Регулярные выражения

"Регулярные выражения также можно сравнить с иностранным языком — когда вы начинаете изучать язык, он перестает казаться белибердой". (Джеффри Фридл)

- Регулярные выражения — это отдельный язык для работы с текстовой информацией. Регулярные выражения использовались в Unix задолго до создания Perl. Но широкое распространение Perl в свое время сделало регулярные выражения популярными на разных платформах. А в ходе развития языка Perl была отточена система обозначений для регулярных выражений, ставшая фактическим стандартом. В Perl механизмы работы с регулярными выражениями встроены в ядро языка, поэтому применять их естественно, легко и удобно. А благодаря эффективной реализации "движка" регулярных выражений, в Perl они обрабатываются чрезвычайно быстро.

Регулярные выражения в Perl несколькими способами:

- для поиска в тексте строк по определенному образцу;
- для разделения текста на части по указанному набору разделителей;
- для извлечения из строки подстрок, соответствующих заданному шаблону;
- для замены в тексте найденных соответствий на новые значения.

Поиск соответствий

Чаще всего регулярные выражения используются в операции сопоставления, которая проверяет, соответствует ли текст указанному образцу.

- ***Образец*** (pattern) — это символьная последовательность для сопоставления, записанная в специальной нотации. Простейший образец — это строковый литерал, представляющий собой последовательность символов, которая будет отыскиваться в тексте.

Для указания, к какой строке применить операцию сопоставления, используется операция привязки =~ к строке:

```
'В строке образец есть' =~ /образец/; # образец найден
```

Обычно поиск образца выполняется с учетом регистра, но можно игнорировать регистр при сопоставлении строки с образцом, если в операции сопоставления задать модификатор /i (ignore case):

```
use locale;
'В строке образец есть' =~ /Образец/; # образец НЕ найден!
'В строке образец есть' =~ /Образец/i; # образец найден
```

- В скалярном контексте операция сопоставления возвращает '1', если образец в строке найден, и пустую строку "", если соответствие образцу не найдено.

```
$text = 'Черный кот в темной комнате'; # ищем в этом тексте
$found = $text =~ /кот/; # в $found будет '1'
print 'Кошки нет!' unless $text =~ /кошка/; # вернет ''
# операция отрицательной привязки
```

```
print 'Кошки нет!' if $text !~ /кошка/;# вернет '1'
```

Если операция привязки к строке не используется, образец отыскивается в переменной по умолчанию $_. Выражение перед поиском интерполируется, поэтому

- весь образец поиска или его часть может содержаться в переменной.

```
$_ = 'Счастье - это когда тебя понимают.'; # переменная поиска
$pattern = 'Счастье'; # образец для сопоставления
print "$pattern найдено!" if /$pattern/;
```

Для успешного сопоставления строки образцу достаточно найти в строке первое совпадение. В этом примере образец совпадет с началом подстроки 'которого':

```
$text = 'У которого из котов зеленые глаза?'; # ищем здесь
$any = $text =~ /кот/; # образец совпал с началом 'которого'
```

В операции сопоставления программист может задавать ***ограничители для образца***:

- перед ограничителями указывается буква m// (Операцию сопоставления часто именно так и называют: операция m//.)
- В качестве ограничителей могут выступать различного вида скобки или парные небуквенные символы, например:

m($pattern) m{$pattern} m[$pattern] m<$pattern> m|$pattern| m!$pattern! m"$pattern" m#$pattern#

Задать собственные ограничители бывает особенно полезно, когда в шаблон поиска входит наклонная черта. Из двух приведенных вариантов второй смотрится гораздо понятнее:

/\/usr\/bin\/perl/

m{/usr/bin/perl}

Недаром обилие левых и правых наклонных черт в первом варианте называют "ученическим синдромом зубочисток" (LTS — Learning Toothpick Syndrome). *В приводимых до сих пор примерах операцию сопоставления с литералом в качестве образца вполне можно заменить вызовом функции index().*

Метасимволы (metacharacter)

Очень часто требуется искать в тексте не конкретные строки, а символьные последовательности, определенные приблизительно: "число в скобках", "четвертое слово с начала строки", "список из пар имя = значение, разделенных запятыми" и тому подобное. В таких случаях в строке поиска задается шаблон, который описывает такую последовательность.

