THIRD EDITION

PHP Cookbook

David Sklar, Adam Trachtenberg

O'REILLY®

Beijing • Cambridge • Farnham • Köln • Sebastopol • Tokyo

ТРЕТЬЕ ИЗДАНИЕ

PHP

Рецепты программирования

Дэвид Скляр, Адам Трахтенберг

ПИТЕР®

Москва · Санкт-Петербург · Нижний Новгород · Воронеж
Ростов-на-Дону · Екатеринбург · Самара · Новосибирск
Киев · Харьков · Минск
2015

ББК 32.988.02-018
УДК 004.738.5
С43

Скляр Д., Трахтенберг А.

С43 PHP. Рецепты программирования. 3-е изд. — СПб.: Питер, 2015. — 784 с.: ил. — (Серия «Бестселлеры O'Reilly»).

ISBN 978-5-496-01592-9

Третье издание этой популярной книги представляет собой подборку готовых решений наиболее распространенных задач на языке PHP. Изложен материал, интересный каждому разработчику: базовые типы данных, операции с ними, файлы cookie, функции PHP, аутентификация пользователей, работа со слоями, проблемы безопасности, ускорение действия программ, работа в сети, создание графических изображений, обработка ошибок, отладка сценариев и написание тестов. Даны рецепты, затрагивающие основы объектно-ориентированного программирования и новые функциональные возможности PHP. Каждый рецепт является самодостаточным и показывает весь путь решения задачи.

Третье издание книги полностью обновлено под версию PHP 5.4, а также включает ряд новых разделов по работе с данными.

12+ (В соответствии с Федеральным законом от 29 декабря 2010 г. № 436-ФЗ.)

ББК 32.988.02-018
УДК 004.738.5

ISBN 978-1449363758 англ.

ISBN 978-5-496-01592-9

Краткое содержание

Оглавление

Предисловие

Технология PHP лежит в основе миллионов динамических веб-приложений. Широкий набор возможностей, доступный синтаксис и поддержка разных операционных систем и веб-серверов делают PHP идеальным языком как для ускоренной веб-разработки, так и для методичного конструирования сложных систем.

Одним из главных факторов успеха PHP как языка веб-сценариев стало его происхождение от инструмента обработки форм HTML и создания веб-страниц. Благодаря этому обстоятельству PHP отлично сочетается с веб-программированием. Вдобавок PHP отличается выдающейся «всеядностью» в отношении внешних приложений и библиотек. PHP может взаимодействовать со многими базами данных и поддерживает многочисленные протоколы Интернета, упрощает разбор данных форм и выдачу запросов HTTP. Эта ориентированность на веб-разработку проявляется в рецептах и примерах этой книги.

Перед вами набор решений типичных задач, встречающихся при программировании на PHP. Мы постарались включить в книгу материал, который представляет интерес для широкого спектра пользователей, от новичков до экспертов. Если нам это удалось, вы узнаете что-то новое (а может, много всего!) из книги. Одни рецепты предназначены для тех, кто занимается повседневным программированием на PHP, а другие — для разработчиков, осваивающих PHP с опытом работы на другом языке.

Исходный код и двоичные файлы PHP доступны для бесплатной загрузки по адресу *http://www.php.net/*. На сайте PHP также можно найти инструкции по установке, подробную документацию и ссылки на сетевые ресурсы, посвященные PHP, пользовательские группы, списки рассылки и т. д.

Для кого предназначена эта книга

Эта книга предназначена для программистов, желающих решать практические задачи на PHP. Если вы пока ничего не знаете о PHP, воспользуйтесь каким-нибудь учебником для начинающих.

Читателям, уже знакомым с PHP, книга поможет справиться с конкретными задачами и упростит жизнь (по крайней мере в том, что касается программирования). Вы узнаете, как на PHP выполняются такие операции, как отправка электронной почты или разбор данных JSON, — которые вы, возможно, уже умеете выполнять на других языках. Книга станет верным помощником для программистов, занимающихся переработкой приложений с других языков на PHP.

О чем говорится в книге

Никто не ожидает, что вы сядете и прочитаете книгу от корки до корки (хотя если так — мы будем только рады!). Программисты PHP постоянно сталкиваются с множеством разнообразных задач по широкому спектру тем. Обращайтесь к книге тогда, когда у вас появится конкретная задача. Каждый рецепт содержит самостоятельное объяснение, которое станет хорошей отправной точкой для дальнейшей работы. Если в рецепте упоминаются темы, выходящие за его рамки, то в него включаются указатели на другие рецепты, сетевые и автономные ресурсы.

Если вы решите прочитать целую главу — тоже хорошо. Рецепты обычно следуют от простых к сложным, а в конце многих глав приводятся примеры программ, которые связывают воедино весь представленный материал. Во вводной части каждой главы приводится обзор материала, включая необходимые сведения общего характера, а также упоминаются рецепты, представляющие особый интерес.

Первые четыре главы книги посвящены основным типам данных. В главе 1 рассматриваются такие базовые операции, как обработка подстрок, преобразования регистра символов, разбиение строк и разбор данных, разделенных запятыми. В главе 2 рассматриваются операции с вещественными числами, генерирование случайных чисел, преобразование систем счисления и числовое форматирование. Глава 3 демонстрирует работу с датой и временем, их форматирование, обработку часовых поясов и летнего времени, а также определение времени с точностью до миллисекунд. В главе 4 представлены операции с массивами: перебор, слияние, обратная перестановка элементов, сортировка и извлечение отдельных элементов.

Далее идут три главы, в которых рассматриваются основные структурные элементы программ. В главе 5 рассматриваются такие возможности работы с переменными PHP, как значения по умолчанию, статические переменные и получение строковых представлений сложных типов данных. Рецепты главы 6 относятся к использованию функций в PHP: обработке аргументов, передаче и возвращению переменных по ссылке, созданию функций во время выполнения и области видимости переменных. В главе 7 рассматриваются объектно-ориентированные возможности PHP, от простейших до таких нетривиальных возможностей, как специальные методы, деструкторы, управление доступом, отражение, типажи и пространства имен.

После типов данных и структурных элементов программ следуют шесть глав с темами, занимающими центральное место в веб-программировании. В главе 8 рассматриваются cookie, заголовки, аутентификация, работа со строками запросов и другие ключевые возможности веб-приложений. Глава 9 посвящена обработке и проверке данных, вводимых на формах, отображению многостраничных форм, выводу форм с сообщениями об ошибках, а также защите от таких потенциальных опасностей, как межсайтовые сценарные атаки и повторная отправка форм. Глава 10 объясняет различия между базами данных SQL и DBM, а на

примере уровня абстрагирования доступа к базам данных PDO показывает, как подключиться к базе данных, присвоить уникальные значения идентификаторов, прочитать строки, изменить данные, экранировать специальные символы и сохранить в журнале отладочную информацию. В главе 11 рассматривается встроенный сеансовый модуль PHP, который позволяет сохранять информацию о пользователе при переходе от страницы к странице. Также в этой главе описаны некоторые проблемы безопасности, возникающие при работе с сеансовыми данными. Основной темой главы 12 является язык разметки XML: расширение SimpleXML и функции DOM, использование XPath и XSLT, чтение и запись каналов RSS и Atom. В главе 13 рассматриваются темы, полезные для приложений PHP, интегрированных с внешними сайтами и клиентским кодом JavaScript: обращение по URL-адресам, чистка разметки HTML и реакция на запросы Ajax.

Следующие три главы посвящены сетевым взаимодействиям. В главе 14 обсуждаются нюансы использования веб-служб (на примере внешних REST-совместимых служб) из вашего кода. В главе 15 вам предстоит взглянуть на работу веб-служб с обратной стороны — речь пойдет об организации обработки REST-запросов. В обеих главах обсуждаются вопросы аутентификации, работы с заголовками и обработки ошибок. В главе 16 обсуждаются другие сетевые сервисы: отправка сообщений электронной почты, использование LDAP и преобразования DNS.

Следующая часть книги содержит несколько глав, посвященных возможностям и расширениям PHP, способствующим построению мощных, безопасных, удобных и эффективных приложений. Глава 17 показывает, как создавать графические изображения; в ее рецептах рассматривается вывод текста, линий, многоугольников и кривых. В главе 18 основное внимание уделяется вопросам безопасности: предотвращению фиксации сеанса и межсайтовых сценарных атак, работе с паролями и шифрованию данных. Глава 19 рассказывает, как сделать ваше приложение удобным для пользователей из других стран; также в ней приводятся рецепты локализации текста, даты и времени, денежных сумм и графики, а также работа с текстом в кодировке UTF-8. В главе 20 подробно рассматривается обработка ошибок и ведение журналов, тогда как в главе 21 рассматриваются средства отладки, написание тестов и использование встроенного веб-сервера PHP. В главе 22 объясняется, как сравнить быстродействие двух функций и добиться выполнения программ на максимальной скорости. В главе 23 рассматриваются регулярные выражения, сохранение текста из тегов HTML, вызов функций PHP из регулярных выражений, использование максимального и минимального поиска.

Главы 24 и 25 посвящены файловой системе. Глава 24 ограничивается операциями с файлами: открытие и закрытие, использование временных файлов, установление блокировки, передача сжатых файлов, обработка содержимого файлов. В главе 25 речь пойдет о каталогах и метаданных файлов, а в ее рецептах рассматривается изменение разрешений и владельца, перемещение и удаление файлов, обработка всех файлов в каталоге.

Наконец, в двух последних главах рассматриваются темы, выходящие за рамки возможностей PHP. В главе 26 вы узнаете, что можно сделать при помощи PHP за пределами веб-программирования. В этих рецептах рассматриваются темы, относящиеся к режиму командной строки, такие как разбор аргументов программы и чтение паролей. Глава 27 знакомит читателя с репозиториями Composer, PEAR (PHP Extension and Application Repository) и PECL (PHP Extension Community Library). Composer и PEAR предоставляют доступ к коду PHP, предоставляющему функции и расширения для PHP. PECL также содержит подборку расширений PHP, но написанных на C. Модули PEAR и PECL неоднократно используются в книге, а в главе 27 описан процесс их установки и обновления.

Другие ресурсы

Сайты

В Интернете доступно огромное количество справочных материалов по PHP, от справочных руководств с аннотациями до периодики со статьями и учебниками. Быстрое подключение к Интернету может оказаться полезнее большой книжной полки с литературой по PHP. Ниже перечислены некоторые важные сайты.

- The Annotated PHP Manual (*http://php.net/manual/en/index.php*) — сайт содержит официальную документацию по функциям и возможностям PHP, а также пользовательские комментарии (поддерживаются 11 языков).
- Списки рассылки PHP (*http://php.net/mailing-lists.php*) — по тематике PHP ведутся многочисленные списки рассылки, посвященные установке, программированию, расширениям PHP и другим вопросам; также имеется веб-интерфейс к спискам рассылки (*http://news.php.net/*), доступный только для чтения.
- Ресурсы поддержки PHP (*http://us3.php.net/support.php*) — полезная подборка ресурсов с информацией о группах пользователей PHP, мероприятиях и других каналах поддержки.
- Composer (*https://getcomposer.org/*) — менеджер зависимостей для PHP, предоставляющий структурированный механизм объявления и установки зависимостей в проектах.
- PEAR (*http://pear.php.net/*) — разработчики описывают PEAR как «инфраструктуру и систему распространения многократно используемых компонентов PHP». Здесь вы найдете много полезных классов PHP и примеров кода. За дополнительной информацией о PEAR обращайтесь к главе 27.
- PECL (*http://pecl.php.net/*) — описывается как «репозиторий для расширений PHP, со справочником расширений и инструментами, упрощающими загруз-

ку и разработку расширений PHP». За дополнительной информацией о PECL обращайтесь к главе 27.

- PHP.net: A Tourist's Guide (*http://php.net/sites.php*) — описание различных сайтов, входящих в иерархию *php.net*.
- PHP: The Right Way (*http://www.phptherightway.com/*) — краткий справочник, который задуман как всесторонний источник информации о приемах программирования PHP. Сайт станет отличной отправной точкой для разработчиков, интересующихся идиомами программирования на PHP.
- Planet PHP (*http://www.planet-php.net/*) — агрегатор публикаций в блогах разработчиков, посвященных PHP.
- Блоги SitePoint, посвященные PHP (*http://www.sitepoint.com/php/*) — полезная подборка информации о PHP.

Благодарности

Прежде всего мы должны поблагодарить всех, кто не пожалел своего времени, творческой энергии и мастерства для того, чтобы поднять PHP до нынешнего уровня. Выдающимися усилиями добровольцев были созданы не только сотни тысяч строк кода, но и подробная документация, инфраструктура контроля качества, множество дополнительных приложений и библиотек, а также преуспевающее сообщество пользователей по всему миру. Возможность написать новую книгу для мира PHP стало для нас честью и захватывающим приключением.

Мы выражаем благодарность всем нашим рецензентам: Полу Хаффу (Paul Huff), Питеру Макинтайру (Peter MacIntyre), Саймону Макинтайру (Simon MacIntyre) и Рассу Уману (Russ Uman). Отдельное спасибо Крису Шифлету (Chris Shiflett) и Клею Лавлейсу (Clay Lovelace) за их творческий вклад во второе издание книги.

Спасибо всем сотрудникам издательства O'Reilly, благодаря которым появилась эта книга: это Рэчел Румелиотис (Rachel Roumeliotis), Элисон Макдональд (Allyson MacDonald), Мелани Ярбро (Melanie Yarbrough) и Мария Гулик (Maria Gulick), а также бесчисленные трудяги, обеспечившие плавный ход производственного процесса.

Дэвид Скляр

Еще раз хочу поблагодарить Адама. Мы проработали вместе (тем или иным образом) уже 18 лет, и из них 17 занимались программированием PHP. До сих пор не существует другого соавтора, с которым я хотел бы работать над этой книгой (если уж быть совсем откровенным — кроме Бена Франклина, если бы его удалось вернуть к жизни).

Спасибо всем членам моей семьи. Благодаря вам у меня было время и место для работы над книгой. А теперь вам придется найти время и место, чтобы все это прочитать!

Адам Трахтенберг

Дэвид, пожалуй, мне трудно соревноваться с Беном Франклином. И кстати, я предпочел бы видеть официальным талисманом PHP индюка, а не слона. Большое спасибо за твою поддержку все эти годы, начиная с дней PHP/FI. Без тебя эта книга так бы и осталась на стадии замыслов.

Огромное спасибо моим друзьям и семье за их поддержку и содействие. Я люблю своих двоих сыновей — даже того, кто напомнил мне, что дети не соглашаются на отсрочку, если книга не закончена через 40 недель. Наконец, отдельное спасибо моей жене Элизабет Энн; мне следовало бы почаще прислушиваться к ее мудрым советам.

От издательства

Ваши замечания, предложения, вопросы отправляйте по адресу comp@piter.com (издательство «Питер», компьютерная редакция).

Мы будем рады узнать ваше мнение!

На веб-сайте издательства www.piter.com вы найдете подробную информацию о наших книгах.

1 Строки

1.0. Введение

Строки в PHP представляют собой последовательности байтов, например «Once upon a time» или даже «111211211». При чтении из файла или выводе в браузере данные представляются в виде строк.

Строки PHP являются *двоично-безопасными* (то есть могут содержать нуль-байты), они могут увеличиваться и уменьшаться по мере необходимости. Их размер ограничивается только объемом памяти, доступной для PHP.

Обычно строки PHP являются ASCII-строками. Иначе говоря, для работы с данными, не входящими в набор ASCII, например текстом в UTF-8 или другой многобайтовой кодировке, приходится предпринимать дополнительные усилия (глава 19).

По аналогии со строками Perl и командного процессора Unix строки PHP могут инициализироваться тремя способами: в одиночных кавычках, двойных кавычках и в формате *heredoc*. В строках, заключенных в одиночные кавычки, из всех специальных символов должны экранироваться (escaped) только обратная косая черта (\) и одиночная кавычка ('). Ниже приведены примеры четырех строк, заключенных в одиночные кавычки:

```
print 'I have gone to the store.';
print 'I\'ve gone to the store.';
print 'Would you pay $1.75 for 8 ounces of tap water?';
print 'In double-quoted strings, newline is represented by \n';
```

Команды выводят следующие результаты:

```
I have gone to the store.
I've gone to the store.
Would you pay $1.75 for 8 ounces of tap water?
In double-quoted strings, newline is represented by \n
```

В этом примере представлен непосредственно выводимый текст. При просмотре в браузере все фразы будут выводиться в одну линию, потому что в HTML для вставки разрывов строк необходима дополнительная разметка.

Так как в строках, заключенных в одиночные кавычки, PHP не проверяет интерполяцию переменных и почти никакие служебные комбинации, этот способ определения строк быстр и прямолинеен.

В строках, заключенных в двойные кавычки, экранированные одиночные кавычки не распознаются. С другой стороны, в них поддерживается интерполяция переменных и служебные комбинации, перечисленные в таблице 1.1.

Таблица 1.1. Служебные комбинации в строках, заключенных в двойные кавычки

Служебная комбинация	Символ
\n	Новая строка (ASCII 10)
\r	Возврат курсора (ASCII 13)
\t	Табуляция (ASCII 9)
\\	Обратная косая черта
\$	Знак доллара
\"	Кавычка
\0 — \777	Восьмеричное значение
\x0 — \xFF	Шестнадцатеричное значение

В листинге 1.1 приведены примеры строк, заключенных в двойные кавычки.

Листинг 1.1. Строки, заключенные в двойные кавычки

```
print "I've gone to the store.";
print "The sauce cost \$10.25.";
$cost = '$10.25';
print "The sauce cost $cost.";
print "The sauce cost \$\061\060.\x32\x35.";
```

Код листинга 1.1 выводит следующий текст:

```
I've gone to the store.
The sauce cost $10.25.
The sauce cost $10.25.
The sauce cost $10.25.
```

Последняя строка листинга 1.1 выводит правильное значение, потому что символу `1` в кодировке ASCII соответствует десятичный код 49 — или 061 в восьмеричной записи. Символу `0` в ASCII соответствует десятичный код 48 (060 в восьмеричной записи); символу 2 — десятичный код 50 (32 в шестнадцатеричной записи); символу 5 — десятичный код 53 (35 в шестнадцатеричной записи).

Строки в формате *heredoc* поддерживают все интерполяции и служебные комбинации строк в двойных кавычках, но не требуют экранирования символа двойной кавычки. Строка в формате *heredoc* начинается с последовательности <<< и маркера. Этот же маркер (не имеющий начальных или завершающих пропусков), за которым следует символ «;» для завершения команды (если необходимо), завершает строку *heredoc*. Пример определения строки в формате *heredoc* представлен в листинге 1.2.

Листинг 1.2. Определение строки в формате heredoc

```
print <<< END
It's funny when signs say things like:
   Original "Root" Beer
   "Free" Gift
   Shoes cleaned while "you" wait
or have other misquoted words.
END;
```

Код листинга 1.2 выводит следующий текст:

```
It's funny when signs say things like:
   Original "Root" Beer
   "Free" Gift
   Shoes cleaned while "you" wait
or have other misquoted words.
```

Разрывы строк, пробелы и двойные кавычки сохраняются в *heredoc*-строках. По общепринятым соглашениям идентификатор конца строки обычно записывается прописными буквами, и в нем учитывается регистр символов. В листинге 1.3 приведены еще две действительные строки в формате *heredoc*.

Листинг 1.3. Дополнительные примеры строк в формате heredoc

```
print <<< PARSLEY
It's easy to grow fresh:
Parsley
Chives
on your windowsill
PARSLEY;

print <<< DOGS
If you like pets, yell out:
DOGS AND CATS ARE GREAT!
DOGS;
```

Формат *heredoc* особенно удобен для вывода разметки HTML с интерполированными переменными, потому что он избавляет от необходимости экранировать двойные кавычки, встречающиеся в элементах HTML. В листинге 1.4 представлен пример использования формата *heredoc* для вывода разметки HTML.

Листинг 1.4. Вывод разметки HTML из heredoc-строки

```
if ($remaining_cards > 0) {
    $url = '/deal.php';
```

```
    $text = 'Deal More Cards';
} else {
    $url = '/new-game.php';
    $text = 'Start a New Game';
}

print <<< HTML
There are <b>$remaining_cards</b> left.
<p>
<a href="$url">$text</a>
HTML;
```

В листинге 1.4 за конечным маркером должен следовать символ «;», который сообщает PHP о завершении команды. Впрочем, в некоторых случаях этот символ не используется. Один из таких случаев представлен в листинге 1.5: здесь строка в формате *heredoc* используется с оператором конкатенации строк.

Листинг 1.5. Конкатенация со строкой в формате heredoc

```
$html = <<< END
<div class="$divClass">
<ul class="$ulClass">
<li>
END
. $listItem . '</li></div>';

print $html;
```

Если предположить, что переменным `$divClass`, `$ulClass` и `$listItem` присвоены некие разумные значения, результат выполнения листинга 1.5 будет выглядеть примерно так:

```
<div class="class1">>
<ul class="class2">
<li> The List Item </li></div>
```

В листинге 1.5 выражение должно продолжаться в следующей строке, поэтому символ «;» не используется. Также следует учесть, что для распознавания ограничителя конца строки оператор конкатенации `.` должен находиться в разных строках с ограничителем конца строки.

Формат *nowdoc* похож на *heredoc*, но он не поддерживает интерполяцию строк. Таким образом, строки в формате *nowdoc* связаны с *heredoc* примерно так же, как строки в одиночных кавычках со строками в двойных кавычках. Лучше всего использовать их при наличии блока, не являющегося кодом PHP (например, JavaScript), который нужно вывести как часть страницы HTML или отправить другой программе.

Например, при использовании jQuery это выглядит так:

```
$js = <<<'__JS__'
$.ajax({
  'url': '/api/getStock',
  'data': {
```

```
    'ticker': 'LNKD'
  },
  'success': function( data ) {
    $( "#stock-price" ).html( "<strong>$" + data + "</strong>" );
  }
});
__JS__;

print $js;
```

Для обращения к отдельным байтам строки используется синтаксис индексирования `[]`. Первый байт строки имеет индекс 0. В листинге 1.6 из строки извлекается один байт.

Листинг 1.6. Обращение к отдельному байту в строке

```
$neighbor = 'Hilda';
print $neighbor[3];
```

Результат выполнения листинга 1.6:

```
d
```

1.1. Обращение к подстрокам

Задача

Требуется узнать, содержит ли строка заданную подстроку, например проверить, содержит ли адрес электронной почты знак `@`.

Решение

Воспользуйтесь функцией `strpos()`, как показано в листинге 1.7.

Листинг 1.7. Поиск подстроки с использованием функции strpos( )

```
if (strpos($_POST['email'], '@') === false) {
    print 'There was no @ in the e-mail address!';
}
```

Комментарий

Функция `strpos()` возвращает индекс первой позиции в строке, в которой была найдена заданная подстрока. Если найти подстроку не удалось, `strpos()` возвращает `false`. Если строка начинается с искомой подстроки, `strpos()` возвращает 0, потому что индекс 0 соответствует началу строки. Чтобы отличить возвращаемое значение 0 от `false`, следует использовать оператор тождественности (`===`) или нетождественности (`!==`) вместо обычных операторов равенства (`==`) или

неравенства (`!=`). Листинг 1.7 сравнивает возвращаемое значение `strpos()` с `false` при помощи оператора `===`. Проверка проходит успешно только в том случае, если `strpos()` возвращает `false` — но не 0 или какое-либо другое число.

См. также

Документация по функции `strpos()`[1].

1.2. Выделение подстрок

Задача

Требуется выделить часть строки начиная с заданной позиции (например, первые восемь символов имени пользователя, введенного на форме).

Решение

Воспользуйтесь функцией `substr()`, как показано в листинге 1.8.

Листинг 1.8. Выделение подстроки с использованием функции substr()

```
$substring = substr($string,$start,$length);
$username = substr($_GET['username'],0,8);
```

Комментарий

Если значения `$start` и `$length` положительны, `substr()` возвращает `$length` символов строки начиная с позиции `$start`. Первый символ строки находится в позиции 0. В листинге 1.9 используются положительные значения `$start` и `$length`.

Листинг 1.9. Использование substr() с положительными значениями $start и $length

```
print substr('watch out for that tree',6,5);
```

Результат выполнения листинга 1.9:

```
out f
```

Если значение `$length` не указано, `substr()` возвращает строку от `$start` до конца исходной строки, как показано в листинге 1.10.

Листинг 1.10. Использование substr() с положительным значением $start без $length

```
print substr('watch out for that tree',17);
```

---

[1] Здесь и далее: документацию по функциям можно найти на сайте *www.php.net*.

Результат выполнения листинга 1.10:

```
t tree
```

Если значение `$start` больше длины строки, `substr()` возвращает `false`.

Если `$start` в сумме с `$length` выходит за конец строки, `substr()` возвращает всю оставшуюся часть строки начиная с `$start`, как показано в листинге 1.11.

Листинг 1.11. Использование substr() с длиной, выходящей за конец строки

```
print substr('watch out for that tree',20,5);
```

Результат выполнения листинга 1.11:

```
ree
```

Если значение `$start` отрицательно, то `substr()` при определении начальной позиции подстроки начинает отсчет от конца строки, как показано в листинге 1.12.

Листинг 1.12. Использование substr() с отрицательным значением $start

```
print substr('watch out for that tree',-6);
print substr('watch out for that tree',-17,5);
```

Результат выполнения листинга 1.12:

```
t tree
out f
```

С отрицательными значениями `$start`, выходящими за начало строки (например, если значение `$start` равно –27 для строки из 20 символов), `substr()` ведет себя так, как если бы значение `$start` было равно 0.

Если значение `$length` отрицательно, то для определения конечной позиции подстроки функция `substr()` отсчитывает символы от конца строки, как показано в листинге 1.13.

Листинг 1.13. Использование substr() с отрицательным значением $length

```
print substr('watch out for that tree',15,-2);
print substr('watch out for that tree',-4,-1);
```

Результат выполнения листинга 1.13:

```
hat tr
tre
```

См. также

Документация по функции `substr()`.

1.3. Замена подстрок

Задача

Требуется заменить подстроку другой строкой, например скрыть все цифры номера кредитной карты, кроме четырех последних, перед выводом.

Решение

Воспользуйтесь функцией `substr_replace()`, как показано в листинге 1.14.

Листинг 1.14. Замена подстроки с использованием функции substr_replace()

```
// Все символы от позиции $start до конца $old_string
// заменяются содержимым $new_substring
$new_string = substr_replace($old_string,$new_substring,$start);
// $length символов, начиная с позиции $start,
// заменяются содержимым $new_substring
$new_string = substr_replace($old_string,$new_substring,$start,$length);
```

Комментарий

Без аргумента `$length` функция `substr_replace()` заменяет все символы от позиции `$start` до конца строки. Если значение `$length` задано, то заменяется указанное количество символов:

```
print substr_replace('My pet is a blue dog.','fish.',12);
print substr_replace('My pet is a blue dog.','green',12,4);
$credit_card = '4111 1111 1111 1111';
print substr_replace($credit_card,'xxxx ',0,strlen($credit_card)-4);

My pet is a fish.
My pet is a green dog.
xxxx 1111
```

Если значение `$start` отрицательно, то положение новой подстроки определяется отсчетом `$start` символов с конца `$old_string`, а не с начала:

```
print substr_replace('My pet is a blue dog.','fish.',-9);
print substr_replace('My pet is a blue dog.','green',-9,4);

My pet is a fish.
My pet is a green dog.
```

Если оба параметра `$start` и `$length` равны 0, то новая подстрока вставляется в начало `$old_string`:

```
print substr_replace('My pet is a blue dog.','Title: ',0,0);

Title: My pet is a blue dog.
```

Функция `substr_replace()` удобна для сокращения текста, слишком длинного для вывода, или для вывода части текста с созданием ссылки. Код листинга 1.15 выводит первые 25 символов сообщения, за которыми следует многоточие — ссылка на страницу для вывода дополнительного текста.

Листинг 1.15. Вывод длинного текста с многоточием

```
$r = mysql_query("SELECT id,message FROM messages WHERE id = $id") or die();
$ob = mysql_fetch_object($r);
printf('<a href="more-text.php?id=%d">%s</a>',
       $ob->id, substr_replace($ob->message,' ...',25));
```

Страница more-text.php, упоминаемая в листинге 1.15, может использовать идентификатор сообщения, переданный в строке запроса, для получения и вывода полного текста сообщения.

См. также

Документация по функции `substr_replace()`.

1.4. Обработка строки по одному байту

Задача

Требуется последовательно обработать каждый байт в строке.

Решение

Переберите байты строки в цикле `for`. Код листинга 1.16 подсчитывает гласные буквы в строке.

Листинг 1.16. Обработка строки по байтам

```
$string = "This weekend, I'm going shopping for a pet chicken.";
$vowels = 0;
for ($i = 0, $j = strlen($string); $i < $j; $i++) {
    if (strstr('aeiouAEIOU',$string[$i])) {
        $vowels++;
    }
}
```

Комментарий

Посимвольная обработка строк позволяет легко реализовать программу подсчета серий символов по принципу «посмотреть и сказать», приведенную в листинге 1.17.

Листинг 1.17. Подсчет серий символов

```
function lookandsay($s) {
    // Возвращаемое значение инициализируется пустой строкой
    $r = '';
    // Переменная $m содержит подсчитываемый символ; инициализируем
    // первым символом в строке
    $m = $s[0];
    // В $n хранится количество обнаруженных $m, инициализируется 1
    $n = 1;
    for ($i = 1, $j = strlen($s); $i < $j; $i++) {
        // Если символ совпадает с предыдущим
        if ($s[$i] == $m) {
            // Увеличить счетчик этого символа
            $n++;
        } else {
            // Иначе добавить счетчик и символ к возвращаемому значению
            $r .= $n.$m;
            // Текущий символ назначается искомым
            $m = $s[$i];
            // Счетчик сбрасывается до 1
            $n = 1;
        }
    }
    // Вернуть построенную строку, а также последнее значение счетчика и символ
    return $r.$n.$m;
}

for ($i = 0, $s = 1; $i < 10; $i++) {
    $s = lookandsay($s);
    print "$s\n";
}
```

Код листинга 1.17 выводит следующий результат:

```
1
11
21
1211
111221
312211
13112221
1113213211
31131211131221
13211311123113112211
```

Принцип «посмотреть и сказать» назван так, потому что для получения каждого элемента программа «просматривает» предыдущий элемент и «говорит», что он содержит. Например, первый элемент 1 — это «одна единица», или «11». Эта строка содержит две единицы, или «21». Она состоит из одной двойки и одной единицы, поэтому четвертый элемент последовательности содержит «1211», и т. д.

См. также

Документация по циклам `for`; принцип «посмотреть и сказать» *http://bit.ly/1g2X0sD*.

1.5. Обратная перестановка строки по словам или байтам

Задача

Требуется переставить в обратном порядке слова или байты в строке.

Решение

Воспользуйтесь функцией `strrev()` для получения обратной перестановки строки по байтам, как показано в листинге 1.18.

Листинг 1.18. Обратная перестановка строки по байтам

```
print strrev('This is not a palindrome.');
```

Код листинга 1.18 выводит следующий результат:

```
.emordnilap a ton si sihT
```

Чтобы переставить строку по словам, разбейте ее по границам слов, переставьте слова, а затем соедините их, как показано в листинге 1.19.

Листинг 1.19. Обратная перестановка строки по словам

```
$s = "Once upon a time there was a turtle.";
// Разбиение строки по словам
$words = explode(' ',$s);
// Массив слов
$words = array_reverse($words);
// Построение строки
$s = implode(' ',$words);
print $s;
```

Код листинга 1.19 выводит следующий результат:

```
turtle. a was there time a upon Once
```

Комментарий

Обратная перестановка строки по словам также может быть выполнена в одной строке кода, как показано в листинге 1.20.

Листинг 1.20. Компактная обратная перестановка строки по словам

```
$reversed_s = implode(' ',array_reverse(explode(' ',$s)));
```

См. также

Рецепт 24.7 — последствия использования других символов (вместо пробела) как границы слова; документация по функциям `strrev()` и `array_reverse()`.

1.6. Генерирование случайной строки

Задача

Требуется сгенерировать случайную строку.

Решение

Воспользуйтесь функцией `str_rand()`:

```
function str_rand($length = 32,
    $characters = ⏎
'0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ') {
if (!is_int($length) || $length < 0) {
    return false;
}

    $characters_length = strlen($characters) - 1;
$string = '';
for ($i = $length; $i > 0; $i--) {
$string .= $characters[mt_rand(0, $characters_length)];
}

    return $string;
}
```

Комментарий

В PHP имеются встроенные функции для генерирования случайных чисел, но не для случайных строк. Функция `str_rand()` возвращает 32-символьную строку, состоящую из букв и цифр.

Передайте функции целое число, чтобы изменить длину возвращаемой строки. Чтобы использовать альтернативный набор символов, передайте их в виде строки во втором аргументе. Например, для получения кода Морзе из 16 знаков можно воспользоваться следующим вызовом:

```
print str_rand(16, '.-');
.--..-.-.--.----
```

См. также

Рецепт 2.5 — генерирование случайных чисел.

1.7. Сжатие и свертка табуляций

Задача

Требуется заменить пробелы табуляциями (или табуляции — пробелами) в строке с сохранением выравнивания текста по позициям табуляции. Например, такая

необходимость может возникнуть для стандартизации вывода отформатированного текста.

Решение

Воспользуйтесь функцией `str_replace()` для замены пробелов символами табуляции или наоборот, как показано в листинге 1.21.

Листинг 1.21. Замена табуляций и пробелов

```
$rows = $db->query('SELECT message FROM messages WHERE id = 1');
$obj = $rows->fetch(PDO::FETCH_OBJ);

$tabbed = str_replace(' ' , "\t", $obj->message);
$spaced = str_replace("\t", ' ' , $obj->message);

print "With Tabs: <pre>$tabbed</pre>";
print "With Spaces: <pre>$spaced</pre>";
```

Однако преобразование функцией `str_replace()` не сохраняет позиции табуляции. Если вы хотите, чтобы позиции табуляции следовали через восемь символов, то в строке, начинающейся со слова из пяти букв и символа табуляции, последний должен быть заменен тремя пробелами, а не одним. Используйте функцию `tab_expand()` из листинга 1.22 для преобразования табуляций в пробелы с сохранением позиций табуляции.

Листинг 1.22. tab_expand()

```
function tab_expand($text) {
    while (strstr($text,"\t")) {
        $text = preg_replace_callback('/^([^\t\n]*)(\t+)/m',
                                      'tab_expand_helper', $text);
    }
    return $text;
}

function tab_expand_helper($matches) {
    $tab_stop = 8;
    return $matches[1] .
    str_repeat(' ',strlen($matches[2]) *
                       $tab_stop - (strlen($matches[1]) % $tab_stop));
}

$spaced = tab_expand($obj->message);
```

Обратное преобразование пробелов в табуляции может быть выполнено функцией `tab_unexpand()` из листинга 1.23.

Листинг 1.23. tab_unexpand()

```
function tab_unexpand($text) {
    $tab_stop = 8;
    $lines = explode("\n",$text);
    foreach ($lines as $i => $line) {
        // Преобразование табуляций в пробелы
```

```
        $line = tab_expand($line);
        $chunks = str_split($line, $tab_stop);
        $chunkCount = count($chunks);
        // Просмотр всех фрагментов, кроме последнего
        for ($j = 0; $j < $chunkCount - 1; $j++) {
            $chunks[$j] = preg_replace('/ {2,}$/',"\t",$chunks[$j]);
        }
        // Если последний фрагмент содержит пробелы, заполняющие
        // позицию табуляции, преобразовать их в табуляцию. В противном
        // случае оставить их без изменения.
        if ($chunks[$chunkCount-1] == str_repeat(' ', $tab_stop)) {
            $chunks[$chunkCount-1] = "\t";
        }
        // Объединение фрагментов
        $lines[$i] = implode('',$chunks);
    }
    // Объединение строк
    return implode("\n",$lines);
}

$tabbed = tab_unexpand($obj->message);
```

Обе функции получают строковый аргумент и возвращают строку, полученную в результате преобразования.

Комментарий

Обе функции предполагают, что позиции табуляции следуют через каждые восемь пробелов, но размер позиций можно изменить, задавая значение переменной `$tab_stop`.

Регулярное выражение в `tab_expand()` совпадает с группой табуляций и всем текстом в строке, предшествующим этой группе табуляций. Совпадение с текстом перед табуляциями необходимо из-за того, что длина этого текста влияет на количество заменяемых пробелов, чтобы дальнейший текст был выровнен по следующей позиции табуляции. Функция не заменяет каждую табуляцию восемью пробелами; она изменяет текст после табуляций так, чтобы он выравнивался по позициям табуляции.

Аналогичным образом функция `tab_unexpand()` не ограничивается поиском восьми последовательных пробелов и их заменой одним символом табуляции. Она разбивает каждую строку на фрагменты, состоящие из 8 символов, и заменяет завершающие пробелы в каждом из этих фрагментов (минимум два) табуляциями. Такая замена не только сохраняет выравнивание текста по позициям табуляции, но и экономит память, занимаемую строкой.

См. также

Документация по функциям `str_replace()`, `preg_replace_callback()` и `str_split()`. В Рецепте 23.10 функция `preg_replace_callback()` рассматривается более подробно.

1.8. Управление регистром символов

Задача

Требуется привести символы к верхнему или нижнему регистру либо иным образом изменить регистр символов строки. Например, имена в списке должны начинаться с прописных букв, но остальные буквы должны быть строчными.

Решение

Воспользуйтесь функцией `ucfirst()` или `ucwords()` для преобразования к верхнему регистру первой буквы одного или нескольких слов, как показано в листинге 1.24.

Листинг 1.24. Преобразование регистра символов

```
print ucfirst("how do you do today?");
print ucwords("the prince of wales");
```

Код листинга 1.24 выводит следующий результат:

```
How do you do today?
The Prince Of Wales
```

Для преобразования регистра целых строк используются функции `strtolower()` и `strtoupper()`, как показано в листинге 1.25.

Листинг 1.25. Изменение регистра символов в строках

```
print strtoupper("i'm not yelling!");
print strtolower('<A HREF="one.php">one</A>');
```

Код листинга 1.25 выводит следующий результат:

```
I'M NOT YELLING!
<a href="one.php">one</a>
```

Комментарий

Функция `ucfirst()` преобразует к верхнему регистру первый символ в строке:

```
print ucfirst('monkey face');
print ucfirst('1 monkey face');
```

Результат выглядит так:

```
Monkey face
1 monkey face
```

Обратите внимание: во втором случае не будет получена строка «1 Monkey face».

Функция `ucwords()` преобразует к верхнему регистру первый символ каждого слова в строке:

```
print ucwords('1 monkey face');
print ucwords("don't play zone defense against the philadelphia 76-ers");
```

Результат выглядит так:

```
1 Monkey Face
Don't Play Zone Defense Against The Philadelphia 76-ers
```

Как и ожидалось, `ucwords()` не преобразует букву «t» в «don't», а также букву «e» в «76-ers». Для `ucwords()` словом является любая последовательность символов, не являющихся пропусками (whitespace), за которыми следует один или несколько символов-пропусков. Как ', так и - не являются пропусками, поэтому `ucwords()` не считает «t» в «don't» или «e» в «76-ers» начальными символами слов.

Функции `ucfirst()` и `ucwords()` не изменяют регистр букв, не являющихся первыми буквами в слове:

```
print ucfirst('macWorld says I should get an iBook');
print ucwords('eTunaFish.com might buy itunaFish.Com!');
```

Результат выглядит так:

```
MacWorld says I should get an iBook
ETunaFish.com Might Buy ItunaFish.Com!
```

Функции `strtolower()` и `strtoupper()` работают с целыми строками, а не с отдельными символами. Все алфавитные символы преобразуются к нижнему регистру функцией `strtolower()` и к верхнему регистру функцией `strtoupper()`:

```
print strtolower("I programmed the WOPR and the TRS-80.");
print strtoupper('"since feeling is first" is a poem by e. e. cummings.');
```

Результат выглядит так:

```
i programmed the wopr and the trs-80.
"SINCE FEELING IS FIRST" IS A POEM BY E. E. CUMMINGS.
```

При определении нижнего и верхнего регистра эти функции учитывают параметры локального контекста (locale).

См. также

Глава 19 содержит дополнительную информацию о локальных контекстах; документация по функциям `ucfirst()`, `ucwords()`, `strtolower()` и `strtoupper()`.

1.9. Интерполяция функций и выражений в строках

Проблема

Требуется включить в строку результат выполнения функции или выражения.

Решение

Воспользуйтесь оператором конкатенации строк (`.`), как показано в листинге 1.26, если включаемое значение не может находиться внутри строки.

Листинг 1.26. Конкатенация строк

```
print 'You have '.($_POST['boys'] + $_POST['girls']).' children.';
print "The word '$word' is ".strlen($word).' characters long.';
print 'You owe '.$amounts['payment'].' immediately.';
print "My circle's diameter is ".$circle->getDiameter().' inches.';
```

Комментарий

Переменные, свойства объектов и элементы массивов (если индекс не заключен в двойные кавычки) можно включать прямо в строки, заключенные в двойные кавычки:

```
print "I have $children children.";
print "You owe $amounts[payment] immediately.";
print "My circle's diameter is $circle->diameter inches.";
```

Интерполяция в строках, заключенных в двойные кавычки, накладывает некоторые ограничения на синтаксис интерполируемых элементов. В приведенном примере конструкция `$amounts['payment']` должна быть записана в виде `$amounts[payment]`. Чтобы более сложное выражение правильно интерполировалось в строке, заключите его в фигурные скобки. Например:

```
print "I have {$children} children.";
print "You owe {$amounts['payment']} immediately.";
print "My circle's diameter is {$circle->getDiameter()} inches.";
```

Прямая интерполяция и конкатенация строк также работает со строками в формате *heredoc*. Интерполяция конкатенаций с *heredoc* выглядит немного странно, потому что закрывающий ограничитель *heredoc* и оператор конкатенации должны находиться в разных строках:

```
print <<< END
Right now, the time is
END
. strftime('%c') . <<< END
 but tomorrow it will be
END
. strftime('%c',time() + 86400);
```

Кроме того, при интерполяции строк *heredoc* обязательно следует включить необходимые пробелы, чтобы строка выглядела нормально. В приведенном примере текст `Right now, the time is` должен включать завершающий пробел, а у `but tomorrow it will be` должны присутствовать оба пробела, начальный и завершающий.

См. также

Рецепт 5.4 — синтаксис интерполяции переменных (например, `${"amount_$i"}`); документация по оператору конкатенации строк.

1.10. Удаление начальных или конечных пропусков в строке

Задача

Требуется удалить пропуски в начале или конце строки, например чтобы убрать лишние пробелы из данных, введенных пользователем, перед их проверкой.

Решение

Воспользуйтесь функцией `ltrim()`, `rtrim()` или `trim()`. Функция `ltrim()` удаляет пропуски в начале строки, `rtrim()` — в конце строки и `trim()` — в начале и в конце:

```
$zipcode = trim($_GET['zipcode']);
$no_linefeed = rtrim($_GET['text']);
$name = ltrim($_GET['name']);
```

Комментарий

В контексте этих функций пропуск (whitespace) определяется как один из следующих символов: новая строка, возврат курсора, пробел, горизонтальная и вертикальная табуляция, нуль-символ.

Отсечение пропусков экономит память и позволяет более точно управлять выводом отформатированных данных или текста, например в тегах `<pre>`. Если вы выполняете сравнения с пользовательским вводом, сначала уберите лишние пробелы, чтобы пользователю, который по ошибке ввел свой почтовый индекс 98052 с несколькими пробелами, не пришлось исправлять ошибку (которой на самом деле нет). Также отсечение лишних пробелов перед сравнениями гарантирует, что, например, строка «`salami\n`” будет равна `"salami"`. Также полезно нормализовать строковые данные посредством отсечения пропусков перед сохранением в базе данных.

Функции `trim()` также могут удалять из строк символы, заданные пользователем. Такие символы передаются во втором аргументе. Вы можете определять диапазоны символов, разделяя первый и последний символы диапазона двумя точками:

```
// Удаление цифр и пробелов из начала строки
print ltrim('10 PRINT A$',' 0..9');
// Удаление точки с запятой с конца строки
print rtrim('SELECT * FROM turtles;',';');
```

Результат выглядит так:

```
PRINT A$
SELECT * FROM turtles
```

PHP также предоставляет функцию `chop()`, которая является псевдонимом для `rtrim()`. Впрочем, лучше все же использовать `rtrim()`, потому что функция PHP `chop()` по своему поведению отличается от функции Perl `chop()` (которая все равно считается устаревшей и замененной `chomp()`), и ее использование может сбить с толку других разработчиков, читающих ваш код.

См. также

Документация по функциям `trim()`, `ltrim()` и `rtrim()`.

1.11. Генерирование данных, разделенных запятыми

Задача

Требуется отформатировать данные в виде списка значений, разделенных запятыми (CSV, Comma-Separated Values), чтобы их можно было импортировать в электронную таблицу или базу данных.

Решение

Воспользуйтесь функцией `fputcsv()` для построения строки значений, разделенных запятыми, на базе массива данных.

Листинг 1.27 записывает данные из `$sales` в файл.

Листинг 1.27. Генерирование данных, разделенных запятыми

```
$sales = array( array('Northeast','2005-01-01','2005-02-01',12.54),
                array('Northwest','2005-01-01','2005-02-01',546.33),
                array('Southeast','2005-01-01','2005-02-01',93.26),
                array('Southwest','2005-01-01','2005-02-01',945.21),
                array('All Regions','--','--',1597.34) );

$filename = './sales.csv';
$fh = fopen($filename,'w') or die("Can't open $filename");
foreach ($sales as $sales_line) {
    if (fputcsv($fh, $sales_line) === false) {
        die("Can't write CSV line");
    }
}
fclose($fh) or die("Can't close $filename");
```

Комментарий

Чтобы вывести данные в формате CSV на печать (вместо записи в файл), используйте специальный выходной поток `php://output`, как показано в листинге 1.28.

Листинг 1.28. Вывод данных, разделенных запятыми

```
$sales = array( array('Northeast','2005-01-01','2005-02-01',12.54),
                array('Northwest','2005-01-01','2005-02-01',546.33),
                array('Southeast','2005-01-01','2005-02-01',93.26),
                array('Southwest','2005-01-01','2005-02-01',945.21),
                array('All Regions','--','--',1597.34) );

$fh = fopen('php://output','w');
foreach ($sales as $sales_line) {
    if (fputcsv($fh, $sales_line) === false) {
        die("Can't write CSV line");
    }
}
fclose($fh);
```

Чтобы поместить данные в формате CSV в строку (вместо вывода на печать или записи в файл), объедините метод, продемонстрированный в листинге 1.28, с буферизацией вывода из листинга 1.29.

Листинг 1.29. Размещение данных, разделенных запятыми, в строке

```
$sales = array( array('Northeast','2005-01-01','2005-02-01',12.54),
                array('Northwest','2005-01-01','2005-02-01',546.33),
                array('Southeast','2005-01-01','2005-02-01',93.26),
                array('Southwest','2005-01-01','2005-02-01',945.21),
                array('All Regions','--','--',1597.34) );

ob_start();
$fh = fopen('php://output','w') or die("Can't open php://output");
foreach ($sales as $sales_line) {
    if (fputcsv($fh, $sales_line) === false) {
        die("Can't write CSV line");
    }
}
fclose($fh) or die("Can't close php://output");
$output = ob_get_contents();
ob_end_clean();
```

См. также

Документация по функции `fputcsv()`; Рецепт 8.13 — буферизация вывода.

1.12. Разбор данных, разделенных запятыми

Проблема

Имеется набор данных, разделенных запятыми (формат CSV), например файл, экспортированный из Excel или базы данных. Требуется извлечь записи и поля в формат, с которым удобно работать в PHP.

Решение

Если данные в формате CSV хранятся в файле (или доступны по URL-адресу), откройте файл функцией `fopen()` и прочитайте данные функцией `fgetcsv()`. Листинг 1.30 выводит CSV-данные в виде таблицы HTML.

Листинг 1.30. Чтение данных CSV из файла

```
$fp = fopen($filename,'r') or die("can't open file");
print "<table>\n";
while($csv_line = fgetcsv($fp)) {
    print '<tr>';
    for ($i = 0, $j = count($csv_line); $i < $j; $i++) {
        print '<td>'.htmlentities($csv_line[$i]).'</td>';
    }
    print "</tr>\n";
}
print "</table>\n";
fclose($fp) or die("can't close file");
```

Комментарий

По умолчанию `fgetcsv()` читает полную строку данных. Если средняя строка данных занимает более 8192 байт, ваша программа может работать быстрее, если вы зададите длину явно (вместо того, чтобы доверить ее определение PHP). Для этого `fgetcsv()` передается второй аргумент, значение которого превышает максимальную длину строки данных в файле CSV. (Не забудьте учесть пропуски в конце строк.) Если передаваемая длина строки равна 0, PHP возвращается к поведению по умолчанию.

Функция `fgetcsv()` может получать необязательный третий аргумент — ограничитель, используемый вместо запятой (`,`). Однако использование альтернативного ограничителя в каком-то смысле противоречит смыслу формата CSV как простого механизма обмена табличными данными.

Не поддавайтесь искушению обойти `fgetcsv()`, просто прочитать строку и вызвать `explode()` для запятых. Обработка формата CSV не настолько тривиальна — в частности, она справляется со значениями полей, которые содержат литераль-

ные запятые, которые не должны интерпретироваться как ограничители полей. Использование `fgetcsv()` защищает вас и ваш код от коварных ошибок.

См. также

Документация по функции `fgetcsv()`.

1.13. Генерирование записей с полями фиксированной длины

Задача

Требуется отформатировать записи данных так, чтобы каждое поле занимало указанное количество символов.

Решение

Воспользуйтесь функцией `pack()` с форматной строкой, определяющей последовательность строк, дополненных пробелами.

В листинге 1.31 массив данных преобразуется в записи с полями фиксированной длины.

Листинг 1.31. Генерирование записей данных с полями фиксированной длины

```
$books = array( array('Elmer Gantry', 'Sinclair Lewis', 1927),
                array('The Scarlatti Inheritance','Robert Ludlum', 1971),
                array('The Parsifal Mosaic','William Styron', 1979) );

foreach ($books as $book) {
    print pack('A25A15A4', $book[0], $book[1], $book[2]) . "\n";
}
```

Комментарий

Форматная строка `A25A14A4` приказывает `pack()` преобразовать последующие аргументы в строку из трех фрагментов, дополняемых пробелами: из 25 символов, 14 символов и 4 символов. Для полей, дополняемых пробелами в записях фиксированной длины, функция `pack()` предоставляет компактное решение.

Но чтобы поля дополнялись другими символами вместо пробелов, необходимо воспользоваться функциями `substr()` и `str_pad()`, которые соответственно проверяют, что значения полей не слишком длинные и не слишком короткие. В листинге 1.32 массив записей преобразуется в записи фиксированной длины с полями, дополняемыми точками.

Листинг 1.32. Генерирование полей фиксированной длины без использования pack()

```
$books = array( array('Elmer Gantry', 'Sinclair Lewis', 1927),
                array('The Scarlatti Inheritance','Robert Ludlum', 1971),
                array('The Parsifal Mosaic','William Styron', 1979) );

foreach ($books as $book) {
    $title  = str_pad(substr($book[0], 0, 25), 25, '.');
    $author = str_pad(substr($book[1], 0, 15), 15, '.');
    $year   = str_pad(substr($book[2], 0, 4), 4, '.');
    print "$title$author$year\n";
}
```

См. также

Документация по функциям `pack()` и `str_pad()`. В Рецепте 1.17 форматные строки `pack()` рассматриваются более подробно.

1.14. Разбор данных с полями фиксированной длины

Задача

Требуется разобрать записи с полями фиксированной длины на строки.

Решение

Воспользуйтесь функцией `substr()`, как показано в листинге 1.33.

Листинг 1.33. Разбор записей с полями фиксированной длины с использованием функции substr()

```
$fp = fopen('fixed-width-records.txt','r',true) or die ("can't open file");
while ($s = fgets($fp,1024)) {
    $fields[1] = substr($s,0,25);  // Первое поле:  первые 25 символов
    $fields[2] = substr($s,25,15); // Второе поле: следующие 15 символов
    $fields[3] = substr($s,40,4);  // Третье поле:  следующие 4 символа
    $fields = array_map('rtrim', $fields); // Удаление завершающих пропусков
    // Функция обработки полей
    process_fields($fields);
}
fclose($fp) or die("can't close file");
```

Также можно воспользоваться функцией `unpack()`, как показано в листинге 1.34.

Листинг 1.34. Разбор записей с полями фиксированной длины с использованием функции unpack()

```
function fixed_width_unpack($format_string,$data) {
  $r = array();
```

```
  for ($i = 0, $j = count($data); $i < $j; $i++) {
    $r[$i] = unpack($format_string,$data[$i]);
  }
  return $r;
}
```

Комментарий

В качестве примера данных, в которых для каждого поля выделяется фиксированное количество символов, можно рассмотреть список книг, авторов и дат издания:

```
$booklist=<<<END
Elmer Gantry             Sinclair Lewis 1927
The Scarlatti InheritanceRobert Ludlum  1971
The Parsifal Mosaic      Robert Ludlum  1982
Sophie's Choice          William Styron 1979
END;
```

В каждой строке название книги занимает первые 25 символов, имя автора — следующие 15 символов и год издания — следующие 4 символа. Зная ширину полей, мы можем легко использовать функцию `substr()` для разбора строки в массив по полям:

```
$books = explode("\n",$booklist);
for($i = 0, $j = count($books); $i < $j; $i++) {
  $book_array[$i]['title'] = substr($books[$i],0,25);
  $book_array[$i]['author'] = substr($books[$i],25,15);
  $book_array[$i]['publication_year'] = substr($books[$i],40,4);
}
```

Разбиение `$booklist` в массив позволяет использовать один код цикла независимо от того, работает ли он со строкой или серией строк данных, прочитанных из файла.

Чтобы сделать цикл еще более гибким, можно задать имена и размеры полей в отдельном массиве, который можно передать функции разбора, как показано в функции `fixed_width_substr()` из листинга 1.35.

Листинг 1.35. Функция fixed_width_substr( )

```
function fixed_width_substr($fields,$data) {
  $r = array();
  for ($i = 0, $j = count($data); $i < $j; $i++) {
    $line_pos = 0;
    foreach($fields as $field_name => $field_length) {
      $r[$i][$field_name] = rtrim(substr($data[$i],$line_pos,$field_length));
      $line_pos += $field_length;
    }
  }
  return $r;
}
```

```
$book_fields = array('title' => 25,
                     'author' => 15,
                     'publication_year' => 4);
$book_array = fixed_width_substr($book_fields,$booklist);
```

Переменная `$line_pos` отслеживает начало каждого поля и продвигается на ширину предыдущего поля при перемещении по строке данных. Функция `rtrim()` используется для удаления завершающих пропусков из каждого поля.

Функция `unpack()` может использоваться как замена `substr()` для извлечения полей. Вместо того чтобы задавать имена и ширину полей в ассоциативном массиве, для `unpack()` создается форматная строка. Функция извлечения полей фиксированного размера с использованием `unpack()` похожа на функцию `fixed_width_unpack()`, приведенную в листинге 1.36.

Листинг 1.36. Функция fixed_width_unpack()

```
function fixed_width_unpack($format_string,$data) {
  $r = array();
  for ($i = 0, $j = count($data); $i < $j; $i++) {
    $r[$i] = unpack($format_string,$data[$i]);
  }
  return $r;
}
```

Так как формат `A` для функции `unpack()` обозначает строку, дополненную пробелами, вызывать `rtrim()` для отсечения завершающих пробелов необязательно.

Когда разобранные поля будут сохранены в `$book_array` любой из функций, их можно вывести, например, в виде таблицы HTML:

```
$book_array = fixed_width_unpack('A25title/A15author/A4publication_year',
                                 $books);
print "<table>\n";
// Вывод строки заголовка
print '<tr><td>';
print join('</td><td>',array_keys($book_array[0]));
print "</td></tr>\n";
// Вывод строк таблицы с данными
foreach ($book_array as $row) {
    print '<tr><td>';
    print join('</td><td>',array_values($row));
    print "</td></tr>\n";
}
print "</table>\n";
```

Соединение данных с ограничителем `</td><td>` строит строку таблицы, в которой отсутствует первый тег `<td>` и последний `</td>`. При построении полной строки таблицы до соединенных данных выводится фрагмент `<tr><td>`, а после них — фрагмент `</td></tr>`.

Функции `substr()` и `unpack()` предоставляют эквивалентные возможности, если поля фиксированного размера содержат строки, но функция `unpack()` лучше подходит тогда, когда элементы полей не ограничиваются простыми строками.

Если все поля имеют одинаковый размер, функция `str_split()` обеспечивает удобную сокращенную запись для разбиения входных данных. Она возвращает массив, содержащий фрагменты строки. В листинге 1.37 функция `str_split()` разбивает строку на 32-байтовые блоки.

Листинг 1.37. Разбиение строк функцией str_split( )

```
$fields = str_split($line_of_data,32);
// $fields[0] - байты 0 - 31
// $fields[1] - байты 32 - 63
//  и т. д.
```

См. также

Рецепт 1.17 и сайт PHP с дополнительной информацией о функции `unpack()`; документация по функции `str_split()`; Рецепт 4.8 с описанием `join()`.

1.15. Разбиение строк на фрагменты

Задача

Требуется разбить строку на фрагменты. Допустим, вы хотите последовательно перебрать все строки, введенные пользователем в поле формы `<textarea>`.

Решение

Воспользуйтесь функцией `explode()`, если фрагменты разделяются постоянной строкой:

```
$words = explode(' ','My sentence is not very complicated');
```

Если разделители описываются Perl-совместимым регулярным выражением, воспользуйтесь функцией `preg_split()` :

```
$words = preg_split('/\d\. /','my day: 1. get up 2. get dressed 3. eat toast');
$lines = preg_split('/[\n\r]+/',$_POST['textarea']);
```

Используйте флаг `/i` с функцией `preg_split()`, чтобы поиск разделителей осуществлялся без учета регистра символов:

```
$words = preg_split('/ x /i','31 inches x 22 inches X 9 inches');
```

Комментарий

Простейшее решение основано на использовании `explode()`. Передайте строку-разделитель, разбиваемую строку и необязательный аргумент с количеством возвращаемых элементов:

```
$dwarves = 'dopey,sleepy,happy,grumpy,sneezy,bashful,doc';
$dwarf_array = explode(',',$dwarves);
```

Этот фрагмент создает массив `$dwarf_array` из семи элементов, так что `print_r($dwarf_array)` выводит следующий результат:

```
Array
(
    [0] => dopey
    [1] => sleepy
    [2] => happy
    [3] => grumpy
    [4] => sneezy
    [5] => bashful
    [6] => doc
)
```

Если заданный лимит меньше количества возможных фрагментов, то в последнем фрагменте передается весь остаток:

```
$dwarf_array = explode(',',$dwarves,5);
print_r($dwarf_array);
```

Результат выглядит так:

```
Array
(
    [0] => dopey
    [1] => sleepy
    [2] => happy
    [3] => grumpy
    [4] => sneezy,bashful,doc
)
```

Функция `explode()` интерпретирует разделитель буквально. Если вы укажете запятую и пробел как разделитель, то строка будет разбиваться по запятой, за которой следует пробел, но не по запятой или пробелу.

Функция `preg_split()` обладает большей гибкостью. Вместо строкового литерала она использует в качестве разделителя Perl-совместимое регулярное выражение. С функцией `preg_split()` можно использовать различные расширения регулярных выражений Perl, а также такие приемы, как включение текста разделителя в возвращаемый массив строк:

```
$math = "3 + 2 / 7 - 9";
$stack = preg_split('/ *([+\-\/*]) */',$math,-1,PREG_SPLIT_DELIM_CAPTURE);
print_r($stack);
```

Результат выглядит так:

```
Array
(
    [0] => 3
    [1] => +
    [2] => 2
    [3] => /
    [4] => 7
```

```
    [5] => -
    [6] => 9
)
```

В этом примере разделитель ищет четыре математических оператора (+, -, /, *) с необязательными начальными или конечными пробелами. Флаг `PREG_SPLIT_DELIM_CAPTURE` приказывает `preg_split()` включить совпадения частей регулярного выражения в круглых скобках в возвращаемый массив строк. В скобки заключен только символьный класс математических операторов, так что возвращаемый массив не будет содержать пробелов.

См. также

Регулярные выражения более подробно рассматриваются в главе 23; документация по функциям `explode()` и `preg_split()`.

1.16. Перенос текста по заданной длине строки

Задача

Требуется организовать перенос текста в строке, например вывести текст в тегах `<pre>` и `</pre>`, но так, чтобы он помещался в окне браузера обычного размера.

Решение

Воспользуйтесь функцией `wordwrap()`:

```
$s = "Four score and seven years ago our fathers brought forth on this continent ⏎
a new nation, conceived in liberty and dedicated to the proposition ⏎
that all men are created equal.";

print "<pre>\n".wordwrap($s)."\n</pre>";
```

Результат выглядит так:

```
<pre>
Four score and seven years ago our fathers brought forth on this continent
a new nation, conceived in liberty and dedicated to the proposition that
all men are created equal.
</pre>
```

Комментарий

По умолчанию функция `wordwrap()` переносит текст с длиной строки 75 символов. Необязательный второй аргумент задает другую длину строки:

```
print wordwrap($s,50);
```

Результат выглядит так:

```
Four score and seven years ago our fathers brought
forth on this continent a new nation, conceived in
liberty and dedicated to the proposition that all
men are created equal.
```

Вместо `\n` в качестве разрывов строк могут использоваться и другие символы. Для создания двойных пробелов используйте последовательность `"\n\n"`:

```
print wordwrap($s,50,"\n\n");
```

Результат выглядит так:

```
Four score and seven years ago our fathers brought

forth on this continent a new nation, conceived in

liberty and dedicated to the proposition that all

men are created equal.
```

У функции `wordwrap()` есть необязательный четвертый аргумент, который управляет обработкой слов, длина которых превышает заданную длину строки. Если этот аргумент равен 1, эти слова переносятся. В противном случае они выходят за заданную длину строки:

```
print wordwrap('jabberwocky',5) . "\n";
print wordwrap('jabberwocky',5,"\n",1);
```

Результат выглядит так:

```
jabberwocky
jabbe
rwock
y
```

См. также

Документация по `wordwrap()`.

1.17. Хранение двоичных данных в строках

Задача

Требуется разобрать строку со значениями, закодированными в двоичной структуре, или закодировать значения в строке. Например, вы хотите хранить числа в их двоичном представлении вместо последовательности ASCII-символов.

Решение

Воспользуйтесь функцией `pack()` для сохранения двоичных данных в строке:

```
$packed = pack('S4',1974,106,28225,32725);
```

Для извлечения двоичных данных из строки используется функция `unpack()`:

```
$nums = unpack('S4',$packed);
```

Комментарий

Первый аргумент `pack()` содержит форматную строку, которая описывает способ кодирования данных, передаваемых в остальных аргументах. Форматная строка `S4` приказывает `pack()` сгенерировать четыре коротких 16-разрядных числа без знака с машинным порядком байтов. Если использовать числа 1974, 106, 28 225 и 32 725 на машине с прямым (little-endian) порядком байтов, вызов вернет восемь байтов: 182, 7, 106, 0, 65, 110, 213 и 127. Каждая пара байтов соответствует одному из входных чисел: 7 * 256 + 182 =1974; 0 * 256 + 106 =106; 110 * 256 + 65 = 28 225; 127 * 256 + 213 = 32 725.

Первый аргумент `unpack()` также содержит форматную строку, а во втором передаются декодируемые данные. Для форматной строки `S4` и 8-байтовой последовательности, сгенерированной функцией `pack()`, создается массив исходных чисел с четырьмя элементами. Команда `print_r($nums)` выводит следующий результат:

```
Array
(
    [1] => 1974
    [2] => 106
    [3] => 28225
    [4] => 32725
)
```

При вызове `unpack()` за форматными символами может следовать строка, которая используется в качестве ключа массива. Пример:

```
$nums = unpack('S4num',$packed);
print_r($nums);
```

Результат выглядит так:

```
Array
(
    [num1] => 1974
    [num2] => 106
    [num3] => 28225
    [num4] => 32725
)
```

Форматные символы в аргументе `unpack()` должны разделяться символом `/`:

```
$nums = unpack('S1a/S1b/S1c/S1d',$packed);
print_r($nums);
```

Результат выглядит так:

```
Array
(
    [a] => 1974
    [b] => 106
    [c] => 28225
    [d] => 32725
)
```

В таблице 1.2 перечислены форматные символы, которые могут использоваться с функциями `pack()` и `unpack()`.

Таблица 1.2. Форматные символы функций pack() и unpack()

Форматный символ	Тип данных
a	Строка, дополненная NUL
A	Строка, дополненная пробелами
h	Шестнадцатеричная строка, сначала младший полубайт
H	Шестнадцатеричная строка, сначала старший полубайт
c	signed char
C	unsigned char
s	signed short (16-разрядное, машинный порядок байтов)
S	unsigned short (16-разрядное, машинный порядок байтов)
n	unsigned short (16-разрядное, обратный порядок байтов)
v	unsigned short (16-разрядное, прямой порядок байтов)
i	signed int (машинный размер и порядок байтов)
I	unsigned int (машинный размер и порядок байтов)
l	signed long (32-разрядное, машинный порядок байтов)
L	unsigned long (32-разрядное, машинный порядок байтов)
N	unsigned long (32-разрядное, обратный порядок байтов)
V	unsigned long (32-разрядное, прямой порядок байтов)
f	float (машинный размер и представление)
d	double (машинный размер и представление)
x	Байт NUL
X	Возврат на один байт
@	NUL-заполнение до абсолютной позиции

Для форматных символов a, A, h и H число после форматного символа обозначает длину строки. Например, A25 обозначает строку из 25 символов, дополненную пробелами. Для других форматных символов число обозначает последовательное количество экземпляров соответствующего типа в строке. Знак * используется для обозначения остатка доступных данных.

Функция unpack() может использоваться для преобразования типов данных. В следующем примере массив $ascii заполняется ASCII-кодами всех символов из $s:

```
$s = 'platypus';
$ascii = unpack('c*',$s);
print_r($ascii);
```

Результат выглядит так:

```
Array
(
    [1] => 112
    [2] => 108
    [3] => 97
    [4] => 116
    [5] => 121
    [6] => 112
    [7] => 117
    [8] => 115
)
```

См. также

Документация по функциям pack() и unpack().

1.18. Программа: загрузка файла в формате CSV

Объединение функции header() для изменения типа контента, выводимого вашей программой PHP, с функцией fputcsv() для форматирования данных позволяет передавать браузерам CSV-файлы, которые будут автоматически передаваться электронной таблице (или другому приложению, настроенному в клиентской системе для обработки файлов в формате CSV).

Программа из листинга 1.38 форматирует результаты запроса SQL SELECT в виде данных CSV и предоставляет правильные заголовки для обработки данных браузером.

Листинг 1.38. Загрузка файла в формате CSV

```
$db = new PDO('sqlite:/usr/local/data/sales.db');
$query = $db->query('SELECT region, start, end, amount FROM sales', PDO::FETCH_
```

```
NUM);
$sales_data = $db->fetchAll();

// Открытие файлового дескриптора для fputcsv()
$output = fopen('php://output','w') or die("Can't open php://output");
$total = 0;

// Сообщаем браузеру, что будет передаваться файл CSV
header('Content-Type: application/csv');
header('Content-Disposition: attachment; filename="sales.csv"');

// Вывод заголовка
fputcsv($output,array('Region','Start Date','End Date','Amount'));
// Вывод каждой строки данных и увеличение $total
foreach ($sales_data as $sales_line) {
    fputcsv($output, $sales_line);
    $total += $sales_line[3];
}

// Вывод завершителя и закрытие файлового дескриптора
fputcsv($output,array('All Regions','--','--',$total));
fclose($output) or die("Can't close php://output");
```

Листинг 1.38 отправляет два заголовка, обеспечивающих правильную обработку браузером выходных данных в формате CSV. Первый заголовок `Content-Type` сообщает браузеру, что выводит не разметку HTML, а данные CSV. Второй заголовок, `Content-Disposition`, приказывает браузеру не выводить данные, а попытаться загрузить внешнюю программу для их обработки. Атрибут `filename` этого заголовка определяет имя файла по умолчанию, которое должно использоваться браузером для загружаемого файла.

Если вы захотите создать разные представления для одних данных, объедините код форматирования в одной странице и используйте переменную строки запроса для определения типа форматирования. В листинге 1.39 переменная `format` управляет тем, в каком формате должны возвращаться результаты запроса SQL `SELECT` — таблицы HTML или CSV.

Листинг 1.39. Выбор формата (CSV или HTML)

```
$db = new PDO('sqlite:/usr/local/data/sales.db');
$query = $db->query('SELECT region, start, end, amount FROM sales', PDO::FETCH_
NUM);
$sales_data = $db->fetchAll();

$total = 0;
$column_headers = array('Region','Start Date','End Date','Amount');
// Выбор формата
$format = $_GET['format'] == 'csv' ? 'csv' : 'html';

// Вывод начальных данных, соответствующих формату
if ($format == 'csv') {
    $output = fopen('php://output','w') or die("Can't open php://output");
    header('Content-Type: application/csv');
```

```
    header('Content-Disposition: attachment; filename="sales.csv"');
    fputcsv($output,$column_headers);
 } else {
    echo '<table><tr><th>';
    echo implode('</th><th>', $column_headers);
    echo '</th></tr>';
 }

foreach ($sales_data as $sales_line) {
    // Вывод строки в соответствии с форматом
    if ($format == 'csv') {
        fputcsv($output, $sales_line);
    } else {
        echo '<tr><td>' . implode('</td><td>', $sales_line) . '</td></tr>';
   }
    $total += $sales_line[3];
}
$total_line = array('All Regions','--','--',$total);

// Вывод завершителя, соответствующего формату
if ($format == 'csv') {
    fputcsv($output,$total_line);
    fclose($output) or die("Can't close php://output");
 } else {
    echo '<tr><td>' . implode('</td><td>', $total_line) . '</td></tr>';
    echo '</table>';
 }
```

При выполнении листинга 1.39 с включением в строку запроса параметра `format=csv` программа возвращает данные в формате CSV. При любом другом значении `format` в строке запроса программа возвращает данные в формате HTML. Логика присваивания `$format` легко расширяется на другие выходные форматы, например JSON. Если программа содержит много мест, в которых вы хотите реализовать возможность загрузки одних данных в нескольких форматах, упакуйте код листинга 1.39 в функцию, которая получает массив данных и спецификатор формата и возвращает нужный результат.

2 Числа

2.0. Введение

В повседневной жизни мы легко узнаем числа: 15:00 — текущее время, а $1.29 — стоимость пинты молока... Или число π — отношение длины окружности к диаметру. Числа бывают очень большими, как, например, число Авогадро, которое равно примерно 6×10^{23}. В PHP поддерживаются все эти варианты записи чисел.

Однако в PHP все эти числа делятся на две категории: целые числа и числа с плавающей точкой. Целые числа не имеют дробной части; например, к этой категории относятся –4, 0, 5 и 1975. Числа с плавающей точкой имеют дробную часть, как, например, числа –1,23, 0,0, 3,14159 и 9,9999999999.

К счастью, обычно вам не нужно беспокоиться о различиях между этими категориями, потому что PHP автоматически преобразует целые числа в числа с плавающей точкой, и наоборот. Это позволяет забыть о подробностях, а также означает, что результат выражения 3/2 равен 1,5, а не 1, как в некоторых языках программирования. PHP также автоматически преобразует строки в числа, и наоборот. Например, результат `1+"1"` равен 2.

Впрочем, в отдельных случаях это блаженное неведение может создать проблемы. Во-первых, числа не могут быть бесконечно большими или малыми[1]; наименьшее значение равно $2{,}2e^{-308}$, а наибольшее — около $1{,}8e^{308}$. Для представления больших (или меньших) чисел следует использовать библиотеки BCMath или GMP (Рецепт 2.15).

Далее, числа с плавающей точкой представляют числа не абсолютно точно, а с небольшой погрешностью. В большинстве случаев эта погрешность допустима, но иногда она создает проблемы. Например, человек автоматически преобразует

[1] Вообще говоря, ограничения зависят от платформы, но эти значения достаточно распространены, потому что они происходят от 64-разрядного стандарта IEEE 754.

6 с бесконечной серией девяток в дробной части в 7, но для PHP это всего лишь 6 с серией девяток. Следовательно, если вы запросите у PHP целую часть этого числа, то получите 6, а не 7. По тем же причинам, если цифра в 200-м разряде дробной части является значимой для вас, не используйте формат с плавающей точкой — вам нужны библиотеки BCMath и GMP. Однако в большинстве случаев PHP отлично справляется с числами и позволяет работать с ними по тем же правилам, что и в реальной жизни.

2.1. Проверка значения переменной

Задача

Требуется проверить, содержит ли переменная числовое значение, даже если она имеет строковый тип. Также возможна другая ситуация: убедиться в том, что переменная не только содержит число, но и явно объявлена как числовая.

Решение

Чтобы выяснить, содержит ли переменная число, воспользуйтесь функцией `is_numeric()`:

```
foreach ([5, '5', '05', 12.3, '16.7', 'five', 0xDECAFBAD, '10e200']
          as $maybeNumber) {
    $isItNumeric = is_numeric($maybeNumber);
    $actualType  = gettype($maybeNumber);
    print "Is the $actualType $maybeNumber numeric? ";
    if (is_numeric($maybeNumber)) {
        print "yes";
    } else {
        print "no";
    }
    print "\n";
}
```

Этот пример выводит следующий результат:

```
Is the integer 5 numeric? yes
Is the string 5 numeric? yes
Is the string 05 numeric? yes
Is the double 12.3 numeric? yes
Is the string 16.7 numeric? yes
Is the string five numeric? no
Is the integer 3737844653 numeric? yes
Is the string 10e200 numeric? yes
```

Комментарий

Разнообразие чисел огромно. Нельзя считать, что некое значение является числом просто потому, что оно состоит из символов от 0 до 9. Что делать с точкой (раз-

делителем дробной части) или знаком «минус»? Их нельзя просто включить в набор проверяемых символов, потому что минус может располагаться только в начале числа, а точка может быть только одна. Также не стоит забывать о шестнадцатеричных числах и экспоненциальной (научной) записи.

Чтобы узнать, содержит ли переменная данные, которые либо являются числовыми, либо могут быть преобразованы в число, не обязательно писать собственную функцию — воспользуйтесь функцией `is_numeric()`.

На самом деле между этими двумя случаями существуют различия. Формально целое число 5 и строка 5 с точки зрения PHP различаются. Однако в большинстве случаев эти различия несущественны, поэтому функция `is_numeric()` приносит пользу.

Функция `is_numeric()` правильно разбирает дробные числа (например, `5.1`); с другой стороны, для чисел с разделителями групп разрядов (например, `5,100`) функция `is_numeric()` возвращает `false`.

Чтобы убрать разделители групп разрядов из числа перед вызовом `is_numeric()`, воспользуйтесь функцией `str_replace()`:

```
$number = "5,100";

// Этот вызов is_numeric() возвращает false
$withCommas = is_numeric($number);

// Этот вызов is_numeric() возвращает true
$withoutCommas = is_numeric(str_replace(',', '', $number));
```

Чтобы узнать, относится ли число к конкретному числовому типу, используйте разнообразные взаимосвязанные функции с именами, которые говорят сами за себя: `is_float()` (или `is_double()`, или `is_real()`; это одно и то же) и `is_int()` (или `is_integer()`, или `is_long()`).

Для проверки введенных данных вместо функции `is_numeric()` следует использовать методы, описанные в Рецепте 9.3. В нем также рассказано, как проверяются положительные и отрицательные числа, дробные числа и другие форматы числовых данных.

См. также

Рецепт 9.3 — проверка числовых данных, введенных пользователем; документация по функциям `is_numeric()` и `str_replace()`.

2.2. Сравнение чисел с плавающей точкой

Задача

Требуется проверить, равны ли два числа с плавающей точкой.

Решение

Выберите небольшой интервал («дельта») и проверьте, превышает ли абсолютная величина разности этих двух чисел указанный интервал:

```
$delta = 0.00001;
$a = 1.00000001;
$b = 1.00000000;
if (abs($a - $b) < $delta) {
   print '$a and $b are equal enough.';
}
```

Комментарий

В двоичном представлении чисел с плавающей точкой используется конечное число разрядов мантиссы и экспоненты. При выходе за пределы разрядности происходит переполнение. В результате PHP (а также некоторые другие языки) иногда считают, что два равных числа различны, потому что они различаются в конечных разрядах.

Чтобы избежать подобных проблем, вместо проверки `$a == $b` следует проверить, отличается ли первое число от второго на очень малый интервал (`$delta`). Размер интервала определяется минимальной разностью, начиная с которой два числа считаются разными. Для получения абсолютного значения разности используется функция `abs()`.

См. также

Рецепт 2.3 — округление чисел с плавающей точкой; документация по числам с плавающей точкой в PHP.

2.3. Округление чисел с плавающей точкой

Задача

Требуется округлить число с плавающей точкой до целого числа или до заданного количества разрядов в дробной части.

Решение

Округление числа до ближайшего целого выполняется функцией `round()`:

```
$number = round(2.4);
printf("2.4 rounds to the float %s", $number);
```

Результат выглядит так:

```
2.4 rounds to the float 2
```

Чтобы округление выполнялось в бóльшую сторону, используйте функцию `ceil()`:

```
$number = ceil(2.4);
printf("2.4 rounds up to the float %s", $number);
```

Результат выглядит так:

```
2.4 rounds up to the float 3
```

Округление в меньшую сторону выполняется функцией `floor()`:

```
$number = floor(2.4);
printf("2.4 rounds down to the float %s", $number);
```

Результат выглядит так:

```
2.4 rounds down to the float 2
```

Комментарий

Если число находится ровно посередине между двумя числами, PHP округляет его в направлении от 0:

```
$number = round(2.5);
printf("Rounding a positive number rounds up: %s\n", $number);
$number = round(-2.5);
printf("Rounding a negative number rounds down: %s\n", $number);
```

Результат выглядит так:

```
Rounding a positive number rounds up: 3
Rounding a negative number rounds down: -3
```

Как было сказано в Рецепте 2.2, числа с плавающей точкой не всегда представляют точные значения, что связано с особенностями их хранения в компьютере. Иногда такие погрешности создают путаницу. Значение, которое на первый взгляд имеет дробную часть «0.5», вместо этого может иметь дробную часть «.499999...9» (с длинной последовательностью девяток) или «.500000...1» (с множеством нулей и завершающей единицей).

PHP автоматически включает небольшой «разброс» в вычисления с округлением, так что вам об этом беспокоиться не придется.

Для округления до заданного количества знаков в дробной части функция `round()` может получать необязательный аргумент. Допустим, вы вычисляете суммарную цену товаров в корзине пользователя:

```
$cart = 54.23;
$tax = $cart * .05;
```

```
$total = $cart + $tax;
$final = round($total, 2);

print "Tax calculation uses all the digits it needs: $total, but ";
print "round() trims it to two decimal places: $final";
```

Результат выглядит так:

```
Tax calculation uses all the digits it needs: 56.9415, but round()
trims it to two decimal places: 56.94
```

Для округления числа в меньшую сторону используется функция `floor()`:

```
$number1 = floor(2.1);  // floor(2.1) дает 2.0
$number2 = floor(2.9);  // floor(2.9) тоже дает 2.0
$number3 = floor(-2.1); // floor(-2.1) дает -3.0
$number4 = floor(-2.9); // floor(-2.9) тоже дает 3.0
```

Для округления числа в большую сторону используется функция `ceil()`:

```
$number1 = ceil(2.1);  // ceil(2.1) дает 3.0
$number2 = ceil(2.9);  // ceil(2.9) тоже дает 3.0
$number3 = ceil(-2.1); // ceil(-2.1) дает -2.0
$number4 = ceil(-2.9); // ceil(-2.9) тоже дает 2.0
```

См. также

Рецепт 2.2 — сравнение чисел с плавающей точкой; документация по функциям `ceil()`, `floor()`, `round()` и форматным строкам `printf` (`%s`).

2.4. Работа с последовательностями целых чисел

Задача

Требуется выполнить фрагмент кода для каждого целого числа из диапазона.

Решение

Воспользуйтесь циклом `for`:

```
$start = 3;
$end = 7;
for ($i = $start; $i <= $end; $i++) {
    printf("%d squared is %d\n", $i, $i * $i);
}
```

В качестве приращения можно использовать другие значения, кроме 1. Пример:

```
$start = 3;
$end = 7;
for ($i = $start; $i <= $end; $i += 2) {
    printf("The odd number %d squared is %d\n", $i, $i * $i);
}
```

Если вы хотите сохранить числа для использования, помимо перебора воспользуйтесь методом `range()`:

```
$numbers = range(3, 7);
foreach ($numbers as $n) {
        printf("%d squared is %d\n", $n, $n * $n);
}
foreach ($numbers as $n) {
        printf("%d cubed is %d\n", $n, $n * $n * $n);
}
```

Комментарий

Подобные циклы очень часто используются в программах. Допустим, вы строите график функции и хотите вычислить результаты для набора точек по горизонтальной оси. А может, вы — студент, отсчитывающий время в секундах до конца лекции.

Цикл `for` использует один целочисленный счетчик, а произвольный выбор приращения `$i` позволяет лучше контролировать ход выполнения цикла. Кроме того, значение `$i` можно изменять внутри цикла.

В последнем примере Решения функция `range()` возвращает массив значений от `$start` до `$end`. Преимуществом `range()` является лаконичность, но у этого приема есть и недостаток: большой массив может занимать лишнюю память.

Если вы хотите, чтобы функция `range()` использовала нестандартные приращения, передайте третий аргумент, определяющий величину каждого шага. Например, вызов `range(1, 10, 2)` возвращает массив с элементами 1, 3, 5, 7, 9. Значение `$start` может быть больше `$end`; в этом случае числа, возвращаемые `range()`, упорядочиваются по убыванию.

Функция `range()` также может использоваться для генерирования символьных последовательностей:

```
print_r(range('l', 'p'));
```

Результат выглядит так:

```
Array
(
    [0] => l
    [1] => m
    [2] => n
    [3] => o
    [4] => p
)
```

Учтите, что символьные последовательности, сгенерированные функцией `range()`, состоят из байтов ASCII, так что с многобайтовыми символами Юникода функция работать не будет.

Начиная с PHP 5.5 для работы с сериями можно использовать генераторы — функции, которые вместо вызова `return` для возвращения значения вызывает `yield` (возможно, в цикле). Вызов такой функции может быть применен всюду, где может использоваться массив, и вы работаете с сериями значений, передаваемых ключевому слову `yield`. В следующем примере генератор используется для построения списка квадратов:

```
function squares($start, $stop) {
    if ($start < $stop) {
        for ($i = $start; $i <= $stop; $i++) {
            yield $i => $i * $i;
        }
    }
    else {
        for ($i = $stop; $i >= $start; $i--) {
            yield $i => $i * $i;
        }
    }
}

foreach (squares(3, 15) as $n => $square) {
    printf("%d squared is %d\n", $n, $square);
}
```

PHP продолжает вызывать функцию `squares()`, пока та вызывает `yield`. Ключ и значение, передаваемое `yield`, могут использоваться в `foreach`, как обычный элемент массива.

Генераторы удобны прежде всего тем, что в них можно реализовать произвольное поведение для создания каждого значения (то, что запрограммировано в вашей функции), но значения генерируются по требованию. Вам не нужно резервировать память (или вычислительные ресурсы) для предварительного создания всего массива, как в случае с `range()`, прежде чем начать работать с ним.

См. также

Рецепт 4.3 — инициализация массива диапазоном целых чисел; документация по функции `range()`.

2.5. Генерирование случайных чисел в заданном диапазоне

Задача

Требуется сгенерировать случайное число в заданном диапазоне.

Решение

Воспользуйтесь функцией `mt_rand()`:

```
$lower = 65;
$upper = 97;
// Случайное число в диапазоне от $upper до $lower включительно
$random_number = mt_rand($lower, $upper);
```

Комментарий

Случайные числа пригодятся в разных ситуациях: для вывода случайного изображения на странице, генерирования исходной позиции в игре, выборки случайной записи из базы данных или генерирования уникального идентификатора сеанса. Чтобы сгенерировать случайное число в диапазоне, заданном двумя конечными точками, передайте `mt_rand()` два аргумента: минимальное и максимальное число, которое может быть возвращено в результате вызова. Вызов `mt_rand()` без аргументов возвращает число в диапазоне от 0 до максимального случайного числа, которое может быть получено вызовом `mt_getrandmax()`.

Генерирование действительно случайного числа — непростая задача для компьютера. Компьютеры специализируются на методичном выполнении инструкций; со спонтанностью они справляются значительно хуже. Если вы хотите, чтобы компьютер возвращал случайные числа, придется предоставить ему конкретный набор повторяемых команд; сам факт повторяемости противоречит случайности.

В PHP имеются два генератора случайных чисел — классическая функция `rand()` и улучшенная функция `mt_rand()`. Сокращение «MT» происходит от «Mersenne Twister» — имени французского богослова и математика Марена Мерсенна, «вихря Мерсенна» — разновидности генератора, основанного на свойствах простых «чисел Мерсенна». Функция `mt_rand()` менее предсказуема и работает быстрее, поэтому мы предпочитаем использовать ее вместо `rand()`.

См. также

Рецепт 2.7 — генерирование случайных чисел с неравномерным распределением; Рецепт 1.6 — генерирование случайных строк; документация по функциям `mt_rand()` и `rand()`.

2.6. Генерирование предсказуемых случайных чисел

Задача

Требуется генерировать предсказуемые случайные числа, чтобы гарантировать повторяемость поведения программы.

Решение

Инициализируйте генератор случайных чисел заранее известным значением с использованием `mt_srand()` (или `srand()`):

```
<?php

function pick_color() {
    $colors = array('red','orange','yellow','blue','green','indigo','violet');
    $i = mt_rand(0, count($colors) - 1);
    return $colors[$i];
}
mt_srand(34534);
$first = pick_color();
$second = pick_color();
// Так как mt_srand() передается конкретное значение, каждый раз
// будут выбираться одни и те же цвета: красный (red) и желтый (yellow)
print "$first is red and $second is yellow.";
```

Комментарий

Чтобы получить непредсказуемые случайные числа, поручите инициализацию («раскрутку») счетчика PHP. Инициализация генератора случайных чисел известным значением может быть полезна для получения от генератора предсказуемой серии значений. Например, такая возможность может быть полезна при написании тестов. Если ваш модульный тест проверяет поведение функции, выбирающей случайный элемент из массива, а числа действительно случайны, проверяемое условие будет изменяться при каждом выполнении теста. Но если в начале теста функция `mt_srand()` (или `srand()`) будет вызываться с конкретным значением, при каждом выполнении теста будет генерироваться одна и та же последовательность случайных чисел.

См. также

Документация по функциям `mt_srand()` и `srand()`.

2.7. Генерирование случайных чисел с неравномерным распределением

Задача

Требуется генерировать случайные числа с неравномерным распределением, чтобы некоторые числа появлялись чаще других. Например, серия показов рекламных баннеров должна распределяться пропорционально количеству показов, оставшихся для каждой рекламной кампании.

Решение

Воспользуйтесь функцией `rand_weighted()`, приведенной в листинге 2.1.

Листинг 2.1. rand_weighted()

```
// Функция возвращает случайно выбранный ключ из взвешенного набора
function rand_weighted($numbers) {
    $total = 0;
    foreach ($numbers as $number => $weight) {
        $total += $weight;
        $distribution[$number] = $total;
    }
    $rand = mt_rand(0, $total - 1);
    foreach ($distribution as $number => $weights) {
        if ($rand < $weights) { return $number; }
    }
}
```

Комментарий

Представьте, что вместо массива с количествами оставшихся показов у вас имеется массив, в котором для каждой рекламы количество элементов соответствует количеству оставшихся показов. Вы просто выбираете невзвешенную случайную позицию в массиве, и соответствующая реклама отображается на странице.

При миллионах оставшихся показов этот метод приведет к огромным затратам памяти. Вместо этого вы вычисляете размер массива (суммируя оставшиеся показы), выбираете случайное число из диапазона с размером предполагаемого массива, после чего по массиву определяете, какая реклама соответствует выбранному числу. Пример:

```
$ads = array('ford' => 12234, // Реклама, оставшиеся показы
             'att'  => 33424,
             'ibm'  => 16823);

$ad = rand_weighted($ads);
```

Генераторы PHP 5.5 позволяют выбрать взвешенное случайное число без предварительного построения массива распределения:

```
function incremental_total($numbers) {
    $total = 0;
    foreach ($numbers as $number => $weight) {
        $total += $weight;
        yield $number => $total;
    }
}

// Функция возвращает случайно выбранный ключ из взвешенного набора
function rand_weighted_generator($numbers) {
```

```
    $total = array_sum($numbers);
    $rand = mt_rand(0, $total - 1);
    foreach (incremental_total($numbers) as $number => $weight) {
        if ($rand < $weight) { return $number; }
    }
}
```

См. также

Рецепт 2.5 — генерирование случайных чисел в заданном диапазоне.

2.8. Вычисление логарифмов

Задача

Требуется вычислить логарифм числа.

Решение

Для вычисления натуральных логарифмов (по основанию e) используйте функцию `log()`:

```
// $log == примерно 2.30258
$log = log(10);
```

Для десятичных логарифмов (по основанию 10) используйте функцию `log10()`:

```
// $log10 == 1
$log10 = log10(10);
```

Для вычисления других логарифмов основание передается во втором аргументе `log()`:

```
// log 10 по основанию 2 равен примерно 3.32
$log2 = log(10, 2);
```

Комментарий

Функции `log()` и `log10()` определены только для положительных значений. При вызове для чисел, меньших либо равных 0, эти функции возвращают `NAN` («Not A Number», то есть «не число»).

См. также

Документация по функциям `log()` и `log10()`.

2.9. Вычисление экспоненты

Задача

Требуется возвести число в степень.

Решение

Для возведения числа e в заданную степень используйте функцию `exp()`:

```
// Переменная $exp (e в квадрате) равна примерно 7.389
$exp = exp(2);
```

Для возведения чисел в произвольную степень используется функция `pow()`:

```
// $exp (2^e) == примерно 6.58
$exp = pow( 2, M_E);
// $pow1 (2^10) == 1024
$pow1 = pow( 2, 10);
// $pow2 (2^-2) == 0.25
$pow2 = pow( 2, -2);
// $pow3 (2^2.5) == примерно 5.656
$pow3 = pow( 2, 2.5);
// $pow4 ( (-2)^10 ) == 1024
$pow4 = pow(-2, 10);
// is_nan($pow5) возвращает true, потому что экспонента
// отрицательного числа 2 с дробным показателем
// не является вещественным числом.
$pow5 = pow(-2, -2.5);
```

Комментарий

Встроенная константа `M_E` является приближенным представлением числа e (приблизительно 2,7182818284590452354). Таким образом, вызовы `exp($n)` и `pow(M_E,$n)` идентичны.

Числа, создаваемые функциями `exp()` и `pow()`, могут быть очень большими; если ваши потребности выходят за пределы максимального размера PHP (почти $1.8e^{308}$), обращайтесь к Рецепту 2.15 за информацией об использовании функций с произвольной точностью. Для функций `exp()` и `pow()` PHP возвращает `INF` (бесконечность), если результат слишком велик, или `NAN` («не число») в случае ошибки.

См. также

Документация по функциям `pow()`, `exp()`, описание стандартных математических констант.

2.10. Форматирование чисел

Задача

Требуется вывести число с разделителями групп разрядов и дробной части. Например, вы хотите вывести количество посетителей страницы или процент людей, проголосовавших в опросе.

Решение

Если в качестве разделителей групп разрядов и дробной части всегда используются конкретные символы, воспользуйтесь функцией `number_format()`:

```
$number = 1234.56;
// $formatted1 содержит "1,235" - 1234.56 округляется в большую сторону,
// а символ , является разделителем групп разрядов
$formatted1 = number_format($number);

// Второй аргумент задает количество знаков в дробной части.
// $formatted2 содержит 1,234.56
$formatted2 = number_format($number, 2);

// Третий аргумент задает символ разделителя дробной части.
// Четвертый аргумент задает разделитель групп разрядов.
// $formatted3 is 1.234,56
$formatted3 = number_format($number, 2, ",", ".");
```

Чтобы отформатировать строку по стандартам конкретного локального контекста, воспользуйтесь классом `NumberFormatter`:

```
$number = '1234.56';

// $formatted1 содержит 1,234.56
$usa = new NumberFormatter("en-US", NumberFormatter::DEFAULT_STYLE);
$formatted1 = $usa->format($number);

// $formatted2 содержит 1 234,56
// Обратите внимание: между 1 и 2 находится неразрывный пробел (\u00A0)
$france = new NumberFormatter("fr-FR", NumberFormatter::DEFAULT_STYLE);
$formatted2 = $france->format($number);
```

Комментарий

Функция `number_format()` форматирует число с разделителями групп разрядов и дробной части. По умолчанию число округляется до ближайшего целого. Если вы хотите сохранить все число, но не знаете заранее, сколько цифр содержит дробная часть, используйте следующий прием:

```
$number = 31415.92653; // ваше число
list($int, $dec) = explode('.', $number);
```

```
// $formatted содержит 31,415.92653
$formatted = number_format($number, strlen($dec));
```

Класс `NumberFormatter`, часть расширения `intl`, использует обширные правила форматирования из библиотеки ICU, чтобы предоставить простой и мощный механизм форматирования чисел по правилам любой страны мира. Возможны даже такие хитроумные трюки, как преобразование числа в текстовую запись:

```
$number = '1234.56';
$france = new NumberFormatter("fr-FR", NumberFormatter::SPELLOUT);
// $formatted содержит "mille-deux-cent-trente-quatre virgule cinq six"
$formatted = $france->format($number);
```

В Рецепте 19.4 класс `NumberFormatter` обсуждается более подробно.

См. также

Глава 19 — интернационализация и локализация; документация по функциям `number_format()` и `NumberFormatter`.

2.11. Форматирование денежных сумм

Задача

Требуется вывести имеющееся число с разделителями групп разрядов и дробной части по стандартам денежных сумм. Например, вы хотите вывести цены товаров в корзине покупателя.

Решение

Воспользуйтесь классом `NumberFormatter` со стилем `NumberFormatter::CURRENCY`:

```
$number = 1234.56;
// В США используются знаки $ , и .
// $formatted1 содержит $1,234.56
$usa = new NumberFormatter("en-US", NumberFormatter::CURRENCY);
$formatted1 = $usa->format($number);

// Во Франции используются знаки , и €
// $formatted2 содержит 1 234,56 €
$france = new NumberFormatter("fr-FR", NumberFormatter::CURRENCY);
$formatted2 = $france->format($number);
```

Комментарий

Со стилем `NumberFormatter::CURRENCY` при форматировании в число вставляется знак денежной единицы, разделители групп разрядов и дробной части для ло-

кального контекста, используемого при создании экземпляра объекта `NumberFormatter`. Предполагается, что в денежной сумме используется денежная единица, связанная с локальным контекстом, — доллары США для локального контекста `en-US`, евро для контекста `fr-FR` и т. п.

Чтобы получить правильный формат для денежной единицы, отличной от той, что связана с локальным контекстом, используйте метод `formatCurrency()`. Во втором аргументе передается используемая денежная единица. Как, например, отформатировать сумму в евро в локальном контексте США?

```
$number = 1234.56;

// В США для сумм в евро используются знаки € , и .
// $formatted содержит €1,234.56
$usa = new NumberFormatter("en-US", NumberFormatter::CURRENCY);
$formatted = $usa->formatCurrency($number, 'EUR');
```

В стандарте ISO-4217 перечислены трехбуквенные коды различных денежных единиц.

В Рецепте 19.5 использование `NumberFormatter` для форматирования денежных сумм рассматривается более подробно.

См. также

Глава 19 — интернационализация и локализация; документация по кодам денежных единиц ISO-4217 и `NumberFormatter`.

2.12. Вывод формы множественного числа

Задача

Требуется правильно образовать множественное число на основании значения переменной. Например, возвращаемый текст должен зависеть от количества совпадений, обнаруженных при поиске.

Решение

Воспользуйтесь условным выражением:

```
$number = 4;
print "Your search returned $number " . ($number == 1 ? 'hit' : 'hits') . '.';
```

Результат выглядит так:

```
Your search returned 4 hits.
```

Комментарий

Другое возможное решение — использовать одну функцию для хранения всех форм множественного числа, как показано в функции `may_pluralize()` в листинге 2.2.

Листинг 2.2. may_pluralize()

```
function may_pluralize($singular_word, $amount_of) {

    // Массив специальных форм множественного числа
    $plurals = array(
        'fish' => 'fish',
        'person' => 'people',
    );

    // Форма единственного числа
    if (1 == $amount_of) {
        return $singular_word;
    }

    // Больше одного экземпляра, специальное множественное число
    if (isset($plurals[$singular_word])) {
        return $plurals[$singular_word];
    }
    // Больше одного экземпляра, множественное число строится
    // по стандартным правилам: в конец слова добавляется 's'
    return $singular_word . 's';
}
```

Несколько примеров:

```
$number_of_fish = 1;
// $out1 содержит "I ate 1 fish."
$out1 = "I ate $number_of_fish " . may_pluralize('fish', $number_of_fish) . '.';

$number_of_people = 4;
// $out2 содержит "Soylent Green is people!"
$out2 = 'Soylent Green is ' . may_pluralize('person', $number_of_people) . '!';
```

Если вы собираетесь использовать несколько множественных форм в своем коде, такая функция, как `may_pluralize()`, упростит его чтение. Чтобы использовать эту функцию, передайте `may_pluralize()` форму единственного числа в первом аргументе и количество экземпляров во втором. Внутри функции определяется большой массив `$plurals`, в котором хранятся специальные формы множественного числа. Если значение `$amount` равно 1, функция возвращает исходное слово. Если значение больше 1, возвращается специальная форма, если она присутствует в массиве. По умолчанию множественное число образуется добавлением в конец слова символа «s».

Функция `may_pluralize()` в приведенном виде реализует правила создания множественного числа для английского языка (США). Разумеется, в других языках используются другие правила. Если ваше приложение выводит данные только на одном языке, разумно создать функцию-аналог `may_pluralize()` с логикой,

относящейся к этому языку. Если приложение должно выводить данные на многих языках, потребуется более полное решение. Эта возможность рассматривается в главе 19.

См. также

Рецепт 19.2 — образование формы множественного числа в разных локальных контекстах.

2.13. Вычисление тригонометрических функций

Задача

Требуется получить значение тригонометрической функции: синус, косинус, тангенс и т. д.

Решение

В PHP имеется встроенная поддержка многих тригонометрических функций: `sin()`, `cos()` и `tan()`:

```
// Косинус 2 pi равен 1, $result = 1
$result = cos(2 * M_PI);
```

Также доступны обратные тригонометрические функции: `asin()`, `acos()` и `atan()`:

```
// Арктангенс pi/4 равен приблизительно 0.665773
$result = atan(M_PI / 4);
```

Комментарий

Углы для этих функций должны задаваться в радианах, а не в градусах (если это обстоятельство создает проблемы, обращайтесь к Рецепту 2.14).

Функция `atan2()` получает две переменные, `$x` и `$y`, и вычисляет `atan($x/$y)`. При этом всегда возвращается правильный знак, потому что при определении квадранта результата используются оба параметра.

Секанс, косеканс и котангенс легко вычисляются вручную по значениям `sin()`, `cos()` и `tan()`:

```
$n = 0.707;
// Секанс 0.707 равен приблизительно 1.53951
$secant    = 1 / sin($n);

// Косеканс 0.707 равен приблизительно 1.31524
```

```
$cosecant  = 1 / cos($n);

// Котангенс 0.707 равен приблизительно 1.17051
$cotangent = 1 / tan($n);
```

Также можно использовать гиперболические функции: `sinh()`, `cosh()` и `tanh()`, и, разумеется, `asinh()`, `acosh()` и `atanh()`. Впрочем, обратные функции не поддерживаются в PHP для Windows до версии 5.3.0.

См. также

Рецепт 2.14 — выполнение тригонометрических операций в градусах вместо радианов; документация по функциям `sin()`, `cos()`, `tan()`, `asin()`, `acos()`, `atan()` и `atan2()`.

2.14. Выполнение тригонометрических операций в градусах

Задача

Требуется использовать тригонометрические функции со значениями углов, заданными в градусах.

Решение

Используйте функции `deg2rad()` и `rad2deg()` с входными и выходными данными:

```
$degree = 90;
// Косинус 90 градусов равен 0
$cosine = cos(deg2rad($degree));
```

Комментарий

По определению 360 градусов равны 2π радиан, так что преобразования легко выполняются вручную. Впрочем, функции используют встроенное значение π, поэтому они гарантируют высокую точность ответа. Для использования числа в других вычислениях применяйте константу `M_PI`, равную 3,14159265358979323846.

Встроенная поддержка градов в PHP отсутствует. Это сознательное решение, а не ошибка.

См. также

Рецепт 2.13 — основы использования тригонометрических функций; документация по функциям `deg2rad()` и `rad2deg()`.

2.15. Работа с очень большими или очень малыми числами

Задача

Требуется работать с числами, слишком большими (или слишком малыми) для встроенного в PHP формата с плавающей точкой.

Решение

Воспользуйтесь библиотекой BCMath или GMP.

Использование BCMath:

```
// $sum = "9999999999999999"
$sum = bcadd('1234567812345678', '8765432187654321');
```

Использование GMP:

```
$sum = gmp_add('1234567812345678', '8765432187654321');
// $sum - ресурс GMP, а не строка; для преобразования
// используется функция gmp_strval()
print gmp_strval($sum); // Выводится 9999999999999999
```

Комментарий

Библиотека BCMath очень проста в использовании. Числа передаются в виде строк, а функция возвращает сумму (разность, произведение и т. д.) в виде строки. Впрочем, диапазон операций, применимых к числам с использованием BCMath, ограничивается базовой арифметикой.

Другой вариант — библиотека GMP. Хотя многие функции из семейства GMP получают в аргументах целые числа и строки, числа обычно передаются в виде *ресурсов*, которые фактически представляют собой указатели на внутренние представления чисел. Таким образом, в отличие от функций BCMath, возвращающих строки, функции GMP возвращают только ресурсы. Ресурс передается любой функции GMP и работает как число.

Единственный недостаток заключается в том, что для просмотра или использования ресурса функцией, не входящей в библиотеку GMP, вам придется явно преобразовать его функцией `gmp_strval()` или `gmp_intval()`.

Функции GMP достаточно свободно относятся к получаемым данным; листинг 2.3 наглядно доказывает это.

Листинг 2.3. Суммирование чисел с использованием GMP

```
$four = gmp_add(2, 2);            // Можно передавать числа
$eight = gmp_add('4', '4');       // или строки
$twelve = gmp_add($four, $eight); // или ресурсы GMP
```

Возможности GMP отнюдь не ограничиваются сложением: библиотека позволяет возводить числа в степень, очень быстро вычислять большие факториалы, находить наибольший общий делитель и выполнять другие математические операции, как показано в листинге 2.4.

Листинг 2.4. Выполнение нетривиальных математических операций с использованием GMP

```
// Возведение числа в степень
$pow = gmp_pow(2, 10);

// Очень быстрое вычисление больших факториалов
$factorial = gmp_fact(20);

// Поиск наибольшего общего делителя
$gcd = gmp_gcd(123, 456);

// Другие экзотические операции
$legendre = gmp_legendre(1, 7);
```

Не следует полагать, что библиотеки BCMath и GMP всегда доступны во всех конфигурациях PHP. Библиотека BCMath поставляется с PHP, поэтому, скорее всего, она будет доступна. Однако GMP в поставку PHP не входит, поэтому вам придется загрузить ее, установить и приказать PHP использовать ее в процессе настройки. Чтобы узнать, можно ли использовать BCMath и GMP, проверьте значения `function_defined('bcadd')` и `function_defined('gmp_init')`.

Другая альтернатива для выполнения математических операций с большой точностью — библиотека PECL big_int. Пример ее использования продемонстрирован в листинге 2.5.

Листинг 2.5. Суммирование чисел с использованием big_int

```
$two  = bi_from_str('2');
$four = bi_add($two, $two);
// Использование bi_to_str() для получения строк из ресурсов big_int
print bi_to_str($four); // Выводит 4
// Очень быстрое вычисление больших факториалов
$factorial = bi_fact(20);
```

Библиотека работает быстрее BCMath и почти не уступает GMP по мощи. Но если GMP лицензируется на условиях LGPL, big_int распространяется по лицензии в стиле BSD.

См. также

Документация по BCMath, big_int и GMP.

2.16. Преобразование между системами счисления

Задача

Требуется преобразовать число из одной системы счисления в другую.

Решение

Воспользуйтесь функцией `base_convert()`:

```
// Шестнадцатеричное число (по основанию 16)
$hex = 'a1';
// Преобразование из шестнадцатеричной системы счисления в десятичную.
// $decimal содержит '161'
$decimal = base_convert($hex, 16, 10);
```

Комментарий

Функция `base_convert()` получает строку, представляющую число в одной системе счисления, и создает строку с ее правильным представлением в другой системе. Она работает с любыми основаниями систем счисления от 2 до 36 включительно (для оснований более 10 в качестве дополнительных цифр используются буквы от a до z). В первом аргументе передается преобразуемое число, за ним следуют основание текущей системы и основание новой системы счисления.

Также существуют специализированные функции для преобразования из десятичной системы счисления в другие часто используемые системы с основаниями 2, 8 и 16, и обратно. Это функции `bindec()` и `decbin()`, `octdec()` и `decoct()`, `hexdec()` и `dechex()`:

```
// Переход от основания 2 к основанию 10
// $a = 27
$a = bindec(11011);
// Переход от основания 8 к основанию 10
// $b = 27
$b = octdec(33);
// Переход от основания 16 к основанию 10
// $c = 27
$c = hexdec('1b');
// Переход от основания 10 к основанию 2
// $d = '11011'
$d = decbin(27);
// $e = '33'
$e = decoct(27);
// $f = '1b'
$f = dechex(27);
```

Обратите внимание: специализированные функции, выполняющие преобразование к десятичной системе, возвращают целые числа. Функции, выполняющие преобразование из десятичной системы, возвращают строки.

Другое возможное решение основано на использовании семейства функций `printf()`, позволяющих преобразовывать десятичные числа в двоичные, восьмеричные и шестнадцатеричные, с расширенными возможностями форматирования: выводом начальных нулей и выбором между символами верхнего и нижнего регистра для шестнадцатеричных чисел.

Например, для вывода цветовых значений HTML используется спецификатор формата %02X:

```
$red = 0;
$green = 102;
$blue = 204;
// $color содержит '#0066CC'
$color = sprintf('#%02X%02X%02X', $red, $green, $blue);
```

См. также

Документация по возможностям форматирования `base_convert()` и `sprintf()`.

2.17. Математические операции в других системах счисления

Задача

Требуется выполнить математические операции с числами, отформатированными не в десятичной, а в восьмеричной или шестнадцатеричной системе счисления. Например, вы хотите вычислить безопасные веб-цвета в шестнадцатеричном представлении.

Решение

Поставьте перед числом префикс, который сообщает PHP, что число записано не по основанию 10. Префикс `0b` обозначает запись в двоичной системе счисления (основание 2), префикс `0` — восьмеричную (основание 8), а префикс `0x` — шестнадцатеричную (основание 16). Если `$a = 100`, `$b = 0144`, `$c = 0x64`, и `$d = 0b1100100`, PHP считает, что числа `$a`, `$b`, `$c` и `$d` равны.

Вот как выполнить отсчет от 1 до 15 (дес.) в шестнадцатеричной записи:

```
for ($i = 0x1; $i < 0x10; $i++) {
    print "$i\n";
}
```

Комментарий

Даже если использовать в цикле `for` числа, отформатированные в шестнадцатеричной системе, по умолчанию все числа выводятся в десятичном виде. Иначе говоря, код в Решении не выведет ..., 8, 9, a, b, ... Для печати в шестнадцатеричном виде следует использовать один из методов, перечисленных в Рецепте 2.16. Пример:

```
for ($i = 0x1; $i < 0x10; $i++) { print dechex($i) . "\n"; }
```

Как правило, в вычислении проще использовать десятичную систему. Тем не менее в некоторых ситуациях логичнее переключиться на другое основание, например при выполнении байтовых операций. Популярный алгоритм хеширования «Times 33», разработанный Дэном Бернстайном (Dan Bernstein), предоставляет удобный и быстрый способ хеширования строки произвольной длины в целочисленное значение. Начните с «волшебного числа» 5381, которое определяет исходное значение хеш-кода. Затем для каждого байта в хешируемой строке к хеш-коду прибавляется этот байт и предыдущее значение хеша, умноженное на 32. Если напрямую перевести это описание на PHP, получится код следующего вида:

```
function times_33_hash($str) {
   $h = 5381;
   for ($i = 0, $j = strlen($str); $i < $j; $i++) {
       // Сдвиг $h влево на 5 разрядов - простой способ умножения на 32
       $h += ($h << 5) + ord($str[$i]);
   }
   return $h;
}
```

Этот код не является полностью правильным. Иногда он выдает странные результаты, например вызов `times_33_hash("Once, I ate a papaya.")` возвращает не целое число, а число с плавающей точкой, причем очень-очень большое (приблизительно $2{,}28375 \times 10^{19}$). Многократные умножения и сложения, повторяемые для каждого байта в строке, приводят к переполнению целочисленного представления PHP, поэтому в дело вмешается автоматическое преобразование к типу с плавающей точкой (с потерей точности). Чтобы исправить проблему, достаточно объединить логической операцией ИЛИ значение хеша с маской значимых битов, которые должны остаться в хеше. В шестнадцатеричной системе эти значимые биты выражаются намного нагляднее, чем в десятичной. Например, если в хеш-код должны войти 32 бита, добавьте в цикл операцию применения маски:

```
function times_33_hash($str) {
   $h = 5381;
   for ($i = 0, $j = strlen($str); $i < $j; $i++) {
       // Сдвиг $h влево на 5 разрядов - простой способ умножения на 32
       $h += ($h << 5) + ord($str[$i]);
       // Остаются только младшие 32 бита $h
       $h = $h & 0xFFFFFFFF;
   }
   return $h;
}
```

Каждая шестнадцатеричная цифра F представляет 4 бита, так что последовательность из восьми F дает 32-разрядную маску. Вместо 0xFFFFFFFF в коде можно использовать десятичную маску 4294967295, но смысл этого значения далеко не очевиден.

Восьмеричные и шестнадцатеричные числовые выражения поддерживались в PHP в течение многих версий, но префикс 0b для двоичных чисел появился только в PHP 5.4.

См. также

Рецепт 2.16 — дополнительная информация о преобразовании между системами счисления; сообщение Дэна Бернстайна в группе `comp.lang.c` о хешировании «Times 33».

2.18. Определение расстояния между двумя географическими точками

Задача

Требуется определить расстояние между двумя точками на земной поверхности.

Решение

Используйте функцию `sphere_distance()`, как показано в листинге 2.6.

Листинг 2.6. Определение расстояния между двумя точками

```
function sphere_distance($lat1, $lon1, $lat2, $lon2, $radius = 6378.135) {
    $rad = doubleval(M_PI/180.0);

    $lat1 = doubleval($lat1) * $rad;
    $lon1 = doubleval($lon1) * $rad;
    $lat2 = doubleval($lat2) * $rad;
    $lon2 = doubleval($lon2) * $rad;

    $theta = $lon2 - $lon1;
    $dist = acos(sin($lat1) * sin($lat2) +
                 cos($lat1) * cos($lat2) *
                 cos($theta));
    if ($dist < 0) { $dist += M_PI; }
    // По умолчанию используется экваториальный радиус Земли в километрах
    return $dist = $dist * $radius;
}

// Нью-Йорк (10040)
```

```
$lat1 = 40.858704;
$lon1 = -73.928532;

// Сан-Франциско (94144)
$lat2 = 37.758434;
$lon2 = -122.435126;

$dist = sphere_distance($lat1, $lon1, $lat2, $lon2);

// От Нью-Йорка до Сан-Франциско около 2570 миль.
// $formatted содержит 2570.18
$formatted = sprintf("%.2f", $dist * 0.621); // Форматирование
                                             // и преобразование в мили
```

Комментарий

Так как поверхность Земли не плоская, вычислить точное расстояние между двумя точками по теореме Пифагора не удастся. Вместо этого приходится использовать алгоритм дуги большого круга, как сделано в функции `sphere_distance()`.

В первых четырех аргументах передаются географические координаты двух точек. Сначала передается широта и долгота отправной точки, а затем широта и долгота конечной точки. Возвращаемое значение определяет расстояние между ними в километрах.

Код в листинге 2.6 вычисляет расстояние между Нью-Йорком и Сан-Франциско, преобразует расстояние в мили и форматирует значение до двух знаков в дробной части.

Поскольку Земля не является идеальной сферой, вычисления получаются приближенными и могут содержать погрешность до 0,5%.

В необязательном пятом аргументе `sphere_distance()` может передаваться радиус сферы. Это позволяет, например, вычислить расстояние между двумя точками на Марсе:

```
$martian_radius = 3397;
$dist = sphere_distance($lat1, $lon1, $lat2, $lon2, $martian_radius);
$formatted = sprintf("%.2f", $dist * 0.621); // Форматирование
                                             // и преобразование в мили
```

См. также

Рецепт 2.13 — тригонометрические функции; Википедия — статья о радиусе Земли.

3 Дата и время

3.0. Введение

На первый взгляд задача отображения и работы с датами кажется достаточно элементарной, но она усложняется в зависимости от разнообразия потребностей пользователей. Ваши пользователи живут в разных часовых поясах? Скорее всего да, если только вы не строите интрасеть или сайт с очень узкой географической аудиторией. Ваших пользователей пугают метки вида «2015-07-20» и они предпочитают более знакомые представления вида «Saturday July 20, 2015 (2:56 P.M.)»? Вычислить количество часов между 10:00 и 19:00 сегодняшнего дня несложно. А как насчет интервала между 15:00 сегодняшнего дня и полуднем первого дня следующего месяца? Вычисление разностей дат рассматривается в Рецептах 3.5 и 3.6.

Вычисления и манипуляции с датами еще больше усложняются введением летнего времени (DST, Daylight Saving Time). Из-за действия летнего времени появляются несуществующие моменты времени (на большей части территории США от 2:00 до 3:00 в один день весной) и время, существующее дважды (на большей части территории США от 1:00 до 2:00 в один день осенью). Одни пользователи могут жить в местах, поддерживающих летнее время, другие — нет. В Рецепте 3.10 рассматриваются способы работы с часовыми поясами и летним временем.

Работа с временем в программах существенно упрощается благодаря двум соглашениям. Во-первых, во внутреннем представлении программы работают с всемирным скоординированным временем (UTC, Coordinated Universal Time — также используется сокращение GMT от «Greenwich Mean Time») — своего рода «патриархом» семейства часовых поясов без летнего времени. Этот часовой пояс находится на долготе 0 градусов, а все остальные часовые пояса выражаются в смещениях от него (положительных или отрицательных). Во-вторых, время рассматривается не как массив значений месяца, дня, года, минут, секунд и т. д.,

а как количество секунд, прошедших от начала *эпохи* Unix: полночи 1 января 1970 года (конечно, по шкале UTC). Такое представление существенно упрощает вычисление интервалов, и в PHP существует много функций, помогающих легко переходить между количеством секунд от начала эпохи (временной меткой) и представлением, понятным для человека.

Функция `mktime()` создает временные метки из заданного набора компонентов, а функция `date()` для заданной временной метки возвращает отформатированную строку. В листинге 3.1 эти функции используются для определения дня недели, на который пришелся Новый год в 1986 году.

Листинг 3.1. Использование функций mktime() и date()

```
$stamp = mktime(0,0,0,1,1,1986);
print date('l',$stamp);
```

Результат выполнения листинга 3.1:

```
Wednesday
```

В листинге 3.1 функция `mktime()` возвращает временную метку для полуночи 1 января 1986 года. Форматный символ `l` приказывает функции `date()` вернуть полное название дня недели, соответствующего заданной временной метке. Другие форматные символы, поддерживаемые `date()`, перечислены в Рецепте 3.4.

Чтобы упростить обработку даты и времени в коде, присвойте конфигурационной переменной `date.timezone` соответствующий часовой пояс (или вызовите функцию `date_default_timezone_set()` перед выполнением каких-либо операций с временем или датой). Чтобы в вычислениях с датой всегда использовалось время UTC, присвойте `date.timezone` значение `UTC`. Далее, как объясняется в Рецепте 3.4, можно организовать вывод времени и даты в формате, соответствующем часовому поясу и местонахождению пользователя.

В этой книге термином «временная метка» обозначается количество секунд от начала эпохи Unix. Под термином «компоненты даты» следует понимать массив или группу компонентов времени и даты — таких как день, месяц, год, час, минута и секунда. Термин «отформатированная строка времени» (или «отформатированная строка даты» и т. п.) обозначает строку, содержащую сгруппированные компоненты времени и даты, например «2002-03-12», «Wednesday, 11:23 A.M.» или «February 25».

Используя временные метки для внутреннего представления времени, вы избегаете всех проблем, связанных с «проблемой 2000 года», потому что разность между 946702799 (31 декабря 1999 г., 23:59:59 UTC) и 946702800 (1 января 2000 г., 00:00:00 UTC) интерпретируется так же, как разность между любыми другими временными метками. Впрочем, приходится учитывать потенциальную «проблему 2038 года». Момент времени 3:14:07 19 января 2038 года (UTC) наступит через 2147483647 секунд от полуночи 1 января 1970 года. Чем же так примечательно число 2147483647? Это $2^{31}-1$, наибольшее целое значение при использовании 32-разрядного представления знаковых целых (32-й разряд используется для знака).

Функции PHP, использующие встроенную библиотеку работы с временем (такие, как `date()`, `mktime()` и методы класса `DateTime`), хранят временные метки в виде 64-разрядных целых чисел. Диапазон представимых дат расширяется до 600 миллиардов лет; скорее всего, для ваших вычислений этого будет достаточно. По этой причине (а также для простоты) в этой главе для работы с временем и датой используются эти функции — вместо таких функций, как `strftime()` и `gmstrftime()`. Последние зависят от системных вызовов, которые могут не обеспечивать нужной функциональности.

3.1. Определение текущей даты и времени

Задача

Требуется определить текущую дату или время.

Решение

Воспользуйтесь функцией `date()` и передайте ей форматную строку, как показано в листинге 3.2.

Листинг 3.2. Определение текущей даты и времени

```
print date('r');
```

Конечно, вывод зависит от времени и даты запуска, но результат выполнения листинга 3.2 выглядит примерно так:

```
Fri, 01 Feb 2013 14:23:33 -0500
```

Также можно воспользоваться объектом `DateTime`. Его метод `format()` работает так же, как функция `date()`:

```
$when = new DateTime();
print $when->format('r');
```

Если вас интересуют отдельные компоненты даты/времени, используйте функцию `getdate()` или `localtime()`. Листинг 3.3 показывает, как работают эти функции.

Листинг 3.3. Определение компонентов времени

```
$now_1 = getdate();
$now_2 = localtime();
print "{$now_1['hours']}:{$now_1['minutes']}:{$now_1['seconds']}\n";
print "$now_2[2]:$now_2[1]:$now_2[0]";
```

Результат выполнения листинга 3.3 выглядит примерно так:

```
18:23:45
18:23:45
```

Комментарий

Функция `date()` (и объект `DateTime`) может генерировать разнообразные строки отформатированного времени и даты. Возможности форматирования более подробно рассматриваются в Рецепте 3.4. С другой стороны, функции `localtime()` и `getdate()` возвращают массивы, элементы которых представляют компоненты заданной даты или времени.

Ассоциативный массив `getdate()` возвращает пары «ключ/значение», перечисленные в таблице 3.1.

Таблица 3.1. Массив, возвращаемый функцией getdate()

Ключ	Значение
seconds	Секунды
minutes	Минуты
hours	Часы
mday	День месяца
wday	День недели в числовом представлении (воскресенье — 0, суббота — 6)
mon	Месяц в числовом представлении
year	Год (4 цифры)
yday	День года в числовом представлении (например, 299)
weekday	День недели в текстовом представлении (например, «Friday»)
month	Месяц в текстовом представлении (например, «January»)
0	Секунды от начала эпохи (результат, возвращаемый функцией time())

В листинге 3.4 показано, как использовать функцию `getdate()` для вывода месяца, дня и года.

Листинг 3.4. Определение месяца, дня и года

```
$a = getdate();
printf('%s %d, %d',$a['month'],$a['mday'],$a['year']);
```

Листинг 3.4 выводит следующий результат:

```
February 4, 2013
```

Передайте в аргументе `getdate()` временную метку, чтобы возвращаемый массив содержал значения местного времени для этой метки. Пример вывода месяца, дня и года для временной метки 163727100 показан в листинге 3.5.

Листинг 3.5. Функция getdate() с временной меткой

```
$a = getdate(163727100);
printf('%s %d, %d',$a['month'],$a['mday'],$a['year']);
```

Код листинга 3.5 выводит следующий результат:

```
March 10, 1975
```

Функция `localtime()` также возвращает массив с компонентами даты и времени. Она тоже получает временную метку в первом необязательном аргументе и флаг во втором необязательном аргументе. Если второй аргумент истинен, то `localtime()` возвращает ассоциативный массив вместо массива с числовыми индексами. Ключи массива соответствуют полям структуры `tm_struct`, возвращаемой функцией C `localtime()`; список приведен в таблице 3.2.

Таблица 3.2. Массив, возвращаемый функцией localtime()

Позиция	Ключ	Значение
0	tm_sec	Секунды
1	tm_min	Минуты
2	tm_hour	Часы
3	tm_mday	День месяца
4	tm_mon	Месяц года (январь — 0)
5	tm_year	Год (считая с 1990 года)
6	tm_wday	День недели (воскресенье — 0)
7	tm_yday	День года
8	tm_isdst	Признак действия летнего времени

Листинг 3.6 демонстрирует использование функции `localtime()` для вывода текущей даты в формате *месяц/день/год*.

Листинг 3.6. Использование функции localtime()

```
$a = localtime();
$a[4] += 1;
$a[5] += 1900;
print "$a[4]/$a[3]/$a[5]";
```

Код листинга 3.6 выводит следующий результат:

```
2/4/2013
```

Месяц увеличивается на 1 перед выводом, потому что `localtime()` начинает отсчет месяцев с 0 (январь). Если текущим месяцем является январь, мы хотим, чтобы выводился номер 1. По тем же причинам год увеличивается на 1900, так как `localtime()` начинает отсчет лет с 0 (для года 1900).

Функции `getdate()` и `localtime()` используют одну внутреннюю реализацию для генерирования возвращаемых компонентов даты и времени. Они отличаются только форматом возвращаемых массивов и составом возвращаемых данных.

(Например, функция `localtime()` включает признак действия летнего времени для заданного момента.)

Часовой пояс, используемый функциями `getdate()` и `localtime()` для вычислений, задается конфигурационной переменной `date.timezone` или вызовом функции `date_default_timezone_set()`.

См. также

Документация по функции `date()`, классу `DateTime`, функциям `getdate()` и `localtime()`.

3.2. Преобразование компонентов времени и даты во временную метку

Задача

Требуется определить, какая временная метка соответствует набору компонентов времени и даты.

Решение

Воспользуйтесь функцией `mktime()`, если компоненты даты и времени заданы в местном часовом поясе, как показано в листинге 3.7.

Листинг 3.7. Получение временной метки

```
// 19:45:03, 10 марта 1975 года, местное время
// "Местным" считается восточное время США
$then = mktime(19,45,3,3,10,1975);
```

Если компоненты даты и времени заданы в часовом поясе GMT, используется функция `gmmktime()`.

Листинг 3.8. Получение временной метки для времени GMT

```
// 19:45:03, 10 марта 1975 г., время GMT
$then = gmmktime(19,45,3,3,10,1975);
```

Если компоненты даты и времени заданы отформатированной строкой времени, используйте метод `DateTime::createFromFormat()`, как показано в листинге 3.9.

Листинг 3.9. Получение временной метки по отформатированной строке времени

```
// 19:45:03, 10 марта 1975 года, конкретный часовой пояс
$then = DateTime::createFromFormat(DateTime::ATOM, "1975-03-10T19:45:03-04:00");
```

Комментарий

Функции `mktime()` и `gmmktime()` получают компоненты даты и времени (час, минута, секунда, месяц, день, год) и возвращают соответствующую им временную метку эпохи Unix. Функция `mktime()` интерпретирует компоненты как относящиеся к местному времени, тогда как функция `gmmktime()` интерпретирует их как дату и время в UTC.

В листинге 3.10 `$stamp_future` задается временная метка для времени 15:25 3 декабря 2024 года. Временную метку эпохи можно передать функции `date()` для получения отформатированной строки времени.

Листинг 3.10. Работа с временными метками

```
date_default_timezone_set('America/New_York');
// $stamp_future содержит 1733257500
$stamp_future = mktime(15,25,0,12,3,2024);
// $formatted содержит '2024-12-03T15:25:00-05:00'
$formatted = date('c', $stamp_future);
```

Вызовы `mktime()` в листинге 3.10 делаются для часового пояса Нью-Йорка (Америка), поэтому при использовании `gmmktime()` будет получена временная метка, которая на 18 000 секунд (пять часов) меньше предыдущей (листинг 3.11).

Листинг 3.11. Временные метки и gmmktime()

```
date_default_timezone_set('America/New_York');
// $stamp_future содержит 1733239500 - значение, на 18000
// меньшее 1733257500
$stamp_future = gmmktime(15,25,0,12,3,2024);
```

Метод `createFromFormat()` класса `DateTime` обладает большей гибкостью. Вместо готовых компонентов даты он получает отформатированное время или строку даты и описание структуры этой строки. Далее он выделяет компоненты и вычисляет правильную временную метку. Кроме форматных строк из Рецепта 3.4, поддерживаемых функцией `date()`, метод `createFromFormat()` также использует символы, перечисленные в таблице 3.3.

Таблица 3.3. Форматные символы метода DateTime::`createFromFormat()`

Символ	Значение
пробел или табуляция	
#	Любой из байтов-разделителей ;, :, /, ., ,, -, (,)
;, :, /, ., ,, -, (,)	Литеральный символ
?	Любой байт (не символ, только один байт)
*	Любое количество байтов до следующей цифры или разделителя
!	Сброс всех полей к значениям «начала эпохи Unix» (без этого все незаданные поля будут заполнены текущей датой/временем)
|	Сброс всех неразобранных полей значениями «начала эпохи Unix»
+	Неразобранные завершающие данные приводят к выдаче предупреждения вместо ошибки

Листинг 3.12 показывает, как использовать метод `DateTime::createFromFormat()` для выделения компонентов времени из большей строки.

Листинг 3.12. Использование DateTime::createFromFormat()

```
$text = "Birthday: May 11, 1918.";
$when = DateTime::createFromFormat("*: F j, Y.|", $text);
// $formatted содержит "Saturday, 11-May-18 00:00:00 UTC"
$formatted = $when->format(DateTime::RFC850);
```

См. также

Рецепт 3.3 — обратное преобразование временной метки в компоненты времени и даты; документация по функциям `mktime()` и `gmmktime()`, `date_default_timezone_set()` и методу `DateTime::createFromFormat()`.

3.3. Преобразование временной метки в компоненты времени и даты

Задача

Требуется получить компоненты времени и даты, соответствующие заданной временной метке.

Решение

Передайте временную метку функции `getdate()`: `$time_parts = getdate(163727100);`.

Комментарий

Компоненты времени, возвращаемые функцией `getdate()`, перечислены в таблице 3.1. Значения этих компонентов определяются относительно часового пояса, назначенного в PHP. Если вы хотите, чтобы компоненты времени задавались относительно другого часового пояса, измените часовой пояс функцией `date_default_timezone_set()`, а затем верните прежнее значение после вызова `getdate()`. Вы также можете создать объект `DateTime`, связать его с конкретным часовым поясом, а затем получить нужные компоненты времени и даты методом `format()` объекта:

```
$when = new DateTime("@163727100");
$when->setTimezone(new DateTimeZone('America/Los_Angeles'));
$parts = explode('/', $when->format('Y/m/d/H/i/s'));
// Год, месяц, день, часы, минуты, секунды
// $parts - массив ('1975', '03','10', '16','45', '00'))
```

Символ @ сообщает объекту DateTime, что оставшаяся часть аргумента конструктора содержит временную метку. При указании временной метки как исходного значения DateTime игнорирует часовой пояс, также переданный конструктору, поэтому для его назначения потребуется дополнительный вызов setTimezone(). Когда это будет сделано, метод format() сможет сгенерировать все необходимые компоненты.

См. также

Рецепт 3.2 — преобразование времени и даты во временные метки; Рецепт 3.10 — дополнительная информация о работе с часовыми поясами; документация по функциям getdate() и DateTime.

3.4. Вывод даты или времени в заданном формате

Задача

Требуется вывести дату или время, отформатированные определенным образом.

Решение

Воспользуйтесь функцией date() или методом DateTime::format(), как показано в листинге 3.13.

Листинг 3.13. Использование date() и DateTime::format( )

```
print date('d/M/Y') . "\n";
$when = new DateTime();
print $when->format('d/M/Y');
```

Результат выполнения листинга 3.13 выглядит примерно так:

```
06/Feb/2013
06/Feb/2013
```

Комментарий

Функция date() и метод DateTime::format() используют один внутренний код для генерирования отформатированных строк времени и даты. Эти гибкие функции могут строить отформатированные строки времени с разнообразными компонентами. Форматные символы этих функций перечислены в таблице 3.4.

Таблица 3.4. Форматные символы date()

Тип	Символ	Описание	Диапазон значений
Часы	H	Час в числовом представлении, 24-часовая шкала, начальный нуль	00–23
Часы	h	Час в числовом представлении, 12-часовая шкала, начальный нуль	01–12
Часы	G	Час в числовом представлении, 24-часовая шкала	0–23
Часы	g	Час в числовом представлении, 12-часовая шкала	1–12
Часы	A	Обозначение AM/PM	AM, PM
Часы	a	Обозначение AM/PM	am, pm
Минуты	i	День месяца в числовом представлении, начальный нуль	00–59
Секунды	s	День месяца в числовом представлении	00–59
Секунды	u	Микросекунды, строка	000000–999999
День	d	День месяца в числовом представлении, начальный нуль	01–31
День	j	День месяца в числовом представлении	1–31
День	z	День года в числовом представлении	0–365
День	N	День недели в числовом представлении (понедельник = 1)	1–7
День	w	День недели в числовом представлении (воскресенье = 0)	0–6
День	S	Английский суффикс порядкового числительного для дня месяца в текстовом представлении	«st», «th», «nd», «rd»
Неделя	D	Сокращенное название дня недели	Mon, Tue, Wed, Thu, Fri, Sat, Sun
Неделя	l	Полное название дня недели	Monday, Tuesday, Wednesday Thursday, Friday, Saturday, Sunday
Неделя	W	Номер недели в году (ISO 8601:1988) в числовом представлении; неделей 1 считается первая неделя текущего года, состоящая минимум из 4 дней; первым днем недели считается понедельник	1–53
Месяц	F	Полное название месяца	January–December
Месяц	M	Сокращенное название месяца	Jan–Dec
Месяц	m	Месяц в числовом представлении, начальный нуль	01–12

Таблица 3.4 (*продолжение*)

Тип	Символ	Описание	Диапазон значений
Месяц	n	Месяц в числовом представлении	1–12
Месяц	t	Длина месяца в днях в числовом представлении	28, 29, 30, 31
Год	Y	Год в числовом представлении, включая век	например, 2016
Год	y	Год без века в числовом представлении	например, 16
Год	o	Год с веком по стандарту ISO 8601, в числовом представлении; год из четырех цифр соответствует номеру недели ISO; то же, что Y, кроме того, что если номер недели ISO принадлежит предыдущему или следующему году, используется этот год	например, 2016
Год	L	Флаг високосного года (да = 1)	0, 1
Часовой пояс	O	Смещение в часах от GMT в формате ±ЧЧММ (например, −0400 +0230)	−1200–+1200
Часовой пояс	P	То же, что O, но с двоеточием	−12:00 –+12:00
Часовой пояс	Z	Смещение в секундах от GMT; к западу от GMT — отрицательное, к востоку от GMT — положительное	-43200–50400
Часовой пояс	e	Идентификатор часового пояса	например, America/New_York
Часовой пояс	T	Сокращенное обозначение часового пояса	например, EDT
Часовой пояс	I	Флаг летнего времени (да = 1)	0, 1
Составной тип	c	Дата и время в формате ISO 8601	например, 2012-09-06T15:29:34+0000
Составной тип	r	Дата в формате RFC 2822	например, Thu, 22 Aug 2002 16:01:07 +0200
Прочее	U	Количество секунд от начала эпохи Unix	0–2147483647
Прочее	B	Интернет-время Swatch	000–999

Форматные символы, генерирующие текст вместо чисел (такие как F, M или D), выдают результат на английском языке. О том, как сгенерировать строки даты и времени на других языках, рассказано в Рецепте 19.3.

Также для популярных форматов даты существуют удобные константы, которые представляют форматную строку, передаваемую `date()` или `DateTime::format()`. Эти константы перечислены в таблице 3.5.

Таблица 3.5. Константы, используемые с date()

Константа	Константа класса	Значение	Пример	Использование
DATE_ATOM	DateTime::ATOM	Y-m-d\TH:i:sP	2013-02-22T20:25:31+00:00	«Atom Syndication Format», раздел 3.3
DATE_ISO8601	DateTime::ISO8601	Y-m-d\TH:i:sO	2013-02-22T20:25:31+0000	ISO 8601 (http://www.w3.org/TR/NOTE-datetime)
DATE_RFC822	DateTime::RFC822	D, d M y H:i:s O	Fri, 22 Feb 13 20:25:31 +0000	Сообщения электронной почты (http://www.faqs.org/rfcs/rfc822.html)
DATE_RFC850	DateTime::RFC850	l, d-M-y H:i:s T	Friday, 22-Feb-13 20:25:31 UTC	Сообщения Usenet (http://www.faqs.org/rfcs/rfc850.html)
DATE_RFC1036	DateTime::RFC1036	D, d M y H:i:s O	Fri, 22 Feb 13 20:25:31 +0000	Сообщения Usenet (http://www.faqs.org/rfcs/rfc1036.html)
DATE_RFC1123	DateTime::RFC1123	D, d M Y H:i:s O	Fri, 22 Feb 2013 20:25:31 +0000	http://www.faqs.org/rfcs/rfc1123.html
DATE_RFC2822	DateTime::RFC2822	D, d M Y H:i:s O	Fri, 22 Feb 2013 20:25:31 +0000	Сообщения электронной почты (http://www.faqs.org/rfcs/rfc2822.html)
DATE_RFC3339	DateTime::RFC3339	Y-m-d\TH:i:sP	2013-02-22T20:25:31+00:00	http://www.faqs.org/rfcs/rfc3339.html
DATE_RSS	DateTime::RSS	D, d M Y H:i:s O	Fri, 22 Feb 2013 20:25:31 +0000	Поставки RSS (http://cyber.law.harvard.edu/rss/rss.html)
DATE_W3C	DateTime::W3C	Y-m-d\TH:i:sP	2013-02-22T20:25:31+00:00	http://www.w3.org/TR/NOTE-datetime

См. также

Документация по функции `date()` и методу `DateTime::format()`; Рецепт 19.3 — генерирование отформатированных строк времени и даты на разных языках.

3.5. Вычисление разности двух дат

Задача

Требуется вычислить интервал времени между двумя датами. Например, вы хотите сообщить пользователю, сколько времени прошло с момента его последнего посещения сайта.

Решение

Создайте для каждой даты объект `DateTime`. Воспользуйтесь методом `DateTime::diff()` для получения объекта `DateInterval`, описывающего разность между датами.

В листинге 3.14 выводится разность между двумя датами в неделях, днях, часах, минутах и секундах.

Листинг 3.14. Вычисление разности между двумя датами

```
// 19:32:56, 10 мая 1965 года
$first = new DateTime("1965-05-10 7:32:56pm",
         new DateTimeZone('America/New_York'));
// 4:29:11, 20 ноября 1962 года
$second = new DateTime("1962-11-20 4:29:11am",
          new DateTimeZone('America/New_York'));
$diff = $second->diff($first);

printf("The two dates have %d weeks, %s days, " .
       "%d hours, %d minutes, and %d seconds " .
       "elapsed between them.",
       floor($diff->format('%a') / 7),
       $diff->format('%a') % 7,
       $diff->format('%h'),
       $diff->format('%i'),
       $diff->format('%s'));
```

Код листинга 3.14 выводит следующий результат:

```
The two dates have 128 weeks, 6 days, 15 hours, 3 minutes, and 45 seconds
elapsed between them.
```

Комментарий

С вычислением разности дат связаны некоторые нюансы, о которых следует знать. Прежде всего, 1962 и 1965 годы предшествуют началу эпохи Unix. Впрочем, благодаря 600-миллиардному диапазону встроенной библиотеки PHP это не создает проблем.

Также учтите, что метод `DateTime::diff()` возвращает результат, который представляет разность времени, не всегда совпадающую с абсолютной величиной интервала. Две даты в листинге 3.14 находятся по разные стороны от переходной даты летнего времени, поэтому фактическое прошедшее время будет на час меньше выведенного (из-за «лишнего» часа, появляющегося при осеннем переходе на стандартное время).

Для вычисления разности постройте объекты `DateTime` на основе временных меток местного времени, а затем примените к этим объектам метод `DateTime::diff()`, как показано в листинге 3.15.

Листинг 3.15. Вычисление разности между двумя датами

```
// 19:32:56, 10 мая 1965 года
$first_local = new DateTime("1965-05-10 7:32:56pm",
                            new DateTimeZone('America/New_York'));
// 4:29:11, 20 ноября 1962 г.
$second_local = new DateTime("1962-11-20 4:29:11am",
                             new DateTimeZone('America/New_York'));

$first = new DateTime('@' . $first_local->getTimestamp());
$second = new DateTime('@' . $second_local->getTimestamp());

$diff = $second->diff($first);

printf("The two dates have %d weeks, %s days, " .
       "%d hours, %d minutes, and %d seconds " .
       "elapsed between them.",
       floor($diff->format('%a') / 7),
       $diff->format('%a') % 7,
       $diff->format('%h'),
       $diff->format('%i'),
       $diff->format('%s'));
```

Код листинга 3.15 выводит следующий результат:

```
The two dates have 128 weeks, 6 days, 14 hours, 3 minutes, and 45 seconds
elapsed between them.
```

Как видите, результат на час отличается от вывода в листинге 3.14. Объекты `DateTime`, созданные с форматной строкой `@` и временной меткой, всегда относятся к часовому поясу UTC, поэтому разность между ними не зависит от летнего времени и других местных поправок.

На момент написания книги еще не была исправлена ошибка PHP 52480, проявляющаяся в некоторых редких интервальных вычислениях с некоторыми значениями часов и смещениями часовых поясов. Эту ошибку можно обойти, используя UTC в качестве часового пояса для вычисления интервалов.

См. также

Документация по методу `DateTime::diff()` и объекту `DateInterval`. Дополнительная информация об ошибке PHP 52480.

3.6. Определение дня недели, месяца или года

Задача

Требуется узнать день недели, месяца или года. Например, вы хотите выводить специальное сообщение каждый понедельник или первого числа каждого месяца.

Решение

Передайте правильные аргументы `date()` или `DateTime::format()`, как показано в листинге 3.16.

Листинг 3.16. Определение дня недели, месяца или года

```
print "Today is day " . date('d') . ' of the month and ' . date('z') .
      ' of the year.';
print "\n";

$birthday = new DateTime('January 17, 1706', new DateTimeZone('America/New_
York'));

print "Benjamin Franklin was born on a " . $birthday->format('l') . ", " .
"day " . $birthday->format('N') . " of the week.";
```

Комментарий

Функции `date()` и `DateTime::format()` используют одинаковые форматные символы. В таблице 3.6 перечислены все форматные символы дней и недель.

Таблица 3.6. Числовые форматные символы дней и недель

Тип	Символ	Описание	Диапазон значений
День	d	День месяца в числовом представлении, начальный нуль	01–31
День	j	День месяца в числовом представлении	1–31
День	z	День года в числовом представлении	0–365
День	N	День недели в числовом представлении (понедельник = 1)	1–7
День	w	День недели в числовом представлении (воскресенье = 0)	0–6
День	S	Английский суффикс порядкового числительного для дня месяца в текстовом представлении	«st», «th», «nd», «rd»
Неделя	D	Сокращенное название дня недели	Mon, Tue, Wed, Thu, Fri, Sat, Sun
Неделя	l	Полное название дня недели	Monday, Tuesday, Wednesday Thursday, Friday, Saturday, Sunday
Неделя	W	Номер недели в году (ISO 8601:1988) в числовом представлении; неделей 1 считается первая неделя текущего года, состоящая минимум из 4 дней; первым днем недели считается понедельник	1–53

Чтобы сообщение выводилось только по понедельникам, используйте форматный символ `w`, как показано в листинге 3.17.

Листинг 3.17. Проверка дня недели

```
if (1 == date('w')) {
    print "Welcome to the beginning of your work week.";
}
```

Вычисление номера недели и дня недели может осуществляться разными способами; постарайтесь выбрать наиболее подходящий. В стандарте ISO (ISO 8601) говорится, что недели начинаются с понедельников, а дни недели нумеруются с 1 (понедельник) до 7 (воскресенье). Неделей 1 считается первая неделя года, в которую входит четверг. Другими словами, неделей 1 является та, в которой большинство дней принадлежит этому году. Номера недель лежат в диапазоне от 01 до 53.

В других стандартах нумерации недель значения лежат в диапазоне от 00 до 53, а дни недели 53 могут перекрываться с днями недели 00 следующего года.

Если вы в своих программах будете действовать последовательно, проблем быть не должно. Однако будьте внимательны при взаимодействии с другими программами PHP или базами данных. Например, функция MySQL `DAYOFWEEK()` считает первым днем недели воскресенье, но нумерует дни от 1 до 7, как предписывает стандарт ODBC. С другой стороны, функция `WEEKDAY()` считает первым днем недели понедельник и нумерует дни от 0 до 6. Функция `WEEK()` позволяет выбрать, с какого дня должна начинаться неделя (воскресенье или понедельник), но она несовместима со стандартом ISO.

См. также

Документация по функциям `date()` и `DateTime::format()`; функции MySQL `DAYOFWEEK()`, `WEEKDAY()` и `WEEK()` документированы на сайте MySQL.

3.7. Проверка существования введенной даты

Задача

Требуется проверить дату на «действительность». Например, вы хотите убедиться в том, что пользователь не ввел дату рождения 30 февраля 1962 года.

Решение

Воспользуйтесь функцией `checkdate()`:

```
// $ok содержит true - 10 марта 1993 года является действительной датой
$ok = checkdate(3, 10, 1993);
// $not_ok содержит false - 30 февраля 1962 года не является
// действительной датой
$not_ok = checkdate(2, 30, 1962);
```

Комментарий

Функция `checkdate()` возвращает `true`, если значение `$month` лежит в диапазоне от 1 до 12, значение `$year` — в диапазоне от 1 до 32 767, и значение `$day` — в диапазоне от 1 до максимально допустимого количества дней для заданных `$month` и `$year`. Функция `checkdate()` правильно обрабатывает високосные годы, а даты формируются по григорианскому календарю.

Из-за широкого диапазона действительных лет следует провести дополнительную проверку пользовательского ввода, например, если вы хотите получить реальную, а не вымышленную дату рождения. Самая долгая подтвержденная продолжительность человеческой жизни составляет 122 года. Функция `checkbirthdate()` из листинга 3.18 проверяет, что в соответствии с введенной датой рождения возраст пользователя составляет от 18 до 122 лет.

Листинг 3.18. checkbirthdate()

```
function checkbirthdate($month,$day,$year) {
    $min_age = 18;
    $max_age = 122;

    if (! checkdate($month,$day,$year)) {
        return false;
    }

    $now = new DateTime();
    $then_formatted = sprintf("%d-%d-%d", $year, $month, $day);
    $then = DateTime::createFromFormat("Y-n-j|",$then_formatted);
    $age = $now->diff($then);

    if (($age->y < $min_age)|| ($age->y > $max_age)) {
        return FALSE;
    }
    else {
        return TRUE;
    }
}

// Проверка 3 декабря 1974 года
if (checkbirthdate(12,3,1974)) {
    print "You may use this web site.";
} else {
    print "You are too young (or too old!!) to proceed.";
}
```

Функция сначала использует `checkdate()` для проверки действительности значений `$month`, `$day` и `$year`. Если все значения действительны, она строит два объекта `DateTime`: для текущего момента времени и для переданных значений месяца, дня и года. Вызов `sprintf()` нормализует переданные значения в формат целых чисел без начальных нулей, который соответствует требованиям форматной строки `Y-n-j`, передаваемой `DateTime::createFromFormat()`. Завершающий

символ | в форматной строке приказывает `DateTime::createFromFormat()` инициализировать незаданные компоненты часов, минут и секунд нулями.

После того как два объекта `DateTime` будут созданы, остается проверить возраст, определяемый заданной датой рождения. Для этого функция просто вызывает `DateTime::diff()` и проверяет свойство `y` объекта `DateInterval` (количество лет в интервале) на принадлежность диапазону допустимых значений.

Функция возвращает `true`, если указанная дата предшествует текущей дате ровно на `$min_age` лет, но `false`, если указанная дата приходится ровно на `$max_age` лет позже текущей даты. Таким образом, возраст будет считаться действительным в 18-й день рождения, но не в 123-й.

См. также

Документация по функции `checkdate()`; Википедия — информация о Жанне Кальман, долгожительнице с самой большой подтвержденной продолжительностью жизни.

3.8. Разбор строк даты и времени

Задача

Требуется преобразовать дату или время в строке в формат, который может использоваться в вычислениях. Например, преобразовать строки вида «last Thursday» или «February 9, 2004» во временную метку от начала эпохи.

Решение

Разбор строки даты или времени в произвольном формате проще всего выполняется функцией `strtotime()`. Эта функция преобразует разнообразные строковые представления даты и времени во временные метки, как показано в листинге 3.19.

Листинг 3.19. Разбор строк функцией strtotime()

```
$a = strtotime('march 10'); // По умолчанию текущий год
$b = strtotime('last thursday');
$c = strtotime('now + 3 months');
```

Комментарий

Функция `strtotime()` использует достаточно сложную и разнообразную грамматику. Она реализует спецификацию GNU Date Input Formats (доступна на сайте GNU) и ряд расширений.

Функция `strtotime()` «понимает» текстовые обозначения текущего времени:

```
$a = strtotime('now');
print date(DATE_RFC850, $a);
print "\n";

$a = strtotime('today');
print date(DATE_RFC850, $a);

Tuesday, 12-Feb-13 19:12:14 UTC
Tuesday, 12-Feb-13 00:00:00 UTC
```

Она позволяет задавать время и даты разными способами:

```
$a = strtotime('5/12/2014');
print date(DATE_RFC850, $a);
print "\n";

$a = strtotime('12 may 2014');
print date(DATE_RFC850, $a);

Monday, 12-May-14 00:00:00 UTC
Monday, 12-May-14 00:00:00 UTC
```

Также поддерживаются относительные определения времени и даты:

```
$a = strtotime('last thursday');   // 12 февраля 2013 года
print date(DATE_RFC850, $a);
print "\n";

$a = strtotime('2015-07-12 2pm + 1 month');
print date(DATE_RFC850, $a);

Thursday, 07-Feb-13 00:00:00 UTC
Wednesday, 12-Aug-15 14:00:00 UTC
```

Функция поддерживает часовые пояса. В следующем коде компонент времени (`2pm`) не изменяется, потому что идентификатор часового пояса, используемый PHP по умолчанию (`America/New_York`), совпадает с часовым поясом строки, передаваемой `strtotime()` (EDT — сокращение часового пояса для летнего времени в Нью-Йорке):

```
date_default_timezone_set('America/New_York');
$a = strtotime('2012-07-12 2pm America/New_York + 1 month');
print date(DATE_RFC850, $a);

Sunday, 12-Aug-12 14:00:00 EDT
```

Но если выбрать для PHP идентификатор часового пояса `America/Denver` (за два часа до `America/New_York`), то при передаче `strtotime()` той же строки будет выдано время в Нью-Йорке, соответствующее 14:00 в Денвере (за два часа до часового пояса Нью-Йорка):

```
date_default_timezone_set('America/New_York');
$a = strtotime('2012-07-12 2pm America/Denver + 1 month');
```

```
print date(DATE_RFC850, $a);

Sunday, 12-Aug-12 16:00:00 EDT
```

Расширенная грамматика `strtotime()` также применяется при создании объекта `DateTime`. Итак, хотя функция `strtotime()` чрезвычайно удобна для простого получения временных меток, вы можете передать те же строки `new DateTime()` и построить объект `DateTime`, с которым можно выполнять дальнейшие операции.

Если вы располагаете строкой даты или времени, формат которой известен, но не обрабатывается `strtotime()`, вы все равно можете создать объект `DateTime` на базе строки методом `DateTime::createFromFormat()`. В листинге 3.20 представлен пример использования `DateTime::createFromFormat()` для разбора строк даты, записанных в порядке «день-месяц-год» (по умолчанию в PHP используется порядок «месяц-день-год»).

Листинг 3.20. Разбор строк даты в нестандартном формате

```
$dates = array('01/02/2015', '03/06/2015', '09/08/2015');

foreach ($dates as $date) {
    $default = new DateTime($date);
    $day_first = DateTime::createFromFormat('d/m/Y|', $date);
    printf("The default interpretation is %s\n  but day-first is %s.\n",
           $default->format(DateTime::RFC850),
           $day_first->format(DateTime::RFC850));
}
```

Код листинга 3.20 выводит следующий результат:

```
The default interpretation is Friday, 02-Jan-15 00:00:00 UTC
  but day-first is Sunday, 01-Feb-15 00:00:00 UTC.
The default interpretation is Friday, 06-Mar-15 00:00:00 UTC
  but day-first is Wednesday, 03-Jun-15 00:00:00 UTC.
The default interpretation is Tuesday, 08-Sep-15 00:00:00 UTC
  but day-first is Sunday, 09-Aug-15 00:00:00 UTC.
```

См. также

Документация по функциям `strtotime()` и `DateTime::createFromFormat()`; правила с описаниями строк, разбираемых функцией `strtotime()`.

3.9. Операции сложения и вычитания с датами

Задача

Требуется прибавить или вычесть интервал из заданной даты.

Решение

Примените объект `DateInterval` к объекту `DateTime` методом `DateTime::add()` или `DateTime::sub()`, как показано в листинге 3.21.

Листинг 3.21. Прибавление и вычитание интервала

```
$birthday = new DateTime('March 10, 1975');

// Какой день был за 40 недель до $birthday?
$human_gestation = new DateInterval('P40W');
$birthday->sub($human_gestation);
print $birthday->format(DateTime::RFC850);
print "\n";

// А если бы это был слон, а не человек?
$elephant_gestation = new DateInterval('P616D');
$birthday->add($elephant_gestation);
print $birthday->format(DateTime::RFC850);
```

Комментарий

Методы `add()` и `sub()` класса `DateTime` изменяют объект `DateTime`, для которого они вызываются, на величину, заданную в интервале. Среднее время вынашивания потомства у человека составляет 40 недель, так что интервал `P40W` определяет дату за 40 недель до даты рождения — приблизительное время зачатия. С другой стороны, у слона среднее время вынашивания составляет примерно 616 дней. Таким образом, прибавление интервала `P616D` к дате зачатия определяет ожидаемую дату рождения для слона, зачатого одновременно с человеком.

Метод `modify()` объекта `DateTime` вместо объекта `DateInterval` получает строку, которую «понимает» функция `strtotime()`. Это позволяет легко находить относительные даты вида «next Tuesday» для заданного объекта. Например, выборы в США проходят во вторник после первого понедельника в ноябре (то есть в первый вторник ноября, если только он не является первым днем месяца; тогда выборы проводятся в следующий вторник). С методом `DateTime::modify()` дата дня выборов определяется следующим образом:

```
$year = 2016;
$when = new DateTime("November 1, $year");
if ($when->format('D') != 'Mon') {
    $when->modify("next Monday");
}
$when->modify("next Tuesday");

print "In $year, US election day is on the " .
    $when->format('jS') . ' day of November.';
```

Форматный символ `D` определяет день недели. Таким образом, если первым днем ноября не является понедельник, то вызов `$when->modify("next Monday")` пере-

мещает объект `DateTime` к следующему понедельнику. Далее следующий вызов `modify()` находит первый вторник после этого.

См. также

Документация по созданию объектов `DateInterval` методами `DateTime::add()`, `DateTime::sub()` и `DateTime::modify()`.

3.10. Вычисление времени с учетом часовых поясов и летнего времени

Задача

Требуется вычислять время в разных часовых поясах. Например, вы хотите предоставить пользователям информацию, скорректированную для их местного времени (вместо местного времени сервера).

Решение

Воспользуйтесь объектами `DateTimeZone` при создании объектов `DateTime`, и PHP выполнит всю работу за вас, как показано в листинге 3.22.

Листинг 3.22. Простое использование объекта часового пояса

```
$nowInNewYork = new DateTime('now', new DateTimeZone('America/New_York'));
$nowInCalifornia = new DateTime('now', new DateTimeZone('America/Los_Angeles'));

printf("It's %s in New York but %s in California.",
       $nowInNewYork->format(DateTime::RFC850),
       $nowInCalifornia->format(DateTime::RFC850));
```

Результат выглядит так:

```
It's Friday, 15-Feb-13 14:50:25 EST in New York but
Friday, 15-Feb-13 11:50:25 PST in California.
```

Обратите внимание: результат не только приводится к местному часовому поясу (выводимые часы отличаются на 3), но и включает правильное обозначение часового пояса. Если в используемом часовом поясе действует летнее время, оно учитывается автоматически.

Часовой пояс PHP по умолчанию задается параметром конфигурации `date.timezone` в начале обработки запроса. Его можно изменить вызовом `date_default_time_zone_set()`; этот часовой пояс начинает использоваться по умолчанию до следующего изменения или конца запроса. В листинге 3.23 текущее время выводится дважды — для Нью-Йорка и для Парижа.

Листинг 3.23. Смена часового пояса функцией date_default_timezone_set()

```
$now = time();
date_default_timezone_set('America/New_York');
print date(DATE_RFC850, $now);
print "\n";

date_default_timezone_set('Europe/Paris');
print date(DATE_RFC850, $now);
```

Листинг 3.23, как и листинг 3.22, выводит локализованные значения времени и часовые пояса.

Комментарий

Так как объекты `DateTime` взаимодействуют с объектами `DateTimeZone` (а другие функции — такие как `date()`, — учитывают настройки часового пояса системного уровня), вы можете легко менять часовые пояса с получением соответствующего отформатированного вывода. Информация часового пояса, которую учитывает PHP, также включает сведения о действии летнего времени.

Часовые пояса, поддерживаемые PHP, перечислены в руководстве *PHP Manual*. Названия часовых поясов — такие как `America/New_York`, `Europe/Paris` и `Africa/Dar_es_Salaam`, — отражают структуру популярной базы данных `zoneinfo`. Если вы захотите обновить базу данных часовых поясов без обновления всей установки PHP, установите (или обновите) расширение `timezonedb` из репозитория PECL. В него входит база данных часовых поясов для PHP, находящаяся под управлением IANA.

См. также

Документация по функциям `date_default_timezone_set()`, `date_default_timezone_get()` и классу `DateTimeZone`; часовые пояса, известные PHP; информация о базе данных часовых поясов IANA (*http://www.iana.org/time-zones*); расширение `timezonedb` из репозитория PECL.

3.11. Получение времени с высокой точностью

Задача

Требуется изменить время с точностью, превышающей секундную, например, чтобы сгенерировать уникальный идентификатор или провести хронометраж вызова функции.

Решение

Воспользуйтесь функцией `microtime(true)`, чтобы получить текущее время в секундах и микросекундах. В листинге 3.24 вызов `microtime(true)` используется для измерения продолжительности 1000 поисков совпадений регулярного выражения.

Листинг 3.24. Хронометраж с использованием microtime()

```
$start = microtime(true);
for ($i = 0; $i < 1000; $i++) {
    preg_match('/age=\d{1,5}/',$_SERVER['QUERY_STRING']);
}
$end = microtime(true);
$elapsed = $end - $start;
```

Комментарий

Без передачи аргумента, результатом вычисления которого является `true`, функция `microtime()` возвращает строку с микросекундной составляющей времени от начала эпохи, пробелом и секундами от начала эпохи. Например, возвращаемое значение `0.41644100 1026683258` означает, что от начала эпохи прошли 1026683258,41644100 секунд. При этом обеспечивается бóльшая точность, чем при использовании `float`, но вычисления усложняются.

С выходом PHP 5.4.0 в сверхглобальный массив `$_SERVER` включается элемент `REQUEST_TIME_FLOAT`. Он содержит время (в микросекундах) на начало обработки запроса. Это позволяет легко определить продолжительность обработки запроса в любой момент времени — достаточно вычислить результат выражения `microtime(true)-$_SERVER['REQUEST_TIME_FLOAT']`.

Время с микросекундами может использоваться для генерирования уникальных идентификаторов. В сочетании с идентификатором текущего процесса оно гарантирует уникальность идентификатора (при условии, что процесс не генерирует более одного идентификатора в микросекунду). В листинге 3.25 функция `microtime()` (со строковым форматом возвращаемого значения) используется для генерирования такого идентификатора.

Листинг 3.25. Генерирование идентификатора с использованием функции microtime()

```
list($microseconds,$seconds) = explode(' ',microtime());
$id = $seconds.$microseconds.getmypid();
```

Учтите, что метод из листинга 3.25 не обладает стопроцентной надежностью в многопоточных системах, в которых существует отличная от нуля (хотя и очень малая) вероятность того, что два программных потока одного процесса вызовут `microtime()` в одну микросекунду.

См. также

Документация по функциям `microtime()`.

3.12. Генерирование временных диапазонов

Задача

Требуется получить все дни недели или месяца, например чтобы вывести список встреч на неделю.

Решение

Воспользуйтесь классом `DatePeriod`, появившимся в PHP 5.3.0. Его конструктор получает гибкий набор параметров для управления длиной диапазона, временем между элементами и количеством элементов в диапазоне.

Для создания объекта `DatePeriod` необходимо задать начало, конец и шаг диапазона. В следующем примере строится диапазон для представления всех дней в августе 2014 года:

```
// Начать с 1 августа
$start = new DateTime('August 1, 2014');
// Остановиться 31 августа (конечная дата не включается)
$end = new DateTime('September 1, 2014');
// С шагом в 1 день
$interval = new DateInterval('P1D');
$range1 = new DatePeriod($start, $interval, $end);
```

Тот же результат можно получить другим способом:

```
// Начать с 1 августа
$start = new DateTime('August 1, 2014');
// С шагом в 1 день
$interval = new DateInterval('P1D');
// Повторить 30 раз от начала.
$recurrences = 30;
$range2 = new DatePeriod($start, $interval, $recurrences);
```

Наконец, третий способ использует формат для описания диапазонов дат ISO 8601:

```
$range3 = new DatePeriod('R30/2014-08-01T00:00:00Z/P1D');
```

Класс `DatePeriod` реализует интерфейс `Traversable`, поэтому после создания объекта достаточно передать его `foreach`, и вы получите объект `DateTime` для каждого элемента в диапазоне:

```
foreach ($range1 as $d) {
    print "A day in August is " . $d->format('d') . "\n";
}
```

Комментарий

По умолчанию `DatePeriod` включает начальную границу, но не включает конечную границу диапазона. Чтобы исключить также и начальную границу, передайте константу `DatePeriod::EXCLUDE_START_DATE` в последнем аргументе конструктора.

`DatePeriod` реализует только `Traversable`, но не другие предоставляемые PHP интерфейсы, позволяющие объекту «играть роль массива», так что вы не сможете получить все значения одновременно, например передав объект `implode()`. Вам придется использовать `foreach` для накопления нужных значений в обычном массиве.

См. также

Документация по `DatePeriod` и `DateInterval()`.

3.13. Использование негригорианских календарей

Задача

Требуется использовать негригорианский календарь (например, юлианский, еврейский или французский республиканский).

Решение

Календарное расширение PHP предоставляет функции преобразования для работы с юлианским календарем, а также французским республиканским и еврейским. Для использования этих функций необходимо предварительно загрузить календарное расширение.

Эти функции используют в качестве промежуточного формата юлианское *количество дней* от начала юлианской эпохи. Функция `cal_to_jd()` преобразует месяц, день и год в юлианское количество дней; функция `cal_from_jd()` преобразует юлианское количество дней в месяц, день и год заданного календаря. В листинге 3.26 выполняется преобразование юлианского количества дней к привычному григорианскому календарю.

Листинг 3.26. Преобразование между юлианским и григорианским календарем

```
// 8 марта 1876 года
// $jd содержит 2406323 - дату, выраженную юлианским количеством дней
$jd = gregoriantojd(3,9,1876);
$gregorian = cal_from_jd($jd, CAL_GREGORIAN);
/* $gregorian is array('date' => '3/9/1876',
                       'month' => 3,
```

```
                        'day' => 9,
                        'year' => 1876,
                        'dow' => 4,
                        'abbrevdayname' => 'Thu',
                        'dayname' => 'Thursday',
                        'abbrevmonth' => 'Mar',
                        'monthname' => 'March'));
*/
```

Действительный диапазон дат для григорианского календаря простирается от 4714 года до нашей эры до 9999 года нашей эры.

Комментарий

Для преобразования между юлианским количеством дней и юлианским календарем используйте константу `CAL_JULIAN`, как показано в листинге 3.27.

Листинг 3.27. Использование юлианского календаря

```
// 29 февраля 1900 г. (не является високосным по григорианскому календарю)
// $jd содержит 2415092, юлианское количество дней
$jd = cal_to_jd(CAL_JULIAN, 2, 29, 1900);

$julian = cal_from_jd($jd, CAL_JULIAN);
/* $julian - массив('date' => '2/29/1900',
                    'month' => 2,
                    'day' => 29,
                    'year' => 1900,
                    'dow' => 2,
                    'abbrevdayname' => 'Tue',
                    'dayname' => 'Tuesday',
                    'abbrevmonth' => 'Feb',
                    'monthname' => 'February'));
*/

$gregorian = cal_from_jd($jd, CAL_GREGORIAN);
/* $gregorian - массив ('date' => '3/13/1900',
                        'month' => 3,
                        'day' => 13,
                        'year' => 1900,
                        'dow' => 2,
                        'abbrevdayname' => 'Tue',
                        'dayname' => 'Tuesday',
                        'abbrevmonth' => 'Mar',
                        'monthname' => 'March'));
*/
```

Действительный диапазон юлианского календаря простирается от 4713 года до нашей эры до 9999 года нашей эры, но поскольку юлианский календарь был создан в 46 году до нашей эры, его использование для предшествующих дат может рассердить формалистов.

Для преобразования между юлианским количеством дней и французским республиканским календарем используйте константу `CAL_FRENCH` (см. листинг 3.28).

Листинг 3.28. Использование французского республиканского календаря

```
// 13 флореаля XI года
// $jd содержит 2379714, юлианское количество дней
$jd = cal_to_jd(CAL_FRENCH, 8, 13, 11);
$french = cal_from_jd($jd, CAL_FRENCH);
/* $french - массив ('date' => '8/13/11',
                     'month' => 8,
                     'day' => 13,
                     'year' => 11,
                     'dow' => 2,
                     'abbrevdayname' => 'Tue',
                     'dayname' => 'Tuesday',
                     'abbrevmonth' => 'Floreal',
                     'monthname' => 'Floreal'));
*/

// 3 мая 1803 г. - продажа Луизианы США
$gregorian = cal_from_jd($jd, CAL_GREGORIAN);
/* $gregorian - массив ('date' => '5/3/1803',
                        'month' => 5,
                        'day' => 3,
                        'year' => 1803,
                        'dow' => 2,
                        'abbrevdayname' => 'Tue',
                        'dayname' => 'Tuesday',
                        'abbrevmonth' => 'May',
                        'monthname' => 'May'));
*/
```

Действительный диапазон французского республиканского календаря простирается от сентября 1792 года до сентября 1806 года; он невелик, но поскольку календарь использовался только с октября 1793 года до января 1806 года, этого вполне достаточно. Обратите внимание: названия месяцев, возвращаемые `cal_from_jd()`, не содержат диакритических знаков, например вместо `Floréal` возвращается строка `Floreal`.

Для преобразования между юлианским количеством дней и еврейским календарем используйте константу `CAL_JEWISH`, как показано в листинге 3.29.

Листинг 3.29. Использование еврейского календаря

```
// 25 день месяца Кислев 5774 года - первая ночь/день праздника Ханука
// $jd содержит 2456625, юлианское количество дней
$jd = cal_to_jd(CAL_JEWISH, 3, 25, 5774);
$jewish = cal_from_jd($jd, CAL_JEWISH);
/* $jewish - массив ('date' => '3/25/5774',
                     'month' => 3,
                     'day' => 25,
                     'year' => 5774,
                     'dow' => 4,
                     'abbrevdayname' => 'Thu',
                     'dayname' => 'Thursday',
                     'abbrevmonth' => 'Kislev',
                     'monthname' => 'Kislev'));
```

```
*/

// 28 ноября 2013 года - День Благодарения в США
$gregorian = cal_from_jd($jd, CAL_GREGORIAN);
/* $gregorian - массив ('date' => '11/28/2013',
                        'month' => 11,
                        'day' => 28,
                        'year' => 2013,
                        'dow' => 4,
                        'abbrevdayname' => 'Thu',
                        'dayname' => 'Thursday',
                        'abbrevmonth' => 'Nov',
                        'monthname' => 'November'));
*/
```

Действительный диапазон еврейского календаря начинается с 3761 года до нашей эры (год 1 по еврейскому календарю). Месяц Адар всегда возвращается в виде `AdarI` независимо от того, приходится он на високосный год или нет. В високосные годы месяц Адар II возвращается в виде `AdarII`.

См. также

Документация по календарным функциям; история григорианского календаря.

3.14. Программа: календарь

Класс `LittleCalendar` из листинга 3.31 выводит календарь на месяц (по аналогии с программой Unix cal). Листинг 3.30 демонстрирует использование класса, включая использование стилей оформления.

Листинг 3.30. Использование LittleCalendar()

```
<style type="text/css">
.prev { text-align: left; }
.next { text-align: right; }
.day, .month, .weekday { text-align: center; }
.today { background: yellow; }
.blank { }
</style>
<?php
// Если в строке запроса не указан месяц или год,
// выводится календарь на текущий месяц
$month = isset($_GET['month']) ? intval($_GET['month']) : date('m');
$year = isset($_GET['year']) ? intval($_GET['year']) : date('Y');

$cal = new LittleCalendar($month, $year);

print $cal->html();
```

Класс `LittleCalendar` умеет выводить календарь на месяц в разных форматах. Его метод `prepare()` определяет информацию о каждом дне месяца и начальные/

завершающие заполнители. Затем внутренние методы, вызываемые `generate()` на основании аргумента, создают форматирование для разных контекстов. Метод `html()` строит календарь HTML, подходящий для отображения на веб-странице. Метод `text()` создает текстовый календарь для вывода на консоль.

Листинг 3.31. Класс LittleCalendar

```
class LittleCalendar {

    /** DateTime */
    protected $monthToUse;

    protected $prepared = false;

    protected $days = array();

    public function __construct($month, $year) {
    /* Построение объекта DateTime для выводимого месяца */
        $this->monthToUse = DateTime::createFromFormat('Y-m|',
                                                       sprintf("%04d-%02d",
                                                               $year, $month));
        $this->prepare();
    }

    protected function prepare() {
        // Построение массива с информацией о каждом дне месяца,
        // включая необходимые заполнители в начале и в конце.
        // В первой строке выводятся названия дней недели

      foreach (array('Su', 'Mo','Tu','We','Th','Fr','Sa') as $dow) {
          $endOfRow = ($dow == 'Sa');
          $this->days[] = array('type' => 'dow',
                                'label' => $dow,
                                'endOfRow' => $endOfRow);
      }

        // Затем выводятся заполнители до первого дня недели
        for ($i = 0, $j = $this->monthToUse->format('w'); $i < $j; $i++) {
            $this->days[] = array('type' => 'blank');
        }

        // Затем по одному элементу для каждого дня месяца
        $today = date('Y-m-d');
        $days = new DatePeriod($this->monthToUse,
                               new DateInterval('P1D'),
                               $this->monthToUse->format('t') - 1);
        foreach ($days as $day) {
            $isToday = ($day->format('Y-m-d') == $today);
            $endOfRow = ($day->format('w') == 6);
            $this->days[] = array('type' => 'day',
                                  'label' => $day->format('j'),
                                  'today' => $isToday,
                                  'endOfRow' => $endOfRow);
        }

        // Далее выводятся заполнители до конца месяца,
```

```
    // если конец недели не пришелся на последний день месяца
    if (! $endOfRow) {
        for ($i = 0, $j = 6 - $day->format('w'); $i < $j; $i++) {
            $this->days[] = array('type' => 'blank');
        }
    }
}

public function html($opts = array()) {
    if (! isset($opts['id'])) {
        $opts['id'] = 'calendar';
    }
    if (! isset($opts['month_link'])) {
        $opts['month_link'] =
            '<a href="'.htmlentities($_SERVER['PHP_SELF']) . '?' .
            'month=%d&amp;year=%d">%s</a>';
    }
    $classes = array();
    foreach (array('prev','month','next','weekday','blank','day','today')
             as $class) {
        if (isset($opts['class']) && isset($opts['class'][$class])) {
            $classes[$class] = $opts['class'][$class];
        }
        else {
         $classes[$class] = $class;
     }
 }
 /* Построение объекта DateTime для предыдущего месяца */
 $prevMonth = clone $this->monthToUse;
 $prevMonth->modify("-1 month");
 $prevMonthLink = sprintf($opts['month_link'],
                          $prevMonth->format('m'),
                          $prevMonth->format('Y'),
                          '&laquo;');

 /* Построение объекта DateTime для следующего месяца */
 $nextMonth = clone $this->monthToUse;
 $nextMonth->modify("+1 month");
 $nextMonthLink = sprintf($opts['month_link'],
                           $nextMonth->format('m'),
                           $nextMonth->format('Y'),
                          '&raquo;');

    $html = '<table id="'.htmlentities($opts['id']).'">
     <tr>
     <td class="'.htmlentities($classes['prev']).'">' .
        $prevMonthLink . '</td>
     <td class="'.htmlentities($classes['month']).'" colspan="5">'.
        $this->monthToUse->format('F Y') .'</td>
     <td class="'.htmlentities($classes['next']).'">' .
        $nextMonthLink . '</td>
     </tr>';

    $html .= '<tr>';

    $lastDayIndex = count($this->days) - 1;
    foreach ($this->days as $i => $day) {
```

```
            switch ($day['type']) {
            case 'dow':
                $class = 'weekday';
                $label = htmlentities($day['label']);
                break;
            case 'blank':
                $class = 'blank';
                $label = ' ';
                break;
            case 'day':
                $class = $day['today'] ? 'today' : 'day';
                $label = htmlentities($day['label']);
                break;
            }
            $html .=
                '<td class="' . htmlentities($classes[$class]).'">'.
               $label . '</td>';

           if (isset($day['endOfRow']) && $day['endOfRow']) {
               $html .= "</tr>\n";
               if ($i != $lastDayIndex) {
                   $html .= '<tr>';
               }
           }
       }
       $html .= '</table>';
       return $html;
   }

   public function text() {
        $lineLength = strlen('Su Mo Tu We Th Fr Sa');
        $header = $this->monthToUse->format('F Y');
        $headerSpacing = floor(($lineLength - strlen($header))/2);

        $text = str_repeat(' ', $headerSpacing) . $header . "\n";

        foreach ($this->days as $i => $day) {
            switch ($day['type']) {
            case 'dow':
                $text .= sprintf('% 2s', $day['label']);
                break;
            case 'blank':
                $text .= '   ';
                break;
            case 'day':
                $text .= sprintf("% 2d", $day['label']);
                break;
            }
            $text .= (isset($day['endOfRow']) && $day['endOfRow']) ? "\n" : " ";
        }
        if ($text[strlen($text)-1] != "\n") {
            $text .= "\n";
        }
        return $text;
    }
}
```

Конструктор `LittleCalendar` строит объект `DateTime` для месяца, на который требуется вывести календарь. Затем вызывается метод `prepare()`, который выполняет подготовительную работу по заполнению переменной `$days` массивом информации о каждом из выводимых дней (или заполнителей). Сначала `prepare()` помещает в `$days` элементы для каждого дня недели (как строку заголовка), а потом — заполнители, вычисленные на основании дня недели первого дня месяца. Далее в `$days` помещается элемент для каждого дня месяца, и в завершение — заполнители до конца месяца (если потребуется).

