Лабораторная № 11

13. Работа с датами

Даты встречаются в повседневной жизни настолько часто, что человеку легко иметь с ними дело, не задумываясь Однако причуды человеческого календаря значительно усложняют его использование в программах К счастью, PHP предоставляет мощные инструментальные средства для работы с датами, делая манипулирование последними относительно простым занятием.

Определение даты с помощью функции `time()`

Встроенная в PHP функция `time()` предоставляет вам всю необходимую информацию о текущей дате и времени. Данная функция возвращает целочисленное значение и не принимает аргументов. Возвращаемое функцией `time()` число не очень похоже на дату, однако это действительно текущая дата.

```
print time(); // вывод будет таким: 1044262012
```

Целочисленное значение, возвращаемое функцией `time()`, равно количеству секунд, которое прошло с полуночи 1 января 1970 года. Эта дата известна как точка UNIX, а количество секунд, прошедшее с того момента, называется *time stamp* (отметка времени), мы же в дальнейшем будем называть это число *абсолютным временем*. Язык PHP предоставляет удобные инструментальные средства для преобразования абсолютного времени в привычную для человека форму.

Преобразование абсолютного времени с помощью `getdate()`

Теперь, когда вы получили абсолютное время, необходимо преобразовать его в удобный для пользователя вид. Функция `getdate()` принимает аргументом абсолютное время и возвращает ассоциативный массив, содержащий информацию о дате. Если вы опустите абсолютное время при вызове, функция `getdate()` будет работать с текущим абсолютным временем, которое возвращает функция `time()`. В таблице 13.1 приведен список и описание элементов массива, возвращаемого функцией `getdate()`.

Таблица 13.1 — Описание ассоциативного массива, возвращаемого функцией `getdate()`

Ключ	Описание	Пример
`seconds`	Секунды после целой минуты (0–59)	`28`
`minutes`	Минуты после целого часа (0–59)	`7`
`hours`	Часы с начала суток (0–23)	`12`
`mday`	День месяца (1–31)	`20`
`wday`	День недели (0–ВС, 1–ПН,… 6–СБ)	`4`
`mon`	Месяц (1–12)	`1`
`year`	Год (4 разряда)	`2003`
`yday`	День года (0–365)	`19`
`weekday`	День недели (название дня)	`Thursday`
`month`	Месяц в году (название месяца)	`January`
`0`	Абсолютное время	`948370048`

В программе из листинга 13.1 для получения информации о дате из абсолютного времени используется функция `getdate()`. При выводе на экран каждого элемента данного массива мы воспользовались конструкцией `foreach`.

Листинг 13.1. Получение информации о времени с помощью `getdate()`

```
<html> <head>
<title> Листинг 13-1. Получение информации о времени
        с помощью getdate()</title> </head> <body>

<?php
$date_array = getdate();
// будет использована текущая дата
foreach ($date_array as $key => $val)
        {print "$key = $val<br>";}
```

```
print "<p>Сегодня:
$date_array[mday]-$date_array[mon]-$date_array[year]";
?>
</body> </html>
```

Результат работы этой программы приведен ниже.

seconds = 52
minutes = 46
hours = 14
mday = 3
wday = 1
mon = 2
year = 2003
yday = 33
weekday = Monday
month = February
0 = 1044262012

Сегодня: 3-2-2003

Функция `getdate()` возвращает дату в соответствии с локальным часовым поясом.

Преобразование абсолютного времени с помощью функции `date()`

Функция `getdate()` удобна в том случае, если вам нужны элементы массива, который возвращает эта функция. Однако иногда может понадобиться просто вывести текущую дату в виде строки. Функция `date()` возвращает отформатированную строку, где записана дата. Вы можете управлять форматом возвращаемой строки с помощью достаточно большого количества аргументов, которые вы должны передать этой функции в виде строки.

В дополнение к строке формата функция `date()` принимает абсолютное время. В табл. 13.2 приведены символы, которые может содержать строка формата. Любая другая информация, включенная в строку формата, будет вставлена в возвращаемую строку.

Таблица 13.2 — Коды формата, принимаемые функцией `date()`

Код формата	**Описание**	**Пример**
`a`	"am" или "pm", нижний регистр	`pm`
`A`	"AM" или "PM", верхний регистр	`PM`
`d`	День месяца	`20`
`D`	День недели (три буквы)	`Thu`
`F`	Название месяца	`January`
`h`	Час (12-часовой формат — дополнение нулями)	`02`
`H`	Час (24-часовой формат — дополнение нулями)	`14`
`g`	Час (12-часовой формат — без дополнения нулями)	` 2`
`G`	Час (24-часовой формат — без дополнения нулями)	`14`
`i`	Минуты	`47`
`j`	День месяца	`20`
`l`	День недели (название)	`Thursday`
`L`	Високосный год ("1" — да, "0" — нет)	`1`
`m`	Месяц года (число, дополненное нулями)	`01`
`M`	Месяц года (три буквы)	`Jan`
`n`	Месяц года (число без дополнения нулями)	`1`
`s`	Секунды	`24`
`U`	Абсолютное время	`948372444`
`y`	Год (два разряда)	`03`
`Y`	Год (четыре разряда)	`2003`
`z`	День года (0–365)	`19`
`Z`	Время в секундах от GMT	`0`

В листинге 13.2 показано использование символов формата.

