Perl-совместимые регулярные выражения (PCRE)

При использовании любой PCRE функции необходимо заключать шаблон в разделители – любой символ не являющийся буквой, цифрой или обратной косой чертой. Допустимо использовать парные скобки.

Примеры:

```
/foo bar/
#^[^0-9]$#
{^\s+(\s+)?$}
```

Если необходимо использовать разделитель внутри шаблона, его нужно проэкранировать с помощью обратной косой черты.

```
/http:\/\//
```

Метасимволы

^ – декларирует начало строки
$ – конец строки
\b – начало или конец слова
\B – любая позиция кроме начала и конца слова
. – соответствует любому символу, кроме перевода строки
[ ] – начало и конец описания символьного класса (часть шаблона, заключенная в квадратные скобки)
| – начало ветки условного выбора
( ) – начало и конец подмаски
? – квантификатор, означающим отсутствие либо ровно 1 вхождение, также преобразует жадные квантификаторы в ленивые
* – квантификатор, означающий 0 или более вхождений
+ – квантификатор, означающий 1 или более вхождений
{ } – начало и конец количественного квантификатора, например: ”батарейка A{2,3}”

Внутри символьных классов используются следующие метасимволы:

\ – общий экранирующий символ
^ – означает отрицание класса
- – означает символьный интервал

Указание общего типа символов

\d – любая десятичная цифра
\D – любой символ, кроме десятичной цифры
\s – любой пробельный символ
\S – любой непробельный символ
\w – любая буква или цифра (слово)
\W – не буква и не цифра

Модификаторы шаблонов

/i – игнорирование регистра
/x – пропуск пробелов и комментариев
/m – многострочный поиск
/s – однострочный поиск

Разбор элементов адресной строки с параметрами

```
# Переменная, содержащая строку текста
$a="http://tut.rloc.ru/zform.php?obj=POIS&button1=send&button2=take";
# Строка – шаблон поиска
$tem="{//(.*)/(.*)\?(\w+=.*)&(\w+=.*)&(\w+=.*)}";
# Вызов функции поиска совпадений
preg_match($tem,$a,$match);
Вывод содержимого карманов совпадений
for ($i=0;$i < count($match);$i=$i+1){
echo '$match['."$i".'] = '.$match[$i]."<br/>";
}
```

Результат выполнения скрипта

```
$match[0] = //tut.rloc.ru/zform.php?obj=POIS&button1=send&button2=take
$match[1] = tut.rloc.ru
$match[2] = zform.php
$match[3] = obj=POIS
$match[4] = button1=send
$match[5] = button2=take
```

Жадные и ленивые квантификаторы (greedy and reluctant quantifiers)

```
$a="http://tut.rloc.ru/zform.php?obj=POIS&button1=send&button2=take";
$tem="{//(.*)/(.*)\?(\w+=.*)&}";
preg_match($tem,$a,$match);
```

Результат выполнения скрипта – жадное поглощение квантификатором *

```
$match[0] = //tut.rloc.ru/zform.php?obj=POIS&button1=send&
$match[1] = tut.rloc.ru
$match[2] = zform.php
$match[3] = obj=POIS&button1=send
```

Измененная строка-шаблон

```
$tem="{//(.*)/(.*)\?(\w+=.*?)&}";
```

Результат выполнения скрипта – ленивое поглощение квантификатором *

```
$match[0] = //tut.rloc.ru/zform.php?obj=POIS&button1=send&
$match[1] = tut.rloc.ru
$match[2] = zform.php
$match[3] = obj=POIS&button1=send
```

Функции PHP для работы с регулярными выражениями

preg_match — выполняет проверку на соответствие регулярному выражению

```
int preg_match ( string $pattern , string $subject [, array
&$matches [, int $flags = 0 [, int $offset = 0 ]]] )
```

Параметры

`pattern` – искомый шаблон, строка.

`subject` – входная строка.

`matches` – если указан дополнительный параметр matches, он будет заполнен результатами поиска. Элемент $matches[0] будет содержать часть строки, соответствующую вхождению всего шаблона, $matches[1] - часть строки, соответствующую первой подмаске, и так далее.

`flags` – может принимать значение `PREG_OFFSET_CAPTURE`
В случае, если этот флаг указан, для каждой найденной подстроки в массиве matches будет указана ее позиция в исходной строке.

`offset` – указывает начальную позицию для поиска (в байтах).

preg_match_all — выполняет глобальный поиск шаблона в строке и ищет все совпадения

```
int preg_match_all ( string $pattern , string $subject [,
array   [, int $flags = PREG_PATTERN_ORDER [, int $offset =
0 ]]] )
```

Результаты поиска помещаются в массив `$matches`, который оказывается двумерным: `$matches[B][N]`, где B – номер открывающей скобки (кармана), N – номер совпадения.

Значение параметра `$flags =PREG_PATTERN_ORDER` предполагает сортировку по номеру совпадения.

preg_replace — выполняет поиск и замену по регулярному выражению

```
mixed preg_replace ( mixed $pattern , mixed $replacement ,
mixed $subject [, int $limit = -1 [, int &$count ]] )
```

Параметры

`pattern` – искомый шаблон, строка.

`replacement` – строка или массив строк для замены. Если этот параметр является строкой, а `pattern` является массивом, все шаблоны будут заменены этой строкой. Если и `pattern` и `replacement` являются массивами, каждый элемент pattern будет заменен соответствующим элементом из `replacement`. Если массив `replacement` содержит меньше элементов, чем массив `pattern`, то все лишние шаблоны из `pattern` будут заменены пустыми строками.

`subject` – строка или массив строк для замены.

`limit` – максимально возможное количество замен каждого шаблона для каждой строки subject. По умолчанию равно -1 (без ограничений).

`count` – если указана, то эта переменная будет заполнена количеством произведенных замен.

Пример

```
$string = "Хорошие студенты аккуратно посещают занятия";
$patterns[0] = "/Хорошие/";
$patterns[1] = "/аккуратно/";
$patterns[2] = "/посещают/";
$replacements[2] = "Плохие";
$replacements[1] = "регулярно";
$replacements[0] = "пропускают";
echo preg_replace($patterns, $replacements, $string);
```

Результат работы скрипта

```
"Плохие студенты регулярно пропускают занятия"
```

Конструкции языка PHP

Условные операторы:

```
if (логическое_выражение_1)
оператор_1;
elseif (логическое_выражение_2)
оператор_2;
else
оператор_3;
```

Циклы:

```
while
do-while
for
foreach
break
continue
```

Конструкция выбора: `switch-case`

```
switch(выражение) {
    case значение1: команды1; [break;]
    case значение2: команды2; [break;]
    . . .
    case значениеN: командыN; [break;]
    [default: команды_по_умолчанию; [break]]
}
```

Конструкция возврата значений: `return`

Конструкции включений:

`require(имя_файла)` – включает файл в сценарий до его выполнения

`include(имя_файла)` – включает файл в сценарий во время его выполнения

`require_once()` – гарантирует однократное включение файла в сценарий

`include_once()` – гарантирует однократное включение файла в сценарий