- ***Шаблон*** — это образец, в котором, помимо литеральных значений, содержатся метасимволы.

Метасимволы — это знаки, имеющие специальное значение при записи образцов. Следующие метасимволы применяются при записи регулярных выражений:

{} [] () ^ $. | * + ? \

При необходимости включить в образец поиска один из этих знаков не как метасимвол, а как обыкновенный символ, нужно отменить его особое значение ("экранировать"), поставив перед ним обратную косую черту (backslash):

$text =~ m"\."; #содержится ли в тексте точка?

Как метасимвол точка обозначает в регулярном выражении один любой символ, кроме знака перевода новой строки (\n). Например, для поиска похожих слов можно составить такой шаблон:

само.а.

соответствуют: 'самовар', 'самокат', 'самосад'... # НЕ соответствуют: 'самолюб', 'самогон', 'самоход'...

В регулярном выражении можно задать несколько вариантов образца, любой из которых будет считаться соответствием строки образцу. Варианты образца — это набор возможных альтернатив, разделенных знаком "вертикальная черта" ('|'), который называется "метасимвол альтернатив" (alternation metacharacter). Поиск считается успешным, если найдено соответствие любой из альтернатив:

```
$text = 'Черная кошка в темной комнате'; # будем искать здесь
print "Нашли кошку!" if $text =~ /кот|кошка|котенок/;
```

Сравнение текста с вариантами образца выполняется слева направо, поэтому, если начало альтернатив совпадает, более длинную альтернативу нужно помещать в начало списка вариантов. Иначе всегда будет найдена более короткая. Значит шаблон в предыдущем примере правильнее записать в виде /котенок|кот|кошка/, чтобы в первую очередь поискать котенка, а затем - кота:

```
$text = 'Черный котенок в темной комнате'; # ищем здесь
print "Нашли котенка!" if $text =~ /кот.нок|кот|кошка/;
```

Чтобы сделать образец более универсальным, в первой альтернативе литерал заменен на шаблон с метасимволом "точка", чтобы находились соответствия слову "котенок" в любом написании — через "е" и через "ё".

- Часто применение регулярного выражения с альтернативами выглядит гораздо изящнее, чем длинное условное выражение:

```
return if $command =~ /exit|quit|stop|bye/i;
```

Если в образце после выбора из нескольких альтернатив применяются другие шаблоны или литералы, то конструкцию выбора нужно заключить в круглые группирующие скобки.

```
$lotr =~ / (Sam|Tom) Smit/; # один из Смитов
```

Если нужно найти одно из слов *frog, bog, log, flog* или *clog:*

```
$lotr =~ /(fr|b|l|fl|cl)og/;
```

Классы символов

С помощью метасимволов [] можно обозначить в шаблоне один символ из заданного набора. Для этого нужно определить класс символов, указав в квадратных скобках набор символов, включаемых в класс. Классы символов похожи на шаблон с вариантами, в

котором альтернативами могут быть только отдельные символы. Ради примера запишем шаблон для слов, отличающихся первой буквой из указанного набора:

```
[вклрт]от                    # соответствуют: 'вот','кот','лот','рот','тот'
```

Пример шаблона с несколькими классами символов, каждый из которых представляет одну букву в последовательности из четырех символов:

```
[мс][ул][хо][ан]

# соответствуют: 'муха', 'слон' # а также: 'суоа', 'млхн', 'слоа' и так далее
```

В классе символов вместо перечисления можно указывать ***диапазон*** от начального до конечного символа, разделенных минусом:

[0-9] вместо [0123456789]

[A-Z] вместо [ABCDEFGHIJKLMNOPQRSTUVWXYZ]

Можно указывать несколько диапазонов в одном классе, запишем шаблон для шестнадцатеричной цифры:

```
[0-9a-fA-F]                  # соответствуют: '5', 'b', 'D' и так далее
```

Чтобы включить в символьный класс знак '-', нужно поместить его в начале или в конце перечисленных в классе символов или экранировать обратной чертой.

