

02 | 量词与贪婪：小小的正则，也可能把CPU拖垮！

2020-06-15 涂伟忠

正则表达式入门课

[进入课程 >](#)



讲述：涂伟忠

时长 16:41 大小 15.29M



你好，我是涂伟忠。在上一讲中，我们已经学习了正则中和一些元字符相关的内容。这一节我们讲一下正则中的三种模式，贪婪匹配、非贪婪匹配和独占模式。

这些模式会改变正则中量词的匹配行为，比如匹配一到多次；在匹配的时候，匹配长度是尽可能长还是要尽可能短呢？如果不知道贪婪和非贪婪匹配模式，我们写的正则很可能是错误的，这样匹配就达不到期望的效果了。

为什么会有贪婪与非贪婪模式？



由于本节内容和量词相关的元字符密切相关，所以我们先来回顾一下正则中表示量词的元字符。



在这 6 种元字符中, 我们可以用 {m,n} 来表示 (*) (+) (?) 这 3 种元字符:

元字符	同义表示方法	示例
*	{0,}	ab* 可以匹配 a 或 abbb
+	{1,}	正则 ab+ 可以匹配 ab 或 abbb 但不能匹配 a
?	{0,1}	正则 (\\+86-)?\\d{11} 可以匹配 +86-13800138000 或 13800138000

表示量词的星号 (*) 和 加号 (+) 可能没你想象的那么简单, 我用一个例子给你讲解一下。我们先看一下加号 (+), 使用 a+ 在 aaabb 中查找, 可以看到只有一个输出结果:

REGULAR EXPRESSION

1 match, 3 steps (~0ms)

/ a+

/ gm

TEST STRING

SWITCH TO UNIT TESTS ▶

aaabb



对应的 Python 代码如下：

```
1 >>> import re
2 >>> re.findall(r'a+', 'aaabb')
3 ['aaa']
4
```

复制代码

加号应该很容易理解，我们再使用 `a*` 在 `aaabb` 这个字符串中进行查找，这次我们看到可以找到 4 个匹配结果。

REGULAR EXPRESSION

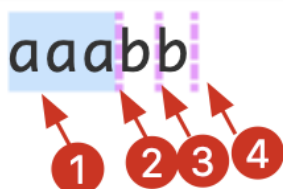
4 matches, 14 steps (~0ms)

/ a*

/ gm

TEST STRING

SWITCH TO UNIT TESTS ▶



使用 Python 示例如下，我们可以看到输出结果，也是得到了 4 个匹配结果：

```
1 >>> import re
2 >>> re.findall(r'a*', 'aaabb')
3 ['aaa', '', '', '']
4
```

复制代码

但这一次的结果匹配到了三次空字符串。为什么会匹配到空字符串呢？因为星号 (*) 代表 0 到多次，匹配 0 次就是空字符串。到这里，你可能会疑问，如果这样，aaa 部分应该也有空字符串，为什么没匹配上呢？

这就引入了我们今天要讲的话题，贪婪与非贪婪模式。这两种模式都必须满足匹配次数的要求才能匹配上。贪婪模式，简单说就是尽可能进行最长匹配。非贪婪模式呢，则会尽可能进行最短匹配。正是这两种模式产生了不同的匹配结果。

贪婪、非贪婪与独占模式

贪婪匹配 (Greedy)

首先，我们来看一下贪婪匹配。在正则中，表示次数的量词默认是贪婪的，在贪婪模式下，会尝试尽可能最大长度去匹配。

首先，我们来看一下在字符串 aaabb 中使用正则 a* 的匹配过程。

字符串	aaabb
下标	012345

匹配	开始	结束	说明	匹配内容
第1次	0	3	到第一个字母b发现不满足，输出aaa	aaa
第2次	3	3	匹配剩下的bb，发现匹配不上，输出空字符串	空字符串
第3次	4	4	匹配剩下的b，发现匹配不上，输出空字符串	空字符串
第4次	5	5	匹配剩下空字符串，输出空字符串	空字符串

a* 在匹配开头的 a 时，会尝试尽量匹配更多的 a，直到第一个字母 b 不满足要求为止，匹配上三个 a，后面每次匹配时都得到了空字符串。

相信看到这里你也发现了，贪婪模式的特点就是尽可能进行最大长度匹配。所以要不要使用贪婪模式是根据需求场景来定的。如果我们想尽可能最短匹配呢？那就要用到非贪婪匹配模式了。