Внутри `prepare()` для получения необходимой информации о каждом дне месяца метод `format()` вызывается для объектов `DateTime`. Так мы получаем информацию дня недели для заполнителей, а также информацию каждого дня. Дни месяца определяются перебором по диапазону `DatePeriod` для используемого месяца с однодневным интервалом.

Хотя метод `prepare()` предоставляет достаточно информации для построения макета календаря, непосредственное форматирование поручается другим методам. Метод `html()` строит календарь в формате HTML, а метод `text()` использует текстовый формат.

Метод `html()` получает в необязательном аргументе массив параметров. В `$opts['month_link']` можно передать форматную строку в стиле `printf()` для изменения способа вывода ссылок на следующий и предыдущий месяцы, а также атрибута `id` таблицы. По умолчанию `id` содержит значение `calendar`.

Также можно передать имена классов для разных элементов макета. Они размещаются в параметре `class`, который содержит массив. В массиве `class` можно задать классы `prev`, `month`, `next`, `weekday`, `blank`, `day` и `today`. В листинг 3.30 включены стили, определяющие базовый макет таблицы, в том числе выделение текущего дня желтым цветом.

Метод `html()` определяет предыдущий и следующий месяцы (при помощи `DateTime::modify()`), чтобы сгенерировать правильные ссылки для перехода к предыдущему и следующему месяцу. После создания короткого заголовка метод перебирает вычисленные дни, помещая каждый день в соответствующую ячейку таблицы. В конце каждой недели строка таблицы закрывается.

Метод `text()` содержит аналогичную логику, но с другим форматом вывода. Он генерирует заголовок с месяцем и годом, а затем перебирает вычисленные дни, добавляя разрыв строки в конце каждой недели.

Субклассируя `LittleCalendar`, можно добавить другие адаптированные варианты календаря. Например, вы можете определить метод `colorText()`, использующий служебные коды ANSI для вывода текущего дня другим цветом.

4 Массивы

4.0. Введение

Массивы используются для хранения списков: людей, книг, метрик и т. д. Если вам потребуется сохранить группу взаимосвязанных элементов данных в массиве, используйте массив. Элементы массива (как и элементы списка, записанного на листе бумаги) располагаются в определенном порядке. Обычно каждый новый элемент вставляется за последним элементом массива, но при необходимости новый элемент можно вставить между парой существующих элементов массива PHP.

В большинстве языков используются массивы с числовым индексированием, или *числовые массивы* (обычно они называются просто *массивами*). Чтобы обратиться к элементу числового массива, достаточно знать его позицию в массиве, называемую *индексом*. Позиции определяются по номерам: номера начинаются с 0 и последовательно возрастают на 1.

В некоторых языках также поддерживается другая разновидность массивов: *ассоциативные массивы*, также называемые *хешами*, *картами* или *словарями*. В ассоциативном массиве в качестве индексов используются не целые числа, а строки. Таким образом, в числовом массиве президентов США элемент «Abraham Lincoln» может иметь индекс 16, а в ассоциативном массиве ему может быть присвоен индекс «Honest». Но если у числовых массивов существует четкий порядок элементов, определяемый их ключами, ассоциативные массивы часто не предоставляют никаких гарантий относительно порядка ключей. Элементы добавляются в определенном порядке, но позднее определить этот порядок уже не удастся.

В языках, поддерживающих как числовые, так и ассоциативные массивы, числовой массив `$presidents` и ассоциативный массив `$presidents` обычно являются разными массивами. Каждая разновидность массивов обладает своим поведением, и работать с ним нужно соответствующим образом. В PHP поддерживаются

как числовые, так и ассоциативные массивы, но они не обладают независимым поведением. В PHP числовые массивы являются ассоциативными, а ассоциативные массивы — числовыми. К какой же разновидности относится каждый отдельный массив? К обеим сразу и ни к одной. Четкой границы между этими разновидностями не существует. На первый взгляд это выглядит странно, особенно если вы привыкли к четко определенному поведению, но вскоре становится ясно, что такая гибкость полезна.

Чтобы сохранить в массиве сразу несколько значений за один шаг, используйте функцию `array()`:

```
$fruits = array('Apples', 'Bananas', 'Cantaloupes', 'Dates');
```

Теперь элемент массива `$fruits[2]` содержит значение `'Cantaloupes'`.

Функция `array()` удобна при работе с короткими списками заранее известных значений. Тот же массив может быть создан следующей серией присваиваний:

```
$fruits[0] = 'Apples';
$fruits[1] = 'Bananas';
$fruits[2] = 'Cantaloupes';
$fruits[3] = 'Dates';
```

и:

```
$fruits[] = 'Apples';
$fruits[] = 'Bananas';
$fruits[] = 'Cantaloupes';
$fruits[] = 'Dates';
```

В PHP 5.4 также поддерживается компактный синтаксис операций с массивами, позаимствованный из JavaScript:

```
$fruits = ['Apples', 'Bananas', 'Cantaloupes', 'Dates'];
```

Присваивание значения массиву без указания индекса — сокращенная запись для добавления нового элемента в конце массива. PHP проверяет длину `$fruits` и использует ее как позицию присваиваемого значения. Конечно, при этом предполагается, что `$fruits` не присвоено скалярное значение (например, 3) или объект. При попытке интерпретировать значение, не являющееся массивом, как массив PHP протестует; впрочем, если переменная используется впервые, PHP автоматически преобразует его в массив и начинает индексирование с 0.

Аналогичную функциональность предоставляет функция `array_push()`, которая заносит новое значение на верх стека массива. Однако запись `$foo[]` больше соответствует традиционному стилю PHP и быстрее работает. Но в некоторых ситуациях `array_push()` лучше выражает стековую природу выполняемых операций — особенно в сочетании с функцией `array_pop()`, которая извлекает последний элемент из массива и возвращает его.

До сих пор мы сохраняли в массивах только целые числа и строки. Однако PHP позволяет присвоить элементу массива любой тип данных: логическое значение,

целое число, число с плавающей точкой, строку, объект, ресурс, NULL и даже другой массив. Таким образом, вы можете загрузить массивы или объекты прямо из базы данных и разместить их в массиве:

```
while ($row = mysqli_fetch_assoc($r)) {
    $fruits[] = $row;
}

while ($obj = mysqli_fetch_object($s)) {
    $vegetables[] = $obj;
}
```

Первый цикл while создает массив массивов, второй создает массив объектов. За дополнительной информацией о хранении нескольких элементов под одним ключом обращайтесь к Рецепту 4.2.

Для определения массива со строковыми ключами вместо целочисленных также можно воспользоваться функцией array(), но в определениях пар «ключ/значение» используется синтаксис =>:

```
$fruits = array('red' => 'Apples', 'yellow' => 'Bananas',
                'beige' => 'Cantaloupes', 'brown' => 'Dates');
```

Теперь элемент $fruits['beige'] содержит строку 'Cantaloupes'. Данная конструкция является сокращенной записью для следующей серии:

```
$fruits['red'] = 'Apples';
$fruits['yellow'] = 'Bananas';
$fruits['beige'] = 'Cantaloupes';
$fruits['brown'] = 'Dates';
```

Короткий синтаксис может быть применен и в этом случае:

```
$fruits = [
    'red' => 'Apples',
    'yellow' => 'Bananas',
    'beige' => 'Cantaloupes',
    'brown' => 'Dates'
];
```

Каждый массив может содержать только одно уникальное значение для каждого ключа. Команда:

```
$fruits['red'] = 'Strawberry';
```

переопределяет значение 'Apples'. С другой стороны, ничто не мешает добавить новый ключ:

```
$fruits['orange'] = 'Orange';
```

Чем больше вы программируете на PHP, тем чаще вы будете использовать ассоциативные массивы вместо числовых. Вместо создания числового массива со строковыми значениями можно создать ассоциативный массив и разместить ваши значения в виде ключей. При желании в значении элемента сохраняется допол-

нительная информация. Быстродействие программы при этом не снижается; вдобавок операции поиска по ключу или изменения значений выполняются просто, потому что ключ уже известен.

Чтобы перебрать элементы массива и обработать некоторые из них (или все), проще всего воспользоваться циклом `foreach`:

```
$fruits = array('red' => 'Apples', 'yellow' => 'Bananas',
                'beige' => 'Cantaloupes', 'brown' => 'Dates');
foreach ($fruits as $color => $fruit) {
    print "$fruit are $color.\n";
}
```

При каждой итерации PHP присваивает следующий ключ переменной `$color`, а значение, связанное с этим ключом, — переменной `$fruit`. После перебора всех элементов цикл завершается.

Чтобы разбить массив на отдельные переменные, воспользуйтесь функцией `list()`:

```
$fruits = array('Apples', 'Bananas', 'Cantaloupes', 'Dates');
list($red, $yellow, $beige, $brown) = $fruits;
```

4.1. Определение массивов с начальным индексом, отличным от 0

Задача

Требуется присвоить значения нескольких элементов массива в одной операции, но при этом начальный индекс не должен быть равен 0.

Решение

Прикажите функции `array()` использовать другой начальный индекс при помощи синтаксиса `=>`:

```
$presidents = array(1 => 'Washington', 'Adams', 'Jefferson', 'Madison');
```

Комментарий

Первый элемент массивов PHP — как и во многих, хотя и не всех языках программирования — имеет индекс 0. Однако в некоторых ситуациях логичнее хранить данные начиная с индекса 1 (и это касается не только программистов, работающих на Pascal!).

Джордж Вашингтон был первым президентом США, а не нулевым, поэтому при выводе списка президентов проще действовать так:

```
foreach ($presidents as $number => $president) {
    print "$number: $president\n";
}
```

а не так:

```
foreach ($presidents as $number => $president) {
    $number++;
    print "$number: $president\n";
}
```

Данная возможность не ограничивается значением 1; можно использовать любое целое число:

```
$reconstruction_presidents = array(16 => 'Lincoln', 'Johnson', 'Grant');
// или
$reconstruction_presidents = [16 => 'Lincoln', 'Johnson', 'Grant'];
```

Также конструкция `=>` может использоваться многократно в одном вызове:

```
$whig_presidents = array(9 => 'Harrison', 'Tyler', 12 => 'Taylor', 'Fillmore');
// или
$whig_presidents = [9 => 'Harrison', 'Tyler', 12 => 'Taylor', 'Fillmore'];
```

PHP даже разрешает отрицательные числа при вызове `array()`. (Более того, этот метод работает даже для нецелочисленных ключей.) Формально при этом вы получаете ассоциативный массив, хотя, как мы уже говорили, граница между числовыми и ассоциативными массивами в PHP часто оказывается размытой; это один из таких случаев:

```
$us_leaders = array(-1 => 'George II', 'George III', 'Washington');
// или
$us_leaders = [-1 => 'George II', 'George III', 'Washington'];
```

Конечно, в одном определении `array()` можно смешивать числовые и строковые ключи, но такой синтаксис трудно понять, а необходимость в нем возникает редко:

```
$presidents = array(1 => 'Washington', 'Adams', 'Honest' => 'Lincoln',
                    'Jefferson');
// или
$presidents = [1 => 'Washington', 'Adams', 'Honest' => 'Lincoln', 'Jefferson'];
```

Эта запись эквивалентна следующей:

```
$presidents[1]         = 'Washington';   // Ключ 1
$presidents[]          = 'Adams';        // Ключ 1 + 1 => 2
$presidents['Honest']  = 'Lincoln';      // Ключ 'Honest'
$presidents[]          = 'Jefferson';    // Ключ 2 + 1 => 3
```

См. также

Документация по функции `array()`.

4.2. Хранение нескольких элементов для каждого ключа

Задача

Требуется связать несколько элементов с каждым ключом.

Решение

Сохраните элементы в массиве:

```
$fruits = array('red' => array('strawberry','apple'),
                'yellow' => array('banana'));
```

Или воспользуйтесь объектом:

```
while ($obj = mysqli_fetch_assoc($r)) {
    $fruits[] = $obj;
}
```

Комментарий

В PHP ключи должны быть уникальны на уровне массива, поэтому связать несколько элементов с одним ключом не удастся (это приведет к замене старого значения). Вместо этого сохраните значения в анонимном массиве:

```
$fruits = array();
$fruits['red'][] = 'strawberry';
$fruits['red'][] = 'apple';
$fruits['yellow'][] = 'banana';

print_r($fruits);
```

Результат выглядит так:

```
Array
(
    [red] => Array
        (
            [0] => strawberry
            [1] => apple
        )
    [yellow] => Array
        (
            [0] => banana
        )
)
```

Или если элементы обрабатываются в цикле:

```
while (list($color,$fruit) = mysqli_fetch_assoc($r)) {
    $fruits[$color][] = $fruit;
}
```

Чтобы вывести значения элементов, переберите содержимое массива:

```
foreach ($fruits as $color => $color_fruit) {
    // $color_fruit - массив
    foreach ($color_fruit as $fruit) {
        print "$fruit is colored $color.<br>";
    }
}
```

Или воспользуйтесь функцией `array_to_comma_string()` из Рецепта 4.9:

```
foreach ($fruits as $color => $color_fruit) {
    print "$color colored fruits include " .
        array_to_comma_string($color_fruit) . "<br>";
}
```

См. также

Рецепт 4.9 — вывод содержимого массивов с запятыми.

4.3. Инициализация массива диапазоном целых чисел

Задача

Требуется сохранить в массиве серию последовательных целых чисел.

Решение

Воспользуйтесь функцией `range($start, $stop)`:

```
$cards = range(1, 52);
```

Комментарий

Если приращение должно быть отлично от 1, передайте его `range()` в третьем аргументе.

Сохранение в массиве серии нечетных чисел:

```
$odd = range(1, 52, 2);
```

Для четных чисел:

```
$even = range(2, 52, 2);
```

См. также

Рецепт 2.4 — работа с сериями целых чисел; документация по функции `range()`.

4.4. Перебор элементов массива

Задача

Требуется в цикле перебрать элементы массива и выполнить операцию с некоторыми (или всеми) элементами.

Решение

Воспользуйтесь циклом `foreach`:

```
foreach ($array as $value) {
    // Операция с $value
}
```

Для получения ключей и значений массива:

```
foreach ($array as $key => $value) {
    // ...
}
```

Также можно воспользоваться циклом `for`:

```
for ($key = 0, $size = count($array); $key < $size; $key++) {
   // ...
}
```

Наконец, можно использовать `each()` в сочетании с `list()` и `while`:

```
reset($array); // Внутренний указатель устанавливается в начало массива
while (list($key, $value) = each ($array)) {
    // ...
}
```

Комментарий

Цикл `foreach` является самым лаконичным способом перебора массива:

```
// foreach со значениями
foreach ($items as $cost) {
    // ...
}

// foreach с ключами и значениями
foreach($items as $item => $cost) {
    // ...
}
```

С циклом `foreach` PHP перебирает содержимое копии массива вместо исходного массива. Напротив, при использовании `each()` и `for` PHP перебирает исходный массив. Следовательно, при изменении массива внутри цикла ожидаемое пове-

дение можно и не получить. Если вам понадобится изменить массив, обращайтесь к нему напрямую:

```
foreach ($items as $item => $cost) {
    if (! in_stock($item)) {
        unset($items[$item]);           // Обращение к массиву напрямую
    }
}
```

Переменные, возвращаемые `foreach()`, не являются псевдонимами для исходных значений в массиве: это копии, поэтому их изменение не отражается в массиве. Вот почему изменять следует `$items[$item]` вместо `$cost`.

При использовании `each()` PHP отслеживает текущую позицию внутри цикла. Чтобы начать сначала после первого прохода, вызовите `reset()`, чтобы вернуть указатель в начало массива. В противном случае вызов `each()` вернет `false`.

Цикл `for` подходит только для массивов с последовательными целочисленными ключами. Если вы не изменяете размер массива внутри цикла, заново вызывать `count()` для `$items` при каждом проходе было бы неэффективно, поэтому мы всегда используем переменную `$size` для хранения размера массива:

```
for ($item = 0, $size = count($items); $item < $size; $item++) {
    // ...
}
```

Если вы предпочитаете эффективный перебор с одной переменной, считайте в обратном направлении:

```
for ($item = count($items) - 1; $item >= 0; $item--) {
    // ...
}
```

Версия цикла `for` для ассоциативного массива выглядит так:

```
for (reset($array); $key = key($array); next($array) ) {
    // ...
}
```

Она не работает, если какой-либо элемент содержит строку, интерпретируемую как `false`, так что даже абсолютно законное значение (например, 0) может привести к преждевременному завершению цикла. По этой причине данный синтаксис используется редко, а мы его приводим только для того, чтобы вам было проще разобраться в старом коде PHP.

Наконец, функция `array_map()` используется для передачи каждого элемента функции для обработки:

```
// Преобразование всех слов к нижнему регистру
$lc = array_map('strtolower', $words);
```

В первом аргументе `array_map()` передается функция, которая должна изменять отдельный элемент, а во втором — перебираемый массив.

В общем случае эта функция уступает перечисленным ранее методам в гибкости, но она хорошо подходит для обработки и слияния нескольких массивов.

Если вы не уверены в том, являются ли обрабатываемые данные скалярным значением или массивом, необходимо защититься от возможного вызова `foreach` для «не массива». Для этого можно воспользоваться функцией `is_array()`:

```
if (is_array($items)) {
    // Код с циклом foreach для массива
} else {
    // Код для скалярного значения
}
```

Другой способ основан на преобразовании всех переменных в массив функцией `settype()`:

```
settype($items, 'array');
// Код цикла для массивов
```

В этом случае скалярное значение преобразуется в массив из одного элемента, а код становится более компактным за счет некоторого снижения эффективности.

См. также

Рецепт 4.24 — использование генераторов для эффективного перебора больших или высокозатратных наборов данных; документация по циклам `for`, `foreach`, `while`, функциям `each()`, `reset()` и `array_map()`.

4.5. Удаление элементов из массива

Задача

Требуется удалить из массива один или несколько элементов.

Решение

Чтобы удалить один элемент, воспользуйтесь функцией `unset()`:

```
unset($array[3]);
unset($array['foo']);
```

Для удаления нескольких несмежных элементов также используется функция `unset()`:

```
unset($array[3], $array[5]);
unset($array['foo'], $array['bar']);
```

Чтобы удалить несколько смежных элементов, воспользуйтесь функцией `array_splice()`:

```
array_splice($array, $offset, $length);
```

Комментарий

При использовании этих функций пропадают все ссылки на эти элементы. Если вы хотите оставить ключ в массиве, но связать его с пустым значением, присвойте элементу пустую строку:

```
$array[3] = $array['foo'] = '';
```

Кроме синтаксиса, между использованием `unset()` и присваиванием элементу `''` существует смысловая разница. Первое означает: «Элемент больше не существует», а второе — «Элемент существует, но его значение представляет собой пустую строку».

Если вы работаете с числами, возможно, лучше будет связать с ключом значение 0. Если, допустим, компания прекратила производство деталей модели XL1000, она обновляет свои данные складского учета командой:

```
unset($products['XL1000']);
```

Но если детали XL1000 временно отсутствуют на складе, но на следующей неделе с завода придет новая партия, лучше поступить иначе:

```
$products['XL1000'] = 0;
```

При вызове `unset()` для элемента PHP корректирует массив так, чтобы цикл по-прежнему работал правильно. Массив не сжимается для заполнения образовавшихся «дыр». Именно это мы имеем в виду, говоря, что все массивы являются ассоциативными, даже если на первый взгляд кажутся числовыми. Пример:

```
// Создание "числового" массива
$animals = array('ant', 'bee', 'cat', 'dog', 'elk', 'fox');
print $animals[1];  // Выводит 'bee'
print $animals[2];  // Выводит 'cat'
count($animals);    // Возвращает 6

// unset()
unset($animals[1]); // Удаляет элемент $animals[1] = 'bee'
print $animals[1];  // Ничего не выводит и выдает ошибку E_NOTICE
print $animals[2];  // Выводит 'cat'
count($animals);    // Возвращает 5, хотя элемент $array[5] содержит 'fox'

// Добавление нового элемента
$animals[] = 'gnu'; // Добавляет новый элемент
print $animals[1];  // Ничего не выводит, также выдает ошибку E_NOTICE
print $animals[6];  // Выводит 'gnu'
count($animals);    // Возвращает 6

// Присваивание ''
$animals[2] = '';   // Значение обнуляется
print $animals[2];  // Выводит ''
count($animals);    // Возвращает 6, count не уменьшается
```

Чтобы перейти к плотно заполненному числовому массиву, воспользуйтесь функцией `array_values()`:

```
$animals = array_values($animals);
```

Также функция `array_splice()` автоматически переиндексирует массивы для устранения «дыр»:

```
// Создание "числового" массива
$animals = array('ant', 'bee', 'cat', 'dog', 'elk', 'fox');
array_splice($animals, 2, 2);
print_r($animals);

Array
(
    [0] => ant
    [1] => bee
    [2] => elk
    [3] => fox
)
```

Такая возможность может пригодиться, если вы работаете с массивом как с очередью и хотите удалить элементы из очереди, не теряя возможность произвольного доступа. Для безопасного удаления первого или последнего элемента из массива используются функции `array_shift()` и `array_pop()` соответственно.

Впрочем, если вы часто сталкиваетесь с проблемами из-за «дыр» в массивах, скорее всего, вы еще не освоили «менталитет PHP». Поищите в Рецепте 4.4 способы перебора массива, не связанные с использованием циклов `for`.

См. также

Рецепт 4.4 — способы перебора; документация по функциям `unset()`, `array_splice()` и `array_values()`.

4.6. Изменение размера массива

Задача

Требуется изменить размер массива — увеличить или уменьшить его по сравнению с текущим размером.

Решение

Для увеличения массива используется функция `array_pad()`:

```
// Начать с трех элементов
$array = array('apple', 'banana', 'coconut');
// Увеличить до пяти
$array = array_pad($array, 5, '');
```

Теперь вызов `count($array)` возвращает 5, а два последних элемента, `$array[3]` и `$array[4]`, содержат пустые строки.

Для усечения массива можно воспользоваться функцией `array_splice()`:

```
// Без присваивания
array_splice($array, 2);
```

Этот вызов удаляет из `$array` все элементы, кроме первых двух.

Комментарий

Массивы PHP не имеют заранее определенного размера, поэтому их размеры можно изменять динамически.

Функция `array_pad()` дополняет массив новыми элементами. В первом аргументе передается массив, а второй аргумент определяет размер массива и направление дополнения. Чтобы дополнить массив вправо, передайте положительное целое число; для дополнения влево используется отрицательное значение. Третий аргумент содержит значение, которое должно быть присвоено вновь созданным элементам. Функция возвращает измененный массив без изменения оригинала.

Несколько примеров:

```
// Создание массива из четырех элементов, с дополнением вправо
// строкой 'dates'
$array = array('apple', 'banana', 'coconut');
$array = array_pad($array, 4, 'dates');
print_r($array);

Array
(
    [0] => apple
    [1] => banana
    [2] => coconut
    [3] => dates
)

// Создание массива из шести элементов, с дополнением влево
// строками 'zucchini'
$array = array_pad($array, -6, 'zucchini');
print_r($array);
Array
(
    [0] => zucchini
    [1] => zucchini
    [2] => apple
    [3] => banana
    [4] => coconut
    [5] => dates
)
```

Будьте внимательны: вызов `array_pad($array, 4, 'dates')` гарантирует, что `$array` содержит *минимум* четыре элемента; он не добавляет четыре новых элемента. Если массив `$array` уже содержит четыре и более элемента, `array_pad()` вернет неизмененную копию.

Кроме того, если объявить значение для четвертого элемента `$array[4]`:

```
$array = array('apple', 'banana', 'coconut');
$array[4] = 'dates';
print_r($array);
```

вы получите массив из четырех элементов с индексами 0, 1, 2 и 4:

```
Array
(
    [0] => apple
    [1] => banana
    [2] => coconut
    [4] => dates
)
```

Фактически PHP превращает его в ассоциативный массив, имеющий целочисленные ключи.

Функция `array_splice()`, в отличие от `array_pad()`, имеет побочный эффект в виде модификации исходного массива. Она возвращает усеченный массив; именно поэтому возвращаемое значение не присваивается `$array`. Но как и в случае с `array_pad()`, усечение может производиться как слева, так и справа. Вызов `array_splice()` со значением –2 отсекает последние два элемента с конца:

```
// Создание массива из четырех элементов
$array = array('apple', 'banana', 'coconut', 'dates');
// Усечение до трех элементов
array_splice($array, 3);
// Удаление последнего элемента, эквивалент array_pop()
array_splice($array, -1);
// Остались только элементы 'apple' и 'banana'
print_r($array);
```

См. также

Документация по функциям `array_pad()` и `array_splice()`.

4.7. Слияние массивов

Задача

Требуется объединить два массива в один.

Решение

Воспользуйтесь функцией `array_merge()`:

```
$garden = array_merge($fruits, $vegetables);
```

Комментарий

Функция `array_merge()` работает как с заранее определенными массивами, так и с массивами, определяемыми «на месте» при помощи `array()`:

```
$p_languages = array('Perl', 'PHP');
$p_languages = array_merge($p_languages, array('Python'));
print_r($p_languages);
Array
(
    [0] => Perl
    [1] => PHP
    [2] => Python
)
```

Соответственно, объединяемые массивы могут быть как уже существующими (как в случае с `$p_languages`), так и анонимными (как `array('Python')`).

Использовать `array_push()` не удастся, потому что PHP не производит автоматического преобразования массива в серию независимых переменных, и в итоге вы получите вложенный массив:

```
array_push($p_languages, array('Python'));
print_r($p_languages);
Array
(
    [0] => Perl
    [1] => PHP
    [2] => Array
        (
            [0] => Python
        )

)
```

При слиянии массивов, содержащих только числовые ключи, происходит перенумерация массивов, так что значения не теряются. При слиянии массивов со строковыми ключами значения из второго массива заменяют значения с одинаковыми ключами. Для массивов с обоими типами ключей реализуются оба типа поведения. Например:

```
$lc = array('a', 'b' => 'b'); // Значения - буквы нижнего регистра
$uc = array('A', 'b' => 'B'); // Значения - буквы верхнего регистра
$ac = array_merge($lc, $uc);  // Все регистры?
print_r($ac);
Array
(
    [0] => a
    [b] => B
    [1] => A
)
```

Буква `A` (в верхнем регистре) была размещена под индексом 1 вместо 0 для предотвращения коллизии и присоединена в конце итогового массива. Буква `B`

(в верхнем регистре) заменила букву b нижнего регистра на ее исходном месте в массиве.

Для слияния массивов также может использоваться оператор +. Для ключей с одинаковыми именами, присутствующих в обоих массивах, используется значение из левого операнда. Переупорядочение для предотвращения коллизий при этом не выполняется. В приведенном примере:

```
print_r($uc + $lc);
print_r($lc + $uc);
Array
(
    [0] => A
    [b] => B
)
Array
(
    [0] => a
    [b] => b
)
```

Так как оба значения, a и A, имеют ключ 0, а значения b и B имеют ключ b, в итоге в объединенных массивах остаются только два элемента.

В первом случае результат $a+$b представляет собой $b, а во втором результат $b+$a представляет собой $a.

Впрочем, для двух массивов с разными ключами проблем не будет, а новый массив будет содержать объединение двух массивов.

См. также

Документация по функции array_merge().

4.8. Преобразование массива в строку

Задача

Требуется преобразовать имеющийся массив в отформатированную строку.

Решение

Воспользуйтесь функцией join():

```
// Построение списка, разделенного запятыми
$string = join(',', $array);
```

Или постройте строку в цикле:

```
$string = '';
```

```
foreach ($array as $key => $value) {
    $string .= ",$value";
}
$string = substr($string, 1); // Удаление "," в начале
```

Комментарий

Если решение с `join()` работает в вашей ситуации, выбирайте его; оно работает быстрее любого решения на базе цикла PHP. Однако функции `join()` не хватает гибкости. Прежде всего она вставляет ограничители только *между* элементами, а не заключает элементы в них. Чтобы заключить элементы в HTML-теги `<b>` и разделить их запятыми, приходится действовать так:

```
$left  = '<b>';
$right = '</b>';
$html = $left . join("$right,$left", $html) . $right;
```

Кроме того, `join()` не различает значения. Если вы хотите включить подмножество элементов, цикл придется проводить самостоятельно:

```
$string = '';
foreach ($fields as $key => $value) {
    // Пароль не включается
    if ('password' != $key) {
        $string .= ",<b>$value</b>";
    }
}
$string = substr($string, 1); // Удаление "," в начале
```

Обратите внимание: разделитель всегда добавляется к каждому значению, а затем отсекается за пределами цикла. И хотя включать в строку то, что будет удалено из нее позднее, вроде бы расточительно, такое решение намного «чище» и эффективнее (в большинстве случаев), чем попытка встраивания логики в цикл:

```
$string = '';
foreach ($fields as $key => $value) {
    // Пароль не включается
    if ('password' != $value) {
        if (!empty($string)) { $string .= ','; }
        $string .= "<b>$value</b>";
    }
}
```

Теперь `$string` приходится проверять при каждом присоединении промежуточного значения. Такое решение хуже простого вызова `substr()`. Кроме того, ограничитель (в данном случае запятая) включается в начало, а не в конец, потому что усечение строки в начале выполняется быстрее, чем в конце.

См. также

Рецепт 4.9 — вывод массива с разделением запятыми; документация по функциям `join()` и `substr()`.

4.9. Вывод массива с разделением запятыми

Задача

Требуется вывести массив, разделяя элементы запятыми, и вывести *and* перед последним элементом, если массив содержит более двух элементов.

Решение

Используйте функцию `array_to_comma_string()`, приведенную в листинге 4.1.

Листинг 4.1. array_to_comma_string()

```
function array_to_comma_string($array) {
    switch (count($array)) {
    case 0:
        return '';

    case 1:
        return reset($array);

    case 2:
        return join(' and ', $array);

    default:
        $last = array_pop($array);
        return join(', ', $array) . ", and $last";
    }
}
```

Комментарий

Если у вас имеется список элементов, часто бывает нужно вывести его в грамматически правильной форме. Было бы неудобно использовать для вывода конструкции вида:

```
$thundercats = array('Lion-O', 'Panthro', 'Tygra', 'Cheetara', 'Snarf');
print 'ThunderCat good guys include ' . join(', ', $thundercats) . '.';
```

Приведенная реализация несколько усложняется из-за того, что функция `array_to_comma_string()` должна работать с любыми массивами, а не только с числовыми, у которых индексы начинаются с 0. Если ограничиться только этим подмножеством, для массива размером 1 можно вернуть `$array[0]`. Но если индексы массива не начинаются с 0, `$array[0]` содержит пустое значение. Мы используем тот факт, что функция `reset()`, сбрасывающая внутренний указатель массива, также возвращает значение его первого элемента.

По сходным причинам для получения конечного элемента можно вызвать `array_pop()` — вместо того, чтобы предполагать, что он находится в позиции `$array-[count($array)-1]`. Это позволяет использовать `join()` с `$array`.

Также обратите внимание на то, что код секции 2 правильно работает и для секции 1. А код секции `default` работает (хотя и неэффективно) для секции 2; однако свойство транзитивности в данном случае не действует, поэтому код `default` не может использоваться для элементов с размером 1.

См. также

Рецепт 4.8 — преобразование массива в строку; документация по функциям `join()`, `array_pop()` и `reset()`.

4.10. Проверка присутствия ключа в массиве

Задача

Требуется проверить, содержит ли массив заданный ключ.

Решение

Используйте функцию `array_key_exists()` для проверки присутствия ключа (с любым связанным значением):

```
if (array_key_exists('key', $array)) {
    /* Существует значение $array['key'] */
}
```

Используйте функцию `isset()` для проверки ключа, с которым связано любое значение, кроме `null`:

```
if (isset($array['key'])) { /* there is a non-null value for 'key' in $array */ }
```

Комментарий

Функция `array_key_exists()` полностью игнорирует значения — она только сообщает, присутствует ли в массиве элемент с заданным ключом. С другой стороны, функция `isset()` ведет себя с ключами массива так же, как с любыми другими переменными: при значении `null` функция `isset()` возвращает `false`. За дополнительной информацией об истинности значений переменных обращайтесь к «Введению» главы 5.

См. также

Документация по функциям `isset()` и `array_key_exists()`.

4.11. Проверка присутствия элемента в массиве

Задача

Требуется определить, содержит ли массив некоторое значение.

Решение

Воспользуйтесь функцией `in_array()`:

```
if (in_array($value, $array)) {
    // В массиве $array имеется элемент со значением $value
}
```

Комментарий

Функция `in_array()` проверяет, содержит ли элемент массива заданное значение:

```
$book_collection = array('Emma', 'Pride and Prejudice', 'Northhanger Abbey');
$book = 'Sense and Sensibility';

if (in_array($book, $book_collection)) {
    echo 'Own it.';
} else {
    echo 'Need it.';
}
```

По умолчанию `in_array()` сравнивает элементы оператором `==`. Чтобы использовать жесткую проверку равенства `===`, передайте `true` в третьем параметре `in_array()`:

```
$array = array(1, '2', 'three');

in_array(0, $array);        // true
in_array(0, $array, true);  // false
in_array(1, $array);        // true
in_array(1, $array, true);  // true
in_array(2, $array);        // true
in_array(2, $array, true);  // false
```

Результат первой проверки `in_array(0, $array)` равен `true`, потому что при сравнении числа 0 со строкой `three` PHP преобразует `three` в целое число. Такой строки не существует, поэтому результат равен 0. Соответственно, `in_array()` считает, что совпадение существует.

Следовательно, при сравнении чисел с данными, которые могут содержать строки, безопаснее использовать жесткое сравнение.

Если вам приходится многократно вызывать `in_array()` для одного массива, возможно, лучше создать ассоциативный массив, в котором элементы исходного массива становятся ключами нового ассоциативного массива. Поиск элементов с использованием `in_array()` выполняется за линейное время; в ассоциативных массивах время поиска становится постоянным.

Если вы не можете создать ассоциативный массив напрямую и вам приходится преобразовывать традиционный массив с целочисленными ключами, воспользуйтесь функцией `array_flip()`, чтобы поменять местами ключи и значения массива:

```
$book_collection = array('Emma',
                         'Pride and Prejudice',
                         'Northhanger Abbey');

// Преобразование числовых массивов в ассоциативные массивы
$book_collection = array_flip($book_collection);
$book = 'Sense and Sensibility';

if (isset($book_collection[$book])) {
    echo 'Own it.';
} else {
    echo 'Need it.';
}
```

Учтите, что при этом несколько ключей, с которыми связываются одинаковые значения, сжимаются в один элемент обращенного массива.

См. также

Рецепт 4.12 — определение позиции значения в массиве; документация по функциям `in_array()` и `array_flip()`.

4.12. Определение позиции значения в массиве

Задача

Требуется узнать, присутствует ли значение в массиве. Если значение присутствует в массиве, требуется узнать его ключ.

Решение

Воспользуйтесь функцией `array_search()`. Эта функция возвращает ключ найденного значения. Если значение отсутствует в массиве, то возвращается значение `false`:

```
$position = array_search($value, $array);
if ($position !== false) {
    // Элемент в позиции $position имеет значение $value в массиве $array
}
```

Комментарий

Функция `in_array()` сообщает, содержит ли массив некоторое значение; функция `array_search()` позволяет узнать, *где находится* это значение. Но поскольку функция `array_search()` корректно обрабатывает ситуации, в которых значение не найдено, лучше использовать `array_search()` вместо `in_array()`. Различия в скорости незначительны, а дополнительная информация может оказаться полезной:

```
$favorite_foods = array(1 => 'artichokes', 'bread', 'cauliflower',
                        'deviled eggs');
$food = 'cauliflower';
$position = array_search($food, $favorite_foods);

if ($position !== false) {
    echo "My #$position favorite food is $food";
} else {
    echo "Blech! I hate $food!";
}
```

Используйте проверку `!==` с `false`, потому что если строка расположена в позиции 0 массива, `$position` интерпретируется как `false` и условие будет сочтено ложным, а это не то, что вам нужно.

Если значение встречается в массиве многократно, гарантируется только то, что `array_search()` вернет один из экземпляров (а не первый экземпляр).

См. также

Рецепт 4.11 — проверка присутствия элемента в массиве; документация по функции `array_search()`; более сложные средства поиска в массивах с использованием регулярных выражений рассматриваются в описании функции `preg_replace()` на сайте PHP и в главе 23.

4.13. Поиск элементов, удовлетворяющих некоторому условию

Задача

Требуется найти в массиве элементы, удовлетворяющие некоторому условию.

Решение

Воспользуйтесь циклом `foreach`:

```
$movies = array(/*...*/);
foreach ($movies as $movie) {
    if ($movie['box_office_gross'] < 5000000) { $flops[] = $movie; }
}
```

Или функцией `array_filter()`:

```
$movies = array(/* ... */);
$flops = array_filter($movies, function ($movie) {
    return ($movie['box_office_gross'] < 5000000) ? 1 : 0;
});
```

Комментарий

Решение с циклом `foreach` элементарно: вы перебираете данные и присоединяете к возвращаемому массиву только те элементы, которые удовлетворяют проверяемому условию.

Если вас интересует только первый такой элемент, цикл прерывается командой `break`:

```
$movies = array(/*...*/);
foreach ($movies as $movie) {
    if ($movie['box_office_gross'] > 200000000) { $blockbuster = $movie; break; }
}
```

Также можно вернуть управление прямо из функции:

```
function blockbuster($movies) {
    foreach ($movies as $movie) {
        if ($movie['box_office_gross'] > 200000000) { return $movie; }
    }
}
```

Однако при использовании `array_filter()` можно сначала создать анонимную функцию, возвращающую `true` для значений, которые должны остаться, и `false` для исключаемых значений. Далее массив обрабатывается так же, как в `foreach`.

Преждевременный выход из `array_filter()` невозможен, поэтому цикл `foreach` обладает большей гибкостью и его проще понять. Это один из тех немногочисленных случаев, когда встроенная функция PHP не имеет явных преимуществ перед кодом пользовательского уровня.

См. также

Документация по функции `array_filter()` и анонимным функциям.

4.14. Поиск элемента с наибольшим или наименьшим значением

Задача

Имеется массив; требуется найти в нем элемент с наибольшим или наименьшим значением (например, для вычисления диапазона при построении гистограммы).

Решение

Для поиска наибольшего элемента используйте функцию `max()`:

```
$largest = max($array);
```

Для поиска наибольшего элемента используйте функцию `min()`:

```
$smallest = min($array);
```

Комментарий

Обычно функция `max()` возвращает большее из двух значений, но если передать ей массив, она проводит поиск по всему массиву. К сожалению, получить индекс наибольшего элемента при использовании `max()` не удастся. Для этого необходимо отсортировать массив в обратном порядке, чтобы наибольший элемент находился в позиции 0:

```
arsort($array);
```

Теперь наибольшим элементом будет `$array[0]`.

Если вы не хотите нарушать порядок элементов в исходном массиве, создайте копию и отсортируйте ее:

```
$copy = $array;
arsort($copy);
```

То же можно сказать и о `min()`, но вместо `arsort()` в этом случае используется функция `asort()`.

Обе функции, `max()` и `min()`, выдают предупреждение при передаче пустого массива.

См. также

Рецепт 4.16 — сортировка массива; документация по функциям `max()`, `min()`, `arsort()` и `asort()`.

4.15. Перестановка в обратном порядке

Задача

Требуется переставить элементы массива в обратном порядке.

Решение

Воспользуйтесь функцией `array_reverse()`:

```
$array = array('Zero', 'One', 'Two');
$reversed = array_reverse($array);
```

Комментарий

Функция `array_reverse()` переставляет элементы массива в обратном порядке. Тем не менее часто удается обойтись без лишней перестановки. Если вы хотите переставить элементы только что отсортированного массива, измените процедуру сортировки, чтобы она использовала обратный порядок. Если потребовалось переставить элементы списка, который вы собираетесь перебрать и обработать, просто измените направление цикла. Вместо:

```
for ($i = 0, $size = count($array); $i < $size; $i++) {
    // ...
}
```

используйте обратный цикл:

```
for ($i = count($array) - 1; $i >=0 ; $i--) {
    // ...
}
```

Как обычно, цикл `for` следует использовать только для плотно упакованного массива, не содержащего «дыр».

В некоторых ситуациях возможно другое решение — обращение порядка размещения элементов в массиве. Например, если массив заполняется серией записей из базы данных, измените запрос и включите в него условие `ORDER DESC`. За информацией о точном синтаксисе команды обращайтесь к справочнику по базе данных.

См. также

Документация по функции `array_reverse()`.

4.16. Сортировка массива

Задача

Требуется отсортировать массив конкретным способом.

Решение

Чтобы отсортировать массив в соответствии с традиционным определением сортировки, воспользуйтесь функцией `sort()`:

```
$states = array('Delaware', 'Pennsylvania', 'New Jersey');
sort($states);
```

Чтобы выполнить числовую сортировку, передайте во втором аргументе `sort()` константу `SORT_NUMERIC`:

```
$scores = array(1, 10, 2, 20);
sort($scores, SORT_NUMERIC);
```

Вместо лексикографического порядка (1, 10, 2, 20) массив сортируется по возрастанию элементов (1, 2, 10, 20).

Комментарий

Функция `sort()` не охраняет связи «ключ/значение»; вместо этого элементы переиндексируются по возрастанию начиная с 0.

Чтобы сохранить связи «ключ/значение», используйте функцию `asort()`. Обычно эта функция `asort()` используется для ассоциативных массивов, но она также пригодится для массивов с индексами, несущими полезную информацию:

```
$states = array(1 => 'Delaware', 'Pennsylvania', 'New Jersey');
asort($states);
while (list($rank, $state) = each($states)) {
    print "$state was the #$rank state to join the United States\n";
}
```

Функция `natsort()` сортирует массив с применением алгоритма «естественной сортировки». При использовании этого алгоритма можно смешивать в элементах строки и числа и все равно получить правильный ответ:

```
$tests = array('test1.php', 'test10.php', 'test11.php', 'test2.php');
natsort($tests);
```

После сортировки элементы размещаются в порядке `'test1.php'`, `'test2.php'`, `'test10.php'` и `'test11.php'`. При естественной сортировке число 10 следует после числа 2, тогда как при традиционной сортировке их порядок противоположен. Для естественной сортировки без учета регистра символов используется функция `natcasesort()`.

Чтобы отсортировать массив в обратном порядке, используйте функцию `rsort()` или `arsort()` (аналог `rsort()` с сохранением ключей). Этим функциям также может передаваться константа `SORT_NUMERIC`. Функции `natrsort()` или `natcasersort()` не существуют.

См. также

Рецепт 4.17 — сортировка с пользовательской функцией сравнения; Рецепт 4.18 — сортировка нескольких массивов; документация по функциям `sort()`, `asort()`, `natsort()`, `natcasesort()`, `rsort()` и `arsort()`.

4.17. Сортировка массива по вычисляемому полю

Задача

Требуется реализовать собственную процедуру сортировки.

Решение

Воспользуйтесь функцией `usort()` в сочетании с пользовательской функцией сравнения:

```
$tests = array('test1.php', 'test10.php', 'test11.php', 'test2.php');
// Естественная сортировка в обратном порядке
usort($tests, function ($a, $b) {
    return strnatcmp($b, $a);
});
```

Комментарий

Функция сравнения должна возвращать положительное значение, если `$a > $b`; 0, если `$a == $b`; и отрицательное значение, если `$a < $b`. Чтобы выполнить обратную сортировку, возвращайте противоположное значение. Функция из решения — `strnatcmp()` — соблюдает эти правила.

Чтобы провести сортировку в обратном порядке, вместо умножения возвращаемого значения `strnatcmp($a, $b)` на –1 достаточно поменять порядок аргументов `strnatcmp($b, $a)`.

Функция сравнения не обязана быть «оберткой» для существующей функции сортировки или анонимной функции. Например, функция `date_sort()` из листинга 4.2 демонстрирует сортировку дат.

Листинг 4.2. date_sort()

```
// Получает даты в форме "ММ/ДД/ГГГГ"
function date_sort($a, $b) {
    list($a_month, $a_day, $a_year) = explode('/', $a);
    list($b_month, $b_day, $b_year) = explode('/', $b);

    if ($a_year  > $b_year ) return  1;
    if ($a_year  < $b_year ) return -1;

    if ($a_month > $b_month) return  1;
    if ($a_month < $b_month) return -1;

    if ($a_day   > $b_day  ) return  1;
    if ($a_day   < $b_day  ) return -1;

    return 0;
}

$dates = array('12/14/2000', '08/10/2001', '08/07/1999');
usort($dates, 'date_sort');
```

При проведении сортировки `usort()` пересчитывает возвращаемые значения функции сравнения каждый раз, когда потребуется сравнить два элемента; это замедляет сортировку. Чтобы избежать лишней работы, вы можете кэшировать сравниваемые значения, как показано в функции `array_sort()` из листинга 4.3.

Листинг 4.3. array_sort()

```
function array_sort($array, $map_func, $sort_func = '') {
    $mapped = array_map($map_func, $array);    // Кэширование значений $map_
func()

    if ('' === $sort_func) {
        asort($mapped);                        // asort() работает быстрее
usort()
    }  else {
        uasort($mapped, $sort_func);           // Необходимо сохранять ключи
    }

    while (list($key) = each($mapped)) {
        $sorted[] = $array[$key];              // Использование отсортированных
ключей
    }

    return $sorted;
}
```

Чтобы избежать лишней работы, функция `array_sort()` использует временный массив `$mapped` для кэширования возвращаемых значений. Она сортирует `$mapped` с использованием порядка сортировки по умолчанию или алгоритма сортировки, заданного пользователем. Важно, что используемая функция сортировки сохраняет отношения «ключ/значение». По умолчанию используется функция `asort()`, потому что она работает быстрее, чем `uasort()`. (В конце концов, функция `array_sort()` написана именно из-за недостаточной эффективности `uasort()`.) Наконец, `array_sort()` создает отсортированный массив `$sorted`, используя от-

сортированные ключи в `$mapped` для индексирования значений исходного массива.

Для небольших массивов или простых функций сортировки `usort()` работает быстрее, но с ростом объема вычислений `array_sort()` превосходит `usort()`. В следующем примере элементы сортируются по длине строки — относительно быстрая специальная сортировка:

```
function u_length($a, $b) {
    $a = strlen($a);
    $b = strlen($b);

    if ($a == $b) return  0;
    if ($a  > $b) return  1;
                  return -1;
}

function map_length($a) {
    return strlen($a);
}

$tests = array('one', 'two', 'three', 'four', 'five',
               'six', 'seven', 'eight', 'nine', 'ten');

// Быстрее для < 5 элементов с u_length()
usort($tests, 'u_length');

// Быстрее для >= 5 элементов с map_length()
$tests = array_sort($tests, 'map_length');
```

В этом примере `array_sort()` начинает работать быстрее `usort()`, когда массив достигает пяти элементов.

См. также

Рецепт 4.16 — основы сортировки; Рецепт 4.18 — сортировка нескольких массивов; документация по функциям `usort()`, `asort()`, `array_map()` и анонимным функциям.

4.18. Сортировка нескольких массивов

Задача

Требуется отсортировать несколько массивов или многомерный массив.

Решение

Воспользуйтесь функцией `array_multisort()`.

Чтобы отсортировать несколько массивов одновременно, передайте их `array_multisort()`:

```
$colors = array('Red', 'White', 'Blue');
$cities = array('Boston', 'New York', 'Chicago');

array_multisort($colors, $cities);
print_r($colors);
print_r($cities);
Array
(
    [0] => Blue
    [1] => Red
    [2] => White
)
Array
(
    [0] => Chicago
    [1] => Boston
    [2] => New York
)
```

Чтобы отсортировать разные измерения одного массива, передайте элементы массива:

```
$stuff = array('colors' => array('Red', 'White', 'Blue'),
               'cities' => array('Boston', 'New York', 'Chicago'));

array_multisort($stuff['colors'], $stuff['cities']);
print_r($stuff);
Array
(
    [colors] => Array
        (
            [0] => Blue
            [1] => Red
            [2] => White
        )

    [cities] => Array
        (
            [0] => Chicago
            [1] => Boston
            [2] => New York
        )
)
```

Чтобы изменить тип сортировки, как и в случае с `sort()`, передайте после массива константу `SORT_REGULAR`, `SORT_NUMERIC` или `SORT_STRING`. Чтобы изменить порядок сортировки, в отличие от `sort()`, передайте после массива `SORT_ASC` или `SORT_DESC`. После массива также можно передать как тип, так и порядок сортировки.

Комментарий

Функция `array_multisort()` может отсортировать несколько массивов одновременно либо многомерный массив — по одному или нескольким измерениям.

Массивы рассматриваются как столбцы таблицы, сортируемой по строкам. Первый массив определяет основную сортировку; все элементы других массивов переупорядочиваются на основании порядка сортировки первого массива. Если элементы первого массива при сравнении оказываются равными, порядок сортировки определяется вторым массивом, и т. д.

По умолчанию используются параметры сортировки `SORT_REGULAR` и `SORT_ASC`. Более того, они автоматически восстанавливаются после каждого массива, поэтому передавать эти два значения не нужно (разве что для того, чтобы смысл кода стал более очевидным):

```
$numbers = array(0, 1, 2, 3);
$letters = array('a', 'b', 'c', 'd');
array_multisort($numbers, SORT_NUMERIC, SORT_DESC,
                $letters, SORT_STRING , SORT_DESC);
```

В этом примере элементы массивов переставляются в обратном порядке.

См. также

Рецепт 4.16 — простая сортировка; Рецепт 4.17 — сортировка с пользовательской функцией; документация по функции `array_multisort()`.

4.19. Сортировка массива с использованием метода вместо функции

Задача

Требуется определить пользовательский алгоритм сортировки для упорядочения элементов массива. При этом вместо функции должен использоваться метод объекта.

Решение

Передайте вместо имени функции массив, содержащий имена класса и метода:

```
usort($access_times, array('dates', 'compare'));
```

Комментарий

Как и в случае с пользовательской функцией сортировки, метод объекта должен получать два аргумента и возвращать 1, 0 или −1, если первый параметр соответственно больше второго, равен или меньше его:

```
class sort {
    // Обратное сравнение строк
    static function strrcmp($a, $b) {
```

```
        return strcmp($b, $a);
    }
}

usort($words, array('sort', 'strrcmp'));
```

Кроме того, метод должен быть объявлен статическим (`static`). Также можно использовать экземпляр класса:

```
class Dates {
    public function compare($a, $b) { /* compare here */ }
}

$dates = new Dates;

usort($access_times, array($dates, 'compare'));
```

См. также

Глава 7 — дополнительная информация о классах и объектах; Рецепт 4.17 — пользовательская сортировка массивов.

4.20. Случайная перестановка

Задача

Требуется переставить элементы массива в случайном порядке.

Решение

Воспользуйтесь функцией `shuffle()`:

```
shuffle($array);
```

Комментарий

Как ни странно, сгенерировать качественную случайную перестановку не так просто. Более того, до выхода PHP 4.3 функция PHP `shuffle()` не создавала действительно случайной перестановки. Она перемещала элементы, но некоторые комбинации были более вероятными, чем другие.

По этой причине в пользовательском коде рекомендуется использовать функцию PHP `shuffle()` там, где это возможно.

См. также

Документация по функции `shuffle()`.

4.21. Удаление дубликатов из массива

Задача

Требуется исключить из массива дубликаты (повторяющиеся значения).

Решение

Если массив уже построен, воспользуйтесь функцией `array_unique()`, которая возвращает новый массив без дубликатов:

```
$unique = array_unique($array);
```

Если массив строится одновременно с обработкой результатов, для числовых массивов решение может выглядеть так:

```
foreach ($_GET['fruits'] as $fruit) {
    if (!in_array($fruit, $array)) { $array[] = $fruit; }
}
```

Решение для ассоциативных массивов:

```
foreach ($_GET['fruits'] as $fruit) {
    $array[$fruit] = $fruit;
}
```

Комментарий

После завершения обработки для удаления дубликатов лучше всего использовать `array_unique()`. В цикле обработки для устранения дубликатов можно проверить, присутствует ли элемент с таким значением в массиве.

Существует еще одно решение, превосходящее по скорости `in_array()`: создайте гибридный массив, в котором ключи и значения элементов совпадают. При этом теряется линейная сложность `in_array()`, но появляется возможность использования функций массивов, работающих со значениями массивов вместо ключей.

Как выясняется, быстрее использовать метод ассоциативного массива, а затем вызвать `array_values()` для результата (или `array_keys()`, но `array_values()` работает чуть быстрее), чем создавать числовой массив напрямую и вызывать `in_array()`.

См. также

Документация по функции `array_unique()`.

4.22. Применение функции к каждому элементу массива

Задача

Требуется применить функцию или метод к каждому элементу массива. Таким образом выполняется одновременное преобразование всего набора данных.

Решение

Воспользуйтесь функцией `array_walk()`:

```
$names = array('firstname' => "Baba",
               'lastname'  => "O'Riley");

array_walk($names, function (&$value, $key) {
    $value = htmlentities($value, ENT_QUOTES);
});

foreach ($names as $name) {
    print "$name\n";
}

Baba
O'Riley
```

Для вложенных данных используется функция `array_walk_recursive()`:

```
$names = array('firstnames' => array("Baba", "Bill"),
               'lastnames'  => array("O'Riley", "O'Reilly"));
array_walk_recursive($names, function (&$value, $key) {
    $value = htmlentities($value, ENT_QUOTES);
});

foreach ($names as $nametypes) {
    foreach ($nametypes as $name) {
        print "$name\n";
    }
}

Baba
Bill
O'Riley
O'Reilly
```

Комментарий

Циклы часто используются для перебора элементов массива. Один из вариантов — перебор данных в цикле `foreach`. Также можно воспользоваться функцией `array_walk()`; эта функция получает массив и функцию обратного вызова, об-

рабатывающую элементы массива. Функция обратного вызова получает два параметра: значение и ключ. Она также может получать необязательный третий параметр с дополнительными данными, которые могут использоваться в функции обратного вызова.

Следующий пример обеспечивает кодирование всех данных в массиве `$names` по правилам HTML. Анонимная функция обратного вызова получает значения из массива, передает их `htmlentities()` для кодирования ключевых сущностей HTML и присваивает результат `$value`:

```
$names = array('firstname' => "Baba",
               'lastname'  => "O'Riley");

array_walk($names, function (&$value, $key) {
    $value = htmlentities($value, ENT_QUOTES);
});

foreach ($names as $name) {
    print "$name\n";
}

Baba
O'Riley
```

`Функция array_walk()` работает «на месте», а не возвращает измененную копию массива. А значит, если вы хотите изменить элементы, значения должны передаваться по ссылке. В таких случаях, как в приведенном примере, перед именем параметра ставится символ `&`. Тем не менее это необходимо только тогда, когда вы хотите изменить массив.

Если вы работаете с набором вложенных массивов, используйте функцию `array_walk_recursive()`:

```
$names = array('firstnames' => array("Baba", "Bill"),
               'lastnames'  => array("O'Riley", "O'Reilly"));
array_walk_recursive($names, function (&$value, $key) {
    $value = htmlentities($value, ENT_QUOTES);
});

foreach ($names as $nametypes) {
    foreach ($nametypes as $name) {
        print "$name\n";
    }
}

Baba
Bill
O'Riley
O'Reilly
```

Функция `array_walk_recursive()` передает функции обратного вызова только элементы, не являющиеся массивами, так что вам не придется изменять функцию обратного вызова при переходе с `array_walk()`.

См. также

Документация по функциям `array_walk()`, `array_walk_recursive()`, `htmlentities()` и анонимным функциям.

4.23. Вычисление объединения, пересечения и разности двух массивов

Задача

Имеется пара массивов. Требуется вычислить их объединение (все элементы), пересечение (элементы, входящие в *оба* массива) или разность (элементы, входящие в одно множество, но не в оба).

Решение

Вычисление объединения:

```
$union = array_unique(array_merge($a, $b));
```

Вычисление пересечения:

```
$intersection = array_intersect($a, $b);
```

Вычисление простой разности:

```
$difference = array_diff($a, $b);
```

Вычисление симметричной разности:

```
$difference = array_merge(array_diff($a, $b), array_diff($b, $a));
```

Комментарий

Многие необходимые компоненты этих вычислений уже встроены в PHP; все сводится лишь к применению их в правильной последовательности.

Чтобы вычислить объединение, вы создаете из двух массивов один большой массив, содержащий все значения. Но функция `array_merge()` допускает дубликаты при слиянии двух числовых массивов, поэтому мы вызываем `array_unique()` для их устранения. Это может привести к появлению «дыр», потому что `array_unique()` не выполняет уплотнения массива. Впрочем, это не создает серьезных проблем, потому что циклы `foreach` и `each()` легко справляются с неплотно заполненными массивами.

Функция вычисления пересечения называется `array_intersection()` и не требует никакой дополнительной работы с вашей стороны.

Функция `array_diff()` возвращает массив, содержащий все уникальные элементы `$old`, отсутствующие в `$new`. Это называется *простой разностью*:

```
$old = array('To', 'be', 'or', 'not', 'to', 'be');
$new = array('To', 'be', 'or', 'whatever');
$difference = array_diff($old, $new);
print_r($difference);
```

Итоговый массив `$difference` содержит `'not'` и `'to'`, потому что `array_diff()` учитывает регистр символов. Значение `'whatever'` в результат не входит, потому что оно отсутствует в `$old`.

Чтобы вычислить обратную разность (то есть, другими словами, найти уникальные элементы `$new`, отсутствующие в `$old`), поменяйте аргументы местами:

```
$old = array('To', 'be', 'or', 'not', 'to', 'be');
$new = array('To', 'be', 'or', 'whatever');
$reverse_diff = array_diff($new, $old);
print_r($reverse_diff);
```

Массив `$reverse_diff` содержит только значение `'whatever'`.

Если вы хотите применить к `array_diff()` функцию или другой фильтр, напишите собственный алгоритм вычисления разности:

```
// Реализация вычисления разности без учета регистра; diff -i

$seen = array();
foreach ($new as $n) {
    $seen[strtolower($n)]++;
}

foreach ($old as $o) {
    $o = strtolower($o);
    if (!$seen[$o]) { $diff[$o] = $o; }
}
```

Первый цикл `foreach` строит ассоциативный массив для хранения таблицы поиска. Далее в цикле перебираются элементы `$old`, и если элемент не удается найти в таблице, то он добавляется в `$diff`.

Возможно, сочетание `array_diff()` с `array_map()` немного ускорит работу:

```
$diff = array_diff(array_map('strtolower', $old), array_map('strtolower', $new));
```

При симметричной разности в результат включаются элементы, которые присутствуют в `$a`, но отсутствуют в `$b`, а также те, которые присутствуют в `$b`, но отсутствуют в `$a`:

```
$difference = array_merge(array_diff($a, $b), array_diff($b, $a));
```

При такой формулировке задачи алгоритм реализуется элементарно. Вы дважды вызываете `array_diff()` и находите две разности, после чего выполняете их слияние в один массив. Вызывать `array_unique()` не нужно, потому что массивы сознательно строились так, чтобы в них не было совпадающих элементов.

См. также

Документация по функциям `array_unique()`, `array_intersect()`, `array_diff()`, `array_merge()` и `array_map()`.

4.24. Эффективный перебор больших или высокозатратных наборов данных

Задача

Требуется перебрать содержимое списка данных. При этом весь список занимает много памяти или очень медленно строится.

Решение

Воспользуйтесь генератором:

```
function FileLineGenerator($file) {
    if (!$fh = fopen($file, 'r')) {
        return;
    }

    while (false !== ($line = fgets($fh))) {
        yield $line;
    }
    fclose($fh);
}

$file = FileLineGenerator('log.txt');
foreach ($file as $line) {
    if (preg_match('/^rasmus: /', $line)) { print $line; }
}
```

Комментарий

Генераторы предоставляют простой механизм эффективного перебора без затрат, связанных с загрузкой всех данных в память. Поддержка генераторов появилась в PHP 5.5.

Генератор представляет собой функцию, которая возвращает итеративный объект. При переборе по объекту PHP многократно вызывает генератор для получения следующего значения, которое возвращается функцией-генератором при помощи ключевого слова `yield`.

В отличие от нормальных функций, которые каждый раз начинают выполнение с нуля, PHP сохраняет текущее состояние функции между вызовами генератора. Это позволяет сохранить любую информацию, необходимую для получения следующего генерируемого значения.

Если данных больше нет, функция просто возвращает управление без команды `return`, даже пустой (попытка вызова `return` в генераторе недопустима).

Идеальный пример использования генератора — обработка всех строк файла. Проще всего воспользоваться функцией `file()`: она открывает файл, загружает каждую строку в элемент массива и закрывает файл. Однако при этом весь файл хранится в памяти.

```
$file = file('log.txt');
foreach ($file as $line) {
    if (preg_match('/^rasmus: /', $line)) { print $line; }
}
```

Другой вариант — использовать стандартные функции чтения файлов с передачей управления вашему коду, который обрабатывает каждую прочитанную строку. Обычно такой код плохо читается и непригоден для повторного использования:

```
function print_matching_lines($file, $regex) {
    if (!$fh = fopen('log.txt','r')) {
        return;
    }
    while(false !== ($line = fgets($fh))) {
        if (preg_match($regex, $line)) { print $line; }
    }
    fclose($fh);
}

print_matching_lines('log.txt', '/^rasmus: /');
```

Но если упаковать код обработки файла в генератор, вы получаете и то и другое — обобщенную функцию для эффективного перебора строк файла и четкость синтаксиса, как если бы все данные хранились в массиве:

```
function FileLineGenerator($file) {
    if (!$fh = fopen($file, 'r')) {
        return;
    }

    while (false !== ($line = fgets($fh))) {
        yield $line;
    }

    fclose($fh);
}

$file = FileLineGenerator('log.txt');
foreach ($file as $line) {
    if (preg_match('/^rasmus: /', $line)) { print $line; }
}
```

В генераторе управление передается между циклом и функцией командой `yield`. При первом вызове генератора выполнение начинается от начала функции и приостанавливается при достижении команды `yield`, возвращающей значение.

В приведенном примере функция-генератор `FileLineGenerator()` перебирает строки файла. После открытия файла функция `fgets()` вызывается в цикле. Пока

еще остаются непрочитанные строки, цикл возвращает итератору `$line`. В конце файла цикл останавливается, файл закрывается, а функция завершается. Так как `yield` не выполняется, происходит выход из `foreach`.

Теперь функция `FileLineGenerator()` может использоваться везде, где потребуется выполнить перебор строк файла. Приведенный пример выводит строки, начинающиеся с префикса `rasmus:`. Следующий пример выводит случайную строку из файла:

```
$line_number = 0;
foreach (FileLineGenerator('sayings.txt') as $line) {
    $line_number++;
    if (mt_rand(0, $line_number - 1) == 0) {
        $selected = $line;
    }
}

print $selected . "\n";
```

Как видите, в совершенно другом сценарии использования `FileLineGenerator()` используется повторно без каких-либо изменений. На этот раз генератор не сохраняется в переменной, а вызывается из цикла `foreach`.

«Вернуть» генератор обратно невозможно. Генераторы работают только в одном направлении.

См. также

Рецепт 4.4 — методы перебора; глава 24 — чтение из файлов; документация по генераторам.

4.25. Работа с объектом в синтаксисе массива

Задача

Имеется объект; требуется читать и записывать в него данные так, как если бы он был массивом. Это позволило бы объединить преимущества объектно-ориентированного проектирования со знакомым интерфейсом массива.

Решение

Реализуйте интерфейс SPL `ArrayAccess`:

```
class FakeArray implements ArrayAccess {

    private $elements;

    public function __construct() {
        $this->elements = array();
    }

    public function offsetExists($offset) {
```

```
        return isset($this->elements[$offset]);
    }

    public function offsetGet($offset) {
        return $this->elements[$offset];
    }

    public function offsetSet($offset, $value) {
        return $this->elements[$offset] = $value;
    }

    public function offsetUnset($offset) {
        unset($this->elements[$offset]);
    }
}

$array = new FakeArray;

$array['animal'] = 'wabbit';

if (isset($array['animal']) &&
    $array['animal'] == 'wabbit') {

    unset($array['animal']);
}

if (!isset($array['animal'])) {
    print "Well, what did you expect in an opera? A happy ending?\n";
}
```

Комментарий

Интерфейс `ArrayAccess` позволяет работать с данными объекта по тем же принципам, что и с данными массивов. Это позволяет использовать преимущества объектно-ориентированного проектирования (такие как иерархии классов или реализация дополнительных методов в объектах), но при этом дать возможность пользователям работать с объектом в знакомом интерфейсе. Кроме того, появляется возможность создания «псевдомассива», хранящего свои данные на внешнем носителе (например, в общей памяти или в базе данных).

Реализация `ArrayAccess` состоит из четырех методов: `offsetExists()`, проверяющего, определен ли элемент; `offsetGet()`, возвращающего значение элемента; `offsetSet()`, присваивающего элементу новое значение; и `offsetUnset()`, удаляющего элемент и его значение.

В следующем примере данные хранятся локально в свойстве объекта:

```
class FakeArray implements ArrayAccess {

    private $elements;

    public function __construct() {
        $this->elements = array();
    }

    public function offsetExists($offset) {
        return isset($this->elements[$offset]);
    }
```

```
    public function offsetGet($offset) {
        return $this->elements[$offset];
    }

    public function offsetSet($offset, $value) {
        return $this->elements[$offset] = $value;
    }

    public function offsetUnset($offset) {
        unset($this->elements[$offset]);
    }
}
```

Конструктор объекта инициализирует свойство `$elements` новым массивом. В нем будут храниться ключи и значения. Это свойство определяется как закрытое (`private`), так что с данными можно работать только через методы доступа, определенные в интерфейсе.

Следующие четыре метода реализуют все необходимое для работы с массивом. Так как метод `offsetExists()` должен проверять, задано ли значение элемента массива, он возвращает значение `isset($this->elements[$offset])`.

Методы `offsetGet()` и `offsetSet()` взаимодействуют со свойством `$elements` так, как это обычно делается с массивом.

Наконец, метод `offsetUnset()` просто вызывает `unset()` для элемента. В отличие от других трех методов, операция не возвращает значения. Это объясняется тем, что `unset()` — команда, а не функция.

Теперь можно создать экземпляр `FakeArray` и работать с ним как с массивом:

```
$array = new FakeArray;

$array['animal'] = 'wabbit';

if (isset($array['animal']) &&
    $array['animal'] == 'wabbit') {

    unset($array['animal']);
}

if (!isset($array['animal'])) {
    print "Well, what did you expect in an opera? A happy ending?\n";
}
```

При каждой операции вызывается один из методов: присваивание значения `$array['animal']` приводит к выполнению `offsetSet()`, проверка `isset($array['animal'])` активизирует `offsetExists()`, `offsetGet()` вступает в игру при сравнении `$array['animal'] == 'wabbit'`, а `offsetUnset()` вызывается для `unset($array['animal'])`.

См. также

Глава 7 — дополнительная информация об объектах; описание интерфейса `ArrayAccess`.

5 Переменные

5.0. Введение

Переменные, наряду с условной логикой, являются одним из ключевых факторов мощи и гибкости компьютерных программ. Если рассматривать переменную как «ведро», на котором написано имя, а внутри лежит некое значение, PHP позволяет создавать обычные ведра; ведра, в которых лежат имена других ведер; ведра, в которых лежат числа и строки; ведра с массивами других ведер, ведра с объектами... и вообще практически любые разновидности этой аналогии, которые только можно себе представить.

Переменная может находиться в одном из двух значений: определенном или неопределенном. Переменная, которой присвоено любое значение (истинное или ложное, пустое или непустое и т. д.), является определенной. Функция `isset()` возвращает `true`, если переданная ей переменная определена. Чтобы превратить определенную переменную в неопределенную, вызовите для нее `unset()` или присвойте `null`. Функции `unset()` могут передаваться скалярные значения, массивы и объекты. Также можно передать `unset()` сразу несколько переменных, чтобы все они стали неопределенными:

```
unset($vegetables);
unset($fruits[12]);
unset($earth, $moon, $stars);
```

Если переменная присутствует в строке запроса URL-адреса, то она определяется в соответствующем суперглобальном массиве, даже если ей не присвоено значение. Так, адрес

http://www.example.com/set.php?chimps=&monkeys=12

присваивает переменной `$_GET['monkeys']` значение 12, а переменной `$_GET['chimps']` присваивается пустая строка.

Все неопределенные переменные также пусты. Определенные переменные могут быть как пустыми, так и непустыми. Пустые переменные содержат значения, которые интерпретируются как `false` в логическом контексте. Эти значения перечислены в таблице 5.1.

Таблица 5.1. Значения, интерпретируемые как false

Тип	Значение
integer	0
double	0.0
string	" " (пустая строка)
string	"0"
boolean	false
array	array() (пустой массив)
null	NULL
object	Объект без свойств (только до выхода PHP 5)

Все значения, не представленные в таблице 5.1, не являются пустыми. В эту категорию входит и строка `"00"`, и строка `" "`, содержащая только пробел.

В версии 5.5 функции empty() можно передавать произвольные выражения.

Значение переменной может интерпретироваться как логическая истина (`true`) или логическая ложь (`false`). В таблице 5.1 перечислены все значения, интерпретируемые в PHP как `false`. Все остальные значения истинны. Языковая конструкция `isset()` проверяет, является ли переменная определенной. Языковая конструкция `empty()` проверяет, является ли значение пустым. До выхода PHP версии 5.5 в аргументах `empty()` могли передаваться только переменные. В PHP 5.5 появилась возможность передачи `empty()` произвольных выражений.

Константы и возвращаемые значения функций могут быть ложными, но до выхода PHP 5.5 они не могли быть пустыми. Например, в листинге 5.1 представлен допустимый пример использования `empty()` (в любой версии PHP), потому что `$first_name` является переменной.

Листинг 5.1. Допустимая проверка переменной

```
if (empty($first_name)) { .. }
```

С другой стороны, в версиях PHP до 5.5 код листинга 5.2 возвращает ошибку разбора, потому что 0 (константа) и возвращаемое значение `get_first_name()` не могут быть пустыми.

Листинг 5.2. Недопустимая проверка до PHP 5.5

```
if (empty(0)) { .. }
if (empty(get_first_name())) { .. }
```

5.1. Предотвращение путаницы между == и =

Задача

Требуется избежать случайного присваивания значений при сравнении переменной и константы.

Решение

Используйте конструкцию:

```
if (12 == $dwarves) { ... }
```

вместо:

```
if ($dwarves == 12) { ... }
```

Если константа находится в левой части, для оператора присваивания будет выдана ошибка разбора. Иначе говоря, PHP выдает ошибку для следующего фрагмента:

```
if (12 = $dwarves) { ... }
```

но фрагмент

```
if ($dwarves = 12) { ... }
```

будет нормально выполнен: переменной `$dwarves` присваивается значение 12, после чего будет выполнен код внутри блока. (Результат `$dwarves = 12` равен 12, что интерпретируется как `true`.)

Комментарий

Размещение константы в левой части оператора сравнения приводит сравнение к типу константы. Это может создать проблемы при сравнении целого числа с переменной, которая может быть как целым числом, так и строкой. Выражение `0 == $dwarves` истинно, если переменная `$dwarves` равна 0, но оно также истинно, если `$dwarves` содержит строку `sleepy`. Из-за того что слева от оператора сравнения находится целое число (0), PHP преобразует правую часть (строку `sleepy`) в целое число (0) перед сравнением. Чтобы этого не происходило, используйте оператор тождественности `0 === $dwarves`.

См. также

Документация по оператору =; *www.php.net/operators.comparison* (== и ===).

5.2. Определение значения по умолчанию

Задача

Требуется определить значение по умолчанию для переменной, которой еще не было назначено значение. Например, переменной назначается фиксированное значение по умолчанию, которое может переопределяться данными, вводимыми пользователем, или переменной окружения.

Решение

Используйте функцию `isset()` для определения значения по умолчанию для переменной, которая уже может иметь значение:

```
if (! isset($cars)) {
    $cars = $default_cars;
}
```

Используйте тернарный оператор `(a ? b : c)` для того, чтобы задать новой переменной значение (возможно, значение по умолчанию):

```
$cars = isset($_GET['cars']) ? $_GET['cars'] : $default_cars;
```

Комментарий

Функция `isset()` играет важную роль при определении значений по умолчанию. Без нее значение, не являющееся значением по умолчанию, не может быть равно 0 или чему-либо другому, интерпретируемому как `false`. Рассмотрим следующую команду присваивания:

```
$cars = isset($_GET['cars']) ? $_GET['cars'] : $default_cars;
```

Если `$_GET['cars']` содержит 0, то `$cars` присваивается `$default_cars`, хотя 0 вполне может быть действительным значением `$cars`.

Альтернативный синтаксис проверки для массивов основан на использовании функции `array_key_exists()`:

```
$cars = array_key_exists('cars', $_GET) ? $_GET['cars'] : $default_cars;
```

Единственное различие между `isset()` и `array_key_exists()` заключается в том, что если ключ существует, но содержит `null`, то `array_key_exists()` возвращает `true`, тогда как `isset()` возвращает `false`:

```
$vehicles = array('cars' => null);
// array_key_exists() возвращает TRUE, потому что ключ присутствует.
$ake_result = array_key_exists('cars', $vehicles);

// isset() возвращает false, потому что значение ключа равно NULL
$isset_result = isset($vehicles['cars']);
```

Чтобы легко задать несколько значений по умолчанию, используйте массив. Ключами этого массива являются имена переменных, а значениями — значения по умолчанию для каждой переменной:

```
$defaults = array('emperors'  => array('Rudolf II','Caligula'),
                  'vegetable' => 'celery',
                  'acres'     => 15);

foreach ($defaults as $k => $v) {
    if (! isset($GLOBALS[$k])) { $GLOBALS[$k] = $v; }
}
```

Так как переменные задаются в глобальном пространстве имен, приведенный код не подойдет для назначения закрытых переменных внутри функции. Для этого используются *косвенные* переменные:

```
foreach ($defaults as $k => $v) {
    if (! isset($$k)) { $$k = $v; }
}
```

В этом примере при первой итерации цикла `$k` содержит строку `emperors`, поэтому запись `$$k` соответствует `$emperors`.

См. также

Документация по функциям `isset()`, `array_key_exists()` и косвенные переменные.

5.3. Переключение значений без использования временных переменных

Задача

Требуется поменять местами значения двух переменных без использования дополнительных переменных для промежуточного хранения данных.

Решение

Поменять местами значения `$a` и `$b` можно так:

```
$a = 'Alice';
$b = 'Bob';
```

```
list($a,$b) = array($b,$a);
// Теперь $a содержит 'Bob', а $b содержит 'Alice'
```

Комментарий

Языковая конструкция PHP `list()` позволяет присваивать значения из массива отдельным переменным. Правая часть команды, `array()`, позволяет конструировать массивы из отдельных значений. Присваивание массива, возвращаемого `array()`, переменным из `list()` позволяет менять порядок этих значений. Такое решение работает не только с двумя, но и с большим количеством значений:

```
$yesterday = 'pleasure';
$today = 'sorrow';
$tomorrow = 'celebrate';

list($yesterday,$today,$tomorrow) = array($today,$tomorrow,$yesterday);
// Теперь $yesterday содержит 'sorrow', $today содержит 'celebrate'
// а $tomorrow содержит 'pleasure'
```

Этот способ не дает выигрыша в скорости и применяется в основном для ясности кода.

См. также

Документация по функциям `list()` и `array()`.

5.4. Динамическое создание имени переменной

Задача

Требуется построить имя переменной динамически. Например, нужно использовать имена переменных, соответствующие именам полей из базы данных.

Решение

Используйте синтаксис косвенных переменных PHP — поставьте `$` перед переменной, содержащей имя нужной вам переменной:

```
$animal = 'turtles';
$turtles = 103;
print $$animal;
```

Результат выглядит так:

```
103
```

Комментарий

Если перед именем переменной стоят два знака `$`, PHP берет значение этой переменной и использует его как имя «итоговой» переменной. В приведенном выше примере выводится значение 103, потому что `$animal` содержит `'turtles'`; следовательно, `$$animal` соответствует `$turtles`, а значение этой переменной равно 103.

Синтаксис `{...}` позволяет строить более сложные выражения, обозначающие имена переменных:

```
$stooges = array('Moe','Larry','Curly');
$stooge_moe = 'Moses Horwitz';
$stooge_larry = 'Louis Feinberg';
$stooge_curly = 'Jerome Horwitz';

foreach ($stooges as $s) {
  print "$s's real name was ${'stooge_'.strtolower($s)}.\n";
}
```

PHP вычисляет выражение в фигурных скобках и использует его как имя переменной. В этом выражении даже могут присутствовать вызовы функций (такие как `strtolower()`).

Синтаксис косвенных переменных также удобен при переборе переменных с именами, построенными по одной схеме. Допустим, вы выдаете запрос к таблице базы данных, содержащей поля с именами `title_1`, `title_2` и т. д. Если вы хотите проверить, совпадает ли имеющаяся строка с одним из этих полей, проще всего перебрать их следующим образом:

```
for ($i = 1; $i <= $n; $i++) {
    $t = "title_$i";
    if ($title == $$t) { /* Есть совпадение */ }
}
```

Конечно, такие значения проще сохранить в массиве, но если вы занимаетесь сопровождением старого кода, в котором используется этот прием (и этот код нельзя изменять), косвенные переменные будут полезны.

Синтаксис с фигурными скобками также необходим при разрешении неоднозначностей с элементами массивов. Смысл косвенной переменной `$$donkeys[12]` неочевиден: во-первых, это может быть 12-й элемент массива `$donkeys`, который используется как имя переменной; это можно записать в виде `${$donkeys[12]}`. Во-вторых, это может быть использование скалярного значения переменной `$donkeys` как имени массива, с обращением к 12-му элементу этого массива. Этот вариант записывается в виде `${$donkeys}[12]`.

Синтаксис не ограничивается двумя знаками `$` — их может быть три и больше, но на практике более двух уровней косвенности встречаются редко.

См. также

Документация по косвенным переменным.

5.5. Сохранение значения локальной переменной между вызовами функции

Задача

Текущее значение локальной переменной должно сохраняться между вызовами функции.

Решение

Объявите статическую (`static`) переменную:

```
function track_times_called() {
    static $i = 0;
    $i++;
    return $i;
}
```

Комментарий

Переменная, объявленная внутри функции как статическая, сохраняет свое значение между вызовами, и при последующих вызовах функции вы сможете работать с сохраненным значением. Пример использования статической переменной представлен в функции `check_the_count()` из листинга 5.3.

Листинг 5.3. check_the_count()

```
function check_the_count($pitch) {
    static $strikes = 0;
    static $balls   = 0;

    switch ($pitch) {
    case 'foul':
        if (2 == $strikes) break; // Ничего не происходит
    case 'strike':
        $strikes++;
        break;
    case 'ball':
        $balls++;
        break;
    }

    if (3 == $strikes) {
        $strikes = $balls = 0;
        return 'strike out';
    }
    if (4 == $balls) {
        $strikes = $balls = 0;
        return 'walk';
    }
    return 'at bat';
```

```
}

$pitches = array('strike', 'ball', 'ball', 'strike', 'foul','strike');
$what_happened = array();
foreach ($pitches as $pitch) {
    $what_happened[] = check_the_count($pitch);
}

// Вывод результатов
var_dump($what_happened);
```

Код листинга 5.3 выводит следующий результат:

```
array(6) {
  [0]=>
  string(6) "at bat"
  [1]=>
  string(6) "at bat"
  [2]=>
  string(6) "at bat"
  [3]=>
  string(6) "at bat"
  [4]=>
  string(6) "at bat"
  [5]=>
  string(10) "strike out"
}
```

Хотя статические переменные сохраняют значения между вызовами функции, это происходит только в границах одного выполнения сценария. Статическая переменная, с которой вы работали в одном запросе, не сохранит свое значение в следующем запросе той же страницы.

См. также

Документация по статическим переменным.

5.6. Совместный доступ к переменным между процессами

Задача

Требуется организовать совместное использование информации процессами, с быстрым доступом к общим данным.

Решение

Воспользуйтесь функциональностью хранилища данных из расширения APC, как показано в листинге 5.4.

Листинг 5.4. Работа с хранилищем данных APC

```
// Получение старого значения
$population = apc_fetch('population');
// Работа с данными
$population += ($births + $immigrants - $deaths - $emigrants);
// Запись нового значения на прежнее место
apc_store('population', $population);
```

Если расширение APC недоступно, воспользуйтесь одним из двух расширений для работы с общей памятью, входящих в поставку PHP: shmop или System V shared memory.

В shmop вы создаете блок для чтения и записи данных, как показано в листинге 5.5.

Листинг 5.5. Использование функций общей памяти shmop

```
// Создание ключа
$shmop_key = ftok(__FILE__, 'p');
// Создание блока общей памяти из 16384 байт
$shmop_id = shmop_open($shmop_key, "c", 0600, 16384);
// Чтение всего сегмента общей памяти
$population = shmop_read($shmop_id, 0, 0);
// Работа с данными
$population += ($births + $immigrants - $deaths - $emigrants);
// Сохранение значения в сегменте общей памяти
$shmop_bytes_written = shmop_write($shmop_id, $population, 0);
// Проверяем, что данные успешно поместились в сегменте
if ($shmop_bytes_written != strlen($population)) {
   echo "Can't write all of: $population\n";
}
// Закрытие дескриптора
shmop_close($shmop_id);
```

При использовании расширения System V shared memory данные хранятся в общем сегменте памяти, а для обеспечения монопольного доступа к общей памяти используется семафор, как показано в листинге 5.6.

Листинг 5.6. Использование System V shared memory

```
$semaphore_id = 100;
$segment_id   = 200;
// Получение дескриптора семафора, связанного с используемым
// сегментом общей памяти
$sem = sem_get($semaphore_id,1,0600);
// Получение монопольного доступа к семафору
sem_acquire($sem) or die("Can't acquire semaphore");
// Получение дескриптора сегмента общей памяти
$shm = shm_attach($segment_id,16384,0600);
// Каждое значение, хранящееся в сегменте,
// представлено целочисленным идентификатором
$var_id = 3476;
// Получение значения из сегмента общей памяти
if (shm_has_var($shm, $var_id)) {
    $population = shm_get_var($shm,$var_id);
}
```

```
// Или инициализация, если значение еще не было присвоено
else {
    $population = 0;
}
// Работа со значением
$population += ($births + $immigrants - $deaths - $emigrants);
// Сохранение значения в сегменте общей памяти
shm_put_var($shm,$var_id,$population);
// Освобождение дескриптора сегмента общей памяти
shm_detach($shm);
// Освобождение семафора, чтобы другие процессы могли захватить его
sem_release($sem);
```

Комментарий

Если расширение APC доступно, то хранилище данных позволяет исключительно удобно организовать обмен информацией между процессами PHP для разных запросов. Функция `apc_store()` получает ключ и значение и сохраняет значение, связанное с указанным ключом. Также в третьем аргументе `apc_store()` может передаваться необязательное время жизни (TTL, Time To Live) для ограничения продолжительности хранения значения в кэше в секундах.

Чтобы прочитать сохраненные данные, передайте ключ функции `apc_fetch()`. Поскольку `apc_fetch()` возвращает сохраненное значение или `false` в случае неудачи, результат успешного вызова, вернувший `false`, может быть трудно отличить от неудачного вызова. Для таких случаев `apc_fetch()` поддерживает второй аргумент, передаваемый по ссылке; в нем содержится признак успешного вызова (`true` или `false`):

```
// Тест не пройден!
apc_store('passed the test?', false);

// $results содержит false, потому что хранимое значение ложно.
// $success содержит true, потому что вызов apc_fetch() завершился успешно.
$results = apc_fetch('passed the test?', $success);
```

Кроме хранения и выборки данных, APC также обеспечивает возможность более сложных операций с данными. Функции `apc_inc()` и `apc_dec()` выполняют атомарный инкремент и декремент хранимого числа, что делает их очень удобными для реализации быстрых счетчиков. Также можно реализовать упрощенную блокировку с использованием функции `apc_add()`, которая вставляет переменную в хранилище данных только в том случае, если значение с указанным ключом еще не существует. В листинге 5.7 показано, как это делается.

Листинг 5.7. Использование apc_add() для реализации блокировки

```
function update_recent_users($current_user) {
    $recent_users = apc_fetch('recent-users', $success);
    if ($success) {
        if (! in_array($current_user, $recent_users)) {
            array_unshift($recent_users, $current_user);
```

```
        }
    }
    else {
        $recent_users = array($current_user);
    }
    $recent_users = array_slice($recent_users, 0, 10);
    apc_store('recent-users', $recent_users);
}

$tries = 3;
$done = false;

while ((! $done) && ($tries-- > 0)) {
    if (apc_add('my-lock', true, 5)) {
        update_recent_users($current_user);
        apc_delete('my-lock');
        $done = true;
    }
}
```

В листинге 5.7 вызов `apc_add('my-lock', true, 5)` означает «Вставить значение `true` с ключом `my-lock` только в том случае, если оно еще не существует, и ограничить срок его жизни пятью секундами». Итак, если операция выполняется успешно, то все последующие запросы, которые попытаются сделать то же самое (в ближайшие пять секунд), завершатся неудачей, пока вызов `apc_delete('my-lock')` в первом запросе не удалит запись из хранилища данных. Например, вызов `update_recent_users()` в цикле, как в приведенном примере, ведет массив десяти последних пользователей. Цикл трижды пытается установить блокировку, после чего завершается.

Если расширение APC недоступно, используйте расширение общей памяти для реализации совместного доступа к данным (хотя это и потребует большего объема работы).

Сегмент общей памяти представляет собой блок оперативной памяти, к которому могут обращаться разные процессы (например, процессы веб-сервера, занимающиеся обработкой запросов). Расширения shmop и System V shared memory решают аналогичную задачу, позволяя сохранять информацию и быстро и эффективно использовать ее в разных запросах, но они работают по-разному и в результате используют несколько отличающиеся интерфейсы.

Интерфейс функций shmop напоминает традиционные операции с файлами. Вы открываете сегмент, читаете данные, записываете их и закрываете сегмент. Как и в случае с файлами, встроенная структура в данных отсутствует — вы работаете с последовательностью символов. В листинге 5.5 сначала создается общий блок памяти. В отличие от файла, необходимо заранее объявить максимальный размер блока. В данном примере он составляет 16 384 байта:

```
// Создание ключа
$shmop_key = ftok(__FILE__, 'p');
// Создание блока общей памяти с размером 16 384 байта
$shmop_id = shmop_open($shmop_key, "c", 0600, 16384);
```

Если файлы различаются по именам, то сегменты shmop различаются по ключам. В отличие от имен файлов, ключи представляют собой не строки, а целые числа, и запомнить их нелегко. По этой причине лучше воспользоваться функцией `ftok()` для преобразования удобного и понятного имени (в данном примере имени файла в форме `__FILE__`) в формат, подходящий для `shmop_open()`. Функция `ftok()` также получает односимвольный идентификатор проекта. Это позволяет предотвратить конфликты при случайном повторном использовании той же строки. В данном примере используется символ `p` (сокращение от PHP).

Функции `shmop_create()` передается ключ, флаг, разрешения (в восьмеричной форме) и размер блока. Список флагов приведен в таблице 5.2. Разрешения работают по тем же принципам, что и файловые разрешения; значение 0600 означает, что пользователь, создавший блок, может выполнять операции чтения и записи с ним. В этом контексте под «пользователем» имеется в виду не только процесс, создавший семафор, но и любой процесс с тем же идентификатором пользователя. Разрешения 0600 хорошо подходят для большинства ситуаций, в которых процессы веб-сервера работают от имени одного пользователя.

Таблица 5.2. Флаги shmop_open()

Флаг	Описание
a	Открыть только для чтения.
c	Создать новый сегмент. Если сегмент уже существует, он открывается для чтения и записи.
w	Открыть для чтения и записи.
n	Создать новый сегмент. Если сегмент уже существует, происходит ошибка. Режим полезен для предотвращения ситуаций «гонки».

Получив дескриптор, вы можете прочитать данные из сегмента функцией `shmop_read()` и работать с данными:

```
// Чтение всего сегмента памяти
$population = shmop_read($shmop_id, 0, 0);
// Обработка данных
$population += ($births + $immigrants - $deaths - $emigrants);
```

Этот код читает весь сегмент. Чтобы прочитать меньшее количество данных, измените второй и третий параметры. Второй параметр определяет начальную позицию, а третий — длину читаемых данных. Если длина равна 0, данные читаются до конца сегмента.

Чтобы сохранить модифицированные данные, вызовите функцию `shmop_write()` и освободите дескриптор функцией `shmop_close()`:

```
// Сохранение значения в сегменте общей памяти
$shmop_bytes_written = shmop_write($shmop_id, $population, 0);
// Проверяем, что данные успешно поместились в сегменте
if ($shmop_bytes_written != strlen($population)) {
```

```
   echo "Can't write all of: $population\n";
}
// Закрытие дескриптора
shmop_close($shmop_id);
```

Сегменты общей памяти имеют фиксированную длину. Если не принять меры предосторожности, вы можете попытаться записать больше данных, чем позволяет свободное место. Чтобы выявить такую ситуацию, сравните значение, возвращаемое `shmop_write()`, с длиной данных. Значения должны совпасть. Если функция `shmop_write()` вернула меньшее значение, значит, ей удалось записать это количество байт, прежде чем в сегменте кончилось место.

В отличие от shmop, функции System V shared memory похожи на функции массивов. Чтобы обратиться к части сегмента, вы указываете ключ, после чего операции с данными выполняются напрямую. В зависимости от сохраняемых данных этот прямой доступ может оказаться более удобным.

С другой стороны, это приводит к усложнению интерфейса; кроме того, System V shared memory также требует организовать управление блокировкой через семафор.

Семафор следит за тем, чтобы разные процессы не мешали друг другу при обращении к сегменту общей памяти. Прежде чем процесс сможет использовать сегмент, он должен захватить семафор. Завершив работу с сегментом, процесс освобождает семафор, и тот становится доступным для других процессов.

Чтобы захватить семафор, воспользуйтесь функцией `sem_get()` для получения идентификатора семафора. Первый аргумент `sem_get()` содержит целочисленный ключ семафора. В качестве ключа можно использовать любое целое число при условии, что все программы, которым нужен доступ к этому конкретному семафору, будут использовать один ключ. Если семафор с заданным ключом еще не существует, он будет создан; максимальное количество процессов, которые могут обращаться к семафору, определяется вторым аргументом `sem_get()` (в данном случае 1); разрешения доступа к семафору определяются третьим аргументом `sem_get()` (0600). Разрешения работают так же, как для файлов и shmop, например:

```
$semaphore_id = 100;
$segment_id   = 200;
// Получение дескриптора семафора, связанного с используемым
// сегментом общей памяти
$sem = sem_get($semaphore_id,1,0600);
// Получение монопольного доступа к семафору
sem_acquire($sem) or die("Can't acquire semaphore");
```

Функция `sem_get()` возвращает идентификатор, ссылающийся на системный семафор. Идентификатор используется для захвата семафора функцией `sem_acquire()`. Функция ожидает до тех пор, пока ей удастся взять семафор под контроль (возможно, для этого придется дождаться освобождения семафора другим процессом), после чего возвращает `true`. В случае ошибки возвращается

false. Возможными ошибками могут быть недействительные разрешения или нехватка памяти для создания семафора. После того как семафор будет захвачен, программа может читать данные из сегмента общей памяти:

```
// Получение дескриптора сегмента общей памяти
$shm = shm_attach($segment_id,16384,0600);
// Каждое значение, хранящееся в сегменте,
// представлено целочисленным идентификатором
$var_id = 3476;
// Получение значения из сегмента общей памяти
if (shm_has_var($shm, $var_id)) {
    $population = shm_get_var($shm,$var_id);
}
// Или инициализация, если значение еще не было присвоено
else {
    $population = 0;
}
// Работа со значением
$population += ($births + $immigrants - $deaths - $emigrants);
```

Сначала программа устанавливает связь с конкретным сегментом общей памяти вызовом shm_attach(). Как и в случае с sem_get(), первый аргумент shm_attach() содержит целочисленный ключ, однако на этот раз он определяет нужный сегмент, а не семафор. Если сегмент с заданным ключом не существует, он создается другими аргументами. Второй аргумент (16384) содержит размер сегмента в байтах, а последний аргумент (0600) определяет разрешения доступа для сегмента. Вызов shm_attach(200,16384,0600) создает 16-килобайтный сегмент общей памяти, доступный для чтения и записи только для пользователя, который его создал. Функция возвращает идентификатор, необходимый для чтения и записи в сегмент общей памяти.

После присоединения к сегменту для чтения переменных из него используется функция shm_get_var($shm, $var_id). Она обращается к сегменту, определяемому $shm, и читает значение переменной с целочисленным ключом $var_id. В общей памяти могут храниться любые переменные. После того как переменная будет прочитана, с ней можно работать так же, как с любой другой переменной. Вызов shm_put_var($shm, $var_id ,$population) записывает значение $population обратно в сегмент общей памяти, в переменную $var_id.

Работа с сегментом общей памяти завершена. Отсоединитесь от него вызовом shm_detach() и освободите семафор вызовом sem_release(), чтобы он мог использоваться другим процессом:

```
// Освобождение дескриптора сегмента общей памяти
shm_detach($shm);
// Освобождение семафора, чтобы другие процессы могли захватить его
sem_release($sem);
```

Главное преимущество этой технологии — скорость. Но поскольку данные хранятся в оперативной памяти, их объем ограничен, и данные не сохраняются при

перезагрузке (если только не принять специальных мер для записи информации на диск перед загрузкой и их повторной загрузки при запуске).

System V shared memory не может использоваться в Windows, но функции shmop работают нормально.

См. также

Документация по apc, shmop, System V shared memory и функциям семафоров.

5.7. Строковое представление сложных типов данных

Задача

Требуется получить строковое представление массива или объекта для сохранения в файле или базе данных. Такая строка должна обеспечивать простое восстановление исходного массива или объекта.

Решение

Воспользуйтесь функцией `serialize()` для *сериализации* данных, то есть кодирования переменных и их значений в текстовую форму:

```
$pantry = array('sugar' => '2 lbs.','butter' => '3 sticks');
$fp = fopen('/tmp/pantry','w') or die ("Can't open pantry");
fputs($fp,serialize($pantry));
fclose($fp);
```

Восстановление переменных осуществляется функцией `unserialize()`:

```
// $new_pantry будет содержать массив:
// array('sugar' => '2 lbs.','butter' => '3 sticks'
$new_pantry = unserialize(file_get_contents('/tmp/pantry'));
```

Чтобы упростить взаимодействие с другими языками (за счет некоторого снижения быстродействия), воспользуйтесь функцией `json_encode()` для сериализации данных:

```
$pantry = array('sugar' => '2 lbs.','butter' => '3 sticks');
$fp = fopen('/tmp/pantry.json','w') or die ("Can't open pantry");
fputs($fp,json_encode($pantry));
fclose($fp);
```

Закодированные данные восстанавливаются функцией `json_decode()`:

```
// $new_pantry будет содержать массив:
//  array('sugar' => '2 lbs.','butter' => '3 sticks')
$new_pantry = json_decode(file_get_contents('/tmp/pantry.json'), TRUE);
```

Комментарий

Сериализованная строка PHP в `$pantry` выглядит примерно так:

```
a:2:{s:5:"sugar";s:6:"2 lbs.";s:6:"butter";s:8:"3 sticks";}
```

Версия, закодированная по правилам JSON, выглядит так:

```
{"sugar":"2 lbs.","butter":"3 sticks"}
```

Дополнительные символы, отсутствующие в JSON-строке, содержат информацию о типах и длинах значений. Строка выглядит тяжеловесно, но зато немного быстрее декодируется. Если вы ограничиваетесь передачей данных между приложениями PHP, встроенной сериализации вполне достаточно. Если вам нужно работать с другими языками, используйте JSON.

И встроенная сериализация и JSON сохраняют достаточно информации для восстановления всех значений в массиве, но само имя переменной ни в одном из сериализованных форматов не сохраняется. JSON не умеет различать объекты и ассоциативные массивы в своем формате сериализации, поэтому при вызове `json_decode()` необходимо выбрать, что именно вам нужно. Если второй аргумент равен `true`, как в приведенном примере, вызов строит ассоциативный массив. Без аргумента то же представление JSON будет декодировано в объект класса `stdClass` с двумя свойствами: `sugar` и `butter`.

Если сериализованные данные передаются между страницами в URL-адресе, вызовите для них функцию `urlencode()`, чтобы обеспечить экранирование URL-метасимволов:

```
$shopping_cart = array('Poppy Seed Bagel' => 2,
                       'Plain Bagel' => 1,
                       'Lox' => 4);
print '<a href="next.php?cart='.urlencode(serialize($shopping_cart)).
       '">Next</a>';
```

Сериализованные данные, сохраняемые в базе данных, всегда должны экранироваться. В Рецепте 10.9 рассказано, как осуществить безопасное экранирование значений для вставки в базу данных.

При десериализации объекта PHP автоматически вызывает его метод `__wakeUp()`. Это позволяет объекту восстановить любое состояние, не сохраняемое между сериализациями (например, подключение к базе данных). Восстановление может привести к изменению окружения — вы должны хорошо понимать, что делаете. За дополнительной информацией обращайтесь к Рецепту 7.19.

См. также

Документация по функциям `serialize()`, `unserialize()`, `json_encode()` и `json_decode()`. Рецепт 10.9 — безопасная вставка значений в базу данных, Рецепт 7.19 — сериализация объектов.

5.8. Вывод содержимого переменной в строковом виде

Задача

Требуется проанализировать значения, хранящиеся в переменной. Переменная может содержать сложный многомерный массив или объект, поэтому просто вывести содержимое переменной или перебрать элементы в цикле не удастся.

Решение

Воспользуйтесь функцией `var_dump()`, `print_r()` или `var_export()` в зависимости от того, что именно нужно сделать. Функции `var_dump()` и `print_r()` строят представление переменных в формате, понятном для человека.

Вывод функции `print_r()` получается чуть более компактным:

```
$info = array('name' => 'frank', 12.6, array(3, 4));
print_r($info);
```

Результат выглядит так:

```
Array
(
    [name] => frank
    [0] => 12.6
    [1] => Array
        (
            [0] => 3
            [1] => 4
        )
)
```

С другой стороны, при выполнении этого фрагмента:

```
$info = array('name' => 'frank', 12.6, array(3, 4));
var_dump($info);
```

будет получен следующий результат:

```
array(3) {
  ["name"]=>
  string(5) "frank"
  [0]=>
  float(12.6)
  [1]=>
  array(2) {
    [0]=>
    int(3)
    [1]=>
    int(4)
  }
}
```

Функция `var_export()` генерирует код PHP, при выполнении которого определяется экспортируемая переменная:

```
$info = array('name' => 'frank', 12.6, array(3, 4));
var_export($info);
```

Результат:

```
array (
  'name' => 'frank',
  0 => 12.6,
  1 =>
  array (
    0 => 3,
    1 => 4,
  ),
)
```

Комментарий

Три функции, упоминаемые в Решении, отличаются прежде всего подходом к обработке рекурсии. Так как все функции рекурсивно обрабатывают содержимое переменных, если в переменной содержатся ссылки, указывающие на саму переменную, выполнение войдет в бесконечный цикл.

Если функция `var_dump()` или `print_r()` уже встречала переменную, то вместо вывода информации о переменной она выводит строку `*RECURSION*` и продолжает перебор остальной выводимой информации. Функция `var_export()` действует аналогично, но вместо `*RECURSION*` выводится `null`, чтобы выходные данные представляли собой корректный исполняемый код PHP.

Возьмем массивы `$user_1` и `$user_2`, каждый из которых ссылается на другой массив в элементе `friend`:

```
$user_1 = array('name' => 'Max Bialystock',
                'username' => 'max');
$user_2 = array('name' => 'Leo Bloom',
                'username' => 'leo');

// Макс и Лео - друзья
$user_2['friend'] = &$user_1;
$user_1['friend'] = &$user_2;

// У Макса и Лео есть работа
$user_1['job'] = 'Swindler';
$user_2['job'] = 'Accountant';
```

Вызов `print_r($user_2)` выводит следующий результат:

```
Array
(
    [name] => Leo Bloom
    [username] => leo
    [friend] => Array
```

```
        (
            [name] => Max Bialystock
            [username] => max
            [friend] => Array
                (
                    [name] => Leo Bloom
                    [username] => leo
                    [friend] => Array
 *RECURSION*
                    [job] => Accountant
                )
            [job] => Swindler
        )
    [job] => Accountant
)
```

Когда `print_r()` встречает ссылку на `$user_1` во второй раз, она выводит `*RECURSION*` вместо того, чтобы углубляться в массив. Далее функция продолжает работу, выводя оставшиеся элементы `$user_1` и `$user_2`. Функция `var_dump()` работает аналогично:

```
array(4) {
  ["name"]=>
  string(9) "Leo Bloom"
  ["username"]=>
  string(3) "leo"
  ["friend"]=>
  &array(4) {
    ["name"]=>
    string(14) "Max Bialystock"
    ["username"]=>
    string(3) "max"
    ["friend"]=>
    &array(4) {
      ["name"]=>
      string(9) "Leo Bloom"
      ["username"]=>
      string(3) "leo"
      ["friend"]=>
      *RECURSION*
      ["job"]=>
      string(10) "Accountant"
    }
    ["job"]=>
    string(8) "Swindler"
  }
  ["job"]=>
  string(10) "Accountant"
}
```

То же самое делает и функция `var_export()`, но она выводит `null` вместо `*RECURSION*`:

```
array (
  'name' => 'Leo Bloom',
```

```
  'username' => 'leo',
  'friend' =>
  array (
    'name' => 'Max Bialystock',
    'username' => 'max',
    'friend' =>
    array (
      'name' => 'Leo Bloom',
      'username' => 'leo',
      'friend' => NULL,
      'job' => 'Accountant',
    ),
    'job' => 'Swindler',
  ),
  'job' => 'Accountant',
)
```

Функции `print_r()` и `var_export()` получают второй аргумент; если он равен `true`, то функция возвращает строковое представление переменной вместо того, чтобы отправлять его на вывод. Чтобы сохранить результаты работы `var_dump()`, придется воспользоваться буферизацией вывода:

```
ob_start();
var_dump($user);
$dump = ob_get_contents();
ob_end_clean();
```

Результаты `var_dump($user)` помещаются в `$dump`.

См. также

Рецепт 8.13 — буферизация вывода; документация по функциям `print_r()`, `var_dump()` и `var_export()`.

6 Функции

6.0. Введение

Функции помогают создавать структурированный код, пригодный для повторного использования. Они позволяют абстрагироваться от второстепенных подробностей, делая код более гибким и понятным. Без использования функций программы создают массу проблем с сопровождением, потому что вам приходится постоянно обновлять идентичные блоки кода в разных местах и разных файлах.

Функция получает набор аргументов и возвращает значение:

```
function add($a, $b) {
    return $a + $b;
}
$total = add(2, 2);
// $total содержит 4
```

Функция объявляется ключевым словом `function`, за которым следует имя функции и параметры, заключенные в круглые скобки. Чтобы вызвать функцию, просто укажите ее имя и значения аргументов. Если функция возвращает значение, присвойте результат функции переменной, как показано в предыдущем примере.

Предварительно объявлять функцию перед вызовом не обязательно. PHP обрабатывает весь файл, прежде чем выполнять его, поэтому объявления и вызовы функций можно чередовать в любой последовательности. При этом в PHP нельзя переопределять функции. Обнаружив функцию с именем, идентичным имени уже существующей функции, PHP выдает фатальную ошибку и аварийно завершается.

Иногда стандартная процедура с передачей фиксированного количества аргументов и возвращением одного значения плохо подходит для конкретной ситуации в вашем коде. Возможно, вы не знаете заранее, сколько параметров потребуется вашей функции. А может быть, функция почти всегда вызывается

с одними и теми же значениями параметров, и вам не хочется загромождать код лишней информацией. А может, из функции нужно вернуть более одного значения.

Эта глава поможет вам в решении подобных проблем в PHP. Мы начнем с обсуждения разных способов передачи аргументов функции. В Рецептах 6.1–6.6 рассматриваются механизмы передачи аргументов по значению, по ссылке и в виде именованных параметров; определение для параметров значений по умолчанию и функции с переменным количеством параметров.

Следующие четыре Рецепта посвящены возвращению значений из функции: Рецепт 6.7 описывает возвращение по ссылке; в Рецепте 6.8 рассматривается возвращение нескольких переменных; Рецепт 6.9 объясняет, как пропускать отдельные возвращаемые значения; и в Рецепте 6.10 рассматривается оптимальный способ возвращения управления и проверки ошибок при вызове функции. Последние три Рецепта показывают, как вызывать разные функции в зависимости от значения переменной, как решать проблемы с областью видимости переменных и как динамически создавать функции. Если вам нужно, чтобы переменная сохраняла свое значение между вызовами функции, обращайтесь к Рецепту 5.5.

6.1. Обращение к параметрам функции

Задача

Требуется получить доступ к значениям, переданным функции.

Решение

Используйте имена из прототипа функции:

```
function commercial_sponsorship($letter, $number) {
    print "This episode of Sesame Street is brought to you by ";
    print "the letter $letter and number $number.\n";
}

commercial_sponsorship('G', 3);

$another_letter = 'X';
$another_number = 15;
commercial_sponsorship($another_letter, $another_number);
```

Комментарий

Для кода функции несущественно, как передаются значения: как строки, числа, массивы или другие виды переменных. С ними можно работать одинаково и обращаться к ним по именам из прототипа.

Если явно не указано обратное, все не-объектные значения передаются функциям и возвращаются ими по значению, а не по ссылке (по умолчанию объекты передаются по ссылке). Это означает, что PHP создает копию переданного значения и предоставляет ее вашему коду для дальнейшей работы. Изменения, вносимые в копию, не отражаются на исходном значении. Например, функция:

```
function add_one($number) {
    $number++;
}
$number = 1;
add_one($number);
print $number;
```

выводит следующий результат:

```
1
```

Если бы переменная передавалась по ссылке, то значение `$number` в глобальной области действия было бы равно 2.

Во многих языках передача переменных по ссылке обладает дополнительным преимуществом: она работает заметно быстрее, чем передача по значению. Хотя в PHP передача по ссылке тоже выполняется быстрее, различия в скорости невелики. По этой причине мы рекомендуем передавать переменные по ссылке только тогда, когда это действительно необходимо, и никогда не применять передачу по ссылке в целях оптимизации.

См. также

Рецепт 6.3 — передача по ссылке; Рецепт 6.7 — возвращение данных по ссылке.

6.2. Определение значений по умолчанию для параметров функции

Задача

Параметр должен иметь значение по умолчанию, если иное значение не указывается при вызове функции. Например, функция для заключения текста в теги HTML может иметь параметр для имени тега; если его значение не задано, по умолчанию используется тег `strong`.

Решение

Укажите значение параметра по умолчанию в прототипе функции:

```
function wrap_in_html_tag($text, $tag = 'strong') {
    return "<$tag>$text</$tag>";
}
```

Комментарий

Пример, приведенный в Решении, назначает для тега значение по умолчанию `strong`. Например, вызов:

```
print wrap_in_html_tag("Hey, a mountain lion!");
```

выводит:

```
<strong>Hey, a mountain lion!</strong>
```

А при явной передаче параметра:

```
print wrap_in_html_tag("Look over there!", "em");
```

результат выглядит так:

```
<em>Look over there!</em>
```

При присваивании значений по умолчанию следует помнить о двух важных обстоятельствах. Во-первых, все параметры со значениями по умолчанию должны перечисляться после параметров, не имеющих значений по умолчанию. В противном случае PHP не сможет определить, какие параметры не были указаны при вызове, в каких случаях следует взять значение по умолчанию, а какие аргументы переопределяют его. Следовательно, функция `wrap_in_html_tag()` не может быть определена так:

```
function wrap_in_html_tag($tag = 'strong', $text)
```

Если вы поступите так и передадите `wrap_in_html_tag()` только один аргумент, то PHP присвоит значение `$tag` и выдаст предупреждение об отсутствии второго аргумента.

Во-вторых, присваиваемое значение по умолчанию должно быть константой, например строкой или числом. Оно не может быть переменной. Возвращаясь к примеру `wrap_in_html_tag()`, следующее объявление недопустимо:

```
$my_favorite_html_tag = 'blink';
function wrap_in_html_tag($text, $tag = $my_favorite_html_tag) {
    return "<$tag>$text</$tag>";
}
```

Если по умолчанию переменная не должна содержать ничего, можно присвоить параметру пустую строку:

```
function wrap_in_html_tag($text, $tag = '') {
    if (empty($tag)) { return $text; }
    return "<$tag>$text</$tag>";
}
```

Если значение `$tag` не передается, функция возвращает исходную строку. Если передается непустое имя тега, то возвращается строка, заключенная между соответствующими тегами.

В зависимости от обстоятельств, для `$tag` также может использоваться значение по умолчанию 0 или `NULL`. Вероятно, в функции `wrap_in_html_tag()` теги с пустым значением нежелательны, но в некоторых случаях этот вариант может быть допустим. Так, в следующем примере при отсутствии аргумента используется сообщение по умолчанию, а при передаче пустой строки — пустое сообщение:

```
function log_db_error($message = NULL) {
    if (is_null($message)) {
        $message = "Couldn't connect to DB";
    }
    error_log("[DB] [$message]");
}
```

См. также

Рецепт 6.6 — создание функций с переменным количеством аргументов.

6.3. Передача по ссылке

Задача

Требуется передать переменную функции так, чтобы в ней сохранились все изменения, внесенные в ее значение внутри функции.

Решение

Чтобы аргумент передавался функции по ссылке, а не по значению, поставьте знак & перед именем параметра в прототипе функции:

```
function wrap_in_html_tag(&$text, $tag = 'strong') {
  $text = “<$tag>$text</$tag>”;
}
```

Теперь возвращать строку нет необходимости, потому что оригинал был модифицирован «на месте».

Комментарий

Передача переменной по ссылке позволяет избежать лишней работы по возвращению переменной и присваиванию возвращаемого значения исходной переменной. Эта возможность также может пригодиться, если функция должна вернуть логический признак успешного выполнения (`true` или `false`), но при этом значения аргументов должны быть изменены внутри функции.

Переключаться между передачей параметра по ссылке и по значению нельзя; вы выбираете либо одно, либо другое. Другими словами, невозможно приказать

PHP интерпретировать переменную в одном случае как передаваемую по ссылке, а в другом – как передаваемую по значению.

Кроме того, если параметр объявлен для передачи по ссылке, в нем нельзя передать константу (строку, число и т. д.); в противном случае происходит аварийное завершение с фатальной ошибкой.

См. также

Рецепт 6.7 — возвращение значений по ссылке.

6.4. Именованные параметры

Задача

Требуется передать аргументы функции по имени (а не по их позиции в списке при вызове функции).

Решение

В PHP нет встроенной поддержки именованных параметров, в отличие от некоторых других языков. Однако эту поддержку можно смоделировать, передавая функции один параметр, который представляет собой ассоциативный массив:

```
function image($img) {
    $tag  = '<img src="' . $img['src'] . '" ';
    $tag .= 'alt="' . (isset($img['alt']) ? $img['alt'] : '') .'"/>';
    return $tag;
}

// $image1 содержит '<img src="cow.png" alt="cows say moo"/>'
$image1 = image(array('src' => 'cow.png', 'alt' => 'cows say moo'));

// $image2 содержит '<img src="pig.jpeg" alt=""/>'
$image2 = image(array('src' => 'pig.jpeg'));
```

Комментарий

Хотя именованные параметры усложняют код внутри функций, они упрощают код вызова этих функций. Так как код функции находится в одном месте, а вызывается во многих, программа становится более понятной.

Так как параметры функции выделены в ассоциативный массив, PHP не сможет предупредить о случайной опечатке в имени параметра. Вам придется действовать более осторожно, потому что парсер не обнаруживает такие ошибки. Кроме того, вы не сможете воспользоваться возможностью определения значения по умол-

чанию для параметра. К счастью, этот недостаток можно обойти, добавив простой код в начало функции:

```
function image($img) {
    if (! isset($img['src']))    { $img['src']    = 'cow.png';      }
    if (! isset($img['alt']))    { $img['alt']    = 'milk factory'; }
    if (! isset($img['height'])) { $img['height'] = 100;            }
    if (! isset($img['width']))  { $img['width']  = 50;             }
    /* ... */
}
```

При помощи функции `isset()` вы убеждаетесь в том, что значение каждого параметра определено; в противном случае параметру присваивается значение по умолчанию.

Также желаемого результата можно добиться при помощи функции `array_merge()`:

```
function image($img) {
    $defaults = array('src'    => 'cow.png',
                      'alt'    => 'milk factory',
                      'height' => 100,
                      'width'  => 50
                     );
    $img = array_merge($defaults, $img);
    /* ... */
}
```

Если один ключ присутствует в обоих массивах, переданных `array_merge()`, то используется значение из второго массива. В приведенном примере это означает, что все значения из `$img` переопределят значения из `$defaults`. Если ключ отсутствует в `$img`, то используется значение из `$defaults`.

См. также

Рецепт 6.6 — создание функций с переменным количеством аргументов.

6.5. Контроль типа аргументов

Задача

Требуется проследить за тем, чтобы значения аргументов относились к определенным типам.

Решение

Используйте указания типов для параметров в определении функции. Указание типа размещается перед именем параметра в объявлении функции:

```
function drink_juice(Liquid $drink) {
    /* ... */
}
function enumerate_some_stuff(array $values) {
    /* ... */
}
```

Комментарий

Указание типа может быть именем класса, именем интерфейса, ключевым словом `array` (начиная с PHP 5.1) или ключевым словом `callable` (начиная с PHP 5.4). Если во время выполнения для параметра с указанием типа передается значение, не соответствующее указанию, PHP выдает ошибку `E_RECOVERABLE_ERROR`.

Если назначить параметру с указанием типа значение по умолчанию `null`, то этот параметр может принимать значение `null` или значение соответствующего типа. В следующем коде функция `must_be_an_array()` требует, чтобы ей передавался параметр-массив. Во всех остальных случаях выдается ошибка `E_RECOVERABLE_ERROR`. Функция `array_or_null_is_ok()` устанавливает менее жесткие требования — если ей передается параметр, то он должен содержать массив или `null`. Если параметр не задан, то локальная переменная `$fruits` будет равна `null`:

```
function must_be_an_array(array $fruits) {
    foreach ($fruits as $fruit) {
        print "$fruit\n";
    }
}

function array_or_null_is_ok(array $fruits = null) {
    if (is_array($fruits)) {
        foreach ($fruits as $fruit) {
            print "$fruit\n";
        }
    }
}
```

См. также

Документация по указаниям типов.

6.6. Создание функций с переменным количеством аргументов

Задача

Требуется определить функцию, которая может получать переменное количество аргументов.

Решение

Передайте функции один аргумент-массив и разместите аргументы в массиве:

```
// Вычисление среднего арифметического для набора чисел
function mean($numbers) {
    // Инициализация для подавления предупреждений
    $sum = 0;

    // Количество элементов в массиве
    $size = count($numbers);

    // Перебор массива и суммирование чисел
    for ($i = 0; $i < $size; $i++) {
        $sum += $numbers[$i];
    }

    // Деление на количество слагаемых
    $average = $sum / $size;

    // Функция возвращает среднее арифметическое
    return $average;
}

// $mean содержит 96.25
$mean = mean(array(96, 93, 98, 98));
```

Комментарий

У задачи есть два хороших решения; выбор зависит от стиля программирования и личных предпочтений. Более традиционный для PHP метод описан в Решении. Мы предпочитаем его, потому что массивы часто используются в PHP, поэтому все программисты хорошо знакомы с массивами и их поведением.

Хотя этот метод сопряжен с некоторыми лишними затратами, упаковка переменных в массив применяется очень часто. В Рецепте 6.4 она используется для создания именованных параметров, а в Рецепте 6.8 — для возвращения из функции нескольких значений. Кроме того, в синтаксисе обращения к данным из массива и работы с ними внутри функции используются такие базовые конструкции, как `$array[$i]` и `count($array)`.

С другой стороны, кому-то такое решение может показаться громоздким. В PHP предусмотрено альтернативное решение, основанное на прямом доступе к списку аргументов, как показано в листинге 6.1.

Листинг 6.1. Обращение к параметрам функций без использования списка аргументов

```
// Вычисление среднего арифметического для набора чисел
function mean() {
    // Инициализация для подавления предупреждений
    $sum = 0;

    // Аргументы, переданные функции
```

```
    $size = func_num_args();

    // Перебор массива и суммирование чисел
    for ($i = 0; $i < $size; $i++) {
        $sum += func_get_arg($i);
    }

    // Деление на количество слагаемых
    $average = $sum / $size;

    // Функция возвращает среднее арифметическое
    return $average;
}

// $mean содержит 96.25
$mean = mean(96, 93, 98, 98);
```

В этом примере используются функции для получения информации об аргументах, переданных функции, из которой они вызываются. Функция `func_num_args()` возвращает количество аргументов, переданных вызывающей функции — в данном случае функции `mean()`. Далее функция `func_get_arg()` используется для получения конкретного значения аргумента для каждой позиции.

Для вызова `mean(96, 93, 98, 98)` функция `func_num_args()` возвращает 4. Первый аргумент находится в позиции 0, поэтому перебор осуществляется от 0 до 3, а не от 1 до 4 (поэтому в цикле `for` переменная `$i` изменяется от 0 до значения, меньшего `$size`). Как видите, логика аналогична той, которая использовалась в Решении с передачей массива. Не беспокойтесь о потенциальной неэффективности вызова `func_get_arg()` в цикле — на самом деле этот способ работает быстрее, чем вариант с передачей массива.

Третья версия этой функции использует `func_get_args()` для получения массива со всеми значениями, переданными функции. Она представляет собой гибрид предыдущих двух версий (листинг 6.2).

Листинг 6.2. Обращение к параметрам функций без использования списка аргументов

```
// Вычисление среднего арифметического для набора чисел
function mean() {
    // Инициализация для подавления предупреждений
    $sum = 0;

    // Аргументы, переданные функции
    $size = func_num_args();

    // Перебор аргументов и суммирование чисел
    foreach (func_get_args() as $arg) {
        $sum += $arg;
    }

    // Деление на количество слагаемых
    $average = $sum / $size;
```

```
    // Функция возвращает среднее арифметическое
    return $average;
}

// $mean содержит 96.25
$mean = mean(96, 93, 98, 98);
```

Такое решение не требует лишнего места для хранения чисел во временном массиве при передаче их `mean()`, но внутри функции код продолжает работать с ними так, словно числа хранятся в массиве.

См. также

Рецепт 6.8 — возвращение нескольких значений из функции; документация по функциям `func_num_args()`, `func_get_arg()` и `func_get_args()`.

6.7. Возвращение значений по ссылке

Задача

Требуется вернуть значение по ссылке, а не по значению. Тем самым предотвращается создание лишнего дубликата переменной.

Решение

Синтаксис возвращения переменной по значению сходен с синтаксисом передачи по ссылке. Однако вместо того, чтобы ставить знак `&` перед именем параметра, поставьте его перед именем функции:

```
function &array_find_value($needle, &$haystack) {
    foreach ($haystack as $key => $value) {
        if ($needle == $value) {
            return $haystack[$key];
        }
    }
}
```

Также при вызове функции необходимо использовать оператор присваивания `=&` вместо простого оператора `=`:

```
$band =& array_find_value('The Doors', $artists);
```

Комментарий

Возвращение ссылки из функции позволяет работать с возвращаемым значением так, чтобы эти изменения напрямую отразились на исходной переменной.

Следующий код ищет в массиве первый элемент с заданным значением и возвращает первое найденное значение. Допустим, вы ищете Принса в списке знаменитостей из Миннесоты, чтобы обновить его имя:

```
function &array_find_value($needle, &$haystack) {
    foreach ($haystack as $key => $value) {
        if ($needle == $value) {
            return $haystack[$key];
        }
    }
}

$minnesota = array('Bob Dylan', 'F. Scott Fitzgerald',
                   'Prince', 'Charles Schultz');

$prince =& array_find_value('Prince', $minnesota);

$prince = 'O(+>'; // ASCII-версия нечитаемого псевдонима Принса

print_r($minnesota);
```

Результат выглядит так:

```
Array
(
    [0] => Bob Dylan
    [1] => F. Scott Fitzgerald
    [2] => O(+>
    [3] => Charles Schultz
)
```

Без возможности возвращения значений по ссылке пришлось бы возвращать ключ массива, а затем заново обращаться к исходному массиву:

```
function array_find_value($needle, &$haystack) {
    foreach ($haystack as $key => $value) {
        if ($needle == $value) {
            return $key;
        }
    }
}

$minnesota = array('Bob Dylan', 'F. Scott Fitzgerald',
                   'Prince', 'Charles Schultz');

$prince = array_find_value('Prince', $minnesota);
// ASCII-версия нечитаемого псевдонима Принса
$minnesota[$prince] = 'O(+>';
```

При возвращении значения по ссылке из функции должна возвращаться ссылка на переменную, а не строка. Например, следующая реализация недопустима:

```
function &array_find_value($needle, &$haystack) {
    foreach ($haystack as $key => $value) {
        if ($needle == $value) {
            $match = $haystack[$key];
```

```
        }
    }

    return "$match is found in position $key";
}
```

Дело в том, что `"$match is found in position $key"` является строкой, а возвращение ссылки на нечто, не являющееся переменной, не имеет смысла. PHP выдает ошибку `E_NOTICE`.

В отличие от передачи значений функциям, при которой аргумент передается либо по ссылке, либо по значению, вы можете отказаться от присваивания ссылки и просто получить возвращенное значение. Используйте оператор `=` вместо `=&`, и PHP присвоит значение вместо ссылки.

См. также

Рецепт 6.3 — передача значений по ссылке.

6.8. Возвращение нескольких значений из функции

Задача

Требуется вернуть из функции более одного значения.

Решение

Верните массив и воспользуйтесь функцией `list()` для разделения элементов:

```
function array_stats($values) {
    $min = min($values);
    $max = max($values);
    $mean = array_sum($values) / count($values);

    return array($min, $max, $mean);
}

$values = array(1,3,5,9,13,1442);
list($min, $max, $mean) = array_stats($values);
```

Комментарий

С точки зрения эффективности это решение неидеально: оно требует лишних затрат на создание массива и его последующее уничтожение. Рассмотрим следующий пример:

```
function time_parts($time) {
    return explode(':', $time);
}

list($hour, $minute, $second) = time_parts('12:34:56');
```

Функция получает строку с временем и вызывает `explode()` для разбиения строки на элементы массива. Когда `time_parts()` вернет управление, функция `list()` берет каждый элемент и сохраняет его в скалярной переменной. Хотя такая схема немного неэффективна, другие возможные решения хуже, потому что они приводят к появлению малопонятного кода.

Одна из альтернатив — передача значений по ссылке. Однако такое решение может показаться неуклюжим и противоестественным, потому что передача необходимых переменных функции не всегда выглядит логично. Пример:

```
function time_parts($time, &$hour, &$minute, &$second) {
    list($hour, $minute, $second) = explode(':', $time);
}
time_parts('12:34:56', $hour, $minute, $second);
```

Не зная прототип функции, вы вряд ли сразу поймете, что `$hour`, `$minute` и `$second` по сути являются возвращаемыми значениями `time_parts()`.

Также можно использовать глобальные переменные, но они загромождают глобальное пространство имен, а вы вряд ли сразу поймете, какие переменные незаметно изменяются функцией. Пример:

```
function time_parts($time) {
    global $hour, $minute, $second;
    list($hour, $minute, $second) = explode(':', $time);
}
time_parts('12:34:56');
```

В данном случае все понятно, потому что определение функции находится прямо над вызовом. Но если бы функция находилась в другом файле или была написана другим разработчиком, решение с глобальными переменными выглядело бы более загадочно и создавало риск неочевидных ошибок.

Если значение изменяется внутри функции, мы рекомендуем вернуть это значение и присвоить его переменной, если только у вас нет веских доводов против этого (например, существенных проблем с производительностью). Такие решения получаются более понятными и создают меньше проблем с сопровождением.

См. также

Рецепт 6.3 — передача значений по ссылке; Рецепт 6.12 — область видимости переменных.

6.9. Пропуск отдельных возвращаемых значений

Задача

Функция возвращает несколько значений, но вас интересуют только некоторые из них.

Решение

Пропустите лишние переменные внутри `list()`:

```
// Нас интересуют только минуты
function time_parts($time) {
    return explode(':', $time);
}
list(, $minute,) = time_parts('12:34:56');
```

Комментарий

Код, приведенный в Решении, на первый взгляд кажется неправильным, но в действительности это корректный код PHP. Чтобы избежать путаницы, не злоупотребляйте этой возможностью; но если функция возвращает много значений, а вам нужны только одно-два из них, она может пригодиться. Один из примеров такого рода: при чтении данных функцией `fgetcsv()`, которая возвращает массив с полями из строки, можно использовать конструкции вроде следующей:

```
while ($fields = fgetcsv($fh, 4096)) {
    print $fields[2] . "\n";  // Третье поле
}
```

При работе с пользовательской, а не встроенной функцией, также можно определить строковые ключи для возвращаемого массива, потому что трудно запомнить, например, что элемент массива 2 ассоциируется с переменной `$rank`:

```
while ($fields = read_fields($filename)) {
    $rank = $fields['rank']; // Третье поле теперь называется rank
    print "$rank\n";
}
```

Самый эффективный метод выглядит так:

```
while (list(,,$rank,,) = fgetcsv($fh, 4096)) {
    print "$rank\n";          // Прямое присваивание $rank
}
```

Будьте внимательны и не перепутайте количество запятых; это приведет к появлению ошибки в программе.

См. также

Рецепт 1.12 — чтение файлов с использованием функции `fgetcsv()`.

6.10. Возвращение признака ошибки

Задача

Требуется вернуть из функции признак ошибки, то есть неудачного выполнения.

Решение

Верните значение `false`:

```
function lookup($name) {
    if (empty($name)) { return false; }
    /* ... */
}

$name = 'alice';

if (false !== lookup($name)) {
    /* Поиск прошел успешно */
} else {
    /* Сохранить информацию об ошибке */
}
```

Комментарий

В PHP значения, не интерпретируемые как `true`, не стандартизированы и могут легко привести к ошибкам. Как следствие, ваши функции должны возвращать предопределенное ключевое слово `false`, потому что оно лучше всего подходит для проверки логических значений.

Другие возможные значения — `''` или 0. Но хотя все три значения интерпретируются в `if` как логическая ложь, между ними существует различие. А иногда возвращаемое значение 0 является содержательным результатом, а вам хотелось бы иметь возможность вернуть признак ошибки.

Например, функция `strpos()` возвращает позицию первой подстроки в строке. Если подстрока не найдена, `strpos()` возвращает `false`. Если подстрока обнаружена, функция возвращает целое число. Таким образом, вы могли бы записать вызов функции поиска подстроки следующим образом:

```
if (strpos($string, $substring)) { /* Подстрока найдена! */ }
```

Но если `$substring` располагается в самом начале `$string`, то возвращается значение 0. К сожалению, в `if` оно интерпретируется как `false`, поэтому условный

код не выполняется. Правильный способ обработки возвращаемого значения `strpos()` выглядит так:

```
if (false !== strpos($string, $substring)) { /* Подстрока найдена! */ }
```

См. также

«Введение» главы 5 — дополнительная информация об истинности переменных; документация по функциям `strpos()` и `empty()`.

6.11. Вызов разных функций в зависимости от значения переменной

Задача

Требуется вызывать разные функции в зависимости от значения переменной.

Решение

Воспользуйтесь функцией `call_user_func()`:

```
function get_file($filename) { return file_get_contents($filename); }

$function = 'get_file';
$filename = 'graphic.png';

// Вызывает get_file('graphic.png')
call_user_func($function, $filename);
```

Если функции вызываются с разным количеством аргументов, используйте `call_user_func_array()`:

```
function get_file($filename) { return file_get_contents($filename); }
function put_file($filename, $d) {
    return file_put_contents($filename, $d); }

if ($action == 'get') {
    $function = 'get_file';
    $args = array('graphic.png');
} elseif ($action == 'put') {
    $function = 'put_file';
    $args = array('graphic.png', $graphic);
}

// Вызывает get_file('graphic.png')
// Вызывает put_file('graphic.png', $graphic)
call_user_func_array($function, $args);
```

Комментарий

Функции `call_user_func()` и `call_user_func_array()` несколько отличаются от стандартных функций PHP. В первом аргументе передается не выводимая строка и не суммируемое число, а имя выполняемой функции. Концепция передачи имени функции для вызова иногда называется *обратным вызовом* (callback). Функция `call_user_func_array()` весьма удобна при обратном вызове из функции, которая может получать переменное количество аргументов. В таких ситуациях вместо того, чтобы встраивать логику внутрь функции, можно получить все аргументы при помощи `func_get_args()`:

```
// Функция для вывода сообщений, получающая форматирование
// в стиле printf. Выводит временную метку, строку и символ новой строки.
function logf() {
    $date = date(DATE_RSS);
    $args = func_get_args();
    return print "$date: " . call_user_func_array('sprintf', $args) . "\n";
}

logf('<a href="%s">%s</a>','http://developer.ebay.com','eBay Developer Program');
```

Функция `logf()` имеет такой же интерфейс, как и семейство `printf`: первый аргумент содержит форматный спецификатор, а во втором содержатся данные, интерполируемые в строку на основании форматных кодов. Так как за кодом форматирования может следовать произвольное количество аргументов, использовать `call_user_func()` не удастся. Вместо этого функция `func_get_args()` получает все аргументы в массиве и передает этот массив `sprintf` с использованием `call_user_func_array()`.

В этом конкретном примере также можно воспользоваться функцией `vsprintf()`; она представляет собой разновидность `sprintf()`, которая, как и `call_user_func_array()`, получает массив аргументов:

```
// Функция для вывода сообщений, получающая форматирование
// в стиле printf. Выводит временную метку, строку и символ новой строки.
function logf() {
    $date = date(DATE_RSS);
    $args = func_get_args();
    $format = array_shift($args);

    return print "$date: " . vsprintf($format, $args) . "\n";
}
```

Если количество вариантов вызываемых функций больше двух, используйте ассоциативный массив с именами функций:

```
$dispatch = array(
    'add'      => 'do_add',
    'commit'   => 'do_commit',
    'checkout' => 'do_checkout',
    'update'   => 'do_update'
);
```

```
$cmd = (isset($_REQUEST['command']) ? $_REQUEST['command'] : '');

if (array_key_exists($cmd, $dispatch)) {
    $function = $dispatch[$cmd];
    call_user_func($function); // Вызов функции
} else {
    error_log("Unknown command $cmd");
}
```

Код берет имя команды из запроса и выполняет эту функцию. Обратите внимание на проверку, которая убеждается в том, что команда присутствует в списке допустимых команд. Тем самым предотвращается вызов из вашего кода произвольной функции, переданной в запросе (например, `phpinfo()`). Это повышает уровень безопасности кода и упрощает выявление ошибок.

Другое преимущество заключается в том, что с одной функцией может быть связано сразу несколько команд, благодаря чему можно определять длинные и короткие имена:

```
$dispatch = array(
    'add'      => 'do_add',
    'commit'   => 'do_commit',   'ci' => 'do_commit',
    'checkout' => 'do_checkout', 'co' => 'do_checkout',
    'update'   => 'do_update',   'up' => 'do_update'
);
```

См. также

Документация по функциям `array_key_exists()`, `call_user_func()`, `call_user_func_array()` и `isset()`.

6.12. Обращение к глобальной переменной внутри функции

Задача

Требуется обратиться к глобальной переменной внутри функции.

Решение

Введите глобальную переменную в локальную область видимости при помощи ключевого слова `global`:

```
function eat_fruit($fruit) {
   global $chew_count;
   for ($i = $chew_count; $i > 0; $i--) {
       /* ... */
   }
}
```

Или обратитесь к ней напрямую через `$GLOBALS`:

```
function eat_fruit($fruit) {
   for ($i = $GLOBALS['chew_count']; $i > 0; $i--) {
       /* ... */
   }
}
```

Комментарий

Если вы используете несколько глобальных переменных внутри функции, ключевое слово `global` упростит понимание синтаксиса функции, особенно если глобальные переменные интерполируются в строках.

Ключевое слово `global` также может использоваться для введения нескольких глобальных переменных в локальную область видимости с перечислением переменных в списке, разделенном запятыми:

```
global $age,$gender,shoe_size;
```

Имена глобальных переменных также могут указываться в синтаксисе косвенных переменных:

```
$which_var = 'age';
global $$which_var; // refers to the global variable $age
```

Однако при вызове `unset()` для переменной, вводимой в локальную область видимости ключевым словом `global`, переменная сбрасывается только внутри этой функции. Чтобы перевести в неопределенное состояние переменную в глобальной области видимости, вызовите `unset()` для элемента массива `$GLOBALS`:

```
$food =  'pizza';
$drink = 'beer';
function party() {
    global $food, $drink;
    unset($food);             // Съесть пиццу
    unset($GLOBALS['drink']); // Выпить пиво
}
print "$food: $drink\n";
party();
print "$food: $drink\n";
```

Результат выглядит так:

```
pizza: beer
pizza:
```

Как видите, переменная `$food` сохранила прежнее значение, тогда как переменная `$drink` была сброшена. Объявление переменной с ключевым словом `global` внутри функции сходно с присваиванием ссылки на глобальную переменную локальной переменной:

```
$food = $GLOBALS['food'];
```

См. также

Документация по области видимости переменных и ссылкам на переменные.

6.13. Создание динамических функций

Задача

Требуется создать и определить функцию во время выполнения программы.

Решение

Воспользуйтесь синтаксисом замыканий для определения функции и сохранения ее в переменной:

```
$increment = 7;
$add = function($i, $j) use ($increment) { return $i + $j + $increment; };

$sum = $add(1, 2);
```

Значение `$sum` теперь равно 10. Если вы работаете с PHP версии до 5.3.0, воспользуйтесь функцией `create_function()`:

```
$increment = 7;

$add = create_function('$i,$j', 'return $i+$j + ' . $increment. ';');
$sum = $add(1, 2);
```

Комментарий

Синтаксис *замыканий* (closure) намного приятнее использования `create_function()`. У `create_function` список аргументов и тело функции записываются в виде литеральных строк. Это означает, что PHP не может разобрать их синтаксис до стадии выполнения, и вам приходится учитывать возможное присутствие одиночных и двойных кавычек, а также правила интерполяции переменных.

В синтаксисе замыканий PHP сможет проверить анонимные функции на стадии компиляции точно так же, как это делается с остальным кодом. При этом используется тот же синтаксис, что и для написания функций, за одним исключением: в объявлении `use()` после списка аргументов могут перечисляться переменные из области видимости, содержащей определение замыкания, которые должны быть доступны внутри замыкания. В приведенном примере конструкция `use($increment)` означает, что внутри замыкания `$increment` имеет то же значение, что и в области видимости, в которой определяется замыкание (7).

Анонимные функции часто применяются для создания пользовательских функций сортировки для `usort()` или `array_walk()`:

```
$files = array('ziggy.txt', '10steps.doc', '11pants.org', "frank.mov");
// Файлы в порядке, обратном порядку естественной сортировки
usort($files, function($a, $b) { return strnatcmp($b, $a); });
// Теперь $files содержит
// array('ziggy.txt', 'frank.mov','11pants.org','10steps.doc')
```

См. также

Рецепт 4.17 — информация о функции `usort()`; документация по функциям `create_function()` и `usort()`.

7 Классы и объекты

7.0. Введение

Ранние версии PHP были чисто процедурными: разработчик мог определять функции, но не объекты. В PHP 3 появилась крайне рудиментарная поддержка объектов, написанная «на скорую руку». В 1997 году никто не ожидал стремительного роста численности программистов PHP или того, что на PHP будут создаваться масштабные программные проекты. В то время ограниченные возможности объектов не создавали проблем.

За прошедшие годы в PHP была реализована расширенная объектно-ориентированная функциональность; тем не менее группа разработки не переработала базовый объектно-ориентированный код для нормальной работы с объектами и классами. В результате, несмотря на общее повышение эффективности PHP 4, написать сложную объектно-ориентированную программу было достаточно сложно, если вообще возможно.

В PHP 5 эти проблемы были решены за счет использования ядра Zend Engine 2 (ZE2). Оно позволило включить в PHP более современные объектно-ориентированные возможности без потери высокой степени обратной совместимости с миллионами написанных сценариев PHP. В последних версиях PHP 5 объектно-ориентированный инструментарий был дополнительно усовершенствован. Сегодня он позволяет разработчикам создавать полнофункциональные объектно-ориентированные приложения.

Читателей, не имеющих опыта работы с объектно-ориентированным программированием, ждет немало сюрпризов. Хотя решение некоторых задач упрощается, в ряде случаев ваши возможности *ограничиваются*.

На первый взгляд ограничения выглядят противоестественно, но в действительности они помогают быстро писать безопасный код, потому что их целью является повторное использование кода и инкапсуляция данных. Эти ключевые инструменты объектно-ориентированного программирования будут рассмотрены

позднее в этой главе. А начнем мы с краткого введения в объектно-ориентированное программирование, его терминологию и концепции.

Класс представляет собой пакет, состоящий из данных и методов для работы с этими данными. Данные состоят из переменных, называемых *свойствами*. Вторая «половина» класса представляет собой набор функций, работающих со свойствами, — такие функции называются *методами*.

Определяя класс, вы не определяете объект, с которым будет работать ваша программа. Вместо этого вы определяете своего рода «шаблон» для создания объектов. На базе этого шаблона создаются конкретные объекты, также называемые *экземплярами*. Программа может работать с многими объектами одного класса подобно тому, как у человека может быть много книг.

Классы образуют четко определенную иерархию. Каждый класс, находящийся на нижних уровнях иерархии, обладает более четкой специализацией, чем расположенные выше классы. Специализированные классы называются *производными* классами, а тот класс, который они «уточняют», называется *родительским* классом. Например, родительский класс может представлять здание. Здания дополнительно подразделяются на жилые и коммерческие. Жилые здания делятся на малоэтажные и многоэтажные. И так далее. Родительский класс самого верхнего уровня также называется *базовым* классом.

И малоэтажные, и многоэтажные здания обладают тем же набором свойств, что и все жилые здания, — подобно тому, как у жилых и коммерческих зданий тоже имеются общие аспекты. Когда классы используются для выражения отношений «родитель—потомок», производный класс наследует свойства и методы, определенные в родительском классе. Таким образом, разработчик может заново использовать код родительского класса, и ему остается лишь написать код, адаптирующий новый производный класс к конкретной ситуации. Этот механизм, называемый *наследованием*, составляет одно из важнейших преимуществ классов перед функциями. Процесс определения производного класса на основе родительского класса называется *субклассированием*, или *расширением*.

Классы PHP легко определяются и создаются:

```
class guest_book {
 public $comments;
 public $last_visitor;
 function update($comment, $visitor) {
 ...
 }
}
```

Ключевое слово `class` определяет класс подобно тому, как ключевое слово `function` определяет функцию. Свойства объявляются с ключевым словом `public`. Объявление метода идентично определению функции.

Для создания объекта (или экземпляра) используется ключевое слово `new`:

```
$gb = new guest_book;
```

Создание объектов более подробно рассматривается в Рецепте 7.1.

Класс может содержать объявления свойств с ключевым словом `public`. Включать такое объявление необязательно, но оно содержит полезную информацию обо всех переменных класса. Так как PHP не заставляет разработчика заранее объявлять все переменные, при создании такой переменной внутри класса PHP не выдаст сообщение об ошибке и вообще никак не уведомит пользователя. Кроме того, список переменных в начале определения класса будет дезинформировать пользователя, потому что он не соответствует списку переменных, фактически используемых в классе.

При объявлении свойству можно присвоить значение:

```
public $last_visitor = 'Donnan';
```

В правой стороне этой конструкции может находиться только константа:

```
public $last_visitor = 'Donnan'; // Можно
public $last_visitor = 9; // Можно
public $last_visitor = array('Jesse'); // Можно
public $last_visitor = pick_visitor(); // Нельзя
public $last_visitor = 'Chris' . '9'; // Нельзя
```

При попытке присвоить любое другое значение происходит аварийное завершение PHP.

Чтобы присвоить переменной неконстантное значение, выполните присваивание в методе класса:

```
class guest_book {
  public $last_visitor;
  public function update($comment, $visitor) {
    if (!empty($comment)) {
      array_unshift($this->comments, $comment);
      $this->last_visitor = $visitor;
    }
  }
}
```

Если посетитель оставил комментарий, он добавляется в начало массива комментариев, а этот человек становится последним посетителем гостевой книги. `$this` — специальная переменная, содержащая ссылку на текущий объект. Итак, для обращения к свойству `$last_visitor` объекта из метода объекта используется конструкция вида `$this->last_visitor`.

Если неконстантное значение должно присваиваться переменной при создании объекта, присвойте его в конструкторе класса. Конструктор класса представляет собой метод, автоматически вызываемый при создании нового объекта; ему присваивается фиксированное имя `__construct()`:

```
class guest_book {
  public $comments;
  public $last_visitor;
```

```
  public function __construct($user) {
    $dbh = mysqli_connect('localhost', 'username', 'password', 'sites');
    $user = mysqli_real_escape_string($dbh, $user);
    $sql = "SELECT comments, last_visitor FROM guest_books WHERE user='$user'";
    $r = mysqli_query($dbh, $sql);

    if ($obj = mysqli_fetch_object($dbh, $r)) {
      $this->comments = $obj->comments;
      $this->last_visitor = $obj->last_visitor;
    }
  }
}

$gb = new guest_book('stewart');
```

Конструкторы рассматриваются в Рецепте 7.2.

Будьте внимательны и не введите по ошибке `$this->$size`. Эта запись допустима, но по смыслу она отличается от `$this->size`. Она обращается к свойству объекта, имя которого определяется значением, хранящимся в переменной `$size`. Обычно значение `$size` не определено, поэтому значение `$this->$size` оказывается пустым. Об именах свойств рассказано в Рецепте 5.4.

В PHP 5.4 возможен вызов метода или обращение к свойству непосредственно при создании объекта:

```
$last_visitor = (new guest_book('stewart'))->last_visitor;
$last_visitor = (new guest_book('stewart'))->getLastVisitor();
```

Помимо `->` для обращения к методам или переменным классов может использоваться запись `::`. Этот синтаксис предназначен для обращения к статическим методам класса. Статические методы работают одинаково для каждого экземпляра класса, потому что они не зависят от данных конкретных экземпляров. В статическом методе не может использоваться ссылка `$this`. Пример:

```
class convert {
 // Преобразование из шкалы Цельсия
 // в шкалу Фаренгейта
 public static function c2f($degrees) {
   return (1.8 * $degrees) + 32;
 }
}
$f = convert::c2f(100); // 212
```

Для реализации наследования посредством расширения существующего класса используется ключевое слово `extends`:

```
class xhtml extends xml {
  // ...
}
```

Производные классы наследуют родительские методы и при желании могут реализовать собственные, специализированные версии. Пример:

```
class DB {
 public $result;

 function getResult() {
  return $this->result;
 }

 function query($sql) {
  error_log("query() must be overridden by a database-specific child");
  return false;
 }
}

class MySQL extends DB {
 function query($sql) {
  $this->result = mysql_query($sql);
 }
}
```

Класс `MySQL` наследует неизменную реализацию `getResult()` от родительского класса `DB`, но определяет собственную, предназначенную для `MySQL` реализацию метода `query()`. Чтобы явно вызвать метод родительского класса, поставьте перед именем метода префикс `parent::`:

```
function escape($sql) {
 $safe_sql = mysql_real_escape_string($sql); // Экранирование специальных
символов
 $safe_sql = parent::escape($safe_sql); // Родительский метод заключает $sql в ''
 return $safe_sql;
}
```

Вызов переопределенных методов рассматривается в Рецепте 7.14.

7.1. Создание объектов

Задача

Требуется создать новый объект (экземпляр класса).

Решение

Определите класс, а затем воспользуйтесь ключевым словом `new` для создания объекта этого класса:

```
class user {
    function load_info($username) {
       // Загрузка профиля из базы данных
    }
}

$user = new user;
$user->load_info($_GET['username']);
```

Комментарий

Вы можете создать несколько экземпляров с данными одного объекта:

```
$adam = new user;
$adam->load_info('adam');
$dave = new user;
$dave->load_info('adam');
```

В программе существуют два независимых объекта, которые содержат одинаковую информацию. Их можно сравнить с близнецами: при «рождении» их не отличить друг от друга, но дальше каждый из них живет своей жизнью.

См. также

Рецепт 7.10 — дополнительная информация о копировании и клонировании объектов; документация по классам и объектам.

7.2. Определение конструкторов

Задача

Требуется определить метод, вызываемый при создании объекта. Например, при создании объект должен автоматически заполняться информацией, загружаемой из базы данных.

Решение

Определите метод с именем `__construct()`:

```
class user {
    function __construct($username, $password) {
        // ...
    }
}
```

Комментарий

Метод с именем `__construct()` (с двумя символами подчеркивания перед словом `construct`) используется для построения объекта:

```
class user {
  public $username;

  function __construct($username, $password) {
     if ($this->validate_user($username, $password)) {
       $this->username = $username;
     }
  }
```

```
}
$user = new user('Grif', 'Mistoffelees'); // Использование
                                          // встроенного конструктора
```

Для обратной совместимости с PHP 4, если PHP 5 не находит метод с именем `__construct()`, но находит метод с именем, совпадающим с именем класса (правило назначения имени конструктора PHP 4), то этот метод используется как конструктор класса.

Наличие стандартного имени у всех конструкторов упрощает вызов конструктора родительского класса (потому что вам не нужно знать имя родительского класса), а также не требует модификации конструктора в случае переименования класса.

См. также

Рецепт 7.14 — дополнительная информация о вызове конструкторов родительских классов; документация по конструкторам.

7.3. Определение деструкторов объектов

Задача

Требуется определить метод, автоматически вызываемый при уничтожении объекта. Например, уничтожение объекта должно сопровождаться автоматическим сохранением информации в базе данных.

Решение

Объекты автоматически уничтожаются при завершении сценария. Принудительное уничтожение объектов осуществляется вызовом `unset()`:

```
$car = new car; // Покупка новой машины
// ...
unset($car);    // Машина отправляется на свалку
```

Чтобы заставить PHP автоматически вызывать метод при уничтожении объекта, определите в классе метод с именем `__destruct()`:

```
class car {
    function __destruct() {
        // ...
    }
}
```

Комментарий

Обычно объекты не обязательно уничтожать «вручную», но в большом цикле вызов `unset()` может предотвратить лавинообразное нарастание затрат памяти.

В PHP поддерживаются деструкторы объектов. Деструктор похож на конструктор, но вызывается он при уничтожении объекта. Даже если вы не уничтожите объект вызовом `unset()`, PHP все равно вызовет деструктор, когда определит, что объект больше не используется. Это может произойти при завершении сценария, но может произойти и намного ранее.

Деструктор обычно используется для освобождения ресурсов, удерживаемых объектом. Например, деструктор `Database` может разорвать связь с базой данных и освободить ресурсы подключения. В отличие от конструктора, деструктору нельзя передать информацию, потому что вы не знаете, когда он будет выполнен.

Следовательно, если вашему деструктору необходима информация, связанная с конкретным экземпляром, сохраните ее в свойстве:

```
// Деструктор
class Database {
    function __destruct() {
         db_close($this->handle); // Закрытие подключения к базе данных
    }
}
```

Деструкторы выполняются до того, как PHP прекратит обработку запроса и завершит отправку данных. Следовательно, в них можно выполнять операции вывода, записывать информацию в базу данных и даже проверять связь с удаленным сервером.

Однако вы не можете предполагать, что PHP будет уничтожать объекты в каком-то определенном порядке. Следовательно, в деструкторе нельзя ссылаться на другие объекты, потому что он уже может быть уничтожен PHP. Это не приведет к сбою программы, но ваш код будет вести себя непредсказуемо (и скорее всего, некорректно).

См. также

Документация по функции `unset()`.

7.4. Управление доступом

Задача

Требуется ограничить доступ к некоторым методам и свойствам, чтобы они были доступны только в классах, связанных определенными отношениями с классом объекта.

Решение

Используйте ключевые слова `public`, `protected` и `private`:

```
class Person {
    public $name;     // Переменная доступна везде
    protected $age;   // Доступна в классе и в производных классах
    private $salary;  // Доступна только в этом классе

    public function __construct() {
    // ...
    }

    protected function set_age() {
    // ...
    }

    private function set_salary() {
    // ...
    }
}
```

Комментарий

PHP позволяет ограничить доступ к методам и свойствам. Существуют три уровня видимости:

- открытый (`public`);
- защищенный (`protected`);
- закрытый (`private`).

Объявление метода или свойства с ключевым словом `public` означает, что он может вызываться или изменяться кем угодно[1].

Также метод или свойство можно пометить ключевым словом `protected`; в этом случае доступ к нему будет возможен только из текущего класса и производных классов, расширяющих этот класс.

Последний уровень видимости — `private` — устанавливает самые жесткие ограничения. Такие свойства и методы доступны только внутри этого класса.

Если вы не сталкивались с этой концепцией, управление доступом выглядит немного странно. Однако управление доступом делает код более надежным, потому что оно способствует *инкапсуляции данных* — одному из основополагающих принципов объектно-ориентированного программирования.

При написании кода неизбежно выясняется, что какая-то часть — способ хранения данных, передаваемые функции параметры, структура базы данных — работает не так хорошо, как хотелось бы: она работает слишком медленно, недостаточно эффективно, не позволяет добавить новую функциональность и т. д. Короче говоря, она нуждается в улучшении.

---

[1] До выхода PHP 5 все методы и свойства объектов были открытыми. — *Примеч. ред.*

Доработка кода — хорошее дело, если только она не приводит к случайному нарушению работы других частей системы. В программах, спроектированных с высокой степенью инкапсуляции, нижележащие структуры данных и таблицы баз данных не используются напрямую. Вместо этого определяется набор функций, и все обращения к данным проходят через эти функции.

Предположим, имеется таблица базы данных, в которой хранится информация об именах и адресах электронной почты. Программа с плохой инкапсуляцией данных напрямую обращается к таблице, когда ей потребуется получить адрес электронной почты:

```
$name   = 'Rasmus Lerdorf';

$sqlite = new PDO('sqlite:/usr/local/users.db');

$rows = $db->query("SELECT email FROM users WHERE name LIKE '$name'");
$row = $rows->fetch();
$email  = $row['email'];
```

В программе с хорошей инкапсуляцией будет использоваться промежуточная функция:

```
function getEmail($name) {
    $sqlite = new PDO("sqlite:/usr/local/users.db");

    $rows = $db->query("SELECT email FROM users WHERE name LIKE '$name'");
    $row = $rows->fetch();
    $email  = $row['email'];
    return $email;
}

$email = getEmail('Rasmus Lerdorf');
```

Использование `getEmail()` обладает многими преимуществами, среди которых и сокращение объема кода, который необходимо написать для получения адреса электронной почты. Но кроме этого оно позволяет безопасно изменять схему базы данных: ведь вам достаточно изменить всего один запрос в `getEmail()` вместо того, чтобы просматривать каждую строку каждого файла в поисках мест, в которых осуществляется выборка данных из таблицы. Также появляется возможность с относительной простотой переключаться с одной базы данных на другую. Трудно написать хорошо инкапсулированную программу без управления доступом, потому что в такой программе сообщить «Не трогать!» можно только при помощи комментариев и условных соглашений.

Объекты ограждают подробности внутренней реализации от постороннего доступа. В результате пользователи не будут зависеть от кода, который может измениться в будущем, и им приходится использовать ваши функции для обращения к данным. Такие функции называются *методами доступа*, потому что они открывают доступ к защищенной информации. При изменении структуры кода достаточно обновить методы доступа, и весь остальной код сохранит работоспособность.

Пометка составляющих класса ключевым словом `protected` или `private` указывает на то, что они могут измениться в будущем, поэтому пользователи не должны работать с ними, чтобы не нарушить инкапсуляции.

Ограничение доступа — нечто большее, чем общепринятая условность. PHP действительно запрещает вызывать закрытые методы или читать закрытые свойства за пределами класса. Следовательно, с точки зрения внешнего пользователя такие методы и свойства вообще не существуют, потому что обратиться к ним невозможно.

В объектно-ориентированном программировании существует неявный контракт между автором и пользователем класса. Пользователи обещают забыть о подробностях реализации, а автор обещает, что открытые методы всегда будут работать, даже если он внесет изменения в реализацию класса.

Ключевые слова `protected` и `private` обеспечивают защиту от использования за пределами класса. Следовательно, решение о выборе того или иного уровня защиты в действительности определяется здравым смыслом — ожидаете ли вы, что кому-то понадобится вызывать этот метод в производном классе?

Если класс будет использоваться только вами (или вашей группой), выбор уровня `private` вместо `protected` позволит обеспечить более высокий уровень защиты и застраховаться от любых случайностей. Если потребуется, степень защиты можно будет снизить в будущем. Если же вы планируете распространять код в виде пакета, то использование уровня `protected` позволит другим расширить результаты вашей работы без модификации основной библиотеки.

7.5. Запрет на изменение классов и методов

Задача

Требуется запретить другим разработчикам переопределять конкретные методы в производных классах или даже полностью запретить расширение всего класса.

Решение

Пометьте нужные методы или весь класс ключевым словом `final`:

```
final public function connect($server, $username, $password) {
    // Определение метода
}
```

и:

```
final class MySQL {
    // Определение класса
}
```

Комментарий

Вообще говоря, наследование — полезная штука, и все же иногда его стоит ограничивать.

Главная причина для объявления метода с ключевым словом `final` — риск, который может быть связан с его переопределением: повреждение данных, условие гонки, возможность сбоев или взаимной блокировки из-за того, что пользователь забыл установить (или, наоборот, освободить) блокировку или семафор.

Чтобы запретить переопределение метода, включите ключевое слово `final` в начало объявления метода:

```
final public function connect($server, $username, $password) {
    // Определение метода
}
```

Тем самым предотвращается создание другой версии метода `connect()` при субклассировании.

Чтобы запретить субклассирование для всего класса, не нужно помечать каждый метод ключевым словом `final`. Вместо этого включите его в заголовок класса:

```
final class MySQL {
    // Определение класса
}
```

Класс с ключевым словом `final` не может субклассироваться, то есть использоваться в качестве родительского класса. В этом отношении он отличается от класса, в котором все методы имеют пометку `final`, — последний может расширяться и дополняться новыми методами, хотя никакие существующие методы изменяться при этом не могут.

7.6. Определение строкового представления объекта

Задача

Требуется управлять представлением объекта при его выводе из PHP.

Решение

Реализация метода `__toString()`:

```
class Person {
    // Остальной код класса
    public function __toString() {
        return "$this->name <$this->email>";
    }
}
```

Комментарий

PHP предоставляет объектам возможность управлять процессом их преобразования в строковую форму. Это позволяет вывести представление объекта удобным способом, без большого объема дополнительного кода.

При выполнении команды `echo` или `print` для отдельного объекта PHP вызывает метод `__toString()` этого объекта.

Пример:

```
class Person {
    protected $name;
    protected $email;

    public function setName($name) {
        $this->name = $name;
    }

    public function setEmail($email) {
        $this->email = $email;
    }

    public function __toString() {
        return "$this->name <$this->email>";
    }
}
```

После этого представление объекта выводится следующим образом:

```
$rasmus = new Person;
$rasmus->setName('Rasmus Lerdorf');
$rasmus->setEmail('rasmus@php.net');
print $rasmus;

Rasmus Lerdorf <rasmus@php.net>
```

При вызове `print` PHP автоматически вызывает метод `__toString()` и возвращает строковое представление объекта.

Метод *должен* возвращать строку; в противном случае PHP выдаст сообщение об ошибке. Хотя это кажется очевидным, иногда разработчиков подводит механизм автоматического преобразования типа, который здесь не работает.

Например, с точки зрения разработчика, строка `'9'` часто считается эквивалентом целого числа 9, потому что PHP обычно переключается между ними в зависимости от контекста и почти всегда с правильным результатом.

Однако функция `__toString()` не может возвращать целые числа. Если вы подозреваете, что метод может вернуть нестроковое значение, рассмотрите возможность явного преобразования типа результата:

```
class TextInput {
    // Остальной код класса
```

```
    public function __toString() {
        return (string) $this->label;
    }
}
```

Преобразуя `$this->label` в `string`, вы можете не беспокоиться о том, что кто-то может снабдить текстовое поле числовой меткой.

До выхода PHP 5.2 функциональность `__toString()` обладала рядом ограничений. Следовательно, если вы часто используете `__toString()` в своем коде, лучше работать с PHP 5.2 и выше.

7.7. Результат сходного поведения в разных классах

Задача

Некоторые методы должны использоваться разными классами, но при этом наследование всех классов от одного родительского класса выглядит нелогично.

Решение

Определите интерфейс и объявите, что ваши классы реализуют этот интерфейс:

```
interface NameInterface {
    public function getName();
    public function setName($name);
}

class Book implements NameInterface {
    private $name;

    public function getName() {
        return $this->name;
    }

    public function setName($name) {
        return $this->name = $name;
    }
}
```

Интерфейс `NameInterface` определяет два метода для присваивания имени объекту. Класс `Book` объявляет о реализации интерфейса `NameInterface` и определяет два метода в теле класса.

Если вы захотите включить код, реализующий интерфейс, в другое место, определите *типаж* (trait) и объявите об его использовании другими классами:

```
trait NameTrait {
    private $name;
```

```
    public function getName() {
        return $this->name;
    }

    public function setName($name) {
        return $this->name = $name;
    }
}

class Book {
    use NameTrait;
}

class Child {
    use NameTrait;
}
```

Типаж `NameTrait` определяет и реализует два метода, необходимых для присваивания имени объекту. Так, класс `Book` объявляет об использовании типажа `NameTrait`, после чего вы можете вызвать два метода из тела класса.

Комментарий

В объектно-ориентированном программировании объекты должны совместно решать общие задачи. Следовательно, у разработчика должна быть возможность потребовать, чтобы класс (и возможно, не один) реализовал методы, необходимые для его правильного взаимодействия с системой.

Например, приложение электронного магазина должно обладать некоторым объемом информации о каждом продаваемом товаре. Эти товары могут относиться к разным классам: книги, CD, DVD и т. д. Тем не менее каждый товар в каталоге независимо от его типа должен как минимум обладать именем (и если уж на то пошло, ему также должна быть назначена цена и идентификатор).

Механизм, заставляющий классы поддерживать определенный набор методов, называется *интерфейсом*. Определение интерфейса сходно с определением класса:

```
interface NameInterface {
    public function getName();
    public function setName($name);
}
```

Вместо ключевого слова `class` интерфейс использует ключевое слово `interface`. Внутри интерфейса определяются прототипы методов, но не предоставляются их реализации.

Приведенный фрагмент создает интерфейс с именем `NameInterface`. Любой класс, реализующий `NameInterface`, должен реализовать два метода, указанных в интерфейсе: `getName()` и `setName()`.

Если класс поддерживает все методы интерфейса, он *реализует* этот интерфейс. Обязательство по реализации интерфейса указывается в определении класса:

```
class Book implements NameInterface {
    private $name;

    public function getName() {
        return $this->name;
    }

    public function setName($name) {
        return $this->name = $name;
    }
}
```

Если класс реализует не все методы, входящие в интерфейс, или реализует их с другими прототипами, PHP выдает фатальную ошибку.

Класс может реализовать столько интерфейсов, сколько потребуется. Например, интерфейс `ListenInterface` может определять способ загрузки аудиоролика для товара. В этом случае классы `CD` и `DVD` будут реализовать `ListenInterface`, а класс `Book` не будет (если, конечно, он не представляет аудиокнигу).

При использовании интерфейсов очень важно, чтобы классы объявлялись до создания объектов; в противном случае PHP иногда начинает путаться с реализацией интерфейсов. Чтобы не нарушить работу существующих приложений, это не является обязательным требованием, но лучше не нарушать его.

Если вы хотите проверить, реализует ли класс конкретный интерфейс, воспользуйтесь функцией `class_implements()`:

```
class Book implements NameInterface {
    // ...
}

$interfaces = class_implements('Book');
if (isset($interfaces['NameInterface'])) {
    // Book реализует NameInterface
}
```

Также можно воспользоваться классами `Reflection`:

```
class Book implements NameInterface {
    // ...
}

$rc = new ReflectionClass('Book');
if ($rc->implementsInterface('NameInterface')) {
  print "Book implements NameInterface\n";
}
```

Чтобы организовать совместное использование кода между двумя классами, используйте типажи:

```
trait NameTrait {
    private $name;
    public function getName() {
        return $this->name;
    }
    public function setName($name) {
```

```
        return $this->name = $name;
    }
}
class Book {
    use NameTrait;
}
$book = new Book;
$book->setName('PHP Cookbook');
print $book->getName();
```

Интерфейсы могут работать вместе с типажами. Более того, так даже рекомендуется поступать:

```
class Book implements NameInterface {
    use NameTrait;
}
```

Интерфейсы устанавливают четкие контракты с явно выраженными обязательствами относительно поведения объектов. Типажи позволяют организовать совместное использование кода между объектами, не имеющими отношения наследования типа «является частным случаем»; это всего лишь программный механизм, позволяющий избежать копирования и вставки кода в нескольких местах.

Сочетание интерфейсов с типажами извлекает наибольшую выгоду из обеих технологий. Интерфейс предоставляет контракт для набора классов, а типаж позволяет реализовать его. Далее конкретный класс может выбрать между использованием типажа или полноценной реализацией интерфейса. Например, вы можете установить ограничение, согласно которому каждый объект `Book` должен обладать уникальным именем, или это имя должно храниться в базе данных. В таких случаях `NameTrait` для ваших потребностей будет недостаточно.

Класс может реализовать несколько интерфейсов или типажей, которые перечисляются через запятую:

```
class Book implements NameInterface, SizeInterface {
    use NameTrait, SizeTrait;
}
```

См. также

Рецепт 7.20 — дополнительная информация о классах `Reflection`; документация по функции `class_implements()`, интерфейсам и типажам.

7.8. Создание абстрактных базовых классов

Задача

Требуется создать *абстрактный* класс — иначе говоря, класс, который используется только как общая база для производных классов (без возможности создания объектов этого класса).

Решение

Пометьте класс ключевым словом `abstract`:

```
abstract class Database {
  // ...
}
```

Ключевое слово `abstract` размещается перед определением `class`.

В классе также должен быть определен как минимум один абстрактный метод. Для этого ключевое слово `abstract` размещается в начале определения метода:

```
abstract class Database {
    abstract public function connect();
    abstract public function query();
    abstract public function fetch();
    abstract public function close();
}
```

Комментарий

Абстрактные классы лучше всего использовать для групп объектов, связанных отношением типа «является частным случаем». По этой причине логично заставить такие объекты наследовать от общего родителя — но при этом потомки «материальны», а родитель абстрактен.

Для примера возьмем класс `Database`. База данных — реальный объект, поэтому создание класса `Database` выглядит логично. С другой стороны, существуют Oracle, MySQL, Postgres, MSSQL и сотни других баз данных, но загрузить и установить «базу данных вообще» невозможно. Вы должны выбрать конкретную базу данных.

PHP позволяет определять классы, объекты которых не могут создаваться. Такие классы называются абстрактными. Для примера возьмем класс `Database`:

```
abstract class Database {
    abstract public function connect($server, $username, $password, $database);
    abstract public function query($sql);
    abstract public function fetch();
    abstract public function close();
}
```

Чтобы пометить класс как абстрактный, поместите ключевое слово `abstract` перед `class`.

Абстрактные классы должны содержать как минимум один метод с пометкой `abstract`. Эти методы называются *абстрактными методами*. Класс `Database` содержит четыре абстрактных метода: `connect()`, `query()`, `fetch()` и `close()`. Эти четыре метода образуют базовую функциональность, необходимую для работы с базой данных.

Если класс содержит абстрактный метод, сам класс тоже должен быть объявлен абстрактным. Тем не менее абстрактные классы могут содержать неабстрактные методы (хотя в классе `Database` таких нет).

Абстрактные методы, как и методы интерфейсов, не реализуются внутри абстрактного класса. Вместо этого абстрактные методы реализуются в производном классе, расширяющем абстрактного родителя. Например, вы можете использовать класс `MySQL`:

```
class MySQL extends Database {
    protected $dbh;
    protected $query;

    public function connect($server, $username, $password, $database) {
        $this->dbh = mysqli_connect($server, $username,
                                    $password, $database);
    }

    public function query($sql) {
        $this->query = mysqli_query($this->dbh, $sql);
    }

    public function fetch() {
        return mysqli_fetch_row($this->dbh, $this->query);
    }

    public function close() {
        mysqli_close($this->dbh);
    }
}
```

При реализации абстрактных методов должны соблюдаться прототипы методов. Так, в приведенном примере `query()` получает единственный аргумент `$sql`.

Если производный класс не реализует все абстрактные методы родительского класса, он тоже является абстрактным, поэтому другой класс должен осуществить дальнейшее субклассирование. Например, такая иерархия может появиться, если вы хотите создать два класса `MySQL`: первый осуществляет выборку информации в виде объектов, а другой возвращает массивы.

К абстрактным методам предъявляются два требования:

- Абстрактные методы не могут определяться как закрытые (`private`), потому что они должны использоваться при наследовании.
- Абстрактные методы не могут определяться с ключевым словом `final`, потому что они должны переопределяться.

Абстрактные классы и интерфейсы похожи, и все же это разные концепции. Прежде всего, класс может реализовать несколько интерфейсов, но расширять он может только один абстрактный класс. Кроме того, в интерфейсе можно определять только прототипы методов — никаких реализаций. Напротив, в абстрактном классе должен присутствовать только один абстрактный метод, наряду с которым могут быть другие неабстрактные методы и даже свойства.

Абстрактные классы также следует использовать в ситуациях, в которых действует правило «является частным случаем». Например, можно сказать: «MySQL является частным случаем базы данных», поэтому класс `Database` разумно сделать

абстрактным. С другой стороны, вы не можете сказать: «Книга является частным случаем ее названия», поэтому `NameInterface` следует оформить в виде интерфейса.

7.9. Присваивание ссылок на объекты

Задача

Требуется связать два объекта так, чтобы при обновлении одного объекта также обновлялся другой.

Решение

Воспользуйтесь оператором `=` так, чтобы присвоить один объект другому по ссылке:

```
$adam = new user;
$dave = $adam;
```

Комментарий

При присваивании объекта оператором `=` вместо новой копии объекта создается ссылка на первый объект. Таким образом, модификация одного объекта автоматически приводит к изменению другого.

Такое поведение отличается от того, как PHP поступает с другими видами переменных, для которых выполняется копирование по значению:

```
$adam = new user;
$adam->load_info('adam');
$dave = $adam;
```

После выполнения присваивания `$dave` и `$adam` представляют собой два имени одного объекта.

См. также

Рецепт 7.10 — клонирование объектов; документация по ссылкам.

7.10. Клонирование объектов

Задача

Требуется создать копию объекта.

Решение

Копирование объектов по ссылке осуществляется оператором =:

```
$rasmus = $zeev;
```

Копирование объектов по значению осуществляется ключевым словом clone:

```
$rasmus = clone $zeev;
```

Комментарий

PHP обычно копирует объекты по ссылке, а не по значению. Когда вы присваиваете существующий объект новой переменной, новая переменная становится всего лишь дополнительным именем для существующего объекта. Обращение к объекту по новому или старому имени приводит к одинаковым результатам.

Чтобы создать независимый экземпляр значения с тем же содержимым (этот процесс называется копированием по значению), воспользуйтесь ключевым словом clone. В противном случае второй объект будет содержать обычную ссылку на первый.

Процесс клонирования копирует каждое свойство из первого объекта во второй. В копировании участвуют свойства, содержащие объекты, так что клонированный объект в результате может содержать те же ссылки на объекты, что и оригинал.

Часто такое поведение оказывается нежелательным. Для примера возьмем композитную версию класса Person, которая содержит объект Address:

```
class Address {
  protected $city;
  protected $country;
  public function setCity($city) { $this->city = $city; }
  public function getCity() { return $this->city; }
  public function setCountry($country) { $this->country = $country; }
  public function getCountry() { return $this-> country;}
}

class Person {
 protected $name;
 protected $address;

 public function __construct() { $this->address = new Address; }
 public function setName($name) { $this->name = $name; }
 public function getName() { return $this->name; }
 public function __call($method, $arguments) {
  if (method_exists($this->address, $method)) {
   return call_user_func_array( array($this->address, $method), $arguments);
  }
 }
}
```

Композитным называется класс, который содержит встроенный объект другого класса, благодаря чему разработчик может легко работать как с исходным, так

и с встроенным классом. Здесь важно запомнить, что свойство `$address` содержит объект `Address`.

Следующий пример показывает, что происходит при клонировании объекта такого класса:

```
$rasmus = new Person;
$rasmus->setName('Rasmus Lerdorf');
$rasmus->setCity('Sunnyvale');

$zeev = clone $rasmus;
$zeev->setName('Zeev Suraski');
$zeev->setCity('Tel Aviv');

print $rasmus->getName() . ' lives in ' . $rasmus->getCity() . '.';
print $zeev->getName() . ' lives in ' . $zeev->getCity() . '.';
```

```
Rasmus Lerdorf lives in Tel Aviv.
Zeev Suraski lives in Tel Aviv.
```

Интересно... Вызов `setName()` работает правильно, потому что свойство `$name` содержит строку и поэтому копируется по значению. Но так как свойство `$address` содержит объект, оно копируется по ссылке и `getCity()` выдает неправильный результат.