Помещенные в символьный класс [], все метасимволы (кроме ']') рассматриваются как обычные символы. Поэтому так могут выглядеть шаблоны для поиска знака препинания или одной из скобок:

```
[-.,;:!?]                    # знаки препинания
[()[\]{}]                    # скобки: \] представляет скобку ']'
```

Иногда требуется выразить понятие "все, кроме указанных символов": для этого в описании класса символов *сразу после открывающей квадратной скобки* ставится метасимвол отрицания ('^').

Например, так можно записать шаблоны для "любого символа, кроме знаков препинания" или "любого нецифрового символа":

```
[^-.,;:!?]                   # все, кроме этих знаков препинания
[^0-9]                       # не цифры
```

Чтобы включить в символьный класс символ '^', нужно поставить его не первым в списке символов или отменить его специальное значение с помощью символа '\':

[*^] или так: [\^]

Метапоследовательности

Для сокращенной записи классов символов в регулярных выражениях предусмотрены специальные обозначения, состоящие из латинской буквы с обратной косой чертой перед ней. Вот они:

```
\d      - любая десятичная цифра, то есть [0-9]
\D      - любой символ, кроме цифры: [^0-9] или [^\d]
\w      - то же, что и  [a-zA-Z0-9_]
\W      - противоположность символа \w, то есть [^\w]
```

\s - пробельный символ любого вида: пробел, \t (табулятор), \n (конец строки), \r (возврат каретки) или \f (прогон страницы)

\S - любой не пробельный символ, то есть [^\s]

С помощью этих метасимволов можно составлять гораздо более интересные образцы. Например, проверим, ***содержится ли в тексте число из четырех цифр, окруженное любыми пробельными символами***:

text: "Альбом 'Dire Straits'\tГод 1978\tВремя 41:21".

```
\s\d\d\d\s                    # найдет ' 1978\t'
```

Квантификаторы

Записывать ***несколько метасимволов подряд для указания в шаблоне последовательности из однотипных символов*** утомительно и неудобно, да и ошибиться при этом легко. Облегчить жизнь составителям регулярных выражений помогают квантификаторы.

Квантификатор (quantifier) — это обозначение ***числа повторений предыдущего шаблона*** при поиске соответствия.

Количество повторений может задаваться одним или парой десятичных чисел в фигурных скобках:

{n} повторяется точно n раз

{n,} повторяется n и более раз

{n,m} повторяется от n до m раз включительно

Квантификатор, также иногда называемый множителем, указывается сразу после конструкции в шаблоне, которую нужно повторить несколько раз, например:

```
\d{5}          # ровно пять цифр, то есть: \d\d\d\d\d
\s{1,}         # один и более пробельных символов
[A-Z]{1,8}     # от 1 до 8 заглавных латинских букв
```

Сокращения метасимволов

Для наиболее часто встречающихся квантификаторов предусмотрены удобные односимвольные сокращения:

* повторяется 0 или более раз: то же, что {0,}

? повторяется 0 или 1 раз: то же, что {0,1}

\+ повторяется как минимум 1 раз: то же, что {1,}

Составим регулярное выражение с использованием односимвольных квантификаторов, чтобы найти в тексте "***идентификатор, перед которым могут стоять пробельные символы и за которым стоит хотя бы один из перечисленных знаков препинания***":

```
\s*\w+[-.,;?]+     # соответствует, например: ' count--;' ‘ word...’
car*t              # соответствуют: ‘carted’, ‘cat’, ‘carrrt’
```

Группировка шаблонов

Если квантификатор нужно применить к нескольким шаблонам, то нужно сгруппировать шаблоны, заключив их в круглые скобки. Составим регулярное выражение для поиска IP-адреса, которое ***находит число, состоящее от одной до трех цифр, за которым стоит точка (\.), причем эта последовательность повторяется ровно три раза*** ({3}), после 4-й группы цифр точки нет:

```
text: 'address=208.201.239.36'

(\d{1,3}\.){3}\d{1,3}                    # соответствие: '208.201.239.36'
```

- "При составлении шаблона главное, чтобы регулярное выражение соответствовало тому, что нужно, и не соответствовало тому, что не нужно".

Правила использования регулярных выражений

• Обычно поиск по шаблону в строке ведется слева направо.