非贪婪匹配 (Lazy)

那么如何将贪婪模式变成非贪婪模式呢？我们可以在量词后面加上英文的问号 (?)，正则就变成了 a*?。此时的匹配结果如下：

REGULAR EXPRESSION

9 matches, 27 steps (~0ms)

/ a*?

/ gm

TEST STRING

SWITCH TO UNIT TESTS ▶

aaabb



复制代码

```
1 >>> import re
2 >>> re.findall(r'a*', 'aaabb') # 贪婪模式
3 ['aaa', '', '', '']
4 >>> re.findall(r'a*?', 'aaabb') # 非贪婪模式
5 ['', 'a', '', 'a', '', 'a', '', '']
```

这一次我们可以看到，这次匹配到的结果都是单个的 a，就连每个 a 左边的空字符串也匹配上了。

到这里你可能就明白了，非贪婪模式会尽可能短地去匹配，我把这两者之间的区别写到了下面这张图中。

贪婪：表示次数的量词，默认是贪婪的
默认尽可能多地去匹配

贪婪&非贪婪

非贪婪：“数量”元字符后加？（英文问号）
找出长度最小且满足要求的

为了让你加深理解，我们再来看一个示例，这一次让我们查找一下引号中的单词。

从下面这个示例中，我们可以很容易看出两者对比上的差异。左右的文本是一样的，其中有两对双引号。不同之处在于，左边的示例中，不加问号时正则贪婪匹配，匹配上了从第一个引号到最后一个引号之间的所有内容；而右边的图是非贪婪匹配，找到了符合要求的结果。

REGULAR EXPRESSION	1 match, 6 steps (~1ms)
<code>r' ".+" </code> 贪婪匹配 <code>gm</code>	
TEST STRING	SWITCH TO UNIT TESTS ▾
<code>"the little cat" is a toy, it looks "a little bad".</code>	

REGULAR EXPRESSION	2 matches, 32 steps (~0ms)
<code>r' ".+?" </code> 非贪婪匹配 <code>gm</code>	
TEST STRING	SWITCH TO UNIT TESTS ▾
<code>"the little cat" is a toy, it looks "a little bad".</code>	

独占模式 (Possessive)

不管是贪婪模式，还是非贪婪模式，都需要发生回溯才能完成相应的功能。但是在一些场景下，我们不需要回溯，匹配不上返回失败就好了，因此正则中还有另外一种模式，独占模式，它类似贪婪匹配，但匹配过程不会发生回溯，因此在一些场合下性能会更好。

你可能会问，那什么是回溯呢？我们来看一些例子，例如下面的正则：

```
regex = "xy{1,3}z"
```

```
text = "xyyz"
```

在匹配时，`y{1,3}`会尽可能长地去匹配，当匹配完 `xyy` 后，由于 `y` 要尽可能匹配最长，即三个，但字符串中后面是个 `z` 就会导致匹配不上，这时候正则就会**向前回溯**，吐出当前字符 `z`，接着用正则中的 `z` 去匹配。

x	y	y	z		x	y	y	z
xy{1,3}z					xy{1,3}z			
匹配不上，回溯（即z会吐出来），再用z去匹配								

如果我们把这个正则改成非贪婪模式，如下：

```
regex = "xy{1,3}?z"
```

```
text = "xyyz"
```

由于 `y{1,3}?` 代表匹配 1 到 3 个 `y`，尽可能少地匹配。匹配上一个 `y` 之后，也就是在匹配上 `text` 中的 `xy` 后，正则会使用 `z` 和 `text` 中的 `xy` 后面的 `y` 比较，发现正则 `z` 和 `y` 不匹配，这时正则就会**向前回溯**，重新查看 `y` 匹配两个的情况，匹配上正则中的 `xyy`，然后再用 `z` 去匹配 `text` 中的 `z`，匹配成功。

