Операция замены в Perl (perl regex)

[<< Предыдущая] [ИНДЕКС] [Исправить] [src / Печать] [Следующая >>]

Ключевые слова: perl, regex, (найти похожие документы)

From: Alex <aleksander@jabber.finet.ru.>
Date: Sun, 18 Sep 2007 17:02:14 +0000 (UTC)
Subject: Операция замены в Perl

Оригинал: http://ylsoftware.com/?action=news&na=viewfull&news=227
Автор оригинального текст (на английском языке) - Randal L. Schwartz
Оригинальный текст

Если вы использовали Perl больше 15 минут, вы без сомнения видели (и возможно использовали) очень полезную операцию замены, которая обычно выглядит как s/old/new/. Давайте поговорим о вещах, которые вы возможно уже знаете, и о некоторых вещах, которые вы возможно еще не знаете, используемых при операциях замены.

Наиболее важная вещь, о которой следует сказать - это то, что операция замены по умолчанию действует для переменной "$_":

```
$_ = "hello";
s/ell/ipp/; # $_ теперь "hippo"
```

Левая часть замены - регулярное выражение, значит к нему применимы все правила для регулярных выражений:

```
$_ = "hello";
s/e.*l/ipp/; # $_ теперь "hippo"
```

Здесь ".*" заменяет любое количество любых символов. Если использовать выражение ".*?", то будет изменена только часть текста до первого вхождения символа (в данном случае символа "l"). Например:

```
$_ = "hello";
s/e.*?l/ipp/; # $_ теперь "hipplo"
```

Кроме того, мы можем использовать вместо переменной "$_" другие, для этого надо использовать конструкцию "=~". В отличие от операции сравнения, мы должны определить переменную типа lvalue (как имя переменной), а не rvalue (результат выражения).

```
my $text = "hello";
$text =~ s/ell/ipp/; # $text теперь "hippo"
```

Я иногда узнаю, что люди путаются в возвращаемых значениях операции замены. Итак, если в переменной $text сейчас содержится новое значение, будет ли оно тем же, что и в более широком контексте?

```
my $text = "hello";
my $result = ($text =~ s/ell/ipp/);
```

Ответ нет. Несмотря на то, что операция замены действительно меняет переменную $text, она возвращает значение true/false, а не результат замены. В данном случае $result равен true. Это свойство используется в операция с условиями:

```
if (s/foo/bar/) { # если foo было найдено, то оно стало bar...
  ... выполняем какие-гибудь действия ...
} else {
  ... мы не нашли foo, и $_ не изменилось ...
}
```

Замена выполняется в первом возможном месте:

```
$_ = "hello";
s/l/p/; # $_ теперь "heplo";
s/l/p/; # $_ теперь "heppo";
```

Повторение операции для всех неперекрывающихся совпадений произойдет, если мы добавим суффикс "g":

```
$_ = "hillo";
s/l/p/g; # $_ is now "hippo";
```

Важно слово "неперекрывающихся". Perl ищет каждое новое совпадение после конца предыдущего совпадения. Таким образом, результат замены,

подобной этой, вначале может удивить:

```
$_ = "aaa,bbb,ccc,ddd,eee,fff,ggg";
s/,.*?,/,XXX,/g; # заменяем все поля на XXX
```

Когда мы проверим результат, мы увидим:

```
aaa,XXX,ccc,XXX,eee,XXX,ggg
```

Почему же не произошло полной замены? В первом совпадении мы получили ,bbb, и заменили его на ,XXX,. Но после этого ,ccc, не будет считаться как совпадение из-за того что является перекрывающимся (то есть является концом предыдущей замены)!

Мы можем исправить это если добавим предпросмотр замыкающей запятой:

```
$_ = "aaa,bbb,ccc,ddd,eee,fff,ggg";
s/,.*?(?=,)/,XXX/g; # заменяем все поля на XXX
```

Сейчас замыкающая запятая не является частью строки поиска, поэтому она не пропускается при поиске. Заметьте, что я также изменил строку для замены, теперь она не добавляет запятую в конце. Теперь результат будет лучше:

```
aaa,XXX,XXX,XXX,XXX,XXX,ggg
```

Все еще не заменено начало. Это понятно, так как мы требуем запятую перед буквами. И не заменен конец, так как мы требуем замыкающую запятую, даже если она не является частью совпадения. Мы можем решить обе эти проблемы немного исправив текст для замены:

```
$_ = "aaa,bbb,ccc,ddd,eee,fff,ggg";
s/(^|(?<=,)).*?((?=,)|$)/XXX/g; # заменяем все поля на XXX
```

Это работает, но выглядит не красиво. Мы можем переписать эту строку следующим образом:

```
s/
  (
    ^        # начало строки
    |        # или
    (?<=,)   # запятая слева
```

```
  )
  .*?        # сколько угодно знаков
  (
    (?=,)    # запятая справа
    |        # или
    $        # конец строки
  )
/XXX/gx;
```

Так гораздо приятнее читать.

