04 | 慎重使用正则表达式

2019-05-28 刘超

你好，我是刘超。

上一讲，我在讲String对象优化时，提到了Split()方法，该方法使用的正则表达式可能引起回溯问题，今天我们就来深入了解下，这究竟是怎么回事？

开始之前，我们先来看一个案例，可以帮助你更好地理解内容。

在一次小型项目开发中，我遇到过这样一个问题。为了宣传新品，我们开发了一个小程序，按照之前评估的访问量，这次活动预计参与用户量30W+，TPS（每秒事务处理量）最高3000左右。

这个结果来自我对接口做的微基准性能测试。我习惯使用ab工具（通过yum -y install httpd-tools可以快速安装）在另一台机器上对http请求接口进行测试。

我可以通过设置-n请求数/-c并发用户数来模拟线上的峰值请求，再通过TPS、RT（每秒响应时间）以及每秒请求时间分布情况这三个指标来衡量接口的性能，如下图所示（图中隐藏部分为我的服务器地址）：

```
[root@...  ~]# ab -n100000 -c 1000 http://...
This is ApacheBench, Version 2.3 <$Revision: 1430300 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking 10.11.100.146 (be patient)
Completed 10000 requests
Completed 20000 requests
Completed 30000 requests
Completed 40000 requests
Completed 50000 requests
Completed 60000 requests
Completed 70000 requests
Completed 80000 requests
Completed 90000 requests
Completed 100000 requests
Finished 100000 requests


Server Software:
Server Hostname:        ...
Server Port:            ----

Document Path:          ...
Document Length:        36801 bytes

Concurrency Level:      1000
Time taken for tests:   30.024 seconds
Complete requests:      100000
Failed requests:        0
Write errors:           0
Total transferred:      3692000000 bytes
HTML transferred:       3680100000 bytes
Requests per second:    3330.63 [#/sec] (mean)
Time per request:       300.243 [ms] (mean)
Time per request:       0.300 [ms] (mean, across all concurrent requests)
Transfer rate:          120084.87 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0  117 384.3     11    7018
Processing:     4  171 236.6    136    6066
Waiting:        2  148 236.7    110    6061
Total:         13  288 444.7    164    7239

Percentage of the requests served within a certain time (ms)
  50%    164
  66%    202
  75%    232
  80%    261
  90%   1063
  95%   1167
  98%   1281
  99%   1396
 100%   7239 (longest request)
```

就在做性能测试的时候，我发现有一个提交接口的TPS一直上不去，按理说这个业务非常简单，存在性能瓶颈的可能性并不大。

我迅速使用了排除法查找问题。首先将方法里面的业务代码全部注释，留一个空方法在这里，再看性能如何。这种方式能够很好地区分是框架性能问题，还是业务代码性能问题。

我快速定位到了是业务代码问题，就马上逐一查看代码查找原因。我将插入数据库操作代码加上之后，TPS稍微下降了，但还是没有找到原因。最后，就只剩下Split() 方法操作了，果然，我将Split()方法加入之后，TPS明显下降了。

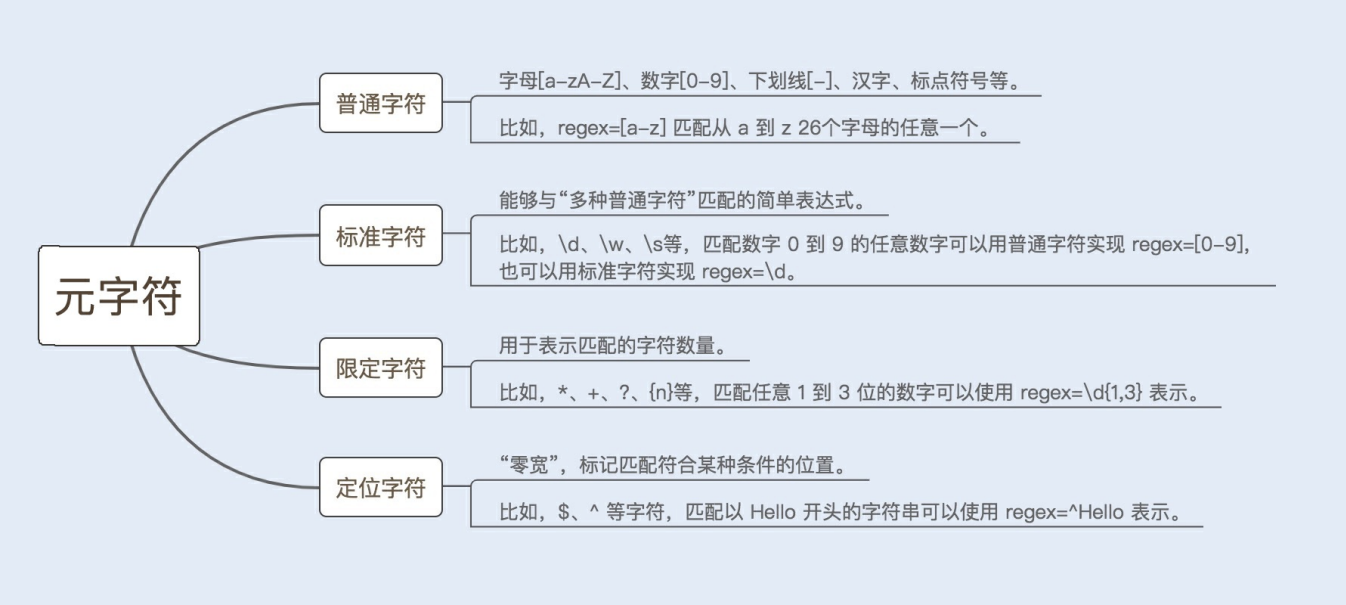
可是一个Split()方法为什么会影响到TPS呢？下面我们就来了解下正则表达式的相关内容，学完了答案也就出来了。