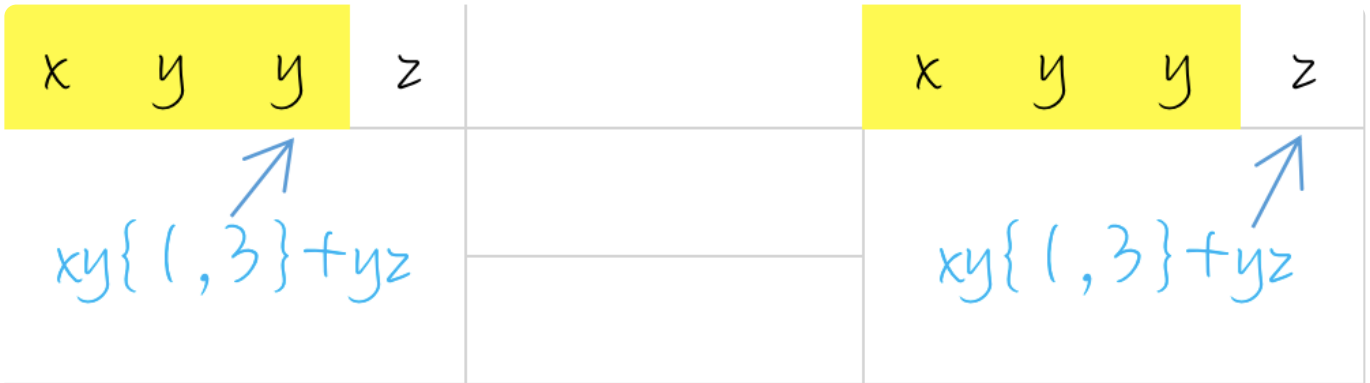
x	y	y	z		x	y	y	z
xy{1,3}?z					xy{1,3}?z			
正则z匹配不上，回溯，重新尝试匹配两个y的情况								

了解了回溯，我们再看下独占模式。

独占模式和贪婪模式很像，独占模式会尽可能多地去匹配，如果匹配失败就结束，不会进行回溯，这样的话就比较节省时间。具体的方法就是在量词后面加上加号（+）。


```
regex = "xy{1,3}+yz"
```

```
text = "xyyz"
```



$y\{1,3\}+$ 尽可能多地匹配了两个 y ，不回溯导致正则 z 前面的 y 匹配不上

需要注意的是 Python 和 Go 的标准库目前都不支持独占模式，会报错，如下所示：

```
1 >>> import re
2 >>> re.findall(r'xy{1,3}+yz', 'xyyz')
3 error: multiple repeat at position 7
4
```

复制代码


报错显示，加号 (+) 被认为是重复次数的元字符了。如果要测试这个功能，我们可以安装 PyPI 上的 regex 模块。

```
1 注意：需要先安装 regex 模块, pip install regex
2
3 >>> import regex
4 >>> regex.findall(r'xy{1,3}z', 'xyyz') # 贪婪模式
5 ['xyyz']
6 >>> regex.findall(r'xy{1,3}+z', 'xyyz') # 独占模式
7 ['xyyz']
8 >>> regex.findall(r'xy{1,2}+yz', 'xyyz') # 独占模式
9 []
```