Листинг 13.2. Форматирование даты с помощью функции `date()`

```
<html> <head>
<title> Листинг 13-2. Форматирование даты
        с помощью функции date() </title>
</head> <body>
<?php
print date("d-m-y G:i.s<br>", time());
     // 10-02-03  13:45.01
print "<p>Now ".date("j of F Y, g:i a", time());
     // Now 10 of February 2003, 1:45 pm
?>
 </body> </html>
```

Функция `date()` возвращает данные о времени в соответствии с локальным временным поясом. Если вы хотите получить дату в формате GMT, то нужно использовать функцию `gmdate()`, которая работает так же, как и `date()`.

Создание абсолютного времени с помощью функции `mktime()`

Вы уже можете получать информацию о текущем времени, но вам еще не известно, как работать с произвольными датами. Функция `mktime()` возвращает абсолютное время, которое вы затем можете использовать с функциями `date()` или `getdate()`. Функция `mktime()` принимает до шести целочисленных аргументов в следующем порядке:

```
час – минуты – секунды – месяц - день месяца - год
```

В примере из листинга 13.3 для получения абсолютного времени используется функция `mktime()`. Это абсолютное время затем передается функции `date()`.

Листинг 13.3. Создание абсолютного времени с помощью функции `mktime()`

```
<html> <head>
<title> Листинг 13-3. Создание абсолютного времени
        с помощью функции mktime()</title>
</head> <body>
```

```
<?php
// создадим абсолютное время, соответствующее дате
// 1-09-03 13:45
$ts = mktime(13, 45, 0, 9, 1, 2003);
print date("<p>d-m-y G:i.s", $ts);
// 01-09-03 13:45.00
print date("<p>j of F Y, g:i a", $ts);
// 1 of September 2003, 1:45 pm
?>
</body> </html>
```

Вы можете опустить некоторые аргументы функции `mktime()`. При этом будет использовано значение, соответствующее текущему времени. Кроме того, функция `mktime()` автоматически транслирует значения, выходящие за допустимые пределы. Таким образом время, равное 25 часам, будет транслировано в 1.00 am следующего по календарю дня, с учетом месяца, дня и года.

Проверка даты с помощью `checkdate()`

Иногда вам может понадобиться принимать информацию о дате от пользователя. Перед обработкой даты или сохранением ее в базе данных вы должны убедиться, что эта дата лежит в допустимых пределах. Функция `checkdate()` принимает три параметра: месяц, день и год. Эта функция возвращает `true`, если месяц лежит между 1 и 12, день месяца лежит в приемлемом для данного месяца диапазоне (с учетом високосных годов), а год находится в диапазоне между 0 и 32767. Будьте внимательны при работе с датами: дата может находиться в правильном диапазоне, но не приниматься остальными функциями работы с датами. Например, приведенная ниже строка вернет значение `true`:

```
checkdate(2, 8, 1724)
```

Однако если вы попытаетесь создать дату, передавая эти значения функции `mktime()`, то получите абсолютное время, равное «–1». Существует эмпирическое правило: нельзя использовать функцию `mktime()` для годов, ранее 1902, а также следует использовать ее осторожно для годов до 1970.

14. Регулярные выражения

Возможности эффективной организации, поиска и распространения информации давно представляли интерес для специалистов в области компьютерных технологий. Поскольку информация в основном представляет собой текст, состоящий из алфавитно-цифровых символов, разработка средств поиска и обработки информации по шаблонам, описывающим текст, стала предметом серьезных теоретических исследований.

Поиск по шаблону позволяет не только находить определенные фрагменты текста, но и заменять их другими фрагментами. Одним из стандартных примеров поиска по шаблону являются команды поиска/замены в текстовых редакторах — например, в MS Word. Всем пользователям UNIX хорошо известны такие программы, как sed, awk и grep; богатство возможностей этих программ в значительной степени обусловлено средствами поиска по шаблону. Механизмы поиска по шаблону решают четыре основные задачи:

- поиск строк, в точности совпадающих с заданным шаблоном;
- поиск фрагментов строк, совпадающих с заданным шаблоном;
- замену строк и подстрок по шаблону;
- поиск строк, с которыми заданный шаблон *не совпадает*.