什么是正则表达式？

很基础，这里带你简单回顾一下。

正则表达式是计算机科学的一个概念，很多语言都实现了它。正则表达式使用一些特定的元字符来检索、匹配以及替换符合规则的字符串。

构造正则表达式语法的元字符，由普通字符、标准字符、限定字符（量词）、定位字符（边界字符）组成。详情可见下图：



正则表达式引擎

正则表达式是一个用正则符号写出的公式，程序对这个公式进行语法分析，建立一个语法分析树，再根据这个分析树结合正则表达式的引擎生成执行程序（这个执行程序我们把它称作状态机，也叫状态自动机），用于字符匹配。

而这里的正则表达式引擎就是一套核心算法，用于建立状态机。

目前实现正则表达式引擎的方式有两种：DFA自动机（Deterministic Final Automata 确定有限状态自动机）和NFA自动机（Non deterministic Finite Automaton 非确定有限状态自动机）。

对比来看，构造DFA自动机的代价远大于NFA自动机，但DFA自动机的执行效率高于NFA自动机。

假设一个字符串的长度是n，如果用DFA自动机作为正则表达式引擎，则匹配的时间复杂度为O(n)；如果用NFA自动机作为正则表达式引擎，由于NFA自动机在匹配过程中存在大量的分支和回溯，假设NFA的状态数为s，则该匹配算法的时间复杂度为O（ns）。

