ziishaned Merge pull request #178 from niexia/patch-1 2 years ago

517 lines (356 loc) · 19.7 KB

LEARN REGEX THE EASY WAY

翻译：

- English

master learn-regex / translations / README-cn.md Top

Preview Code Blame Raw

- 日本語
- 한국어
- Turkish
- Greek
- Magyar
- Polish
- Русский
- Tiếng Việt

- فارسی
- עברית

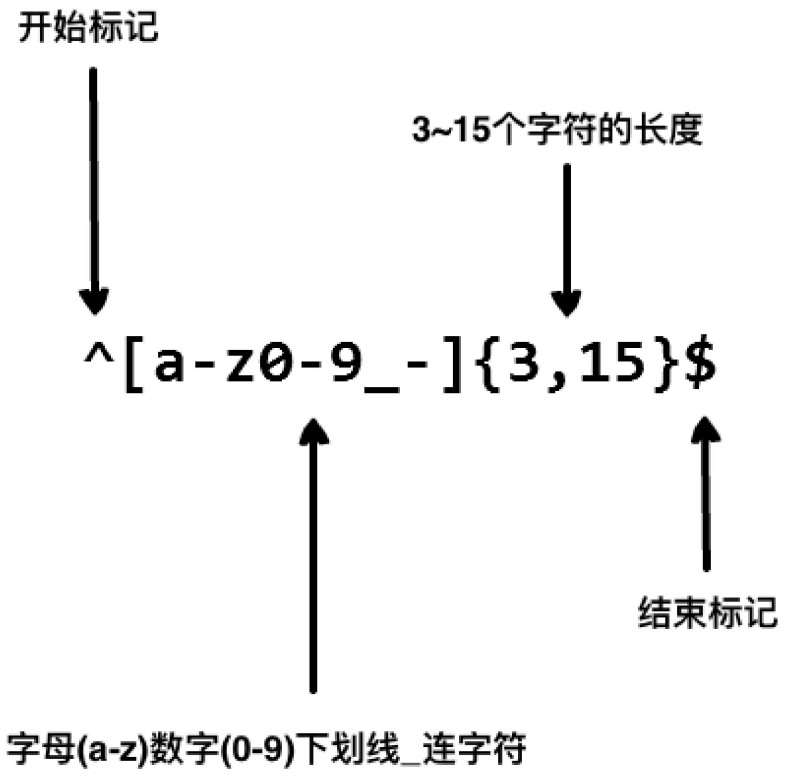
什么是正则表达式?

Download PDF

> 正则表达式是一组由字母和符号组成的特殊文本，它可以用来从文本中找出满足你想要的格式的句子。

一个正则表达式是一种从左到右匹配主体字符串的模式。 “Regular expression”这个词比较拗口，我们常使用缩写的术语“regex”或“regexp”。 正则表达式可以从一个基础字符串中根据一定的匹配模式替换文本中的字符串、验证表单、提取字符串等等。

想象你正在写一个应用，然后你想设定一个用户命名的规则，让用户名包含字符、数字、下划线和连字符，以及限制字符的个数，好让名字看起来没那么丑。 我们使用以下正则表达式来验证一个用户名：



以上的正则表达式可以接受 `john_doe` 、 `jo-hn_doe` 、 `john12_as` 。 但不匹配 `Jo` ，因为它包含了大写的字母而且太短了。

目录

1. 基本匹配

正则表达式其实就是在执行搜索时的格式，它由一些字母和数字组合而成。 例如：一个正则表达式 `the` ，它表示一个规则：由字母 `t` 开始，接着是 `h` ，再接着是 `e` 。

```
"the" => The fat cat sat on the mat.
```