Такая разновидность клонирования объектов называется *поверхностным клонированием* (или поверхностным копированием). С другой стороны, *глубокое клонирование* сопровождается клонированием всех задействованных объектов.

Для управления процессом клонирования объектов в классе реализуется метод `__clone()`. Когда этот метод существует, PHP позволяет `__clone()` переопределить свое поведение по умолчанию:

```
class Person {
 // ... То же, что прежде
 public function __clone() {
  $this->address = clone $this->address;
 }
}
```

Если метод `__clone()` не существует, PHP автоматически предоставляет поверхностную копию переменной, хранящейся в `$this`.

Так как все свойства уже скопированы, вам остается заменить нежелательные значения. В данном случае с `$name` проблем нет, но `$address` необходимо клонировать явно.

Теперь клонирование работает правильно:

```
$rasmus = new Person;
$rasmus->setName('Rasmus Lerdorf');
$rasmus->setCity('Sunnyvale');

$zeev = clone $rasmus;
$zeev->setName('Zeev Suraski');
$zeev->setCity('Tel Aviv');
```

```
print $rasmus->getName() . ' lives in ' . $rasmus->getCity() . '.';
print $zeev->getName() . ' lives in ' . $zeev->getCity() . '.';
```

```
Rasmus Lerdorf lives in Sunnyvale.
Zeev Suraski lives in Tel Aviv.
```

Применение оператора `clone` к объектам, хранящимся в свойствах, заставляет PHP проверить, содержат ли какие-либо из этих объектов метод `__clone()`. Если такой метод существует, то PHP вызывает его. Процедура повторяется для всех объектов на следующих уровнях вложенности.

Этот процесс, правильно клонирующий весь объект, наглядно демонстрирует, почему он называется «глубоким копированием».

См. также

Рецепт 7.9 — дополнительная информация о присваивании ссылок на объекты.

7.11. Переопределение обращений к свойствам

Задача

Требуется определить функции, выполняемые при чтении и записи свойств объектов. Это позволит вам написать обобщенный код для обращения к свойствам вашего класса.

Решение

Используйте специальные методы `__get()` и `__set()` для перехвата обращений к свойствам.

Чтобы улучшить эту абстракцию, также можно реализовать методы `__isset()` и `__unset()`, чтобы класс правильно вел себя при проверке свойства вызовом `isset()` или его удалением вызовом `unset()`.

Комментарий

Перегрузка свойств позволяет скрыть от пользователя фактическое местонахождение свойств объекта и структур данных, используемых для их хранения.

Допустим, класс `Person` хранит переменные в массиве `$__data`. (Имя переменной не обязано начинаться с двух символов подчеркивания — они только напоминают о том, что переменная используется специальным методом.)

```
class Person {
    private $__data = array();
```

```
    public function __get($property) {
        if (isset($this->__data[$property])) {
            return $this->__data[$property];
        } else {
            return false;
        }
    }

    public function __set($property, $value) {
        $this->__data[$property] = $value;
    }
}
```

Пример использования:

```
$johnwood = new Person;
$johnwood->email = 'jonathan@wopr.mil'; // Запись $user->__data['email']
print $johnwood->email;                 // Чтение $user->__data['email']

jonathan@wopr.mil
```

При записи данных `__set()` заменяет значение элемента в `$__data`. Аналогичным образом `__get()` перехватывает операции чтения и возвращает соответствующий элемент массива.

Применение этих методов и массива для хранения данных упрощает инкапсуляцию данных объекта. Вместо того, чтобы писать пару методов доступа для каждого свойства класса, достаточно написать методы `__get()` и `__set()`.

Наличие `__get()` и `__set()` позволяет использовать то, что на первый взгляд кажется открытым свойством (например, `$johnwood->name`), без нарушения инкапсуляции. Дело в том, что операции чтения и записи не выполняются с переменными свойств напрямую, а проходят через методы доступа.

Метод `__get()` получает имя свойства в своем единственном параметре. В этом методе мы проверяем, присутствует ли значение свойства в `$__data`. Если свойство обнаружено, то метод возвращает его значение; в противном случае возвращается `false`.

При чтении значения $johnwood->name фактически происходит вызов __get('name'), который возвращает $__data['name'], но для внешнего пользователя это несущественно.

Метод `__set()` получает два аргумента: имя свойства и новое значение. В остальном логика метода аналогична логике `__get()`.

Кроме сокращения количества методов в классах, эти специальные методы также упрощают реализацию централизованной проверки входных и выходных данных. Например, вот как можно ограничить использование свойств класса заранее определенным списком:

```
class Person {
    // person и email заносятся в список допустимых свойств
    protected $__data = array('person' => false, 'email' => false);
    public function __get($property) {
        if (isset($this->__data[$property])) {
            return $this->__data[$property];
        } else {
            return false;
        }
    }

    // Присваивание разрешается только для заранее
    // определенных свойств
    public function __set($property, $value) {
        if (isset($this->__data[$property])) {
            return $this->__data[$property] = $value;
        } else {
            return false;
        }
    }
}
```

В обновленной версии кода имена допустимых свойств объекта явно перечисляются при определении свойства `$__data`. Затем внутри `__set()` функция `isset()` проверяет, что все операции присваивания выполняются только с допустимыми именами свойств.

Предотвращение посторонних операций чтения и записи — причина, по которой свойство `$__data` объявляется защищенным (`protected`), а не открытым (`public`). В противном случае стало бы возможно выполнение кода следующего вида:

```
$person = new Person;
$person->__data['fake_property'] = 'fake_data';
```

Обратите внимание на эту важную подробность реализации. Если пользователи будут расширять класс, они могут добавить свойство, конфликтующее со свойством, которое должно обрабатываться `__get()` и `__set()`. По этой причине свойство в приведенном примере называется `$__data` (с двумя символами подчеркивания в начале).

Чтобы снабжать префиксами все «настоящие» свойства в классах, используются специальные методы доступа для предотвращения коллизий между обычными свойствами и свойствами, которые должны обрабатываться через `__get()` и `__set()`.

У использования `__get()` и `__set()` имеются некоторые недостатки. Во-первых, эти методы способны перехватывать только отсутствующие свойства. Если вы определите свойство в своем классе, методы `__get()` и `__set()` не будут вызываться PHP при обращении к этому свойству.

Это справедливо даже в том случае, если свойство, к которому вы обращаетесь, недоступно в текущей области видимости (например, при чтении свойства, которое существует в классе, но недоступно из-за объявления `private`). В таких случаях PHP выдает фатальную ошибку:

```
PHP Fatal error:  Cannot access private property...
```

Во-вторых, эти методы полностью исключают любую форму наследования свойств. Если родительский объект содержит метод `__get()` и вы реализуете собственную версию `__get()` в потомке, ваш объект будет работать некорректно, потому что родительский метод `__get()` не вызывается.

Строго говоря, проблема обходится вызовом `parent::__get()`, но вам придется выполнить операцию явно вместо того, чтобы получить ее «бесплатно» в контексте объектно-ориентированной архитектуры.

Наконец, иллюзия неполна, потому что она не распространяется на методы `isset()` и `unset()`. Например, при проверке `isset()` для перегруженного свойства вы получите неправильный ответ, потому что PHP не знает о необходимости вызова `__get()`.

Недостаток можно исправить, реализовав в классе собственные версии этих методов с именами `__isset()` и `__unset()`:

```
class Person {
    // person и email заносятся в список допустимых свойств

    protected $data = array('person' => false, 'email' => false);

    public function __get($property) {
        if (isset($this->data[$property])) {
            return $this->data[$property];
        } else {
            return null;
        }
    }

    // Присваивание разрешается только для заранее
    // определенных свойств
    public function __set($property, $value) {
        if (isset($this->data[$property])) {
            $this->data[$property] = $value;
        }
    }

    public function __isset($property) {
        return isset($this->data[$property]);
    }

    public function __unset($property) {
        if (isset($this->data[$property])) {
            unset($this->data[$property]);
        }
    }
}
```

Метод `__isset()` проверяет элемент `$data` и возвращает `true` или `false` в зависимости от состояния проверяемого свойства.

Аналогичным образом `__unset()` возвращает результат применения `unset()` к *реальному* свойству — или `false`, если оно не определено.

Реализация этих двух методов не обязательна при использовании `__get()` и `__set()`, но лучше предоставить ее, потому что трудно предсказать, как будут

использоваться свойства объектов. Отсутствие реализации этих методов может привести к путанице, когда другой разработчик (а может быть, даже вы сами) не будет знать (или забудет), что класс использует специальные методы доступа.

Другие причины для отказа от использования специальных методов доступа:

- Они работают относительно медленно — медленнее как прямого обращения к свойствам, так и явного написания методов доступа для всех свойств.
- Они препятствуют использованию классов `Reflection` и таких инструментов автоматического документирования кода, как `phpDocumentor`.
- Они не могут использоваться со статическими свойствами.

См. также

Документация по перегруженным методам.

7.12. Вызов методов объекта, возвращаемого другим методом

Задача

Требуется вызвать метод объекта, возвращаемого другим методом.

Решение

Вызовите второй метод прямо из первого:

```
$orange = $fruit->get('citrus')->peel();
```

Комментарий

PHP сначала вызывает `$fruit->get('citrus')`, а затем вызывает метод `peel()` для возвращенного объекта.

Вы можете спроектировать свои классы так, чтобы упростить сцепление вызовов. Пример:

```
$tweet = new Tweet;
$tweet->from('@rasmus')
    ->withStatus('PHP 6 released! #php')
    ->send();
```

Объединяя в цепочку вызовы методов, вы строите класс `Tweet` по сегментам, а затем отправляете сообщение.

Чтобы такие цепочки работали, следует возвращать `$this` из каждого сцепляемого метода. Таким образом текущий контекст сохраняется в каждом последующем методе. Так как пользователи могут выбирать вызываемые методы (и порядок их вызова), один метод должен вызываться последним — в нашем случае это метод `send()`. В нем содержится вся логика объединения компонентов и выполнения нужной операции.

В действительности этот код не отправляет сообщение (так как для Twitter API требуется поддержка OAuth), но он хорошо демонстрирует методику проектирования:

```
class Tweet {
    protected $data;

    public function from($from) {
        $data['from'] = $from;
        return $this;
    }

    public function withStatus($status) {
        $data['status'] = $status;
        return $this;
    }

    public function inReplyToId($id) {
        $data['id'] = $id;
        return $this;
    }

    public function send() {
        // Генерирование запроса Twitter API с использованием
        // информации из $data
        // Отправка https://api.twitter.com/1.1/statuses/update.json
        // с http_build_query($data)
        return $id;
    }
}

$tweet = new Tweet;
$id = $tweet->from('@rasmus')
    ->withStatus('PHP 6 released! #php')
    ->send();

$reply = new Tweet;
$id2 = $reply->withStatus('I <3 Unicode!')
    ->from('@a')
    ->inReplyToId($id)
    ->send();
```

Сцепленные вызовы могут выглядеть очень элегантно, но важно не злоупотреблять ими. Они лучше всего подходят применительно к областям с четко определенным языком (например, SQL или протоколом отправки сообщений). В приведенном примере используется Tweeter API, но электронная почта и SMS тоже подойдут.

См. также

Документация по Twitter API.

7.13. Композиция объектов

Задача

Требуется объединить два и более объектов, чтобы для внешнего пользователя они работали как единое целое.

Решение

Используйте композицию и специальные методы `__call()` и `__callStatic()` для перехвата вызовов и их соответствующего перенаправления:

```
class Address {
    protected $city;

    public function setCity($city) {
        $this->city = $city;
    }

    public function getCity() {
        return $this->city;
    }
}

class Person {
    protected $name;
    protected $address;

    public function __construct() {
        $this->address = new Address;
    }

    public function setName($name) {
        $this->name = $name;
    }

    public function getName() {
        return $this->name;
    }

    public function __call($method, $arguments) {
        if (method_exists($this->address, $method)) {
            return call_user_func_array(
                array($this->address, $method), $arguments);
        }
    }
}

$rasmus = new Person;
$rasmus->setName('Rasmus Lerdorf');
```

```
$rasmus->setCity('Sunnyvale');

print $rasmus->getName() . ' lives in ' . $rasmus->getCity() . '.';
```

При построении каждого объекта `Person` создается объект `Address`. Вызовы методов, не определенные в `Person`, перехватываются методом `__call()` и там, где это применимо, передаются вызовом `call_user_func_array()`.

Для перенаправления статических методов используется метод `__callStatic()`.

Комментарий

Для этих двух классов нельзя сказать, что `Person` «является частным случаем» `Address` и наоборот. Следовательно, расширение одного класса другим выглядит бессмысленно.

С другой стороны, эти две сущности разумно представить в виде отдельных классов, чтобы они обеспечивали максимум гибкости и возможностей повторного использования, а также предотвращали дублирование кода. Соответственно, нужно проверить, применяется ли здесь другой тип отношений — «содержит». В данном случае информация о человеке «содержит» адрес, и эти два класса разумно объединить посредством композиции.

С композицией один объект выступает в роли контейнера для одного или нескольких дополнительных объектов. Это альтернативное решение проблемы множественного наследования, потому что в объекте легко выделяются несколько меньших компонентов.

Например, объект `Person` содержит объект `Address`. Разумеется, у каждого человека есть адрес, адреса принадлежат не только людям; они также есть у коммерческих и других организаций. Следовательно, вместо жесткого кодирования адресов внутри `Person` логичнее создать отдельный класс `Address`, который может использоваться несколькими классами. Вот как это работает на практике:

```
class Address {
    protected $city;

    public function setCity($city) {
        $this->city = $city;
    }

    public function getCity() {
        return $this->city;
    }
}
class Person {
    protected $name;
    protected $address;

    public function __construct() {
        $this->address = new Address;
    }

    public function setName($name) {
```

```
        $this->name = $name;
    }

    public function getName() {
        return $this->name;
    }

    public function __call($method, $arguments) {
        if (method_exists($this->address, $method)) {
            return call_user_func_array(
                array($this->address, $method), $arguments);
        }
    }
}
```

Класс `Address` содержит свойство `city` и два метода доступа для работы с данными, `setCity()` и `getCity()`.

Класс `Person` содержит методы `setName()` и `getName()`, по аналогии с `Address`, но также в нем присутствуют еще два метода: `__construct()` и `__call()`.

Его конструктор создает объект `Address` и сохраняет его в защищенном свойстве `$address`. Это позволяет методам класса `Person` обращаться к `$address`, но предотвращает несанкционированный внешний доступ к данным класса.

В идеале при вызове метода, существующего в `Address`, PHP автоматически выполняет его. Этого не происходит, потому что `Person` не расширяет `Address`. Вам придется самостоятельно написать код, который связывает эти вызовы между собой.

Один из вариантов — методы-обертки (wrappers). Пример:

```
class Person {
    public function setCity($city) {
        $this->address->setCity($city);
    }
}
```

Метод `setCity()` передает свои данные методу `setCity()`, хранящемуся в `$address`. Это просто, но утомительно, потому что вам придется писать обертку для каждого метода.

Использование `__call()` позволяет автоматизировать процесс за счет централизации этих методов в одном месте:

```
class Person {
    public function __call($method, $arguments) {
            if (method_exists($this->address, $method)) {
                return call_user_func_array(
                    array($this->address, $method), $arguments);
            }
    }
}
```

Метод `__call()` перехватывает любые вызовы неопределенных методов в классе. При вызове ему передаются два аргумента: имя метода и массив параметров, передаваемых методу. Первый аргумент позволяет узнать, какой метод был вызван, чтобы вы могли определить, нужно ли перенаправить его в `$address`.

В данном случае должны передаваться вызовы методов, которые являются действительными методами класса `Address`. Существование проверяется вызовом `method_exists()`, при котором объект передается в первом параметре, а имя метода — во втором.

Если функция возвращает `true`, значит, метод действителен и его можно вызвать. К сожалению, остается неприятная проблема распаковки аргументов из массива `$arguments`.

Редко используемая функция `call_user_func_array()` решает эту проблему. Эта функция позволяет вызвать пользовательскую функцию и передать ей аргументы в виде массива. В первом параметре передается имя функции, а во втором — массив аргументов.

Однако в этом случае вместо функции должен вызываться метод объекта. Для таких ситуаций предусмотрен специальный синтаксис: вместо имени функции передается массив из двух элементов. В первом элементе хранится объект, а во втором — имя метода.

С этим синтаксисом `call_user_func_array()` вызывает метод объекта. Результат `call_user_func_array()` возвращается исходной вызывающей стороне, или возвращаемые значения попросту игнорируются.

В следующем примере для объекта `Person` вызываются два метода: один определен в `Person`, а другой в `Address`:

```
$rasmus = new Person;
$rasmus->setName('Rasmus Lerdorf');
$rasmus->setCity('Sunnyvale');

print $rasmus->getName() . ' lives in ' . $rasmus->getCity() . '.';
```

Хотя `setCity()` и `getCity()` не являются методами `Person`, они были включены в класс посредством композиции.

Следует учитывать, что в класс могут быть встроены дополнительные объекты; а может быть, вы захотите проявить бóльшую избирательность в отношении методов, доступных для внешнего пользователя. Для этого потребуется организовать несложную фильтрацию по имени метода.

См. также

Документация по специальным методам.

7.14. Обращение к переопределенным методам

Задача

Требуется обратиться к методу родительского класса, переопределенному в производном классе.

Решение

Укажите префикс `parent` перед именем метода:

```
class shape {
    function draw() {
        // Вывод на экран
    }
}

class circle extends shape {
   function draw($origin, $radius) {
      // Проверка данных
      if ($radius > 0) {
          parent::draw();
          return true;
      }

      return false;
   }
}
```

Комментарий

Если метод родительского класса переопределяется в производном классе, то для вызова родительской версии метода необходимо использовать специальную форму записи.

В Решении метод `draw()` переопределяется в производном классе `circle`, потому что реализация производного класса должна получать специальные параметры и проверять данные. Однако при этом все равно нужно выполнить обобщенную операциию `shape::draw()`, которая выполняет непосредственный вывод, поэтому мы вызываем `parent::draw()` в своем методе, если `$radius` больше 0.

Префикс `parent::` может использоваться только кодом внутри класса. Вызов `parent::draw()` за пределами класса приведет к ошибке разбора. Например, если бы реализация `circle::draw()` ограничивалась проверкой радиуса, то попытка вызвать `shape::draw()` для выполнения вывода работать не будет[1]:

```
$circle = new circle;
if ($circle->draw($origin, $radius)) {
    $circle->parent::draw();
}
```

Это правило распространяется и на конструкторы объектов, поэтому на практике часто встречаются конструкции следующего вида:

```
class circle {
    function __construct($x, $y, $r) {
        // Сначала вызывается конструктор shape
```

[1] При этом выдается сообщение об ошибке unexpected T_PAAMAYIM_NEKUDOTAYIM, что означает «два двоеточия» на иврите.

```
        parent::__construct();
        // Теперь выполняются действия, специфические для circle
    }
}
```

См. также

Рецепт 7.2 — дополнительная информация о конструкторах объектов; документация по родительским классам и функции `get_parent_class()`.

7.15. Динамическое создание методов

Задача

Требуется динамически предоставлять методы без их явного определения.

Решение

Воспользуйтесь специальными методами `__call()` и `__callStatic()` для перехвата вызовов методов и их соответствующего перенаправления.

Этот прием лучше всего использовать при создании объектно-реляционных отображений (ORM, Object Relational Map) или классов-заместителей (proxy). Например, вы хотите предоставить методы `findBy()`, которые преобразуются в запросы к базе данных или REST-совместимым API.

Предположим, вы хотите, чтобы приложение могло возвращать информацию о пользователях по различным критериям поиска: идентификатору, адресу электронной почты, номеру телефона. Конечно, можно создать по одному методу на каждый критерий: `findById()`, `findByEmail()`, `findByPhone()`. Однако код этих методов в основном идентичен, поэтому его можно разместить в одном месте. В этом вам поможет метод `__callStatic()`:

```
class Users {
    static function find($args) {
        // Здесь размещается реальная логика -
        // например, запрос к базе данных:
        // SELECT user FROM users WHERE $args['field'] = $args['value']
    }

    static function __callStatic($method, $args) {
        if (preg_match('/^findBy(.+)$/', $method, $matches)) {
            return static::find(array('field' => $matches[1],
                                      'value' => $args[0]));
        }
    }
}

$user = User::findById(123);
$user = User::findByEmail('rasmus@php.net');
```

При вызове `findById()` PHP передает запрос `__callStatic()`. Внутри метода результат ищет запросы, начинающиеся с `findBy`, и извлекает остальные символы. Полученное значение и первый аргумент функции упаковываются и передаются методу `Users::find()`, в котором размещается «настоящая» логика.

См. также

Документация по перегруженным методам; Рецепт 7.18 — дополнительная информация о вызове статических методов.

7.16. Полиморфизм методов

Задача

Требуется выполнить разный код в зависимости от количества и типа аргументов, переданных методу.

Решение

PHP не обладает встроенной поддержкой полиморфизма. Тем не менее полиморфизм можно смоделировать при помощи различных функций проверки типа. Следующая функция `combine()` использует методы `is_numeric()`, `is_string()`, `is_array()` и `is_bool()`:

```
// combine() выполняет суммирование чисел, конкатенацию строк,
// слияние массивов и поразрядную операцию AND.
function combine($a, $b) {
    if (is_int($a) && is_int($b))     {
        return $a + $b;
    }

    if (is_float($a) && is_float($b))   {
        return $a + $b;
    }

    if (is_string($a) && is_string($b))  {
        return "$a$b";
    }

    if (is_array($a) && is_array($b))   {
        return array_merge($a, $b);
    }

    if (is_bool($a) && is_bool($b))    {
        return $a & $b;
    }

    return false;
}
```

Комментарий

Так как тип переменной не может объявляться в прототипе метода, PHP не позволяет условно выполнять разные методы в зависимости от их сигнатуры, как это делается в Java и C++. Тем не менее можно взять одну функцию и воспользоваться конструкцией `switch`, чтобы смоделировать этот механизм «вручную».

Например, PHP позволяет редактировать графические изображения средствами GD. В классе изображения было бы удобно иметь возможность передавать либо местоположение изображения (удаленное или локальное), либо дескриптор, присвоенный PHP существующему потоку графических данных. Следующий класс `Image` реализует эту возможность:

```
class Image {
    protected $handle;
    function ImageCreate($image) {
        if (is_string($image)) {
            // Простая попытка угадать тип файла

            // Получение суффикса файла
            $info = pathinfo($image);
            $extension = strtolower($info['extension']);
            switch ($extension) {
            case 'jpg':
            case 'jpeg':
                $this->handle = ImageCreateFromJPEG($image);
                break;
            case 'png':
                $this->handle = ImageCreateFromPNG($image);
                break;
            default:
                die('Images must be JPEGs or PNGs.');
            }
        } elseif (is_resource($image)) {
            $this->handle = $image;
        } else {
            die('Variables must be strings or resources.');
        }
    }
}
```

В этом примере любая передаваемая строка рассматривается как путь к файлу, и функция `pathinfo()` выделяет расширение файла. Зная расширение, мы пытаемся предположить, какая из функций `ImageCreateFrom()` правильно откроет изображение и создаст дескриптор.

Если передается не строка, значит, мы имеем дело с потоком GD, который имеет тип `resource`. Так как преобразование в этом случае не требуется, поток присваивается `$handle` напрямую. Конечно, если бы этот класс использовался в реальном приложении, то в нем была бы реализована более надежная обработка ошибок.

Полиморфизм также распространяется на методы с разным количеством аргументов. Код определения количества аргументов внутри метода идентичен коду

обработки функции с переменным количеством аргументов с использованием `func_num_args()`. Эта возможность обсуждается в Рецепте 6.6.

См. также

Рецепт 6.6 — функции с переменным количеством аргументов; документация по функциям `is_string()`, `is_resource()` и `pathinfo()`.

7.17. Определение констант класса

Задача

Требуется определять константы на уровне класса, а не на глобальном уровне.

Решение

Определите константы как свойства класса, но используйте метку `const`:

```
class Math {
    const pi = 3.14159; // Универсальные
    const  e = 2.71828; // константы
}
$area = math::pi * $radius * $radius;
```

Комментарий

PHP переосмысливает концепцию глобальных констант применительно к классам. По сути, константы представляют собой свойства, значение которых должно оставаться неизменным.

Константы классов объявляются с ключевым словом `const`:

```
class Math {
    const pi = 3.14159; // Универсальные
    const  e = 2.71828; // константы
}

$area = math::pi * $radius * $radius;
```

К константам, как и к статическим свойствам, можно обращаться без предварительного создания нового объекта; при этом используется синтаксис из двух двоеточий (`::`). Чтобы использовать константу внутри класса, снабдите ее имя префиксом `self::`.

В отличие от свойств, константы не имеют префикса `$`:

```
class Circle {
    const pi = 3.14159;
```

```
    protected $radius;
    public function __construct($radius) {
        $this->radius = $radius;
    }

    public function circumference() {
        return 2 * self::pi * $this->radius;
    }
}

$c = new circle(1);
print $c->circumference();
```

Этот пример создает окружность с радиусом 1, после чего вызывает метод `circumference` для вычисления длины окружности:

```
define('pi', 10); // Глобальная константа

class Circle {
    const pi = 3.14159; // Константа класса
    protected $radius;

    public function __construct($radius) {
        $this->radius = $radius;
    }

    public function circumference() {
        return 2 * pi * $this->radius;
    }

}

$c = new circle(1);
print $c->circumference();
```

Сюрприз! PHP использует значение 10 вместо 3,14159, поэтому новый ответ будет равен 20 вместо 6,28318.

И хотя вероятность случайного переопределения π невелика (скорее всего, вы все равно будете использовать встроенную константу `M_PI`), такие просчеты возможны.

Константе нельзя присвоить значение выражения, и константы не могут использовать информацию, переданную сценарию:

```
// Недопустимо
class permissions {
    const    read = 1 << 2;
    const   write = 1 << 1;
    const execute = 1 << 0;
}

// Недопустимо и небезопасно
class database {
    const debug = $_REQUEST['debug'];
}
```

Ни константы класса `permissions`, ни константа `debug` из класса `database` недопустимы, потому что их значения не фиксированы. Недопустим даже первый пример, `1 << 2`, в котором PHP не нужно читать внешние данные.

Так как к константам нужно указывать префикс `self::` или имя класса, имя класса не может динамически вычисляться во время выполнения. Оно должно быть объявлено заранее. Пример:

```
class Constants {
    const pi = 3.14159;

    // Остальной код класса
}

$class = 'Constants';

print $class::pi;
```

В этом случае происходит ошибка разбора несмотря на то, что такие конструкции допустимы для неконстантных выражений (например, `$class->pi`).

См. также

Документация по константам классов.

7.18. Определение статических методов и свойств

Задача

Требуется определить в классе методы, которые могут вызываться без создания объекта.

Решение

Объявите статический метод:

```
class Format {
    public static function number($number, $decimals = 2,
                                  $decimal = '.', $thousands = ',') {
        return number_format($number, $decimals, $decimal, $thousands);
    }
}

print Format::number(1234.567);

1,234.57
```

Комментарий

Иногда требуется определить в объекте набор методов, которые должны вызываться без создания объекта (то есть конкретного экземпляра). В PHP объявление метода с ключевым словом `static` позволяет вызывать его напрямую:

```
class Format {
    public static function number($number, $decimals = 2,
                                  $decimal = '.', $thousands = ',') {
        return number_format($number, $decimals, $decimal, $thousands);
    }
}

print Format::number(1234.567);
```

```
1,234.57
```

Поскольку статическим методам не нужен конкретный объект, используйте имя класса вместо объекта. Не ставьте знак `$` перед именем класса.

При обращении к статическому методу вместо стрелки (`->`) используется двойное двоеточие (`::`) — оно сообщает PHP, что метод является статическим. Таким образом, в приведенном примере для обращения к методу `number()` класса `Format` используется запись `Format::number()`.

Числовое форматирование не зависит от других свойств или методов объекта. По этой причине логично объявить метод статическим. Например, в приложении интернет-магазина это позволит отформатировать цены товаров в корзине всего одной строкой кода и при этом использовать объект вместо глобальной функции.

Внутри класса, в котором определен статический метод, для обращения к нему используется обозначение `self`:

```
class Format {
    public static function number($number, $decimals = 2,
                                  $decimal = '.', $thousands = ',') {
        return number_format($number, $decimals, $decimal, $thousands);
    }
    public static function integer($number) {
        return self::number($number, 0);
    }
}

print Format::number(1234.567)  . "\n";
print Format::integer(1234.567) . "\n";
```

```
1,234.57
1,235
```

Здесь метод `integer()` обращается к другому методу, определенному в классе `Format`, а именно `number()`. По этой причине он использует запись `self::number()`.

Статические методы не работают с конкретным экземпляром класса, в котором они определяются. Во время нахождения в методе PHP не создает временный

объект, с которым выполняются операции. По этой причине в статическом методе не может использоваться обозначение `$this` — не существует никакого «текущего» объекта, с которым можно было бы работать. Вызов статического метода в этом отношении похож на вызов обычной функции.

С использованием `self::` связано одно потенциальное затруднение: это ключевое слово не следует тем же правилам наследования, что и нестатические методы. В данном случае `self::` всегда относится к классу, в котором находится определение, независимо от того, вызывается ли метод из этого класса или из его потомка. Чтобы изменить это поведение, используйте спецификатор `static::`, как в следующем примере ORM:

```
class Model {
    protected static function validateArgs($args) {
        throw new Exception("You need to override this in a subclass!");
    }

    public static function find($args) {
        static::validateArgs($args);
        $class = get_called_class();
        // Здесь может выполняться запрос к базе данных, например:
        // SELECT * FROM $class WHERE ...
    }
}

class Bicycle extends Model {
    protected static function validateArgs($args) {
        return true;
    }
}

Bicycle::find(['owner' => 'peewee']);
```

С `self::` PHP осуществляет привязку к методу `Model::validateArgs()`, не поддерживающему проверку данных для конкретной модели. С другой стороны, со спецификатором `static::` PHP откладывает привязку до того момента, когда будет знать, из какого класса вызывается метод. Такой механизм называется *поздней статической привязкой*.

Внутри метода `find()` для генерирования кода SQL необходимо имя вызывающего класса. Вы не сможете использовать классы `Reflection` и константу `__CLASS__`, потому что они возвращают `Model`; для получения необходимой информации на стадии выполнения вызывается метод `get_called_class()`.

В PHP также имеется механизм *статических свойств*. Эти свойства являются общими для всех объектов класса. Таким образом, статические свойства напоминают глобальные переменные с областью видимости уровня класса. Одна из причин использования статических свойств — совместное использование подключения к базе данных несколькими объектами `Database`. По соображениям эффективности не стоит создавать новое подключение к базе данных при каждом создании экземпляра `Database`. Вместо этого подключение осуществляется в самый первый раз, а затем повторно используется всеми дополнительными экземплярами:

```
class Database {
    private static $dbh = NULL;
    public function __construct($server, $username, $password) {
        if (self::$dbh == NULL) {
            self::$dbh = db_connect($server, $username, $password);
        } else {
            // Повторное использование существующего подключения
        }
    }
}

$db  = new Database('db.example.com', 'web', 'jsd6w@2d');
// Выполнение группы запросов

$db2 = new Database('db.example.com', 'web', 'jsd6w@2d');
// Дополнительные запросы
```

Для статических свойств, как и статических методов, используется синтаксис с двойным двоеточием (`::`). Для ссылки на статическое свойство в классе используется специальный префикс `self`. Для статических свойств и методов `self` играет ту же роль, что и `$this` для свойств и методов объектов.

Конструктор использует запись `self::$dbh` для обращения к статическому свойству `connection`. После создания объекта `$db` в `dbh` по-прежнему содержится `NULL`, поэтому конструктор вызывает `db_connect()` для установления нового подключения к базе данных.

При создании `$db2` это не происходит, потому что в `dbh` уже хранится дескриптор базы данных.

См. также

Документация по ключевому слову `static`.

7.19. Управление сериализацией объектов

Задача

Требуется управлять поведением объекта в процессе сериализации (`serialize()`) и (`unserialize()`). Управление сериализацией может быть полезно при открытии и закрытии подключений к удаленным ресурсам: базам данных, файлам и веб-службам.

Решение

Определите специальные методы `__sleep()` и `__wakeUp()`:

```
class LogFile {
    protected $filename;
    protected $handle;
```

```
    public function __construct($filename) {
        $this->filename = $filename;
        $this->open();
    }

    private function open() {
        $this->handle = fopen($this->filename, 'a');
    }

    public function __destruct($filename) {
        fclose($this->handle);
    }

    // Вызывается при сериализации объекта.
    // Метод должен возвращать массив сериализуемых свойств объекта.
    public function __sleep() {
        return array('filename');
    }

    // Вызывается при десериализации объекта.
    public function __wakeUp() {
        $this->open();
    }
}
```

Комментарий

При сериализации объекта в PHP сохраняются все свойства объекта. Однако в сериализацию не входят подключения и полученные дескрипторы, относящиеся к внешним ресурсам: базам данных, файлам и веб-службам.

Их приходится заново получать при десериализации объекта, в противном случае объект будет вести себя некорректно. Это можно сделать явно в коде, но лучше поручить PHP сделать автоматически.

Задача решается при помощи специальных методов `__sleep()` и `__wakeUp()`. При вызове `serialize()` для объекта PHP вызывает `__sleep()`; при вызове `unserialize()` вызывается метод `__wakeUp()`.

Класс `LogFile` в Решении содержит пять простых методов. Конструктор получает имя файла и сохраняет его на будущее. Метод `open()` открывает файл и сохраняет его дескриптор, который закрывается в деструкторе объекта.

Метод `__sleep()` возвращает массив свойств, которые должны сохраняться во время сериализации объекта. Так как дескрипторы файлов не сохраняются между сериализациями, возвращается только массив `array('filename')` — это все, что нужно сохранить.

Именно поэтому при повторной сериализации объекта файл приходится открывать заново. Это делается внутри метода `__wakeUp()`, вызывающего тот же метод `open()`, который использовался конструктором.

Так как метод `__wakeUp()` не может получать аргументы, он должен получить имя файла из другого источника. К счастью, метод может обращаться к свойствам объекта; поэтому мы и сохранили в них имя файла.

Важно понимать, что один объект может быть сериализован несколько раз за время обработки одного запроса и даже может использоваться после сериализации. Следовательно, в `__sleep()` не следует делать ничего такого, что могло бы помешать любой из этих двух операций. Метод `__sleep()` должен использоваться только для исключения свойств, которые не должны сериализоваться, потому что они занимают слишком много места на диске или вычисляются на основе других данных и должны пересчитываться или обновляться иным образом при десериализации объекта.

Этим и объясняется то, что вызов `fclose()` присутствует в деструкторе, но не в `__sleep()`.

См. также

Документация по специальным методам; функция `unserialize()` и функция `serialize()`.

7.20. Интроспекция

Задача

Требуется проанализировать объект и узнать, какие методы и свойства он содержит; это позволит вам писать обобщенный код, работающий с произвольным объектом независимо от его типа.

Решение

Используйте классы `Reflection` для получения информации об объекте.

Чтобы получить краткую сводку для класса, вызовите `Reflection::export()`:

```
// Получение информации о car
Reflection::export(new ReflectionClass('car'));
```

Или проверьте существование конкретных компонентов класса:

```
$car = new ReflectionClass('car');
if ($car->hasMethod('retractTop')) {
    // Содержит метод retractTop
}
```

Комментарий

Ситуации, в которых у вас имеется объект, но вы не можете просмотреть его исходный код, встречаются нечасто. Тем не менее классы `Reflection` позволяют на программном уровне получить информацию как по объектно-ориентированным

сущностям (классам, методам, свойствам), так и по обычным функциям, расширениям и т. д.

Интроспекция может пригодиться в проектах, применяемых к широкому диапазону разных классов: например, автоматизированного создания документации по классам, обобщенным объектным отладчикам и средствам сохранения состояния (таким как `serialize()`).

Для демонстрации классов `Reflection` в листинге 7.1 приведен пример класса `Person`, в котором используются многие объектно-ориентированные возможности PHP.

Листинг 7.1. Класс Person

```
class Person {
    public $name;
    protected $spouse;
    private $password;
    public function __construct($name) {
        $this->name = $name
    }

   public function getName() {
        return $name;
    }

    protected function setSpouse(Person $spouse) {
        if (!isset($this->spouse)) {
            $this->spouse = $spouse;
        }
    }

    private function setPassword($password) {
        $this->password = $password;
    }
}
```

Чтобы получить краткую сводку для данного класса, вызовите `Reflection::export()`:

```
Reflection::export(new ReflectionClass('Person'));
```

```
Class [ <user> class Person ] {
  @@ /www/reflection.php 3-25

  - Constants [0] {
  }

  - Static properties [0] {
  }

  - Static methods [0] {
  }

  - Properties [3] {
    Property [ <default> public $name ]
    Property [ <default> protected $spouse ]
    Property [ <default> private $password ]
  }
```

```
  - Methods [4] {
    Method [ <user> <ctor> public method _ _construct ] {
      @@ /www/reflection.php 8 - 10

      - Parameters [1] {
        Parameter #0 [ $name ]
      }
    }

    Method [ <user> public method getName ] {
      @@ /www/reflection.php 12 - 14
    }

    Method [ <user> protected method setSpouse ] {
      @@ /www/reflection.php 16 - 20

      - Parameters [1] {
        Parameter #0 [ Person or NULL $spouse ]
      }
    }

    Method [ <user> private method setPassword ] {
      @@ /www/reflection.php 22 - 24

      - Parameters [1] {
        Parameter #0 [ $password ]
      }
    }
  }
}
```

Статический метод `Reflection::export()` получает объект класса `ReflectionClass` и возвращает довольно подробную информацию о нем. Как видите, в сводке указано количество констант, статических свойств, статических методов, свойств и методов класса. Каждая запись делится на составные части. Например, во всех записях указываются спецификаторы видимости (`private`, `protected` или `public`), а под определением методов приводятся списки параметров.

`Reflection::export()` не только сообщает информацию о файле, в котором все определяется, но даже выдает номера строк! Это позволяет извлечь код из файла и разместить его в документации.

В листинге 7.2 приведен короткий сценарий командной строки, который ищет файл с заданным именем и начальный номер строки метода или функции.

Листинг 7.2. Использование интроспекции для поиска определений функций и методов

```
if ($argc < 2) {
    print "$argv[0]: function/method, classes1.php [, ... classesN.php]\n";
    exit;
}

// Получение имени функции
$function = $argv[1];

// Включение файлов
foreach (array_slice($argv, 2) as $filename) {
```

```
    include_once $filename;
}

try {
    if (strpos($function, '::')) {
        // Это метод
        list ($class, $method) = explode('::', $function);
        $reflect = new ReflectionMethod($class, $method);
    } else {
        // Это функция
        $reflect = new ReflectionFunction($function);
    }

    $file = $reflect->getFileName();
    $line = $reflect->getStartLine();

    printf ("%s | %s | %d\n", "$function()", $file, $line);
} catch (ReflectionException $e) {
    printf ("%s not found.\n", "$function()");
}
```

В первом аргументе передается имя функции или метода, а в остальных аргументах передаются включаемые файлы.

На следующем шаге мы определяем, что представляет первый аргумент — метод или функцию. Так как методы задаются в формате *класс*::*метод*, их можно отличать от функций при помощи `strpos()`.

Если в аргументе передается метод, вызов `explode()` отделяет имя класса от метода и оба имени передаются `ReflectionMethod`. Если это функция, мы можем сразу создать объект `ReflectionFunction` без всяких сложностей.

Так как `ReflectionMethod` расширяет `ReflectionFunction`, мы можем вызывать `FileName()` и `getStartLine()` для любого из этих классов. Полученная информация выводится вызовом `printf()`.

При попытке создать объект `ReflectionMethod` или `ReflectionFunction` с именем неопределенного метода эти классы выдают исключение `ReflectionException`. В данном примере это исключение перехватывается с выводом сообщения об ошибке.

Более сложный сценарий, который выводит ту же информацию для всех методов и функций, определенных пользователем, приведен в Рецепте 7.24.

Если вам нужно просто в общих чертах ознакомиться с объектом и вы не хотите возиться с классами `Reflection`, воспользуйтесь функцией `var_dump()`, `var_export()` или `print_r()` для вывода значений объекта. Все эти функции выводят информацию по-разному; `var_export()` может вернуть данные вместо того, чтобы выводить их.

См. также

Рецепт 5.8 — дополнительная информация о выводе переменных; документация по интроспекции и функциям `var_dump()`, `var_export()` и `print_r()`.

7.21. Проверка объекта на принадлежность к определенному классу

Задача

Требуется проверить, является ли объект экземпляром заданного класса.

Решение

Чтобы проверить, является ли значение, переданное в аргументе функции, объектом заданного класса, укажите имя этого класса в прототипе функции:

```
public function add(Person $person) {
        // Добавление $person в адресную книгу
    }
}
```

В других контекстах используйте оператор `instanceof`:

```
$media = get_something_from_catalog();
if ($media instanceof Book) {
  // Выполнить действия для книги
} else if ($media instanceof DVD) {
  // Смотреть фильм
}
```

Комментарий

Один из способов обеспечения контроля над объектами основан на использовании *рекомендаций типов*. Рекомендация типа сообщает PHP о том, что объект, переданный функции или методу, относится к определенному классу. Для этого имя класса указывается в прототипе функции или метода. Также можно потребовать, чтобы аргумент был массивом, при помощи ключевого слова `array`. Впрочем, этот механизм работает только для классов и массивов, но не для других типов переменных. Например, рекомендации типа не могут использоваться со строками или целыми числами.

Например, следующий фрагмент требует, чтобы первый аргумент метода `add()` класса `AddressBook` относился к типу `Person`:

```
class AddressBook {
    public function add(Person $person) {
        // Добавление $person в адресную книгу
    }
}
```

Если теперь при вызове `add()` передать строку, PHP выдаст фатальную ошибку:

```
$book = new AddressBook;
$person = 'Rasmus Lerdorf';
```

```
$book->add($person);
PHP Fatal error:  Argument 1 must be an object of class Person in...
```

Размещение рекомендации типа в первом аргументе объявления функции эквивалентно добавлению в функцию следующего кода PHP:

```
public function add($person) {
        if (!($person instanceof Person)) {
                die("Argument 1 must be an instance of Person");
        }
}
```

Оператор `instanceof` проверяет, является ли объект экземпляром заданного класса. Приведенный код проверяет, что объект `$person` относится к классу `Person`.

Оператор `instanceof` также возвращает `true` для классов, производных от проверяемого класса. Пример:

```
class Person { /* ... */ }

class Kid extends Person { /* ... */ }

$kid = new Kid;

if ($kid instanceof Person) {
        print "Kids are people, to.\n";
}

Kids are people, too.
```

Наконец, при помощи оператора `instanceof` можно проверить, реализует ли класс заданный интерфейс:

```
interface Nameable {
    public function getName();
    public function setName($name);
}

class Book implements Nameable {
        private $name;
    public function getName() {
        return $this->name;
    }

    public function setName($name) {
        return $this->name = $name;
    }
}

$book = new Book;
if ($book instanceof Book) {
        print "You can name a Book.\n";
}

You can name a Book
```

Дополнительным преимуществом рекомендаций типов является включение документации API непосредственно в класс. Если вы видите, что конструктор класса получает тип `Event`, то вы точно знаете, что ему нужно передать. Кроме того, код и «документация» всегда синхронизированы, потому что документация генерируется непосредственно по определению класса.

Рекомендации типов также могут использоваться в определениях интерфейсов, что позволяет уточнить определение интерфейса.

Впрочем, за рекомендации типов приходится расплачиваться сокращением гибкости. В параметре с рекомендацией типа не может передаваться более одного типа объектов, так что рекомендации накладывают некоторые ограничения на проектирование иерархий классов.

Кроме того, нарушение рекомендации типа имеет весьма серьезные последствия — сценарий аварийно завершается с фатальной ошибкой. В контексте веб-приложений желательно управлять обработкой ошибок и осуществлять корректное восстановление при ошибках такого рода. Реализация собственного механизма проверки типов в методах позволяет при желании вывести страницу с описанием ошибки.

Наконец, в отличие от некоторых языков, рекомендации типов не могут применяться к возвращаемым значениям, поэтому вы не сможете потребовать, чтобы некоторая функция возвращала объект определенного типа.

См. также

Документация по рекомендациям типов и `instanceof`.

7.22. Автоматическая загрузка файлов классов при создании объекта

Задача

Вам не хотелось бы включать все определения классов в каждую страницу. Вместо этого нужно динамически загружать только те файлы, которые необходимы для этой страницы.

Решение

Воспользуйтесь специальным методом `__autoload()`:

```
function __autoload($class_name) {
    include "$class_name.php";
}

$person = new Person;
```

Комментарий

Обычно при попытке создания объекта класса с отсутствующим определением выдается фатальная ошибка, потому что PHP не может найти искомое определение. По этой причине для страниц обычно загружаются все потенциально используемые классы независимо от того, используются они фактически или нет.

Побочным эффектом такого подхода является возрастание времени обработки, потому что PHP приходится обрабатывать все классы, даже неиспользуемые. Одно из возможных решений — загрузка недостающего кода по мере надобности функцией `__autoload()`, которая вызывается при создании экземпляров неопределенных классов.

Например, следующая реализация включает все классы, используемые вашим сценарием:

```
function __autoload($class_name) {
    include "$class_name.php";
}
$person = new Person;
```

В единственном параметре функции `__autoload()` передается имя класса. В приведенном примере функция присоединяет к `$class_name` расширение `.php` и пытается включить файл с полученным именем. Таким образом, при создании нового объекта `Person` функция ищет файл `Person.php` в пути `include_path`.

Если `__autoload()` не удается загрузить определение класса для создаваемого объекта, PHP завершается с фатальной ошибкой, как и при отсутствии определения класса без автозагрузки классов.

Если вы следуете соглашениям об именах PSR-0, используйте код с GitHub. В этом случае можно поступить следующим образом:

```
use Mysite\Person;
$person = new Person;
```

Если класс `Person` не определен, то имя передается `__autoload()`. Функция вычисляет файл на основании пространства имен и имени класса.

Хотя использование `__autoload()` немного увеличивает время обработки при добавлении класса, но она вызывается только один раз для класса. Создание других объектов того же класса не приводит к повторным вызовам `__autoload()`.

Прежде чем применять `__autoload()`, проведите хронометражные тесты и убедитесь в том, что дополнительные затраты на открытие, чтение и закрытие нескольких файлов не превосходят затраты на разбор неиспользуемых классов.

В частности, при использовании opcode-кэша — такого, как `OPcache` — использование `__autoload()` и `include_once` может повредить производительности. Для достижения оптимального результата следует включать все файлы в начало сценария и следить за тем, чтобы никакие файлы не включались дважды.

См. также

Рецепт 27.3 — дополнительная информация о PSR-0; документация по автозагрузке.

7.23. Динамическое создание объекта

Задача

Требуется создать объект, но имя класса остается неизвестным до выполнения кода. Например, на локализованном сайте должен создаваться объект, соответствующий конкретному языку. Тем не менее до запроса страницы вы не знаете, какой язык следует выбрать.

Решение

Укажите переменную в качестве имени класса:

```
$language = $_REQUEST['language'];
$valid_langs = array('en_US' => 'US English',
                     'en_UK' => 'British English',
                     'es_US' => 'US Spanish',
                     'fr_CA' => 'Canadian French');

if (isset($valid_langs[$language]) && class_exists($language)) {
    $lang = new $language;
}
```

Комментарий

Иногда полное имя класса, объект которого требуется создать, неизвестно до момента выполнения, но вы знаете его часть. Тем не менее хотя следующий фрагмент содержит действительный код PHP:

```
$class_name = 'Net_Ping';
$class = new $class_name;               // new Net_Ping
```

этот фрагмент недействителен:

```
$partial_class_name = 'Ping';
$class = new "Net_$partial_class_name"; // new Net_Ping
```

А этот фрагмент допустим:

```
$partial_class_name = 'Ping';
$class_prefix = 'Net_';

$class_name = "$class_prefix$partial_class_name";
$class = new $class_name;               // new Net_Ping
```

Таким образом, вы не сможете создать объект, если его имя класса строится посредством конкатенации на том же шаге. С другой стороны, простые имена пере-

менных допустимы, поэтому проблема решается предварительным выполнением конкатенации.

См. также

Документация по функции `class_exists()`.

7.24. Программа: whereis

Хотя такие инструменты, как phpDocumentor, предоставляют подробную информацию о целых сериях классов, иногда бывает полезно быстро получить список всех функций и методов, определенных в списке файлов.

Программа в листинге 7.3 перебирает файлы по списку, включает их, а затем использует классы `Reflection` для получения информации о них. После построения списка программа сортирует функции и методы в алфавитном порядке и выводит полученный список.

Листинг 7.3. whereis

```
if ($argc < 2) {
    print "$argv[0]: classes1.php [, ...]\n";
    exit;
}

// Включение файлов
foreach (array_slice($argv, 1) as $filename) {
    include_once $filename;
}

// Получение информации о методах и функциях.
// Начинаем с классов.
$methods = array();
foreach (get_declared_classes() as $class) {
    $r = new ReflectionClass($class);
    // Исключение встроенных классов
    if ($r->isUserDefined()) {
        foreach ($r->getMethods() as $method) {
            // Eliminate inherited methods
            if ($method->getDeclaringClass()->getName() == $class) {
                $signature = "$class::" . $method->getName();
                $methods[$signature] = $method;
            }
        }
    }
}

// Затем добавляются функции
$functions = array();
$defined_functions = get_defined_functions();
foreach ($defined_functions['user'] as $function) {
    $functions[$function] = new ReflectionFunction($function);
}
```

```
// Методы сортируются в алфавитном порядке по классам
function sort_methods($a, $b) {
    list ($a_class, $a_method) = explode('::', $a);
    list ($b_class, $b_method) = explode('::', $b);

    if ($cmp = strcasecmp($a_class, $b_class)) {
        return $cmp;
    }

    return strcasecmp($a_method, $b_method);
}
uksort($methods, 'sort_methods');

// Алфавитная сортировка функций.
// Из списка нужно исключить функцию сортировки.
unset($functions['sort_methods']);
// Сортировка
ksort($functions);

// Вывод информации
foreach (array_merge($functions, $methods) as $name => $reflect) {
    $file = $reflect->getFileName();
    $line = $reflect->getStartLine();

    printf ("%-25s | %-40s | %6d\n", "$name()", $file, $line);
}
```

Код использует классы `Reflection` и пару функций PHP, `get_declared_classes()` и `get_declared_functions()`, которые не входят в классы `Reflection`, но упрощают интроспекцию.

Важно отфильтровать из результатов все встроенные классы и функции PHP; в противном случае отчет будет содержать информацию не столько о вашем коде, сколько о вашей установке PHP. Эта задача может решаться двумя разными способами. Так как `get_declared_classes()` не различает пользовательские и внутренние классы, поэтому для проверки вызывается метод `ReflectionClass ::isUserDefined()`. Вызов функции `get_defined_function()` выполняет вычисления за вас, помещая информацию в элемент массива `user`.

Выходные данные удобнее просматривать в отсортированном виде, поэтому сценарий сортирует массивы методов и функций. Так как один метод может присутствовать сразу в нескольких классах, необходимо использовать пользовательский метод сортировки `sort_methods()`, которая сначала сравнивает два метода по именам классов, а затем по именам методов.

После того как данные будут отсортированы, остается относительно простая задача: перебрать объединенные массивы, собрать имена файлов и номера начальных строк и вывести отчет.

Результаты выполнения сценария для класса PEAR `HTTP`:

```
HTTP::Date()              | /usr/lib/php/HTTP.php                    |     38
HTTP::head()              | /usr/lib/php/HTTP.php                    |    144
HTTP::negotiateLanguage() | /usr/lib/php/HTTP.php                    |     77
HTTP::redirect()          | /usr/lib/php/HTTP.php                    |    186
```

8 Основы веб-программирования

8.0. Введение

Вероятно, веб-программирование — главная причина, по которой вы читаете эту книгу. Именно ради веб-программирования была написана первая версия PHP, и эта же причина обеспечивает нынешнюю популярность этой технологии. С PHP можно легко создавать динамические веб-программы, которые способны сделать почти все. В других главах рассматриваются различные возможности PHP: веб-службы, регулярные выражения, работа с базами данных и файловый ввод/вывод. Все эти возможности являются частью веб-программирования, но эта глава посвящена ключевым концепциям и организационным вопросам, которые расширяют ваши возможности веб-программирования.

Запросы HTTP не имеют состояния; каждый последующий запрос не связан с предыдущим. С другой стороны, cookie могут связывать разные запросы от одного пользователя. Это упрощает построение таких подсистем, как покупательская корзина в интернет-магазине или отслеживание истории поиска пользователя. Рецепты 8.1, 8.2 и 8.3 показывают, как выполняются операции записи, чтения и удаления cookie — маленьких фрагментов текста, которые сервер приказывает браузеру передавать вместе с запросами, выдаваемыми браузером.

Также данные могут передаваться в строке запроса и в теле запроса. В Рецепте 8.4 описан процесс построения URL-адреса, включающего строку запроса, с кодированием специальных символов и обработкой сущностей HTML. Аналогичным образом в Рецепте 8.5 представлена информация о чтении данных, передаваемых в теле запроса и не являющихся данными формы, что не позволяет PHP автоматически преобразовать их в `$_POST`.

В следующих рецептах продемонстрировано использование аутентификации для защиты веб-страниц паролями. Специальные средства PHP для работы с базовой аутентификацией HTTP объясняются в Рецепте 8.6. На практике чаще рекомен-

дуется использовать собственную реализацию аутентификации на основе cookie, как объясняется в Рецепте 8.7.

`Cookies` и `Authentication` — два вполне конкретных заголовка HTTP. О том, как прочитать любой заголовок HTTP, рассказано в Рецепте 8.8, а записи заголовков посвящен Рецепт 8.9.

В Рецепте 8.10 рассматривается назначение кода статуса HTTP. В Рецепте 8.11 читатели узнают, как перенаправить пользователя на веб-страницу, отличную от изначально запрошенной.

Три следующих рецепта относятся к управлению выводом. Рецепт 8.12 показывает, как выполнить принудительную отправку вывода браузеру. В Рецепте 8.13 объясняются функции буферизации вывода, позволяющие перехватить выходные данные или отложить вывод до обработки всей страницы. Автоматическое сжатие вывода рассматривается в Рецепте 8.14.

Следующие два рецепта посвящены взаимодействию с внешними переменными: переменными окружения и параметрами конфигурации PHP. В Рецептах 8.15 и 8.16 рассматриваются переменные окружения. Если вы используете веб-сервер Apache, то можете использовать средства из Рецепта 8.17 для взаимодействия с другими модулями Apache из программ PHP.

В Рецепте 8.18 рассматривается идентификация мобильных браузеров, чтобы вы могли предоставить пользователям альтернативные версии своего сайта.

В эту главу также включены три программы, демонстрирующие некоторые из описанных концепций. В Рецепте 8.19 описана проверка учетных записей пользователей, основанная на отправке новым пользователям сообщения электронной почты со специальной ссылкой. Если пользователь не перейдет по ссылке в течение недели после получения сообщения, его учетная запись удаляется. В Рецепте 8.20 рассматривается небольшой пример вики-системы, в которой каждая страница сайта доступна для редактирования из браузера. Рецепт 8.21 показывает, как разобрать заголовок HTTP `Range` для возвращения заданных частей файла. Это позволяет клиенту возобновить прерванную загрузку с той точки, на которой она была прервана.

8.1. Запись cookie

Задача

Требуется записать cookie, чтобы ваш сайт мог распознавать последующие запросы от того же браузера.

Решение

Вызовите функцию `setcookie()` с именем и значением cookie:

```
setcookie('flavor','chocolate chip');
```

Комментарий

Cookie передаются с заголовками HTTP, поэтому если вы не используете буферизацию вывода, функцию `setcookie()` необходимо вызвать перед генерированием вывода.

Чтобы управлять поведением cookie, передайте `setcookie()` дополнительные аргументы. В третьем аргументе `setcookie()` передается срок действия, представленный в виде временной метки. Например, в следующем примере cookie перестает действовать 3 декабря 2014 года в 12:00 GMT:

```
setcookie('flavor','chocolate chip',1417608000);
```

Если третий аргумент `setcookie()` отсутствует (или содержит пустое значение), cookie перестает действовать с закрытием браузера. Кроме того, во многих системах срок действия cookie не может превышать 2147483647 — максимальную временную метку, которая может быть представлена 32-разрядным целым числом (см. «Введение» в главе 3).

Четвертый аргумент `setcookie()` содержит путь. Значение cookie отправляется серверу только при запросе страниц, путь к которым начинается с заданной строки. Например, в следующем примере cookie отправляется только для страниц, путь которых начинается с `/products/`:

```
setcookie('flavor','chocolate chip',0,'/products/');
```

В URL-адресе страницы, записывающей cookie, компонент пути не обязан начинаться с `/products/`, но cookie будет возвращаться серверу только для таких страниц.

Пятый аргумент `setcookie()` определяет домен. Значение cookie отправляется серверу только для страниц с именем хоста, завершающимся заданным доменом. В следующем фрагменте в первом примере cookie отправляется всем хостам домена `example.com`, а во втором — только с запросами к хосту `jeannie.example.com`:

```
setcookie('flavor','chocolate chip',0,'','.example.com');
setcookie('flavor','chocolate chip',0,'','jeannie.example.com');
```

Если бы в первом примере был указан домен `example.com` вместо `.example.com`, то отправка выполнялась бы только для хоста `example.com` (но не `www.example.com` или `jeannie.example.com`). Если домен не указан при вызове `setcookie()`, то браузер отправляет cookie только с запросами с тем же именем хоста, что и для запроса, записавшего cookie.

В последнем из необязательных аргументов `setcookie()` передается флаг. Если он равен `true`, то браузер будет принимать cookie только по соединению SSL. Эта возможность может быть полезна, если cookie содержит конфиденциальную информацию, но следует помнить, что данные cookie хранятся на компьютере пользователя в виде незашифрованного текста.

Разные браузеры по-разному работают с cookie, особенно в отношении точности совпадения строк пути и домена, а также способов определения приоритета

между одноименными cookie. Полезная сводка этих различий представлена на странице `setcookie()` электронной документации.

См. также

Рецепт 8.2 — чтение значений cookie; Рецепт 8.3 — удаление cookie; Рецепт 8.13 — буферизация вывода; документация по функции `setcookie()`; расширенная спецификация cookie в RFC 2965.

8.2. Чтение cookie

Задача

Требуется прочитать значение cookie, записанное ранее.

Решение

Обратитесь к суперглобальному массиву `$_COOKIE`:

```
if (isset($_COOKIE['flavor'])) {
    print "You ate a {$_COOKIE['flavor']} cookie.";
}
```

Комментарий

Значение cookie недоступно в массиве `$_COOKIE` во время обработки запроса, в котором оно было записано. Другими словами, вызов функции `setcookie()` не изменяет значения `$_COOKIE`. Тем не менее при последующих запросах cookie, отправленные на сервер, хранятся в `$_COOKIE`.

Когда браузер отправляет cookie на сервер, отправляется только значение. Из `$_COOKIE` невозможно получить информацию о домене, пути, сроке действия или статусе защиты cookie, потому что браузер не отправляет эту информацию на сервер.

Чтобы вывести имена и значения всех cookie, отправленных конкретным запросом, переберите в цикле содержимое массива `$_COOKIE`:

```
foreach ($_COOKIE as $cookie_name => $cookie_value) {
    print "$cookie_name = $cookie_value <br/>";
}
```

См. также

Рецепт 8.1 — запись cookie; Рецепт 8.3 — удаление cookie.

8.3. Удаление cookie

Задача

Требуется удалить cookie, чтобы браузер не отправлял значение cookie серверу. Например, cookie используется для отслеживания информации о входе пользователя на сайт, и пользователь завершил сеанс работы на сайте.

Решение

Вызовите `setcookie()` без значения cookie и со сроком действия, завершившимся в прошлом:

```
setcookie('flavor','',1);
```

Комментарий

Срок действия рекомендуется задавать в далеком прошлом на случай рассинхронизации часов на сервере и компьютере пользователя. Например, если сервер считает, что сейчас 15:06, а на компьютере пользователя часы показывают 15:02, cookie со сроком действия до 15:05 не удаляется на компьютере пользователя, хотя на сервере это время уже прошло.

Вызов `setcookie()` для удаления cookie должен получать те же аргументы (кроме значения и времени), что и вызов `setcookie()` для записи cookie; не забудьте передать путь, домен и флаг защиты, если они используются.

См. также

Рецепт 8.1 — запись cookie; Рецепт 8.2 — чтение cookie; документация по функции `setcookie()`.

8.4. Построение строки запроса

Задача

Требуется сгенерировать строку запроса с заданными парами «имя/значение».

Решение

Воспользуйтесь функцией `http_build_query()`:

```
$vars = array('name' => 'Oscar the Grouch',
              'color' => 'green',
              'favorite_punctuation' => '#');
```

```
$query_string = http_build_query($vars);
$url = '/muppet/select.php?' . $query_string;
```

Комментарий

URL-адрес, построенный в Решении, выглядит так:

```
/muppet/select.php?name=Oscar+the+Grouch&color=green&favorite_punctuation=%23
```

Так как в URL-адресах и строках запросов не все символы считаются действительными, функция кодирует данные в подходящий формат. Например, в строке запроса пробелы представлены знаками `+`. Специальные символы обозначаются шестнадцатеричными кодами, например символ `#` имеет обозначение `%23`, потому что ASCII-код `#` равен 35, что соответствует 23 в шестнадцатеричной записи.

Хотя кодирование, выполняемое `http_build_query()`, предотвращает искажение генерируемого URL-адреса специальными символами в именах и значениях переменных, у вас могут возникнуть проблемы с именами переменных, начинающимися с имен сущностей HTML. Рассмотрим следующий частичный URL-адрес для получения информации о стереосистеме:

```
/stereo.php?speakers=12&cdplayer=52&amp=10
```

Символу & соответствует сущность HTML `&`; следовательно, браузер может интерпретировать этот URL-адрес как:

```
/stereo.php?speakers=12&cdplayer=52&=10
```

Есть три способа предотвращения искажения URL-адресов встроенными сущностями. Во-первых, можно выбирать имена переменных, которые невозможно спутать с сущностями, например `_amp` вместо `amp`. Во-вторых, можно преобразовать символы с эквивалентными сущностями HTML перед выводом URL-адреса. Для этого используется функция `htmlentities()`:

```
$url = '/muppet/select.php?' . htmlentities($query_string);
```

Полученный URL-адрес выглядит так:

```
/muppet/select.php?name=Oscar+the+Grouch&color=green&favorite_punctuation=%23
```

В-третьих, можно сменить разделитель аргументов `&` на `&`;, задав конфигурационной директиве `arg_separator.input` значение `&`. Далее функция `http_build_query()` будет объединять пары «имя/значение» с разделителем `&`:

```
ini_set('arg_separator.input', '&amp;');
```

См. также

Документация по функциям `http_build_query()` и `htmlentities()`.

8.5. Чтение тела запроса POST

Задача

Требуется напрямую обратиться к телу запроса (а не к разобранным данным, которые PHP помещает в `$_POST` для вас). Допустим, вы хотите обработать документ XML, который был отправлен как часть запроса веб-службы.

Решение

Прочитайте данные из потока `php://input`:

```
$body = file_get_contents('php://input');
```

Комментарий

Суперглобальный массив `$_POST` предназначен для работы с отправленными переменными форм HTML, но он не предоставляет низкоуровневого доступа ко всему телу запроса. На помощь приходит поток `php://input`. Прочитайте все данные вызовом `file_get_contents()`, или, если тело запроса ожидается достаточно большое, читайте его по частям с использованием `fread()`.

Если конфигурационная директива `always_populate_raw_post_data` включена, то низкоуровневые данные также помещаются в глобальную переменную `$HTTP_RAW_POST_DATA`. Тем не менее, чтобы ваш код был наиболее универсальным, используйте `php://input` — это решение будет работать даже в том случае, если режим `always_populate_raw_post_data` отключен.

См. также

Документация по `php://input` и `always_populate_raw_post_data`; глава 24 — средства чтения файлов.

8.6. Использование аутентификации HTTP

Задача

Требуется использовать PHP для защиты частей сайта паролями. Вместо того чтобы хранить пароли во внешнем файле и поручить аутентификацию веб-серверу, вы хотите реализовать логику проверки пароля в программе PHP.

Решение

Суперглобальные переменные `$_SERVER['PHP_AUTH_USER']` и `$_SERVER['PHP_AUTH_PW']` содержат имя пользователя и пароль (если они были предоставлены поль-

зователем). Чтобы запретить доступ к странице, отправьте заголовок `WWW-Authenticate`, идентифицирующий защищенную область как часть ответа с кодом статуса HTTP 401:

```
http_response_code(401);
header('WWW-Authenticate: Basic realm="My Website"');
echo "You need to enter a valid username and password.";
exit();
```

Комментарий

Получив заголовок 401, браузер отображает диалоговое окно для ввода имени пользователя и пароля. Эти учетные данные (имя пользователя и пароль), если они принимаются сервером, связываются с защищенной областью в заголовке `WWW-Authenticate`. Код, проверяющий учетные данные аутентификации, должен выполняться до отправки какого-либо вывода браузеру, потому что из него могут отправляться заголовки. Например, можно воспользоваться такой функцией, как `validate()` из листинга 8.1.

Листинг 8.1. validate()

```
function validate($user, $pass) {
    /* Заменить соответствующей проверкой имени пользователя
       и пароля - например, проверкой по базе данных */
    $users = array('david' => 'fadj&32',
                   'adam'  => '8HEj838');

    if (isset($users[$user]) && ($users[$user] === $pass)) {
        return true;
    } else {
        return false;
    }
}
```

В листинге 8.2 приведен пример использования `validate()`.

Листинг 8.2. Использование функции проверки

```
if (! validate($_SERVER['PHP_AUTH_USER'], $_SERVER['PHP_AUTH_PW'])) {
    http_response_code(401);
    header('WWW-Authenticate: Basic realm="My Website"');
    echo "You need to enter a valid username and password.";
    exit;
}
```

Содержимое функции `validate()` следует заменить логикой проверки правильности пароля. Также можно изменить строку защищенной области и текст сообщения, которое должно выводиться в случае нажатия кнопки отмены в окне аутентификации браузера.

Кроме базовой аутентификации, PHP также поддерживает *дайджест-аутентификацию*. При базовой аутентификации имена и пароли отправляются в неза-

щищенном виде по сети, с минимальной маскировкой в виде кодирования Base64. При дайджест-аутентификации сам пароль никогда не передается от браузера к серверу — отправляется только хеш-код пароля вместе с другими значениями. Тем самым сокращается вероятность перехвата сетевого трафика и его воспроизведения атакующей стороной. Повышенная безопасность, обеспечиваемая дайджест-аутентификацией, означает, что код реализации будет сложнее простого сравнения паролей. В листинге 8.3 представлены функции для реализации дайджест-аутентификации в соответствии с RFC 2617.

Листинг 8.3. Использование дайджест-аутентификации

```
/* Заменить соответствующей проверкой имени пользователя
   и пароля - например, проверкой по базе данных */
$users = array('david' => 'fadj&32',
               'adam'  => '8HEj838');
$realm = 'My website';

$username = validate_digest($realm, $users);

// При недействительных данных аутентификации управление
// никогда не достигнет этой точки.
print "Hello, " . htmlentities($username);

function validate_digest($realm, $users) {
    // Ошибка, если клиент не предоставил дайджест
    if (! isset($_SERVER['PHP_AUTH_DIGEST'])) {
        send_digest($realm);
    }
    // Ошибка, если дайджест не удается разобрать
    $username = parse_digest($_SERVER['PHP_AUTH_DIGEST'], $realm, $users);
    if ($username === false) {
        send_digest($realm);
    }
    // В дайджесте указано правильное имя пользователя
    return $username;
}

function send_digest($realm) {
    http_response_code(401);
    $nonce = md5(uniqid());
    $opaque = md5($realm);
    header("WWW-Authenticate: Digest realm=\"$realm\" qop=\"auth\" ".
           "nonce=\"$nonce\" opaque=\"$opaque\"");
    echo "You need to enter a valid username and password.";
    exit;
}

function parse_digest($digest, $realm, $users) {
    // Необходимо найти в заголовке дайджеста следующие значения:
    // username, uri, qop, cnonce, nc и response
    $digest_info = array();
    foreach (array('username','uri','nonce','cnonce','response') as $part) {
        // Ограничителем может быть символ ' или " или ничего (для qop и nc)
        if (preg_match('/'.$part.'=([\'"]?)(.*?)\1/', $digest, $match)) {
            // Если компонент найден, сохранить его для вычислений
```

```
            $digest_info[$part] = $match[2];
        } else {
            // Если компонент отсутствует, проверка дайджеста невозможна;
            return false;
        }
    }
    // Убедиться в правильности предоставленного значения qop
    if (preg_match('/qop=auth(,|$)/', $digest)) {
        $digest_info['qop'] = 'auth';
    } else {
        return false;
    }
    // Убедиться в правильности переданного временного значения
    if (preg_match('/nc=([0-9a-f]{8})(,|$)/', $digest, $match)) {
        $digest_info['nc'] = $match[1];
    } else {
        return false;
    }

    // Теперь убедиться в том, что из заголовка дайджеста были извлечены
    // все необходимые значения, и выполнить алгоритмические вычисления,
    // необходимые для проверки правильности информации.
    //
    // Эти вычисления описаны в разделах 3.2.2, 3.2.2.1
    // и 3.2.2.2 документа RFC 2617.
    // Алгоритм MD5
    $A1 = $digest_info['username'] . ':' . $realm . ':' .
        $users[$digest_info['username']];
    // qop содержит 'auth'
    $A2 = $_SERVER['REQUEST_METHOD'] . ':' . $digest_info['uri'];
    $request_digest = md5(implode(':', array(md5($A1), $digest_info['nonce'],
        $digest_info['nc'],
    $digest_info['cnonce'], $digest_info['qop'], md5($A2))));
    // Отправленные данные соответствуют вычисленным?
    if ($request_digest != $digest_info['response']) {
        return false;
    }
    // Все нормально, вернуть имя пользователя
    return $digest_info['username'];
}
```

Ни базовая аутентификация, ни дайджест-аутентификация HTTP не могут использоваться при выполнении PHP как программы CGI. Если вы не можете выполнять PHP в виде серверного модуля, используйте cookie-аутентификацию, рассматриваемую в Рецепте 8.7.

Другой недостаток аутентификации HTTP заключается в том, что она не предоставляет простых средств для завершения сеанса, кроме выхода из браузера. В электронной документации PHP приводятся некоторые рекомендации для организации завершения сеанса, работающие с разной степенью успеха для разных комбинацией веб-серверов и браузеров.

Однако существует прямолинейный способ принудительного завершения сеанса пользователя с истечением заданного интервала времени: включение времени

в строку защищенной области. Браузеры используют одну комбинацию имени пользователя и пароля каждый раз, когда у них запрашиваются учетные данные для одной защищенной области. Изменение имени защищенной области заставляет браузер запросить у пользователя новые учетные данные. В листинге 8.4 используется базовая аутентификация с принудительным завершением сеанса ежедневно в полночь.

Листинг 8.4. Принудительное завершение сеанса с базовой аутентификацией

```
if (! validate($_SERVER['PHP_AUTH_USER'],$_SERVER['PHP_AUTH_PW'])) {
    $realm = 'My Website for '.date('Y-m-d');
    http_response_code(401);
    header('WWW-Authenticate: Basic realm="'.$realm.'"');
    echo "You need to enter a valid username and password.";
    exit;
}
```

Также можно организовать тайм-аут уровня конкретного пользователя без изменения имени защищенной области; для этого следует сохранить время начала сеанса или обращения к защищенной странице. Функция `validate_date()` в листинге 8.5 сохраняет время начала сеанса в базе данных и принудительного завершения сеанса по истечении более 15 минут с момента последнего запроса защищенной страницы пользователем.

Листинг 8.5. validate_date()

```
function validate_date($user,$pass) {
    $db = new PDO('sqlite:/databases/users');

    // Подготовка и выполнение
    $st = $db->prepare('SELECT password, last_access
                        FROM users WHERE user LIKE ?');
    $st->execute(array($user));

    if ($ob = $st->fetchObject()) {
        if ($ob->password == $pass) {
            $now = time();
            if (($now - $ob->last_access) > (15 * 60)) {
                return false;
            } else {
                // Обновление времени последнего обращения
                $st2 = $db->prepare('UPDATE users SET last_access = "now"
-                                    WHERE user LIKE ?');

                $st2->execute(array($user));
                return true;
            }
        }
    }

    return false;
}
```

См. также

Рецепт 8.7; электронная документация PHP — раздел, посвященный аутентификации HTTP.

8.7. Аутентификация с использованием cookie

Задача

Требуется усилить контроль за процедурой начала сеанса, например вывести собственную форму для ввода учетных данных.

Решение

Сохраните статус аутентификации в cookie или как часть сеансовых данных. Если процедура подключения пользователя проходит успешно, сохраните его имя пользователя (или другое уникальное значение) в cookie. Также включите хеш имени пользователя и секретного слова, чтобы пользователь не мог просто подделать cookie аутентификации с именем:

```
$secret_word = 'if i ate spinach';
if (validate($_POST['username'],$_POST['password'])) {
    setcookie('login',
              $_POST['username'].','.md5($_POST['username'].$secret_word));
}
```

Комментарий

При использовании аутентификации с cookie приходится выводить собственную форму ввода учетных данных вроде той, что представлена в листинге 8.6.

Листинг 8.6. Пример формы ввода учетных данных аутентификации

```
<form method="POST" action="login.php">
Username: <input type="text" name="username"> <br>
Password: <input type="password" name="password"> <br>
<input type="submit" value="Log In">
</form>
```

Для проверки имени пользователя и пароля можно воспользоваться функцией `validate()` из листинга 8.1. Единственное различие заключается в том, что этой функции в качестве учетных данных передаются значения `$_POST['username']` и `$_POST['password']` вместо `$_SERVER['PHP_AUTH_USER']` и `$_SERVER['PHP_AUTH_PW']`. Если пароль проходит проверку, сервер отправляет cookie с именем пользователя, хеш-кодом имени пользователя и секретным словом. Хеш-код предотвращает фальсификацию посредством отправки cookie с именем пользователя.

После того как пользователь пройдет проверку, странице достаточно проверить действительность отправленного значения cookie для выполнения специальных операций для этого пользователя. Листинг 8.7 демонстрирует один из способов решения этой задачи.

Листинг 8.7. Проверка cookie

```
unset($username);
if (isset($_COOKIE['login'])) {
    list($c_username, $cookie_hash) = split(',', $_COOKIE['login']);
    if (md5($c_username.$secret_word) == $cookie_hash) {
        $username = $c_username;
    } else {
        print "You have sent a bad cookie.";
    }
}

if (isset($username)) {
    print "Welcome, $username.";
} else {
    print "Welcome, anonymous user.";
}
```

Если вы используете встроенную поддержку сеансов, добавьте имя пользователя и хеш в сеанс и избегайте отправки отдельных cookie. При подключении нового пользователя установите дополнительную переменную в сеансе вместо того, чтобы отправлять cookie, как показано в листинге 8.8.

Листинг 8.8. Хранение информации подключения в сеансе

```
if (validate($_POST['username'],$_POST['password'])) {
    $_SESSION['login'] =
        $_POST['username'].','.md5($_POST['username'].$secret_word);
}
```

Код проверки, приведенный в листинге 8.9, почти не изменился; просто на этот раз вместо `$_COOKIE` в нем используется `$_SESSION`.

Листинг 8.9. Проверка сеансовой информации

```
unset($username);
if (isset($_SESSION['login'])) {
    list($c_username,$cookie_hash) = explode(',',$_SESSION['login']);
    if (md5($c_username.$secret_word) == $cookie_hash) {
        $username = $c_username;
    } else {
        print "You have tampered with your session.";
    }
}
```

Применение аутентификации с использованием cookie или сеансовых данных вместо базовой аутентификации HTTP существенно упрощает завершение работы пользователя: достаточно удалить его cookie или переменную из данных сеанса. Другое преимущество хранения аутентификационной информации в се-

ансе заключается в том, что вы можете связать действия пользователей после подключения с их действиями до подключения или после завершения сеанса. С базовой аутентификацией HTTP невозможно связать запросы, ассоциированные с именем пользователя, с запросами, выданными тем же пользователем до ввода учетных данных. Поиск запросов с того же IP-адреса ненадежен, особенно если пользователь находится за брандмауэром или прокси-сервером. Если вы используете сеансовые данные, можно изменить процедуру подключения и сохранять информацию о связи идентификатора сеанса с именем пользователя с применением кода, приведенного в листинге 8.10.

Листинг 8.10. Сохранение информации о входе и выходе

```
if (validate($_POST['username'], $_POST['password'])) {
    $_SESSION['login'] =
        $_POST['username'].','.md5($_POST['username'].$secret_word);
    error_log('Session id '.session_id().' log in as '.$_POST['username']);
}
```

Листинг 8.10 записывает сообщение в журнал, но с таким же успехом он мог бы записывать информацию в базу данных, используемую при анализе трафика и использования сайта.

Одна из опасностей использования сеансовых идентификаторов заключается в том, что сеансы могут перехватываться. Если Алиса угадает идентификатор сеанса Боба, она сможет выдать себя за Боба перед веб-сервером. Сеансовый модуль имеет две необязательные конфигурационные директивы, которые затрудняют подбор идентификаторов сеанса. Директива `session.entropy_file` содержит путь к устройству или файлу, генерирующему элемент случайности, например `/dev/random` или `/dev/urandom`. Директива `session.entropy_length` содержит количество байтов, читаемых из файла энтропии при создании идентификатора сеанса.

Как бы вы ни затрудняли подбор идентификаторов сеансов, они могут быть похищены при пересылке в виде простого текста между сервером и браузером пользователя. Базовой аутентификации HTTP также присуща эта проблема. Используйте SSL для защиты от анализа сетевого трафика, как описано в Рецепте 18.13.

См. также

Рецепт 8.6; Рецепт 20.9 — ошибки ведения журнала; Рецепт 18.9 — проверка данных с использованием хеш-кодов; документация по функциям `setcookie()` и `md5()`.

8.8. Чтение заголовка HTTP

Задача

Требуется прочитать заголовок из запроса HTTP.

Решение

Если вам нужен одиночный заголовок, обратитесь к суперглобальному массиву `$_SERVER`:

```
// Заголовок User-Agent
echo $_SERVER['HTTP_USER_AGENT'];
```

Для получения всех заголовков вызовите функцию `getallheaders()`:

```
$headers = getallheaders();
echo $headers['User-Agent'];
```

Комментарий

Заголовки HTTP позволяют браузеру (или любому приложению) передать дополнительную информацию о запросе. Например, заголовок `Content-Type` описывает тело запроса (что было отправлено — веб-форма или данные JSON), `Accept-Language` содержит список предпочтительных языков (канадский английский или канадский французский?), а `User-Agent` определяет пользовательского агента (имя и описание браузера).

Иногда веб-сервер автоматически обрабатывает заголовки и действует соответствующим образом, особенно в том, что касается низкоуровневых подробностей запроса (например, предоставления данных из кэша или сжатия данных). В других случаях PHP разбирает нужные заголовки по мере надобности, как показано в Рецептах 8.2 и 8.6.

Но встречаются ситуации, в которых требуется прочитать конкретный заголовок из программного кода. Один из примеров — разбор заголовка `ETag` для проверки версии ресурса.

В таких случаях следует работать с суперглобальным массивом `$_SERVER`. PHP выделяет заголовки запросов HTTP в отдельное пространство имен, снабжая имя заголовка префиксом `HTTP_`. Кроме того, все имена заголовков переводятся в верхний регистр, чтобы упростить их нахождение. (Это нормально, потому что регистр символов в именах заголовков игнорируется.)

Итак, для заголовка `ETag`, если он отправлен, будет храниться в элементе `$_SERVER['HTTP_ETAG']`. Если модификация имен полей кажется вам эстетически неприемлемой, данные также можно получить вызовом `getallheaders()['Etag']`.

См. также

Рецепт 8.9 — запись заголовков HTTP.

8.9. Запись заголовка HTTP

Задача

Требуется записать заголовок HTTP.

Решение

Вызовите функцию `header()`:

```
// Графика в формате PNG
header('Content-Type: image/png');
```

Комментарий

Веб-сервер и PHP часто заполняют все необходимые заголовки правильными значениями. Например, при возвращении страницы HTML заголовок `Content-Length` или `Transfer-Encoding` задается автоматически, чтобы браузер мог определить размер ответа.

Функция `header()` позволяет явно задать эти значения, если они не могут быть вычислены сервером или если вы хотите изменить поведение по умолчанию.

Например, многие веб-серверы настроены на отправку заголовка `Content-Type` со значением `text/html` для всех страниц, обрабатываемых PHP. Чтобы использовать PHP для создания файла JSON, можно изменить содержимое `Content-Type` из самого сценария:

```
header('Content-Type: application/json');
```

Если записать один заголовок несколько раз, то отправлено будет только последнее значение. Чтобы изменить это поведение, передайте `true` во втором аргументе функции:

```
header('WWW-Authenticate: Basic realm="http://server.example.com/"');
header('WWW-Authenticate: OAuth realm="http://server.example.com/"', true);
```

Если вы поддерживаете несколько способов аутентификации, возвращение нескольких заголовков `WWW-Authenticate` вполне допустимо. В таком случае пользователь может использовать либо базовую аутентификацию HTTP, либо OAuth.

См. также

Документация по функции `header()`; Рецепт 8.8 — чтение заголовков HTTP.

8.10. Отправка конкретного кода статуса HTTP

Задача

Требуется явно задать код статуса HTTP. Например, вы хотите указать, что пользователю не разрешен просмотр страницы или что страница не найдена.

Решение

Воспользуйтесь функцией `http_response_code()` для назначения кода ответа:

```
http_response_code(401);
```

Комментарий

Ваш веб-сервер возвращает код статуса HTTP 200 (OK) для большинства страниц, обрабатываемых PHP. Однако существует множество других кодов статуса (или кодов ответа), которые вам также могут пригодиться.

Некоторые популярные коды заслуживают собственных рецептов. При перенаправлении на другую страницу следует отправить код статуса 302 (Found); эта тема рассматривается в Рецепте 8.11. Если пользователю не разрешено просматривать страницу, отправляется код 401 (Unauthorized). За дополнительной информацией по этой теме обращайтесь к Рецептам 8.6 и 8.7.

Также стоит отметить код 304 (Not Modified) для условных запросов GET, когда контент возвращается только в том случае, если он изменился с момента последнего запроса. Эта возможность обычно используется, когда пользователь выполняет периодический опрос вашего сайта и вы хотите ему сообщить, что никакой новой информации не появилось.

Также следует помнить о пользующемся дурной славой коде 404 (Not Found), который указывает на отсутствие страницы. Обычно он предоставляется веб-сервером. Но если вы хотите поддерживать динамические URL-адреса, для которых не существует никаких физических файлов на диске, а URL-адрес обрабатывается с выдачей ответа на основании информации из базы данных, то попытки обращения по недействительному URL-адресу придется обрабатывать самостоятельно.

Превосходным примером служит система WordPress, реагирующая на URL-адреса с включением категорий или дат (например, `/category/php/` или `/2014/11/03/`). При добавлении категории или публикации с новой датой WordPress можно настроить для автоматического ответа на запросы с URL-адресами, соответствующими этой схеме, хотя файлов с таким местоположением не существует.

При вызове функции `http_response_code()` предоставляется код статуса, а PHP берет на себя настройку строки `Status-Line`. Для некоторых кодов статуса, включая код 204 (No Content), в спецификации HTTP явно указано, что вы не должны предоставлять тело сообщения. В таких случаях лучше выполнить `exit()` для немедленного завершения сценария. Тем самым предотвращается случайное добавление контента в будущем:

```
http_response_code(204);
exit();
```

Если вы работаете с PHP 5.3, используйте `header()` и передайте код статуса в третьем параметре:

```
header('HTTP/1.0 204 No Content', true, 204);
```

См. также

Описание кодов статуса в спецификации HTTP 1.1.

8.11. Перенаправление запросов

Задача

Требуется автоматически направить пользователя на новый URL-адрес. Например, после успешного сохранения данных формы пользователь должен попасть на страницу, подтверждающую сохранение данных.

Решение

Перед выводом данных воспользуйтесь функцией `header()` для отправки заголовка `Location` с новым URL-адресом, после чего вызовите `exit()` для предотвращения дальнейшего вывода:

```
header('Location: http://www.example.com/confirm.html');
exit();
```

Комментарий

Чтобы передать переменные новой странице, включите их в строку запроса URL-адреса, как показано в листинге 8.11.

Листинг 8.11. Перенаправление с использованием переменных в строке запроса

```
header('Location: http://www.example.com/?monkey=turtle');
exit();
```

URL-адрес перенаправления должен включать протокол и имя хоста; одного пути недостаточно. В листинге 8.12 приведены примеры правильного и неправильного заголовка `Location`.

Листинг 8.12. Правильный и неправильный заголовки Location

```
// Правильное перенаправление
header('Location: http://www.example.com/catalog/food/pemmican.php');
// Неправильное перенаправление
header('Location: /catalog/food/pemmican.php');
```

Загрузка URL-адреса, на который перенаправляется пользователь, осуществляется методом GET. Перенаправление с загрузкой методом POST невозможно. Впрочем, в JavaScript можно имитировать перенаправление на базе POST — для этого следует сгенерировать форму, которая отправляется (методом POST) автоматически. Когда браузер (с включенной поддержкой JavaScript) получает страницу из листинга 8.13, он немедленно отправляет включенную форму методом POST.

Листинг 8.13. Перенаправление с использованием формы

```
<html>
  <body onload="document.getElementById('redirectForm').submit()">
    <form id='redirectForm' method='POST' action='/done.html'>
      <input type='hidden' name='status' value='complete'/>
      <input type='hidden' name='id' value='0u812'/>
      <input type='submit' value='Please Click Here To Continue'/>
    </form>
  </body>
</html>
```

Форма в листинге 8.13 имеет идентификатор `redirectForm`, поэтому код в атрибуте `onload` элемента `<body/>` отправляет форму. Действие `onload` не будет выполнено при отключенной поддержке JavaScript. В такой ситуации пользователь видит кнопку Please Click Here To Continue.

См. также

Документация по функции `header()`.

8.12. Принудительная отправка вывода браузеру

Задача

Требуется форсировать отправку вывода браузеру. Например, перед выполнением медленного запроса к базе данных вы хотите передать пользователю информацию об изменении текущего состояния.

Решение

Воспользуйтесь функцией `flush()`:

```
print 'Finding identical snowflakes...';
flush();
$sth = $dbh->query(
    'SELECT shape,COUNT(*) AS c FROM snowflakes GROUP BY shape HAVING c > 1');
```

Комментарий

Функция `flush()` отправляет вывод, накопленный во внутреннем буфере PHP, веб-серверу. Впрочем, веб-сервер может использовать собственную внутреннюю буферизацию, которая откладывает момент получения данных браузером. Кроме того, некоторые браузеры не отображают данные непосредственно при получении, а некоторые версии Internet Explorer не выводят страницу, пока не получат минимум 256 байт. Чтобы заставить IE вывести страницу, выведите пробелы в начале страницы, как показано в листинге 8.14.

Листинг 8.14. Принудительный вывод контента в IE

```
print str_repeat(' ',300);
print 'Finding identical snowflakes...';
flush();
$sth = $dbh->query(
    'SELECT shape,COUNT(*) AS c FROM snowflakes GROUP BY shape HAVING c > 1');
```

См. также

Рецепт 24.13; документация по функции `flush()`.

8.13. Буферизация вывода

Задача

Требуется начать генерировать вывод до того, как вы завершили отправку заголовков или cookie.

Решение

Вызовите функцию `ob_start()` в начале страницы и функцию `ob_end_flush()` в конце. После этого вы сможете чередовать команды, генерирующие вывод, с командами, отправляющими заголовки. Вывод не будет отправлен до вызова `ob_end_flush()`:

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

I haven't decided if I want to send a cookie yet.
```

```
<?php setcookie('heron','great blue'); ?>

Yes, sending that cookie was the right decision.

<?php
ob_end_flush();
```

Комментарий

Функции `ob_start()` может передаваться имя функции обратного вызова для обработки выходного буфера. Данная возможность может быть полезна для заключительной обработки всего контента страницы (например, сокрытия адресов электронной почты от роботов, собирающих адреса). Пример:

```
<?php
function mangle_email($s) {
    return preg_replace('/([^@\s]+)@([-a-z0-9]+\.)+[a-z]{2,}/is',
                        '<$1@...>',
                        $s);
}

ob_start('mangle_email');
?>

I would not like spam sent to ronald@example.com!

<?php
ob_end_flush();
```

Функция `mangle_email()` преобразует вывод к следующему виду:

```
I would not like spam sent to <ronald@...>!
```

Конфигурационная директива `output_buffering` включает буферизацию вывода для всех страниц:

```
output_buffering = On
```

Директива `output_handler` назначает функцию обратного вызова для обработки выходного буфера для всех страниц:

```
output_handler=mangle_email
```

Выполнение `output_handler` автоматически включает `output_buffering` в состояние `on`.

См. также

Документация по функциям `ob_start()`, `ob_end_flush()` и буферизации вывода.

8.14. Сжатие вывода

Задача

Требуется отправить сжатый контент браузерам, поддерживающим автоматическую распаковку.

Решение

Добавьте следующую настройку в файл php.ini:

```
zlib.output_compression=1
```

Комментарий

Браузер сообщает серверу, что он может принимать сжатые ответы, при помощи заголовка `Accept-Encoding`. Если браузер отправляет заголовок `Accept-Encoding: gzip` или `Accept-Encoding: deflate`, а установка PHP была построена с расширением zlib, то конфигурационная директива `zlib.output_compression` приказывает PHP сжимать выходные данные с применением соответствующего алгоритма перед отправкой их браузеру. Браузер распаковывает данные перед отображением.

Уровень сжатия можно отрегулировать конфигурационной директивой `zlib.output_compression_level`:

```
; минимальное сжатие
zlib.output_compression_level=1
; максимальное сжатие
zlib.output_compression_level=9
```

Более высокие уровни сжатия уменьшают объем данных, передаваемых от сервера к браузеру, но увеличивают затраты процессорного времени сервера на сжатие данных.

См. также

Документация по расширению zlib.

8.15. Чтение переменных окружения

Задача

Требуется получить значение переменной окружения.

Решение

Воспользуйтесь функцией `getenv()`:

```
$path = getenv('PATH');
```

Комментарий

Переменные окружения (environment variables) представляют собой набор именованных значений, связанных с процессом. Например, в Unix вызов `getenv('HOME')` возвращает домашний каталог пользователя:

```
print getenv('HOME'); // Домашний каталог пользователя
```

По умолчанию PHP автоматически загружает переменные окружения в `$_ENV`. Тем не менее директивы `php.ini-development` и `php.ini-production` отключают загрузку окружения по соображениям быстродействия.

Если вы часто обращаетесь к переменным окружения, включите загрузку массива `$_ENV`, добавив символ `E` в конфигурационную директиву `variables_order`. После этого вы сможете читать значения из суперглобального массива `$_ENV`. Пример:

```
$name = $_ENV['USER'];
```

Функция `getenv()` недоступна при выполнении PHP в виде модуля ISAPI.

См. также

Рецепт 8.16 — запись переменных окружения; документация по функции `getenv()`; информация о переменных окружения в PHP.

8.16. Запись переменных окружения

Задача

Требуется задать значение переменной окружения в сценарии или в конфигурации сервера. Настройка переменных окружения в конфигурации сервера или на уровне хоста позволяет устанавливать разную конфигурацию виртуальных хостов.

Решение

Чтобы задать переменную окружения в сценарии, воспользуйтесь функцией `putenv()`:

```
putenv('ORACLE_SID=ORACLE'); // Настройка расширения oci
```

Чтобы задать переменную окружения в файле Apache httpd.conf, используйте директиву SetEnv:

```
SetEnv DATABASE_PASSWORD password
```

Переменные, заданные в httpd.conf, присутствуют в суперглобальном массиве PHP $_SERVER, но не в результатах getenv() или $_ENV.

Комментарий

Преимущество настройки переменных окружения в файле httpd.conf заключается в том, что вы можете назначить более ограниченные разрешения чтения по сравнению со сценариями PHP. Так как файлы PHP должны быть доступны для чтения процессу веб-сервера, обычно это позволяет другим пользователям системы просматривать их. Хранение паролей в httpd.conf позволяет избежать размещения пароля в общедоступном файле. Кроме того, если вы используете несколько имен хостов, отображаемых на один корневой каталог документов, вы можете настроить сценарии так, чтобы их поведение различалось в зависимости от имени хоста.

Допустим, имеются имена хостов members.example.com и guests.example.com. Версия для зарегистрированных участников (members) требует аутентификации и предоставляет расширенный доступ. Гостевая версия (guests) предоставляет ограниченный набор возможностей, но не требует аутентификации. В листинге 8.15 показано, как может работать эта схема.

Листинг 8.15. Изменение поведения на основании значения переменной окружения

```
$version = (isset($_SERVER['SITE_VERSION']) ? $_SERVER['SITE_VERSION'] :
'guest');
// Если пользователь не ввел учетные данные,
// перенаправить на http://guest.example.com
if ('members' == $version) {
    if (!authenticate_user($_POST['username'], $_POST['password'])) {
        header('Location: http://guest.example.com/');
        exit;
    }
}
include_once "${version}_header"; // Загрузка специального заголовка
```

См. также

Рецепт 8.15 — получение значений переменных окружения; документация по функции putenv(); информация о настройке переменных окружения в Apache.

8.17. Взаимодействие с Apache

Задача

Требуется организовать передачу данных от PHP другим частям процесса обработки запросов Apache (в частности, записывать данные в `access_log`).

Решение

Воспользуйтесь функцией `apache_note()`:

```
// Прочитать значение
$session = apache_note('session');
// Записать значение
apache_note('session', $session);
```

Комментарий

В процессе обработки запроса от клиента Apache выполняет определенную последовательность действий; PHP является лишь одним из звеньев этой цепочки. Apache также осуществляет перераспределение URL, аутентификацию пользователей, регистрацию запросов в журнале и т. д. Во время обработки запроса каждый обработчик имеет доступ к набору пар «ключ/значение» — так называемой *таблице уведомлений* (notes table). Функция `apache_note()` предоставляет доступ к таблице уведомлений для получения информации, записанной обработчиками на более ранних стадиях процесса, и записи информации для обработчиков более поздних стадий.

Например, если вы используете сеансовый модуль для отслеживания пользователей и сохранения переменных между запросами, его можно интегрировать с анализом журналов для определения среднего количества просмотров страниц на одного пользователя. Используйте `apache_note()` в сочетании с журнальным модулем для записи идентификатора сеанса прямо в `access_log` для каждого запроса. Сначала добавьте идентификатор сеанса в таблицу уведомлений при помощи кода из листинга 8.16.

Листинг 8.16. Добавление идентификатора сеанса в таблицу уведомлений

```
// Получение идентификатора сеанса
// и включение его в таблицу уведомлений Apache
apache_note('session_id', session_id());
```

См. также

Документация по функции `apache_note()`; информация о ведении журналов Apache.

8.18. Перенаправление мобильных браузеров

Задача

Требуется перенаправить браузер для мобильного или планшетного устройства на альтернативный сайт или альтернативный контент, оптимизированный для устройства.

Решение

Используйте объект, возвращенный функцией `get_browser()`, для определения мобильного браузера:

```
if ($browser->ismobilebrowser) {
    // Вывод контента для мобильных устройств
} else {
    // Вывод контента для настольных устройств
}
```

Комментарий

Функция `get_browser()` проверяет переменную окружения (заданную веб-сервером) и сравнивает ее с браузерами, перечисленными во внешнем файле описания браузеров. Из-за проблем, связанных с лицензированием, файл описания браузеров не входит в поставку PHP. Одним из источников служит сайт Browscap (*http://browscap.org/*); загрузите с него файл php_browscap.ini.

После загрузки файла необходимо сообщить PHP его местонахождение; для этого следует указать в конфигурационной директиве `browscap` путь к файлу. Если вы используете PHP в режиме CGI, включите директиву в файл php.ini:

```
browscap=/usr/local/lib/php_browscap.ini
```

После того как устройство будет идентифицировано как мобильное, запрос перенаправляется на версию сайта или на страницу, оптимизированную для мобильных устройств:

```
header('Location: http://m.example.com/');
```

Также существует упрощенное решение — вместо вызова `get_browser()` вы можете разобрать `$_SERVER['HTTP_USER_AGENT']` самостоятельно.

См. также

Документация по функции `get_browser()`; Рецепт 8.11 — перенаправление запросов; Рецепт 8.8 — чтение заголовков HTTP.

8.19. Программа: (де)активизация учетных записей

Когда пользователь регистрируется на вашем сайте, будет полезно убедиться в том, что он ввел правильный адрес электронной почты. Чтобы проверить адрес, отправьте сообщение по указанному адресу. Если пользователь за несколько дней не посетит специальный URL-адрес, включенный в сообщение, его учетная запись деактивизируется.

Система состоит из трех частей. Первая часть — программа notify-user.php, которая отправляет сообщение новому пользователю и предлагает ему посетить проверочный URL-адрес, — приведена в листинге 8.18. Вторая часть (листинг 8.19) — страница verify-user.php — обрабатывает проверочный URL-адрес и помечает пользователей как прошедших проверку. Третья часть — программа delete-user.php — деактивизирует учетные записи пользователей, не посетивших проверочный URL-адрес за определенный промежуток времени. Эта программа приведена в листинге 8.20.

Код SQL в листинге 8.17 создает таблицу для хранения информации о пользователях.

Листинг 8.17. Код SQL для создания таблицы проверки пользователей

```
CREATE TABLE users (
 email VARCHAR(255) NOT NULL,
 created_on DATETIME NOT NULL,
 verify_string VARCHAR(16) NOT NULL,
 verified TINYINT UNSIGNED
);
```

В листинге 8.17 определяется минимальный объем информации, необходимой для проверки пользователей. Возможно, вы захотите хранить более подробные описания. При создании учетной записи пользователя сохраните информацию в таблице users и отправьте пользователю сообщение с предложением подтвердить регистрацию. Код в листинге 8.18 предполагает, что адрес электронной почты пользователя хранится в переменной `$email`.

Листинг 8.18. notify-user.php

```
// Подключение к базе данных
$db = new PDO('sqlite:users.db');

$email = 'david';

// Генерирование verify_string
$verify_string = '';
for ($i = 0; $i < 16; $i++) {
    $verify_string .= chr(mt_rand(32,126));
}
```

```
// Вставка записи пользователя в базу данных.
// При этом используется функция datetime(), специфическая для SQLite
$sth = $db->prepare("INSERT INTO users " .
                    "(email, created_on, verify_string, verified) " .
                    "VALUES (?, datetime('now'), ?, 0)");
$sth->execute(array($email, $verify_string));

$verify_string = urlencode($verify_string);
$safe_email = urlencode($email);

$verify_url = "http://www.example.com/verify-user.php";

$mail_body=<<<_MAIL_
To $email:

Please click on the following link to verify your account creation:

$verify_url?email=$safe_email&verify_string=$verify_string

If you do not verify your account in the next seven days, it will be
deleted.
_MAIL_;

mail($email,"User Verification",$mail_body);
```

Проверочная страница, на которую направляется пользователь по ссылке из сообщения, обновляет таблицу `users` при получении правильной информации (листинг 8.19).

Листинг 8.19. verify-user.php

```
// Подключение к базе данных
$db = new PDO('sqlite:users.db');

$sth = $db->prepare('UPDATE users SET verified = 1 WHERE email = ? '.
                    ' AND verify_string = ? AND verified = 0');

$res = $sth->execute(array($_GET['email'], $_GET['verify_string']));
var_dump($res, $sth->rowCount());
if (! $res) {
    print "Please try again later due to a database error.";
} else {
    if ($sth->rowCount() == 1) {
        print "Thank you, your account is verified.";
    } else {
        print "Sorry, you could not be verified.";
    }
}
```

Статус проверки пользователя обновляется только в том случае, если адрес электронной почты и проверочная строка соответствуют информации из записи в базе данных, которая еще не прошла проверку. Последняя часть (листинг 8.20) представляет собой небольшую программу для удаления непроверенных пользователей по истечении заданного интервала.

Листинг 8.20. delete-user.php

```
// Подключение к базе данных
$db = new PDO('sqlite:users.db');

$window = '-7 days';

$sth = $db->prepare("DELETE FROM users WHERE verified = 0 AND ".
                    "created_on < datetime('now',?)");
$res = $sth->execute(array($window));

if ($res) {
    print "Deactivated " . $sth->rowCount() . " users.\n";
} else {
    print "Can't delete users.\n";
}
```

Программа из листинга 8.20 ежедневно запускается для очистки таблицы от непроверенных пользователей. Если вы хотите изменить промежуток времени, в течение которого пользователи должны подтвердить регистрацию, измените переменную `$window` и обновите текст сообщения в соответствии с новым значением.

8.20. Программа: Tiny Wiki

Программа из листинга 8.21 объединяет разные концепции, представленные в этой главе, и реализует полнофункциональную систему вики — сайта, страницы которого могут редактироваться пользователями. Структура программы типична для простых PHP-программ такого типа. В первой части кода определяются различные параметры конфигурации. Затем следует секция `if/else`, которая решает, что нужно делать (вывести страницу, сохранить правку и т. д.), в зависимости от значений отправленной формы или переменных из URL. Оставшаяся часть программы состоит из функций, вызываемых из секции `if/else` — функции для вывода заголовка и завершителя страницы, загрузки сохраненного контента страницы и вывода формы редактирования страницы.

Программа Tiny Wiki использует внешнюю библиотеку PHP Markdown Мишеля Фортина (Michel Fortin) для преобразования удобного и компактного синтаксиса Markdown в HTML.

Листинг 8.21. Tiny Wiki

```
<?php
// Регистрация PSR-0-совместимого автозагрузчика классов
spl_autoload_register(function($class){
    require preg_replace('{\\\\|_(?!.*\\\\)}', DIRECTORY_SEPARATOR,
        trim($class, '\\')).'.php';
});

// Markdown используется для разметки текста.
```

```
// Библиотека доступна по адресу http://michelf.ca/projects/php-markdown/
use \Michelf\Markdown;

// Каталог, в котором будут храниться страницы вики.
// Проследите за тем, чтобы каталог был доступен для записи веб-серверу.
define('PAGEDIR', dirname(__FILE__) . '/pages');

// Получение имени страницы или использование значения по умолчанию.
$page = isset($_GET['page']) ? $_GET['page'] : 'Home';

// Выбор действия: отображение формы редактирования, сохранение
// формы редактирования или отображение страницы.

// Вывод запрошенной формы редактирования
if (isset($_GET['edit'])) {
    pageHeader($page);
    edit($page);
    pageFooter($page, false);
}
// Сохранение отправленной формы редактирования
else if (isset($_POST['edit'])) {
    file_put_contents(pageToFile($_POST['page']), $_POST['contents']);
    // Перенаправление на обычное представление
    // только что отредактированной страницы.
    header('Location: http://'.$_SERVER['HTTP_HOST'] . $_SERVER['SCRIPT_NAME'] .
           '?page='.urlencode($_POST['page']));
    exit();
}
// Отображение страницы
else {
    pageHeader($page);
    // Если страница существует, вывести ее и завершитель со ссылкой "Edit"
    if (is_readable(pageToFile($page))) {
        // Получить контент страницы из файла, в котором она хранится
        $text = file_get_contents(pageToFile($page));
        // Преобразовать синтаксис Markdown (с использованием
        // библиотеки Markdown, загруженной выше)
        $text = Markdown::defaultTransform($text);
        // Создать ссылку на другие страницы вики
        $text = wikiLinks($text);
        // Отображение страницы
        echo $text;
        // Вывод завершителя
        pageFooter($page, true);
    }
    // Если страница не существует, вывести форму редактирования
    // и завершитель без ссылки "Edit"
    else {
        edit($page, true);
        pageFooter($page, false);
    }
}

// Заголовок страницы - очень простой, только название
// и обычная разметка HTML
```

```
function pageheader($page) { ?>
<html>
<head>
<title>Wiki: <?php echo htmlentities($page) ?></title>
</head>
<body>
<h1><?php echo htmlentities($page) ?></h1>
<hr/>
<?php
}
// Завершитель страницы - временная метка последнего изменения,
// необязательная ссылка "Edit" и ссылка для возврата
// к главной странице вики
function pageFooter($page, $displayEditLink) {
    $timestamp = @filemtime(pageToFile($page));
    if ($timestamp) {
        $lastModified = strftime('%c', $timestamp);
    } else {
        $lastModified = 'Never';
    }
    if ($displayEditLink) {
        $editLink = ' - <a href="?page='.urlencode($page).'&edit=true">Edit</a>';
    } else {
        $editLink = '';
    }
?>
<hr/>
<em>Last Modified: <?php echo $lastModified ?></em>
<?php echo $editLink ?> - <a href="<?php echo $_SERVER['SCRIPT_NAME'] ?>">Home</
a>
</body>
</html>
<?php
}

// Отображение формы редактирования. Если страница уже существует,
// включить ее текущий контент в форму
function edit($page, $isNew = false) {
    if ($isNew) {
        $contents = '';
?>
<p><b>This page doesn't exist yet.</b> To create it, enter its contents below
and click the <b>Save</b> button.</p>
    <?php } else {
        $contents = file_get_contents(pageToFile($page));
    }
?>
<form method='post' action='<?php echo htmlentities($_SERVER['SCRIPT_NAME']) ?>'>
<input type='hidden' name='edit' value='true'/>
<input type='hidden' name='page' value='<?php echo htmlentities($page) ?>'/>
<textarea name='contents' rows='20' cols='60'>
<?php echo htmlentities($contents) ?></textarea>
<br/>
<input type='submit' value='Save'/>
</form>
```

```
<?php
}

// Генерирование имени файла по содержимому страницы. Вызов md5()
// предотвращает проблемы безопасности из-за нежелательных символов в $page
function pageToFile($page) {
    return PAGEDIR.'/'.md5($page);
}

// Преобразование текста вида [something] в странице в ссылку HTML
// на страницу вики "something"
function wikiLinks($page) {
    if (preg_match_all('/\[([^\]]+?)\]/', $page, $matches, PREG_SET_ORDER)) {
        foreach ($matches as $match) {
            $page = str_replace($match[0], '<a href="'.$_SERVER['SCRIPT_NAME'].
'?page='.urlencode($match[1]).'">'.htmlentities($match[1]).'</a>', $page);
        }
    }
    return $page;
}
```

См. также

Информация об установке и использовании пакетов, включая информацию о PSR-0, в Рецепте 27.3.

8.21. Программа: HTTP Range

Программа в листинге 8.22 реализует функциональность, которая позволяет клиенту запросить одну или несколько частей файла. Чаще всего данная возможность используется для возобновления прерванной загрузки файла (например, видеоролика, просмотр которого был прерван).

Обычно веб-сервер способен сделать это за вас. Он разбирает заголовок, загружает выбранные части файла и возвращает их браузеру (вместе с необходимыми данными HTTP).

Но если вы продаете мультимедийный контент (например, подкасты или музыку), скорее всего, вы не захотите предоставлять доступ к этим файлам. В противном случае файлы сможет загрузить любой пользователь, знающий URL-адрес, — а вам бы хотелось, чтобы файлы могли быть прочитаны только пользователями, которые эти файлы купили. И по этой причине вы не сможете ограничиться одним веб-сервером, придется использовать PHP.

Рецепт 17.11 показывает, как ограничить файл от прямых обращений, но он работает только с целыми файлами. Программа расширяет возможности этого простого примера и позволяет передавать только части файла, запрашиваемого браузером.

На первый взгляд задача не кажется особо сложной. Тем не менее в спецификации HTTP 1.1 появились некоторые возможности, повышающие сложность: множественные диапазоны (с другим синтаксисом ответов), смещения от конца файла (например, «только последние 1000 байт»), специальные коды статуса и заголовки для недействительных запросов.

Кроме перевода требований спецификации в программный код, эта программа демонстрирует методику чтения и отправки кодов статуса HTTP и заголовков. В нее также интегрированы решения из других рецептов, включая Рецепт 1.6.

Листинг 8.22. HTTP Range

```
// При желании добавьте здесь аутентификацию.

// Файл
$file = __DIR__ . '/numbers.txt';
$content_type = 'text/plain';

// Убедиться в том, что файл доступен для чтения,
// и получить его размер
if (($filelength = filesize($file)) === false) {
    error_log("Problem reading filesize of $file.");
}

// Разобрать заголовок для определения информации,
// необходимой для отправки ответа
if (isset($_SERVER['HTTP_RANGE'])) {
    // В ограничителях игнорируется регистр символов
    if (!preg_match('/bytes=\d*-\d*(,\d*-\d*)*$/i', $_SERVER['HTTP_RANGE'])) {
        error_log("Client requested invalid Range.");
        send_error($filelength);
        exit;
    }
    /*
    Спецификация: "Если клиент запрашивает несколько байтовых
    диапазонов в одном запросе, сервер ДОЛЖЕН вернуть их в порядке их
    следования в запросе."
    */
    $ranges = explode(',',
        substr($_SERVER['HTTP_RANGE'], 6)); // Все после bytes=
    $offsets = array();
    // Извлечение и проверка каждого смещения.
    // Остаются только смещения, прошедшие проверку
    foreach ($ranges as $range) {
        $offset = parse_offset($range, $filelength);
        if ($offset !== false) {
            $offsets[] = $offset;
        }
    }

    /*
    В зависимости от количества запрошенных действительных диапазонов
    ответ должен возвращаться в разных форматах.
    */
```

```
switch (count($offsets)) {
case 0:
    // Нет действительных диапазонов
    error_log("Client requested no valid ranges.");
    send_error($filelength);
    exit;
    break;
case 1:
    // Один диапазон, отправить стандартный ответ
    http_response_code(206); // Частичный контент
    list($start, $end) = $offsets[0];
    header("Content-Range: bytes $start-$end/$filelength");
    header("Content-Type: $content_type");
    // Установка переменных, обеспечивающих повторное использование
    // кода между этим и следующим случаем.
    // Примечание: диапазон 0-0 имеет длину 1 байт (границы включаются)
    $content_length = $end - $start + 1;
    $boundaries = array(0 => '', 1 => '');
    break;
default:
    // Несколько действительных диапазонов, отправить ответ
    // из нескольких частей
    http_response_code(206);  // Частичный контент
    $boundary = str_rand(32); // Строка для разделения частей

    /*
    Необходимо вычислить Content-Length для всего ответа,
    но загрузка полного ответа в строку может привести к большим
    затратам памяти, поэтому значение вычисляется по смещениям.
    Заодно в процессе вычислений будут вычислены границы.
    */
    $boundaries = array();
    $content_length = 0;

   foreach ($offsets as $offset) {
       list($start, $end) = $offset;
           // Используется для разбивки на секции
       $boundary_header =
           "\r\n" .
           "--$boundary\r\n" .
           "Content-Type: $content_type\r\n" .
           "Content-Range: bytes $start-$end/$filelength\r\n" .
           "\r\n";
       $content_length += strlen($boundary_header) + ($end - $start + 1);
       $boundaries[] = $boundary_header;
   }

    // Добавить закрывающую границу
    $boundary_header = "\r\n--$boundary--";
    $content_length += strlen($boundary_header);
    $boundaries[] = $boundary_header;
    // Отсечь лишнюю комбинацию \r\n в первой границе
    $boundaries[0] = substr($boundaries[0], 2);
    $content_length -= 2;
    // Заменить специальным составным значением Content-Type
```

```
        $content_type = "multipart/byteranges; boundary=$boundary";
    }
} else {
    // Отправить весь файл.
    // Настроить переменные так, как если бы данные были получены
    // из заголовка Range.
    $start = 0;
    $end = $filelength - 1;
    $offset = array($start, $end);
    $offsets = array($offset);
    $content_length = $filelength;
    $boundaries = array(0 => '', 1 => '');
}
// Сообщить, что будет передаваться.
header("Content-Type: $content_type");
header("Content-Length: $content_length");

// Передача данных
$handle = fopen($file, 'r');
if ($handle) {
    $offsets_count = count($offsets);
// Вывести каждый ограничитель и соответствующую часть файла
    for ($i = 0; $i < $offsets_count; $i++) {
        print $boundaries[$i];
        list($start, $end) = $offsets[$i];
        send_range($handle, $start, $end);
    }
    // Закрывающий ограничитель
    print $boundaries[$i];
    fclose($handle);
}

// Переход к соответствующей позиции в файле
// и вывод запрашиваемой части по фрагментам
function send_range($handle, $start, $end) {
    $line_length = 4096; // "Волшебное" значение
    if (fseek($handle, $start) === -1) {
        error_log("Error: fseek() fail.");
    }

    $left_to_read = $end - $start + 1;
    do {
        $length = min($line_length, $left_to_read);
        if (($buffer = fread($handle, $length)) !== false) {
            print $buffer;
        } else {
            error_log("Error: fread() fail.");
        }
    } while ($left_to_read -= $length);
}

// Отправка заголовка ошибки
function send_error($filelength) {
    http_response_code(416);
    header("Content-Range: bytes */$filelength"); // Необходимо для кода 416.
```

```
}

// Преобразование смещения в начальную и конечную позиции в файле,
// или возвращение false для недействительного значения
function parse_offset($range, $filelength) {
    /*
    Спецификация: "Значение first-byte-pos в byte-range-spec определяет
    смещение первого байта в диапазоне".
    Спецификация: "Значение last-byte-pos определяет смещение
    последнего байта в диапазоне; указанные позиции являются включающими".
    */
    list($start, $end) = explode('-', $range);

    /*
    Спецификация: "Значение suffix-byte-range-spec используется
    для определения суффикса тела сущности с длиной, задаваемой
    значением suffix-length".
    */
    if ($start === '') {
        if ($end === '' || $end === 0) {
            // Запрашивается диапазон "-" или "-0"
            return false;
        } else {
            /*
            Спецификация: "Если сущность короче заданного значения
            suffix-length, то используется все тело сущности".
            Спецификация: "Байтовые смещения начинаются с нуля".
            */
            $start = max(0, $filelength - $end);
            $end = $filelength - 1;
        }
    } else {
        /*
        Спецификация: "Если значение last-byte-pos отсутствует или оно
        больше либо равно текущей длине тела сущности, то значение last-byte-pos
        принимается равным текущей длине тела сущности в байтах,
        уменьшенной на единицу".
        */
        if ($end === '' || $end > $filelength - 1) {
            $end = $filelength - 1;
        }

        /*
        Спецификация: "Если значение last-byte-pos присутствует, то оно
        ДОЛЖНО быть больше либо равно first-byte-pos для этого диапазона
        byte-range-spec, или диапазон byte-range-spec считается
        синтаксически недействительным".
        Проверка также перехватывает случаи start > filelength
        */
        if ($start > $end) {
            return false;
        }
    }

    return array($start, $end);
```

```
}

// Генерирование случайной строки для ограничения секций в ответе
function str_rand($length = 32,
$characters = '0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ') {
    if (!is_int($length) || $length < 0) {
        return false;
    }
    $characters_length = strlen($characters) - 1;
    $string = '';

    for ($i = $length; $i > 0; $i--) {
        $string .= $characters[mt_rand(0, $characters_length)];
    }

    return $string;
}
```

Для простоты демонстрационный файл numbers.txt содержит следующий набор цифр:

```
01234567890123456789
```

Следующий пример показывает, как работает частичная передача при выдаче запросов из программы командной строки *curl* встроенному веб-серверу PHP. Ниже приведено полное содержимое файла с подробной версией вывода HTTP (без заголовков Range):

```
$ curl -v http://localhost:8000/range.php
* About to connect() to localhost port 8000 (#0)
*   Trying ::1...
* connected
* Connected to localhost (::1) port 8000 (#0)
> GET /range.php HTTP/1.1
> User-Agent: curl/7.24.0
> Host: localhost:8000
> Accept: */*
>
[Sun Aug 18 14:33:36 2013] ::1:59812 [200]: /range.php
< HTTP/1.1 200 OK
< Host: localhost:8000
< Connection: close
< X-Powered-By: PHP/5.4.9
< Content-Type: text/plain
< Content-Length: 10
<
* Closing connection #0
0123456789
```

Только первые пять байтов:

```
$ curl -v -H 'Range: bytes=0-4' http://localhost:8000/range.php
* About to connect() to localhost port 8000 (#0)
*   Trying ::1...
* connected
```

```
* Connected to localhost (::1) port 8000 (#0)
> GET /range.php HTTP/1.1
> User-Agent: curl/7.24.0
> Host: localhost:8000
> Accept: */*
> Range: bytes=0-4
>
[Sun Aug 18 14:30:52 2013] ::1:59798 [206]: /range.php
< HTTP/1.1 206 Partial Content
< Host: localhost:8000
< Connection: close
< X-Powered-By: PHP/5.4.9
< Content-Range: bytes 0-4/10
< Content-Type: text/plain
< Content-Length: 5
<
* Closing connection #0
01234
```

Обратите внимание: на этот раз используется код статуса 206 вместо 200, а заголовок HTTP `Content-Range` сообщает, какие байты были возвращены.

Последние пять байтов:

```
$ curl -v -H 'Range: bytes=-5' http://localhost:8000/range.php
* About to connect() to localhost port 8000 (#0)
*   Trying ::1...
* connected
* Connected to localhost (::1) port 8000 (#0)
> GET /range.php HTTP/1.1
> User-Agent: curl/7.24.0
> Host: localhost:8000
> Accept: */*
> Range: bytes=-5
>
[Sun Aug 18 14:30:33 2013] ::1:59796 [206]: /range.php
< HTTP/1.1 206 Partial Content
< Host: localhost:8000
< Connection: close
< X-Powered-By: PHP/5.4.9
< Content-Range: bytes 5-9/10
< Content-Type: text/plain
< Content-Length: 5
<
* Closing connection #0
56789
```

Первые пять и последние пять байтов:

```
$ curl -v -H 'Range: bytes=0-4,-5' http://localhost:8000/range.php
* About to connect() to localhost port 8000 (#0)
*   Trying ::1...
* connected
* Connected to localhost (::1) port 8000 (#0)
> GET /range.php HTTP/1.1
> User-Agent: curl/7.24.0
```

```
> Host: localhost:8000
> Accept: */*
> Range: bytes=0-4,-5
>
[Sun Aug 18 14:30:12 2013] ::1:59794 [206]: /range.php
< HTTP/1.1 206 Partial Content
< Host: localhost:8000
< Connection: close
< X-Powered-By: PHP/5.4.9
< Content-Type: multipart/byteranges; boundary=ALLIeNOkvwgKk0ib91ZNph5qi8fHo2ai
< Content-Length: 236
<
--ALLIeNOkvwgKk0ib91ZNph5qi8fHo2ai
Content-Type: text/plain
Content-Range: bytes 0-4/10

01234
--ALLIeNOkvwgKk0ib91ZNph5qi8fHo2ai
Content-Type: text/plain
Content-Range: bytes 5-9/10

56789
* Closing connection #0
--ALLIeNOkvwgKk0ib91ZNph5qi8fHo2ai--
```

Заголовок `Content-Type` изменяется со значения `text/plain` на `multipart/byteranges; boundary=ALLIeNOkvwgKk0ib91ZNph5qi8fHo2ai`. «Настоящие» заголовки `Content` переместились в соответствующие секции.

Так как речь идет о полном содержимом файла, оно может поставляться так, как если бы оно запрашивалось без заголовка `Range`.

Недействительный запрос (байты 20–24 не существуют):

```
$ curl -v -H 'Range: bytes=20-24' http://localhost:8000/range.php
* About to connect() to localhost port 8000 (#0)
*   Trying ::1...
* connected
* Connected to localhost (::1) port 8000 (#0)
> GET /range.php HTTP/1.1
> User-Agent: curl/7.24.0
> Host: localhost:8000
> Accept: */*
> Range: bytes=20-24
>
[Sun Aug 18 14:32:17 2013] Client requested no valid ranges.
[Sun Aug 18 14:32:17 2013] ::1:59806 [416]: /range.php
< HTTP/1.1 416 Requested Range Not Satisfiable
< Host: localhost:8000
< Connection: close
< X-Powered-By: PHP/5.4.9
< Content-Range: bytes */10
< Content-type: text/html
<
* Closing connection #0
```

На этот раз возвращается третий код статуса 416 с полезным заголовком, который сообщает допустимый диапазон значений для запроса: `Content-Range: bytes */10`.

Наконец, рассмотрим комбинацию допустимого и недопустимого значения:

```
$ curl -v -H 'Range: bytes=0-4,20-24' http://localhost:8000/range.php
* About to connect() to localhost port 8000 (#0)
*   Trying ::1...
* connected
* Connected to localhost (::1) port 8000 (#0)
> GET /range.php HTTP/1.1
> User-Agent: curl/7.24.0
> Host: localhost:8000
> Accept: */*
> Range: bytes=0-4,20-24
>
[Sun Aug 18 14:31:27 2013] ::1:59801 [206]: /range.php
< HTTP/1.1 206 Partial Content
< Host: localhost:8000
< Connection: close
< X-Powered-By: PHP/5.4.9
< Content-Range: bytes 0-4/10
< Content-Type: text/plain
< Content-Length: 5
<
* Closing connection #0
01234
```

Запрос включает хотя бы один действительный диапазон, поэтому недействительные диапазоны игнорируются, и ответ не отличается от того, который будет получен при запросе только первых пять байтов.

9 Формы

9.0. Введение

Одной из самых сильных сторон PHP является органичная интеграция переменных форм в программы. Благодаря ей веб-программирование становится простым и элегантным, а цикл перехода от веб-форм к коду PHP и выводу HTML существенно ускоряется.

Впрочем, вместе с удобствами приходит ответственность: вы должны проследить за тем, чтобы предоставленная пользователем информация, которая так легко перетекает в вашу программу, содержала правильные данные. Внешнему вводу нельзя доверять ни при каких условиях, поэтому очень важно проверять все вводимые данные. Рецепты 9.2–9.9 показывают, как проверять типичные виды информации, а также содержат общие рекомендации по проверке произвольных данных форм. В Рецепте 9.10 рассматривается экранирование сущностей HTML для безопасного отображения данных, введенных пользователем. Рецепт 9.11 объясняет, как обрабатывать файлы, отправленные пользователем.

HTTP относится к категории протоколов *без состояния* — в нем нет встроенного механизма, позволяющего сохранить информацию с одной страницы, чтобы обратиться к ней из других страниц. В Рецептах 9.12, 9.13 и 9.14 представлены обходные решения этой фундаментальной проблемы для определения того, какой пользователь выдал тот или иной запрос к веб-серверу.

При обработке страницы PHP проверяет URL-адрес и переменные формы, отправленные файлы, действующие cookie, переменные веб-сервера и окружения. Полученные данные напрямую доступны в следующих массивах: `$_GET`, `$_POST`, `$_FILES`, `$_COOKIE`, `$_SERVER` и `$_ENV`. В них хранятся соответственно все переменные, заданные в строке запроса, в теле запроса, отправкой файлов, cookie, веб-сервером и окружением, в котором работает веб-сервер. Также существует `$_REQUEST` — один большой массив, в котором содержатся значения из шести массивов.

При размещении элементов в $_REQUEST, если два массива содержат одноименные ключи, PHP разрешает конфликт в соответствии с конфигурационной директивой variables_order. По умолчанию variables_order использует порядок EGPCS (или GPCS, если вы используете конфигурационный файл php.ini-recommended). Итак, PHP сначала добавляет в $_REQUEST переменные окружения, а затем добавляет в массив переменные строки запроса, отправленные значения, данные cookie и переменные веб-сервера в указанном порядке. Например, так как C в порядке по умолчанию следует после P, cookie с именем username заменяет отправленную переменную с именем username. Следует заметить, что со значением GPCS из файла php.ini-recommended массив $_ENV не заполняется переменными окружения.

Хотя массив $_REQUEST может быть удобен, обычно удобнее работать со специализированными массивами. В этом случае вы точно знаете, что получаете, и вам не нужно беспокоиться о том, что изменение variables_order повлияет на поведение программы.

Все эти массивы являются *автоглобальными*. Этот термин обозначает глобальность внутри функции или класса — они всегда находятся в области видимости.

В версиях PHP до 5.4.0 поддерживалась конфигурационная директива с именем register_globals. Если она была установлена, то все эти переменные также становились доступны в виде переменных глобального пространства имен. Итак, значение $_GET['password'] также доступно под именем $password. При всем удобстве такой подход создает серьезные проблемы с безопасностью, потому что злоумышленник может легко задать переменные снаружи и заменить доверенные внутренние переменные. Если вы используете старую версию PHP, проследите за тем, чтобы эта директива была отключена.

В листинге 9.1 приведена простая форма, на которой пользователю предлагается ввести имя. При отправке формы информация передается hello.php.

Листинг 9.1. Простая форма HTML

```
<form action="hello.php" method="post">
<p>What is your first name?</p>
<input type="text" name="first_name" />
<input type="submit" value="Say Hello" />
</form>
```

Текстовому полю на форме присвоено имя first_name, а форма использует метод post. Это означает, что при отправке формы в $_POST['first_name'] будет храниться строка, введенная пользователем. (Конечно, элемент может быть и пустым, если пользователь ничего не ввел.)

В листинге 9.2 приведено содержимое программы hello.php, которая выводит информацию из формы.

Листинг 9.2. Простейшая обработка формы на PHP

```
echo 'Hello, ' . $_POST['first_name'] . '!';
```

Если ввести на форме из листинга 9.1 имя Twinkle, то листинг 9.2 выведет сообщение:

```
Hello, Twinkle!
```

Программа из листинга 9.2 предельно проста, и в ней пропущены два важных шага, которые должны присутствовать во всех приложениях обработки форм PHP: проверка данных (которая гарантирует, что данные, введенные на форме, являются допустимыми для вашей программы) и экранирование вывода (чтобы злоумышленники не могли использовать ваш сайт для атак). Проверка данных рассматривается в Рецептах 9.2–9.9, а экранирование вывода — в Рецепте 9.10.

9.1. Обработка ввода

Задача

Требуется использовать одну страницу HTML для отображения формы и последующей обработки данных, введенных на этой форме. Иначе говоря, вы хотите избежать создания лишних страниц для обработки последовательных этапов операции.

Решение

Используйте переменную `$_SERVER['REQUEST_METHOD']` для определения того, был ли запрос отправлен методом `get` или `post`. Если использовался метод `get`, выведите форму. Если использовался метод `post`, обработайте форму. В листинге 9.3 форма из листинга 9.1 объединяется с кодом из листинга 9.2 в одну программу, которая выбирает выполняемую операцию в зависимости от значения `$_SERVER['REQUEST_METHOD']`.

Листинг 9.3. Выбор операции в зависимости от метода запроса

```
<?php if ($_SERVER['REQUEST_METHOD'] == 'GET') { ?>
<form action="<?php echo htmlentities($_SERVER['SCRIPT_NAME']) ?>" method="post">
What is your first name?
<input type="text" name="first_name" />
<input type="submit" value="Say Hello" />
</form>
<?php } else {
    echo 'Hello, ' . $_POST['first_name'] . '!';
}
```

Комментарий

Работа с формами упрощается в том случае, если все части находятся в одном файле (или используются одним файлом), а отображаемые секции определяются контекстом. Метод `get` (который используется вашим браузером при вводе

URL-адреса или щелчке на ссылке), по сути, означает: «Эй, сервер, передай мне то, что у тебя есть». Метод `post` (используемый браузером при отправке формы, у которой атрибут `method` имеет значение `post`) означает: «Эй, сервер, вот тебе данные, которые должны что-то изменить». Таким образом, типичным ответом на запрос `get` является форма HTML, а типичной реакцией на запрос `post` — результаты обработки этой формы. В листинге 9.3 «обработка» ограничивается простым выводом приветствия. В более типичных приложениях выполняются более сложные действия, например сохранение информации в базе данных или отправка сообщений электронной почты.

Хотя спецификация XHTML требует, чтобы атрибут `method` элемента `<form/>` записывался в нижнем регистре (`get` или `post`), по спецификации HTTP браузер должен использовать символы верхнего регистра (`GET` или `POST`) при отправке серверу метода запроса. В `$_SERVER['REQUEST_METHOD']` хранится значение, отправленное браузером, поэтому на практике оно всегда содержит символы верхнего регистра.

Также для упрощения сопровождения страниц не стоит жестко кодировать путь к странице в атрибуте `action` формы; это не позволит вам переименовать или переместить страницу без ее редактирования. Вместо этого в качестве значения `action` используется переменная `$_SERVER['SCRIPT_NAME']`. PHP для каждого запроса присваивает этой переменной имя файла текущего сценария (относительно корневого каталога документов).

Если вы используете инфраструктуры для разработки веб-приложений, у них имеются собственные правила объединения отображения форм с обработкой результатов. Хотя в этой книге мы не будем ориентироваться ни на какую конкретную инфраструктуру, четкое разделение между кодом представления (отображения информации для пользователя) и «бизнес-логикой» (обработкой данных, предоставленных пользователем) упростит сопровождение кода и его понимание. Для форм сколько-нибудь сложнее листинга 9.3 выделение логики отображения формы в шаблон принесет заметную пользу. Существует множество отличных языков для работы с шаблонами, но для простоты в этой книге в качестве шаблонного языка будет использоваться PHP.

При такой переработке листинг 9.3 преобразуется в три файла: один отображает форму по запросу `get`, другой обрабатывает результаты по запросу `post`, а третий решает, какую операцию следует выполнить.

Код отображения формы выглядит так:

```
<form action="<?= htmlentities($_SERVER['SCRIPT_NAME']) ?>" method="post">
What is your first name?
<input type="text" name="first_name" />
<input type="submit" value="Say Hello" />
</form>
```

Логика обработки формы:

```
Hello, <?= $_POST['first_name'] ?> !
```

И наконец, логика принятия решения:

```
if ($_SERVER['REQUEST_METHOD'] == 'GET') {
    include __DIR__ . '/getpost-get.php';
}
else {
    include __DIR__ . '/getpost-post.php';
}
```

Логика принятия решений предполагает, что код отображения формы хранится в файле getpost-get.php, а код обработки формы — в файле getpost-post.php и что все три файла находятся в одном каталоге. Константа `__DIR__` сообщает программе, что искать следует в том же каталоге, в котором находится исполняемый код включаемых файлов.

Стратегия разбиения страниц на несколько файлов будет использоваться в других рецептах этой главы.

См. также

Рецепт 9.12 — обработка многостраничных форм.

9.2. Проверка ввода на форме: обязательные поля

Задача

Требуется убедиться в том, что для элемента формы было задано значение, например, что текстовое поле не осталось пустым.

Решение

Воспользуйтесь функцией `filter_has_var()` для проверки присутствия элемента в соответствующем входном массиве, как показано в листинге 9.4.

Листинг 9.4. Проверка обязательного поля

```
if (! filter_has_var(INPUT_POST, 'flavor')) {
   print 'You must enter your favorite ice cream flavor.';
}
```

Комментарий

Функция `filter_has_var()` проверяет входные данные, полученные PHP, перед внесением каких-либо модификаций в вашем коде. Последовательное использование различных функций-фильтров, рассматриваемых позднее в этой главе, обеспечивает необходимую проверку и защитную обработку данных, введенных

пользователем. Первый аргумент `filter_has_var()` указывает, где следует искать; значение `INPUT_POST` проверяет данные `POST` в теле запроса. Другие возможные значения — `INPUT_GET` (переменные из строки запроса), `INPUT_COOKIE` (cookie), `INPUT_SERVER` (серверная информация, которая сохраняется в `$_SERVER`) и `INPUT_ENV` (переменные окружения).

Разные типы элементов форм, которые не были заполнены пользователем, поразному ведут себя в данных `GET` и `POST`. Пустые текстовые поля, текстовые области и поля отправки файлов приводят к появлению элементов, значение которых представляет собой строку нулевой длины. Для флажков и переключателей, которые не были установлены, никакие элементы в данных `GET` или `POST` не создаются. Браузеры обычно заставляют пользователя выбрать один из вариантов в раскрывающихся меню, поддерживающих одиночный выбор, но раскрывающиеся меню с множественным выбором, в которых нет выбранных вариантов, работают аналогично флажкам — они не создают никаких элементов в данных `GET` или `POST`.

Что еще хуже, запросы могут поступать не только от браузеров. Ваши программы PHP могут получать запросы от других программ, любознательный хакер может построить запрос вручную, а злоумышленник может создавать запросы в попытке найти уязвимость в вашей системе. Чтобы ваш код был как можно более надежным, всегда проверяйте, что некоторый элемент существует в наборе входных данных, прежде чем применять к нему другие стратегии проверки. Кроме того, если стратегия проверки подразумевает, что элемент представляет собой массив значений (как в листинге 9.14), убедитесь в этом при помощи флага фильтрации `FILTER_REQUIRE_ARRAY`.

В листинге 9.5 функции `filter_has_var()`, `filter_input()` и `strlen()` используются для максимально жесткой проверки данных формы.

Листинг 9.5. Проверка данных формы

```
// Убедиться в том, что значение $_POST['flavor'] существует,
// прежде чем проверять его длину
if (! (filter_has_var(INPUT_POST, 'flavor') &&
       (strlen(filter_input(INPUT_POST, 'flavor')) > 0))) {
    print 'You must enter your favorite ice cream flavor.';
}

// Значение $_POST['color'] не является обязательным, но если оно
// задано, то после защитной обработки оно должно содержать
// более 5 символов
if (filter_has_var(INPUT_POST, 'color') &&
    (strlen(filter_input(INPUT_POST, 'color', FILTER_SANITIZE_STRING)) <= 5)) {
    print 'Color must be more than 5 characters.';
}
// Убедиться в том, что $_POST['choices'] существует
// и представляет собой массив
if (! (filter_has_var(INPUT_POST, 'choices') &&
       filter_input(INPUT_POST, 'choices', FILTER_DEFAULT,
                    FILTER_REQUIRE_ARRAY))) {
    print 'You must select some choices.';
}
```

Вызов `filter_input()` с двумя аргументами применяет фильтр по умолчанию, который не изменяет входные данные. В листинге 9.5 отправленное значение `flavor` никак не преобразуется. Фильтр `FILTER_SANITIZE_STRING`, примененный к отправленному значению `color`, удаляет теги HTML, удаляет двоичные символы, не входящие в набор ASCII, и кодирует амперсанды (`&`). Фильтр `FILTER_DEFAULT` применяется к `choices` для явного задания фильтра по умолчанию. Это существенно в последней части листинга 9.5, потому что флаг фильтра `FILTER_REQUIRE_ARRAY` должен передаваться в четвертом аргументе `filter_input()`.

Возможно, в минуту слабости вам захочется использовать `empty()` вместо `strlen()` для проверки того, что в текстовом поле было введено значение. Не поддавайтесь искушению; вы создадите себе проблемы, потому что строка из одного символа `0` интерпретируется как `false` по правилам логических вычислений PHP. Это может привести к ошибкам в проверке форм, если, например, кто-нибудь введет 0 в текстовом поле children, в результате чего элемент `$_POST['children']` будет содержать 0. Тогда вызов `empty($_POST['children'])` вернет `true` — что с точки зрения проверки формы свидетельствует об ошибке.

См. также

Документация по функциям `filter_has_var()` и `filter_input()`, список фильтров защитной обработки, список флагов фильтров; Рецепт 9.5 — информация о проверке раскрывающихся меню, Рецепт 9.6 — информация о проверке переключателей, Рецепт 9.7 — информация о проверке флажков.

9.3. Проверка ввода на форме: числа

Задача

Требуется убедиться в том, что в поле на форме было введено число. Например, вы хотите проверить, что в поле «Возраст пользователя» введено конкретное значение (13, 56 и т. д.), а не текст «В расцвете лет» или что-нибудь в этом роде.

Решение

Если вас интересуют целые числа, воспользуйтесь фильтром `FILTER_VALIDATE_INT`, как показано в листинге 9.6.

Листинг 9.6. Проверка чисел с фильтром FILTER_VALIDATE_INT

```
$age = filter_input(INPUT_POST, 'age', FILTER_VALIDATE_INT);
if ($age === false) {
   print "Submitted age is invalid.";
}
```

Если вас интересуют дробные числа, используйте фильтр FILTER_VALIDATE_FLOAT, как показано в листинге 9.7.

Листинг 9.7. Проверка чисел с фильтром FILTER_VALIDATE_FLOAT

```
$price = filter_input(INPUT_POST, 'price', FILTER_VALIDATE_FLOAT);
if ($price === false) {
   print "Submitted price is invalid.";
}
```

Комментарий

С фильтрами FILTER_VALIDATE_INT и FILTER_VALIDATE_FLOAT функция filter_input() возвращает число указанного типа (целое или с плавающей точкой), если входная строка представляет число, соответствующее фильтру, или false в противном случае.

Существует несколько флагов, управляющих поведением числовых фильтров. Флаг FILTER_FLAG_ALLOW_OCTAL приказывает FILTER_VALIDATE_INT поддерживать восьмеричную запись. Другими словами, для отправленной строки 017 возвращается целое число 15. С похожим флагом FILTER_FLAG_ALLOW_HEX отправленная строка 0x2f возвращается в виде целого числа 47.

Флаг FILTER_FLAG_ALLOW_THOUSAND изменяет поведение фильтра FILTER_VALIDATE_FLOAT, разрешая использование запятых для разделения групп разрядов. Без него строка 5,252 будет сочтена недействительной, а с ним она правильно проходит проверку как число с плавающей точкой 5252.

В некоторых видах проверки могут использоваться регулярные выражения. В листинге 9.8 приведены регулярные выражения для проверки целого и дробного числа.

Листинг 9.8. Проверка чисел по регулярному выражению

```
// Шаблон совпадает с необязательным знаком -,
// за которым следует минимум одна цифра
if (! preg_match('/^-?\d+$/',$_POST['rating'])) {
    print 'Your rating must be an integer.';
}

// Шаблон совпадает с необязательным знаком -, за которым следует
// необязательная последовательность цифр (целая часть),
// затем необязательная точка и как минимум одна цифра.
if (! preg_match('/^-?\d*\.?\d+$/',$_POST['temperature'])) {
   print 'Your temperature must be a number.';
}
```

Многие фанатичные сторонники оптимизации считают, что регулярных выражений следует избегать, потому что они работают относительно медленно. Однако в данном случае простые регулярные выражения практически не уступают функциям фильтров по эффективности. Если вы увереннее чувствуете себя

с регулярными выражениями или используете их в других контекстах, этот вариант может быть удобен. Регулярное выражение также позволяет рассматривать действительные числа (например, 782364.238723123), которые не могут храниться в формате с плавающей точкой PHP без потери точности. Например, такая возможность может пригодиться для хранения значений широты и долготы, которые вы планируете хранить в строковом виде.

См. также

Рецепт 9.2 — информация о проверке обязательных полей; список фильтров проверки; список флагов фильтров.

9.4. Проверка ввода на форме: адреса электронной почты

Задача

Требуется проверить адрес электронной почты, введенный пользователем.

Решение

Используйте фильтр `FILTER_VALIDATE_EMAIL`, как показано в листинге 9.9. Этот фильтр проверяет действительность адреса электронной почты по правилам RFC 5321 (в основном).

Листинг 9.9. Проверка адреса электронной почты

```
$email = filter_input(INPUT_POST, 'email', FILTER_VALIDATE_EMAIL);
if ($email === false) {
    print "Submitted email address is invalid.";
}
```

Комментарий

RFC 5321 объединяет несколько документов RFC, относящихся к работе с электронной почтой, и определяет стандарты действительных адресов электронной почты. Фильтр `FILTER_VALIDATE_EMAIL` использует регулярное выражение, базирующееся на этих правилах, хотя он и не поддерживает комментарии или свертку пробелов.

Фильтр проверяет только синтаксическую правильность конкретного адреса. Он поможет предотвратить ввод адреса вида *bingolover2261@example* вместо *bingolover2261@example.com*, но ничего не скажет о том, что произойдет при отправке сообщения по этому адресу. Более того, фильтр не скажет, действительно

ли адрес находится под контролем человека, предоставившего адрес. Для проверок такого рода необходимо отправить на указанный адрес сообщение для подтверждения. В нем пользователю предлагается выполнить некую операцию (получив это сообщение, щелкните на ссылке), которая подтвердит, что именно этот человек ввел адрес на форме. Или же в сообщении может быть сказано, что делать пользователю, если он не вводил этот адрес на форме, например щелкнуть на ссылке в сообщении, чтобы уведомить о вводе ошибочного адреса. В Рецепте 8.19 продемонстрирована система с отправкой сообщения со ссылкой, на которой пользователь должен щелкнуть для подтверждения адреса.

См. также

RFC 5321

9.5. Проверка ввода на форме: раскрывающиеся меню

Задача

Требуется проверить, что в раскрывающемся меню, сгенерированном элементом HTML `<select/>`, выбран действительный вариант.

Решение

Воспользуйтесь массивом значений для построения меню. Затем проверьте выбор пользователя, убедившись в том, что значение присутствует в массиве. В листинге 9.10 для проверки используется функция `in_array()`.

Листинг 9.10. Проверка раскрывающегося меню функцией in_array()

```
// Построение меню
$choices = array('Eggs','Toast','Coffee');
echo "<select name='food'>\n";
foreach ($choices as $choice) {
   echo "<option>$choice</option>\n";
}
echo "</select>";

// Последующая проверка выбранного варианта
if (! in_array($_POST['food'], $choices)) {
    echo "You must select a valid choice.";
}
```

Меню, сгенерированное в листинге 9.10, выглядит так:

```
<select name='food'>
<option>Eggs</option>
<option>Toast</option>
```

```
<option>Coffee</option>
</select>

<select name='food'>
<option>Eggs</option>
<option>Toast</option>
<option>Coffee</option>
</select>
```

При работе с меню, которое задает атрибуты `value` для каждого элемента `<option/>`, используйте функцию `array_key_exists()` для проверки ввода, как показано в листинге 9.11.

Листинг 9.11. Проверка раскрывающегося меню функцией array_key_exists()

```
// Построение меню
$choices = array('eggs' => 'Eggs Benedict',
                 'toast' => 'Buttered Toast with Jam',
                 'coffee' => 'Piping Hot Coffee');
echo "<select name='food'>\n";
foreach ($choices as $key => $choice) {
   echo "<option value='$key'>$choice</option>\n";
}
echo "</select>";

// Последующая проверка выбранного варианта
if (! array_key_exists($_POST['food'], $choices)) {
    echo "You must select a valid choice.";
}
```

Меню, сгенерированное в листинге 9.11, выглядит так:

```
<select name='food'>
<option value='eggs'>Eggs Benedict</option>
<option value='toast'>Buttered Toast with Jam</option>
<option value='coffee'>Piping Hot Coffee</option>
</select>
```

Комментарий

Методы в листингах 9.10 и 9.11 генерируют немного различающиеся меню. Листинг 9.10 использует массив `$choices` с автоматической нумерацией ключей и выводит элементы `<option/>`. Листинг 9.11 использует массив `$choices` с явно заданными ключами и выводит элементы `<option/>` с атрибутами `value`, которые определяются ключами.

В любом случае стратегия проверки остается неизменной: мы убеждаемся в том, что значение, отправленное для элемента формы, входит в число допустимых вариантов. Для запросов, отправленных «правильными» браузерами, это правило никогда не нарушается — обычно браузеры не позволяют создать фиктивный выбор для раскрывающегося меню. Однако следует помнить, что запросы к программам PHP не всегда поступают от «правильных» браузеров. Они с таким же успехом могут поступить от браузера, содержащего программную ошибку, или

от 11-летнего умника со спецификацией HTTP и telnet-клиентом командной строки. Так как всегда приходится учитывать возможность поступления вредоносных, вручную построенных запросов HTTP, важно проверять ввод даже в том случае, когда большинство пользователей никогда не столкнется с этой ошибкой.

См. также

Документация по функциям `in_array()` и `array_key_exists()`.

9.6. Проверка ввода на форме: переключатели

Задача

Требуется убедиться в том, что в группе переключателей (radio buttons) выбран действительный переключатель.

Решение

Воспользуйтесь массивом значений для построения набора переключателей. Затем проверьте выбор пользователя, убедившись в том, что отправленное значение присутствует в массиве. В листинге 9.12 для проверки используется функция `array_key_exists()`.

Листинг 9.12. Проверка переключателя

```
// Построение набора переключателей
$choices = array('eggs' => 'Eggs Benedict',
                 'toast' => 'Buttered Toast with Jam',
                 'coffee' => 'Piping Hot Coffee');
foreach ($choices as $key => $choice) {
   echo "<input type='radio' name='food' value='$key'/> $choice \n";
}

// Последующая проверка отправленного переключателя
if (! array_key_exists($_POST['food'], $choices)) {
    echo "You must select a valid choice.";
}
```

Комментарий

Проверка переключателя в листинге 9.12 очень похожа на проверку раскрывающихся меню из листинга 9.11. В обоих случаях используется одна и та же схема — мы определяем данные, описывающие варианты, генерируем разметку HTML, а потом используем данные, определенные на первом шаге, для проверки отправленного значения. Различается в основном генерируемый код HTML.

Среди различий между раскрывающимися меню и переключателями можно выделить обработку значений по умолчанию. Если в HTML вариант по умолчанию для раскрывающегося меню не указан явно, то используется первый вариант из меню. Но если в HTML не указан вариант по умолчанию для набора переключателей, то никакой вариант не используется по умолчанию.

Чтобы гарантировать, что один из переключателей будет выбран в «правильном» браузере, укажите вариант по умолчанию при помощи атрибута `checked="checked"`. В следующем коде по умолчанию используется вариант `toast`:

```
// По умолчанию
$defaults['food'] = 'toast';
// Построение набора переключателей
$choices = array('eggs' => 'Eggs Benedict',
                 'toast' => 'Buttered Toast with Jam',
                 'coffee' => 'Piping Hot Coffee');
foreach ($choices as $key => $choice) {
    echo "<input type='radio' name='food' value='$key'";
    if ($key == $defaults['food']) {
        echo ' checked="checked"';
    }
    echo "/> $choice \n";
}

// Последующая проверка отправленного переключателя
if (! array_key_exists($_POST['food'], $choices)) {
    echo "You must select a valid choice.";
}
```

Кроме того, для защиты от отсутствующих значений во вредоносных запросах, построенных вручную, используйте функцию `filter_has_var()` для проверки того, что для переключателя было отправлено значение (см. Рецепт 9.2).

См. также

Рецепт 9.2 — информация о проверке обязательных полей; документация по функции `array_key_exists()`.

9.7. Проверка ввода на форме: флажки

Задача

Требуется убедиться в том, что в отправленных данных установлены только действительные флажки.

Решение

Если вы работаете с одним флажком, проверьте, что значение отправлено и оно является действительным. Если значение флажка отсутствует, значит, флажок

не установлен. Листинг 9.13 проверяет, был ли флажок установлен, снят или значение в отправленных данных недействительно.

Листинг 9.13. Проверка одного флажка

```
// Построение флажка
$value = 'yes';
echo "<input type='checkbox' name='subscribe' value='yes'/> Subscribe?";

// Последующая проверка флажка
if (filter_has_var(INPUT_POST, 'subscribe')) {
    // Значение отправлено, и оно действительно
    if ($_POST['subscribe'] == $value) {
        $subscribed = true;
    } else {
        // Значение отправлено, но оно недействительно
        $subscribed = false;
        print 'Invalid checkbox value submitted.';
    }
} else {
    // Значение не отправлено
    $subscribed = false;
}
if ($subscribed) {
    print 'You are subscribed.';
} else {
    print 'You are not subscribed';
}
```

Флажки, образующие группу, генерируются на основании массива значений. Затем вызов `array_intersect()` проверяет, что набор отправленных значений содержится в наборе допустимых значений, как показано в листинге 9.14.

Листинг 9.14. Проверка группы флажков

```
// Построение группы флажков
$choices = array('eggs' => 'Eggs Benedict',
                 'toast' => 'Buttered Toast with Jam',
                 'coffee' => 'Piping Hot Coffee');
foreach ($choices as $key => $choice) {
   echo "<input type='checkbox' name='food[]' value='$key'/> $choice \n";
}

// Последующая проверка отправленных флажков
if (array_intersect($_POST['food'], array_keys($choices)) != $_POST['food']) {
    echo "You must select only valid choices.";
}
```

Комментарий

Чтобы код PHP правильно обрабатывал значения нескольких выделенных флажков, атрибут `name` флажков должен завершаться парой квадратных скобок `[]`, как описано в Рецепте 9.17. Эти множественные значения форматируются в `$_POST`

как массив. Так как для флажков в листинге 9.14 используется имя `food[]`, элемент `$_POST['food']` содержит массив значений установленных флажков.

Функция `array_intersect()` находит все элементы `$_POST['food']`, которые также присутствуют в `array_keys($choices)`. Иначе говоря, она фильтрует отправленные варианты (`$_POST['food']`), пропуская только приемлемые значения — ключи из массива `$choices`. Если все значения из `$_POST['food']` являются приемлемыми, то результат `array_intersect($_POST['food'], array_keys($choices))` представляет собой немодифицированную копию `$_POST['food']`. Итак, если результат не равен `$_POST['food']`, значит, были отправлены недействительные значения.

Для флажков действуют те же проблемы со значениями по умолчанию, что и для переключателей. Как и при работе с переключателями, используйте правила из Рецепта 9.2 и убедитесь в наличии отправленных данных, прежде чем переходить к дальнейшей проверке.

См. также

Рецепт 9.2 — информация о проверке обязательных полей; документация по функции `array_intersect()`.

9.8. Проверка ввода на форме: дата и время

Задача

Требуется проверить введенную пользователем дату или время. Например, вы хотите убедиться в том, что пользователь не пытается запланировать встречу на 45-е августа или срок действия его кредитной карты еще не истек.

Решение

Если месяц, день и год вводятся на форме в разных элементах, передайте эти значения функции `checkdate()`, как показано в листинге 9.15. Функция сообщит, является ли действительной данная комбинация месяца, дня и года.

Листинг 9.15. Проверка даты

```
if (! checkdate($_POST['month'], $_POST['day'], $_POST['year'])) {
   print "The date you entered doesn't exist!";
}
```

Чтобы проверить, что введенная дата предшествует или следует после некоторой пороговой даты, преобразуйте введенные значения во временную метку, вычислите временную метку для пороговой даты и сравните два значения. Листинг 9.16 проверяет, что до конца срока действия кредитной карты осталось еще достаточно времени.

Листинг 9.16. Проверка срока действия кредитной карты

```
// Начало месяца, в котором заканчивается срок действия
// кредитной карты
$expires = mktime(0, 0, 0, $_POST['month'], 1, $_POST['year']);
// Начало предыдущего месяца
$lastMonth = strtotime('last month', $expires);
if (time() > $lastMonth) {
   print "Sorry, that credit card expires too soon.";
}
```

Комментарий

Функция `checkdate()` удобна прежде всего тем, что она учитывает високосные годы и количество дней в каждом месяце, избавляя вас от утомительных сравнений каждого компонента даты. Для диапазонных проверок (проверки того, что дата или время наступает до, после или между двумя другими датами или моментами времени) проще всего работать с временными метками.

См. также

Глава 3 — нюансы обработки даты и времени.

9.9. Проверка ввода на форме: кредитные карты

Задача

Требуется убедиться в том, что пользователь не ввел вымышленный номер кредитной карты.

Решение

Функция `is_valid_credit_card()`, приведенная в листинге 9.17, проверяет номер кредитной карты на правильность синтаксиса.

Листинг 9.17. Проверка номера кредитной карты

```
function is_valid_credit_card($s) {
    // Удаление нецифровых символов и перестановка в обратном порядке
    $s = strrev(preg_replace('/[^\d]/','',$s));
    // Вычисление контрольной суммы
    $sum = 0;
    for ($i = 0, $j = strlen($s); $i < $j; $i++) {
        // Четные цифры используются без изменений
        if (($i % 2) == 0) {
            $val = $s[$i];
        } else {
            // Нечетные цифры удваиваются с вычитанием 9,
```

```
            //  если результат больше 9
            $val = $s[$i] * 2;
            if ($val > 9) { $val -= 9; }
        }
        $sum += $val;
    }
    // Число действительно, если сумма кратна 10
    return (($sum % 10) == 0);
}
if (! is_valid_credit_card($_POST['credit_card'])) {
    print 'Sorry, that card number is invalid.';
}
```

Комментарий

Для защиты от случайных ошибок при вводе номера кредитной карты используется *алгоритм Луна*. Этот алгоритм, реализованный функцией `is_valid_credit_card()` из листинга 9.17, выполняет вычисления с отдельными цифрами номера для проверки правильности числа.

Процесс проверки кредитной карты отчасти напоминает проверку адреса электронной почты. Синтаксическая проверка (то есть проверка того, что переданная последовательность символов соответствует стандарту) реализуется относительно легко. С семантической проверкой дело обстоит сложнее. Номер кредитной карты 4111 1111 1111 1111 проходит проверку функции из листинга 9.17, но действительным он не является. Это хорошо известное тестовое число, сходное с номером карты Visa (поэтому оно часто используется в книгах в качестве примера).

Как говорилось ранее, проверка адреса электронной почты требует внешней проверки — обычно посредством отправки на адрес сообщения, содержащего ссылку для подтверждения. Внешняя проверка номера кредитной карты основана на обращении к платежной системе с информацией о владельце (имя, адрес) и получении подтверждения.

Синтаксическая проверка хорошо защищает от случайных опечаток при вводе, но разумеется, для проверки номеров кредитных карт ее недостаточно.

См. также

Рецепт 9.4 — информация о проверке адресов электронной почты; описание алгоритма Луна.

9.10. Предотвращение межсайтовых сценарных атак

Задача

Требуется организовать безопасный вывод данных, введенных пользователем, на странице HTML. Например, вы хотите разрешить пользователю добавлять

комментарии к сообщениям в блоге, не беспокоясь о том, что содержащийся в комментарии код JavaScript или разметка HTML создаст проблемы.

Решение

Обработайте пользовательский ввод функцией `htmlentities()`, прежде чем отображать его на странице, как показано в листинге 9.18.

Листинг 9.18. Экранирование сущностей HTML

```
print 'The comment was: ';
print htmlentities($_POST['comment']);
```

Комментарий

PHP содержит пару функций для экранирования сущностей HTML. Простейшая функция `htmlspecialchars()` экранирует четыре символа: < > " и &. В зависимости от необязательных параметров она также может выполнять преобразования по принципу «вместо или в дополнение». Для более сложного кодирования используется функция `htmlentities()`; она кодирует любые символы, имеющие эквивалентные сущности HTML. В листинге 9.19 представлена функция `htmlspecialchars()` в действии.

Листинг 9.19. Экранирование сущностей HTML

```
$html = "<a href='fletch.html'>Stew's favorite movie.</a>\n";
print htmlspecialchars($html);                // Двойные кавычки
print htmlspecialchars($html, ENT_QUOTES);    // Одиночные и двойные кавычки
print htmlspecialchars($html, ENT_NOQUOTES);  // Ни то, ни другое
```

Код листинга 9.19 выводит следующий результат:

```
&lt;a href=&quot;fletch.html&quot;&gt;Stew's favorite movie.&lt;/a&gt;
&lt;a href=&quot;fletch.html&quot;&gt;Stew&#039;s favorite movie.&lt;/a&gt;
&lt;a href="fletch.html"&gt;Stew's favorite movie.&lt;/a&gt;
```

По умолчанию `htmlentities()` и `htmlspecialchars()` используют набор символов UTF-8 (для PHP 5.4.0; до этого по умолчанию использовалась кодировка ISO-8859-1). Чтобы использовать другой набор символов, передайте его в третьем аргументе. Например, чтобы использовать BIG5, используйте вызов `htmlentities($string, ENT_QUOTES, "BIG5")`.

См. также

Рецепты 18.4 и 19.12; документация по функциям `htmlentities()` и `htmlspecialchars()`.

9.11. Обработка отправленных файлов

Задача

Требуется обработать файл, отправленный пользователем. Например, вы строите сайт для размещения фотографий и хотите реализовать сохранение пользовательских фотографий.

Решение

Воспользуйтесь массивом `$_FILES` для получения информации об отправленных файлах. В листинге 9.20 отправленный файл сохраняется в каталоге /tmp вебсервера.

Листинг 9.20. Отправка файла

```
<?php if ($_SERVER['REQUEST_METHOD'] == 'GET') { ?>
<form method="post" action="<?php echo htmlentities($_SERVER['SCRIPT_NAME']) ?>"
      enctype="multipart/form-data">
<input type="file" name="document"/>
<input type="submit" value="Send File"/>
</form>
<?php } else {
    if (isset($_FILES['document']) &&
    ($_FILES['document']['error'] == UPLOAD_ERR_OK)) {
        $newPath = '/tmp/' . basename($_FILES['document']['name']);
        if (move_uploaded_file($_FILES['document']['tmp_name'], $newPath)) {
            print "File saved in $newPath";
        } else {
            print "Couldn't move file to $newPath";
        }
    } else {
        print "No valid file uploaded.";
    }
}
```

Комментарий

Информация об отправленных файлах хранится в суперглобальном массиве `$_FILES`. Для каждого элемента `file` на форме в `$_FILES` создается массив, ключом которого является имя элемента `file`. Например, форма в листинге 9.20 содержит элемент `file` с именем `document`, поэтому `$_FILES['document']` содержит информацию об отправленном файле. Каждый массив, соответствующий одному файлу, содержит пять элементов:

- `name` — имя отправленного файла; заполняется браузером и может содержать как полное имя вместе с путем, так и простое имя файла.
- `type` — MIME-тип содержимого файла, предоставленный браузером.

- `size` — размер файла в байтах, вычисленный сервером.
- `tmp_name` — каталог временного хранения файла на сервере.
- `error` — код ошибки, описывающей проблемы с отправкой файла (если они были).

Допустимые значения элемента `error`:

- `UPLOAD_ERR_OK` (0) — отправка прошла успешно (без ошибок).
- `UPLOAD_ERR_INI_SIZE` (1) — размер отправленного файла превышает значение конфигурационной директивы `upload_max_filesize`.
- `UPLOAD_ERR_FORM_SIZE` (2) — размер отправленного файла превышает значение элемента `MAX_FILE_SIZE` формы.
- `UPLOAD_ERR_PARTIAL` (3) — отправлена только часть файла.
- `UPLOAD_ERR_NO_FILE` (4) — файл отсутствует.
- `UPLOAD_ERR_NO_TMP_DIR` (6) — отправка завершилась неудачей из-за отсутствия временного каталога для хранения файла.
- `UPLOAD_ERR_CANT_WRITE` (7) — PHP не может записать файл на диск.
- `UPLOAD_ERR_EXTENSION` (8) — отправка остановлена расширением PHP.

Функция `is_uploaded_file()` подтверждает, что файл, который вы собираетесь обработать, был отправлен пользователем. Всегда проверяйте значение `tmp_name` перед тем, как обрабатывать отправленный файл наряду с другими файлами. Тем самым вы защититесь от действий злоумышленников, которые попытаются заставить ваш код обработать системный файл как отправленный пользователем.

Файл также можно переместить в каталог для постоянного хранения; используйте функцию `move_uploaded_file()`, как показано в листинге 9.20. Также следует проверить, что перемещаемый файл действительно является отправленным файлом. Обратите внимание: в `tmp_name` хранится полный путь к файлу, а не только базовое имя. При необходимости используйте функцию `basename()` для отсечения начального пути.

Обязательно проследите за тем, что PHP обладает разрешениями чтения и записи для каталога, в котором хранятся временные файлы (задается конфигурационной директивой `upload_tmp_dir`), и каталога, в который вы пытаетесь скопировать файл. PHP часто выполняется под специальным именем пользователя (например, `nobody` или `apache`) вместо персонального имени пользователя.

Обработка файлов может быть нетривиальной задачей, потому что не все браузеры отправляют одинаковую информацию. Важно сделать все правильно, иначе вы рискуете создать проблемы с безопасностью. В конце концов, вы разрешаете постороннему пользователю отправить на ваш компьютер любой файл на его усмотрение; для злоумышленника это может открыть возможность взлома или нарушения работоспособности компьютера.

По этой причине в PHP предусмотрены механизмы, позволяющие установить ограничения на отправку файлов, включая возможность полного запрета отправки файлов. Итак, если у вас возникают проблемы с обработкой отправленных файлов, проверьте, что файл не отклоняется из-за возможных проблем с безопасностью.

Для выполнения такой проверки убедитесь в том, что в конфигурационном файле директиве `file_uploads` присвоено значение `On`. Затем проверьте, что размер файла не превышает значение `upload_max_filesize`; по умолчанию порог составляет 2 Мбайт, чтобы злоумышленник не мог вызвать сбой на компьютере, заполнив весь жесткий диск огромным файлом. Также существует директива `post_max_size`, управляющая максимальным размером всех отправляемых данных в одном запросе; начальное значение этого параметра составляет 8 Мбайт.

С точки зрения различий между браузерами и ошибками пользователей, если вы не находите в `$_FILES` ожидаемых данных, убедитесь в том, что в открывающий тег формы был включен атрибут `enctype="multipart/form-data"`. Это необходимо PHP для правильной обработки файлов.

Кроме того, если файл для отправки не выбран, PHP присваивает `tmp_name` пустую строку. Чтобы убедиться в том, что файл был отправлен и он не пуст (хотя в некоторых обстоятельствах пустые файлы могут быть именно тем, что требуется), проверьте, что значение `tmp_name` определено, а размер `size` больше 0. Наконец, не все браузеры обязательно отправляют для файла один и тот же тип MIME; отправляемый тип зависит от того, что они знают о разных типах файлов.

См. также

Документация по отправке файлов и функции `basename()`.

9.12. Работа с многостраничными формами

Задача

Требуется использовать форму, которая состоит из нескольких страниц и сохраняет данные между страницами. Например, форма может использоваться для проведения опроса, вопросы которого не помещаются на одной странице.

Решение

Используйте механизм отслеживания сеанса для сохранения информации формы между этапами, а также переменную для отслеживания отображаемого этапа. Листинг 9.21 отображает четыре файла на форме, состоящей из двух страниц, и выводит собранные данные.

Листинг 9.21. Создание многостраничной формы

```
Логика принятия решений (stage.php):
// Включение сеансов
session_start();

// Определение используемого этапа
if (($_SERVER['REQUEST_METHOD'] == 'GET') || (! isset($_POST['stage']))) {
    $stage = 1;
} else {
    $stage = (int) $_POST['stage'];
}

// Проверяем, что номер этапа не слишком мал и не слишком велик
$stage = max($stage, 1);
$stage = min($stage, 3);

// Сохранение отправленных данных
if ($stage > 1) {
    foreach ($_POST as $key => $value) {
        $_SESSION[$key] = $value;
    }
}

include __DIR__ . "/stage-$stage.php";
```

Первая страница формы (stage-1.php):

```
<form action='<?= htmlentities($_SERVER['SCRIPT_NAME']) ?>' method='post'>

Name: <input type='text' name='name'/> <br/>
Age:  <input type='text' name='age'/> <br/>

<input type='hidden' name='stage' value='<?= $stage + 1 ?>'/>
<input type='submit' value='Next'/>
</form>
```

Вторая страница формы (stage-2.php):

```
<form action='<?= htmlentities($_SERVER['SCRIPT_NAME']) ?>' method='post'>

Favorite Color: <input type='text' name='color'/> <br/>
Favorite Food:  <input type='text' name='food'/> <br/>

<input type='hidden' name='stage' value='<?= $stage + 1 ?>'/>
<input type='submit' value='Done'/>
```

Страница вывода результатов (stage-3.php):

```
Hello <?= htmlentities($_SESSION['name']) ?>.
You are <?= htmlentities($_SESSION['age']) ?> years old.
Your favorite color is <?= htmlentities($_SESSION['color']) ?>
and your favorite food is <?= htmlentities($_SESSION['food']) ?>.
```

Комментарий

В начале каждого этапа в листинге 9.21 все отправленные переменные формы копируются в `$_SESSION`. Тем самым обеспечивается их доступность при после-

дующих запросах, в том числе и при коде этапа 3, который выводит все сохраненные данные.

Сеансовые данные PHP идеально подходят для таких задач, поскольку все данные сеанса хранятся на сервере. В результате запросы остаются небольшими (не нужно заново отправлять все, что было введено на предыдущем этапе), а проверка выполняется с большей эффективностью. Каждый блок данных проверяется только при отправке.

См. также

Рецепт 11.1 — информация о работе с сеансовыми данными.

9.13. Повторное отображение форм со встроенными сообщениями об ошибках

Задача

Приложение обнаруживает проблемы в данных, введенных на форме. Требуется вывести сообщения об ошибках рядом с проблемными полями вместо обобщенного сообщения об ошибке в верхней части формы. Также требуется сохранить значения, введенные пользователем на форме, чтобы пользователю не пришлось заполнять ее заново.

Решение

В процессе проверки храните информацию об ошибках формы в массиве, ключами которого являются имена элементов. Затем, когда наступает время отображения формы, выведите соответствующее сообщение об ошибке рядом с каждым элементом. Для сохранения данных, введенных пользователем, используется соответствующая идиома HTML: атрибут `value` (с кодированием сущностей) для большинства элементов `<input/>`, атрибут `checked='checked'` для переключателей и флажков и атрибут `selected='selected'` для элементов `<option/>` в раскрывающихся меню. Листинг 9.22 отображает форму с текстовым полем, флажком и раскрывающимся меню и проверяет введенные данные.

Листинг 9.22. Повторное отображение формы с сообщениями об ошибках и введенными данными

Основная логика и функция проверки:

```
// Тестовые данные для раскрывающегося меню
$flavors = array('Vanilla','Chocolate','Rhinoceros');

// Настройка пустых значений по умолчанию при отсутствии выбранного варианта
```

```
$defaults = array('name' => '',
                  'age' => '',
                  'flavor' => array());
foreach ($flavors as $flavor) {
    $defaults['flavor'][$flavor] = '';
}

if ($_SERVER['REQUEST_METHOD'] == 'GET') {
    $errors = array();
    include __DIR__ . '/show-form.php';
} else {
    // Запрос относится к типу POST, проверить данные на форме
    $errors = validate_form();
    if (count($errors)) {
        // Если обнаружены ошибки, повторно отобразить форму
        // с сообщениями об ошибках, сохранив значения по умолчанию
        if (isset($_POST['name'])) { $defaults['name'] = $_POST['name']; }
        if (isset($_POST['age'])) { $defaults['age'] = "checked='checked'"; }
        foreach ($flavors as $flavor) {
            if (isset($_POST['flavor']) && ($_POST['flavor'] == $flavor)) {
                $defaults['flavor'][$flavor] = "selected='selected'";
            }
        }
        include __DIR__ . '/show-form.php';
    } else {
        // Данные формы проверены, вывести сообщение для пользователя.
        // Вероятно, в реальном приложении будет выполнено
        // перенаправление или включен другой файл для отображения.
        print 'The form is submitted!';
    }
}

function validate_form() {
    global $flavors;

    // Изначально ошибки отсутствуют
    $errors = array();

    // Имя обязательно для заполнения, должно содержать не менее трех символов
    if (! (isset($_POST['name']) && (strlen($_POST['name']) > 3))) {
        $errors['name'] = 'Enter a name of at least 3 letters';
    }
    if (isset($_POST['age']) && ($_POST['age'] != '1')) {
        $errors['age'] = 'Invalid age checkbox value.';
    }
    // Выбор из меню не обязателен, но если значение отправлено,
    // оно должно присутствовать в $flavors
    if (isset($_POST['flavor']) && (! in_array($_POST['flavor'], $flavors))) {
        $errors['flavor'] = 'Choose a valid flavor.';
    }

    return $errors;
}
```

Форма (show-form.php):

```
<form action='<?= htmlentities($_SERVER['SCRIPT_NAME']) ?>' method='post'>
<dl>
```

```
<dt>Your Name:</dt>
<?php if (isset($errors['name'])) { ?>
  <dd class="error"><?= htmlentities($errors['name']) ?></dd>
<?php } ?>
<dd><input type='text' name='name'
     value='<?= htmlentities($defaults['name']) ?>'/></dd>
<dt>Are you over 18 years old?</dt>
<?php if (isset($errors['age'])) { ?>
  <dd class="error"><?= htmlentities($errors['age']) ?></dd>
<?php } ?>
<dd><input type='checkbox' name='age' value='1'
        <?= $defaults['age'] ?>/> Yes</dd>
<dt>Your favorite ice cream flavor:</dt>
<?php if (isset($errors['flavor'])) { ?>
  <dd class="error"><?= htmlentities($errors['flavor']) ?></dd>
<?php } ?>
<dd><select name='flavor'>
<?php foreach ($flavors as $flavor) { ?>
<option <?= isset($defaults['flavor'][$flavor]) ?
            $defaults['flavor'][$flavor] :
            "" ?>><?= htmlentities($flavor) ?></option>
<?php } ?>
</select></dd>
</dl>
<input type='submit' value='Send Info'/>
</form>
```

Комментарий

Если форма была отправлена с недействительными данными, пользователю будет намного приятнее получить ту же форму с сообщениями об ошибках в соответствующих местах вместо безликого сообщения об ошибке в верхней части формы. Функция `validate_form()` в листинге 9.22 строит массив сообщений об ошибках, которые используются кодом отображения формы для вывода сообщений в нужных местах.

Следующим логичным шагом станет расширение `validate_form()` с обработкой различных видов проверки данных и включением в show-form.php разметки HTML, чтобы на форме присутствовали необходимые элементы ввода.

См. также

Рецепты 9.2–9.9 — разные стратегии проверки ввода на форме.

9.14. Защита от повторной отправки одной формы

Задача

Требуется предотвратить повторную отправку одной формы.

Решение

Включите в форму скрытое поле с уникальным значением. При проверке данных проверьте, отправлялась ли ранее форма с этим значением, и если отправлялась — отклоните отправку данных. Если же форма еще не отправлялась, обработайте данные и запишите значение для использования в будущем. Также можно использовать JavaScript для блокировки кнопки Submit после отправки формы.

В листинге 9.23 функции `uniqid()` и `md5()` используются для включения в форму уникального поля. Кроме того, в обработчик `onsubmit` формы включается небольшой фрагмент кода JavaScript, который блокирует кнопку Submit при отправке.

Листинг 9.23. Включение уникального идентификатора в форму

```
<form method="post" action="<?php echo $_SERVER['SCRIPT_NAME'] ?>"
      onsubmit="document.getElementById('submit-button').disabled = true;">
<!-- ...Здесь вставляются обычные элементы формы... -->
<input type='hidden' name='token' value='<?= md5(uniqid()) ?>'/>
<input type='submit' value='Save Data' id='submit-button'/>
</form>
```

Чтобы узнать, не отправлялась ли форма ранее, листинг 9.24 сравнивает отправленный маркер со значением, хранящимся в базе данных SQLite.

Листинг 9.24. Проверка формы на повторную отправку