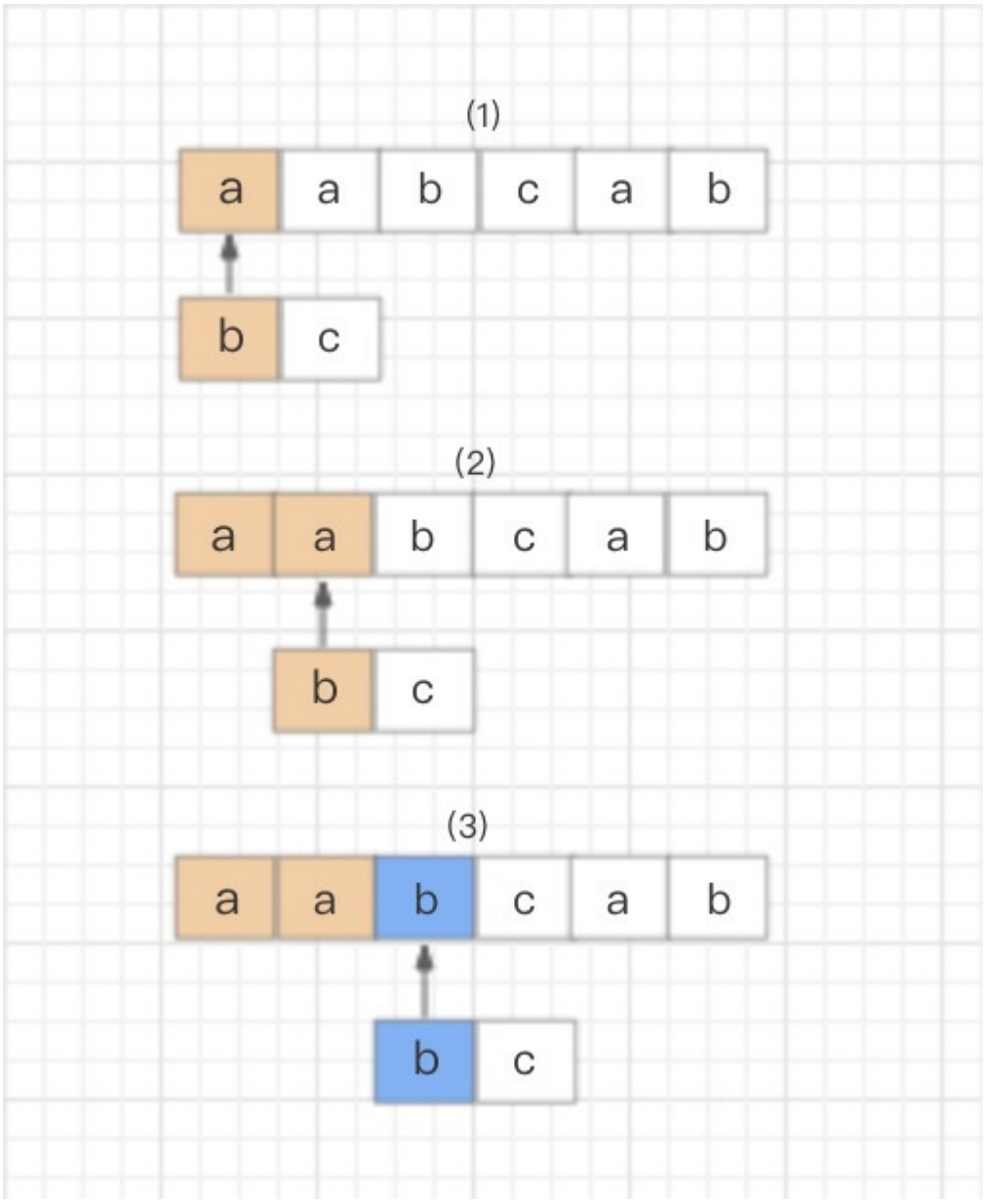
NFA自动机的优势是支持更多功能。例如，捕获group、环视、占有优先量词等高级功能。这些功能都是基于子表达式独立进行匹配，因此在编程语言里，使用的正则表达式库都是基于NFA实现的。

那么NFA自动机到底是怎么进行匹配的呢？我以下面的字符和表达式来举例说明。

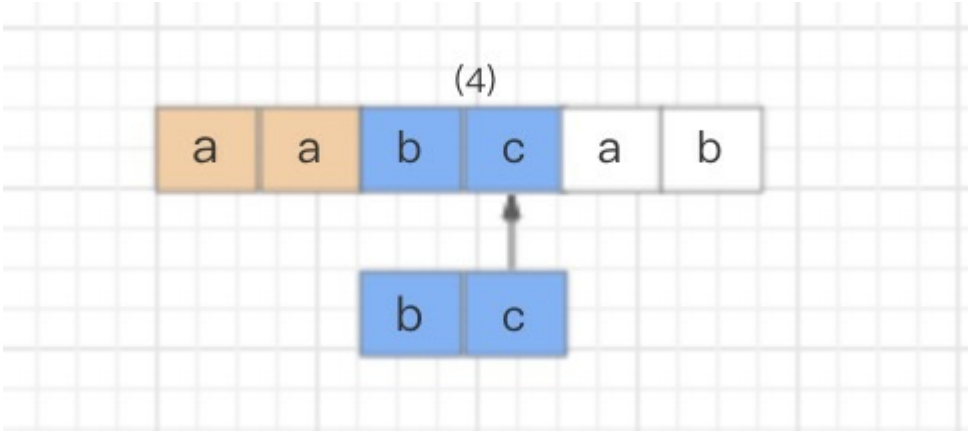
text="aabcab"
regex="bc"