复制代码

你也可以使用 Java 或 Perl 等其它语言来测试独占模式，查阅相关文档，看一下你所用的语言对独占模式的支持程度。

这个正则比较长，但很好理解，中括号里面代表多选一，我们简化一下，就成下面这样：

 复制代码


```
1 ^([符合要求的组成1]|[符合要求的组成2])+ $
```

脱字符（^）代表以这个正则开头，美元符号（\$）代表以正则结尾，我们后面会专门进行讲解。这里可以先理解成整个店铺名称要能匹配上正则，即起到验证的作用。

你需要留意的是，正则中有个加号（+），表示前面的内容出现一到多次，进行贪婪匹配，这样会导致大量回溯，占用大量 CPU 资源，引发线上问题，我们只需要将贪婪模式改成独占模式就可以解决这个问题。

我之前说过，要根据具体情况来选择合适的模式，在这个例子中，匹配不上时证明店铺名不合法，不需要进行回溯，因此我们可以使用独占模式，但要注意并不是说所有的场合都可以用独占模式解决，我们要首先保证正则能满足功能需求。

仔细再看一下这个正则，你会发现“组成 1”和“组成 2”部分中，A-Za-z 英文字母在两个集合里面重复出现了，这会导致回溯后的重复判断。这里要强调一下，并不是说有回溯就会导致问题，你应该尽量减少回溯后的计算量，这些在后面的原理讲解中我们会进一步学习。

