Урок 8

Регулярные выражения

В этой главе вы научитесь выполнять сложные операции со строками при помощи регулярных выражений. Вы узнаете, как с помощью регулярных выражений можно реализовать проверку строк и сложный поиск с заменой.

Введение в регулярные выражения

С помощью регулярных выражений можно написать правила, которые позволяют идентифицировать определенный формат строки. Довольно сложные правила описываются с помощью нескольких символов. Поэтому поначалу они могут показаться немного сложными.

В самом простом виде регулярное выражение содержит строку, которую нужно найти. В более сложном виде задается детальный шаблон символов и строка разбивается на части на основе этого шаблона.

Типы регулярных выражений

В PHP поддерживается два типа регулярных выражений: основанные на расширенном POSIX-синтаксисе, которые рассматриваются в этом уроке, и “перловские” — Perl-Compatible Regular Expression (PCRE).

Оба типа позволяют выполнять одинаковые функции, используя различный синтаксис. Поэтому не нужно знать оба варианта. Тому, кто уже знает Perl, лучше освоить функциональность PCRE вместо POSIX-синтаксиса.

Подробная документация по PCRE находится по адресу: www.php.net/manual/en/ref.pcre.php.

Возможности ereg

Функция ereg в PHP позволяет проверить строку на соответствие регулярному выражению. В примере ниже с помощью простого регулярного выражения находится подстрока PHP в переменной $phrase:

```
$phrase = "Я люблю PHP";
if (ereg("PHP", $phrase)) {
    echo "Выражение найдено";
}
```

Если запустить этот сценарий, можно убедиться, что выражение действительно обнаружило соответствие с переменной $phrase.

Регулярные выражения зависят от регистра, поэтому, если выражение в нижнем регистре, пример не сработает. Чтобы выполнить безразличное к регистру сравнение, нужно использовать eregi:

```
if (eregi("php", $phrase)) {
    echo "Выражение найдено";
}
```

Производительность

Вышеприведенные примеры можно реализовать с помощью менее сложных функций со строками из урока 6, “Обработка строк”, таких как strstr. Сценарий выполняется быстрее, если использовать строковые функции вместо ereg для простых проверок.

Соответствие набору символов

Кроме поиска точной последовательности символов в строке, с помощью регулярных выражений можно искать по списку символов, если заключить их в квадратные скобки. Для этого нужно перечислить все символы, и при обнаружении одного из них, устанавливается соответствие.

Ниже приводится выражение, эквивалентное ранее приведенному с функцией eregi:

```
if (ereg("[Pp][Hh][Pp]", $phrase)) {
    echo " Выражение найдено ";
}
```

В этом выражении проверяется наличие букв P, H и P в заданном порядке следования. При этом не учитывается регистр.

С помощью символа "дефис" можно задать диапазон символов. Для этого нужно поставить его между двумя буквами или цифрами. Например, [A-Z] соответствует любой букве латинского алфавита, а [0-4] соответствует любой цифре от нуля до четырех.

Следующее выражение истинно только в том случае, если $phrase содержит одну из букв в верхнем регистре:

```
if (ereg("[A-Z]", $phrase)) ...
```

С помощью символа ^ можно указать, каких символов не должно быть в искомом выражении. В примере ниже истинное значение возвращается, только если $phrase содержит как минимум один нецифровой символ:

```
if (ereg("[^0-9]", $phrase)) ...
```

Общие классы символов

Можно использовать различные наборы символов в регулярных выражениях. Чтобы проверить наличие буквенных символов, нужно следующее регулярное выражение:

```
[A-Za-z0-9]
```

Аналогичный по работе класс символов можно записать в более наглядном виде:

```
[[:alnum:]]
```

Символы [: и :] свидетельствуют о том, что выражение содержит класс символов. Существующие наборы классов перечислены в табл. 8.1.

Таблица 8.1. Классы символов для регулярных выражений

Название класса	*Описание*
alnum	Все буквы латинского алфавита и цифры, A–Z, a–z, и 0–9
alpha	Все буквы A–Z и a–z
digit	Все цифры 0–9

Окончание табл. 8.1

Название класса	*Описание*
lower	Все буквы в нижнем регистре a–z
print	Все печатные символы, включая пробел
punct	Все символы пунктуации, кроме пробела и класса alnum
space	Все разделительные символы, включая табуляцию и перевод строки
upper	Все буквы в верхнем регистре A–Z

Проверка положения

Все выражения, рассмотренные выше, проверяют наличие шаблона в произвольном месте строки. Но регулярные выражения позволяют также указывать позицию, где ожидается определенный шаблон.