NFA自动机会读取正则表达式的每一个字符，拿去和目标字符串匹配，匹配成功就换正则表达式的下一个字符，反之就继续和目标字符串的下一个字符进行匹配。分解一下过程。

首先，读取正则表达式的第一个匹配符和字符串的第一个字符进行比较，b对a，不匹配；继续换字符串的下一个字符，也是a，不匹配；继续换下一个，是b，匹配。



然后，同理，读取正则表达式的第二个匹配符和字符串的第四个字符进行比较，c对c，匹配；继续读取正则表达式的下一个字符，然而后面已经没有可匹配的字符了，结束。



这就是NFA自动机的匹配过程，虽然在实际应用中，碰到的正则表达式都要比这复杂，但匹配方法是一样的。

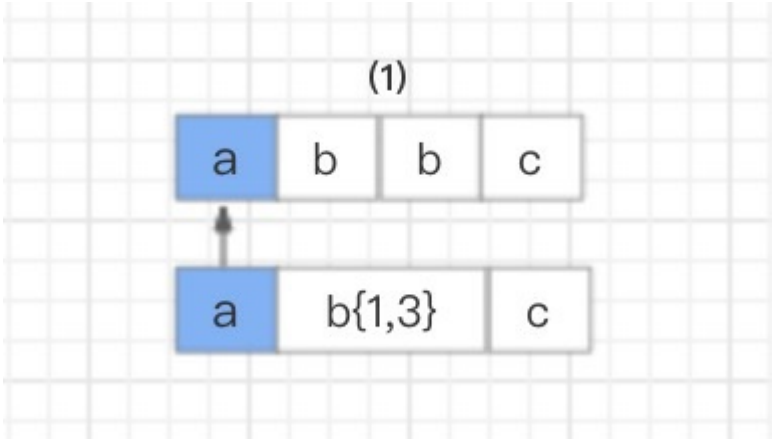
NFA自动机的回溯

用NFA自动机实现的比较复杂的正则表达式，在匹配过程中经常会引起回溯问题。大量的回溯会长时间地占用CPU，从而带来系统性能开销。我来举例说明。

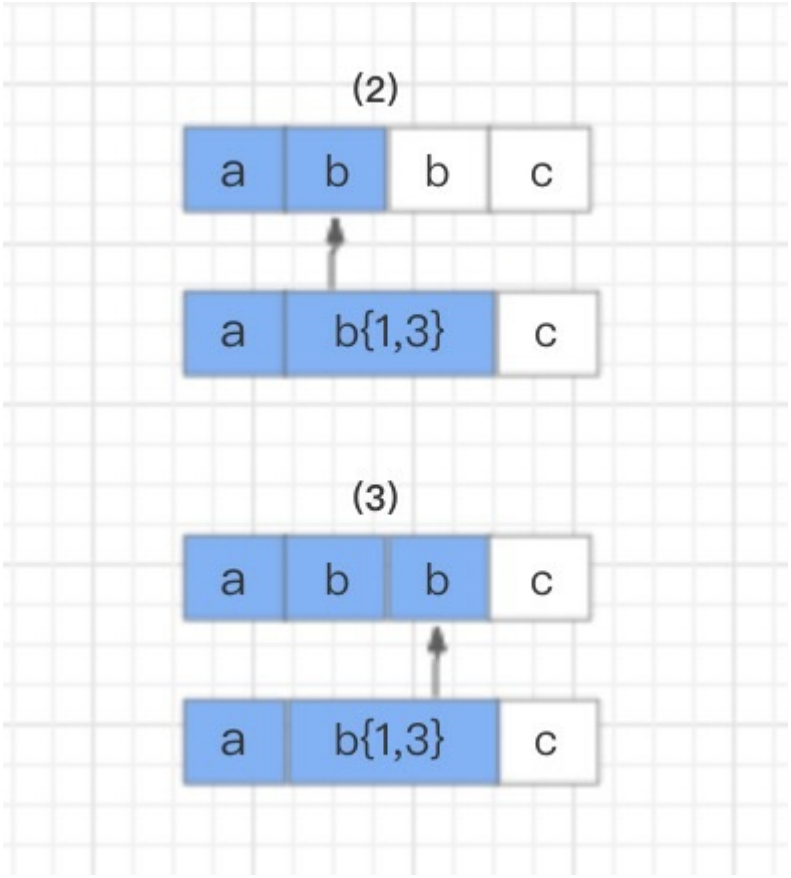
text="abbc"
regex="ab{1,3}c"