• Поиск соответствия шаблону выполняется в строке слева направо. Таким образом, первым будет найден текст, расположенный ближе к началу строки (*принцип торопливости*). Однако это совсем не означает, что следующие соответствия шаблону найдены не будут. Хотя существуют и исключения...

Пример:

text = 'У которого из котов зеленые глаза?';

кот образец совпал с 'которого'

• Прежде всего программа пытается найти самую длинную строку, соответствующую шаблону. Обнаружив первое совпадение, Perl просматривает всю строку до конца в поиске более длинной фразы, соответствующей заданному шаблону. Для описания этого свойства регулярных выражений придуман специальный термин — *"жадность* "регулярных выражений.

Пример:

будем искать "***более одного символа, за которыми идет буква 'й' и пробел***":

```
my $text = 'Какой хороший компакт-диск!';

$text =~ /.+й\s/; # жадный квантификатор

# найдено соответствие: 'Какой хороший '
```

Это произошло потому, что по умолчанию квантификаторы подразумевают максимальную последовательность символов, соответствующих указанному шаблону. Такое поведение квантификаторов называется "*жадным*" (greedy quantifier). Чтобы заставить квантификатор вести себя не "жадно", а "*лениво*" (lazy quantifier), нужно поставить сразу после него символ '?'. Тогда ***квантификатор будет описывать минимальную последовательность символов, соответствующих образцу***. Исправленный с учетом этого образец найдет то, что нужно:

```
$text =~ /.+?й\s/; # ленивый квантификатор

# найдено соответствие: 'Какой '
```

Таким же образом можно ограничивать "жадность" и других квантификаторов, заставляя их прекращать поиск как можно раньше, что обычно и требуется в большинстве ситуаций.

Символы привязки (утверждения)

Обозначение	Описание	Шаблон	Соответствие
^	Соответствие должно находиться в начале строки	^\d{2}	«32» в «32,43,54»
$	Соответствие должно находиться в конце строки или до символа \n при многострочном поиске	\d{2}$	«54» в «32,43,54»
\b	Соответствие должно находиться на границе алфавитно-цифрового символа (\w) и не алфавитно-цифрового (\W)	\b\d{2}	«32», «54» в «32 a43 54»
\B	Соответствие не должно находиться на границе	\B\d{2}	«43» в «32 a43 54»
\G	Соответствие должно находиться на позиции конца предыдущего соответствия	\G\d	«3», «2», «4» в «324.758»

Часто нам бывает небезразлично, ***в каком месте содержимое строки совпадет с шаблоном***. Мы бы хотели уточнить: "в начале строки", "в конце слова" и так далее.

Для того чтобы более точно задать положение в тексте, где должно быть найдено соответствие, в регулярных выражениях можно указывать так называемые ***утверждения***.

- Утверждение (assertion) не соответствует какому-либо символу, а ***совпадает с определенной позицией в тексте***. Поэтому их можно воспринимать как ***мнимые символы нулевого размера***.

Чаще всего используются следующие утверждения:

^ позиция в начале строки

$ позиция в конце строки (или перед \n в конце строки)

\b граница слова: позиция между \w и \W или \W и \w

\B любая позиция, кроме границы слова \b

Вот пример шаблонов поиска, где уточняется, что нужно проверить наличие числа в определенном месте строки:

```
$log = '20060326 05:55:25 194.67.18.73 ... 200 797';
print "Число в начале\n" if $log =~ /^\d+/;
print "Число в конце\n" if $log =~ /\d+$/;
```

Утверждение, которое используется для фиксирования части образца относительно положения в строке, иногда называется якорем (anchor). Якори применяются, чтобы указать, в каком именно месте строки нужно искать соответствие образцу.

Граница слова

\b граница слова (между \w и \W или \W и \w)

ПРИМЕРЫ

Извлечение всех чисел из строки

\d+\.?\d*|\.\d+

Поиск всех слов записанных символами верхнего регистра

\b[^\Wa-z0-9_]+\b

Поиск всех слов, начинающихся с буквы верхнего регистра

\b[^\Wa-z0-9_][^\WA-Z0-9_]*\b

Найдет fact, first, fought,... — слова, начинающиеся на f, заканчивающиеся на t

\bf[a-z]+?t\b