Появление Web породило необходимость в более быстрых и эффективных средствах поиска данных, которые бы позволяли пользователям со всего мира находить нужную информацию среди миллиардов web-страниц. Поисковые системы, он-лайновые финансовые службы и сайты электронной коммерции — все это стало бы абсолютно бесполезным без средств анализа гигантских объемов данных в этих секторах. Действительно, средства обработки строковой информации являются жизненно важной составляющей практически любого сектора, так или иначе связанного с современными информационными технологиями.

В этой главе основное внимание посвящено средствам обработки строк в PHP. Мы рассмотрим некоторые стандартные строковые функции (в языке их больше 60!), а из приведенных определений и примеров вы получите сведения, необходимые для создания web-приложений. Но прежде чем переходить к специфике PHP, необходимо познакомиться с базовым механизмом, благодаря которому становится возможным поиск по шаблону. Речь идет о регулярных выражениях.

Регулярные выражения

Регулярные выражения лежат в основе всех современных технологий поиска по шаблону. Регулярное выражение представляет собой последовательность простых и служебных символов, описывающих искомый текст. Иногда регулярные выражения бывают простыми и понятными (например, слово dog), но часто в них присутствуют служебные символы, обладающие особым смыслом в синтаксисе регулярных выражений, — например, `<(?)>.*<\/.?>`.

В PHP существуют два семейства функций, каждое из которых относится к определенному типу регулярных выражений: в стиле POSIX или в стиле Perl. Каждый тип регулярных выражений обладает собственным синтаксисом и рассматривается в соответствующей части главы.

Синтаксис регулярных выражений (POSIX)

Структура регулярных выражений POSIX чем-то напоминает структуру типичных математических выражений — различные элементы (операторы) объединяются друг с другом и образуют более сложные выражения. Однако именно смысл объединения элементов делает регулярные выражения таким мощным и выразительным средством. Возможности не ограничиваются поиском литерального текста (например, конкретного слова или числа); вы можете провести поиск строк с разной семантикой, но похожим синтаксисом — например, всех тегов HTML в файле.

Простейшее регулярное выражение совпадает с одним литеральным символом — например, выражение `"я"` совпадает в таких строках, как `"я"`, `"яма"` и `"синяя"`. Выражение, полученное при объединении нескольких литеральных символов, совпадает по тем же правилам — например, последовательность `"лес"` совпадает в любой строке, содержащей эти символы (например, `"прелестный"`, `"колесо"` или `"лесной"`).

Оператор `|` (вертикальная черта) проверяет совпадение одной из нескольких альтернатив. Например, регулярное выражение `"php|html"` проверяет строку на наличие `"php"` или `"html"`.

Квадратные скобки

Квадратные скобки (`[ ]`) имеют особый смысл в контексте регулярных выражений — они означают «любой символ из перечисленных в скобках». В отличие от регулярного выражения `"php"`, которое совпадает во всех строках, содержащих литеральный текст `"php"`, выражение `[php]` совпадает в любой строке, содержащей символы `"p"` или `"h"`. Квадратные скобки играют важную роль при работе с регулярными выражениями, поскольку в процессе поиска часто возникает задача поиска символов из заданного интервала.

Ниже перечислены некоторые часто используемые интервалы:

- `[0-9]` — совпадает с любой десятичной цифрой от 0 до 9;
- `[a-z]` — совпадает с любым символом нижнего регистра от a до z;
- `[A-Z]` — совпадает с любым символом верхнего регистра от A до Z;
- `[a-Z]` — совпадает с любым символом нижнего или верхнего регистра от a до Z.

Конечно, перечисленные выше интервалы всего лишь демонстрируют общий принцип. Например, вы можете воспользоваться интервалом `[0-3]` для обозначения любой десятичной цифры от `0` до `3` или интервалом `[в-л]` для обозначения любого символа нижнего регистра от `в` до `л`. Короче говоря, интервалы определяются совершенно произвольно.

Квантификаторы

Существует особый класс служебных символов, обозначающих количество повторений отдельного символа или конструкции, заключенной в квадратные скобки. Эти служебные символы (`+`, `*` и `{...}`) называются *квантификаторами*. Принцип их действия проще всего пояснить на примерах:

- `p+` означает один или несколько символов `p`, стоящих подряд;
- `p*` означает ноль и более символов `p`, стоящих подряд;
- `p?` означает ноль или один символ `p`;
- `p{2}` означает два символа `p`, стоящих подряд;
- `p{2,3}` означает от двух до трех символов `p`, стоящих подряд;
- `p{2,}` означает минимум два и более символов `p`, стоящих подряд.
- `p{,3}` означает не более трех символов `p`, стоящих подряд.

