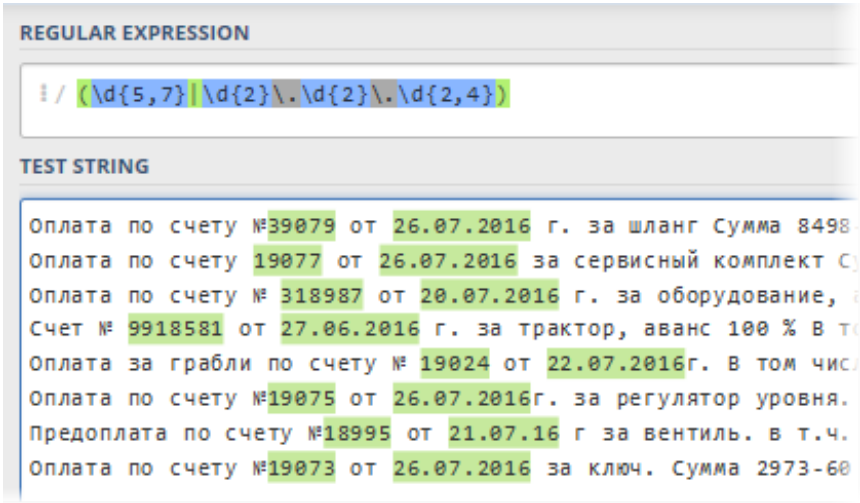
Шпаргалка за регулярними виразами

cinex.ho.ua/shpargalka-za-regulyarnymy-vyrazamy

Cinex 20 квітня 2019 р.



Шпаргалка являє собою загальне керівництво по шаблонах регулярних виразів без урахування специфіки будь-якої мови. Вона представлена у вигляді таблиці, що міститься на одному друкованому аркуші формату A4. Створена під ліцензією Creative Commons на базі шпаргалки, автором якої є Dave Child

Завантажити regexp в PDF

Завантажити regexp в PNG

Потренуватися і протестувати свої регулярки можна на спеціальних сервісах.

Пам'ятайте, що різні мови програмування підтримують регулярні вирази в різному ступені, тому ви можете зіткнутися з ситуацією, коли деякі із зазначених можливостей не будуть працювати. Для тих же, хто тільки знайомиться з регулярними виразами, пропонується цей переклад авторських коментарів до шпаргалки. Він познайомить вас з деякими техніками, що застосовуються при побудові шаблонів регулярних виразів.

Якоря

Якоря в регулярних виразах вказують на початок або кінець чого-небудь. Наприклад, рядки або слова. Вони представлені певними символами. Наприклад, шаблон, відповідний рядку, що починається з цифри, повинен мати такий вигляд:

```
1  ^[0-9]+
```

Тут символ ^ позначає початок рядка. Без нього шаблон відповідав би будь-якому рядку, що містить цифру.

Символьні класи

Символьні класи в регулярних виразах відповідають відразу деякого набору символів. Наприклад, \d відповідає будь цифрі від 0 до 9 включно, \w відповідає буквах і цифрам, а \W - всім символам, крім букв і цифр. Шаблон, що ідентифікує букви, цифри і пробіл, виглядає так:

```
1  \w\s
```

POSIX

POSIX - це відносно нове доповнення сімейства регулярних виразів. Ідея, як і в випадку з символьними класами, полягає в використанні скорочень, що представляють деяку групу символів.

Твердження

Спочатку практично у всіх виникають труднощі з розумінням тверджень, проте познайомившись з ними ближче, ви будете використовувати їх досить часто. Твердження надають спосіб сказати: «я хочу знайти в цьому документі кожне слово, яке включає букву" q ", за якою не слідує " werty "».

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

Наведений вище код починається з пошуку будь-яких символів, крім пробілу ([^ \ s] *), за якими слідує q. Потім парсер досягає «дивиться вперед» твердження. Це автоматично робить попередній елемент (символ, групу або символьний клас) умовним - він буде відповідати шаблоном, тільки якщо твердження вірне. У нашому випадку, твердження є негативним (?!), Тобто воно буде вірним, якщо те, що в ньому шукається, що не буде знайдено.

Отже, парсер перевіряє кілька наступних символів по запропонованими шаблономи (werty). Якщо вони знайдені, то твердження помилкове, а значить символ q буде «проігнорований», Тобто не буде відповідати шаблоном. Якщо ж werty, не знайдено, то твердження вірне, і з q все в порядку. Потім триває пошук будь-яких символів, крім пробілу ([^ \ s] *).

Зразки шаблонів

У цій групі представлені зразки шаблонів. З їх допомогою ви можете побачити, як можна використовувати регулярні вирази в щоденній практиці. Однак зауважте, що вони не обов'язково будуть працювати в будь-якій мові програмування, оскільки кожен з них має індивідуальними особливостями і різним рівнем підтримки регулярних виразів.