```
if ($_SERVER['REQUEST_METHOD'] == 'POST') {
    $db = new PDO('sqlite:/tmp/formjs.db');
    $db->beginTransaction();
    $sth = $db->prepare('SELECT * FROM forms WHERE token = ?');
    $sth->execute(array($_POST['token']));
    if (count($sth->fetchAll())) {
        print "This form has already been submitted!";
        $db->rollBack();
    } else {
        /* Здесь размещается код проверки остальных данных формы -
         * проверьте все, прежде чем записывать маркер */
        $sth = $db->prepare('INSERT INTO forms (token) VALUES (?)');
        $sth->execute(array($_POST['token']));
        $db->commit();
        print "The form is submitted successfully.";
    }
}
```

Комментарий

Пользователи нередко отправляют форму повторно. Это происходит по разным причинам, чаще всего из-за случайного повторного нажатия кнопки отправки. Может, они нажали кнопку возврата в своем браузере, чтобы проверить или отредактировать информацию, а потом снова нажали кнопку отправки вместо кнопки Вперед. А может, повторная отправка была намеренной: пользователь пытается лишний раз проголосовать в интернет-опросе или в розыгрыше призов.

Наше Решение предотвращает случайные ошибки и затрудняет работу злоумышленников, которые сознательно выполняют повторную отправку. Тем не менее мошенничество не исключается полностью: для этого придется добавить на форму контрольное изображение (captcha) или контрольный вопрос.

Решение предотвращает загромождение базы данных лишними копиями одной записи. Генерирование маркера, размещаемого на форме, позволяет однозначно идентифицировать этот конкретный экземпляр формы, даже при запрете cookie. Функция `uniqid()` генерирует приемлемый одноразовый маркер. Функция `md5()` не вносит дополнительный случайный элемент, но ограничивает символы, которые в нем могут встречаться. Результат `uniqid()` может представлять собой комбинацию из букв и других символов; результат `md5()` состоит только из цифр и букв `abcdef`.

Возможно, у вас возникнет соблазнительная идея обойтись без генерирования случайного маркера и вместо него использовать число, на 1 большее количества записей, уже находящихся в базе данных. У такого подхода есть (как минимум) два недостатка. Во-первых, он создает потенциальную «ситуацию гонки» (race condition). Что произойдет, если другой пользователь начнет заполнять форму до того, как первый пользователь заполнит ее? Вторая форма будет содержать тот же маркер, что и первая, и произойдет конфликт. Проблему можно обойти созданием новой пустой записи в базе данных при запросе формы, чтобы второму пользователю досталось число, на 1 большее первого. С другой стороны, это приведет к появлению пустых записей в базе данных, если пользователь передумает заполнять форму.

Во-вторых, такое решение позволяет тривиально отредактировать другую запись в базе данных, вручную заменив идентификатор другим числом. В зависимости от параметров безопасности фиктивный запрос `get` или `post` позволяет изменить данные без особых сложностей. С другой стороны, случайный маркер невозможно угадать простым переходом на другое число.

См. также

Рецепт 18.9 — дополнительная информация об использовании хеш-кодов для проверки данных; документация по функциям `uniqid()` и `md5()`. Простая реализация контрольных изображений доступна от Google (*http://www.google.com/recaptcha/*).

9.15. Предотвращение внедрения глобальных переменных

Задача

Вы используете старую версию PHP и хотите обратиться к переменным ввода данных на форме — но так, чтобы злоумышленники не могли задавать произвольные значения глобальных переменных в вашей программе.

Решение

Самое простое решение — использовать PHP версии 5.4.0 и выше. Начиная с этой версии, конфигурационная директива `register_globals` — источник проблемы внедрения глобальных переменных — была устранена.

Если вы работаете с более ранней версией PHP, отключите конфигурационную директиву `register_globals` и работайте с переменными только из массивов `$_GET`, `$_POST` и `$_COOKIE`, чтобы точно знать, откуда взялись значения.

Для этого убедитесь в том, что в файле php.ini присутствует директива `register_globals = Off`. Если вы не имеете доступа для записи к файлу php.ini, в котором включена директива `register_globals`, вам придется серьезно поговорить с системным администратором или найти нового провайдера, не использующего неправильные настройки, которым уже больше 10 лет. Если вы используете PHP с веб-сервером Apache, настроенным на использование файлов .htaccess уровня каталогов, отключите `register_globals`, добавив директиву `php_flag register_globals off` в файл .htaccess.

Комментарий

Если директива `register_globals` находится в состоянии on, внешние переменные, включая переменные форм и cookie, импортируются прямо в глобальное пространство имен. Это очень удобно, но если вы будете недостаточно внимательно проверять переменные и места их определения, в системе безопасности могут появиться дефекты. Почему? Потому что в вашей внутренней реализации могут использоваться переменные, которые должны быть недоступны извне и которые были перезаписаны без вашего ведома.

В листинге 9.25 приведен простой пример: представьте, что у вас имеется страница для ввода имени пользователя и пароля. Если введенные данные успешно проходят проверку, вы возвращаете идентификатор пользователя и используете его для поиска и вывода личной информации.

Листинг 9.25. Код, небезопасный при использовании директивы register_globals

```
$username = $dbh->quote($_GET['username']);
$password = $dbh->quote($_GET['password']);

$sth = $dbh->query("SELECT id FROM users WHERE username = $username AND
                    password = $password");

if (1 == $sth->numRows()) {
    $row = $sth->fetchRow(DB_FETCHMODE_OBJECT);
    $id = $row->id;
} else {
    "Print bad username and password";
}

if (!empty($id)) {
    $sth = $dbh->query("SELECT * FROM profile WHERE id = $id");
}
```

Обычно значение `$id` задается только вашей программой и является результатом контролируемой выборки из базы данных. Но если кто-то изменит строку запроса и передаст значение `$id`, возникают проблемы. С включенной директивой `register_globals` ваш сценарий все равно сможет выполнить второй запрос к базе данных и вернуть результаты даже после неудачи при поиске имени и пароля. Без `register_globals` переменная `$id` остается неопределенной, потому что задаются только значения `$_REQUEST['id']` и `$_GET['id']`.

Конечно, существуют и другие решения этой проблемы, даже при использовании `register_globals`. Можно изменить структуру кода так, чтобы подобные лазейки стали невозможными. Один из способов показан в листинге 9.26.

Листинг 9.26. Решение проблемы с register_globals

```
$sth = $dbh->query("SELECT id FROM users WHERE username = $username AND
                    password = $password");

if (1 == $sth->numRows()) {
    $row = $sth->fetchRow(DB_FETCHMODE_OBJECT);
    $id = $row->id;
    if (!empty($id)) {
        $sth = $dbh->query("SELECT * FROM profile WHERE id = $id");
    }
} else {
    "Print bad username and password";
}
```

В листинге 9.26 переменная `$id` содержит значение только в том случае, если оно было задано явно в результате обращения к базе данных. Впрочем, иногда бывает трудно реализовать подобную схему из-за структуры программы. Другое решение основано на ручном вызове `unset()` или инициализации всех переменных в начале сценария. Тем самым нежелательное значение `$id` удаляется до того, как ему представится возможность повлиять на ваш код. Но так как PHP не требует обязательной инициализации переменных, вы можете забыть об этом, и тогда в программе появится ошибка, о которой PHP вас не предупредит.

С учетом всех этих причин лучше просто отключить `register_globals`.

См. также

Документация по функции `register_globals`.

9.16. Работа с переменными, имена которых содержат точки

Задача

Требуется обработать переменную, имя которой содержит точку. При отправке формы вам не удается найти переменную в `$_GET` или `$_POST`.

Решение

Замените точку в имени переменной символом подчеркивания. Например, если на форме присутствует элемент ввода с именем `hot.dog`, вы можете обратиться к нему из кода PHP как к переменной `$_GET['hot_dog']` или `$_POST['hot_dog']`.

Комментарий

На ранних стадиях становления PHP, когда директива `register_globals` включалась по умолчанию, переменная формы с именем `hot.dog` не могла превратиться в `$hot.dog` — точки в именах переменных запрещены. Для решения этой проблемы символ «.» заменялся символом «_». Хотя у `$_GET['hot.dog']` и `$_POST['hot.dog']` такой проблемы не существует, преобразование выполняется по соображениям совместимости независимо от состояния директивы `register_globals`.

См. также

Документация по внешним переменным.

9.17. Использование элементов форм с множественным выбором

Задача

Некоторые элементы формы позволяют пользователю выбрать несколько вариантов (например, в раскрывающемся меню или в группе флажков), но PHP «видит» только одно из отправленных значений.

Решение

Завершите имя элемента формы парой квадратных скобок (`[]`). В листинге 9.27 представлена группа флажков с правильным именем.

Листинг 9.27. Назначение имени группы флажков

```
<input type="checkbox" name="boroughs[]" value="bronx"> The Bronx
<input type="checkbox" name="boroughs[]" value="brooklyn"> Brooklyn
<input type="checkbox" name="boroughs[]" value="manhattan"> Manhattan
<input type="checkbox" name="boroughs[]" value="queens"> Queens
<input type="checkbox" name="boroughs[]" value="statenisland"> Staten Island
```

Далее отправленные данные интерпретируются как массив, содержащийся в `$_GET` или `$_POST`, как показано в листинге 9.28.

Листинг 9.28. Обработка группы флажков

```
print 'I love ' . join(' and ', $_POST['boroughs']) . '!';
```

Комментарий

Размещение квадратных скобок `[]` в конце имени элемента формы сообщает PHP, что входные данные следует интерпретировать как массив, а не как скалярное значение. Когда PHP обнаруживает, что с переменной связано более одного значения, сохраняются все эти значения. Если первые три флажка в листинге 9.27 установлены, результат эквивалентен включению кода из листинга 9.29 в начало программы.

Листинг 9.29. Программный эквивалент отправки элемента формы с множественным выбором

```
$_POST['boroughs'][] = "bronx";
$_POST['boroughs'][] = "brooklyn";
$_POST['boroughs'][] = "manhattan";
```

Похожий синтаксис работает и для многомерных массивов. Например, разметка флажка может иметь вид `<input type="checkbox" name="population[NY][NYC]" value="8336697">`. Если флажок установлен, то элемент формы присваивает `$_POST['population']['NY']['NYC']` значение 8336697.

См. также

«Введение» к главе 4 — дополнительная информация о массивах.

9.18. Создание раскрывающихся меню на основании текущей даты

Задача

Требуется создать серию раскрывающихся меню, основанных на значении текущей даты.

Решение

Создайте объект `DateTime` и переберите интересующие вас дни, изменяя объект методом `modify()`.

Листинг 9.30 генерирует значения `<option/>` для сегодняшнего дня и для шести следующих дней. В приведенном примере «сегодняшним днем» является 8 апреля 2013 года.

Листинг 9.30. Генерирование вариантов раскрывающихся меню на основании текущей даты

```
$options = array();
$when = new DateTime();
// Вывод дней одной недели
for ($i = 0; $i < 7; ++$i) {
    $options[$when->getTimestamp()] = $when->format("D, F j, Y");
    $when->modify("+1 day");
}

foreach ($options as $value => $label) {
    print "<option value='$value'>$label</option>\n";
}
```

Если запустить код листинга 9.30 8 апреля 2013 года, он выводит следующий результат:

```
<option value='1365450257'>Mon, April 8, 2013</option>
<option value='1365536657'>Tue, April 9, 2013</option>
<option value='1365623057'>Wed, April 10, 2013</option>
<option value='1365709457'>Thu, April 11, 2013</option>
<option value='1365795857'>Fri, April 12, 2013</option>
<option value='1365882257'>Sat, April 13, 2013</option>
<option value='1365968657'>Sun, April 14, 2013</option>
```

Комментарий

В листинге 9.30 значение каждой даты задается в формате временной метки Unix, потому что с ней удобнее работать в программе. Конечно, вы можете использовать любой формат, который покажется вам более удобным и подходящим.

Использование методов `DateTime#modify()` и `DateTime#format()` освобождает вас от всех хлопот, связанных с вычислениями часовых поясов. Переход на летнее время для соответствующих часовых поясов также будет обрабатываться корректно.

См. также

Глава 3, особенно Рецепт 3.9; документация по функции `DateTime`.

10 Базы данных

10.0. Введение

Базы данных занимают центральное место во многих веб-приложениях. В базе данных можно хранить практически любую информацию, которую вы планируете использовать для поиска и обновления: списки пользователей, каталоги продуктов, заголовки новостей и т. д. Одной из причин, по которым PHP так хорошо подходит для веб-программирования, является всесторонняя поддержка баз данных. PHP может взаимодействовать практически с любыми базами данных — как с реляционными, так и с другими. Также в PHP реализована поддержка ODBC, и даже если ваша любимая база данных не входит в список, но поддерживает ODBC, ее все равно можно будет использовать с PHP.

Базы данных DBM, рассматриваемые в Рецепте 10.1, представляют собой простые, мощные и эффективные неструктурированные файлы, однако структура данных таких баз ограничивается парами «ключ/значение». Если данные можно организовать в виде ассоциаций «ключ/значение», базы данных DBM отлично справятся с этой задачей.

Однако возможности PHP в полной мере раскрываются в сочетании с базами данных SQL. Эта комбинация используется в большинстве рецептов этой главы. Работа с базами данных SQL бывает нетривиальной, но они отличаются исключительной мощью. Чтобы использовать PHP с конкретной базой данных SQL, необходимо явно приказать PHP включить поддержку этой базы данных при компиляции. Если PHP строится с поддержкой динамической загрузки модулей, поддержка базы данных также может быть построена в виде динамического модуля.

В примерах этой главы используется уровень работы с базами данных PDO для PHP 5. При работе с PDO разработчик использует одни и те же функции PHP независимо от того, с каким ядром базы данных он взаимодействует. И хотя синтаксис SQL может зависеть от базы данных, код PHP остается неизменным. В этом отношении PDO обеспечивает абстракцию доступа к данным, а не общую

абстракцию базы данных. Существуют и другие библиотеки, которые пытаются решить задачу общей абстракции базы данных, — они скрывают от разработчика такие подробности реализации, как работа с датами и типы столбцов, за программной прослойкой. Подобные абстракции могут сэкономить время, если вы пишете программу, которая должна использоваться с множеством разных баз данных, но они могут вызвать другие проблемы. При написании кода SQL, ориентированного на конкретный тип базы данных, вы можете использовать особенности этой базы данных для достижения максимального быстродействия.

В комплект поставки PHP 5 входит SQLite — мощная база данных, не требующая отдельного сервера. Это отличное решение при небольшом объеме трафика, если вы не хотите возиться с администрированием более мощного сервера баз данных. В листинге 10.2 обсуждаются достоинства и недостатки SQLite.

Во многих примерах кода SQL в этой главе используется таблица с информацией о знаках зодиака. Структура таблицы представлена в листинге 10.1, а данные приведены в листинге 10.2.

Листинг 10.1. Структура тестовой таблицы

```
CREATE TABLE zodiac (
  id INT UNSIGNED NOT NULL,
  sign CHAR(11),
  symbol CHAR(13),
  planet CHAR(7),
  element CHAR(5),
  start_month TINYINT,
  start_day TINYINT,
  end_month TINYINT,
  end_day TINYINT,
  PRIMARY KEY(id)
);
```

Листинг 10.2. Тестовые данные

```
INSERT INTO zodiac VALUES (1,'Aries','Ram','Mars','fire',3,21,4,19);
INSERT INTO zodiac VALUES (2,'Taurus','Bull','Venus','earth',4,20,5,20);
INSERT INTO zodiac VALUES (3,'Gemini','Twins','Mercury','air',5,21,6,21);
INSERT INTO zodiac VALUES (4,'Cancer','Crab','Moon','water',6,22,7,22);
INSERT INTO zodiac VALUES (5,'Leo','Lion','Sun','fire',7,23,8,22);
INSERT INTO zodiac VALUES (6,'Virgo','Virgin','Mercury','earth',8,23,9,22);
INSERT INTO zodiac VALUES (7,'Libra','Scales','Venus','air',9,23,10,23);
INSERT INTO zodiac VALUES (8,'Scorpio','Scorpion','Mars','water',10,24,11,21);
INSERT INTO zodiac VALUES (9,'Sagittarius','Archer','Jupiter','fire',11,22,12,↵
21);
INSERT INTO zodiac VALUES (10,'Capricorn','Goat','Saturn','earth',12,22,1,19);
INSERT INTO zodiac VALUES (11,'Aquarius','Water Carrier','Uranus','air',1,20,2,↵
18);
INSERT INTO zodiac VALUES (12,'Pisces','Fishes','Neptune','water',2,19,3,20);
```

В Рецептах 10.3–10.8 изложены основные сведения о подключении к серверу баз данных, отправке запросов и получению результатов, а также использованию запросов, изменяющих данные в базе данных. Так как подключение к базе данных

рассматривается в Рецепте 10.3, в коде последующих рецептов этот код опускается, чтобы читателю было проще сосредоточиться на специфике запросов и обработки результатов.

Типичная PHP-программа получает информацию из полей формы HTML и сохраняет ее в базе данных. Некоторые символы (например, апострофы и обратная косая черта) имеют в SQL особый смысл; будьте внимательны, если данные формы содержат эти символы.

В версиях PHP до 5.4.0 существовал механизм *автоматического экранирования*, который пытался упростить задачу разработчика. При включении конфигурационной директивы `magic_quotes_gpc` в переменных, получаемых из get-запросов, post-запросов и cookie, символы одиночной и двойной кавычки, обратной косой черты и нуль-символы экранировались обратной косой чертой. Также можно было включить директиву `magic_quotes_runtime` для автоматического экранирования кавычек, обратных косых черт и нуль-символов от внешних источников (например, запросов к базам данных или текстовых файлов). Например, если режим `magic_quotes_runtime` включен, при чтении файла в массив функцией `file()` специальные символы в этом массиве экранируются обратной косой чертой.

К сожалению, режим автоматического экранирования обычно создает больше проблем, чем решает их. Если отправленные данные формы должны использоваться в любом другом контексте, помимо запроса SQL (например, для отображения на странице), экранирование придется отменять, чтобы страница нормально смотрелась. Если вы используете версию PHP до 5.4.0, различные конфигурационные директивы, связанные с автоматическим экранированием, следует отключить. Правильный способ экранирования пользовательского ввода запросов к базам данных рассматривается в Рецепте 10.7, где объясняется механизм связывания параметров PDO. Кроме того, в Рецепте 10.9 экранирование специальных символов в запросах рассматривается более подробно. Общие методы отладки, используемые для обработки ошибок при выполнении запросов к базам данных, рассматриваются в Рецепте 10.10.

Операции с базами данных, рассматриваемые в следующей группе рецептов, сложнее простых запросов. Рецепт 10.11 показывает, как автоматически генерировать уникальные значения, которые могут использоваться в качестве идентификаторов записей. В Рецепте 10.12 рассматривается построение запросов по списку полей во время выполнения. Таким образом упрощается управление запросами `INSERT` и `UPDATE` с большим количеством столбцов. Рецепт 10.13 объясняет, как отображать ссылки для постраничного перебора итогового набора, с отображением некоторого количества записей на каждой странице. Для ускорения обращений к базе данных можно кэшировать запросы и их результаты, как объясняется в Рецепте 10.14.

В Рецепте 10.15 описана методика управления доступом к одному подключению базы данных из нескольких мест большой программы. Затем Рецепт 10.16 объединяет многие темы, обсуждавшиеся в этой главе, в одну полнофункциональную программу для хранения сообщений интернет-форума в базе данных.

Кроме баз данных SQL, PHP также может работать с многими базами данных, условно относящимися к категории NoSQL — хранилищ данных, предоставляющих другие модели организации и выборки информации. Таких баз данных слишком много, чтобы их можно было рассмотреть здесь, поэтому мы рассмотрим одну из них — Redis — в Рецепте 10.17.

10.1. Использование баз данных DBM

Задача

Имеются данные, которые легко могут быть представлены парами «ключ/значение». Требуется организовать их безопасное хранение и выполнять очень быструю выборку по ключу.

Решение

Воспользуйтесь уровнем абстракции DBA для работы с базой данных «в стиле DBM», как показано в листинге 10.3.

Листинг 10.3. Использование базы данных DBM

```
$dbh = dba_open(__DIR__ . '/fish.db','c','db4') or die($php_errormsg);
// Выборка и изменение значений
if (dba_exists('flounder',$dbh)) {
  $flounder_count = dba_fetch('flounder',$dbh);
  $flounder_count++;
  dba_replace('flounder',$flounder_count, $dbh);
  print "Updated the flounder count.";
} else {
  dba_insert('flounder',1, $dbh);
  print "Started the flounder count.";
}

// Удаление данных
dba_delete('tilapia',$dbh);

// Какая рыба есть в наличии?
for ($key = dba_firstkey($dbh); $key !== false; $key = dba_nextkey($dbh)) {
   $value = dba_fetch($key, $dbh);
   print "$key: $value\n";
}

dba_close($dbh);
```

Комментарий

PHP поддерживает много внутренних реализаций DBM, таких как GDBM, NDBM, QDBM, DB2, DB3, DB4, DBM и CDB. Уровень абстракции DBA позво-

ляет использовать одни и те же функции для любой внутренней реализации DBM. Все реализации хранят пары «ключ/значение». Разработчик может перебирать все ключи в базе данных, получить значение, связанное с конкретным ключом, и также проверять существование пары с заданным ключом. И ключи, и значения представляют собой строки.

Программа в листинге 10.4 ведет список имен и паролей в виде базы данных DBM. В первом аргументе командной строки передается имя пользователя, а во втором — пароль. Если заданное имя пользователя уже существует в базе данных, то пароль заменяется указанным; в противном случае комбинация имени и пароля добавляется в базу данных.

Листинг 10.4. Хранение имен и паролей в базе данных DBM

```
$user = $argv[1];
$password = $argv[2];

$data_file = '/tmp/users.db';

$dbh = dba_open($data_file,'c','db4') or die("Can't open db $data_file");

if (dba_exists($user,$dbh)) {
    print "User $user exists. Changing password.";
} else {
    print "Adding user $user.";
}

dba_replace($user,$password,$dbh) or die("Can't write to database $data_file");

dba_close($dbh);
```

Функция `dba_open()` возвращает дескриптор файла DBM (или `false` в случае ошибки). Функция получает три аргумента. В первом передается имя файла DBM. Второй аргумент содержит режим открытия файла. В режиме `r` существующая база данных открывается только для чтения, а в режиме `w` существующая база данных открывается для чтения и записи. В режиме `c` база данных открывается для чтения и записи, а если база данных не существует — она создается. Наконец, режим `n` работает так же, как `c`, но если база данных уже существует, режим `n` очищает ее. Третий аргумент `dba_open()` определяет используемый обработчик DBM; в приведенном примере используется `db4`.

Чтобы узнать, какие обработчики DBM откомпилированы в установку PHP, воспользуйтесь функцией `dba_handlers()`. Функция возвращает массив с информацией обо всех поддерживаемых обработчиках.

Чтобы узнать, существует ли ключ в базе данных DBM, используйте функцию `dba_exists()`. Функция получает два аргумента: строковый ключ и файловый дескриптор DBM. Она ищет ключ в файле DBM и возвращает `true`, если ключ найден (или `false`, если найти его не удалось). Функция `dba_replace()` получает три аргумента: строковый ключ, строковое значение и файловый дескриптор DBM. Она помещает заданную пару «ключ/значение» в файл DBM. Если элемент с заданным ключом уже существует, текущее значение заменяется новым.

Функция `dba_close()` закрывает базу данных. Файл DBM, открытый вызовом `dba_open()`, автоматически закрывается в конце запроса, но для закрытия долгосрочных подключений, созданных вызовом `dba_popen()`, необходим явный вызов `dba_close()`.

Функции `dba_firstkey()` и `dba_nextkey()` могут использоваться для перебора всех ключей в файле DBM, а функция `dba_fetch()` — для выборки значений, связанных с каждым ключом. Программа в листинге 10.5 вычисляет суммарную длину всех паролей в файле DBM.

Листинг 10.5. Суммирование длин паролей в базе данных DBM

```
$data_file = '/tmp/users.db';
$total_length = 0;
$dbh = dba_open($data_file,'r','db4');
$dbh or die("Can't open database $data_file");

$k = dba_firstkey($dbh);
while ($k) {
    $total_length += strlen(dba_fetch($k,$dbh));
    $k = dba_nextkey($dbh);
}

print "Total length of all passwords is $total_length characters.";

dba_close($dbh);
```

Функция `dba_firstkey()` инициализирует `$k` первым ключом в файле DBM. При каждой итерации цикла `while` функция `dba_fetch()` получает значение, связанное с ключом `$k`, и `$total_length` увеличивается на длину значения (вычисляемую функцией `strlen()`). При вызове `dba_nextkey()` переменной `$k` присваивается следующий ключ в файле.

Для хранения сложных данных в файле данных DBM можно воспользоваться функцией `serialize()`. Листинг 10.6 сохраняет в базе данных DBM структурированную информацию о пользователе; структура сериализуется перед сохранением и десериализуется при чтении.

Листинг 10.6. Хранение структурированных данных в базе данных DBM

```
$dbh = dba_open('users.db','c','db4') or die($php_errormsg);

// Чтение и десериализация данных
$exists = dba_exists($_POST['username'], $dbh);
if ($exists) {
    $serialized_data = dba_fetch($_POST['username'], $dbh) or die($php_errormsg);
    $data = unserialize($serialized_data);
} else {
    $data = array();
}

// Обновление значений
if ($_POST['new_password']) {
```

```
    $data['password'] = $_POST['new_password'];
}
$data['last_access'] = time();

// Запись данных обратно в файл
if ($exists) {
    dba_replace($_POST['username'],serialize($data), $dbh);
} else {
    dba_insert($_POST['username'],serialize($data), $dbh);
}

dba_close($dbh);
```

Хотя листинг 10.6 позволяет сохранить данные нескольких пользователей в одном файле, вы не сможете провести поиск, скажем, по времени последнего обращения пользователя без перебора всех ключей в файле. Если вам необходимы поисковые операции такого рода, разместите данные в базе данных SQL.

Базы данных DBM — шаг вперед по сравнению с обычными текстовыми файлами, но они не поддерживают большинства функций баз данных SQL. Структура данных ограничивается парами «ключ/значение», а надежность блокировки сильно зависит от обработчика DBM. Тем не менее обработчики DBM хорошо подходят для данных с частыми обращениями только для чтения.

См. также

Рецепт 5.7 — сериализация данных; документация по функциям DBA; дополнительная информация об обработчиках DB4 DBM на сайте Oracle.

10.2. Использование базы данных SQLite

Задача

Требуется использовать реляционную базу данных без запуска отдельного серверного процесса.

Решение

Воспользуйтесь SQLite. Эта мощная, надежная система управления базами данных проста в использовании, и для нее не нужно запускать отдельный сервер. База данных SQLite представляет собой обычный файл. Листинг 10.7 создает базу данных SQLite, создает в ней таблицу (если она еще не существует), после чего заполняет ее данными.

Листинг 10.7. Создание базы данных SQLite

```
<programlisting>$db = new PDO('sqlite:/tmp/zodiac');

// Создание таблицы и атомарная вставка данных
$db-&gt;beginTransaction();
```

```
// Проверка поиска таблицы с именем 'zodiac'
$q = $db-&gt;query("SELECT name FROM sqlite_master WHERE type = 'table'" .
                " AND name = 'zodiac'");
// Если запрос не вернул ни одной записи, создать таблицу
// и заполнить ее данными
if ($q-&gt;fetch() === false) {
    $db-&gt;exec(&lt;&lt;&lt;_SQL_
CREATE TABLE zodiac (
  id INT UNSIGNED NOT NULL,
  sign CHAR(11),
  symbol CHAR(13),
  planet CHAR(7),
  element CHAR(5),
  start_month TINYINT,
  start_day TINYINT,
  end_month TINYINT,
  end_day TINYINT,
  PRIMARY KEY(id)
)
_SQL_
);

    // Команды SQL
    $sql=&lt;&lt;&lt;_SQL_
INSERT INTO zodiac VALUES (1,'Aries','Ram','Mars','fire',3,21,4,19);
INSERT INTO zodiac VALUES (2,'Taurus','Bull','Venus','earth',4,20,5,20);
INSERT INTO zodiac VALUES (3,'Gemini','Twins','Mercury','air',5,21,6,21);
INSERT INTO zodiac VALUES (4,'Cancer','Crab','Moon','water',6,22,7,22);
INSERT INTO zodiac VALUES (5,'Leo','Lion','Sun','fire',7,23,8,22);
INSERT INTO zodiac VALUES (6,'Virgo','Virgin','Mercury','earth',8,23,9,22);
INSERT INTO zodiac VALUES (7,'Libra','Scales','Venus','air',9,23,10,23);
INSERT INTO zodiac VALUES (8,'Scorpio','Scorpion','Mars','water',10,24,11,21);
INSERT INTO zodiac VALUES (9,'Sagittarius','Archer','Jupiter','fire',11,22,12,
                           <?pdf-cr?>21);
INSERT INTO zodiac VALUES (10,'Capricorn','Goat','Saturn','earth',12,22,1,19);
INSERT INTO zodiac VALUES (11,'Aquarius','Water Carrier','Uranus','air',1,20,2,
                           <?pdf-cr?>18);
INSERT INTO zodiac VALUES (12,'Pisces','Fishes','Neptune','water',2,19,3,20);
_SQL_;

    // Выделение каждой строки SQL и ее выполнение
    foreach (explode("\n",trim($sql)) as $q) {
        $db-&gt;exec(trim($q));
    }
    $db-&gt;commit();
} else {
    // Ничего не произошло, отмена транзакции
    $db-&gt;rollback();
}</programlisting>
```

Комментарий

Так как базы данных SQLite представляют собой обычные файлы, все предостережения и скрытые ловушки, относящиеся к работе с файлами в PHP, примени-

мы и к базам данных SQLite. Пользователь, с правами которого работает процесс PHP, должен иметь разрешения для чтения и записи в каталог, в котором находится база данных SQLite. Очень желательно разместить этот каталог где-то за пределами корневого каталога документов вашего веб-сервера. Если файл базы данных может читаться напрямую веб-сервером, то пользователь, угадавший его местонахождение, сможет получить все данные в обход всех ограничений, встроенных в запросы ваших PHP-программ.

В PHP расширение `sqlite` предоставляет как обычный доступ SQLite, так и драйвер PDO для SQLite версии 2. Расширение `pdo_sqlite` предоставляет драйвер PDO для SQLite версии 3. Если вы начинаете с нуля, используйте драйвер PDO для SQLite 3, потому что он работает быстрее и обладает большей функциональностью. Если у вас уже установлена база данных SQLite 2, подумайте об использовании драйверов PDO для перехода на SQLite 3.

В листинге 10.7 упоминается специальная системная таблица `sqlite_master`, которая содержит информацию о других таблицах; с ее помощью можно легко определить, существует ли некоторая таблица. В других базах данных для получения подобных системных метаданных используются другие механизмы.

См. также

Документация по SQLite и `sqlite_master`.

10.3. Подключение к базе данных SQL

Задача

Требуется получить доступ к базе данных SQL для сохранения или выборки информации. Вряд ли можно создать динамичный сайт без использования базы данных.

Решение

Создайте новый объект `PDO` с соответствующей строкой подключения. В листинге 10.8 продемонстрировано создание объекта `PDO` для нескольких разновидностей баз данных.

Листинг 10.8. Подключение через PDO

```
// MySQL получает параметры в строке
$mysql = new PDO('mysql:host=db.example.com', $user, $password);
// Параметры в списке разделяются символом ;
$mysql = new PDO('mysql:host=db.example.com;port=31075', $user, $password);
$mysql = new PDO('mysql:host=db.example.com;port=31075;dbname=food', $user,
                 $password);
// Подключение к локальному серверу MySQL
```

```
$mysql = new PDO('mysql:unix_socket=/tmp/mysql.sock', $user, $password);

// PostgreSQL также получает параметры в строке
$pgsql = new PDO('pgsql:host=db.example.com', $user, $password);
// Параметры в списке разделяются символами ' ' или ;
$pgsql = new PDO('pgsql:host=db.example.com port=31075', $user, $password);
$pgsql = new PDO('pgsql:host=db.example.com;port=31075;dbname=food', $user,
                 $password);
// При желании можно включить имя пользователя и пароль в DSN.
$pgsql = new PDO("pgsql:host=db.example.com port=31075 dbname=food user=$user
                 password=$password");

// Oracle
// Если имя базы данных определено в tnsnames.ora, просто включите его
// в DSN как значение параметра dbname
$oci = new PDO('oci:dbname=food', $user, $password);
// В противном случае укажите URI "мгновенного клиента" (Instant Client)
$oci = new PDO('oci:dbname=//db.example.com:1521/food', $user, $password);

// Sybase (если PDO использует библиотеку Sybase ct-lib)
$sybase = new PDO('sybase:host=db.example.com;dbname=food', $user, $password);
// Microsoft SQL Server (если PDO использует библиотеки MS SQL Server)
$mssql = new PDO('mssql:host=db.example.com;dbname=food', $user, $password);
// DBLib (для FreeTDS)
$dblib = new PDO('dblib:host=db.example.com;dbname=food', $user, $password);

// ODBC -- предопределенное подключение
$odbc = new PDO('odbc:food');
// ODBC -- специальное подключение. Предоставьте информацию, необходимую
// используемому драйверу.
$odbc = new PDO('odbc:Driver={Microsoft Access Driver
                (*.mdb)};DBQ=C:\\data\\food.mdb;Uid=Chef');

// SQLite получает только имя файла -- ни имени пользователя, ни пароля
$sqlite = new PDO('sqlite:/usr/local/zodiac.db');
$sqlite = new PDO('sqlite:c:/data/zodiac.db');
// SQLite также может работать с временными базами данных в памяти
$sqlite = new PDO('sqlite::memory:');
// В SQLite v2 DSN выглядят аналогично DSN v3
$sqlite2 = new PDO('sqlite2:/usr/local/old-zodiac.db');
```

Комментарий

Если все прошло нормально, конструктор PDO возвращает новый объект, который может использоваться для выдачи запросов к базе данных. При возникновении проблем выдается исключение `PDOException`.

Как видно из листинга 10.8, форма DSN сильно зависит от базы данных, к которой вы пытаетесь подключиться. Впрочем, в общем случае в первом аргументе конструктора PDO передается строка с описанием местонахождения и имени базы данных, а во втором и третьем аргументах — имя пользователя и пароль для подключения. Однако в общем случае первый аргумент конструктора `PDO` содержит строку, описывающую местонахождение и имя базы данных, а второй

и третий — имя пользователя и пароль для подключения к базе данных. Учтите, что для использования конкретной внутренней реализации PDO соответствующая поддержка должна быть включена при построении PHP. Чтобы определить, какие внутренние реализации PDO поддерживаются вашей конфигурацией PHP, используйте выходные данные метода `PDO::getAvailableDrivers()`.

См. также

Рецепт 10.4 — запрос к базе данных SQL; Рецепт 10.6 — модификация базы данных SQL; документация по PDO.

10.4. Запрос к базе данных SQL

Задача

Требуется прочитать информацию из базы данных.

Решение

Воспользуйтесь функцией `PDO::query()` для передачи запроса SQL базе данных. Затем в цикле `foreach` обработайте каждую строку данных результата, как показано в листинге 10.9.

Листинг 10.9. Отправка запроса базе данных

```
$st = $db->query('SELECT symbol,planet FROM zodiac');
foreach ($st->fetchAll() as $row) {
    print "{$row['symbol']} goes with {$row['planet']} <br/>\n";
}
```

Комментарий

Метод `query()` возвращает объект `PDOStatement`. Его метод `fetchAll()` предоставляет компактный механизм выполнения операции с каждой записью, возвращенной запросом.

Метод `fetch()` возвращает одну строку данных, как показано в листинге 10.10.

Листинг 10.10. Выборка отдельных строк данных

```
$rows = $db->query('SELECT symbol,planet FROM zodiac ORDER BY planet');
$firstRow = $rows->fetch();
print "The first results are that {$firstRow['symbol']} goes with ⏎
{$firstRow['planet']}";
```

Каждый вызов `fetch()` возвращает следующую строку данных из результирующего набора. Если следующая строка данных недоступна, `fetch()` возвращает `false`.

По умолчанию `fetch()` возвращает массив, в который каждый столбец строк данных результирующего набора входит дважды: с индексом, соответствующим имени столбца, и с числовым индексом. Это означает, что переменная `$firstRow` из листинга 10.10 содержит четыре элемента: `$firstRow[0]` содержит `Archer`, `$firstRow[1]` содержит `Jupiter`, `$firstRow['symbol']` содержит `Archer`, и `$firstRow['planet']` содержит `Jupiter`.

Чтобы вызов `fetch()` возвращал строки в другом формате, передайте во втором аргументе `query()` константу `PDO::FETCH_*`. Также одна из констант может передаваться в первом аргументе `fetch()`. Допустимые константы и их влияние на возвращаемые данные `fetch()` перечислены в таблице 10.1.

Таблица 10.1. Константы PDO::FETCH_*

Константа	Формат
PDO::FETCH_BOTH	Массив с числовыми и строковыми (имена столбцов) ключами; используется по умолчанию
PDO::FETCH_NUM	Массив с числовыми ключами
PDO::FETCH_ASSOC	Массив со строковыми (имена столбцов) ключами
PDO_FETCH_OBJ	Объект класса stdClass, имена свойств которого соответствуют именам столбцов
PDO_FETCH_LAZY	Объект класса PDORow, имена свойств которого соответствуют именам столбцов. Свойства не заполняются до первого обращения, поэтому этот формат хорошо подходит для строк данных с большим количеством столбцов. Внимание: если вы сохраните возвращенный объект и произведете выборку новой строки, сохраненный объект будет обновлен значениями из новой строки

Кроме вариантов из таблицы 10.1, существуют и другие варианты структурирования возвращаемых данных. Впрочем, они требуют чего-то большего, чем простая передача константы при вызове `query()` или `fetch()`.

В сочетании с `bindColumn()` режим выборки `PDO::FETCH_BOUND` позволяет создать переменные, значения которых обновляются при каждом вызове `fetch()`. Листинг 10.11 показывает, как это работает.

Листинг 10.11. Связывание столбцов с переменными

```
$row = $db->query('SELECT symbol,planet FROM zodiac',PDO::FETCH_BOUND);
// Значение столбца 'symbol' сохраняется в $symbol
$row->bindColumn('symbol', $symbol);
// Значение второго столбца ('planet') сохраняется в $planet
$row->bindColumn(2, $planet);
while ($row->fetch()) {
    print "$symbol goes with $planet. <br/>\n";
}
```

В листинге 10.11 при каждом вызове `fetch()` переменным `$symbol` и `$planet` присваиваются новые значения. Обратите внимание: при вызове `bindColumn()`

можно использовать как имя столбца, так и число. Нумерация столбцов начинается с 1.

При использовании с `query()` константы `PDO::FETCH_INTO` и `PDO::FETCH_CLASS` помещают строки данных результата в специализированные объекты конкретных классов. Чтобы использовать эти режимы, сначала создайте класс, расширяющий встроенный класс `PDOStatement`. Листинг 10.12 расширяет `PDOStatement` методом, выводящим среднюю длину значений столбцов, а затем создает запрос для его использования.

Листинг 10.12. Расширение PDOStatement

```
class AvgStatement extends PDOStatement {
    public function avg() {
        $sum = 0;
        $vars = get_object_vars($this);
        // Удаление встроенной переменной 'queryString' класса PDOStatement
        unset($vars['queryString']);
        foreach ($vars as $var => $value) {
            $sum += strlen($value);
        }
        return $sum / count($vars);
    }
}
$row = new AvgStatement;
$results = $db->query('SELECT symbol,planet FROM zodiac',PDO::FETCH_INTO, $row);

// При каждом вызове fetch() переменная $row заполняется заново
while ($results->fetch()) {
    print "$row->symbol belongs to $row->planet (Average: {$row->avg()}) <br/>
    \n";
}
```

В листинге 10.12 второй и третий аргументы `query()` приказывают PDO: «При каждой выборке новой строки данных помещать значения свойств в переменную `$row`». Далее в цикле `while()` доступны свойства `$row`, а также вновь определенный метод `avg()`.

Режим `PDO::FETCH_INTO` полезен для хранения данных в одном объекте (например, признака отображения четной или нечетной строки данных) между вызовами `fetch()`. Если для каждой строки данных должен создаваться новый объект, используйте режим `PDO::FETCH_CLASS`. Передайте его `query()`, как это делается с `PDO::FETCH_INTO`, но в третьем аргументе `query()` укажите имя класса, а не экземпляр объекта. Класс, передаваемый с `PDO::FETCH_CLASS`, должен расширять `PDOStatement`.

См. также

Рецепт 10.5 — другие способы выборки данных; Рецепт 10.6 — модификация базы данных SQL; Рецепт 10.7 — эффективное повторение запросов; документация по PDO.

10.5. Выборка строк данных без цикла

Задача

Требуется организовать выполнение запросов и получение возвращаемых данных.

Решение

Используйте метод `fetchAll()` для одновременного получения всех результатов запроса, как показано в листинге 10.13.

Листинг 10.13. Одновременное получение всех результатов

```
$st = $db->query('SELECT planet, element FROM zodiac');
$results = $st->fetchAll();
foreach ($results as $i => $result) {
   print "Planet $i is {$result['planet']} <br/>\n";
}
```

Комментарий

Метод `fetchAll()` полезен при выполнении операций, зависящих от всех строк данных, возвращаемых запросом, например подсчета возвращаемых строк данных или обработки строк данных в нарушение исходного порядка. Как и `fetch()`, метод `fetchAll()` по умолчанию представляет каждую строку данных как массив, использующий как числовые, так и строковые ключи, и получает различные константы `PDO::FETCH_*` для модификации этого поведения.

Метод `fetchAll()` также может получать другие константы, влияющие на возвращаемые результаты. Чтобы выбрать из результатов только один столбец, передайте константу `PDO::FETCH_COLUMN` и второй аргумент с индексом нужного столбца. Первому столбцу соответствует индекс 0, а не 1.

См. также

Рецепт 10.4 — запросы к базам данных SQL и дополнительная информация о режимах выборки; Рецепт 10.6 — модификация базы данных SQL; Рецепт 10.7 — эффективное повторение запросов; документация по PDO.

10.6. Модификация данных в базах данных SQL

Задача

Требуется добавить, удалить или изменить данные в базе данных SQL.

Решение

Используйте метод PDO::exec() для отправки команды INSERT, DELETE или UPDATE, как показано в листинге 10.14.

Листинг 10.14. Использование PDO::exec()

```
$db->exec("INSERT INTO family (id,name) VALUES (1,'Vito')");
$db->exec("DELETE FROM family WHERE name LIKE 'Fredo'");
$db->exec("UPDATE family SET is_naive = 1 WHERE name LIKE 'Kay'");
```

Запрос также можно подготовить методом PDO::prepare(), а потом выполнить его методом PDOStatement::execute(), как показано в листинге 10.15.

Листинг 10.15. Подготовка и выполнение запроса

```
$st = $db->prepare('INSERT INTO family (id,name) VALUES (?,?)');
$st->execute(array(1,'Vito'));

$st = $db->prepare('DELETE FROM family WHERE name LIKE ?');
$st->execute(array('Fredo'));

$st = $db->prepare('UPDATE family SET is_naive = ? WHERE name LIKE ?');
$st->execute(array(1,'Kay'));
```

Комментарий

Метод exec() отправляет полученную команду базе данных. Для запросов INSERT, UPDATE и DELETE он возвращает количество строк данных, задействованных в выполнении запроса.

Методы prepare() и execute() особенно полезны для запросов, которые должны выполняться многократно. Запрос, подготовленный вызовом prepare(), можно будет выполнить с новыми значениями без повторной подготовки. В листинге 10.16 однократно подготовленный запрос выполняется трижды.

Листинг 10.16. Повторное использование подготовленной команды

```
st = $db->prepare('DELETE FROM family WHERE name LIKE ?');
$st->execute(array('Fredo'));
$st->execute(array('Sonny'));
$st->execute(array('Luca Brasi'));
```

См. также

Рецепт 10.7 — информация о повторении запросов; документация по методам PDO::exec(), PDO::prepare() и PDOStatement::execute().

10.7. Эффективное повторение запросов

Задача

Требуется выполнить один запрос несколько раз, подставляя в него разные значения.

Решение

Создайте запрос вызовом `PDO::prepare()`, а затем выполните его вызовом `execute()` для подготовленной команды, возвращаемой `prepare()`. Заполнители в запросе, переданном `prepare()`, заменяются данными при вызове `execute()`, как показано в листинге 10.17.

Листинг 10.17. Повторное выполнение подготовленных команд

```
// Подготовка
$st = $db->prepare("SELECT sign FROM zodiac WHERE element LIKE ?");
// Первое выполнение
$st->execute(array('fire'));
while ($row = $st->fetch()) {
    print $row[0] . "<br/>\n";
}
// Повторное выполнение
$st->execute(array('water'));
while ($row = $st->fetch()) {
    print $row[0] . "<br/>\n";
}
```

Комментарий

Значения, передаваемые `execute()`, называются *связанными параметрами* — каждое значение связывается с заполнителем в запросе. Два главных преимущества связанных параметров — безопасность и скорость. При использовании связанных параметров не нужно беспокоиться об атаках внедрения SQL. PDO автоматически экранирует каждый параметр, так что специальные символы нейтрализуются. Кроме того, при вызове `prepare()` многие внутренние подсистемы баз данных разбирают и оптимизируют запросы, так что каждый вызов `execute()` выполняется быстрее вызова `exec()` или `query()` с полностью сформированным запросом в строке, построенной самостоятельно.

В листинге 10.17 первый вызов `execute()` выполняет запрос `SELECT sign FROM zodiac WHERE element LIKE 'fire'`. Второй вызов `execute()` выполняет запрос `SELECT sign FROM zodiac WHERE element LIKE 'water'`.

При каждом вызове `execute()` значение второго аргумента подставляется на место заполнителя `?`. Если заполнителей несколько, аргументы помещаются в массив в порядке их следования в запросе. В листинге 10.18 представлены вызовы `prepare()` и `execute()` с двумя заполнителями.

Листинг 10.18. Запросы с несколькими заполнителями

```
$st = $db->prepare(
    "SELECT sign FROM zodiac WHERE element LIKE ? OR planet LIKE ?");

// SELECT sign FROM zodiac WHERE element LIKE 'earth' OR planet LIKE 'Mars'
$st->execute(array('earth','Mars'));
```

Кроме заполнителей `?` PDO также поддерживает именованные заполнители. При большом количестве заполнителей такой синтаксис упрощает их чтение. Вместо `?` включите в запрос имя заполнителя (которое должно начинаться с двоеточия), а затем используйте эти имена (без двоеточий) в качестве ключей массива параметров, передаваемого `execute()`. В листинге 10.19 показаны именованные заполнители в действии.

Листинг 10.19. Использование именованных заполнителей

```
$st = $db->prepare(
   "SELECT sign FROM zodiac WHERE element LIKE :element OR planet LIKE :planet");
// SELECT sign FROM zodiac WHERE element LIKE 'earth' OR planet LIKE 'Mars'
$st->execute(array('planet' => 'Mars', 'element' => 'earth'));
$row = $st->fetch();
```

С именованными заполнителями запросы проще читаются, а значения можно передавать `execute()` в любом порядке. Однако следует помнить, что каждое имя заполнителя может встречаться в запросе только один раз. Если вы захотите включить одно значение в запрос повторно, используйте два разноименных заполнителя; таким образом, значение будет дважды передаваться в массиве, передаваемом `execute()`.

Кроме `?` и именованных заполнителей `prepare()` предоставляет третий способ включений значений в запрос: `bindParam()`. Этот метод автоматически связывает содержимое переменной с указанным заполнителем. Пример использования `bindParam()` представлен в листинге 10.20.

Листинг 10.20. Использование метода bindParam()

```
$pairs = array('Mars' => 'water',
               'Moon' => 'water',
               'Sun' => 'fire');
$st = $db->prepare(
  "SELECT sign FROM zodiac WHERE element LIKE :element AND planet LIKE
  :planet");
$st->bindParam(':element', $element);
$st->bindparam(':planet', $planet);
foreach ($pairs as $planet => $element) {
    // Передавать execute() ничего не нужно -
    // значения берутся из $element и $planet
    $st->execute();
    var_dump($st->fetch());
}
```

В листинге 10.20 при вызове `execute()` не нужно передавать никакие значения. Два вызова `bindParam()` указывают PDO: «При каждом выполнении `$st` подставлять содержимое переменной `$element` вместо `:element`, а содержимое пере-

менной `$planet` — вместо `:planet`». Содержимое этих переменных на момент вызова `bindParam()` роли не играет — важны лишь значения, содержащиеся в них в момент вызова `execute()`. Так как команда `foreach` помещает ключи массива в `$planet`, а значения массива в `$element`, в запрос будут подставлены ключи и значения из `$pairs`.

Если вы используете заполнители `?` с `prepare()`, передайте в первом аргументе `bindParam()` позицию заполнителя вместо имени параметра. Нумерация позиций начинается с 1, а не с 0.

Метод `bindParam()` решает, как поступить с переданным значением, в зависимости от типа PHP этого значения. Чтобы заставить `bindParam()` интерпретировать значение как относящееся к конкретному типу, передайте константу типа в третьем аргументе. Константы типов, поддерживаемые `bindParam()`, перечислены в таблице 10.2.

Таблица 10.2. Константы PDO::PARAM_*

Константа	Тип
PDO::PARAM_NULL	NULL
PDO::PARAM_BOOL	boolean
PDO::PARAM_INT	integer
PDO::PARAM_STR	string
PDO::PARAM_LOB	«большой объект»

Тип `PDO::PARAM_LOB` особенно удобен, потому что с ним параметр интерпретируется как поток данных. Такое представление позволяет эффективно сохранять содержимое файлов (и вообще всего, что может быть представлено потоком данных, например удаленного URL-адреса) в таблице базы данных. В листинге 10.21 функция `glob()` используется для сохранения содержимого всех файлов каталога в таблице базы данных.

Листинг 10.21. Сохранение содержимого файлов в базе данных с использованием PDO::PARAM_LOB

```
$st = $db->prepare('INSERT INTO files (path,contents) VALUES (:path,:contents)');
$st->bindParam(':path',$path);
$st->bindParam(':contents',$fp,PDO::PARAM_LOB);
foreach (glob('/usr/local/*') as $path) {
    // Получение файлового дескриптора, с которым может
    // работать PDO::PARAM_LOB
    $fp = fopen($path,'r');
    $st->execute();
}
```

Нюансы использования `PDO::PARAM_LOB` зависят от конкретной базы данных. Например, с Oracle ваш запрос должен создавать пустой дескриптор LOB и находиться внутри транзакции. Правильный синтаксис продемонстрирован в примере «Inserting an image into a database: Oracle» в *man*-странице PDO.

См. также

Документация по методам PDO::prepare(), PDOStatement::execute(), PDOStatement::bindParam() и режиму PDO::PARAM_LOB.

10.8. Получение количества строк данных, возвращаемых запросом

Задача

Требуется узнать, сколько строк данных вернул запрос SELECT либо сколько строк данных изменил запрос INSERT, UPDATE или DELETE.

Решение

При выполнении запроса INSERT, UPDATE или DELETE методом PDO::exec() в возвращаемом значении exec() содержится количество измененных строк данных.

При выполнении запроса INSERT, UPDATE или DELETE методами PDO::prepare() и PDOStatement::execute() для получения количества измененных строк следует вызвать метод PDOStatement::rowCount(), как показано в листинге 10.22.

Листинг 10.22. Получение количества строк данных методом rowCount()

```
$st = $db->prepare('DELETE FROM family WHERE name LIKE ?');
$st->execute(array('Fredo'));
print "Deleted rows: " . $st->rowCount();
$st->execute(array('Sonny'));
print "Deleted rows: " . $st->rowCount();
$st->execute(array('Luca Brasi'));
print "Deleted rows: " . $st->rowCount();
```

Если вы выполняете команду SELECT, существует только один надежный способ узнать количество полученных строк — получите их все при помощи fetchAll() и вызовите функцию count(), как показано в листинге 10.23.

Листинг 10.23. Определение количества строк данных для команды SELECT

```
$st = $db->query('SELECT symbol,planet FROM zodiac');
$all= $st->fetchAll(PDO::FETCH_COLUMN, 1);
print "Retrieved ". count($all) . " rows";
```

Комментарий

Хотя некоторые подсистемы баз данных предоставляют PDO информацию о количестве строк данных, полученных в результате выборки SELECT (чтобы вы могли воспользоваться методом rowCount()), эта возможность поддерживается не для всех баз данных. Следовательно, полагаться на это поведение не стоит.

С другой стороны, выборка всех данных большого результирующего набора может оказаться неэффективной. В таком случае можно запросить размер результирующего набора у базы данных. Используйте то же условие `WHERE`, но вместо списка полей запросите в команде `SELECT` результат вызова функции `COUNT(*)`.

См. также

Документация по методам `PDOStatement::rowCount` и `PDO::exec()`.

10.9. Экранирование в запросах

Задача

Требуется сделать текстовые или двоичные данные безопасными для выполнения запросов.

Решение

Записывайте все свои запросы с заполнителями, чтобы методы `prepare()` и `execute()` могли экранировать символы в строках за вас. В Рецепте 10.7 описаны разные варианты использования заполнителей.

Если вам все же придется выполнять экранирование самостоятельно, используйте метод `PDO::quote()`. Например, такая необходимость может возникнуть при экранировании специальных символов SQL в пользовательском вводе, как показано в листинге 10.24.

Листинг 10.24. Ручное экранирование

```
$safe = $db->quote($_GET['searchTerm']);
$safe = strtr($safe,array('_' => '\_', '%' => '\%'));
$st = $db->query("SELECT * FROM zodiac WHERE planet LIKE $safe");
```

Комментарий

Метод `PDO::quote()` обеспечивает необходимое экранирование текстовых и двоичных данных, но, возможно, вам также потребуется экранировать специальные символы SQL `%` и `_`, чтобы команды `SELECT`, использующие оператор `LIKE`, возвращали правильные результаты. Если элемент `$_GET['searchTerm']` содержит `Melm%`, а код листинга 10.24 не вызовет `strtr()`, запрос вернет строки данных, у которых столбец `planet` содержит `Melmac`, `Melmacko`, `Melmacedonia` и вообще любой текст, начинающийся с `Melm`.

Так как `%` является специальным символом SQL, который обозначает совпадение произвольного количества символов (по аналогии с * в синтаксисе командного процессора), а специальный символ _ обозначает ровно один символ (по аналогии

с ? в синтаксисе командного процессора), эти символы также необходимо экранировать обратной косой чертой.

Функция `strtr()` должна вызываться после `PDO::quote()`. В противном случае `PDO::quote()` экранирует символы \, добавленные `strtr()`. Если сначала вызывается `PDO::quote()`, то `Melm_` преобразуется в `Melm\_`, что интерпретируется базой данных как «строка Melm, за которой следует литеральный символ подчеркивания». Если же вызвать `PDO::quote()` после `strtr()`, `Melm_` превращается в `Melm\\_`, что интерпретируется базой данных как «строка Melm, за которой следует литеральный символ обратной косой черты, за которым следует специальный символ подчеркивания». То же самое произойдет, если экранировать специальные символы SQL, а затем использовать полученное значение как связанный параметр.

Экранирование значений заполнителей также происходит при включении режима `magic_quotes_gpc` или `magic_quotes_runtime`. Аналогичным образом, при вызове `PDO::quote()` для значения в действующем режиме автоматического экранирования значение все равно будет экранировано. Чтобы код был максимально универсальным, удалите символы \, появившиеся из-за автоматического экранирования, перед использованием запроса с заполнителями или вызовами `PDO::quote()`. Пример такой проверки представлен в листинге 10.25.

Листинг 10.25. Проверка автоматического экранирования

```
// Директива magic_quotes_sybase также может повлиять на ситуацию
if (get_magic_quotes_gpc() && (! ini_get('magic_quotes_sybase'))) {
    $fruit = stripslashes($_GET['fruit']);
} else {
    $fruit = $_GET['fruit'];
}
$st = $db->prepare('UPDATE orchard SET trees = trees - 1 WHERE fruit = ?');
$st->execute(array($fruit));
```

Если сервер находится под вашим контролем, отключите режим автоматического экранирования; это существенно упростит вашу жизнь. Но если вы пытаетесь написать максимально универсальный код, который будет работать в любой среде, неподконтрольной вам, об этой проблеме следует помнить.

См. также

Документация по методу `PDO::quote()` и режиму автоматического экранирования.

10.10. Работа с журналом отладочной информации и сообщений об ошибках

Задача

Требуется получить доступ к информации, упрощающей отладку проблем с базой данных. Например, если при выполнении запроса происходит ошибка, вы хотите увидеть, какое сообщение об ошибке возвращает база данных.

Решение

Чтобы получить код ошибки в случае неудачного выполнения операции, используйте метод `PDO::errorCode()` или `PDOStatement::errorCode()`. Соответствующий метод `errorInfo()` возвращает более подробную информацию. В листинге 10.26 обрабатывается ошибка, которая происходит при попытке обращения к несуществующей таблице.

Листинг 10.26. Вывод информации об ошибке

```
$st = $db->prepare('SELECT * FROM imaginary_table');
if (! $st) {
    $error = $db->errorInfo();
    print "Problem ({$error[2]})";
}
```

Комментарий

Метод `errorCode()` возвращает код ошибки, состоящий из пяти символов. PDO использует коды ошибок SQL 92 SQLSTATE. Согласно этому стандарту, 00000 означает «нет ошибки», так что вызов `errorCode()`, возвращающий 00000, является признаком успеха.

Метод `errorInfo()` возвращает массив из трех элементов. Первый элемент содержит код SQLSTATE из пяти символов (тот же, который возвращается при вызове `errorCode()`). Второй элемент содержит код ошибки, специфический для конкретной подсистемы базы данных. Третий элемент содержит сообщение об ошибке, специфическое для конкретной подсистемы базы данных.

Помните, что метод `errorCode()` или `errorInfo()` должен вызываться для того же объекта, для которого вызывался метод, проверяемый на наличие ошибки. В листинге 10.26 метод `prepare()` вызывается для объекта PDO, так что метод `errorInfo()` вызывается для объекта `PDO`. Если вы хотите проверить, успешно ли выполнился вызов `fetch()` для объекта `PDOStatement`, вызовите `errorCode()` или `errorInfo()` для объекта `PDOStatement`.

Это правило не действует при создании новых объектов `PDO`. Если попытка создания объекта завершается неудачей, `PDO` выдает исключение. Дело в том, что при возникновении ошибки не будет объекта, для которого можно было бы вызвать `errorCode()` или `errorInfo()`. Сообщение в исключении содержит подробное описание причин ошибки при подключении.

Чтобы уровень PDO выдавал исключения при возникновении *любой* ошибки, вызовите `setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION)` для объекта PDO после его создания. Это позволит организовать централизованную обработку ошибок баз данных вместо загромождения кода многократными вызовами `errorCode()` и `errorInfo()`. Листинг 10.27 выполняет серию операций с базой данных, заключенную в блок `try/catch`.

Листинг 10.27. Обработка исключений при работе с базой данных

```
try {
    $db = new PDO('sqlite:/tmp/zodiac.db');
```

```
    // Выдавать исключение для любой ошибки базы данных
    $db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
    $st = $db->prepare('SELECT * FROM zodiac');
    $st->execute();
    while ($row = $st->fetch(PDO::FETCH_NUM)) {
        print implode(',',$row). "<br/>\n";
    }
} catch (Exception $e) {
    print "Database Problem: " . $e->getMessage();
}
```

Обработка ошибок PDO в виде исключений также удобна при использовании транзакций. Если проблема с запросом возникнет после начала транзакции, просто отмените транзакцию при обработке исключения.

Кроме режима ошибок-исключений также существует похожий режим ошибок-предупреждений. Вызов `setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_WARNING)` приказывает PDO выдавать предупреждения при обнаружении ошибок работы с базами данных. Если вы предпочитаете работать с обычными ошибками PHP вместо исключений, этот режим вам подойдет. Создайте пользовательский обработчик вызовом `set_error_handler()` для обработки событий уровня `E_WARNING` и вы сможете обрабатывать в нем проблемы, возникающие при работе с базой данных.

Независимо от выбранного режима PDO выдает исключение, если ошибка происходит при создании исходного объекта `PDO`. Если вы используете PDO, настоятельно рекомендуется назначить обработчик исключений по умолчанию вызовом `set_exception_handler()`. Без обработчика по умолчанию неперехваченное исключение приводит к выводу полной трассировки стека, если режим `display_errors` активен. Если исключение выдается при подключении к базе данных, трассировка может содержать конфиденциальную информацию, включая учетные данные для подключения.

См. также

Документация по методам `PDO::errorCode()`, `PDO::errorInfo()`, `PDOStatement::errorCode()`, `PDOStatement::errorInfo()`, функциям `set_exception_handler()` и `set_error_handler()`. На с. 619 стандарта SQL 92 приведены коды ошибок SQLSTATE, поддерживаемые PDO, но некоторые подсистемы баз данных могут выдавать ошибки, отсутствующие в списке.

10.11. Создание уникальных идентификаторов

Задача

Требуется присвоить уникальный идентификатор пользователю, публикации или любому другому объекту при его добавлении в базу данных.

Решение

Воспользуйтесь функцией PHP `uniqid()`. Чтобы ограничить набор символов, входящих в идентификатор, обработайте значение функцией `md5()`; она возвращает строку, состоящую только из цифр и букв a–f. В листинге 10.28 приведены примеры создания идентификаторов с использованием обоих способов.

Листинг 10.28. Создание уникальных идентификаторов

```
$st = $db->prepare('INSERT INTO users (id, name) VALUES (?,?)');
$st->execute(array(uniqid(), 'Jacob'));
$st->execute(array(md5(uniqid()), 'Ruby'));
```

Также идентификаторы можно генерировать средствами базы данных. Например, SQLite 3 и MySQL поддерживают столбцы `AUTOINCREMENT`, которые автоматически включают целочисленные идентификаторы во вставляемые строки данных.

Комментарий

Функция `uniqid()` использует текущее время (в микросекундах) и случайное число для генерирования строки, которую чрезвычайно трудно подобрать. Функция `md5()` хеширует переданное ей значение. Она не добавляет в идентификатор никакого случайного фактора, но ограничивает состав символов, которые могут в него входить. Результаты `md5()` не содержат знаков препинания, так что вам не нужно беспокоиться о проблемах экранирования.

Если вы предпочитаете доверить генерирование уникальных идентификаторов базе данных, используйте соответствующий синтаксис при создании таблицы. В листинге 10.29 показано, как в SQLite создать таблицу со столбцом, содержащим целочисленный идентификатор, который автоматически увеличивается при каждой вставке новой строки данных.

Листинг 10.29. Создание автоматически увеличиваемого столбца в SQLite

```
// Тип INTEGER PRIMARY KEY AUTOINCREMENT сообщает SQLite,
// что столбцу должны присваиваться автоматически увеличиваемые
// идентификаторы.
$db->exec(<<<_SQL_
  CREATE TABLE users (
    id  INTEGER PRIMARY KEY AUTOINCREMENT,
    name VARCHAR(255)
  )
_SQL_
);
// Включать значение 'id' не нужно - SQLite присваивает его автоматически
$st = $db->prepare('INSERT INTO users (name) VALUES (?)');
// Значение 'id' в этих строках данных задается автоматически
foreach (array('Jacob','Ruby') as $name) {
    $st->execute(array($name));
}
```

Листинг 10.30 показывает, как сделать то же самое в MySQL.

Листинг 10.30. Создание автоматически увеличиваемого столбца в MySQL

```
// Тип AUTO_INCREMENT сообщает MySQL, что столбцу должны
// присваиваться автоматически увеличиваемые идентификаторы.
// Столбец должен быть первичным ключом
$db->exec(<<<_SQL_
  CREATE TABLE users (
    id  INT NOT NULL AUTO_INCREMENT,
    name VARCHAR(255),
    PRIMARY KEY(id)
  )
_SQL_
);

// Включать значение 'id' не нужно - MySQL присваивает его автоматически
$st = $db->prepare('INSERT INTO users (name) VALUES (?)');
// Значение 'id' в этих строках данных задается автоматически
foreach (array('Jacob','Ruby') as $name) {
    $st->execute(array($name));
}
```

Если идентификаторы автоматически создаются базой данных, их можно получить методом `PDO::lastInsertId()`. Вызов `lastInsertId()` для объекта `PDO` возвращает автоматически сгенерированный идентификатор последней вставленной строки данных. Некоторые подсистемы баз данных также позволяют передать `lastInsertId()` имя последовательности для получения последнего значения из этой последовательности. Некоторые подсистемы баз данных вообще не поддерживают `PDO::lastInsertId()`. В этом случае `PDO::lastInsertId()` вызывает ошибку с кодом SQLSTATE, равным `IM001`.

См. также

Документация по функциям `uniqid()`, `md5()`, методу `PDO::lastInsertId()`, SQLite и `AUTOINCREMENT`, MySQL и `AUTO_INCREMENT`.

10.12. Программное построение запросов

Задача

Требуется сконструировать запрос `INSERT` или `UPDATE` по массиву имен полей. Например, вы хотите вставить в базу данных запись нового пользователя. Вместо того чтобы жестко кодировать каждое поле этой записи (имя пользователя, адрес электронной почты, почтовый адрес, дата рождения и т. д.), вы помещаете имена полей в массив и используете массив для построения запроса. Такой код проще в сопровождении, особенно при условном выполнении запросов `INSERT` или `UPDATE` с одним набором полей.

Решение

Чтобы сконструировать запрос `UPDATE`, постройте массив пар «поле/значение», а затем объедините элементы массива функцией `implode()`, как показано в листинге 10.31.

Листинг 10.31. Построение запроса UPDATE

```
// Список имен полей
$fields = array('symbol','planet','element');
$update_fields = array();
$update_values = array();
foreach ($fields as $field) {
    $update_fields[] = "$field = ?";
    // Предполагается, что данные поступают от формы
    $update_values[] = $_POST[$field];
}
$st = $db->prepare("UPDATE zodiac SET " .
                   implode(',', $update_fields) .
                   'WHERE sign = ?');
// Добавление 'sign' в массив значений
$update_values[] = $_GET['sign'];
// Выполнение запроса
$st->execute($update_values);
```

С запросами `INSERT` происходит то же самое, хотя синтаксис SQL выглядит немного иначе, как показано в листинге 10.32.

Листинг 10.32. Построение запроса INSERT

```
// Список имен полей
$fields = array('symbol','planet','element');
$placeholders = array();
$values = array();
foreach ($fields as $field) {
    // Один заполнитель на поле
    $placeholders[] = '?';
    // Предполагается, что данные поступают от формы
    $values[] = $_POST[$field];
}
$st = $db->prepare('INSERT INTO zodiac (' .
                   implode(',',$fields) .
                   ') VALUES (' .
                   implode(',', $placeholders) .
                   ')');
// Выполнение запроса
$st->execute($values);
```

Комментарий

С заполнителями такие задачи решаются проще простого. Так как заполнители обеспечивают экранирование предоставленных данных, вы можете легко занести данные, предоставленные пользователем, в сгенерированные на программном уровне запросы.

Если последовательно генерируемые целочисленные идентификаторы используются как первичные ключи, два способа построения запросов можно объединить в одну функцию. Функция определяет, существует ли запись, а затем генерирует правильный запрос с новым идентификатором, как показано в функции `build_query()` из листинга 10.33.

Листинг 10.33. build_query()

```
function build_query($db,$key_field,$fields,$table) {
    $values = array();
    if (! empty($_POST[$key_field])) {
        $update_fields = array();
        foreach ($fields as $field) {
            $update_fields[] = "$field = ?";
            // Предполагается, что данные поступают от формы
            $values[] = $_POST[$field];
        }
        // Значение ключевого поля добавляется в массив $values
        $values[] = $_POST[$key_field];
        $st = $db->prepare("UPDATE $table SET " .
                   implode(',', $update_fields) .
                   "WHERE $key_field = ?");
    } else {
        // Начать с уникального идентификатора.
        // Если значение должно генерироваться базой данных,
        // используйте NULL.
        $values[] = md5(uniqid());
        $placeholders = array('?');
        foreach ($fields as $field) {
            // Один заполнитель на каждое поле
            $placeholders[] = '?';
            // Предполагается, что данные поступают от формы
            $values[] = $_POST[$field];
        }
        $st = $db->prepare("INSERT INTO $table ($key_field," .
                           implode(',',$fields) . ') VALUES ('.
                           implode(',',$placeholders) .')');
    }
    $st->execute($values);
    return $st;
}
```

При помощи этой функции можно создать простую страницу для редактирования всей информации в таблице `zodiac`, как показано в листинге 10.34.

Листинг 10.34. Простая страница для добавления/редактирования записей

```
// Файл, в котором определяется build_query()
include __DIR__ . '/buildquery.php';

$db = new PDO('sqlite:/tmp/zodiac.db');
$db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

$fields = array('sign','symbol','planet','element',
```

```
                'start_month','start_day','end_month','end_day');

$cmd = isset($_REQUEST['cmd']) ? $_REQUEST['cmd'] : 'show';

switch ($cmd) {
 case 'edit':
 try {
    $st = $db->prepare('SELECT ' . implode(',',$fields) .
                       ' FROM zodiac WHERE id = ?');
    $st->execute(array($_GET['id']));
    $row = $st->fetch(PDO::FETCH_ASSOC);
 } catch (Exception $e) {
     $row = array();
 }
 case 'add':
     print '<form method="post" action="' .
           htmlentities($_SERVER['PHP_SELF']) . '">';
     print '<input type="hidden" name="cmd" value="save">';
     print '<table>';
     if ('edit' == $cmd) {
         printf('<input type="hidden" name="id" value="%d">',
                $_GET['id']);
     }
     foreach ($fields as $field) {
         if ('edit' == $cmd) {
             $value = htmlentities($row[$field]);
         } else {
             $value = '';
         }
         printf('<tr><td>%s: </td><td><input type="text" name="%s" value="%s">',
                $field,$field,$value);
         printf('</td></tr>');
     }
     print '<tr><td></td><td><input type="submit" value="Save"></td></tr>';
     print '</table></form>';
     break;
 case 'save':
     try {
       $st = build_query($db,'id',$fields,'zodiac');
       print 'Added info.';
     } catch (Exception $e) {
       print "Couldn't add info: " . htmlentities($e->getMessage());
     }
     print '<hr>';
 case 'show':
 default:
     $self = htmlentities($_SERVER['PHP_SELF']);
     print '<ul>';
     foreach ($db->query('SELECT id,sign FROM zodiac') as $row) {
         printf('<li> <a href="%s?cmd=edit&id=%s">%s</a>',
                $self,$row['id'],htmlentities($row['sign']));
     }
     print '<hr><li> <a href="'.$self.'?cmd=add">Add New</a>';
     print '</ul>';
     break;
}
```

Команда `switch` управляет действиями, выполняемыми программой, в зависимости от значения `$_REQUEST['cmd']`. Если `$_REQUEST['cmd']` содержит значение `add` или `edit`, программа отображает форму с текстовыми полями для каждого поля в массиве `$fields`, как показано на рис. 10.1. Если `$_REQUEST['cmd']` содержит значение `edit`, то значения строки с заданным идентификатором `$id` загружаются из базы данных и отображаются на форме. Если `$_REQUEST['cmd']` содержит команду `save`, программа использует функцию `build_query()` для генерирования соответствующего запроса для вставки (`INSERT`) или обновления (`UPDATE`) информации в базе данных. После сохранения (или если значение `$_REQUEST['cmd']` не задано) программа выводит список всех знаков зодиака, как показано на рис. 10.2.

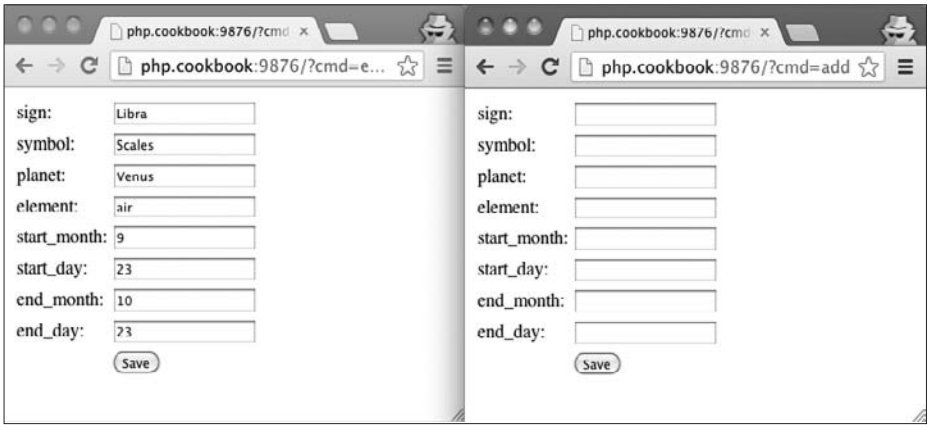

Рис. 10.1. Редактирование и добавление записи

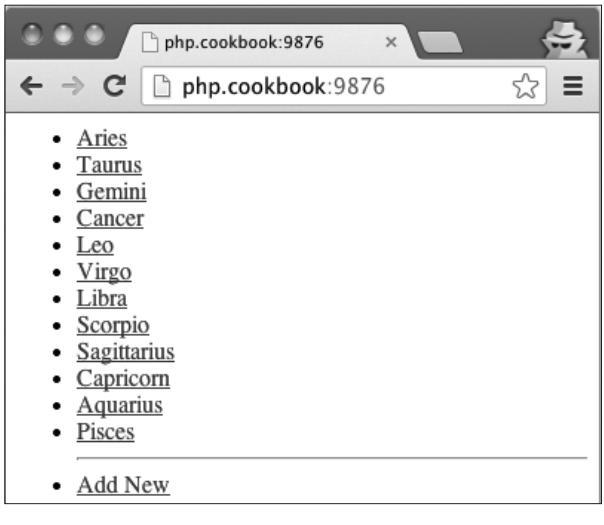

Рис. 10.2. Перечень записей

Выбор команды, которую строит `build_query()` (`INSERT` или `UPDATE`), зависит от присутствия переменной запроса `$_REQUEST['id']` (потому что `id` передается в `$key_field`). Если значение `$_REQUEST['id']` не пусто, то функция строит запрос `UPDATE` для изменения строки данных с заданным идентификатором. Если значение `$_REQUEST['id']` пусто (или вообще не задано), функция генерирует новый идентификатор и использует его в запросе `INSERT`, добавляющем строку в таблицу. Чтобы вызов `build_query()` соблюдал настройку `AUTOINCREMENT` базы данных, начните `$values` с состояния `null` вместо `md5(uniqid())`.

См. также

Рецепт 10.7 — информация о заполнителях, функциях `prepare()` и `execute()`; документация по методам `PDO::prepare()` и `PDOStatement::execute()`.

10.13. Создание страничных ссылок на серии записей

Задача

Требуется организовать страничный вывод большого набора данных и предоставить ссылки для перемещения по этому набору.

Решение

Используйте синтаксис базы данных для выборки части записей, соответствующих вашему запросу. Листинг 10.35 показывает, как это сделать в SQLite.

Листинг 10.35. Страничный вывод данных с использованием SQLite

```
// Выборка пяти строк данных после первых трех
foreach ($db->query('SELECT * FROM zodiac ' .
                    'ORDER BY sign LIMIT 5 ' .
                    'OFFSET 3') as $row) {
     // Обработка каждой строки
}
```

Функции `indexed_links()` и `print_link()` в этом рецепте помогают организовать страничный вывод информации. В листинге 10.36 представлен пример использования этих функций.

Листинг 10.36. Страничный вывод результатов

```
$offset = isset($_GET['offset']) ? intval($_GET['offset']) : 1;
if (! $offset) { $offset = 1; }
$per_page = 5;
$total = $db->query('SELECT COUNT(*) FROM zodiac')->fetchColumn(0);

$limitedSQL = 'SELECT * FROM zodiac ORDER BY id ' .
```

```
              "LIMIT $per_page OFFSET " . ($offset-1);
$lastRowNumber = $offset - 1;

foreach ($db->query($limitedSQL) as $row) {
    $lastRowNumber++;
    print "{$row['sign']}, {$row['symbol']} ({$row['id']}) <br/>\n";
}

indexed_links($total,$offset,$per_page);
print "<br/>";
print "(Displaying $offset - $lastRowNumber of $total)";
```

Комментарий

Функция `print_link()` приведена в листинге 10.37, а функция `indexed_links()` — в листинге 10.38.

Листинг 10.37. print_link()

```
function print_link($inactive,$text,$offset='') {
    if ($inactive) {
        print "<span class='inactive'>$text</span>";
    } else {
        print "<span class='active'>".
              "<a href='" . htmlentities($_SERVER['PHP_SELF']) .
              "?offset=$offset'>$text</a></span>";
    }
}
```

Листинг 10.38. indexed_links()

```
function indexed_links($total,$offset,$per_page) {
    $separator = ' | ';

    // Вывод ссылки "<<Prev"
    print_link($offset == 1, '<< Prev', max(1, $offset - $per_page));

    // Вывод всех групп, кроме последней
    for ($start = 1, $end = $per_page;
         $end < $total;
         $start += $per_page, $end += $per_page) {
             print $separator;
             print_link($offset == $start, "$start-$end", $start);
    }

    /* Вывод последней группы -
     * сейчас $start указывает на элемент в начале
     * последней группы
     */

    /* Текст должен содержать диапазон только в том случае, если
     * последняя страница содержит более одного элемента. Например,
     * ссылка для последней группы в наборе из 11 элементов с 5 элементами
     * на страницу должна содержать текст "11", а не "11-11"
     */
    $end = ($total > $start) ? "-$total" : '';
```

```
    print $separator;
    print_link($offset == $start, "$start$end", $start);

    // Вывод ссылки "Next>>"
    print $separator;
    print_link($offset == $start, 'Next >>',$offset + $per_page);
}
```

Чтобы использовать эти функции, прочитайте нужное подмножество данных при помощи функций PDO и выведите их. Вызовите функцию `indexed_links()` для вывода индексных ссылок.

После подключения к базе данных необходимо убедиться в том, что `$offset` содержит соответствующее значение — начальную запись отображаемого набора данных. Чтобы начать от начала результирующего набора, переменная `$offset` должна быть равна 1. Переменная `$per_page` содержит количество записей, отображаемых на каждой странице, а `$total` содержит общее количество записей во всем результирующем наборе. В нашем примере отображаются все записи со знаками зодиака, поэтому `$total` присваивается количество всех строк данных в таблице.

Запрос SQL, который загружает информацию из таблицы в нужном порядке, выглядит так:

```
$limitedSQL = 'SELECT * FROM zodiac ORDER BY id ' .
"LIMIT $per_page OFFSET " . ($offset-1);
```

Ключевые слова `LIMIT` и `OFFSET` приказывают SQLite вернуть нужное подмножество всех подходящих записей.

Соответствующие строки данных загружаются вызовом `$db->query($limitedSQL)`, после чего выводится информация из каждой строки данных. После вывода данных вызов `indexed_links()` строит навигационные ссылки. Результат выполнения, при котором значение `$offset` не задано (или равно 1), показан на рис. 10.3.

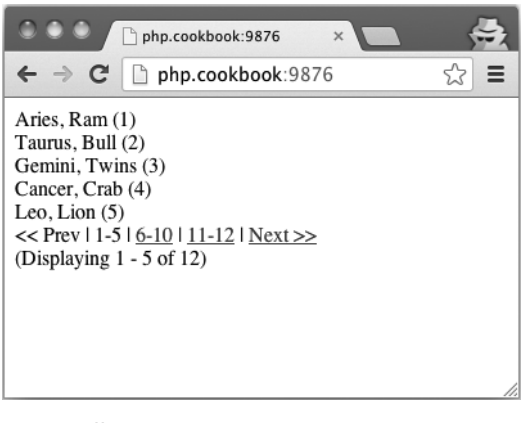

Рис. 10.3. Страничный вывод данных с использованием indexed_links()

На рис. 10.3 ссылки `6-10`, `11-12` и `Next >>` ведут на одну страницу с измененными аргументами `$offset`. Ссылки `<< Prev` и `1-5` недоступны, потому что они ведут на страницу, отображаемую в настоящий момент.

См. также

Описание страничного вывода в инфраструктуре Solar и информация о разновидностях синтаксиса страничного вывода в разных базах данных.

10.14. Кэширование запросов и результатов

Задача

Требуется избежать лишних (и потенциально затратных) запросов к базам данных в том случае, если их результаты остались неизменными.

Решение

Используйте пакет Cache_Lite из репозитория PEAR, позволяющий легко организовать кэширование произвольных данных. В данном случае следует кэшировать результаты запроса `SELECT` и использовать текст запроса как ключ кэша.

Листинг 10.39 демонстрирует кэширование результатов запросов с применением Cache_Lite.

Листинг 10.39. Кэширование результатов запроса

```
require_once 'Cache/Lite.php';

$opts = array(
   // Каталог для размещения кэшированных данных
   'cacheDir' => '/tmp/',
   // В кэше будут храниться массивы
  'automaticSerialization' => true,
  // Продолжительность хранения данных в кэше
  'lifeTime' => 600 /* десять минут */);

// Создание кэша
$cache = new Cache_Lite($opts);

// Подключение к базе данных
$db = new PDO('sqlite:/tmp/zodiac.db');

// Определение запроса и его параметров
$sql = 'SELECT * FROM zodiac WHERE planet = ?';
$params = array($_GET['planet']);
```

```
// Получение уникального ключа кэша
$key = cache_key($sql, $params);

// Попытка получения результатов из кэша
$results = $cache->get($key);

if ($results === false) {
    // Результаты не найдены, выполнить запрос и поместить результаты в кэш
    $st = $db->prepare($sql);
    $st->execute($params);
    $results = $st->fetchAll();
    $cache->save($results);
}

// $results содержит наши данные, откуда бы они ни были получены
foreach ($results as $result) {
    print "$result[id]: $result[planet], $result[sign] <br/>\n";
}

function cache_key($sql, $params) {
    return md5($sql .
               implode('|',array_keys($params)) .
               implode('|',$params));
}
```

Комментарий

Cache_Lite — универсальный и эффективный механизм кэширования произвольной информации. Конструктор Cache_Lite получает массив параметров, управляющих его поведением. Два самых важных параметра (листинг 10.39) — `automaticSerialization` (упрощает хранение массивов в кэше) и `cacheDir` (определяет место хранения файлов кэша). Проследите за тем, чтобы параметр `cacheDir` завершался символом `/`.

Кэш всего лишь связывает ключи со значениями. Разработчик должен позаботиться о том, чтобы ключ кэша однозначно идентифицировал кэшируемые данные — в данном случае запрос SQL и связанные с ним параметры. Функция `cache_key` вычисляет соответствующий ключ, после чего листинг 10.39 проверяет, находится ли результат в кэше. Если его там нет, запрос передается базе данных, а результаты сохраняются в кэше для следующего обращения.

Учтите, что в кэш нельзя поместить объект `PDO` или `PDOStatement`, — необходимо провести выборку результатов, а затем поместить результаты в кэш.

По умолчанию данные остаются в кэше в течение 1 часа. Продолжительность хранения можно изменить, передав другое значение (в секундах) в параметре `lifeTime` при создании нового объекта `Cache_Lite`. Если срок жизни данных не должен истекать автоматически, передайте `null`.

Содержимое кэша не изменяется при изменении базы данных запросами `INSERT`, `UPDATE` или `DELETE`. Если в кэше находятся результаты команды `SELECT` для данных, которые уже не существуют в базе данных, необходимо явно очистить кэш вы-

зовом метода `Cache_Lite::clean()`. Также существует возможность удаления из кэша отдельных элементов; для этого при вызове `Cache_Lite::remove()` передается ключ кэша.

Функция `cache_key()` в листинге 10.39 учитывает регистр символов. Это означает, что если в кэше хранятся результаты `SELECT * FROM zodiac`, то при выполнении запроса `SELECT * from zodiac` результаты не будут обнаружены и запрос будет выполнен повторно. Последовательное использование регистра, интервалов и порядка полей при построении запросов SQL обеспечит более эффективное использование кэша.

Одно из преимуществ решений уровня PHP (таких как Cache_Lite) заключается в их независимости от базы данных. С другой стороны, привязка к конкретной базе данных позволяет использовать механизмы кэширования запросов, реализованные в этой базе. Вследствие своей более плотной интеграции с базой данных такие кэши более разумно управляют сроком жизни кэшированных данных при их изменении. В частности, вы можете прочитать в документации о том, как включить кэш запросов MySQL.

См. также

Документация Cache_Lite.

10.15. Использование подключения к базе данных в любой точке программы

Задача

Имеется программа, содержащая большое количество функций и классов. Требуется создать одно подключение к базе данных, с которым можно будет легко работать из любой точки программы.

Решение

Воспользуйтесь статическим методом класса, который возвращает существующее подключение или создает подключение, если оно не существует (листинг 10.40).

Листинг 10.40. Создание подключения к базе данных в статическом методе класса

```
class DBCxn {
    // DSN для подключения
    public static $dsn = 'sqlite:c:/data/zodiac.db';
    public static $user = null;
    public static $pass = null;
    public static $driverOpts = null;

    // Внутренняя переменная для хранения подключения
```

```
    private static $db;
    // Клонирование или создание экземпляров запрещено
    final private function __construct() { }
    final private function __clone() { }

    public static function get() {
        // Если подключение не существует, оно создается
        if (is_null(self::$db)) {
            self::$db = new PDO(self::$dsn, self::$user, self::$pass,
                                self::$driverOpts);
        }
        // Вернуть подключение
        return self::$db;
    }
}
```

Комментарий

Метод `DBCxn::get()`, определенный в листинге 10.40, решает сразу две задачи: его можно вызывать из любой точки программы, не беспокоясь об области видимости переменной, и он не позволяет создать в программе более одного экземпляра подключения.

Чтобы изменить подключение, предоставляемое методом `DBCxn::get()`, измените свойства `$dsn`, `$user`, `$pass` и `$driverOpts`. Если вам понадобится управлять несколькими подключениями к базе данных во время одного выполнения сценария, преобразуйте `$dsn` и `$db` в массивы и передайте `get()` аргумент, идентифицирующий используемое подключение. В листинге 10.41 приведена версия `DBCxn` для работы с тремя разными базами данных.

Листинг 10.41. Использование нескольких подключений к базам данных

```
class DBCxn {
    // DSN для подключения
    public static $dsn =
       array('zodiac' => 'sqlite:c:/data/zodiac.db',
             'users' => array('mysql:host=db.example.com','monty','7f2iuh'),
             'stats' => array('oci:statistics', 'statsuser','statspass'));

    // Внутренняя переменная для хранения подключения
    private static $db = array();
    // Клонирование или создание экземпляров запрещено
    final private function __construct() { }
    final private function __clone() { }

    public static function get($key) {
        if (! isset(self::$dsn[$key])) {
            throw new Exception("Unknown DSN: $key");
        }
        // Если подключение не существует, оно создается
        if (! isset(self::$db[$key])) {
            if (is_array(self::$dsn[$key])) {

                $c = new ReflectionClass('PDO');
```

```
                self::$db[$key] = $c->newInstanceArgs(self::$dsn[$key]);
            } else {
                self::$db[$key] = new PDO(self::$dsn[$key]);
            }
        }
        // Вернуть подключение
        return self::$db[$key];
    }
}
```

В листинге 10.41 методу `DBCxn::get()` должен передаваться ключ, который определяет используемый элемент `$dsn`. Код метода `get()` несколько усложняется, потому что он должен обрабатывать переменное количество аргументов конструктора `PDO`. Некоторым базам данных (например, SQLite) достаточно одного аргумента; другим могут потребоваться два, три или четыре аргумента. В листинге 10.41 метод `ReflectionClass::newInstanceArgs()` используется для компактного вызова конструктора с передачей аргументов в массиве.

См. также

Документация по методам `PDO::__construct()` и `ReflectionClass::newInstanceArgs()`.

10.16. Программа: база данных сообщений интернет-форума

При хранении и выборке древовидной структуры сообщений интернет-форума необходимо действовать очень внимательно, чтобы сообщения выводились в правильном порядке. Поиск потомков каждого сообщения и построение дерева сообщений легко приводит к построению рекурсивной «паутины» запросов. Обычно пользователи просматривают списки сообщений и читают отдельные сообщения намного чаще, чем публикуют свои сообщения. Дополнительные действия при сохранении нового сообщения в базе данных позволят существенно упростить и повысить эффективность выборки списка сообщений.

Для хранения сообщений будет использоваться таблица со следующей структурой:

```
CREATE TABLE message (
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  posted_on DATETIME NOT NULL,
  author CHAR(255),
  subject CHAR(255),
  body MEDIUMTEXT,
  thread_id INT UNSIGNED NOT NULL,
  parent_id INT UNSIGNED NOT NULL,
  level INT UNSIGNED NOT NULL,
  thread_pos INT UNSIGNED NOT NULL
);
```

Первичный ключ `id` идентифицирует конкретное сообщение. В поле `posted_on` хранятся время и дата публикации сообщения, а в полях `author`, `subject` и `body` — автор, тема и тело сообщения (кто бы мог подумать?). Остальные поля предназначены для хранения информации об иерархических связях между сообщениями. Целочисленное поле `thread_id` определяет конкретное обсуждение, то есть цепочку сообщений. Все сообщения одного обсуждения обладают одинаковым идентификатором `thread_id`. Если сообщение является ответом на другое сообщение, то в поле `parent_id` хранится идентификатор «родительского» сообщения. Поле `level` определяет позицию сообщения в обсуждении. Первое сообщение в потоке имеет уровень 0. Ответы на сообщения этого уровня имеют уровень 1, ответы на сообщения уровня 1 имеют уровень 2 и т. д. В обсуждении может быть много сообщений с одинаковыми значениями `level` и `parent_id`. Например, если кто-то открывает новое обсуждение сообщением о преимуществах BeOS перед CP/M, возмущенные ответы фанатов CP/M будут иметь уровень 1, а поле `parent_id` этих ответов будет содержать идентификатор исходного сообщения.

Последнее поле, `thread_pos`, содержит информацию, которая упрощает вывод сообщений. При выводе все сообщения одного обсуждения упорядочиваются по значению `thread_pos`.

Правила вычисления значений `thread_pos`:

- У первого сообщения в обсуждении `thread_pos = 0`.
- Для нового сообщения *N*, если обсуждение не содержит сообщений с тем же родителем, что у *N*, значение `thread_pos` на 1 больше значения `thread_pos` родителя.
- Для нового сообщения *N*, если в обсуждении имеются сообщения с тем же родителем, что у *N*, значение `thread_pos` на 1 больше максимального значения `thread_pos` по всем сообщениям с тем же родителем.
- После определения `thread_pos` для нового сообщения *N* у всех сообщений в обсуждении, у которых значение `thread_pos` больше либо равно `thread_pos` сообщения *N*, это значение увеличивается на 1 (чтобы освободить место для *N*).

Программа message.php, приведенная в листинге 10.42, сохраняет сообщения в базе данных и вычисляет значение `thread_pos`. Пример вывода показан на рис. 10.4 (с. 371).

Листинг 10.42. message.php

```
$board = new MessageBoard();
$board->go();

class MessageBoard {
    protected $db;
    protected $form_errors = array();
    protected $inTransaction = false;

    public function __construct() {
        set_exception_handler(array($this,'logAndDie'));
```

```
        $this->db = new PDO('sqlite:/tmp/message.db');
        $this->db->setAttribute(PDO::ATTR_ERRMODE,PDO::ERRMODE_EXCEPTION);
    }

    public function go() {
        // Значение $_REQUEST['cmd'] определяет выполняемую команду
        $cmd = isset($_REQUEST['cmd']) ? $_REQUEST['cmd'] : 'show';
        switch ($cmd) {
            case 'read':          // Чтение отдельного сообщения
              $this->read();
              break;
            case 'post':          // Отображение формы для ввода сообщения
              $this->post();
              break;
            case 'save':          // Сохранение отправленного сообщения
              if ($this->valid()) { // Если сообщение прошло проверку,
                  $this->save();    // сохранить его
                  $this->show();    // и вывести список сообщений
              } else {
                  $this->post();    // Иначе снова открыть форму отправки
              }
              break;
            case 'show':          // Вывести список сообщений по умолчанию
            default:
              $this->show();
              break;
        }
    }

    // save() сохраняет сообщение в базе данных
    protected function save() {

        $parent_id = isset($_REQUEST['parent_id']) ?
                     intval($_REQUEST['parent_id']) : 0;

        // Сообщение не должно изменяться в то время,
        // когда мы работаем с ним.
        $this->db->beginTransaction();
        $this->inTransaction = true;

        // Сообщение является ответом?
        if ($parent_id) {
            // Получить обсуждение, уровень и thread_pos
            // родительского сообщения
            $st = $this->db->prepare("SELECT thread_id,level,thread_pos
                                 FROM message WHERE id = ?");
            $st->execute(array($parent_id));
            $parent = $st->fetch();

            // Уровень ответа на 1 больше уровня родителя
            $level = $parent['level'] + 1;

            /* Чему равно максимальное значение  thread_pos среди всех
               сообщений данного обсуждения, имеющих того же родителя? */
            $st = $this->db->prepare('SELECT MAX(thread_pos) FROM message
                    WHERE thread_id = ? AND parent_id = ?');
```

```
            $st->execute(array($parent['thread_id'], $parent_id));
            $thread_pos = $st->fetchColumn(0);

            // Имеются ли существующие ответы на то же
            // родительское сообщение?
            if ($thread_pos) {
                // Это значение thread_pos следует за максимальным
                // из существующих
                $thread_pos++;
            } else {
                // Это первый ответ, поместить его сразу после родителя
                $thread_pos = $parent['thread_pos'] + 1;
            }

            /* Увеличение thread_pos всех сообщений обсуждения, которые
            следуют за этим */
            $st = $this->db->prepare('UPDATE message SET thread_pos = thread_pos
                                + 1 WHERE thread_id = ? AND thread_pos >= ?');
            $st->execute(array($parent['thread_id'], $thread_pos));

            // Новое сообщение должно сохраняться со значением thread_id
            // своего родителя
            $thread_id = $parent['thread_id'];
        } else {
            // Сообщение не является ответом, а значит,
            // открывает новое обсуждение
            $thread_id = $this->db->query('SELECT MAX(thread_id) + 1 FROM
                                        message')->fetchColumn(0);
            // Если строки данных еще не существуют,
            // начать со значения thread_id = 1
            if (! $thread_id) {
                $thread_id = 1;
            }
            $level = 0;
            $thread_pos = 0;
        }

        /* Вставка сообщения в базу данных. Использование prepare()
           и execute() обеспечивает необходимое экранирование всех полей */
        $st = $this->db->prepare("INSERT INTO message (id,thread_id,parent_id,
                        thread_pos,posted_on,level,author,subject,body)
                        VALUES (?,?,?,?,?,?,?,?,?)");
        $st->execute(array(null,$thread_id,$parent_id,$thread_pos,
                           date('c'),$level,$_REQUEST['author'],
                           $_REQUEST['subject'],$_REQUEST['body']));

        // Закрепление всех операций
        $this->db->commit();
        $this->inTransaction = false;
    }

    // show() выводит список всех сообщений
    protected function show() {
        print '<h2>Message List</h2><p>';

        /* Сообщения упорядочиваются по обсуждению (thread_id)
```

```
            и позиции в обсуждении (thread_pos) */
        $st = $this->db->query("SELECT id,author,subject,LENGTH(body)
              AS body_length,posted_on,level FROM message
                       ORDER BY thread_id,thread_pos");
        while ($row = $st->fetch()) {
            // Сообщения с уровнем > 0 выводятся с отступами
            print str_repeat(' ',4 * $row['level']);
            // Вывод информации о сообщении со ссылкой для его чтения
            $when = date('Y-m-d H:i', strtotime($row['posted_on']));
            print "<a href='" . htmlentities($_SERVER['PHP_SELF']) .
            "?cmd=read&amp;id={$row['id']}'>" .
            htmlentities($row['subject']) . '</a> by ' .
            htmlentities($row['author']) . ' @ ' .
             htmlentities($when) .
             " ({$row['body_length']} bytes) <br/>";
        }

        // Для отправки сообщений, которые не являются ответами
        print "<hr/><a href='" .
              htmlentities($_SERVER['PHP_SELF']) .
              "?cmd=post'>Start a New Thread</a>";
    }

    // read() выводит отдельное сообщение
    public function read() {

        /* Убедиться в том, что переданный идентификатор является
           целым числом и действительно соответствует сообщению */
        if (! isset($_REQUEST['id'])) {
            throw new Exception('No message ID supplied');
        }
        $id = intval($_REQUEST['id']);
        $st = $this->db->prepare("SELECT author,subject,body,posted_on
                                  FROM message WHERE id = ?");
        $st->execute(array($id));
        $msg = $st->fetch();
        if (! $msg) {
            throw new Exception('Bad message ID');
        }

        /* Не отображать разметку HTML, введенную пользователем,
           но вывести символы новой строки как разрывы строк HTML */
        $body = nl2br(htmlentities($msg['body']));

        // Вывести сообщения со ссылками для ответа
        // и возврата к списку  сообщений
        $self = htmlentities($_SERVER['PHP_SELF']);
        $subject = htmlentities($msg['subject']);
        $author = htmlentities($msg['author']);
        print<<<_HTML_
<h2>$subject</h2>
<h3>by $author</h3>
<p>$body</p>
<hr/>
<a href="$self?cmd=post&parent_id=$id">Reply</a>
<br/>
```

```
<a href="$self?cmd=list">List Messages</a>
_HTML_;
    }

    // post() отображает форму для отправки сообщений
    public function post() {
        $safe =  array();
        foreach (array('author','subject','body') as $field) {
            // Экранирование в значениях полей по умолчанию
            if (isset($_POST[$field])) {
                $safe[$field] = htmlentities($_POST[$field]);
            } else {
                $safe[$field] = '';
            }
            // Сообщения об ошибках выводятся красным цветом
            if (isset($this->form_errors[$field])) {
                $this->form_errors[$field] = '<span style="color: red">' .
                   $this->form_errors[$field] . '</span><br/>';
            } else {
                $this->form_errors[$field] = '';
            }
        }

        // Сообщение является ответом?
        if (isset($_REQUEST['parent_id']) &&
        $parent_id = intval($_REQUEST['parent_id'])) {
        // Передать parent_id при отправке данных формы
        $parent_field =
            sprintf('<input type="hidden" name="parent_id" value="%d" />',
                    $parent_id);
        // Если тема не указана, использовать тему родительского сообщения
        if (! strlen($safe['subject'])) {
             $st = $this->db->prepare('SELECT subject FROM message WHERE
                           id = ?');
             $st->execute(array($parent_id));
             $parent_subject = $st->fetchColumn(0);
            /* Вставить префикс 'Re: ' в тему родительского сообщения,
               если она существует и еще не имеет префикса 'Re:' */
            $safe['subject'] = htmlentities($parent_subject);
            if ($parent_subject && (! preg_match('/^re:/i',$parent_subject))) {
                $safe['subject'] = "Re: {$safe['subject']}";
            }
          }
        } else {
            $parent_field = '';
        }

    // Отображение формы с ошибками и значениями по умолчанию
    $self = htmlentities($_SERVER['PHP_SELF']);
    print<<<_HTML_
<form method="post" action="$self">
<table>
<tr>
 <td>Your Name:</td>
 <td>{$this->form_errors['author']}
     <input type="text" name="author" value="{$safe['author']}" />
```

```
</td>
<tr>
 <td>Subject:</td>
 <td>{$this->form_errors['subject']}
     <input type="text" name="subject" value="{$safe['subject']}" />
</td>
<tr>
 <td>Message:</td>
 <td>{$this->form_errors['body']}
     <textarea rows="4" cols="30" wrap="physical"
               name="body">{$safe['body']}</textarea>
</td>
<tr><td colspan="2"><input type="submit" value="Post Message" /></td></tr>
</table>
$parent_field
<input type="hidden" name="cmd" value="save" />
</form>
_HTML_;
    }

    // validate() проверяет, что каждое поле содержит данные
    public function valid() {
        $this->form_errors = array();
        if (! (isset($_POST['author']) && strlen(trim($_POST['author'])))) {
            $this->form_errors['author'] = 'Please enter your name.';
        }
        if (! (isset($_POST['subject']) && strlen(trim($_POST['subject'])))) {
            $this->form_errors['subject'] = 'Please enter a message subject.';
        }
        if (! (isset($_POST['body']) && strlen(trim($_POST['body'])))) {
            $this->form_errors['body'] = 'Please enter a message body.';
        }

        return (count($this->form_errors) == 0);
    }

    public function logAndDie(Exception $e) {
        print 'ERROR: ' . htmlentities($e->getMessage());
        if ($this->db && $this->db->inTransaction()) {
            $this->db->rollback();
        }
        exit();
    }
}
```

Чтобы правильно организовать параллельный доступ к данным, метод `save()` должен получить монопольный доступ к таблице `msg` между моментами начала вычисления `thread_pos` нового сообщения и фактической вставки нового сообщения в базу данных. Для этого мы использовали методы `beginTransaction()` и `commit()` уровня PDO. Обратите внимание: `logAndDie()`, обработчик исключения, отменяет транзакцию при возникновении ошибок в ходе ее выполнения. Хотя PDO всегда вызывает `rollback()` в конце сценария, если в нем была начата транзакция, явное включение вызова в `logAndDie()` более наглядно демонстрирует суть происходящего для читателя кода.

Рис. 10.4. База данных сообщений с разделением на обсуждения

Поле `level` может использоваться при выводе сообщений для ограничения загружаемой информации. Если глубина обсуждений становится слишком большой, такие ограничения предотвратят чрезмерное разрастание страниц. Листинг 10.43 показывает, как вывести только первое сообщение из каждого обсуждения, вместе с ответами на это первое сообщение.

Листинг 10.43. Ограничение глубины обсуждения

```
$st = $this->db->query(
    "SELECT * FROM message WHERE level <= 1 ORDER BY thread_id,thread_pos");
while ($row = $st->fetch()) {
    // Вывод каждого сообщения
}
```

Если вы хотите организовать форум на своем сайте, возможно, вам стоит воспользоваться одним из существующих пакетов PHP, например FUDForum или Vanilla Forums.

10.17. Использование Redis

Задача

Требуется использовать хранилище пар «ключ/значение» Redis из программы PHP.

Решение

Если вы можете устанавливать расширения PECL, установите расширение redis и используйте его:

```
$redis = new Redis();
$redis->connect('127.0.0.1');
$redis->set('counter', 0);
$redis->incrBy('counter', 7);
$counter = $redis->get('counter');
print $counter;
```

Если это невозможно, используйте библиотеку Predis:

```
require 'Predis/Autoloader.php';
Predis\Autoloader::register();

$redis = new Predis\Client(array('host' => '127.0.0.1'));
$redis->set('counter', 0);
$redis->incrBy('counter', 7);
$counter = $redis->get('counter');
print $counter;
```

Комментарий

Хотя расширение redis и библиотека Predis различаются по способу установки и подключения к серверу, между ними много общего. В обоих случаях вы получаете объект, представляющий подключение к серверу Redis (или пулу серверов Redis), причем методы этого объекта соответствуют различным операциям, которые могут передаваться серверу (серверам).

Для установки расширения redis используется команда *pecl*:

```
pecl install redis
```

Для установки Predis используется *pear*:

```
pear channel-discover pear.nrk.io
pear install nrk/Predis
```

Если вы используете менеджер пакетов Composer, установите Predis с использованием строки зависимостей `"predis/predis"`.

См. также

Сервер Redis также доступен в Интернете. За информацией о расширении redis и его документацией обращайтесь на страницу PECL. Документация по библиотеке Predis доступна в GitHub. Дополнительную информацию о Composer можно найти на сайте Composer.

11 Сеансы и долгосрочное хранение данных

11.0. Введение

По мере развития веб-технологий необходимость сохранения текущего состояния стала типичным требованием. Отслеживание состояния, то есть информации конкретного посетителя в процессе его перемещения по сайту, стало настолько привычным делом, что теперь его считают чем-то само собой разумеющимся.

С учетом распространения веб-приложений, хранящих информацию о своих посетителях (покупательские корзины, дистанционное банковское обслуживание, персонализация домашних страниц порталов, сайты социальных сетей), трудно представить нашу повседневную деятельность в Интернете без приложений с поддержкой состояния.

Протокол HTTP, используемый веб-серверами и клиентами для взаимодействия друг с другом, был спроектирован как протокол без состояния. Однако PHP предоставляет набор удобных функций управления состоянием, которые существенно упрощают реализацию отслеживания состояния. В этой главе основное внимание уделяется нескольким полезным приемам, о которых следует помнить при разработке приложений с состоянием.

Основной целью сеанса (session) является поддержание состояния, относящегося к конкретному посетителю, между запросами. В некоторых приложениях также необходим эквивалентный механизм эффективного хранения состояния, не относящегося к конкретному посетителю, в течение некоторого времени на стороне сервера. Это называется долгосрочным хранением данных.

В Рецепте 11.1 рассматривается сеансовый модуль PHP, позволяющий легко связать долгосрочные данные с посетителем в процессе его перемещения по сайту. В Рецептах 11.2 и 11.3 рассказано о перехвате сеанса, устранении сеансовых уязвимостей и о том, как избежать их.

Сеансовые данные по умолчанию хранятся в неструктурированных файлах в каталоге /tmp сервера. Рецепты 11.4 и 11.5 объясняют, как сохранить сеансовые данные в другом хранилище (например, Memcached или базе данных), а также обсуждаются достоинства и недостатки разных решений.

Рецепт 11.6 демонстрирует возможности использования общей памяти для задач, не ограничивающихся хранением сеансовых данных, а в Рецепте 11.7 продемонстрирована методика хранения сводной информации, полученной в результате анализа журнальных файлов.

11.1. Отслеживание сеанса

Задача

Требуется поддерживать информацию о пользователе во время его перемещения по сайту.

Решение

Воспользуйтесь сеансовым модулем. Функция `session_start()` инициализирует сеанс, а обращение к элементу суперглобального массива `$_SESSION` сообщает PHP о необходимости отслеживания соответствующей переменной:

```
session_start();
if (! isset($_SESSION['visits'])) {
    $_SESSION['visits'] = 0;
}
$_SESSION['visits']++;
print 'You have visited here '.$_SESSION['visits'].' times.';
```

Комментарий

Сеансовый модуль отслеживает пользователей, передавая им cookie со случайно сгенерированным идентификатором сеанса.

По умолчанию PHP сохраняет сеансовые данные в файлах в каталоге /tmp на вашем сервере. Каждый сеанс сохраняется в отдельном файле. Чтобы изменить каталог, в котором хранятся файлы, укажите новый каталог в конфигурационной директиве `session.save_path` в php.ini или воспользуйтесь функцией `ini_set()`. Также можно вызвать функцию `session_save_path()` с новым каталогом для смены каталога, но это необходимо сделать перед началом сеанса или обращением к каким-либо сеансовым переменным.

Чтобы сеанс автоматически начинался при каждом запросе, задайте директиве `session.auto_start` в файле php.ini значение 1. В режиме `session.auto_start` вызывать `session_start()` не обязательно; если у вас есть возможность изменить файл php.ini — это самое простое решение.

Если конфигурационная директива `session.use_trans_sid` включена и PHP обнаруживает, что пользователь не принимает cookie с идентификатором сеанса, то идентификатор сеанса автоматически добавляется во все URL-адреса и формы. Для примера возьмем код вывода URL:

```
print '<a href="train.php">Take the A Train</a>';
```

Если сеансы включены, но пользователь не принимает cookie, то браузеру будет передана строка вида:

```
<a href="train.php?PHPSESSID=2eb89f3344520d11969a79aea6bd2fdd">⏎
Take the A Train</a>
```

В этом примере `PHPSESSID` — имя, а `2eb89f3344520d11969a79aea6bd2fdd` — идентификатор сеанса. PHP добавляет их в URL, чтобы они передавались следующей странице. В формы включается скрытый элемент для передачи идентификатора сеанса.

Вследствие разнообразных проблем безопасности, связанных с встраиванием идентификаторов сеансов в URL, по умолчанию такое поведение отключается. Чтобы разрешить прозрачное включение идентификаторов сеансов в URL, необходимо включить директиву `session.use_trans_sid` в файле `php.ini` или вызовом `ini_set('session.use_trans_sid', true)` в сценариях перед началом сеанса.

Директива `session.use_trans_sid` удобна, но она может создать проблемы, связанные с безопасностью. Так как URL-адреса содержат идентификаторы сеансов, их распространение позволяет любому обладателю URL выдать себя за пользователя, которому был присвоен идентификатор сеанса. Пользователь, который скопировал URL из браузера и вставил его в сообщение электронной почты, разосланное друзьям, невольно разрешает всем этим друзьям (а также всем, кому будет переслано это сообщение) посетить сайт, выдавая себя за него.

Что еще хуже, когда пользователь щелкает на ссылке, которая направляет его на другой сайт, браузер пользователя передает URL-адрес со встроенным идентификатором сеанса как URL источника запроса на внешний сайт. Даже если владельцы внешнего сайта не производят сознательный поиск по URL источников запросов, журналы источников часто непреднамеренно становятся доступными для поисковых систем. Введите строку «PHPSESSID referer» в своей любимой поисковой системе, и вы наверняка найдете журналы со встроенными идентификаторами сеансов.

Кроме того, к перенаправлениям с заголовком `Location` автоматическая модификация не применяется, поэтому вам придется добавлять идентификатор сеанса самостоятельно при помощи константы `SID`:

```
$redirect_url = 'http://www.example.com/airplane.php';
if (defined('SID') && (!isset($_COOKIE[session_name()]))) {
    $redirect_url .= '?' . SID;
}

header("Location: $redirect_url");
```

Функция `session_name()` возвращает имя cookie, в котором хранится идентификатор сеанса; этот код присоединяет константу `SID` к `$redirect_url`, если константа определена и данные cookie сеанса не установлены.

См. также

Документация по функциям `session_start()` и `session_save_path()`. Сеансовый модуль содержит ряд конфигурационных директив для управления различными параметрами сеансов (например, продолжительностью сеансов и способом кэширования). Эти параметры описаны в разделе «Sessions» электронной документации.

11.2. Предотвращение перехвата сеанса

Задача

Требуется помешать атакующему получить доступ к сеансу другого пользователя.

Решение

Разрешите передачу идентификаторов сеансов только через cookie и сгенерируйте дополнительный маркер сеанса, передаваемый через URL. К сеансу смогут обратиться только запросы, содержащие действительный идентификатор сеанса и действительный маркер сеанса:

```
ini_set('session.use_only_cookies', true);
session_start();

$salt     = 'YourSpecialValueHere';
$tokenstr = strval(date('W')) . $salt;
$token    = md5($tokenstr);

if (!isset($_REQUEST['token']) || $_REQUEST['token'] != $token) {
    // Запрос учетных данных
    exit;
}

$_SESSION['token'] = $token;
output_add_rewrite_var('token', $token);
```

Комментарий

В этом примере для создания маркера с автоматическим сдвигом текущий номер недели объединяется с затравкой (salt), выбранной пользователем. В этом случае маркер остается действительным в течение разумного промежутка времени. Затравка предотвращает возможность вычисления хеша MD5 даты, относящейся

к далекому будущему, и использования его для продления сеанса. Не зная выбранную затравку, злоумышленник не сможет легко получить действительный маркер. Затем проверяется присутствие маркера в запросе, и если он не найден — запрашиваются новые учетные данные. Если же маркер найден, он должен добавляться в генерируемые ссылки. Функция `output_add_rewrite_var()` легко справляется с этой задачей.

Учтите, что этот механизм не помешает атакующему, способному перехватывать весь трафик между пользователем и сервером (например, по незащищенной сети WiFi). Для предотвращения атак такого рода лучше всего использовать протокол SSL.

См. также

Рецепты 18.1 и 11.3 — дополнительная информация о повторном генерировании идентификаторов для предотвращения фиксации сеанса.

11.3. Предотвращение фиксации сеанса

Задача

Требуется защитить приложение от атак фиксации сеанса, при которых атакующий заставляет пользователя использовать заранее определенный идентификатор сеанса.

Решение

Потребуйте использования сеансовых cookie без присоединения идентификаторов к URL, с частым генерированием нового идентификатора сеанса:

```
ini_set('session.use_only_cookies', true);
session_start();
if (!isset($_SESSION['generated'])
    || $_SESSION['generated'] < (time() - 30)) {
    session_regenerate_id();
    $_SESSION['generated'] = time();
}
```

Комментарий

Сначала этот код включает сеансовое поведение PHP, при котором используются только cookie. Это делается для того, чтобы PHP игнорировал идентификатор сеанса при включении его в URL.

После начала сеанса присваивается значение переменной, в которой будет отслеживаться последнее время генерирования идентификатора сеанса. Требуя

регулярного генерирования нового значения (каждые 30 секунд в этом примере), мы существенно сокращаем возможности атакующего по получению действительного идентификатора сеанса.

Сочетание этих двух методов практически устраняет риск фиксации сеанса. Атакующему будет нелегко получить действительный идентификатор сеанса, потому что он так часто изменяется, а поскольку идентификаторы могут передаваться только в cookie, атака на базе URL становится невозможной. Наконец, благодаря включению директивы `session.use_only_cookies` сеансовые cookie не останутся в истории браузера или журналах источников запросов сервера.

См. также

«Session Fixation Vulnerability in Web-based Applications» (*http://www.acros.si/papers/session_fixation.pdf*); Рецепт 18.1 — информация о повторном генерировании идентификаторов сеансов при повышении привилегий.

11.4. Хранение сеансовых данных в Memcached

Задача

Требуется сохранить сеансовые данные в быстром хранилище, с которым могут работать несколько веб-серверов.

Решение

Воспользуйтесь обработчиком, встроенным в расширение `memcached`, для хранения сеансовых данных на одном или нескольких серверах Memcached. С установленным расширением `memcached` задайте конфигурационной директиве `session.save_handler` значение `memcached`, а затем задайте в директиве `session.save_path` хост и порт вашего сервера Memcached. Например, если сервер Memcached работает на порте 11211 хоста 10.5.7.12, задайте `session.save_path` значение `10.5.7.12:11211`. Если вы используете несколько серверов Memcached, включите в `session.save_path` список значений *хост*`:`*порт*, разделенных запятыми.

После задания директив `session.save_handler` и `session.save_path`, сообщающих PHP, как следует хранить сеансовую информацию в Memcached, вам ничего не придется дополнительно делать в коде, использующем `$_SESSION`. Так как подсистема хранения сеансовых данных работает прозрачно, достаточно изменить конфигурацию, и она заработает.

Если вы используете схему согласованного хеширования с несколькими серверами Memcached для распределения значений по серверам, задайте в конфигурационной директиве `memcached.sess_consistent_hash` значение `on`. Это гарантирует,

что сеансовые данные также будут распределяться между множественными серверами Memcached.

Обратите внимание: кроме расширения PHP `memcached` также существует расширение `memcache` (без `d` на конце). В нем также имеется встроенный обработчик сеансовых данных. Чтобы использовать его, задайте `session.save_handler` значение `memcache`. Конфигурационная директива `session.save_path` используется для определения серверов Memcached, но они используют несколько иной синтаксис, чем расширение `memcached`. Перед именем хоста указывается префикс соответствующего протокола (например, `tcp://`), и вы можете добавить пары «имя/значение» по аналогии со строками запроса для задания любых параметров, принимаемых `Memcache::addServer()`. Например, `session.save_path` можно задать значение `tcp://10.5.7.12:11211?weight=3,tcp://10.5.7.13:11211?weight=5`, которое определяет два сервера, 10.5.7.12 и 10.5.7.12, работающих на порте 11211, но имеющих разные веса.

Оба расширения могут хранить сеансовые данные в Memcache. Расширение `memcached` поддерживает различные схемы сжатия для хранения больших блоков данных в Memcache. Расширение `memcache` обладает меньшей функциональностью, но не зависит от внешних библиотек.

Следует учесть, что сам сервер Memcached не сохраняет хранимые данные между перезапусками — данные только хранятся в памяти во время работы сервера. Это означает, что сеансовые данные в Memcached могут пропасть, если на сервере Memcached произойдет сбой или он будет перезапущен.

См. также

Документация о настройке расширений `memcached` и `memcache`; информация о Memcached.

11.5. Хранение сеансовых данных в базе данных

Задача

Требуется хранить сеансовые данные в базе данных вместо файлов. Если несколько веб-серверов имеют доступ к одной базе данных, то сеансовые данные будут доступны для всех серверов.

Решение

Используйте класс в сочетании с функцией `session_set_save_handler()`, чтобы определить средства управления сеансом с поддержкой баз данных. Например,

в листинге 11.1 представлен класс, использующий PDO для хранения сеансовой информации в таблице базы данных.

Листинг 11.1. Обработчик сеансовых данных с поддержкой базы данных

```
/** Реализация SessionHandlerInterface обязательна для PHP 5.4
 * и приведет к ошибкам в предыдущих версиях.
 */
class DBHandler implements SessionHandlerInterface {

    protected $dbh;

    public function open($save_path, $name) {
        try {
            $this->connect($save_path, $name);
            return true;
        } catch (PDOException $e) {
            return false;
        }
    }

    public function close() {
        return true;
    }

    public function destroy($session_id) {
        $sth = $this->dbh->prepare("DELETE FROM sessions WHERE session_id = ?");
        $sth->execute(array($session_id));
        return true;
    }

    public function gc($maxlifetime) {
        $sth = $this->dbh->prepare("DELETE FROM sessions WHERE last_update < ?");
        $sth->execute(array(time() - $maxlifetime));
        return true;
    }

    public function read($session_id) {
        $sth = $this->dbh->prepare("SELECT session_data FROM sessions WHERE
                                    session_id = ?");
        $sth->execute(array($session_id));
        $rows = $sth->fetchAll(PDO::FETCH_NUM);
        if (count($rows) == 0) {
            return '';
        } else {
            return $rows[0][0];
        }
    }

    public function write($session_id, $session_data) {
        $now = time();
        $sth = $this->dbh->prepare("UPDATE sessions SET session_data = ?,
                                    last_update = ? WHERE session_id = ?");
        $sth->execute(array($session_data, $now, $session_id));
        if ($sth->rowCount() == 0) {
            $sth2 = $this->dbh->prepare('INSERT INTO sessions (session_id,
                                         session_data, last_update)
                                         VALUES (?,?,?)');
            $sth2->execute(array($session_id, $session_data, $now));
```

```
        }
    }

    public function createTable($save_path, $name, $connect = true) {
        if ($connect) {
            $this->connect($save_path, $name);
        }
        $sql=<<<_SQL_
CREATE TABLE sessions (
  session_id VARCHAR(64) NOT NULL,
  session_data MEDIUMTEXT NOT NULL,
  last_update TIMESTAMP NOT NULL,
  PRIMARY KEY (session_id)
)
_SQL_;
        $this->dbh->exec($sql);
    }

    protected function connect($save_path) {
        /* Поиск имени пользователя и пароля в DSN
           как параметров "строки запроса" */
        $parts = parse_url($save_path);
        if (isset($parts['query'])) {
            parse_str($parts['query'], $query);
            $user = isset($query['user']) ? $query['user'] : null;
            $password = isset($query['password']) ? $query['password'] : null;
            $dsn = $parts['scheme'] . ':';
            if (isset($parts['host'])) {
                $dsn .= '//' . $parts['host'];
            }
            $dsn .= $parts['path'];
            $this->dbh = new PDO($dsn, $user, $password);
        } else {
            $this->dbh = new PDO($save_path);
        }
        $this->dbh->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
        // Очень простой способ создания таблицы sessions,
        // если она не существует.
        try {
            $this->dbh->query('SELECT 1 FROM sessions LIMIT 1');
        } catch (Exception $e) {
            $this->createTable($save_path, NULL, false);
        }
    }
}
```

Комментарий

Одним из самых полезных аспектов сеансового модуля является его абстракция сохранения сеансовых данных. Функция `session_set_save_handler()` приказывает PHP использовать разные функции для разных операций с сеансовыми данными (таких как сохранение сеанса и сохранение сеансовых данных).

В PHP 5.4 и выше `session_set_save_handler()` передается экземпляр класса, реализующий интерфейс `SessionHandlerInterface`. В более ранних версиях явно

определенного интерфейса не было, но определяемые методы были теми же: открытые методы `open()`, `close()`, `destroy()`, `gc()`, `read()` и `write()` вызывались из внутреннего кода обработки сеансовых данных PHP при необходимости. Чтобы использовать этот обработчик, создайте объект класса и передайте его `session_set_save_handler()`:

```
include __DIR__ . '/db.php';
ini_set('session.save_path', 'sqlite:/tmp/sessions.db');
session_set_save_handler(new DBHandler);

session_start();
if (! isset($_SESSION['visits'])) {
    $_SESSION['visits'] = 0;
}
$_SESSION['visits']++;
print 'You have visited here '.$_SESSION['visits'].' times.';
```

В этом коде предполагается, что класс `DBHandler` определяется в файле с именем db.php, находящемся в том же каталоге. После того, как в директиве `session.save_path` будет задана строка PDO DSN, описывающая базу данных с таблицей `sessions`, вызов `session_set_save_handler(new DBHandler)` — все, что необходимо для подключения обработчика к PHP. С этого момента код, использующий сеансовые данные, ничем не отличается от кода, использующего стандартный обработчик PHP.

В листинге 11.1 определяется дополнительный открытый метод `createTable()`, который предоставляет удобный способ создания таблицы для хранения сеансовых данных. Метод `connect()` вызывает его, если не сможет найти таблицу `sessions`.

Функции `createTable()` и `open()` также получают имя сеанса в отдельной переменной. По умолчанию действует значение `PHPSESSID`; PHP использует его как имя cookie при записи cookie с идентификатором сеанса. Если вам понадобится различать несколько сеансов с разными именами, ассоциированных с одним пользователем, измените класс `DBHandler`, встроив аргумент `$name` в имя таблицы базы данных или в дополнительный столбец таблицы `sessions`.

См. также

Документация по функции `session_set_save_handler()` и интерфейсу `SessionHandlerInterface`.

11.6. Хранение произвольных данных в общей памяти

Задача

Требуется предоставить доступ к блоку данных для всех серверных процессов через общую память.

Решение

Если данные должны совместно использоваться только процессами PHP, воспользуйтесь расширением APC, как показано в Рецепте 5.6. Если вы хотите открыть доступ к данным также и для других процессов, воспользуйтесь классом `pc_Shm` (листинг 11.2).

Например, для сохранения строки в общей памяти используется метод `pc_Shm::write()`, которому при вызове передаются ключ, длина и значение:

```
$shm = new pc_Shm();
$secret_code = 'land shark';
$shm->write('mysecret', strlen($secret_code), $secret_code);
```

Комментарий

Класс `pc_Shm` представлен в листинге 11.2.

Листинг 11.2. Хранение произвольных данных в общей памяти

```
class pc_Shm {

    protected $tmp;
    public function __construct($tmp = '') {
        if (!function_exists('shmop_open')) {
            trigger_error('pc_Shm: shmop extension is required.', E_USER_ERROR);
            return;
        }

        if ($tmp != '' && is_dir($tmp) && is_writable($tmp)) {
            $this->tmp = $tmp;
        } else {
            $this->tmp = '/tmp';
        }
    }

    public function read($id, $size) {
        $shm = $this->open($id, $size);
        $data = shmop_read($shm, 0, $size);
        $this->close($shm);
        if (!$data) {
            trigger_error('pc_Shm: could not read from shared memory block',
                          E_USER_ERROR);
            return false;
        }
        return $data;
    }

    public function write($id, $size, $data) {
        $shm = $this->open($id, $size);
        $written = shmop_write($shm, $data, 0);
        $this->close($shm);
        if ($written != strlen($data)) {
            trigger_error('pc_Shm: could not write entire length of data',
                          E_USER_ERROR);
```

```
            return false;
        }
        return true;
    }

    public function delete($id, $size) {
        $shm = $this->open($id, $size);
        if (shmop_delete($shm)) {
            $keyfile = $this->getKeyFile($id);
            if (file_exists($keyfile)) {
                unlink($keyfile);
            }
        }
        return true;
    }

    protected function open($id, $size) {
        $key = $this->getKey($id);
        $shm = shmop_open($key, 'c', 0644, $size);
        if (!$shm) {
            trigger_error('pc_Shm: could not create shared memory segment',
                          E_USER_ERROR);
            return false;
        }
        return $shm;
    }

    protected function close($shm) {
        return shmop_close($shm);
    }

    protected function getKey($id) {
        $keyfile = $this->getKeyFile($id);
        if (! file_exists($keyfile)) {
            touch($keyfile);
        }
        return ftok($keyfile, 'R');
    }

    protected function getKeyFile($id) {
        return $this->tmp . DIRECTORY_SEPARATOR . 'pcshm_' . $id;
    }
}
```

Так как `pc_Shm` использует для работы с общей памятью стандартные системные функции, другие программы (на каком бы языке они ни были написаны) также смогут обращаться к этим данным. Например, короткая программа на языке C из листинга 11.3 может читать данные, записанные `pc_Shm`.

Листинг 11.3. Чтение данных в общей памяти из кода C

```
#include <sys/ipc.h>
#include <sys/shm.h>

#include <stdio.h>

int main(int argc, char **argv) {
```

```
    char *id;
    size_t size;

    if (argc != 3) {
        fprintf(stderr, "Usage: %s ID SIZE\n", argv[0]);
        return -1;
    }

    id = argv[1];
    size = atoi(argv[2]);

    char *path;
    asprintf(&path, "/tmp/pcshm_%s", id);
    key_t token = ftok(path, (int) 'R');
    int shmid = shmget(token, size, 0);
    void *ptr = shmat(shmid, 0, SHM_RDONLY);
    printf("%*s\n", (int) size, (char *) ptr);
    shmdt(ptr);
    free(path);
}
```

Откомпилируйте эту программу и вызовите ее с аргументами `mysecret` и 10 (или любой другой достаточно длительной задержкой); программа выведет данные, записанные в общую память кодом PHP.

Важно помнить, что в отличие от пар «ключ/значение» в обычных массивах PHP, функции `shmop` должны выделять блок памяти конкретного размера, который, как ожидается, будут занимать хранимые данные. По этой причине операциям чтения и записи должен передаваться размер блока.

См. также

Рецепт 5.6 — дополнительная информация о применении APC для организации совместного доступа к памяти из процессов PHP; документация по функциям PHP `shmop`.

11.7. Кэширование вычисленных результатов в сводных таблицах

Задача

Требуется собрать статистику по таблицам с журналами, размеры которых слишком велики для эффективной обработки запросов в реальном времени.

Решение

Создайте таблицу для хранения сводных данных по полной таблице. Обращайтесь с запросами к сводной таблице, чтобы генерировать отчеты *почти* в реальном времени.

Комментарий

Допустим, вы ведете журнал запросов, использованных посетителями в поисковых системах типа Google или Yahoo! для выхода на ваш сайт, и храните данные этих запросов в MySQL. Таблица с условиями поиска имеет следующую структуру:

```
CREATE TABLE searches
(
  searchterm    VARCHAR(255) NOT NULL,  -- условие поиска, определяемое
                                        -- разбором HTTP_REFERER
  dt            DATETIME NOT NULL,      -- дата запроса
  source        VARCHAR(15) NOT NULL    -- сайт, на котором выполнялся поиск
);
```

Если вам повезет и ваш сайт будет получать тысячи и десятки тысяч посещений в час с основных поисковых систем, таблица `searches` за несколько месяцев может разрастись до неуправляемых размеров.

Возможно, вы предпочтете генерировать отчеты, демонстрирующие закономерности привлечения посетителей из разных поисковых систем; это поможет вам определить, в какой поисковой системе следует приобретать рекламу.

Создайте сводную таблицу с интересующими вас данными, а затем ежечасно выдавайте запрос к полному набору данных. Результат запроса сохраняется в сводной таблице для ускоренной выборки при построении отчетов. Структура сводной таблицы может выглядеть так:

```
CREATE TABLE searchsummary
(
  searchterm    VARCHAR(255) NOT NULL,  -- search term
  source        VARCHAR(15) NOT NULL,   -- site where search was performed
  sdate         DATE NOT NULL,          -- date search performed
  searches      INT UNSIGNED NOT NULL,  -- number of searches
  PRIMARY KEY (searchterm, source, sdate)
);
```

Сценарий построения отчета средствами PDO выдает запрос к таблице `searchsummary`. Если результаты недоступны, то данные берутся из таблицы `searches`, а результат кэшируется в `searchsummary`:

```
$st = $db->prepare('SELECT COUNT(*)
                    FROM
                       searchsummary
                    WHERE
                       sdate = ?');
$st->execute(array(date('Y-m-d', strtotime('yesterday'))));

$row = $st->fetch();

// Совпадения в кэше отсутствуют
if ($row[0] == 0) {
    $st2 = $db->prepare('SELECT
```

```
                        searchterm,
                        source,
                        date(dt) AS sdate,
                        COUNT(*) as searches
                      FROM
                       searches
                      WHERE
                        date(dt) = ?');
    $st2->execute(array(date('Y-m-d', strtotime('yesterday'))));

    $stInsert = $db->prepare('INSERT INTO searchsummary
                            (searchterm,source,sdate,searches)
                            VALUES (?,?,?,?)');
    while ($row = $st2->fetch(PDO::FETCH_NUM)) {
        $stInsert->execute($row);
    }
}
```

При использовании сводной таблицы высокозатратный запрос к полной таблице выполняется только один раз, а все последующие запросы выбирают по одной строке сводных данных на каждое условие поиска.

См. также

Рецепт 10.6 — информация о методах `PDO::prepare()` и `PDOStatement::execute()`.

12 XML

12.0. Введение

XML — популярный формат передачи данных, определения конфигурации и передачи сообщений. Хотя во многих распространенных ситуациях его вытеснил формат JSON, XML по-прежнему играет важную роль в жизни разработчика. Воспользовавшись несколькими расширениями, вы сможете выполнять любые операции чтения и записи XML в коде PHP.

XML предоставляет разработчикам средства структурированной разметки данных с использованием тегов, образующих древовидную иерархию. Вообще говоря, XML можно рассматривать как расширенную версию CSV: формат XML может использоваться для хранения записей, разбитых на поля. Однако в отличие от простого списка данных, разделенных запятыми, XML позволяет наряду с данными включать имена полей, типы и атрибуты.

Также XML может рассматриваться как язык представления документов. Например, эта книга была написана с использованием XML. Она разделена на главы; каждая глава делится на рецепты; каждый рецепт состоит из разделов «Задача», «Решение» и «Комментарий». В каждом разделе в тексте дополнительно выделяются абзацы, таблицы, рисунки и листинги. Аналогичным образом в статье на веб-странице возможно выделить заголовок, преамбулу, информацию об авторах, сам текст, врезки, сопутствующие ссылки и дополнительный контент.

Разметка XML в целом похожа на HTML: в обоих случаях для разметки текста используются теги, заключенные в угловые скобки < и >. Тем не менее разметка XML одновременно и жестче, и свободнее разметки HTML. Он жестче, потому что все контейнерные теги должны явно закрываться. Разметка не может содержать открывающие элементы без соответствующих закрывающих тегов. Она свободнее, потому что разработчик не ограничивается использованием заранее определенного набора тегов, таких как `<a>`, `<img>` и `<h1>`. Вместо этого он может свободно выбирать имена тегов, которые лучше описывают его данные.

Другие принципиальные различия между XML и HTML — учет регистра символов, обязательные кавычки в атрибутах и пропуски. В HTML теги <B> и <b> эквивалентны; в XML это два разных тега. В HTML кавычки, в которые заключаются атрибуты, часто могут опускаться; в XML они обязательны. Таким образом, вы всегда должны использовать синтаксис:

```
<элемент атрибут="значение">
```

Кроме того, парсеры HTML обычно игнорируют пропуски, так что серия из 20 смежных пробелов рассматривается как один пробел. Парсеры XML сохраняют пропуски, если им явно не приказать поступать иначе. Так как все элементы должны закрываться, пустые элементы должны завершаться символами />. Например, в HTML разрыв строки обозначается тегом
, тогда как в XHTML (разметка HTML, которая является действительной по правилам XML) он записывается в виде
[1].

Для документов XML также устанавливается еще одно ограничение. Когда документ XML разбирается в дерево элементов, элемент верхнего уровня называется *корневым* элементом. Подобно тому как у дерева существует только один ствол, документ XML должен иметь ровно один корневой элемент. В приведенном выше примере с книгой это означает, что теги глав должны быть заключены в объединяющий тег книги. Если вы хотите заключить в документ несколько книг, упакуйте их в другой контейнер (например, представляющий книжную полку). Это ограничение распространяется только на корень документа: у деревьев от ствола может отходить много ветвей, и на одной полке могут стоять несколько книг.

Итак, разобравшись с правилами, рассмотрим конкретный пример. Допустим, вы работаете в библиотеке и хотите перевести свою картотеку в формат XML. Начните с базового набора тегов XML:

```
<book>
    <title>PHP Cookbook</title>
    <author>Sklar, David and Trachtenberg, Adam</author>
    <subject>PHP</subject>
</book>
```

В дальнейшем вы сможете добавлять новые элементы или изменять уже существующие. Например, тег автора <author> можно разделить на имя и фамилию или же разрешить множественное вхождение элементов для книги с несколькими авторами.

В PHP входит набор расширений XML, которые:

- работают совместно как единое целое;
- стандартизированы по одной библиотеке XML: libxml2;

[1] Именно по этой причине nl2br() по умолчанию выводит
; этот вывод является XHTML-совместимым.

- полностью соответствуют спецификациям W3C;
- обеспечивают эффективную обработку данных;
- предоставляют необходимые инструменты для практической работы с XML.

Кроме того, в соответствии с основополагающим принципом PHP — простотой создания веб-приложений — существует специальное расширение, которое упрощает чтение и изменение документов XML. Расширение SimpleXML позволяет работать с данными в документах XML как с массивами и объектами, с перебором их в цикле `foreach` и редактированием их «на месте» с присваиванием новых значений переменным.

Первые два рецепта этой главы посвящены разбору разметки XML. В Рецепте 12.1 показано, как записывать XML без применения дополнительных средств. О том, как использовать расширение DOM для стандартизированной записи XML, рассказано в Рецепте 12.2.

Обратная задача — разбор существующей разметки XML — является темой следующих трех рецептов. Материал разделен в соответствии со сложностью и размером разбираемого документа XML. В Рецепте 12.3 рассматривается разбор базовых документов XML. Если вам понадобятся более сложные средства разбора XML, переходите к Рецепту 12.4. Если же ваши документы XML чрезвычайно велики и требуют больших затрат памяти, обратитесь к Рецепту 12.5. Если же вы впервые используете XML и не уверены в том, какой рецепт вам лучше подходит, опробуйте их последовательно — сложность кода возрастает по мере роста требований.

В Рецепте 12.6 рассматривается XPath — стандарт W3C для извлечения конкретной информации из документов XML. Считайте, что это некий аналог регулярных выражений для XML. XPath — одна из самых полезных, но недостаточно часто используемых частей семейства спецификаций XML. Каждый, кто регулярно работает с XML, должен быть знаком с XPath.

Технология XSLT позволяет взять таблицу стилей XSL и преобразовать XML в визуальный вывод. Отделение контента от представления позволяет взять одну таблицу стилей для браузеров, другую — для мобильных телефонов, третью — для печати... и все это без изменения самого контента. Эта тема рассматривается в Рецепте 12.7.

После знакомства с XSLT два следующих рецепта показывают, как организовать передачу информации между PHP и XSLT. В Рецепте 12.8 представлена передача данных из PHP в таблицу стилей XSLT; Рецепт 12.9 показывает, как получить доступ к PHP из таблицы стилей XSLT.

Документ XML, соответствующий структурным правилам XML, называется *синтаксически корректным*. Однако в отличие от языка HTML с конкретным набором элементов и атрибутов, которые должны располагаться в положенных местах, в XML таких ограничений не существует.

И все же в некоторых случаях будет полезно проследить за тем, чтобы документы XML соответствовали спецификации. Это позволяет инструментам (браузе-

рам, программам чтения RSS, вашим собственным сценариям) легко обрабатывать входные данные. Если документ XML подчиняется всем правилам, установленным в спецификации, он называется *действительным*. В Рецепте 12.10 рассказано о том, как проверить документ XML на действительность.

Одно из главных ограничений PHP связано с обработкой наборов символов и кодировок документов. Строки PHP не связываются ни с какой конкретной кодировкой, но все расширения XML требуют ввода в кодировке UTF-8 и выдают данные в UTF-8. Следовательно, если вы должны использовать набор символов, несовместимый с UTF-8, вы должны вручную преобразовать свои данные как перед отправкой расширению XML, так и после их возвращения. В Рецепте 12.11 рассматриваются лучшие способы решения этой задачи.

Глава завершается несколькими рецептами, посвященными чтению и записи некоторых распространенных видов документов XML, а именно RSS и Atom — двух самых популярных форматов синдикации данных, которые также могут пригодиться для передачи других типов данных, включая сообщения в блогах, подкастов и даже данных геопозиционирования.

В книге также рассматриваются *REST-совместимые* API. Эта тема настолько важна, что для нее выделено две самостоятельные главы. В главе 14 вы узнаете, как пользоваться REST-совместимыми API, а глава 15 рассказывает, как создать собственную реализацию REST-совместимого API.

12.1. Генерирование XML в строковом формате

Задача

Требуется сгенерировать разметку XML. Например, вы хотите создать представление данных своей программы в формате XML, чтобы разобрать его в другой программе.

Решение

Переберите данные и выведите их, заключая в соответствующие теги XML:

```
header('Content-Type: text/xml');
print '<?xml version="1.0"?>' . "\n";
print "<shows>\n";

$shows = array(array('name'     => 'Modern Family',
                     'channel'  => 'ABC',
                     'start'    => '9:00 PM',
                     'duration' => '30'),

               array('name'     => 'Law & Order: SVU',
```

```
                    'channel'  => 'NBC',
                    'start'    => '9:00 PM',
                    'duration' => '60'));

foreach ($shows as $show) {
    print "    <show>\n";
    foreach($show as $tag => $data) {
        print "        <$tag>" . htmlspecialchars($data) . "</$tag>\n";
    }
    print "    </show>\n";
}

print "</shows>\n";
```

Комментарий

Ручной вывод разметки XML в основном сводится к многочисленным циклам `foreach` при переборе массивов. Тем не менее о некоторых неочевидных подробностях стоит упомянуть особо. Прежде всего, необходимо вызвать `header()` для назначения правильного заголовка `Content-Type` для документа. Так как вместо HTML передается разметка XML, используется значение `text/xml`.

Затем, в зависимости от настроек конфигурационной директивы `short_open_tag`, попытка вывода объявления XML может привести к непреднамеренной обработке PHP. Так как `<?` из `<?xml version="1.0"?>` является коротким открывающим тегом PHP, для вывода объявления в браузере следует либо отключить директиву, либо выводить строку из PHP. В нашем решении будет использован второй способ.

Наконец, сущности необходимо экранировать. Например, символ `&` в названии сериала `Law & Order` должен быть заменен на `&`. Экранирование данных осуществляется вызовом `htmlspecialchars()`.

Результат выполнения примера из Решения приведен в листинге 12.1.

Листинг 12.1. Программа передач

```
<?xml version="1.0"?>
<shows>
    <show>
        <name>Modern Family</name>
        <channel>ABC</channel>
        <start>9:00 PM</start>
        <duration>30</duration>
    </show>
    <show>
        <name>Law &amp; Order: SVU</name>
        <channel>NBC</channel>
        <start>9:00 PM</start>
        <duration>60</duration>
    </show>
</shows>
```

См. также

Рецепт 12.2 — генерирование XML с использованием DOM; документация по функции `htmlspecialchars()`.

12.2. Генерирование XML с использованием DOM

Задача

Требуется сгенерировать разметку XML, но сделать это более осмысленно по сравнению с простым использованием `print` с циклами.

Решение

Воспользуйтесь расширением DOM для создания объекта `DOMDocument`. После построения документа вызовите `DOMDocument::save()` или `DOMDocument::saveXML()`, чтобы сгенерировать синтаксически корректный документ XML:

```
// Создание нового документа
$dom = new DOMDocument('1.0');

// Создание корневого элемента <book> и присоединение его к документу
$book = $dom->appendChild($dom->createElement('book'));

// Создание элемента title и присоединение его к $book
$title = $book->appendChild($dom->createElement('title'));

// Назначение текста и атрибута cover элемента $title
$title->appendChild($dom->createTextNode('PHP Cookbook'));
$title->setAttribute('edition', '3');

// Создание и присоединение элементов author к $book
$sklar = $book->appendChild($dom->createElement('author'));
// Создание и присоединение текста для каждого элемента
$sklar->appendChild($dom->createTextNode('Sklar'));

$trachtenberg = $book->appendChild($dom->createElement('author'));
$trachtenberg->appendChild($dom->createTextNode('Trachtenberg'));

// Вывод отформатированной версии документа DOM в формате XML
$dom->formatOutput = true;
echo $dom->saveXML();

<?xml version="1.0"?>
<book>
  <title edition="3">PHP Cookbook</title>
  <author>Sklar</author>
  <author>Trachtenberg</author>
</book>
```

Комментарий

Вызовы методов DOM происходят по определенной схеме. Вы создаете объект как элемент или текстовый узел, добавляете и устанавливаете любые необходимые атрибуты, а затем присоединяете его к дереву в нужной точке.

Прежде чем создавать элементы, создайте новый документ с передачей в его единственном аргументе версии XML:

```
$dom = new DOMDocument('1.0');
```

Теперь создайте новые элементы, принадлежащие документу. Несмотря на связь с конкретным документом, узлы включаются в дерево документа только после присоединения:

```
$book_element = $dom->createElement('book');
$book = $dom->appendChild($book_element);
```

Здесь создается новый элемент `book`, который присваивается объекту `$book_element`. Для создания корня документа `$book_element` присоединяется как потомок `$dom`. Результат, `$book`, обозначает конкретный элемент и его положение в объекте DOM.

Все узлы создаются вызовом методов для `$dom`. После того как узел будет создан, его можно присоединить к любому элементу дерева. Элемент, для которого вызывается метод `appendChild()`, определяет, где именно в дереве будет находиться узел. В предыдущем примере `$book_element` присоединяется к `$dom`. Элемент, присоединяемый к `$dom`, является узлом верхнего уровня, или корневым узлом.

Новый дочерний элемент также можно присоединить к `$book`. Так как `$book` является потомком по отношению к `$dom`, новый элемент становится «внуком» `$dom`:

```
$title_element = $dom->createElement('title');
$title = $book->appendChild($title_element);
```

Вызов `$book->appendChild()` размещает элемент `$title_element` под элементом `$book`.

Чтобы добавить текст в тегах `<title></title>`, создайте текстовый узел вызовом `createTextNode()` и присоедините его к `$title`:

```
$text_node = $dom->createTextNode('PHP Cookbook');
$title->appendChild($text_node);
```

Так как узел `$title` уже присоединен к документу, присоединять его заново к `$book` не нужно.

Порядок присоединения потомков к узлам роли не играет. Следующие четыре строки, в которых текстовый узел сначала присоединяется к узлу `$title_element`, который затем присоединяется к `$book`, эквивалентен предыдущему фрагменту:

```
$title_element = $dom->createElement('title');
$text_node = $dom->createTextNode('PHP Cookbook');
$title_element->appendChild($text_node);
```

```
$book->appendChild($title_element);
```

Чтобы добавить атрибут, вызовите для узла метод `setAttribute()`, передавая имя атрибута и его значение в аргументах:

```
$title->setAttribute('edition', '3');
```

Если теперь вывести элемент `title`, он будет выглядеть так:

```
<title edition="3">PHP Cookbook</title>
```

Когда дерево будет построено, документ можно вывести в строку или в файл:

```
// Строковое представление документа XML выводится в $books
$books = $dom->saveXML();

// Документ XML записывается в books.xml
$dom->save('books.xml');
```

По умолчанию методы генерируют вывод XML в одну длинную строку без пропусков, отступов и разрывов строк. Проблема решается установкой атрибута `formatOutput` объекта `DOMDocument` в состояние `true`:

```
// Вывод отформатированной версии документа DOM в формате XML
$dom->formatOutput = true;
```

После этого расширение DOM сгенерирует разметку XML следующего вида:

```
<?xml version="1.0"?>
<book>
  <title cover="soft">PHP Cookbook</title>
</book>
```

См. также

Рецепт 12.1 — запись XML без расширения DOM; Рецепт 12.4 — разбор XML с использованием DOM; документация по `DOMDocument` и функциям DOM.

12.3. Разбор базовых документов XML

Задача

Требуется разобрать базовый документ XML, построенный по известной схеме. При этом вас не интересуют более экзотические возможности XML (такие как инструкции по обработке).

Решение

Воспользуйтесь расширением SimpleXML. В следующем примере разметка XML читается из файла:

```
$sx = simplexml_load_file(__DIR__ . '/address-book.xml');

foreach ($sx->person as $person) {
    $firstname_text_value = $person->firstname;
    $lastname_text_value = $person->lastname;

    print "$firstname_text_value $lastname_text_value\n";
}
```

```
David Sklar
Adam Trachtenberg
```

Комментарий

Как бы фантастически это ни прозвучало, расширение SimpleXML справляется с совершенно нереальной задачей: обеспечить простое взаимодействие с XML. Хотите ли вы прочитать конфигурационный файл, написанный на XML, разобрать канал RSS, обработать результат REST-запроса — SimpleXML прекрасно справится с этими задачами. Это расширение не подойдет для более сложных операций, связанных с XML, например чтения документов с заранее неизвестным форматом или обращения к инструкциям по обработке или комментариям.

SimpleXML преобразует элементы в свойства объектов. Текст между тегами назначается свойству. Если в одном месте находятся несколько одноименных элементов (например, несколько тегов `<people>`), они размещаются в списке.

Атрибуты элементов становятся элементами массивов; ключ массива определяет имя атрибута, а его значение — значение атрибута.

Чтобы обратиться к одному значению, постройте ссылку на него в синтаксисе методов объектов. В качестве примера воспользуемся следующим фрагментом XML:

```
<firstname>David</firstname>
```

Если эта разметка представляется объектом SimpleXML `$firstname`, для обращения к `David` достаточно использовать запись:

```
$firstname
```

SimpleXML предполагает, что если узел содержит только текст, то вас интересует именно текст. Соответственно, команда `print $firstname` делает именно то, что и ожидалось: она выводит `David`.

Для перебора множественных элементов лучше всего подходят итеративные конструкции (такие как `foreach`). Примеры кода будут представлены ниже.

Атрибуты хранятся в элементах массивов. Например, следующий фрагмент выводит атрибут `id` первого элемента `person`:

```
$ab = simplexml_load_file(__DIR__ . '/address-book.xml');

// Атрибут id первого элемента person
print $ab->person['id'] . "\n";
```

Результат:

```
1
```

Следующая простая адресная книга в формате XML будет использоваться в дальнейших примерах.