Прочие служебные символы

Служебные символы `$` и `^` совпадают не с символами, а с определенными позициями в строке. Например, выражение `p$` означает строку, которая завершается символом `"p"`, а выражение `^p` — строку, начинающуюся с символа `"p"`.

- Конструкция `[^a-zA-Z]` совпадает с любым символом, *не входящим* в указанные интервалы (`a-z` и `A-Z`).
- Служебный символ `.` (точка) означает «любой символ». Например, выражение `"p.p"` совпадает с символом `"p"`, за которым следует произвольный символ, после чего опять следует символ `"p"`.

Объединение служебных символов приводит к появлению более сложных выражений. Рассмотрим несколько примеров:

- `^.{2}$` — любая строка, содержащая *ровно* два символа;
- `<b>(.*)</b>` — произвольная последовательность символов, заключенная между `<b>` и `</b>`;
- `p(hp)*` — символ `"p"`, за которым следует ноль и более экземпляров последовательности `"hp"` (например, `"phphphp"`).

Иногда требуется найти служебные символы в строках вместо того, чтобы использовать их в описанном специальном контексте. Для этого служебные символы экранируются обратной косой чертой (`\`). Например, для поиска денежной суммы в долларах можно воспользоваться выражением `\$[0-9]+`, то есть «знак доллара, за которым следует одна или несколько десятичных цифр». Обратите внимание на обратную косую черту перед `$`. Возможными совпадениями для этого регулярного выражения являются `$42`, `$560` и `$3`.

Стандартные интервальные выражения (символьные классы)

Для удобства программирования в стандарте POSIX были определены некоторые стандартные интервальные выражения, также называемые *символьными классами* (character classes). Символьный класс определяет один символ из заданного интервала — например, букву алфавита или цифру:

`[[:alpha:]]` — алфавитный символ (`aA-zZ`);

`[[:digit:]]` — цифра (`0-9`);

`[[:alnum:]]` — алфавитный символ (`aA-zZ`) или цифра (`0-9`);

`[[:space:]]` — пропуски (символы новой строки, табуляции и т. д.).

Функции PHP для работы с регулярными выражениями (POSIX-совместимые)

В настоящее время PHP поддерживает семь функций поиска с использованием регулярных выражений в стиле POSIX:

- `ereg();`
- `ereg_replace();`
- `eregi();`
- `eregi_replace();`
- `split();`
- `spliti();`
- `sql_regcase().`

ereg()

Функция `ereg()` ищет в заданной строке совпадение для шаблона. Если совпадение найдено, возвращается TRUE, в противном случае возвращается FALSE.

Поиск производится с учетом регистра алфавитных символов. Пример использования `ereg()` для поиска в строках доменов `".ru"`:

```
$is_ru = ereg("(\.)(ru$)", $domain);
// Поскольку точка является служебным символом,
// ее необходимо экранировать
// Функция возвращает TRUE,
// если $domain завершается символами ".ru"
// В частности, поиск будет успешным для строк
// "www.ngs.ru" и "helen@mail.ru"
```

Обратите внимание: из-за присутствия служебного символа `$` регулярное выражение совпадает только в том случае, если строка завершается символами `".ru"`. Например, оно совпадет в строке `"www.ngs.ru",` но не совпадет в строке `"www.ngs.ru/weather".`

Третий параметр «совпадения» содержит массив совпадений для всех подвыражений, заключенных в регулярном выражении в круглые скобки. В листинге 14.1 показано, как при помощи этого массива разделить URL на несколько сегментов.

Листинг 14.1. Применение функции `ereg()`

```
<html> <head>
<title> Листинг 14-1. Применение функции ereg()
</title> </head> <body>
<?php
$url = "http://www.ngs.ru";
// Разделить $url на три компонента:
// "http://www", "ngs" и "ru"
$www_url = ereg("^(http://www)\.([[:alnum:]]+)\.
([[:alnum:]]+)",
     $url, $regs);
if ($www_url) { // Если переменная $www_url содержит URL
foreach ($regs as $val) {print "<p>$val";}
}
?>
</body> </html>
```

При выполнении сценария в листинге 14.1 будет получен следующий результат:

```
http://www.ngs.ru
http://www
ngs
ru
```

ereg_replace()

Функция `ereg_replace()` ищет в заданной строке совпадение для шаблона и заменяет его новым фрагментом. Функция `ereg_replace()` работает по тому же принципу, что и `ereg()`, но ее возможности расширены от простого поиска до поиска с заменой. После выполнения замены функция возвращает модифицированную строку. Если совпадения отсутствуют, строка остается в прежнем состоянии. Функция `ereg_replace()`, как и `ereg()`, учитывает регистр символов. Ниже приведен простой пример, демонстрирующий применение этой функции:

```
$copy_date = "Copyright 2002";
$copy_date = ereg_replace("([0-9]+)",
                          "2003", $copy_date);
print $copy_date; // Выводится строка "Copyright 2003"
```

