Шпаргалка по регулярным выражениям

exlab.net/tools/sheets/regexp.html

Шпаргалка представляет собой общее руководство по шаблонам регулярных выражений без учета специфики какого-либо языка. Она представлена в виде таблицы, помещающейся на одном печатном листе формата A4. Создана под лицензией Creative Commons на базе шпаргалки, автором которой является Dave Child (подробнее).

Помните, что различные языки программирования поддерживают регулярные выражения в разной степени, поэтому вы можете столкнуться с ситуацией, когда некоторые из указанных возможностей не будут работать. Для тех же, кто только знакомится с регулярными выражениями, предлагается этот перевод авторских комментариев к шпаргалке. Он познакомит вас с некоторыми техниками, применяемыми при построении шаблонов регулярных выражений.

Якоря

Якоря в регулярных выражениях указывают на начало или конец чего-либо. Например, строки или слова. Они представлены определенными символами. К примеру, шаблон, соответствующий строке, начинающейся с цифры, должен иметь следующий вид:

```
^[0-9]+
```

Здесь символ `^` обозначает начало строки. Без него шаблон соответствовал бы любой строке, содержащей цифру.

Символьные классы

Символьные классы в регулярных выражениях соответствуют сразу некоторому набору символов. Например, `\d` соответствует любой цифре от 0 до 9 включительно, `\w` соответствует буквам и цифрам, а `\W` — всем символам, кроме букв и цифр. Шаблон, идентифицирующий буквы, цифры и пробел, выглядит так:

```
\w\s
```

POSIX

POSIX — это относительно новое дополнение семейства регулярных выражений. Идея, как и в случае с символьными классами, заключается в использовании сокращений, представляющих некоторую группу символов.

Утверждения

Поначалу практически у всех возникают трудности с пониманием утверждений, однако познакомившись с ними ближе, вы будете использовать их довольно часто. Утверждения предоставляют способ сказать: «я хочу найти в этом документе каждое слово, включающее букву “q”, за которой не следует “werty”».

```
[^\s]*q(?!werty)[^\s]*
```

Приведенный выше код начинается с поиска любых символов, кроме пробела (`[^\s]*`), за которыми следует `q` . Затем парсер достигает «смотрящего вперед» утверждения. Это автоматически делает предшествующий элемент (символ, группу или символьный класс) условным — он будет соответствовать шаблону, только если утверждение верно. В нашем случае, утверждение является отрицательным (`?!`), т. е. оно будет верным, если то, что в нем ищется, не будет найдено.

Итак, парсер проверяет несколько следующих символов по предложенному шаблону (`werty`). Если они найдены, то утверждение ложно, а значит символ `q` будет «проигнорирован», т. е. не будет соответствовать шаблону. Если же `werty` не найдено, то утверждение верно, и с `q` все в порядке. Затем продолжается поиск любых символов, кроме пробела (`[^\s]*`).

Образцы шаблонов

В этой группе представлены образцы шаблонов. С их помощью вы можете увидеть, как можно использовать регулярные выражения в ежедневной практике. Однако заметьте, что они не обязательно будут работать в любом языке программирования, поскольку каждый из них обладает индивидуальными особенностями и различным уровнем поддержки регулярных выражений.

Кванторы

Кванторы позволяют определить часть шаблона, которая должна повторяться несколько раз подряд. Например, если вы хотите выяснить, содержит ли документ строку из от 10 до 20 (включительно) букв «a», то можно использовать этот шаблон:

```
a{10,20}
```

По умолчанию кванторы — «жадные». Поэтому квантор `+` , означающий «один или больше раз», будет соответствовать максимально возможному значению. Иногда это вызывает проблемы, и тогда вы можете сказать квантору перестать быть жадным (стать «ленивым»), используя специальный модификатор. Посмотрите на этот код:

```
".*"
```

Этот шаблон соответствует тексту, заключенному в двойные кавычки. Однако, ваша исходная строка может быть вроде этой:

```
<a href="helloworld.htm" title="Привет, Мир">Привет, Мир</a>
```

Приведенный выше шаблон найдет в этой строке вот такую подстроку:

```
"helloworld.htm" title="Привет, Мир"
```

Он оказался слишком жадным, захватив наибольший кусок текста, который смог.

```
".*?"
```

Этот шаблон также соответствует любым символам, заключенным в двойные кавычки. Но ленивая версия (обратите внимание на модификатор `?`) ищет наименьшее из возможных вхождений, и поэтому найдет каждую подстроку в двойных кавычках по отдельности:

```
"helloworld.htm"
"Привет, Мир"
```

Специальные символы

Регулярные выражения используют некоторые символы для обозначения различных частей шаблона. Однако, возникает проблема, если вам нужно найти один из таких символов в строке, как обычный символ. Точка, к примеру, в регулярном выражении обозначает «любой символ, кроме переноса строки». Если вам нужно найти точку в строке, вы не можете просто использовать «`.`» в качестве шаблона — это приведет к нахождению практически всего. Итак, вам необходимо сообщить парсеру, что эта точка должна считаться обычной точкой, а не «любым символом». Это делается с помощью знака экранирования.