```
<?xml version="1.0"?>
<address-book>
    <person id="1">
        <!--David Sklar-->
        <firstname>David</firstname>
        <lastname>Sklar</lastname>
        <city>New York</city>
        <state>NY</state>
        <email>sklar@php.net</email>
    </person>

    <person id="2">
        <!--Adam Trachtenberg-->
        <firstname>Adam</firstname>
        <lastname>Trachtenberg</lastname>
        <city>San Francisco</city>
        <state>CA</state>
        <email>amt@php.net</email>
    </person>
</address-book>
```

Извлечение имени (`firstname`) и фамилии (`lastname`) средствами SimpleXML:

```
$sx = simplexml_load_file(__DIR__ . '/address-book.xml');

foreach ($sx->person as $person) {
    $firstname_text_value = $person->firstname;
    $lastname_text_value = $person->lastname;

    print "$firstname_text_value $lastname_text_value\n";
}
```

```
David Sklar
Adam Trachtenberg
```

При использовании SimpleXML возможно напрямую перебирать элементы в цикле `foreach`. В данном примере итерация производится по ссылке `$sx->person`, содержащей все узлы `person`.

Также возможен прямой вывод объектов SimpleXML:

```
foreach ($sx->person as $person) {
    print "$person->firstname $person->lastname\n";
}
```

```
David Sklar
Adam Trachtenberg
```

PHP интерполирует простые объекты SimpleXML в строках, заключенных в кавычки, и получает хранящееся в них текстовое содержимое.

См. также

Рецепт 12.4 — разбор сложных документов XML; Рецепт 12.5 — разбор больших документов XML; документация по SimpleXML; дополнительная информация о библиотеке C libxml2.

12.4. Разбор сложных документов XML

Задача

Имеется сложный документ XML, например документ, требующий интроспекции для определения схемы или применения более экзотических возможностей XML (скажем, инструкций по обработке или комментариев).

Решение

Воспользуйтесь расширением DOM. Оно предоставляет полноценный интерфейс ко всем аспектам спецификации XML:

```
// $node - узел <book cover="soft">PHP Cookbook</book>, разобранный в DOM
$type = $node->nodeType;
switch($type) {
case XML_ELEMENT_NODE:
    // Я тег, у меня есть свойство tagname.
    print $node->tagName;  // Выводит свойство tagname: "book"
    break;
case XML_ATTRIBUTE_NODE:
    // Я атрибут, у меня есть свойства name и value.
    print $node->name;  // Выводит свойство name: "cover"
    print $node->value; // Выводит свойство value: "soft"
    break;
case XML_TEXT_NODE:
    // Я блок текста внутри элемента.
    // У меня есть свойства name и content.
    print $node->nodeName;  // Выводит свойство name: "#text"
    print $node->nodeValue; // Выводит текстовое содержимое: "PHP Cookbook"
    break;
default:
    // Другой тип
    break;
}
```

```
book
```

Комментарий

Стандарт W3C DOM предоставляет независимый от платформы и языка механизм определения структуры и содержимого документа. DOM позволяет прочитать документ XML в дерево, состоящее из узлов, а затем организовать пере-

мещение по дереву для получения информации о конкретном элементе или элементах, удовлетворяющих заданному критерию. Это называется *разбором на базе дерева*.

Кроме того, разработчик может изменять структуру дерева посредством создания, изменения и удаления узлов. Собственно, функции DOM позволяют построить новый документ XML с нуля; см. Рецепт 12.2.

Одно из главных преимуществ DOM заключается в том, что в соответствии со спецификацией W3C во многих языках функции DOM реализуются примерно одинаково, что существенно упрощает перевод логики и инструкций с одного языка на другой.

DOM — большая и сложная тема. За дополнительной информацией обращайтесь к спецификации.

Функции DOM в PHP являются объектно-ориентированными. Чтобы перейти от одного узла к другому, используйте такие свойства, как `$node->childNodes` (массив объектов узлов) и `$node->parentNode` (объект родительского узла). Таким образом, чтобы обработать узел, проверьте его тип и вызовите соответствующий метод, как показано в следующем примере:

```
// $node - узел <book cover="soft">PHP Cookbook</book>, разобранный в DOM
$type = $node->nodeType;
switch($type) {
case XML_ELEMENT_NODE:
    // Я тег, у меня есть свойство tagname.
    print $node->tagName;  // Выводит свойство tagname: "book"
    break;
case XML_ATTRIBUTE_NODE:
    // Я атрибут, у меня есть свойства name и value.
    print $node->name;  // Выводит свойство name: "cover"
    print $node->value; // Выводит свойство value: "soft"
    break;
case XML_TEXT_NODE:
    // Я блок текста внутри элемента.
    // У меня есть свойства name и content.
    print $node->nodeName;  // Выводит свойство name: "#text"
    print $node->nodeValue; // Выводит текстовое содержимое: "PHP Cookbook"
    break;
default:
    // Другой тип
    break;
}
```

Чтобы автоматически провести поиск конкретных элементов по дереву DOM, используйте метод `getElementsByTagname()`. Вот как он работает с деревом, содержащим записи о нескольких книгах:

```
<books>
    <book>
        <title>PHP Cookbook</title>
        <author>Sklar</author>
        <author>Trachtenberg</author>
```

```
        <subject>PHP</subject>
    </book>
    <book>
        <title>Perl Cookbook</title>
        <author>Christiansen</author>
        <author>Torkington</author>
        <subject>Perl</subject>
    </book>
</books>
```

Следующий фрагмент кода находит всех авторов (`author`):

```
// Поиск и вывод всех авторов
$authors = $dom->getElementsByTagname('author');

// Перебор элементов author
foreach ($authors as $author) {
    // childNodes содержит значения author
    $text_nodes = $author->childNodes;

    foreach ($text_nodes as $text) {
         print $text->nodeValue . "\n";
    }
}
```

```
Sklar
Trachtenberg
Christiansen
Torkington
```

Метод `getElementsByTagname()` возвращает массив объектов, представляющих узлы элементов. Перебирая потомков каждого элемента, вы можете добраться до текстового узла, связанного с этим элементом. Далее извлекаются значения узлов, которые в данном случае представляют имена авторов книг.

См. также

Рецепт 12.3 — разбор простых документов XML; Рецепт 12.5 — разбор больших документов XML; документация по DOM; дополнительная информация о библиотеке C libxml2.

12.5. Разбор больших документов XML

Задача

Требуется разобрать большой документ XML. Документ настолько огромен, что его не удастся полностью разместить в памяти; это исключает применение SimpleXML или DOM. Таким образом, документ придется загружать по частям.

Решение

Воспользуйтесь расширением XMLReader:

```
$reader = new XMLReader();
$reader->open(__DIR__ . '/card-catalog.xml');
/* Перебор содержимого документа */
while ($reader->read()) {
    /* Если текущим является элемент с именем 'author' */
    if($reader->nodeType == XMLREADER::ELEMENT &&
    $reader->localName == 'author') {
        /* Перейти к текстовому узлу и вывести его содержимое */
        $reader->read();
        print $reader->value . "\n";
    }
}
```

Комментарий

Парсеры XML делятся на две категории: одни хранят в памяти сразу весь документ, а другие в любой момент времени загружают в память небольшую часть документа.

Первые называются *древовидными парсерами*, потому что документ преобразуется в структуру данных, называемую деревом. Расширения SimpleXML и DOM из Рецептов 12.3 и 12.4 относятся к категории древовидных парсеров. С древовидным парсером работать проще, но PHP расходует больше памяти. С большинством документов XML это не создает проблем, но с очень большими документами XML могут возникнуть серьезные трудности с быстродействием.

Другую категорию парсеров XML составляют *потоковые парсеры*. Они не хранят весь документ в памяти, а читают данные по одному узлу, предоставляя разработчику возможность обрабатывать данные в реальном времени. После перехода к следующему узлу старый узел пропадает (если только программа не сохранит его явно для последующего использования). Потоковые парсеры работают быстрее и занимают меньше памяти, но требуют написания кода обработки документа.

Самый простой способ обработки данных XML с использованием потокового парсера основан на использовании расширения XMLReader. Это расширение базируется на API XmlTextReader языка C#. По сравнению с технологией SAX (Simple API for XML), расширение XMLReader более интуитивно, обладает большей функциональностью и быстрее работает.

Начните с создания нового экземпляра класса `XMLReader` и указания местонахождения данных XML:

```
// Создание нового объекта XMLReader
$reader = new XMLReader();

// Загрузка из файла или по URL
$reader->open('document.xml');
```

```
// Или загрузка из переменной PHP
$reader->XML($document);
```

Чаще всего данные загружаются из внешнего источника методом `XMLReader::open()`, но также возможна загрузка из существующей переменной PHP вызовом `XMLReader::XML()`.

После настройки объекта можно переходить к обработке данных. Сначала текущая позиция находится в начале документа. Для перемещения по документу можно использовать навигационные методы, предоставляемые классом `XMLReader`: `XMLReader::read()` и `XMLReader::next()`. Первый метод читает блок данных XML, следующий непосредственно за текущей позицией. Второй метод осуществляет переход к следующему одноранговому элементу («брату») после текущей позиции.

Например, возьмем следующую разметку XML:

```
<books>
    <book isbn="1565926811">
        <title>PHP Cookbook</title>
        <author>Sklar</author>
        <author>Trachtenberg</author>
        <subject>PHP</subject>
    </book>
    <book isbn="0596003137">
        <title>Perl Cookbook</title>
        <author>Christiansen</author>
        <author>Torkington</author>
        <subject>Perl</subject>
    </book>
</books>
```

Когда объект переходит к первому элементу `<book>`, метод `read()` осуществляет переход к следующему элементу под `<book>` (формально это пропуск между `<book>` и `<title>`). С другой стороны, метод `next()` перемещается к следующему элементу `<book>` и пропускает все поддерево `PHP Cookbook`.

Методы возвращают `true`, если им удалось успешно переместиться к другому узлу, и `false`, если переход не состоялся. Они часто используются в циклах `while` следующего вида:

```
/* Цикл по содержимому документа */
while ($reader->read()) {
    /* Обработка XML */
}
```

Объект читает весь документ XML по одному фрагменту. Внутри `while()` проанализируйте значение `$reader` и обработайте его соответствующим образом.

Обычно в цикле проверяется тип узла. Это позволяет узнать, был ли достигнут нужный элемент (а затем проверить имя элемента), закрывающий элемент, атрибут, блок текста, пропуск или любая другая часть документа XML. Для проверки используется атрибут `nodeType`:

```
/* Цикл по содержимому документа */
while ($reader->read()) {
    /* Если текущим является элемент с именем 'author' */
    if($reader->nodeType == XMLREADER::ELEMENT &&
    $reader->localName == 'author') {
        /* Обработка элемента author */
    }
}
```

Этот код проверяет, является ли узел элементом, и если является — имеет ли он имя `author`. Полный список возможных значений `nodeType` приведен в таблице 12.1.

Таблица 12.1. Типы узлов XMLReader

Тип узла	Описание
XMLReader::NONE	Тип узла не указан
XMLReader::ELEMENT	Начало элемента
XMLReader::ATTRIBUTE	Узел атрибута
XMLReader::TEXT	Текстовый узел
XMLReader::CDATA	Узел CDATA
XMLReader::ENTITY_REF	Узел ссылки на сущность
XMLReader::ENTITY	Узел объявления сущности
XMLReader::PI	Узел инструкций по обработке
XMLReader::COMMENT	Узел комментария
XMLReader::DOC	Узел документа
XMLReader::DOC_TYPE	Узел типа документа
XMLReader::DOC_FRAGMENT	Узел фрагмента документа
XMLReader::NOTATION	Узел обозначения
XMLReader::WHITESPACE	Узел пропуска
XMLReader::SIGNIFICANT_WHITESPACE	Узел значащего пропуска
XMLReader::END_ELEMENT	Конец элемента
XMLReader::END_ENTITY	Конец сущности
XMLReader::XML_DECLARATION	Узел объявления XML

Далее вы можете решить, как обработать этот элемент и содержащиеся в нем данные. Например, можно вывести имена всех авторов в картотеке:

```
$reader = new XMLReader();
$reader->open(__DIR__ . '/card-catalog.xml');

/* Цикл по содержимому документа */
while ($reader->read()) {
    /* Если текущим является элемент с именем 'author' */
    if($reader->nodeType == XMLREADER::ELEMENT &&
    $reader->localName == 'author') {
```

```
        /* Переход к текстовому узлу и его вывод */
        $reader->read();
        print $reader->value . "\n";
    }
}
```

```
Sklar
Trachtenberg
Christiansen
Torkington
```

После того как будет достигнут элемент `<author>`, вызовите `$reader->read()` для перехода к содержащемуся в нем тексту. После этого вы найдете имена авторов в `$reader->value`.

Атрибут `XMLReader::value` предоставляет доступ к значению узла. Это относится только к узлам, имеющим осмысленное содержимое, например текстовым узлам или CDATA. Во всех остальных случаях (например, для узлов элементов) этому атрибуту присваивается пустая строка.

В таблице 12.2 приведен полный список свойств `XMLReader`, включая свойство `value`.

Таблица 12.2. Свойства объекта XMLReader

Имя	Тип	Описание
attributeCount	int	Количество атрибутов узла
baseURI	string	Базовый URI-идентификатор узла
depth	int	Глубина узла в дереве (начиная с уровня 0)
hasAttributes	bool	Признак наличия атрибутов у узла
hasValue	bool	Признак наличия текстового значения
isDefault	bool	Признак использования значения по умолчанию из DTD
isEmptyElement	bool	Признак пустого тега элемента
localName	string	Локальное имя узла
name	string	Уточненное имя узла
namespaceURI	string	URI пространства имен, связанного с узлом
nodeType	int	Тип узла
prefix	string	Префикс пространства имен, связанного с узлом
value	string	Текстовое значение узла
xmlLang	string	Область видимости xml:lang

Осталось упомянуть об одном важном аспекте функциональности `XMLReader` — атрибутах. `XMLReader` поддерживает специальный набор методов для обращения к данным атрибутов текущего узла элемента, включая методы `moveToAttribute()`, `moveToFirstAttribute()` и `moveToNextAttribute()`.

Метод `moveToAttribute()` позволяет указать имя атрибута. Например, следующий код использует картотеку в формате XML для вывода всех номеров ISBN:

```
$reader = new XMLReader();
$reader->XML($catalog);

/* Цикл по содержимому документа */
while ($reader->read()) {
    /* Если текущим является элемент с именем 'book' */
    if ($reader->nodeType == XMLREADER::ELEMENT &&
    $reader->localName == 'book') {
        $reader->moveToAttribute('isbn');
        print $reader->value . "\n";
    }
}
```

После того как элемент `<book>` будет найден, вызовите `moveToAttribute('isbn')` для перехода к атрибуту `isbn`, чтобы прочитать его значение и вывести его:

```
1565926811
0596003137
```

В примерах этого рецепта выводится информация по всем книгам. Тем не менее эти примеры легко изменить так, чтобы они получали данные только по одной конкретной книге. Например, следующий код объединяет фрагменты кода примеров для эффективного вывода всех данных книги `Perl Cookbook`:

```
$reader = new XMLReader();
$reader->XML($catalog);

// Номер ISBN книги Perl Cookbook - 0596003137
// Использование массива упрощает добавление других номеров ISBN
$isbns = array('0596003137' => true);

/* Цикл по содержимому документа для поиска первого элемента <book> */
while ($reader->read()) {
    /* Если текущим является элемент с именем 'book' */
    if ($reader->nodeType == XMLREADER::ELEMENT &&
        $reader->localName == 'book') {
        break;
    }
}

/* Перебор элементов <book> для поиска нужных номеров ISBN */
do {
    if ($reader->moveToAttribute('isbn') &&
        isset($isbns[$reader->value])) {
        while ($reader->read()) {
            switch ($reader->nodeType) {
            case XMLREADER::ELEMENT:
                print $reader->localName . ": ";
                break;
            case XMLREADER::TEXT:
                print $reader->value . "\n";
                break;
            case XMLREADER::END_ELEMENT;
```

```
                if ($reader->localName == 'book') {
                    break 2;
                }
            }
        }
    }
} while ($reader->next());

title: Perl Cookbook
author: Christiansen
author: Torkington
subject: Perl
```

Первый цикл `while()` последовательно перебирает содержимое документа до того, как найдет первый элемент `<book>`.

После того как позиционирование будет выполнено, программа выходит из цикла и начинает проверять номера ISBN. Это делается в цикле `do...while()`, использующем вызов `$reader->next()` для перемещения по списку `<book>`. Использовать обычный цикл `while()` нельзя, потому что будет пропущен первый элемент `<book>`. Кроме того, это превосходный пример использования `$reader->next()` вместо `$reader->read()`.

Если номер ISBN совпадает со значением в `$isbns`, нужно обработать данные в текущем элементе `<book>`. Эта задача решается с использованием еще одного цикла `while()` и `switch()`.

Конструкция `switch()` состоит из трех секций: открывающий элемент, текст элемента и закрывающий элемент. Если элемент открывается, то выводится имя элемента и двоеточие. Для текста выводятся текстовые данные. Наконец, при закрытии элемента необходимо проверить, закрывается ли элемент `<book>`. В этом случае достигнут конец данных конкретной книги и программа должна выйти из цикла `do... while()`. Эта задача решается командой `break 2;` — управление возвращается на два уровня вместо обычного одного.

См. также

Рецепт 12.3 — разбор простых документов XML; Рецепт 12.4 — разбор сложных документов XML; документация по XMLReader; дополнительная информация о функциях `XMLReader` библиотеки C libxml2.

12.6. Извлечение информации с использованием XPath

Задача

Требуется обращаться к данным XML со сложными запросами без разбора документа по узлам.

Решение

Воспользуйтесь средствами XPath.

Поддержка XPath доступна в SimpleXML:

```
$s = simplexml_load_file(__DIR__ . '/address-book.xml');
$emails = $s->xpath('/address-book/person/email');

foreach ($emails as $email) {
    // Обработать $email
}
```

И в DOM:

```
$dom = new DOMDocument;
$dom->load(__DIR__ . '/address-book.xml');
$xpath = new DOMXPath($dom);
$emails = $xpath->query('/address-book/person/email');

foreach ($emails as $email) {
    // Обработать $email
}
```

Комментарий

Если не считать самых простых документов, редко когда существует простой механизм просмотра нужных данных элемент за элементом. По мере усложнения файлов XML и требований к разбору обычно бывает проще использовать XPath, чем фильтровать данные в `foreach`.

PHP содержит класс `XPath`, в конструкторе которого передается объект DOM. После этого можно проводить поиск по объекту, получая в ответ узлы DOM. Расширение SimpleXML также поддерживает XPath, и эту поддержку проще использовать благодаря ее тесной интеграции с объектом SimpleXML.

DOM поддерживает запросы XPath, но такой запрос не применяется непосредственно к объекту DOM. Вместо этого в программе создается объект `DOMXPath`:

```
$dom = new DOMDocument;
$dom->load(__DIR__ . '/address-book.xml');
$xpath = new DOMXPath($dom);
$emails = $xpath->query('/address-book/person/email');
```

Создайте экземпляр `DOMXPath`, передав конструктору объект `DOMDocument`. Чтобы выполнить запрос XPath, вызовите `query()` передачей текста запроса в аргументе. При этом вы получите список узлов DOM, соответствующих запросу; элементы списка можно перебрать в `foreach`:

```
$dom = new DOMDocument;
$dom->load(__DIR__ . '/address-book.xml');
$xpath = new DOMXPath($dom);
$emails = $xpath->query('/address-book/person/email');
```

```
foreach ($emails as $e) {
    $email = $e->firstChild->nodeValue;
    // Обработать $email
}
```

После создания нового объекта `DOMXPath` вызывается метод `DOMXPath::query()`, которому в первом параметре передается запрос XPath (`/people/person/email` в этом примере). Функция возвращает список подходящих узлов DOM.

По умолчанию `DOMXPath::query()` работает со всем документом XML. Чтобы ограничить поиск по части дерева, передайте поддерево в последнем параметре `query()`. Например, чтобы получить все имена и фамилии из адресной книги, загрузите все узлы `person` и обратитесь с запросом по каждому узлу в отдельности:

```
$dom = new DOMDocument;
$dom->load(__DIR__ . '/address-book.xml');
$xpath = new DOMXPath($dom);
$people = $xpath->query('/address-book/person');

foreach ($people as $p) {
    $fn = $xpath->query('firstname', $p);
    $firstname = $fn->item(0)->firstChild->nodeValue;

    $ln = $xpath->query('lastname', $p);
    $lastname = $ln->item(0)->firstChild->nodeValue;

    print "$firstname $lastname\n";
}
```

В цикле `foreach` вызов `DOMXPath::query()` получает узлы `firstname` и `lastname`. На этот раз, в дополнение к запросу XPath, методу также передается значение `$p`, вследствие чего поиск ограничивается данным узлом.

В отличие от расширения DOM, все объекты SimpleXML имеют встроенный метод `xpath()`. Вызов этого метода выдает запрос к текущему объекту с использованием XPath и возвращает объект SimpleXML с соответствующими узлами, чтобы вам не приходилось создавать отдельный объект для использования XPath. Единственным аргументом метода является запрос XPath.

А вот как производится поиск всех адресов электронной почты в адресной книге из нашего примера:

```
$s = simplexml_load_file(__DIR__ . '/address-book.xml');
$emails = $s->xpath('/address-book/person/email');
foreach ($emails as $email) {
    // Обработать $email
}
```

Эта запись короче, потому что она не требует разыменования `firstChild` или получения `nodeValue`.

SimpleXML также справляется с более сложными задачами. Так как `xpath()` возвращает объекты SimpleXML, вы можете обращаться к ним с запросами напрямую:

```
$s = simplexml_load_file(__DIR__ . '/address-book.xml');
$people = $s->xpath('/address-book/person');

foreach($people as $p) {
    list($firstname) = $p->xpath('firstname');
    list($lastname) = $p->xpath('lastname');

    print "$firstname $lastname\n";
}
```

```
David Sklar
Adam Trachtenberg
```

Внутренние запросы XPath возвращают только один элемент, поэтому для его извлечения из массива используется `list`.

См. также

Документация по DOM XPath; официальная спецификация XPath.

12.7. Преобразование XML с использованием XSLT

Задача

Имеется документ XML и таблица стилей XSLT. Требуется преобразовать документ с использованием XSLT и сохранить результаты. Это позволяет применять таблицы стилей к данным и создавать разные версии контента для разных средств передачи информации.

Решение

Воспользуйтесь PHP-расширением XSLT:

```
// Загрузка шаблона XSL
$xsl = new DOMDocument;
$xsl->load(__DIR__ . '/stylesheet.xsl');

// Создание объекта XSLTProcessor
$xslt = new XSLTProcessor();
// Загрузка таблицы стилей
$xslt->importStylesheet($xsl);

// Загрузка входного файла XML
$xml = new DOMDocument;
$xsl->load(__DIR__ . '/address-book.xml');
// Преобразование в строку
$results = $xslt->transformToXML($xml);
```

```
// Преобразование в файл
$results = $xslt->transformToURI($xml, 'results.txt');

// Преобразование в объект DOM
$results = $xslt->transformToDoc($xml);
```

Преобразованный текст сохраняется в `$results`.

Комментарий

Документы XML описывают содержимое данных, но не содержат никакой информации о том, как эти данные должны отображаться. Но когда контент XML объединяется с таблицей стилей на языке XSL (eXtensible Stylesheet Language), этот контент отображается в соответствии с правилами конкретного визуального представления.

Между XML и XSL находится «прослойка» XSLT (eXtensible Stylesheet Language Transformations). Преобразования XSLT применяют серию правил, перечисленных в таблице стилей, к данным XML. Итак, по аналогии с тем, как PHP разбирает ваш код и объединяет его с пользовательским вводом для формирования динамической страницы, программа XSLT использует XSL и XML для вывода новой страницы, содержащей новые данные в формате XML, HTML или любом другом формате, для которого можно предоставить формальное описание.

Существует несколько программных реализаций XSLT, каждая из которых обладает своим набором функций и ограничениями. PHP поддерживает только процессор libxslt.

Работа с XSLT в PHP состоит из двух основных шагов: подготовки объекта XSLT и запуска преобразования для каждого файла XML.

Начните с загрузки таблицы стилей с использованием DOM. Затем создайте новый объект `XSLTProcessor` и импортируйте документ XSLT, передавая его методу `importStylesheet()` только что созданного объекта DOM:

```
// Загрузка шаблона XSL
$xsl = new DOMDocument;
$xsl->load(__DIR__ . '/stylesheet.xsl');

// Создание нового объекта XSLTProcessor
$xslt = new XSLTProcessor();
// Загрузка таблицы стилей
$xslt->importStylesheet($xsl);
```

Преобразователь готов к работе. Любой объект DOM может быть преобразован одним из трех способов — в строку, в файл или обратно в другой объект DOM:

```
// Загрузка входного файла с данными XML
$xml = new DOMDocument;
$xsl->load(__DIR__ . '/stylesheet.xsl');

// Преобразование в строку
```

```
$results = $xslt->transformToXML($xml);

// Преобразование в файл
$results = $xslt->transformToURI($xml, 'results.txt');

// Преобразование в объект DOM
$results = $xslt->transformToDoc($xml);
```

При вызове `transformToXML()` или `transformToDoc()` расширение возвращает полученную строку или объект. С другой стороны, `transformToURI()` возвращает количество байтов, записанных в файл, а не сам документ.

Методы возвращают `false` при возникновении ошибок, поэтому правильный способ выполнения проверки выглядит так:

```
if (false === ($results = $xslt->transformToXML($xml))) {
    // Произошла ошибка
}
```

Оператор `===` предотвращает путаницу между возвращаемым значением 0 и ошибкой.

См. также

Документация по функциям XSL: *http://www.php.net/xsl*.

12.8. Настройка параметров XSLT из PHP

Задача

Требуется задать параметры таблицы стилей XSLT из PHP.

Решение

Воспользуйтесь методом `XSLTProcessor::setParameter()`:

```
// Также можно взять из $_GET['city'];
$city = 'San Francisco';

$dom  = new DOMDocument;
$dom->load(__DIR__ . '/address-book.xml');
$xsl  = new DOMDocument;
$xsl->load(__DIR__ . '/stylesheet.xsl');

$xslt = new XSLTProcessor();
$xslt->importStylesheet($xsl);
$xslt->setParameter(NULL, 'city', $city);
print $xslt->transformToXML($dom);
```

Этот код присваивает параметру XSLT `city` значение, хранящееся в переменной PHP `$city`.

Комментарий

Для передачи данных из PHP в таблицу стилей XSLT используется метод `setParameter()`. Он позволяет выполнять такие операции, как фильтрация данных таблицы стилей, на основании данных, введенных пользователем.

Например, следующая программа позволяет искать людей по названию города:

```
// Также можно взять из $_GET['city'];
$city = 'San Francisco';

$dom  = new DOMDocument;
$dom->load(__DIR__ . '/address-book.xml');
$xsl  = new DOMDocument;
$xsl->load(__DIR__ . '/stylesheet.xsl');

$xslt = new XSLTProcessor();
$xslt->importStylesheet($xsl);
$xslt->setParameter(NULL, 'city', $city);
print $xslt->transformToXML($dom);
```

Программа использует следующую таблицу стилей:

```
<?xml version="1.0" ?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="@*|node()">
  <xsl:copy>
    <xsl:apply-templates select="@*|node()"/>
  </xsl:copy>
</xsl:template>

<xsl:template match="/address-book/person">
  <xsl:if test="city=$city">
    <xsl:copy>
      <xsl:apply-templates select="@*|node()"/>
    </xsl:copy>
  </xsl:if>
</xsl:template>
</xsl:stylesheet>
```

Программа в сочетании с таблицей стилей дает следующий результат:

```
<?xml version="1.0"?>
<address-book>

    <person id="2">
        <!--Adam Trachtenberg-->
        <firstname>Adam</firstname>
        <lastname>Trachtenberg</lastname>
        <city>San Francisco</city>
        <state>CA</state>
        <email>amt@php.net</email>
    </person>
</address-book>
```

Сценарий PHP выполняет стандартное преобразование XSLT, за исключением вызова `$xslt->setParameter(NULL, city, $city)`. В первом аргументе передается пространство имен параметра, во втором — имя параметра, а в третьем — значение параметра. В данном случае значение, хранящееся в переменной PHP `$city` (`San Francisco`), присваивается параметру XSLT `city`, который не находится в пространстве имен. Результат эквивалентен размещению следующего элемента в файле XSLT:

```
<xsl:param name="city">San Francisco</xsl:param>
```

Обычно обращения к параметрам в таблицах стилей выглядят так же, как обращения к переменным PHP, — перед именем ставится знак доллара (`$`). В примере с таблицей стилей создается шаблон для поиска узлов `/address-book/person`.

Внутри шаблона проверяется условие `city=$city`; другими словами, равен ли потомок `city` текущего узла значению параметра `city`? Если значения совпадают, то потомок копируется; в противном случае записи исключаются.

В приведенном примере `city` содержит значение `San Francisco`, поэтому запись с данными `David` удаляется, а запись с данными `Adam` остается.

См. также

Документация по методу `XSLTProcessor::setParameter()`.

12.9. Вызов функций PHP из таблиц стилей XSLT

Задача

Требуется вызывать функции PHP из таблицы стилей XSLT.

Решение

Для включения необходимой функциональности вызовите метод `XSLTProcessor::registerPHPFunctions()`:

```
$xslt = new XSLTProcessor();
$xslt->registerPHPFunctions();
```

Далее используйте функцию `function()` или `functionString()` из своей таблицы стилей:

```
<?xml version="1.0" ?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:php="http://php.net/xsl"
```

```
    xsl:extension-element-prefixes="php">
<xsl:template match="/">
    <xsl:value-of select="php:function('strftime', '%c')" />
</xsl:template>

</xsl:stylesheet>
```

Комментарий

Параметры XSLT прекрасно подходят для передачи данных из PHP в XSLT. Тем не менее для передачи в обратном направлении они не очень подходят. Параметры не могут использоваться для извлечения информации из таблицы стилей во время преобразования. В идеале хотелось бы вызывать функции PHP из таблицы стилей и возвращать информацию PHP.

К счастью, существует метод, реализующий эту функциональность, — `registerPHPFunctions()`:

```
$xslt = new XSLTProcessor();
$xslt->registerPHPFunctions();
```

Этот метод позволяет вызывать из таблиц стилей любые функции PHP. Данная возможность отключена по умолчанию, потому что она создает угрозу для безопасности при обработке таблиц стилей, находящихся под контролем других людей.

Работают как встроенные, так и пользовательские функции. В таблице стилей необходимо определить пространство имен и вызвать методы `function()` или `functionString()`, как показано ниже:

```
<?xml version="1.0" ?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:php="http://php.net/xsl"
    xsl:extension-element-prefixes="php">

<xsl:template match="/">
    <xsl:value-of select="php:function('strftime', '%c')" />
</xsl:template>

</xsl:stylesheet>
```

В начале таблицы стилей определяется пространство имен PHP: *http://php.net/xsl*. В данном примере будет использоваться префикс пространства имен `php`; присвойте `extension-element-prefixes` значение `php`, чтобы процессор XSLT знал, что перед ним функции.

Чтобы вызвать функцию PHP, используйте синтаксис `php:function()`. В первом параметре передается имя функции; в остальных параметрах передаются аргументы. В данном случае передается имя функции `strftime` с единственным аргументом `%c`; в этом случае `strftime` возвращает текущую дату и время.

В следующем примере таблица стилей из файла strftime.xsl используется для обработки документа XML, состоящего из одного элемента:

```
$dom  = new DOMDocument;
$dom->loadXML('<blank/>');
$xsl  = new DOMDocument;
$xsl->load(__DIR__ . '/strftime.xsl');

$xslt = new XSLTProcessor();
$xslt->importStylesheet($xsl);
$xslt->registerPHPFunctions();
print $xslt->transformToXML($dom);
```

```
Mon Jul 22 06:01:10 2014
```

Происходит стандартная обработка XSLT, но с дополнительным вызовом `registerPHPFunctions()` для активизации поддержки функций PHP.

Также могут возвращаться объекты DOM. Пример из листинга 12.2 получает адресную книгу XML и преобразует все адреса электронной почты, заменяя имя хоста тремя точками. Вся остальная информация в документе остается неизменной.

Листинг 12.2. Защита адресов электронной почты от спама

```
function mangle_email($nodes) {
    return preg_replace('/([^@\s]+)@([-a-z0-9]+\.)+[a-z]{2,}/is',
                        '$1@...',
                        $nodes[0]->nodeValue);
}

$dom  = new DOMDocument;
$dom->load(__DIR__ . '/address-book.xml');
$xsl  = new DOMDocument;
$xsl->load(__DIR__ . '/mangle-email.xsl');

$xslt = new XSLTProcessor();
$xslt->importStylesheet($xsl);
$xslt->registerPhpFunctions();
print $xslt->transformToXML($dom);
```

В таблице стилей создайте специальный шаблон для элементов `/address-book/person/email`. Пример:

```
<?xml version="1.0" ?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:php="http://php.net/xsl"
  xsl:extension-element-prefixes="php">

<xsl:template match="@*|node()">
  <xsl:copy>
    <xsl:apply-templates select="@*|node()"/>
  </xsl:copy>
</xsl:template>

<xsl:template match="/address-book/person/email">
  <xsl:copy>
```

```
    <xsl:value-of select="php:function('mangle_email', node())" />
  </xsl:copy>
</xsl:template>
</xsl:stylesheet>
```

Первый шаблон обеспечивает неизменность элементов, а второй передает текущий узел PHP для обработки. Во втором шаблоне функция `mangle_email()` вместо строки получает текущий узел, представленный в XPath как `node()`. Будьте внимательны — вызов `node()` не должен быть заключен в кавычки, в противном случае будет передан литеральный текст `node()`.

Узлы становятся объектами DOM внутри PHP и всегда поступают в виде массива. В данном случае `mangle_email()` знает, что всегда имеет дело только с одним объектом, который является объектом `DOMText`, поэтому адрес электронной почты берется из `$nodes[0]->nodeValue`.

Если вы уверены в том, что для вас представляет интерес только текстовая часть узла, воспользуйтесь функцией `functionString()`. Эта функция преобразует узлы в строки PHP, что позволяет опустить обращения к массиву и разыменование `nodeValue`:

```
function mangle_email($email) {
    return preg_replace('/([^@\s]+)@([-a-z0-9]+\.)+[a-z]{2,}/is',
                        '$1@...',
                        $email);
}

// Остальной код не изменился
```

Новый шаблон таблицы стилей для `/address-book/person/email` выглядит так:

```
<xsl:template match="/address-book/person/email">
  <xsl:copy>
    <xsl:value-of
       select="php:functionString('mangle_email', node())" />
  </xsl:copy>
</xsl:template>
```

Функция `mangle_email()` теперь обрабатывает `$email` вместо `$nodes[0]->nodeValue`, потому что шаблон теперь вызывает функцию `functionString()`.

Методы `function()` и `functionString()` чрезвычайно полезны, но их применение противоречит запланированной роли XSLT как механизма преобразований, независимого от языка программирования. При вызове функций PHP из XSLT не удастся легко использовать таблицы стилей в проектах на базе Java, Perl и других языков, потому что они не могут вызывать PHP. Следовательно, перед использованием этой возможности следует задуматься о балансе между удобством и универсальностью кода.

См. также

Документация по методу `XSLTProcessor::registerPHPFunctions()`.

12.10. Проверка действительности документов XML

Задача

Требуется убедиться в том, что документ XML соответствует правилам некоторой схемы: XML Schema, Relax NG или DTD.

Решение

Воспользуйтесь расширением DOM.

Чтобы проверить действительность объекта DOM на соответствие схеме, хранящейся в файле, вызовите `DOMDocument::schemaValidate()` или `DOMDocument::relaxNGValidate()`:

```
$file = __DIR__ . '/address-book.xml';
$schema = __DIR__ . '/address-book.xsd';
$ab = new DOMDocument;
$ab->load($file);

if ($ab->schemaValidate($schema)) {
    print "$file is valid.\n";
} else {
    print "$file is invalid.\n";
}
```

Если в начале документа XML указана схема DTD, вызовите `DOMDocument::validate()` для проверки его действительности по схеме DTD.

Чтобы проверить объект DOM по схеме, хранящейся в переменной, вызовите метод `DOMDocument::schemaValidateSource()` или `DOMDocument::relaxNGValidateSource()`:

```
$file = __DIR__ . '/address-book.xml';
$ab = new DOMDocument;
$ab->load($file);

$schema = file_get_contents(__DIR__ . '/address-book.xsd');

if ($ab->schemaValidateSource($schema)) {
    print "XML is valid.\n";
} else {
    print "XML is invalid.\n";
}
```

Комментарий

Схемы (schemas) определяют спецификацию для документов XML. Существует несколько способов кодирования схем с разным синтаксисом (хотя и решающих одну и ту же задачу). Некоторые популярные форматы: DTD (Document Type

Definition), XML Schema и Relax NG. Схемы DTD появились раньше других, но они записываются не на XML и имеют другие недостатки, и с ними бывает трудно работать. Схемы XML Schema и Relax NG появились позднее, и их разработчики попытались решить некоторые проблемы, окружающие DTD.

PHP в своей поддержке проверки действительности использует библиотеку libxml2. Соответственно, разработчик может проверять файлы по всем трем типам. Наибольшая гибкость проверки достигается с XML Schema и Relax NG, но поддержка XML Schema реализована не полностью. С большинством документов XML Schema быть не должно, но libxml2 может не справиться с некоторыми сложными схемами или при использовании более экзотических возможностей.

В PHP расширение DOM поддерживает проверку действительности для схем DTD, XML Schema и Relax NG, тогда как SimpleXML предоставляет только проверку XML Schema.

Проверка действительности любого файла с использованием DOM проходит примерно одинаково независимо от формата схемы. Чтобы выполнить проверку, вызовите соответствующий метод для объекта DOM. Пример:

```
$file = __DIR__ . '/address-book.xml';
$schema = __DIR__ . '/address-book.xsd';
$ab = new DOMDocument;
$ab->load($file);

if ($ab->schemaValidate($schema)) {
    print "$file is valid.\n";
} else {
    print "$file is invalid.\n";
}
```

Если файл проходит проверку, метод возвращает `true`. При наличии ошибок возвращается `false`, а в журнал ошибок записывается соответствующее сообщение.

Если схема хранится в строке, используйте метод `DOMDocument::schemaValidateSource()` вместо `schemaValidate()`.

Таблица 12.3. Методы проверки действительности по схеме DOM

Имя метода	Тип схемы	Способ хранения данных
schemaValidate	XML Schema	Файл
schemaValidateSource	XML Schema	Строка
relaxNGValidate	Relax NG	Файл
relaxNGValidateSource	Relax NG	Строка
validate	DTD	–

Все методы проверки действительности работают примерно одинаково, так что для перехода на другую схему проверки действительности в предыдущем примере достаточно изменить имя метода.

И XML Schema и Relax NG поддерживают проверку действительности для файлов и строк. Объект DOM может проверяться только по схеме DTD, определенной в начале документа.

См. также

Спецификация XML Schema и Relax NG.

12.11. Преобразование кодировки контента

Задача

Расширения PHP для работы с XML используют UTF-8, но ваши данные хранятся в другой кодировке.

Решение

Воспользуйтесь библиотекой `iconv`, чтобы преобразовать данные перед их передачей расширению XML:

```
$utf_8 = iconv('ISO-8859-1', 'UTF-8', $iso_8859_1);
```

После завершения обработки преобразуйте данные обратно:

```
$iso_8859_1 = iconv('UTF-8', 'ISO-8859-1', $utf_8);
```

Комментарий

Кодировка символов — одна из слабых сторон PHP. При попытке использования расширений XML с данными в произвольной кодировке могут возникнуть проблемы.

Для простоты все расширения XML используют исключительно кодировку UTF-8. Это означает, что все они получают данные в UTF-8 и выводят данные в UTF-8. Если ваши данные хранятся в ASCII, беспокоиться не о чем; UTF-8 является надмножеством ASCII. Но если вы работаете в другой кодировке, рано или поздно возникнут проблемы.

Чтобы решить эту проблему, воспользуйтесь расширением `iconv` для ручного преобразования вашей кодировки в UTF-8 и обратно. Например, преобразование из ISO-8859-1 в UTF-8 выполняется так:

```
$utf_8 = iconv('ISO-8859-1', 'UTF-8', $iso_8859_1);
```

Функция `iconv` поддерживает два специальных модификатора для итоговой кодировки: `//TRANSLIT` и `//IGNORE`. Первый сообщает `iconv`, что каждый раз, ког-

да символ не удается точно воспроизвести в итоговой кодировке, следует попытаться сгенерировать его приближенное представление в виде серии других символов. Со вторым модификатором `iconv` просто игнорирует все непреобразуемые символы.

Допустим, строка `$geb` содержит текст `Gödel, Escher, Bach`. Попытка прямого преобразования в ASCII приводит к ошибке:

```
echo iconv('UTF-8', 'ASCII', $geb);

PHP Notice:  iconv(): Detected an illegal character in input string...
```

С модификатором `//IGNORE` преобразование выполняется:

```
echo iconv('UTF-8', 'ASCII//IGNORE', $geb);
```

Однако результат выглядит некрасиво, в нем нет буквы ö:

```
Gdel, Escher, Bach
```

Лучше всего воспользоваться режимом `//TRANSLIT`:

```
echo iconv('UTF-8', 'ASCII//TRANSLIT', $geb);
```

На этот раз строка выглядит немного лучше:

```
G"odel, Escher, Bach
```

Будьте осторожны с модификатором `//TRANSLIT`, потому что он может привести к увеличению количества символов. Например, один символ ö заменяется двумя символами: " и o.

См. также

Рецепт 19.12 — дополнительная информация о работе с текстом в кодировке UTF-8; документация по расширению iconv; домашняя страница GNU libiconv (*http://www.gnu.org/software/libiconv/*).

12.12. Чтение каналов RSS и Atom

Задача

Требуется загружать каналы RSS и Atom и обрабатывать их содержимое. Это позволит вам интегрировать каналы новостей с нескольких сайтов в ваше приложение.

Решение

Воспользуйтесь парсером MagpieRSS. В следующем примере читается канал RSS для списка рассылки `php.announce`:

```
require __DIR__ . '/magpie/rss_fetch.inc';

$feed = 'http://news.php.net/group.php?group=php.announce&format=rss';

$rss = fetch_rss( $feed );

print "<ul>\n";
foreach ($rss->items as $item) {
    print '<li><a href="' . $item['link'] . '">' . $item['title'] .
          "</a></li>\n";
}
print "</ul>\n";
```

Комментарий

RSS — простой и удобный формат синдикации статей или заголовков в виде разметки XML. Многие новостные сайты (такие как New York Times и Washington Post) предоставляют каналы RSS, обновляемые при публикации новых статей. Поддержка RSS также реализована во многих блогах. Сайт PHP также публикует каналы RSS для многих списков рассылки PHP.

Atom также представляет собой формат синдикации на базе XML. Он расширяет многие концепции RSS, включая возможность чтения и записи данных Atom, и пытается предоставить более четко определенный синтаксис синдикации по сравнению с RSS, потому что спецификация RSS не всегда однозначно определяет, что разрешено или запрещено в поставках новостей.

С MagpieRSS загрузка и разбор RSS и Atom выполняются просто:

```
$feed = 'http://news.php.net/group.php?group=php.announce&format=rss';

$rss = fetch_rss($feed);
```

В этом примере читается канал RSS для списка рассылки `php.announce`. Затем данные разбираются вызовом `fetch_rss()` и сохраняются в переменной `$rss`.

Хотя этот канал ведется в формате RSS 0.93, передавать эту информацию MagpieRSS не нужно. Функция `fetch_rss()` определяет формат синдикации, включая Atom, и форматирует документ соответствующим образом.

Затем к каждому элементу RSS можно обратиться как к ассоциативному массиву через свойство `items`:

```
print "<ul>\n";

foreach ($rss->items as $item) {
    print '<li><a href="' . $item['link'] . '">' . $item['title'] .
          "</a></li>\n";
}

print "</ul>\n";
```

Цикл `foreach` создает неупорядоченный список элементов, при этом каждый аголовок (`title`) элемента связывается с URL-адресом для чтения полной статьи

(рис. 12.1). Кроме обязательных полей `title` и `link`, элемент может содержать необязательное поле `description` с кратким описанием.

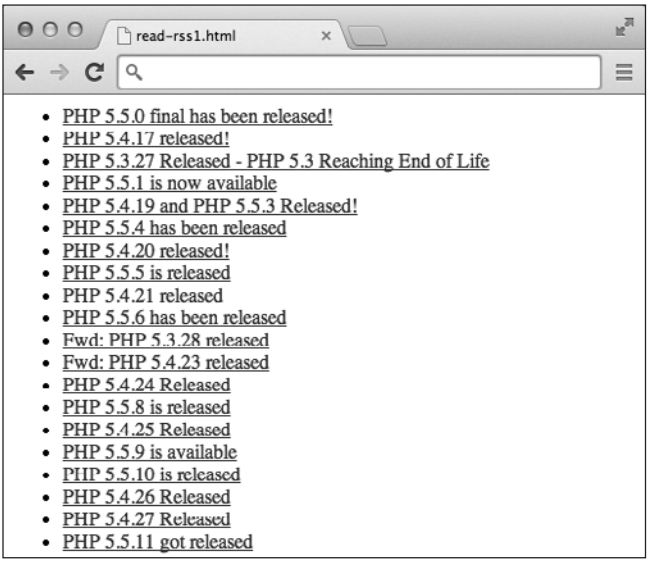

Рис. 12.1. RSS-канал php.announce

Каждый канал также содержит общую информацию о поставляемых данных (рис. 12.2). Для получения этих данных обратитесь к атрибуту `channel`:

```
$feed = 'http://news.php.net/group.php?group=php.announce&format=rss';
$rss = fetch_rss($feed);

print "<ul>\n";

foreach ($rss->channel as $key => $value) {
    print "<li>$key: $value</li>\n";
}

print "</ul>\n";
```

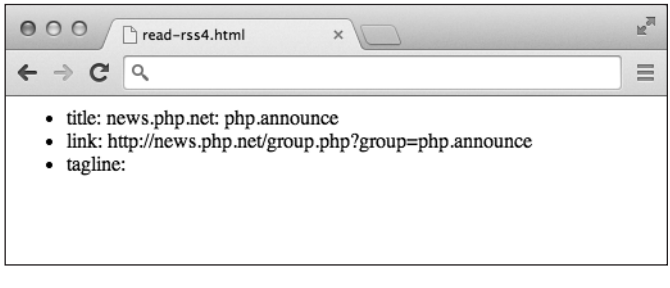

Рис. 12.2. Информация о канале RSS php.announce

См. также

Домашняя страница MagpieRSS; Википедия — дополнительная информация о RSS.

12.13. Генерирование каналов RSS

Задача

Требуется сгенерировать канал RSS на основе ваших данных, чтобы организовать синдикацию контента.

Решение

Воспользуйтесь следующим классом:

```
class rss2 extends DOMDocument {
    private $channel;

    public function __construct($title, $link, $description) {
        parent::__construct();
        $this->formatOutput = true;

        $root = $this->appendChild($this->createElement('rss'));
        $root->setAttribute('version', '2.0');

        $channel= $root->appendChild($this->createElement('channel'));

        $channel->appendChild($this->createElement('title', $title));
        $channel->appendChild($this->createElement('link', $link));
        $channel->appendChild($this->createElement('description',
                                                   $description));

        $this->channel = $channel;
    }
    public function addItem($title, $link, $description) {
        $item = $this->createElement('item');
        $item->appendChild($this->createElement('title', $title));
        $item->appendChild($this->createElement('link', $link));
        $item->appendChild($this->createElement('description', $description));
        $this->channel->appendChild($item);
    }
}

$rss = new rss2('Channel Title', 'http://www.example.org',
                'Channel Description');

$rss->addItem('Item 1', 'http://www.example.org/item1',
              'Item 1 Description');
$rss->addItem('Item 2', 'http://www.example.org/item2',
              'Item 2 Description');

print $rss->saveXML();
```

Комментарий

Формат RSS построен на базе XML, так что вы можете пользоваться всеми средствами генерирования XML из расширения DOM.

Приведенный в Решении код расширяет класс `DOMDocument` для построения дерева DOM с созданием элементов и присоединением их в соответствующей структуре.

Конструктор класса инициализирует элементы `<rss>` и `<channel>`. Он получает три аргумента: название канала, ссылку и описание:

```
public function __construct($title, $link, $description) {
        parent::__construct();
        $this->formatOutput = true;

        $root = $this->appendChild($this->createElement('rss'));
        $root->setAttribute('version', '2.0');

        $channel= $root->appendChild($this->createElement('channel'));

        $channel->appendChild($this->createElement('title', $title));
        $channel->appendChild($this->createElement('link', $link));
        $channel->appendChild($this->createElement('description',
                                                   $description));

        $this->channel = $channel;
    }
```

Вызов `parent::__construct()` выполняет конструктор `DOMDocument::__construct()`. Теперь можно переходить к построению самого документа.

Сначала задайте атрибуту `formatOutput` значение `true`. При этом в выходные данные добавляются отступы и разрывы строк, чтобы было проще читать.

Затем создайте корневой элемент документа `rss` и задайте его атрибуту `version` значение 2.0, потому что эти данные поставляются в формате RSS 2.0.

Все непосредственные данные размещаются в элементе `channel` под узлом `rss`, так что на следующем шаге следует взять этот элемент и инициализировать его дочерние элементы `title`, `link` и `description`.

Данные берутся из аргументов, переданных конструктору. При этом используется полезная особенность метода `createElement()`, который позволяет задать имя элемента и текстовый узел с данными за один вызов — это расширение спецификации DOM для PHP.

Наконец, элемент `channel` сохраняется, чтобы к нему было удобнее обращаться в будущем.

Когда основной контент будет определен, используйте метод `addItem()` для добавления элементов `item`:

```
public function addItem($title, $link, $description) {
    $item = $this->createElement('item');
```

```
        $item->appendChild($this->createElement('title', $title));
        $item->appendChild($this->createElement('link', $link));
        $item->appendChild($this->createElement('description', $description));
        $this->channel->appendChild($item);
    }
```

Так как элементы `item` содержат те же данные, что и `channel`, этот код почти идентичен коду в конструкторе.

`title`, `link` и `description` являются обязательными элементами канала, для `item`. Единственное требование заключается в том, что элемент `item` должен содержать *либо* заголовок (`title`), *либо* описание (`description`).

Для простоты в этом коде используются все три элемента. Кроме того, он не предоставляет возможностей для добавления дополнительных элементов `channel` или `item`, например даты публикации или идентификатора GUID, однозначно описывающего статью.

Через 43 строки простой класс RSS 2.0 завершается. Используйте его следующим образом:

```
$rss = new rss2('Channel Title', 'http://www.example.org',
                'Channel Description');

$rss->addItem('Item 1', 'http://www.example.org/item1',
              'Item 1 Description');
$rss->addItem('Item 2', 'http://www.example.org/item2',
              'Item 2 Description');

print $rss->saveXML();
```

```
<?xml version="1.0"?>
<rss version="2.0">
  <channel>
    <title>Channel Title</title>
    <link>http://www.example.org</link>
    <description>Channel Description</description>
    <item>
      <title>Item 1</title>
      <link>http://www.example.org/item1</link>
      <description>Item 1 Description</description>
    </item>
    <item>
      <title>Item 2</title>
      <link>http://www.example.org/item2</link>
      <description>Item 2 Description</description>
    </item>
  </channel>
</rss>
```

Создайте новый экземпляр класса `rss2` и передайте данные канала. Затем вызовите метод `addItem()` для добавления отдельных статей в канал. Когда данные будут готовы, класс можно преобразовать в XML вызовом родительского метода `DOMDocument::saveXML()`.

12.14. Генерирование каналов Atom

Задача

Требуется сгенерировать канал Atom на основе ваших данных, чтобы организовать синдикацию контента.

Решение

Воспользуйтесь следующим классом:

```
class atom1 extends DOMDocument {
    private $ns;

    public function __construct($title, $href, $name, $id) {
        parent::__construct();
        $this->formatOutput = true;

        $this->ns = 'http://www.w3.org/2005/Atom';

        $root = $this->appendChild($this->createElementNS($this->ns, 'feed'));

        $root->appendChild($this->createElementNS($this->ns, 'title', $title));
        $link = $root->appendChild($this->createElementNS($this->ns, 'link'));
        $link->setAttribute('href', $href);
        $root->appendChild($this->createElementNS($this->ns, 'updated',
            date(DATE_ATOM)));
        $author = $root->appendChild($this->createElementNS($this->ns,
                                                            'author'));

        $author->appendChild($this->createElementNS($this->ns, 'name', $name));
        $root->appendChild($this->createElementNS($this->ns, 'id', $id));
    }

    public function addEntry($title, $link, $summary) {
        $entry = $this->createElementNS($this->ns, 'entry');
        $entry->appendChild($this->createElementNS($this->ns, 'title', $title));
        $entry->appendChild($this->createElementNS($this->ns, 'link', $link));

        $id = uniqid('http://example.org/atom/entry/ids/');
        $entry->appendChild($this->createElementNS($this->ns, 'id', $id));

        $entry->appendChild($this->createElementNS($this->ns, 'updated',
            date(DATE_ATOM)));
        $entry->appendChild($this->createElementNS($this->ns, 'summary',
            $summary));

        $this->documentElement->appendChild($entry);
    }
}
$atom = new atom1('Channel Title', 'http://www.example.org',
                'John Quincy Atom', 'http://example.org/atom/feed/ids/1');
```

```
$atom->addEntry('Item 1', 'http://www.example.org/item1',
              'Item 1 Description', 'http://example.org/atom/entry/ids/1');

$atom->addEntry('Item 2', 'http://www.example.org/item2',
              'Item 2 Description', 'http://example.org/atom/entry/ids/2');

print $atom->saveXML();
```

Комментарий

Класс `atom1` имеет практически такую же структуру, как класс `rss2` из Рецепта 12.13. За более подробным описанием общей структуры кода и поведения расширения DOM обращайтесь к разделу «Комментарий» Рецепта 12.13. А здесь будут рассмотрены различия между RSS и Atom, а также соответствующие изменения, вносимые в класс.

Спецификация Atom сложнее спецификации RSS. Она требует, чтобы элементы размещались в пространстве имен, заставляет генерировать уникальные идентификаторы для канала и отдельных статей, а также даты их последнего обновления.

Кроме того, несмотря на общее сходство структуры, в Atom используется другая терминология. Корневому элементу присваивается имя `feed`, а отдельным статьям — имя `entry`. Описание канала необязательно, но автор должен быть указан. Внутри статей элемент `description` содержит краткую сводку.

Наконец, концепция элемента `channel` отсутствует. И данные канала и статьи располагаются непосредственно под элементом документа.

Обновленная версия конструктора выглядит так:

```
public function __construct($title, $href, $name, $id) {
        parent::__construct();
        $this->formatOutput = true;

        $this->ns = 'http://www.w3.org/2005/Atom';

        $root = $this->appendChild($this->createElementNS($this->ns, 'feed'));

        $root->appendChild(
            $this->createElementNS($this->ns, 'title', $title));
        $link = $root->appendChild(
            $this->createElementNS($this->ns, 'link'));
        $link->setAttribute('href', $href);
        $root->appendChild($this->createElementNS(
            $this->ns, 'updated', date(DATE_ATOM)));
        $author = $root->appendChild(
            $this->createElementNS($this->ns, 'author'));
        $author->appendChild(
            $this->createElementNS($this->ns, 'name', $name));
        $root->appendChild(
            $this->createElementNS($this->ns, 'id', $id));
    }
```

Все элементы Atom существуют в пространстве имен XML *http://www.w3.org/2005/Atom*. Соответственно, все методы `atom1` используют метод `DOMDocument::createElementNS()`, который представляет собой версию `DOMDocument::createElement()` с пространством имен. Пространство имен Atom хранится в `atom1::ns`, что позволяет легко обратиться к нему при необходимости.

Конструктор получает четыре аргумента: название, ссылку, имя автора и идентификатор канала. Значения `title` и `id` определяются по аналогии с элементами `channel` RSS. С другой стороны, ссылка задается как атрибут `href` элемента `link`, а имя является потомком элемента `author`.

Кроме того, существует элемент `updated`, которому присваивается время последнего обновления. В приведенном примере он инициализируется текущим временем и форматируется с использованием встроенной константы PHP `DATE_ATOM`.

Метод `addItem()` переименовывается в `addEntry()` в соответствии со спецификацией Atom:

```
public function addEntry($title, $link, $summary, $id) {
        $entry = $this->createElementNS($this->ns, 'entry');
        $entry->appendChild(
            $this->createElementNS($this->ns, 'title', $title));
        $entry->appendChild(
            $this->createElementNS($this->ns, 'link', $link));
        $entry->appendChild(
            $this->createElementNS($this->ns, 'id', $id));
        $entry->appendChild(
            $this->createElementNS($this->ns, 'updated', date(DATE_ATOM)));
        $entry->appendChild(
            $this->createElementNS($this->ns, 'summary', $summary));
        $this->documentElement->appendChild($entry);
    }
```

Конструктор очень похож на своего предшественника, если не считать добавления новых элементов (таких как `id` и `updated`).

В совокупности все работает примерно так:

```
$atom = new atom1('Channel Title', 'http://www.example.org',
                'John Quincy Atom', 'http://example.org/atom/feed/ids/1');

$atom->addEntry('Item 1', 'http://www.example.org/item1',
              'Item 1 Description', 'http://example.org/atom/entry/ids/1');

$atom->addEntry('Item 2', 'http://www.example.org/item2',
              'Item 2 Description', 'http://example.org/atom/entry/ids/2');

print $atom->saveXML();
```

```
<?xml version="1.0"?>
<feed xmlns="http://www.w3.org/2005/Atom">
  <title>Channel Title</title>
  <link href="http://www.example.org"/>
  <updated>2006-10-23T22:33:59-07:00</updated>
```

```
  <author>
    <name>John Quincy Atom</name>
  </author>
  <id>http://example.org/atom/feed/ids/1</id>
  <entry>
    <title>Item 1</title>
    <link>http://www.example.org/item1</link>
    <id>http://example.org/atom/entry/ids/1</id>
    <updated>2014-10-23T20:23:32-07:00</updated>
    <summary>Item 1 Description</summary>
  </entry>
  <entry>
    <title>Item 2</title>
    <link>http://www.example.org/item2</link>
    <id>http://example.org/atom/entry/ids/2</id>
    <updated>2014-10-23T21:53:44-07:00</updated>
    <summary>Item 2 Description</summary>
  </entry>
</feed>
```

Как и класс `rss2`, `atom1` реализует только малое подмножество полной спецификации. Этого достаточно для генерирования действительной поставки данных, но если вам понадобится что-то большее, придется заняться расширением класса.

См. также

Домашняя страница Atom; вики Atom; дополнительная информация о формате Atom.

13 Автоматизация в веб-приложениях

Как правило, PHP является частью веб-сервера и занимается отправкой контента браузерам. Даже при запуске из командной строки PHP обычно выполняет некую задачу, а затем выводит результат. Однако PHP может принести пользу и в роли веб-клиента, выполняя загрузку данных по URL-адресам и выполняя необходимые операции с содержимым. Тема загрузки данных по URL-адресам из кода PHP рассматривается в главе 14, а в этой главе вы узнаете, как обрабатываются полученные данные.

Рецепты 13.1–13.6 помогут вам в обработке содержимого страниц. Рецепт 13.1 демонстрирует, как выделить некоторые слова в странице цветом, например для пометки условий поиска. «Очистка» разметки HTML упростит ее разбор и повысит степень соответствия стандартам; эта тема рассматривается в Рецепте 13.2. В Рецепте 13.3 приведена функция для поиска всех ссылок на странице, важный структурный компонент поискового бота или программы проверки ссылок. Преобразования между простым текстом и разметкой HTML рассматриваются в Рецептах 13.4 и 13.5. Рецепт 13.6 показывает, как удалить из веб-страницы все теги HTML и PHP. В Рецептах 13.7 и 13.8 обсуждаются возможности взаимодействия между PHP и JavaScript. В Рецепте 13.7 рассматривается использование PHP для ответов на запросы, выдаваемые из JavaScript, с учетом кэширования и возможности использования альтернативных типов содержимого. Рецепт 13.8 предоставляет полноценный пример интеграции PHP-JavaScript с использованием популярного и мощного инструментария jQuery.

В двух примерах программ используется функция извлечения ссылок из Рецепта 13.3. Программа в Рецепте 13.9 сканирует ссылки на странице и сообщает, какие из них остаются действительными, какие были перемещены, а какие перестали работать. Программа в Рецепте 13.10 выводит информацию об актуальности ссылок. Она сообщает время последней модификации страницы, на которую ведет ссылка, и была ли эта страница перемещена.

13.1. Пометки в веб-страницах

Задача

Требуется вывести веб-страницу (например, результаты поиска) с цветовым выделением некоторых слов.

Решение

Постройте массив с заменой для каждого выделяемого слова. Затем разбейте страницу на «элементы HTML» и «текст между элементами HTML» и примените замену только к тексту между элементами HTML. В листинге 13.1 цветовое выделение слов, находящихся в `$words`, применяется к разметке HTML из `$body`.

Листинг 13.1. Разметка веб-страницы

```
$body = '
<p>I like pickles and herring.</p>

<a href="pickle.php"><img src="pickle.jpg"/>A pickle picture</a>

I have a herringbone-patterned toaster cozy.

<herring>Herring is not a real HTML element!</herring>
';

$words = array('pickle','herring');
$replacements = array();
foreach ($words as $i => $word) {
    $replacements[] = "<span class='word-$i'>$word</span>";
}

// Страница разбивается по разделителям, которые
// приближенно определяются как элементы HTML.
$parts = preg_split("{(<(?:\"[^\"]*\"|'[^']*'|[^'\">])*>)}",
                    $body,
                    -1,  // Количество фрагментов не ограничивается
                    PREG_SPLIT_DELIM_CAPTURE);
foreach ($parts as $i => $part) {
    // Пропустить, если фрагмент является элементом HTML
    if (isset($part[0]) && ($part[0] == '<')) { continue; }
    // Заключить слова в теги <span/>
    $parts[$i] = str_replace($words, $replacements, $part);
}
// Восстановление тела страницы
$body = implode('',$parts);

print $body;
```

Комментарий

Листинг 13.1 выводит следующий результат:

```
<p>I like <span class='word-0'>pickle</span>s and <span class='word-1'>⏎
herring</span>.</p>
```

```
<a href="pickle.php"><img src="pickle.jpg"/>A <span class='word-0'>pickle</span>⏎
 picture</a>

I have a <span class='word-1'>herring</span>bone-patterned toaster cozy.

<herring>Herring is not a real HTML element!</herring>
```

Каждое слово из `$words` (`pickle` и `herring`) заключается в теги `<span/>` с определенным атрибутом `class`. Таблица стилей CSS используется для связывания с этим классом конкретных визуальных атрибутов, например ярко-желтого фона или рамки.

Регулярное выражение в листинге 13.1 «нарезает» `$body` на последовательность фрагментов, разделенных элементами HTML. Это позволяет ограничить замену текстом между элементами HTML и исключить из нее элементы HTML и атрибуты, в значениях которых может присутствовать условие поиска. Регулярное выражение неплохо справляется с поиском элементов HTML, но в аномальной, плохо сформированной разметке с непарными или неэкранированными кавычками оно может запутаться.

Так как функция `str_replace()` учитывает регистр символов, заменяются только строки, точно соответствующие словам из `$words`. Последнее вхождение `Herring` в листинге 13.1 не выделяется, потому что оно начинается с прописной буквы. Чтобы поиск выполнялся без учета регистра, следует переключиться с `str_replace()` на регулярные выражения (использовать `str_ireplace()` не удастся, потому что замена должна сохранять регистр совпавшей части). В листинге 13.2 приведен измененный код, использующий регулярные выражения для выполнения замены.

Листинг 13.2. Разметка веб-страницы с использованием регулярных выражений

```
$body = '
<p>I like pickles and herring.</p>

<a href="pickle.php"><img src="pickle.jpg"/>A pickle picture</a>

I have a herringbone-patterned toaster cozy.

<herring>Herring is not a real HTML element!</herring>
';

$words = array('pickle','herring');
$patterns = array();
$replacements = array();
foreach ($words as $i => $word) {
    $patterns[] = '/' . preg_quote($word) .'/i';
    $replacements[] = "<span class='word-$i'>\\0</span>";
}

// Страница разбивается по разделителям, которые
// приближенно определяются как элементы HTML.
$parts = preg_split("{(<(?:\"[^\"]*\"|'[^']*'|[^'\">])*>)}",
```

```
                    $body,
                    -1,  // Количество фрагментов не ограничивается
                    PREG_SPLIT_DELIM_CAPTURE);
foreach ($parts as $i => $part) {
    // Пропустить, если фрагмент является элементом HTML
    if (isset($part[0]) && ($part[0] == '<')) { continue; }
    // Заключить слова в теги <span/>
    $parts[$i] = preg_replace($patterns, $replacements, $part);
}

// Восстановление тела страницы
$body = implode('',$parts);

print $body;
```

Два основных отличия листинга 13.2 — построение массива `$patterns` в цикле в начале программы и использование `preg_replace()` (с массивом `$patterns`) вместо `str_replace()`. Символ `i` в конце каждого элемента `$patterns` активизирует поиск без учета регистра. `\\0` в замене обеспечивает соответствие регистра замены регистру совпадения.

Переход на регулярные выражения также упрощает предотвращение совпадений в подстроках. Как в листинге 13.1, так и в листинге 13.2 будет выделена подстрока `herring` в `herringbone`. Чтобы этого не произошло, замените команду

```
$patterns[] = '/' . preg_quote($word) .'/i';
```

в листинге 13.2 командой

```
$patterns[] = '/\b' . preg_quote($word) .'\b/i';
```

Дополнительные метасимволы `\b` в шаблоне сообщают `preg_replace()`, что совпадение должно обнаруживаться только для отдельно стоящих слов.

См. также

Документация по функциям `str_replace()`, `str_ireplace()`, `preg_replace()` и `preg_split()`. Если вы с недоверием относитесь к идее использования регулярных выражений для разбора HTML, см. Комментарий к Рецепту 13.2 (в конце).

13.2. Удаление некорректной или нестандартной разметки HTML

Задача

В имеющейся разметке HTML присутствует некорректный синтаксис, который нужно удалить. Подобная чистка упрощает разбор и гарантирует, что создаваемые страницы соответствуют стандартам.

Решение

Воспользуйтесь расширением PHP Tidy. Это расширение при помощи популярной мощной библиотеки HTML Tidy превращает хаотичную мешанину из тегов в синтаксически корректную, соответствующую стандартам разметку HTML или XHTML.

Пример 13.3 показывает, как исправить некорректный файл.

Листинг 13.3. Исправление файла HTML с использованием Tidy

```
$fixed = tidy_repair_file('bad.html');
file_put_contents('good.html', $fixed);
```

Комментарий

В библиотеке HTML Tidy за прошедшее время были собраны многочисленные правила и функции для исправления разнообразных нарушений HTML. К счастью, чтобы пользоваться Tidy, вам не нужно знать об этих правилах. Просто передайте имя файла функции `tidy_repair_file()`, и вы получите исправленную версию. Допустим, файл bad.html содержит следующую разметку:

```
<img src="monkey.jpg">
<b>I <em>love</b> monkeys</em>.
```

Листинг 13.3 записывает в файл good.html следующую разметку:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 3.2//EN">
<html>
<head>
<title></title>
</head>
<body>
<img src="monkey.jpg"> <b>I <em>love</em> monkeys</b>.
</body>
</html>
```

Решение Tidy содержит множество конфигурационных параметров, влияющих на генерируемый результат. Параметры передаются `tidy_repair_file()` во втором аргументе, который содержит массив параметров и значений. В листинге 13.4 используется параметр `output-xhtml`, который приказывает Tidy сгенерировать действительную разметку XHTML.

Листинг 13.4. Генерирование XHTML с использованием Tidy

```
$config = array('output-xhtml' => true);
$fixed = tidy_repair_file('bad.html', $config);
file_put_contents('good.xhtml', $fixed);
```

Листинг 13.4 записывает в good.xhtml следующий вывод:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title></title>
</head>
<body>
<img src="monkey.jpg" /> <b>I <em>love</em> monkeys</b>.
</body>
</html>
```

Если источником является строка, а не файл, используйте `tidy_repair_string()`. Функция получает первый аргумент с разметкой HTML, а не имя файла.

Исправленный код XHTML, сгенерированный Tidy, также предоставляет возможность пометки HTML (как в Рецепте 13.1) без использования регулярных выражений. После того как разметка HTML будет преобразована в синтаксически корректный документ XHTML, ее можно корректно обработать и преобразовать функциями PHP DOM. В листинге 13.5 показано, как это делается.

Листинг 13.5. Разметка веб-страницы средствами Tidy и DOM

```
$body = '
<p>I like pickles and herring.</p>

<a href="pickle.php"><img src="pickle.jpg"/>A pickle picture</a>

I have a herringbone-patterned toaster cozy.

<herring>Herring is not a real HTML element!</herring>
';

$words = array('pickle','herring');
$patterns = array();
$replacements = array();
foreach ($words as $i => $word) {
    $patterns[] = '/' . preg_quote($word) . '/i';
    $replacements[] = "<span class='word-$i'>$word</span>";
}

/* Приказать Tidy сгенерировать разметку XHTML */
$xhtml = tidy_repair_string($body, array('output-xhtml' => true));

/* XHTML загружается как документ XML */
$doc = new DOMDocument;
$doc->loadXml($xhtml);

/* При преобразовании HTML в документ XHTML
 * Tidy размещает входную разметку HTML в элементе <body/>
 * документа XHTML */
$body = $doc->getElementsByTagName('body')->item(0);

/* Обход всех текстовых узлов и пометка слов в случае необходимости */
$xpath = new DOMXpath($doc);
foreach ($xpath->query("descendant-or-self::text()", $body) as $textNode) {
    $replaced = preg_replace($patterns, $replacements, $textNode->wholeText);
    if ($replaced !== $textNode->wholeText) {
        $fragment = $textNode->ownerDocument->createDocumentFragment();
        /* Гарантирует, что подузлы <span/> будут созданы правильно */
```

```
        $fragment->appendXml($replaced);
        $textNode->parentNode->replaceChild($fragment, $textNode);
    }
}
/* Построение разметки XHTML, состоящей из всего содержимого <body/> */
$markedup = '';
foreach ($body->childNodes as $node) {
    $markedup .= $doc->saveXml($node);
}
print $markedup;
```

В листинге 13.5 команда `preg_replace()` для добавления разметки применяется ко всем текстовым узлам дерева DOM, полученного при загрузке исправленной версии входной разметки HTML в объект `DOMDocument`. А самое замечательное — мы можем быть уверены в том, что замена применяется только к тексту. Любая некорректная разметка HTML, которая могла бы сбить с толку регулярное выражение для поиска HTML в Рецепте 13.1, будет исправлена Tidy до создания объекта `DOMDocument`.

Недостаток такого решения заключается в том, что в зависимости от степени некорректности входной разметки HTML результат преобразования Tidy может оказаться не таким, какой вы ожидаете. Вот какой результат выдает листинг 13.5:

```
<p>I like <span class="word-0">pickle</span>s and <span class="word-1">herring⏎
</span>.</p>
<a href="pickle.php"><img src="pickle.jpg" />A <span class="word-0">pickle⏎
</span> picture</a> I
have a <span class="word-1">herring</span>bone-patterned toaster cozy. ⏎
<span class="word-1">herring</span> is not a real HTML element!
```

Обратите внимание на последнюю часть: `<span class="word-1">herring</span> is not a real HTML element!`. Поскольку `<herring/>` не является действительным элементом XHTML, Tidy удаляет `<herring>` и `</herring>`, оставляя внутренний текст. Это разумная мера для получения действительного документа XHTML, но если вы к ней не готовы, она может стать неприятным сюрпризом.

См. также

Документация по функциям `tidy_repair_file()` и `tidy_repair_string()`; документация по параметрам конфигурации Tidy. Спецификация XHTML 1.0; Рецепт 12.4 — информация о функциях DOM в PHP.

13.3. Извлечение ссылок из файлов HTML

Задача

Требуется извлечь URL-адреса, указанные в документе HTML.

Решение

Воспользуйтесь Tidy для преобразования документа в XHTML, а затем примените запрос XPath для поиска всех ссылок, как показано в листинге 13.6.

Листинг 13.6. Извлечение ссылок с использованием Tidy и XPath

```
$html=<<<_HTML_
<p>Some things I enjoy eating are:</p>
<ul>
<li><a href="http://en.wikipedia.org/wiki/Pickle">Pickles</a></li>
<li><a href="http://www.eatingintranslation.com/2011/03/great_ny_noodle.html">
                Salt-Baked Scallops</a></li>
<li><a href="http://www.thestoryofchocolate.com/">Chocolate</a></li>
</ul>
_HTML_;

$doc = new DOMDocument();
$opts = array('output-xhtml' => true,
              // Предотвратить путаницу с сущностями
              'numeric-entities' => true);
$doc->loadXML(tidy_repair_string($html,$opts));
$xpath = new DOMXPath($doc);
// Сообщить $xpath о пространстве имен XHTML
$xpath->registerNamespace('xhtml','http://www.w3.org/1999/xhtml');
foreach ($xpath->query('//xhtml:a/@href') as $node) {
    $link = $node->nodeValue;
    print $link . "\n";
}
```

Если расширение Tidy недоступно, воспользуйтесь функцией `pc_link_extractor()`, приведенной в листинге 13.7.

Листинг 13.7. Извлечение ссылок без использования Tidy

```
$html=<<<_HTML_
<p>Some things I enjoy eating are:</p>
<ul>
<li><a href="http://en.wikipedia.org/wiki/Pickle">Pickles</a></li>
<li><a href="http://www.eatingintranslation.com/2011/03/great_ny_noodle.html">
                Salt-Baked Scallops</a></li>
<li><a href="http://www.thestoryofchocolate.com/">Chocolate</a></li>
</ul>
_HTML_;

$links = pc_link_extractor($html);
foreach ($links as $link) {
    print $link[0] . "\n";
}

function pc_link_extractor($html) {
    $links = array();
    preg_match_all('/<a\s+.*?href=[\"\']?([^\"\' >]*)[\"\']?[^>]*>(.*?)<\/a>/i',
                   $html,$matches,PREG_SET_ORDER);
    foreach($matches as $match) {
```

```
        $links[] = array($match[1],$match[2]);
    }
    return $links;
}
```

Комментарий

Документ XHTML, сгенерированный Tidy при включении режима параметра `output-xhtml`, может содержать другие сущности, кроме четырех, определенных в базовой спецификации XML (<, >, &, "). Включение режима `numeric-entities` предотвращает появление других сущностей в сгенерированном документе XHTML. Обнаружив их, `DOMDocument` сообщает о неопределенных сущностях. Альтернатива — оставить численные сущности, но присвоить `$doc->resolveExternals` значение `true`. Тем самым вы приказываете `DOMDocument` получить схему DTD (Document Type Definition), упомянутую в файле, и использовать ее для разрешения сущностей. Tidy генерирует XML с соответствующей схемой DTD. К недостаткам такого подхода следует отнести то, что DTD URL ссылается на ресурс, хранящийся на внешнем веб-сервере, так что вашей программе придется загружать этот ресурс при каждом выполнении.

XHTML представляет собой XML-номенклатуру для выражения разметки HTML. Соответственно, все элементы XHTML (знакомые нам `<a/>`, `<h1/>` и т. д.) существуют в пространстве имен. Чтобы запросы XPath нормально работали, необходимо связать пространство имен с префиксом (эту задачу решает метод `registerNamespace()`), а затем использовать в запросе XPath.

Функция `pc_link_extractor()` — полезная альтернатива на случай недоступности Tidy. Ее регулярное выражение не работает с некоторыми ссылками (например, сконструированными с использованием шестнадцатеричных служебных последовательностей), но с большинством синтаксически корректных конструкций HTML проблем быть не должно. Функция возвращает массив. Каждый элемент массива сам по себе является массивом, состоящим из двух элементов. Первый элемент определяет цель, на которую указывает ссылка, а второй определяет текст, связанный со ссылкой (якорь).

Выражение XPath в листинге 13.6 получает только ссылки, но не якоря. В листинге 13.8 продемонстрировано альтернативное решение, которое выдает как ссылки, так и якоря.

Листинг 13.8. Извлечение ссылок и якорей с использованием Tidy и XPath

```
$html=<<<_HTML_
<p>Some things I enjoy eating are:</p>
<ul>
<li><a href="http://en.wikipedia.org/wiki/Pickle">Pickles</a></li>
<li><a href="http://www.eatingintranslation.com/2011/03/great_ny_noodle.html">
                Salt-Baked Scallops</a></li>
<li><a href="http://www.thestoryofchocolate.com/">Chocolate</a></li>
</ul>
_HTML_;
```

```
$doc = new DOMDocument();
$opts = array('output-xhtml'=>true,
              'wrap' => 0,
              // Предотвратить путаницу с сущностями
              'numeric-entities' => true);
$doc->loadXML(tidy_repair_string($html,$opts));
$xpath = new DOMXPath($doc);
// Сообщить $xpath о пространстве имен XHTML
$xpath->registerNamespace('xhtml','http://www.w3.org/1999/xhtml');
foreach ($xpath->query('//xhtml:a') as $node) {
    $anchor = trim($node->textContent);
    $link = $node->getAttribute('href');
    print "$anchor -> $link\n";
}
```

В листинге 13.8 запрос XPath находит все узлы элементов `<a/>`. Свойство `textContent` узла содержит якорный текст, а ссылка находится в атрибуте `href`. Дополнительный параметр `'wrap' => 0` приказывает Tidy не создавать переносы в сгенерированной разметке XHTML. В результате все якоря остаются в одной строке при извлечении.

См. также

Документация по классу `DOMDocument`, методам `DOMXPath::query()`, `DOMXPath::registerNamespace()`, функциям `tidy_repair_file()` и `preg_match_all()`; Рецепт 13.2 — дополнительная информация о Tidy; XPath; XHTML.

13.4. Преобразование простого текста в HTML

Задача

Требуется преобразовать простой текст в разметку HTML с осмысленным форматированием.

Решение

Начните с кодирования сущностей вызовом `htmlentities()`. Затем преобразуйте текст в различные структуры HTML. Функция `pc_text2html()` из листинга 13.9 содержит базовые преобразования для ссылок и разрывов абзацев.

Листинг 13.9. pc_text2html()

```
function pc_text2html($s) {
  $s = htmlentities($s);
  $grafs = split("\n\n",$s);
  for ($i = 0, $j = count($grafs); $i < $j; $i++) {
```

```
    // Ссылка на URL-адреса http или ftp
    $grafs[$i] = preg_replace('/((ht|f)tp:\/\/[^\s&]+)/',
                              '<a href="$1">$1</a>',$grafs[$i]);

    // Ссылка на адрес электронной почты
    $grafs[$i] = preg_replace('/[^@\s]+@([-a-z0-9]+\.)+[a-z]{2,}/i',
        '<a href="mailto:$1">$1</a>',$grafs[$i]);

    // Начать с нового абзаца
    $grafs[$i] = '<p>'.$grafs[$i].'</p>';
  }
  return implode("\n\n",$grafs);
}
```

Комментарий

Чем больше вы знаете о том, что собой представляет преобразуемый текст, тем лучше пройдет преобразование. Например, если выделение обозначается звездочками (*) или символами косой черты (/), в которые заключаются слова, в функцию можно добавить соответствующие правила, как показано в листинге 13.10.

Листинг 13.10. pc_text2html()

```
$grafs[$i] = preg_replace('/(\A|\s)\*([^*]+)\*(\s|\z)/',
                          '$1<b>$2</b>$3',$grafs[$i]);
$grafs[$i] = preg_replace('{(\A|\s)/([^/]+)/(\s|\z)}',
                          '$1<i>$2</i>$3',$grafs[$i]);
```

См. также

Документация по функции `preg_replace()`.

13.5. Преобразование HTML в простой текст

Задача

Требуется преобразовать разметку HTML в удобочитаемый отформатированный текст.

Решение

Воспользуйтесь классом `html2text`. В листинге 13.11 представлен пример его использования.

Листинг 13.11. Преобразование HTML в простой текст

```
require_once 'class.html2text.inc';
/* Функции file_get_contents() передается путь или URL-адрес
   обрабатываемого файла HTML */
$html = file_get_contents(__DIR__ . '/article.html');
$converter = new html2text($html);
$plain_text = $converter->get_text();
```

Комментарий

В класс `html2text` встроены многочисленные правила форматирования, чтобы сгенерированный простой текст содержал визуальное выделение заголовков, абзацев и т. д. Кроме того, в конец сгенерированного текста включается список всех ссылок, входящих в HTML.

Версия 1.0 класса html2text в нескольких местах использует модификатор /e с preg_replace(). В PHP 5.5 эта возможность считается устаревшей, поэтому вы можете получить предупреждения, если уровень ошибок настроен соответствующим образом. Чтобы избавиться от предупреждений, замените шаблоны, заканчивающиеся на /ie, так, чтобы они заканчивались только на /i, в строках 153, 156, 157, 164 и 170.

См. также

Дополнительная информация о классе `html2text` и ссылки для загрузки.

13.6. Удаление тегов HTML и PHP

Задача

Требуется удалить теги HTML и PHP из строки или файла. Например, вы хотите убедиться в том, что строка не содержит разметки HTML, прежде чем выводить ее, или кода PHP — прежде чем передавать ее `eval()`.

Решение

Воспользуйтесь функцией `strip_tags()` или `filter_var()` для удаления тегов HTML и PHP из строки, как показано в листинге 13.12.

Листинг 13.12. Удаление тегов HTML и PHP

```
$html = '<a href="http://www.oreilly.com">I <b>love computer books.</b></a>';
$html .= '<?php echo "Hello!" ?>';
print strip_tags($html);
print "\n";
print filter_var($html, FILTER_SANITIZE_STRING);
```

Код листинга 13.12 выводит следующий результат:

```
I love computer books.
I love computer books.
```

Для удаления тегов из потока в процессе чтения используется потоковый фильтр `string.strip_tags` (листинг 13.13).

Листинг 13.13. Удаление тегов HTML и PHP из потока

```
$stream = fopen(__DIR__ . '/elephant.html','r');
stream_filter_append($stream, 'string.strip_tags');
print stream_get_contents($stream);
```

Комментарий

И функции `strip_tags()`, и фильтру `string.strip_tags` можно запретить удаление некоторых тегов. Во втором аргументе `strip_tags()` передается строка, содержащая разрешенные теги. Теги задаются без учета регистра символов, и для пар тегов задается только открывающий тег. Например, для удаления из `$html` всех тегов, кроме `<b></b>` и `<i></i>`, используется вызов `strip_tags($html,'<b><i>')`.

С фильтром `string.strip_tags` аналогичная строка передается в четвертом аргументе `stream_filter_append()`. Третий аргумент `stream_filter_append()` управляет тем, применяется ли фильтр при чтении (`STREAM_FILTER_READ`), записи (`STREAM_FILTER_WRITE`) или чтении/записи (`STREAM_FILTER_ALL`). Листинг 13.14 делает то же, что листинг 13.13, но разрешает теги `<b></b><i></i>`.

Листинг 13.14. Удаление некоторых тегов HTML и PHP из потока

```
$stream = fopen(__DIR__ . '/elephant.html','r');
stream_filter_append($stream, 'string.strip_tags',STREAM_FILTER_READ,'b,i');
print stream_get_contents($stream);
```

Функция `stream_filter_append()` также может получать массив имен тегов вместо строки: `array('b','i')` вместо `'<b><i>'`.

Ни strip_tags(), ни потоковый фильтр не удаляют из разрешенных тегов атрибуты. Это означает, что атрибут, изменяющий визуальное оформление (например, style) или выполняющий код JavaScript (любой обработчик события) будет сохранен. Если вы отображаете «усеченный» текст неизвестного происхождения в браузере без предварительного экранирования, это может привести к межсайтовым сценарным атакам.

Более надежное решение, которое обходит все проблемы, связанные с неправильной реакцией `strip_tags()` на «битые» теги или с наличием неудаленных опасных атрибутов — создание «белого списка» безопасных тегов и атрибутов в разметке HTML. При таком подходе вы не удаляете «плохие» элементы (с риском того, что ваш «черный список» не полон), а оставляете «хорошие». Класс `TagStripper` в листинге 13.5 работает именно по такому принципу.

Листинг 13.15. Удаление тегов по «белому списку»

```
class TagStripper {
    protected $allowed =
        array(
              /* Разрешить <a/> только с атрибутом "href" */
              'a'=> array('href' => true),
              /* Разрешить <p/> без атрибутов */
              'p' => array());

    public function strip($html) {
        /* Приказать Tidy сгенерировать разметку XHTML */
        $xhtml = tidy_repair_string($html, array('output-xhtml' => true));

        /* Загрузка некорректной разметки HTML в DOMDocument */
        $dirty = new DOMDocument;
        $dirty->loadXml($xhtml);
        $dirtyBody = $dirty->getElementsByTagName('body')->item(0);

        /* Создание пустого объекта DOMDocument для чистой разметки HTML */
        $clean = new DOMDocument();
        $cleanBody = $clean->appendChild($clean->createElement('body'));

        /* Копирование разрешенных узлов из $dirtyBody в $cleanBody */
        $this->copyNodes($dirtyBody, $cleanBody);

        /* Возвращение содержимого чистого тела документа */
        $stripped = '';
        foreach ($cleanBody->childNodes as $node) {
            $stripped .= $clean->saveXml($node);
        }
        return trim($stripped);
    }

    protected function copyNodes(DOMNode $dirty, DOMNode $clean) {
        foreach ($dirty->attributes as $name => $valueNode) {
            /* Копирование разрешенных атрибутов */
            if (isset($this->allowed[$dirty->nodeName][$name])) {
                $attr = $clean->ownerDocument->createAttribute($name);
                $attr->value = $valueNode->value;
                $clean->appendChild($attr);
            }
        }
        foreach ($dirty->childNodes as $child) {
            /* Копирование разрешенных элементов */
            if (($child->nodeType == XML_ELEMENT_NODE) &&
                (isset($this->allowed[$child->nodeName]))) {
                    $node = $clean->ownerDocument->createElement(
                    $child->nodeName);
                    $clean->appendChild($node);
                    /* Анализ потомков разрешенного элемента */
                    $this->copyNodes($child, $node);
            }
            /* Копирование текста */
            else if ($child->nodeType == XML_TEXT_NODE) {
                $text = $clean->ownerDocument->createTextNode(
                $child->textContent);
```

```
                $clean->appendChild($text);
            }
        }
    }
}
```

Получив входную разметку HTML, метод `strip()` класса в листинге 13.15 преобразует его в XHTML при помощи Tidy, а затем обходит дерево DOM элементов, копируя в новую структуру DOM только разрешенные атрибуты и элементы. Затем метод возвращает содержимое новой структуры DOM.

Пример использования `TagStripper`:

```
$html=<<<_HTML_
<a href=foo onmouseover="bad()" >this is some</b>
stuff
    <p>This should be OK, as <a href="beep">well</a> as this. </p>
<script>alert('whoops')<p>This gets removed.</p></script>

<p>But this <script>bad</script> stuff has the script removed.</p>
_HTML_;

$ts = new TagStripper();
print $ts->strip($html);
```

Выходная разметка выглядит так:

```
<a href="foo">this is some stuff</a>
<p>This should be OK, as <a href="beep">well</a> as this.</p>

<p>But this  stuff has the script removed.</p>
```

Исходный набор разрешенных элементов и атрибутов, определяемый свойством `$allowed` класса `TagStripper` в листинге 13.15, намеренно содержит минимальное количество элементов. Добавьте новые элементы и атрибуты, которые вам потребуются.

См. также

Документация по функциям `strip_tags()`, `stream_filter_append()` и потоковым фильтрам; Рецепт 18.4 — более подробная информация о межсайтовых сценарных атаках.

13.7. Обработка запросов Ajax

Задача

Требуется использовать JavaScript для того, чтобы создать внутристраничные запросы с использованием `XMLHTTPRequest` и отправить данные в ответ на такой запрос.

Решение

Задайте соответствующий заголовок `Content-Type` и сгенерируйте данные, отформатированные соответствующим образом.

Листинг 13.16 отправляет небольшой документ XML в ответ на запрос.

Листинг 13.16. Отправка запроса XML

```
<?php header('Content-Type: text/xml'); ?>
<menu>
 <dish type="appetizer">Chicken Soup</dish>
 <dish type="main course">Fried Monkey Brains</dish>
</menu>
```

В листинге 13.17 функция `json_encode()` используется для отправки ответа JSON.

Листинг 13.17. Отправка ответа JSON

```
$menu = array();
$menu[] = array('type' => 'appetizer',
                'dish' => 'Chicken Soup');
$menu[] = array('type' => 'main course',
                'dish' => 'Fried Monkey Brains');
header('Content-Type: application/json');
print json_encode($menu);
```

Комментарий

С точки зрения PHP отправка ответа запросу на базе `XMLHTTPRequest` ничем не отличается от любого другого ответа. Вы отправляете все необходимые заголовка, а затем выдаете некий текст. Впрочем, кое-что все же отличается: сами заголовки, а обычно еще и внешний вид текста.

Формат JSON особенно удобен для ответов такого рода, потому что работать с данными в формате JSON из JavaScript проще простого. Вывод листинга 13.17 выглядит так:

```
[{"type":"appetizer","dish":"Chicken Soup"},
 {"type":"main course","dish":"Fried Monkey Brains"}]
```

Здесь закодирован двухэлементный массив хешей JavaScript. Функция `json_encode()` вместе с парной `json_decode()` позволяет легко преобразовать структуры данных PHP (скаляры, массивы и объекты) в строки JSON, и наоборот.

При использовании подобных ответов также важно обратить внимание на кэширование. В разных браузерах используются разные нетривиальные стратегии кэширования запросов, выданных из JavaScript. Если ваши ответы отправляют динамические данные (как это обычно происходит), скорее всего, кэшировать их не следует. Два основных инструмента «антикэширования» — заголовки и модификация URL. В листинге 13.19 приведен полный набор заголовков, которые можно выдать из PHP для предотвращения кэширования ответа браузером.

Листинг 13.18. Заголовки для предотвращения кэширования

```
header("Expires: 0");
header("Last-Modified: " . gmdate("D, d M Y H:i:s") . " GMT");
header("Cache-Control: no-store, no-cache, must-revalidate");
// Заголовок для IE
header("Cache-Control: post-check=0, pre-check=0", false);
// Для HTTP/1.0
header("Pragma: no-cache");
```

Другое средство предотвращения кэширования — модификация URL — требует содействия со стороны кода JavaScript, выдающего запрос. В конец строки каждого запроса добавляется пара «имя/значение» с произвольным значением. URL-адреса всех выдаваемых запросов отличаются, что предотвращает нежелательное вмешательство механизма кэширования. Для генерирования таких значений удобно воспользоваться функцией JavaScript `Math.random()`.

См. также

Документация по функции `header()`; дополнительная информация о `XMLHTTPRequest`, JSON и расширении `json`. Хорошее введение в тему кэширования HTTP содержится в книге Майкла Рэдвина «HTTP Caching and Cache-Busting for Content Publishers». В разделе 13 документа RFC 2616 кэширование HTTP описано во всех подробностях.

13.8. Интеграция с JavaScript

Задача

Часть страницы должна обновляться данными со стороны сервера без перезагрузки всей страницы (например, список должен заполняться результатами поиска).

Решение

Воспользуйтесь инструментарием JavaScript (таким как jQuery) для такой организации клиентской части, чтобы некоторое действие пользователя (например, нажатие кнопки) инициировало запрос к серверу. Напишите код PHP, генерирующий запрос с правильными данными; затем воспользуйтесь инструментарием JavaScript для правильного размещения результатов в странице.

В листинге 13.19 показан простой документ HTML, который загружает jQuery и код из листинга 13.20. В листинге 13.20 содержится код «прослойки» JavaScript, которая отправляет запрос серверу при нажатии кнопки Search и размещает результаты в правильном месте страницы. В листинге 13.21 содержится код PHP, который выполняет поиск и отправляет ответ в формате JSON.

Листинг 13.19. Базовая разметка HTML для интеграции с JavaScript

```
<!-- Загрузка jQuery -->
<script type="text/javascript"
        src="//code.jquery.com/jquery-1.9.1.min.js"></script>
<!-- Загрузка нашего кода JavaScript -->
<script type="text/javascript" src="search.js"></script>
<!-- Элементы ввода -->
<input type="text" id="q" />
<input type="button" id="go" value="Search"/>
<hr/>
<!-- Для размещения вывода -->
<div id="output"></div>
```

Листинг 13.20. «Прослойка» для интеграции с JavaScript

```
// Код выполняется при загрузке страницы
$(document).ready(function() {
    // Функция search() вызывается при нажатии кнопки 'go'
    $("#go").click(search);
});
function search() {
    // Содержимое текстового поля
    var q = $("#q").val();
    // Отправка запроса на сервер.
    // Первый аргумент определяет путь к странице поиска.
    // Второй аргумент передает параметр строки запроса.
    // Третий аргумент определяет функцию, выполняемую с результатами.
    $.get('/search.php', { 'q': q }, showResults);
}

// Обработка результатов
function showResults(data) {
    var html = '';
    // Если имеются результаты...
    if (data.length > 0) {
        html = '<ul>';
        // Они используются для построения списка
        for (var i in data) {
            var escaped = $('<div/>').text(data[i]).html();
            html += '<li>' + escaped + '</li>';
        }
        html += '</ul>';
    } else {
        html = 'No results.';
    }
    // Размещение разметки HTML на странице
    $("#output").html(html);
}
```

Листинг 13.21. Код PHP, генерирующий ответ для JavaScript

```
$results = array();
$q = isset($_GET['q']) ? $_GET['q'] : '';

// Подключение к базе данных из главы 10
```

```
$db = new PDO('sqlite:/tmp/zodiac.db');

// Выполнение запроса
$st = $db->prepare('SELECT symbol FROM zodiac WHERE planet LIKE ? ');
$st->execute(array($q.'%'));
// Построение массива результатов
while ($row = $st->fetch()) {
    $results[] = $row['symbol'];
}

if (count($results) == 0) {
    $results[] = "No results";
}

// Выдача заголовков для предотвращения кэширования
header("Expires: 0");
header("Last-Modified: " . gmdate("D, d M Y H:i:s") . " GMT");
header("Cache-Control: no-store, no-cache, must-revalidate");

// Параметры для IE
header("Cache-Control: post-check=0, pre-check=0", false);

// Для HTTP/1.0
header("Pragma: no-cache");

// Ответ в формате JSON
header('Content-Type: application/json');

// Вывод данных JSON
print json_encode($results);
```

Комментарий

Разметка HTML в листинге 13.19 минимальна: она состоит из нескольких элементов и вызовов для загрузки внешних сценариев. Код JavaScript рекомендуется отделять от HTML — по аналогии с отделением логики представления от бизнес-логики на стороне сервера. Первый тег `<script/>` в листинге 13.19 загружает jQuery из CDN. Это удобно, если вы не хотите устанавливать jQuery на своем сервере. Символы `//` в начале URL-адреса — удобный прием, чтобы решение работало как на страницах `http`, так и на страницах `https`.

Второй тег должен содержать ссылку на каталог, в котором вы разместили код из листинга 13.20. Эти функции JavaScript связывают элементы HTML в листинге 13.19 с кодом на стороне сервера в листинге 13.21. Первый вызов `$(document).ready` сообщает браузеру: «Когда загрузка страницы завершится, выполнить код JavaScript, который обеспечит выполнение функции `search()` при нажатии кнопки». У веб-страницы, «нафаршированной» кодом JavaScript, нет четкой процедурной последовательности выполнения от начала до конца. Вместо этого страница предоставляет в распоряжение пользователя различные способы взаимодействия — нажатие кнопок, ввод текста в полях, щелчки на ссылках и т. д. Обычно код JavaScript создает различные обработчики событий — функции, выполняемые как реакция на нажатия, ввод и другие события.

В листинге 13.20 функция `search()` использует функцию jQuery `$.get` для отправки запроса серверу, с передачей содержимого текстового поля в параметре строки запроса `q`. Вторй аргумент `$.get` означает, что при поступлении запроса его следует передать функции `showResults()`.

Функция `showResults()`, в свою очередь, получает эти результаты и строит по ним список HTML. После того как список будет построен, разметка HTML заносится в тег `<div/> output`.

Вероятно, код листинга 13.21 является самой знакомой частью этой тройки. Он очень похож на любой другой сценарий PHP «провести поиск в базе данных на основании данных, введенных пользователем», если не считать способа возвращения результатов. Вместо вывода HTML средства, описанные в Рецепте 13.7, используются для возвращения некэшируемого ответа JSON.

Приложения, основанные на операциях на стороне клиента, запрограммированные на JavaScript, требуют другой парадигмы программирования по сравнению с типичными приложениями PHP. Вместо генерирования целых динамических страниц вам приходится заниматься генерированием блоков динамических данных для вывода или удобной обработки логикой на стороне клиента. Инструментарий jQuery предоставляет мощную платформу для построения таких приложений. Он ограждает вас от многих хлопотных аспектов JavaScript-программирования — межбраузерных несовместимостей, низкоуровневой реализации асинхронного ввода/вывода и других рутинных задач.

См. также

Рецепт 13.7 — отправка ответов JSON; дополнительная информация о jQuery.

13.9. Программа: поиск устаревших ссылок

Программа stale-links.php в листинге 13.22 строит список ссылок в странице и выводит информацию об их состоянии (рабочая ссылка, нерабочая ссылка, ссылка перемещена). При запуске программе передается URL-адрес для проверки ссылок:

```
http://oreilly.com: OK
https://members.oreilly.com: MOVED: https://members.oreilly.com/account/login
http://shop.oreilly.com/basket.do: OK
http://shop.oreilly.com: OK
http://radar.oreilly.com: OK
http://animals.oreilly.com: OK
http://programming.oreilly.com: OK
...
```

Программа stale-links.php использует расширение cURL для получения вебстраниц (см. листинг 13.22). Сначала она загружает URL-адрес, указанный в командной строке. Когда страница будет загружена, программа использует средства XPath из Рецепта 13.3 для получения списка ссылок на странице. Затем после

включения в начало каждой ссылки базового URL-адреса (если это необходимо) программа обращается по ссылке. Так как нас интересуют только заголовки ответов, вместо GET используется метод HEAD, для чего устанавливается режим `CURLOPT_NOBODY`. Установка режима `CURLOPT_HEADER` приказывает `curl_exec()` включить заголовки ответа в возвращаемую строку. В зависимости от кода ответа выводится статус ссылки и ее новое местонахождение в случае перемещения.

Листинг 13.22. stale-links.php

```
if (! isset($_SERVER['argv'][1])) {
    die("No URL provided.\n");
}

$url = $_SERVER['argv'][1];

// Загрузка страницы
list($page,$pageInfo) = load_with_curl($url);

if (! strlen($page)) {
    die("No page retrieved from $url");
}

// Преобразование в XML для простоты разбора
$opts = array('output-xhtml' => true,
              'numeric-entities' => true);
$xml = tidy_repair_string($page, $opts);
$doc = new DOMDocument();
$doc->loadXML($xml);
$xpath = new DOMXPath($doc);
$xpath->registerNamespace('xhtml','http://www.w3.org/1999/xhtml');

// Вычисление базового URL-адреса для относительных ссылок
$baseURL = '';
// Проверить наличие в странице <base href=""/>
$nodeList = $xpath->query('//xhtml:base/@href');
if ($nodeList->length == 1) {
    $baseURL = $nodeList->item(0)->nodeValue;
}
// Тег <base href=""/> отсутствует, построить базовый URL на основе $url
else {
    $URLParts = parse_url($pageInfo['url']);
    if (! (isset($URLParts['path']) && strlen($URLParts['path']))) {
        $basePath = '';
    } else {
        $basePath = preg_replace('#/[^/]*$#','',$URLParts['path']);
    }
    if (isset($URLParts['username']) || isset($URLParts['password'])) {
        $auth = isset($URLParts['username']) ? $URLParts['username'] : '';
        $auth .= ':';
        $auth .= isset($URLParts['password']) ? $URLParts['password'] : '';
        $auth .= '@';
    } else {
        $auth = '';
    }
```

```
    $baseURL = $URLParts['scheme'] . '://' .
               $auth . $URLParts['host'] .
               $basePath;
}

// Сохранение всех посещенных ссылок, чтобы каждая ссылка
// посещалась не более одного раза
$seenLinks = array();
// Получение всех ссылок
$links = $xpath->query('//xhtml:a/@href');
foreach ($links as $node) {
    $link = $node->nodeValue;
    // Разрешение относительных ссылок
    if (! preg_match('#^(http|https|mailto):#', $link)) {
        if (((strlen($link) == 0)) || ($link[0] != '/')) {
            $link = '/' . $link;
        }
        $link = $baseURL . $link;
    }
    // Если ссылка встречалась ранее, она пропускается
    if (isset($seenLinks[$link])) {
        continue;
    }
    // Ссылка помечается как посещенная
    $seenLinks[$link] = true;
    // Вывод посещенной ссылки
    print $link.': ';
    flush();
    list($linkHeaders, $linkInfo) = load_with_curl($link, 'HEAD');
    // Дальнейшие действия определяются кодом ответа.
    // Коды 2xx означают, что со страницей все нормально
    if (($linkInfo['http_code'] >= 200) && ($linkInfo['http_code'] < 300)) {
        $status = 'OK';
    }
    // Коды 3xx - признак перенаправления
    else if (($linkInfo['http_code'] >= 300) && ($linkInfo['http_code'] < 400)) {
        $status = 'MOVED';
        if (preg_match('/^Location: (.*)$/m',$linkHeaders,$match)) {
                $status .= ': ' . trim($match[1]);
        }
    }
    // Другие коды ответов - признак ошибки
    else {
        $status = "ERROR: {$linkInfo['http_code']}";
    }
    // Вывод информации о ссылке
    print "$status\n";
}

function load_with_curl($url, $method = 'GET') {
    $c = curl_init($url);
    curl_setopt($c, CURLOPT_RETURNTRANSFER, true);
    if ($method == 'GET') {
        curl_setopt($c,CURLOPT_FOLLOWLOCATION, true);
    }
```

```
    else if ($method == 'HEAD') {
        curl_setopt($c, CURLOPT_NOBODY, true);
        curl_setopt($c, CURLOPT_HEADER, true);
    }
    $response = curl_exec($c);
    return array($response, curl_getinfo($c));
}
```

13.10. Программа: проверка актуальности ссылок

Листинг 13.23 представляет собой модификацию программы в листинге 13.22, которая выводит список ссылок и время их последнего изменения. Если сервер, на котором находится URL, не предоставляет информацию о времени последнего изменения, программа выводит время запроса URL-адреса. Если программе не удалось успешно загрузить данные по URL-адресу, она выводит код статуса, полученный при попытке обращения. При запуске программе передается URL-адрес для проверки ссылок:

```
http://oreilly.com: OK; Last Modified: Fri, 24 May 2013 18:09:11 GMT
https://members.oreilly.com: MOVED: https://members.oreilly.com/account/login
http://shop.oreilly.com/basket.do: OK
http://shop.oreilly.com: OK
http://radar.oreilly.com: OK; Last Modified: Fri, 24 May 2013 20:40:56 GMT
http://animals.oreilly.com: OK; Last Modified: Fri, 24 May 2013 20:40:18 GMT
http://programming.oreilly.com: OK; Last Modified: Fri, 24 May 2013 20:42:44 GMT
...
```

Приведенные результаты были получены при запуске программы около 20:43 (GMT) 24 мая 2013 года. Если за ссылкой не указывается время последнего изменения, это означает, что сервер не предоставил соответствующую информацию, так что страница, вероятно, является динамической.

Программа проверки актуальности ссылок концептуально почти не отличается от программы поиска устаревших ссылок. Она использует те же средства для извлечения ссылок и страницы и тот же код для загрузки URL.

После того как страница будет загружена, URL-адрес каждой ссылки загружается методом `head`. Однако вместо того, чтобы просто выводить новое местонахождение перемещенных ссылок, программа выводит отформатированную версию заголовка `Last-Modified` (если она доступна).

Листинг 13.23. fresh-links.php

```
error_reporting(E_ALL);

if (! isset($_SERVER['argv'][1])) {
    die("No URL provided.\n");
}
```

```
$url = $_SERVER['argv'][1];

// Загрузка страницы
list($page, $pageInfo) = load_with_curl($url);

if (! strlen($page)) {
    die("No page retrieved from $url");
}

// Преобразование в XML для простоты разбора
$opts = array('output-xhtml' => true,
              'numeric-entities' => true);
$xml = tidy_repair_string($page, $opts);
$doc = new DOMDocument();
$doc->loadXML($xml);
$xpath = new DOMXPath($doc);
$xpath->registerNamespace('xhtml','http://www.w3.org/1999/xhtml');

// Вычисление базового URL-адреса для относительных ссылок
$baseURL = '';
// Проверить наличие в странице <base href=""/>
$nodeList = $xpath->query('//xhtml:base/@href');
if ($nodeList->length == 1) {
    $baseURL = $nodeList->item(0)->nodeValue;
}
// Тег <base href=""/> отсутствует, построить базовый URL на основе $url
else {
    $URLParts = parse_url($pageInfo['url']);
    if (! (isset($URLParts['path']) && strlen($URLParts['path']))) {
        $basePath = '';
    } else {
        $basePath = preg_replace('#/[^/]*$#','',$URLParts['path']);
    }
    if (isset($URLParts['username']) || isset($URLParts['password'])) {
        $auth = isset($URLParts['username']) ? $URLParts['username'] : '';
        $auth .= ':';
        $auth .= isset($URLParts['password']) ? $URLParts['password'] : '';
        $auth .= '@';
    } else {
        $auth = '';
    }
    $baseURL = $URLParts['scheme'] . '://' .
               $auth . $URLParts['host'] .
               $basePath;
}

// Сохранение всех посещенных ссылок, чтобы каждая ссылка
// посещалась не более одного раза
$seenLinks = array();

// Получение всех ссылок
$links = $xpath->query('//xhtml:a/@href');

foreach ($links as $node) {
    $link = $node->nodeValue;
```

```
    // Разрешение относительных ссылок
    if (! preg_match('#^(http|https|mailto):#', $link)) {
        if (((strlen($link) == 0)) || ($link[0] != '/')) {
            $link = '/' . $link;
        }
        $link = $baseURL . $link;
    }
    // Если ссылка встречалась ранее, она пропускается
    if (isset($seenLinks[$link])) {
        continue;
    }
    // Ссылка помечается как посещенная
    $seenLinks[$link] = true;
    // Вывод посещенной ссылки
    print $link.': ';
    flush();

    list ($linkHeaders, $linkInfo) = load_with_curl($link, 'HEAD');
    // Дальнейшие действия определяются кодом ответа.
    // Коды 2xx означают, что со страницей все нормально
    if (($linkInfo['http_code'] >= 200) && ($linkInfo['http_code'] < 300)) {
        $status = 'OK';
    }
    // Коды 3xx - признак перенаправления
    else if (($linkInfo['http_code'] >= 300) && ($linkInfo['http_code'] < 400)) {
        $status = 'MOVED';
        if (preg_match('/^Location: (.*)$/m',$linkHeaders,$match)) {
                $status .= ': ' . trim($match[1]);
        }
    }
    // Другие коды ответов - признак ошибки
    else {
        $status = "ERROR: {$linkInfo['http_code']}";
    }
    if (preg_match('/^Last-Modified: (.*)$/mi', $linkHeaders, $match)) {
        $status .= "; Last Modified: " . trim($match[1]);
    }
    // вывод информации о ссылке
    print "$status\n";
}

function load_with_curl($url, $method = 'GET') {
    $c = curl_init($url);
    curl_setopt($c, CURLOPT_RETURNTRANSFER, true);
    if ($method == 'GET') {
        curl_setopt($c,CURLOPT_FOLLOWLOCATION, true);
    }
    else if ($method == 'HEAD') {
        curl_setopt($c, CURLOPT_NOBODY, true);
        curl_setopt($c, CURLOPT_HEADER, true);
    }
    $response = curl_exec($c);
    return array($response, curl_getinfo($c));
}
```

14 Потребление REST-совместимых API

14.0. Введение

Когда вам потребуется получить прогноз погоды в Нью-Йорке, последние твиты пользователя `@rasmus` или обновить сохраненный файл, вы можете написать короткий сценарий REST для обработки этих данных в удобном формате.

REST — разновидность веб-API, при которой выдаются запросы к URL с использованием традиционных методов HTTP (таких как GET и POST). URL-адрес и тело запроса, часто в формате JSON или XML, описывают ресурс, с которым выполняется операция, а метод сообщает серверу, какую именно операцию следует выполнить.

Метод GET является признаком чтения существующих данных, тогда как метод POST означает, что вы добавляете новый ресурс. Метод PUT используется для замены ресурса или создания ресурса с заданным именем. И разумеется, метод DELETE удаляет ресурс.

Главное преимущество REST заключается в том, что эта технология базируется на существующих стандартах. Многие разработчики знакомы с HTTP или JSON, поэтому они легко и быстро осваивают REST. К недостаткам можно отнести отсутствие стандартной схемы передаваемых или возвращаемых данных. Каждый сайт может использовать тот формат, который ему кажется наиболее подходящим. Для небольших служб это не создает проблем, но при недостаточно серьезном проектировании с увеличением масштаба может повыситься уровень сложности.

Тем не менее формат REST очень популярен, а его простота остается ключевым фактором успеха.

Основы создания запросов REST изложены в Рецепте 14.1.

В рецептах этой главы рассказано о том, как сгенерировать запрос HTTP для нужного вызова REST. Так как данные (почти всегда) возвращаются в одном

из стандартных форматов файлов, нет необходимости объяснять, как происходит разбор результатов. С учетом природы документов REST и того, что схема ответа вам обычно известна, расширения JSON и SimpleXML часто оказываются наилучшим вариантом. Эти расширения рассматриваются в Рецептах 5.7 и 12.3.

В PHP существует много способов обращений к удаленным URL-адресам. Выбор того или иного способа зависит от ваших потребностей в простоте, уровне контроля и портируемости кода. В этой главе рассмотрены три способа: стандартные файловые функции, расширение cURL и класс `HTTP_Request2` из PEAR. Эти три способа обычно делают все, что вам может потребоваться, и по крайней мере один из них будет доступен при любой конфигурации сервера или возможности установки дополнительных расширений. Среди других способов получения данных по удаленным URL-адресам стоит упомянуть расширение pecl_http, которое пока находится на стадии разработки, но предоставляет ряд перспективных возможностей и использование функции `fsockopen()` с открытием сокета для отправки сконструированного вами запроса HTTP.

Стандартные файловые функции — такие, как `file_get_contents()`, — просты и удобны. Они автоматически поддерживают перенаправление, так что если использовать такую функцию для загрузки каталога по адресу *http://www.example.com/people*, а сервер перенаправит вас по адресу *http://www.example.com/people/*, вы получите содержимое индексной страницы каталога, а не сообщение о перемещении URL-адреса. Стандартные файловые функции также работают с HTTP и FTP. Недостаток этого способа — необходимость включения конфигурационной директивы `allow_url_fopen`.

Расширение cURL — мощный и универсальный инструмент. Оно базируется на популярной библиотеке libcurl и предоставляет быстрый, обладающий широкими возможностями настройки механизм для обработки различных сетевых запросов. Если это расширение доступно на вашем сервере, мы рекомендуем использовать его.

Если директива `allow_url_fopen` отключена, а расширение cURL недоступно, вас выручит модуль PEAR `HTTP_Request2`. Как и все модули PEAR, он состоит из обычного кода PHP, так что если вы можете сохранить файл PHP на сервере — используйте этот вариант. `HTTP_Request2` поддерживает практически все, что может потребоваться при запросе удаленного URL-адреса, включая модификацию заголовков и тела запроса, использование произвольного метода и выборку заголовков запросов.

В Рецептах 14.1–14.7 рассказано, как создавать различные типы запросов HTTP, настроить заголовки, метод, тело и временные параметры. Рецепт 14.8 поможет заглянуть во «внутренности» запроса HTTP, чтобы проанализировать заголовки запроса и ответа. Если запрос, выданный из программы, не дает искомых результатов, анализ заголовков часто помогает выяснить суть проблемы.

14.1. Получение данных по URL-адресу методом GET

Задача

Требуется получить данные по URL-адресу, например включить часть одного сайта в контент другого сайта.

Решение

Передайте URL-адрес функции `file_get_contents()`:

```
$page = file_get_contents('http://www.example.com/robots.txt');
```

Или воспользуйтесь расширением cURL:

```
$c = curl_init('http://www.example.com/robots.txt');
curl_setopt($c, CURLOPT_RETURNTRANSFER, true);
$page = curl_exec($c);
curl_close($c);
```

Также можно воспользоваться классом `HTTP_Request2` из репозитория PEAR:

```
require_once 'HTTP/Request2.php';
$r = new HTTP_Request2('http://www.example.com/robots.txt');
$page = $r->send()->getBody();
```

Комментарий

Функция `file_get_contents()`, как и все функции PHP, для работы с файлами использует поддержку *потоков данных* PHP. Это означает, что функция может работать как с локальными файлами, так и с различными сетевыми ресурсами, включая URL-адреса HTTP. Однако существует одна загвоздка — конфигурационная директива `allow_url_fopen` должна быть включена (впрочем, обычно так оно и бывает).

Это позволяет легко загружать документы с удаленных серверов, например документа XML, как в следующем примере:

```
$url = 'http://rss.news.yahoo.com/rss/oddlyenough';
$rss = simplexml_load_file($url);
print '<ul>';
foreach ($rss->channel->item as $item) {
   print '<li><a href="' .
         htmlentities($item->link) .
         '">' .
         htmlentities($item->title) .
         '</a></li>';
}
print '</ul>';
```

Чтобы загрузить страницу, включающую переменные строки запроса, используйте функцию `http_build_query()` для создания строки запроса. Функция получает массив пар «ключ/значение» и возвращает одну строку, в которой все служебные символы экранированы должным образом. При этом вы отвечаете за символ `?` в URL-адресе, начинающем строку запроса. Например:

```
$vars = array('page' => 4, 'search' => 'this & that');
$qs = http_build_query($vars);
$url = 'http://www.example.com/search.php?' . $qs;
$page = file_get_contents($url);
```

Чтобы загрузить защищенную страницу, включите в URL-адрес имя пользователя и пароль. В следующем примере используется имя `david` с паролем `hax0r`:

```
$url = 'http://david:hax0r@www.example.com/secrets.php';
$page = file_get_contents($url);
```

Или с cURL:

```
$c = curl_init('http://www.example.com/secrets.php');
curl_setopt($c, CURLOPT_RETURNTRANSFER, true);
curl_setopt($c, CURLOPT_USERPWD, 'david:hax0r');
$page = curl_exec($c);
curl_close($c);
```

Аналогично с `HTTP_Request2`:

```
require 'HTTP/Request2.php';
$r = new HTTP_Request2('http://www.example.com/secrets.php');
$r->setAuth('david', 'hax0r', HTTP_Request2::AUTH_DIGEST);
$page = $r->send()->getBody();
```

PHP-обертка `http` автоматически отслеживает перенаправления. Функции `file_get_contents()` и `fopen()` поддерживают аргумент *контекста потока*, при помощи которого можно задать параметры загрузки потока. Один из этих параметров — `max_redirects` — задает максимальное количество отслеживаемых перенаправлений.

В следующем примере `max_redirects` присваивается значение 1, которое отключает отслеживание перенаправлений:

```
$url = 'http://www.example.com/redirector.php';
// Определение параметров
$options = array('max_redirects' => 1 );
// Создание контекста с параметрами потока http
$context = stream_context_create(array('http' => $options));
// Параметры передаются file_get_contents. Второй аргумент указывает,
// нужно ли использовать путь include (в нашем случае это не нужно).
print file_get_contents($url, false, $context);
```

Параметр `max_redirects` на самом деле определяет не количество отслеживаемых перенаправлений, а максимальное количество запросов, выдаваемых при отслеживании цепочки перенаправлений. А именно значение 1 приказывает PHP выдать не более одного запроса — то есть не отслеживать перенаправления. Значение 2 приказывает PHP выдать не более двух запросов — то есть отслеживать

не более одного перенаправления (значение 0 ведет себя как значение 1, то есть PHP ограничивается одним запросом).

Если в цепочке перенаправлений PHP приходится выдавать больше запросов, чем разрешает `max_redirects`, то PHP выдает предупреждение.

Расширение cURL отслеживает перенаправления только при включенном режиме `CURLOPT_FOLLOWLOCATION`:

```
$c = curl_init('http://www.example.com/redirector.php');
curl_setopt($c, CURLOPT_RETURNTRANSFER, true);
curl_setopt($c, CURLOPT_FOLLOWLOCATION, true);
$page = curl_exec($c);
curl_close($c);
```

Чтобы задать максимальное количество перенаправлений, отслеживаемых cURL, задайте `CURLOPT_FOLLOWLOCATION` значение `true`, а затем присвойте `CURLOPT_MAXREDIRS` это максимальное значение.

Класс `HTTP_Request2` отслеживает перенаправления, если параметр `follow_redirects` равен `true`, как показано ниже:

```
require 'HTTP/Request2.php';

$r = new HTTP_Request2('http://www.example.com/redirector.php');
$r->setConfig(array(
    'follow_redirects' => true,
    'max_redirects'    => 1
));

$page = $r->send()->getBody();
print $page;
```

Расширение cURL может выполнять ряд других операций при загрузке страницы. Как было показано в предыдущих примерах, при включенном режиме `CURLOPT_RETURNTRANSFER` функция `curl_exec()` возвращает тело запрашиваемой страницы. Если режим `CURLOPT_RETURNTRANSFER` не включен, то `curl_exec()` выводит тело ответа.

Чтобы записать полученную страницу в файл, откройте файловый дескриптор для записи вызовом `fopen()` и присвойте этот дескриптор параметру `CURLOPT_FILE`. В следующем примере расширение cURL используется для сохранения удаленной страницы в локальном файле:

```
$fh = fopen('local-copy-of-files.html','w') or die($php_errormsg);
$c = curl_init('http://www.example.com/files.html');
curl_setopt($c, CURLOPT_FILE, $fh);
curl_exec($c);
curl_close($c);
```

Чтобы передать ресурс cURL и содержимое загруженной страницы функции, назначьте параметру `CURLOPT_WRITEFUNCTION` функцию обратного вызова для этой функции (имя функции или массив, первым элементом которого является объект или строка с именем класса, а вторым — имя метода). Функция обратного вызова (она называется *функцией записи*) должна возвращать количество передан-

ных байтов. Учтите, что для больших ответов функция обратного вызова может вызываться несколько раз, потому что cURL обрабатывает ответ по фрагментам. В следующем примере функция записи cURL используется для сохранения содержимого страницы в базе данных:

```
class PageSaver {
    protected $db;
    protected $page ='';

    public function __construct() {
        $this->db = new PDO('sqlite:./pages.db');
    }

    public function write($curl, $data) {
        $this->page .= $data;
        return strlen($data);
    }

    public function save($curl) {
        $info = curl_getinfo($curl);
        $st = $this->db->prepare('INSERT INTO pages '.
                          '(url,page) VALUES (?,?)');
        $st->execute(array($info['url'], $this->page));
    }
}

// Создание экземпляра PageSaver
$pageSaver = new PageSaver();
// Создание ресурсов cURL
$c = curl_init('http://www.example.com/');
// Назначение функции записи
curl_setopt($c, CURLOPT_WRITEFUNCTION, array($pageSaver,'write'));
// Выполнение запроса
curl_exec($c);
// Сохранение накопленных данных
$pageSaver->save($c);
```

См. также

Рецепт 14.2 — отправка данных по URL-адресу методом POST; документация по функциям `file_get_contents()`, `simplexml_load_file()`, `stream_context_create()`, `curl_init()`, `curl_setopt()`, `curl_exec()`, `curl_getinfo()` и `curl_close()`; класс PEAR `HTTP_Request2`.

14.2. Обращение по URL-адресу с методом POST и данными формы

Задача

Требуется отправить документ методом POST с использованием данных, отформатированных в виде формы HTML.

Решение

Задайте метод и параметры контекста потока при использовании потока `http`:

```
$url = 'http://www.example.com/submit.php';
// Отправленные данные формы, закодированные в формате
// строки запроса с парами "имя/значение"
$body = 'monkey=uncle&amp;rhino=aunt';
$options = array('method' => 'POST',
                 'content' => $body,
                 'header' => 'Content-type: application/x-www-form-urlencoded');
// Создание контекста потока
$context = stream_context_create(array('http' => $options));
// Контекст передается file_get_contents()
print file_get_contents($url, false, $context);
```

При использовании расширения cURL задайте параметры `CURLOPT_POST` и `CURLOPT_POSTFIELDS`:

```
$url = 'http://www.example.com/submit.php';
// Отправленные данные формы, закодированные в формате
// строки запроса с парами "имя/значение"
$body = 'monkey=uncle&rhino=aunt';
$c = curl_init($url);
curl_setopt($c, CURLOPT_POST, true);
curl_setopt($c, CURLOPT_POSTFIELDS, $body);
curl_setopt($c, CURLOPT_RETURNTRANSFER, true);
$page = curl_exec($c);
curl_close($c);
```

При использовании класса `HTTP_Request2` передайте значение `HTTP_Request2::METHOD_POST` методу `setMethod()` и используйте сцепленные вызовы `addPostParameter()` для каждой пары «имя/значение» в отправляемых данных:

```
require 'HTTP/Request2.php';

$url = 'http://www.example.com/submit.php';
$r = new HTTP_Request2($url);

$r->setMethod(HTTP_Request2::METHOD_POST)
    ->addPostParameter('monkey', 'uncle')
    ->addPostParameter('rhino','aunt');

$page = $r->send()->getBody();
```

Комментарий

Отправка запроса методом POST требует особой обработки аргументов форм. Для запросов GET эти аргументы включаются в строку запроса, но для запросов POST они передаются в теле запроса. Дополнительно в запрос должен быть включен заголовок `Content-Length`, который сообщает серверу предполагаемый размер контента в теле запроса.

Несмотря на различия в способе задания метода и контента в теле запроса, каждый из примеров в Решении автоматически добавляет правильный заголовок `Content-Length`.

Если для отправки запроса POST используется контент потока, не забудьте задать параметру `method` значение `post` (с соблюдением регистра символов).

См. также

Рецепт 14.1 — получение данных по URL-адресу методом GET; документация по функции `curl_setopt()` и параметрам потоков; класс PEAR `HTTP_Request2`; RFC 2616.

14.3. Обращение по URL-адресу с произвольным методом и телом POST

Задача

Требуется обратиться по URL-адресу с использованием произвольного метода, например POST, PUT или DELETE. Запрос POST или PUT может содержать отформатированные данные (например, в формате JSON или XML).

Решение

Задайте параметры контекста потока `method`, `header` и `content` при использовании потока `http`:

```
$url = 'http://www.example.com/meals/123';
$header = "Content-Type: application/json";
// Тело запроса в формате JSON
$body = '[{
    "type": "appetizer",
    "dish": "Chicken Soup"
}, {
    "type": "main course",
    "dish": "Fried Monkey Brains"
}]';

$options = array('method' => 'put',
    'header' => $header,
    'content' => $body);
// Создание контекста потока
$context = stream_context_create(array('http' => $options));
// Контекст передается file_get_contents()
print file_get_contents($url, false, $context);
```

При использовании расширения cURL присвойте параметру `CURLOPT_CUSTOMREQUEST` имя метода. Чтобы включить тело запроса, присвойте `CURLOPT_HTTPHEADER` значение `Content-Type`, а само тело присваивается `CURLOPT_POST`:

```
$url = 'http://www.example.com/meals/123';
// Тело запроса в формате JSON
$body = '[{
    "type": "appetizer",
    "dish": "Chicken Soup"
}, {
    "type": "main course",
    "dish": "Fried Monkey Brains"
}]';
$c = curl_init($url);
curl_setopt($c, CURLOPT_CUSTOMREQUEST, 'PUT');
curl_setopt($c, CURLOPT_HTTPHEADER, array('Content-Type: application/json'));
curl_setopt($c, CURLOPT_POSTFIELDS, $body);
curl_setopt($c, CURLOPT_RETURNTRANSFER, true);
$page = curl_exec($c);
curl_close($c);
```

При использовании `HTTP_Request2` вызовите метод `setMethod()` с константой метода, `setHeader()` с `Content-Type`, и `setBody()` с содержимым тела запроса:

```
require 'HTTP/Request2.php';
$url = 'http://www.example.com/meals/123';
// Тело запроса в формате JSON
$body = '[{
    "type": "appetizer",
    "dish": "Chicken Soup"
}, {
    "type": "main course",
    "dish": "Fried Monkey Brains"
}]';
$r = new HTTP_Request2($url);
$r->setMethod(HTTP_Request2::METHOD_PUT);
$r->setHeader('Content-Type', 'application/json');
$r->setBody($body);

$page = $r->send()->getBody();
```

Комментарий

Многие API в стиле REST не ограничиваются методами GET и POST для модификации ресурсов; они также требуют использования методов PUT или DELETE.

Примеры, приведенные в Решении, выдают запросы HTTP PUT с данными в формате JSON. Если ваши данные хранятся в другом формате (например, XML), измените `Content-Type` соответствующим образом. Если у запроса нет тела (например, для запроса HTTP DELETE), задайте только метод.

Метод PUT часто используется для создания или изменения содержимого конкретного ресурса. Расширение cURL поддерживает три специальных параметра, упрощающих выполнение таких операций: `CURLOPT_PUT`, `CURLOPT_INFILE` и `CURLOPT_INFILESIZE`. Чтобы отправить файл методом PUT с использованием cURL, присвойте `CURLOPT_PUT` значение `true`, `CURLOPT_INFILE` — файловый дескриптор,

открытый для отправляемого файла, и `CURLOPT_INFILESIZE` — размер файла. Пример приведен в листинге 14.1.

Листинг 14.1. Отправка файла с использованием расширения cURL и метода PUT

```
$url = 'http://www.example.com/upload.php';
$filename = '/usr/local/data/pictures/piggy.jpg';
$fp = fopen($filename,'r');
$c = curl_init($url);
curl_setopt($c, CURLOPT_PUT, true);
curl_setopt($c, CURLOPT_INFILE, $fp);
curl_setopt($c, CURLOPT_INFILESIZE, filesize($filename));
curl_setopt($c, CURLOPT_RETURNTRANSFER, true);
$page = curl_exec($c);
print $page;
curl_close($c);
```

См. также

Документация по функции `curl_setopt()` и параметрам потоков; класс PEAR `HTTP_Request2`; раздел 5.1.1 документа RFC 2616 — описание методов запросов; список распространенных типов `Content-Type`.

14.4. Обращение по URL-адресу с cookie

Задача

Требуется получить страницу, для которой вместе с запросом необходимо отправить cookie.

Решение

При использовании cURL воспользуйтесь параметром `CURLOPT_COOKIE`:

```
$c = curl_init('http://www.example.com/needs-cookies.php');
curl_setopt($c, CURLOPT_COOKIE, 'user=ellen; activity=swimming');
curl_setopt($c, CURLOPT_RETURNTRANSFER, true);
$page = curl_exec($c);
curl_close($c);
```

При использовании `HTTP_Request2` воспользуйтесь методом `addCookie()`:

```
require 'HTTP/Request2.php';
$r = new HTTP_Request2('http://www.example.com/needs-cookies.php');
$r->addCookie('user', 'ellen');
$r->addCookie('activity', 'swimming');
$page = $r->send()->getBody();
echo $page;
```

Комментарий

Для передачи cookie серверу используется заголовок запроса `Cookie`. Хотя на практике это всего лишь обычный заголовок HTTP, из-за его важности в расширении cURL и в пакете `HTTP_Request2` существуют специальные функции для записи cookie.

В примерах из раздела «Решение» отправляются два cookie: с именем `user` и значением `ellen` и с именем `activity` и значением `swimming`.

Чтобы запросить страницу, которая записывает cookie, а затем выдавать последующие запросы с включением этих недавно записанных cookie, можно воспользоваться хранилищем cookie расширения cURL. При первом запросе параметру `CURLOPT_COOKIEJAR` присваивается имя файла, в котором будут храниться cookie. При последующих запросах присвойте `CURLOPT_COOKIEFILE` то же имя файла; cURL прочитает cookie из файла и отправит их с запросом. Эта возможность особенно полезна для серий запросов, в которых первый запрос осуществляет вход на сайт с записью cookie сеанса или аутентификации, а затем остальные запросы должны включать записанные cookie, чтобы они были признаны действительными.

Пример такой серии запросов:

```
// Временный файл для хранения cookie
$cookie_jar = tempnam('/tmp','cookie');

// Вход
$c = curl_init('https://bank.example.com/login.php?user=donald&password=b1g$');
curl_setopt($c, CURLOPT_RETURNTRANSFER, true);
curl_setopt($c, CURLOPT_COOKIEJAR, $cookie_jar);
$page = curl_exec($c);
curl_close($c);

// Получение баланса
$c = curl_init('http://bank.example.com/balance.php?account=checking');
curl_setopt($c, CURLOPT_RETURNTRANSFER, true);
curl_setopt($c, CURLOPT_COOKIEFILE, $cookie_jar);
$page = curl_exec($c);
curl_close($c);

// Внесение средств
$c = curl_init('http://bank.example.com/deposit.php');
curl_setopt($c, CURLOPT_POST, true);
curl_setopt($c, CURLOPT_POSTFIELDS, 'account=checking&amount=122.44');
curl_setopt($c, CURLOPT_RETURNTRANSFER, true);
curl_setopt($c, CURLOPT_COOKIEFILE, $cookie_jar);
$page = curl_exec($c);
curl_close($c);

// Удаление хранилища cookie
unlink($cookie_jar) or die("Can't unlink $cookie_jar");
```

Будьте внимательны с выбором места для хранилища. Оно должно располагаться в каталоге, доступном веб-серверу для записи, но если файл будет доступен

для чтения другим пользователям, то они смогут похитить учетные данные аутентификации, хранящиеся в cookie.

Класс `HTTP_Request2` предоставляет аналогичный режим отслеживания cookie. Для его включения необходимо вызвать метод `setCookieJar()`. Затем, при выдаче нескольких запросов с тем же объектом `HTTP_Request2`, cookies автоматически переходят между запросами. Пример:

```
require 'HTTP/Request2.php';

$r = new HTTP_Request2;
$r->setCookieJar(true);

// Вход
$r->setUrl('https://bank.example.com/login.php?user=donald&password=b1gmoney$');
$page = $r->send()->getBody();

// Получение баланса
$r->setUrl('http://bank.example.com/balance.php?account=checking');
$page = $r->send()->getBody();

// Внесение средств
$r->setUrl('http://bank.example.com/deposit.php');
$r->setMethod(HTTP_Request2::METHOD_POST)
    ->addPostParameter('account', 'checking')
    ->addPostParameter('amount','122.44');

$page = $r->send()->getBody();
```

См. также

Документация по `curl_setopt()`, классу PEAR `HTTP_Request2`; RFC 6265 и статья Дэвида М. Кристола http://bit.ly/1lMEBWy.

14.5. Обращение по URL-адресу с произвольными заголовками

Задача

Требуется обратиться по URL-адресу, требующему отправки конкретных заголовков при запросе страницы.

Решение

Задайте параметр контекста `header` при использовании потока `http`. Значение `header` должно представлять собой одну строку. Заголовки разделяются комбинацией возврата курсора и новой строки (`\r\n` в строке, заключенной в двойные кавычки). Пример:

```
$url = 'http://www.example.com/special-header.php';
$header = "X-Factor: 12\r\nMy-Header: Bob";
$options = array('header' => $header);
// Создание контекста потока
$context = stream_context_create(array('http' => $options));
// Контекст передается file_get_contents()
print file_get_contents($url, false, $context);
```

При использовании cURL присвойте параметру `CURLOPT_HTTPHEADER` массив отправляемых заголовков:

```
$c = curl_init('http://www.example.com/special-header.php');
curl_setopt($c, CURLOPT_RETURNTRANSFER, true);
curl_setopt($c, CURLOPT_HTTPHEADER, array('X-Factor: 12', 'My-Header: Bob'));
$page = curl_exec($c);
curl_close($c);
```

При использовании `HTTP_Request2` вызовите метод `setHeader()`:

```
require 'HTTP/Request2.php';

$r = new HTTP_Request2('http://www.example.com/special-header.php');
$r->setHeader(array('X-Factor' => 12, 'My-Header','Bob'));
$page = $r->send()->getBody();
print $page;
```

Комментарий

В cURL предусмотрены специальные параметры для присваивания заголовков запроса `Referer` и `User-Agent` — `CURLOPT_REFERER` и `CURLOPT_USERAGENT`. Пример использования этих параметров:

```
$c = curl_init('http://www.example.com/submit.php');
curl_setopt($c, CURLOPT_RETURNTRANSFER, true);
curl_setopt($c, CURLOPT_REFERER, 'http://www.example.com/form.php');
curl_setopt($c, CURLOPT_USERAGENT, 'cURL via PHP');
$page = curl_exec($c);
curl_close($c);
```

См. также

Документация по обертке потока `http`, функции `curl_setopt()` и классу PEAR `HTTP_Request2`.

14.6. Обращение по URL-адресу с тайм-аутом

Задача

Требуется обратиться по удаленному URL-адресу, но при этом избежать слишком долгого ожидания в случае занятости или медленной работы удаленного сервера.

Решение

При использовании потока `http` задайте параметр конфигурации `default_socket_timeout`:

```
// 15-секундный тайм-аут
ini_set('default_socket_timeout', 15);
$page = file_get_contents('http://slow.example.com/');
```

Программа ожидает установления связи с удаленным сервером до 15 секунд. Изменение `default_socket_timeout` распространяется на все новые сокеты или удаленные подключения, созданные при конкретном запуске сценария.

При использовании cURL присвойте значение параметру `CURLOPT_CONNECTTIMEOUT`:

```
$c = curl_init('http://slow.example.com/');
curl_setopt($c, CURLOPT_RETURNTRANSFER, true);
curl_setopt($c, CURLOPT_CONNECTTIMEOUT, 15);
$page = curl_exec($c);
curl_close($c);
```

При использовании `HTTP_Request2` присвойте значение элементу `timeout` в массиве параметров, передаваемом конструктору `HTTP_Request2`:

```
require_once 'HTTP/Request2.php';

$r = new HTTP_Request2('http://slow.example.com/');
$r->setConfig(array(
    'connect_timeout' => 15
));

$page = $r->send()->getBody();
```

Комментарий

Даже самый надежный, критически важный сервер корпоративного класса может испытывать перебои в работе. Возможны и другие варианты: удаленный сервер, от которого вы зависите, работает нормально, но не способен обрабатывать запросы из-за проблем с сетью между вашим и удаленным сервером. Ограничение промежутка времени, в течение которого PHP ожидает подключения к удаленному серверу, может быть полезно, если использование данных с удаленных источников является частью процесса конструирования страницы.

Все приемы, описанные в разделе «Решение», ограничивают продолжительность ожидания подключения к удаленному серверу со стороны PHP. Но после того, как связь будет установлена, на время получения ответа никакие ограничения уже не распространяются. Если вам так важно своевременно получать ответы, дополнительно ограничьте продолжительность ожидания данных из подключенного сокета.

Для потокового подключения используйте функцию `stream_set_timeout()`. Этой функции должен передаваться ресурс потока, поэтому поток необходимо открыть

вызовом `fopen()` — вызов `file_get_contents()` здесь не подойдет. В следующем примере тайм-аут чтения ограничивается 20 секундами:

```
$url = 'http://slow.example.com';
$stream = fopen($url, 'r');
stream_set_timeout($stream, 20);
$response_body = stream_get_contents($stream);
```

При использовании cURL присвойте параметру `CURLOPT_TIMEOUT` максимальный промежуток времени, в течение которого должен отработать вызов `curl_exec()`. В него включается как тайм-аут подключения, так и время чтения всего тела ответа:

```
curl_setopt($c, CURLOPT_TIMEOUT, 35);
```

При использовании `HTTP_Request2` величина тайм-аута включается в конфигурационный массив. Значение задается в секундах. В следующем примере оно составляет 20 секунд:

```
require_once 'HTTP/Request2.php';

$r = new HTTP_Request2('http://slow.example.com/');
$r->setConfig(array(
    'timeout' => 20
));

$page = $r->send()->getBody();
```

Хотя назначение тайм-аута для подключения и чтения данных может повышать производительность, оно также может привести к искажению данных. Ваш сценарий может получить лишь часть ответа до истечения тайм-аута. Если вы установили тайм-аут, обязательно проверьте весь полученный ответ. Также в ситуациях, в которых критично быстрое генерирование страницы, запросите внешние данные в отдельном процессе и запишите их в локальный кэш. Это позволит вашим страницам использовать кэш без опасения тайм-аута или частичных ответов.

См. также

Документация по функциям `curl_setopt()`, `stream_set_timeout()`, `default_socket_timeout` и классу PEAR `HTTP_Request2`.

14.7. Обращение по URL-адресу по протоколу HTTPS

Задача

Требуется запросить страницу по защищенному URL-адресу.

Решение

Воспользуйтесь средствами, описанными в Рецептах 14.1 и 14.2; передайте URL-адрес, начинающийся с префикса https.

Комментарий

Если при построении PHP была включена библиотека OpenSSL, все функции могут обращаться не только по обычным URL-адресам, но и по защищенным URL-адресам. Чтобы узнать, включена ли поддержка SSL в вашей конфигурации PHP, найдите раздел «openssl» в выходных данных `phpinfo()`.

См. также

Рецепты 14.1 и 14.2 — обращение по URL-адресам; OpenSSL Project (*http://www.openssl.org/*).

14.8. Отладка низкоуровневой передачи данных HTTP

Задача

Требуется проанализировать запрос HTTP, который отправляется браузером серверу, и соответствующий ответ HTTP. Например, ваш сервер не предоставил ожидаемого ответа на конкретный запрос, и вы хотите получить точную информацию о компонентах запроса.

Решение

Для простых запросов подключитесь к серверу по Telnet и введите заголовки запроса. Простой обмен данными может выглядеть так:

```
POST /submit.php HTTP/1.1
User-Agent: PEAR HTTP_Request2 class ( http://pear.php.net/ )
Content-Type: application/x-www-form-urlencoded
Connection: close
Host: www.example.com
Content-Length: 12

monkey=uncle
```

Комментарий

Когда вы вводите заголовки запросов, веб-сервер не знает, что данные поступают от вас, а не от браузера, отправившего запрос. Однако некоторые веб-серверы устанавливают тайм-аут для ожидания запроса, поэтому запрос стоит ввести за-

ранее, а затем просто скопировать его в Telnet. В первой строке запроса указывается имя запроса (POST), пробел и путь к нужному файлу (/submit.php), затем пробел и используемый протокол (HTTP/1.1). Следующая строка — заголовок `Host` — сообщает серверу, какой виртуальный хост следует использовать, если один IP-адрес разделяется несколькими IP-адресами. Пустая строка сообщает серверу об окончании запроса, после чего сервер выдает свой ответ: сначала идут заголовки, затем пустая строка и затем тело ответа. Программа Netcat также может пригодиться для решения подобных задач.

Вставка текста в Telnet быстро надоедает, а с методом POST выдавать запросы таким способом еще сложнее. Если запрос выдается через класс `HTTP_Request2`, для получения заголовков и тела ответа можно воспользоваться методами `getResponseHeader()` и `getResponseBody()`:

```
require 'HTTP/Request2.php';
$r = new HTTP_Request2('http://www.example.com/submit.php');
$r = new HTTP_Request2('http://localhost/submit.php');
$r->setMethod(HTTP_Request2::METHOD_POST)
    ->addPostParameter('monkey', 'uncle');
$response = $r->send();

$response_headers = $response->getHeader();
$response_body    = $response->getBody();
```

Чтобы получить конкретный заголовок ответа, передайте имя заголовка методу `getResponseHeader()`. Имя заголовка должно быть записано в нижнем регистре. Без аргумента `getResponseHeader()` возвращает массив, содержащий все заголовки ответов. `HTTP_Request2` сохраняет исходящий запрос. Для обращения к нему используется метод `getLastEvent()`:

```
require 'HTTP/Request2.php';

$r = new HTTP_Request2('http://www.example.com/submit.php');
$r = new HTTP_Request2('http://localhost/submit.php');
$r->setMethod(HTTP_Request2::METHOD_POST)
    ->addPostParameter('monkey', 'uncle');
$response = $r->send();

print_r($r->getLastEvent());
```

Запрос выглядит примерно так:

```
POST /submit.php HTTP/1.1
User-Agent: PEAR HTTP_Request2 class ( http://pear.php.net/ )
Content-Type: application/x-www-form-urlencoded
Connection: close
Host: www.example.com
Content-Length: 12

monkey=uncle
```

Получить доступ к заголовкам ответа с потоком `http` возможно, но вам придется использовать функцию, которая предоставляет ресурс потока, — такую, как `fopen()`.

В метаданных, получаемых при передаче этого ресурса потока функции `stream_get_meta_data()` после выдачи запроса, присутствует набор заголовков ответа. Следующий пример показывает, как обратиться к заголовкам ответа с ресурсом потока:

```
$url = 'http://www.example.com/submit.php';
$stream = fopen($url, 'r');
$metadata = stream_get_meta_data($stream);
// Заголовки хранятся в 'wrapper_data'
foreach ($metadata['wrapper_data'] as $header) {
    print $header . "\n";
}
// Получение тела функцией stream_get_contents()
$response_body = stream_get_contents($stream);
```

Функция `stream_get_meta_data()` возвращает массив с информацией о потоке. Элемент `wrapper_data` этого массива содержит данные, относящиеся к конкретной обертке. Для обертки `http` это заголовки ответов, по одному на каждый элемент подмассива. Результат выглядит примерно так:

```
HTTP/1.1 200 OK
Date: Sun, 07 May 2014 18:24:37 GMT
Server: Apache/2.2.2 (Unix)
Last-Modified: Sun, 07 May 2014 01:58:12 GMT
ETag: "1348011-7-16167502"
Accept-Ranges: bytes
Content-Length: 7
Connection: close
Content-Type: text/plain
```

Функция `fopen()` получает необязательный аргумент с контекстом потока. Если вы хотите использовать контекст, укажите его в четвертом аргументе `fopen()` (второй аргумент содержит режим, а третий — необязательный флаг, определяющий, нужно ли использовать `include_path` при поиске файла).

Если вы используете cURL, для включения заголовков ответов в вывод `curl_exec()` следует установить флаг `CURLOPT_HEADER`:

```
$c = curl_init('http://www.example.com/submit.php');
curl_setopt($c, CURLOPT_HEADER, true);
curl_setopt($c, CURLOPT_POST, true);
curl_setopt($c, CURLOPT_POSTFIELDS, 'monkey=uncle&amp;rhino=aunt');
curl_setopt($c, CURLOPT_RETURNTRANSFER, true);

$response_headers_and_page = curl_exec($c);
curl_close($c);
```

Чтобы записать заголовки ответа прямо в файл, откройте файловый дескриптор при помощи функции `fopen()` и присвойте его параметру `CURLOPT_WRITEHEADER`, как показано ниже:

```
$fh = fopen('/tmp/curl-response-headers.txt','w') or die($php_errormsg);
$c = curl_init('http://www.example.com/submit.php');
curl_setopt($c, CURLOPT_POST, true);
curl_setopt($c, CURLOPT_POSTFIELDS, 'monkey=uncle&amp;rhino=aunt');
curl_setopt($c, CURLOPT_RETURNTRANSFER, true);
```

```
curl_setopt($c, CURLOPT_WRITEHEADER, $fh);
$page = curl_exec($c);
curl_close($c);
fclose($fh) or die($php_errormsg);
```

При использовании cURL параметр `CURLOPT_VERBOSE` заставляет `curl_exec()` и `curl_close()` выводить в стандартный поток ошибок отладочную информацию, включая содержимое запроса:

```
$c = curl_init('http://www.example.com/submit.php');
curl_setopt($c, CURLOPT_VERBOSE, true);
curl_setopt($c, CURLOPT_POST, true);
curl_setopt($c, CURLOPT_POSTFIELDS, 'monkey=uncle&amp;rhino=aunt');
curl_setopt($c, CURLOPT_RETURNTRANSFER, true);
$page = curl_exec($c);
curl_close($c);
```

Результат выглядит примерно так:

```
* Connected to www.example.com (10.1.1.1)
> POST /submit.php HTTP/1.1
Host: www.example.com
Pragma: no-cache
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, */*
Content-Length: 23
Content-Type: application/x-www-form-urlencoded

monkey=uncle&rhino=aunt* Connection #0 left intact
* Closing connection #0
```

Так как cURL выводит отладочную информацию в стандартный поток ошибок, а не в стандартный выходной поток, данные невозможно перехватить посредством буферизации вывода. Впрочем, вы можете открыть файловый дескриптор для записи и присвоить его `CURLOUT_STDERR` для направления отладочной информации в файл:

```
$fh = fopen('/tmp/curl.out','w') or die($php_errormsg);
$c = curl_init('http://www.example.com/submit.php');
curl_setopt($c, CURLOPT_VERBOSE, true);
curl_setopt($c, CURLOPT_POST, true);
curl_setopt($c, CURLOPT_POSTFIELDS, 'monkey=uncle&amp;rhino=aunt');
curl_setopt($c, CURLOPT_RETURNTRANSFER, true);
curl_setopt($c, CURLOPT_STDERR, $fh);
$page = curl_exec($c);
curl_close($c);
fclose($fh) or die($php_errormsg);
```

При использовании cURL существует еще один способ обращения к данным заголовков — написание *заголовочной функции*. Такая функция похожа на функцию записи cURL, но вызывается она для обработки заголовков ответа, а не тела ответа. В следующем примере определяется класс `HeaderSaver`, метод `header()` которого может использоваться как заголовочная функция для накопления заголовков ответа:

```
class HeaderSaver {
    public $headers = array();
```

```
    public $code = null;

    public function header($curl, $data){
        if (is_null($this->code) &&
            preg_match('@^HTTP/\d\.\d (\d+) @',$data,$matches)) {
            $this->code = $matches[1];
        } else {
            // Удаление завершающей новой строки
            $trimmed = rtrim($data);
            if (strlen($trimmed)) {
                // Если эта строка начинается с пробела или табуляции,
                // она является продолжением предыдущего заголовка
                if (($trimmed[0] == ' ') || ($trimmed[0] == "\t")) {
                    // Начальные пропуски сворачиваются в один пробел
                    $trimmed = preg_replace('@^[ \t]+@',' ', $trimmed);
                    $this->headers[count($this->headers)-1] .= $trimmed;
                }
                // В противном случае это новый заголовок
                else {
                    $this->headers[] = $trimmed;
                }
            }
        }
        return strlen($data);
    }

}

$h = new HeaderSaver();
$c = curl_init('http://www.example.com/plankton.php');
// Регистрация заголовочной функции
curl_setopt($c, CURLOPT_HEADERFUNCTION, array($h,'header'));
curl_setopt($c, CURLOPT_RETURNTRANSFER, true);
$page = curl_exec($c);
// Теперь $h содержит данные
print 'The response code was: ' . $h->code . "\n";
print "The response headers were: \n";
foreach ($h->headers as $header) {
    print "  $header\n";
}
```

Согласно стандарту HTTP 1.1, заголовки могут занимать несколько строк; для этого каждая строка продолжения должна начинаться минимум с одного пробела или табуляции. Тем не менее массивы заголовков, возвращаемые `stream_get_meta_data()` и `HTTP_Request2::getResponseHeader()`, некорректно обрабатывают многострочные заголовки. Дополнительные строки заголовка рассматриваются как отдельные заголовки. Тем не менее этот код правильно присоединяет дополнительные строки в многострочных заголовках.

См. также

Документация по функциям `curl_setopt()`, `stream_get_meta_data()` и `fopen()` и классу PEAR `HTTP_Request2`; RFC 2616 — синтаксис запроса HTTP (правила,

касающиеся многострочных заголовков, приведены в разделе 4.2. Программа netcat доступна на сайте проекте GNU (*http://netcat.sourceforge.net/*).

14.9. Выдача запросов OAuth 1.0

Задача

Требуется выдать запрос с цифровой подписью OAuth 1.0.

Решение

Воспользуйтесь расширением PECL oauth.

Комментарий

С помощью технологии OAuth 1.0 провайдеры API дают пользователям возможность предоставить сторонним разработчикам безопасный доступ к их учетным записям без передачи имени пользователя и пароля.

Вместо этого используются два набора открытых и закрытых маркеров для снабжения запросов цифровой подписью. Один набор маркеров предназначен для вашего приложения; он используется при каждом запросе. Другой набор содержит маркеры, отличающиеся в зависимости от пользователя.

Два открытых маркера передаются для идентификации приложения и пользователя. Два закрытых маркера, также называемых *секретами*, используются для подписывания запросов. Подпись строится с использованием метода HTTP и URL-адреса запроса, а также нескольких других фрагментов метаданных (например, временной метки).

При получении запроса провайдер API проверяет подпись и другие части запроса, чтобы убедиться в его «законности». Так как секретные ключи имеются только у вас и у провайдера, провайдер API знает, что в случае совпадения подписей запрос должен был поступить от вас. Подписи расходятся, значит, запрос ложный, и его следует отклонить.

При использовании расширения PECL oauth не нужно беспокоиться о конкретике самого алгоритма. Нужно лишь знать общую последовательность процедуры авторизации.

1. Вы получаете исходный набор пользовательских маркеров. Они также называются *маркерами запросов* или *временными маркерами*, потому что используются только во время процесса авторизации, а не для выдачи реальных вызовов API.
2. Пользователь перенаправляется на провайдера API.

3. Пользователь входит на этот сайт, который выполняет аутентификацию и просит пользователя авторизовать ваше приложение для выдачи вызовов API от его имени.
4. После того как пользователь авторизует ваше приложение, провайдер API перенаправляет пользователя обратно к вашему приложению, передавая два блока данных: тот же временный открытый ключ, который вы предоставили для сопоставления ответов с соответствующими пользователями, и идентификатор PIN для предотвращения атак фиксации сеанса.
5. PIN заменяется постоянными маркерами OAuth для пользователя.
6. Вы выдаете вызовы API от имени пользователя.

См. также

Документация по расширению oauth; RFC 5849 — спецификация OAuth 1.0.

14.10. Выдача запросов OAuth 2.0

Задача

Требуется выдать запрос с цифровой подписью OAuth 2.0.

Решение

Воспользуйтесь функциями потоков.

Комментарий

С помощью технологии OAuth 2.0 провайдеры API дают пользователям возможность предоставить сторонним разработчикам безопасный доступ к их учетным записям без передачи имени пользователя и пароля.

Вместо этого используется маркер, идентифицирующий приложение и участника. Он также называется маркером *носителя* (bearer), потому что API принимает этот маркер для идентификации от любой стороны, представившей его. Для защиты от похищения маркера запросы OAuth 2.0 передаются по протоколу SSL.

Так как OAuth 2.0 отказывается от подписей OAuth 1.0, специальное расширение не понадобится. Вы можете использовать те же функции HTTP, что и обычно.

Последовательность операций OAuth 2.0 проходит по следующей схеме.

1. Пользователь перенаправляется к провайдеру API с передачей самостоятельно сгенерированного секретного значения, называемого *состоянием* (state),

и URL-адреса, на который пользователь должен быть перенаправлен после входа.

2. Пользователь входит на сайт, который выполняет аутентификацию и просит пользователя авторизовать ваше приложение для выдачи вызовов API от его имени.
3. После того как пользователь авторизует ваше приложение, провайдер API перенаправляет пользователя обратно к вашему приложению, передавая два блока данных: состояние, которое было предоставлено вами для сопоставления каждого ответа с пользователем, и код.
4. Код заменяется постоянным маркером OAuth для пользователя, с передачей идентификатора приложения и секрета, по которому вы будете идентифицироваться.
5. Вы выдаете вызовы API от имени пользователя.

В примере «Hello World» REST-совместимый API сервиса LinkedIn используется для приветствия пользователя по имени:

```
// Внесите необходимые изменения
define('API_KEY',      'YOUR_API_KEY_HERE');
define('API_SECRET',   'YOUR_API_SECRET_HERE');
define('REDIRECT_URI', 'http://' . $_SERVER['SERVER_NAME'] .
                                   $_SERVER['SCRIPT_NAME']);
define('SCOPE',        'r_fullprofile r_emailaddress rw_nus');

// Вероятно, стоит воспользоваться базой данных
session_name('linkedin');
session_start();

// Последовательность операций OAuth 2
if (isset($_GET['error'])) {
    // От LinkedIn получена ошибка
    print $_GET['error'] . ': ' . $_GET['error_description'];
    exit;
} elseif (isset($_GET['code'])) {
    // Пользователь авторизовал ваше приложение
    if ($_SESSION['state'] == $_GET['state']) {
        // Получение маркера для вызовов API
        getAccessToken();
    } else {
        // Межсайтовая подделка запроса? А может, перепутаны состояния?
        exit;
    }
} else {
    if ((empty($_SESSION['expires_at'])) || (time() > $_SESSION['expires_at'])) {
        // Срок действия маркера истек, сбросить состояние
        $_SESSION = array();
    }
    if (empty($_SESSION['access_token'])) {
        // Start authorization process
        getAuthorizationCode();
```

```
    }
}

// Поздравляем! Маркер действителен, теперь получить профиль
$user = fetch('GET', '/v1/people/~:(firstName)');
print "Hello $user->firstName.\n";
exit;

function getAuthorizationCode() {
    $params = array('response_type' => 'code',
                    'client_id' => API_KEY,
                    'scope' => SCOPE,
                    'state' => uniqid('', true), // уникальная длинная строка
                    'redirect_uri' => REDIRECT_URI,
              );

    // Запрос на аутентификацию
    $url = 'https://www.linkedin.com/uas/oauth2/authorization?' .
            http_build_query($params);

    // Необходимо для идентификации запроса, когда он вернется к нам
    $_SESSION['state'] = $params['state'];

    // Пользователь перенаправляется для аутентификации
    header("Location: $url");
    exit;
}

function getAccessToken() {
    $params = array('grant_type' => 'authorization_code',
                    'client_id' => API_KEY,
                    'client_secret' => API_SECRET,
                    'code' => $_GET['code'],
                    'redirect_uri' => REDIRECT_URI,
              );

    // Запрос маркера доступа
    $url = 'https://www.linkedin.com/uas/oauth2/accessToken?' .
            http_build_query($params);

    // Приказываем потокам использовать запрос POST
    $context = stream_context_create(
                    array('http' =>
                        array('method' => 'POST',
                        )
                    )
                );

    // Получение информации маркера доступа
    $response = file_get_contents($url, false, $context);

    // Декодирование в "родной" объект PHP
    $token = json_decode($response);

    // Сохранение маркера и срока действия
    $_SESSION['access_token'] = $token->access_token; // Защитить!
```

```
    $_SESSION['expires_in']   = $token->expires_in; // относительное время
                                                    // (в секундах)
    $_SESSION['expires_at']   = time() + $_SESSION['expires_in']; // абсолютное
                                                                  // время

    return true;
}

function fetch($method, $resource, $body = '') {
    $params = array('oauth2_access_token' => $_SESSION['access_token'],
                    'format' => 'json',
              );

    // Необходимо использовать HTTPS
    $url = 'https://api.linkedin.com' . $resource . '?' .
           http_build_query($params);
    // Приказываем потоку выдать запрос (GET, POST, PUT, DELETE)
    $context = stream_context_create(
                    array('http' =>
                        array('method' => $method,
                        )
                    )
                );

    // Фокус
    $response = file_get_contents($url, false, $context);

    // Декодирование в "родной" объект PHP
    return json_decode($response);
}
```

Для других провайдеров API последовательность операций OAuth остается неизменной, но необходимо изменить ключи и URL-адреса, а также сам вызов API.

См. также

Документация по расширению `oauth`; RFC 6749 — спецификация OAuth 2.0; LinkedIn Developer Network (*https://developer.linkedin.com/*).

15 Предоставление доступа к REST-совместимым API

15.0. Введение

Открытый доступ к API с использованием REST-операций позволяет практически кому угодно обращаться к вашему приложению из программного кода. Неважно, какой язык при этом используется. Так как REST в качестве синтаксиса использует базовый протокол Всемирной паутины, никакие специальные библиотеки не нужны. Если разработчик может выдавать запросы HTTP, он также может обращаться с вызовами к вашим REST-совместимым API.

REST не предопределяет конкретного синтаксиса для запросов, схемы передаваемых данных и даже способа сериализации. Это всего лишь архитектурный стиль, который предоставляет набор паттернов и общих правил. Каждый сайт свободен реализовать свои API так, как считает нужным, — при условии соблюдения рекомендаций.

Фундаментальной единицей REST является *ресурс*. Ресурсами могут быть люди, объекты и вообще все, с чем должны выполняться операции. Ресурсы определяются местоположением, с использованием URL-адресов (или по имени с использованием URN-имен[1]). Ресурсы имеют представления, то есть различные способы описания ресурса. Обычно представления используют стандартный формат данных: JSON, XML, HTML, PDF, PNG и т. д.

По стандартной схеме URL-адреса форматируются по схеме */версия/ресурс/ключ*. Например, Расмуса Лердорфа можно найти по адресу *http://api.example*.

---

[1] Сокращение URL означает «Uniform Resource Locator», то есть «унифицированный локатор (указатель) ресурса», а URN — «Uniform Resource Name», то есть «унифицированное имя ресурса». Ресурс может иметь как имя, так и местоположение. Например, спецификации HTTP 1.1 соответствует URN-имя urn:ietf:rfc:2616 и URL-адрес https://www.ietf.org/rfc/rfc2616.txt. URN и URL объединяются общим термином URI (Uniform Resource Identifier, унифицированный идентификатор ресурса).

com/v1/people/rasmus. Этот адрес соответствует человеку (person), идентифицируемому как «rasmus», с использованием версии 1.0. API.

В формате JSON ресурс может быть представлен следующим образом:

```
{
    "firstName": "Rasmus"
    "lastName": "Lerdorf"
}
```

В REST запрашиваемое действие описывается методами HTTP (такими как GET и POST). Таким образом, для обработки REST-совместимого запроса необходимо знать как URL-адрес, так и метод HTTP. Рецепт 15.1 демонстрирует, как направить запрос по нужному URL-адресу на основании метода HTTP, а Рецепт 15.2 показывает, как это делается с «чистыми» URL-адресами.

Каждый метод обладает четко определенным поведением. Например, GET сообщает серверу, что вы хотите получить существующий ресурс, а POST — что вы хотите добавить новый ресурс. Метод PUT используется для модификации ресурсов или создания ресурса с заданным именем. И конечно, метод DELETE удаляет ресурсы.

Помимо этого технология REST накладывает ряд других ограничений на структуру API. А именно некоторые методы должны быть безопасными, а другие — идемпотентными.

Безопасные методы (такие как GET) не изменяют ресурсы — собственно, поэтому они и называются безопасными. Другие методы (такие как POST и DELETE) безопасными не являются. Они могут иметь побочные эффекты изменения состояния системы за счет создания, модификации или удаления ресурсов (наверное, это именно то, что вам нужно).

Небезопасные методы дополнительно разделяются на две подкатегории в зависимости от идемпотентности. Многократный вызов идемпотентного метода эквивалентен его однократному вызову. Например, если вызвать DELETE для ресурса, повторные попытки вызова DELETE могут привести к ошибке, но другие ресурсы при этом не удаляются. С другой стороны, попытка повторного выполнения запроса POST может привести к созданию двух новых ресурсов. В таблице 15.1 представлена краткая сводка поведения методов.

Таблица 15.1. Поведение методов HTTP

Метод HTTP	Описание	Безопасность	Идемпотентность
GET	Чтение ресурса	Да	Да
POST	Создание ресурса	Нет	Нет
PUT	Обновление ресурса	Нет	Да
DELETE	Удаление ресурса	Нет	Да

Поведение каждого метода и способы обработки запросов для чтения, создания, обновления и удаления ресурсов рассматриваются в Рецептах 15.3, 15.4, 15.5 и 15.6.

REST использует коды статуса HTTP для обозначения результата запроса (успех или неудача). Коды 200+ обозначают успешное завершение; 300+ — для ответа серверу требуется дополнительное действие; 400+ — ошибка клиента; 500+ — ошибка сервера. Например, для успешного запроса GET возвращается код 200; для запроса ресурса по новому URL-адресу — код 301; при попытке чтения несуществующего ресурса — код 404; при запросе к серверу во время технического обслуживания (временная недоступность) — код 503.

Стандартные коды статуса и уместность их использования оговорены в контексте рецептов. Обобщенные рекомендации по возвращению ошибок рассматриваются в Рецепте 15.7.

Ничто не мешает ресурсу иметь несколько представлений. Скажем, XML-версия *http://api.example.com/v1/people/rasmus* может выглядеть так:

```
<person>
    <firstName>Rasmus</firstName>
    <lastName>Lerdorf</lastName>
</person>
```

Другой пример — текстовый документ, существующий в HTML- и PDF-версиях, или изображение, существующее в форматах JPEG и PNG. О том, как предоставить доступ к нескольким версиям одного ресурса, рассказано в Рецепте 15.8.

В рецептах этой главы мы постарались заложить основу проектирования и реализации REST-совместимых API в PHP. Тем не менее в целом эта задача выходит далеко за рамки одной главы. При использовании REST приходится учитывать множество аспектов, включая кэширование и гиперсреду как ядро состояния приложения (HATEOAS), которые в книге не упоминаются. Чтобы больше узнать по этой теме, обратитесь к источнику: исходному документу с описанием REST в диссертации Роя Филдинга (*https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm*). Как нетрудно догадаться, материал излагается в академическом стиле.

Более практичная книга, которая преобразует каноны REST в практический материал, — «RESTful Web APIs» Леонарда Ричардсона (Leonard Richardson), Майка Амундсена (Mike Amundsen) и Сэма Руби (Sam Ruby) (издательство O'Reilly). «REST in Practice» Джима Уэббера (Jim Webber), Саваса Парастатидиса (Savas Parastatidis) и Иэна Робинсона (Ian Robinson) (издательство O'Reilly) — практическое руководство по проектированию REST-совместимых архитектур. Наконец, чрезвычайно глубокое введение в REST, написанное в стиле серии «Сборники рецептов», — «RESTful Web Services Cookbook», автор Суббу Алламараджу (Subbu Allamaraju) (издательство O'Reilly).

В главе 8 рассматриваются возможности реализации дополнительных концепций REST-совместимого сервиса, хотя и не всегда с использованием REST-совместимой терминологии. Например, аутентификация рассматривается в Рецепте 8.6, а чтение и запись заголовков HTTP — в Рецептах 8.8 и 8.9.

Хотя для самостоятельного создания REST-совместимых API необходимо понимание основ REST-совместимого проектирования, вам не придется писать весь служебный код с нуля. К сожалению, никакого «официального», подходящего на все случаи жизни REST-совместимого фреймворка для PHP не существует (кто-то может пошутить, что PHP — это он и есть). Однако существует ряд фреймворков PHP, упрощающих работу по предоставлению доступа к ресурсам. Некоторые из них — полноценные фреймворки MVC с абстракциями ORM, позволяющими создавать серьезные приложения с предоставлением доступа к REST-совместимым API; другие — микрофреймворки, предоставляющие минимальный фасад на основе существующего кода. В этой области ведется активная разработка, так что вы наверняка найдете пакет, идеально подходящий для вашей ситуации.

15.1. Предоставление доступа к ресурсу и обработка запроса

Задача

Требуется предоставить доступ к ресурсу и обработать запросы в зависимости от метода HTTP.

Решение

Воспользуйтесь методом `$_SERVER['REQUEST_METHOD']` для маршрутизации запроса:

```
$request = explode('/', $_SERVER['PATH_INFO']);

$method = strtolower($_SERVER['REQUEST_METHOD']);
switch($method) {
    case 'get':
        // Обработка запроса GET
        break;
    case 'post':
        // Обработка запроса POST
        break;
    case 'put':
        // Обработка запроса PUT
        break;
    case 'delete':
        // Обработка запроса DELETE
        break;
    default:
        // Нереализованный метод
        http_response_code(405);
}
```

Комментарий

При обработке запроса REST-совместимого ресурса необходимо знать как запрашиваемый ресурс, так и действие, выполняемое клиентом.

Тем не менее между ресурсами и обрабатывающими их сценариями PHP не всегда удается установить однозначное соответствие. Конечно, исключения тоже встречаются: например, в качестве ключа ресурса книги может использоваться ISBN-код книги. Таким образом, «PHP Cookbook» соответствует URL */v1/books.php/9781449363758*, «Learning PHP 5» — URL */v1/books.php/9780596005603*, и т. д.

Но создавать отдельные файлы во всех этих местах было бы нерационально. Вместо этого используется один файл books.php, которому код ISBN передается в параметре. Во многих сценариях можно передать код ISBN в параметре запроса (например, */v1/books.php?isbn=9781449363758*) и прочитать его в коде PHP конструкцией `$_GET['isbn']`.

Однако с REST для идентификации ресурсов используется символ `/`. И с URL-адресом вида */v1/books.php/9781449363758* не удастся использовать стандартные суперглобальные переменные PHP. Вместо этого путь разбирается на компоненты, для чего выполняется разбивка `$_SERVER['PATH_INFO']` по разделителям «/»:

```
$request = explode('/', $_SERVER['PATH_INFO']);
```

Запрос к */v1/books.php/9781449363758* преобразуется в массив:

```
Array
(
    [0] =>
    [1] => 9781449363758
)
```

Затем запрос обрабатывается в зависимости от метода HTTP, что позволяет обрабатывать запросы GET, PUT, POST и DELETE в разных функциях. Для этого используется `$_SERVER['REQUEST_METHOD']`:

```
$method = strtolower($_SERVER['REQUEST_METHOD']);
switch($method) {
    case 'get':
        // Обработка запроса GET
        get_book($request);
        break;
    case 'post':
        // Обработка запроса POST
        post_book($request);
        break;
    case 'put':
        // Обработка запроса PUT
        put_book($request);
        break;
    case 'delete':
        // Обработка запроса DELETE
```

```
        delete_book($request);
        break;
    default:
        // Нереализованный метод
        http_response_code(405);
}
```

Так как вы можете отказаться от реализации некоторых методов для ресурса, команда `switch` позволяет легко вставить нужные методы, сохранив при этом поведение по умолчанию — возвращение кода статуса HTTP 405 («Недопустимый метод»).

Возможно, вам будет удобно связать REST-совместимые ресурсы, доступ к которым вы предоставляете, с классами PHP. Эти классы могут иметь методы `get()`, `post()` и т. д. Пример:

```
class books {
    static public function get($request) {
        // Обработка запроса GET
    }

    static public function post($request) {
        // Обработка запроса POST
    }

    // Другие методы
}
class albums {
    static public function get($request) {
        // Обработка запроса GET
    }
}
```

Далее можно изменить процедуру маршрутизации так, чтобы все ресурсы обрабатывались одной страницей index.php, вместо создания отдельного файла для каждого ресурса:

```
// Разбиение URL и извлечение корневого ресурса
$request = explode('/', $_SERVER['PATH_INFO']);
$resource = array_shift($request);

// Обрабатываются только действительные ресурсы
$resources = array('books' => true, 'music' => true);
if (! array_key_exists($resource, $resources)) {
    http_response_code(404);
    exit;
}

// Запрос передается соответствующей функции в зависимости от метода
$method = strtolower($_SERVER["REQUEST_METHOD"]);
switch($method) {
    case 'get':
    case 'post':
    case 'put':
    case 'delete':
```

```
    // Все остальные поддерживаемые методы, такие как HEAD
        if (method_exists($resource, $method)) {
            call_user_func(array($resource, $method), $request);
            break;
        }
        // Сквозная передача управления
    default:
        http_response_code(405);
}
```

Сначала URL-адрес разбивается по символам `/`, после чего программа извлекает первый элемент — ресурс (книга, альбом и т. д.).

Далее мы убеждаемся в том, что ресурс входит в число допустимых. Например, запрос */v1/movies/fletch* порождает ошибку 404, потому что такой ресурс не существует.

Наконец, мы проверяем, содержит ли класс с именем, совпадающим с именем ресурса, метод с именем, совпадающим с именем метода HTTP. Если такой метод присутствует, он вызывается при помощи функции `call_user_func()`.

Если же метод не найден, возвращается код ответа 405 («Недопустимый метод»). Также программа ограничивается обработкой методов `get()`, `post()`, `put()` и `delete()`, чтобы пользователи не могли вызывать другие методы класса.

См. также

Рецепт 15.2 — предоставление доступа к «чистым» URL-адресам; документация по суперглобальному массиву `$_SERVER` и `http_response_code()`.

15.2. Использование «чистых» путей для доступа к ресурсам

Задача

Требуется «очистить» URL-адреса, чтобы они не включали расширения файлов.

Решение

Воспользуйтесь модулем Apache `mod_rewrite`, чтобы связать путь с вашим сценарием PHP:

```
RewriteEngine on
RewriteBase /v1/
RewriteCond %{REQUEST_FILENAME} !-f
RewriteCond %{REQUEST_FILENAME} !-d
RewriteRule ^(.*)$ index.php?PATH_INFO=$1 [L,QSA]
```

Затем используйте `$_GET['PATH_INFO']` вместо `$_SERVER['PATH_INFO']`:

```
$request = explode('/', $_GET['PATH_INFO']);
```

Комментарий

Модуль `mod_rewrite` позволяет предоставлять доступ к элегантным URL-адресам вида */v1/books/9781449363758* даже в том случае, если файл по указанному пути не существует. Без него пришлось бы использовать более громоздкие URL вида */v1/books.php/9781449363758*. Если вы работаете с другим веб-сервером (например, nginx), используйте его синтаксис для реализации перенаправления URL-адресов.

Код, приведенный в решении, сообщает Apache, что если файл или каталог по заданному пути не найден, запрос следует перенаправить сценарию `index.php`. Дополнительно программа извлекает путь и передает его в параметре запроса `PATH_INFO`, чтобы информация исходного URL-адреса оставалась доступной для правильной обработки запроса.

В сценарии путь разбивается на компоненты по символу «/»:

```
$request = explode('/', $_GET['PATH_INFO']);
```

Например, запрос к */v1/books.php/9781449363758* преобразуется в массив:

```
Array
(
    [0] => books
    [1] => 9781449363758
)
```

Теперь можно выполнить необходимые действия в зависимости от ресурса и пути, как описано в Рецепте 15.1.

См. также

Рецепт 15.1 — перенаправление запросов к ресурсам в зависимости от метода HTTP; документация по модулю Apache `mod_rewrite`.

15.3. Предоставление доступа к ресурсу для чтения

Задача

Требуется организовать доступ к ресурсу для чтения.

Решение

Читайте запросы с методом GET. Возвращайте структурированные результаты в таких форматах, как JSON, XML или HTML. Не изменяйте никакие ресурсы.

Так, для запроса GET к ресурсу *http://api.example.com/v1/jobs/123*:

```
GET /v1/jobs/123 HTTP/1.1
Host: api.example.com
```

используется следующий код PHP:

```
// Допустим, данные были получены из базы данных или другого хранилища
$job[123] = [
    'id' => 123,
    'position' => [
        'title' => 'PHP Developer',
        ],
    ];

$json = json_encode($job[123]);

// Ресурс существует 200: OK
http_response_code(200);

// Информация возвращается в формате JSON
header('Content-Type: application/json');

print $json;
```

При этом генерируется следующий ответ HTTP:

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 61

{
    "id": 123,
    "position": {
        "title": "PHP Developer"
    }
}
```

Комментарий

Самый распространенный тип запросов REST — запросы для чтения данных. Операции чтения в REST соответствуют запросам HTTP GET. Этот же метод HTTP используется вашим браузером для чтения страниц HTML, так что при написании сценариев PHP вы почти всегда занимаетесь обработкой запросов GET.

Таким образом, процедура предоставления доступа к ресурсу REST для чтения достаточно прямолинейна:

1. Убедитесь в том, что используется метод HTTP GET.
2. Разберите URL-адрес для определения конкретного запроса, и возможно, ключа.

3. Получите необходимую информацию (возможно, из базы данных).
4. Отформатируйте данные в необходимую структуру.
5. Верните данные вместе с необходимыми заголовками HTTP.

Первые два шага рассматриваются в Рецепте 15.1, а третий зависит от вашего приложения. После выборки данных следующим шагом должно стать их форматирование для вывода. Часто используются форматы JSON или XML (или оба сразу), но подойдет любой структурированный формат: например, HTML, YAML или даже CSV.

Следующий пример получает запись и преобразует ее в JSON:

```
// Допустим, данные были получены из базы данных или другого хранилища
$job[123] = [
    'id' => 123,
    'position' => [
        'title' => 'PHP Developer',
        ],
    ];

$json = json_encode($job[123]);
```

После получения тела ответа следующим шагом должна стать отправка соответствующих заголовков HTTP. Так как запись была найдена, возвращается код статуса 200 (OK), а заголовку `Content-Type` присваивается значение, соответствующее формату JSON:

```
// Ресурс существует, код 200: OK
http_response_code(200);
// Данные возвращаются в формате JSON
header('Content-Type: text/json');
```

Наконец, отправьте сами данные:

```
print $json;
```

Если вакансия с идентификатором 123 не существует, сообщите об этом вызывающей стороне при помощи кода статуса 404:

```
// Ресурс не найден, код 404: Не найдено
http_response_code(404);
```

Запросы GET также не могут изменять систему. Другими словами, чтение ресурса не должно приводить к изменению этого ресурса или любой другой части ваших данных. Формально это требование называется «безопасностью операции».

См. также

Рецепт 14.1 — обращение по URL-адресу методом GET; Рецепты 15.4, 15.5 и 15.6 — предоставление доступа к ресурсам с использованием других методов.

15.4. Создание ресурса

Задача

Требуется реализовать возможность создания нового ресурса в системе.

Решение

Принимайте запросы с методом POST. Прочитайте тело POST; верните признак успеха и местонахождение нового ресурса.

Для запроса POST с URL-адресом *http://api.example.com/v1/jobs*:

```
POST /v1/jobs HTTP/1.1
Host: api.example.com
Content-Type: application/json
Content-Length: 49

{
    "position": {
        "title": "PHP Developer"
    }
}
```

используется следующий код PHP:

```
if ($_SERVER["REQUEST_METHOD"] == 'POST') {
    $body = file_get_contents('php://input');
    switch(strtolower($_SERVER['HTTP_CONTENT_TYPE'])) {
        case "application/json":
            $job = json_decode($body);
            break;
        case "text/xml":
            // Разбор данных
            break;
    }

    // Проверка данных

    // Создание нового ресурса
    $id = create($job); // Возвращает идентификатор 456
    $json = json_encode(array('id' => $id));
    http_response_code(201); // Ресурс создан
    $site = 'https://api.example.com';
    header("Location: $site/" . $_SERVER['REQUEST_URI'] . "/$id");
    header('Content-Type: application/json');
    print $json;
}
```

При этом генерируется следующий результат:

```
HTTP/1.1 201 Created
Location: https://api.example.com/jobs/456
```

```
Content-Type: application/json
Content-Length: 15

{
    "id": 456
}
```

Если клиенту разрешено указывать идентификатор (и он ему известен), используйте метод PUT:

```
PUT /v1/jobs/456 HTTP/1.1
Host: api.example.com
Content-Type: application/json
Content-Length: 49

{
    "position": {
        "title": "PHP Developer"
    }
}
```

Код PHP выглядит так:

```
if ($_SERVER["REQUEST_METHOD"] == 'PUT') {

    $body = file_get_contents('php://input');
    switch(strtolower($_SERVER['HTTP_CONTENT_TYPE'])) {
        case "application/json":
            $job = json_decode($body);
            break;
        case "text/xml":
            // Разбор данных
            break;
    }

    // Проверка данных

    // Создание нового ресурса
    $request = explode('/', substr($_SERVER['PATH_INFO'], 1));
    $resource = array_shift($request);
    $id = create($job, $request[0]); // С идентификатором из запроса
    $json = json_encode(array('id' => $id));
    http_response_code(201); // Ресурс создан
    $site = 'https://api.example.com';
    header("Location: $site/" . $_SERVER['REQUEST_URI'] );
    print $json;
}
```

Комментарий

Стандартный способ добавления новых ресурсов основан на обращении метода HTTP POST к родительскому ресурсу (или коллекции). Например, чтобы добавить в систему новую вакансию, отправьте данные методом POST по URL-адресу */v1/jobs* (в отличие от конкретного ресурса вида */v1/jobs/123*).

Сервер должен разобрать данные, проверить их и присвоить идентификатор вновь созданной записи. Пример:

```
if ($_SERVER["REQUEST_METHOD"] == 'POST') {
    $body = file_get_contents('php://input');
    switch(strtolower($_SERVER['HTTP_CONTENT_TYPE'])) {
        case "application/json":
            $job = json_decode($body);
            break;
        case "text/xml":
            $job = simplexml_load_string($body);
            break;
    }

    // Проверка данных

    // Создание нового ресурса
    $id = create($job); // Возвращает идентификатор
}
```

PHP автоматически разбирает данные стандартных форм HTML в `$_POST`. Для большинства REST API тело POST содержит данные в формате JSON (или XML, или другом формате).

Для этого вам придется читать и разбирать данные самостоятельно. Низкоуровневое тело POST доступно в специальном потоке *php://input*; прочитайте его в переменную вызовом `file_get_contents()`.

Чтобы узнать, какой формат данных был отправлен, проверьте заголовок HTTP `Content-Type`. Это делается при помощи суперглобальной переменной `$_SERVER['HTTP_CONTENT_TYPE']`. Даже если вы поддерживаете только один формат (например, JSON), убедитесь в том, что клиент использует именно этот формат.

В зависимости от `Content-Type` используйте соответствующую функцию, например `json_decode()` или `simplexml_load_string()`, для десериализации данных в PHP.

Теперь можно выполнить необходимую бизнес-логику, которая проверит входные данные, добавит ресурс и сгенерирует уникальный идентификатор для этой записи.

Если все прошло нормально, выдайте признак успеха и верните местонахождение нового ресурса:

```
http_response_code(201); // Ресурс создан
$site = 'https://api.example.com';
header("Location: $site/" . $_SERVER['REQUEST_URI'] . "/$id");
print $json;
```

Код статуса 201 означает, что ресурс был создан; этот код предпочтительнее более общего кода 200 (OK). Кроме того, рекомендуется вернуть местонахождение ресурса — либо через заголовок HTTP `Location`, либо в теле запроса. Первый вариант лучше соответствует стилю REST, но некоторые клиенты считают, что

результаты удобнее разбирать в теле, чем в заголовке. Заголовок HTTP `Location` должен содержать абсолютный URL-адрес.

Если с запросом возникли проблемы, верните код статуса из диапазона 4xx. Если возможно, верните сообщение с информацией о том, как клиент может исправить свой запрос.

Например, если обязательное поле отсутствует или документ синтаксически корректен, но имеет неправильную схему, верните код 422 («Необрабатываемая сущность»):

```
http_response_code(422); // Необрабатываемая сущность

$error_body = [
    "error" => "12",
    "message" => "Missing required field: job title"
];

print json_encode($error_body);
```

Если вам не удается найти конкретный код ошибки для проблемы, всегда можно воспользоваться кодом 400 («Неверный запрос»).

Если ваша система не может определенно сказать, правилен запрос или нет, верните код 202 («Принято»). Это правильный способ пассивно-агрессивного оповещения о поведении, не связанном никакими обязательствами. Чаще всего он используется при обработке запросов в асинхронной очереди, когда REST-сервер передает запрос другой системе, но эта система не возвращает ответ немедленно.

Если клиенту известен идентификатор, связанный с новой записью (вместо его присваивания вами), разрешите выдачу запросов PUT прямо к выбранному местонахождению (а не к родительскому ресурсу). Пример:

```
PUT /v1/jobs/123 HTTP/1.1
Host: api.example.com
Content-Type: application/json
Content-Length: 49

{
    "position": {
        "title": "PHP Developer"
    }
}

if ($_SERVER["REQUEST_METHOD"] == 'PUT') {

    $body = file_get_contents('php://input');
    switch(strtolower($_SERVER['HTTP_CONTENT_TYPE'])) {
        case "application/json":
            $job = json_decode($body);
            break;
        case "text/xml":
            // Разбор данных
```

```
            break;
    }
    // Проверка данных
    // Создание нового ресурса
    $request = explode('/', substr($_SERVER['PATH_INFO'], 1));
    $resource = array_shift($request);
    $id = create($job, $request[0]); // С идентификатором из запроса
    $json = json_encode(array('id' => $id));
    http_response_code(201); // Создание ресурса
    $site = 'https://api.example.com';
    header("Location: $site/" . $_SERVER['REQUEST_URI'] . "/$id");
    print $json;
}
```

Независимо от метода запроса (PUT или POST) может использоваться один набор ответов. Вызов PUT для уже существующего ресурса (например, если ресурс */v1/job/123* уже определен) перезаписывает текущее значение. В этом случае верните код ответа 200 (OK) вместо 201 («Создано»), если только вы не используете некоторую разновидность контроля версий, которая предотвращает такое присваивание. В таком случае верните код 409 («Конфликт»).

Запросы POST не являются безопасными, поэтому они могут иметь побочные эффекты. Кроме того, они не идемпотентны, так что повторная выдача одного запроса приведет к повторному созданию нескольких ресурсов. Этим запросы POST отличаются от запросов PUT, которые тоже не являются безопасными, но обладают свойством идемпотентности: многократная выдача запроса PUT эквивалентна его однократной выдаче.

См. также

Рецепт 14.2 — обращение по URL-адресу с использованием метода POST; Рецепты 15.3, 15.5 и 15.6 — предоставление доступа к ресурсам с использованием других методов; документация по `php://input`.

15.5. Редактирование ресурса

Задача

Требуется реализовать возможность обновления ресурсов.

Решение

Принимайте запросы, использующие метод PUT. Прочитайте тело запроса. Верните признак успеха.

Для запроса PUT с URL-адресом *http://api.example.com/v1/jobs/123*:

```
PUT /v1/jobs/123 HTTP/1.1
Host: api.example.com
```

```
Content-Type: application/json
Content-Length: 49

{
    "position": {
        "title": "PHP Developer"
    }
}
```

используется следующий код PHP:

```
if ($_SERVER["REQUEST_METHOD"] == 'PUT') {
    $body = file_get_contents('php://input');
    switch(strtolower($_SERVER['HTTP_CONTENT_TYPE'])) {
        case "application/json":
            $job = json_decode($body);
            break;
        case "text/xml":
            // Разбор данных
            break;
    }

    // Проверка данных
    // Модификация ресурса
    $request = explode('/', substr($_SERVER['PATH_INFO'], 1));
    $resource = array_shift($request);
    $id = update($job, $request[0]); // С идентификатором из запроса
    http_response_code(204); // Нет контента
}
```

При этом генерируется следующий результат:

```
HTTP/1.1 204 No Content
```

Комментарий

Для обновления ресурсов принимайте запросы PUT. Ресурс, предоставленный в теле запроса, заменяет текущий ресурс. Логика разбора запроса объясняется в разделе 15.4.

Если запрос был обработан успешно, верните код 204 («Нет контента»). Не возвращайте код 201, потому что ресурс уже существует. Можно вернуть код 200, но код 204 предпочтителен, если вы не возвращаете тело HTTP. С этим кодом клиент точно знает, что ничего не было потеряно.

Запросы PUT не являются безопасными, но они идемпотентны, потому что ресурс, изменяемый методом PUT, полностью заменяет текущий экземпляр. К сожалению, это означает, что исправление всего одной опечатки потребует повторной передачи всего ресурса.

Некоторые сайты поддерживают частичное обновление методом PUT. Например, следующий запрос не изменяет весь ресурс, а ограничивается обновлением почтового индекса:

```
PUT /v1/jobs/123 HTTP/1.1
Host: api.example.com
Content-Type: application/json
Content-Length: 43
{
    "location" {
        "postalCode": 94043
    }
}
```

При таком подходе бывает трудно отличить, когда поле удаляется, а когда его значение намеренно не предоставляется. Для частичных обновлений рекомендуется использовать метод PATCH, что позволяет выбирать разное поведение для методов PUT и PATCH.

См. также

Рецепт 14.3 — обращение по URL-адресу с произвольным методом; Рецепты 15.3, 15.4 и 15.6 — предоставление доступа к ресурсам с другими методами; RFC 5789 — метод HTTP PATCH.

15.6. Удаление ресурса

Задача

Требуется реализовать возможность удаления ресурсов.

Решение

Принимайте запросы, использующие метод DELETE. Верните признак успеха.

Для запроса DELETE с URL-адресом *http://api.example.com/v1/jobs/123*:

```
DELETE /v1/jobs/123 HTTP/1.1
Host: api.example.com
```

используется следующий код PHP:

```
if ($_SERVER["REQUEST_METHOD"] == 'DELETE') {
    // Удаление ресурса

    $request = explode('/', substr($_SERVER['PATH_INFO'], 1));
    $resource = array_shift($request);
    $success = delete($request[0]); // С идентификатором из запроса

    http_response_code(204); // Нет контента
}
```

При этом генерируется следующий результат:

```
HTTP/1.1 204 No Content
```

Комментарий

Для удаления ресурсов принимайте запросы DELETE. Если запрос был обработан успешно, верните код 204 («Нет контента»). Можно вернуть код 200, но код 204 предпочтителен, если вы не возвращаете тело HTTP. С этим кодом клиент точно знает, что ничего не было потеряно.

Если ресурс не существует (потому что он никогда не существовал или был удален ранее), верните код 404 («Не найдено»). Если доступ к ресурсу безвозвратно утрачен (в отличие от ситуаций, когда ресурс никогда не существовал или был временно удален с возможностью восстановления), верните код 410 («Удален»). Данная возможность часто используется, если весь родительский ресурс потерял актуальность, например если вы отказались от поддержки возможности обработки вакансий.

Запросы DELETE не являются безопасными, но они идемпотентны, потому что многократное удаление одного ресурса приводит к тому же результату, что и его однократное удаление.

См. также

Рецепт 14.3 — обращение по URL-адресу с произвольным методом; Рецепты 15.3, 15.4 и 15.5 — предоставление доступа к ресурсам с другими методами.

15.7. Сообщения об ошибках и сбоях

Задача

Требуется сообщить о произошедшей ошибке.

Решение

Верните код статуса 4xx для ошибок на стороне клиента. Предоставьте сообщение с дополнительной информацией:

```
http_response_code(401); // Не авторизован

$error_body = [
    "error" => "Unauthorized",
    "code" => 1,
    "message" => "Only authenticated users can read " . $_SERVER['REQUEST_URI'],
    "url" => "http://developer.example.com/error/1"
];

print json_encode($error_body);
```

Верните код статуса 5xx для ошибок на стороне сервера. Предоставьте сообщение с дополнительной информацией:

```
http_response_code(503); // Сайт недоступен
$error_body = [
    "error" => "Down for maintenance",
    "code" => 2,
    "message" => "Check back in two hours.",
    "url" => "http://developer.example.com/error/2"
];
print json_encode($error_body);
```

Комментарий

Полезные и содержательные сообщения об ошибках порадуют пользователей ваших API. Хорошее сообщение об ошибке содержит конкретную информацию, объясняет, что пошло не так и как решить проблему (если это возможно).

Для REST-совместимых серверов эта информация делится на две части: код статуса HTTP и сообщение об ошибке, возвращаемое в теле ответа. Коды состояния HTTP делятся на две большие группы.

Семейство кодов 4xx обозначает ошибки на стороне клиента, такие как неверные учетные данные аутентификации (401), запрет на обращение к ресурсу (403) или недоступность ресурса (410).

Не стоит полагать, что ошибка 4xx в чем-то обвиняет клиента, иногда невозможно знать заранее, что запрос окажется ошибочным (например, если пользователь отозвал маркер авторизации или сервер объявил API устаревшим без предварительного оповещения). Скорее речь идет о проблемах, которые могут быть исправлены клиентом посредством предоставления правильной информации (например, действительных учетных данных аутентификации), изменения запроса (обращение только к тем ресурсам, которые ему разрешено запрашивать) или полного отказа от запроса (нет — значит нет).

Семейство кодов 5xx обозначает ошибки на стороне сервера (дело не в тебе, дело во мне): недоступность сервиса (503), неожиданная ошибка из-за программного сбоя (500) и т. д.

Все эти проблемы полностью неподконтрольны клиенту и могут быть устранены только провайдером API. Их невозможно решить простым изменением запроса. Вместо этого придется ждать, пока на сервере будет исправлена ошибка, завершены технические работы или восстановлена способность обработки трафика.

В таблице 15.2 перечислены распространенные коды статуса HTTP, используемые для передачи информации об ошибках.

Впрочем, одного лишь кода обычно недостаточно для полного описания ошибки. По этой причине следует предоставить в ответе сообщение об ошибке — в идеале в формате, совпадающем с форматом запроса (например, JSON или XML).

Минимальное сообщение об ошибке представляет собой строку текста с описанием проблемы. Но хотя строка объясняет суть ошибки, написать код разбора строки достаточно сложно, поэтому лучше включить числовой код ошибки

Таблица 15.2. Коды статуса HTTP, используемые для передачи информации об ошибках

Код статуса	Смысл	Описание
400	Неверный запрос	Недопустимый синтаксис или другая ошибка общего характера
401	Не авторизован	Необходимо предоставить действительные данные аутентификации
403	Запрещен доступ	Доступ к ресурсу запрещен по другим причинам, кроме недействительных данных аутентификации
404	Не найдено	Ресурс не существует (но может появиться в будущем)
405	Метод недопустим	Метод не может вызываться для данного ресурса
410	Удален	Ресурс не существует и не будет существовать
429	Слишком много запросов	Нарушение квоты или ограничения частоты
500	Внутренняя ошибка сервера	Общая ошибка сервера
503	Сервис недоступен	Сервер перегружен запросами или отключен для выполнения технических работ

и короткую строку. Дополнительно можно включить URL-адрес страницы, которая содержит более подробное описание проблемы или дает возможность пользователям задать вопросы по поводу ее решения.

Пример:

```
$http_error_code = 401;
$error_body = [
    "error" => "Unauthorized",
    "code" => 1,
    "message" => "Only authenticated users can read " . $_SERVER['REQUEST_URI'],
    "url" => "http://developer.example.com/error/1"
];
http_response_code($http_error_code); // Не авторизован
print json_encode($error_body);
```

См. также

RFC 2616 — разделы 10.4 и 10.5.

15.8. Поддержка нескольких форматов

Задача

Требуется поддерживать несколько форматов (например, JSON и XML).

Решение

Используйте расширения файлов:

```
http://api.example.com/people/rasmus.json
http://api.example.com/people/rasmus.xml
// Разбиение URL-адреса
$request = explode('/', $_SERVER['PATH_INFO']);
// Выделение корневого ресурса и типа
$resource = array_shift($request);
$file = array_pop($request);
$dot = strrpos($file, ".");
if ($dot === false) { // Обратите внимание: три знака равенства
    $request[] = $file;
    $type = 'json'; // Значение по умолчанию
} else {
    $request[] = substr($file, 0, $dot);
    $type = substr($file, $dot + 1);
}
```

Или обеспечьте поддержку заголовка HTTP `Accept`, чтобы разрешить запросы к адресам вида *http://api.example.com/people/rasmus*:

```
GET /people/rasmus HTTP/1.1
Host: api.example.com
Accept: application/json,text/html

require_once 'HTTP2.php';
$http = new HTTP2;
$supportedTypes = array(
    'application/json',
    'text/xml',
);
$type = $http->negotiateMimeType($supportedTypes, false);
if ($type === false) {
    http_response_code(406); // Неприемлемо
    $error_body = 'Choose one of: ' . join(',', $supportedTypes);
    print json_encode($error_body);
} else {
    // Форматирование ответа в зависимости от $type
}
```

Если все остальные варианты не подходят, читайте параметр запроса:

```
http://api.example.com/people/rasmus?format=json
http://api.example.com/people/rasmus?format=xml
$type = $_GET['format'];
```

Комментарий

Если REST-совместимый API поддерживает несколько форматов (например, JSON и XML), разработчик может сообщить, какой формат он хочет использовать, несколькими способами.

Первый вариант — использовать расширения файлов, такие как *http://api.example.com/people/rasmus.json* и *http://api.example.com/people/rasmus.xml*. Так как речь идет не о «настоящих» файлах, данный способ требует разбора `$_SERVER['PATH_INFO']`:

```
// Разбиение URL-адреса
$request = explode('/', $_SERVER['PATH_INFO']);

// Выделение корневого ресурса и типа
$resource = array_shift($request);
$file = array_pop($request);
$dot = strrpos($file, '.');
if ($dot === false) { // Обратите внимание: три знака равенства
    $request[] = $file;
    $type = 'json'; // Значение по умолчанию
} else {
    $request[] = substr($file, 0, $dot);
    $type = substr($file, $dot + 1);
}

// $type содержит json, xml  и т. д.
```

Мы выделяем файловую часть из URL-адреса и ищем в ней символ «.» в конце. Если он присутствует, программа извлекает имя ресурса и тип при помощи `substr()`. Если символ не найден, мы возвращаемся к значению по умолчанию.

Недостаток этого способа заключается в том, что клиент может запрашивать только один конкретный тип представления. Если он запрашивает версию JSON, а вы ее не поддерживаете, клиент не сможет указать допустимый альтернативный формат в том же запросе.

Сразу несколько представлений ресурса могут быть доступны по одному адресу, например *http://api.example.com/people/rasmus*. В таком случае клиент может перечислить возможные форматы в порядке предпочтения. Это позволит вам согласовать с клиентом наиболее подходящий формат для возвращения ресурса.

В таком случае клиент передает информацию о предпочтительных форматах с использованием заголовка HTTP `Accept`:

```
GET /people/rasmus HTTP/1.1
Host: api.example.com
Accept: application/json,text/html
```

К сожалению, правильно разобрать заголовок `Accept` не так просто, поэтому для этого лучше воспользоваться библиотекой, например HTTP2 из репозитория PEAR:

```
require_once 'HTTP2.php';
$http = new HTTP2;
$supportedTypes = array(
    'application/json',
    'text/xml',
);
```

```
$type = $http->negotiateMimeType($supportedTypes, false);
if ($type === false) {
    http_response_code(406); // Неприемлемо
    $error_body = 'Choose one of: ' . join(',', $supportedTypes);
    print json_encode($error_body);
} else {
    // Форматирование ответа в зависимости от $type
}
```

В этом решении вы указываете, что поддерживаются форматы JSON и XML, а затем используете функцию `$http->negotiateMimeType()` для возвращения наиболее предпочтительного формата из предоставленного списка.

Наконец, формат можно передать в параметре запроса:

```
$type = $_GET['format'];
```

Несмотря на свою простоту и легкость реализации, такое решение не считается признаком REST-совместимой архитектуры.

См. также

Класс PEAR HTTP2; заголовок HTTP Accept.