У средств поиска с заменой в языке PHP имеется одна интересная возможность — использование обратных ссылок на части основного выражения, заключенные в круглые скобки. Обратные ссылки похожи на элементы третьего необязательного параметра (массива совпадений) функции `ereg()` за одним исключением: обратные ссылки записываются в виде `\\0, \\1, \\2` и т.д., где `\\0` соответствует всей строке, `\\1` — успешному совпадению первого подвыражения и т. д. Выражение может содержать до 9 обратных ссылок. В листинге 14.2 все ссылки на URL в тексте заменяются работающими гиперссылками.

Листинг 14.2. Применение функции `ereg_replace()`

```
<html> <head>
<title> Листинг 14-2. Применение функции ereg_replace()
</title> </head> <body>
<?php
$url = "Мой сайт (http://aaa.narod.ru)";
$url = ereg_replace("http://(([A-Za-z0-9.\-])*)",
"<a href=\"\\0\">\\0</a>", $url);
print $url;
//Выводится строка: Мой сайт
//(<a href="http://aaa.narod.ru">http://aaa.narod.ru</a>)
?>
</body> </html>
```

eregi()

Функция `eregi()` ищет в заданной строке совпадение для шаблона. Поиск производится *без учета* регистра алфавитных символов. Функция `eregi()` особенно удобна при проверке правильности введенных строк (например, паролей). Использование функции `eregi()` продемонстрировано в следующем примере:

```
$password = "abc";
if (!eregi("[[:alnum:]]{8,10}", $password)){
print "Неверный пароль! Длина пароля должна быть от 8 до
10 символов."; }
```

В результате выполнения этого фрагмента выводится сообщение об ошибке, поскольку длина строки `"abc"` не входит в разрешенный интервал от 8 до 10 символов. **Внимание!** Шаблон `[:alnum:]` корректно работает **только для латинских символов.** Для русских символов поиск без учета регистра возможен только так: `$is_rus = eregi("[а-я]", $rus_text);`

eregi_replace()

Функция `eregi_replace()` работает точно так же, как `ereg_replace()`, за одним исключением: поиск производится без учета регистра символов.

split()

Функция `split()` разбивает строку на элементы, границы которых определяются по заданному шаблону. Необязательный параметр определяет максимальное количество элементов, на которые делится строка слева направо. Если шаблон содержит алфавитные символы, функция `split()` работает с учетом регистра символов. Следующий пример демонстрирует использование функции `split()` для разбиения IP-адреса на триплеты:

```
$ip = "62.76.77.134"; // IP-адрес
$ip_arr = split ("\.", $ip);
// Поскольку точка является служебным символом,
// ее необходимо экранировать

print "$ip_arr[0] <br>"; // Выводит "62"
print "$ip_arr[1] <br>"; // Выводит "76"
print "$ip_arr[2] <br>"; // Выводит "77"
print "$ip_arr[3] <br>"; // Выводит "134"
```

spliti()

Функция `spliti()` работает точно так же, как ее прототип `split()`, за одним исключением: она не учитывает регистра символов. Разумеется, регистр символов важен лишь в том случае, если шаблон содержит алфавитные символы. Для других символов выполнение `spliti()` полностью аналогично `split()`.

sql_regcase()

Вспомогательная функция `sql_regcase()` заключает каждый символ входной строки в квадратные скобки и добавляет к нему парный символ. Если алфавитный символ существует в двух вариантах (верхний и нижний регистры), выражение в квадратных скобках будет содержать оба варианта; в противном случае исходный символ повторяется дважды. Функция `sql_regcase()` особенно удобна при использовании PHP с программными пакетами, поддерживающими регулярные выражения в одном регистре. Пример преобразования строки функцией `sql_regcase()`:

```
$version = "php 4.0";
print sql_regcase($version);
// Выводится строка [Pp][Hh][Pp] 4.0
```

Синтаксис регулярных выражений в стиле Perl

Perl давно считается одним из самых лучших языков обработки текстов. Синтаксис Perl позволяет осуществлять поиск и замену даже для самых сложных шаблонов. Разработчики PHP сочли, что не стоит заново изобретать уже изобретенное, а лучше сделать знаменитый синтаксис регулярных

выражений Perl доступным для пользователей PHP. Так появились функции для работы с регулярными выражениями в стиле Perl.

Диалект регулярных выражений Perl не так уж сильно отличается от диалекта POSIX. В сущности, синтаксис регулярных выражений Perl является отдаленным потомком реализации POSIX, вследствие чего синтаксис POSIX почти совместим с функциями регулярных выражений стиля Perl.