Символ ^, когда не является частью класса переменных, указывает на начало строки, а $ — на окончание. С помощью этих символов проверяется, находится ли символ, соответственно, в начале или в конце переменной $phrase:

```
if (ereg("^[a-z]", $phrase)) ...
if (ereg("[a-z]$", $phrase)) ...
```

Чтобы убедиться в том, что вся строка соответствует регулярному выражению, можно поместить его между символами ^ и $. Например в следующем условии проверяется, что $number содержит только одну цифру.

```
if (ereg("^[[:digit:]]$", $number) ...
```

Знак доллара

Если с помощью выражения нужно найти символ $, необходимо разделить его обратной косой чертой \$, тогда он не трактуется как конец строки.

Если выражение находится в двойных кавычках, нужно использовать \\$. Иначе знак $ сначала будет интерпретироваться как переменная в строке, а после первой обратной косой черты — как конец строки.

Поиск с помощью символов подстановки

Символ “точка” (.) в регулярном выражении является символом подстановки и соответствует произвольному символу. Например, следующее выражение найдет соответствие с любым словом из четырех символов с двумя буквами *o* посередине:

```
if (ereg("^.oo.$", $word)) ...
```

Символы ^ и $ указывают на начало и конец строки, а каждая точка может быть произвольным символом. Этот шаблон соответствует словам *book* или *tool*, но не *buck* или *stool*.

Символы подстановки

Регулярное выражение с одной точкой соответствует строке, содержащей хотя бы один символ. Нужно использовать символы ^ и $ чтобы наложить ограничение на длину строки.

Повторяющиеся шаблоны

Выше приводились шаблоны для отдельного символа или класса символов в строке. Показывались способы указать произвольный символ, чтобы задать широкий спектр шаблонов в регулярных выражениях. Есть также специальные символы, позволяющие указать, сколько раз отдельный шаблон повторяется в строке.

Символ (*) указывает на то, что последовательность встречается ноль или больше раз, а плюс (+) — как минимум один раз.

Два примера ниже с символами * и + очень похожи. Оба проверяют наличие буквенно-цифровых символов произвольной длины. Первое условие также соответствует пустой строке, потому что звездочка допускает ноль и больше совпадений [[:alnum::]]:

```
if (ereg("^[[:alnum:]]*$", $phrase)) ...
if (ereg("^[[:alnum:]]+$", $phrase)) ...
```

Нужно использовать круглые скобки, чтобы выделить последовательность искомых символов, которые нужно повторить. Например, условие ниже соответствует строке с чередующейся последовательностью букв и цифр:

```
if (ereg("^([[:alpha:]][[:digit:]])+$", $string)) ...
```

Символ плюс указывает на то, что последовательность буква/цифра должна встречаться не менее одного раза. Чтобы задать фиксированное количество повторений, нужно указать номер в фигурных скобках. Можно задать одно или два числа через запятую для указания диапазона, как показывается в примере ниже:

```
if (ereg("^([[:alpha:]][[:digit:]]){2,3}$", $string)) ...
```

Это выражение соответствует четырем или шести символам с чередующимися буквами и цифрами. Но единичная комбинация буквы и цифры или более трех повторений не соответствуют этой комбинации.

Знак вопроса (?) указывает на то, что комбинация должна встретиться не больше одного раза. Его поведение аналогично действию комбинации `{0,1}`.

Несколько практических примеров

В большинстве случаев регулярные выражения используются для того, чтобы проверить корректность данных, введенных пользователем. Ниже приводится несколько практических примеров использования регулярных выражений.

Почтовый индекс

Имеется почтовый индекс заказчика, который находится в переменной `$zip`, и нужно убедиться, что он введен в правильном формате. Почтовый индекс Соединенных Штатов состоит из пяти цифр и опционально сопровождается дефисом с четырьмя цифрами после него. Выражение ниже проверяет почтовый индекс на соответствие этому формату:

```
if (ereg("^[[:digit:]]{5}(-[[:digit:]]{4})?$", $zip)) ...
```

Первая часть регулярного выражения проверяет, что `$zip` начинается с пяти цифр. Вторая часть выражения заключена в круглые скобки, и после них ставится знак вопроса, который указывает на то, что выражение в скобках необязательное. Во второй части выполняется проверка на дефис и четыре цифры за ним.

Даже если второй части выражения нет, знак $ указывает на окончание строки. Таким образом, чтобы условие выполнилось, после указанных выражений не должно быть никаких символов. Поэтому такое выражение вернет истинное значение, если на вход подать почтовый индекс в виде 90210 или 90210-1234.

Телефонные номера

Предположим, нужно убедиться в том, что телефонный номер задан в формате (555)555-5555. В нем нет необязательных частей. Но круглые скобки имеют специальное значение для регулярных выражений. Поэтому их нужно разделить обратной косой чертой.

Следующее выражение проверяет соответствие этому формату:

```
if (ereg("^\([[:digit:]]{3}\)[[:digit:]]{3}-
[[:digit:]]{4}$",
                $telephone)) ...
```

Email адрес

Нужно учитывать много различных параметров при проверке электронных адресов. В самом простом случае почтовый адрес для домена .com выглядит как somename@somedomain.com.