16 Сервисы Интернета

16.0. Введение

Еще до появления HTTP существовали FTP, IMAP, POP3 и много других протоколов с загадочными именами. Многие пользователи быстро оценили браузеры — ведь браузер предоставлял интегрированную среду, в которой можно было проверить почту, передать файлы и просмотреть документы, не отвлекаясь на технические подробности коммуникационных протоколов. PHP предоставляет функции для работы с этими протоколами (как встроенные, так и доступные через PEAR). Это позволяет использовать PHP для создания интерфейсных приложений, выполняющих всевозможные сетевые задачи (например, поиск доменных имен или отправка электронной почты). Хотя PHP упрощает эту работу, важно понимать достоинства и недостатки каждого протокола.

Рецепты 16.1–16.3 посвящены самому популярному сервису — электронной почте. В Рецепте 16.1 показано, как отправить простое сообщение. В Рецепте 16.2 описаны сообщения в кодировке MIME, позволяющие передавать как простой текст, так и сообщения в формате HTML. Протоколы IMAP и POP3, используемые для чтения почтовых ящиков, рассматриваются в Рецепте 16.3.

В Рецепте 16.4 рассматривается передача файлов по FTP — протоколу отправки и получения файлов по Интернету. Серверы FTP могут требовать от пользователя входа с паролем или же разрешают анонимное использование.

Поиск на серверах LDAP является темой Рецепта 16.5, а в Рецепте 16.6 рассматривается аутентификация пользователей на серверах LDAP. Серверы LDAP используются как адресные книги и централизованные хранилища информации о пользователях. Они оптимизируются для быстрой выборки информации и могут настраиваться для репликации данных, гарантирующей высокую надежность и малое время отклика.

Глава завершается рецептами, относящимися к сетевым операциям. Рецепт 16.7 посвящен преобразованиям DNS — как преобразованиям имен доменов в IP-

адреса, так и наоборот. В Рецепте 16.8 рассказано о том, как проверить доступность хоста при помощи модуля PEAR ping. О том, как получить информацию о доменах, вы узнаете в Рецепте 16.9.

Некоторые сетевые протоколы также рассматриваются в других главах. Об использовании протокола HTTP подробно рассказано в главе 13. Протоколы, объединяющие HTTP и JSON, рассматриваются в главах 14 и 15. В этих двух главах обсуждается использование и предоставление доступа к ресурсам в REST-совместимых API.

16.1. Отправка почты

Задача

Требуется отправить сообщение электронной почты. Отправка может быть прямой реакцией на действие пользователя (например, на регистрацию на вашем сайте) или же происходить периодически в заданное время (например, рассылка еженедельного бюллетеня).

Решение

Воспользуйтесь классом `ezcMailComposer` из библиотеки Zeta Component:

```
$message = new ezcMailComposer();
$message->from = new ezcMailAddress('webmaster@example.com');
$message->addTo(new ezcMailAddress('adam@example.com', 'Adam'));
$message->subject = 'New Version of PHP Released!';
$body = 'Go to http://www.php.net and download it today!';
$message->plainText = $body;
$message->build();

$sender = new ezcMailMtaTransport();
$sender->send($message);
```

Если решение с классом `ezcMailComposer` недоступно, воспользуйтесь встроенной функцией PHP `mail()`:

```
$to = 'adam@example.com';
$subject = 'New Version of PHP Released!';
$body = 'Go to http://www.php.net and download it today!';

mail($to, $subject, $body);
```

Комментарий

Класс `ezcMailComposer` из библиотеки Zetacomponent строит сообщения электронной почты. Способ отправки сообщения компонентом зависит от используемой реализации `ezcMailTransport`. В приведенном примере во внутренней реализации

ezcMailMtaTransport() задействована функция PHP mail(), использующая конфигурацию PHP. Класс ezcMailSmtpTransport может использоваться для прямого взаимодействия с сервером SMTP:

```
$message = new ezcMailComposer();
$message->from = new ezcMailAddress('webmaster@example.com');
$message->addTo(new ezcMailAddress('adam@example.com', 'Adam'));
$message->subject = 'New Version of PHP Released!';
$body = 'Go to http://www.php.net and download it today!';
$message->plainText = $body;
$message->build();

$host = 'smtpauth.example.com';
$username = 'philb';
$password = 'jf430k24';
$port = 587;

$smtpOptions = new ezcMailSmtpTransportOptions();
$smtpOptions->preferredAuthMethod = ezcMailSmtpTransport::AUTH_LOGIN;

$sender = new ezcMailSmtpTransport($host, $username, $password, $port,
                                   $smtpOptions);

$sender->send($message);
```

Если вариант с классом ezcMailComposer невозможен, используйте встроенную функцию mail(). Программа, используемая mail() для отправки почты, задается конфигурационной переменной sendmail_path в файле php.ini. Если вы работаете в Windows, присвойте переменной SMTP имя хоста сервера SMTP. Адрес From берется из переменной sendmail_from.

В первом параметре mail() передается адрес электронной почты получателя, во втором — тема сообщения, а в последнем — тело сообщения. В четвертом параметре (который не является обязательным) можно передать дополнительные заголовки. Например, вот как добавить заголовки Reply-To и Organization:

```
$to = 'adam@example.com';
$subject = 'New Version of PHP Released!';
$body = 'Go to http://www.php.net and download it today!';
$header = "Reply-To: webmaster@example.com\r\n"
         ."Organization: The PHP Group";

mail($to, $subject, $body, $header);
```

Разделяйте заголовки символами \r\n, но не добавляйте \r\n за последним заголовком.

Независимо от выбранного способа стоит написать функцию-обертку, используемую для отправки почты. Пересылка всей почты через эту функцию упрощает ведение журнала и проверки всех отправленных сообщений:

```
function mail_wrapper($to, $subject, $body, $headers) {
    mail($to, $subject, $body, $headers);
    error_log("[MAIL][TO: $to]");
}
```

Каждое отправленное сообщение регистрируется в журнале с сохранением информации о получателе. Сохраненная информация содержит временную метку, благодаря которой вам будет проще разобраться с жалобами на попытки использования вашего сайта для рассылки спама. Также можно создать список заблокированных адресов электронной почты, по которым отправка сообщений с сайта запрещается, или проверять адреса всех получателей, чтобы сократить количество возвратов, и т. д.

См. также

Рецепт 9.4 — проверка адресов электронной почты; Рецепт 16.2 — отправка электронной почты с контентом MIME; Рецепт 16.3 — дополнительная информация о получении почты; документация по функции `mail()`; класс PEAR `Mail`; класс `ezcMailComposer`; RFC 822.

16.2. Отправка почты с контентом MIME

Задача

Требуется отправить сообщение электронной почты с данными MIME. Например, нужно отправить сообщение, состоящее из нескольких частей, содержащих простой текст и HTML. Почтовый клиент с поддержкой MIME автоматически отобразит нужную часть.

Решение

Воспользуйтесь классом `ezcMailComposer` из библиотеки Zetacomponent, задав значения свойств `plainText` и `htmlText` следующим образом:

```
$message = new ezcMailComposer();
$message->from = new ezcMailAddress('webmaster@example.com');
$message->addTo(new ezcMailAddress('adam@example.com', 'Adam'));
$message->subject = 'New Version of PHP Released!';
$body = 'Go to http://www.php.net and download it today!';
$message->plainText = $body;
$html = '<html><body><b>Hooray!</b> New PHP Version!</body></html>';
$message->htmlText = $html;
$message->build();

$sender = new ezcMailMtaTransport();
$sender->send($message);
```

Комментарий

Если задать свойство `htmlText`, класс `ezcMailComposer` выполнит всю «черную работу» по созданию заголовков и ограничителей за вас, чтобы программы чтения сообщений могли правильно интерпретировать сообщения.

Класс `ezcMailComposer` позволяет легко включить встроенное изображение. Просто укажите соответствующий локальный путь в атрибуте `src` тега `<img/>`:

```
$message = new ezcMailComposer();
$message->from = new ezcMailAddress('webmaster@example.com');
$message->addTo(new ezcMailAddress('adam@example.com', 'Adam'));
$message->subject = 'New Version of PHP Released!';
$body = 'Go to http://www.php.net and download it today!';
$message->plainText = $body;
$html = '<html>Me: <img src="file:///home/me/picture.png"/></html>';
$message->htmlText = $html;
$message->build();

$sender = new ezcMailMtaTransport();
$sender->send($message);
```

При вызове `$message->build()` класс `ezcMailComposer` ищет файл, указанный в теге `<img/>`, включает его содержимое в сообщение как вложение и обновляет разметку HTML, включая в нее ссылку на вложенное сообщение.

Чтобы добавить в сообщение вложение (например, графику или архив), вызовите метод `addFileAttachment()` или `addStringAttachment()`:

```
$message = new ezcMailComposer();
$message->from = new ezcMailAddress('webmaster@example.com');
$message->addTo(new ezcMailAddress('adam@example.com', 'Adam'));
$message->subject = 'New Version of PHP Released!';
$body = 'Go to http://www.php.net and download it today!';
$message->plainText = $body;
$message->addFileAttachment('/home/me/details.png','image','png');
$message->addStringAttachment('extra.txt','Some text', 'text/plain');

$message->build();

$sender = new ezcMailMtaTransport();
$sender->send($message);
```

См. также

Рецепт 16.1 — отправка простых сообщений; Рецепт 16.3 — дополнительная информация о получении почты; документация по классу `ezcMailComposer` библиотеки Zetacomponent.

16.3. Чтение почты с использованием протокола IMAP или POP3

Задача

Требуется прочитать почту с использованием протокола IMAP или POP3, например для создания почтового клиента с веб-интерфейсом.

Решение

Воспользуйтесь расширением IMAP, которое поддерживает как протокол IMAP, так и POP3:

```
// Открыть подключение IMAP
$mail = imap_open('{mail.server.com:143}',      'username', 'password');

// или открыть подключение POP3
$mail = imap_open('{mail.server.com:110/pop3}', 'username', 'password');

// Получить список всех заголовков
$headers = imap_headers($mail);

// Получить объект заголовка для последнего сообщения в почтовом ящике
$last = imap_num_msg($mail);
$header = imap_header($mail, $last);

// Получить тело того же сообщения
$body = imap_body($mail, $last);

// Закрыть подключение
imap_close($mail);
```

Комментарий

Библиотека, используемая PHP для поддержки IMAP и POP3, предоставляет неимоверно широкий набор возможностей, позволяющих написать практически полноценный почтовый клиент. Однако со всеми этими возможностями приходит и сложность. Достаточно сказать, что в PHP существуют 73 разные функции, имена которых начинаются со слова `imap`, причем при этом даже не учитывается, что некоторые из них также поддерживают POP3 и NNTP.

И все же основная схема взаимодействия с почтовым сервером достаточно прямолинейна. Как и многие операции в PHP, она начинается с открытия подключения и получения дескриптора:

```
$mail = imap_open('{mail.server.com:143}', 'username', 'password');
```

Эта команда открывает подключение IMAP к серверу с именем `mail.server.com` через порт 143. Во втором и третьем аргументе передается имя пользователя и пароль.

Чтобы вместо этого открыть подключение POP3, присоедините к серверу и порту суффикс `/pop3`. Так как сервер POP3 обычно работает на порте 110, добавьте после имени сервера `:110`:

```
$mail = imap_open('{mail.server.com:110/pop3}', 'username', 'password');
```

Чтобы зашифровать подключение с SSL, добавьте суффикс `/ssl` по аналогии с тем, как это было сделано для `pop3`. Также убедитесь в том, что при сборке установки PHP использовался конфигурационный параметр `--with-imap-ssl` наряду с `--with-imap`. Также необходимо построить саму системную библиотеку

IMAP с поддержкой SSL. Если вы используете самоподписанный сертификат и хотите предотвратить попытки проверки, добавьте ключ `/novalidate-cert`. Наконец, подключения SSL чаще всего передают данные через порт 993 или 995. Все эти параметры могут следовать в произвольном порядке, так что следующая команда вполне допустима:

```
$mail = imap_open('{mail.server.com:993/novalidate-cert/pop3/ssl}',
                  'username', 'password');
```

Заключая переменную в фигурные скобки вместо строки в кавычках (`{$var}`), вы точно сообщаете PHP, какую переменную следует интерполировать. Таким образом, чтобы использовать интерполированные переменные в первом параметре `imap_open()`, экранируйте открывающий символ `{`:

```
$server = 'mail.server.com';
$port = 993;

$mail = imap_open("\{$server:$port}", 'username', 'password');
```

После открытия подключения можно обращаться к почтовому серверу с разными запросами. Так, для получения списка всех сообщений в почтовом ящике используется функция `imap_headers()`:

```
$headers = imap_headers($mail);
```

Она возвращает массив, в котором каждый элемент представляет собой отформатированную строку, соответствующую сообщению:

```
   A   189) 5-Aug-2007 Beth Hondl           an invitation (1992 chars)
```

Чтобы получить конкретное сообщение, используйте функции `imap_header()` и `imap_body()` для получения объекта заголовка и строки тела:

```
$header = imap_header($message_number);
$body   = imap_body($message_number);
```

Функция `imap_header()` возвращает объект с многими полями. Самые полезные из них — поля `subject`, `fromaddress` и `udate`. Полный список полей приведен в таблице 16.1.

Таблица 16.1. Поля imap_header()

Имя	Описание	Тип	Пример
date или Date	Дата в формате RFC 822: date('r')	String	Fri, 16 Aug 2002 01:52:24 -0400
subject или Subject	Тема сообщения	String	Re: PHP Cookbook Revisions
message_id	Уникальный идентификатор сообщения	String	<20030410020818.33915.php@news.example.com>
toaddress	Адрес, по которому было отправлено сообщение	String	php-general@lists.php.net

Таблица 16.1 (*продолжение*)

Имя	Описание	Тип	Пример
to	Разобранная версия поля toaddress	Object	mailbox: 'php-general', host: 'lists-php.net'
fromaddress	Адрес, с которого было отправлено сообщение	String	Ralph Josephs <ralph@example.net>
from	Разобранная версия поля fromaddress	Object	personal: 'Ralph Josephs', mailbox: 'ralph', host: 'example.net'
reply_toaddress	Адрес, на который следует отвечать для связи с автором	String	rjosephs@example.net
reply_to	Разобранная версия поля reply_toaddress	Object	Mailbox: 'rjosephs', host: 'example.net'
senderaddress	Отправитель сообщения; почти всегда идентичен полю from, но в отличие от него однозначно идентифицирует отправителя	String	Ralph Josephs <ralph@example.net>
sender	Разобранная версия поля reply_toaddress	Object	Personal: Ralph Josephs, mailbox: ralph, host: 'example.net'
Recent	Признак нового сообщения (появившегося с момента последней проверки почты пользователем)	String	Y или N
Unseen	Признак невидимого сообщения	String	Y или " "
Flagged	Признак помеченного сообщения	String	Y или " "
Answered	Признак отправки ответа на это сообщение	String	Y или " "
Deleted	Признак удаленного сообщения	String	Y или " "
Draft	Признак черновика	String	Y или " "
Size	Размер сообщения в байтах	String	1345
udate	Временная метка Unix для даты сообщения	Int	1013480645
Mesgno	Номер сообщения в группе	String	34943

Элемент `body` представляет собой обычную строку, но если сообщение состоит из нескольких частей (например, содержит как HTML, так и простой текст), `$body` содержит обе части и строки MIME, описывающие их:

```
------=_Part_1046_3914492.1008372096119
Content-Type: text/plain; charset=us-ascii
Content-Transfer-Encoding: 7bit

Plain-Text Message
```

```
------=_Part_1046_3914492.1008372096119
Content-Type: text/html
Content-Transfer-Encoding: 7bit

<html>HTML Message</html>
------=_Part_1046_3914492.1008372096119--
```

Чтобы не столкнуться с этой проблемой, используйте функцию `imap_fetchstructure()` в сочетании с `imap_fetchbody()` для получения информации о формате тела и извлечении только интересующих вас частей:

```
// Получение простого текста для сообщения $n
$st = imap_fetchstructure($mail, $n);
if (!empty($st->parts)) {
    for ($i = 0, $j = count($st->parts); $i < $j; $i++) {
        $part = $st->parts[$i];
        if ($part->subtype == 'PLAIN') {
             $body = imap_fetchbody($mail, $n, $i+1);
        }
     }
} else {
    $body = imap_body($mail, $n);
}
```

Если сообщение состоит из нескольких частей, `$st->parts` содержит массив объектов, описывающих эти части.

Свойство `part` содержит целое число, описывающее тип MIME основного тела. В таблице 16.2 представлено соответствие числовых кодов и типов MIME. Свойство `subtype` содержит подтип MIME и сообщает, является ли часть простым текстом (`plain`), разметкой HTML (`html`), графикой PNG (`png`) или другим типом, например `octet-stream`.

Таблица 16.2. Коды типов MIME для IMAP

Число	Тип MIME	Константа PHP	Описание	Примеры
0	text	TYPETEXT	Неформатированный текст	Простой текст, HTML, XML
1	multipart	TYPEMULTIPART	Сообщение, состоящее из нескольких частей	Смешанный контент, данные форм
2	message	TYPEMESSAGE	Инкапсулированное сообщение	Новости, HTTP
3	application	TYPEAPPLICATION	Данные приложения	Поток октетов, PDF, Zip
4	audio	TYPEAUDIO	Звуковой файл	MP3
5	image	TYPEIMAGE	Графическое изображение	GIF, JPEG, PNG
6	video	TYPEVIDEO	Видеоролик	MPEG, Quicktime
7	other	TYPEOTHER	Все остальное	Модели VRML

См. также

Рецепты 16.1 и 16.2 — дополнительная информация об отправке почты; документация по функциям `imap_open()`, `imap_header()`, `imap-body()` и IMAP вообще.

16.4. Получение и отправка файлов с использованием протокола FTP

Задача

Требуется переслать файлы с использованием протокола FTP.

Решение

Воспользуйтесь функциями FTP, встроенными в PHP:

```
$c = ftp_connect('ftp.example.com')     or die("Can't connect");
ftp_login($c, $username, $password)     or die("Can't login");
ftp_put($c, $remote, $local, FTP_ASCII) or die("Can't transfer");
ftp_close($c)                           or die("Can't close");
```

Также можно воспользоваться расширением cURL:

```
$c = curl_init("ftp://$username:$password@ftp.example.com/$remote");
// $local - место хранения файла на локальном компьютере
$fh = fopen($local, 'w') or die($php_errormsg);
curl_setopt($c, CURLOPT_FILE, $fh);
curl_exec($c);
curl_close($c);
```

Комментарий

Протокол FTP предоставляет механизм передачи файлов между компьютерами. В отличие от серверов HTTP, настроить сервер FTP для отправки и получения файлов относительно несложно.

Встроенные функции FTP не требуют дополнительных библиотек, но их необходимо явно активизировать ключом `--enable-ftp`. Так как эти функции специализируются на пересылке данных FTP, ими просто пользоваться для пересылки файлов.

Все операции FTP начинаются с установления связи с вашего компьютера (локального клиента) на другом компьютере (удаленный сервер):

```
$c = ftp_connect('ftp.example.com')     or die("Can't connect");
```

После подключения необходимо передать имя пользователя или пароль; удаленный сервер проверяет вас и разрешает начать сеанс:

```
ftp_login($c, $username, $password)     or die("Can't login");
```

Некоторые серверы FTP поддерживают так называемый *анонимный режим* FTP. В анонимном режиме пользователь может работать, не имея учетной записи в удаленной системе. При использовании анонимного режима FTP вводится имя пользователя `anonymous`, а в качестве пароля используется адрес вашей электронной почты.

Пример передачи файлов функциями `ftp_put()` и `ftp_get()`:

```
ftp_put($c, $remote, $local,  FTP_ASCII) or die("Can't transfer");
ftp_get($c, $local,  $remote, FTP_ASCII) or die("Can't transfer");
```

Функция `ftp_put()` копирует файл с вашего компьютера на удаленный сервер; функция `ftp_get()` копирует файл с удаленного сервера на ваш компьютер. В приведенном фрагменте `$remote` содержит путь к удаленному файлу, а `$local` определяет местонахождение файла на вашем компьютере.

Последний параметр этой функции определяет режим передачи файлов. С параметром `FTP_ASCII` файл передается в режиме ASCII-текста; в этом режиме завершители строк автоматически преобразуются при переходе к другой операционной системе. С параметром `FTP_BINARY` файл передается в двоичном режиме, в котором преобразования завершителей строк не выполняются.

Используйте функции `ftp_fget()` и `ftp_fput()` для получения или отправки файла по существующему файловому указателю (открытому функцией `fopen()`) вместо пути файловой системы. Например, в следующем примере читается файл, который записывается по существующему файловому указателю `$fp`:

```
$fp = fopen($file, 'w');
ftp_fget($c, $fp, $remote, FTP_ASCII)   or die("Can't transfer");
```

Наконец, для разрыва связи с удаленным хостом вызывается функция `ftp_close()`:

```
ftp_close($c)  or die("Can't close");
```

Интервал тайм-аута (в секундах) задается вызовом `ftp_set_option()`:

```
// Установить продолжительность тайм-аута, равную двум минутам:
set_time_limit(120);
$c = ftp_connect('ftp.example.com');
ftp_set_option($c, FTP_TIMEOUT_SEC, 120);
```

Тайм-аут по умолчанию составляет 90 секунд; с другой стороны, значение `max_execution_time` для сценария PHP по умолчанию составляет 30 секунд. Если подключения разрываются слишком рано, проверьте оба параметра.

Чтобы использовать расширение cURL, следует загрузить cURL и включить конфигурационный параметр `-with-curl` при построении PHP. Затем создайте дескриптор cURL вызовом `curl_init()` и укажите, что вы собираетесь делать, вызовом `curl_setopt()`. Функция `curl_setopt()` получает три параметра: ресурс

cURL, константа изменяемого параметра cURL и значение, присваиваемое второму параметру. В Решении используется константа `CURLOPT_FILE`:

```
$c = curl_init("ftp://$username:$password@ftp.example.com/$remote");
// $local - место хранения файла на локальном клиенте
$fh = fopen($local, 'w') or die($php_errormsg);
curl_setopt($c, CURLOPT_FILE, $fh);
curl_exec($c);
curl_close($c);
```

Функции `curl_init()` передается используемый URL-адрес. Так как URL начинается с *ftp://*, cURL знает, что нужно использовать протокол FTP. Вместо отдельного вызова для установления связи с удаленным сервером имя пользователя и пароль встраиваются прямо в URL. Затем указывается место для хранения файла на сервере. Теперь мы открываем файл с именем `$local` для записи и передаем дескриптор при вызове `curl_setopt()` как значение `CURLOPT_FILE`. При передаче файла cURL автоматически выполняет запись в этот дескриптор. Когда все будет готово, вызовите `curl_exec()` для запуска операции, а затем `curl_close()` для закрытия подключения.

См. также

Документация по расширению FTP и cURL; RFC 959.

16.5. Поиск адресов с использованием LDAP

Задача

Требуется запросить адресную информацию с сервера LDAP.

Решение

Воспользуйтесь расширением LDAP для PHP:

```
$ds = ldap_connect('ldap.example.com')                 or die($php_errormsg);
ldap_bind($ds)                                         or die($php_errormsg);
$sr = ldap_search($ds, 'o=Example Inc., c=US', 'sn=*') or die($php_errormsg);
$e  = ldap_get_entries($ds, $sr)                       or die($php_errormsg);

for ($i=0; $i < $e['count']; $i++) {
    echo $info[$i]['cn'][0] . ' (' . $info[$i]['mail'][0] . ')<br>';
}

ldap_close($ds)                                        or die($php_errormsg);
```

Комментарий

На серверах LDAP (Lightweight Directory Access Protocol) хранится справочная информация (имена, адреса и т. д.), и вы можете обращаться к ним с запросами.

Во многих отношениях такой сервер работает как база данных, но он оптимизирован для хранения информации о людях.

Кроме того, вместо линейной структуры, характерной для базы данных, сервер LDAP позволяет создавать информационные иерархии. Например, работники могут быть сгруппированы по отделам (маркетинговый, технический и т. д.) или по географическому признаку (Северная Америка, Европа, Азия). Это позволяет легко найти всех работников конкретного подразделения компании.

При использовании LDAP хранилище адресов называется *источником данных*. Каждая запись в источнике данных обладает глобально-уникальным идентификатором, называемым *отличительным именем* (distinguished name). Отличительное имя включает как имя работника, так и информацию о компании. Например, Джону Смиту, работающему в американской компании Example Inc., присваивается отличительное имя `cn=John Q. Smith, o=Example Inc., c=US`. В LDAP сокращение `cn` означает стандартное имя, `o` — организацию и `c` — страну.

Поддержку LDAP необходимо включить параметром `--with-ldap`. Сервер LDAP можно загрузить по адресу *http://www.openldap.org/*. В этом рецепте предполагается, что читатель уже обладает базовыми знаниями о LDAP.

Взаимодействие с сервером LDAP состоит из четырех шагов: подключения, аутентификации, поиска записей и отключения. Помимо поиска также можно добавлять, изменять и удалять записи.

Работа начинается с подключения к серверу LDAP и аутентификации с использованием процесса, называемого связыванием (binding):

```
$ds = ldap_connect('ldap.example.com')         or die($php_errormsg);
ldap_bind($ds)                                 or die($php_errormsg);
```

Если `ldap_bind()` передается только дескриптор подключения `$ds`, функция выполняет анонимное связывание. Если вы хотите использовать конкретное имя пользователя и пароль, передайте их во втором и третьем параметрах:

```
ldap_bind($ds, $username, $password)            or die($php_errormsg);
```

После установления связи с сервером можно запрашивать с него информацию. Так как информация образует иерархию, во втором параметре должно передаваться базовое отличительное имя. Также при вызовах могут передаваться критерии поиска. Например, следующий запрос находит всех работников с фамилией `Jones`, работающих в американской компании Example Inc.:

```
$sr = ldap_search($ds, 'o=Example Inc., c=US', 'sn=Jones') or die($php_errormsg);
$e  = ldap_get_entries($ds, $sr)                           or die($php_errormsg);
```

После того как `ldap_search()` проведет поиск и вернет результаты, используйте функцию `ldap_get_entries()` для получения записей данных. Затем переберите элементы массива записей `$e`:

```
for ($i=0; $i < $e['count']; $i++) {
    echo $e[$i]['cn'][0] . ' (' . $e[$i]['mail'][0] . ')<br>';
}
```

Вместо того чтобы вызвать `count($e)`, мы используем заранее вычисленный размер, хранящийся в `$e['count']`. В цикле для каждой записи выводится имя и адрес электронной почты:

```
David Sklar (sklar@example.com)
Adam Trachtenberg (adam@example.com)
```

Функция `ldap_search()` проводит поиск по всему поддереву, начиная с заданного базового отличительного имени. Чтобы ограничить результаты определенным уровнем, используйте функцию `ldap_list()`. Так как поиск выполняется по меньшему набору записей, `ldap_list()` может работать намного быстрее `ldap_search()`.

См. также

Рецепт 16.6 — аутентификация пользователей на серверах LDAP; документация по LDAP; RFC 2251.

16.6. Использование сервера LDAP для аутентификации пользователей

Задача

Требуется ограничить доступ к некоторым частям сайта, разрешив его только проверенным пользователям. Вместо того чтобы проверять людей по базе данных или с использованием базовой авторизации HTTP, необходимо использовать сервер LDAP.

Хранение всей информации о пользователях на сервере LDAP упрощает централизованное администрирование.

Решение

Воспользуйтесь поддержкой аутентификации LDAP в классе PEAR `Auth`:

```
$options = array('host'     => 'ldap.example.com',
                 'port'     => '389',
                 'base'     => 'o=Example Inc., c=US',
                 'userattr' => 'uid');

$auth = new Auth('LDAP', $options);

// Начало проверки.
// Отображение экрана для ввода учетных данных
$auth->start();

if ($auth->getAuth()) {
```

```
    // Контент для проверенных пользователей
} else {
    // Контент для анонимных пользователей
}

// Разрыв связи
$auth->logout();
```

Комментарий

Серверы LDAP предназначены для хранения, поиска и выборки адресов, и для этой цели они подходят лучше, чем стандартные базы данных (например, MySQL или Oracle). Серверы LDAP работают очень быстро, позволяют легко организовать управление доступом за счет предоставления разных разрешений разным группам пользователей, а к серверу можно обращаться с запросами из множества разных программ. Например, многие клиенты электронной почты могут использовать сервер LDAP как адресную книгу, и если вы адресуете сообщение пользователю «John Smith», сервер вернет адрес электронной почты *jsmith@example.com*.

Класс PEAR `Auth` обеспечивает возможность проверки пользователей по файлам, базам данных и серверам LDAP. В первом параметре передается способ аутентификации, а во втором — массив информации о том, как должна выполняться проверка. Пример:

```
$options = array('host'     => 'ldap.example.com',
                 'port'     => '389',
                 'base'     => 'o=Example Inc., c=US',
                 'userattr' => 'uid');

$auth = new Auth('LDAP', $options);
```

Этот фрагмент создает новый объект `Auth`, который проверяет пользователей при помощи сервера LDAP, находящегося по адресу *ldap.example.com* с портом 389. При проверке используется базовое имя каталога `o=Example Inc., c=US`, а имена пользователей проверяются по атрибуту `uid`. В поле `uid` (User Identifier, то есть «Идентификатор пользователя») обычно хранится имя пользователя для сайта или более общей учетной записи. Если ваш сервер не сохраняет атрибуты `uid` для каждого пользователя, используйте атрибут `cn`. В этом поле хранится полное имя пользователя вида «John Q. Smith».

Метод `Auth::auth()` также получает необязательный третий параметр — имя функции, отображающей форму для ввода учетных данных. Эта форма может иметь произвольный формат; единственное требование заключается в том, что полям для ввода данных должны быть присвоены имена `username` и `password`. Кроме того, форма должна отправлять данные методом POST:

```
$options = array('host'     => 'ldap.example.com',
                 'port'     => '389',
                 'base'     => 'o=Example Inc., c=US',
                 'userattr' => 'uid');
```

```
function pc_auth_ldap_signin() {
    $action = htmlentities($_SERVER['PHP_SELF']);
    print<<<_HTML_
<form method="post" action="$action">
Name: <input name="username" type="text"><br />
Password: <input name="password" type="password"><br />
<input type="submit" value="Sign In">
</form>
_HTML_;
}

$auth = new Auth('LDAP', $options, 'pc_auth_ldap_signin');
```

После того как объект `Auth` будет создан, аутентификация пользователя выполняется вызовом `Auth::start()`:

```
$auth->start();
```

Если пользователь уже прошел проверку, ничего не происходит. Для анонимного пользователя отображается форма ввода учетных данных. Чтобы проверить пользователя, `Auth::start()` подключается к серверу LDAP, осуществляет анонимное связывание и ищет адрес, для которого пользовательский атрибут, указанный в конструкторе, совпадает с именем пользователя, полученным от формы:

```
$options['userattr'] = $_POST['username'];
```

Если метод `Auth::start()` находит ровно одного человека, подходящего по заданному критерию, он получает отличительное имя пользователя и пытается выполнить аутентифицированное связывание, используя отличительное имя и пароль с формы как учетные данные для входа. Сервер LDAP сравнивает пароль с атрибутом `userPassword`, связанным с отличительным именем. Если значения совпадают, то аутентификация проходит успешно.

Вызов `Auth::getAuth()` возвращает логический признак, описывающий статус пользователя:

```
if ($auth->getAuth()) {
    print 'Welcome member! Nice to see you again.';
} else {
    print 'Welcome guest. First time visiting?';
}
```

Класс `Auth` использует встроенный сеансовый модуль для отслеживания пользователей, так что после прохождения аутентификации пользователь остается проверенным до истечения сеанса или его явного завершения вызовом:

```
$auth->logout();
```

См. также

Рецепт 16.5 — поиск на серверах LDAP; класс PEAR `Auth`.

16.7. Выполнение поиска DNS

Задача

Требуется найти доменное имя по IP-адресу или наоборот.

Решение

Воспользуйтесь функцией `gethostbyname()` или `gethostbyaddr()`:

```
$ip   = gethostbyname('www.example.com'); // 93.184.216.119
$host = gethostbyaddr('93.184.216.119'); // www.example.com
```

Комментарий

Обычно адресу, возвращаемому `gethostbyname()`, можно доверять — но имя, возвращаемое `gethostbyaddr()`, не является полностью надежным. Сервер DNS, ответственный за IP-адрес, может вернуть любое имя. Обычно администраторы настраивают сервер DNS так, чтобы тот возвращал правильное имя хоста, но злоумышленник может настроить свой сервер DNS для возвращения неправильных имен. Один из способов борьбы с подобными трюками — вызвать `gethostbyname()` для имени хоста, возвращаемого `gethostbyaddr()`, и убедиться в том, что имя преобразуется в исходный IP-адрес.

Если какая-либо из этих функций не может найти IP-адрес или доменное имя, она вместо `false` возвращает переданный ей аргумент. Проверка ошибок выполняется следующим образом:

```
$host = 'this is not a good host name!';
if ($host == ($ip = gethostbyname($host))) {
    // Ошибка
}
```

Этот фрагмент присваивает возвращаемое значение `gethostbyname()` переменной `$ip`, а также проверяет, что значение `$ip` не совпадает с исходным значением `$host`.

Иногда одно имя хоста может отображаться на несколько IP-адресов. Чтобы найти все хосты, используйте функцию `gethostbynamel()`:

```
$hosts = gethostbynamel('www.yahoo.com');
print_r($hosts);
```

Результат выглядит примерно так (конкретные IP-адреса зависят от вашего местонахождения):

```
Array
(
    [0] => 98.139.183.24
    [1] => 98.139.180.149
)
```

В отличие от `gethostbyname()` и `gethostbyaddr()`, функция `gethostbynamel()` возвращает массив, а не строку.

Также возможно выполнение более сложных задач, связанных с DNS. Например, функция `getmxrr()` позволяет получать записи MX:

```
getmxrr('yahoo.com', $hosts, $weight);
for ($i = 0; $i < count($hosts); $i++) {
    echo "$weight[$i] $hosts[$i]\n";
}
```

Если функция `gethostbyname()` получает A-записи IPv4, а функция `getmxrr()` получает MX-записи, функция `dns_get_record()` получает запись DNS указанного типа. Например, эта возможность может пригодиться для получения AAAA-записей IPv6:

```
$addrs = dns_get_record('www.yahoo.com', DNS_AAAA);
print_r($addrs);
```

Результат выглядит примерно так (как и в предыдущем примере, конкретный вывод зависит от вашего местонахождения):

```
Array
(
    [0] => Array
        (
            [host] => ds-any-fp3-real.wa1.b.yahoo.com
            [type] => AAAA
            [ipv6] => 2001:4998:f00b:1fe::3001
            [class] => IN
            [ttl] => 34
        )
    [1] => Array
        (
            [host] => ds-any-fp3-real.wa1.b.yahoo.com
            [type] => AAAA
            [ipv6] => 2001:4998:f00d:1fe::3001
            [class] => IN
            [ttl] => 34
        )
    [2] => Array
        (
            [host] => ds-any-fp3-real.wa1.b.yahoo.com
            [type] => AAAA
            [ipv6] => 2001:4998:f00d:1fe::3000
            [class] => IN
            [ttl] => 34
        )
    [3] => Array
        (
            [host] => ds-any-fp3-real.wa1.b.yahoo.com
            [type] => AAAA
            [ipv6] => 2001:4998:f00b:1fe::3000
            [class] => IN
            [ttl] => 34
        )
)
```

Каждый элемент возвращаемого массива представляет собой подмассив с информацией о типе записи, имени хоста, адресе IPv6, классе записи и сроке жизни (продолжительности кэширования).

За информацией о том, как обозначить интересующий вас тип записи DNS, обращайтесь к документации `dns_get_record()`.

Чтобы выполнять зонный перенос, динамические обновления DNS и другие операции, воспользуйтесь пакетом PEAR `Net_DNS2`.

См. также

Документация по функциям `gethostbyname()`, `gethostbyaddr()`, `gethostbynamel()`, `getmxrr()` и `dns_get_record()`; пакет PEAR `Net_DNS2`.

16.8. Проверка доступности хоста

Задача

Требуется проверить связь с хостом. Это позволит вам определить, работает ли хост и доступен ли он с вашего компьютера.

Решение

Воспользуйтесь пакетом PEAR `Net_Ping`:

```
require 'Net/Ping.php';

$ping = Net_Ping::factory();
if ($ping->checkHost('www.oreilly.com')) {
    print 'Reachable';
} else {
    print 'Unreachable';
}

$data = $ping->ping('www.oreilly.com');
```

Комментарий

Программа *ping* пытается отправить сообщение с вашего компьютера на другой компьютер. Если все проходит нормально, вы получаете статистику с хронометражом передачи. Ошибка означает, что программа *ping* по какой-то причине не может связаться с заданным хостом.

В случае ошибки `Net_Ping::checkhost()` возвращает `false`, а `Net_Ping::ping()` возвращает константу `PING_HOST_NOT_FOUND`. Если возникнут проблемы с запуском

ping (потому что `Net_Ping` представляет собой простую обертку для программы), возвращается `PING_FAILED`.

При успешном выполнении возвращается объект `Net_Ping_Result`. Этот объект содержит разнообразные методы для получения информации об операции *ping*. Пример:

```
require 'Net/Ping.php';
$ping = Net_Ping::factory();

$result = $ping->ping('www.oreilly.com');
print<<<_INFO_
Ping of www.oreilly.com ({$result->getTargetIp()})
with {$result->getTransmitted()} requests had
a minimum time of {$result->getMin()} ms and
a maximum time of {$result->getMax()} ms.
_INFO_
;
```

Результат выглядит примерно так:

```
Ping of www.oreilly.com (23.67.61.152)
with 3 requests had
a minimum time of 35.4 ms and
a maximum time of 40.586 ms.
```

Метод `Net_Ping::setArgs()` позволяет изменить некоторые параметры выполнения программы *ping*. Например, вызов `$ping->setArgs(array('count' => 7))` приказывает `Net_Ping` отправить семь проверочных пакетов вместо стандартного количества (обычно три или четыре).

См. также

Пакет PEAR `Net_Ping`.

16.9. Получение информации о доменном имени

Задача

Требуется получить контактную информацию или другие сведения о доменном имени.

Решение

Воспользуйтесь классом PEAR `Net_Whois`:

```
require 'Net/Whois.php';
$server = 'whois.godaddy.com';
```

```
$query  = 'oreilly.com';
$whois = new Net_Whois();
$data = $whois->query($query, $server);
```

Комментарий

Метод `Net_Whois::query()` возвращает длинную строку, содержимое которой лишний раз демонстрирует, сколько трудностей возникает с разбором результатов Whois. Например, код в Решении заносит в `$data` длинный текст, начало которого выглядит так:

```
Domain Name: OREILLY.COM
Registrar URL: http://www.godaddy.com
Updated Date: 2013-04-22 17:52:42
Creation Date: 1997-05-26 23:00:00
Registrar Expiration Date: 2014-05-25 23:00:00
Registrar: GoDaddy.com, LLC
Registrant Name: O'Reilly Media, Inc.
Registrant Organization: O'Reilly Media, Inc.
Registrant Street: 1005 Gravenstein Highway North
Registrant City: Sebastopol
Registrant State/Province: California
Registrant Postal Code: 95472
Registrant Country: United States
Admin Name: Admin Contact
Admin Organization: O'Reilly Media, Inc.
Admin Street: 1005 Gravenstein Highway North
Admin City: Sebastopol
Admin State/Province: California
Admin Postal Code: 95472
Admin Country: United States
Admin Phone: +1.7078277000
Admin Fax: +1.7078290104
Admin Email: nic-ac@oreilly.com
Tech Name: Tech Contact
Tech Organization: O'Reilly Media, Inc.
Tech Street: 1005 Gravenstein Highway North
Tech City: Sebastopol
…
```

Разные домены используют разные серверы Whois, а разные серверы Whois возвращают результаты в разных форматах. Чтобы узнать правильный сервер Whois для домена, начните с выдачи запроса к *whois.iana.org*. Результат, полученный от сервера, будет содержать строку, начинающуюся с `whois:`; она обозначает сервер, который следует использовать для верхнего уровня интересующего вас домена. После этого можно обратиться к указанному серверу за информацией о домене:

```
require 'Net/Whois.php';
$query  = 'oreilly.com';

$iana_server = 'whois.iana.org';

$whois = new Net_Whois();
```

```
$iana_data = $whois->query($query, $iana_server);
preg_match('/^whois:\s+(.+)$/m', $iana_data, $matches);
$tld_whois_server = $matches[1];

$tld_data = $whois->query($query, $tld_whois_server);

print $tld_data;
```

Затем в зависимости от полученной информации может последовать запрос к дополнительному серверу. Например, из второго запроса в приведенном коде следует, что ответственным сервером Whois для *oreilly.com* является сервер *whois.godaddy.com*.

См. также

Класс PEAR `Net_Whois`.

17 Графика

17.0. Введение

При помощи библиотеки GD можно использовать PHP для создания приложений, использующих динамическую графику для отображения биржевых котировок, результатов опросов, показателей производительности системы и даже программирования игр. Тем не менее графическое программирование не похоже на работу в Photoshop или GIMP; вы не рисуете линии мышью, а точно определяете тип, размер и позицию каждой фигуры.

У GD существует свой API, и PHP старается следовать его синтаксису и правилам выбора имен функций. Таким образом, читатели, знакомые с GD по другим языкам (таким как C или Perl), смогут легко использовать GD в коде PHP. Если же вы еще не имели дела с GD, придется потратить некоторое время на изучение, но вскоре вы начнете рисовать, как Пикассо.

Набор функций GD сильно зависит от того, с какой версией GD вы работаете и какие возможности были включены в процессе настройки конфигурации. GD может поддерживать форматы GIF, JPEG, PNG и WBMP. GD читает PNG и JPEG почти без потери качества. Кроме того, GD поддерживает альфа-каналы PNG, что позволяет вам назначить уровень прозрачности для каждого пиксела.

Помимо поддержки многих форматов файлов, GD позволяет рисовать точки, линии, прямоугольники, многоугольники, дуги, эллипсы и круги с использованием любых цветов. В Рецепте 17.1 рассматриваются прямолинейные, а в Рецепте 17.2 — криволинейные фигуры. О том, как заполнить фигуру узором вместо сплошной заливки, рассказано в Рецепте 17.3.

GD поддерживает вывод текста с использованием разнообразных шрифтов, включая встроенные шрифты и TrueType. Рецепт 17.4 демонстрирует тонкости работы трех основных функций вывода текста, а в Рецепте 17.5 объясняется, как обеспечить выравнивание текста по центру. Эти два рецепта закладывают осно-

ву для Рецепта 17.6, в котором графический шаблон объединяется с данными реального времени для создания динамических изображений. GD также позволяет создавать прозрачные изображения в форматах GIF и PNG. Определение прозрачности и ее использование при заполнении рассматривается в Рецепте 17.7.

Кроме создания новых изображений, вы также можете работать с существующими изображениями, например добавить идентификационный «водяной знак», наложив текст или графику поверх изображения. Эта тема рассматривается в Рецепте 17.8. Рецепт 17.9 посвящен генерированию миниатюр — уменьшенных версий изображений. За информацией об извлечении метаданных из цифровых фотографий и других изображений, использующих формат EXIF, обращайтесь к Рецепту 17.10.

Рецепт 17.11 отходит от темы GD: он показывает, как организовать защищенную поставку изображений с ограничением пользовательского доступа. Наконец, программа-пример в конце главы получает результаты опроса и строит динамическую гистограмму, которая показывает, какой процент пользователей проголосовал за каждый ответ.

GD поставляется вместе с PHP. В разделе электронной документации PHP, посвященном GD, также указано местонахождение дополнительных библиотек, необходимых для поддержки дополнительных графических форматов (например, JPEG и PNG) и шрифтов (например, TrueType).

Существует два простых способа узнать, какая версия GD установлена на вашем сервере (и установлена ли она вообще) и как она настроена. Первый способ — вызов `phpinfo()`. В верхней части должен присутствовать ключ `--with-gd`, а ниже на странице должен присутствовать раздел `gd` с дополнительной информацией об установленной версии GD и активных возможностях. Также можно проверить возвращаемое значение `function_exists('imagecreate')`. Если функция возвращает `true`, значит, библиотека GD установлена. Функция `imagetypes()` возвращает битовое поле с информацией о доступных графических форматах. За дополнительной информацией об использовании этой функции рассказано на сайте PHP. Если вам потребуется одна из отключенных возможностей, постройте PHP самостоятельно или обратитесь за помощью к администратору.

Базовый процесс построения изображения состоит из трех шагов: создания изображения, добавления на холст графических элементов и текста и отображения или сохранения. Пример:

```
$image = ImageCreateTrueColor(200, 50); // По умолчанию черный

// Выбрать серый цвет фона
$grey = 0xCCCCCC;
ImageFilledRectangle($image, 0, 0, 200 - 1, 50 - 1, $grey);

// Нарисовать белый прямоугольник
$white = 0xFFFFFF;
ImageFilledRectangle($image, 50, 10, 150, 40, $white);

// Отправка в формате PNG
```

```
header('Content-type: image/png');
ImagePNG($image);
ImageDestroy($image);
```

Результат выполнения этого кода (белый прямоугольник на сером фоне) изображен на рис. 17.1.

Рис. 17.1. Белый прямоугольник на сером фоне

Все начинается с создания *холста* (canvas). Функция `ImageCreateTrueColor()` не возвращает реальное изображение, а лишь предоставляет дескриптор; сама графика появится лишь после того, как вы прикажете PHP вывести изображение. При помощи функции `ImageCreateTrueColor()` можно «жонглировать» несколькими изображениями одновременно.

В параметрах `ImageCreateTrueColor()` передается ширина и высота графики в пикселах. В данном случае размеры составляют 200 пикселов в ширину и 50 пикселов в высоту. По умолчанию для вновь созданных холстов используется черный цвет фона.

Кроме создания новых изображений вы также можете редактировать существующие изображения. Чтобы открыть графический файл, вызовите `ImageCreateFromPNG()` или функцию с аналогичным именем (`ImageCreateFromGIF()`, `ImageCreateFromJPEG()`, `ImageCreateFromWBMP()`, ...) для открытия файла в другом формате. Единственный аргумент содержит имя файла, который может храниться как локально, так и на удаленном сервере:

```
// Открыть файл PNG на локальной машине
$graph = ImageCreateFromPNG('/path/to/graph.png');
// Открыть файл JPEG на удаленном сервере
$icon  = ImageCreateFromJPEG('http://www.example.com/images/icon.jpeg');
```

Вызов функции `ImageFilledRectangle()` размещает на холсте прямоугольник. `ImageFilledRectangle()` получает несколько параметров: изображение, на котором рисуется прямоугольник, координаты x и y левого верхнего угла прямоугольника, координаты x и y правого нижнего угла, и наконец, цвет контура фигуры.

Цвет задается числом, представляющим компоненты RGB (аналогично тому, как это делается с HTML и CSS). Например, шестнадцатеричный код HTML белого цвета равен `#FFFFFF`. В PHP это значение записывается в шестнадцатеричном формате `0xFFFFFF` (или 16777215 в десятичной записи).

Другой вариант — использование функции `ImageAllocate()`, которой передается объект холста и цветовые составляющие (красная, зеленая и синяя):

```
$color = ImageAllocate($image, $r, $g, $b);

// Например, белый
```

```
$white = ImageAllocate($image, 0xFF, 0xFF, 0xFF); // Шестнадцатеричная
$white = ImageAllocate($image,  255,  255,  255); // Десятичная

// Или...
$grey = ImageColorAllocate($image, 204, 204, 204);
$orange = ImageColorAllocate($image, 0xE9, 0x52, 0x22);
```

Чтобы нарисовать прямоугольник поверх черного цвета фона, используемого по умолчанию, выберите цвет и разместите заполненный прямоугольник на холсте. Так как начало координат находится в точке (0,0), используйте эти значения для первой пары координат *x* и *y*. Уменьшите на 1 ширину и высоту во второй паре координат:

```
// Серый цвет фона
$grey = 0xCCCCCC;
ImageFilledRectangle($image, 0, 0, 200 - 1, 50 - 1, $grey);
```

После этого можно переходить к рисованию других элементов. Например, следующий фрагмент рисует на холсте `$image` прямоугольник от точки (50,10) до точки (150,40), заполненный белым цветом:

```
// Нарисовать белый прямоугольник
$white = 0xFFFFFF;
ImageFilledRectangle($image, 50, 10, 150, 40, $white);
```

Возможно, это кому-то покажется неожиданным, но точка с координатами (0,0) находится в левом верхнем углу. Таким образом, при перемещении вниз по холсту координаты увеличиваются. Например, на холсте высотой 50 пикселов вертикальная координата точки, смещенной на 10 пикселов от верхнего края, равна 10 (а не 40 и не –10).

Изображение полностью готово, и его можно передать пользователю. Начните с передачи заголовка `Content-Type`, чтобы браузер знал тип передаваемого изображения. В данном примере используется изображение PNG. Затем PHP передает изображение PNG функцией `ImagePNG()`:

```
header('Content-type: image/png');
ImagePNG($image);
```

Чтобы записать изображение на диск (вместо того, чтобы отправлять его браузеру), передайте `ImagePNG()` второй аргумент с местонахождением файла:

```
ImagePNG($image, '/path/to/your/new/image.png');
```

Поскольку файл не будет передаваться браузеру, вызывать `header()` не нужно. Укажите путь и имя файла и проследите за тем, чтобы у PHP были необходимые разрешения для записи в выбранное место.

PHP автоматически уничтожает изображение при завершении сценария, но если вам потребуется вручную освободить занимаемую память, вызовите функцию `ImageDestroy($image)`, и PHP немедленно уничтожит его:

```
ImageDestroy($image);
```

17.1. Рисование линий, прямоугольников и многоугольников

Задача

Требуется нарисовать линию, прямоугольник или многоугольник — предположим, вы собираетесь построить гистограмму или график биржевых котировок. Возможно, вам также нужно управлять замыканием контура или заливкой.

Решение

Чтобы нарисовать линию, вызовите функцию `ImageLine()`:

```
$width = 200;
$height = 50;
$image = ImageCreateTrueColor($width, $height);

$background_color = 0xFFFFFF; // Белый
ImageFilledRectangle($image, 0, 0, $width - 1, $height - 1, $background_color);

$x1 = $y1 = 0 ; // 0
$x2 = $y2 = $height - 1; // 49
$color = 0xCCCCCC; // Серый

ImageLine($image, $x1, $y1, $x2, $y2, $color);

header('Content-type: image/png');
ImagePNG($image);
ImageDestroy($image);
```

Чтобы нарисовать незаполненный прямоугольник, используйте функцию `ImageRectangle()`:

```
ImageRectangle($image, $x1, $y1, $x2, $y2, $color);
```

Чтобы нарисовать заполненный прямоугольник, используйте функцию `ImageFilledRectangle()`:

```
ImageFilledRectangle($image, $x1, $y1, $x2, $y2, $color);
```

Чтобы нарисовать незаполненный многоугольник, используйте функцию `ImagePolygon()`:

```
$points = array($x1, $y1, $x2, $y2, $x3, $y3);
ImagePolygon($image, $points, count($points)/2, $color);
```

Чтобы нарисовать заполненный многоугольник, используйте функцию `ImageFilledPolygon()`:

```
$points = array($x1, $y1, $x2, $y2, $x3, $y3);
ImageFilledPolygon($image, $points, count($points)/2, $color);
```

Комментарий

Прототипы всех пяти функций, приведенных в Решении, похожи. В первом параметре передается холст, на котором будет осуществляться рисование. В следующей группе параметров передаются координаты, определяющие местонахождение фигуры. Для функции `ImageLine()` четыре координаты определяют конечные точки линии, а для функции `ImageRectangle()` они определяют противоположные углы прямоугольника. Например, вызов `ImageLine($image, 0, 0, 100, 100, $color)` рисует диагональную линию. При передаче тех же параметров `ImageRectangle()` будет нарисован прямоугольник с углами в точках (0,0), (100,0), (0,100) и (100,100). Обе фигуры изображены на рис. 17.2.

Рис. 17.2. Диагональная линия и квадрат

Функция `ImagePolygon()` устроена немного иначе, потому что ей может передаваться переменное количество вершин. Во втором параметре передается массив координат x и y. Функция начинает с первой пары точек и рисует отрезки от вершины к вершине, после чего завершает контур фигуры, возвращаясь к исходной точке. Многоугольник должен содержать не менее трех точек (шесть элементов в массиве). Третий параметр определяет количество вершин в фигуре; так как оно всегда равно половине количества элементов в массиве точек, можно использовать значение `count($points)/2`, потому что оно позволяет обновить массив вершин без прерывания вызова `ImageLine()`.

Например, следующий фрагмент рисует прямоугольный треугольник на рис. 17.3.

```
$size = 50;
$image = ImageCreateTrueColor($size, $size);

$background_color = 0xFFFFFF // Белый
ImageFilledRectangle($image, 0, 0, $size - 1, $size - 1, $background_color);

// Три точки прямоугольного треугольника
$x1 = $y1 = 0 ;           // ( 0, 0)
$x2 = $y2 = $size - 1;    // (49,49)
$x3 = 0; $y3 = $size - 1; // ( 0,49)

$gray = 0xCCCCCC; // Серый

$points = array($x1, $y1, $x2, $y2, $x3, $y3);
ImagePolygon($image, $points, count($points)/2, $gray);

header('Content-type: image/png');
ImagePNG($image);
ImageDestroy($image);
```

Рис. 17.3. Прямоугольный треугольник

Наконец, все функции получают последний параметр, определяющий цвет линии. Обычно передается цветовое значение (например, `0xCCCCCC`), но также могут использоваться константы `IMG_COLOR_STYLED` или `IMG_COLOR_STYLEDBRUSHED`, если вы хотите рисовать прерывистые линии (см. Рецепт 17.3).

Все эти функции рисуют незаполненные фигуры. Чтобы нарисованная фигура автоматически заполнялась цветом рисования, используйте функции `ImageFilledRectangle()` и `ImageFilledPolygon()` с такими же наборами аргументов, как и у их «родственников» без заполнения.

См. также

Рецепт 17.2 — рисование других типов фигур; Рецепт 17.3 — рисование кистью и с применением стилей; документация по функциям `ImageLine()`, `ImageRectangle()` и `ImagePolygon()`.

17.2. Рисование дуг, эллипсов и кругов

Задача

Требуется нарисовать заполненную или незаполненную криволинейную фигуру, например круговую диаграмму с результатами опроса.

Решение

Чтобы нарисовать дугу, используйте функцию `ImageArc()`:

```
ImageArc($image, $x, $y, $width, $height, $start, $end, $color);
```

Чтобы нарисовать эллипс, используйте функцию `ImageEllipse()`:

```
ImageEllipse($image, $x, $y, $width, $height, $color);
```

Чтобы нарисовать круг, используйте функцию `ImageEllipse()` с одинаковыми значениями `$width` и `$height`:

```
ImageEllipse($image, $x, $y, $diameter, $diameter, $color);
```

Комментарий

Исключительная гибкость функции `ImageArc()` позволяет строить разнообразные кривые. Как и многие функции GD, в первом параметре эта функция получает

объект холста. Следующие два параметра определяют координаты *x* и *y* центральной позиции дуги. За ними следуют параметры ширины и высоты дуги.

В шестом и седьмом параметрах передаются начальный и конечный угол (в градусах). Значение 0 соответствует «трем часам» на циферблате. Дальнейший отсчет ведется по часовой стрелке: 90 — шесть часов, 180 — девять часов и т. д. (Будьте внимательны — это поведение характерно не для всех функций GD. Например, поворот текста осуществляется по направлению против часовой стрелки.) Так как в точке `($x,$y)` находится центр дуги, полукруг от 0 до 180 градусов начинается не в точке `($x,$y)`, а в точке `($x+($diameter/2),$y)`. Как обычно, последний параметр определяет цвет дуги.

Функция `ImageEllipse()` аналогична `ImageArc()`, но начальный и конечный угол для нее не указываются — для них используются фиксированные значения 0 и 360. Так как круг представляет собой эллипс, ширина которого равна высоте, для рисования круга передайте в обоих параметрах его диаметр.

Например, следующий фрагмент рисует незаполненный черный круг с диаметром 100 пикселов, расположенный по центру холста (слева на рис. 17.4):

```
$size = 100;
$image = ImageCreateTrueColor($size, $size);
$background_color = 0xFFFFFF; // white
ImageFilledRectangle($image, 0, 0, $size - 1, $size - 1, $background_color);
$black = 0x000000;
ImageEllipse($image, $size / 2, $size / 2, $size - 1, $size - 1, $black);
```

Чтобы нарисовать заполненный эллипс или круг, вызовите функцию `ImageFilledEllipse()`:

```
ImageFilledEllipse($image, $size / 2, $size / 2, $size - 1, $size - 1, $black);
```

Результат изображен в правой части рис. 17.4.

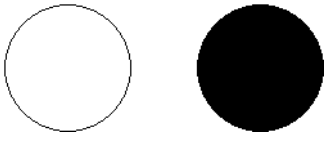

Рис. 17.4. Незаполненный черный круг и заполненный черный круг

Функция `ImageFilledArc()` получает дополнительный пятый параметр, описывающий стиль рисования и заполнения дуги. Константа `IMG_ARC_CHORD` соединяет начальную и конечную точки прямой линией, а константа `IMG_ARC_PIE` рисует только саму дугу. Чтобы отказаться от заполнения дуги, используйте константу `IMG_ARC_NOFILL`. Объединение констант поразрядной операцией ИЛИ позволяет строить секторы и другие интересные фигуры. Пример:

```
$styles = [IMG_ARC_PIE,
           IMG_ARC_CHORD,
```

```
            IMG_ARC_PIE | IMG_ARC_NOFILL,
            IMG_ARC_PIE | IMG_ARC_NOFILL | IMG_ARC_EDGED];

$size = 100;
$image = ImageCreateTrueColor($size * count($styles), $size);

$background_color = 0xFFFFFF; // Белый
ImageFilledRectangle($image, 0, 0,
    $size * count($styles) - 1, $size * count($styles) - 1, $background_color);
$black = 0x000000; // То есть 0

for ($i = 0; $i < count($styles); $i++) {
    ImageFilledArc($image, $size / 2 + $i * $size, $size / 2,
        $size - 1, $size - 1, 0, 135, $black, $styles[$i]);
}

header('Content-type: image/png');
ImagePNG($image);
ImageDestroy($image);
```

Фрагмент рисует фигуры, изображенные на рис. 17.5.

Рис. 17.5. Разнообразные секторы

См. также

Рецепт 17.2 — дополнительная информация о рисовании других видов фигур; документация по функциям `ImageArc()`, `ImageFilledArc()`, `ImageEllipse()` и `ImageFilledEllipse()`.

17.3. Рисование узорных линий

Задача

Требуется рисовать фигуры с использованием узорных линий (вместо используемых по умолчанию сплошных линий).

Решение

Чтобы при рисовании фигур использовались узорные линии, вызовите функцию `ImageSetStyle()`, а затем при рисовании передайте цвет изображения `IMG_COLOR_STYLED`:

```
// Черно-белая пунктирная линия с отрезками по два пиксела
$black = 0x000000;
$white = 0xFFFFFF;
```

```
$style = array($black, $black, $white, $white);
ImageSetStyle($image, $style);

ImageLine($image, 0, 0, 50, 50, IMG_COLOR_STYLED);
ImageFilledRectangle($image, 50, 50, 100, 100, IMG_COLOR_STYLED);
```

Комментарий

Узор линии определяется массивом цветов; элементы массива представляют пикселы кисти. Часто цвет повторяется в нескольких последовательных элементах для увеличения размера полос в узоре.

Например, следующий код рисует квадрат с чередованием белых и черных пикселов (слева на рис. 17.6):

```
// Черно-белая пунктирная линия с отрезками по два пиксела
$style = array($white, $black);
ImageSetStyle($image, $style);
ImageFilledRectangle($image, 0, 0, 49, 49, IMG_COLOR_STYLED);
```

Рис. 17.6. Два квадрата с чередованием белых и черных пикселов

Тот же самый квадрат с узором из пяти белых пикселов, за которыми следуют пять черных пикселов, изображен в правой части рис. 17.6:

```
// Черно-белая пунктирная линия толщиной 5 пикселов
$style = array($white, $white, $white, $white, $white,
               $black, $black, $black, $black, $black);
ImageSetStyle($image, $style);
ImageFilledRectangle($image, 0, 0, 49, 49, IMG_COLOR_STYLED);
```

Узоры выглядят совершенно по-разному, хотя оба стиля представляют собой последовательность белых и черных пикселов.

См. также

Рецепты 17.1 и 17.2 — дополнительная информация о рисовании фигур; документация по функции `ImageSetStyle()`.

17.4. Вывод текста

Задача

Требуется вывести текст как графическое изображение (например, для создания динамической кнопки или счетчика посещений).

Решение

Для встроенных шрифтов GD используется функция `ImageString()`:

```
ImageString($image, 1, $x, $y, 'I love PHP Cookbook', $text_color);
```

Для шрифтов TrueType используется функция `ImageFTText()`:

```
ImageFTText($image, $size, 0, $x, $y, $text_color, '/path/to/font.ttf',
            'I love PHP Cookbook');
```

Комментарий

Чтобы разместить текст на холсте, вызовите функцию `ImageString()`. Как и другие функции графического вывода GD, `ImageString()` получает несколько параметров: объект холста для рисования, номер шрифта, координаты x и y левого верхнего угла текста, выводимый текст и, наконец, цвет символов.

Функция `ImageString()` поддерживает пять вариантов размера шрифта, от 1 до 5. Номер 1 соответствует самому мелкому шрифту, а номер 5 — самому крупному (рис. 17.7). Для любых значений за пределами этого диапазона генерируется размер, эквивалентный ближайшему допустимому значению.

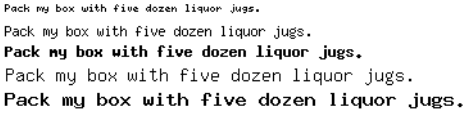

Рис. 17.7. Размеры встроенных шрифтов GD

Чтобы текст выводился по вертикали, а не по горизонтали, воспользуйтесь функцией `ImageStringUp()`. Результат показан на рис. 17.8:

```
ImageStringUp($image, 1, $x, $y, 'I love PHP Cookbook', $text_color);
```

Рис. 17.8. Вертикальный текст

Чтобы использовать шрифты TrueType, необходимо также установить библиотеку FreeType и включить поддержку FreeType в конфигурации PHP. Для включения FreeType 2.x используйте параметр `--with-freetype-dir=DIR`.

Как и `ImageString()`, функция `ImageFTText()` выводит строку на холсте, но получает несколько отличающиеся параметры, которые передаются в другом порядке:

```
$image = ImageCreateTrueColor(200, 50);
ImageFilledRectangle($image, 0, 0, 199, 49, 0xFFFFFF); // Белый

$size = 20;
$angle = 0;
$x = 20;
$y = 35;
$text_color = 0x000000; // Черный
$text = 'Hello PHP!';
$fontpath =  __DIR__ . '/stocky/stocky.ttf';

ImageFTText($image, $size, $angle, $x, $y, $text_color, $fontpath,
            $text);

header('Content-type: image/png');
ImagePNG($image);
```

Аргумент `$size` определяет размер шрифта в пикселах; `$angle` — угол поворота в градусах (по направлению против часовой стрелки). Аргумент `$fontpath` определяет полный путь к файлу шрифта TrueType.

В отличие от `ImageString()`, координаты `($x,$y)` определяют положение левого нижнего угла базовой линии первого символа. (Базовая линия определяет положение нижнего края большинства символов; такие символы, как «g» и «j», опускаются ниже базовой линии, а символы «a» и «z» располагаются на базовой линии.)

Текст, выводимый этим фрагментом, изображен на рис. 17.9.

Рис. 17.9. Текст, оформленный шрифтом TrueType

См. также

Рецепт 17.5 — выравнивание текста по центру; документация по функциям `ImageString()`, `ImageStringUp()` и `ImageFTText()`.

17.5. Выравнивание текста по центру

Задача

Требуется вывести текст в центре изображения.

Решение

Определите размеры изображения и граничного прямоугольника текста. Используя эти координаты, вычислите координаты базовой точки для вывода текста.

Для шрифтов TrueType используется функция `ImageFTCenter()`:

```
function ImageFTCenter($image, $size, $angle, $font, $text, $extrainfo =
                       array()) {
    // Определение размера изображения
    $xi = ImageSX($image);
    $yi = ImageSY($image);

    // Определение размера текста
    $box = ImageFTBBox($size, $angle, $font, $text, $extrainfo);

    $xr = abs(max($box[2], $box[4]));
    $yr = abs(max($box[5], $box[7]));

    // Вычисление центральной точки
    $x = intval(($xi - $xr) / 2);
    $y = intval(($yi + $yr) / 2);

    return array($x, $y);
}
```

Пример:

```
list($x, $y) = ImageFTCenter($image, $size, $angle, $font, $text);
ImageFTText($image, $size, $angle, $x, $y, $fore, $font, $text);
```

Для встроенных шрифтов GD следует использовать функцию `ImageStringCenter()`:

```
function ImageStringCenter($image, $text, $font) {

    // Размеры шрифтов
    $width  = array(1 => 5, 6, 7, 8, 9);
    $height = array(1 => 6, 8, 13, 15, 15);

    // Определение размера изображения
    $xi = ImageSX($image);
    $yi = ImageSY($image);

    // Определение размера текста
    $xr = $width[$font] * strlen($text);
    $yr = $height[$font];

    // Вычисление центральной точки
    $x = intval(($xi - $xr) / 2);
    $y = intval(($yi - $yr) / 2);
    return array($x, $y);
}
```

Пример:

```
list($x, $y) = ImageStringCenter($image, $text, $font);
ImageString($image, $font, $x, $y, $text, $fore);
```

Комментарий

Две функции, приведенные в Решении, возвращают координаты *x* и *y* для вывода текста. В зависимости от типа шрифта, размера и настроек используются разные способы вычисления координат.

Для шрифтов TrueType функции `ImageFTCenter()` передается изображение, созданное вызовом `ImageCreateTrueColor()` (или других функций того же семейства), и параметры, определяющие параметры вывода. Четыре параметра — размер шрифта, угол, выводимый текст и шрифт — передаются всегда. Последний параметр — массив дополнительной информации, который может передаваться `ImageFTBBox()`, — не обязателен.

Функции `ImageSX()` и `ImageSY()` используются для определения размеров холста; они возвращают ширину и высоту графики. Затем вызывается функция `ImageFTBBox()`, которая возвращает восемь чисел: координаты (*x*, *y*) четырех углов текста, от левого нижнего с перемещением против часовой стрелки; вторая пара координат для правого нижнего угла и т. д. Так как координаты задаются относительно базовой линии текста, обычно они отличны от 0. Например, буква «g» в нижнем регистре опускается ниже большинства букв, поэтому в этом случае координата *y* для левого нижнего угла отрицательна.

Располагая этими значениями, можно вычислить правильные параметры для выравнивания по центру. Поскольку координаты левого верхнего угла холста равны (0,0), а функции `ImageFTText()` нужен левый нижний угол, формулы вычисления `$x` и `$y` различаются. Для `$x` вычисляется разность между размерами холста и текста; она определяет величину полей, окружающих текст. Интервал делится на два для определения количества пикселов, резервируемых слева от текста. Для `$y` делается то же самое, но с суммированием `$yi` и `$yr`. Сложение этих чисел дает координату дальней стороны прямоугольника — именно то, что нужно, из-за инвертирования координаты *y* в GD.

Координаты левого нижнего угла в этих вычислениях намеренно игнорируются. Так как основная часть текста располагается выше базовой линии, прибавление пикселов нижних выносных элементов только ухудшает код, так как оно нарушает визуальную центровку текста.

Выравнивание текста по центру осуществляется следующим образом:

```
list($x, $y) = ImageFTCenter($image, $size, $angle, $font, $text);
ImageFTText($image, $size, $angle, $x, $y, $color, $font, $text);
```

К сожалению, этот пример не подходит для встроенных шрифтов GD. Не существует функции для получения размера строки, выводимой встроенным шрифтом. Тем не менее с некоторыми изменениями можно адаптировать приведенный код.

Встроенные шрифты имеют фиксированную ширину, что позволяет легко измерить размер символа и создать функцию, возвращающую размер текста на основании его длины. Данные в таблице 17.1 не являются стопроцентно точными, но получаемые возвращаемые результаты точны до 1 или 2 пикселов; в большинстве случаев этого должно быть достаточно.

Таблица 17.1. Размеры символов встроенных шрифтов GD

Номер шрифта	Ширина	Высота
1	5	6
2	6	8
3	7	13
4	8	15
5	9	15

В функции `ImageStringCenter()` ширина строки вычисляется умножением длины строки на ширину символа; высота равна одному символу. Помните, что `ImageString()` получает координату *y* верхней границы текста, поэтому при вычислении `$y` следует вернуться к знаку «минус».

Примеры горизонтального выравнивания текста по центру с использованием всех пяти шрифтов:

```
$w = 400; $h = 75;
$image = ImageCreateTrueColor($w, $h);
ImageFilledRectangle($image, 0, 0, $w-1, $h-1, 0xFFFFFF);

$color = 0x000000; // Черный
$text = 'Pack my box with five dozen liquor jugs.';

for ($font = 1, $y = 5; $font <= 5; $font++, $y += 20) {
    list($x) = ImageStringCenter($image, $text, $font);
    ImageString($image, $font, $x, $y, $text, $color);
}
```

Результат показан на рис. 17.10.

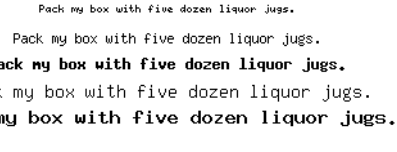

Рис. 17.10. Выравнивание по центру для встроенных шрифтов GD

См. также

Рецепт 17.4 — вывод текста; документация по функциям `ImageSX()`, `ImageSY()` и `ImageFTBBox()`.

17.6. Построение динамических изображений

Задача

Требуется создать изображение по существующему шаблону и динамическим данным (обычно текстовым), например сформировать счетчик посещений.

Решение

Загрузите шаблон, определите позицию для правильного размещения текста (например, для его выравнивания по центру), добавьте текст на холст и передайте изображение браузеру:

```
include 'imageftcenter.php';

// Параметры конфигурации
$image    = ImageCreateFromPNG('/path/to/button.png'); // Графический шаблон
$size     = 24;
$angle    = 0;
$color    = 0x000000;
$fontfile = '/path/to/font.ttf'; // Измените
$text     = $_GET['text']; // Или любой другой источник

// Вывод текста по центру
list($x, $y) = ImageFTCenter($image, $size, $angle, $fontfile, $text);
ImageFTText($image, $size, $angle, $x, $y, $color, $fontfile, $text);

// Сохранение прозрачности
ImageColorTransparent($image,
    ImageColorAllocateAlpha($image, 0, 0, 0, 127));
ImageAlphaBlending($image, false);
ImageSaveAlpha($image, true);

// Отправка изображения
header('Content-type: image/png');
ImagePNG($image);

// Освобождение ресурсов
ImagePSFreeFont($font);
ImageDestroy($image);
```

Комментарий

Построить динамическое изображение средствами GD несложно; для этого достаточно объединить несколько рецептов. В начале кода Решения из готового шаблона кнопки загружается изображение; оно создает фон, на который накладывается текст. В приведенном коде текст берется из строки запроса. Также строка может читаться из базы данных (для счетчика посещений) или загружаться с удаленного сервера (биржевые котировки, значки прогнозов погоды).

После этого выполняются другие подготовительные действия: загрузка шрифта с указанием его размера, цвета и фона. Перед выводом текста необходимо определить его позицию; функция `ImageFTCenter()` из Рецепта 17.5 хорошо справляется с этой задачей. Остается вернуть изображение и выгрузить данные из памяти.

Например, следующий код генерирует страницу HTML и теги изображений с динамическими кнопками, как показано на рис. 17.11:

```
<?php
if (isset($_GET['button'])) {

    // Параметры конфигурации
    $image    = ImageCreateFromPNG(__DIR__ . '/button.png');
    $text     = $_GET['button']; // Динамически сгенерированный текст
    $font     = '/Library/Fonts/Hei.ttf';
    $size     = 24;
    $color    = 0x000000;
    $angle    = 0;

    // Вывод текста по центру
    list($x, $y) = ImageFTCenter($image, $size, $angle, $font, $text);
    ImageFTText($image, $size, $angle, $x, $y, $color, $font, $text);

    // Сохранение прозрачности
    ImageColorTransparent($image,
        ImageColorAllocateAlpha($image, 0, 0, 0, 127));
    ImageAlphaBlending($image, false);
    ImageSaveAlpha($image, true);

    // Отправка изображения
    header('Content-type: image/png');
    ImagePNG($image);

    // Освобождение ресурсов
    ImagePSFreeFont($font);
    ImageDestroy($image);

} else {
    $url = htmlentities($_SERVER['PHP_SELF']);
?>
<html>
<head>
    <title>Sample Button Page</title>
</head>
<body>
    <img src="<?php echo $url; ?>?button=Previous"
         alt="Previous" width="132" height="46">
    <img src="<?php echo $url; ?>?button=Next"
         alt="Next"     width="132" height="46">
</body>
</html>
<?php
}
?>
```

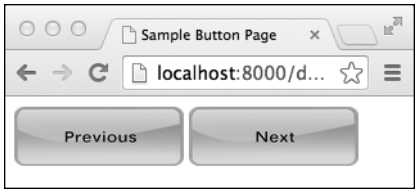

Рис. 17.11. Страница с кнопками

Если в `$_GET['button']` передается значение, сценарий генерирует кнопку и отправляет PNG. Если значение `$_GET['button']` не задано, выводится базовая страница HTML с двумя встроенными вызовами сценария с запросами изображений кнопок — для Previous и для Next. Вероятно, можно создать более общее решение с отдельной страницей button.php, которая возвращает только графику; в качестве источника изображения будет задаваться ссылка на эту страницу.

См. также

Рецепт 17.4 — вывод текста; Рецепт 17.5 — выравнивание текста по центру; глава 9 «Graphics» книги «Programming PHP, Third Edition» Расмуса Лердорфа, Кевина Татро и Питера Макинтайра (издательство O'Reilly).

17.7. Назначение и получение цвета прозрачности

Задача

Требуется назначить один из цветов изображения прозрачным. При наложении изображения на подложку сквозь прозрачные участки изображения просматривается фон.

Решение

Воспользуйтесь функцией `ImageColorTransparent()`:

```
$color = 0xFFFFFF;
ImageColorTransparent($image, $color);
```

Комментарий

Форматы GIF и PNG поддерживают прозрачность; формат JPEG ее не поддерживает. Для ссылок на прозрачный цвет из GD используется константа `IMG_COLOR_TRANSPARENT`. Например, следующий фрагмент рисует пунктирную линию с чередованием черных и прозрачных участков:

```
// Пунктирная линия из черных и прозрачных участков
$style = array($black, $black, IMG_COLOR_TRANSPARENT, IMG_COLOR_TRANSPARENT);
ImageSetStyle($image, $style);
```

Чтобы узнать текущий выбор прозрачного цвета, возьмите возвращаемое значение `ImageColorTransparent()` и передайте его `ImageColorsForIndex()`:

```
$transparent = ImageColorsForIndex($image, ImageColorTransparent($image));
print_r($transparent);
```

Результат:

```
Array
(
    [red] => 255
    [green] => 255
    [blue] => 255
)
```

Функция `ImageColorsForIndex()` возвращает массив с красной, зеленой и синей составляющими. В данном случае в качестве прозрачного используется белый цвет.

См. также

Документация по функциям `ImageColorTransparent()` и `ImageColorsForIndex()`.

17.8. Наложение водяных знаков

Задача

Требуется наложить водяной знак на изображение.

Решение

Если водяной знак имеет прозрачный фон, вызовите функцию `ImageCopy()` для использования альфа-канала:

```
$image = ImageCreateFromPNG('/path/to/image.png');
$stamp = ImageCreateFromPNG('/path/to/stamp.png');

$margin = ['right' => 10, 'bottom' => 10]; // Смещение от края

ImageCopy($image, $stamp,
    imagesx($image) - imagesx($stamp) - $margin['right'],
    imagesy($image) - imagesy($stamp) - $margin['bottom'],
    0, 0, imagesx($stamp), imagesy($stamp));
```

В противном случае вызовите `ImageCopyMerge()` с дополнительным параметром — уровнем непрозрачности:

```
$image = ImageCreateFromPNG('/path/to/image.png');
$stamp = ImageCreateFromPNG('/path/to/stamp.png');

$margin = ['right' => 10, 'bottom' => 10]; // Смещение от края
$opacity = 50; // От 0 до 100%

ImageCopyMerge($image, $stamp,
    imagesx($image) - imagesx($stamp) - $margin['right'],
    imagesy($image) - imagesy($stamp) - $margin['bottom'],
    0, 0, imagesx($stamp), imagesy($stamp),
    $opacity);
```

Комментарий

Водяной знак, наложенный поверх изображения, должен быть ясно видимым, но при этом через него должно просматриваться исходное изображение. Кроме того, чтобы результат лучше смотрелся, водяной знак желательно отодвинуть от края. В приведенном примере используется граница величиной 10 пикселов.

Функции `ImageCopy()` и `ImageCopyMerge()` накладывают одно изображение поверх другого. В нашем случае водяной знак накладывается на базовое изображение.

В первых двух аргументах передаются приемник (изображение) и источник (водяной знак). Далее следуют координаты точки, в которой должен находиться водяной знак. Мы хотим разместить его в правом нижнем углу, поэтому координата x равна ширине изображения-приемника за вычетом ширины водяного знака и правого поля; аналогичная формула используется и для координаты y, но вместо ширины используется высота. Чтобы избежать жесткого фиксирования размеров изображений в сценарии, вычисляйте их динамически при помощи функций `ImageSX()` и `ImageSY()`.

Разобравшись с местонахождением водяного знака, передайте четыре координаты, определяющие размеры копируемой части. В нашем примере передаются значения 0, 0, `ImageSX($stamp)` и `ImageSY($stamp)` для копирования всего изображения, но с таким же успехом можно ограничиться копированием части изображения.

Наконец, функция `ImageCopyMerge()` получает дополнительный аргумент, определяющий уровень непрозрачности копируемого изображения. Значения лежат в диапазоне от 0 до 100; чем меньше число, тем менее заметен водяной знак.

Следующий фрагмент генерирует изображение с водяным знаком, показанное на рис. 17.12:

```
$image = ImageCreateFromJPEG(__DIR__ . '/iguana.jpg');

// Водяной знак
$w = 400; $h = 75;
$stamp = ImageCreateTrueColor($w, $h);
ImageFilledRectangle($stamp, 0, 0, $w-1, $h-1, 0xFFFFFF);

// Сопроводительный текст
```

```
$color = 0x000000; // Черный
ImageString($stamp, 4, 10, 10,
    'Galapagos Land Iguana by Nicolas de Camaret', $color);
ImageString($stamp, 4, 10, 28,
    'http://flic.kr/ndecam/6215259398', $color);
ImageString($stamp, 2, 10, 46,
    'Licence at http://creativecommons.org/licenses/by/2.0.', $color);

// Добавление водяного знака
$margin = ['right' => 10, 'bottom' => 10]; // Смещение от края
$opacity = 50; // between 0 and 100%
ImageCopyMerge($image, $stamp,
    imagesx($image) - imagesx($stamp) - $margin['right'],
    imagesy($image) - imagesy($stamp) - $margin['bottom'],
    0, 0, imagesx($stamp), imagesy($stamp),
    $opacity);

// Отправка
header('Content-type: image/png');
ImagePNG($image);
ImageDestroy($image);
ImageDestroy($stamp);
```

Рис. 17.12. Игуана с водяным знаком

См. также

Документация по `ImageCopy()`, `ImageCopyMerge()`, `ImageSX()` и `ImageSY()`.

17.9. Создание миниатюр

Задача

Требуется создать миниатюры, то есть уменьшенные версии изображений.

Решение

Воспользуйтесь функцией `ImageCopyResampled()` и масштабируйте изображение до нужных размеров.

Пропорциональное уменьшение:

```
$filename = __DIR__ . '/php.png';
$scale = 0.5; // Scale

// Изображения
$image = ImageCreateFromPNG($filename);
$thumbnail = ImageCreateTrueColor(
    ImageSX($image) * $scale,
    ImageSY($image) * $scale);

// Сохранение прозрачности
ImageColorTransparent($thumbnail,
    ImageColorAllocateAlpha($thumbnail, 0, 0, 0, 127));
ImageAlphaBlending($thumbnail, false);
ImageSaveAlpha($thumbnail, true);

// Масштабирование с копированием
ImageCopyResampled($thumbnail, $image, 0, 0, 0, 0,
    ImageSX($thumbnail), ImageSY($thumbnail),
    ImageSX($image), ImageSY($image));

// Отправка
header('Content-type: image/png');
ImagePNG($thumbnail);
ImageDestroy($image);
ImageDestroy($thumbnail);
```

Уменьшение до прямоугольника фиксированного размера:

```
// Прямоугольник

$filename = __DIR__ . '/php.png';
// Размеры миниатюры
$w = 50; $h = 20;

// Изображения
$original = ImageCreateFromPNG($filename);
$thumbnail = ImageCreateTrueColor($w, $h);

// Сохранение прозрачности
ImageColorTransparent($thumbnail,
```

```
    ImageColorAllocateAlpha($thumbnail, 0, 0, 0, 127));
ImageAlphaBlending($thumbnail, false);
ImageSaveAlpha($thumbnail, true);

// Масштабирование с копированием
$x = ImageSX($original);
$y = ImageSY($original);
$scale = min($x / $w, $y / $h);
ImageCopyResampled($thumbnail, $original,
    0, 0, ($x - ($w * $scale)) / 2, ($y - ($h * $scale)) / 2,
    $w, $h, $w * $scale, $h * $scale);

// Отправка
header('Content-type: image/png');
ImagePNG($thumbnail);
ImageDestroy($original);
ImageDestroy($thumbnail);
```

Комментарий

Миниатюры позволяют быстро вывести большое количество изображений в ограниченном пространстве. Сложнее всего подобрать оптимальный алгоритм для масштабирования и/или обрезки, если исходные изображения имеют сильно различающиеся размеры и пропорции.

При пропорциональном уменьшении сохраняется все изображение, однако этот способ хорошо работает только в том случае, если все изображения имеют приблизительно одинаковые размеры и форму. В противном случае получается неуклюжий макет с набором изображений странной формы.

Другой вариант — уменьшение до фиксированного размера. Он упрощает размещение миниатюр, но если пропорции некоторых изображений сильно отличаются от исходных, некоторые изображения получатся очень маленькими. В приведенном рецепте этот недостаток исправляется выделением наибольшего пропорционального прямоугольника из центра изображения.

Например, на рис. 17.13 приведена масштабированная и усеченная версия изображения игуаны с рис. 17.12.

Рис. 17.13. Игуана после масштабирования и усечения

Два алгоритма работают по одному принципу: сначала загружается исходное изображение и создается объект холста для миниатюры; затем миниатюра настраивается для сохранения прозрачности; далее вычисляются параметры, передаваемые функции `ImageCopyResampled()`, выполняющей основную работу по

масштабированию исходного изображения до нового размера и его копированию на холст миниатюры; наконец, миниатюра сохраняется в формате PNG.

При пропорциональном масштабировании размеры холста миниатюры вычисляются умножением исходных размеров на некоторый коэффициент. Функции `ImageSX()` и `ImageSY()` позволяют динамически получить нужные данные:

```
$thumbnail = ImageCreateTrueColor(
    ImageSX($image) * $scale,
    ImageSY($image) * $scale);
```

Когда наступает момент масштабирования, эти две функции снова используются для указания размеров изображений:

```
// Масштабирование и копирование
ImageCopyResampled($thumbnail, $image, 0, 0, 0, 0,
    ImageSX($thumbnail), ImageSY($thumbnail),
    ImageSX($image), ImageSY($image));
```

Функция `ImageCopyResampled()` получает 10 (да, целых 10!) аргументов: в первых двух аргументах передаются два изображения, новое и существующее. В следующих двух передаются координаты x и y для размещения скопированного изображения; в нашем случае заполняется вся миниатюра, поэтому обе координаты всегда равны 0. В пятом и шестом аргументах передаются аналогичные координаты для исходного изображения. Они также равны 0, потому что все изображение уменьшается до миниатюры.

В последних четырех аргументах передаются еще две пары координат. Первая пара определяет ширину и высоту приемного прямоугольника, а вторая — те же размеры исходного прямоугольника. Здесь в обоих случаях используются полные размеры холста.

При масштабировании до постоянного размера холст миниатюры создается очень просто: он имеет выбранную вами ширину и высоту:

```
$thumbnail = ImageCreateTrueColor($w, $h);
```

Впрочем, код масштабирования и копирования изображения получается более сложным:

```
// Масштабирование и копирование
$x = ImageSX($original);
$y = ImageSY($original);
$scale = min($x / $w, $y / $h);
ImageCopyResampled($thumbnail, $original,
    0, 0, ($x - ($w * $scale)) / 2, ($y - ($h * $scale)) / 2,
    $w, $h, $w * $scale, $h * $scale);
```

Сначала вычисляется наименьший возможный пропорциональный прямоугольник, который может поместиться в исходном изображении. Для этого определяется меньшее из двух отношений ширины и высоты для оригинала и миниатюры. Далее размеры выделяемой части исходного прямоугольника вычисляются по формуле `$w * $scale` и `$h * $scale`.

Отсечение можно выполнить и от точки (0,0), но, скорее всего, центральная часть лучше даст представление об изображении в целом. По этой причине смещение в оригинале вычисляется вычитанием половины размера масштабируемого прямоугольника от середины изображения: `$x - ($w * $scale)) / 2 and ($y - ($h * $scale)) / 2`.

Когда вся необходимая информация будет готова, функция `ImageCopyResampled()` выполняет свою работу. Изображение плавно масштабируется до симпатичной картинки меньшего размера.

См. также

Документация по функциям `ImageCopyResampled()`, `ImageSX()` и `ImageSY()`.

17.10. Чтение данных EXIF

Задача

Требуется извлечь метаданные из файла изображения. По этим данным можно узнать, когда была сделана фотография, размер изображения и тип MIME.

Решение

Воспользуйтесь функцией `exif_read_data()`:

```
$exif = exif_read_data('beth-and-seth.jpeg');

print_r($exif);

Array
(
    [FileName] => beth-and-seth.jpg
    [FileDateTime] => 1096055414
    [FileSize] => 182080
    [FileType] => 2
    [MimeType] => image/jpeg
    [SectionsFound] => APP12
    [COMPUTED] => Array
        (
            [html] => width="642" height="855"
            [Height] => 855
            [Width] => 642
            [IsColor] => 1
        )

    [Company] => Ducky
    [Info] =>
)
```

Комментарий

Формат EXIF (Exchangeable Image File Format) — стандарт встраивания метаданных в изображения. Он поддерживается большинством цифровых камер, поэтому этот механизм часто используется для получения расширенных данных в фотогалереях.

PHP поддерживает ряд функций для работы с EXIF. Эти функции не требуют внешних библиотек, но их необходимо явно включить при помощи флага конфигурации `--enable-exif`.

Для чтения данных проще всего воспользоваться методом `exif_read_data()`. Функция возвращает массив с различными метаданными, включая дату создания фотографии, тип MIME (который может использоваться для передачи изображения) и размеры изображения:

```
$exif = exif_read_data('beth-and-seth.jpeg');
```

Используйте значение `html` для прямого встраивания атрибутов высоты и ширины в тег `<img>`.

Также функции EXIF могут использоваться для получения миниатюры, связанной с изображением. Для этого следует вызвать функцию `exif_thumbnail()`:

```
$thumb = exif_thumbnail('beth-and-seth.jpeg', $width, $height, $type);
```

Функция `exif_thumbnail()` получает четыре параметра. В первом передается имя файла; еще три представляют собой передаваемые по ссылке переменные для хранения ширины, высоты и типа изображения. Функция возвращает миниатюрное изображение в формате двоичной строки или `false` в случае ошибки.

Чтобы напрямую передать изображение, используйте функцию `image_type_to_mime_type()` для получения правильного типа MIME. Передайте данные в виде заголовка HTTP, затем выведите изображение:

```
$thumb = exif_thumbnail('beth-and-seth.jpeg', $width, $height, $type);

if ($thumb !== false) {
    $mime = image_type_to_mime_type($type);
    header("Content-type: $mime");
    print $thumb;
} else {
    print "Sorry. No thumbnail.";
}
```

Также можно создать ссылку `<img>`:

```
$file = 'beth-and-seth.jpeg';
$thumb = exif_thumbnail($file, $width, $height, $type);

if ($thumb !== false) {
    $img = "<img src=\"$file\" alt=\"Beth and Seth\"
                width=\"$width\" height=\"$height\">";
    print $img;
}
```

См. также

Документация по функциям `exif_read_data()` и `exif_thumbnail()`.

17.11. Защита изображений

Задача

Требуется управлять доступом пользователей к изображениям.

Решение

Не храните изображения в корневом каталоге документов — сохраните их в другом месте. Чтобы передать файл, откройте его вручную и отправьте браузеру:

```
header('Content-Type: image/png');
readfile('/path/to/graphic.png');
```

Комментарий

Первая строка кода в Решении отправляет браузеру заголовок `Content-Type`, чтобы браузер знал тип поступающего объекта и отображал его соответствующим образом. Вторая строка открывает файл на диске (или по удаленному URL-адресу) для чтения, читает его, направляет в браузер и закрывает файл.

Типичный способ поставки изображений использует тег `<img>` с атрибутом `src`, содержащим ссылку на файл на вашем сайте. Если вы хотите защитить изображения, вероятно, стоит использовать защиту паролем. Два основных механизма — базовая аутентификация HTTP и дайджест-аутентификация — рассматриваются в Рецепте 8.6.

Впрочем, типичный способ не всегда оказывается лучшим. Во-первых, что делать, если вы хотите ограничить состав файлов, которые могут просматриваться пользователями, но осложняя жизнь пользователей вводом имен и паролей? Одно из возможных решений — создавать ссылки только на существующие файлы; если пользователь не может щелкнуть на ссылке, он не сможет просмотреть файл. Впрочем, пользователь может создать закладки для старых файлов или попытаться угадать другие имена файлов на основании вашей схемы назначения имен и ввести URL-адрес вручную в браузере.

Чтобы доступ к контенту был действительно ограничен, пользователи не должны иметь возможность угадать имя и просматривать изображения. Обычно в таких ситуациях ограниченной группе пользователей (чаще всего репортерам) предоставляется версия для предварительного ознакомления, чтобы они могли написать материалы по теме или были готовы распространять их в момент снятия ограничений. Теоретически проблему можно решить, позаботившись о том, что

в корневом каталоге документов хранится только допустимый контент, но для этого потребуются многочисленные «переброски» файлов между каталогами. Лучше действовать иначе — хранить все файлы в одном фиксированном месте и предоставлять только файлы, прошедшие проверку в коде.

Допустим, вы заключили контракт с издательством на распространение одного из его комиксов на вашем сайте. Тем не менее издательство не хочет, чтобы вы создавали виртуальный архив, и поэтому вашим пользователям разрешается просматривать комиксы только за последние две недели, а за остальными комиксами они должны обращаться на официальный сайт. Кроме того, вы получаете комиксы до их официального выпуска, но не хотите позволять другим пользователям бесплатно просматривать их; пользователи должны приходить на ваш сайт ежедневно.

Решение приведено ниже. Поступающие файлы снабжаются временной меткой, поэтому вы можете легко определить, какой файл к какому дню относится. Для блокировки комиксов за пределами 14-дневного окна используется следующий код:

```
// Просмотр комикса разрешен, если его временная метка
// находится в допустимых границах

// Вычисление текущей даты
list($now_m,$now_d,$now_y) = explode(',',date('m,d,Y'));
$now = mktime(0,0,0,$now_m,$now_d,$now_y);

// Двухчасовые допуски с обеих сторон учитывают возможное действие
// летнего времени
$min_ok = $now - 14*86400 - 7200; // 14 дней назад
$max_ok = $now + 7200;            // Сейчас
$mo = (int) $_GET['mo'];
$dy = (int) $_GET['dy'];
$yr = (int) $_GET['yr'];

// Определение временной метки запрашиваемого комикса
$asked_for = mktime(0,0,0,$mo,$dy,$yr);

// Сравнение дат
if (($min_ok > $asked_for) || ($max_ok < $asked_for)) {
    echo 'You are not allowed to view the comic for that day.';
} else {
    header('Content-type: image/png');
    readfile("/www/comics/{$mo}{$dy}{$yr}.png");
}
```

См. также

Рецепт 24.5 — чтение файлов.

17.12. Программа: генерирование гистограммы по результатам опроса

При выводе результатов опроса разноцветная гистограмма часто оказывается нагляднее простого вывода текста. Функция, приведенная в листинге 17.1, использует GD для создания изображения, отображающего накопленные результаты опроса.

Листинг 17.1. Построение гистограммы

```
function bar_chart($question, $answers) {
    // Определение цветов для рисования полос
    $colors = array(0xFF6600, 0x009900, 0x3333CC,
        0xFF0033, 0xFFFF00, 0x66FFFF, 0x9900CC);

    $total = array_sum($answers['votes']);

    // Определение интервалов и других служебных констант
    $padding = 5;
    $line_width = 20;
    $scale = $line_width * 7.5;
    $bar_height = 10;

    $x = $y = $padding;

    // Для рисования создается большое изображение,
    // поскольку длина изображения неизвестна заранее
    $image = ImageCreateTrueColor(150, 500);
    ImageFilledRectangle($image, 0, 0, 149, 499, 0xE0E0E0);
    $black = 0x000000;

    // Вывод вопроса
    $wrapped = explode("\n", wordwrap($question, $line_width));
    foreach ($wrapped as $line) {
        ImageString($image, 3, $x, $y , $line, $black);
        $y += 12;
    }

    $y += $padding;

    // Вывод ответов
    for ($i = 0; $i < count($answers['answer']); $i++) {

        // Форматирование процента
        $percent = sprintf('%1.1f', 100*$answers['votes'][$i]/$total);
        $bar = sprintf('%d', $scale*$answers['votes'][$i]/$total);

        // Определение цвета
        $c = $i % count($colors); // Если полос больше, чем цветов
        $text_color = $colors[$c];

        // Рисование столбца и процента ответов
        ImageFilledRectangle($image, $x, $y, $x + $bar,
```

```
                        $y + $bar_height, $text_color);
        ImageString($image, 3, $x + $bar + $padding, $y,
                    "$percent%", $black);

         $y += 12;

         // Вывод ответа
         $wrapped = explode("\n", wordwrap($answers['answer'][$i], $line_width));
         foreach ($wrapped as $line) {
             ImageString($image, 2, $x, $y, $line, $black);
             $y += 12;
         }

         $y += 7;
     }

     // Изображение обрезается при копировании
     $chart = ImageCreateTrueColor(150, $y);
     ImageCopy($chart, $image, 0, 0, 0, 0, 150, $y);

     // Для PHP 5.5+
     // $chart = ImageCrop($image, array('x' => 0, 'y' => 0,
     //                                  'width' => 150, 'height' => $y));
     // Поставка изображения
     header ('Content-type: image/png');
     ImagePNG($chart);
     // Освобождение ресурсов
     ImageDestroy($image);
     ImageDestroy($chart);
}
```

Чтобы вызвать эту программу, создайте массив, содержащий два параллельных массива: `$answers['answer']` и `$answer['votes']`. Элемент `$i` каждого массива содержит текст ответа и общее количество голосов, поданных за ответ `$i`. На рис. 17.14 представлен примерный результат:

```
// Акт II, сцена II.
$question = 'What a piece of work is man?';

$answers['answer'][] = 'Noble in reason';
$answers['votes'][]  = 29;

$answers['answer'][] = 'Infinite in faculty';
$answers['votes'][]  = 22;

$answers['answer'][] = 'In form, in moving, how express and admirable';
$answers['votes'][]  = 59;

$answers['answer'][] = 'In action how like an angel';
$answers['votes'][]  = 45;

bar_chart($question, $answers);
```

В нашем примере ответы были присвоены вручную, но в реальном опросе информация может извлекаться из базы данных.

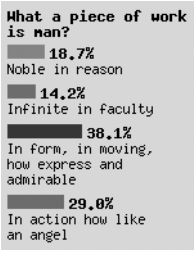

Рис. 17.14. Графическая гистограмма с результатами опроса

Эта программа может стать хорошей отправной точкой, но из-за использования встроенных шрифтов GD в ней задействовано много «волшебных чисел», соответствующих высоте и ширине шрифта. Кроме того, интервалы между ответами тоже жестко фиксируются. Если вы перейдете на более современные шрифты (например, TrueType), алгоритмы управления этими числами придется обновить.

В начале функции определяется набор комбинаций RGB; они используются как цвета для рисования полос. Определяются различные константы — такие как `$line_width`, максимальное количество символов в строке. Переменная `$bar_height` определяет высоту полос, а `$scale` масштабирует длину полосы в зависимости от максимальной длины по всем полосам. Отступ `$padding` смещает результаты на пять пикселов от края холста.

Затем программа создает очень большой холст для рисования гистограммы; позднее холст обрезается до нужного размера, но определить максимальный размер заранее бывает достаточно сложно. В качестве фона гистограммы выбирается цвет #E0E0E0, то есть светло-серый.

Чтобы ограничить ширину гистограммы до разумных пределов, функция `wordwrap()` разбивает текст вопроса `$question` до нужной длины строк, а затем вызывает для него `explode()` по завершителям `\n`. Так создается массив строк правильного размера, элементы которого перебираются в цикле для последовательного вывода строк.

После вывода вопроса можно переходить к ответам. Сначала числа в результатах форматируются функцией `sprintf()`. Для форматирования общего процента голосов для заданного ответа с одним знаком в дробной части используется спецификатор `%1.1f`. Длина полосы, соответствующей результату, вычисляется аналогично, но вместо умножения на сто число умножается на масштабный множитель `$scale`, с получением целого результата.

Цвет полосы извлекается из массива `$colors`, содержащего триплеты RGB. Затем функция `ImageFilledRectangle()` рисует полосу, а функция `ImageString()` выводит процент справа от полосы. После добавления отступов текст ответа выводится по тому же алгоритму, который использовался для вывода вопроса.

После вывода всех ответов общий размер гистограммы сохраняется в `$y`. Теперь изображение можно обрезать до нужного размера, но специальной функции об-

резания не существует. Чтобы обойти это ограничение, создайте новый холст нужного размера и вызовите `ImageCopy()` для той части исходного холста, которую нужно сохранить. Затем изображение правильного размера в формате PNG передается функцией `ImagePNG()`, а память освобождается двумя вызовами `ImageDestroy()`.

Как было сказано в начале раздела, эта функция рисования гистограмм написана «на скорую руку». Она работает и успешно решает некоторые проблемы (например, обеспечивая перенос строк), но идеальной ее не назовешь. Например, возможности ее настройки сильно ограничены — большинство параметров встроены прямо в код. И все же она наглядно демонстрирует, как при помощи функций GD строится вполне практичное графическое приложение.

18 Безопасность и шифрование

18.0. Введение

Безопасность веб-приложений — важная тема, привлекающая внимание как разработчиков, создающих веб-приложения, так и злоумышленников, пытающихся использовать дефекты в этих приложениях. Написанные вами приложения наверняка подвергнутся многочисленным атакам, и вы должны быть к этому готовы.

Многие уязвимости в веб-приложениях обусловлены незаслуженным доверием к данным, предоставленным третьими сторонами. Все входные данные должны считаться потенциально опасными, пока не будет доказано обратное. Отображая непроверенные данные для своих пользователей, вы создаете опасность межсайтовых сценарных атак (XSS). Рецепт 18.4 объясняет, как избежать этой опасности посредством экранирования вывода. Использование потенциально опасных данных в запросах SQL открывает уязвимости внедрения SQL. Рецепт 18.5 показывает, как от них избавиться.

Все данные, полученные от третьих сторон (включая данные, предоставленные пользователями), должны сначала проверяться на действительность. Этот процесс называется *фильтрацией*. Рецепт 18.3 показывает, как обеспечить фильтрацию всех входных данных.

Не все проблемы безопасности могут решаться фильтрацией входных данных и экранированием вывода. Фиксация сеанса — атака, рассматриваемая в Рецепте 18.1, — заставляет жертву использовать идентификатор сеанса, выбранный атакующим. Межсайтовые фальсификации запросов — атаки, рассматриваемые в Рецепте 18.2, — заставляют жертву отправить запрос, выбранный атакующим.

С областью безопасности тесно связано шифрование — мощный инструмент, повышающий безопасность ваших приложений. Однако его, как и любой другой инструмент, следует применять осмотрительно.

Шифрование искажает исходную структуру данных. Некоторые зашифрованные данные не могут быть восстановлены без запредельной вычислительной работы (так называемое *одностороннее шифрование*, или *хеширование*). Другие методы шифрования работают в двух направлениях: данные сначала шифруются, а затем восстанавливаются.

PHP предоставляет в ваше распоряжение инструменты для шифрования и защиты данных. Некоторые из них — такие, как функция `password_hash()`, — входят в базовый набор функций PHP, а другие входят в расширения, которые должны быть явно включены при компиляции PHP (например, mcrypt, mhash и cURL).

В Рецепте 18.7 рассматривается функция `password_hash()`, которая позволяет безопасно хешировать пароль для хранения.

Полнофункциональная библиотека шифрования mcrypt предоставляет различные алгоритмы и режимы шифрования. Благодаря поддержке разных механизмов шифрования библиотека mcrypt особенно удобна для обмена зашифрованными данными с другими системами или с программами, написанными не на PHP. Библиотека mcrypt подробно рассматривается в Рецепте 18.10.

PHP предоставляет средства защиты данных посредством надежного шифрования, однако шифрование является лишь одной из частей большой, часто сложной картины. К зашифрованным данным можно получить доступ при помощи ключа, поэтому защита ключа также играет очень важную роль. Если ключи шифрования будут доступны для посторонних пользователей (например, если они хранятся в файле, к которому можно получить доступ через веб-сервер, или в файле, доступном для других пользователей в среде совместного хостинга), данные окажутся под угрозой независимо от того, насколько безопасен выбранный вами алгоритм шифрования.

Конфиденциальные данные должны защищаться не только на сервере, но и при передаче по сети между сервером и пользователями. Данные, отправленные по обычному каналу HTTP, будут «видны» каждому, кто получит доступ к сети в произвольной точке между сервером и пользователем. В Рецепте 18.13 рассказано, как использовать SSL для предотвращения перехвата и анализа проходящего трафика. За более полным обсуждением темы безопасности приложений PHP обращайтесь к книге «PHP Security» Криса Шифлетта (издательство O'Reilly).

18.1. Предотвращение фиксации сеанса

Задача

Требуется принять меры к тому, чтобы идентификатор сеанса пользователя не мог быть предоставлен третьей стороной, например злоумышленником, который пытается перехватить сеанс пользователя.

Решение

Повторно генерируйте идентификатор сеанса функцией `session_regenerate_id()` при любых изменениях в привилегиях пользователя, например после успешного входа:

```
session_regenerate_id();
$_SESSION['logged_in'] = true;
```

Комментарий

Сеансы позволяют создавать переменные, значения которых сохраняются между запросами. Чтобы система сеансовых данных работала, каждый запрос пользователя должен содержать идентификатор сеанса, однозначно определяющий сеанс.

По умолчанию PHP получает идентификатор сеанса, отправленный в cookie, но если параметр `session.use_only_cookies` равен 1, то идентификатор сеанса будет приниматься в URL. Атакующий может обманным путем заставить жертву перейти на ваше приложение по ссылке со встроенным идентификатором сеанса:

```
<a href="http://example.org/login.php?PHPSESSID=1234">Click Here!</a>
```

Пользователь, переходящий по ссылке, продолжит сеанс с идентификатором 1234. Атакующий теперь знает идентификатор сеанса пользователя, и сможет попытаться перехватить сеанс пользователя, передавая тот же идентификатор.

Если пользователь не вводит свои учетные данные и не выполняет никаких действий, отличающих его от других пользователей приложения, атакующий ничего не выиграет от перехвата сеанса. Следовательно, принимая меры к тому, чтобы идентификатор сеанса генерировался заново при каждой смене уровня привилегий, вы фактически устраняете всякий риск атак фиксации сеанса. PHP берет на себя задачи обновления хранилища сеансовых данных и распространения нового идентификатора сеанса, так что вам остается лишь вызвать единственную функцию в нужный момент.

В PHP 5.5.2 появился новый параметр конфигурации `session.use_strict_mode`, который помогает предотвратить перехват сеанса. Когда этот режим активен, PHP принимает только уже инициализированные идентификаторы сеансов. Если браузер отправляет новый идентификатор сеанса, PHP отвергает его и генерирует новый идентификатор.

См. также

Рецепт 11.2 — параметры конфигурации для предотвращения перехвата и фиксации сеанса; Рецепт 11.3 — схема повторного генерирования идентификаторов сеансов с учетом прошедшего времени.

18.2. Защита от фальсификации форм

Задача

Требуется убедиться в том, что отправка данных формы была выполнена корректно и намеренно.

Решение

Добавьте на форму скрытое поле с одноразовым маркером. Сохраните этот маркер в сеансе пользователя:

```
<?php

session_start();
$_SESSION['token'] = md5(uniqid(mt_rand(), true));

?>

<form action="buy.php" method="POST">
<input type="hidden" name="token" value="<?php echo $_SESSION['token']; ?>" />
<p>Stock Symbol: <input type="text" name="symbol" /></p>
<p>Quantity: <input type="text" name="quantity" /></p>
<p><input type="submit" value="Buy Stocks" /></p>
</form>
```

Когда вы получаете запрос, представляющий отправку данных формы, проверьте маркеры и убедитесь в том, что они совпадают:

```
session_start();

if ((! isset($_SESSION['token'])) ||
    ($_POST['token'] != $_SESSION['token'])) {

    /* Запросить пароль у пользователя. */
} else {
    /* Продолжить. */
}
```

Комментарий

Этот прием помогает защититься от категории атак, называемых межсайтовой фальсификацией запросов (CSRF, Cross-Site Request Forgery). Все атаки этой категории заставляют жертву отправлять запросы на целевой сайт без ее ведома. Как правило, жертва обладает определенным уровнем привилегий на целевом сайте, и эти атаки позволяют атакующему выполнять действия, которые для него обычно недоступны. Представьте, что Алиса вошла на сайт социальной сети с использованием cookie, после чего переходит на другой сайт. Этот сайт может отобразить форму, которая выглядит безобидно, но в действительности отправляет

свои данные на URL-адрес сайта социальной сети. Так как браузер Алисы при отправке формы передает cookie входа, сайт социальной сети без защиты от CSRF не сможет отличить злонамеренную отправку данных формы от корректной.

Добавление маркера на форму не предотвращает возможную фальсификацию своего же запроса самим пользователем, но ее вам все равно не предотвратить, да и вообще о ней не стоит беспокоиться. Фильтрация входных данных, описанная в Рецепте 18.3, заставит запросы подчиняться вашим правилам. Прием, показанный в этом Рецепте, всего лишь поможет убедиться в том, что запрос был выдан сознательно.

18.3. Обеспечение фильтрации входных данных

Задача

Требуется обеспечить фильтрацию всех входных данных перед использованием.

Решение

Инициализируйте пустой массив для хранения фильтрованных данных. Проверив некоторые данные на действительность, сохраните их в массиве:

```
$filters = array('name' => array('filter' => FILTER_VALIDATE_REGEXP,
                                 'options' => array('regexp' => '/^[a-z]+$/i')),
                 'age' => array('filter' => FILTER_VALIDATE_INT,
                                'options' => array('min_range' => 13)));
$clean = filter_input_array(INPUT_POST, $filters);
```

Комментарий

Жесткие правила назначения имен помогут следить за тем, какие входные данные были отфильтрованы. Инициализация `$clean` пустым массивом предотвращает возможность внедрения данных в массив; чтобы данные появились в нем, они должны быть добавлены явно. В приведенном коде вызов `filter_input_array()` инициализирует `$clean` так, чтобы массив мог содержать только фильтрованную информацию.

Когда вы установите такие правила, как использование `$clean`, очень важно, чтобы в вашей бизнес-логике использовались только данные из этого массива.

См. также

Рецепты 9.2–9.9 — подробное обсуждение проверки данных форм для разных типов данных.

18.4. Предотвращение межсайтовых сценарных атак

Задача

Требуется надежно защитить приложения PHP от межсайтовых сценарных атак (XSS, Cross-Site Scripting).

Решение

Экранируйте весь вывод HTML функцией `htmlentities()`, указав правильную кодировку символов:

```
/* Обратите внимание на кодировку. */
header('Content-Type: text/html; charset=UTF-8');

/* Инициализация массива для экранированных данных. */
$html = array();

/* Экранирование фильрованных данных. */
$html['username'] = htmlentities($clean['username'], ENT_QUOTES, 'UTF-8');

echo "<p>Welcome back, {$html['username']}.</p>";
```

Комментарий

Функция `htmlentities()` заменяет каждый символ эквивалентной сущностью HTML (если она существует). Например, символ `>` заменяется сущностью `>`. И хотя экранирование вроде бы приводит к модификации данных, его целью является сохранение данных в другом контексте. Когда браузер отображает последовательность `>` как разметку HTML, она отображается на экране в виде символа `>`.

Атаки XSS пытаются использовать ситуацию, при которой данные, предоставленные третьей стороной, включаются в HTML без должного экранирования. Хитроумный атакующий может предоставить код, который при интерпретации браузером будет чрезвычайно опасным для ваших пользователей. С `htmlentities()` вы можете быть уверены в том, что такие сторонние данные будут отображаться на экране, а не интерпретироваться браузером.

См. также

Рецепт 9.10 — предотвращение межсайтовых сценарных атак в контексте отправки данных форм.

18.5. Предотвращение внедрения SQL

Задача

Требуется исключить уязвимости внедрения SQL в приложении PHP.

Решение

Воспользуйтесь библиотекой баз данных, выполняющей необходимое экранирование для базы данных, например PDO:

```
$statement = $db->prepare("INSERT
                           INTO   users (username, password)
                           VALUES (:username, :password)");

$statement->bindParam(':username', $clean['username']);
$statement->bindParam(':password', $clean['password']);

$statement->execute();
```

Комментарий

Использование связанных параметров гарантирует, что данные никогда не попадут в контекст, в котором они будут интерпретироваться специальным образом; соответственно, никакое значение не сможет изменить формат запроса SQL.

См. также

Глава 10 — информация о PDO (особенно Рецепты 10.6 и 10.7); документация по PDO.

18.6. Хранение паролей отдельно от файлов

Задача

Требуется использовать пароль для подключения к базе данных (например). Пароль не должен храниться в файлах PHP на сайте на случай, если посторонний получит доступ к файлам.

Решение

Сохраните пароль в переменной окружения в файле, загружаемом веб-сервером при запуске. Затем просто обратитесь к переменной окружения в своем коде:

```
$db = new PDO($dsn, $_SERVER['DB_USER'], $_SERVER['DB_PASSWORD']);
```

Комментарий

Хотя описанный способ исключает пароли из исходного кода страниц, они становятся доступными в других местах, где их тоже приходится защищать. Что еще важнее, следует убедиться в отсутствии открытых для просмотра страниц, вызывающих `phpinfo()`. Так как вызов `phpinfo()` выводит все переменные окружения, он также выведет все хранящиеся пароли. Кроме того, содержимое `$_SERVER` не должно становиться доступным другими способами, например вызовом функции `print_r()`.

Затем, особенно при использовании совместного хостинга, проследите за тем, чтобы переменные окружения были настроены так, чтобы они были доступны только вашему виртуальному хосту, а не всем пользователям. Для Apache это можно сделать присваиванием переменных в файле отдельно от главного конфигурационного файла:

```
SetEnv  DB_USER     "susannah"
SetEnv  DB_PASSWORD "y23a!t@ce8"
```

В директиве `<VirtualHost>` для сайта в главном конфигурационном файле (httpd.conf) этот отдельный файл включается следующим образом:

```
Include "/usr/local/apache/database-passwords"
```

Убедитесь в том, что отдельный файл, содержащий пароль (например, /usr/local/apache/database-passwords), недоступен для чтения никому, кроме пользователя, управляющего соответствующим виртуальным хостом. В процессе запуска и чтения конфигурационных файлов Apache обычно выполняется с правами root, поэтому он может прочитать включаемый файл. Дочерний процесс, обрабатывающий запросы, обычно выполняется с правами непривилегированного пользователя, так что защищенный файл будет защищен от чтения из вредоносных сценариев.

См. также

Документация по директиве Apache `Include`.

18.7. Хранение паролей

Задача

Требуется организовать хранение паролей пользователей, чтобы они могли войти на ваш сайт.

Решение

При входе или регистрации пользователя хешируйте выбранный пароль средствами bcrypt и сохраните хешированный пароль в базе данных пользователей.

В PHP 5.5 и выше можно воспользоваться встроенной функцией `password_hash()`:

```
/* Инициализация массива для фильтрованных данных. */
$clean = array();

/* Хеширование пароля. */
$hashed_password = password_hash($_POST['password'], PASSWORD_DEFAULT);

/* Разрешены алфавитно-цифровые имена. */
if (ctype_alnum($_POST['username'])) {
    $clean['username'] = $_POST['username'];
} else {
    /* Ошибка */
}

/* Сохранение информации пользователя в базе данных. */
$st = $db->prepare('INSERT
            INTO   users (username, password)
            VALUES (?, ?)');
$st->execute(array($clean['username'], $hashed_password));
```

Затем, когда пользователь введет свои учетные данные, при помощи функции `password_verify()` проверьте, совпадает ли указанный пароль с сохраненным, хешированным значением:

```
/* Инициализация массива для фильтрованных данных. */
$clean = array();

/* Разрешены алфавитно-цифровые имена. */
if (ctype_alnum($_POST['username'])) {
    $clean['username'] = $_POST['username'];
} else {
    /* Ошибка */
}

$stmt = $db->prepare('SELECT password
                      FROM   users
                      WHERE  username = ?');
$stmt->execute(array($clean['username']));
$hashed_password = $stmt->fetchColumn();

if (password_verify($_POST['password'], $hashed_password)) {
    /* Вход выполнен успешно. */
    print "Login OK!";
} else {
    /* Неудачная попытка. */
}
```

Если вы не используете PHP 5.5, но работаете в PHP 5.3.7 и выше, установите библиотеку password_compat (*https://github.com/ircmaxell/password_compat*), чтобы получить доступ к реализациям `password_hash()` и `password_verify()`.

Если вы используете более старую версию PHP, в следующем комментарии описаны некоторые варианты организации безопасного хранения паролей.

Комментарий

Хранение хешированных паролей предотвращает взлом учетных записей пользователей при получении посторонними доступа к базе данных имен и паролей (хотя такие несанкционированные обращения предвещают другие проблемы с безопасностью).

Функции `password_hash()` и `password_verify()` решают две задачи, затрудняющие возможность использования хешированных паролей даже в том случае, если они станут доступны злоумышленнику. Прежде всего, в хешируемое значение включается строка «затравки». Даже в том случае, если два пользователя выберут одинаковые текстовые пароли, хешированные версии их паролей будут различаться. Если злоумышленник вычислит пароль одного пользователя, он не сможет легко получить другой.

Вторая интересная особенность этих функций — использование алгоритма (в настоящее время bcrypt) с регулируемыми затратами. Вычислительная мощь компьютеров злоумышленников постепенно растет, но вы можете легко увеличить затраты (вычислительные) на восстановление текстовых паролей по хешированным.

Из-за трудностей восстановления хешированных паролей такой механизм хранения несколько повышает уровень защиты. С другой стороны, вы не сможете получить доступ к текстовым паролям даже в случае необходимости. Например, если пользователь забудет свой пароль, вы не сможете напомнить ему. Лучшее, что можно сделать в такой ситуации, — сгенерировать новый пароль и сообщить его пользователю. О том, как действовать при потере пароля, рассказано в Рецепте 18.8.

В PHP версий до 5.3.7 достаточно безопасные хеши паролей можно вычислить при помощи встроенной функции `crypt()`:

```
/* Инициализация массива для фильтрованных данных. */
$clean = array();

/* Генерирование затравки. '$2a$' приказывает функции crypt()
 * использовать алгоритм Blowfish, а 08 - выполнить
 * 256 (2^8) циклов хеширования */
$salt = '$2a$08$';
/* Хеши Blowfish имеют длину 22 байта, при этом каждый байт
 * находится в диапазонах 0-9, A-Z, a-z */
for ($i = 0; $i < 22; $i++) {
    $r = mt_rand(0, 61);
    if ($r < 10) {
        $c = ord('0') + $r;
    }
    else if ($r < 36) {
        $c = ord('A') + $r - 10;
    }
    else {
        $c = ord('a') + $r - 36;
    }
    $salt .= chr($c);
```

```
}

$hashed_password = crypt($_POST['password'], $salt);

/* Разрешены алфавитно-цифровые имена. */
if (ctype_alnum($_POST['username'])) {
    $clean['username'] = $_POST['username'];
} else {
    /* Ошибка */
}

/* Сохранение информации пользователя в базе данных. */
$st = $db->prepare('INSERT
            INTO   users (username, password)
            VALUES (?, ?)');
$st->execute(array($clean['username'], $hashed_password));
```

Затем происходит проверка паролей: программа читает сохраненную «затравку» и передает ее функции `crypt()` вместе с паролем, введенным пользователем:

```
/* Инициализация массива для фильтрованных данных. */
$clean = array();

/* Разрешены алфавитно-цифровые имена. */
if (ctype_alnum($_POST['username'])) {
    $clean['username'] = $_POST['username'];
} else {
    /* Ошибка */
}

$stmt = $db->prepare('SELECT password
                      FROM   users
                      WHERE  username = ?');
$stmt->execute(array($clean['username']));
$hashed_password = $stmt->fetchColumn();
$salt = substr($hashed_password, 0, strlen('$2a$08$') + 22);

if (crypt($_POST['password'], $salt) === $hashed_password) {
    /* Вход выполнен успешно. */
    print "Login OK!";
} else {
    /* Неудачная попытка. */
}
```

При использовании `crypt()` необходимо получить соответствующую затравку (и префикс `$2a$08$`, который сообщает `crypt()` об использовании Blowfish) из сохраненных данных, чтобы передать ее функции `crypt()` с паролем, введенным пользователем. Тем самым гарантируется, что перехешированное значение совпадет в случае совпадения паролей.

Если вы используете версию PHP ранее 5.3.0 и библиотека вашей системы, используемая `crypt()`, не включает поддержку Blowfish, то приведенный код работать не будет. Чтобы проверить поддержку Blowfish, проверьте значение константы `CRYPT_BLOWFISH`. Если она равна 0, значит, Blowfish не поддерживается.

В этом случае можно либо перейти на PHP версии 5.3 (включающей собственную реализацию Blowfish), либо воспользоваться другой функцией хеширования пароля, например `sha1()`. Если вы хотите обеспечить максимальную безопасность паролей, лучше обновить версию PHP. Алгоритм SHA1 работает намного быстрее, чем Blowfish, поэтому нападающему будет проще подобрать текстовый пароль, подходящий для заданного хешированного значения.

См. также

Рецепт 18.11 — информация о хранении шифрованных данных; библиотека password_compat; документация по функциям `password_hash()`, `password_verify()`, `crypt()` и `sha1()`.

18.8. Восстановление утраченных паролей

Задача

Требуется назначить пользователю новый пароль вместо утраченного.

Решение

Сгенерируйте новый пароль и отправьте его по адресу электронной почты пользователя (который должен храниться в системе):

```
/* Генерирование нового пароля. */
$new_password = '';
for ($i = 0; $i < 8; $i++) {
    $new_password .= chr(mt_rand(33, 126));
}

/* Хеширование нового пароля. */
$hashed_password = password_hash($new_password, PASSWORD_DEFAULT);

/* Сохранение нового хешированного пароля в базе данных. */
$st = $db->prepare('UPDATE users
            SET    password = ?
            WHERE  username = ?');

$st->execute(array($hashed_password, $clean['username']));

/* Отправка пользователю нового пароля по электронной почте. */
mail($clean['email'], 'New Password', "Your new password is: $new_password");
```

Обратите внимание: в этом коде используется функция `password_hash()`, поддерживаемая только в PHP 5.5. Если вы используете более старую версию PHP, ознакомьтесь с рекомендациями в разделе «Комментарий» Рецепта 18.7.

Комментарий

Если пользователь забыл свой пароль, а вы храните хешированные пароли, как рекомендуется в Рецепте 18.7, переслать ему забытый пароль не удастся. Односторонняя природа хеширования не позволит восстановить текстовый вариант пароля.

Вместо этого приходится генерировать новый пароль и отправлять его по электронной почте. Адрес, по которому отправляется новый пароль, должен быть зарегистрирован за пользователем; в противном случае вы не знаете, действительно ли новый адрес принадлежит этому пользователю. Возможно, адрес был предоставлен злоумышленником, который пытается выдать себя за настоящего пользователя.

Так как сообщение с новым паролем не хешируется, код Решения не включает имя пользователя в сообщение. Это сделано для того, чтобы сократить риск похищения пароля атакующим в случае перехвата. Чтобы новый пароль вообще не раскрывался в сообщении, предложите пользователю подтвердить свою личность, ответив на один или несколько личных вопросов (ответы на эти вопросы должны храниться в файле): «Как звали вашего первого питомца?» или «Девичья фамилия вашей матери?» — любой вопрос, ответ на который вряд ли будет известен злоумышленнику. Если пользователь правильно ответит на вопросы, разрешите ему выбрать новый пароль.

Один из компромиссов между безопасностью и удобочитаемостью — генерирование паролей из слов, разделенных цифрами:

```
$words = array('mother', 'basset', 'detain', 'sudden', 'fellow', 'logged',
               'remove', 'snails', 'direct', 'serves', 'daring', 'chirps',
               'reward', 'snakes', 'uphold', 'wiring', 'nurses', 'regent',
               'ornate', 'dogmas', 'mended', 'hinges', 'verbal', 'grimes',
               'ritual', 'drying', 'chests', 'newark', 'winged', 'hobbit');

$word_count = count($words);
$password = sprintf('%s%02d%s',
                    $words[mt_rand(0,$word_count - 1)],
                    mt_rand(0,99),
                    $words[mt_rand(0,$word_count - 1)]);
echo $password;
```

Этот код генерирует пароли, состоящие из двух шестибуквенных слов, разделенных двумя цифрами, например «mother43hobbit» или «verbal68nurses». Пароли получаются длинными, но они легко запоминаются благодаря словам.

Отправка нового пароля по адресу электронной почты пользователя неявно подразумевает, что пользователю, читающему почту по этому адресу, разрешен вход на сайт. Руководствуясь этим предположением, также можно отправить пользователю «одноразовый» URL-адрес. При посещении этого адреса выводится страница для сброса пароля. Если URL-адрес достаточно трудно подобрать, вы можете быть уверены в том, что страница сброса будет доступна только для получателя сообщения.

См. также

Рецепт 18.7 — информация о хранении хешированных паролей.

18.9. Проверка данных с использованием хешей

Задача

Требуется убедиться в том, что пользователи не изменяют данные, отправленные им в cookie или элементе формы.

Решение

Вместе с данными отправьте результат их хеширования с использованием затравки. При получении данных вычислите хеш полученного значения с той же затравкой. Если результаты не совпадут, значит, пользователь изменил данные.

Вот как генерируется хеш-код в скрытом поле формы:

```
<?php

/* Определение затравки. */
define('SALT', 'flyingturtle');

$id = 1337;
$idcheck = hash_hmac('sha1', $id, SALT);

?>
<input type="hidden" name="id" value="<?php echo $id; ?>" />
<input type="hidden" name="idcheck" value="<?php echo $idcheck; ?>" />
```

Проверка данных скрытого поля формы при отправке выполняется следующим образом:

```
/* Инициализация массива для фильтрованных данных. */
$clean = array();

/* Определение затравки. */
define('SALT', 'flyingturtle');

if (hash_hmac('sha1', $_POST['id'], SALT) === $_POST['idcheck']) {
    $clean['id'] = $_POST['id'];
} else {
    /* Ошибка */
}
```

Комментарий

В процессе обработки отправленных данных формы вычислите хеш-код отправленного значения `$_POST['id']` с той же затравкой. Если значение совпадет

с `$_POST['idcheck']`, то значение `$_POST['id']` не было изменено пользователем. Если значения не совпадают, значит, полученное значение `$_POST['id']` отлично от отправленного.

Чтобы использовать тот же прием с cookie, добавьте хеш к значению cookie вызовом `implode()`:

```
/* Определение затравки. */
define('SALT', 'flyingturtle');

$name = 'Ellen';

$namecheck = hash_hmac('sha1', $name, SALT);

setcookie('name', implode('|',array($name, $namecheck)));
```

Выделите хеш из значения cookie вызовом `explode()`:

```
/* Определение затравки. */
define('SALT', 'flyingturtle');

list($cookie_value, $cookie_check) = explode('|', $_COOKIE['name'], 2);

if (hash_hmac('sha1', $cookie_value, SALT) === $cookie_check) {
    $clean['name'] = $cookie_value;
} else {
    /* Ошибка */
}
```

Конечно, использование хеш-кода для проверки данных в форме или cookie зависит от затравки, использованной при вычислении хеша. Если злоумышленник узнает затравку, хеширование не предоставит никакой защиты. Помимо тщательной защиты затравки, рекомендуется как можно чаще изменять ее. Для дополнительной защиты используйте разные затравки, выбирая затравку для конкретного хеш-кода на основании некоторого свойства значения `$id` (например, 10 разных слов, выбираемых по формуле `$id%10`). Таким образом, даже если одно из слов станет известно злоумышленнику, ущерб будет относительно невелик.

См. также

Документация по функции `hash_hmac()`.

18.10. Шифрование и дешифрование данных

Задача

Требуется выполнить шифрование и дешифрование данных с применением одного из популярных алгоритмов.

Решение

Воспользуйтесь расширением PHP mcrypt:

```
$algorithm = MCRYPT_BLOWFISH;
$key = 'That golden key that opens the palace of eternity.';
$data = 'The chicken escapes at dawn. Send help with Mr. Blue.';
$mode = MCRYPT_MODE_CBC;

$iv = mcrypt_create_iv(mcrypt_get_iv_size($algorithm, $mode),
                       MCRYPT_DEV_URANDOM);

$encrypted_data = mcrypt_encrypt($algorithm, $key, $data, $mode, $iv);
$plain_text = base64_encode($encrypted_data);
echo $plain_text . "\n";

$encrypted_data = base64_decode($plain_text);
$decoded = mcrypt_decrypt($algorithm, $key, $encrypted_data, $mode, $iv);
// trim() удалит все завершающие NULL-байты, которые
// могут быть добавлены функцией mcrypt_decrypt() для выравнивания
// вывода по границе 8-байтовых блоков
echo trim($decoded) . "\n";
```

Результат выполнения кода выглядит так:

```
Cd4Uzc1c5lDxxWc7rXv+mbsElwj2ENrYg5HAPiaOpe7Wr8UAG5aXD9CoG6NdKoOWLSumg9ffSnE=
The chicken escapes at dawn. Send help with Mr. Blue.
```

Комментарий

Расширение mcrypt предоставляет интерфейс к mcrypt — библиотеке, реализующей множество разных алгоритмов шифрования. Шифрование и дешифрование данных выполняются функциями `mcrypt_encrypt()` и `mcrypt_decrypt()` соответственно. Каждая из этих функций получает пять аргументов. Первый аргумент определяет используемый алгоритм. Чтобы узнать, какие алгоритмы поддерживает mcrypt в вашей системе, вызовите функцию `mcrypt_list_algorithms()`. Во втором аргументе передается ключ шифрования, а в третьем — данные для шифрования или дешифрования. Четвертый аргумент определяет режим шифрования или дешифрования (список поддерживаемых режимов возвращается функцией `mcrypt_list_modes()`). Пятый аргумент содержит вектор инициализации (IV, Initialization Vector), который используется в некоторых режимах в процессе шифрования или дешифрования.

Все аргументы, кроме данных, должны совпадать при шифровании/дешифровании. Если используемый режим требует передачи вектора инициализации, вектор можно передать в незашифрованном виде (в сочетании с зашифрованным текстом).

В разных обстоятельствах уместны разные режимы. В режиме CBC (Cipher Block Chaining) данные шифруются по блокам, а зашифрованное значение каждого блока (вместе с ключом) используется для вычисления зашифрованного значения следующего блока. Вектор инициализации влияет на зашифрованное значение

первого блока. Режимы CFB (Cipher Feedback) и OFB (Output Feedback) также используют вектор инициализации, но данные шифруются по единицам, меньшим размера блока. Учтите, что в режиме OFB при использовании единиц, меньших размера блока, могут возникнуть проблемы с безопасностью. В режиме ECB (Electronic Code Book) данные шифруются в раздельных блоках, не зависящих друг от друга. Режим ECB не использует вектор инициализации. При многократном использовании он также менее безопасен, чем другие режимы, потому что конкретный фрагмент обычного текста с конкретным ключом всегда дает один и тот же зашифрованный текст. Константы для выбора режима шифрования перечислены в таблице 18.1.

Таблица 18.1. Константы режимов mcrypt

Константа	Описание
MCRYPT_MODE_ECB	Режим Electronic Code Book
MCRYPT_MODE_CBC	Режим Cipher Block Chaining
MCRYPT_MODE_CFB	Режим Cipher Feedback
MCRYPT_MODE_OFB	Режим Output Feedback с 8 битами обратной связи
MCRYPT_MODE_NOFB	Режим Output Feedback с n битами обратной связи, где n — размер блока, используемый алгоритмом
MCRYPT_MODE_STREAM	Режим Stream Cipher для таких алгоритмов, как RC4 и WAKE

Размер блока зависит от алгоритма. Чтобы узнать размер блока конкретного алгоритма, вызовите `mcrypt_get_block_size()`. Размер вектора инициализации также определяется алгоритмом и режимом. Функции `mcrypt_create_iv()` и `mcrypt_get_iv_size()` позволяют легко сгенерировать случайный вектор инициализации:

```
$iv = mcrypt_create_iv(mcrypt_get_iv_size($algorithm, $mode),
                       MCRYPT_DEV_URANDOM);
```

В первом аргументе `mcrypt_create_iv()` передается размер вектора, а во втором — источник рандомизации. Поддерживаются три источника рандомизации: `MCRYPT_DEV_RANDOM` — чтение из псевдоустройства `/dev/random`, `MCRYPT_DEV_URANDOM` — чтение из псевдоустройства `/dev/urandom` и `MCRYPT_RAND` — внутренний генератор случайных чисел. Псевдоустройства генерирования случайных чисел поддерживаются не всеми операционными системами. Чтобы получить поток случайных чисел без повторений, вызовите `srand()` перед использованием `MCRYPT_RAND`.

См. также

Документация расширения mcrypt; библиотека mcrypt. Выбор подходящего алгоритма и его безопасное использование требует осторожности и планирования: дополнительную информацию о mcrypt и используемых алгоритмах шифрования можно найти в документации расширения mcrypt, на домашней странице mcrypt и странице Википедии с описанием /dev/urandom и /dev/urandom. Среди хороших

книг по криптографии можно упомянуть «Applied Cryptography» Брюса Шнейера (Bruce Schneier) (издательство Wiley) и «Cryptography: Theory and Practice» Дугласа P. Стинсона (Douglas R. Stinson) (издательство Chapman & Hall).

18.11. Хранение шифрованных данных в файле или в базе данных

Задача

Требуется сохранить шифрованные данные для последующего чтения и дешифрования веб-сервером.

Решение

Сохраните вместе с зашифрованными данными дополнительную информацию, необходимую для дешифрования (алгоритм, режим шифрования, вектор инициализации) — но только не ключ:

```
/* Шифрование данных. */
$algorithm  = MCRYPT_BLOWFISH;
$mode = MCRYPT_MODE_CBC;
$iv = mcrypt_create_iv(mcrypt_get_iv_size($algorithm, $mode),
                       MCRYPT_DEV_URANDOM);
$ciphertext = mcrypt_encrypt($algorithm, $_POST['key'], $_POST['data'],
                             $mode, $iv);

/* Хранение шифрованных данных. */
$st = $db->prepare('INSERT
            INTO   noc_list (algorithm, mode, iv, data)
            VALUES (?, ?, ?, ?)');
$st->execute(array($algorithm, $mode, $iv, $ciphertext));
```

Чтобы расшифровать данные, получите ключ у пользователя и примените его к сохраненным данным:

```
$row = $db->query('SELECT *
                   FROM   noc_list
                   WHERE  id = 27')->fetch();
$plaintext = mcrypt_decrypt($row['algorithm'],
                            $_POST['key'],
                            $row['data'],
                            $row['mode'],
                            $row['iv']);
```

Комментарий

Сценарий save-crypt.php из листинга 18.1 сохраняет зашифрованные данные в файле.

Листинг 18.1. save-crypt.php

```
function show_form() {
    $html = array();
    $html['action'] = htmlentities($_SERVER['PHP_SELF'], ENT_QUOTES, 'UTF-8');

    print<<<FORM
<form method="POST" action="{$html['action']}">
<textarea name="data"
          rows="10" cols="40">Enter data to be encrypted here.</textarea>
<br />
Encryption Key: <input type="text" name="key" />
<br />
<input name="submit" type="submit" value="Save" />
</form>
FORM;
}

function save_form() {
    $algorithm  = MCRYPT_BLOWFISH;
    $mode = MCRYPT_MODE_CBC;

    /* Шифрование данных. */
    $iv = mcrypt_create_iv(mcrypt_get_iv_size($algorithm, $mode),
                           MCRYPT_DEV_URANDOM);
    $ciphertext = mcrypt_encrypt($algorithm,
                                 $_POST['key'],
                                 $_POST['data'],
                                 $mode,
                                 $iv);

    /* Сохранение зашифрованных данных. */
    $filename = tempnam('/tmp','enc') or exit($php_errormsg);
    $file = fopen($filename, 'w') or exit($php_errormsg);
    if (FALSE === fwrite($file, $iv.$ciphertext)) {
        fclose($file);
        exit($php_errormsg);
    }

    fclose($file) or exit($php_errormsg);

    return $filename;
}

if (isset($_POST['submit'])) {
    $file = save_form();
    echo "Encrypted data saved to file: $file";
} else {
    show_form();
}
```

В листинге 18.2 приведена соответствующая программа get-crypt.php, которая получает имя файла и ключ и возвращает дешифрованные данные.

Листинг 18.2. get-crypt.php

```
function show_form() {
    $html = array();
```

```
    $html['action'] = htmlentities($_SERVER['PHP_SELF'], ENT_QUOTES, 'UTF-8');

    print<<<FORM
<form method="POST" action="{$html['action']}">
Encrypted File: <input type="text" name="file" />
<br />
Encryption Key: <input type="text" name="key" />
<br />
<input name="submit" type="submit" value="Display" />
</form>
FORM;
}

function display() {
    $algorithm  = MCRYPT_BLOWFISH;
    $mode = MCRYPT_MODE_CBC;

    $file = fopen($_POST['file'], 'r') or exit($php_errormsg);
    $iv = fread($file, mcrypt_get_iv_size($algorithm, $mode));
    $ciphertext = fread($file, filesize($_POST['file']));
    fclose($file);

    $plaintext = mcrypt_decrypt($algorithm, $_POST['key'], $ciphertext,
                                $mode, $iv);
    echo "<pre>$plaintext</pre>";
}

if (isset($_POST['submit'])) {
    display();
} else {
    show_form();
}
```

В этих двух программах алгоритм шифрования и режим жестко запрограммированы, поэтому хранить эту информацию в файле не нужно. Файл состоит из вектора инициализации, за которым следуют зашифрованные данные. В ограничителе после вектора инициализации (IV) нет необходимости, потому что `mcrypt_get_iv_size()` возвращает точное количество байтов, которые должны быть прочитаны программой дешифрования для получения всего вектора. Все, что следует после него, — зашифрованные данные.

Шифрование файлов методом, описанным в этом рецепте, обеспечивает защиту при получении злоумышленником доступа к серверу, на котором хранятся файлы. Без ключа или огромного объема вычислительных ресурсов атакующий не сможет прочитать файлы. Тем не менее безопасность, обеспечиваемая зашифрованными файлами, будет подорвана, если данные для шифрования и ключи передаются между сервером и браузером сервера в открытом виде. Сторона, перехватывающая или отслеживающая сетевой трафик, сможет увидеть данные еще до того, как они будут зашифрованы. Для предотвращения подобных перехватов следует использовать SSL.

Дополнительный риск при шифровании данных веб-сервером, как в этом рецепте, связан с видимостью данных до их шифрования и записи в файл. Пользователь,

обладающий доступом к серверу с привилегиями администратора или root, может просмотреть содержимое памяти процесса веб-сервера и попытаться найти в ней незашифрованные данные и ключ. Если операционная система выгружает образ памяти процесса веб-сервера на диск, незашифрованные данные также могут быть доступны в файле подкачки. Такие атаки бывает сложно провести, но их последствия оказываются разрушительными. После того как зашифрованные данные окажутся в файле, их не сможет прочитать даже атакующий с привилегированным доступом к веб-серверу, но если атакующий сможет «подсмотреть» незашифрованные данные до их сохранения, шифрование никакой пользы не принесет.

См. также

Рецепт 18.13 — SSL и защита данных при пересылке по сети; документация по функциям `mcrypt_encrypt()`, `mcrypt_decrypt()`, `mcrypt_create_iv()` и `mcrypt_get_iv_size()`.

18.12. Обмен зашифрованными данными с другим сайтом

Задача

Требуется организовать безопасный обмен данными с другим сайтом.

Решение

Если другой сайт загружает данные с вашего сайта, разместите их на странице с парольной защитой. Данные также можно предоставить в зашифрованном виде, с паролем или без. Если вам потребуется передать данные на другой сайт, отправьте (возможно, зашифрованные) данные методом POST на URL-адрес с парольной защитой.

Комментарий

Следующая страница запрашивает имя пользователя и пароль, а затем шифрует и выводит содержимое файла с отчетом о вчерашней активности учетной записи:

```
$user = 'bank';
$password = 'fas8uj3';

if ($_SERVER['PHP_AUTH_USER'] != $user ||
    $_SERVER['PHP_AUTH_PW'] != $password) {
    header('WWW-Authenticate: Basic realm="Secure Transfer"');
    header('HTTP/1.0 401 Unauthorized');
```

```
    echo "You must supply a valid username and password for access.";
    exit;
}

header('Content-type: text/plain; charset=UTF-8');
$filename = strftime('/usr/local/account-activity.%Y-%m-%d', time() - 86400);
$data = implode('', file($filename));

$algorithm  = MCRYPT_BLOWFISH;
$mode = MCRYPT_MODE_CBC;
$key  = "There are many ways to butter your toast.";

/* Шифрование данных. */
$iv = mcrypt_create_iv(mcrypt_get_iv_size($algorithm, $mode),
                                          MCRYPT_DEV_URANDOM);
$ciphertext = mcrypt_encrypt($algorithm, $key, $data, $mode, $iv);

echo base64_encode($iv.$ciphertext);
```

Соответствующий код для получения зашифрованной страницы и дешифрования информации выглядит так:

```
$user = 'bank';
$password = 'fas8uj3';
$algorithm  = MCRYPT_BLOWFISH;
$mode = MCRYPT_MODE_CBC;
$key  = "There are many ways to butter your toast.";

$url = 'https://bank.example.com/accounts.php';

$c = curl_init($url);
curl_setopt($c, CURLOPT_USERPWD, "$user:$password");
curl_setopt($c, CURLOPT_RETURNTRANSFER, TRUE);
$data = curl_exec($c);
if (FALSE === $data) {
    exit("Transfer failed: " . curl_error($c));
}

$binary_data = base64_decode($data);
$iv_size = mcrypt_get_iv_size($algorithm, $mode);
$iv = substr($binary_data, 0, $iv_size);
$ciphertext = substr($binary_data, $iv_size, strlen($binary_data));

echo mcrypt_decrypt($algorithm, $key, $ciphertext, $mode, $iv);
```

Программа восстановления выполняет все действия программы шифрования, но в обратном порядке. Она получает зашифрованные данные в кодировке Base64, сообщая имя пользователя и пароль. Затем данные декодируются из Base64 и отделяются от вектора инициализации. Наконец, программа дешифрует данные и выводит их.

В приведенном примере имя пользователя и пароль передаются по сети в виде простого текста, если только связь не осуществляется по каналу SSL. С другой стороны, при использовании SSL вряд ли понадобится шифровать содержимое

файла. Мы включили в программу как запрос пароля, так и шифрование файла, просто чтобы показать, как это можно сделать.

Однако возможна ситуация, в которой будут полезны как парольная защита, так и шифрование файла: если файл не дешифруется автоматически при получении. Автоматизированная программа может получить зашифрованный файл и поместить его (все еще в зашифрованном виде) в место, где к нему можно будет обратиться позднее. В этом случае хранение ключа шифрования в программе получения данных не является обязательным.

См. также

Рецепт 18.13 — SSL и защита данных при передаче по сети; документация по функциям `mcrypt_encrypt()` и `mcrypt_decrypt()`.

18.13. Обнаружение SSL

Задача

Требуется узнать, использовался ли для передачи запроса протокол SSL.

Решение

Проверьте значение `$_SERVER['HTTPS']`:

```
if ('on' == $_SERVER['HTTPS']) {
  echo 'The secret ingredient in Coca-Cola is Soylent Green.';
} else {
  echo 'Coca-Cola contains many delicious natural and artificial flavors.';
}
```

Комментарий

Протокол SSL работает на более низком уровне, чем HTTP. Веб-сервер и браузер согласовывают подключение с уровнем защиты, который зависит от их возможностей, после чего сообщения HTTP передаются по этому защищенному подключению. С точки зрения злоумышленника, перехватывающего трафик, перед ним бессмысленный поток байтов, из которого невозможно извлечь полезную информацию.

У разных веб-серверов действуют разные требования к использованию SSL, поэтому за подробностями следует обращаться к документации сервера. Никакие специальные изменения для работы на базе SSL в PHP вносить не нужно.

Кроме изменения кода в зависимости от `$_SERVER['HTTPS']` вы также можете включить передачу cookie только по каналам SSL. Если последний аргумент

setcookie() равен true, то браузер вернет cookie серверу только по защищенному подключению:

```
/* Создать cookie с именем "sslonly" и значением "yes", срок действия которого
истекает с завершением текущего сеанса. */
if ('on' === $_SERVER['HTTPS']) {
    setcookie('sslonly', 'yes', 0, '/', 'example.org', true);
}
```

Хотя браузер возвращает такие cookie серверу только по подключению SSL, сервер отправляет их браузеру (при вызове setcookie() в странице) независимо от того, была ли запрошена страница, создавшая cookie, по подключению SSL. Если в cookie размещаются конфиденциальные данные, убедитесь в том, что cookie создается тоже только по запросу SSL. Также следует помнить, что данные cookie хранятся на компьютере пользователя в незашифрованном виде.

См. также

Документация по функции setcookie().

18.14. Шифрование электронной почты и GPG

Задача

Требуется отправлять сообщения электронной почты в зашифрованном виде. Допустим, вы принимаете на своем сайте заказы и хотите отправить на фабрику сообщение с подробной информацией о заказе. Шифрование сообщений предотвратит передачу конфиденциальной информации (например, номеров кредитных карт) в незащищенном виде.

Решение

Воспользуйтесь функциями расширения gnupg для шифрования тела сообщений электронной почты средствами GPG (GNU Privacy Guard) перед отправкой:

```
$plaintext_body = 'Some sensitive order data';
$recipient = 'ordertaker@example.com';

$g = gnupg_init();
gnupg_seterrormode($g, GNUPG_ERROR_WARNING);
// Идентификационная метка ключа получателя
$a = gnupg_addencryptkey($g, "5495F0CA9C8F30A9274C2259D7EBE8584CEF302B");
// Идентификационная метка ключа отправителя
$b = gnupg_addsignkey($g, "520D5FC5C85EF4F4F9D94E1C1AF1F7C5916FC221",
                 "passphrase");
```

```
$encrypted_body = gnupg_encryptsign($g, $plaintext_body);

mail($recipient, 'Web Site Order', $encrypted_body);
```

Для дешифрования сообщений используются средства GPG, PGP (Pretty Good Privacy) или плагин почтового клиента, поддерживающий одну из этих программ.

Комментарий

Приведенный в Решении код использует расширение PHP ghupg, которое в свою очередь зависит от библиотеки GPGME для выполнения операций шифрования и цифровой подписи сообщений по стандарту OpenPGP.

Ресурс, возвращаемый `gnupg_init()`, используется в остальных вызовах функций как контейнер для конкретных настроек, связанных с выполняемым шифрованием. Затем вызов `gnupg_seterrormode($g, GNUPG_ERROR_WARNING)` обеспечивает выдачу предупреждений PHP при возникновении проблем с операциями GnuPG.

В этом примере сообщение шифруется и снабжается цифровой подписью. Шифрование гарантирует, что сообщение может быть расшифровано и прочитано только его предполагаемым получателем. Подпись позволяет получателю убедиться в том, что сообщение было получено от предполагаемого отправителя.

Идентификационная метка ключа, передаваемая `gnupg_addencryptkey()`, указывает, какой ключ должен использоваться для шифрования сообщения. Только пользователь с доступом к закрытому ключу, связанному с этой меткой, сможет расшифровать сообщение.

Идентификационная метка ключа, передаваемая `gnupg_addsignkey()`, указывает, какой ключ должен использоваться для цифровой подписи сообщения. Третий аргумент `gnupg_addsignkey()` содержит пароль, связанный с этим закрытым ключом.

Функции расширения gnupg ищут ключи в том же месте, где и исполняемый файл командной строки *gpg*: в каталоге с именем `.gnupg` внутри домашнего каталога (или домашнего каталога пользователя, от имени которого работает PHP). Чтобы приказать PHP искать ключи в другом месте, присвойте нужный каталог переменной окружения `GNUPGHOME`.

После того как для ресурса gnupg будут назначены ключи, вызов `gnupg_encryptsign()` создаст зашифрованное подписанное сообщение. По умолчанию это значение преобразуется в простой ASCII-текст.

Если вам потребуется найти идентификационную метку для передачи `gnupg_addencryptkey()` или `gnupg_addsignkey()`, используйте функцию `gnupg_keyinfo()`, как показано ниже:

```
$email = 'friend@example.com';

$g = gnupg_init();
```

```
$keys = gnupg_keyinfo($g, $email);
if (count($keys) == 1) {
    $fingerprint = $keys[0]['subkeys'][0]['fingerprint'];
    print "Fingerprint for $email is $fingerprint";
}
else {
    print "Expected 1, found " . count($keys) .
        " keys for $email";
}
```

Для заданного ресурса gnupg и строки поиска функция `gnupg_keyinfo()` возвращает массив, содержащий информацию о каждом ключе из набора, у которого идентификатор UID (или его часть) совпадает со строкой поиска. Каждый элемент в этом возвращаемом массиве сам представляет собой массив, состоящий из множества элементов и подмассивов с подробной информацией о ключе. Ключ `fingerprint` первого элемента массива `subkeys` содержит метку, которая может передаваться другим функциям gnupg.

См. также

Домашняя страница GNU Privacy Guard и информация о библиотеке GPGME; страница PECL — информация о расширении gnupg и документация по нему.

19 Интернационализация и локализация

19.0. Введение

Каждому программисту, работающему на PHP, рано или поздно приходится учить английский хотя бы для того, чтобы разобраться в именах функций и языковых конструкциях. PHP может создавать приложения, которые общаются с пользователем практически на любом языке. У некоторых приложений круг пользователей включает людей, говорящих на разных языках. Если вам потребуется, скажем, взять приложение, написанное для франкоязычных пользователей, и адаптировать его для немецкого языка, поддержка интернационализации и локализации в PHP упростит вашу задачу.

Рецепты этой главы используют функциональность расширения PHP intl для использования инициализации и локализации. Расширение работает на базе мощной библиотеки ICU. Эта популярная библиотека также имеет реализации для C/C++ и Java. А следовательно, концепции, которые вы освоите в PHP, пригодятся, если вам придется заниматься глобализацией в других языках (программирования).

Расширение intl включается в поставку PHP версии 5.3.0 и выше. Чтобы использовать его с PHP 5.2.0 и выше, установите его из репозитария PECL.

Интернационализацией называется процесс переработки приложения, спроектированного только для одного локального контекста, и его реструктуризации для того, чтобы оно могло использоваться во многих разных локальных контекстах.

Локализацией называется процесс добавления поддержки нового локального контекста в интернационализированное приложение.

Локальный контекст (locale) представляет собой набор параметров, описывающих правила форматирования текста и языковые стандарты в некоторой области мира. В частности, в локальные контексты включаются следующие правила:

- *Упорядочение* — правила сортировки текста (какие буквы предшествуют другим буквам в алфавитном порядке).
- *Числа* — правила вывода числовых данных (включая денежные суммы): способ группировки разрядов, символы-разделители групп разрядов и дробной части, обозначение отрицательных значений.
- *Время и дата* — правила форматирования и вывода времени и даты: названия месяцев и дней, использование 24- или 12-часовой шкалы.
- *Сообщения* — текстовые сообщения для приложений, которые должны выводить информацию на нескольких языках.

Идентификатор локального контекста состоит из нескольких компонентов, разделенных подчеркиваниями. Первая часть определяет код языка (сокращение, обозначающее язык, например «en» для английского или «pt» для португальского). Коды языков состоят из двух букв и определяются в стандарте ISO 639-1.

Далее идет необязательный *код письменности*, который указывает, какой набор символов должен использоваться для представления текста в этом локальном контексте (например, «Arab» для арабского алфавита или «Cyrl» для кириллицы). Коды письменностей перечислены в стандарте ISO 15924.

Далее идет необязательный *код страны*, по которому различаются страны, говорящие на разных вариантах одного языка. Например, «en_US» соответствует американской разновидности английского языка, а «en_UK» — английскому языку, на котором говорят в Великобритании; «pt_BR» соответствует бразильскому португальскому языку, а «pt_PT» — «португальскому португальскому». Коды стран состоят из двух букв и определяются в стандарте ISO 3166.

Чтобы в программе можно было определять дополнительные различия в пределах одного языка и страны, следующий необязательный компонент идентификатора локального контекста может содержать *код варианта*. Коды вариантов, документированные в реестре языковых субтегов IANA, описывают такие нюансы, как использование бискайского диалекта баскского языка (вариант `biscayan`) или использование «высокой норвежской» орфографии норвежского языка (вариант `hognorsk`). Скорее всего, в повседневном использовании локальных контекстов коды вариантов вам не понадобятся.

После экзотического кода варианта может следовать необязательный список ключевых слов с префиксом `@`. Эти ключевые слова представляют собой пары «имя=значение», разделенные символом «;». Они предоставляют дополнительную возможность для передачи расширенной информации о локальном контексте. Например, идентификатор `fr_CA@currency=USD` обозначает локальный контекст французского языка для Канады, но с использованием долларов США в качестве денежной единицы.

Чтобы помочь вам разобраться с локальными контекстами, Рецепт 19.1 показывает, как выбрать локальный контекст в соответствии с требованием браузера пользователя.

Для правильной локализации простого текста, чисел, даты и времени, денежных сумм применяются разные методы. Локализация также может применяться к внешним ресурсам, используемым вашей программой, например изображениям и включаемым файлам. Локализация такого контента рассматривается в Рецептах 19.2–19.7.

Сортировка с учетом локального контекста является темой Рецепта 19.8, а обработке больших объемов локализованных данных посвящен Раздел 19.9.

Из Рецептов 19.10–19.12 вы узнаете, как обеспечить нормальную работу программ с разнообразными кодировками символов, чтобы они справлялись с такими строками, как à l'Opéra-Théâtre и 優之良品. Одно из возможных решений — хранение всего текста, обрабатываемого программой, в кодировке UTF-8. Эта схема позволяет обрабатывать как латинские символы из привычной кодировки ISO-8859-1, так и символы других систем письменности из многих стран мира. Основная тема этих рецептов — использование UTF-8 для реализации цельного, независимого от языка пользовательского интерфейса.

19.1. Определение локального контекста пользователя

Задача

Требуется использовать правильный локальный контекст, установленный браузером пользователя.

Решение

Передайте входящий заголовок HTTP `Accept-Language` функции `Locale::acceptFromHttp()` для получения правильного идентификатора локального контекста:

```
if (isset($_SERVER['HTTP_ACCEPT_LANGUAGE'])) {
    $localeToUse = Locale::acceptFromHttp($_SERVER['HTTP_ACCEPT_LANGUAGE']);
}
else {
    $localeToUse = Locale::getDefault();
}
```

Комментарий

В разделе 14.4 документа RFC 2616, содержащего описание HTTP, представлены правила передачи браузером вместе с запросом заголовка `Accept-Language`, определяющего предпочтительные языки для ответа на запрос. Функция `Locale::acceptFromHttp()` просматривает эти значения и возвращает идентификатор локального контекста ICU, соответствующий предпочтительному языку.

Если предпочтительный язык не задан, используйте локальный контекст, возвращаемый функцией `Locale::getDefault()`; его значение берется из конфигурационной директивы `intl.default_locale` (если она не была переопределена вызовом `Locale::setDefault()`).

См. также

Документация по методам `Locale::acceptFromHttp()`, `Locale::getDefault()` и `Locale::setDefault()`; RFC 2616.

19.2. Локализация текстовых сообщений

Задача

Требуется вывести текстовые сообщения на языке, соответствующем локальному контексту.

Решение

Создайте каталог сообщений. Извлеките подходящую строку из каталога и передайте ее объекту `MessageFormatter`, чтобы отформатировать для вывода:

```
$messages = array();
$messages['en_US'] =
    array('FAVORITE_FOODS' => 'My favorite food is {0}.',
          'FRIES' => 'french fries',
          'CANDY' => 'candy',
          'CHIPS' => 'potato chips',
          'EGGPLANT' => 'eggplant');
$messages['en_GB'] =
    array('FAVORITE_FOODS' => 'My favourite food is {0}.',
          'FRIES' => 'chips',
          'CANDY' => 'sweets',
          'CHIPS' => 'crisps',
          'EGGPLANT' => 'aubergine');

foreach (array('en_US', 'en_GB') as $locale) {
    $candy = new MessageFormatter($locale, $messages[$locale]['CANDY']);
    $favs = new MessageFormatter($locale, $messages[$locale]['FAVORITE_FOODS']);
    print $favs->format(array($candy->format(array()))) . "\n";
}
```

Результат выполнения выглядит так:

```
My favorite food is candy.
My favourite food is sweets.
```

Комментарий

В первом аргументе конструктора `MessageFormatter` передается локальный контекст, для которого должно форматироваться сообщение. Второй аргумент задает схему сообщения. Мощь `MessageFormatter` происходит от специальных частей шаблона, заключенных в фигурные скобки. В этих позициях в шаблон подставляются аргументы, переданные методу `format()`. В приведенном коде `{0}` в шаблоне in заменяется первым элементом массива, переданного `format()`, `{1}` в шаблоне заменяется вторым элементом массива и т. д.

Проще всего обозначать аргументы шаблона числами — `{0}` для первого аргумента, `{1}` для второго и т. д. Затем значение первого элемента массива, переданного `format()`, заменяет `{0}`, значение второго заменяет `{1}` и т. д. В приведенном выше примере шаблон `FAVORITE_FOODS` содержит один заполнитель, а в шаблоне `CANDY` заполнителей нет, поэтому массив, переданный `$favs->format()`, содержит один элемент, а массив, переданный `$candy->format()`, пуст.

Откровенно говоря, с примитивными аргументами вида `{0}` результат не впечатляет. Возможности `MessageFormatter` проявляются с более сложными аргументами. Для примера возьмем сообщение, в котором текст должен изменяться в зависимости от количества задействованных объектов: например, «You have one item in your shopping cart» и «You have two items in your shopping cart». Вот как это требование выражается средствами `MessageFormatter`:

```
$messages = array();
$messages['en_US'] =
    array('CART' => "You have {0,spellout} " .
                    "{0, plural, " .
                    " =1    {item} " .
                    " other {items} } " .
                    "in your shopping cart.");
$messages['fr_FR'] =
    array('CART' => "Vous {0, plural, " .
                    " =0 {n'avez pas d'articles} ".
                    " =1 {avez un article} ".
                    " other {avez {0,spellout} articles}} ".
                    "dans votre panier.");

$fmts = array();
foreach (array_keys($messages) as $locale) {
    $fmts[$locale] = new MessageFormatter($locale, $messages[$locale]['CART']);
}

for ($i = 0; $i < 10; $i++) {
    foreach ($fmts as $locale => $obj) {
        print $obj->format(array($i)) . "\n";
    }
}
```

Результат выполнения кода выглядит так:

```
You have zero items in your shopping cart.
Vous n'avez pas d'articles dans votre panier.
You have one item in your shopping cart.
Vous avez un article dans votre panier.
You have two items in your shopping cart.
Vous avez deux articles dans votre panier.
You have three items in your shopping cart.
Vous avez trois articles dans votre panier.
You have four items in your shopping cart.
Vous avez quatre articles dans votre panier.
You have five items in your shopping cart.
Vous avez cinq articles dans votre panier.
You have six items in your shopping cart.
Vous avez six articles dans votre panier.
You have seven items in your shopping cart.
Vous avez sept articles dans votre panier.
You have eight items in your shopping cart.
Vous avez huit articles dans votre panier.
You have nine items in your shopping cart.
Vous avez neuf articles dans votre panier.
```

Аргументы шаблона в этом примере не ограничиваются простым номером `{0}`. Аргумент `{0,spellout}` означает: «Использовать аргумент 0, но интерпретировать его как тип `spellout`». Это встроенный тип ICU, который преобразует числа в их эквиваленты, записанные словами. Так как класс `MessageFormatter` учитывает локальный контекст, он знает, какие слова следует использовать, например «three» для английского языка или «trois» для французского. Шаблон также включает аргумент типа `plural`, позволяющий использовать разный текст в зависимости от значения аргумента. В английском языке выводится текст `item`, если аргумент равен 1, а в остальных случаях выводится текст `items`. Во французском языке для обеспечения правильной грамматики вывода различаются 0, 1 и все остальные числа.

С типом аргумента `plural` решение в процессе форматирования принимается в зависимости от числового значения аргумента. Более общий тип аргумента `select` позволяет делать выбор на основании произвольных значений. Например, эта возможность пригодится для выбора разных слов в зависимости от мужского или женского рода аргумента. В английском языке это выглядит примерно так:

```
$message = '{0, select, f {She} m {He} other {It}} went to the store.';
$fmt = new MessageFormatter('en_US', $message);
print $fmt->format(array('f')) . "\n";
print $fmt->format(array('m')) . "\n";
print $fmt->format(array('Unknown')) . "\n";
```

Результат:

```
She went to the store.
He went to the store.
It went to the store.
```

Если аргумент 0 содержит `f`, в вывод интерполируется строка «She». Для аргумента `m` подставляется строка «He». Во всех остальных случаях используется «It».

В PHP 5.5 класс `MessageFormatter` поддерживает не только числовые, но и именованные аргументы. Вам остается только следить за тем, чтобы ссылка на аргумент в шаблоне соответствовала ключу массива, использованному при вызове `format()`. Пример:

```
$message = 'I like to eat {food} and {drink}.';
$fmt = new MessageFormatter('en_US', $message);
print $fmt->format(array('food' => 'eggs',
                         'drink' => 'water'));
```

Результат:

```
I like to eat eggs and water.
```

Чтобы получить доступ к этой возможности в старой версии PHP, установите версию 3.0 (или выше) расширения intl из репозитория PECL.

См. также

Документация класса `MessageFormatter`. Так как реализация `MessageFormatter` зависит от ICU, много полезной информации также можно найти в документации ICU по форматированию сообщений и аргументов, особенно в ICU User Guide и ICU 53.1.

19.3. Локализация даты и времени

Задача

Требуется вывести дату и время в формате, соответствующем локальному контексту.

Решение

Используйте тип аргумента `date` или `time` с необязательным стилем `short`, `medium`, `long` или `full` в сообщении `MessageFormatter`:

```
$when = 1376943432; // Секунды от начала эпохи
$message = "It is {0,time,short} on {0,date,medium}.";
$fmt = new MessageFormatter('en_US', $message);
print $fmt->format(array($when));
```

Результат:

```
It is 4:17 PM on Aug 19, 2013.
```

Используйте шаблон форматирования с типом аргумента `date` или `time` в сообщении `MessageFormatter`:

```
$when = 1376943432; // Секунды от начала эпохи
$message = "Maintenant: {0,date,eeee dd MMMM y}";
$fmt = new MessageFormatter('fr_FR', $message);
print $fmt->format(array($when));
```

Результат:

```
Maintenant: lundi 19 août 2013
```

Метод `format()` класса `IntlDateFormatter`:

```
$when = 1376943432; // Секунды от начала эпохи
$fmt = new IntlDateFormatter('en_US', IntlDateFormatter::FULL,
                             IntlDateFormatter::FULL);
print $fmt->format($when);
```

Результат:

```
Monday, August 19, 2013 at 8:17:12 PM GMT
```

Комментарий

Типы аргументов `date` и `time` класса `MessageFormatter` упрощают включение локализованных представлений даты и времени в выводимый текст. Эти «готовые» форматы не только учитывают настройки локального контекста и выводят правильные компоненты в правильном порядке (например, с размещением дня перед номером месяца или наоборот), но и переводят месяцы и дни на соответствующий язык.

Если вы хотите лучше контролировать формат даты и времени, включаемые в сообщение, определите форматный шаблон. В таблице 19.1 перечислены элементы, которые могут включаться в шаблон даты/времени.

Таблица 19.1. Символы шаблонов даты/времени

Тип	Символ	Описание	Пример
Часы	a	Половина дня	PM
Часы	h	Часы по 12-часовой шкале (1–12)	3
Часы	K	Часы по 12-часовой шкале (0–11)	3
Часы	H	Часы по 24-часовой шкале (0–23)	15
Часы	k	Часы по 24-часовой шкале (1–24)	15
Минуты	m	Минуты (0–59)	8
Секунды	s	Секунды (0–59)	7
Секунды	S	Десятые доли секунд (0–9)	0
Секунды	SS	Сотые доли секунд (00–99)	00
Секунды	SSS	Миллисекунды (000–999)	000

Тип	Символ	Описание	Пример
Секунды	A	Миллисекунды в сутках	54487000
Дни	d	День месяца (1–31)	18
Дни	D	День года (1–366)	78
Дни	EEEEE	День недели, короткое сокращение	T
Дни	EEE	День недели, длинное сокращение	Thu
Дни	EEEE	День недели, название	Thursday
Дни	e	День недели, номер (от 0 или 1 до 6 или 7, локализован)	5
Дни	F	День недели в месяце (например, 3 для «третьей среды»)	3
Дни	g	Модифицированный день юлианского календаря	2453083
Неделя	w	Неделя года с локализованным началом отсчета (1–52)	12
Неделя	W	Неделя месяца (1–5)	3
Месяц	M	Месяц (1–12)	3
Месяц	MMMMM	Месяц, короткое сокращение	M
Месяц	MMM	Месяц, длинное сокращение	Mar
Месяц	MMMM	Название месяца	March
Год	y	Год из 4 цифр	2004
Год	yy	Год из 2 цифр	04
Часовой пояс	z	Часовой пояс, включая летнее время, сокращение	EST
Часовой пояс	zzzz	Часовой пояс, включая летнее время, полное название	Eastern Standard Time
Часовой пояс	Z	Часовой пояс в формате RFC-822	-0500
Часовой пояс	ZZZZ	Часовой пояс как смещение GMT	GMT-05:00
Часовой пояс	ZZZZZ	Часовой пояс в формате ISO-8601	GMT-05:00
Часовой пояс	v	Часовой пояс без летнего времени, сокращение	ET
Часовой пояс	vvvv	Часовой пояс без летнего времени, полное название	Eastern Time
Часовой пояс	VVVV	Часовой пояс как территориальное обозначение	United States Time (New York)
Прочее	Q	Квартал, номер	1
Прочее	QQQ	Квартал, номер с префиксом	Q1
Прочее	QQQQ	Квартал в текстовой записи	1st quarter
Прочее	G	Эра (BC, AD)	AD
Прочее	''	Две одиночные кавычки преобразуются в одну в выходных данных	'

Временные метки эпохи PHP и объекты `DateTime` не поддерживают миллисекунды, поэтому символы `S`, `SS` и `SSS` всегда соответствуют нулям. Если для повторяющихся символов в таблице не указано специальное значение, в большинстве случаев повторение символов приводит к включению начальных нулей в выводимое значение. Например, символ `m` представляет минуты, а со спецификатором `mm` восемь минут выводятся в формате `08`.

Типы аргументов `date` или `time` в сообщении `MessageFormatter` рассчитаны на получение целочисленного аргумента, представляющего количество секунд от начала эпохи, класс `IntlDateFormatter` позволяет задать интересующую вас дату или время другими способами. Форматируемое значение можно задать в формате объекта `DateTime` или массива компонентов времени, возвращаемых функцией `localtime()`. Пример:

```
$fmt = new IntlDateFormatter('en_US', IntlDateFormatter::FULL,
                             IntlDateFormatter::FULL,
                             'America/Chicago');

// Часовой пояс Z соответствует UTC
$obj = new DateTime('2013-08-20T12:34:56Z');
$parts = array('tm_sec' => 56,
               'tm_min' => 34,
               'tm_hour' => 12,
               'tm_mday' => 20,
               'tm_mon' => 7, /* 0 = январь */
               'tm_year' => 113); /* 0 = 1900 */

print $fmt->format($obj) . "\n";
print $fmt->format($parts) . "\n";
```

Результат:

```
Tuesday, August 20, 2013 at 7:34:56 AM Central Daylight Time
Tuesday, August 20, 2013 at 12:34:56 PM Central Daylight Time
```

Значения форматируются по правилам локального контекста `en_US` и часового пояса Чикаго (США). Таким образом, часы, заданные в объекте `DateTime` — полночь GMT, — преобразуются в 7 утра по стандарту Центрального летнего времени (активный часовой пояс для 20 августа 2013 года в Чикаго). Компоненты времени, указанные при втором вызове `format()`, не включают часовой пояс, поэтому считается, что они относятся к часовому поясу, заданному в конструкторе `IntlDateFormatter`.

Начиная с PHP 5.5 в классе `IntlDateFormatter` также появился вспомогательный метод для создания локализованных отформатированных строк даты/времени за один шаг вместо того, чтобы конструировать новый объект `IntlDateFormatter` с последующим вызовом `format()`. Статический метод `IntlDateFormatter::fromObject()` получает три аргумента: объект `DateTime`, формат и локальный контекст и возвращает отформатированную строку даты/времени. Листинг 19.1 показывает, как это происходит.

Листинг 19.1. Объект DateTime, формат и локальный контекст

```
$obj = new DateTime('2013-08-20T12:34:56');
print IntlDateFormatter::formatObject($obj, 'eeee dd MMMM y', 'es_ES') . "\n";
print IntlDateFormatter::formatObject($obj, IntlDateFormatter::FULL, 'fr_FR') .
                                                                        "\n";

// Первый элемент определяет формат даты, второй - формат времени
$formats = array(IntlDateFormatter::FULL, IntlDateFormatter::SHORT);
print IntlDateFormatter::formatObject($obj, $formats, 'de_DE') . "\n";
```

Результат выглядит так:

```
martes 20 agosto 2013
mardi 20 août 2013 12:34:56 UTC
Dienstag, 20. August 2013 12:34
```

Как видно из приведенного примера, во втором аргументе `formatObject` может передаваться явно заданная строка шаблона формата, одна из констант стилей форматирования `IntlDateFormatter` или массив с двумя константами стиля форматирования. Если передается строка с шаблоном, то используется этот шаблон. Если передается одна константа стиля, она используется и для даты, и для времени. Если передается массив с двумя константами стиля форматирования, то первая используется для стиля даты, а вторая — для стиля времени.

См. также

Рецепт 3.1 — описание `localtime()`; документация по классу `IntlDateFormatter`, методам `IntlDateFormatter::format()` и `IntlDateFormatter::formatObject()`. Форматирование даты и времени ICU, включая символы шаблонов форматирования, рассматривается в руководстве «ICU User Guide».

19.4. Локализация числовых данных

Задача

Требуется вывести числовые данные в формате, соответствующем локальному контексту.

Решение

Используйте тип аргумента `number` с `MessageFormatter`:

```
$message = '{0,number} / {1,number} = {2,number}';
$args = array(5327, 98, 5327/98);

$us = new MessageFormatter('en_US',$message);
$fr = new MessageFormatter('fr_FR',$message);
print $us->format($args) . "\n";
```

```
print $fr->format($args) . "\n";
```

Результат:

```
5,327 / 98 = 54.357
5 327 / 98 = 54,357
```

Комментарий

Обратите внимание: для одного сообщения генерируются разные варианты вывода в зависимости от того, на какой локальный контекст настроен класс `MessageFormatter`. Символы, используемые в качестве разделителя групп разрядов и дробной части, определяются локальным контекстом. Приведенные результаты соответствуют форматированию чисел по умолчанию. Добавление дополнительного параметра стиля позволяет изменить это поведение.

Например, существуют простые сокращения для вывода чисел в виде денежных сумм и процентов:

```
$message = '{0,number,currency},  {0,number,percent}';
$us = new MessageFormatter('en_US',$message);
print $us->format(array(3.33333333));
```

Результат:

```
$3.33,  333%
```

Вместо слов `currency` или `percent` также можно указать форматную строку по правилам класса ICU `DecimalFormat`. Многие символы, которые могут использоваться в таких форматных строках, перечислены в таблице 19.2.