Оставшаяся часть этого раздела будет посвящена краткому знакомству с диалектом регулярных выражений Perl. Рассмотрим простой пример:

```
/food/
```

Обратите внимание: строка `food` заключена между двумя косыми чертами. Как и в стандарте POSIX, вы можете создавать более сложные шаблоны при помощи квантификаторов:

```
/fo+/
```

Этот шаблон совпадает с последовательностью `"fo"`, за которой могут следовать дополнительные символы `"o"`. Например, совпадения будут обнаружены в строках `"food"`, `"fool"` и `"fo4"`.

Рассмотрим другой пример использования квантификатора: `/fo{2,4}/`

Шаблон совпадает с символом `"f"`, за которым следуют от 2 до 4 экземпляров символа `"o"`. К числу потенциальных совпадений относятся строки `"fool"`, `"fooool"` и `"foosball"`.

В регулярных выражениях Perl могут использоваться все квантификаторы, упомянутые в предыдущем разделе для регулярных выражений POSIX.

Модификаторы

Модификаторы заметно упрощают работу с регулярными выражениями. Впрочем, модификаторов много, и в таблице 14.1 приведены лишь наиболее интересные из них. Модификаторы перечисляются сразу же после регулярного выражения — например, `/string/i`.

Таблица 14.1 — Примеры модификаторов

Модификатор	Описание
/m	Фрагмент текста интерпретируется как состоящий из нескольких «логических строк». По умолчанию специальные символы ^ и $ совпадают только в начале и в конце всего фрагмента. При включении «многострочного режима» при помощи модификатора m^ и $ будут совпадать в начале и в конце каждой логической строки внутри фрагмента
/s	По смыслу противоположен модификатору m — при поиске фрагмент интерпретируется как одна строка, а все внутренние символы новой строки игнорируются
/i	Поиск выполняется без учета регистра символов
/x	Пустые символы вне символьных классов не находятся для улучшения читабельности. Для нахождения пустых символов используйте \s
/e	Обрабатывать строку замены в функции preg_replace() как PHP-код
/A	Находит шаблон только в начале строки
/E	Находит шаблон только в конце строки
/U	Сопоставление с минимальным количеством подстрок — будет найдено минимальное количество совпадений

Escape-последовательности

Одной из интересных особенностей Perl является использование escape-последовательности при поиске. Escape-последовательность представляет собой алфавитный символ с префиксом \ — признаком особой интерпретации следующего символа. Например, escape-последовательность \d может использоваться при поиске денежных сумм: /([\d]+)000/

Комбинация \d обозначает любую цифру. Конечно, в процессе поиска часто возникает задача идентификации алфавитно-цифровых символов, поэтому в Perl для них был определена escape-последовательность \w:

```
/<([\w]+)>/
```

Этот шаблон совпадает с конструкциями, заключенными в угловые скобки, — например, тегами HTML. Кстати, escape-последовательность \W имеет прямо

противоположный смысл и используется для идентификации символов, не являющихся алфавитно-цифровыми.

Еще одна полезная escape-последовательность, `\b`, совпадает с границами слов:

```
/па\b/
```

Поскольку escape-последовательность границы слова расположена справа от текста, этот шаблон совпадет в строках `"группа"` и `"Европа"`, но не в строке `"парадокс"`. Противоположная escape-последовательность, `\B`, совпадает с чем угодно, кроме границы слова: `/на\B/`

Шаблон совпадает в таких строках, как `"начало"` и `"канат"`, но не совпадает в строке `"корона"`.

Функции PHP для работы с регулярными выражениями (Perl-совместимые)

В PHP существует пять функций поиска по шаблону с использованием Perl-совместимых регулярных выражений:

- `preg_match();`
- `preg_match_all();`
- `preg_replace();`
- `preg_split();`
- `preg_grep().`

preg_match()

Функция `preg_match()` ищет в заданной строке совпадение для шаблона. Если совпадение найдено, возвращается TRUE, в противном случае возвращается FALSE. В листинге 14.3 функция `preg_match()` используется для проведения поиска без учета регистра, при этом escape-последовательности позволяют значительно упростить регулярные выражения.