Знак экранирования, предшествующий символу вроде точки, заставляет парсер игнорировать его функцию и считать обычным символом. Есть несколько символов, требующих такого экранирования в большинстве шаблонов и языков. Вы можете найти их в правом нижнем углу шпаргалки («Мета-символы»).

Шаблон для нахождения точки таков:

```
\.
```

Другие специальные символы в регулярных выражениях соответствуют необычным элементам в тексте. Переносы строки и табуляции, к примеру, могут быть набраны с клавиатуры, но вероятно собьют с толку языки программирования. Знак экранирования используется здесь для того, чтобы сообщить парсеру о необходимости считать следующий символ специальным, а не обычной буквой или цифрой.

Подстановка строк

Подстановка строк подробно описана в следующем параграфе «Группы и диапазоны», однако здесь следует упомянуть о существовании «пассивных» групп. Это группы, игнорируемые при подстановке, что очень полезно, если вы хотите использовать в шаблоне условие «или», но не хотите, чтобы эта группа принимала участие в подстановке.

Группы и диапазоны

Группы и диапазоны очень-очень полезны. Вероятно, проще будет начать с диапазонов. Они позволяют указать набор подходящих символов. Например, чтобы проверить, содержит ли строка шестнадцатеричные цифры (от 0 до 9 и от A до F), следует использовать такой диапазон:

```
[A-Fa-f0-9]
```

Чтобы проверить обратное, используйте отрицательный диапазон, который в нашем случае подходит под любой символ, кроме цифр от 0 до 9 и букв от A до F:

```
[^A-Fa-f0-9]
```

Группы наиболее часто применяются, когда в шаблоне необходимо условие «или»; когда нужно сослаться на часть шаблона из другой его части; а также при подстановке строк.

Использовать «или» очень просто: следующий шаблон ищет «ab» или «bc»:

```
(ab|bc)
```

Если в регулярном выражении необходимо сослаться на какую-то из предшествующих групп, следует использовать `\n`, где вместо `n` подставить номер нужной группы. Вам может понадобиться шаблон, соответствующий буквам «aaa» или «bbb», за которыми следует число, а затем те же три буквы. Такой шаблон реализуется с помощью групп:

```
(aaa|bbb)[0-9]+\1
```

Первая часть шаблона ищет «aaa» или «bbb», объединяя найденные буквы в группу. За этим следует поиск одной или более цифр (`[0-9]+`), и наконец `\1`. Последняя часть шаблона ссылается на первую группу и ищет то же самое. Она ищет совпадение с текстом, уже найденным первой частью шаблона, а не соответствующее ему. Таким образом, «aaa123bbb» не будет удовлетворять вышеприведенному шаблону, так как `\1` будет искать «aaa» после числа.

Одним из наиболее полезных инструментов в регулярных выражениях является подстановка строк. При замене текста можно сослаться на найденную группу, используя `$n`. Скажем, вы хотите выделить в тексте все слова «wish» жирным начертанием. Для этого вам следует использовать функцию замены по регулярному выражению, которая может выглядеть так:

```
replace(pattern, replacement, subject)
```

Первым параметром будет примерно такой шаблон (возможно вам понадобятся несколько дополнительных символов для этой конкретной функции):

```
([^A-Za-z0-9])(wish)([^A-Za-z0-9])
```

Он найдет любые вхождения слова «wish» вместе с предыдущим и следующим символами, если только это не буквы или цифры. Тогда ваша подстановка может быть такой:

```
$1<b>$2</b>$3
```

Ею будет заменена вся найденная по шаблону строка. Мы начинаем замену с первого найденного символа (который не буква и не цифра), отмечая его `$1`. Без этого мы бы просто удалили этот символ из текста. То же касается конца подстановки (`$3`). В середину мы добавили HTML тег для жирного начертания (разумеется, вместо него вы можете использовать CSS или `<strong>`), выделив им вторую группу, найденную по шаблону (`$2`).

Модификаторы шаблонов

Модификаторы шаблонов используются в нескольких языках, в частности, в Perl. Они позволяют изменить работу парсера. Например, модификатор `i` заставляет парсер игнорировать регистры.

Регулярные выражения в Perl обрамляются одним и тем же символом в начале и в конце. Это может быть любой символ (чаще используется «/»), и выглядит все таким образом:

```
/pattern/
```

Модификаторы добавляются в конец этой строки, вот так:

```
/pattern/i
```

Мета-символы

Наконец, последняя часть таблицы содержит мета-символы. Это символы, имеющие специальное значение в регулярных выражениях. Так что если вы хотите использовать один из них как обычный символ, то его необходимо экранировать. Для проверки наличия скобки в тексте, используется такой шаблон:

```
\(
```