在线练习

正则表达式 `123` 匹配字符串 `123` 。它逐个字符的与输入的正则表达式做比较。

正则表达式是大小写敏感的，所以 `The` 不会匹配 `the` 。

```
"The" => The fat cat sat on the mat.
```

在线练习

2. 元字符

正则表达式主要依赖于元字符。 元字符不代表他们本身的字面意思，他们都有特殊的含义。一些元字符写在方括号中的时候有一些特殊的意思。以下是一些元字符的介绍：

元字符	描述
.	句号匹配任意单个字符除了换行符。
[ ]	字符种类。匹配方括号内的任意字符。
[^]	否定的字符种类。匹配除了方括号里的任意字符
*	匹配>=0个重复的在*号之前的字符。
+	匹配>=1个重复的+号前的字符。
?	标记?之前的字符为可选.
{n,m}	匹配num个大括号之前的字符或字符集 (n <= num <= m).
(xyz)	字符集，匹配与 xyz 完全相等的字符串.
|	或运算符，匹配符号前或后的字符.
\	转义字符,用于匹配一些保留的字符 `[ ] ( ) { } . * + ? ^ $ \ |`
^	从开始行开始匹配.
$	从末端开始匹配.

2.1 点运算符 .

. 是元字符中最简单的例子。 . 匹配任意单个字符，但不匹配换行符。 例如，表达式 `.ar` 匹配一个任意字符后面跟着是 `a` 和 `r` 的字符串。

```
".ar" => The car parked in the garage.
```

在线练习

2.2 字符集

字符集也叫做字符类。 方括号用来指定一个字符集。 在方括号中使用连字符来指定字符集的范围。 在方括号中的字符集不关心顺序。 例如，表达式 `[Tt]he` 匹配 `the` 和 `The` 。

```
"[Tt]he" => The car parked in the garage.
```

在线练习

方括号的句号就表示句号。 表达式 `ar[.]` 匹配 `ar.` 字符串

```
"ar[.]" => A garage is a good place to park a car.
```

在线练习

2.2.1 否定字符集

一般来说 `^` 表示一个字符串的开头，但它用在一个方括号的开头的时候，它表示这个字符集是否定的。 例如，表达式 `[^c]ar` 匹配一个后面跟着 `ar` 的除了 `c` 的任意字符。

```
"[^c]ar" => The car parked in the garage.
```

在线练习

2.3 重复次数

后面跟着元字符 `+` , `*` or `?` 的，用来指定匹配子模式的次数。 这些元字符在不同的情况下有着不同的意思。

2.3.1 * 号

`*` 号匹配 在 `*` 之前的字符出现 `大于等于0` 次。 例如，表达式 `a*` 匹配0或更多个以a开头的字符。表达式 `[a-z]*` 匹配一个行中所有以小写字母开头的字符串。

```
"[a-z]*" => The car parked in the garage #21.
```

在线练习

`*` 字符和 `.` 字符搭配可以匹配所有的字符 `.*` 。 `*` 和表示匹配空格的符号 `\s` 连起来用，如表达式 `\s*cat\s*` 匹配0或更多个空格开头和0或更多个空格结尾的cat字符串。

```
"\s*cat\s*" => The fat cat sat on the concatenation.
```

在线练习

2.3.2 + 号

+ 号匹配 + 号之前的字符出现 >=1 次。 例如表达式 c.+t 匹配以首字母 c 开头以 t 结尾，中间跟着至少一个字符的字符串。

```
"c.+t" => The fat cat sat on the mat.
```

在线练习

2.3.3 ? 号

在正则表达式中元字符 ? 标记在符号前面的字符为可选，即出现 0 或 1 次。 例如，表达式 [T]?he 匹配字符串 he 和 The 。

```
"[T]he" => The car is parked in the garage.
```

在线练习

```
"[T]?he" => The car is parked in the garage.
```

在线练习

2.4 {} 号

在正则表达式中 {} 是一个量词，常用来限定一个或一组字符可以重复出现的次数。 例如， 表达式 [0-9]{2,3} 匹配最少 2 位最多 3 位 0~9 的数字。

```
"[0-9]{2,3}" => The number was 9.9997 but we rounded it off to 10.0.
```

在线练习

我们可以省略第二个参数。 例如， [0-9]{2,} 匹配至少两位 0~9 的数字。

```
"[0-9]{2,}" => The number was 9.9997 but we rounded it off to 10.0.
```

在线练习

如果逗号也省略掉则表示重复固定的次数。 例如， `[0-9]{3}` 匹配3位数字

```
"[0-9]{3}" => The number was 9.9997 but we rounded it off to 10.0.
```