Листинг 14.3. Применение функции `preg_match()`

```
<html> <head>
<title> Листинг 14-3. Применение функции preg_match()
</title> </head> <body>
```

```
<?php
if (preg_match ("/\bweb\b/i", "World Wide Web"))
     {print "<p>Строка найдена.";}
else {print "<p>Строка не найдена.";}

if (preg_match ("/\bweb\b/i", "PHP is the website
                  scripting language."))
     {print "<p>Строка найдена.";}
else {print "<p>Строка не найдена.";}

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

Функция `preg_match()` будет находить строку `"web"`, но только в том случае, если эта строка является целым словом, а не частью слова, как `"webbing"` (тесьма) или `"cobweb"` (паутина).

preg_match_all()

Функция `preg_match_all()` находит все совпадения шаблона в заданной строке.

Следующий пример показывает, как при помощи функции `preg_match_all()` найти весь текст, заключенный между тегами HTML `<b> </b>`:

```
$user_info = "Name <b>Rasmus Lerdorf</b>
<br> Title <b>PHP Guru</b>";

preg_match_all("/<b>(.*)<\/b>/U",
               $user_info, $res_array);

print $res_array[0][0]."<br>".$res_array[0][1]."\n";
```

Результат:

Rasmus Lerdorf
PHP Guru

preg_replace()

Функция `preg_replace()` ведет себя так же, как и функция `ereg_replace()`, за исключением того, что у вас есть расширенные возможности совместимых с Perl регулярных выражений. Функции `preg_replace()` передается регулярное выражение, строка, на которую нужно заменять, и исходная строка. Если была найдена совпадающая подстрока, эта функция возвращает модифицированную строку; в противном

случае возвращается копия исходной строки. Приведенный ниже фрагмент преобразовывает дату в формате `mm/dd/yy` в формат `dd/mm/yy`:

```
$t = "12/25/02, 5/14/03";
$t = preg_replace("|\b(\d+)/(\d+)/(\d+)\b|",
                  "\\2/\\1/\\3", $t);
print "$t<br>";
// выводит "25/12/02, 14/5/03"
```

Обратите внимание, что мы использовали символ (`|`) как символ разделителя.

Другие строковые функции

Кроме функций для работы с регулярными выражениями, описанными в первой части этой главы, в PHP существует более 70 функций для выполнения практически всех мыслимых операций со строками. Подробное перечисление и описание всех функций выходит за рамки этой книги и приведет к обычному повторению информации, приведенной в документации PHP. По этой причине я превратил оставшуюся часть главы в своего рода список FAQ из вопросов, часто встречающихся во многих электронных конференциях PHP и на многих сайтах этой тематики. На мой взгляд, этот способ позволяет гораздо эффективнее описать общие принципы громадной библиотеки строковых функций PHP.

Дополнение и сжатие строк

В процессе форматирования часто возникает необходимость в изменении длины строки посредством дополнения или удаления символов. В PHP существует несколько функций, предназначенных для решения этой задачи.

chop()

Функция `chop()` возвращает строку после удаления из нее завершающих пропусков и символов новой строки. В следующем примере функция `chop()` удаляет лишние символы новой строки:

```
$header = "Table of Contents\n\n"
$header = chop($header)
// $header = "Table of Contents"
```

str_pad()

Функция `str_pad()` выравнивает строку до определенной длины заданными символами и возвращает отформатированную строку.

strstr()

Функция `strstr()` возвращает часть строки, начинающуюся с первого вхождения заданной подстроки. В следующем примере функция `strstr()` используется для выделения имени домена из URL:

```
$url = "http://www.ngs.ru";
$domain = strstr($url, "."); // $domain = "ngs.ru"
```

Преобразование текста в HTML

Быстрое преобразование простого текста к формату web-браузера — весьма распространенная задача. В ее решении вам помогут функции, описанные в этом разделе.

nl2br()

Функция `nl2br()` заменяет все символы новой строки `(\n)` эквивалентными html-командами `<br>`. Символы новой строки могут быть как видимыми (то есть явно включенными в строку), так и невидимыми (например, введенными в редакторе).

htmlspecialchars()

Функция `htmlspecialchars()` заменяет некоторые символы, имеющие особый смысл в контексте HTML, эквивалентными конструкциями HTML. Функция `htmlspecialchars()` в настоящее время преобразует следующие символы:

& преобразуется в `&` **"** преобразуется в `&quot;`

< преобразуется в `<` **>** преобразуется в `>`

В частности, эта функция позволяет предотвратить ввод пользователями разметки HTML в интерактивных web-приложениях (например, в электронных форумах). Ошибки, допущенные в разметке HTML, могут привести к тому, что вся страница будет формироваться неправильно. Впрочем, у этой задачи существует и более эффективное решение — полностью удалить теги из строки функцией `strip_tags()`.

Следующий пример демонстрирует удаление потенциально опасных символов функцией `htmlspecialchars()`:

```
$user_input="I can’t get <<enough>> of PHP & MySQL";
$res = htmlspecialchars($user_input);