Таблица 19.2. Символы шаблонов DecimalFormat

Символ	Описание
0	Цифра
1–9	Цифра с округлением
#	Цифра (для нуля ничего не выводится)
@	Значимая цифра
%	Знак процента, число умножается на 100
¤	Знак денежной единицы
¤¤	Трехбуквенное сокращение денежной единицы
;	Разделитель положительных и отрицательных шаблонов

В следующем коде некоторые из этих символов применяются к разным числам:

```
$args = array(7,159,-0.3782,6.815574);
$messages = array("0", "00", "1", "11", "222",
```

```
                    "#", "##", "@", "@@@",
                    "##%", "¤#", "¤1.11",
                    "¤¤#",
                    "#.##;(#.## !!!)"
                    );

foreach ($messages as $message) {
    $fmt = new MessageFormatter('en_US',"{0,number,$message}\t{1,number,
                                $message}\t"."{2,number,$message}\t
                                {3,number,$message}");
    print "$message:\t" . $fmt->format($args) . "\n";
}
```

Результат выполнения этого кода:

```
0:              7               1590            -0              7
00:             07              159             -00             07
1:              7               159             -0              7
11:             11              154             -00             11
222:            000             222             -000            000
#:              7               159             -0              7
##:             7               159             -0              7
@:              7               200             -0.4            7
@@@:            7.00            159             -0.378          6.82
##%:           700%             15900%          -38%            682%
¤#:            $7              $159            -$0             $7
¤1.11:         $6.66           $158.73         -$0.00          $6.66
¤¤#:            USD7           USD159          -USD0           USD7
#.##;(#.## !!!): 7              159               (0.38 !!!)      6.82
```

Более точное управление числовым форматированием обеспечивает отдельный класс `NumberFormatter`. Его конструктор получает локальный контекст, стиль форматирования и необязательную строку шаблона. Пример:

```
$args = array(7,159,-0.3782,6.815574);

$sci = new NumberFormatter('en_US', NumberFormatter::SCIENTIFIC);
$dur = new NumberFormatter('en_US', NumberFormatter::DURATION);
$ord = new NumberFormatter('en_US', NumberFormatter::ORDINAL);
$pat = new NumberFormatter('en_US', NumberFormatter::PATTERN_DECIMAL, '@@@@');

print $sci->format(10040)    . "\n";
print $dur->format(64)       . "\n";
print $ord->format(15)       . "\n";
print $pat->format(1.357926) . "\n";
```

Результат:

```
1.004E4
1:04
15th
1.358
```

Первый объект, используя стиль `NumberFormatter::SCIENTIFIC`, преобразует 10040 в соответствующую научную запись: `1.004E4`. Второй объект, используя стиль

NumberFormatter::DURATION, преобразует 64 секунды в 1:04 (одна минута и четыре секунды). Третий объект, используя стиль NumberFormatter::ORDINAL, преобразует 15 в 15th. Последний объект со стилем NumberFormatter::PATTERN_DECIMAL использует форматные символы, рассматривавшиеся ранее.

Здесь описана лишь часть возможностей NumberFormatter. Другие возможности более подробно рассмотрены в электронной документации NumberFormatter.

См. также

Рецепт 19.5 — использование NumberFormatter для локализации денежных величин; документация класса NumberFormatter и полный список символов форматов числовых данных ICU.

19.5. Локализация денежных сумм

Задача

Требуется вывести денежные суммы в формате, соответствующем локальному контексту.

Решение

Чтобы применить форматирование по умолчанию, используйте стиль currency для типа аргумента number:

```
$income = 5549.3;
$debit = -25.95;
$fmt = new MessageFormatter('en_US',
                            '{0,number,currency} in and {1,number,currency} out');
print $fmt->format(array($income,$debit));
```

Результат:

```
$5,549.30 in and -$25.95 out
```

Чтобы более точно описать параметры форматирования, используйте метод formatCurrency() класса NumberFormatter:

```
$income = 5549.3;
$debit = -25.95;
$fmt = new NumberFormatter('en_US', NumberFormatter::CURRENCY);
print $fmt->formatCurrency($income, 'USD') . ' in and ' .
   $fmt->formatCurrency($debit, 'EUR') . ' out';
```

Результат:

```
$5,549.30 in and -€25.95 out
```

Комментарий

Стиль `currency` типа аргумента `number` в `MessageFormatter` использует правила форматирования денежных сумм локального контекста экземпляра `MessageFormatter`. Это лаконичный и простой механизм включения денежных величин в выдаваемые сообщения. Результат выполнения кода с `MessageFormatter` выглядит так:

```
$5,549.30 in and -$25.95 out
```

Метод `formatCurrency()` класса `NumberFormatter` упрощает использование других денежных сумм при выводе. В примере с `NumberFormatter` при первом вызове `formatCurrency()` указаны доллары США (`USD`), а при втором — евро (`EUR`), поэтому результат выглядит так:

```
$5,549.30 in and -€25.95 out
```

Хотя числовые форматы, поддерживаемые `MessageFormatter`, позволяют строить сложные правила форматирования денежных величин, часто бывает удобнее выразить эти правила через программный интерфейс, предоставляемый `NumberFormatter`. Пример:

```
$amounts = array( array(152.9, 'USD'),
                  array(328, 'ISK'),
                  array(-1, 'USD'),
                  array(500.53, 'EUR') );

$fmt = new NumberFormatter('en_US', NumberFormatter::CURRENCY);
$fmt->setAttribute(NumberFormatter::PADDING_POSITION,
                   NumberFormatter::PAD_AFTER_PREFIX);
$fmt->setAttribute(NumberFormatter::FORMAT_WIDTH, 15);
$fmt->setTextAttribute(NumberFormatter::PADDING_CHARACTER, ' ');

foreach ($amounts as $amount) {
    print $fmt->formatCurrency($amount[0], $amount[1]) . "\n";
}
```

Фрагмент выводит таблицу с четырьмя значениями в разных валютах; сумма отделяется от знака денежной единицы таким интервалом, чтобы строка имела ширину 15 символов. В качестве символа-заполнителя используется пробел (вместо принятого по умолчанию знака `*`). Результат выглядит так:

```
$        152.90
ISK         328
-$         1.00
€        500.53
```

См. также

Документация по методу `NumberFormatter::formatCurrency()` и различным атрибутам форматирования.

19.6. Локализация графики

Задача

Требуется вывести изображения, содержащие текст. При этом текст должен выводиться на языке локального контекста.

Решение

Создайте отдельный каталог изображений для каждого локального контекста, который вы собираетесь поддерживать, а также глобальный каталог для изображений, не содержащих локализуемого текста. Создайте копии каждого локализуемого изображения в соответствующем специализированном каталоге. Убедитесь в том, что файлы одного изображения в разных каталогах имеют одинаковые имена. Вместо того чтобы напрямую выводить URL-адреса изображений, обрабатывайте их пути как локализуемые строки — либо явно сохраняя их в каталогах изображений, либо вычисляя нужный путь во время выполнения.

Комментарий

Функция `img()` в листинге 19.2 сначала ищет версию изображения, адаптированную для конкретного локального контекста, а затем глобальную версию. Если поиск в обоих случаях не приносит успеха, выводится сообщение в журнал ошибок.

Листинг 19.2. Поиск изображений, адаптированных для локального контекста

```
function img($locale, $f) {
    static $image_base_path = '/usr/local/www/images';
    static $image_base_url  = '/images';

    if (is_readable("$image_base_path/$locale/$f")) {
        return "$image_base_url/$locale/$f";
    } elseif (is_readable("$image_base_path/global/$f")) {
        return "$image_base_url/global/$f";
    } else {
        error_log("l10n error: locale: $locale, image: '$f'");
    }
}
```

Функция `img()` должна знать как путь к файлу изображения в файловой системе (`$image_base_path`), так и путь к изображению от базового URL-адреса сайта (`/images`). По первому пути она проверяет, доступен ли файл для чтения, а по второму строит подходящий URL-адрес изображения.

Локализованное изображение должно храниться во всех каталогах локализации под одним именем. Например, изображение с текстом «New!» на фоне желтой

вспышки должно храниться под именем new.gif как в каталоге images/en_US, так и в каталоге images/es_US, несмотря на то что файл images/es_US/new.gif содержит изображение желтой вспышки с текстом «¡Nuevo!».

Не забудьте, что альтернативный текст в тегах изображений тоже должен быть локализован.

Листинг 19.3 выводит полностью локализованный элемент `<img/>`.

Листинг 19.3. Локализованный элемент <img/>

```
print '<img src="' . img($locale, 'cancel.png') . '" ' .
      'alt="' . $messages[$locale]['CANCEL'] . '"/>';
```

Если локализованные версии некоторого изображения имеют разные размеры, сохраните ширину и высоту изображения в каталоге сообщений. Листинг 19.4 выводит локализованный элемент `<img/>` с атрибутами высоты и ширины.

Листинг 19.4. Локализованный элемент <img/> с высотой и шириной

```
print '<img src="' . img($locale, 'cancel.png') . '" ' .
      'alt="' . $messages[$locale]['CANCEL'] . '" ' .
      'height="' . $messages[$locale]['CANCEL_IMG_HEIGHT'] . '" ' .
      'width="' . $messages[$locale]['CANCEL_IMG_WIDTH'] . '"/>';
```

Локализованные сообщения `CANCEL_IMG_HEIGHT` и `CANCEL_IMG_WIDTH` представляют собой не текстовые строки, а целые числа, описывающие размеры изображения cancel.png в каждом локальном контексте.

Функция `img()` удобна тем, что для поиска нужных файлов она просматривает файловую систему во времени выполнения. Но если коллекция изображений изменяется редко (или ее изменения прогнозируются, например с выходом новой версии программного продукта), возможно, эффективнее будет сохранить в каталоге сообщений сами пути. Для этого вам потребуется выполнить предварительную работу и разобраться в том, какие локальные контексты будут иметь локализованное изображение, а для каких контекстов будет использоваться обобщенная версия.

См. также

Рецепт 19.2 — каталоги сообщений для конкретных локальных контекстов.

19.7. Локализация включаемых файлов

Задача

Требуется включить в страницы файлы, соответствующие определенному локальному контексту.

Решение

Определите локальный контекст и измените значение `include_path`, как показано в листинге 19.5.

Листинг 19.5. Изменение include_path для локализации

```
$base = '/usr/local/php-include';
$locale = 'en_US';

$include_path = ini_get('include_path');
ini_set('include_path',"$base/$locale:$base/global:$include_path");
```

Комментарий

В листинге 19.5 переменная `$base` содержит имя базового каталога для включаемых локализованных файлов. Файлы, не привязанные к конкретному локальному контексту, хранятся в подкаталоге global каталога `$base`, а локализованные файлы — в каталоге, имя которого соответствует локальному контексту (например, `en_US`). Включение каталога локального контекста и глобального каталога в начало списка приводит к тому, что PHP начинает поиск включаемых файлов с этих двух каталогов. Размещение глобального каталога на втором месте гарантирует, что нелокализованная информация будет загружаться только в том случае, если локализованная информация недоступна.

Этот прием напоминает то, что происходило в функции `img()` из Рецепта 19.6. Однако на этот раз мы можем воспользоваться механизмом PHP `include_path`, чтобы поиск в каталоге выполнялся автоматически. Сбрасывайте значение `include_path` как можно раньше в своем коде — желательно в начале файла, загруженного при помощи `auto_prepend_file` при каждом запросе.

См. также

Документация по `include_path` и `auto_prepend_file`.

19.8. Сортировка с учетом локального контекста

Задача

Требуется отсортировать текст с соблюдением правил упорядочения символов конкретного локального контекста.

Решение

Создайте экземпляр объекта `Collator` для своего локального контекста, а затем вызовите его метод `sort()`:

```
$words = array('Малина', 'Клубника', 'Огурец');
$collator = new Collator('ru_RU');
// Выполняет сортировку "на месте", как и sort()
$collator->sort($words);
```

Комментарий

Обычные функции PHP, предназначенные для работы с текстом, интерпретируют строку как простую последовательность байтов. Им ничего не известно о многобайтовых символах, не говоря уже о правилах локального контекста относительно того, какие символы «предшествуют» тем или иным символам в установленной для этого контекста концепции алфавитного порядка. Класс `Collator` использует большую базу данных ICU с информацией о локальных контекстах для решения этой задачи.

Метод `sort()` класса `Collator` является аналогом функции PHP `sort()`. В классе `Collator` также имеется метод `asort()`, который, как и функция PHP `asort()`, сортирует массив с сохранением отношений между ключами и значениями.

См. также

Документация класса `Collator`.

19.9. Управление ресурсами локализации

Задача

Требуется организовать хранение информации о сообщениях и изображениях для разных языков.

Решение

Сохраните справочник сообщений в виде сериализованного массива PHP, связывающего ключи сообщений с сообщениями, соответствующими локальному контексту. Или, если вас интересует совместимость с ICU-совместимым инструментарием или другими языками, воспользуйтесь классом `ResourceBundle`.

Комментарий

По сути, справочник сообщений представляет отношение между ключами и значениями. Справочник сообщений для английского языка может связывать ключ `HELLO_WORLD` со строкой «Hello, World», а испанский — со строкой «Hola, Mundo».

Для работы с информацией таких справочников проще всего рассматривать их как массивы PHP, а затем загружать их из файлов (и сохранять в файлах) как сериализованные массивы. В листинге 19.6 приведена короткая программа, которая определяет справочники сообщений и сохраняет их в файлах.

Листинг 19.6. Сохранение справочников сообщений в виде сериализованных массивов

```
$messages = array();
$messages['en_US'] =
    array('FAVORITE_FOODS' => 'My favorite food is {0}.',
          'FRIES' => 'french fries',
          'CANDY' => 'candy',
          'CHIPS' => 'potato chips',
          'EGGPLANT' => 'eggplant');
$messages['en_GB'] =
    array('FAVORITE_FOODS' => 'My favourite food is {0}.',
          'FRIES' => 'chips',
          'CANDY' => 'sweets',
          'CHIPS' => 'crisps',
          'EGGPLANT' => 'aubergine');

foreach ($messages as $locale => $entries) {
    file_put_contents(__DIR__ . "/$locale.ser", serialize($entries));
}
```

Листинг 19.7 показывает, как загрузить и использовать в программе справочник сообщений, сохраненный кодом листинга 19.6.

Листинг 19.7. Использование справочников сообщений из сериализованных массивов

```
/* Данные могут быть получены от пользователя или браузера */
define('LOCALE', 'en_US');
/* Если полученные данные не заслуживают доверия, добавьте проверку
 * ошибок на случай, если файл не существует или его содержимое
 * не может быть десериализовано . */
$messages = unserialize(file_get_contents(__DIR__ . '/' . LOCALE . '.ser'));

$candy = new MessageFormatter(LOCALE, $messages['CANDY']);
$favs = new MessageFormatter(LOCALE, $messages['FAVORITE_FOODS']);
print $favs->format(array($candy->format(array()))) . "\n";
```

Работа со справочниками сообщений как с сериализованными массивами PHP весьма проста. С другой стороны, этот формат специфичен для PHP. В ICU определяется обобщенный формат обмена данными для обмена данными (такими как справочники сообщений) между разными программами и инструментами — так называемый «пакет ресурсов» (resource bundle). Если вы работаете со средствами локализации или другими языками программирования, поддерживающими пакеты ресурсов ICU, используйте класс `ResourceBundle` для управления ими.

Чтобы создать пакет ресурсов ICU, создайте текстовый файл в правильном формате, а затем обработайте его программой ICU *genrb*, чтобы создать откомпилированную «двоичную» версию пакета. Следующий пример предполагает, что был создан пакет ресурсов ICU со следующим содержимым:

```
en_US {
    FAVORITE_FOODS { "My favorite food is {0}." }
    FRIES { french fries }
    CANDY { candy }
    CHIPS { potato chips }
    EGGPLANT { eggplant }
}
```

Этот пакет ресурсов компилируется программой *genrb* в файл с именем en_US.res. Локальный контекст en_US определяется именем таблицы верхнего уровня в файле. Суффикс .res по умолчанию назначается программой *genrb* всем откомпилированным пакетам ресурсов.

Листинг 19.8 читает содержимое справочника сообщений из пакета и выводит тот же текст, что и листинг 19.7.

Листинг 19.8. Использование справочников сообщений из пакетов ресурсов

```
define('LOCALE', 'en_US');
$bundle = new ResourceBundle(LOCALE, __DIR__);

$candy = new MessageFormatter(LOCALE, $bundle->get('CANDY'));
$favs = new MessageFormatter(LOCALE, $bundle->get('FAVORITE_FOODS'));
print $favs->format(array($candy->format(array()))) . "\n";
```

В листинге 19.8 два аргумента конструктора `ResourceBundle` указывают, как найти правильный откомпилированный пакет ресурсов. Второй аргумент определяет каталог, в котором следует искать файл, а первый аргумент — имя локального контекста, которое обычно совпадает с базовым именем файла. После создания объекта `ResourceBundle` вы можете обращаться к отдельным элементам пакета при помощи метода `get()`. Код, выводящий сообщение `My favorite food is candy.`, почти идентичен листингу 19.7. Отличается только синтаксис чтения строк сообщений из пакета ресурсов (вместо элементов массива).

См. также

Рецепт 19.2 — справочники сообщений; документация класса `ResourceBundle`; руководство по управлению ресурсами ICU, включая синтаксис файлов пакетов ресурсов.

19.10. Выбор кодировки символов для выходных данных

Задача

Требуется убедиться в том, что браузеры правильно обрабатывают текст в кодировке UTF-8, генерируемый вашей программой.

Решение

Задайте конфигурационной директиве PHP `default_encoding` значение `utf-8`. Это гарантирует, что заголовок `Content-Type`, генерируемый PHP для ответов HTML, будет включать фрагмент `charset=utf-8`. С ним браузеры интепретируют содержимое страницы как текст в кодировке UTF-8.

Комментарий

Присваивание `default_encoding` сообщает браузеру, что содержимое страницы должно интерпретироваться в кодировке UTF-8. Тем не менее вы несете ответственность за то, чтобы контент действительно был правильно закодирован в UTF-8, с использованием соответствующих строковых функций. В Рецепте 19.12 подробно объясняется, как это делается.

Если вы не можете изменить конфигурационную директиву `default_encoding`, отправьте соответствующий заголовок `Content-Type` самостоятельно при помощи функции `header()`, как показано в листинге 19.9.

Листинг 19.9. Назначение кодировки символов

```
header('Content-Type: text/html;charset=utf-8');
```

См. также

Рецепт 19.12 — генерирование текста в кодировке UTF-8.

19.11. Назначение кодировки символов для входных данных

Задача

Все данные, поступающие в программу, должны иметь единую кодировку символов, чтобы они правильно обрабатывались программой. Например, все входящие данные, отправляемые формами, должны интерпретироваться в кодировке UTF-8.

Решение

Невозможно гарантировать, что браузеры подчинятся полученным инструкциям по поводу кодировки, но если принять некоторые меры, добропорядочные браузеры будут «играть по правилам».

Во-первых, выполните инструкции из Рецепта 19.10, чтобы ваши программы сообщали браузерам о выдаче текста в кодировке UTF-8. Заголовок `Content-Type`

с указанием кодировки подсказывает браузеру, что в отправляемых формах должна использоваться кодировка, указанная в заголовке.

Во-вторых, включите атрибут `accept-charset="utf-8"` в выводимые элементы `<form/>`. Этот атрибут (хотя он поддерживается не всеми браузерами) подсказывает браузеру закодировать пользовательские данные в кодировке UTF-8, прежде чем отправлять ее серверу.

Комментарий

В общем случае браузеры отправляют данные форм в той кодировке, которая использовалась для генерирования страницы, содержащей форму. Следовательно, установив стандарт вывода в кодировке UTF-8, вы можете рассчитывать на то, что всегда будете получать ввод в UTF-8. Атрибут `accept-charset` элемента `<form/>` является частью спецификации HTML 4.0, но реализован не везде.

См. также

Рецепт 19.10 — отправка вывода в кодировке UTF-8; описание атрибута `accept-charset` элемента `<form/>` на сайте W3C (*http://www.w3.org/TR/REC-html40/interact/forms.html#adef-accept-charset*).

19.12. Работа с текстом в кодировке UTF-8

Задача

Требуется работать с текстом в кодировке UTF-8 в программах, например правильно вычислять длину строк с многбайтовыми символами и следить за тем, чтобы весь текст выводился в виде символов в кодировке UTF-8.

Решение

Используйте различные функции PHP для выполнения операций, необходимых для совместимости с UTF-8.

Если расширение mbstring доступно, используйте его строковые функции для операций со строками UTF-8. В листинге 19.10 функция `mb_strlen()` используется для вычисления количества символов в двух строках в кодировке UTF-8.

Листинг 19.10. Применение функции mb_strlen()

```
// Определение кодировки
mb_internal_encoding('UTF-8');
// Символ ö представляется двумя байтами
$name = 'Kurt Gödel';
```

```
// Каждый символ хангыль представляется тремя байтами
$dinner = '불고기';
$name_len_bytes = strlen($name);
$name_len_chars = mb_strlen($name);

$dinner_len_bytes = strlen($dinner);
$dinner_len_chars = mb_strlen($dinner);

print "$name is $name_len_bytes bytes and $name_len_chars chars\n";
print "$dinner is $dinner_len_bytes bytes and $dinner_len_chars chars\n";
```

Код листинга 19.10 выводит следующий результат:

```
Kurt Gödel is 11 bytes and 10 chars
불고기 is 9 bytes and 3 chars
```

Расширение iconv также предоставляет функции для выполнения операций со строками, использующими многобайтовые кодировки (листинг 19.11).

Листинг 19.11. Использование функций iconv

```
// Определение кодировки
iconv_set_encoding('internal_encoding','UTF-8');
// Символ ö представляется двумя байтами
$name = 'Kurt Gödel';
// Каждый символ хангыль представляется тремя байтами
$dinner = '불고기';

$name_len_bytes = strlen($name);
$name_len_chars = iconv_strlen($name);

$dinner_len_bytes = strlen($dinner);
$dinner_len_chars = iconv_strlen($dinner);

print "$name is $name_len_bytes bytes and $name_len_chars chars\n";
print "$dinner is $dinner_len_bytes bytes and $dinner_len_chars chars\n";

print "The seventh character of $name is " . iconv_substr($name,6,1) . "\n";
print "The last two characters of $dinner are " . iconv_substr($dinner,-2);
```

Необязательный третий аргумент таких функций, как `htmlentities()` и `htmlspecialchars()`, приказывает им интерпретировать входные данные в кодировке UTF-8. Пример представлен в листинге 19.12.

Листинг 19.12. Работа с HTML в кодировке UTF-8

```
$encoded_name = htmlspecialchars($_POST['name'], ENT_QUOTES, 'UTF-8');
$encoded_dinner = htmlentities($_POST['dinner'], ENT_QUOTES, 'UTF-8');
```

Комментарий

Постоянная бдительность — цена, которую приходится платить за удобства кодировок. Если вы выполнили инструкции Рецептов 19.10 и 19.11, то данные,

получаемые вашей программой, должны определяться в кодировке UTF-8, а браузеры будут обрабатывать выходные данные программы как имеющие кодировку UTF-8. Таким образом, вам остаются две обязанности: выполнение операций со строками с учетом кодировки UTF-8 и генерирование текста в кодировке UTF-8.

Первая обязанность заметно упрощается, когда вы примете основную истину интернационализации: символ — это не байт. У этой аксиомы имеется одно важное следствие в области PHP: строковые функции PHP работают с байтами, а не с символами. Например, функция `strlen()` подсчитывает количество байтов в строке, а не количество символов. В доисторические времена кодировки ISO-8859-1 это не создавало проблем — каждый из 256 символов кодировки занимал один байт. С другой стороны, для представления символа UTF-8 используется от 1 до 4 байтов. Расширения mbstring и iconv предоставляют альтернативные версии некоторых строковых функций, работающие с символами, а не с байтами. Эти функции перечислены в таблице 19.3.

Таблица 19.3. Функции, работающие с символами

Обычная функция	Функция mbstring	Функция iconv
strlen()	mb_strlen()	iconv_strlen()
strpos()	mb_strpos()	iconv_strpos()
strrpos()	mb_strrpos()	iconv_strrpos()
substr()	mb_substr()	iconv_substr()
strtolower()	mb_strtolower()	—
strtoupper()	mb_strtoupper()	—
substr_count()	mb_substr_count()	—
ereg()	mb_ereg()	—
eregi()	mb_eregi()	—
ereg_replace()	mb_ereg_replace()	—
eregi_replace()	mb_eregi_replace()	—
split()	mb_split()	—
mail()	mb_send_mail()	—

Для правильной работы расширения mbstring необходимо приказать ему использовать кодировку UTF-8. Это можно сделать из кода сценария, как в листинге 19.19, — при помощи функции `mb_internal_encoding()`. Также нужный режим можно включить на общесистемном уровне, присвоив конфигурационной директиве `mbstring.internal_encoding` значение `UTF-8`.

Похожие требования действуют и для расширения iconv: используйте функцию `iconv_set_encoding()`, как в листинге 19.11, или задайте значение конфигурационной директивы `iconv.internal_encoding`.

Расширение mbstring также предоставляет альтернативные версии для функций семейства ereg, работающих с регулярными выражениями. Впрочем, вы всегда можете использовать строки UTF-8 с функциями регулярных выражений PCRE (`preg_*()`). Модификатор `u` сообщает функции `preg`, что строка шаблона определена в кодировке UTF-8, а также позволяет использовать различные свойства Юникода в шаблонах. В листинге 19.13 свойство Юникода используется для подсчета количества букв нижнего регистра в каждой из двух строк.

Листинг 19.13. Регулярные выражения в кодировке UTF-8

```
$name = 'Kurt Gödel';
$dinner = '불고기';
$name_lower = preg_match_all('/\p{Ll}/u',$name,$match);
$dinner_lower = preg_match_all('/\p{Ll}/u',$dinner,$match);
print "There are $name_lower lowercase letters in $name.\n";
print "There are $dinner_lower lowercase letters in $dinner.\n";
```

Код листинга 19.13 выводит следующий результат:

```
There are 7 lowercase letters in Kurt Gödel.
There are 0 lowercase letters in 불고기.
```

Другие функции упрощают преобразования между другими кодировками и UTF-8. Функции `utf8_encode()` и `utf8_decode()` преобразуют строки между кодировками ISO-8859-1 и UTF-8. Так как кодировка ISO-8859-1 используется по умолчанию во многих ситуациях, эти функции удобны для обеспечения совместимости с данными, изначально не поддерживающими UTF-8. Например, в словарях, используемых расширением pspell, данные часто кодируются в ISO-8859-1. В листинге 19.14 функция `utf8_encode()` преобразует результат `pspell_suggest()` в строку в кодировке UTF-8.

Листинг 19.14. Применение кодировки UTF-8 к строкам ISO-8859-1

```
$lang = isset($_GET['lang']) ? $_GET['lang'] : 'en';
$word = isset($_GET['word']) ? $_GET['word'] : 'asparagus';

$ps = pspell_new($lang);
$check = pspell_check($ps, $word);

print htmlspecialchars($word,ENT_QUOTES,'UTF-8');
print $check ? ' is ' : ' is not ';
print ' found in the dictionary.';
print '<hr/>';

if (! $check) {
    $suggestions = pspell_suggest($ps, $word);
    if (count($suggestions)) {
        print 'Suggestions: <ul>';
        foreach ($suggestions as $suggestion) {
            $utf8suggestion = utf8_encode($suggestion);
            $safesuggestion = htmlspecialchars($utf8suggestion,
                                               ENT_QUOTES,'UTF-8');
```

```
            print "<li>$safesuggestion</li>";
        }
        print '</ul>';
    }
}
```

Возможно, задачу управления кодировкой будет проще понять, если рассматривать ее как аналог кодирования сущностей HTML. В обоих случаях текст требуется обработать и перевести в формат, соответствующий определенному контексту. С кодированием сущностей обычно подразумевается обработка данных, полученных из внешнего источника, функцией `htmlentities()` или `htmlspecialchars()`. В случае кодировок все данные преобразуются в UTF-8 перед обработкой, для строковых операций используются символьные функции, и перед выводом строк следует проверить, что они закодированы в UTF-8.

См. также

Рецепты 19.10 и 19.11 — получение и отправка строк в кодировке UTF-8; документация расширений mbstring и iconv; документация по функциям `htmlentities()` и `htmlspecialchars()`; синтаксис шаблонов PCRE; документация по функциям `utf8_encode()` и `utf8_decode()`.

Среди полезных источников информации по кодировкам символов и PHP можно порекомендовать:

- Статья «Character Sets/Character Encoding Issues» в вики PHP WACT (*http://www.phpwact.org/php/i18n/charsets*).
- «Characters vs. Bytes», автор Тим Брей (Tim Bray) (*http://www.tbray.org/ongoing/When/200x/2003/04/26/UTF*).
- «A Tutorial on Character Code Issues», автор Юкка Корпела (Jukka Korpela) (*http://www.cs.tut.fi/~jkorpela/chars.html*).

20 Обработка ошибок

20.0. Введение

Людей, которые проводят свое время за разработкой веб-приложений, называют программистами, но это название обманчиво: на самом деле бóльшая часть времени тратится на *отладку* приложения. Чем бы вы ни занимались — исправлением опечаток или переработкой фрагментов кода, неэффективно работающих в среде с высокой нагрузкой, — вам наверняка придется провести большую часть рабочего времени за отладкой и тестированием... и снова за отладкой и тестированием. А потом еще и еще.

Скорее всего, о лихорадочных, затягивающихся на всю ночь сеансах отладки в описании вашей работы ничего не говорилось — да и кто бы добровольно подписался на такие развлечения? Тем не менее ошибки, отладка и тестирование являются частью жизни программиста. И если вы заранее вооружитесь эффективными средствами и методами отладки, то время отладки сведется к минимуму и у вас останется больше времени для творческой работы.

К сожалению, многие разработчики не желают тратить время на совершенствование навыков обработки ошибок, отладки и тестирования; не повторяйте их ошибку. Применяя то, что принято называть *пессимистическим программированием*, вы начнете заранее планировать возможные проблемы, а ваше приложение будет готово корректно обработать их, когда эти проблемы возникнут.

Эта глава посвящена ошибкам: поиску источников ошибок, получению информации о текущей ситуации при возникновении ошибки, сокрытию ошибок от пользователей и их регистрации в журнале, чтобы вы могли провести осмысленную процедуру отладки после их возникновения. Глава 21 дополняет материал информацией об использовании отладчика в PHP и написании тестов.

20.1. Поиск и исправление ошибок разбора

Задача

Сценарий PHP не работает из-за фатальной синтаксической ошибки. Требуется быстро найти источник проблемы и продолжить программирование.

Решение

Проверьте строку, указанную в сообщении интерпретатора PHP. Если эта строка не содержит ошибок, двигайтесь в обратном направлении, пока не найдете проблемную строку.

Другой вариант — используйте среду разработки с поддержкой PHP, которая сообщает о синтаксических ошибках во время написания кода. Это поможет выявлять ошибки прямо во время написания кода.

Комментарий

Интерпретатор PHP, как и большинство других языков программирования, чрезвычайно придирчиво относится к сценарному коду. Даже при малейших неточностях интерпретатор PHP останавливает разбор кода и сообщает о возникших проблемах.

Возьмем следующую программу:

```
<?php
if isset($user_firstname) {
  print "Howdy, $user_firstname!";
} else {
  print "Howdy!";
}
```

Сохраните код в файле howdy.php и запустите программу. PHP выводит сообщение об ошибке:

```
Parse error: syntax error, unexpected T_ISSET, expecting '(' in ⏎
/var/www/howdy.php on line 2
```

Судя по сообщению, обнаружена ошибка в строке 2, а точнее, синтаксическая ошибка: что-то о неожиданном `T_ISSET`.

Когда PHP обрабатывает сценарии для преобразования их в формат, понятный компьютеру, каждая строка разбивается на фрагменты, называемые *лексемами*. PHP распознает десятки лексем и знает, какие лексемы и в каком порядке могут встречаться в строке кода PHP. В указанной ошибке упоминание `T_ISSET` означает, что интерпретатор PHP встретил лексему `T_ISSET` там, где ее не должно быть.

Если прочитать сообщение об ошибке чуть дальше, становится ясно, что интерпретатор PHP ожидал увидеть символ (там, где была обнаружена лексема `T_ISSET`. Возвращаемся к строке 2 программы — и действительно, между `if` и вызовом функции `isset()` отсутствует круглая скобка.

Некоторые редакторы с поддержкой синтаксиса PHP могут сообщать о таких проблемах еще до того, как программа перейдет на стадию выполнения кода и интерпретатор сообщит об ошибке. На рис. 20.1 показана программа с ошибкой в среде разработки NetBeans с опережающим предупреждением о будущей ошибке.

Рис. 20.1. Среда разработки NetBeans обнаруживает синтаксическую ошибку еще до ее возникновения

Проблема не всегда решается простым переходом к строке, указанной в сообщении. Иногда ошибка, допущенная несколькими строками ранее, создает проблему, которая не была обнаружена сразу, а проявилась только в контексте строки, указанной в сообщении.

Если у вас возникнут проблемы с поиском источника ошибки, а диагностические инструменты для ее выявления недоступны, помните: если все остальное не ра-

ботает — комментарии вам помогут. Попробуйте закомментировать блоки кода, предшествующие строке из сообщения об ошибке, а затем снова запустите сценарий с ошибкой. Повторяя процесс исключения, вы рано или поздно найдете строку, которая стала причиной ошибки.

См. также

Список лексем парсера PHP (*http://php.net/tokens*).

20.2. Создание классов исключений

Задача

Требуется управлять тем, какая информация об ошибках должна выводиться для пользователя, даже если вы работаете со сторонней библиотекой, которая использует собственный подход к обработке ошибок.

Решение

Используйте поддержку *исключений* PHP 5 и создайте собственный обработчик ошибок, который сделает все необходимое при возникновении ошибок в сторонней библиотеке:

```
class CustomException extends Exception {
    public function __construct($e) {
        // Обеспечить правильную инициализацию
        parent::__construct($e->getMessage(), $e->getCode());

        // Зарегистрировать имеющуюся информацию в журнале
        $msg = "-----------------------------------------------\n";
        $msg .= __CLASS__ . ": [{$this->code}]: {$this->message}\n";
        $msg .= $e->getTraceAsString() . "\n";
        error_log($msg);
    }

    // Перегрузить метод __toString() для подавления
    // "нормального" вывода
    public function __toString() {
        return $this->printMessage();
    }

    // Связать коды ошибок с выходными сообщениями или шаблонами
    public function printMessage() {

        $usermsg = '';
        $code = $this->getCode();

        switch ($code) {
```

```
            case SOME_DEFINED_ERROR_CODE:
                $usermsg = 'Ooops! Sorry about that.';
                break;
            case OTHER_DEFINED_ERROR_CODE:
                $usermsg = "Drat!";
                break;
            default:
                $usermsg = file_get_contents('/templates/general_error.html');
                break;
            }
            return $usermsg;
        }

        // Статический обработчик ошибок по умолчанию
        public static function exception_handler($exception) {
            throw new CustomException($exception);
        }

}

// Обеспечить перехват всех исключений
set_exception_handler('CustomException::exception_handler');

try {
    $obj = new CoolThirdPartyPackage();
} catch (CustomException $e) {
    echo $e;
}
```

Комментарий

В версии PHP 5 в PHP была добавлена поддержка исключений — стандартного механизма, существующего во многих других языках программирования. Исключения обеспечивают корректную обработку непредвиденных ошибочных ситуаций. Они особенно полезны при включении в сценарии кода, работающего со сторонними библиотеками, когда у вас нет полной уверенности в том, как поведет себя код в непредвиденных обстоятельствах, например при потере подключения к базе данных, отсутствии ответа от удаленного сервера или в других аналогичных ситуациях.

Для обработки исключений в сценарий включается структура `try/catch`. Она определяет изолированный блок кода, в котором могут возникнуть проблемы без вреда для остального кода:

```
try {
  // Выполнить операцию
  $obj = new CoolThing();
} catch (CustomException $e) {
  // Что-то пошло не так
  print $e;
}
```

Зачем использовать пользовательские исключения, если в PHP 5 уже имеется вполне работоспособный класс исключений? Дело в том, что класс исключений по умолчанию не обеспечивает *корректной* обработки непредсказуемых результатов. Он просто выводит сообщение об ошибке, которое мало чем отличается от обычных сообщений об ошибках. Если вы хотите организовать более гибкую обработку этих прискорбных событий, пользовательский обработчик события позволит выполнить те действия, которые на ваш взгляд являются наиболее уместными в данной ситуации.

В примере с `CustomException` мы преследуем две цели: сохранить информацию обо всем происшедшем и сделать так, чтобы с точки зрения пользователя все выглядело более или менее прилично.

Метод `__construct()` инициализирует исключение вызовом конструктора родительского класса (то есть конструктора класса исключения по умолчанию); тем самым гарантируется, что все возможные значения будут готовы к использованию методами пользовательского исключения.

Затем вся информация об ошибке сохраняется вызовом `error_log()`, который можно заменить любой другой функцией на ваш выбор. Стремясь к корректной обработке ошибки, мы должны позаботиться о том, чтобы функция могла сохранить информацию, не породив другой ошибки. Например, если ошибка связана со сбоем подключения к базе данных, наверное, не стоит сохранять информацию о ней в таблице на том же сервере баз данных.

Далее класс `CustomException` пишется в расчете на то, что информация об ошибке будет выведена на стороне вызова. Тем не менее такое поведение не является обязательным. С таким же успехом можно использовать блок `try/catch` следующего вида:

```
try {
  // Выполнить операцию
  $obj = new CoolThing();
} catch (CustomException $e) {
  // Что-то пошло не так
  $e->redirectToOhNoPage();
}
```

Сегмент `catch (CustomException $e)` означает, что экземпляр класса `CustomException` будет создан и присвоен переменной `$e`. Далее `$e` содержит объект с заранее определенными значениями и методами, которые связаны с проблемой, породившей исключение, но в остальном это совершенно обычный объект — настолько простой или сложный, каким вы его сделаете.

Одно из основных отличий между стандартным обработчиком ошибок и исключениями — концепция *восстановления*. Ситуация, представленная в этом примере, сильно напоминает использование функции `set_error_handler()`, с которой вы, вероятно, уже знакомы. В пользовательский обработчик включается процедура «зачистки», которая проверяет состояние приложения на момент перехвата исключения, по возможности приводит его в порядок и корректно завершает работу.

Исключения также могут использоваться для восстановления работоспособности приложения в процессе его выполнения. Например, блок `try` может иметь несколько блоков `catch`; такая конструкция выглядит аккуратнее, чем серия блоков `if/else/else/else`:

```
try {
    // Выполнить операцию
    $obj = new CoolThing();
} catch (PossibleException $e) {
    // Возможная проблема
    print "<!-- caught exception $e! -->";
    $obj = new PlanB();
} catch (AnotherPossibleException $e) {
    // И такая возможность тоже была предусмотрена
    print "<!-- aha! caught exception $e -->";
    $obj = new PlanC();
} catch (CustomException $e) {
    // Если все остальное не помогло, перейти к зачистке и завершению
    $e->cleanUp();
    $e->bailOut();
}
```

В этом примере структура `try/catch` используется для обнаружения исключительных ситуаций без прерывания выполнения. Если ни одна из мер, которые можно было предусмотреть заранее, не подходит, остается вариант с выходом в обработчик любых исключений. В блоке `catch` даже можно инициировать новое исключение, чтобы повлиять на порядок появления исключений в блоке `try/catch`, в который может быть заключен текущий выполняемый фрагмент кода.

См. также

Рецепт 20.9 — регистрация ошибок; документация по исключениям.

20.3. Вывод трассировки стека

Задача

Требуется узнать, что происходит в конкретной точке вашей программы и что произошло на пути к этой точке.

Решение

Воспользуйтесь функцией `debug_print_backtrace()`:

```
function stooges() {
  print "woo woo woo!\n";
```

```
  larry();
}

function larry() {
  curly();
}

function curly() {
  moe();
}

function moe() {
  debug_print_backtrace();
}

stooges();
```

Результат выглядит так:

```
woo woo woo!
#0  moe() called at [backtrace.php:12]
#1  curly() called at [backtrace.php:8]
#2  larry() called at [backtrace.php:4]
#3  stooges() called at [backtrace.php:19]
```

Комментарий

Удобная функция `debug_print_backtrace()` позволяет быстро получить представление о том, что происходило в приложении непосредственно перед вызовом конкретной функции.

Чем сложнее ваше приложение, тем больше информации вернет функция трассировки. При отладке большой кодовой базы вы быстрее добьетесь успеха с помощью полноценного отладочного расширения (такого как Xdebug) или интегрированной среды разработки (IDE) с поддержкой точек прерывания, пошагового выполнения, наблюдения за изменениями переменных и т. д., например NetBeans.

Если вам требуется лишь немного больше информации, чем можно получить от вставки в код команд `'Выполняется строка ' . LINE;`, функция `debug_print_backtrace()` хорошо подойдет для ваших целей.

Вывод `debug_print_backtrace()` по умолчанию включает аргументы, переданные каждой функции. Если в аргументах передаются большие массивы или сложные объекты, вывод становится слишком громоздким. Если передать в первом аргументе `debug_print_backtrace()` константу `DEBUG_BACKTRACE_IGNORE_ARGS`, то аргументы исключаются из вывода. Если вас интересует только последовательность вызовов функций, это идеальный вариант.

У функции `debug_print_backtrace()` имеется парная функция `debug_backtrace()`. Вместо того чтобы выводить данные трассировки, `debug_backtrace()` возвращает их в виде массива, содержащего один элемент на кадр стека. Это может быть

полезно, если вы собираетесь вывести отдельные элементы трассировочных данных или хотите работать с ними на программном уровне. В листинге 20.1 информация из `debug_backtrace()` используется для вывода ограниченных данных трассировки, состоящих только из имен функций и классов.

Листинг 20.1. Использование debug_backtrace()

```
function print_parsed_backtrace() {
    $backtrace = debug_backtrace();
    for ($i = 1, $j = count($backtrace); $i < $j; $i++) {
        $frame = $backtrace[$i];
        if (isset($frame['class'])) {
            $function = $frame['class'] . $frame['type'] . $frame['function'];
        } else {
            $function = $frame['function'];
        }
        print $function . '()';
        if ($i != ($j - 1)) {
            print ', ';
        }
    }
}

function stooges() {
  print "woo woo woo!\n";
  Fine::larry();
}

class Fine {
    static function larry() {
        $brothers = new Howard;
        $brothers->curly();
    }
}
class Howard {
    function curly() {
        $this->moe();
    }
    function moe() {
        print_parsed_backtrace();
    }
}

stooges();
```

Результат:

```
woo woo woo!
Howard->moe(), Howard->curly(), Fine::larry(), stooges()
```

В листинге 20.1 вызывается цепочка функций, одна из которых вызывает `print_parsed_backtrace()`. Затем эта функция получает информацию о кадрах стека от `debug_backtrace()` и перебирает их. Цикл `for()` начинается с 1, а не с 0, потому

что первый кадр стека (элемент 0) относится к вызову самой функции `print_parsed_backtrace()`. Если кадр стека соответствует вызову метода, то элемент `class` массива содержит имя класса метода, а элемент `type` содержит `::` для вызова статического метода или `->` для вызова метода экземпляра. Если элемент `class` отсутствует, значит, это вызов обычной функции.

См. также

Документация по функциям `debug_backtrace()` и `debug_print_backtrace()`; NetBeans.

20.4. Чтение конфигурационных переменных

Задача

Требуется получить значение параметра конфигурации PHP.

Решение

Воспользуйтесь функцией `ini_get()`:

```
// Получение пути поиска файлов
$include_path = ini_get('include_path');
```

Комментарий

Чтобы получить значения всех переменных конфигурации за один шаг, вызовите функцию `ini_get_all()`. Функция возвращает переменные в формате ассоциативного массива, каждый элемент которого также является ассоциативным массивом. Второй массив состоит из трех элементов: глобального значения, локального значения и уровня доступа:

```
// Сохранение всех параметров конфигурации в ассоциативном массиве
$vars = ini_get_all();
print_r($vars['date.timezone']);
```

Результат:

```
Array
(
    [global_value] => UTC
    [local_value] => UTC
    [access] => 7
)
```

Значение `global_value` задается в файле php.ini; значение `local_value` учитывает все изменения, внесенные в файле конфигурации веб-сервера, во всех актуальных файлах .htaccess и текущем сценарии. Значение `access` представляет собой числовую константу, описывающую возможности изменения этого значения. Значения `access` представлены в таблице 20.1. Следует учесть, что имя `access` выбрано не совсем удачно, потому что значение всегда можно проверить, но не всегда можно изменять.

Таблица 20.1. Значения уровня доступа

Значение	Константа PHP	Смысл
1	PHP_INI_USER	Любой сценарий, вызов ini_set()
2	PHP_INI_PERDIR	Уровень каталога, с использованием .htaccess
4	PHP_INI_SYSTEM	Системный уровень, с использованием php.ini или httpd.conf
7	PHP_INI_ALL	Любое место: сценарии, каталоги, система

Значение 6 означает, что параметр может быть изменен как на уровне каталога, так и на уровне системы: 2+4=6. На практике не существует переменных, которые могут изменяться только в `PHP_INI_USER` или `PHP_INI_PERDIR`, и все переменные могут изменяться в `PHP_INI_SYSTEM`, поэтому используются значения 4, 6 и 7.

Чтобы получить переменные, принадлежащие конкретному расширению, передайте имя расширения `ini_get_all()`:

```
// Вернуть переменные сеансового модуля
$session = ini_get_all('session');
```

По правилам имена переменных расширений снабжаются префиксом из имени расширения и точки. Так, например, все сеансовые переменные начинаются с `session.`, а переменные PDO — с `pdo`.

Поскольку `ini_get()` возвращает текущее значение конфигурационной директивы, для проверки исходного значения из файла php.ini следует использовать функцию `get_cfg_var()`:

```
$original = get_cfg_var('sendmail_from'); // Адрес изменился?
```

Значение, возвращаемое `get_cfg_var()`, совпадает со значением элемента `global_value` массива, возвращаемого функцией `ini_get_all()`.

См. также

Рецепт 20.5 — присваивание значений конфигурационным переменным; документация по функциям `ini_get()`, `ini_get_all()` и `get_cfg_var()`; полный список конфигурационных переменных, их значений по умолчанию и возможностей их изменения.

20.5. Присваивание значений конфигурационным переменным

Задача

Требуется изменить значение параметра конфигурации PHP.

Решение

Воспользуйтесь функцией `ini_set()`:

```
// Добавление каталога в путь поиска
ini_set('include_path', ini_get('include_path') . ':/home/fezzik/php');
```

Комментарий

Функция `ini_set()` не вносит постоянные изменения в конфигурационные переменные. Новое значение действует только в течение обработки запроса, для которого была вызвана функция `ini_set()`. Чтобы внести долгосрочные изменения, измените значения, хранящиеся в файле php.ini.

Изменение некоторых переменных (например, `asp_tags`) не имеет смысла, потому что к моменту вызова `ini_set()` изменять поведение, на которое влияет эта переменная, уже поздно. Если переменную нельзя изменить, `ini_set()` возвращает `false`.

Тем не менее в некоторых страницах изменение конфигурационных переменных может принести пользу. Например, если вы запустили сценарий из командной строки, присвойте `html_errors` значение `off`.

Чтобы вернуть исходное значение переменной, вызовите функцию `ini_restore()`:

```
ini_restore('sendmail_from'); // Вернуться к значению по умолчанию
```

См. также

Рецепт 20.4 — чтение значений конфигурационных переменных; документация по функциям `ini_set()` и `ini_restore()`.

20.6. Сокрытие сообщений об ошибках от пользователей

Задача

Сообщения об ошибках PHP не должны быть видны пользователям.

Решение

Включите следующие значения в файл php.ini или конфигурационный файл веб-сервера:

```
display_errors =off
log_errors     =on
```

Если серверный файл php.ini недоступен, эти значения можно задать вызовами `ini_set()`:

```
ini_set('display_errors', 'off');
ini_set('log_errors', 'on');
```

Эти настройки приказывают PHP не выводить информацию об ошибках в виде разметки HTML в браузере, а сохранять ее в журнале ошибок сервера.

Комментарий

При включении режима `log_errors` сообщения об ошибках записываются в журнал ошибок сервера. Если вы хотите, чтобы информация об ошибках PHP записывалась в отдельный файл, укажите имя файла в конфигурационной директиве `error_log`:

```
error_log   = /var/log/php.error.log
```

или:

```
ini_set('error_log', '/var/log/php.error.log');
```

Если в директиве `error_log` выбрано значение `syslog`, сообщения об ошибках PHP передаются системному регистратору вызовом `syslog(3)` в Unix, или в журнал событий в Windows. Если значение `error_log` не задано, то сообщения об ошибках сохраняются в стандартном месте — обычно в журнале ошибок веб-сервера. (Для программ командной строки PHP по умолчанию информация направляется в стандартный выходной поток ошибок.)

Многие сообщения об ошибках должны быть видны пользователям (например, сообщения о неправильном заполнении формы), однако пользователей следует оградить от внутренних ошибок, отражающих проблемы в вашем коде. Во-первых, такие сообщения об ошибках выглядят непрофессионально для опытных пользователей и сбивают с толку новичков. Если при сохранении ввода с формы в базе данных что-то пошло не по плану, проверьте код, возвращенный запросом базы данных, и выведите сообщение для пользователя с извинениями и предложением повторить попытку позднее. Вывод невразумительных сообщений прямо из PHP не прибавит уверенности пользователям вашего сайта.

Во-вторых, вывод сообщений об ошибках создает угрозу для безопасности. В зависимости от базы данных и типа ошибки сообщение может содержать информацию о том, как подключиться к вашей базе данных или серверу, и о структуре

данных. Злоумышленники могут воспользоваться этой информацией для организации атаки на сайт.

Например, если сервер базы данных недоступен и вы пытаетесь подключиться к нему вызовом `mysql_connect()`, PHP выдаст следующее предупреждение:

```
<br>
<b>Warning</b>:  Can't connect to MySQL server on 'db.example.com' (111) in
<b>/www/docroot/example.php</b> on line <b>3</b><br>
```

Если сообщение попадет в браузер пользователя, то пользователь узнает, что сервер базы данных находится по адресу *db.example.com*, и может направить усилия на его взлом.

См. также

Рецепт 20.9 — регистрация ошибок; Рецепт 20.5 — настройка параметров конфигурации вызовом `ini_set()`; документация по конфигурационным директивам PHP.

20.7. Настройка обработки ошибок

Задача

Требуется изменить «чувствительность» некоторой страницы к ошибкам. Это позволит вам управлять тем, о каких типах ошибок будет сообщать страница.

Решение

Для настройки типов ошибок, о которых будет сообщать PHP, используйте функцию `error_reporting()`:

```
error_reporting(E_ALL);                // Все ошибки
error_reporting(E_ERROR | E_PARSE);    // Только серьезные ошибки
error_reporting(E_ALL & ~E_NOTICE);    // Все, кроме уведомлений
```

Комментарий

С каждой сгенерированной ошибкой связывается определенный тип. Например, если вы попытаетесь вызвать `array_pop()` для строки, PHP сообщит, что аргумент должен быть массивом. С этим сообщением связан тип ошибки `E_NOTICE` — некритичная ошибка времени выполнения.

По умолчанию используется уровень сообщений об ошибках `E_ALL&~E_NOTICE`, что соответствует всем типам ошибок, кроме уведомлений. Знак `&` представляет логическую операцию И, а знак `~` — логическую операцию НЕ. Однако в конфи-

гурационном файле php.ini назначается уровень сообщений об ошибках E_ALL, что соответствует всем типам ошибок.

В PHP 5.0 появился новый уровень сообщений об ошибках E_STRICT. При включенном режиме E_STRICT во время разработки PHP будет сообщать о различных возможностях усовершенствования вашего кода. Вы будете получать предупреждения об использовании устаревших функций, а также рекомендации по поводу передовых методов программирования. Для PHP 5.0–5.3 режим E_STRICT является единственным уровнем, не включаемым в E_ALL; для получения максимально полной информации об ошибках в ходе разработки выберите значение E_ALL|E_STRICT. Начиная с PHP 5.4 режим E_STRICT включается в E_ALL.

Сообщения об ошибках, относящиеся к категории «уведомлений», указывают на проблемы времени выполнения, которые считаются менее серьезными по сравнению с предупреждениями. Они свидетельствуют даже не о реальных, а о потенциальных проблемах. Например, к категории E_NOTICE относится сообщение «Undefined variable», которое возникает при попытке использования переменной без предварительного присваивания значения:

```
// Генерирует E_NOTICE
foreach ($array as $value) {
    $html .= $value;
}

// Сообщения об ошибках не генерируются
$html = '';
foreach ($array as $value) {
    $html .= $value;
}
```

В первом случае при первом выполнении foreach переменная $html не определена, поэтому при присоединении данных PHP сообщает об использовании неопределенной переменной. Во втором случае $html перед циклом присваивается пустая строка, чтобы избежать выдачи уведомления E_NOTICE. Два приведенных фрагмента генерируют идентичный код, потому что по умолчанию переменнаая инициализируется пустой строкой. С другой стороны, уведомление E_NOTICE может быть полезным, например при опечатке в имени переменной:

```
foreach ($array as $value) {
    $hmtl .= $value; // А должно быть $html !
}

$html = '';
foreach ($array as $value) {
    $hmtl .= $value; // А должно быть $html !
}
```

Пользовательская функция обработки ошибок может анализировать ошибки по типу и предпринимать соответствующие действия. Полный список типов ошибок приведен в таблице 20.2.

Таблица 20.2. Типы ошибок

Значение	Константа	Описание	Возможность перехвата
1	E_ERROR	Неустранимая ошибка	Нет
2	E_WARNING	Исправимая ошибка	Да
4	E_PARSE	Ошибка парсера	Нет
8	E_NOTICE	Потенциальная ошибка	Да
16	E_CORE_ERROR	Аналогично E_ERROR, но генерируется ядром PHP	Нет
32	E_CORE_WARNING	Аналогично E_WARNING, но генерируется ядром PHP	Нет
64	E_COMPILE_ERROR	Аналогично E_ERROR, но генерируется Zend Engine	Нет
128	E_COMPILE_WARNING	Аналогично E_WARNING, но генерируется Zend Engine	Нет
256	E_USER_ERROR	Аналогично E_ERROR, но инициируется вызовом trigger_error()	Да
512	E_USER_WARNING	Аналогично E_WARNING, но инициируется вызовом trigger_error()	Да
1024	E_USER_NOTICE	Аналогично E_NOTICE, но инициируется вызовом trigger_error()	Да
2048	E_STRICT	Сообщения от исполнительной среды с рекомендациями по улучшению качества кода (начиная с PHP 5)	–
4096	E_RECOVERABLE_ERROR	Опасная, но не фатальная ошибка (например, несоответствие типа)	Да
8192	E_DEPRECATED	Предупреждение об использовании устаревшей функции или возможности	Да
16384	E_USER_DEPRECATED	Предупреждение об использовании устаревшей функции или возможности, инициированное в коде	Да
32767	E_ALL	Все ошибки	Нет

Ошибки, помеченные как перехватываемые, могут обрабатываться функцией, зарегистрированной вызовом `set_error_handler()`. Остальные сообщают о проблемах настолько серьезных, что их решение пользователем небезопасно и они должны решаться PHP.

Тип `E_RECOVERABLE_ERROR` появился в PHP 5.2.0. Типы `E_DEPRECATED` и `E_USER_DEPRECATED` появились в PHP 5.3.0.

См. также

Рецепт 20.8 — определение пользовательского обработчика ошибок; документация по функциям `error_reporting()` и `set_error_handler()`; дополнительная информация об ошибках.

20.8. Применение пользовательских обработчиков ошибок

Задача

Требуется создать пользовательский обработчик ошибок, с помощью которого вы будете управлять тем, как PHP сообщает об ошибках.

Решение

Чтобы создать собственную функцию обработки ошибок, воспользуйтесь функцией `set_error_handler()`:

```
set_error_handler('pc_error_handler');

function pc_error_handler($errno, $error, $file, $line) {
    $message = "[ERROR][$errno][$error][$file:$line]";
    error_log($message);
}
```

Комментарий

Пользовательская функция обработки ошибок может анализировать ошибки в зависимости от типа и выполнять соответствующие действия. Список типов ошибок приведен в таблице 20.2 (см. Рецепт 20.7).

Передайте `set_error_handler()` имя функции, и PHP будет передавать все ошибки этой функции. Функция обработки ошибок может получать до пяти параметров. В первом параметре передается тип ошибки, например 8 для `E_NOTICE`. Во втором параметре передается сообщение, выдаваемое для ошибки, например «Undefined variable: html.» В третьем и четвертом аргументах передаются имя файла и номер строки, в которой была обнаружена ошибка. В последнем параметре передается массив со всеми переменными, определенными в текущей области действия, и их значениями.

Например, в следующем примере строка присоединяется к переменной `$html`, которой не было присвоено исходное значение:

```
error_reporting(E_ALL);
set_error_handler('pc_error_handler');
```

```
function pc_error_handler($errno, $error, $file, $line, $context) {
    $message = "[ERROR][$errno][$error][$file:$line]";
    print "$message";
    print_r($context);
}

$form = array('one','two');

foreach ($form as $line) {
    $html .= "<b>$line</b>";
}
```

При генерировании ошибки «Undefined variable» функция `pc_error_handler()` выводит следующее сообщение:

```
[ERROR][8][Undefined variable:  html][err-all.php:16]
```

После начального сообщения об ошибке `pc_error_handler()` выводит большой массив со всеми глобальными переменными, а также переменными окружения, запроса и сеанса.

Ошибки, помеченные в таблице 20.2 как перехватываемые, могут обрабатываться функцией, зарегистрированной вызовом `set_error_handler()`. Остальные сообщают о проблемах настолько серьезных, что их решение пользователем небезопасно и они должны решаться PHP.

См. также

Рецепт 20.7 — разные типы ошибок; документация по функции `set_error_handler()`.

20.9. Регистрация ошибок

Задача

Требуется сохранить информацию об ошибках программы в журнале. Ошибки могут быть любыми, от ошибок разбора и отсутствующих файлов до некорректных запросов к базе данных и потери подключения.

Решение

Воспользуйтесь функцией `error_log()` для записи информации в журнал ошибок:

```
// Ошибка LDAP
if (ldap_errno($ldap)) {
    error_log("LDAP Error #" . ldap_errno($ldap) . ": " . ldap_error($ldap));
}
```

Комментарий

Сохранение информации об ошибках упрощает отладку и исправление ошибок. Всегда сохраняйте информацию о контексте возникновения ошибки:

```
$r = mysql_query($sql);
if (! $r) {
    $error = mysql_error();
    error_log('[DB: query @'.$_SERVER['REQUEST_URI']."][$sql]: $error");
} else {
    // Обработка результатов
}
```

Простая регистрация самого факта ошибки без сопутствующей информации особой пользы не принесет:

```
$r = mysql_query($sql);
if (! $r) {
    error_log("bad query");
} else {
    // Обработка результатов
}
```

Другой полезный прием — включение «волшебных» констант `__FILE__`, `__LINE__`, `__FUNCTION__`, `__CLASS__` и `__METHOD__` в сообщения об ошибках:

```
error_log('['.__FILE__.']['.__LINE__."]: $error");
```

Константа `__FILE__` содержит текущее имя файла, `__LINE__` — текущий номер строки, `__FUNCTION__` — текущее имя функции, `__METHOD__` — текущее имя метода (если актуально) и `__CLASS__` — текущее имя класса (если актуально). Начиная с PHP 5.3.0 `__DIR__` содержит каталог, в котором находится файл `__FILE__`, и `__NAMESPACE__` — текущее пространство имен. Начиная с PHP 5.4.0 `__TRAIT__` содержит текущее имя типажа (если актуально).

См. также

Рецепт 20.6 — сокрытие сообщений об ошибках от пользователей; документация по функции `error_log()`; документация по «волшебным» константам (*http://php.net/language.constants.predefined*).

20.10. Устранение ошибок «заголовки уже отправлены»

Задача

Вы пытаетесь отправить заголовок HTTP или cookie с использованием `header()` или `setcookie()`, но PHP выдает сообщение об ошибке «заголовки уже отправлены» (headers already sent).

Решение

Ошибка возникает при отправке «незаголовочного» вывода перед вызовом `header()` или `setcookie()`.

Перепишите свой код так, чтобы весь вывод выполнялся после отправки заголовков:

```
<?php
// Хорошо
setcookie("name", $name);
print "Hello $name!";
// Плохо
print "Hello $name!";
setcookie("name", $name);
// Хорошо
setcookie("name",$name); ?>
<html><title>Hello</title>
```

Комментарий

Сообщение HTTP состоит из заголовка и тела, которые отправляются клиенту именно в таком порядке. Начав отправку тела, вы уже не сможете отправить никакие заголовки. Следовательно, при вызове `setcookie()` после вывода разметки HTML PHP не сможет отправить соответствующий заголовок `Cookie`.

Также удалите завершающие пропуски во всех включаемых файлах. При включении файла, содержащего пустые строки за пределами тегов `<?php ?>`, эти пустые строки отправляются браузеру. Используйте функцию `trim()` для удаления начальных и конечных пустых строк из файлов:

```
$file = '/path/to/file.php';

// Копирование
copy($file, "$file.bak") or die("Can't copy $file: $php_errormsg");

// Чтение и обрезка
$contents = trim(join('',file($file)));

// Запись
$fh = fopen($file, 'w')  or die("Can't open $file for writing: $php_errormsg");
if (-1 == fwrite($fh, $contents)) { die("Can't write to $file: $php_errormsg");}
fclose($fh)              or die("Can't close $file: $php_errormsg");
```

Вместо того чтобы обрабатывать файлы один за одним, может быть удобнее обрабатывать их по каталогам. В Рецепте 25.7 рассказано, как обработать все файлы в каталоге.

Другой абсолютно законный способ устранения завершающих пропусков — просто убрать закрывающий тег `?>`. Если включаемый файл содержит только код PHP, этот способ гарантирует, что вам не придется возвращаться к файлу для удаления непреднамеренных пропусков.

Если вы не хотите беспокоиться о пустых строках, нарушающих последовательность отправки заголовков, включите буферизацию вывода (см. Рецепт 8.13). Буферизация предотвращает немедленную отправку всего вывода клиенту. При буферизации можно чередовать заголовки и текст тела без ограничений. Однако у пользователей может возникнуть впечатление, что выполнение запросов занимает больше времени, потому что им приходится дольше ожидать, пока браузер выдаст результат.

См. также

Рецепт 8.13 — буферизация вывода; Рецепт 25.7 — обработка всех файлов в каталоге; документация по функции `header()`.

20.11. Сохранение отладочной информации

Задача

Требуется включить команды вывода переменных для упрощения отладки. При этом вам хотелось бы легко переключаться между отладочным и нормальным режимом.

Решение

Включите функцию, которая выводит сообщения в зависимости от определения константы, при помощи конфигурационной директивы `auto_prepend_file`. Сохраните следующий код в файле debug.php:

```
// Включение отладки
define('DEBUG',true);
// Функция отладочного вывода
function pc_debug($message) {
    if (defined('DEBUG') && DEBUG) {
        error_log($message);
    }
}
```

Включите директиву `auto_prepend_file` в файл php.ini или файл .htaccess вашего сайта:

```
auto_prepend_file=debug.php
```

Теперь используйте функцию `pc_debug()` из своего кода для вывода отладочной информации:

```
$sql = 'SELECT color, shape, smell FROM vegetables';
pc_debug("[sql: $sql]"); // Выводится только в том случае,
$r = mysql_query($sql);  // если DEBUG содержит true
```

Комментарий

Отладка — неизбежный побочный эффект написания кода. Существует много приемов, позволяющих быстро находить и устранять ошибки. Многие из них основаны на включении служебной «оснастки», помогающей проверить правильность кода. Чем сложнее программа, тем больше служебного кода требуется. Фредерик Брукс в своей книге «Мифический человеко-месяц» (Addison-Wesley) предполагает, что «объем служебного кода достигает половины объема кода продукта». Качественное предварительное планирование позволяет чисто и эффективно интегрировать «оснастку» в программную логику приложения. Для этого следует заранее спланировать, какие характеристики вы хотите измерять и сохранять и как вы собираетесь структурировать данные, собранные служебным кодом.

Один из методов организации собранных данных основан на назначении разных уровней приоритета разным типам отладочных команд. Отладочная функция выводит информацию только в том случае, если ее приоритет выше текущего:

```
define('DEBUG',2);
function pc_debug($message, $level = 0) {
    if (defined('DEBUG') && ($level > DEBUG)) {
        error_log($message);
    }
}

$sql = 'SELECT color, shape, smell FROM vegetables';
pc_debug("[sql: $sql]", 1); // Не выводится, потому что 1 < 2
pc_debug("[sql: $sql]", 3); // Выводится, потому что 3 > 2
```

Другой прием основан на написании функций-оберток, выводящих дополнительную информацию для оптимизации, например время выполнения запроса к базе данных:

```
function db_query($sql) {
    if (defined('DEBUG') && DEBUG) {
         // Если режим DEBUG включен, запустить хронометраж
         $DEBUG_STRING = "[sql: $sql]<br>\n";
         $starttime = microtime(true);
    }

    $r = mysql_query($sql);

    if (! $r) {
        $error = mysql_error();
        error_log('[DB: query @'.$_SERVER['REQUEST_URI']."][$sql]: $error");
    } elseif (defined(DEBUG) && DEBUG) {
        // Запрос выполнен без ошибок, режим DEBUG включен -
        // хронометраж останавливается.
        $endtime = microtime(true);
        $elapsedtime = $endtime - $starttime;
        $DEBUG_STRING .= "[time: $elapsedtime]<br>\n";
        error_log($DEBUG_STRING);
    }

    return $r;
}
```

Здесь вместо простого сохранения кода SQL в журнале ошибок также сохраняется продолжительность выполнения запроса MySQL в секундах. По ней можно выявить запросы, которые выполняются слишком долго. За дополнительной информацией о хронометраже выполнения кода обращайтесь к Рецепту 22.2.

Наконец, можно использовать пакет `Log` из репозитория PEAR, который предоставляет эффективную инфраструктуру для абстрагирования системы ведения журнала. Пакет `Log` определяет восемь уровней приоритета: `PEAR_LOG_EMERG`, `PEAR_LOG_ALERT`, `PEAR_LOG_CRIT`, `PEAR_LOG_ERR`, `PEAR_LOG_WARNING`, `PEAR_LOG_NOTICE`, `PEAR_LOG_INFO` и `PEAR_LOG_DEBUG`. Пакет `Log` предоставляет мощный набор средств для настройки сохранения информации об ошибках, включая направление информации в базу данных SQLite и/или во временное окно в браузере.

См. также

Документация по функциям `define()`, `defined()` и `error_log()`; главная страница пакета PEAR `Log` (*http://pear.php.net/package/Log*).

21 Технология программирования

21.0. Введение

Ввод выражений, образующих компьютерную программу, — всего лишь начало построения жизнеспособной программной системы. В этой главе речь пойдет о том, что происходит после написания начального варианта кода, а именно о процессах отладки и тестирования программного продукта.

В Рецепте 21.1 рассматривается использование Xdebug — расширения PHP с открытым кодом, позволяющего проводить построчную отладку в реальном времени.

Рецепты 21.2, 21.3 и 21.4 посвящены области модульного тестирования PHP. В них показано, как превратить исправленные ошибки в набор тестов, с которыми вы можете быть уверены, что единожды исправленная ошибка так и останется исправленной.

Рецепт 21.5 представляет простые средства организации тестирования на локальном компьютере, чтобы вы могли работать в защищенном окружении, не опасаясь «сломать» работающий сайт в процессе диагностики.

Наконец, в Рецепте 21.6 рассматривается встроенный веб-сервер, который является частью PHP 5.4.0 и более поздних версий.

21.1. Использование отладочного расширения

Задача

Требуется выполнить интерактивную отладку сценариев во время выполнения.

Решение

Воспользуйтесь расширением Xdebug. В сочетании с Xdebug-совместимой средой разработки вы сможете анализировать структуры данных; назначать точки прерывания; выполнять участки кода с заходом или с обходом процедур в интерактивном режиме.

Комментарий

В этом рецепте основное внимание уделяется средствам интерактивной отладки Xdebug. Чтобы следить за изложением материала, вы должны иметь возможность откомпилировать и установить расширение Zend, для чего потребуются разрешения на редактирование файла php.ini в вашей системе. Функция загрузки расширений PHP `dl()` не работает с Xdebug. Наконец, примеры из этого рецепта предназначены для версии Xdebug 2.2.

Установка расширения Xdebug проходит достаточно прямолинейно. Постройте расширение из исходного кода или воспользуйтесь командой *pecl*:

```
% pecl install xdebug
```

Когда расширение будет откомпилировано и установлено, необходимо отредактировать файл php.ini и внести в него полный путь к модулю xdebug.so вида

```
zend_extension = /usr/lib/php/extensions/no-debug-non-zts-20050922/xdebug.so
```

Каталог, в который *pecl* устанавливает xdebug.so, задается в значении конфигурационной директивы `extension_dir`.

Расширение Xdebug установлено правильно, если при выполнении команды `php -m` из командной строки расширение Xdebug указывается в результатах дважды — в разделе `[PHP Modules]` и в разделе `[Zend Modules]`. Если вы пытаетесь установить Xdebug с версией PHP, с которой вы работаете из браузера, проверьте раздел «xdebug» вывода функции `phpinfo()`.

Также для работы удаленной отладки необходимо присвоить конфигурационной директиве `xdebug.remote_enable` значение `on`.

Впрочем, установка расширения Xdebug — всего лишь половина того, что нужно сделать для интерактивной отладки. Второй половиной станет клиент, который умеет взаимодействовать с Xdebug и поможет вам в анализе программы. В этом рецепте для примера используется среда разработки NetBeans — бесплатная, платформенно-независимая и простая в использовании. Впрочем, расширение Xdebug и его протокол отладки DBGp поддерживаются множеством других сред, как бесплатных, так и коммерческих.

Чтобы настроить NetBeans для взаимодействия с установкой Xdebug, необходимо выполнить несколько операций. Во-первых, убедитесь в том, что в свойствах проекта параметр Project URL указывает на URL-адрес веб-сервера, на котором работает PHP. Затем в разделе PHP панели настроек убедитесь в том, что значе-

ние Debugger Port соответствует настройке PHP `xdebug.remote_port` (обычно 9000).

Если все настроено правильно, то при выполнении команды Debug Project в NetBeans инициируется запрос к домашней странице проекта в браузере, а управление передается первой строке кода PHP домашней страницы проекта. При этом на экране появляются окно кода и панель отслеживания переменных Variables (рис. 21.1).

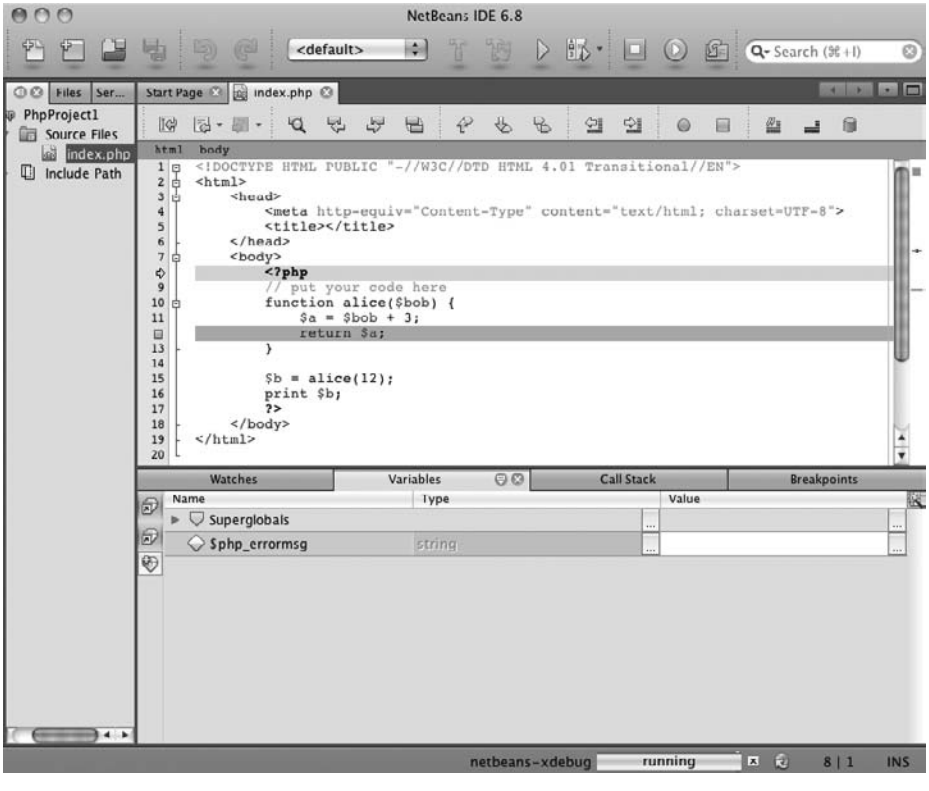

Рис. 21.1. Начало сеанса отладки

Красный квадрат вместо номера строки 12 на иллюстрации появился в результате щелчка на номере строки и вставки точки прерывания. Если нажать клавишу F5, PHP продолжит выполнение кода, пока не будет достигнута точка прерывания (рис. 21.2).

В точке прерывания содержимое панели Variables обновляется значениями локальных переменных `$a` и `$bob`. Xdebug поддерживает широкий спектр возможностей, которые можно ожидать от интерактивного отладчика, — точки прерываний, отслеживание, трассировка и т. д. Конкретный способ их использования зависит от среды разработки. Но когда программа PHP начинает вести себя

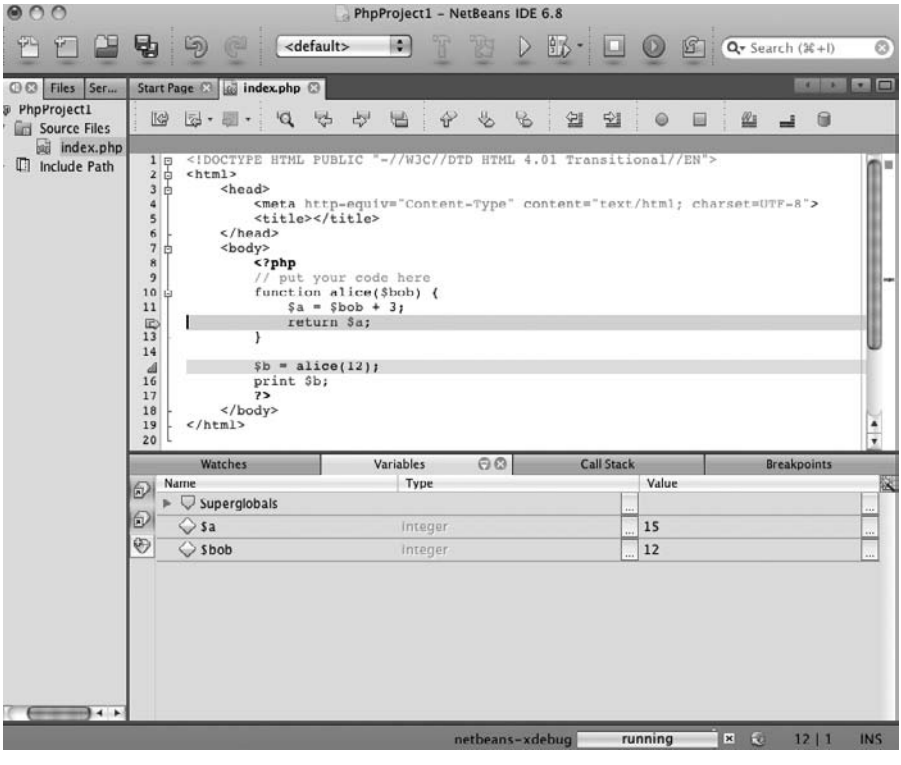

Рис. 21.2. Продолжение сеанса отладки

странно, ничто не сравнится по эффективности с возможностью установить точку прерывания, выполнить программу до конкретной строки, а затем проверить значения переменных в этой точке.

См. также

Документация по Xdebug и протоколу DBGp; возможности среды разработки NetBeans с поддержкой Xdebug для PHP (*https://netbeans.org/features/php/index.html*).

21.2. Написание модульного теста

Задача

Вы работаете над проектом, расширяющим некую базовую функциональность. Вам нужен простой способ убедиться в том, что все продолжает работать нормально по мере разрастания проекта.

Решение

Напишите модульный тест, который тестирует базовые возможности функции или класса и оповещает вас об обнаруженных проблемах.

Простой пример теста, написанного с использованием системы тестирования PHP-QA .phpt:

```
--TEST--
str_replace() function
--FILE--

<?php
$str = 'Hello, all!';
var_dump(str_replace('all', 'world', $str));
?>
--EXPECT--
string(13) "Hello, world!"
```

Пример теста, использующего мощный и популярный пакет PHPUnit:

```
class StrReplaceTest extends PHPUnit_Framework_TestCase
{
    public function testStrReplaceWorks()
    {
        $str = 'Hello, all!';
        $this->assertEquals('Hello, world!', str_replace('all', 'world', $str));
    }
}
```

Комментарий

Существует много способов написания модульных тестов для PHP. Возможно, серии простых тестов .phpt будет достаточно для ваших целей, а может, вам нужно структурированное решение на базе PHPUnit? Мы обсудим оба варианта, но начать нужно с другого: зачем вообще писать модульные тесты?

Создание приложения с нуля на любом языке напоминает чистку луковицы, только наоборот. Вы начинаете с сердцевины и накладываете слой за слоем, пока не получится конечный продукт — луковица.

Чем больше слоев накладывается на сердцевину, тем важнее, чтобы сердцевина продолжала работать как задумано. Как проще всего убедиться в том, что сердцевина продолжает правильно функционировать, особенно после внесения изменений? В этом вам помогут модульные тесты.

В предыдущем примере мы проверяем, что функция `str_replace()` успешно заменяет одну строку другой. При этом для теста несущественно, как написана функция `str_replace()`; важно лишь то, чтобы она продолжала правильно работать.

В простейшем варианте тест сохраняется в файле с суффиксом .phpt (например, str_replace.phpt), после чего используется встроенная программа PEAR для выполнения .phpt:

```
% pear run-tests str_replace.phpt
```

Результат выглядит примерно так:

```
Running 1 tests
PASS str_replace() function[str_replace.phpt]
TOTAL TIME: 00:00
1 PASSED TESTS
0 SKIPPED TESTS
```

Чтобы протестировать несколько разных аспектов базовой функциональности, создайте несколько файлов .phpt и выполните их:

```
% pear run-tests *.phpt
```

Полную информацию о структуре файлов .phpt можно найти по адресу *http://qa.php.net/write-test.php*.

Для написания модульных тестов также можно воспользоваться инфраструктурой модульного тестирования PHPUnit.

Если тест PHPUnit, приведенный в разделе «Решение», хранится в файле StrReplaceTest.php, после установки PHPUnit тест можно выполнить следующим образом:

```
% phpunit StrReplaceTest
```

Эта команда ищет файл с именем StrReplaceTest.php и выполняет определенный в нем тест.

PHPUnit — чрезвычайно мощная инфраструктура тестирования, и ее возможности выходят далеко за рамки простого запуска тестов, как в приведенном примере.

См. также

Документация по модульным тестам .phpt и PHPUnit.

21.3. Написание пакета модульных тестов

Задача

Требуется организовать регулярное выполнение более одного модульного теста.

Решение

Объедините свои модульные тесты в группу, называемую *пакетом модульных тестов*.

Комментарий

Редко встречаются программы настолько простые, что один модульный тест удовлетворит все потребности в тестировании, возникающие на протяжении их жизненного цикла. Со временем приложение разрастается, и возникает необходимость в добавлении новых тестов — либо для тестирования новой функциональности, либо для проверки того, что исправленные ошибки не появятся снова.

Когда библиотека тестов достигает сколько-нибудь заметного размера, становится намного удобнее сгруппировать тесты в пакет модульных тестов. Пакет модульных тестов, несмотря на формальное название, представляет собой обычную обертку для набора тестов, которые можно запустить все сразу (по имени тестового пакета).

С помощью PHPUnit можно создать пакет для проверки множества условий, а не результатов единственной функции `str_replace`. В одном файле можно разместить несколько тестов, относящихся к строковым функциям. Например, включите в файл с именем StringTest.php следующий код:

```
class StringTest extends PHPUnit_Framework_TestCase
{
    function testStrReplace()
    {
        $str = 'Hello, all!';
        $this->assertEquals('Hello, world!', str_replace('all', 'world', $str));
    }

    function testSubstr()
    {
        $str = 'Hello, all!';
        $this->assertEquals('e', substr($str, 1, 1));
    }
}
```

В файле определяются два теста, которые могут запускаться из класса `StringTest`. Создайте аналогичный файл с именем ArrayTest.php, в котором определяются следующие тесты:

```
class ArrayTest extends PHPUnit_Framework_TestCase
{
    function testArrayFlip()
    {
        $array = array('foo' => 'bar', 'cheese' => 'hotdog');
        $flipped = array_flip($array);
        $this->assertEquals('foo', reset($flipped));
```

```
    }

    function testArrayPop()
    {
        $array = array('foo' => 'bar', 'cheese' => 'hotdog');
        $popped = array_pop($array);
        $this->assertEquals('hotdog', $popped);
        $this->assertEquals(1, sizeof($array));
    }
}
```

С четырьмя тестами уже можно собрать пакет, который будет выполнять все эти тесты каждый раз, когда вы захотите удостовериться в правильности работы программы. Если оба файла хранятся в одном каталоге, просто укажите этот каталог в командной строке PHPUnit:

```
% phpunit testDir
```

Если оба тестовых файла находятся в каталоге testDir, результат будет выглядеть примерно так:

```
PHPUnit 3.7.24 by Sebastian Bergmann.
....
Time: 45 ms, Memory: 5.00Mb

OK (4 tests, 5 assertions)
```

Выполняются все файлы, имена которых соответствуют шаблону *Test.php. PHPUnit также осуществляет рекурсивный обход подкаталогов.

Такой подход позволяет включить в автоматизированную систему тестирования большое количество тестов, которые будут запускаться всего одной командой.

См. также

Документация по организации групп тестов в PHPUnit.

21.4. Применение модульного теста к веб-странице

Задача

Ваше приложение не разбито на небольшие блоки, удобные для тестирования, или вы хотите применить модульные тесты к сайту, с которым взаимодействуют посетители.

Решение

Воспользуйтесь интеграцией PHPUnit с Selenium Server для написания тестов, которые выдают запросы HTTP и проверяют различные условия для ответов. Следующие тесты проверяют заданные условия относительно структуры *www.example.com*:

```
class ExampleDotComTest extends PHPUnit_Extensions_SeleniumTestCase
{

    function setUp() {
        $this->setBrowser('firefox');
        $this->setBrowserUrl('http://www.example.com');
    }

    // Простейшая загрузка домашней страницы
    function testHomepageLoading()
    {
        $this->open('http://www.example.com/');
        $this->assertTitle('Example Domain');
    }

    // Тест щелчка на ссылке с получением нужной страницы
    function testClick()
    {
        $this->open('http://www.example.com/');
        $this->clickAndWait('link=More information...');
        $this->assertTitle('IANA — IANA-managed Reserved Domains');
    }
}
```

Результат:

```
PHPUnit 3.7.24 by Sebastian Bergmann.
..
Time: 9.05 seconds, Memory: 3.50Mb

OK (2 tests, 2 assertions)
```

Комментарий

Если ваш сайт управляется (полностью или частично) процедурным кодом PHP, иногда бывает трудно написать содержательные модульные тесты для проверки инкапсулируемой функциональности. Вместо этого можно просто убедиться в том, что сайт работает как предполагалось; если что-то работает не так, можно браться за отладку. Кроме того, тестирование реальных результатов выполнения вашего кода на веб-сервере позволит проверить работу элементов пользовательского интерфейса, ссылок и других элементов, с которыми имеет дело пользователь.

Расширение PHPUnit Selenium обеспечивает интеграцию с Selenium Server — бесплатным платформенно-независимым инструментом для проведения тестирования в браузере. После загрузки и запуска Selenium Server (который представляет собой отдельный Java-файл в формате .jar) тестовые сценарии PHPUnit, расширяющие базовый класс `PHPUnit_Extensions_SeleniumTestCase`, могут взаимодействовать с сервером при помощи некоторых новых методов. В приведенном примере вызов `open()` открывает заданный URL-адрес, вызов `clickAndWait()` имитирует «щелчок» на ссылке в возвращаемой странице для посещения новой страницы, а вызов `assertTitle()` проверяет условие для элемента `<title/>` веб-страницы.

Selenium поддерживает обширный набор команд для проверки условий относительно содержимого веб-страниц, размещения элементов, ссылок и т. д.

См. также

Документация по интеграции PHPUnit с Selenium (*https://phpunit.de/manual/current/en/selenium.html*); документация Selenium Server (*http://docs.seleniumhq.org/*); справочник команд Selenium (*http://release.seleniumhq.org/selenium-core/1.0.1/reference.html*).

21.5. Настройка среды тестирования

Задача

Требуется протестировать сценарии PHP, не рискуя вывести сайт из строя или нарушить параметры его рабочей среды.

Решение

Создайте для приложения среду тестирования при помощи XAMPP.

Комментарий

Из-за сложностей настройки локализованной рабочей среды веб-приложения разработчики часто отказываются от этого шага. Это в свою очередь приводит к нарушению важных принципов разработки, например редактированию файлов на основном сайте с правами привилегированного пользователя. Никогда так не поступайте!

Проект XAMPP предоставляет решения для четырех платформ: Windows 98/NT/2000/XP, Mac OS X, Linux (SuSE/RedHat/Mandrake/Debian) и Solaris. Пакеты содержат синхронизированные версии Apache, MySQL, PHP и PEAR,

phpMyAdmin и eAccelerator. Благодаря простой пошаговой процедуре установки проекта XAMPP создать среду запуска веб-приложения на локальной машине проще простого.

Работаете с большими объемами данных? К сожалению, разработчики сайтов, работающих с большим, часто изменяющимся контентом, часто лишаются возможности применять правильные принципы разработки из-за нехватки тестовых данных. Не совершайте эту оплошность! Просто напишите сценарий, который создает локальную копию вашей структуры данных, и периодически обновляйте локальную копию небольшим подмножеством полного набора. Так вы сможете использовать текущую копию актуальных данных, объем которой будет достаточен для реального тестирования и разработки.

См. также

Домашняя страница проекта XAMPP (*http://www.apachefriends.org/en/xampp.html*).

21.6. Использование встроенного веб-сервера

Задача

Требуется использовать встроенный веб-сервер PHP для быстрого проведения теста или проверки сайта.

Решение

В PHP 5.4.0 и выше выполните программу командной строки *php* с аргументом *-S*, именем хоста и портом для прослушивания — и в вашем распоряжении появится веб-сервер с поддержкой PHP, который поставляет данные из каталога, из которого он был запущен:

```
% php -S localhost:9876
```

Комментарий

При запуске только с аргументом *-S хост:порт* встроенный веб-сервер считает каталог, из которого он был запущен, корневым каталогом документов. Если он запускается из каталога /home/roger, то запрос */files/monkeys.php* соответствует файлу */home/roger/files/monkeys.php*. Если запрос относится к каталогу, а не к конкретному файлу, встроенный веб-сервер сначала ищет в каталоге файл index.php, а затем каталог index.html.

Чтобы использовать другой корневой каталог, передайте его в аргументе *-t* при запуске PHP. Пример:

```
% php -S localhost:9876 -t /var/www
```

Для организации косвенного отображения между URL-адресами запросов и ответами передайте файл с кодом PHP, выполняющим маршрутизацию запросов, в дополнительном аргументе:

```
% php -S localhost:9876 router.php
```

Прежде чем переходить к поиску в пути, соответствующем URL-адресу запроса, PHP выполняет код из файла router.php. PHP переходит к поиску в пути только в том случае, если код возвращает `false`. Это обстоятельство позволяет организовать свободное генерирование ответов. В листинге 21.1 приведен маршрутизатор запросов, который возвращает обменный курс для двух валют, коды которых содержатся в URL.

Листинг 21.1. Маршрутизация запросов для встроенного веб-сервера

```
$parts = explode('/', $_SERVER['REQUEST_URI']);
// Ожидается URI запроса вида /USD/ISK; проверить,
// что он состоит минимум из двух частей и каждая
// содержит три буквы
if (! (isset($parts[1]) &&
       preg_match('/[a-z]{3}/i', $parts[1]) &&
       isset($parts[2]) &&
       preg_match('/[a-z]{3}/i', $parts[2]))) {
    header('Bad Request', true, 400);
    print "Bad Request";
    exit();
}

$quotes = 'http://download.finance.yahoo.com/d/quotes.csv?f=nl1&s=%s%s=X,%s%s=X';
$url = sprintf($quotes,
               urlencode($parts[1]), urlencode($parts[2]),
               urlencode($parts[2]), urlencode($parts[1]));
$response = file_get_contents($url);
$lines = explode("\n", trim($response));
foreach ($lines as $line) {
    list($label, $rate) = str_getcsv($line);
    print "<b>" . htmlentities($label) . "</b>: " .
      htmlentities($rate) . "<br/>";
}
```

Когда листинг 21.1 используется как маршрутизатор запросов встроенным веб-сервером, элемент `$_SERVER['REQUEST_URI']` содержит путь запроса, запрошенный клиентом. В данном примере запросы вида */USD/EUR* должны возвращать обменный курс между долларами США (USD) и евро (EUR). Итак, сначала код убеждается в том, что URI запроса содержит два кода валют, состоящие из трех букв. Затем коды валют подставляются в URL-адрес, который загружает курс с сайта Yahoo! Finance. Ответ от URL представляет собой серию строк, каждая

из которых содержит набор разделенных запятыми полей, описывающих тип курса и его числовое значение. Цикл `foreach` в конце кода выводит курсы с минимальным форматированием HTML. На рис. 21.3 изображен URL-адрес запроса и вывод, полученный для преобразования между исландскими кронами и японскими иенами. В этом примере при запуске сервера используется порт 9876.

Рис. 21.3. Маршрутизация запросов в действии

См. также

Документация по встроенному веб-серверу PHP и Yahoo! Finance CSV API.

22 Оптимизация

22.0. Введение

PHP работает достаточно быстро. Как правило, медленное выполнение частей программы PHP связано с внешними ресурсами — ожиданием завершения запроса к базе данных или получения данных с удаленного URL-адреса. Впрочем, код PHP тоже порой работает менее эффективно, чем мог бы. В этой главе описаны приемы выявления и решения проблем с быстродействием в вашем коде.

В мире программирования идут ожесточенные споры относительно того, когда в процессе разработки следует переходить к оптимизации. Начнете слишком рано, и вы потратите слишком много времени на возню с подробностями, которые могут оказаться несущественными в общей картине; займетесь оптимизацией слишком поздно, и вам придется переписывать большие фрагменты кода приложения.

Оптимизация не происходит в изоляции. В процессе оптимизации кода вы изменяете не только непосредственное время выполнения — ваши правки также влияют на размер кода, удобочитаемость и простоту сопровождения кода. Всегда встречаются ситуации, требующие молниеносного выполнения кода. Тем не менее чаще на первое место выходит время программиста или простота отладки. Попробуйте выдержать баланс между этими факторами, когда будете заниматься оптимизацией своего кода.

Установка акселератора кода — лучшее, что можно сделать для повышения быстродействия PHP. Версия PHP 5.5 включает и строит акселератор PHP Zend OPcache, но OPcache работает в PHP 5.2 и выше. Это расширение рассматривается в Рецепте 22.1.

Если ваше приложение продолжает работать слишком медленно, начните с интеграции аналитических средств в процедуру разработки. Ваша задача — определить, какие части приложения занимают наибольшее время. Трудно заранее определить, где кроются проблемы. Это может быть однократно вызываемая

секция, которая выполняется очень медленно, или же небольшая функция, которая работает быстро, но вызывается слишком часто. Вопрос в том, как быстро выявить проблемную область в коде; с увеличением объема приложения эта задача становится все сложнее.

Существует пять разных способов разбиения приложений на разных уровнях; Рецепт 22.2 показывает, как организовать хронометраж функции, а Рецепт 22.3 объясняет, как легко измерить время выполнения всех вызовов функций в блоке кода. Профилирование кода по командам рассматривается в Рецепте 22.4, а профилирование по секциям — в Рецепте 22.5. Наконец, следующим логическим шагом становится использование расширения отладчика для профилирования приложений в Рецепте 22.6.

Обзор средств нагрузочного тестирования сайта в Рецепте 22.7 напоминает о том, что оптимизация не ограничивается одним кодом — сетевая задержка и оборудование также играют важную роль.

Одним из наиболее распространенных узких мест в сценариях PHP является некорректное использование регулярных выражений. В Рецепте 22.8 объясняются некоторые методы поиска совпадений в тексте без затрат ресурсов, связанных с регулярными выражениями.

22.1. Использование акселератора

Задача

Требуется повысить быстродействие приложений PHP.

Решение

Используйте кэширующий акселератор PHP Zend OPcache, чтобы PHP не приходилось компилировать сценарии в коды операций при каждом запросе.

Комментарий

Работа акселераторов PHP основана на сохранении откомпилированных версий сценариев PHP на диске или в совместной памяти для того, чтобы предотвратить компиляцию при каждом запросе.

Когда вы приказываете ядру PHP запустить программу, оно читает исходный код программы и компилирует его в компактное внутреннее представление, после чего выполняет инструкции в откомпилированном представлении. Завершив выполнение сценария, ядро уничтожает откомпилированное представление.

Акселератор сохраняет (кэширует) откомпилированные команды. Когда ядро PHP снова получает запрос на выполнение той же программы, акселератор перехваты-

вает управление и проверяет наличие откомпилированной версии этой программы. Если откомпилированная версия будет найдена, акселератор приказывает ядру PHP пропустить повторную компиляцию и использовать уже откомпилированную версию. Акселератор можно настроить так, чтобы откомпилированные представления обновлялись по разным критериям, например при изменении исходной программы или только тогда, когда вы прикажете ему это сделать.

В версии PHP 5.5 акселератор Zend OPcache строится и устанавливается автоматически. Если вы используете более раннюю версию PHP, установите акселератор из репозитория GitHub или PECL.

Хотя PHP 5.5 строит акселератор Zend OPcache, по умолчанию он не используется. Отредактируйте файл php.ini и добавьте ссылку на полный путь расширения: *zend_extension=/path/to/php/lib/php/extension/debug-non-zts-20121212/opcache.so*.

Хотя вы сразу заметите явное улучшение, быстродействие можно еще улучшить посредством дополнительной оптимизации. Начните с обновления параметров рабочей конфигурации:

```
opcache.memory_consumption=128
opcache.interned_strings_buffer=8
opcache.max_accelerated_files=4000
opcache.revalidate_freq=60
opcache.fast_shutdown=1
opcache.enable_cli=1
```

В конечном итоге «правильные» настройки определяются сочетанием факторов, зависящих от объема кода, частоты его обновления и вызова, объема памяти в системе и т. д. Чтобы подобрать идеальное сочетание параметров для конкретной системы, необходимо поэкспериментировать и воспользоваться программой нагрузочного тестирования (эта тема рассматривается в Рецепте 22.7).

См. также

Документация OPcache (*http://us2.php.net/opcache*).

22.2. Хронометраж выполнения функций

Задача

Имеется функция. Требуется узнать, сколько времени уходит на ее выполнение.

Решение

Сравните время в миллисекундах до выполнения функции с временем в миллисекундах после ее выполнения. Разность определяет время, проведенное за выполнением самой функции:

```
// Создание длинной строки
$long_str = uniqid(php_uname('a'), true);

// Хронометраж начинается с этого момента
$start = microtime(true);

// Тестируемая функция
$md5 = md5($long_str);

$elapsed = microtime(true) - $start;
echo "That took $elapsed seconds.\n";
```

Комментарий

Чтобы определить, сколько времени потребовалось для выполнения одной функции, специальный хронометражный пакет не понадобится. Всю необходимую информацию можно получить от функции `microtime()`.

Существуют три способа вычисления хеша MD5 в PHP (с одинаковым результатом):

```
// Базовая функция PHP md5()
$hashA = md5('optimize this!');
// Расширение mhash
$hashB = bin2hex(mhash(MHASH_MD5, 'optimize this!'));
// Функция hash()
$hashC = hash('md5', 'optimize this!');
```

Все три переменные — `$hashA`, `$hashB` и `$hashC` — содержат `83f0bb25be8de910670840d66f261cf`. И все же третий способ работает почти вдвое быстрее базовой функции PHP `md5()`.

У оптимизации на базе таких сравнительных тестов есть своя оборотная сторона: вы должны знать, насколько часто функция вызывается в вашем коде и насколько прост в чтении и сопровождении код альтернативных решений.

Например, при выборе хеш-функции, если ваш код должен работать в версиях PHP до 5.12 (боже упаси!), придется либо использовать `md5()` постоянно, либо добавить проверку, которая будет выбирать используемую функцию на основании версии PHP (и возможно, наличия расширения mhash). Абсолютная разность времени между `md5()` и `hash()` имеет порядок десятых долей миллисекунды. При вычислении тысяч и миллионов хеш-кодов вставка лишней проверки для выбора более быстрой функции оправданна. Однако ничтожные доли секунды, сэкономленные на десятке-другом вычислений, не оправдывают лишней сложности.

См. также

Рецепт 3.11 — функция `microtime()`; документация по функции `microtime()`.

22.3. Хронометраж функций

Задача

Имеется блок кода. Требуется узнать, сколько времени занимает выполнение каждой из его функций.

Решение

Воспользуйтесь средствами трассировки Xdebug:

```
xdebug_start_trace('/tmp/factorial-trace');

function factorial($x) {
    return ($x == 1) ? 1 : $x * factorial($x - 1);
}

print factorial(10);

xdebug_stop_trace();
```

Комментарий

Расширение Xdebug предоставляет обширный набор полезных инструментов отладки и профилирования. Оно доступно в репозитории PECL и в виде готового двоичного файла Windows.

Его средства трассировки функций предоставляют информацию о том, какие функции из какой точки вызывались (а при желании — о передаваемых и возвращаемых аргументах). Также сохраняется информация о времени и затратах памяти для каждого вызова.

Для примера с функцией `factorial()` результат выглядит примерно так:

```
TRACE START [2015-01-05 06:32:11]
    0.0005     240136     -> factorial($x = 10) /factorial.php:9
    0.0005     240184       -> factorial($x = 9) /factorial.php:6
    0.0005     240256         -> factorial($x = 8) /factorial.php:6
    0.0006     240304           -> factorial($x = 7) /factorial.php:6
    0.0006     240352             -> factorial($x = 6) /factorial.php:6
    0.0006     240400               -> factorial($x = 5) /factorial.php:6
    0.0007     240448                 -> factorial($x = 4) /factorial.php:6
    0.0007     240496                   -> factorial($x = 3) /factorial.php:6
    0.0007     240544                     -> factorial($x = 2) /factorial.php:6
    0.0008     240592                       -> factorial($x = 1) /factorial.php:6
                                              >=> 1
                                            >=> 2
                                          >=> 6
                                        >=> 24
                                      >=> 120
```

```
                                    >=> 720
                                  >=> 5040
                                >=> 40320
                              >=> 362880
                            >=> 3628800
    0.0010     240136     -> xdebug_stop_trace() /factorial.php:12
    0.0010     240176
TRACE END   [2015-01-05 06:32:11]
```

В первом столбце указано начальное время в секундах; в следующем столбце приведены затраты памяти. Затем следует вызов функции с переданными аргументами. Наконец, указываются имя файла и номер строки. По мере выполнения функций приводятся данные, возвращенные каждой из них.

Xdebug предоставляет несколько возможностей для форматирования этих результатов. В приведенном примере используются следующие параметры конфигурации:

```
xdebug.trace_format=0    ; Обычный текст, понятный для человека
xdebug.collect_params=4  ; Полное содержимое и имена переменных
xdebug.collect_return=1  ; Вывод возвращаемых значений
```

Средства профилирования функций в Xdebug предоставляют простой механизм получения подробной информации обо всем, что происходит в программном блоке. Тем не менее в некоторых ситуациях подобная детализация может оказаться излишней.

См. также

Документация по Xdebug и трассировке функций; Рецепт 21.1 — использование Xdebug для отладки и инструкции по установке.

22.4. Хронометраж по командам

Задача

Требуется узнать, сколько времени занимает выполнение каждой команды в блоке кода.

Решение

Воспользуйтесь конструкцией `declare` и директивой `ticks`:

```
function profile($display = false) {
    static $times;

    switch ($display) {
```

```
    case false:
        // Добавить время в список
        $times[] = microtime();
        break;
    case true:
        // Вернуть затраченное время в микросекундах
        $start = array_shift($times);

        $start_mt = explode(' ', $start);
        $start_total = doubleval($start_mt[0]) + $start_mt[1];

        foreach ($times as $stop) {
            $stop_mt = explode(' ', $stop);
            $stop_total = doubleval($stop_mt[0]) + $stop_mt[1];
            $elapsed[] = $stop_total - $start_total;
        }

        unset($times);
        return $elapsed;
        break;
    }
}

// Регистрация обработчика
register_tick_function('profile');

// Получение начального времени
profile();

// Выполнение кода с сохранением времени для каждой выполненной команды
declare (ticks = 1) {
    foreach ($_SERVER['argv'] as $arg) {
        print "$arg: " . strlen($arg) ."\n";
    }
}

// Вывод затраченного времени
print "---\n";
$i = 0;
foreach (profile(true) as $time) {
    $i++;
    print "Line $i: $time\n";
}
```

Комментарий

Директива `ticks` позволяет организовать многократное выполнение функции в программном блоке. Число, присвоенное `ticks`, определяет количество команд, после выполнения которых срабатывает функция, зарегистрированная вызовом `register_tick_function()`.

В Решении регистрируется одна функция, а функция `profile()` выполняется для каждой команды в блоке `declare`. Если `$_SERVER['argv']` содержит два элемента, `profile()` выполняется шесть раз: при запуске отсчета; два раза для двух итераций

цикла `foreach`; еще два раза при выполнении строки `print strlen($arg)`; и наконец, один раз, когда `foreach` возвращает `false`:

```
Line 1: 5.3882598876953E-5
Line 2: 5.6982040405273E-5
Line 3: 6.2942504882812E-5
Line 4: 6.5803527832031E-5
Line 5: 6.7949295043945E-5
Line 6: 6.9856643676758E-5
```

Возможна и более сложная настройка, например вызов двух функций через каждые три команды:

```
register_tick_function('profile');
register_tick_function('backup');

declare (ticks = 3) {
    // Код...
}
```

Зарегистрированным функциям (в том числе и методам объектов, а не только простым функциям) также могут передаваться дополнительные параметры:

```
// Передать "parameter" функции profile()
register_tick_function('profile', 'parameter');

// Вызвать $car->drive();
$car = new Vehicle;
register_tick_function(array($car, 'drive'));
```

Чтобы выполнять метод объекта, передайте сам объект и имя метода в массиве. Так функция `register_tick_function()` сможет определить, что вы ссылаетесь на объект, а не на функцию.

Чтобы исключить функцию из списка зарегистрированных функций, вызовите `unregister_tick_function()`:

```
unregister_tick_function('profile');
```

См. также

Документация по функциям `register_tick_function()`, `unregister_tick_function()` и блокам `declare`.

22.5. Хронометраж по секциям

Задача

Требуется узнать, сколько времени занимает выполнение каждой секции в блоке кода.

Решение

Воспользуйтесь модулем PEAR Benchmark:

```
require_once 'Benchmark/Timer.php';

$timer = new Benchmark_Timer(true);

$timer->start();
// Начальный код
$timer->setMarker('setup');
// Фрагмент кода
$timer->setMarker('middle');
// Еще один фрагмент кода
$timer->setmarker('done');
// И завершающий фрагмент
$timer->stop();

$timer->display();
```

Комментарий

Пакет PEAR Benchmark позволяет легко и быстро расставить в коде маркеры для определения профилируемых участков на более высоком уровне. Для установки пакета используется менеджер пакетов PEAR:

```
% pear install Benchmark
```

Вызов `setMarker()` сохраняет текущее время. Метод `display()` выводит список маркеров, время их установки и интервалы, отделяющие их от предыдущего маркера:

```
------------------------------------------------------------
marker    time index            ex time               perct
------------------------------------------------------------
Start     1029433375.42507400   -                       0.00%
------------------------------------------------------------
setup     1029433375.42554800   0.00047397613525391    29.77%
------------------------------------------------------------
middle    1029433375.42568700   0.00013899803161621     8.73%
------------------------------------------------------------
done      1029433375.42582000   0.00013303756713867     8.36%
------------------------------------------------------------
Stop      1029433375.42666600   0.00084602832794189    53.14%
------------------------------------------------------------
total     -                     0.0015920400619507    100.00%
------------------------------------------------------------
```

Модуль `Benchmark` также включает класс `Benchmark_Iterate`, который может оценивать время при многократных вызовах одной функции:

```
require 'Benchmark/Iterate.php';

$timer = new Benchmark_Iterate;
```

```
// Пример функции для проведения хронометража
function use_preg($ar) {
    for ($i = 0, $j = count($ar); $i < $j; $i++) {
        if (preg_match('/gouda/',$ar[$i])) {
            // it's gouda
        }
    }
}

// Другая функция
function use_equals($ar) {
    for ($i = 0, $j = count($ar); $i < $j; $i++) {
        if ('gouda' == $ar[$i]) {
            // Условие выполнено
        }
    }
}

// Выполнить use_preg() 1000 раз
$timer->run(1000,'use_preg',
                array('gouda','swiss','gruyere','muenster','whiz'));
$results = $timer->get();
print "Mean execution time for use_preg(): $results[mean]\n";

// Выполнить use_equals() 1000 раз
$timer->run(1000,'use_equals',
                array('gouda','swiss','gruyere','muenster','whiz'));
$results = $timer->get();
print "Mean execution time for use_equals(): $results[mean]\n";
```

Метод `Benchmark_Iterate::get()` возвращает ассоциативный массив. Элемент `mean` этого массива содержит среднее время выполнения одной итерации функции, а элемент `iterations` содержит количество итераций. Время выполнения каждой итерации функции хранится в элементе массива с целочисленным ключом. Например, время первой итерации хранится в `$results[1]`, а время 37-й итерации — в `$results[37]`.

См. также

Информация о классе PEAR `Benchmark`.

22.6. Профилирование с отладочным расширением

Задача

Требуется организовать полноценное профилирование приложений, чтобы вы могли постоянно следить за тем, в каких участках кода программа проводит бóльшую часть своего времени.

Решение

Воспользуйтесь расширением Xdebug из репозитория PECL. После установки Xdebug добавьте строку `xdebug.profiler_enable=1` в конфигурационный файл php.ini для сохранения файла трассировки на диске. Разбор файла трассировки при помощи специальных программ позволит получить информацию о распределении времени при данном запуске сценария PHP.

Комментарий

С установленным расширением Xdebug можно легко запустить сеанс профилирования, который будет сохранять информацию о ходе выполнения приложения.

Чтобы эта информация генерировалась для всех запросов, присвойте конфигурационной переменной `xdebug.profiler_enable` значение 1. Чтобы данные генерировались в зависимости от условия, присвойте `xdebug.profiler_enable` значение `off`, а переменной `xdebug.profiler_enable_trigger` — значение `on`. В такой конфигурации Xdebug будет выполнять профилирование только при передаче переменной GET, POST или cookie с именем `XDEBUG_PROFILE` и произвольным значением.

Условное генерирование, несмотря на более сложный способ активизации, имеет свои преимущества, поскольку вы можете сохранять информацию только по отдельным запросам. В сложных приложениях журналы профилирования могут быть достаточно большими; условное генерирование предотвращает нехватку дискового пространства. Кроме того, если проблема не воспроизводится в тестовой среде, Xdebug может с меньшим риском применяться в условиях реальной эксплуатации (пусть такое решение и не идеально).

Выходные файлы, генерируемые Xdebug, можно сохранять в любом месте, доступном для записи PHP. Каталог задается конфигурационной переменной `xdebug.profiler_output_dir`, а имя файла — переменной `xdebug.profiler_output_name`.

По умолчанию используется файл с именем cachegrind.out., за которым следует идентификатор процесса. По такой структуре трудно определить, какая информация была сохранена в файле. К счастью, Xdebug позволяет использовать разные форматы. Например, с форматом `xdebug.profiler_output_name=cachegrind.out.%R.%t` в имя включается URI запроса и временная метка, а сгенерированным файлам присваиваются имена вида `cachegrind.out._factorial_php.1388986739`.

Выходные файлы обрабатываются специальными приложениями, упрощающими просмотр и анализ данных. Поиски возможностей для оптимизации стоит начинать с функций, выполнение которых занимает больше всего времени.

Популярная программа с графическим интерфейсом KCachegrind используется для анализа данных с целью выявления узких мест и интенсивно используемых участков. Кроссплатформенная версия этой программы QCachegrind доступна

для пользователей, не работающих в среде KDE, например для разработчиков, использующих MacOS X или Windows. Для работы QCachegrind требуется Qt 4.4 и выше.

Следующий (упрощенный) пример выводит значения первых 50 факториалов:

```
function factorial($x) {
    return ($x == 1) ? 1 : $x * factorial($x - 1);
}

for ($i = 1; $i <= 50; $i++) {
    print "$i: " . factorial($i) . "\n";
}
```

Загрузка профильной информации Xdebug в QCachegrind (рис. 22.1) позволяет определить, что функция `factorial()` вызывалась рекурсивно 2450 раз, на что потребовалось 13 376 циклов.

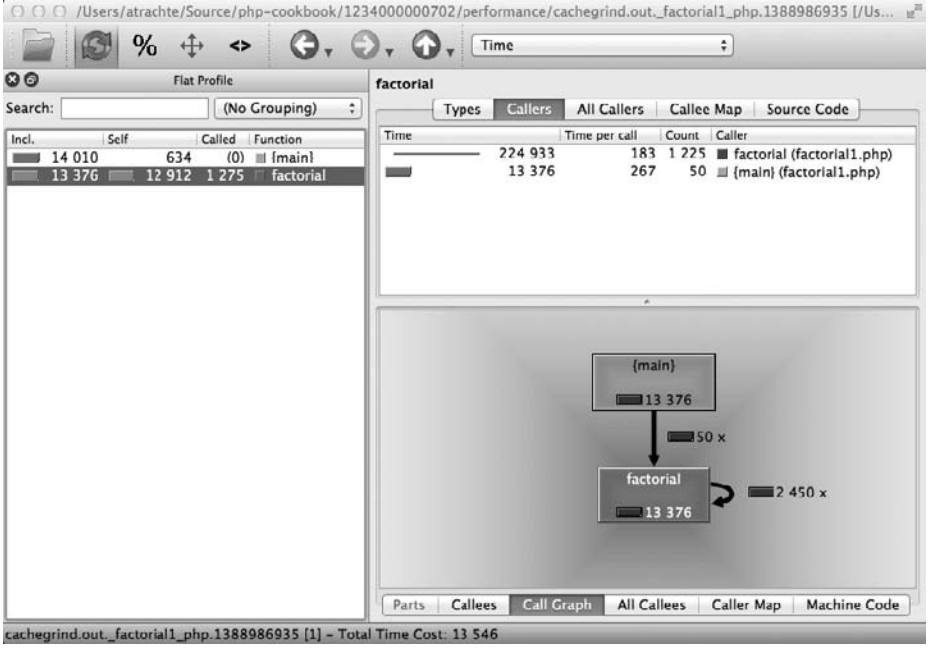

Рис. 22.1. Анализ результатов профилирования в QCacheGrind

Чтобы сократить затраты на рекурсию, мы добавим простейшую реализацию кэширования:

```
function factorial($x) {
    static $cache = [];
    if (isset($cache[$x])) return $cache[$x];
    $cache[$x] = (($x == 1) ? 1 : $x * factorial($x - 1));
```

```
    return $cache[$x];
}

for ($i = 1; $i <= 50; $i++) {
    print "$i: " . factorial($i) . "\n";
}
```

Теперь при загрузке информации профилирования количество циклов процессора уменьшается на порядок — до 643, как видно из рис. 22.2.

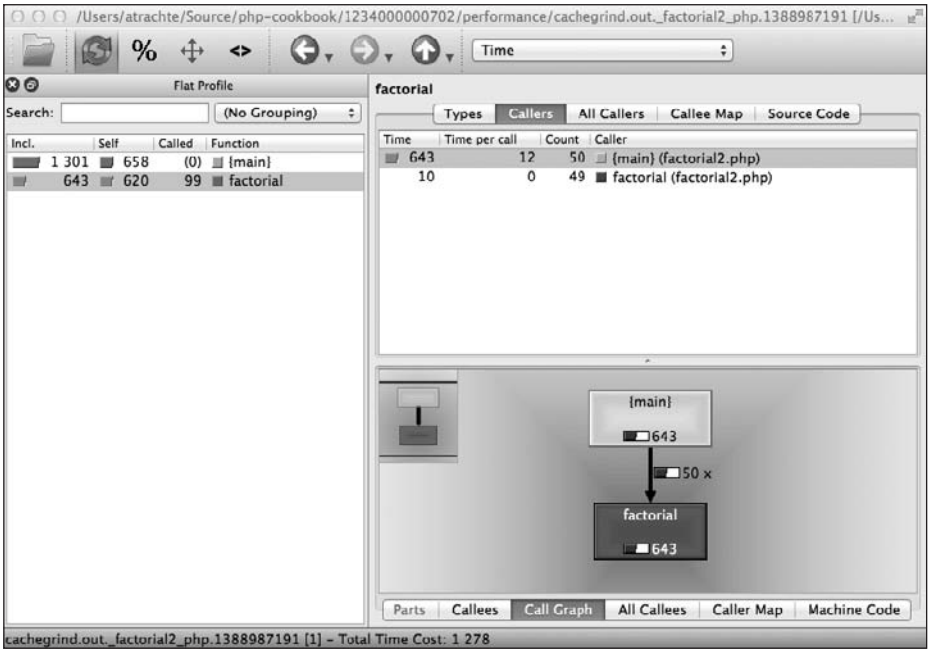

Рис. 22.2. Проверка результатов оптимизации

В зависимости от системы при установке QCachegrind могут возникнуть трудности, потому что эта программа зависит от Qt. Webgrind — приложение, полностью реализованное на PHP, которое разбирает вывод Xdebug. По функциональности оно несколько уступает QCachegrind, но для обычного анализа его достаточно, а устанавливается оно очень просто, потому что представляет собой сценарий PHP.

На рис. 22.3 показано, как выглядят результаты.

См. также

Документация по Xdebug и профилированию; KCachegrind (*http://kcachegrind.sourceforge.net/*); Webgrind (*https://github.com/jokkedk/webgrind*).

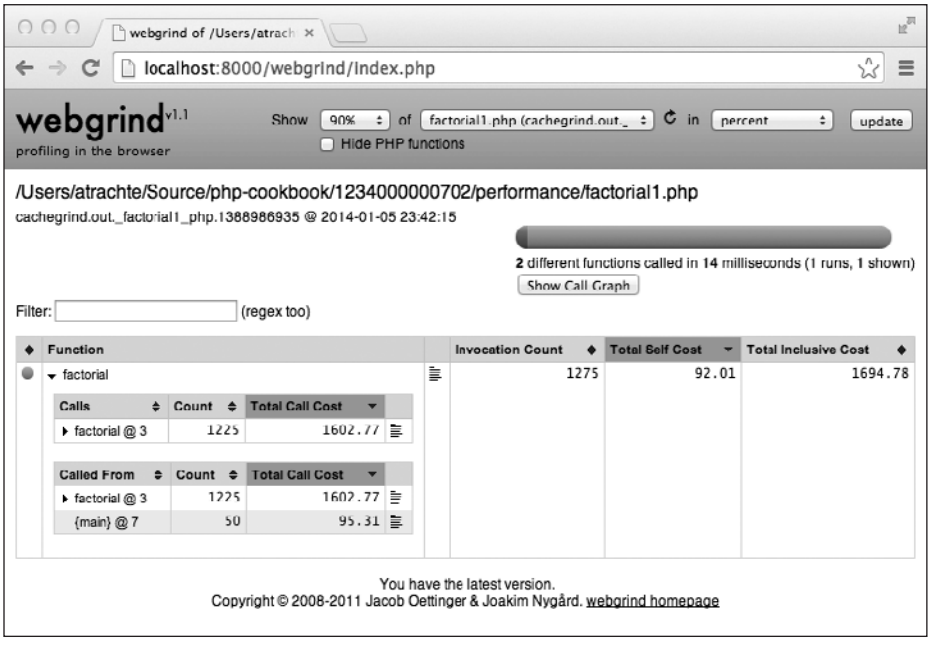

Рис. 22.3. Анализ результатов в Webgrind

22.7. Нагрузочное тестирование сайта

Задача

Требуется проверить, насколько хорошо ваш сайт справляется со значительной нагрузкой.

Решение

Воспользуйтесь средствами нагрузочного и эталонного тестирования для моделирования разных уровней нагрузки.

Комментарий

Нагрузочное тестирование иногда путают с эталонным, но это совершенно разные понятия.

Эталонное тестирование, проводимое отдельным разработчиком, часто представляет собой неформальную деятельность. Чаще всего оно проводится с помощью программы эталонного тестирования HTTP-сервера Apache *ab*, которая должна определять, сколько запросов в секунду может обслуживать сервер. Пример:

```
% /usr/bin/ab -n 1000 -c 100 -k www.example.com/test.php
```

Этот тест возвращает отчет со средним временем ответа на запросы к *http://www.example.com/test.php*, вычисленным на основании 1000 запросов, сгруппированных в пакеты из 100 параллельных запросов.

Хотя такие тесты полезны (они дают разумную оценку того, сколько запросов вы можете обслуживать в секунду при нормальной нагрузке), они ничего не скажут о том, как поведет себя приложение при большой нагрузке. В конце концов, все обращения относятся к одному URL-адресу.

Методология нагрузочного тестирования в конечном счете направлена на то, чтобы вывести ваше веб-приложение из строя. Наращивая нагрузку до критической точки, вы сможете выявить и исправить слабости вашего приложения или поймете, когда нужно добавить дополнительное оборудование. В сочетании с профилированием кода полученная информация также позволяет получить представление о том, какие части приложения должны масштабироваться в первую очередь — иначе говоря, нужно ли включать новые серверы в кластер баз данных, прежде чем устанавливать новые машины для обслуживания взаимодействий клиентов с веб-серверами?

Для проведения нагрузочного тестирования существует превосходная программа с открытым кодом Siege. Ее можно настроить для чтения большого количества URL из конфигурационного файла и их последовательного перебора (регрессионное тестирование) или же для случайных обращений по прочитанному списку URL, что лучше моделирует реальное использование сайта. Siege также может генерировать поток запросов по одному URL-адресу по аналогии с *ab*.

Если установка Siege в вашей системе невозможна, написанный Линкольном Стейном (Lincoln Stein) сценарий torture.pl может стать хорошей альтернативой. Многие концепции проекта Siege были вдохновлены torture.pl, и эти два инструмента выдают похожие отчеты.

См. также

Исходный код и документация Siege (*http://www.joedog.org/2013/07/siege-3-0-3-url-encoding/*); описание *ab* (*http://httpd.apache.org/docs/2.0/programs/ab.html*); исходный код и документация torture.pl (*http://stein.cshl.org/~lstein/torture/*).

22.8. Альтернативы для регулярных выражений

Задача

Требуется повысить быстродействие сценария за счет оптимизации поиска совпадений в строках.

Решение

Замените лишние использования регулярных выражений более быстрыми альтернативными строковыми функциями и функциями проверки символов.

Комментарий

Типичная причина лишних вычислений — применение функций регулярных выражений там, где можно обойтись без них. Допустим, вы проверяете имя пользователя в данных формы на правильность и хотите убедиться в том, что оно состоит только из алфавитно-цифровых символов.

Обычно для решения подобных задач используется регулярное выражение:

```
if (!preg_match('/^[a-z0-9]+$/i', $username)) {
  echo 'please enter a valid username.';
}
```

Ту же проверку можно выполнить намного быстрее при помощи функции `ctype_alnum()`.

Сравним эти два решения, используя средства хронометража из Рецепта 22.2:

```
$username = 'foo411';

$start = microtime(true);

if (!preg_match('/^[a-z0-9]+/i', $username)) {
    echo 'please enter a valid username';
}

$regextime = microtime(true) - $start;

$start = microtime(true);

if (!ctype_alnum($username)) {
    echo 'please enter a valid username';
}

$ctypetime = microtime(true) - $start;

echo "preg_match took:  $regextime seconds\n";
echo "ctype_alnum took: $ctypetime seconds\n";
```

Примерный результат выглядит так:

```
preg_match took:  0.000163078308105 seconds
ctype_alnum took: 9.05990600586E-06 seconds
```

Функция `ctype_alnum()` работает намного быстрее; запись `9.05990600586E-06` — то же, что 0,00000906 секунды, что в 18 раз быстрее проверки по регулярному выражению `preg_match()` при том же результате.

В сложных приложениях замена лишних регулярных выражений эквивалентными альтернативами может обеспечить существенный прирост быстродействия.

Чтобы понять, нужно или нет применять регулярное выражение, попробуйте сформулировать условие в одной короткой фразе. Конечно, существуют некоторые условия (например, «строка содержит действительный адрес электронной почты»), которые невозможно проверить без сложного регулярного выражения. Однако условие «проверить, содержит ли строка A строку B» можно проверить несколькими разными способами, но в конечном счете задача решается очень простой проверкой, не требующей использования регулярных выражений:

```
$haystack = 'The quick brown fox jumps over the lazy dog';
$needle = 'lazy dog';

// Медленное решение (считается устаревшим)
if (ereg($needle, $haystack)) echo 'match!';

// Медленно
if (preg_match("/$needle/", $haystack)) echo 'match!';

// Быстро
if (strstr($haystack, $needle)) echo 'match!';

// Самое быстрое
if (strpos($haystack, $needle) !== false) echo 'match!';
```

Обязательно припомните `ctype` и строковые функции, прежде чем выбирать решение с регулярным выражением, особенно в многократно выполняемых участках кода.

См. также

Документация по функциям `ctype`, строковым функциям и функциям регулярных выражений.

23 Регулярные выражения

23.0. Введение

Регулярные выражения — мощный и филигранный инструмент для поиска по шаблону и обработки текста. Уступая по скорости простым операциям поиска в строках, регулярные выражения отличаются исключительной гибкостью. Они позволяют строить шаблоны для поиска практически любых мыслимых комбинаций символов в простом (хотя и кратком и насыщенным знаками препинания) синтаксисе. Если ваш сайт зависит от данных, поступающих в текстовых файлах (результаты спортивных соревнований, новости, часто обновляемые заголовки), регулярные выражения помогут быстро извлечь полезную информацию.

Глава начинается с краткого обзора базового синтаксиса регулярных выражений, после чего мы перейдем к функциям PHP для работы с регулярными выражениями. Более подробную информацию о тонкостях регулярных выражений можно найти в описании PCRE в электронной документации PHP, а также в книге «Learning PHP» Дэвида Скляра (издательство O'Reilly). Если же вы захотите достичь вершин мастерства в области регулярных выражений, изучайте наиболее полную книгу «Mastering Regular Expressions» Джеффри Фридла (Jeffrey E.F. Friedl) (издательство O'Reilly).

Регулярные выражения пригодятся при преобразовании простого текста в HTML и наоборот. К счастью, ввиду их полезности в PHP существует много встроенных функций для выполнения типичных операций с регулярными выражениями; они упоминаются в рецептах других глав. Рецепт 9.10 показывает, как экранировать сущности HTML; в Рецепте 13.6 рассматривается удаление тегов HTML; Рецепты 13.4 и 13.5 объясняют, как преобразовать простой текст в разметку HTML, и HTML в простой текст соответственно. За информацией о поиске совпадений и проверке адресов электронной почты обращайтесь к Рецепту 9.4.

За прошедшие годы функциональность регулярных выражений расширялась и дополнялась новыми полезными возможностями. В результате PHP поддерживает две группы функций регулярных выражений. Первая группа включает традиционные функции (или функции POSIX), имена которых начинаются с `ereg`; сами функции `ereg` уже являются расширением исходной функциональности. Другая группа включает семейство Perl-совместимых функций с префиксом `preg`.

Функции `preg` используют библиотеку, которая моделирует функциональность языка программирования Perl. В области регулярных выражений Perl реализовано множество полезных возможностей, включая минимальный поиск совпадения, опережающие и ретроспективные проверки и даже рекурсивные шаблоны.

Сейчас не осталось никаких причин для использования функций `ereg`, и в PHP 5.3.0 они официально считаются устаревшими. Эти функции реализуют меньший набор возможностей и работают медленнее функций `preg`. Тем не менее функции `ereg` существовали в PHP за много лет до появления функций `preg`, поэтому многие программисты продолжают использовать их в унаследованном коде (или просто по привычке). К счастью, прототипы двух групп функций практически идентичны, поэтому переход обойдется без особых сложностей (о том, как сделать это и не столкнуться с типичными проблемами, рассказано в Рецепте 23.1).

Регулярное выражение в программе можно рассматривать как язык программирования с сильно ограниченными возможностями. Единственная задача программы на языке регулярных выражений — поиск совпадений в тексте. В шаблонах регулярных выражений большинство символов представляют сами себя; иначе говоря, для регулярного выражения `rhino` будет найдено совпадение в строке, содержащей последовательность из пяти символов `rhino`. Самые интересные возможности регулярных выражений связаны с использованием разных знаков и служебных символов, называемых *метасимволами*. Эти знаки не совпадают сами с собой, а дают указания ядру поиска совпадений.

К числу наиболее часто используемых метасимволов относятся точка (`.`), звездочка (`*`), знак «плюс» (`+`) и вопросительный знак (`?`). (Чтобы отключить специальную интерпретацию метасимвола, то есть искать совпадение для литерала, поставьте перед ним символ `\`.)

- Точка обозначает «любой символ»; таким образом, шаблон `.at` совпадет с последовательностью `bat`, `cat` или `rat`.
- Звездочка обозначает «0 или более вхождений предыдущего объекта». (Пока мы не знаем о других объектах, кроме символов.)
- Плюс похож по смыслу на звездочку, но обозначает «1 или более вхождений предыдущего объекта». Таким образом, шаблон `.+at` совпадет с `brat`, `sprat` и даже подстрокой `cat` внутри `catastrophe`, но не с простым `at`. Чтобы обнаружить совпадение для `at`, замените `+` символом `*`.

- Вопросительный знак сообщает: «Предыдущий объект не является обязательным». Иначе говоря, он совпадает с 0 или 1 вхождением предыдущего объекта. Так, шаблон `colou?r` совпадает как с `color`, так и с `colour`.

Чтобы применить * и + к объектам, размер которых превышает один символ, заключите последовательность символов, образующих объект, в круглые скобки. Скобки позволяют группировать символы для более сложного поиска, а также *захватывают* заключенную в них часть шаблона. Захваченная последовательность может использоваться в `preg_replace()` для изменения строки, а все захваченные совпадения могут быть сохранены в массиве, передаваемом в третьем параметре `preg_match()` и `preg_match_all()`. Функция `preg_match_all()` аналогична `preg_match()`, но она находит все совпадения в строке, не останавливаясь на первом. В листинге 23.1 приведены примеры использования `preg_match()`, `preg_match_all()` и `preg_replace()`.

Листинг 23.1. Использование функций preg

```
if (preg_match('{<title>.+</title>}', $html)) {
    print "The page has a title!\n";
}

if (preg_match_all('/<li>/', $html, $matches)) {
    print 'Page has ' . count($matches[0]) . " list items\n";
}

// Замена жирного шрифта курсивом
$italics = preg_replace('/(<\/?)b(>)/', '$1i$2', $bold);
```

Обычно в качестве ограничителя шаблона (то есть символа, начинающего и завершающего строку шаблона) используется символ `/`. Если ограничитель присутствует как литерал внутри шаблона, он должен экранироваться символом \, поэтому при работе с разметкой HTML или XML такой ограничитель неудобен. В предыдущем коде для решения этой проблемы в первом шаблоне в качестве ограничителей используются парные фигурные скобки. Ограничителем может быть любой символ, не являющийся алфавитно-цифровым или пропуском (кроме \). Если в качестве начального ограничителя используется одна из скобок, то конечным ограничителем должна быть парная скобка.

Если вы хотите искать совпадение для символа из конкретного набора, создайте *символьный класс*, заключив нужные символы в квадратные скобки. Так, символьный класс `[aeiou]` совпадает с одним из перечисленных символов: a, e, i, o или u. Также для определения символьных классов в квадратные скобки могут использоваться диапазоны. Например, класс `[a-z]` представляет любую латинскую букву нижнего регистра, а класс `[a-zA-Z0-9]` — цифру или латинскую букву. Класс `[a-zA-Z0-9_]` совпадает с цифрой, латинской буквой или символом подчеркивания.

Все шаблоны, рассматривавшиеся ранее, никак не ограничивали местонахождения совпадения в тексте. Совпадение шаблона `[a-z0-9]+` будет найдено в строках

grapefruit и c3p0, но также оно будет найдено и в строках grr!!! и *******p. Все четыре строки удовлетворяют критерию, определяемому шаблоном [a-z0-9]+: «Одна или более цифр или латинских букв нижнего регистра».

Якорные метасимволы ограничивают поиск совпадения позициями, которые описываются шаблоном. Так, метасимволы ^ и $ привязывают шаблон к началу и концу строки соответственно. Без них совпадение может быть обнаружено в любой поизции строки. Итак, если [a-z0-9]+ означает «одна или более цифр или латинских букв нижнего регистра», смысл шаблона ^[a-z0-9]+ меняется на «начинается с одной или более цифр или латинских букв нижнего регистра», а [a-z0-9]+$ означает «кончается одной или более цифрами или латинскими буквами нижнего регистра». Наконец, шаблон ^[a-z0-9]+$ означает «содержит только одну или более цифр или латинских букв нижнего регистра». В листинге 23.2 приведены примеры использования символьных классов.

Листинг 23.2. Поиск совпадений с символьными классами и якорными метасимволами

```
$thisFileContents = file_get_contents(__FILE__);
// По адресу http://php.net/language.variables приведено
// регулярное выражение для обозначения действительных имен
// переменных в php. \$ в начале шаблона обозначает литерал $
$matchCount = preg_match_all('/\$[a-zA-Z_\x7f-\xff][a-zA-Z0-9_\x7f-\xff]*/',
                             $thisFileContents, $matches);
print "Matches: $matchCount\n";
foreach ($matches[0] as $variableName) {
    print "$variableName\n";
}
```

Листинг 23.2 выводит все имена используемых переменных:

```
Matches: 8
$thisFileContents
$matchCount
$thisFileContents
$matches
$matchCount
$matches
$variableName
$variableName
```

Иногда интересующий вас набор символов проще определяется по принципу исключения. Символьный класс, начинающийся с символа ^, совпадает с любым символом, кроме входящих в этот класс. Вне символьных классов знак ^ привязывает совпадение к началу строки; в символьном классе он означает «любой символ, кроме перечисленных в квадратных скобках». Например, символьный класс [^aeiou] совпадает с любым символом, кроме гласных латинских букв нижнего регистра.

Не стоит полагать, что символьным классом, обратным по отношению к [aeiou], является [bcdfghjklmnpqrstvwxyz]. Символьный класс [^aeiou] также включает гласные буквы верхнего регистра (AEIOU) и другие знаки.

Вертикальная черта (|) определяет альтернативы. В листинге 23.3 этот символ используется для поиска разных вариантов имен графических файлов в блоке текста.

Листинг 23.3. Поиск с использованием |

```
$text = "The files are cuddly.gif, report.pdf, and cute.jpg.";
if (preg_match_all('/[a-zA-Z0-9]+\.(gif|jpe?g)/',$text,$matches)) {
    print "The image files are: " . implode(',',$matches[0]);
}
```

Пример 23.3 выводит следующий результат:

```
The image files are: cuddly.gif,cute.jpg
```

Мы рассмотрели очень небольшое подмножество мира регулярных выражений. Кое-какие подробности встречаются в следующих рецептах, но на сайте PHP представлена очень полезная информация о Perl-совместимых регулярных выражениях (*http://php.net/pcre*).

23.1. Переход с ereg на preg

Задача

Требуется преобразовать существующий код, использующий функции `ereg`, и перейти на функции `preg`.

Решение

Начните с добавления ограничителей в шаблоны:

```
preg_match('/pattern/', 'string');
```

Для поиска совпадения без учета регистра используйте модификатор `/i` с функцией `preg_match()`:

```
preg_match('/pattern/i', 'string');
```

Если в качестве шаблонов или строк замены используются целые числа, преобразуйте их в шестнадцатеричную форму и определите как служебную последовательность:

```
$hex = dechex($number);
preg_match("/\x$hex/", 'string');
```

Комментарий

Между функциями `ereg` и `preg` существует несколько серьезных отличий. Во-первых, при использовании функций `preg` шаблон не задается в виде обычной

строки (например, `pattern`); он также должен содержать ограничители, как в Perl (например, `/pattern/`[1]). Таким образом, вызов

```
ereg('pattern', 'string');
```

заменяется следующим:

```
preg_match('/pattern/', 'string');
```

При выборе ограничителя следите за тем, чтобы этот символ не встречался внутри регулярного выражения; в противном случае шаблон завершится раньше времени. Если это невозможно, экранируйте все вхождения ограничителя в шаблоне символом `\`. Не делайте это вручную — воспользуйтесь функцией `addcslashes()`.

Например, если ограничителем является символ `/`:

```
$ereg_pattern = '<b>.+</b>';
$preg_pattern = addcslashes($ereg_pattern, '/');
```

то переменная `$preg_pattern` после экранирования содержит `<b>.+<\/b>`.

В семействе `preg` нет параллельного набора функций, работающих без учета регистра символов. Вместо них используется модификатор поиска без учета регистра. Вызов

```
eregi('pattern', 'string');
```

заменяется вызовом:

```
preg_match('/pattern/i', 'string');
```

Модификатор `i` вставляется после закрывающего ограничителя.

Остается упомянуть еще об одном неочевидном различии. Если в качестве шаблона или замены `ereg_replace()` используется число, предполагается, что оно обозначает ASCII-код символа. Таким образом, поскольку 9 является ASCII-кодом символа табуляции (то есть `\t`), следующий фрагмент вставляет символ табуляции в начало строки:

```
$tab = 9;
$replaced = ereg_replace('^', $tab, $string);
```

А замена завершителей строк выполняется так:

```
$converted = ereg_replace(10, 12, $text);
```

---

[1] Или {pattern}, <pattern>, |pattern|, #pattern# — и вообще любые ограничители, какие вам больше нравятся. Если в качестве начального ограничителя используется открывающий парный символ ((, <, [или {), то PHP ожидает, что конечным ограничителем будет другой символ пары (соответственно), >,] или }). Для всех остальных символов PHP ожидает, что конечным ограничителем будет тот же символ.

Чтобы отказаться от этой особенности функций ereg, используйте следующую запись:

```
$tab = '9';
```

С другой стороны, preg_replace() рассматривает число 9 как строку из одного символа '9', а не как заменитель табуляции. Чтобы использовать эти коды символов в preg_replace(), преобразуйте их в шестнадцатеричную форму и снабдите префиксом \x. Например, 9 превращается в \x9 или \x09, а 12 заменяется на \x0c. Также можно использовать комбинации \t , \r и \n для табуляции, возврата курсора и перевода строки соответственно.

См. также

Документация по функциям ereg(), preg_match() и addcslashes().

23.2. Поиск слов

Задача

Требуется извлечь из строки все слова.

Решение

Проще всего воспользоваться служебной комбинацией PCRE \w, обозначающей «символ слова»:

```
$text = "Knock, knock. Who's there? r2d2!";
$words = preg_match_all('/\w+/', $text, $matches);
var_dump($matches[0]);
```

Комментарий

Комбинация \w соответствует буквам, цифрам и символам подчеркивания. Другие знаки и символы в нее не включаются. Таким образом, результат выполнения приведенного кода будет выглядеть так:

```
array(6) {
  [0]=>
  string(5) "Knock"
  [1]=>
  string(5) "knock"
  [2]=>
  string(3) "Who"
  [3]=>
  string(1) "s"
  [4]=>
  string(5) "there"
```

```
  [5]=>
  string(4) "r2d2"
}
```

Результат в основном правилен, если не считать того, что комбинация «Who's» была разделена на «Who» и «s». Чтобы этот шаблон правильно обрабатывал сокращения английского языка, можно искать совпадение для символа слова или для апострофа, заключенного между символами слов:

```
$text = "Knock, knock. Who's there? r2d2!";
$pattern = "/(?:\w'\w|\w)+/";
$words = preg_match_all($pattern, $text, $matches);
var_dump($matches[0]);
```

(Синтаксис `?:` предотвращает «захват» текста, совпавшего с внутренним шаблоном в круглых скобках. Эта тема более подробно рассматривается в Рецепте 23.7.)

Если добавить модификатор `u`, шаблон начинает поддерживать Юникод и правильно распознает слова, выходящие за пределы кодировки ASCII. Пример:

```
$fr = 'Toc, toc. Qui est là? R2D2!';
$fr_words = preg_match_all('/\w+/u', $fr, $matches);
print "The French words are:\n\t";
print implode(', ', $matches[0]) . "\n";

$kr = '노크, 노크, 거기, 누구입니까? R2D2!';
$kr_words = preg_match_all('/\w+/u', $kr, $matches);
print "The Korean words are:\n\t";
print implode(', ', $matches[0]) . "\n";
```

Результат:

```
The French words are:
        Toc, toc, Qui, est, là, R2D2
The Korean words are:
        노크, 노크, 거기, 누구입니까, R2D2
```

Без модификатора `u` в конце шаблона символы, не входящие в ASCII, будут исключаться из совпадений, а результат будет неверным.

См. также

Документация по служебным комбинациям `preg`; Рецепт 19.12 — использование строк в кодировке UTF-8 с функциями регулярных выражений PCRE.

23.3. Поиск n-го совпадения

Задача

Требуется найти *n*-е совпадение слова вместо первого.

Решение

Воспользуйтесь функцией `preg_match_all()`, чтобы получить все совпадения в массиве; затем выберите из массива интересующие вас совпадения, как показано в листинге 23.4.

Листинг 23.4. Поиск n-го совпадения

```
$todo = "1. Get Dressed 2. Eat Jelly 3. Squash every week into a day";

preg_match_all("/\d\. ([^\d]+)/", $todo, $matches);

print "The second item on the todo list is: ";
// $matches[1] - массив всех подстрок, захваченных для ([^\d]+)
print $matches[1][1] . "\n";

print "The entire todo list is: ";
foreach($matches[1] as $match) {
    print "$match\n";
}
```

Комментарий

Так как функция `preg_match()` прекращает поиск после обнаружения первого совпадения, для получения информации о дополнительных совпадениях необходимо использовать функцию `preg_match_all()`. Функция `preg_match_all()` возвращает количество полных совпадений шаблона, обнаруженных в строке. Если не найдено ни одного совпадения, возвращается 0. При обнаружении ошибки (например, неправильного синтаксиса в шаблоне) возвращается `false`.

Третий аргумент `preg_match_all()` заполняется массивом с информацией о различных подстроках, в которых были найдены совпадения шаблона. Первый элемент содержит массив совпадений всего шаблона. Для листинга 23.4 это означает, что элемент `$matches[0]` содержит части `$todo`, совпадающие с `/\d\. ([^\d]+)/`: `1. Get Dressed`, `2. Eat Jelly` и `3. Squash every week into a day`.

Следующие элементы массива `$matches` содержат массивы текста, совпавшего с каждым из внутренних подвыражений, заключенных в круглые скобки. В листинге 23.4 используется всего одно внутреннее подвыражение `([^\d]+)`. Следовательно, `$matches[1]` содержит массив строк, соответствующих шаблону подвыражения: `Get Dressed`, `Eat Jelly` и `Squash every week into a day`.

Если бы шаблон содержал второе подвыражение в круглых скобках, то совпавшие подстроки хранились бы в `$matches[2]`, совпадения третьего подвыражения — в `$matches[3]`, и т. д.

Вместо массива, поделенного по полным и частичным совпадениям, `preg_match_all()` может вернуть массив, поделенный по номерам совпадений: в `$matches[0]` хранится массив первого набора совпадений, в `$matches[1]` — массив второго набора совпадений и т. д. Чтобы включить этот режим, передайте `PREG_SET_ORDER` в четвертом аргументе. Данная возможность особенно полезна, если шаблон

содержит несколько захватывающих подвыражений и вы хотите последовательно перебирать группы подвыражений, как показано в листинге 23.5.

Листинг 23.5. Группировка совпадений подвыражений

```
$todo = "
first=Get Dressed
next=Eat Jelly
last=Squash every week into a day
";

preg_match_all("/([a-zA-Z]+)=(.*)/", $todo, $matches, PREG_SET_ORDER);

foreach ($matches as $match) {
    print "The {$match[1]} action is {$match[2]}\n";
}
```

Листинг 23.5 выводит следующий результат:

```
The first action is Get Dressed
The next action is Eat Jelly
The last action is Squash every week into a day
```

В режиме `PREG_SET_ORDER` каждое значение `$match` в цикле `foreach` содержит совпадения всех подвыражений: `$match[0]` — все совпадение, `$match[1]` — часть перед `=` и `$match[2]` — часть после `=`.

См. также

Документация по функции `preg_match_all()`.

23.4. Выбор между максимальным и минимальным совпадением

Задача

Требуется найти для шаблона наименьшее возможное совпадение (вместо наибольшего).

Решение

Поставьте знак `?` после квантификатора, чтобы изменить эту часть шаблона, как сделано в листинге 23.6.

Листинг 23.6. Квантификатор совпадает с минимально возможным количеством символов

```
// Поиск всех секций <em>emphasized</em>
preg_match_all('@<em>.+?</em>@', $html, $matches);
```

Или воспользуйтесь модификатором U, чтобы переключить все квантификаторы из максимального, «жадного», режима («найти совпадение, содержащее как можно больше символов») в минимальный («найти совпадение, содержащее как можно меньше символов»). Код в листинге 23.7 делает то же, что и код в листинге 23.6.

Листинг 23.7. Поиск совпадения, содержащего как можно меньше символов

```
// Поиск всех секций <em>emphasized</em>
preg_match_all('@<em>.+</em>@U', $html, $matches);
```

Комментарий

По умолчанию все квантификаторы регулярных выражений в PHP работают в максимальном, или «жадном», режиме. Для примера возьмем шаблон `<em>.+</em>`, который обозначает «`<em>`, один или несколько символов, потом снова `</em>`». Этот шаблон применяется к строке `I simply <em>love</em> your <em>work</em>`. Максимальное регулярное выражение найдет одно совпадение: после открывающего элемента `<em>` часть `.+` «поглощает» как можно больше символов, останавливаясь только перед завершающим элементом `</em>`. Часть `.+` совпадает с текстом `love</em> your <em>work`.

С другой стороны, минимальное регулярное выражение находит два совпадения. Первый элемент `<em>` включается в совпадение, как и в предыдущем случае, но затем `.+` останавливается при первой возможности, совпадая только с текстом `love`. Затем начинается поиск второго совпадения, в котором `.+` совпадает с текстом `work`.

Листинг 23.8 демонстрирует, как работают максимальные и минимальные шаблоны.

Листинг 23.8. Минимальный и максимальный поиск совпадений

```
$html = 'I simply <em>love</em> your <em>work</em>';
// Максимальный
$matchCount = preg_match_all('@<em>.+</em>@', $html, $matches);
print "Greedy count: " . $matchCount . "\n";
// Минимальный
$matchCount = preg_match_all('@<em>.+?</em>@', $html, $matches);
print "First non-greedy count: " . $matchCount . "\n";
// Минимальный
$matchCount = preg_match_all('@<em>.+</em>@U', $html, $matches);
print "Second non-greedy count: " . $matchCount . "\n";
```

Код листинга 23.8 выводит следующий результат:

```
Greedy count: 1
First non-greedy count: 2
Second non-greedy count: 2
```

Функции `ereg()` и `ereg_replace()` всегда работают в максимальном режиме. Возможность выбора между максимальным и минимальным поиском совпадений — еще одна причина для перехода на функции PCRE.

Минимальный поиск удобен для простейшего разбора разметки HTML, но он может «сломаться», если разметка не является полностью корректной, например если в ней встречаются непарные теги `<em>`[1]. Если вам потребовалось удалить все теги HTML (или их часть) из блока текста, регулярное выражение лучше не использовать. Вместо этого примените функцию `strip_tags()`; она работает быстрее и без ошибок. За подробностями обращайтесь к Рецепту 13.6.

Наконец, хотя идея минимального поиска совпадений пришла из Perl, модификатор `U` несовместим с Perl; это уникальная возможность Perl-совместимых регулярных выражений PHP. Он инвертирует все квантификаторы: максимальный режим поиска переключается на минимальный, и наоборот. Таким образом, чтобы реализовать максимальный поиск внутри шаблона с завершающим модификатором `/U`, добавьте после квантификатора знак `?` — по аналогии с тем, как максимальный квантификатор обычно превращается в минимальный.

См. также

Рецепт 23.6 — выделение текста из тегов HTML; Рецепт 13.6 — удаление тегов HTML; документация по функции `preg_match_all()`.

23.5. Поиск в файле всех строк, соответствующих шаблону

Задача

Требуется найти в файле все строки, содержащие совпадения шаблона.

Решение

Прочитайте файл в массив и используйте функцию `preg_grep()`.

Комментарий

Задачу можно решить двумя способами. Листинг 23.9 работает быстрее, но использует больше памяти. Функция `file()` помещает каждую строку файла в массив, а функция `preg_grep()` отфильтровывает неподходящие строки.

---

[1] Неприятности возможны даже при корректной разметке HTML, например если теги <b> встречаются в комментариях. Нормальный парсер HTML их проигнорирует, но наш шаблон попытается их обработать.

Листинг 23.9. Быстрый поиск строк с совпадениями шаблона

```
$pattern = "/\bo'reilly\b/i"; // Только книги O'Reilly
$ora_books = preg_grep($pattern, file('/path/to/your/file.txt'));
```

Листинг 23.10 работает медленнее, но более эффективно расходует память. Файл читается по одной строке, а функция `preg_match()` проверяет каждую прочитанную строку.

Листинг 23.10. Эффективный поиск строк с совпадениями шаблона

```
$fh = fopen('/path/to/your/file.txt', 'r') or die($php_errormsg);
while (!feof($fh)) {
    $line = fgets($fh);
    if (preg_match($pattern, $line)) { $ora_books[ ] = $line; }
}
fclose($fh);
```

Поскольку код из листинга 23.9 читает все данные сразу, он работает примерно втрое быстрее листинга 23.10, который разбирает файл строку за строкой, но расходует меньше памяти. Помните, что оба способа работают с отдельными строками файла, и они не могут использовать шаблоны для поиска текста, занимающего несколько строк.

См. также

Рецепт 24.5 — чтение файлов в строки; документация по функции `preg_grep()`.

23.6. Выделение текста из тегов HTML

Задача

Требуется выделить текст из тегов HTML. Например, вы хотите найти содержимое всех заголовочных тегов в документе HTML.

Решение

Прочитайте файл HTML в строку в минимальном режиме поиска совпадений, как показано в листинге 23.11.

Листинг 23.11. Выделение заголовков HTML

```
$html = file_get_contents(__DIR__ . '/example.html');
preg_match_all('@<h([1-6])>(.+?)</h\1>@is', $html, $matches);
foreach ($matches[2] as $text) {
    print "Heading: $text\n";
}
```

Комментарий

Организовать безопасный разбор HTML с использованием простых регулярных выражений достаточно сложно. Кстати, одним из ключевых преимуществ XHTML как раз и является простота проверки и разбора данных.

Например, шаблон в листинге 23.11 не справляется с атрибутами в тегах заголовков, а его возможностей хватает только для поиска парных заголовков, так что текст `<h1>Dr. Strangelove</h1>` будет обработан нормально, потому что он заключен в теги `<h1></h1>`, а с текстом `<h2>How I Learned to Stop Worrying and Love the Bomb</h3>` возникнут проблемы из-за несовпадения открывающего тега `<h2>` с закрывающим тегом.

Этот способ также может использоваться для поиска всего текста в правильно сконструированных тегах `<strong>` и `<em>`, как показано в листинге 23.12.

Листинг 23.12. Извлечение текста из тегов HTML

```
$html = file_get_contents(__DIR__.'/example.html');
preg_match_all('@<(strong|em)>(.+?)</\1>@is', $html, $matches);
foreach ($matches[2] as $text) {
    print "Text: $text\n";
}
```

Однако листинг 23.12 не работает с вложенными заголовками. Если файл example.html содержит `<strong>Dr. Strangelove or: <em>How I Learned to Stop Worrying and Love the Bomb</em></strong>`, листинг 23.12 не сохраняет текст в тегах `<em></em>` в отдельном элементе.

В листинге 23.11 это не создает проблем: заголовки являются элементами блочного уровня, их вложение запрещается. Однако вложение тегов `<strong>` и `<em>` (как строковых (inline) элементов) допустимо.

Регулярные выражения могут быть умеренно полезны при разборе небольших объемов разметки HTML, особенно если структура HTML относительно жестко ограничена (или вы генерируете разметку самостоятельно). Для реализации более универсального и мощного разбора HTML используйте расширение Tidy, предоставляющее интерфейс к популярной библиотеке чистки HTML libtidy. После того как Tidy проведет чистку HTML, вы сможете использовать методы для обращения к частям документа. Или, если приказать Tidy преобразовать HTML в XHTML, вы сможете использовать всю мощь SimpleXML или расширения DOM для того, чтобы выполнять с документом HTML любые нужные операции.

См. также

Рецепт 13.1 — разметка веб-страниц; Рецепт 13.3 — извлечение ссылок из файла HTML; документация по функции `preg_match()` и Tidy.

23.7. Незахватывающие круглые скобки

Задача

Круглые скобки используются в шаблоне для группировки, но текст, совпавший с подвыражением в круглых скобках, не должен появляться в массиве совпадений.

Решение

Поставьте `?:` после открывающей круглой скобки, как показано в листинге 23.13.

Листинг 23.13. Отказ от захвата

```
$html = '<link rel="icon" href="http://www.example.com/icon.gif"/>
<link rel="prev" href="http://www.example.com/prev.xml"/>
<link rel="next" href="http://www.example.com/next.xml"/>';

preg_match_all('/rel="(prev|next)" href="([^"]*?)"/', $html, $bothMatches);
preg_match_all('/rel="(?:prev|next)" href="([^"]*?)"/', $html, $linkMatches);

print '$bothMatches is: '; var_dump($bothMatches);
print '$linkMatches is: '; var_dump($linkMatches);
```

В листинге 23.13 `$bothMatches` содержит значения атрибутов `rel` и `href`. С другой стороны, `$linkMatches` содержит только значения атрибутов `href`. Результат выглядит так:

```
$bothMatches is: array(3) {
  [0]=>
  array(2) {
    [0]=>
    string(49) "rel="prev" href="http://www.example.com/prev.xml""
    [1]=>
    string(49) "rel="next" href="http://www.example.com/next.xml""
  }
  [1]=>
  array(2) {
    [0]=>
    string(4) "prev"
    [1]=>
    string(4) "next"
  }
  [2]=>
  array(2) {
    [0]=>
    string(31) "http://www.example.com/prev.xml"
    [1]=>
    string(31) "http://www.example.com/next.xml"
  }
}
$linkMatches is: array(2) {
  [0]=>
```

```
  array(2) {
    [0]=>
    string(49) "rel="prev" href="http://www.example.com/prev.xml""
    [1]=>
    string(49) "rel="next" href="http://www.example.com/next.xml""
  }
  [1]=>
  array(2) {
    [0]=>
    string(31) "http://www.example.com/prev.xml"
    [1]=>
    string(31) "http://www.example.com/next.xml"
  }
}
```

Комментарий

Отказ от захвата особенно полезен для необязательных подвыражений. Поскольку такие выражения могут отсутствовать в массиве совпавшего текста, необязательное подвыражение может изменить количество фрагментов совпавшего текста, а это усложняет ссылки на конкретный фрагмент совпавшего текста по индексу. Отказ от захвата необязательных подвыражений решает эту проблему. Суть различия представлена в листинге 23.14.

Листинг 23.14. Необязательное подвыражение без сохранения

```
$html = '<link rel="icon" href="http://www.example.com/icon.gif"/>
<link rel="prev" title="Previous" href="http://www.example.com/prev.xml"/>
<link rel="next" href="http://www.example.com/next.xml"/>';
preg_match_all('/rel="(?:prev|next)"(?: title="[^"]+?")? href="([^"]*?)"/',
               $html, $linkMatches);

print '$bothMatches is: '; var_dump($linkMatches);
```

См. также

Документация по синтаксису шаблонов PCRE (*http://php.net/reference.pcre.pattern.syntax*).

23.8. Экранирование специальных символов в регулярных выражениях

Задача

Требуется, чтобы символы * или + интерпретировались в регулярных выражениях как литералы, а не метасимволы. Например, это может быть полезно при вводе пользователем строк, которые должны использоваться в регулярных выражениях.

Решение

Используйте функцию `preg_quote()` для экранирования метасимволов PCRE:

```
$pattern = preg_quote('The Education of H*Y*M*A*N K*A*P*L*A*N').':(\d+)';
if (preg_match("/$pattern/",$book_rank,$matches)) {
    print "Leo Rosten's book ranked: ".$matches[1];
}
```

Комментарий

Символы, которые экранируются функцией `preg_quote()`:

```
. \ + * ? ^ $ [ ] () { } < > = ! | :
```

Метасимволы экранируются символом \.

Также `preg_quote()` можно передать во втором аргументе дополнительный символ, который также должен экранироваться. В этом аргументе часто передается ограничитель шаблона (обычно `/`), чтобы он тоже экранировался. Это может быть важно при включении данных, введенных пользователем, в шаблон регулярного выражения. Следующий код получает от веб-формы `$_GET['search_term']` и ищет в строке `$s` слова, начинающиеся с `$_GET['search_term']`:

```
$search_term = preg_quote($_GET['search_term'],'/');
if (preg_match("/\b$search_term/i",$s)) {
   print 'match!';
}
```

Использование `preg_quote()` обеспечивает правильную интерпретацию регулярного выражения, если, например, в условии поиска будет введена строка `t.c`. Без вызова `preg_quote()` совпадения будут найдены в словах `tic`, `tucker` и всех остальных словах, в которых первой буквой является «t», а третьей — «c». Передача ограничителя `preg_quote()` также гарантирует правильность обработки введенных пользователем данных, содержащих символ `/` (например, «CP/M»).

См. также

Документация по `preg_quote()`.

23.9. Чтение записей с разделителями-шаблонами

Задача

Требуется прочитать записи из файла. Записи разделяются шаблоном, для поиска которого может использоваться регулярное выражение.

Решение

Прочитайте весь файл в строку, а затем выполните разбивку по регулярному выражению:

```
$contents = file_get_contents('/path/to/your/file.txt');
$records = preg_split('/[0-9]+\) /', $contents);
```

Комментарий

Приведенный фрагмент разбивает нумерованный список и размещает его отдельные элементы в элементах массива. Итак, для списка вида:

```
1) Gödel
2) Escher
3) Bach
```

будет создан массив из четырех элементов, с пустым открывающим элементом. Дело в том, что `preg_split()` считает, что ограничители находятся *между* элементами, а в данном случае они располагаются *перед* ними:

```
array(4) {
  [0]=>
  string(0) ""
  [1]=>
  string(7) "Gödel
"
  [2]=>
  string(7) "Escher
"
  [3]=>
  string(5) "Bach
"
}
```

Возможно, кто-то сочтет такое поведение полезным — *n*-й элемент содержит *n*-й элемент списка. Но чтобы сжать массив, из него необходимо удалить первый элемент:

```
$records = preg_split('/[0-9]+\) /', $contents);
array_shift($records);
```

Другая возможная модификация — удаление новых строк из элементов и замена их пустой строкой:

```
$records = preg_split('/[0-9]+\) /', str_replace("\n",'',$contents));
array_shift($records);
```

PHP не позволяет назначить разделитель входных записей чем-либо, кроме символа новой строки, поэтому этот прием также полезен для разбиения записей, разделенных фиксированным текстом. С другой стороны, если вам часто приходится разбивать текст по строкам вместо регулярных выражений, замените `preg_split()` на `explode()` для повышения эффективности операции.

См. также

Рецепт 24.5 — чтение из файла; Рецепт 1.12 — разбор файлов в формате CSV.

23.10. Использование функции PHP в регулярном выражении

Задача

Требуется обработать текст совпадения функцией PHP. Например, требуется декодировать все сущности HTML в захваченных подвыражениях.

Решение

Воспользуйтесь функцией `preg_replace_callback()`. Вместо шаблона замены функции передается функция обратного вызова. Эта функция получает массив совпадений и должна вернуть соответствующую строку замены. Листинг 23.15 выполняет декодирование сущностей между тегами `<code></code>`.

Листинг 23.15. Генерирование строк замены функцией обратного вызова

```
$h = 'The &lt;b&gt; tag makes text bold: <code>&lt;b&gt;bold&lt;/b&gt;</code>';
print preg_replace_callback('@<code>(.*?)</code>@','decode', $h);

// $matches[0] - полный текст совпадения
// $matches[1] - первое захваченное подвыражение
function decode($matches) {
    return html_entity_decode($matches[1]);
}
```

Код листинга 23.15 выводит следующий результат:

```
The <b> tag makes text bold: <b>bold</b>
```

Комментарий

Второй аргумент `preg_replace_callback()` задает функцию, которая будет вызываться для вычисления строк замены. Как обычно при использовании псевдотипа `callable` в PHP, аргументом может быть строка или массив. Строка задает имя функции. Чтобы использовать в качестве функции обратного вызова метод объекта, передайте массив, первый элемент которого содержит объект, а второй — строку с именем метода. Чтобы использовать в качестве функции обратного вызова статический метод класса, передайте массив из двух строк: имени класса и имени метода. В PHP 5.4.0 и выше можно передать переменную, содержащую анонимную функцию, или определить функцию «на месте» в вызове `preg_replace_callback()`.

Функция обратного вызова получает один аргумент: массив совпадений. Элемент 0 этого массива всегда содержит текст совпадения всего шаблона. Если шаблон, переданный `preg_replace_callback()`, содержит подвыражения в круглых скобках, эти совпадения размещаются в последующих элементах массива `$matches`. Ключи массива `$matches` являются числовыми, даже если шаблон содержит именованные подвыражения.

Определение анонимной функции для обратного вызова может привести к большим затратам памяти, если функция определяется внутри вызова `preg_replace_callback()`, находящегося в цикле. Если вы хотите использовать анонимную функцию с `preg_replace_callback()`, сохраните ее в переменной и передайте переменную `preg_replace_callback()` в качестве функции обратного вызова. Листинг 23.16 использует анонимную функцию для применения преобразования из листинга 23.15 к каждой строке в файле.

Листинг 23.16. Генерирование строк замены анонимной функцией

```
$callbackFunction = function($matches) {
    return html_entity_decode($matches[1]);
};

$fp = fopen(__DIR__ . '/html-to-decode.html','r');
while (! feof($fp)) {
    $line = fgets($fp);
    print preg_replace_callback('@<code>(.*?)</code>@',$callbackFunction, $line);
}
fclose($fp);
```

В листинге 23.16 используется синтаксис объявлений анонимных функций, появившийся в PHP 5.3.0. Если вы используете более старую версию PHP, воспользуйтесь `create_function()` для создания функции обратного вызова:

```
$callbackFunction = create_function('$matches',
                        'return html_entity_decode($matches[1]);');
$fp = fopen(__DIR__ . '/html-to-decode.html','r');
while (! feof($fp)) {
    $line = fgets($fp);
    print preg_replace_callback('@<code>(.*?)</code>@',$callbackFunction, $line);
}
fclose($fp);
```

Также следует знать о существовании модификатора `e` (в основном чтобы держаться от него подальше). С ним строка замены интерпретируется как код PHP. В PHP 5.5.0 этот модификатор был объявлен устаревшим, и использовать его не рекомендуется из-за проблем безопасности, связанных со включением пользовательского ввода в текст, с которым работает `preg_replace()`. Если вы встретите код, использующий `preg_replace()` с модификатором `e`, преобразуйте его в код с использованием `preg_replace_callback()`.

См. также

Документация по функциям `preg_replace_callback()`, `preg_replace()`, `create_function()` и псевдотипу `callable`.

24 Файлы

24.0. Введение

Входные и выходные данные веб-приложения обычно передаются между браузером, сервером и базой данных, но в некоторых ситуациях также используется файловая система. Файлы могут использоваться для загрузки удаленных веб-страниц с целью локальной обработки, сохранения информации без базы данных, а также информации, доступ к которой нужен другим программам. Кроме того, по мере развития PHP и выхода за рамки простой поставки веб-страниц, функции файлового ввода/вывода становятся еще более полезными.

Интерфейс файлового ввода/вывода PHP напоминает интерфейс C, хотя и в упрощенном варианте. Основным идентификатором файла для чтения и записи является *файловый дескриптор*. Он определяет связь программы с конкретным файлом и используется для выполнения операций с этим файлом.

В этой главе основное внимание уделяется открытию и закрытию файлов, операциям с файловыми дескрипторами в PHP, а также говорится о том, что можно делать с содержимым файла при его открытии. В главе 25 рассматривается работа с каталогами и метаданными файлов (например, разрешениями).

В следующем примере открывается файл /tmp/cookie-data, в который записывается содержимое конкретного cookie:

```
$fh = fopen('/tmp/cookie-data','w')      or die("can't open file");
if (-1 == fwrite($fh,$_COOKIE['flavor'])) { die("can't write data"); }
fclose($fh)                              or die("can't close file");
```

Функция `fopen()` возвращает файловый дескриптор, если попытка открытия файла оказалась успешной. Если файл открыть не удалось (например, из-за неправильно заданных разрешений), функция возвращает `false` и генерирует ошибку типа `E_WARNING`. Способы открытия файлов рассматриваются в Рецептах 24.1–24.3.

В приведенном примере функция `fwrite()` записывает значение cookie `flavor` через файловый дескриптор. Функция возвращает количество записанных байтов. Если попытка записи завершилась неудачей (например, из-за нехватки дискового пространства), возвращается –1.

Наконец, функция `fclose()` закрывает файловый дескриптор. В конце обработки запроса это происходит автоматически, но обычно рекомендуется явно закрывать все файлы, чтобы избежать проблем при использовании кода в контексте командной строки и освободить системные ресурсы. Кроме того, явное закрытие позволяет проверить возвращаемое значение `fclose()`. Буферизованные данные могут записываться на диск только при вызове `fclose()`, поэтому иногда в этот момент появляются ошибки заполнения диска.

Для работы с файлом PHP необходимы разрешения на чтение и запись в файл. В контексте командной строки эта задача обычно решается тривиально, но при запуске сценариев из веб-сервера может возникнуть путаница. Ваш веб-сервер (а соответственно, и сценарий PHP), вероятно, выполняется с правами конкретного пользователя, созданного специально для этой цели (а может, пользователя `nobody`). По соображениям безопасности разрешения этого пользователя на доступ к файлам часто ограничиваются. Если у вашего сценария возникают проблемы с файловыми операциями, убедитесь в том, что у пользователя или группы веб-сервера (а не у вас!) имеются разрешения на выполнение этой операции. Впрочем, некоторые конфигурации веб-сервера могут выполнять сценарии от вашего имени; в этом случае проследите за тем, чтобы сценарии не могли читать или записывать посторонние файлы, не являющиеся частью сайта.

Так как многие функции для работы с файлами при возникновении ошибки просто возвращают `false`, вам придется потрудиться для получения дополнительной информации об ошибке. При включенной конфигурационной директиве `track_errors` каждое сообщение об ошибке помещается в глобальную переменную `$php_errormsg`. Включение этой переменной в сообщение об ошибке упрощает отладку:

```
$fh = fopen('/tmp/cookie-data','w')      or die("can't open: $php_errormsg");
if (-1 == fwrite($fh,$_COOKIE['flavor'])) { die("can't write: $php_errormsg"); }
fclose($fh)                              or die("can't close: $php_errormsg");
```

При отсутствии разрешения записи в /tmp/cookie-data код аварийно завершается со следующим сообщением:

```
can't open: fopen(/tmp/cookie-data): failed to open stream: Permission denied
```

Windows и Unix используют разные правила при операциях с файлами. Чтобы код обращения к файлам правильно работал в обеих системах, обеспечьте правильную обработку завершителей строк и путей.

Завершитель строки в Windows состоит из двух символов: ASCII 13 (возврат курсора) и ASCII 10 (перевод строки, или новая строка). В Unix он состоит только из символа ASCII 10. Названия символов, появившиеся еще в эпоху пишущих машинок, объясняют, почему при выводе файла с завершителями Unix может возникнуть эффект «ступенчатого» текста. Представьте, что эти имена символов

содержат команды для каретки пишущей машинки или текстового принтера. Возврат курсора сдвигает каретку в начало текущей строки, а перевод строки смещает бумагу на одну строку. Если принтер настроен неправильно, то при обнаружении файла с завершителями Unix он послушно выполнит инструкции и в конце строки перейдет к следующей строке. При этом горизонтальная позиция печати не возвращается к левому краю листа. Следующая строка текста начнется (по горизонтали) там, где завершилась предыдущая строка.

Функции PHP, использующие символ новой строки в качестве завершителя (например, `fgets()`), работают как в Unix, так и в Windows, потому что этот символ находится в конце строки на обеих платформах.

Для удаления символов-завершителей используется функция PHP `rtrim()`:

```
$fh = fopen('/tmp/lines-of-data.txt','r') or die($php_errormsg);
while(false !== ($s = fgets($fh))) {
    $s = rtrim($s);
    // Операции с $s ...
}
fclose($fh)                                or die($php_errormsg);
```

Функция удаляет все конечные пустоты в строке, включая ASCII 13 и ASCII 10 (а также табуляции и пробелы). Если в конце строки имеется пропуск, который бы вы предпочли сохранить, но при этом все равно желаете удалить символы возврата курсора и новой строки, передайте `rtrim()` строку с перечнем удаляемых символов. Другие символы остаются в строке, как показано в следующем фрагменте:

```
$fh = fopen('/tmp/lines-of-data.txt','r') or die($php_errormsg);
while(false !== ($s = fgets($fh))) {
    $s = rtrim($s, "\r\n");
    // Операции с $s ...
}
fclose($fh)                                or die($php_errormsg);
```

В Unix и Windows также используются разные символы для разделения каталогов в путях. В Unix используется символ / (косая черта), а в Windows — символ \ (обратная косая черта). Впрочем, в PHP эта проблема решается легко, потому что версия PHP для Windows также поддерживает / как разделитель каталогов. Например, следующий фрагмент успешно выводит содержимое файла C:\Alligator\Crocodile Menu.txt:

```
$fh = fopen('c:/alligator/crocodile menu.txt','r') or die($php_errormsg);
while(false !== ($s = fgets($fh))) {
    print $s;
}
fclose($fh)                                      or die($php_errormsg);
```

Также здесь используется тот факт, что в именах файлов Windows не учитывается регистр символов — в отличие от имен файлов Unix.

Проблема разбивки строк может быть актуальной не только для кода чтения и записи файлов, но и для исходного кода ваших программ. Если над проектом

работает большая группа разработчиков, проследите за тем, чтобы у всех участников редакторы были настроены на одинаковые разрывы строк.

После открытия файла PHP предоставляет много полезных средств для обработки данных. Как и в интерфейсе ввода/вывода языка C, который послужил прототипом для PHP, для чтения данных из файла используются две функции: `fread()` читает заданное количество байтов, а `fgets()` читает их по строкам (с возможным ограничением максимальной длины в байтах). Следующий пример работает со строками длиной до 256 байт:

```
$fh = fopen('orders.txt','r') or die($php_errormsg);
while (! feof($fh)) {
    $s = fgets($fh,256);
    process_order($s);
}
fclose($fh)                    or die($php_errormsg);
```

Если файл orders.txt содержит строку из 300 байт, `fgets()` вернет только первые 256 байт. Следующий вызов `fgets()` вернет следующие 44 байта и остановится на символе новой строки. Еще один вызов `fgets()` перейдет к следующей строке файла. Без второго аргумента `fgets()` читает данные, пока не будет достигнут конец строки.

Многие операции с содержимым файлов (например, случайный выбор строки — см. Рецепт 24.8) концептуально упрощаются (и требуют меньшего объема кода) при загрузке всего файла в строку или массив. Функция `file_get_contents()` читает весь файл в строку, а функция `file()` помещает каждую строку файла в массив. С другой стороны, за упрощение кода приходится расплачиваться повышенными затратами памяти. Это может быть особенно неприятно при использовании PHP в качестве серверного модуля.

Как правило, когда процесс (например, процесс веб-сервера со встроенным PHP) выделяет память (как это делает PHP для чтения всего файла в строку или массив), эта память не возвращается операционной системе вплоть до завершения процесса. Это означает, что вызов `file_get_contents()` для 1-мегабайтного файла из кода PHP, выполняемого в качестве модуля Apache, увеличивает размер процесса Apache на 1 мегабайт вплоть до его завершения. Повторение этой операции снижает эффективность работы сервера. Конечно, для обработки всего файла как единого целого есть веские причины, но если вы действуете подобным образом, помните о возрастающих затратах памяти.

Если вы предпочитаете избежать загрузки всего файла в память, генераторы позволяют элегантно инкапсулировать логику перебора содержимого файла. Рецепт 4.24 показывает, как организовать перебор строк файла так, словно они хранятся в массиве, но при этом в память будет загружаться только одна строка.

Рецепты 24.17–24.19 относятся к запуску других программ из PHP. Некоторые функции и операторы выполнения программ предоставляют возможность запустить программу и прочитать весь ее вывод сразу (обратные апострофы) или прочитать последнюю строку вывода (`system()`). PHP позволяет использовать

каналы (pipes) для запуска программы с передачей ей входных данных или чтением ее вывода. Так как для чтения каналов используются стандартные функции ввода/вывода (`fgets()` и `fread()`), вы сами решаете, как организовать получение входных данных, а между чтением фрагментов ввода можно выполнять другие операции. Аналогичным образом запись в канал выполняется вызовами `fputs()` и `fwrite()`, так что входные данные другим программам можно передавать с произвольной детализацией.

С каналами возникают те же проблемы разрешений, что и с обычными файлами. Процесс PHP должен обладать разрешением на выполнение программы, открываемой через канал. Если у вас возникают проблемы с открытием каналов (особенно если PHP работает с правами специального пользователя веб-сервера), убедитесь в том, что этому пользователю разрешено выполнение программы, для которой открывается канал.

24.1. Создание или открытие локального файла

Задача

Требуется открыть локальный файл для чтения или записи данных.

Решение

Воспользуйтесь функцией `fopen()`:

```
$fh = fopen('file.txt','r') or die("can't open file.txt: $php_errormsg");
```

Комментарий

В первом аргументе `fopen()` передается открываемый файл; второй аргумент содержит режим открытия файла. Режим определяет, какие операции могут выполняться с файлом (чтение и/или запись), в какую позицию устанавливается файловый указатель после открытия файла (в начало или в конец файла), должен ли файл усекаться до нулевой длины после открытия и должен ли файл создаваться, если он не существует. Режимы открытия файлов перечислены в таблице 24.1.

В режимах `x` и `x+` для уже существующих файлов функция возвращает `false` и генерирует предупреждение. В системах, не поддерживающих стандарт POSIX (например, Windows), при открытии двоичного файла к режиму добавляется символ `b`; в противном случае операции чтения и записи будут сбиваться на символах NUL (ASCII 0). Пример:

```
$fh = fopen('c:/images/logo.gif','rb');
```

Таблица 24.1. Режимы открытия файлов функцией fopen()

Режим	Чтение	Запись	Файловый указатель	Усечение	Создание
r	Да	Нет	Начало	Нет	Нет
r+	Да	Да	Начало	Нет	Нет
w	Нет	Да	Начало	Да	Да
w+	Да	Да	Начало	Да	Да
a	Нет	Да	Конец	Нет	Да
a+	Да	Да	Конец	Нет	Да
x	Нет	Да	Начало	Нет	Да
x+	Да	Да	Начало	Нет	Да

И хотя в системах семейства Unix двоичные файлы правильно обрабатываются и без включения `b` в режим, все равно рекомендуется добавлять его. Тем самым обеспечивается универсальность кода и возможность его выполнения в Unix и Windows.