Кроме того, мы можем использовать альтернативные разделители для левой и правой части выражения замены:

```
$_ = "hello";

s%ell%ipp%; # $_ теперь "hippo"
```

Правила немного сложнее, но это работает точно как хотел Larry Wall. Если разделитель не является специальным символом, который начинает пару, то мы используем его второй раз к обоим частям и для выражения замены и для завершения операции замены, как это показано в примере ниже.

Однако, если мы используем начальный символ парного набора (круглые скобки, фигурные скобки, квадратные скобки или знаки "больше" и "меньше"), мы завершаем выражением соответствующим знаком. Затем мы приступаем к следующему разделителю, используя те же правила. Например, все эти строки делают одно и тоже:

```
s/ell/ipp/;
s%ell%ipp%;
s;ell;ipp;; # не делайте так!
s#ell#ipp#; # одна из моих любимых
s[ell]#ipp#; [] для строки поиска, # для замены
s[ell][ipp]; [] и для строки поиска и для замены
s<ell><ipp>; <> и для строки поиска и для замен
s{ell}(ipp); {} для строки поиска, () для замены
```

Не важно, что закрывающий разделитель используется и для строки поиска и для замены, мы можем добавить этот знак если используем обратную косую черту (backslash):

```
$_ = "hello";
```

```
s/ell/i\/n/; # $_ теперь "hi/no";
s/\/no/res/; # $_ теперь "hires";
```

Вместо множества косых черт, можно использовать другие разделители:

```
$_ = "hello";

s%ell%i/n%; # $_ is now "hi/no";
s%/no%res%; # $_ is now "hires";
```

Удобно то, что если используются парные символы, то выражение можно писать вообще без косых черт:

```
$_ = "aaa,bbb,ccc,ddd,eee,fff,ggg";
s((^|(?<=,)).*?((?=,)|$))(XXX)g; # Заменяем все поля на XXX
```

Не забудьте, что все закрывающие значки составляют пару с откравающими, так тчо располагайте их правильно.

Правая часть операции замены обычно упрощается с помощью переменной, которая является строкой, заключенной в двойные кавычки:

```
$replacement = "ipp";
$_ = "hello";
s/ell/$replacement/; # $_ теперь "hippo"
```

Левая часть выражения тоже упрощается подобной заменой:

```
$pattern = "ell";
$replacement = "ipp";
$_ = "hello";
s/$pattern/$replacement/; # $_ теперь "hippo"
```

Использование таких замен заставляет Perl вычислять регулярные выражения во время выполнения программы. Если это происходит в цикле - это может привести к некоторому замедлению работы программы. Мы можем указать Perl, что это действительно регулярное выражение, используя символ регулярного выражения:

```
$pattern = qr/ell/;
$replacement = "ipp";
$_ = "hello";
```

```
s/$pattern/$replacement/; # $_ теперь "hippo"
```

Операция qr создает объект Regexp, который подставляется в выражение и имеет наибольшую скорость выполнения.

Надеюсь вам понравилась эта статья об операции замены, однако она не может служить заменой справочных страниц, таких как perlre, perlretut, perlrequick, и perlreref. Смотрите их для более полной информации.

[<< Предыдущая] [ИНДЕКС] [Исправить] [src / Печать] [Следующая >>]

Обсуждение

[RSS]

- 1, nrg (ok), 18:13, 12/01/2008 [ответить] +/-

Блин, статья супер, но вот когда я заставлю себя выучить look ahead модификаторы в регулярных выражениях - это большой вопрос :(

- 2, Максим (??), 01:59, 09/05/2020 [ответить] +/-

Супер подробная статья, помогла. Интересовало какие символы лучше использовать для разделения поиска и замены в URL. Использовал %, но иногда в URL попадается. Спасибо.

Добавить комментарий

Имя:

E-Mail:

Заголовок: Операция замены в Perl (perl re

Текст:

Отправить