// $res="I can't get &lt;&lt;enough&gt;&gt;
// of PHP &amp; MySQL"
```

Если функция `htmlspecialchars()` используется в сочетании с `nl2br()`,то последнюю следует вызывать после `htmlspecialchars()`. В противном случае конструкции `<br>`, сгенерированные при вызове `nl2br()`, преобразуются в видимые символы.

Преобразование HTML в простой текст

Иногда возникает необходимость преобразовать файл в формате HTML в простой текст. Функции, описанные ниже, помогут вам в решении этой задачи.

strip_tags()

Функция `strip_tags()` удаляет из строки все теги HTML и PHP, оставляя в ней только текст. Ниже приведен пример удаления из строки всех тегов HTML функцией `strip_tags()`:

```
$user_input="I <b>like</b> PHP and <i> MySQL</i>";
$res = strip_tags($user_input);
//$res = "I like PHP and MySQL";
```

Идентификация браузера

Каждый программист, пытающийся создать удобный web-сайт, должен учитывать различия в форматировании страниц при просмотре сайта в разных браузерах и операционных системах. Хотя консорциум W3 (http://www.w3.org) продолжает публиковать стандарты, которых должны придерживаться программисты при создании web-приложений, разработчики браузеров любят дополнять эти стандарты своими маленькими «усовершенствованиями», что в конечном счете вызывает хаос и путаницу. Разработчики часто решают эту проблему, создавая разные страницы для каждого типа браузера и операционной системы - при этом объем работы значительно увеличивается, но зато итоговый сайт идеально подходит для любого пользователя. Результат — хорошая репутация сайта и уверенность в том, что пользователь посетит его снова.

Чтобы пользователь мог просматривать страницу в формате, соответствующем специфике его браузера и операционной системы, из входящего запроса на получение страницы извлекается информация о браузере и платформе. После получения необходимых данных пользователь перенаправляется на нужную страницу.

Приведенный листинг 14.4 показывает, как использовать функции PHP для работы с регулярными выражениями. Программа определяет тип и версию браузера и операционной системы, после чего выводит полученную информацию в окне браузера.

Но прежде чем переходить к непосредственному анализу программы, необходимо представить один из главных ее компонентов — серверную

переменную PHP `$_SERVER['HTTP_USER_AGENT']`. В этой переменной в строковом формате хранятся различные сведения о браузере и операционной системе пользователя — именно то, что нас интересует. Эту информацию можно легко вывести на экран всего одной командой:

```
print $_SERVER['HTTP_USER_AGENT'];
```

При работе в Internet Explorer 7.0 на компьютере с Windows XP результат будет выглядеть так:

Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; InfoPath.1)

Скрипт при помощи функций обработки строк и регулярных выражений извлекает необходимые данные из `$_SERVER['HTTP_USER_AGENT']`.

Листинг 14.4. Идентификация браузера

```
<html> <head>
<title> Листинг 14-4. Идентификация браузера
</title> </head> <body>
<?php
function browser_info ($agent) {
// Назначение: возвращает тип и версию браузера
    global $type_vers;
// Определить тип браузера
// Искать строку MSIE (Internet Explorer)
    if (ereg('MSIE ([0-9].[0-9]{1,2})',
        $agent, $version)) {
         $browse_type = "Internet Explorer";
         $browse_version = $version[1];}

// Искать строку Opera
    elseif (ereg('Opera ([0-9].[0-9]{1,2})',
                 $agent, $version)) {
        $browse_type = "Opera";
        $browse_version = $version[1];}
// Искать строку Firefox
    elseif (ereg('Firefox/([0-9].[0-9]{1,2})',
            $agent, $version)) {
        $browse_type = "Mozilla Firefox";
        $browse_version = $version[1];}
```

```
// Если это не Internet Explorer, Opera или Firefox -
// значит, мы обнаружили неизвестный браузер.
    else {
        $browse_type = "Неизвестно";
        $browse_version = "Неизвестно";
          }
$type_vers = array($browse_type, $browse_version);
return $type_vers;
} # Конец функции browser_info
function opsys_info($agent) {
// Назначение: возвращает информацию об операционной
// системе пользователя
// Идентифицировать операционную систему
// Искать строку Windows
    if ( strstr ($agent, 'Win') ) {
        $opsys = "Windows";}
// Искать строку Linux
    elseif ( strstr($agent, 'Linux') ) {
        $opsys = "Linux";}
// Неизвестная платформа
    else {
        $opsys = "Неизвестно";
         }
// Вернуть информацию об операционной системе
     return $opsys;
} # Конец функции opsys_info
$agent=$_SERVER['HTTP_USER_AGENT'];
browser_info($agent);
$browse_type = $type_vers[0];
$browse_version = $type_vers[1];
$operating_sys = opsys_info($agent);
print "Тип браузера: $browse_type <br>";
print "Версия браузера: $browse_version <br>";
print "Операционная система: $operating_sys";
?>
</body> </html>
```