Но может быть много вариаций. Например, домены верхнего уровня могут быть из двух символов, как в .ca, или даже из четырех, как .info. Некоторые специфические домены стран состоят из двух частей, как .co.uk или .com.au.

Как видно, регулярное выражение для электронного адреса должно быть довольно общим. Несмотря на это некоторые общие допущения в формате электронных адресов, позволяют создать правила, которые отсеивают много некорректных адресов.

Адрес состоит из двух важных частей, которые разделяются символом @. Символы слева от знака @, составляют название почтового ящика адресата. Они могут быть буквенно-цифровыми и включать несколько других символов.

Предположим, название ящика адресата может быть произвольной длины и состоять из любых символов, кроме символа @. Перед тем как перечислить приемлемые символы, нужно решить, например, включать ли одиночную ка-

вычку? Обычно достаточно убедиться в том, что в электронном адресе только один символ @ и все, что до него, является корректным названием ящика.

В регулярном выражении имя домена должно состоять из двух и более частей, разделенных точкой. Кроме того, можно задать, что последняя часть может быть не меньше двух и не больше четырех символов в длину. Это справедливо для всех доменов верхнего уровня, которые используются в данный момент.

В части домена набор допустимых символов более бедный, чем в названии почтового ящика. Допускается использование буквенно-цифровых символов в нижнем регистре и знака дефис.

Учитывая все приведенные выше условия, получаем следующее регулярное выражение для проверки электронного адреса:

```
if (ereg("^[^@]+@([a-z0-9\-]+\.)+[a-z]{2,4}$", $email)) ...
```

Это выражение разбивается на такие части: вначале любое количество символов, после которых следует символ @. А сразу за ним одна или более частей, состоящих из букв в нижнем регистре, цифр и дефиса. Каждая часть заканчивается точкой, а финальная может быть не меньше двух и не больше четырех букв в длину.

Когда остановиться?

Это выражение можно улучшить. Например, имя домена не может начинаться с дефиса и должно быть не больше 63 символов в длину. Но для проверки электронных адресов этого выражения вполне достаточно.

Разбивка строки на компоненты

Раньше круглые скобки использовались для группировки частей, чтобы выделить повторяющийся шаблон. Кроме того, круглые скобки используются для выделения части выражения. С помощью `ereg` шаблон разбивается на части, в зависимости от круглых скобок.

Третий необязательный параметр `ereg` сохраняет все части шаблона, которые выделяются круглыми скобками в регулярном выражении.

Задействуем выражение с электронным адресом. В примере ниже используется три набора круглых скобок: чтобы отделить имя почтового ящика, имя домена (отдельно от расширения) и расширение домена:

```
$email = "chris@lightwood.net";
if (ereg("^([^@]+)@([a-z\-]+\.)+([a-z]{2,4})$",
$email, $match)) {
    echo "Почтовый ящик: " . $match[1] . "<br>";
    echo "Имя домена: " . $match[2] . "<br>";
    echo "Тип домена: " . $match[3] . "<br>";
}
else {
    echo "Электронный адрес некорректен";
}
```

Если запустить сценарий в браузере, получим следующий вывод:

```
Почтовый ящик: chris
Имя домена: lightwood.
Тип домена: net
```

Отметим, что первый искомый шаблон (почтовый ящик) содержится в элементе `$match[1]`. Массив, как обычно, начинается с нулевого элемента. Дело в том, что элемент `$match[0]` содержит полное найденное выражение.

Поиск и замена

С помощью функции `ereg_replace` выполняется поиск и замена в строке с помощью регулярных выражений. На вход подается три аргумента: искомое регулярное выражение, строка, в которой производится замена, и строка-заменитель. На выходе получаем модифицированную строку.

str_replace

Для простых замен, которые не требуют регулярных выражений, лучше использовать функцию `str_replace` вместо `ereg_replace`. Функция `str_replace` намного эффективней, потому что PHP не пытается искать регулярное выражение.

Например, для того, чтобы удалить телефонный номер перед выводом на экран, можно использовать следующее:

```
echo ereg_replace( "\([[:digit:]]{3}\)[[:digit:]]{3}-
[[:digit:]]{4}$", "(XXX)XXX-XXXX", $string);
```

Аналогично тому, как `eregi` используется вместо `ereg` для независимого от регистра поиска, `eregi_replace` выполняет независимый от регистра поиск и замену.

Резюме

В этом уроке вы научились работать с простыми регулярными выражениями. Подробнее о них можно прочитать в книге Бена Форта *Освой самостоятельно регулярные выражения за 10 минут* (ИД "Вильямс", 2005 г.). В следующем уроке вы узнаете, как манипулировать значениями даты и времени в PHP.