另外，腾讯云技术社区也有类似的技术文章，你如果感兴趣，可以点击[这里](#)  进行查看。

说到这里，你是不是想起了课程开篇里面提到的一句话：

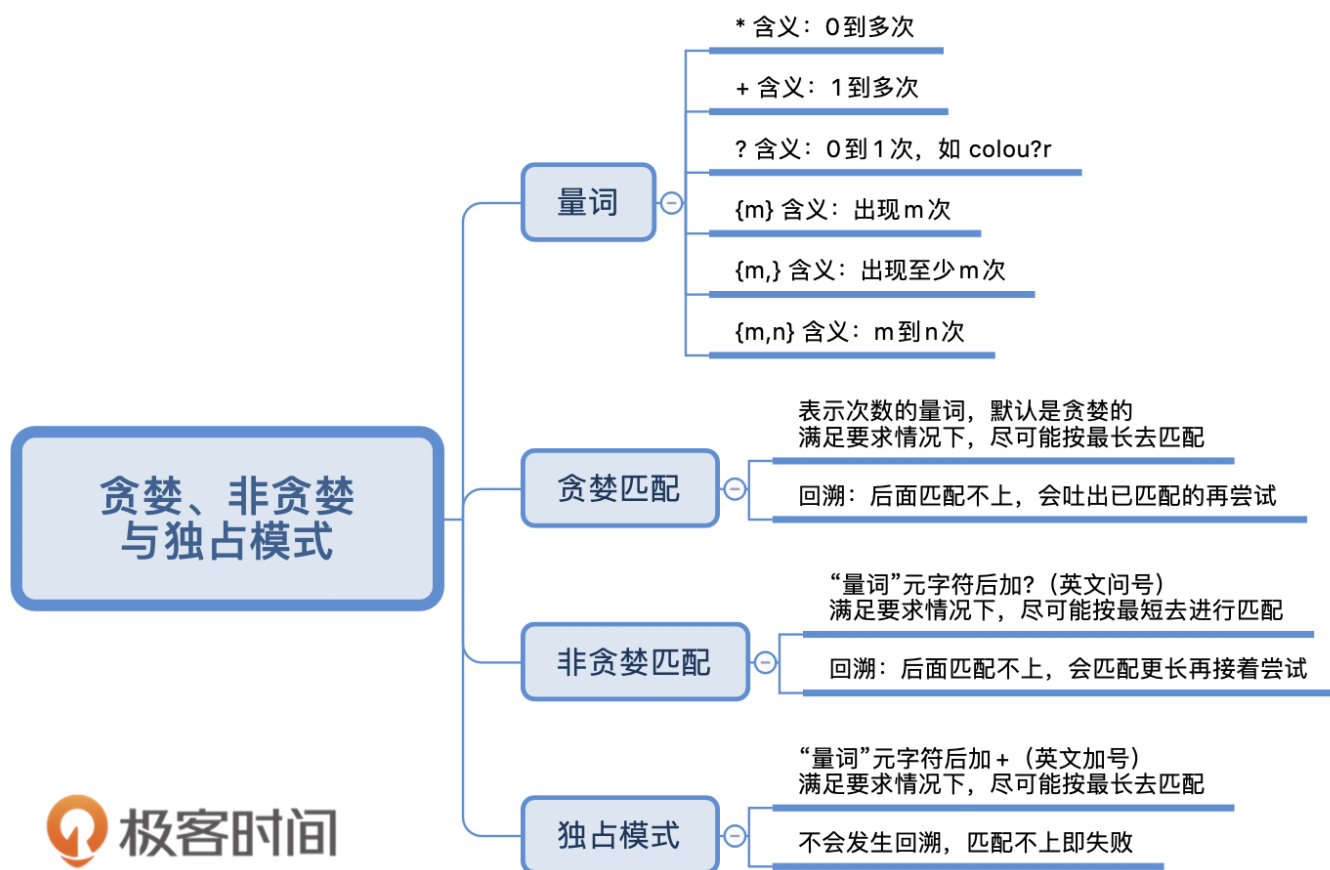
如果你有一个问题，你想到可以用正则来解决，那么你有两个问题了。

Some people, when confronted with a problem, think “I know, I’ll use regular expressions.” Now they have two problems.

所以一个小小的正则，有些时候也可能会把 CPU 拖垮，这也提醒我们在写正则的时候，一定要思考下回溯问题，避免使用低效的正则，引发线上问题。

最后总结

最后我来给你总结一下：正则中量词默认是贪婪匹配，如果想要进行非贪婪匹配需要在量词后面加上问号。贪婪和非贪婪匹配都可能会进行回溯，独占模式也是进行贪婪匹配，但不进行回溯，因此在一些场景下，可以提高匹配的效率，具体能不能用独占模式需要看使用的编程语言的类库的支持情况，以及独占模式能不能满足需求。



课后思考

最后，我们来做一个小练习吧。

有一篇英文文章，里面有很多单词，单词和单词之间是用空格隔开的，在引号里面的一到多个单词表示特殊含义，即引号里面的多个单词要看成一个单词。现在你需要提取出文章中所有的单词。我们可以假设文章中除了引号没有其它的标点符号，有什么方法可以解决这个问题呢？如果用正则来解决，你能不能写出一个正则，提取出文章中所有的单词呢（不要求结果去重）？

we found “the little cat” is in the hat, we like “the little cat”

其中 the little cat 需要看成一个单词

好了，今天的课程就结束了，希望可以帮助到你，也希望你在下方的留言区和我参与讨论，并把文章分享给你的朋友或者同事，一起交流一下。

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 01 | 元字符：如何巧妙记忆正则表达式的基本元件？

精选留言 (30)

写留言



Geek.S.

2020-06-15

以前只知道贪婪模式和懒惰模式，原来还有一个独占模式，贪婪和非贪婪都会发生回溯，结合文中给的案例链接，知道了 NFA 和 DFA (这个老师应该在后面讲匹配原理时会讲到)。难怪余晟老师说学会正则后务必要学会克制。

如果只是判断文本是否符合规则，则可以使用独占模式; 如果需要获取匹配的结果，则根...
展开

作者回复: 对的，务必克制，搞懂原理才能用的更好。

答案是对的



6



一步

2020-06-15

老师，对于文中的这个语句
`regex.findall(r'xy{1,3}+z', 'xyyz')`

这里是独占模式，不进行回溯。这里在尽可能多的匹配第三个 y 的时候匹配失败，不应该是直接匹配失败 返回空数组吗？ 怎么是返回 xyyz 呢？ 如果返回 xyyz 不就进行回溯了...
展开

1

2



pyhhou

2020-06-15

思考题：

".+?"|[^\\s|,|,]+

关于回溯，是不是就是递归调用函数栈的原理？拿 $xy\{1,3\}z$ 匹配 $xyyz$ 举例，步骤如下： ...