Квантори

Квантори дозволяють визначити частину шаблону, яка повинна повторюватися кілька разів поспіль. Наприклад, якщо ви хочете з'ясувати, чи містить документ рядок з від 10 до 20 (включно) букв «a», то можна використовувати цей шаблон:

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

За замовчуванням квантори - «жадібні». Тому квантор +, що означає «один або більше разів», буде відповідати максимально можливому значенню. Іноді це викликає проблеми, і тоді ви можете сказати квантору перестати бути жадібним (стати «ледачим»), використовуючи спеціальний модифікатор. Подивіться на цей код:

```
".*"
```

Цей шаблон відповідає тексту, укладеним в подвійні лапки. Однак, ваша вихідний рядок може бути на кшталт цього:

```
<a href="helloworld.htm" title="Привіт, світ">Привіт, світ</a>
```

Наведений вище шаблон знайде в цьому рядку ось таку підстроку:

```
"helloworld.htm" title="Привіт, світ"
```

Він виявився занадто жадібним, захопивши найбільший шматок тексту, який зміг.

```
".*?"
```

Цей шаблон також відповідає будь-яким символам, укладеними в подвійні лапки. Але лінива версія (зверніть увагу на модифікатор ?) Шукає найменшу з можливих входжень, і тому знайде кожну підстроку в подвійних лапках окремо:

```
"helloworld.htm"
"Привіт, світ"
```

Спеціальні символи

Регулярні вирази використовують деякі символи для позначення різних частин шаблону. Однак, виникає проблема, якщо вам потрібно знайти один з таких символів в рядку, як звичайний символ. Точка, наприклад, в регулярному виразі означає «будь-який символ, крім розриву рядків». Якщо вам потрібно знайти точку в рядку, ви не можете просто використовувати «.» Як шаблон - це призведе до знаходження практично всього. Отже, вам необхідно повідомити парсеру, що ця точка повинна вважатися звичайною точкою, а не «будь-яким символом». Це робиться за допомогою знака екранування.

Знак екранування, що передує символу на кшталт точки, змушує парсер ігнорувати його функцію і вважати звичайним символом. Є кілька символів, які потребують такого захисту, в більшості шаблонів і мов. Ви можете знайти їх в правому нижньому кутку шпаргалки («Мета-символи»).

Шаблон для знаходження точки такий:

```
1  \.
```

Інші спеціальні символи в регулярних виразах відповідають незвичайним елементам в тексті. Перенесення рядка і табуляції, наприклад, можуть бути набрані з клавіатури, але ймовірно зіб'ють з пантелику мови програмування. Знак екранування використовується тут для того, щоб повідомити парсеру про необхідність вважати наступний символ спеціальним, а не звичайною літерою або цифрою.

Підстановка рядків

Підстановка рядків докладно описана в наступному параграфі «Групи і діапазони», однак тут слід згадати про існування «пасивних» груп. Це групи, ігноровані при підстановці, що дуже корисно, якщо ви хочете використовувати в шаблоні умова «або», але не хочете, щоб ця група брала участь в підстановці.

Групи і діапазони

Групи і діапазони дуже-дуже корисні. Ймовірно, простіше буде почати з діапазонів. Вони дозволяють вказати набір відповідних символів. Наприклад, щоб перевірити, чи містить рядок шістнадцятиричні цифри (від 0 до 9 і від A до F), слід використовувати такий діапазон:

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

Щоб перевірити зворотне, використовуйте негативний діапазон, який в нашому випадку підходить під будь-який символ, крім цифр від 0 до 9 і букв від A до F:

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

Групи найбільш часто застосовуються, коли в шаблоні необхідно умова «або»; коли потрібно послатися на частину шаблону з іншої його частини; а також при підстановці рядків.

Використовувати «або» дуже просто: наступний шаблон шукає «ab» або «bc»:

```
1  (ab|bc)
```

Якщо в регулярному виразі необхідно послатися на якусь із попередніх груп, слід використовувати \n, де замість n підставити номер потрібної групи. Вам може знадобитися шаблон, відповідний буквам «aaa» або «bbb», за якими слід число, а потім ті ж три букви. Такий шаблон реалізується за допомогою груп:

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

Перша частина шаблону шукає «aaa» або «bbb», об'єднуючи знайдені букви в групу. Далі йде пошук однієї або більше цифр ([0-9]+), і нарешті \1. Остання частина шаблону посилається на першу групу і шукає те ж саме. Вона шукає збіг з текстом, вже знайденим першою частиною шаблону, а не відповідне йому. Таким чином, «aaa123bbb" не буде задовольняти вищенаведеного шаблоном, так як \1 шукатиме «aaa» після числа.

Одним з найбільш корисних інструментів в регулярних виразах є підстановка рядків. При заміні тексту можна послатися на знайдену групу, використовуючи $n. Скажімо, ви хочете виділити в тексті всі слова «wish» жирним шрифтом. Для цього вам слід використовувати функцію заміни по регулярному виразу, яка може виглядати так:

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

Першим параметром буде приблизно такий шаблон (можливо вам знадобляться кілька додаткових символів для цієї конкретної функції):

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

Він знайде будь-які входження слова «wish» разом з попереднім і наступним символами, якщо тільки це не букви або цифри. Тоді ваша підстановка може бути такою:

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

Нею буде замінений весь знайдений за шаблоном рядок. Ми починаємо заміну з першого знайденого символу (який не буква і не цифра), відзначаючи його $1. Без цього ми б просто видалили цей символ з тексту. Те ж стосується кінця підстановки ($3). В середину ми додали HTML тег для жирного накреслення (зрозуміло, замість нього ви можете використовувати CSS або), виділивши їм другу групу, знайдену за шаблоном ($2).

Модифікатори шаблонів

Модифікатори шаблонів використовуються в декількох мовах, зокрема, в Perl. Вони дозволяють змінити роботу парсеру. Наприклад, модифікатор i змушує парсер ігнорувати регістри.

Регулярне вираження в Perl обрамляються одним і тим же символом на початку і в кінці. Це може бути будь-який символ (частіше використовується «/»), і виглядає все таким чином:

```
/pattern/
```

Модифікатори додаються в кінець цього рядка, ось так:

```
/pattern/i
```

Мета-символи

Нарешті, остання частина таблиці містить мета-символи. Це символи, що мають спеціальне значення в регулярних виразах. Так що якщо ви хочете використовувати один з них як звичайний символ, то його необхідно екранувати. Для перевірки наявності дужки в тексті, використовується такий шаблон:

```
1  \(
```

Джерело