在线练习

2.5（...）特征标群

特征标群是一组写在 `(...)` 中的子模式。`(...)` 中包含的内容将会被看成一个整体，和数学中小括号（）的作用相同。例如, 表达式 `(ab)*` 匹配连续出现 0 或更多个 `ab` 。如果没有使用 `(...)` ，那么表达式 `ab*` 将匹配连续出现 0 或更多个 `b` 。再比如之前说的 `{}` 是用来表示前面一个字符出现指定次数。但如果在 `{}` 前加上特征标群 `(...)` 则表示整个标群内的字符重复 N 次。

我们还可以在 `()` 中用或字符 `|` 表示或。例如， `(c|g|p)ar` 匹配 `car` 或 `gar` 或 `par` .

```
"(c|g|p)ar" => The car is parked in the garage.
```

在线练习

2.6 | 或运算符

或运算符就表示或，用作判断条件。

例如 `(T|t)he|car` 匹配 `(T|t)he` 或 `car` 。

```
"(T|t)he|car" => The car is parked in the garage.
```

在线练习

2.7 转码特殊字符

反斜线 `\` 在表达式中用于转码紧跟其后的字符。用于指定 `{ } [ ] / \ + * . $ ^ | ?` 这些特殊字符。如果想要匹配这些特殊字符则要在其前面加上反斜线 `\` 。

例如 `.` 是用来匹配除换行符外的所有字符的。如果想要匹配句子中的 `.` 则要写成 `\.` 以下这个例子 `\.?` 是选择性匹配 `.`

```
"(f|c|m)at\.?" => The fat cat sat on the mat.
```

在线练习

2.8 锚点

在正则表达式中，想要匹配指定开头或结尾的字符串就要使用到锚点。 `^` 指定开头， `$` 指定结尾。

2.8.1 ^ 号

`^` 用来检查匹配的字符串是否在所匹配字符串的开头。

例如，在 `abc` 中使用表达式 `^a` 会得到结果 `a` 。但如果使用 `^b` 将匹配不到任何结果。因为在字符串 `abc` 中并不是以 `b` 开头。

例如， `^(T|t)he` 匹配以 `The` 或 `the` 开头的字符串。

```
"(T|t)he" => The car is parked in the garage.
```

在线练习

```
"^(T|t)he" => The car is parked in the garage.
```

在线练习

2.8.2 $ 号

同理于 `^` 号， `$` 号用来匹配字符是否是最后一个。

例如， `(at\.)$` 匹配以 `at.` 结尾的字符串。

```
"(at\.)" => The fat cat. sat. on the mat.
```

在线练习

```
"(at\.)$" => The fat cat. sat. on the mat.
```

在线练习

3. 简写字符集

正则表达式提供一些常用的字符集简写。如下:

简写	描述
.	除换行符外的所有字符
\w	匹配所有字母数字，等同于 `[a-zA-Z0-9_]`
\W	匹配所有非字母数字，即符号，等同于： `[^\w]`
\d	匹配数字： `[0-9]`
\D	匹配非数字： `[^\d]`
\s	匹配所有空格字符，等同于： `[\t\n\f\r\p{Z}]`
\S	匹配所有非空格字符： `[^\s]`
\f	匹配一个换页符
\n	匹配一个换行符
\r	匹配一个回车符
\t	匹配一个制表符
\v	匹配一个垂直制表符
\p	匹配 CR/LF（等同于 `\r\n` ），用来匹配 DOS 行终止符

4. 零宽度断言（前后预查）

先行断言和后发断言（合称 lookaround）都属于**非捕获组**（用于匹配模式，但不包括在匹配列表中）。当我们需要一个模式的前面或后面有另一个特定的模式时，就可以使用它们。

例如，我们希望从下面的输入字符串 `$4.44` 和 `$10.88` 中获得所有以 `$` 字符开头的数字，我们将使用以下的正则表达式 `(?<=\$)[0-9\.]*` 。意思是：获取所有包含 `.` 并且前面是 `$` 的数字。

零宽度断言如下：

符号	描述
?=	正先行断言-存在
?!	负先行断言-排除
?<=	正后发断言-存在
?<!	负后发断言-排除

4.1 ?=... 正先行断言

?=... 正先行断言，表示第一部分表达式之后必须跟着 ?=... 定义的表达式。

返回结果只包含满足匹配条件的第一部分表达式。 定义一个正先行断言要使用 () 。在括号内部使用一个问号和等号： (?=...) 。

正先行断言的内容写在括号中的等号后面。 例如，表达式 (T|t)he(?=\sfat) 匹配 The 和 the ，在括号中我们又定义了正先行断言 (?=\sfat) ，即 The 和 the 后面紧跟着 (空格)fat 。

```
"(T|t)he(?=\sfat)" => The fat cat sat on the mat.
```