Для выполнения операций с файлом дескриптор, возвращенный `fopen()`, передается другим функциям ввода/вывода — таким как `fgets()`, `fputs()` и `fclose()`.

Если файл передается `fopen()` без указания пути, то файл открывается в каталоге выполняемого сценария (веб-контекст) или текущем каталоге (контекст командной строки). Также можно приказать `fopen()` искать открываемый файл в пути `include_path`, заданном в файле php.ini; для этого в третьем аргументе передается `true`. В следующем примере файл file.inc ищется в каталогах `include_path`:

```
$fh = fopen('file.inc','r',true) or die("can't open file.inc: $php_errormsg");
```

См. также

Документация по функции `fopen()`.

24.2. Создание временного файла

Задача

Требуется создать файл для временного хранения данных.

Решение

Чтобы создать файл, срок жизни которого ограничивается продолжительностью текущего запроса, воспользуйтесь функцией `tmpfile()`:

```
$temp_fh = tmpfile();
// Запись данных во временный файл
fputs($temp_fh,"The current time is ".strftime('%c'));
// Файл уничтожается при завершении сценария
exit(1);
```

Если файл должен существовать и далее, сгенерируйте имя файла функцией `tempnam()`, а затем воспользуйтесь `fopen()`:

```
$tempfilename = tempnam('/tmp', 'data-');
$temp_fh = fopen($tempfilename, 'w') or die($php_errormsg);
fputs($temp_fh, "The current time is ".strftime('%c'));
fclose($temp_fh)                    or die($php_errormsg);
```

Комментарий

Функция `tmpfile()` создает файл с уникальным именем и возвращает дескриптор. Файл будет уничтожен при вызове `fclose()` для этого дескриптора или при завершении сценария.

Также существует другой способ — сгенерировать имя файла функцией `tempnam()`. Функция получает два аргумента: в первом передается каталог, а во втором префикс имени файла. Если каталог не существует или недоступен для записи, `tempnam()` использует системный временный каталог (переменная окружения `TMPDIR` в Unix, переменная `TMP` в Windows). Следующий пример показывает, какие имена генерируются функцией `tempnam()`:

```
$tempfilename = tempnam('/tmp','data-');
print "Temporary data will be stored in $tempfilename";
```

Результат:

```
Temporary data will be stored in /tmp/data-GawVoL
```

Файл с именем, возвращаемым функцией `tempnam()`, фактически создается, но остается пустым, даже если он не будет явно открыт в вашем сценарии. Это гарантирует, что другая программа не создаст файл с тем же именем между моментом вызова `tempnam()` и временем вызова `fopen()` с этим именем.

См. также

Документация по функциям `tmpfile()` и `tempnam()`.

24.3. Дистанционное открытие файла

Задача

Требуется открыть файл, для обращения к которому используется протокол HTTP или FTP.

Решение

Передайте URL-адрес файла функции `fopen()`:

```
$fh = fopen('http://www.example.com/robots.txt','r') or die($php_errormsg);
```

Комментарий

Когда функции `fopen()` передается имя файла, начинающееся с `http://`, функция загружает заданную страницу запросом HTTP/1.0 GET (хотя заголовок `Host:` также передается для работы с виртуальными хостами). По файловому дескриптору можно обратиться только к телу ответа, но не к заголовкам. HTTP поддерживает только чтение, но не запись файлов.

Если передать `fopen()` имя файла, начинающееся с префикса `ftp://`, функция возвращает указатель на этот файл, полученный через протокол FTP пассивного режима. Через FTP можно открыть файл либо для чтения, либо для записи.

Чтобы открыть функцией `fopen()` URL-адрес с именем пользователя и паролем, включите данные аутентификации в URL-адрес:

```
$fh = fopen('ftp://username:password@ftp.example.com/pub/Index','r');
$fh = fopen('http://username:password@www.example.com/robots.txt','r');
```

Открытие временных файлов функцией `fopen()` реализуется при помощи так называемых *потоковых оберток* PHP. По умолчанию эта возможность включена, но ее можно отключить, присвоив директиве `allow_url_fopen` значение `off` в `php.ini` или конфигурационном файле веб-сервера. Если вам не удается открыть удаленный файл функцией `fopen()`, проверьте конфигурацию сервера.

См. также

Рецепты 14.1–14.7 — обращения по URL-адресам; документация по функции `fopen()`, удаленному доступу к файлам и потоковым оберткам.

24.4. Чтение из стандартного ввода

Задача

Требуется прочитать данные из стандартного ввода (входного потока) в контексте командной строки, например для получения данных с клавиатуры или данных, передаваемых программе PHP по каналу.

Решение

Воспользуйтесь функцией `fopen()` для открытия `php://stdin`:

```
$fh = fopen('php://stdin','r') or die($php_errormsg);
while($s = fgets($fh)) {
    print "You typed: $s";
}
```

Комментарий

В Рецепте 26.3 чтение данных с клавиатуры в контексте командной строки рассматривается более подробно. В веб-контексте чтение данных из стандартного ввода не приносит особой пользы, потому что информация не поступает через стандартный ввод. PHP разбирает тела запросов HTTP POST и запросов на отправку файлов и помещает их в специальные переменные. Тела запросов POST, не связанных с отправкой файлов, могут быть прочитаны из потока `php://input` (см. Рецепт 8.5).

См. также

Рецепт 26.3 — чтение с клавиатуры с контексте командной строки; Рецепт 8.5 — чтение тела запроса POST; документация по функции `fopen()`.

24.5. Чтение файла в строку

Задача

Требуется загрузить все содержимое файла в переменную. Например, вы хотите определить, существует ли в тексте совпадение для регулярного выражения.

Решение

Воспользуйтесь функцией `file_get_contents()`:

```
$people = file_get_contents('people.txt');
if (preg_match('/Names:.*(David|Susannah)/i',$people)) {
    print "people.txt matches.";
}
```

Комментарий

Если содержимое файла требуется сохранить в строке для обработки, то функция `file_get_contents()` прекрасно подойдет. Но если все содержимое файла нужно вывести, существуют и более простые (и более эффективные) способы, чем чтение его в строку с последующим выводом строки. PHP предоставляет две функции для решения этой задачи. Первая функция, `fpassthru($fh)`, выводит все оставшееся содержимое файлового дескриптора `$fh`, а затем закрывает его. Вторая функция, `readfile($filename)`, выводит все содержимое `$filename`.

Функция `readfile()` реализует обертку для изображений, которые должны отображаться только при выполнении некоторого условия. Следующая программа убеждается в том, что запрашиваемое изображение появилось менее недели назад:

```
$image_directory = '/usr/local/images';
if (preg_match('/^[a-zA-Z0-9]+\.(gif|jpe?g)$/',$image,$matches) &&
    is_readable($image_directory."/$image") &&
    (filemtime($image_directory."/$image") >= (time() - 86400 * 7))) {

  header('Content-Type: image/'.$matches[1]);
  header('Content-Length: '.filesize($image_directory."/$image"));

  readfile($image_directory."/$image");

} else {
  error_log("Can't serve image: $image");
}
```

Чтобы обертка работала эффективно, каталог, в котором хранятся изображения (`$image_directory`), должен находиться за пределами корневого каталога документов веб-сервера. В противном случае пользователь может просто обратиться к файлам изображений напрямую. Код проверяет файл изображения по трем условиям. Сначала он убеждается в том, что имя файла, переданное в `$image`, состоит из алфавитно-цифровых символов и завершается суффиксом .gif, .jpg или .jpeg. Также нужно проследить за тем, чтобы в имени файла не было таких символов, как `..` или `/`; это позволило бы злоумышленникам загружать файлы за пределами заданного каталога. Кроме того, функция `is_readable()` проверяет, что программа может прочитать файл. Наконец, функция `filemtime()` проверяет время изменения файла и убеждается в том, что оно не больше 86 400 × 7 секунд (в сутках 86 400, поэтому 86 400 × 7 — неделя[1]).

Если все условия выполнены, изображение можно отправлять. Сначала отправляются два заголовка, которые сообщают браузеру тип MIME и размер файла изображения. Затем функция `readfile()` передает все содержимое файла пользователю.

См. также

Документация по функциям `filesize()`, `fread()`, `fpassthru()` и `readfile()`.

24.6. Подсчет строк, абзацев или записей в файле

Задача

Требуется подсчитать количество строк, абзацев или записей в файле.

[1] При переходе между стандартным и летним временем в сутках может быть менее 86 400 секунд. За подробностями обращайтесь к Рецепту 3.10.

Решение

Для подсчета строк в файле используйте функцию `fgets()`:

```
$lines = 0;

if ($fh = fopen('orders.txt','r')) {
  while (! feof($fh)) {
    if (fgets($fh)) {
      $lines++;
    }
  }
}

print $lines;
```

Так как функция `fgets()` читает строки одну за одной, для подсчета строк достаточно подсчитать количество вызовов перед достижением конца файла.

Чтобы подсчитать абзацы, увеличивайте счетчик только при чтении пустой строки:

```
$paragraphs = 0;

if ($fh = fopen('great-american-novel.txt','r')) {
  while (! feof($fh)) {
    $s = fgets($fh);
    if (("\n" == $s) || ("\r\n" == $s)) {
      $paragraphs++;
    }
  }
}

print $paragraphs;
```

Чтобы подсчитать записи, увеличивайте счетчик только для строк, состоящих только из разделителя записей и пропусков. Вот как выглядит разделитель записей, хранящихся в переменной `$record_separator`:

```
$records = 0;
$record_separator = '--end--';

if ($fh = fopen('great-american-textfile-database.txt','r')) {
  while (! feof($fh)) {
    $s = rtrim(fgets($fh));
    if ($s == $record_separator) {
      $records++;
    }
  }
}

print $records;
```

Комментарий

При подсчете строк переменная `$lines` увеличивается только в том случае, если `fgets()` возвращает `true`. Так как функция `fgets()` перебирает содержимое фай-

ла, она возвращает каждую прочитанную строку. При достижении последней строки возвращается `false`, поэтому значение `$lines` не будет ошибочно увеличиваться. Так как для файла достигается признак конца файла (EOF), функция `feof()` возвращает `true`, и цикл `while` завершается.

Описанный способ подсчета абзацев правильно работает для простого текста, но он приводит к неожиданным результатам для длинной серии пустых строк или файла, не содержащего двух последовательных разрывов строк. Эти проблемы можно решить при помощи функций на базе `preg_split()`. Если файл достаточно мал и может быть прочитан в память, используйте функцию `split_paragraphs()`. Эта функция возвращает массив, содержащий все абзацы в файле.

Пример:

```
function split_paragraphs($file,$rs="\r?\n") {
    $text = file_get_contents($file);
    $matches = preg_split("/(.*?$rs)(?:$rs)+/s",$text,-1,
                          PREG_SPLIT_DELIM_CAPTURE|PREG_SPLIT_NO_EMPTY);
    return $matches;
}
```

Содержимое файла разбивается по двум и более последовательным разрывам строк и возвращается в массиве `$matches`. Регулярное выражение разделителя записей по умолчанию, `\r?\n`, совпадает с завершителями строк как Windows, так и Unix.

Если файл слишком велик для единовременной загрузки в память, используйте функцию `split_paragraphs_largefile()`, которая читает файл фрагментами по 16 Кбайт:

```
function split_paragraphs_largefile($file,$rs="\r?\n") {
    global $php_errormsg;

    $unmatched_text = '';
    $paragraphs = array();

    $fh = fopen($file,'r') or die($php_errormsg);

    while(! feof($fh)) {
        $s = fread($fh,16384) or die($php_errormsg);
        $text_to_split = $unmatched_text . $s;

        $matches = preg_split("/(.*?$rs)(?:$rs)+/s",$text_to_split,-1,
                              PREG_SPLIT_DELIM_CAPTURE|PREG_SPLIT_NO_EMPTY);

        // Если последний фрагмент не завершается двумя разделителями
        // записей, сохранить его для присоединения к нему следующего
        // прочитанного фрагмента.
        $last_match = $matches[count($matches)-1];
        if (! preg_match("/$rs$rs\$/",$last_match)) {
            $unmatched_text = $last_match;
            array_pop($matches);
        } else {
            $unmatched_text = '';
        }
```

```
            $paragraphs = array_merge($paragraphs,$matches);
        }

        // Если после чтения всех фрагментов остается последний фрагмент,
        // который не завершается разделителем записей,
        // он рассматривается как абзац.
        if ($unmatched_text) {
            $paragraphs[] = $unmatched_text;
        }
        return $paragraphs;
}
```

Эта функция использует то же регулярное выражение, что и `split_paragraphs()`, для разбиения файла на абзацы. Обнаружив конец абзаца во фрагменте, прочитанном из файла, она сохраняет остаток текста фрагмента в `$unmatched_text`, а затем присоединяет к нему следующий прочитанный фрагмент. Таким образом, оставшийся текст включается в начало следующего абзаца в файле.

Функция подсчета записей, приведенная в конце Решения, поручает определение длины каждой строки `fgets()`. Если вы сможете предоставить разумную верхнюю границу длины строки, `stream_get_line()` предоставляет более компактный способ подсчета записей. Эта функция читает строку до тех пор, пока не прочитает заданное количество байтов, или не встретит конкретный разделитель (в данном случае разделитель записей):

```
$records = 0;
$record_separator = '--end--';

if ($fh = fopen('great-american-textfile-database.txt','r')) {
    $done = false;
    while (! $done) {
        $s = stream_get_line($fh, 65536, $record_separator);
        if (feof($fh)) {
            $done = true;
        } else {
            $records++;
        }
  }
}

print $records;
```

Предполагается, что длина каждой записи не превышает 64 Кбайт (65 536 байт). Каждый вызов `stream_get_line()` возвращает одну запись, не включающую разделитель записей. Когда функция `stream_get_line()` переходит за последний разделитель записей, она достигает конца файла, и переменной `$done` присваивается `true` для прекращения подсчета записей.

См. также

Документация по функциям `fgets()`, `feof()`, `preg_split()` и `stream_get_line()`.

24.7. Обработка каждого слова в файле

Задача

Требуется обработать каждое слово в файле. Например, вы хотите построить таблицу количества вхождений каждого слова в документ для анализа сходства между документами.

Решение

Прочитайте каждую строку функцией `fgets()`, разделите строки на слова и обработайте каждое слово:

```
$fh = fopen('great-american-novel.txt','r') or die($php_errormsg);
while (! feof($fh)) {
    if ($s = fgets($fh)) {
        $words = preg_split('/\s+/',$s,-1,PREG_SPLIT_NO_EMPTY);
        // Обработка слов
    }
}
fclose($fh) or die($php_errormsg);
```

Комментарий

В следующем примере вычисляется средняя длина слова в файле:

```
$word_count = $word_length = 0;

if ($fh = fopen('great-american-novel.txt','r')) {
  while (! feof($fh)) {
    if ($s = fgets($fh)) {
      $words = preg_split('/\s+/',$s,-1,PREG_SPLIT_NO_EMPTY);
      foreach ($words as $word) {
        $word_count++;
        $word_length += strlen($word);
      }
    }
  }
}

print sprintf("The average word length over %d words is %.02f characters.",
              $word_count,
              $word_length/$word_count);
```

Обработка слов ведется по-разному в зависимости от того, как определяется понятие «слова». Код в этом рецепте использует метасимвол `\s`, определяемый Perl-совместимым ядром регулярных выражений; этот метасимвол определяет пропуск, то есть пробел, табуляцию, новую строку, возврат курсора или перевод страницы.

В Рецепте 1.5 строка разбивается на слова по пробелам; этот способ работает, потому что слова должны разделяться пробелами. Perl-совместимое ядро также поддерживает позиционный метасимвол границы слова (`\b`), который совпадает с позицией между символом слова (алфавитно-цифровым) и символом, не являющимся символом слова (все остальные символы). Использование `\b` вместо `\s` для определения границ слов наиболее заметно проявляется в словах, содержащих встроенные знаки препинания. Например, текст `6 o'clock` состоит из двух слов при разбивке по пропускам (`6` и `o'clock`) и из четырех слов при разбивке по границам слов (`6`, `o`, `'` и `clock`).

См. также

Рецепт 23.2 — применение регулярных выражений для поиска слов; Рецепт 1.5 — разбивка строк на слова; документация по функциям `fgets()`, `preg_split()` и Perl-совместимому расширению регулярных выражений.

24.8. Выбор случайной строки из файла

Задача

Требуется выбрать из файла случайную строку, например выбрать случайную цитату из файла, содержащего сборник цитат.

Решение

Организуйте равномерную случайную выборку из всех строк файла:

```
$line_number = 0;

$fh = fopen(__DIR__ . '/sayings.txt','r') or die($php_errormsg);
while (! feof($fh)) {
    if ($s = fgets($fh)) {
        $line_number++;
        if (mt_rand(0, $line_number - 1) == 0) {
            $line = $s;
        }
    }
}
fclose($fh) or die($php_errormsg);
```

Комментарий

После чтения каждой строки счетчик `$line_number` увеличивается, а приведенный пример генерирует случайное целое число в диапазоне от 0 до `$line_number - 1`. Если число равно 0, то текущая строка сохраняется в переменной результата. После того как все строки будут прочитаны, последняя случайно выбранная строка остается в `$line`.

Этот элегантный алгоритм гарантирует, что каждая строка в файле может быть выбрана с вероятностью $1/n$, не требуя при этом сохранения всех n строк в памяти.

См. также

Документация по функции `mt_rand()`.

24.9. Случайная перестановка всех строк в файле

Задача

Требуется случайным образом переставить все строки в файле. Допустим, у вас имеется файл со смешными цитатами, и вы хотите выбрать одну из них.

Решение

Прочитайте все строки файла в массив функцией `file()`, а затем сгенерируйте случайную перестановку элементов массива:

```
$lines = file(__DIR__ . '/quotes-of-the-day.txt');
if (shuffle($lines)) {
    // Перестановка выполнена
} else {
    die("Failed to shuffle");
}
```

Комментарий

Функция `shuffle()` переставляет элементы массива в случайном порядке, поэтому после перестановки можно выбрать элемент `$lines[0]` как случайную цитату для вывода.

См. также

Рецепт 4.20 — использование функции `shuffle()`; документация по функции `shuffle()`.

24.10. Обработка текстовых полей переменной длины

Задача

Требуется прочитать из файла информацию, состоящую из текстовых полей с разделителями. Допустим, имеется программа, которая генерирует данные

в виде строк с полями, разделенными табуляциями; эти данные нужно перенести в массив.

Решение

Используйте функцию `fgetcsv()` для чтения строк и последующей разбивки полей по разделителю:

```
$delim = '|';

$fh = fopen('books.txt','r') or die("can't open: $php_errormsg");
while (! feof($fh)) {
    $fields = fgetcsv($fh, 1000, $delim);
    // ... Обработка данных ...
    print_r($fields);
}
fclose($fh) or die("can't close: $php_errormsg");
```

Комментарий

Для обработки следующих данных из файла books.txt:

```
Elmer Gantry|Sinclair Lewis|1927
The Scarlatti Inheritance|Robert Ludlum|1971
The Parsifal Mosaic|Robert Ludlum|1982
Sophie's Choice|William Styron|1979
```

записи будут обрабатываться примерно так:

```
$fh = fopen('books.txt','r') or die("can't open: $php_errormsg");
while (! feof($fh)) {
    list($title,$author,$publication_year) = fgetcsv($fh, 1000, '|');
    // ... Обработка данных ...
}
fclose($fh) or die("can't close: $php_errormsg");
```

Функция `fgetcsv()` читает все строки из файла, как и `fgets()`, но разбирает их на отдельные поля. По умолчанию в качестве разделителя используется запятая, но вы можете выбрать другой символ (например, в данном примере используется вертикальная черта).

Алгоритм длины строки `fgetcsv()` должен быть не меньше длины самой большой записи, в противном случае запись будет усечена.

Если разделитель присутствует в какой-либо из записей, `fgetcsv()` сможет правильно разобрать данные, только если они заключены в соответствующие ограничители или экранированы. По умолчанию для этих целей используются двойные кавычки и символ обратной косой черты соответственно. Эту конфигурацию можно изменить:

```
$fh = fopen('books.txt','r') or die("can't open: $php_errormsg");
while (! feof($fh)) {
```

```
    list($title,$author,$publication_year) = fgetcsv($fh, 1000, '|', "'", '*');
    // ... Обработка данных ...
}
fclose($fh) or die("can't close: $php_errormsg");
```

В данном случае записи заключаются в одиночные кавычки, а символ одиночной кавычки экранируется звездочкой.

Если строки в файле распознаются некорректно, включите параметр конфигурации `auto_detect_line_endings` перед открытием файла:

```
ini_set('auto_detect_line_endings', true);
$fh = fopen('books.txt','r') or die("can't open: $php_errormsg");
// ...
```

См. также

Рецепт 1.15 — разбиение строк; Рецепты 1.12 и 1.14 — разбор данных, разделенных запятыми, и полей фиксированной ширины; документация по функции `fgetcsv()`.

24.11. Чтение конфигурационных файлов

Задача

Требуется использовать конфигурационные файлы для инициализации настроек программы.

Решение

Воспользуйтесь функцией `parse_ini_file()`:

```
$config = parse_ini_file('/etc/myapp.ini');
```

Комментарий

Функция `parse_ini_file()` читает конфигурационные файлы, структура которых соответствует структуре главного конфигурационного файла PHP php.ini. Но вместо того чтобы применять настройки из файла к конфигурации PHP, вызов `parse_ini_file()` возвращает значения из файла в виде массива.

Допустим, `parse_ini_file()` передается файл со следующим содержимым:

```
; physical features
eyes=brown
hair=brown
glasses=yes

; other features
```

```
name=Susannah
likes=monkeys,ice cream,reading
```

Возвращаемый массив выглядит так:

```
Array
(
    [eyes] => brown
    [hair] => brown
    [glasses] => 1
    [name] => Susannah
    [likes] => monkeys,ice cream,reading
)
```

Пустые строки и строки, начинающиеся с `;` в конфигурационном файле, игнорируются. Остальные строки с парами *имя=значение* помещаются в массив; имя определяет ключ элемента, а *значение* — конечно, его значение. Такие значения, как `on` и `yes`, возвращаются в виде 1, а такие, как `off` и `no`, — в виде пустой строки.

Чтобы разобрать данные секций конфигурационного файла, передайте 1 во втором аргументе `parse_ini_file()`. Секции в файле начинаются со слова, заключенного в квадратные скобки:

```
[physical]
eyes=brown
hair=brown
glasses=yes

[other]
name=Susannah
likes=monkeys,ice cream,reading
```

Если эти данные хранятся в /etc/myapp.ini, то вызов:

```
$conf = parse_ini_file('/etc/myapp.ini',1);
```

помещает массив в `$conf`:

```
Array
(
    [physical] => Array
        (
            [eyes] => brown
            [hair] => brown
            [glasses] => 1
        )

    [other] => Array
        (
            [name] => Susannah
            [likes] => monkeys,ice cream,reading
        )

)
```

Другой способ работы с конфигурационными данными заключается в том, чтобы сделать конфигурационный файл действительным файлом PHO, который можно будет загрузить директивой `require` вместо вызова `parse_ini_file()`. Если файл config.php содержит следующие данные:

```
<?php
// Внешность
$eyes = 'brown';
$hair = 'brown';
$glasses = 'yes';
// Другие атрибуты
$name = 'Susannah';
$likes = array('monkeys','ice cream','reading');
```

то переменные `$eyes`, `$hair`, `$glasses`, `$name` и `$likes` можно будет инициализировать простой директивой `require 'config.php';`.

Конфигурационный файл, загружаемый `require`, должен содержать действительный код PHP — включая начальный тег `<?php`. Значения переменных, указанных в config.php, присваиваются напрямую, а не помещаются в массив, как для `parse_ini_file()`. Для простых конфигурационных файлов этот прием может и не оправдывать лишних усилий на соблюдение синтаксиса, но он позволяет встроить в конфигурационный файл дополнительную логику, например команды вида:

```
$time_of_day = (date('a') == 'am') ? 'early' : 'late';
```

См. также

Документация по функции `parse_ini_file()`.

24.12. Изменение файла «на месте» без использования временного файла

Задача

Требуется изменить файл без использования временного файла для хранения изменений.

Решение

Прочитайте файл функцией `file_get_contents()`, внесите изменения и перезапишите файл функцией `file_put_contents()`:

```
$contents = file_get_contents('pickles.txt');
$contents = strtoupper($contents);
file_put_contents('pickles.txt', $contents);
```

Комментарий

Следующий пример преобразует текст, заключенный между звездочками или символами /, в текст в тегах HTML <b> или <i>:

```
$contents = file_get_contents('message.txt');
// Преобразование *word* в <b>word</b>
$contents = preg_replace('@\*(.*?)\*@i','<b>$1</b>',$contents);
// Преобразование /word/ в <i>word</i>
$contents = preg_replace('@/(.*?)/@i','<i>$1</i>',$contents);
file_put_contents('message.txt', $contents);
```

Поскольку добавление тегов HTML приводит к увеличению размера файла, весь файл необходимо прочитать в память, а затем обработать. Если изменение приводит только к уменьшению строк (или сохранению прежнего размера), файл можно обрабатывать построчно для экономии памяти.

Например, следующий фрагмент преобразует текст, размеченный тегами <b> и <i>, в текст, размеченный звездочками и символами косой черты:

```
$fh = fopen('message.txt','r+')         or die($php_errormsg);

// Определение количества байтов для чтения
$bytes_to_read = filesize('message.txt');

// Инициализация переменных для хранения позиций
$next_read = $last_write = 0;

// Продолжать, пока остаются байты
while ($next_read < $bytes_to_read) {

    /* Переместиться к позиции следующего чтения, прочитать строку,
       сохранить позицию следующего чтения */
    fseek($fh,$next_read);
    $s = fgets($fh)                     or die($php_errormsg);
    $next_read = ftell($fh);

    // Преобразовать <b>word</b> в *word*
    $s = preg_replace('@<b[^>]*>(.*?)</b>@i','*$1*',$s);

    // Преобразовать <i>word</i> в /word/
    $s = preg_replace('@<i[^>]*>(.*?)</i>@i','/$1/',$s);

    /* Переместиться к завершающей позиции последней записи, записать
     * преобразованную строку и сохранить позицию для следующей записи */
    fseek($fh,$last_write);
    if (-1 == fwrite($fh,$s))            { die($php_errormsg); }
    $last_write = ftell($fh);
}

// Усечение длины файла до фактически записанной
ftruncate($fh,$last_write)              or die($php_errormsg);

// Закрытие файла
fclose($fh)                             or die($php_errormsg);
```

См. также

Рецепты 13.4 и 13.5 — дополнительная информация о преобразовании между простым текстом и HTML; документация по функциям `fseek()`, `rewind()`, `ftruncate()`, `file_get_contents()` и `file_put_contents()`.

24.13. Сброс буферизованного вывода в файл

Задача

Требуется обеспечить запись всех буферизованных данных в файловый дескриптор.

Решение

Воспользуйтесь функцией `fflush()`:

```
fwrite($fh,'There are twelve pumpkins in my house.');
fflush($fh);
```

Вызов гарантирует, что строка `There are twelve pumpkins in my house.` будет записана в файл через дескриптор `$fh`.

Комментарий

Для повышения эффективности системные библиотеки ввода/вывода обычно не записывают данные непосредственно при выполнении операции вывода. Вместо этого записываемые данные накапливаются в буфере, а потом сбрасываются на диск одновременно. Вызов `fflush()` приводит к тому, что все данные, ожидающие в буфере записи, будут фактически записаны на диск.

Принудительная запись буферизованных данных может быть особенно полезна при генерировании журнала обращений или активности. Вызов `fflush()` после вывода каждого сообщения в журнал гарантирует, что администратор или программа, отслеживающие состояние журнала, безотлагательно увидят сообщение.

См. также

Документация по функции `fflush()`.

24.14. Запись в стандартный вывод

Задача

Требуется записать данные в стандартный выходной поток (стандартный вывод).

Решение

Воспользуйтесь командой `echo` или функцией `print()`:

```
print "Where did my pastrami sandwich go?";
echo  "It went into my stomach.";
```

Комментарий

`Print()` — функция, `echo` — языковая конструкция. Это означает, что `print()` возвращает значение, а `echo` этого не делает. В частности, это означает, что в большие выражения можно включать `print()`, но не `echo`:

```
// Можно
(12 == $status) ? print 'Status is good' : error_log('Problem with status!');
// Ошибка разбора
(12 == $status) ? echo 'Status is good' : error_log('Problem with status!');
```

При использовании файловых функций используйте php://stdout в качестве имени файла:

```
$fh = fopen('php://stdout','w') or die($php_errormsg);.
```

Запись в стандартный вывод через файловый дескриптор (вместо простого использования `print()` или `echo`) будет полезна, если вы хотите абстрагировать приемник для записи или намереваетесь отправлять данные в стандартный вывод одновременно с записью в файл. За подробностями обращайтесь к Рецепту 24.15.

Также можно записать данные в стандартный поток ошибок, для чего следует открыть `php://stderr`:

```
$fh = fopen('php://stderr','w');.
```

См. также

Рецепт 24.15 — одновременная запись по нескольким файловым дескрипторам; документация по функции `print()` и `echo`.

24.15. Одновременная запись по нескольким файловым дескрипторам

Задача

Требуется направить данные в несколько файловых дескрипторов одновременно; например, сообщения должны выводиться на экран и в файл.

Решение

Выполняйте вывод в цикле, в котором перебираются все файловые дескрипторы:

```
function multi_fwrite($fhs,$s,$length=NULL) {
  if (is_array($fhs)) {
    if (is_null($length)) {
      foreach($fhs as $fh) {
        fwrite($fh,$s);
      }
    } else {
      foreach($fhs as $fh) {
        fwrite($fh,$s,$length);
      }
    }
  }
}

$fhs = array();
$fhs['file'] = fopen('log.txt','w') or die($php_errormsg);
$fhs['screen'] = fopen('php://stdout','w') or die($php_errormsg);

multi_fwrite($fhs,'The space shuttle has landed.');
```

Комментарий

Если вы не хотите передавать аргумент длины `fwrite()` (или всегда передаете его), исключите эту проверку из `multi_fwrite()`. Следующая версия работает без аргумента `$length`:

```
function multi_fwrite($fhs,$s) {
  if (is_array($fhs)) {
    foreach($fhs as $fh) {
      fwrite($fh,$s);
    }
  }
}
```

См. также

Документация по функции `fwrite()`.

24.16. Экранирование метасимволов командного процессора

Задача

Требуется встроить в командную строку внешние данные, но при этом нужно экранировать специальные символы, чтобы не произошло ничего неожиданного; например, требуется передать пользовательский ввод в аргументе программы.

Решение

Используйте функцию `escapeshellarg()` для обработки аргументов и функцию `escapeshellcmd()` для обработки имен программ:

```
system('ls -al '.escapeshellarg($directory));
system(escapeshellcmd($ls_program).' -al');
```

Комментарий

Использовать неэкранированные символы в командной строке очень опасно. Никогда не передавайте немодифицированный пользовательский ввод функциям PHP, запускающим программы через командный процессор (shell). Всегда экранируйте соответствующие символы в командах и аргументах, это очень важно. Выполнение командных строк с данными, полученными от веб-форм, нетипично, и к этому решению не следует относиться легкомысленно. Тем не менее иногда без запуска внешних программ не обойтись, и экранирование команд и аргументов может вам пригодиться.

Функция `escapeshellarg()` заключает аргументы в одиночные кавычки (и экранирует все внутренние одиночные кавычки). В следующем примере функция `escapeshellarg()` используется для вывода информации о состоянии конкретного процесса:

```
system('/bin/ps '.escapeshellarg($process_id));
```

Использование `escapeshellarg()` гарантирует вывод правильной информации о процессе даже в том случае, если его идентификатор содержит непредвиденный символ (например, пробел). Также оно предотвращает выполнение непредвиденных команд. Если переменная `$process_id` содержит текст `1; rm -rf /`, то вызов `system("/bin/ps $process_id")` не только выводит информацию состояния процесса 1, но и выполняет команду `rm -rf /`.

С другой стороны, вызов `system('/bin/ps'.escapeshellarg($process_id))` выполняет команду `/bin/ps 1; rm -rf`, а это приводит к ошибке, потому что последовательность «1-точка с запятой-пробел-rm-пробел-дефис-rf» не является действительным идентификатором процесса.

Аналогичным образом `escapeshellcmd()` предотвращает выполнение непредвиденных командных строк. Команда `system("/usr/local/bin/formatter-$which_program");` выполняет разные программы в зависимости от значения `$which_program`.

Например, если `$which_program` содержит текст `pdf 12`, то сценарий запустит программу /usr/local/bin/formatter-pdf с аргументом `12`. Но если `$which_program` содержит текст `pdf 12; 56`, то сценарий выполнит /usr/local/bin/formatter-pdf с аргументом `12`, а также попытается запустить программу `56`, что приведет к ошибке.

Для успешной передачи аргументов formatter-pdf понадобится функция `escapeshellcmd()`: `system(escapeshellcmd("/usr/local/bin/formatter-$which_program"));`. Команда выполняет /usr/local/bin/formatter-pdf с передачей двух аргументов: `12;` и `56`.

См. также

Документация по функциям `system()`, `escapeshellarg()` и `escapeshellcmd()`.

24.17. Передача входных данных программе

Задача

Требуется передать входные данные внешней программе, запущенной из сценария PHP. Например, база данных требует запуска внешней программы для индексирования текста и вы хотите передать текст этой программе.

Решение

Откройте канал к программе вызовом `popen()`, запишите данные в канал вызовом `fputs()` или `fwrite()`, а затем закройте канал вызовом `pclose()`:

```
$ph = popen('/usr/bin/indexer --category=dinner','w') or die($php_errormsg);
if (-1 == fputs($ph,"red-cooked chicken\n"))    { die($php_errormsg); }
if (-1 == fputs($ph,"chicken and dumplings\n")) { die($php_errormsg); }
pclose($ph)                                    or die($php_errormsg);
```

Комментарий

В этом примере функция `popen()` используется для выполнения команды *nsupdate*, которая отправляет серверам имен запросы Dynamic DNS Update:

```
$ph = popen('/usr/bin/nsupdate -k keyfile')              or die($php_errormsg);
if (-1 == fputs($ph,"update delete test.example.com A\n")) { die($php_errormsg);}
```

```
if (-1 == fputs($ph,"update add test.example.com 5 A 192.168.1.1\n"))
                                                        { die($php_errormsg);}
pclose($ph)                                            or die($php_errormsg);
```

Программе *nsupdate* через `popen()` передаются две команды. Первая удаляет A-запись *test.example.com*, а вторая добавляет новую A-запись для *test.example.com* с адресом 192.168.1.1.

См. также

Документация по функциям `popen()` и `pclose()`; RFC 2136 — описание Dynamic DNS.

24.18. Получение стандартного вывода от программ

Задача

Требуется прочитать вывод программы, например системной утилиты *route(8)*, предоставляющей информацию о сети.

Решение

Чтобы прочитать весь вывод программы, используйте оператор ` (обратная кавычка):

```
$routing_table = `/sbin/route`;
```

Чтобы данные читались последовательно, откройте канал вызовом `popen()`:

```
$ph = popen('/sbin/route','r') or die($php_errormsg);
while (! feof($ph)) {
    $s = fgets($ph)            or die($php_errormsg);
}
pclose($ph)                    or die($php_errormsg);
```

Комментарий

Оператор ` выполняет программу и возвращает все ее выходные данные в виде одной строки. В системе Linux с 1,6 Гбайт оперативной памяти команда `$s = /usr/bin/free;` помещает в `$s` следующий многострочный текст:

```
             total       used       free     shared    buffers     cached
Mem:      16471704   15488260     983444          0     627820   12076120
-/+ buffers/cache:    2784320   13687384
Swap:            0          0          0
```

Если программа выдает большой объем выходных данных, для сокращения затрат памяти лучше читать данные из канала по одной строке. Если на основании вывода канала строятся отформатированные данные, которые передаются браузеру, их можно выводить по мере получения.

Следующий пример выводит информацию о последних входах в систему Unix, отформатированную в виде таблицы HTML. Для получения информации используется команда /usr/bin/last:

```
// Вывод заголовка таблицы
print<<<_HTML_
<table>
<tr>
 <td>user</td><td>login port</td><td>login from</td><td>login time</td>
 <td>time spent logged in</td>
</tr>
_HTML_;

// Открытие канала для /usr/bin/last
$ph = popen('/usr/bin/last','r') or die($php_errormsg);
while (! feof($ph)) {
    $line = fgets($ph) or die($php_errormsg);

    // Не обрабатывать пустые строки и завершающую строку
    if (trim($line) && (! preg_match('/^wtmp begins/',$line))) {
        $user = trim(substr($line,0,8));
        $port = trim(substr($line,9,12));
        $host = trim(substr($line,22,16));
        $date = trim(substr($line,38,25));
        $elapsed = trim(substr($line,63,10),' ()');

        if ('logged in' == $elapsed) {
            $elapsed = 'still logged in';
            $date = substr_replace($date,'',-5);
        }

        print "<tr><td>$user</td><td>$port</td><td>$host</td>";
        print "<td>$date</td><td>$elapsed</td></tr>\n";
    }
}
pclose($ph) or die($php_errormsg);

print '</table>';
```

См. также

Документация по функциям `popen()`, `pclose()` и оператору `` ` ``.

24.19. Получение стандартного потока ошибок от программы

Задача

Требуется прочитать информацию, выводимую программой в стандартный поток ошибок. Например, вы хотите получить список системных вызовов, выводимый *strace(1)*.

Решение

Перенаправьте стандартный поток ошибок в стандартный вывод, добавив `2>&1` в командную строку, передаваемую `popen()`. Прочитайте содержимое стандартного вывода, открыв канал в режиме `r`:

```
$ph = popen('strace ls 2>&1','r') or die($php_errormsg);
while (!feof($ph)) {
    $s = fgets($ph)               or die($php_errormsg);
}
pclose($ph)                       or die($php_errormsg);
```

Комментарий

В командных процессорах Unix (*sh*) и Windows (*cmd.exe*) стандартному потоку ошибок присваивается манипулятор `2`, а стандартному выводу — манипулятор `1`. Присоединение к команде `2>&1` приказывает командному процессору перенаправить данные, обычно передаваемые файловому манипулятору `2` (стандартная ошибка), в файловый манипулятор `1` (стандартный вывод). В этом случае `fgets()` будет читать и стандартный поток ошибок, и стандартный вывод.

Этот прием позволяет прочитать стандартный поток ошибок, но не дает возможности отличить его от стандартного вывода. Чтобы ограничиться чтением стандартного потока ошибок, следует заблокировать возвращение стандартного вывода по каналу. Для этого следует перенаправить его в псевдоустройство `/dev/null` в Unix или NUL в Windows:

```
// Unix: чтение только стандартного потока ошибок
$ph = popen('strace ls 2>&1 1>/dev/null','r') or die($php_errormsg);

// Windows: чтение только стандартного потока ошибок
$ph = popen('ipxroute.exe 2>&1 1>NUL','r') or die($php_errormsg);
```

См. также

Документация по функции `popen()`; man-страница *popen(3)* — информация о командном процессоре, используемом вашей системой с `popen()`; раздел «Redirection» *man*-страницы *sh(1)* — информация о перенаправлении для Unix; описание перенаправления в разделе «Command Reference» справочной системы — информация о перенаправлении для Windows.

24.20. Блокировка файла

Задача

Требуется получить монопольный доступ к файлу, чтобы предотвратить его возможное изменение во время чтения или обновления. Например, если информация гостевой книги хранится в файле, два пользователя должны иметь возможность добавлять элементы гостевой книги одновременно, не рискуя повредить данные другого пользователя.

Решение

Воспользуйтесь функцией `flock()` для организации необязательной блокировки:

```
$fh = fopen('guestbook.txt','a')         or die($php_errormsg);
flock($fh,LOCK_EX)                       or die($php_errormsg);
fwrite($fh,$_POST['guestbook_entry']) or die($php_errormsg);
fflush($fh)                              or die($php_errormsg);
flock($fh,LOCK_UN)                       or die($php_errormsg);
fclose($fh)                              or die($php_errormsg);
```

Комментарий

Как правило, если вам когда-либо потребуется установить блокировку для доступа к файлу, лучше поискать другое решение задачи. Часто можно (и нужно!) использовать базу данных (или SQLite, если более мощная СУБД недоступна).

Блокировка, предоставляемая функцией `flock()`, называется *необязательной* (advisory), потому что вызов `flock()` не запрещает другим процессам открыть заблокированный файл; он всего лишь позволяет им добровольно сотрудничать в процессе обращения к файлам. Чтобы механизм блокировки работал, все программы, которым потребуется обратиться к файлу, заблокированному вызовом `flock()`, должны устанавливать и снимать блокировку.

Функция `flock()` позволяет устанавливать блокировки двух видов: *монопольную* и *совместную*. Монопольная блокировка, которая определяется значением `LOCK_EX` во втором аргументе `flock()`, может в любой момент времени устанавливаться только одним процессом для конкретного файла. Совместная блоки-

ровка, определяемая значением LOCK_SH, может одновременно устанавливаться несколькими процессами. Прежде чем записывать данные в файл, следует установить монопольную блокировку. Перед чтением из файла устанавливается совместная блокировка.

Если ваш код хотя бы в одном месте устанавливает блокировку файла вызовом flock(), то и при всех остальных обращениях к файлу должна использоваться блокировка. Например, если в одной части программы LOCK_EX используется для получения монопольного доступа к файлу, то в любом месте, где этот файл будет читаться, следует использовать режим LOCK_SH для установления совместной блокировки файла. Если этого не сделать, процесс, осуществляющий чтение из файла, может работать с данными, пока они будут записываться другим процессом.

Чтобы снять блокировку с файла, вызовите flock() со вторым аргументом LOCK_UN. Перед снятием блокировки важно записать все буферизованные данные вызовом fflush(). Другие процессы не смогут установить блокировку, пока эти данные не будут записаны.

По умолчанию flock() переходит в режим ожидания до тех пор, пока не получит блокировку. Чтобы этого не происходило, добавьте во второй аргумент константу LOCK_NB:

```
$fh = fopen('guestbook.txt','a')             or die($php_errormsg);
$tries = 3;
while ($tries > 0) {
    $locked = flock($fh,LOCK_EX | LOCK_NB);
    if (! $locked) {
        sleep(5);
        $tries--;
    } else {
        // don't go through the loop again
        $tries = 0;
    }
}
if ($locked) {
    fwrite($fh,$_POST['guestbook_entry'])    or die($php_errormsg);
    fflush($fh)                              or die($php_errormsg);
    flock($fh,LOCK_UN)                       or die($php_errormsg);
    fclose($fh)                              or die($php_errormsg);
} else {
    print "Can't get lock.";
}
```

Когда запрос на блокировку выполняется без перехода в ожидание, flock() немедленно возвращает управление даже в том случае, если блокировку получить не удалось. Предыдущий пример трижды пытается установить блокировку для guestbook.txt с 5-секундными паузами между попытками.

Блокировка вызовом flock() не работает в некоторых ситуациях, например в некоторых реализациях NFS и старых версиях Windows. Для моделирования блокировки в таких случаях используется отдельный пустой каталог: его присутствие

означает, что файл данных заблокирован. Прежде чем открывать файл, создайте каталог-индикатор, а затем удалите его после завершения работы с файлом. В остальном код работы с файлом остается неизменным:

```
// Цикл вплоть до успешного создания каталога-индикатора блокировки
$locked = 0;
while (! $locked) {
    if (@mkdir('guestbook.txt.lock',0777)) {
        $locked = 1;
    } else {
        sleep(1);
    }
}
$fh = fopen('guestbook.txt','a')         or die($php_errormsg);

if (-1 == fwrite($fh,$_POST['guestbook_entry'])) {
    rmdir('guestbook.txt.lock');
    die($php_errormsg);
}
if (! fclose($fh)) {
    rmdir('guestbook.txt.lock');
    die($php_errormsg);
}
rmdir('guestbook.txt.lock')              or die($php_errormsg);
```

В качестве индикатора блокировки используется каталог, а не файл, потому что функция `mkdir()` не создает каталог, если он уже существует. Это позволит вам в одной операции проверить существование каталога и создать его, если он не существует. С другой стороны, все обработчики ошибок после его создания должны удалить каталог перед завершением. Если каталог останется на своем месте, будущие процессы не смогут установить блокировку, создав каталог.

Если в качестве индикатора почему-либо используется файл, а не каталог, код его создания выглядит примерно так:

```
$locked = 0;
while (! $locked) {
    if (! file_exists('guestbook.txt.lock')) {
        touch('guestbook.txt.lock');
        $locked = 1;
    } else {
        sleep(1);
    }
}
```

При повышенной нагрузке этот код может работать некорректно, потому что он проверяет существование блокировки вызовом `file_exists()`, а затем создает блокировку вызовом `touch()`. После того как один процесс вызовет `file_exists()`, другой может успеть вызвать `touch()` до того, как это сделает первый процесс. Оба процесса будут полагать, что они получили монопольный доступ к файлу, тогда как в действительности его не получил ни один. При использовании `mkdir()` задержка между проверкой существования и созданием отсутствует, поэтому процессу, создавшему каталог, гарантируется монопольный доступ.

См. также

Документация по функции `flock()`.

24.21. Чтение и запись нестандартных типов файлов

Задача

Требуется использовать стандартные файловые функции PHP для работы с данными, которые могут не находиться в физическом файле. Например, вы хотите использовать файловые функции для чтения и записи в общую память или для обработки данных из файла непосредственно при чтении (до того, как они достигнут PHP).

Решение

Воспользуйтесь модулем PEAR Stream_SHM, реализующим потоковую обертку для чтения и записи в общую память:

```
require_once 'Stream/SHM.php';
stream_register_wrapper('shm','Stream_SHM') or die("can't register shm");
$shm = fopen('shm://0xabcd','c');
fwrite($shm, "Current time is: " . time());
fclose($shm);
```

Комментарий

Потоковые обертки обеспечивают низкоуровневые подробности передачи данных между PHP и пользовательским местонахождением или форматом. Класс обертки реализует методы, необходимые PHP для выполнения операций с нестандартным потоком данных: открытия, закрытия, чтения, записи и т. д. Обертка регистрируется с префиксом, который используется при передаче имени файла `fopen()`, `include()` или любой другой файловой функции PHP и гарантирует вызов вашей обертки.

Потоковые обертки хорошо подходят для источников данных, отличных от файлов, но они также могут использоваться для предварительной обработки содержимого файлов на пути к PHP. Майк Набережный демонстрирует элегантный пример применительно к шаблонам. При отключении директивы `short_open_tags` вывод переменной объекта в шаблоне требует относительно громоздкой записи `<?php echo $this->property; ?>`. Решение Майка использует потоковую обертку, с которой `echo $this->` можно заменить символом `@`.

Код обертки выглядит так:

```
<?php
/**
 * Потоковая обертка для преобразования разметки шаблонов, состоящих
 * в основном из PHP, перед include().
 *
 * Решение в значительной мере основано на примере
 * http://www.php.net/manual/en/function.stream-wrapper-register.php
 *
 * @author Mike Naberezny (@link http://mikenaberezny.com)
 * @author Paul M. Jones  (@link http://paul-m-jones.com)
 */
class ViewStream {
    /**
     * Текущая позиция потока.
     *
     * @var int
     */
    private $pos = 0;
    /**
     * Данные для потока.
     *
     * @var string
     */
    private $data;
    /**
     * Статистика потока.
     *
     * @var array
     */
    private $stat;

    /**
     * Открывает файл сценария и преобразует разметку.
     */
    public function stream_open($path, $mode, $options, &$opened_path) {
        // Получение исходного кода сценария
        $path = str_replace('view://', '', $path);
        $this->data = file_get_contents($path);
        /**
         * Если чтение файла завершилось неудачей, локальная статистика
         * обновляется реальной статистикой файла, после чего
         * происходит возврат управления
         */
        if ($this->data===false) {
            $this->stat = stat($path);
            return false;
        }

        /**
         * Преобразование <?= ?> в развернутую форму <?php echo ?>
         *
         * Возможно, было бы также удобно преобразовать блоки кода
         * PHP <? ?>, но лучше вовремя остановиться. Вероятно, лучше
```

```
         * оставить <?php для более крупных блоков, но это ваш выбор.
         * Если вы пойдете по этому пути, явно проверяйте <?xml,
         *  потому что это создаст больше всего проблем.
         */
        if (! ini_get('short_open_tag')) {
            $find = '/\<\?\= (.*)? \?>/';
            $replace = "<?php echo \$1 ?>";
            $this->data = preg_replace($find, $replace, $this->data);
        }

        /**
         * Преобразование @$ в $this->
         *
         * Наверное, было бы разумнее ограничиться поиском @$
         * внутри <?php ?>, но, скорее всего, это не нужно, потому что
         * комбинация @$ встречается относительно редко, а использование
         * str_replace() существенно повышает быстродействие.
         */
        $this->data = str_replace('@$', '$this->', $this->data);
        /**
         * file_get_contents() не обновляет кэш статистики PHP,
         * поэтому повторное выполнение stat() снова приведет
         * к обращению к файловой системе. Так как файл был успешно
         * прочитан, достаточно использовать фиктивную статистику,
         * чтобы избежать жалоб include().
         */
        $this->stat = array('mode' => 0100777,
                            'size' => strlen($this->data));
        return true;
    }
    /**
     * Чтение из потока.
     */
    public function stream_read($count) {
        $ret = substr($this->data, $this->pos, $count);
        $this->pos += strlen($ret);
        return $ret;
    }

    /**
     * Получение текущей позиции в потоке.
     */
    public function stream_tell() {
        return $this->pos;
    }

    /**
     * Проверка достижения конца потока.
     */
    public function stream_eof() {
        return $this->pos >= strlen($this->data);
    }

    /**
     * Статистика потока.
```

```
     */
    public function stream_stat() {
        return $this->stat;
    }

    /**
     * Позиционирование в конкретную точку потока.
     */
    public function stream_seek($offset, $whence) {
        switch ($whence) {
            case SEEK_SET:
                if ($offset < strlen($this->data) && $offset >= 0) {
                $this->pos = $offset;
                    return true;
                } else {
                    return false;
                }
                break;
            case SEEK_CUR:
                if ($offset >= 0) {
                    $this->pos += $offset;
                    return true;
                } else {
                    return false;
                }
                break;
            case SEEK_END:
                if (strlen($this->data) + $offset >= 0) {
                    $this->pos = strlen($this->data) + $offset;
                    return true;
                } else {
                    return false;
                }
                break;

            default:
                return false;
        }
    }
}
```

Пример шаблона:

```
<html> <?= @$hello ?> </html>
```

Совместно они работают следующим образом:

```
/** Потоковая обертка */
require_once dirname(__FILE__) . DIRECTORY_SEPARATOR . 'ViewStream.php';
/**
 * Очень примитивный класс, предназначенный только для демонстрации.
 *
 * @author Mike Naberezny
 * @link   http://mikenaberezny.com/archives/40
 * @link   http://phpsavant.com
 */
```

```
class IdiotSavant {
    public function __construct() {
        if (!in_array('view', stream_get_wrappers())) {
            stream_wrapper_register('view', 'ViewStream');
        }
    }

    public function render($filename) {
        include 'view://' . dirname(__FILE__) . DIRECTORY_SEPARATOR .
        $filename . '.html';
    }
}

// Создание нового представления
$view = new IdiotSavant();

// Присваивание переменной "hello" в области видимости
$view->hello = 'Hello, World!';

// Построение представления по шаблону.
// Результат: "<html> Hello, World! </html>"
$view->render('ExampleTemplate');
```

Код потоковой обертки сохраняется в файле ViewStream.php, а шаблон — в файле ExampleTemplate.html.

См. также

Документация по функции `stream_register_wrapper()`; модуль PEAR Stream_SHM; публикация «Symfony Templates and Ruby's ERb» в блоге Майка Набережного (*http://mikenaberezny.com/2006/02/19/symphony-templates-ruby-erb/*).

24.22. Чтение и запись сжатых файлов

Задача

Требуется прочитать или записать сжатые файлы.

Решение

Воспользуйтесь потоковой оберткой `compress.zlib` или `compress.bzip2` в сочетании со стандартными файловыми функциями.

Чтение данных из файла, сжатого с применением алгоритма *gzip*:

```
$file = __DIR__ . '/lots-of-data.gz';
$fh = fopen("compress.zlib://$file",'r') or die("can't open: $php_errormsg");
if ($fh) {
    while ($line = fgets($fh)) {
```

```
        // $line - следующая строка несжатых данных
    }
    fclose($fh) or die("can't close: $php_errormsg");
}
```

Комментарий

Потоковая обертка `compress.zlib` предоставляет доступ к файлам, сжатым с применением алгоритма *gzip*. Потоковая обертка `compress.bzip2` используется для файлов, сжатых алгоритмом *bzip2*. Обе потоковые обертки поддерживают чтение, запись и присоединение данных к сжатым файлам. Чтобы включить поддержку потоков сжатия `zlib` и `bzip2`, постройте PHP с ключами `--with-zlib` и `--with-bz2` соответственно.

Кроме потоковых оберток, предоставляющих доступ к локальным сжатым файлам, существуют потоковые фильтры, выполняющие сжатие (или восстановление) произвольных потоков на ходу. Фильтры `zlib.deflate` и `zlib.inflate` сжимают и восстанавливают данные по алгоритму zlib «deflate». Фильтры `bzip2.compress` и `bzip2.uncompress` выполняют те же операции для алгоритма bzip2.

Каждый потоковый фильтр после создания должен быть применен к потоку. В следующем примере потоковые фильтры `bzip2` используются для чтения сжатых данных по URL-адресу:

```
$fp = fopen('http://www.example.com/something-compressed.bz2','r');
if ($fp) {
    stream_filter_append($fp, 'bzip2.uncompress');
    while (! feof($fp)) {
        $data = fread($fp);
        // Операции с  $data;
    }
    fclose($fp);
}
```

См. также

Документация по потоковым оберткам сжатия, фильтрам сжатия и функции `stream_filter_append()`; RFC 1950 и 1951 — подробное описание алгоритма *zlib*.

25 Каталоги

25.0. Введение

Кроме непосредственного содержимого файлов, файловая система хранит много дополнительной информации о файлах. В эту информацию включается размер файла, каталог и разрешения доступа. Если вы работаете с файлами, скорее всего, вам также потребуется работать с этими метаданными. PHP предоставляет разнообразные функции для чтения и выполнения операций с каталогами, элементами каталогов и атрибутами файлов. Как и другие части PHP, относящиеся к файлам, эти функции напоминают функции C, которые решают те же задачи (с некоторыми упрощениями).

Для создания файловой структуры используются так называемые *индексные узлы*. Каждый файл (как и любая другая часть файловой системы — каталог, устройство и ссылка) имеет собственный индексный узел, в котором хранится указатель на блоки данных файла, а также все метаданные этого файла. Блоки данных каталогов содержат имена файлов, находящихся в этом каталоге, и индексные узлы всех файлов.

В PHP предусмотрено несколько способов просмотра содержимого каталогов и получения списка находящихся в нем файлов. Класс `DirectoryIterator` предоставляет полноценный объектно-ориентированный интерфейс для перебора содержимого каталогов. В следующем примере `DirectoryIterator` используется для вывода имени каждого файла в каталоге:

```
foreach (new DirectoryIterator('/usr/local/images') as $file) {
    print $file->getPathname() . "\n";
}
```

Функции `opendir()`, `readdir()` и `closedir()` представляют процедурный подход к решению той же задачи. Функция `opendir()` возвращает дескриптор каталога,

функция `readdir()` перебирает файлы, а функция `closedir()` закрывает дескриптор каталога. Пример[1]:

```
$d = opendir('/usr/local/images') or die($php_errormsg);
while (false !== ($f = readdir($d))) {
    print $f . "\n";
}
closedir($d);
```

В примерах этой главы в основном используется класс `DirectoryIterator`.

Файловая система не ограничивается хранением файлов и каталогов. В Unix она также может содержать *символические ссылки* — специальные файлы, содержащие указатели на другие файлы. Удаление ссылки никак не отражается на файле, на который она указывает. Символические ссылки создаются функцией `symlink()`:

```
symlink('/usr/local/images','/www/docroot/images') or die($php_errormsg);
```

Эта команда создает в /www/docroot символическую ссылку с именем images, которая указывает на /usr/local/images.

Чтобы получить информацию о файле, каталоге или ссылке, следует проверить его индексный узел. Для чтения метаданных индексного узла используется функция `stat()` (пример приведен в Рецепте 25.2). В PHP также существуют функции для проверки отдельных атрибутов файла, использующие `stat()` в своей внутренней реализации. Список таких функций приведен в таблице 25.1.

Таблица 25.1. Функции для получения информации о файлах

Имя функции	Возвращаемая информация
file_exists()	Существует ли файл?
fileatime()	Время последнего обращения
filectime()	Время последнего изменения метаданных
filegroup()	Группа (числовое значение)
fileinode()	Номер индексного узла
filemtime()	Время последнего изменения содержимого
fileowner()	Владелец (числовое значение)
fileperms()	Разрешения (числовое значение)
filesize()	Размер

[1] В PHP также имеется функция dir(), которая возвращает объект с методами, повторяющими процедурный подход (открытие, чтение, закрытие). Так как класс DirectoryIterator обладает существенно большими возможностями, используйте его, если вам нужен ОО-интерфейс.

Имя функции	Возвращаемая информация
filetype()	Тип (fifo, char, dir, block, link, file, unknown)
is_dir()	Является каталогом?
is_executable()	Является исполняемым файлом?
is_file()	Является обычным файлом?
is_link()	Является ссылкой?
is_readable()	Доступен для чтения?
is_writable()	Доступен для записи?

В Unix набор операций, которые могут выполняться с файлом (его владельцем, участниками группы файла и всеми остальными пользователями), определяется разрешениями файла. Существуют три типа операций: чтение, запись и исполнение. Для программ под исполнением подразумевается возможность запуска программы; для каталогов — возможность просмотра каталога и получения списка содержащихся в нем файлов.

В разрешениях Unix также могут устанавливаться некоторые флаги: *бит setuid*, *бит setgid* и *бит закрепления* (sticky bit). Бит setuid означает, что при запуске программа должна выполняться с идентификатором пользователя своего владельца. Бит setgid означает, что программа должна выполняться с идентификатором группы, к которой она принадлежит. Для каталога бит setgid означает, что новые файлы в каталоге по умолчанию создаются в той же группе, что и сам каталог. Бит закрепления полезен для каталогов, используемых для совместной работы с файлами, потому что он не позволяет непривилегированным пользователям с разрешениями записи для каталога удалять файлы, находящиеся в этом каталоге, если они не являются владельцами файла или каталога.

При назначении разрешений функцией `chmod()` (см. Рецепт 25.3) значение должно выражаться в восьмеричной записи. Число состоит из четырех цифр. Первая цифра определяет специальные настройки для файлов (такие, как setuid или setgid). Вторая цифра определяет разрешения пользователя — операции, которые может выполнять владелец файла. Третья цифра определяет разрешения группы — операции, которые могут выполняться участниками группы файла. Четвертая цифра определяет разрешения для всех остальных пользователей.

Чтобы вычислить значение каждой цифры, сложите нужные значения из таблицы 25.2. Например, разрешения 0644 означают, что специальные режимы отсутствуют (0), владельцу файла разрешено чтение и запись (6 = 4 + 2, для чтения и записи соответственно), пользователи группы могут читать файл (первая цифра 4), а остальным пользователям также разрешено чтение файла (вторая цифра 4). Набор разрешений 4644 действует так же, не считая того, что для файла также установлен бит setuid.

Таблица 25.2. Разрешения доступа к файлам

Значение	Смысл в разрешениях	Смысл в специальных битах
4	Чтение	setuid
2	Запись	setgid
1	Исполнение	бит закрепления

Разрешения только что созданных файлов и каталогов зависят от значения, называемого *маской* (umask), — кода разрешений, исключаемого из исходных разрешений файла (0666) или каталога (0777). Например, если маска равна 0022, то разрешения по умолчанию для нового файла, созданного вызовом `touch()` или `fopen()`, равны 0644, а разрешения по умолчанию для нового каталога, созданного вызовом `mkdir()`, равны 0755. Для чтения и назначения маски используется функция `umask()`. Функция возвращает текущую маску, а если она вызывается с аргументом — заменяет текущую маску переданным значением. Следующий пример показывает, как назначить вновь создаваемым файлам разрешения, при которых к файлу сможет обращаться только его владелец (и суперпользователь):

```
$old_umask = umask(0077);
touch('secret-file.txt');
umask($old_umask);
```

Первый вызов `umask()` подавляет все разрешения для группы и остальных пользователей. После того как файл будет создан, второй вызов `umask()` восстанавливает предыдущее значение маски. Когда PHP работает в качестве серверного модуля, маска восстанавливает значение по умолчанию в конце каждого запроса. В Windows используется другая (более мощная) система управления разрешениями, поэтому функция PHP `umask()` (а также другие функции управления разрешениями) в Windows недоступна.

25.1. Чтение и запись временных меток

Задача

Требуется узнать время последнего обращения или изменения файла или же обновить временную метку обращения или изменения; например, вы хотите, чтобы на каждой странице сайта выводилась дата ее последней модификации.

Решение

Функции `fileatime()`, `filemtime()` и `filectime()` возвращают время последнего обращения, изменения или изменения метаданных файла:

```
$last_access = fileatime('larry.php');
$last_modification = filemtime('moe.php');
$last_change = filectime('curly.php');
```

Время последнего изменения файла обновляется функцией `touch()`. Без второго аргумента `touch()` заменяет время изменения текущей датой и временем. Чтобы заменить время изменения файла конкретным значением, передайте его функции `touch()` во втором аргументе в формате временной метки эпохи. Следующий пример изменяет время изменения двух файлов без изменения их содержимого:

```
touch('shemp.php');          // Время изменения заменяется текущим временем
touch('joe.php',$timestamp); // Время изменения заменяется на $timestamp
```

Комментарий

Функция `fileatime()` возвращает время последнего открытия файла для чтения или записи. Функция `filemtime()` возвращает время последнего изменения содержимого файла. Функция `filectime()` возвращает последнее время изменения содержимого файла или метаданных (например, владельца или разрешений). Все функции возвращают время в формате временной метки от начала эпохи. Следующий код выводит время последнего обновления страницы на сайте:

```
print "Last Modified: ".strftime('%c',filemtime($_SERVER['SCRIPT_FILENAME']));
```

См. также

Документация по функциям `fileatime()`, `filemtime()` и `filectime()`.

25.2. Чтение метаданных

Задача

Требуется прочитать метаданные файла, например информацию о разрешениях и владельце.

Решение

Воспользуйтесь функцией `stat()`, которая возвращает массив с информацией о файле:

```
$info = stat('harpo.php');
```

Комментарий

Функция `stat()` возвращает массив с числовыми и строковыми индексами, содержащий информацию о файле. Элементы массива перечислены в таблице 25.3.

Таблица 25.3. Информация, возвращаемая функцией stat()

Числовой индекс	Строковый индекс	Значение
0	dev	Устройство
1	ino	Индексный узел
2	mode	Разрешения
3	nlink	Количество ссылок
4	uid	Идентификатор владельца
5	gid	Идентификатор группы
6	rdev	Тип устройства для устройства индексного узла (–1 для Windows)
7	size	Размер (в байтах)
8	atime	Время последнего обращения (временная метка от начала эпохи)
9	mtime	Время последнего изменения содержимого (временная метка от начала эпохи)
10	ctime	Время последнего изменения содержимого или метаданных (временная метка от начала эпохи)
11	blksize	Размер блока ввода/вывода (–1 для Windows)
12	blocks	Количество блоков, выделенных для файла

Элемент `mode` возвращаемого массива содержит разрешения, выраженные в виде целого числа в десятичной записи. Это создает проблемы, потому что разрешения обычно выражаются либо в символьной форме (например, вывод `-rw-r--r--` программы *ls*), либо в восьмеричной целочисленной записи (например, 0644). Чтобы преобразовать разрешения в более понятный восьмеричный формат, используйте функцию `base_convert()`:

```
$file_info = stat('/tmp/session.txt');
$permissions = base_convert($file_info['mode'],10,8);
```

Здесь `$permissions` является восьмеричным числом из шести цифр. Например, если *ls* выводит следующие сведения о файле /tmp/session.txt:

```
-rw-rw-r--    1 sklar    sklar          12 Oct 23 17:55 /tmp/session.txt
```

то элемент `$file_info['mode']` содержит 33204, а `$permissions` содержит 100664. Три последние цифры (664) определяют разрешения для пользователя (чтение и запись), группы (чтение и запись) и остальных пользователей (чтение). Третья цифра, 0, означает, что для файла не установлены биты setuid или setgid. 10 в левых позициях означает, что файл является обычным файлом (а не сокетом, символической ссылкой или другим специальным файлом).

Так как функция `stat()` возвращает массив с числовыми и строковыми индексами, использование `foreach` для перебора по возвращаемому массиву создаст

две копии каждого значения. Вместо этого используйте цикл `for` от элемента 0 до элемента 12 возвращаемого массива.

Вызов `stat()` для символической ссылки возвращает информацию о файле, на который указывает символическая ссылка. Для получения информации о самой символической ссылке используется функция `lstat()`.

У `stat()` существует парная функция `fstat()`, в аргументе которой передается файловый дескриптор (возвращаемый вызовом `fopen()` или `popen()`).

Функция PHP `stat()` использует системную функцию `stat(2)`, вызов которой сопряжен с относительно высокими затратами. Чтобы свести затраты к минимуму, PHP кэширует результат вызова `stat(2)`. Следовательно, если вызвать `stat()` для файла, изменить его разрешения, а потом вызвать `stat()` для того же файла, то вы получите те же результаты. Чтобы заставить PHP обновить метаданные файла, вызовите функцию `clearstatcache()`, которая сбрасывает кэшированную информацию PHP. PHP также использует кэш для других функций, возвращающих метаданные файла: `file_exists()`, `fileatime()`, `filectime()`, `filegroup()`, `fileinode()`, `filemtime()`, `fileowner()`, `fileperms()`, `filesize()`, `filetype()`, `fstat()`, `is_dir()`, `is_executable()`, `is_file()`, `is_link()`, `is_readable()`, `is_writable()` и `lstat()`.

См. также

Документация по функциям `stat()`, `lstat()`, `fstat()` и `clearstatcache()`.

25.3. Изменение разрешений или владельца файла

Задача

Требуется изменить разрешения или владельца файла; например, вы хотите запретить другим пользователям просмотр файла с конфиденциальными данными.

Решение

Воспользуйтесь функцией `chmod()` для изменения разрешений файла:

```
chmod('/home/user/secrets.txt', 0400);
```

Функция `chown()` изменяет владельца файла, а функция `chgrp()` изменяет группу:

```
chown('/tmp/myfile.txt','sklar');           // Пользователь по имени
chgrp('/home/sklar/schedule.txt','soccer'); // Группа по имени
chown('/tmp/myfile.txt',5001);              // Пользователь по uid
chgrp('/home/sklar/schedule.txt',102);      // Группа по gid
```

Комментарий

Разрешения, передаваемые `chmod()`, должны задаваться в виде восьмеричного числа.

Суперпользователь может изменять разрешения, владельца и группу любого файла. Другие пользователи ограничены в правах: они могут изменять разрешения и группу только тех файлов, которые им принадлежат, а владельца они изменять вообще не могут. Кроме того, они могут заменить группу файла только той группой, к которой принадлежит пользователь.

Функции `chmod()`, `chgrp()` и `chown()` не работают в Windows.

См. также

Документация по функциям `chmod()`, `chown()` и `chgrp()`.

25.4. Получение компонентов имени файла

Задача

Требуется получить путь и имя файла; например, вы хотите создать файл в одном каталоге с существующим файлом.

Решение

Для получения имени файла используйте функцию `basename()`, а для получения пути — функцию `dirname()`:

```
$full_name = '/usr/local/php/php.ini';
$base = basename($full_name);  // $base содержит "php.ini"
$dir  = dirname($full_name);   // $dir содержит "/usr/local/php"
```

Используйте функцию `pathinfo()` для получения имени каталога, базового имени и расширения в ассоциативном массиве:

```
$info = pathinfo('/usr/local/php/php.ini');
// $info['dirname'] содержит "/usr/local/php"
// $info['basename'] содержит "php.ini"
// $info['extension'] содержит "ini"
```

Комментарий

Чтобы создать временный файл в одном каталоге с существующим файлом, вызовите функцию `dirname()` для получения каталога, а потом передайте этот каталог `tempnam()`. Пример:

```
$dir = dirname($existing_file);
$temp = tempnam($dir,'temp');
$temp_fh = fopen($temp,'w');
```

Функция `dirname()` особенно полезна в сочетании со специальной константой `__FILE__`, которая содержит полный путь к текущему файлу. Не путайте его с текущим выполняемым сценарием PHP — если сценарий /usr/local/alice.php включает /usr/local/bob.php, то в bob.php константа `__FILE__` содержит `/usr/local/bob.php`.

Следовательно, константа `__FILE__` особенно полезна, когда требуется включить сценарии, находящиеся в одном каталоге с конкретным файлом, но вы не знаете, что это за каталог, и он может отсутствовать в пути поиска. Пример:

```
$currentDir = dirname(__FILE__);
include $currentDir . '/functions.php';
include $currentDir . '/classes.php';
```

Если код находится в каталоге /usr/local, то он включает файлы /usr/local/functions.php и /usr/local/classes.php. Этот прием особенно полезен при распространении кода, предназначенного для других разработчиков. С ним вам не нужно будет требовать вносить изменения в конфигурацию или пути поиска, чтобы ваш код работал корректно. В версии PHP 5.3 вместо `dirname(__FILE__)` используется константа `__DIR__`.

Код с функциями `basename()`, `dirname()` и `pathinfo()` более универсален, чем разбиение полных имен по символу `/`, потому что функции используют разделитель текущей операционной системы. В Windows эти функции интерпретируют символы `/` и `\` как разделители файлов и каталогов; на других платформах используется только символ `/`.

Не существует встроенной функции PHP для объединения частей, полученных при помощи `basename()`, `dirname()` и `pathinfo()`, в полное имя файла. Для этого следует объединить части с использованием символов `.` и встроенной константы `DIRECTORY_SEPARATOR` — `/` в Unix, `\` в Windows.

См. также

Документация по функциям `basename()`, `dirname()`, `pathinfo()` и `__FILE__`.

25.5. Удаление файла

Задача

Требуется удалить файл.

Решение

Воспользуйтесь функцией `unlink()`:

```
$file = '/tmp/junk.txt';
unlink($file) or die ("can't delete $file: $php_errormsg");
```

Комментарий

Функция `unlink()` может удалять только те файлы, которые разрешено удалять пользователю процесса PHP. Если у вас возникнут трудности с `unlink()`, проверьте разрешения файла и способ запуска PHP.

См. также

Документация по функции `unlink()`.

25.6. Копирование и перемещение файла

Задача

Требуется скопировать или переместить файл.

Решение

Для копирования файлов используется функция `copy()`:

```
$old = '/tmp/yesterday.txt';
$new = '/tmp/today.txt';
copy($old,$new) or die("couldn't copy $old to $new: $php_errormsg");
```

Используйте функцию `rename()` для перемещения файлов:

```
$old = '/tmp/today.txt';
$new = '/tmp/tomorrow.txt';
rename($old,$new) or die("couldn't move $old to $new: $php_errormsg");
```

Комментарий

Если скопировать или переместить нужно сразу несколько файлов, вызовите `copy()` или `rename()` в цикле. Вызовы этих функций работают только с одним файлом.

См. также

Документация по функциям `copy()` и `rename()`.

25.7. Обработка всех файлов в каталоге

Задача

Требуется перебрать все файлы в каталоге. Например, вы хотите создать на форме поле `<select/>` со списком всех файлов в каталоге.

Решение

Воспользуйтесь классом `DirectoryIterator` для перебора всех файлов в каталоге:

```
echo "<select name='file'>\n";
foreach (new DirectoryIterator('/usr/local/images') as $file) {
    echo '<option>' . htmlentities($file) . "</option>\n";
}
echo '</select>';
```

Комментарий

Класс `DirectoryIterator` возвращает одно значение для каждого элемента в каталоге. Это значение представляет собой объект с некоторыми полезными характеристиками. Строковое представление объекта содержит имя файла (без начального пути) элемента каталога. Например, если /usr/local/images содержит файлы cucumber.gif и eggplant.png, код в Решении выводит следующий результат:

```
<select name='file'>
<option>.</option>
<option>..</option>
<option>cucumber.gif</option>
<option>eggplant.png</option>
</select>
```

Класс `DirectoryIterator` возвращает объект для каждого элемента каталога, включая `.` (текущий каталог) и `..` (родительский каталог). К счастью, этот объект содержит методы, которые помогают определить тип представляемого элемента. Метод `isDot()` возвращает `true` для элементов `.` или `..`. В следующем примере вызов `isDot()` предотвращает включение этих двух элементов в результат:

```
echo "<select name='file'>\n";
foreach (new DirectoryIterator('/usr/local/images') as $file) {
    if (! $file->isDot()) {
        echo '<option>' . htmlentities($file) . "</option>\n";
    }
}
echo '</select>';
```

В таблице 25.4 перечислены другие методы объектов, возвращаемых классом `DirectoryIterator`.

Таблица 25.4. Информационные методы объекта DirectoryIterator

Имя метода	Возвращаемое значение	Пример
isDir()	Элемент является каталогом?	false
isDot()	Элемент является . или ..?	false
isFile()	Элемент является обычным файлом?	true
isLink()	Элемент является ссылкой?	false

Таблица 25.4 (*продолжение*)

Имя метода	Возвращаемое значение	Пример
isReadable()	Элемент доступен для чтения?	true
isWritable()	Элемент доступен для записи?	true
isExecutable()	Элемент доступен для исполнения?	false
getATime()	Время последнего обращения к элементу	1144509622
getCTime()	Время создания элемента	1144509600
getMTime()	Время последнего изменения элемента	1144509620
getFilename()	Имя файла (без начального пути)	eggplant.png
getPathname()	Полное имя элемента	/usr/local/images/eggplant.png
getPath()	Путь элемента	/usr/local/images
getGroup()	Идентификатор группы элемента	500
getOwner()	Идентификатор владельца элемента	1000
getPerms()	Разрешения элемента (в восьмеричной записи)	16895
getSize()	Размер элемента	328742
getType()	Тип элемента (dir, file, link и т. д.)	file
getInode()	Номер индексного узла элемента	28720

Для получения данных, возвращаемых функциями из таблицы 25.4, используются те же системные функции, что и для данных из таблицы 25.1, в этом случае также действуют предупреждения, касающиеся различий между Unix и Windows.

См. также

Документация по классу `DirectoryIterator`.

25.8. Получение списка файлов по шаблону

Задача

Требуется найти все имена файлов, соответствующие заданному шаблону.

Решение

Используйте субкласс `FilterIterator` в сочетании с `DirectoryIterator`. Субкласс `FilterIterator` должен содержать метод `accept()`, который решает, подходит некоторое значение или нет.

Например, следующий код считает допустимыми только имена, заканчивающиеся стандартными расширениями графических файлов:

```
class ImageFilter extends FilterIterator {
    public function accept() {
        return preg_match('@\.(gif|jpe?g|png)$@i',$this->current());
    }
}
foreach (new ImageFilter(new DirectoryIterator('/usr/local/images')) as $img) {
    print "<img src='".htmlentities($img)."'/>\n";
}
```

Комментарий

Класс `FilterIterator` строится на основе `DirectoryIterator` и может отфильтровывать нежелательные элементы. Метод `accept()` решает, вернуть ли `true` или `false` для каждого элемента, для обращения к которому используется запись `$this->current()`. В Решении функция `accept()` использует регулярное выражение для принятия решения, но ваш код может использовать любую логику, которая вам понадобится.

Если правило может быть выражено простым шаблоном командного процессора (например, `*.*`), для получения подходящих имен файлов можно использовать функцию `glob()`. Например, следующий фрагмент находит все текстовые файлы в конкретном каталоге:

```
foreach (glob('/usr/local/docs/*.txt') as $file) {
   $contents = file_get_contents($file);
   print "$file contains $contents\n";
}
```

Функция `glob()` возвращает массив подходящих имен файлов. Если ни один файл не соответствует шаблону, функция `glob()` возвращает `false`.

См. также

Рецепт 25.9 — рекурсивный перебор файлов в каталоге; документация по классу `FilterIterator` и функции `glob()`; информация о шаблонах командного процессора (*http://www.gnu.org/software/bash/manual/bashref.html#SEC35*).

Задача

Требуется выполнить операцию со всеми файлами в каталоге и во всех его подкаталогах. Например, вы хотите узнать, какое дисковое пространство занимают все файлы в каталоге.

Решение

Воспользуйтесь классами `RecursiveDirectoryIterator` и `RecursiveIteratorIterator`. Класс `RecursiveDirectoryIterator` расширяет `DirectoryIterator`

методом getChildren(), который предоставляет доступ к элементам подкаталога. Класс RecursiveIteratorIterator преобразует иерархию, возвращаемую RecursiveDirectoryIterator, в один список. Следующий пример вычисляет общий размер файлов в каталоге:

```
$dir = new RecursiveDirectoryIterator('/usr/local');
$totalSize = 0;
foreach (new RecursiveIteratorIterator($dir) as $file) {
    $totalSize += $file->getSize();
}
print "The total size is $totalSize.\n";
```

Комментарий

Объекты, которые выдает RecursiveDirectoryIterator (а следовательно, и RecursiveIteratorIterator), не отличаются от объектов, получаемых от DirectoryIterator, поэтому для них доступны все методы из таблицы 25.4.

См. также

Документация по классам RecursiveDirectoryIterator и RecursiveIteratorIterator.

25.10. Создание новых каталогов

Задача

Требуется создать каталог.

Решение

Воспользуйтесь функцией mkdir():

```
mkdir('/tmp/apples',0777) or die($php_errormsg);
```

Комментарий

Второй аргумент mkdir() содержит режим разрешений нового каталога, который должен задаваться восьмеричным числом. Из этого значения исключается текущая маска, а результат определяет разрешения нового каталога. Таким образом, если текущая маска равна 0002, вызов mkdir(/tmp/apples,0777) назначает создаваемому каталогу разрешения 0775 (пользователю и группе разрешены чтение, запись и исполнение; остальным пользователям разрешается только чтение и исполнение).

По умолчанию `mkdir()` создает каталог только в том случае, если существует родительский каталог. Например, если каталог /usr/local/images не существует, создать каталог /usr/local/images/puppies не удастся. Чтобы создать каталог и его родителей, передайте `true` в третьем аргументе `mkdir()`. В этом случае вызов функции рекурсивно создает все недостающие родительские каталоги.

См. также

Документация по функции `mkdir()`.

25.11. Удаление каталога и его содержимого

Задача

Требуется удалить каталог со всем содержимым, включая подкаталоги и их содержимое.

Решение

Используйте классы `RecursiveDirectoryIterator` и `RecursiveIteratorIterator`, указав, что дочерние элементы (файлы и подкаталоги) должны перечисляться до их родителей:

```
function obliterate_directory($dir) {
    $iter = new RecursiveDirectoryIterator($dir);
    foreach (new RecursiveIteratorIterator($iter,
                RecursiveIteratorIterator::CHILD_FIRST) as $f) {
        if ($f->isDir()) {
            rmdir($f->getPathname());
        } else {
            unlink($f->getPathname());
        }
    }
    rmdir($dir);
}

obliterate_directory('/tmp/junk');
```

Комментарий

Конечно, удаление файлов может быть рискованной операцией. Так как встроенная в PHP функция удаления каталогов `rmdir()` работает только с пустыми каталогами, а `unlink()` не поддерживает шаблоны командного процессора, классу `RecursiveIteratorIterator` следует приказать перебирать потомков раньше родителей при помощи константы `CHILD_FIRST`.

См. также

Документация по функции `rmdir()` и классу `RecursiveIteratorIterator`.

25.12. Программа: вывод содержимого каталога веб-сервера

Программа web-ls.php (листинг 25.1) выводит список файлов в корневом каталоге документов веб-сервера, отформатированный по образцу вывода команды Unix *ls*. Имена файлов оформляются в виде ссылок, позволяющих загрузить каждый файл, а имена каталогов оформляются в виде ссылок для просмотра каждого каталога (рис. 25.1).

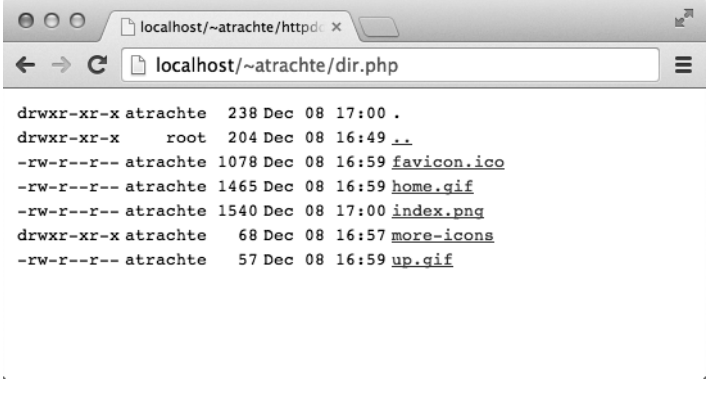

Рис. 25.1. Содержимое корневого каталога

Заметная часть листинга 25.1 строит удобное представление разрешений файла, а основная часть программы находится в цикле `foreach`. Класс `DirectoryIterator` возвращает элемент для каждого элемента в каталоге. Затем различные методы объекта элемента предоставляют информацию об этом файле, а функция `printf()` выводит отформатированную информацию об этом файле.

Функция `mode_string()` и используемые ею константы преобразуют восьмеричное представление режима файла (например, 35316) в удобочитаемую строку (например, `-rwsrw-r--`).

Листинг 25.1. web-ls.php

```
/* Битовые маски для определения разрешений и типа файлов.
 * Приведенные ниже значения соответствуют стандарту POSIX;
 * некоторые системы могут использовать собственные расширения.
 */
```

```
define('S_IFMT',0170000);   // маска для всех типов
define('S_IFSOCK',0140000); // тип: сокет
define('S_IFLNK',0120000);  // тип: символическая ссылка
define('S_IFREG',0100000);  // тип: обычный файл
define('S_IFBLK',0060000);  // тип: блочное устройство
define('S_IFDIR',0040000);  // тип: каталог
define('S_IFCHR',0020000);  // тип: символьное устройство
define('S_IFIFO',0010000);  // тип: fifo
define('S_ISUID',0004000);  // бит set-uid
define('S_ISGID',0002000);  // бит set-gid
define('S_ISVTX',0001000);  // бит закрепления
define('S_IRWXU',00700);    // маска разрешений владельца
define('S_IRUSR',00400);    // владелец: разрешение чтения
define('S_IWUSR',00200);    // владелец: разрешение записи
define('S_IXUSR',00100);    // владелец: разрешение исполнения
define('S_IRWXG',00070);    // маска разрешений группы
define('S_IRGRP',00040);    // группа: разрешение чтения
define('S_IWGRP',00020);    // группа: разрешение записи
define('S_IXGRP',00010);    // группа: разрешение исполнения
define('S_IRWXO',00007);    // маска разрешений других пользователей
define('S_IROTH',00004);    // другие пользователи: разрешение чтения
define('S_IWOTH',00002);    // другие пользователи: разрешение записи
define('S_IXOTH',00001);    // другие пользователи: разрешение исполнения

/* mode_string() - вспомогательная функция, которая получает восьмеричное
 * значение режима и возвращает строку из 10 символов, представляющую
 * тип файла и разрешения. Это PHP-версия функции mode_string()
 * из пакета GNU fileutils.
 */
$mode_type_map = array(S_IFBLK => 'b', S_IFCHR => 'c',
                       S_IFDIR => 'd', S_IFREG => '-',
                       S_IFIFO => 'p', S_IFLNK => 'l',
                       S_IFSOCK => 's');
function mode_string($mode) {
    global $mode_type_map;
    $s = '';
    $mode_type = $mode & S_IFMT;
    // Добавление символа, обозначающего тип
    $s .= isset($mode_type_map[$mode_type]) ?
          $mode_type_map[$mode_type] : '?';

  // Назначение разрешений пользователя
  $s .= $mode & S_IRUSR ? 'r' : '-';
  $s .= $mode & S_IWUSR ? 'w' : '-';
  $s .= $mode & S_IXUSR ? 'x' : '-';

  // Назначение разрешений группы
  $s .= $mode & S_IRGRP ? 'r' : '-';
  $s .= $mode & S_IWGRP ? 'w' : '-';
  $s .= $mode & S_IXGRP ? 'x' : '-';

  // Назначение разрешений других пользователей
  $s .= $mode & S_IROTH ? 'r' : '-';
  $s .= $mode & S_IWOTH ? 'w' : '-';
  $s .= $mode & S_IXOTH ? 'x' : '-';
```

```
  // Изменение букв исполнения для битов set-uid, set-gid и закрепления
  if ($mode & S_ISUID) {
      // 'S' для set-uid без исполнения владельцем
      $s[3] = ($s[3] == 'x') ? 's' : 'S';
  }

  if ($mode & S_ISGID) {
      // 'S' для set-gid без исполнения группой
      $s[6] = ($s[6] == 'x') ? 's' : 'S';
  }

  if ($mode & S_ISVTX) {
      // 'T' для бита закрепления без исполнения другими пользователями
      $s[9] = ($s[9] == 'x') ? 't' : 'T';
  }

  return $s;
}

// Если каталог не задан, начать к корневого каталога документов
$dir = isset($_GET['dir']) ? $_GET['dir'] : '';

// Поиск $dir в файловой системе
$real_dir = realpath($_SERVER['DOCUMENT_ROOT'].$dir);
// Передача корневого каталога документов через realpath
// снимает все проблемы с обычной и обратной косой чертой
$real_docroot = realpath($_SERVER['DOCUMENT_ROOT']);

// Убедиться в том, что $real_dir находится
// внутри корневого каталога документов
if (! (($real_dir == $real_docroot) ||
       ((strlen($real_dir) > strlen($real_docroot)) &&
        (strncasecmp($real_dir,$real_docroot.DIRECTORY_SEPARATOR,
        strlen($real_docroot.DIRECTORY_SEPARATOR)) == 0)))) {
    die("$dir is not inside the document root");
}

// Привести $dir к канонической форме, отделив корневой каталог
// документа от начальной части
$dir = substr($real_dir,strlen($real_docroot)+1);
// Открывается каталог?
if (! is_dir($real_dir)) {
    die("$real_dir is not a directory");
}
print '<pre><table>';

// Прочитать каждый элемент в каталоге
foreach (new DirectoryIterator($real_dir) as $file) {
    // Преобразовать uid в имя пользователя
    if (function_exists('posix_getpwuid')) {
        $user_info = posix_getpwuid($file->getOwner());
    } else {
        $user_info = $file->getOwner();
    }
```

```
// Преобразовать gid в имя группы
if (function_exists('posix_getgrid')) {
    $group_info = $file->getGroup();
} else {
    $group_info = $file->getGroup();
}

// Отформатировать дату для удобства чтения
$date = date('M d H:i',$file->getMTime());

// Преобразовать восьмеричное значение в понятную строку
$mode = mode_string($file->getPerms());

$mode_type = substr($mode,0,1);
if (($mode_type == 'c') || ($mode_type == 'b')) {
    /* Для блочного или символьного устройства вывести
     * основной и дополнительный тип устройства
     * вместо размера файла */
    $statInfo = lstat($file->getPathname());
    $major = ($statInfo['rdev'] >> 8) & 0xff;
    $minor = $statInfo['rdev'] & 0xff;
    $size = sprintf('%3u, %3u',$major,$minor);
} else {
    $size = $file->getSize();
}

// Отформатировать тег <a href=""> для имени файла.
// Для текущего каталога ссылка не создается.
if ('.' == $file->getFilename()) {
    $href = $file->getFilename();
} else {
    // Не включать ".." в ссылку родительского каталога
    if ('..' == $file->getFilename()) {
        $href = urlencode(dirname($dir));
    } else {
        $href = urlencode($dir) . '/' . urlencode($file);
    }
    /* Закодированы должны быть все символы, кроме "/" */
    $href = str_replace('%2F','/',$href);
    // Другие каталоги должны просматриваться web-ls
    if ($file->isDir()) {
        $href = sprintf('<a href="%s?dir=/%s">%s</a>',
                        $_SERVER['PHP_SELF'],$href,$file);
    } else {
        // Если это файл, создать ссылку для загрузки
        $href= sprintf('<a href="%s">%s</a>',$href,$file);
    }
    // Если это символьная ссылка, показать целевой файл
    if ('l' == $mode_type) {
        $href .= ' -&gt; ' . readlink($file->getPathname());
    }
}

// Вывод информации о файле
printf('<tr><td>%s</td><td align="right">%s</td>
```

```
            <td align="right">%s</td><td align="right">%s</td>
            <td align="right">%s</td><td>%s</td></tr>',
           $mode,                // Отформатированная строка режима
           $user_info['name'],   // Имя пользователя владельца
           $group_info['name'],  // Имя группы
           $size,                // Размер файла (или номера для устройств)
           $date,                // Дата и время последнего изменения
           $href);               // Ссылка для просмотра или загрузки
}
print '</table></pre>';
```

25.13. Программа: поиск по сайту

Программа site-search.php может использоваться для выполнения поиска по малым и средним сайтам на базе файлов:

```
class SiteSearch {
    public $bodyRegex = '';
    protected $seen = array();

    public function searchDir($dir) {
        // Массив для хранения подходящих страниц
        $pages = array();

        // Массив для каталогов, которые будут использоваться
        // при рекурсивном просмотре
        $dirs = array();

        // Каталог помечается как просмотренный,
        // чтобы не возвращаться к нему в будущем
        $this->seen[realpath($dir)] = true;

        try {
            foreach (new RecursiveIteratorIterator(
            new RecursiveDirectoryIterator($dir)) as $file) {
                if ($file->isFile() && $file->isReadable() &&
                (! isset($this->seen[$file->getPathname()]))) {
                    // Пометить путь как просмотренный,
                    // чтобы пропустить его, если
                    // мы вернемся к нему в будущем
                    $this->seen[$file->getPathname()] = true;

                    // Загрузить содержимое файла в $text
                    $text = file_get_contents($file->getPathname());

                    // Если условие поиска находится в теле страницы
                    if (preg_match($this->bodyRegex,$text)) {

                    // Построить относительный URI файла,
                    // исключив корневой каталог документов из полного пути
                    $uri = substr_replace($file->getPathname(),'',0,strlen
                    ($_SERVER['DOCUMENT_ROOT']));

                    // Если у страницы есть заголовок, получить его
```

```
                    if (preg_match('#<title>(.*?)</title>#Sis',$text,$match)) {
                        // Добавить заголовок и URI в $pages
                        array_push($pages,array($uri,$match[1]));
                    } else {
                        // В противном случае использовать URI как заголовок
                        array_push($pages,array($uri,$uri));
                    }
                }
                }
            }
        } catch (Exception $e) {
            // Ошибка при открытии каталога
        }
        return $pages;
    }
}

// Вспомогательная функция для алфавитной сортировки
// страниц по заголовкам
function by_title($a,$b) {
        return ($a[1] == $b[1]) ?
               strcmp($a[0],$b[0]) :
               ($a[1] > $b[1]);
}
// Объект SiteSearch для выполнения поиска
$search = new SiteSearch();

// Массив для страниц, подходящих по условию поиска
$matching_pages = array();
// Подкаталоги корневого каталога документов, в которых
// должен выполняться поиск
$search_dirs = array('sports','movies','food');
// Регулярное выражение для поиска. Модификатор "S" приказывает
// ядру PCRE проанализировать регулярное выражение
// для повышения эффективности.
$search->bodyRegex = '#<body>(.*' . preg_quote($_GET['term'],'#').
                     '.*)</body>#Sis';

// Добавить подходящие файлы в каждом каталоге в $matching_pages
foreach ($search_dirs as $dir) {
    $matching_pages = array_merge($matching_pages,
                      $search->searchDir($_SERVER['DOCUMENT_ROOT'].'/'.$dir));
}

if (count($matching_pages)) {
    // Отсортировать подходящие страницы по заголовкам
    usort($matching_pages,'by_title');
    print '<ul>';
    // Вывести каждый заголовок со ссылкой на страницу
    foreach ($matching_pages as $k => $v) {
        print sprintf('<li> <a href="%s">%s</a>',$v[0],$v[1]);
    }
    print '</ul>';
} else {
    print 'No pages found.';
}
```

Программа ищет заданное условие (из `$_GET['term']`) во всех файлах заданного набора каталогов, находящихся в корневом каталоге документов. Список этих каталогов хранится в `$search_dirs`. Кроме того, программа перебирает подкаталоги и переходит по символическим ссылкам, но при этом отслеживает посещенные файлы и каталоги, чтобы избежать зацикливания.

Если будут найдены какие-либо страницы, содержащие искомое условие, программа выводит список ссылок на эти страницы, упорядоченный в алфавитном порядке заголовков. Если страница не имеет заголовка (между тегами `<title>` и `</title>`), то используется относительный URI страницы.

Программа ищет искомое условие между тегами `<body>` и `</body>` в каждом файле. Если в тегах `<body>` содержится большое количество текста, который нужно исключить из поиска, заключите текст, в котором должен проводиться поиск, в специальные комментарии HTML, а затем измените `$body_regex`, чтобы поиск ограничивался этими тегами. Допустим, страница выглядит так:

```
<html>
<head>
        <title>Your Title</title>
</head>
<body>

// Разметка HTML для меню  и т. д.

<!-- search-start -->

<h1>Aliens Invade Earth</h1>

<h3>by H.G. Wells</h3>

<p>Aliens invaded earth today. Uh Oh.</p>

// ...

<!-- search-end -->

// Разметка HTML для завершителей  и т. д.

</body>
</html>
```

Чтобы поиск ограничивался названием книги, автором и кратким описанием в комментариях HTML, замените `$search->bodyRegex` следующим значением:

```
$search->bodyRegex = '#<!-- search-start -->(.*' . preg_quote($_GET['term'],'#').
             '.*)<!-- search-end -->#Sis';
```

Если вы не хотите, чтобы условие поиска совпадало с текстом, находящимся в тегах HTML или PHP, добавьте вызов `strip_tags()` в код, загружающий содержимое файла для поиска:

```
// Загрузить содержимое файла в $text
$text = strip_tags(file_get_contents($file->getPathname()));
```

26 PHP в режиме командной строки

26.0. Введение

Язык PHP создавался для веб-программирования и продолжает использоваться в основном для этой цели. Тем не менее PHP также неплохо справляется с ролью сценарного языка общего назначения. Применение PHP в сценариях, запускаемых из командной строки, особенно полезно в том случае, если они используют код из веб-приложений. Если на сайте поддерживается форум, возможно, вы захотите каждые несколько минут (или часов) запускать программу, которая проверяет новые публикации и сообщает о присутствии некоторых ключевых слов. Написание программы проверки на PHP позволит вам задействовать код основного форумного приложения. Этим вы не только сэкономите время, но и снизите затраты на сопровождение кода.

В сборку PHP включается версия интерфейса командной строки (CLI, Command-Line Interface). Двоичный файл CLI напоминает модули веб-сервера и двоичный файл CGI, но некоторые важные отличия делают его более удобным для взаимодействия с командным процессором. Некоторым конфигурационным директивам в CLI присваиваются жестко фиксированные значения; например, директиве `html_errors` присвоено значение `false`, а директиве `implicit_flush` — значение `true`. Директиве `max_execution_time` присваивается значение 0, что соответствует неограниченному времени выполнения программы. Наконец, директиве `register_argc_argv` присваивается значение `true`; это означает, что для обращения к информации аргументов можно использовать переменные `$argv` и `$argc` вместо `$_SERVER['argv']` и `$_SERVER['argc']`. Обработка аргументов рассматривается в Рецептах 26.1 и 26.2.

Чтобы запустить сценарий, передайте его имя файла в аргументе:

```
% php scan-discussions.php
```

В системе Unix также можно использовать синтаксис `#!` в начале сценария для того, чтобы интерпретатор PHP запускался автоматически. Если двоичный файл PHP находится в каталоге /usr/local/bin, сделайте следующую строку первой строкой вашего сценария:

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

После этого сценарий можно выполнить, просто вводя его имя в командной строке (при условии, что файл имеет разрешение на исполнение).

Если есть вероятность того, что некоторые классы и функции могут использоваться в веб-среде и в командной строке, абстрагируйте код, который должен по-разному вести себя в этих разных условиях (например, генерировать разметку HTML или простой текст или работать с переменными окружения, настроенными веб-сервером). В таких ситуациях бывает полезно проверить, возвращает ли `php_sapi_name()` значение `cli`. В этом случае логика сценария может выглядеть так:

```
if ('cli' == php_sapi_name()) {
  print "Database error: ".mysql_error()."\n";
} else {
  print "Database error.<br/>";
  error_log(mysql_error());
}
```

Этот код изменяет не только форматирование вывода в зависимости от контекста его выполнения (`\n` вместо `<br>`), но и способ выдачи информации. В командной строке человеку, запустившему программу, будет полезно увидеть сообщение об ошибке от MySQL, но в контексте веб-сервера разглашение таких (возможно, конфиденциальных) данных нежелательно. Вместо этого выводится общее сообщение об ошибке, а подробности сохраняются в журнале ошибок сервера для приватного просмотра.

В командной строке поддерживается полезный флаг `-d`, который позволяет задать значения `INI` для конкретного случая без модификации файла php.ini. Например, буферизация вывода включается следующим образом:

```
% php -d output_buffering=1 scan-discussions.php
```

Исполняемый файл CLI также получает аргумент `-r`, за которым следует код PHP без тегов `<?php and ?>`; этот код исполняется CLI. Например, команда вывода текущего времени выглядит так:

```
% php -r 'print strftime("%c");'
```

Чтобы получить полный список параметров командной строки CLI, передайте команду `-h`:

```
% php -h
```

Наконец, исполнительный файл CLI определяет дескрипторы для всех стандартных потоков ввода/вывода в виде констант `STDIN`, `STDOUT` и `STDERR`. Вы можете

использовать эти константы вместо создания собственных дескрипторов вызовом `fopen()`:

```
// Чтение из стандартного ввода
$input = fgets(STDIN,1024);

// Запись в стандартный вывод
fwrite(STDOUT,$jokebook);

// Запись в стандартный поток ошибок
fwrite(STDERR,$error_code);
```

26.1. Разбор аргументов

Задача

Требуется обработать аргументы, переданные в командной строке.

Решение

Количество аргументов хранится в `$argc`, а значения — в `$argv`. Первый элемент, `$argv[0]`, содержит имя выполняемого сценария:

```
if ($argc != 2) {
    die("Wrong number of arguments: I expect only 1.");
}
$size = filesize($argv[1]);
print "I am $argv[0] and report that the size of ";
print "$argv[1] is $size bytes.";
```

Комментарий

Чтобы задать значения параметров на основании флагов, переданных в командной строке, переберите содержимое `$argv` от 1 до `$argc`, как показано в листинге 26.1.

Листинг 26.1. Разбор аргументов командной строки

```
for ($i = 1; $i < $argc; $i++) {
    switch ($argv[$i]) {
    case '-v':
        // set a flag
        $verbose = true;
        break;
    case '-c':
        // Перейти к следующему аргументу
        $i++;
        // Если он задан, сохранить значение
```

```
        if (isset($argv[$i])) {
            $config_file = $argv[$i];
        } else {
            // Если имя файла не задано, прервать выполнение
            die("Must specify a filename after -c");
        }
        break;
    case '-q':
        $quiet = true;
        break;
    default:
        die('Unknown argument: '.$argv[$i]);
        break;
    }
}
```

В этом примере аргументы `-v` и `-q` представляют собой флаги, определяющие значения `$verbose` и `$quiet`, но за аргументом `-c` должна следовать строка, которая присваивается `$config_file`.

Переменные `$argc` и `$argv` удобны, но они не заполняются при отключенной конфигурационной директиве `register_argc_argv`. С другой стороны, в элементах `$_SERVER['argc']` и `$_SERVER['argv']` всегда хранятся количество и значения аргументов. Если вы хотите, чтобы ваш код был наиболее универсальным, используйте эти элементы для получения информации об аргументах.

См. также

Рецепт 26.2 — дополнительная информация о разборе аргументов с использованием `getopt`; документация по `$argc`, `$argv` и `$_SERVER`.

26.2. Разбор аргументов функцией getopt

Задача

Требуется разобрать командную строку с аргументами, которые могут задаваться в короткой или длинной форме, а также могут группироваться.

Решение

Воспользуйтесь встроенной функцией `getopt()`. В PHP 5.3.0 она поддерживает длинную запись, необязательные значения и другие удобные опции:

```
// Получает -a, -b и -c
$opts1 = getopt('abc');
// Получает --alice и --bob
$opts2 = getopt('', array('alice','bob'));
```

Комментарий

Чтобы получить значения аргументов с короткими именами, передайте `getopt()` массив аргументов командной строки и строку аргументов. В следующем примере разрешены аргументы `-a`, `-b` и `-c` — в виде аргументов или в группах:

```
$opts = getopt('abc');
```

Для приведенной строки аргументов `abc` следующие командные строки являются допустимыми:

```
% program.php -a -b -c
% program.php -abc
% program.php -ab -c
```

Метод `getopt()` возвращает массив. Для каждого аргумента, заданного в командной строке, в массиве создается один элемент; его ключом является имя аргумента, а значением — значение аргумента. Как ни странно, для аргументов-флагов, не получающих значения (такие как `a`, `b` и `c` в предыдущем примере), значение соответствующего элемента массива равно `false`. Допустим, программа запускается следующей командной строкой:

```
% program.php -a -b sneeze
```

В этом случае `$opts` содержит следующие элементы:

```
array(2) {
  ["a"]=>
  bool(false)
  ["b"]=>
  bool(false)
}
```

Чтобы показать, что для аргумента должно передаваться значение, поставьте в строке спецификации двоеточие после его имени. Два двоеточия означают, что значение не является обязательным. Таким образом, спецификация `ab:c::` сообщает, что для аргумента `a` значения быть не может, у аргумента `b` оно должно быть, а `c` может получать значение, если оно указано. С этой спецификацией для следующего запуска программы:

```
% program.php -a -b sneeze
```

`$opts` заполняется так:

```
array(2) {
  ["a"]=>
  bool(false)
  ["b"]=>
  string(6) "sneeze"
}
```

Теперь строка `sneeze` уже не игнорируется как неопознанная, а присваивается как значение аргумента `b`.

Чтобы в разбор включались аргументы с длинными именами, передайте во втором аргументе `getopt()` массив с описанием. Каждый аргумент представляется элементом массива; префикс `--` не указывается, а за именем следует `:` (признак обязательного аргумента) или `::` (необязательный аргумент). Первый аргумент `getopt()` (строка аргументов с короткими именами) может быть пустой строкой, но может и не быть — это зависит от того, хотите ли вы также разбирать аргументы с короткими именами. В следующем примере разрешается передача `debug` как аргумента без указания значения, `name` с обязательным значением и `size` с необязательным значением:

```
$opts = getopt('',array('debug','name:','size::'));
```

Допустимые варианты запуска этой программы:

```
% program.php --debug
% program.php --name=Susannah
% program.php --name Susannah
% program.php --debug --size
% program.php --size=56 --name=Susannah
% program.php --name --debug
```

Последний пример допустим (хотя его полезность неочевидна), потому что строка `--debug` рассматривается как значение аргумента `name`, а аргумент `debug` заданным не считается. Значения могут отделяться от аргументов в командной строке либо символом `=`, либо пробелом.

Обратите внимание: для аргументов с длинными именами `getopt()` не включает префикс `--` в массив разобранных аргументов. Для аргумента, заданного в командной строке в виде `--name`, в массиве аргументов создается ключ `name`.

См. также

Рецепт 26.1 — разбор аргументов программы без `getopt()`; документация по функции `getopt()`.

26.3. Чтение с клавиатуры

Задача

Требуется прочитать данные, введенные пользователем с клавиатуры.

Решение

Прочитайте данные из специального файлового дескриптора `STDIN`:

```
print "Type your message. Type '.' on a line by itself when you're done.\n";

$last_line = false;  $message = '';
```

```
while (! $last_line) {
    $next_line = fgets(STDIN,1024);
    if (".\n" == $next_line) {
      $last_line = true;
    } else {
      $message .= $next_line;
    }
}

print "\nYour message is:\n$message\n";
```

Если расширение Readline установлено, используйте функцию `readline()`:

```
$last_line = false; $message = '';
while (! $last_line) {
    $next_line = readline();
    if ('.' == $next_line) {
        $last_line = true;
    } else {
        $message .= $next_line."\n";
    }
}

print "\nYour message is:\n$message\n";
```

Если расширение ncurses установлено, используйте функцию `ncurses_getch()`:

```
$line = '';
ncurses_init();
ncurses_addstr("Type a message, ending with !\n");
/* Вывод нажимаемых клавиш при вводе */
ncurses_echo();
while (($c = ncurses_getch()) != ord("!")) {
    $line .= chr($c);
}
ncurses_end();
print "You typed: [$line]\n";
```

Комментарий

Со специальным файловым дескриптором `STDIN` можно использовать все стандартные функции чтения файлов (`fread()`, `fgets()` и т. д.). В Решении используется функция `fgets()`, которая возвращает ввод по строкам. Если вы используете `fread()`, ввод должен завершаться символом новой строки. Допустим, вы выполняете следующий фрагмент:

```
$msg = fread(STDIN,4);
print "[$msg]";
```

Если затем ввести текст `tomato` и новую строку, то будет выведен результат `[toma]`. Функция `fread()` берет из `STDIN` только четыре символа, как приказано, но ей все равно необходима новая строка как признак выхода из ожидания ввода с клавиатуры.

Расширение Readline предоставляет интерфейс к библиотеке GNU Readline. Функция `readline()` возвращает данные по одной строке без завершителей строк. Readline поддерживает средства редактирования строки в стиле *Emacs* и *vi*. Также расширение может использоваться для хранения истории вводившихся ранее команд:

```
$command_count = 1;
while (true) {
    $line = readline("[$command_count]--> ");
    readline_add_history($line);
    if (is_readable($line)) {
        print "$line is a readable file.\n";
    }
    $command_count++;
}
```

Этот пример выводит перед каждой строкой подсказку, содержащую последовательно увеличивающийся счетчик. Поскольку каждая строка добавлялась в историю Readline функцией `readline_add_history()`, клавиши ↑ и ↓ могут использоваться для перебора ранее вводившихся строк.

Расширение ncurses предоставляет интерфейс к библиотеке GNU ncurses, которая предоставляет высокую степень контроля за событиями клавиатуры, событиями мыши и экранным выводом в текстовом режиме. Основным способом чтения ввода с клавиатуры в ncurses является функция `ncurses_getch()`, которая возвращает ASCII-код нажатой клавиши. Принципиальное отличие ncurses от двух других способов, описанных выше, заключается в том, что нажатие очередной клавиши будет воспринято без ввода новой строки. Функция `ncurses_getch()` возвращает управление сразу же после одного нажатия клавиши. В примере, приведенном в Решении, код выполняется в цикле с многократным вызовом `ncurses_getch()` (и присоединением введенного символа к `$line`), пока не будет введен символ `!`.

См. также

Документация по функциям `fopen()`, `fgets()`, `fread()`, расширению Readline (*http://php.net/readline*) и библиотеке Readline (*http://cnswww.cns.cwru.edu/php/chet/readline/rltop.html*); расширению ncurses (*http://pecl.php.net/package/ncurses*) и библиотеке ncurses (*http://www.gnu.org/software/ncurses/*). Рецепт 26.5 — дополнительная информация о ncurses.

26.4. Выполнение кода PHP для каждой строки входного файла

Задача

Требуется прочитать все содержимое файла и выполнить код PHP для каждой строки. Например, вы хотите создать свою реализацию *grep*, использующую Perl-совместимый механизм регулярных выражений PHP.

Решение

Используйте флаг командной строки `-R` для обработки стандартного ввода:

```
% php -R 'if (preg_match("/$argv[1]/", $argn)) print "$argn\n";'
      php
      < /usr/share/dict/words

ephphatha
```

Чтобы блок кода выполнялся до или после обработки строк, используйте параметры `-B` и `-E` соответственно:

```
% php -B '$count = 0;'
      -R 'if (preg_match("/$argv[1]/", $argn)) $count++;'
      -E 'print "$count\n";'
      php
      < /usr/share/dict/words

1
```

Комментарий

Иногда бывает нужно быстро обработать файл с использованием PHP из командной строки — либо как автономного проекта, либо как цепочки команд. В таких случаях обычно пишется сценарий «на скорую руку» для преобразования данных.

В PHP для упрощения этой задачи существуют три флага командной строки и две специальные переменные: `-R`, `-B`, `-E`, `$argn` и `$argi`.

Флаг `-R` определяет код PHP, который должен выполняться для каждой строки в файле. В этом блоке кода можно обращаться к тексту строки в переменной `$argn`.

Сценарий PHP из следующего примера получает ввод HTML, удаляет теги и выводит результат:

```
php -R 'print strip_tags($argn) . "\n"; ' < index.html
```

Так как PHP автоматически удаляет новую строку в конце входных данных, этот код не только выводит результаты `strip_tags($argn)`, но и выводит новую строку. Он работает с файлом index.html, содержимое которого передается в стандартном входном потоке. Другой механизм для определения файла, который требуется обработать, не предусмотрен.

Следующий пример (чуть менее тривиальный) представляет собой упрощенную версию *grep*; он демонстрирует передачу входных аргументов через массив `$argv`:

```
% php -R 'if (preg_match("/$argv[1]/", $argn)) print "$argn\n";'
      php
      < /usr/share/dict/words

ephphatha
```

Первое значение, передаваемое `preg_match()`, — `/$argv[1]/` — содержит первый аргумент, переданный сценарию. В данном случае это строка `php`, поэтому код ищет в файле /usr/share/dict/words все слова, содержащие *php*. Кстати говоря, *ephphatha* — слово из арамейского языка, означающее «быть открытым».

Помимо отдельных строк также иногда требуется выполнить инициализацию или завершающие действия. Для этих целей используются флаги `-B` и `-E` соответственно.

Следующий код, построенный на базе примера *grep*, содержит общее количество совпадающих строк:

```
% php -B '$count = 0;'
      -R 'if (preg_match("/$argv[1]/", $argn)) $count++;'
      -E 'print "$count\n";'
      php
      < /usr/share/dict/words

1
```

В блоке `-B` переменная `$count` инициализируется значением 0. Затем она увеличивается в блоке `-R` для каждого найденного совпадения. Наконец, в блоке `-E` выводится итоговое значение счетчика.

Чтобы с общим значением счетчика узнать процент строк, содержащих совпадения, используйте переменную `$argi`:

```
% php -B '$count = 0;'
      -R 'if (preg_match("/$argv[1]/", $argn)) $count++;'
      -E 'print "$count/$argi\n";'
      php
      < /usr/share/dict/words

1/234937
```

Переменной `$argi` присваивается текущий номер строки файла, поэтому внутри блока `-E` она содержит общее количество строк.

См. также

Документация по использованию PHP в режиме командной строки (*http://php.net/features.commandline.options*).

26.5. Чтение паролей

Задача

Требуется прочитать текст в режиме командной строки так, чтобы он не отображался на экране в процессе ввода, например при вводе пароля.

Решение

Если расширение ncurses доступно, используйте функцию `ncurses_getch()` для чтения каждого символа; не забудьте проследить за тем, чтобы режим `noecho` был включен:

```
$password = '';
ncurses_init();
ncurses_addstr("Enter your password:\n");
/* Не отображать нажатия клавиш при вводе */
ncurses_noecho();
while (true) {
    // Получить символ с клавиатуры
    $c = chr(ncurses_getch());
    if ( "\r" == $c ||  "\n" == $c ) {
        // Если это новая строка, выйти из цикла - ввод пароля закончен
        break;
    } elseif ("\x08" == $c) {
        /* Если это Backspace, удалить предыдущий символ из $password */
        $password = substr_replace($password,'',-1,1);
    } elseif ("\x03" == $c) {
        // Если это Control-C, очистить $password и выйти из цикла
        $password = NULL;
        break;
    } else {
        // Иначе добавить символ в пароль
        $password .= $c;
    }
}
ncurses_end();
```

Если расширение ncurses недоступно, в системах Unix используйте `/bin/stty` для управления эхо-выводом введенных символов:

```
// Отключить эхо-вывод
`/bin/stty -echo`;
// Прочитать пароль
$password = trim(fgets(STDIN));
// Снова включить эхо-вывод
`/bin/stty echo`;
```

Как библиотека ncurses, так и утилита *stty* недоступны на платформе Windows.

Комментарий

Управление вводом на уровне символов (а также простота переключения эхо-вывода) делает библиотеку ncurses идеальным решением для чтения паролей. Функция `ncurses_getch()` читает символ без эхо-вывода на экране. Она возвращает ASCII-код прочитанного символа, который преобразуется в символ вызовом `chr()`. Далее нужное действие выбирается в зависимости от введенного символа: если это новая строка или возврат курсора, значит, ввод пароля закончен, и цикл прерывается. Если это символ Backspace, программа удаляет символ в конце

пароля. Если это комбинация клавиш Ctrl+C, пароль сбрасывается, а цикл прерывается. Если ни одно из перечисленных условий не выполняется, символ присоединяется к `$password`. При выходе из цикла переменная `$password` содержит введенный пароль.

Если вы работаете в системе Unix, но расширение ncurses недоступно, используйте утилиту /bin/stty для управления характеристиками терминала, чтобы символы не отображались на экране во время ввода.

Следующий код выводит приглашения Login: и Password: и сравнивает введенный пароль с соответствующим зашифрованным паролем из /etc/passwd. Для этого в системе не должны использоваться теневые пароли:

```
print "Login: ";
$username = rtrim(fgets(STDIN)) or die($php_errormsg);

preg_match('/^[a-zA-Z0-9]+$/',$username)
    or die("Invalid username: only letters and numbers allowed");

print 'Password: ';
`/bin/stty -echo`;
$password = rtrim(fgets(STDIN)) or die($php_errormsg);
`/bin/stty echo`;
print "\n";

// Найти соответствующую строку в /etc/passwd
$fh = fopen('/etc/passwd','r')   or die($php_errormsg);
$found_user = 0;
$pattern = '/^' . preg_quote($username) . ':/';
while (! ($found_user || feof($fh))) {
    $passwd_line = fgets($fh,256);
    if (preg_match($pattern,$passwd_line)) {
        $found_user = 1;
    }
}
fclose($fh);

$found_user or die ("Can't find user \"$username\"");

// Разобрать строку из /etc/passwd
$passwd_parts = split(':',$passwd_line);

/* Зашифровать введенный пароль и сравнить его с паролем из /etc/passwd */
$encrypted_password = crypt($password, $password_parts[1]);
if ($encrypted_password == $passwd_parts[1]) {
    print "login successful";
} else {
    print "login unsuccessful";
}
```

См. также

Документация по функциям `readline()` и `chr()`, расширению ncurses и библиотеке ncurses; *man*-страница *stty(1)* в Unix.

26.6. Консольный вывод в цвете

Задача

Требуется использовать разные цвета при выводе на консоль.

Решение

Воспользуйтесь классом PEAR `Console_Color2`:

```
$color = new Console_Color2();

$ok = $color->color('green');
$fail = $color->color('red');

$reset = $color->color('reset');

print $ok . "OK   " . $reset . "Something succeeded!\n";
print $fail . "FAIL " . $reset . "Something failed!\n";
```

Если вы уже используете ncurses, воспользуйтесь функциями консольного вывода в цвете:

```
ncurses_init();
ncurses_start_color();

ncurses_init_pair(1, NCURSES_COLOR_GREEN, NCURSES_COLOR_BLACK);
ncurses_init_pair(2, NCURSES_COLOR_RED, NCURSES_COLOR_BLACK);
ncurses_init_pair(3, NCURSES_COLOR_WHITE, NCURSES_COLOR_BLACK);

ncurses_color_set(1);
ncurses_addstr("OK   ");
ncurses_color_set(3);
ncurses_addstr("Something succeeded!\n");
ncurses_color_set(2);
ncurses_addstr("FAIL ");
ncurses_color_set(3);
ncurses_addstr("Something succeeded!\n");
```

Комментарий

Включая специальные служебные последовательности в вывод на консоль, вы можете приказать консоли выводить текст разными цветами. Вместо того чтобы запоминать «волшебные» числовые коды разных специальных символов, образующие служебные последовательности, воспользуйтесь классом PEAR `Console_Color2`. Метод `color()` этого класса возвращает строку со служебной последовательностью изменения цветов. Если включить одну из таких строк в выходной поток, это приведет к изменению цвета всего последующего текста (до тех пор, пока активный цвет не будет изменен другой служебной последовательностью).

Кроме специального «цвета» `reset` (который возвращает активный цвет по умолчанию) метод `color()` поддерживает следующие имена цветов: `black`, `red`, `green`, `brown`, `blue`, `purple`, `cyan`, `grey` и `yellow`.

Расширение ncurses также предоставляет собственные функции для работы с цветом. Несмотря на различия в синтаксисе логика поведения остается той же: вызовы функции, изменяющие «активный» цвет (`ncurses_color_set()`), чередуются с функциями вывода текста.

См. также

Класс `Console_Color2`; документация по функциям `ncurses_init_pair()` и `ncurses_color_set()`; дополнительная информация о программировании цветов в ncurses (*http://tldp.org/HOWTO/NCURSES-Programming-HOWTO/color.html*) и цветовых служебных последовательностях (*http://en.wikipedia.org/wiki/ANSI_escape_code*).

26.7. Программа: DOM Explorer

Программа dom-explorer.php, приведенная в листинге 26.2, предоставляет режим приглашения для интерактивного анализа документа HTML. Она читает документ HTML по переданному URL-адресу и разбирает его в объект `DOMDocument`. В отображаемом приглашении вводятся команды для получения информации о структуре узлов и содержимом документов.

Кроме того, программа dom-explorer.php использует функцию завершения слов, которая упрощает ввод узлов. Если ввести несколько символов и нажать клавишу Tab, открывается список узлов, соответствующих введенным символам:

```
% php dom-explorer.php http://www.php.net
/html > ls
head body
/html > ls head
title style[1] comment()[1] style[2] comment()[2] meta link[1] link[2] link[3] ⏎
script[1] link[4] script[2]
/html > cat head/title
PHP: Hypertext Preprocessor
/html > cd body
/html/body > ls
text()[1] div[1] text()[2] div[2] text()[3] div[3] text()[4] div[4] text()[5] ⏎
div[5] text()[6] div[6] text()[7] script comment()
/html/body > cd div[2]
/html/body/div[2] > ls
a text()[1] div text()[2]
/html/body/div[2] > cat a

/html/body/div[2] > cat div
downloads |
```

```
documentation | faq
| getting help | mailing lists | licenses | wiki
| reporting bugs | php.net sites | conferences | my
php.net
/html/body/div[2] > exit
```

Код программы dom-explorer.php приведен в листинге 26.2.

Листинг 26.2. dom-explorer.php

```
/* URL-адрес должен задаваться в командной строке */
isset($argv[1]) or die("No URL specified");

/* Загрузить HTML и запустить цикл команд */
$explorer = new DomExplorer($argv[1]);
$explorer->loop();

class DomExplorer {

    public function __construct($url) {
        $html = file_get_contents($url);
        if (false === $html) {
            throw new Exception("Can't retrieve $url");
        }
        /* Преобразовать HTML в действительную разметку XHTML */
        $clean = tidy_repair_string($html, array('output-xhtml' => true));

        /* Загрузить в DOMDocument, скрывая
         * все предупреждения libxml */
        $this->doc = new DOMDocument();
        libxml_use_internal_errors(true);
        if (false === $this->doc->loadHtml($clean)) {
            throw new Exception("Can't parse {$url} as HTML");
        }
        libxml_use_internal_errors(false);
        $this->currentNode = $this->doc->documentElement;
        $this->x = new DOMXPath($this->doc);
    }

    public function loop() {
        /* Функция завершения по нажатии клавиши Tab */
        readline_completion_function(array($this, 'completion'));
        while (true) {
            /* Текущий узел включается в подсказку */
            $line = readline($this->currentNode->getNodePath() . ' > ');
            readline_add_history($line);

            /* Первое введенное слово - команда, остальные - аргументы */
            $parts = explode(' ', $line);
            $cmd = array_shift($parts);

            /* Каждая команда представлена методом;
             * вызвать его, если он существует */
            $cmd_function_name = "cmd_$cmd";
            if (is_callable(array($this, $cmd_function_name))) {
```

```
            try {
                $this->$cmd_function_name($parts);
            } catch (Exception $e) {
                print $e->getMessage() . "\n";
            }
        }
        else {
            print "Unknown Command: $line\n";
        }
    }
}

/**
 * Команда: выход из программы
 */
protected function cmd_exit($args) {
    exit();
}

/**
 * Команда: вывод списка всех дочерних узлов
 * текущего или заданного узла
 */
protected function cmd_ls($args) {
    if (isset($args[0]) && strlen($args[0])) {
        $node = $this->resolvePath($args[0]);
    }
    else {
        $node = $this->currentNode;
    }
    print implode(' ' , $this->getChildNodePaths($node)) . "\n";
}

/**
 * Команда: переход к другому узлу
 */
protected function cmd_cd($args) {
    /* Если аргумент задан, использовать его */
    if (isset($args[0]) && strlen($args[0])) {
        $this->currentNode = $this->resolvePath($args[0]);
    }
    /* В противном случае вернуться к корневому узлу */
    else {
        $this->currentNode = $this->doc->documentElement;
    }
}
/**
 * Команда: вывод текстового содержимого узла
 */
protected function cmd_cat($args) {
    if (isset($args[0]) && strlen($args[0])) {
        $node = $this->resolvePath($args[0]);
        print $node->textContent . "\n";
    }
    else {
```

```
            throw new Exception("cat requires an argument");
        }
    }

    /**
     * Получить все пути узлов, находящихся под заданным,
     * с отсечением пути текущего узла от путей дочерних узлов
     */
    protected function getChildNodePaths($node) {
        $children = array();
        $curdir = $node->getNodePath();
        foreach ($node->childNodes as $node) {
            $path = $node->getNodePath();
            $sub = substr($path, strlen($curdir) + 1);
            $children[] = $sub;
        }
        return $children;
    }

    /**
     * При нажатии Tab вернуть массив путей дочерних узлов
     * как возможных вариантов завершения
     */
    protected function completion($str, $index) {
        return $this->getChildNodePaths($this->currentNode);
    }

    /**
     * Применить выражение xpath относительно текущего узла
     * и убедиться в том, что оно соответствует только одному целевому узлу
     */
    protected function resolvePath($arg) {
        $matches = $this->x->query($arg, $this->currentNode);
        if ($matches === false) {
            throw new Exception("Bad expresion: $arg");
        }
        if ($matches->length == 0) {
            throw new Exception("No match for $arg");
        }
        if ($matches->length > 1) {
            throw new Exception("{$matches->length} matches for arg");
        }
        return $matches->item(0);
    }

}
```

27 Пакеты

27.0. Введение

Пакеты и библиотеки избавляют вас от необходимости делать то, что уже было сделано ранее. Вы просто пользуетесь кодом, который был написан другими разработчиками для их проектов. В этой главе вы научитесь пользоваться тремя источниками пакетов: Composer, PEAR и PECL. Каждый источник предоставляет набор инструментов, которые упрощают интеграцию пакетов в ваш код и даже позволяют публиковать собственный код для общего блага.

Composer представляет собой менеджер зависимостей. Используйте Composer для установки кода (включая все внешние пакеты), необходимого для библиотек, которые вы собираетесь использовать. Composer позволяет легко включить в проекты правильные версии кода — как в начале проекта, так и при последующем обновлении версии пакета. В Packagist, основном репозитории для пакетов Composer, хранятся десятки тысяч пакетов PHP.

PEAR (PHP Extension and Application Repository) — коллекция классов с открытым кодом. Классы PEAR могут использоваться для разбора XML, реализации систем аутентификации, генерирования контрольных изображений, отправки почты с MIME-вложениями, а также для решения множества других типичных (и более экзотических) задач.

PECL (PHP Extension Community Library, произносится «пикл») — семейство расширений для PHP, написанных на языке C. Эти расширения похожи на те, которые включаются в основной выпуск PHP, но предназначены для более специализированных задач — как, например, драйвер базы данных MongoDB или расширение OAuth 1.0.

Все эти источники предназначены для разных целей. Composer принимает все публикации; любой желающий может зарегистрировать свой пакет, который может использоваться кем угодно. Часто в Composer удается найти несколько пакетов, предназначенных для похожих задач, с разным качеством: от перво-

классного до содержащего множество дефектов. С другой стороны, PEAR содержит только тщательно проверенный код с лучшей реализацией для каждой области, и все эти реализации работают в сочетании друг с другом. И если в Composer и PEAR хранится код, написанный на PHP, PECL специализируется на другой стороне мира PHP — расширениях на языке C.

PEAR и PECL также устанавливают одну версию библиотеки, предназначенную для использования во всех проектах. Теоретически такой подход упрощает управление, поскольку администратору нужно обеспечить актуальность только одного набора файлов. Однако на практике часто бывает нужно иметь доступ к двум разным версиям пакета одновременно. Две библиотеки часто используют один базовый пакет (например, поддержку запросов HTTP). Но если первой библиотеке требуется функциональность HTTP версии 2.0, вторая библиотека все еще работает с версией 1.8. А самое неприятное, что версия 2.0 не обладает обратной совместимостью с 1.8. Итак, обновление пакета HTTP для разрешения одной зависимости приведет к нарушению работоспособности другого пакета, причем без каких-либо очевидных возможностей предвидеть или обойти эту проблему при ее возникновении.

Composer действует несколько иначе: для каждого отдельного проекта устанавливаются свои копии пакетов. Это позволяет управлять конкретными версиями, используемыми в любой момент времени, обновлять несколько проектов по разным графикам, а также упаковать код вместе со всеми зависимостями в автономный модуль. Многие разработчики считают, что при таком подходе на управление тратится меньше времени, несмотря на дублирование кода.

Composer не входит в комплект поставки PHP, поэтому начать следует с загрузки и установки Composer на компьютере (эта тема рассматривается в Рецепте 27.1). Вся информация о Composer, включая документацию, последнюю версию продукта и систему отслеживания ошибок, доступна на сайте Composer.

Когда Composer заработает, можно переходить к поиску пакетов, которые вы собираетесь использовать. Рецепт 27.2 посвящен поиску и просмотру репозитория Packagist. После того как нужные пакеты и версии будут найдены, прикажите Composer добавить их.

Например, следующая команда устанавливает новейшую версию 2.x библиотеки PHP_CodeCoverage из PHPUnit:

```
% php composer.phar require phpunit/php-code-coverage:2.*
```

Команда загружает и устанавливает пакет в проект. Чтобы сценарии знали, где найти PHP_CodeCoverage (или любой другой пакет, установленный Composer), включите следующую директиву в начало сценария:

```
require 'vendor/autoload.php';
```

Вот и все! Один файл содержит полную информацию о том, где находится каждый пакет и все связанные с ним классы. Установка пакетов Composer подробно рассматривается в Рецепте 27.3.

За общей информацией о PEAR обращайтесь к документации; список новейших пакетов PEAR доступен по адресу *http://pear.php.net*. На сайте PEAR также имеются ссылки на архивы списков рассылки и RSS-каналы для получения информации о выходе новых пакетов.

Составной частью PEAR является программа *pyrus*, которая упрощает загрузку и установку дополнительных пакетов PEAR. В PHP 5.3+ она также называется программой установки PEAR[1]. О том, как ею пользоваться, рассказано в Рецепте 27.4.

Любой разработчик может использовать инфраструктуру управления классами PEAR в своих проектах. Если разработчик создает собственный пакет в формате PEAR, вы сможете использовать *pyrus* для загрузки и установки файлов с сайта каждого проекта. Этот процесс называется *созданием канала* PEAR. Программа установки PEAR поддерживает разнообразные функции, соответствующие конкретному типу канала, которые будут рассматриваться в рецептах этой главы.

В этой главе объясняется, как найти нужный пакет PEAR и как установить его на компьютере. Так как PEAR и каналы PEAR предоставляют доступ к множеству пакетов, разработчику нужны простые средства для их просмотра. В Рецепте 27.5 описаны различные способы поиска пакетов PEAR. Когда вы узнаете имя пакета и определите, на каком сервере каналов он находится, Рецепт 27.6 поможет вам получить подробную информацию о пакете.

После того как нужный пакет будет найден, запустите программу *pyrus* для загрузки пакета на ваш компьютер и установки его в нужном месте. Установка пакетов PEAR и расширений PECL рассматривается в Рецептах 27.7 и 27.10 соответственно. Рецепт 27.8 показывает, как проверить наличие обновлений пакетов на компьютере и как установить новейшие версии. Удаление пакетов рассматривается в Рецепте 27.9.

За инструкциями и примерами использования конкретных пакетов PEAR обращайтесь на сайт PEAR. Многие пакеты содержат документацию с примерами. Другие обычно хотя бы включают сгенерированную документацию API с примерами использования. Если найти информацию не удалось, попробуйте прочитать начальную секцию PHP-файлов пакета; часто в них приводятся примеры использования.

Найти документацию по расширениям PECL часто оказывается сложнее. Некоторые расширения PECL очень хорошо документированы в основном руководстве по PHP; хорошим примером служит расширение APC. Другие расширения PECL вообще не документируются, а понять, как ими пользоваться, можно только из тестовых сценариев PHP, включенных в пакеты на сайте PECL. В крайнем случае можно получить представление о том, что делает пакет, из его исходного кода.

---

[1] Вам также может попасться более старая программа установки PEAR, которая называется *pear*. Она предназначена для более ранних версий PHP и может пригодиться разработчикам, занимающимся сопровождением старого кода.

Репозитории Composer, PEAR и PECL содержат огромную подборку высококачественного кода, пригодного для многократного использования, поэтому эти проекты приносят огромную пользу сообществу PHP в целом.

27.1. Определение и установка зависимостей Composer

Задача

Требуется использовать Composer, чтобы вы могли устанавливать новые пакеты, обновлять существующие пакеты и получать информацию о них.

Решение

Установите Composer:

```
% curl -sS https://getcomposer.org/installer | php
```

Чтобы выполнить команду, передайте ее имя в первом аргументе командной строки:

```
% php composer.phar команда
```

Комментарий

Чтобы установить Composer, загрузите установочный файл и передайте его PHP. Composer использует PHP, чтобы убедиться в том, что система правильно настроена, обработать все настройки конфигурации и завершить процесс установки. Когда все будет сделано, в текущем каталоге появляется файл с именем composer.phar.

Composer представляет собой сценарий PHP, который можно запустить в командной строке PHP:

```
% php composer.phar команда
```

Также возможно запустить его напрямую, для чего следует поместить composer.phar в один каталог с PHP:

```
% mv composer.phar /usr/local/bin/composer
% composer команда
```

Если при этом возникнут проблемы, убедитесь в том, что у вас имеются разрешения для записи в этот каталог, а файл является исполняемым:

```
% sudo mv composer.phar /usr/local/bin/composer
% sudo chmod +x /usr/local/bin/composer
```

Возможно, копия PHP в вашей системе хранится в другом каталоге. Чтобы узнать его, выполните команду:

```
% which php
/usr/bin/php
```

Когда сценарий Composer заработает, передайте ему нужные команды. Например, установка пакета выполняется следующей командой:

```
% composer install
```

Полный список команд Composer выводится командой `list`.

В Composer поддерживаются команды как для использования, так и для разработки пакетов, поэтому может оказаться, что некоторые команды не представляют для вас интереса. Например, команда `archive` создает новый пакет; если вы работаете только с пакетами других разработчиков, она вам не понадобится. Некоторые часто используемые команды перечислены в таблице 27.1.

Таблица 27.1. Часто используемые команды Composer

Команда	Описание
search	Поиск пакетов
init	Создание файла composer.json
install	Установка зависимостей проекта
update	Обновление зависимостей
self-update	Обновление Composer

См. также

Сайт Composer и документация по установке (*https://getcomposer.org/doc/00-intro.md*).

27.2. Поиск пакетов Composer

Задача

Требуется найти пакеты, которые можно установить при помощи Composer.

Решение

Посетите сайт Packagist (*https://packagist.org/*) или воспользуйтесь Composer для вывода списка или поиска пакетов.

Комментарий

Packagist — основной репозиторий пакетов Composer. Любой разработчик может включить свой пакет в его каталог, а другие пользователи увидят этот пакет при просмотре или поиске.

Для найденного пакета Packagist выводит короткое описание, изображенное на рис. 27.1.

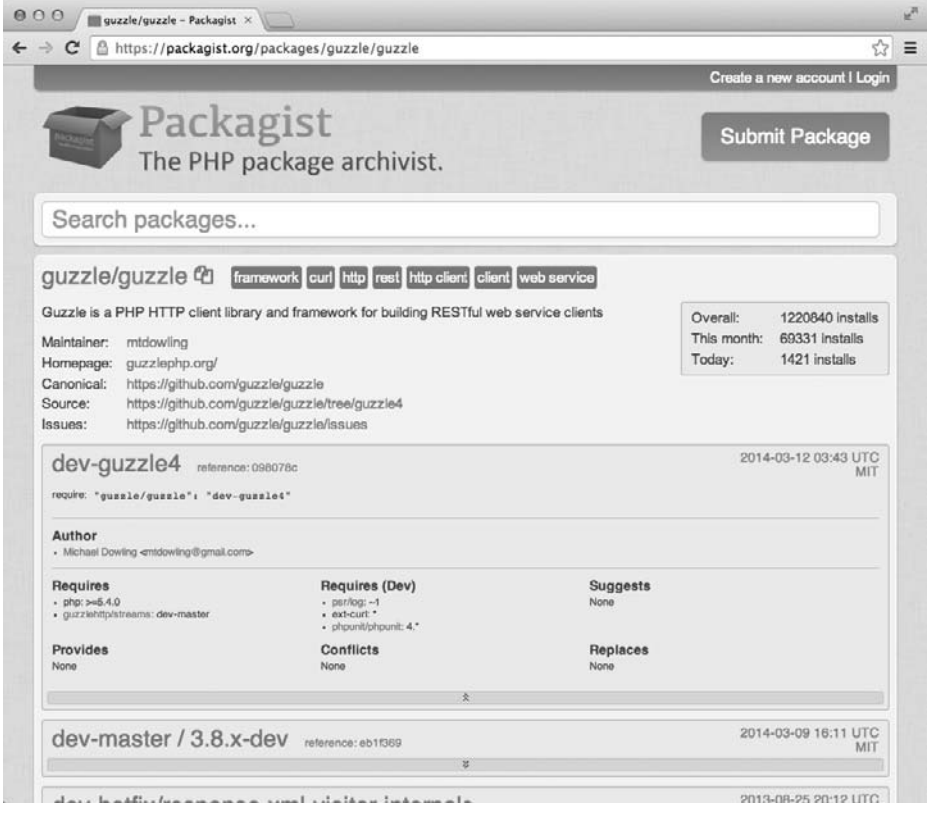

Рис. 27.1. Страница с информацией пакета Guzzle на сайте Packagist

На сайте Packagist не размещается ни исходный код, ни документация. За ними следует обращаться к репозиторию и домашней странице проекта.

Composer может вывести список всех пакетов, известных Packagist; для этой цели используется команда `show`:

```
% php composer.phar show
No composer.json found in the current directory, showing available packages ⏎
from packagist
```

```
platform:
  composer-plugin-api 1.0.0     The Composer Plugin API
  ext-bcmath          0         The bcmath PHP extension
...
  lib-xsl             1.1.26    The xsl PHP library
  php                 5.3.26    The PHP interpreter
  php-64bit           5.3.26    The PHP interpreter (64bit)
available:
  0k/php-oe-json
  0s1r1s/dev-shortcuts-bundle
  0x20h/monoconf
...
  zz/zz
  zzal/cakephp-hash
```

Впрочем, такой список состоит из десятков тысяч строк, поэтому лучше воспользоваться командой `search` для поиска нужного пакета:

```
% php composer.phar search http
No composer.json found in the current directory, showing packages from packagist
guzzle/http HTTP libraries used by Guzzle
illuminate/http
symfony/http-foundation Symfony HttpFoundation Component
symfony/http-kernel Symfony HttpKernel Component
net/http A basic HTTP client
minfraud/http MaxMind minFraud HTTP API
react/http Library for building an evented http server.
techdivision/http HTTP protocol implementation for usage in server context
vinelab/http An http library developed for the laravel framework. aliases
    itself as HttpClient
joomla/http Joomla HTTP Package
minond/http Http helpers
icanboogie/http Provides an API to handle HTTP requests.
aura/http The Aura HTTP package provides objects to build and send HTTP ⏎
requests and responses.
orno/http A wrapper for Symfony\HttpFoundation with some encapsulation and ⏎
convenience methods.
zendframework/zend-http provides an easy interface for performing Hyper-Text ⏎
Transfer Protocol (HTTP) requests
```

См. также

Сайт Packagist (*https://packagist.org/*).

27.3. Установка пакетов Composer

Задача

Требуется установить пакеты с использованием Composer.

Решение

Воспользуйтесь командой Composer `require`:

```
% php composer.phar require источник/пакет:версия
```

Например, для установки новейшей версии 2.x библиотеки PHP_CodeCoverage из PHPUnit вводится следующая команда:

```
% php composer.phar require phpunit/php-code-coverage:2.*
```

Также можно воспользоваться файлом composer.json:

```
{
    "require" : {
        "phpunit/php-code-coverage": "2.*"
    }
}
```

в сочетании с командой Composer `install`:

```
% php composer.phar install
```

Комментарий

Чтобы определить набор устанавливаемых пакетов, Composer читает инструкции из файла composer.json.

Этот файл представляет собой простой документ в формате JSON. Его важнейшим элементом является ключ `require`, который передает Composer список необходимых пакетов. Пример:

```
{
    "require": {
        "phpunit/php-code-coverage": "2.1.*"
    }
}
```

Этот файл сообщает, что вам необходим пакет `php-code-coverage`, опубликованный `phpunit`, и вас устраивает любая версия ветви 2.1.

Создайте такой файл в любом текстовом редакторе или воспользуйтесь командой `require` в Composer:

```
% php composer.phar require источник/пакет:версия
```

Команда создает (или редактирует) файл с добавлением необходимой разметки JSON.

Composer использует комбинацию источника и пакета как простейший механизм определения пространств имен пакетов. Многие люди создают пакеты с одинаковыми простыми именами, например `log`, `json` или `db`, а «пространства имен» позволяют указать, какой именно пакет вам нужен.

В некоторых случаях имена источника и пакета совпадают (например, `guzzle/guzzle`) — это нормально.

Файл composer.json может содержать несколько пакетов с более сложными инструкциями:

```
{
    "require": {
        "phpunit/php-code-coverage": "2.1.*",
        "guzzle/guzzle": ">=3.7.0",
        "monolog/monolog": "1.7.0"
    }
}
```

Этот файл запрашивает любую версию 2.1.x пакета `phpunit/php-code-coverage` (но менее 2.2.0), любую версию `guzzle/guzzle` от 3.7.0 и выше (включая 3.8 и 4.0) и версию 1.7.0 пакета `monolog/monolog` (и никакую другую).

Установка запускается командой `install`:

```
% php composer.phar install
```

Пакеты могут обладать собственными зависимостями, которые задаются производителем кода в отдельном файле composer.json. В нем указывается версия PHP, конкретные расширения PHP (например, cURL) и другие пакеты (например, базовый класс вывода в журнал).

Во время установки Composer автоматически проверяет систему на выполнение этих требований. Если требования не выполнены, Composer пытается исправить ситуацию (загрузить нужные пакеты) или выводит сообщение (если потребуется обновление PHP).

По действующим правилам Composer размещает все установленные пакеты в папке vendor текущего рабочего каталога. Такое размещение позволяет хранить всю информацию в одном месте, а папка легко добавляется в файл .gitignore.

После установки Composer записывает файл с именем composer.lock, содержащий точное описание набора установленных пакетов и их версий. Это позволяет «зафиксировать» конкретный набор пакетов, работающих в вашем приложении. Если в будущем при обновлении одного из пакетов возникнут какие-либо непредвиденные изменения, вы всегда можете вернуться к заведомо работоспособной конфигурации.

Чтобы использовать пакет, включите в начало файла директиву `require` для стандартного автозагрузчика Composer, после чего объявите пространство имен пакета и создайте объект. Ниже приведено начало сценария, в котором используется HTTP-клиент Guzzle:

```
require 'vendor/autoload.php';
use Guzzle\Http\Client;
// Создание клиента для работы с LinkedIn API
$client = new Client('https://api.linkedin.com/{version}', array(
    'version' => 'v1'
));
```

Каким образом Composer с коротким классом автозагрузки удается успешно найти все пакеты? Загадка решается при помощи стандартизации.

Для двух разных пакетов из репозитория Packagist совершенно не гарантируется какое-либо сходство в архитектуре или строении. Тем не менее многие пакеты обязуются соответствовать различным уровням совместимости. Для этого они реализуют набор стандартов, определяемых рабочей группой PHP Frameworks.

Уровень PSR (PHP Standards Recommendation) определяет, как два пакета будут работать совместно друг с другом. Наиболее критическим является уровень PSR-0, стандарт автозагрузки. Если в программе задействовано множество пакетов, каждый из которых использует собственный синтаксис имен файлов и каталогов, будет нелегко определить, как правильно включить их в ваш код.

Пакеты, реализующие PSR-0, обязуются выполнять общий набор правил организации пространств имен, а также выбора имен и структуры файлов PHP и каталогов, в которых они находятся. В результате любой PSR-0-совместимый пакет безопасно сосуществует с любыми другими пакетами PSR-0, а также может загружаться с использованием одной общей функции `autoload`.

Все пакеты, находящиеся под управлением Composer, следуют этому стандарту, что упрощает их использование в ваших программах.

См. также

Документация по использованию Composer; стандарт автозагрузки PSR-0 (*https://github.com/php-fig/fig-standards/blob/master/accepted/PSR-0.md*); пакет PHP_CodeCoverage; проект Guzzle (*http://docs.guzzlephp.org/en/latest/*); Monolog (*https://github.com/Seldaek/monolog*).

27.4. Использование программы установки PEAR

Задача

Требуется использовать программу установки PEAR — *pyrus*. Программа позволяет устанавливать новые пакеты, обновлять и получать информацию о существующих пакетах PEAR.

Решение

Установите Pyrus (*http://pear2.php.net/pyrus.phar*).

Чтобы выполнить команду, передайте ее имя в первом аргументе командной строки:

```
% php pyrus.phar команда
```

Комментарий

Pyrus — программа для управления пакетами PEAR. Она не включается в комплект поставки PHP, поэтому вам придется установить ее самостоятельно. К счастью, Pyrus распространяется в формате автономного архива PHP (phar), так что вам остается только загрузить файл, а затем запустить его в командной строке PHP:

```
% php pyrus.phar --version
Pyrus version 2.0.0a4 SHA-1: 72271D92C3AA1FA96DF9606CD538868544609A52
Using PEAR installation found at /Users/rasmus/lib
php pyrus.phar version 2.0.0a4.
```

Команда `list-packages` выводит список всех установленных пакетов PEAR:

```
% php pyrus.phar list-packages
Pyrus version 2.0.0a4 SHA-1: 72271D92C3AA1FA96DF9606CD538868544609A52
Using PEAR installation found at /Users/rasmus/lib
Listing installed packages [/Users/rasmus/lib]:
[channel pecl.php.net]:
(no packages installed in channel pecl.php.net)
[channel doc.php.net]:
(no packages installed in channel doc.php.net)
[channel __uri]:
(no packages installed in channel __uri)
[channel pear.php.net]:
Archive_Tar 1.3.7 stable
Console_Getopt 1.3.0 stable
PEAR 1.9.4 stable
Structures_Graph 1.0.4 stable
XML_Util 1.2.1 stable
```

Полный список всех команд PEAR выводится командой `help`.

В *pyrus* поддерживаются команды как для использования, так и для разработки пакетов PEAR, поэтому может оказаться, что некоторые команды не представляют для вас интереса. Например, команда `package` создает новый пакет PEAR; если вы только работаете с пакетами других разработчиков, она вам не понадобится. Некоторые часто используемые команды перечислены в таблице 27.2.

Чтобы узнать, где находятся пакеты PEAR, выполните команду PEAR `get php_dir`. Значение `include_path` можно проверить вызовом `ini_get('include_path')` из PHP или узнать его из файла php.ini. Если модификация php.ini невозможна из-за совместной рабочей среды, добавьте каталог в `include_path` в начале сценария перед включением каких-либо файлов PEAR. Работа с конфигурационными переменными в PHP более подробно рассматривается в Рецепте 20.5.

Настройка параметров *pyrus* осуществляется командой:

```
% php pyrus.phar set параметр значение
```

Таблица 27.2. Часто используемые команды программы установки PEAR

Команда	Сокращение	Описание
install	i	Загрузка и установка пакетов
upgrade	up	Обновление установленных пакетов
uninstall	un	Удаление установленных пакетов
list-packages	l	Вывод списка установленных пакетов
list-upgrades	lu	Вывод списка всех доступных обновлений для установленных пакетов
channel-discover	di	Инициализация альтернативного канала PEAR с сервера
list-channels	lc	Вывод списка всех каналов PEAR, настроенных на локальной машине
search	s	Поиск пакетов

Здесь *параметр* — имя изменяемого параметра, а *значение* — его новое значение. Для просмотра всех текущих настроек используется команда `get`:

```
% php pyrus.phar get
Pyrus version 2.0.0a4 SHA-1: 72271D92C3AA1FA96DF9606CD538868544609A52
Using PEAR installation found at /Users/rasmus/lib
System paths:
  php_dir => /Users/rasmus/lib/php
  ext_dir => /usr/lib/php/extensions/no-debug-non-zts-20121212
  cfg_dir => /Users/rasmus/lib/cfg
  doc_dir => /Users/rasmus/lib/docs
  bin_dir => /usr/bin
  data_dir => /Users/rasmus/lib/data
  www_dir => /Users/rasmus/lib/www
  test_dir => /Users/rasmus/lib/tests
  src_dir => /Users/rasmus/lib/src
  php_bin => /usr/bin/php
  php_ini => /private/etc/php.ini
  php_prefix =>
  php_suffix =>
Custom System paths:
User config (from /Users/rasmus/.pear/pearconfig.xml):
  default_channel => pear2.php.net
  auto_discover => 0
  http_proxy =>
  cache_dir => /Users/rasmus/lib/cache
  temp_dir => /Users/rasmus/lib/temp
  verbose => 1
  preferred_state => stable
  umask => 0022
  cache_ttl => 3600
  my_pear_path => /Users/rasmus/lib
  plugins_dir => /Users/rasmus/.pear
(variables specific to pear2.php.net):
  username =>
  password =>
```

```
    preferred_mirror => pear2.php.net
    download_dir => /Users/rasmus/lib/downloads
    openssl_cert =>
    handle =>
    paranoia => 2
Custom User config (from /Users/rasmus/.pear/pearconfig.xml):
(variables specific to pear2.php.net):
```

См. также

Документация по установке Pyrus (*http://pear.php.net/manual/en/installationpyrus.introduction.php*).

27.5. Поиск пакетов PEAR

Задача

Требуется получить список пакетов PEAR с возможностью запросить дополнительную информацию о каждом пакете и решить, хотите ли вы его устанавливать.

Решение

Просмотрите списки пакетов PEAR 2 (*http://pear2.php.net/categories*) и PEAR (*http://pear.php.net/packages.php?php=5*) или воспользуйтесь средствами поиска (*http://pear.php.net/search.php*). Используйте команду `remote-list` для получения списка пакетов PEAR. Просмотрите списки серверов каналов PEAR (*http://pear.php.net/channels/*).

Комментарий

Существует несколько способов просмотра доступных пакетов PEAR (и PEAR-совместимых). Во-первых, списки официальных пакетов PEAR в стиле иерархической структуры каталогов можно просмотреть по адресам *http://pear2.php.net/categories/* и *http://pear.php.net/packages.php?php=5*. Начните с верхнего уровня и откройте интересующие вас категории PEAR.

Также информацию о пакетах можно найти по адресу *http://pear.php.net/search.php*. Поддерживается поиск по имени пакета, автору, категории и дате публикации.

Чтобы запросить у Pyrus список пакетов в канале PEAR, используйте команду `remote-list`:

```
% php pyrus.phar remote-list pear
Pyrus version 2.0.0a4 SHA-1: 72271D92C3AA1FA96DF9606CD538868544609A52
Using PEAR installation found at /Users/rasmus/lib
Remote packages for channel pear:
Audio:
```

```
Key: * = installed, ! = upgrades available
Authentication:
   Auth                      1.6.4     Creating an authentication system.
   Auth_HTTP                 2.1.8     HTTP authentication
   Auth_PrefManager          1.2.2     Preferences management class
   Auth_PrefManager2         2.0.0dev\ Preferences management class
                             1
...
   XML_XPath2                n/a       The PEAR::XML_XPath2 package provided
                                       an XPath/DOM XML manipulation,
                                       maneuvering and query interface.
   XML_XRD                   0.3.0     PHP library to parse and generate
                                       "Extensible Resource Descriptor"
                                       (XRD + JRD) files
   XML_XSLT_Wrapper          0.2.2     Provides a single interface to the
                                       different XSLT interface or commands
   XML_XUL                   0.9.1     Class to build Mozilla XUL applications.
Key: * = installed, ! = upgrades available
```

Команда `remote-list` также может использоваться для получения информации о доступных пакетах с PEAR-совместимых серверов каналов. Для этого необходимо сначала передать Pyrus информацию об альтернативном сервере. Пример:

```
% php pyrus.phar channel-discover pear.drush.org
Pyrus version 2.0.0a4 SHA-1: 72271D92C3AA1FA96DF9606CD538868544609A52
Using PEAR installation found at /Users/rasmus/lib
Discovery of channel pear.drush.org successful

% php pyrus.phar list-channels
Pyrus version 2.0.0a4 SHA-1: 72271D92C3AA1FA96DF9606CD538868544609A52
Using PEAR installation found at /Users/rasmus/lib
Listing channels [/Users/rasmus/lib]:
__uri (__uri)
doc.php.net (phpdocs)
pear.drush.org (drush)
pear.php.net (pear)
pecl.php.net (pecl)

% php pyrus.phar remote-list drush
Pyrus version 2.0.0a4 SHA-1: 72271D92C3AA1FA96DF9606CD538868544609A52
Using PEAR installation found at /Users/rasmus/lib
Remote packages for channel drush:
Default:
   drush                     6.2.0.0   command line shell and Unix scripting
                                       interface for Drupal
Key: * = installed, ! = upgrades available
```

Чтобы установить файл из канала, укажите имя канала и символ / перед именем пакета. Например, установка пакета `drush` из канала `drush` выполняется так:

```
% php pyrus.phar install drush/drush
Using PEAR installation found at /Users/rasmus/lib
Downloading pear.drush.org/drush
Mime-type: application/x-tar
[=========================================================>] 100% (494/494 kb)
Installed pear.drush.org/drush-6.2.0.0
```

См. также

Рецепт 27.6 — получение дополнительной информации о пакете.

27.6. Поиск информации о пакете

Задача

Требуется собрать информацию о пакете: что он делает, кто занимается его сопровождением, какая версия установлена и по какой лицензии он опубликован.

Решение

Воспользуйтесь командой Pyrus `info`:

```
% php pyrus.phar info pear/HTTP2
```

Также можно обратиться за информацией на домашнюю страницу пакета.

Комментарий

Команда `info` предоставляет сводную информацию о пакете:

```
% php pyrus.phar info pear/HTTP2
Pyrus version 2.0.0a4 SHA-1: 72271D92C3AA1FA96DF9606CD538868544609A52
Using PEAR installation found at /Users/rasmus/lib
HTTP2 (pear.php.net Channel)
------------------------------------------------------------------------------
Package type: Version:      1.1.1 (API 1.1.0), Stability:    stable (API stable)
Release Date: 2013-10-23 15:33:41
Package Summary: Miscellaneous HTTP utilities
Package Description Excerpt:
   The HTTP class is a class with static methods for doing
   miscellaneous HTTP related stuff like date formatting,
   language negotiation or HTTP redirection....
(`php pyrus.phar info pear/HTTP2 description` for full description)
Release Notes Excerpt:

   - Fix parsing arguments values without quotes...
(`php pyrus.phar info pear/HTTP2 notes` for full release notes)
```

Если пакет не установлен в вашей системе, программа обращается на удаленный сервер за описанием. Чтобы получить дополнительную информацию (описание пакета или замечания по выпуску), присоедините к запросу условие `description` или `notes`.

Домашняя страница пакета предоставляет более полную информацию и ссылки на ранние выпуски, журнал изменений и средства просмотра репозитория пакета. Также здесь можно просмотреть статистику загрузки пакета. На рис. 27.2 представлен пример страницы с информацией о пакете.

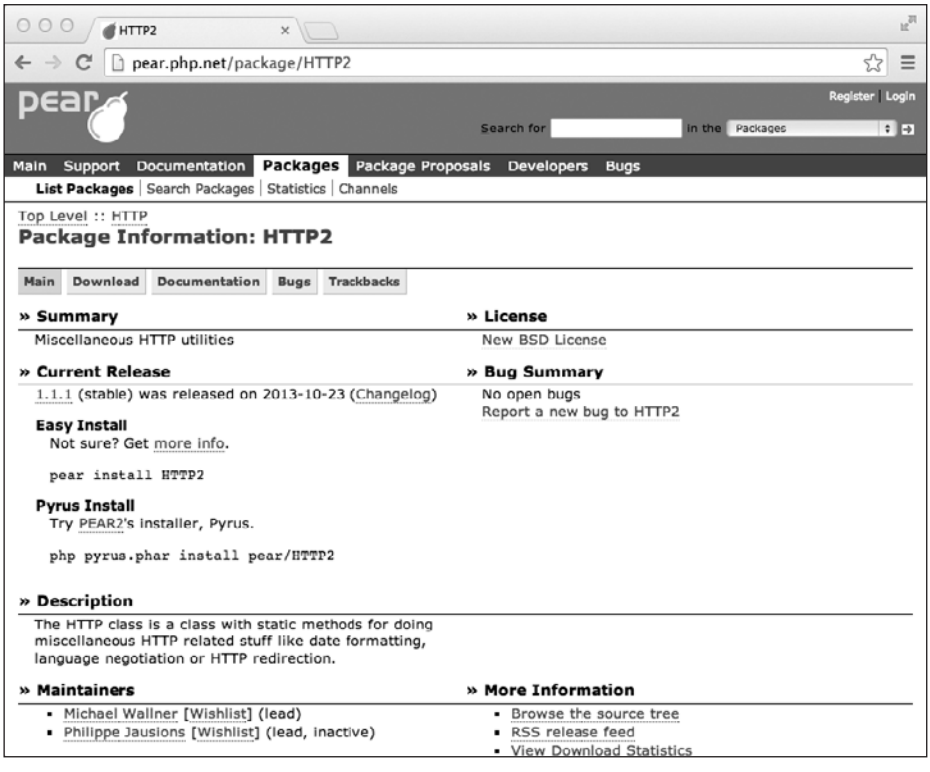

Рис. 27.2. Страница с информацией о пакете HTTP2 на сайте PEAR

См. также

Рецепт 27.5 — поиск пакетов.

27.7. Установка пакетов PEAR

Задача

Требуется установить пакет PEAR.

Решение

Загрузите и установите пакет с сервера канала PEAR с использованием Pyrus:

```
% php pyrus.phar install pear/имя_пакета
```

Пакет также можно установить с другого канала PEAR:

```
% php pyrus.phar install канал/имя_пакета
```

Также возможна установка с произвольного сайта в Интернете:

```
% php pyrus.phar install http://pear.example.com/имя_пакета-1.0.0.tgz
```

А вот как осуществляется установка при наличии локальной копии пакета:

```
% php pyrus.phar install Package_Name-1.0.0.tgz
```

Комментарий

Для установки пакетов PEAR необходимо иметь разрешения записи для каталога, в котором хранятся пакеты; по умолчанию это каталог /usr/local/lib/php/.

Вы также можете запросить несколько пакетов одновременно:

```
% php pyrus.phar install pear/XML_RSS pear/XML_SVG
Pyrus version 2.0.0a4 SHA-1: 72271D92C3AA1FA96DF9606CD538868544609A52
Using PEAR installation found at /Users/rasmus/lib
Downloading pear.php.net/XML_RSS
Mime-type: application/octet-stream
Downloading pear.php.net/XML_SVG==============================>] 100% ( 6/ 6 kb)
Mime-type: application/octet-stream
Installed pear.php.net/XML_RSS-1.0.2==========================>] 100% ( 7/ 7 kb)
Installed pear.php.net/XML_SVG-1.1.0
```

Во время установки пакета Pyrus проверяет доступность всех необходимых функций PHP и пакетов PEAR, от которых зависит новый пакет. Если проверка не проходит, устанавливаются все необходимые зависимости PEAR:

```
% php pyrus.phar install pear/XML_XUL-alpha
Pyrus version 2.0.0a4 SHA-1: 72271D92C3AA1FA96DF9606CD538868544609A52
Using PEAR installation found at /Users/rasmus/lib
Downloading pear.php.net/XML_XUL
Mime-type: application/octet-stream
Downloading pear.php.net/XML_Util2============================>] 100% (26/26 kb)
Mime-type: application/octet-stream
Downloading pear.php.net/XML_Parser2==========================>] 100% (16/16 kb)
Mime-type: application/octet-stream
Installed pear.php.net/XML_XUL-0.9.1==========================>] 100% (11/11 kb)
Installed pear.php.net/XML_Util2-0.1.0
Installed pear.php.net/XML_Parser2-0.1.0
```

Когда пакет PEAR будет установлен, используйте его в своих сценариях PHP вызовом `require_once`. Например, включение пакета `HTTP_Request2` осуществляется следующим образом:

```
require_once 'HTTP/Request2.php';
```

Как правило, если имя пакета содержит символ подчеркивания, этот символ заменяется символом /, а в конец добавляется .php.

Так как пакеты PEAR включатся как обычные файлы PHP, следует убедиться в том, что каталог с пакетами PEAR входит в путь `include_path`; в противном случае `include_once` и `require_once` не смогут найти файлы классов PEAR.

См. также

Рецепт 27.10 — информация об установке пакетов PECL; Рецепт 27.8 — обновление существующих пакетов; Рецепт 27.9 — удаление пакета.

27.8. Обновление пакетов PEAR

Задача

Требуется обновить пакет и заменить его новейшей версией, чтобы использовать дополнительную функциональность и исправить ошибки.

Решение

Проверьте доступность обновлений и прикажите Pyrus обновить нужные пакеты:

```
% php pyrus.phar list-upgrades
% pear upgrade  pear/Package_Name
```

Комментарий

Использование Pyrus упрощает задачу обновления пакетов. Если вам известно, что некоторый пакет устарел, его можно обновить напрямую. Также можно периодически проверять появление новых версий.

Задача решается при помощи команды `list-upgrades`. Команда выводит таблицу, в которой для каждого пакета выводится сервер канала, имя пакета, локальный номер версии и состояние, номер версии и состояние удаленного обновления, а также размер загружаемого обновления:

```
% php pyrus.phar  list-upgrades
Pyrus version 2.0.0a4 SHA-1: 72271D92C3AA1FA96DF9606CD538868544609A52
Using PEAR installation found at /Users/rasmus/lib
(no packages installed in channel __uri)
(no packages installed in channel doc.php.net)
(no upgrades for packages installed in channel pear.drush.org)
Upgrades for channel pear.php.net:
  XML_Beautifier 1.2.2 (stable, released 2010-10-25)
  Console_Getopt 1.3.1 (stable, released 2011-03-08)
  Archive_Tar 1.3.11 (stable, released 2013-02-09)
(no packages installed in channel pecl.php.net)
```

Для обновления конкретного пакета используется команда `upgrade`. Пример:

```
% php pyrus.phar upgrade pear/XML_Beautifier-1.2.2
Pyrus version 2.0.0a4 SHA-1: 72271D92C3AA1FA96DF9606CD538868544609A52
Using PEAR installation found at /Users/rasmus/lib
Downloading pear.php.net/XML_Beautifier
```

```
Mime-type: application/octet-stream
Installed pear.php.net/XML_Beautifier-1.2.2=======>] 100% (14/14 kb)
```

Для команды `list-upgrades` применяется сокращение `lu`, а для команды `upgrade` — сокращение `up`.

В PEAR также поддерживается канал RSS с информацией о новых и обновленных пакетах (*http://pear.php.net/feeds/latest.rss*).

См. также

Рецепты 27.7 и 27.10 — установка пакетов PEAR и PECL; Рецепт 27.9 — удаление пакета; Рецепт 12.12 — разбор данных RSS.

27.9. Удаление пакетов PEAR

Задача

Требуется удалить пакет PEAR из системы.

Решение

Команда `uninstall` приказывает программе установки PEAR удалить установленный пакет:

```
% php pyrus.phar uninstall pear/XML_Beautifier
Pyrus version 2.0.0a4 SHA-1: 72271D92C3AA1FA96DF9606CD538868544609A52
Using PEAR installation found at /Users/rasmus/lib
Uninstalled pear.php.net/XML_Beautifier
```

Комментарий

Удаленный пакет полностью исчезает из системы. Если вы захотите установить его заново, придется действовать так, словно пакет ранее никогда не устанавливался.

При попытке удалить пакет, от которого зависят другие пакеты, Pyrus выдает предупреждение и прерывает процесс удаления. Допустим, имеется следующая установка PEAR:

```
% sudo php pyrus.phar list-packages
Pyrus version 2.0.0a4 SHA-1: 72271D92C3AA1FA96DF9606CD538868544609A52
Using PEAR installation found at /Users/rasmus/lib
Listing installed packages [/Users/rasmus/lib]:
[channel pear.php.net]:
```

```
Archive_Tar 1.3.7 stable
Console_Getopt 1.3.0 stable
HTTP2 1.1.1 stable
PEAR 1.9.4 stable
Structures_Graph 1.0.4 stable
XML_Beautifier 1.2.1 stable
XML_Parser 1.3.4 stable
XML_Parser2 0.1.0 beta
XML_RSS 1.0.2 stable
XML_SVG 1.1.0 stable
% sudo php pyrus.phar list-packages
Pyrus version 2.0.0a4 SHA-1: 72271D92C3AA1FA96DF9606CD538868544609A52
Using PEAR installation found at /Users/rasmus/lib
Listing installed packages [/Users/rasmus/lib]:
[channel pear.php.net]:
Archive_Tar 1.3.7 stable
Console_Getopt 1.3.0 stable
HTTP2 1.1.1 stable
PEAR 1.9.4 stable
Structures_Graph 1.0.4 stable
XML_Beautifier 1.2.1 stable
XML_Parser 1.3.4 stable
XML_Parser2 0.1.0 beta
XML_RSS 1.0.2 stable
XML_SVG 1.1.0 stable
XML_Util 1.2.1 stable
XML_Util2 0.1.0 alpha
```

Вы пытаетесь удалить пакет `XML_Parser`:

```
% php pyrus.phar uninstall pear/XML_Parser
Pyrus version 2.0.0a4 SHA-1: 72271D92C3AA1FA96DF9606CD538868544609A52
Using PEAR installation found at /Users/rasmus/lib
Pyrus\Installer\Exception: Dependency validation failed for some installed
    packages, installation aborted
 Pyrus\Dependency\Exception: channel://pear.php.net/XML_Parser (version >=
     1.0.1, excluded versions: 1.0.1) is required by installed package
     "channel://pear.php.net/XML_RSS"
```

Даже если пакет имеет зависимости, его можно принудительно удалить с флагом `-f` или `--force`; этот флаг приказывает Pyrus игнорировать зависимости. Будьте осторожны при использовании этой возможности.

Команда `uninstall` не позволяет автоматически вернуться к более ранней версии пакета.

Для команды `uninstall` используется сокращение `un`.

См. также

Рецепты 27.7 и 27.10 — установка пакетов PEAR и PECL.

27.10. Установка пакетов PECL

Задача

Требуется установить пакет PECL; для этого расширение PHP, написанное на языке C, строится для использования из PHP.

Решение

Проверьте наличие всех необходимых библиотек расширения и воспользуйтесь встроенной программой установки:

```
% pecl install mailparse
```

Чтобы использовать расширение из PHP, добавьте в файл php.ini соответствующую строку вида:

```
extension=mailparse.so
```

Комментарий

Внешне процесс установки пакетов PECL очень похож на установку пакетов PEAR для кода, написанного на PHP. Тем не менее внутренние процессы очень сильно различаются. Так как расширения PECL написаны на C, программа установки должна откомпилировать расширение и настроить его для работы с установленной версией PHP. Соответственно, в настоящее время пакеты PECL могут строиться на машинах с системой Unix, на которых установлены необходимые средства разработчика.

В отличие от пакетов PEAR на базе PHP, расширения PECL не могут автоматически уведомить об отсутствии библиотеки, необходимой для компиляции расширения. Разработчик сам несет ответственность за предварительную установку этих файлов. Если у вас возникнут проблемы с построением расширений PECL, обратитесь к файлу README и другой документации, поставляемой с пакетом. Программа установки помещает эти файлы в каталог docs в иерархии PEAR.

При установке расширения PECL команда *pecl* загружает дистрибутивный файл, распаковывает его, запускает *phpize* для настройки расширения под версию PHP, установленную на машине, а затем строит и устанавливает расширение. Также она может запросить информацию о местонахождении библиотек:

```
% pecl install memcached
downloading memcached-2.1.0.tgz ...
Starting to download memcached-2.1.0.tgz (39,095 bytes)
.........done: 39,095 bytes
11 source files, building
running: phpize
Configuring for:
```

```
PHP Api Version:         20100412
Zend Module Api No:      20100525
Zend Extension Api No:   220100525
...
Build complete.
...
install ok: channel://pecl.php.net/memcached-2.1.0
You should add "extension=memoize.so" to php.ini
```

Расширения PECL и пакеты PEAR, написанные на PHP, хранятся в разных местах. Для запуска *pecl* необходимо иметь разрешения записи для каталога расширений PHP. По этой причине пакеты желательно устанавливать от имени того же пользователя, который использовался для установки PHP. Также проверьте разрешения на исполнение для этих файлов; так как многие файлы PEAR не являются исполняемыми, текущая маска может не обеспечить правильный набор разрешений для таких исполняемых файлов.

Если PHP и PECL работают в среде Windows, возможно, будет проще загрузить заранее откомпилированные библиотеки DLL со страницы расширения.

См. также

Рецепт 27.7 — установка пакетов PEAR; Рецепт 27.8 — обновление существующих пакетов; Рецепт 27.9 — удаление пакета.

Об авторах

Дэвид Скляр — независимый технологический консультант. Кроме книги «PHP Cookbook», он также является автором книг «Learning PHP 5» (O'Reilly), «Essential PHP Tools» (Apress) и блестящего блога. Дэвид живет в Нью-Йорке и является обладателем ученой степени по информатике, полученной в Йельском университете.

Адам Трахтенберг — руководитель LinkedIn Developer Network, автор книг «Upgrading to PHP 5» и «PHP Cookbook» (O'Reilly). Ранее работал на должности руководителя в области платформ и сервисов для eBay. Адам живет в Маунтин Вью (Калифорния), является обладателем степени бакалавра математики и степени магистра делового администрирования, полученной в Колумбийском университете.

Д. Скляр, А. Трахтенберг

PHP. Рецепты программирования

3-е издание

Перевел с английского *Е. Матвеев*

Заведующий редакцией	*П. Щеголев*
Ведущий редактор	*Ю. Сергиенко*
Художник	*В. Шимкевич*
Корректоры	*С. Беляева, И. Мивриньш*
Верстка	*Л. Соловьева*

ООО «Питер Пресс», 192102, Санкт-Петербург, ул. Андреевская (д. Волкова), 3, литер А, пом. 7Н.

Налоговая льгота — общероссийский классификатор продукции ОК 034-2014, 58.11.12.000 — Книги печатные профессиональные, технические и научные.

Подписано в печать 25.02.15. Формат 70×100/16. Усл. п. л. 63,210. Тираж 1500. Заказ 0000.

Отпечатано в ОАО «Первая Образцовая типография». Филиал «Чеховский Печатный Двор».
142300, Московская область, г. Чехов, ул. Полиграфистов, 1.
Сайт: www.chpk.ru. E-mail: marketing@chpk.ru
факс: 8(496) 726-54-10, телефон: (495) 988-63-87

ИЗДАТЕЛЬСКИЙ ДОМ ПИТЕР®
WWW.PITER.COM

ПРЕДСТАВИТЕЛЬСТВА ИЗДАТЕЛЬСКОГО ДОМА «ПИТЕР»
предлагают профессиональную и популярную литературу по различным направлениям: история и публицистика, экономика и финансы, менеджмент и маркетинг, компьютерные технологии, медицина и психология.

РОССИЯ

Санкт-Петербург: м. «Выборгская», Б. Сампсониевский пр., д. 29а
тел./факс: (812) 703-73-73, 703-73-72; e-mail: sales@piter.com

Москва: м. «Электрозаводская», Семеновская наб., д. 2/1, стр. 1
тел./факс: (495) 234-38-15; e-mail: sales@msk.piter.com

Воронеж: тел.: 8 951 861-72-70; e-mail: voronej@piter.com

Екатеринбург: ул. Бебеля, д. 11а
тел./факс: (343) 378-98-41, 378-98-42; e-mail: office@ekat.piter.com

Нижний Новгород: тел.: 8 960 187-85-50; e-mail: nnovgorod@piter.com

Новосибирск: Комбинатский пер., д. 3
тел./факс: (383) 279-73-92; e-mail: sib@nsk.piter.com

Ростов-на-Дону: ул. Ульяновская, д. 26
тел./факс: (863) 269-91-22, 269-91-30; e-mail: piter-ug@rostov.piter.com

Самара: ул. Молодогвардейская, д. 33а, офис 223
тел./факс: (846) 277-89-79, 229-68-09; e-mail: samara@piter.com

УКРАИНА

Киев: Московский пр., д. 6, корп. 1, офис 33
тел./факс: (044) 490-35-69, 490-35-68; e-mail: office@kiev.piter.com

Харьков: ул. Суздальские ряды, д. 12, офис 10
тел./факс: (057) 7584145, +38 067 545-55-64; e-mail: piter@kharkov.piter.com

БЕЛАРУСЬ

Минск: ул. Розы Люксембург, д. 163
тел./факс: (517) 208-80-01, 208-81-25; e-mail: minsk@piter.com

---

Издательский дом «Питер» приглашает к сотрудничеству зарубежных торговых партнеров или посредников, имеющих выход на зарубежный рынок
тел./факс: (812) 703-73-73; e-mail: spb@piter.com

Издательский дом «Питер» приглашает к сотрудничеству авторов
тел./факс издательства: (812) 703-73-72, (495) 974-34-50

Заказ книг для вузов и библиотек
тел./факс: (812) 703-73-73, доб. 6250; e-mail: uchebnik@piter.com

---

Заказ книг по почте: на сайте www.piter.com; по тел.: (812) 703-73-74, доб. 6225