展开 ▾



2



Robot

2020-06-16

测试结果跟文中的不符

```
>>> import re
>>> re.findall(r'a*?', 'aaabb')
['', '', '', '', '', '']
```

展开 ▾



1



1



Robot

2020-06-16

课后思考：

$/"[^"]+"|[a-z]+/g$

展开 ▾



1



卡尔

2020-06-16

$^([a-zA-Z]+|"[^"]+")\$$

展开 ▾



1



飞

2020-06-15

$[a-z]+| "[^"]+"$

展开 ▾



1



Billions

2020-06-15

$\backslash w+| "[^"]*"$
 $\backslash w+| "[\\w\\s]+?"$

\w+| ".+?"

还有第四种方法吗?



1



中年男子

2020-06-15

还有就是文章中的例子: `xy{1,3}+yz` 去匹配 `xyyz`, 我的理解是用`y{1,3}`尽可能多的去匹配, 也就是 `xyy`之后, 用第三个`y` 去匹配`z`, 因为不回溯, 到这里就失败了, 还没到正则中`z`前面那个`y`。

还请解惑。

展开

作者回复: `y`一到三次独占模式, 虽然只匹配到了两个, 但还是满足了次数要求, 这时候没失败, 继续看下一个, 后面的`y`匹配不上`z`才失败的



1



中年男子

2020-06-15

老师, 我有个问题, 既然是独占模式了, 那`{1,3}` 这种量词是不是就没什么意义? 直接`{3}` 不就行了

作者回复: 不是, 独占模式类似于贪婪模式, 如果只有两个`a`也能匹配上, 直接写`3`只能匹配3个`a`了, 你可以理解成正则中`a`匹配不上字符串中的`b`, 会接着用正则后面的内容去匹配`b`



1

1



Geek.S.

2020-06-15

独占模式是贪婪的, 很好奇有没有非贪婪的独占模式? 老师可不可以分析一下这个问题?



1



c1rew

2020-06-16

\w+| ".+?" (10 matches, 74 steps)

<https://regex101.com/r/eLihse/1/>

\w+| "(.+?)" (10 matches, 104 steps)

只要引号内的the little cat，加括号分组提取 ...

展开 ▾



m

2020-06-16

`^w+| ".+?" /g`

虽然不知道自己写的什么，但是好像匹配上了...

the little cat 需要看成一个单词，但是匹配上的有 ""

展开 ▾



Januarius

2020-06-16

import regex

```
regex.findall(r"""\w+| "[^"]]+"',we found "the little cat" is in the hat, we like "the little cat" ')
```



明明不平凡

2020-06-16

`[^\\ "\s]+| ".+?"`

文本中有可能出现非\w的情况吧？



yaomon

2020-06-16

`\w+| ".+?"`

展开 ▾



sotey

2020-06-16

`"[a-z\s]+"|"[a-z]+`

展开 ▾



HardToGiveName

2020-06-16

之前一直对 贪婪和非贪婪模式马马虎虎，至今才知道居然还有独占模式....收获很大
思考题：

`\w+| ".*?"`

<https://regex101.com/r/PnzZ4k/48>

后面部分的 `".*?"` 主要是想练习一下 非贪婪模式的 最小限度匹配引号中的单词

展开 ▾



爱乐之城

2020-06-16

请问老师，回溯和重复判断那个例子，回溯是吐出之前已匹配的最后一个字符，这和重复判断有什么关系呢？



飞

2020-06-15

老师有地方没理解，在阿里案例中，正则中有个加号（+），独占模式匹配失败就会停止匹配，为何您说这样会导致大量回溯呢？

展开 ▾

1