在线练习

4.2 ?!... 负先行断言

负先行断言 ?! 用于筛选所有匹配结果，筛选条件为 其后不跟随着断言中定义的格式。 正先行断言 定义和 负先行断言 一样，区别就是 = 替换成 ! 也就是 (?!...) 。

表达式 (T|t)he(?!\sfat) 匹配 The 和 the ，且其后不跟着 (空格)fat 。

```
"(T|t)he(?!\sfat)" => The fat cat sat on the mat.
```

在线练习

4.3 ?<= ... 正后发断言

正后发断言 记作 (?<=...) 用于筛选所有匹配结果，筛选条件为 其前跟随着断言中定义的格式。 例如，表达式 (?<=(T|t)he\s)(fat|mat) 匹配 fat 和 mat ，且其前跟着 The 或 the 。

```
"(?<=(T|t)he\s)(fat|mat)" => The fat cat sat on the mat.
```

在线练习

4.4 ?<!... 负后发断言

负后发断言 记作 (?<!...) 用于筛选所有匹配结果，筛选条件为 其前不跟随着断言中定义的格式。 例如，表达式 (?<!(T|t)he\s)(cat) 匹配 cat ，且其前不跟着 The 或 the 。

```
"(?<!(T|t)he\s)(cat)" => The cat sat on cat.
```

在线练习

5. 标志

标志也叫模式修正符，因为它可以用来修改表达式的搜索结果。 这些标志可以任意的组合使用，它也是整个正则表达式的一部分。

标志	描述
i	忽略大小写。
g	全局搜索。
m	多行修饰符：锚点元字符 `^` `$` 工作范围在每行的起始。

5.1 忽略大小写 (Case Insensitive)

修饰语 `i` 用于忽略大小写。 例如，表达式 `/The/gi` 表示在全局搜索 `The` ，在后面的 `i` 将其条件修改为忽略大小写，则变成搜索 `the` 和 `The` ， `g` 表示全局搜索。

```
"The" => The fat cat sat on the mat.
```

在线练习

```
"/The/gi" => The fat cat sat on the mat.
```

在线练习

5.2 全局搜索 (Global search)

修饰符 `g` 常用于执行一个全局搜索匹配，即（不仅仅返回第一个匹配的，而是返回全部）。 例如，表达式 `/.(at)/g` 表示搜索 任意字符（除了换行） `+` `at` ，并返回全部结果。

```
"/.(at)/" => The fat cat sat on the mat.
```

在线练习

```
"/.(at)/g" => The fat cat sat on the mat.
```

在线练习

5.3 多行修饰符 (Multiline)

多行修饰符 `m` 常用于执行一个多行匹配。

像之前介绍的 `(^,$)` 用于检查格式是否是在待检测字符串的开头或结尾。但我们如果想要它在每行的开头和结尾生效，我们需要用到多行修饰符 `m` 。

例如，表达式 `/at(.)?$/gm` 表示小写字符 `a` 后跟小写字符 `t` ，末尾可选除换行符外任意字符。根据 `m` 修饰符，现在表达式匹配每行的结尾。

```
"/.at(.)?$/" => The fat
                cat sat
                on the mat.
```

在线练习

```
"/.at(.)?$/gm" => The fat
                  cat sat
                  on the mat.
```

在线练习

6. 贪婪匹配与惰性匹配 (Greedy vs lazy matching)

正则表达式默认采用贪婪匹配模式，在该模式下意味着会匹配尽可能长的子串。我们可以使用 `?` 将贪婪匹配模式转化为惰性匹配模式。

```
"/(.*at)/" => The fat cat sat on the mat.
```

在线练习

```
"/(.*?at)/" => The fat cat sat on the mat.
```

在线练习

贡献

- 报告问题
- 开放合并请求
- 传播此文档
- 直接和我联系 ziishaned@gmail.com 或 

许可证