这个例子，匹配目的比较简单。匹配以a开头，以c结尾，中间有1-3个b字符的字符串。NFA自动机对其解析的过程是这样的：

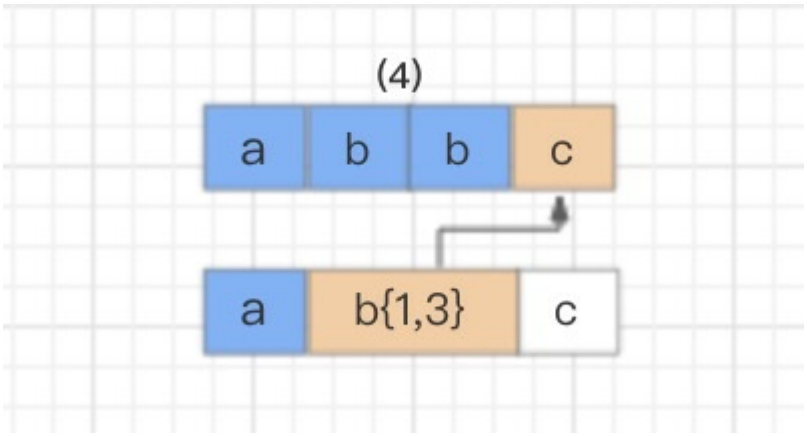
首先，读取正则表达式第一个匹配符a和字符串第一个字符a进行比较，a对a，匹配。



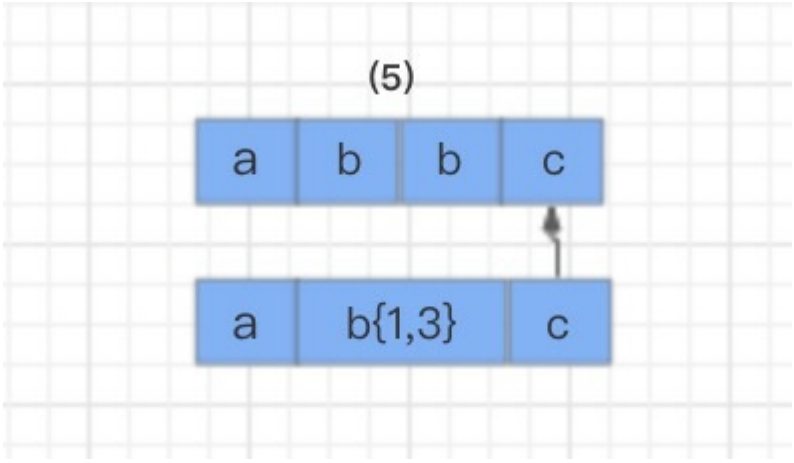
然后，读取正则表达式第二个匹配符b{1,3}和字符串的第二个字符b进行比较，匹配。但因为b{1,3}表示1-3个b字符串，NFA自动机又具有贪婪特性，所以此时不会继续读取正则表达式的下一个匹配符，而是依旧使用 b{1,3}和字符串的第三个字符b进行比较，结果还是匹配。



接着继续使用b{1,3}和字符串的第四个字符c进行比较，发现不匹配了，此时就会发生回溯，已经读取的字符串第四个字符c将被吐出去，指针回到第三个字符b的位置。



那么发生回溯以后，匹配过程怎么继续呢？程序会读取正则表达式的下一个匹配符c，和字符串中的第四个字符c进行比较，结果匹配，结束。



如何避免回溯问题？

既然回溯会给系统带来性能开销，那我们如何应对呢？如果你有仔细看上面那个案例的话，你会发现NFA自动机的贪婪特性就是导火索，这和正则表达式的匹配模式息息相关，一起来了解一下。

1.贪婪模式（Greedy）

顾名思义，就是在数量匹配中，如果单独使用+、？、* 或{min,max} 等量词，正则表达式会匹配尽可能多的内容。

例如，上边那个例子：

text=“abbc”
regex=“ab{1,3}c”

就是在贪婪模式下，NFA自动机读取了最大的匹配范围，即匹配3个b字符。匹配发生了一次失败，就引起了一次回溯。如果匹配结果是“abbbc”，就会匹配成功。

text=“abbbc”
regex=“ab{1,3}c”

2.懒惰模式（Reluctant）

在该模式下，正则表达式会尽可能少地重复匹配字符。如果匹配成功，它会继续匹配剩余的字符串。

例如，在上面例子的字符后面加一个“？”，就可以开启懒惰模式。

text=“abc”
regex=“ab{1,3}?c”

匹配结果是“abc”，该模式下NFA自动机首先选择最小的匹配范围，即匹配1个b字符，因此就避免了回溯问题。

3.独占模式（Possessive）

同贪婪模式一样，独占模式一样会最大限度地匹配更多内容；不同的是，在独占模式下，匹配失败就会结束匹配，不会发生回溯问题。

还是上边的例子，在字符后面加一个“+”，就可以开启独占模式。

text=“abbc”
regex=“ab{1,3}+bc”

结果是不匹配，结束匹配，不会发生回溯问题。讲到这里，你应该非常清楚了，**避免回溯的方法就是：使用懒惰模式和独占模式。**

还有开头那道“一个split()方法为什么会影响到TPS”的存疑，你应该也清楚了吧？

我使用了split()方法提取域名，并检查请求参数是否符合规定。split()在匹配分组时遇到特殊字符产生了大量回溯，我当时是在正则表达式后加了一个需要匹配的字符和“+”，解决了这个问题。

```
\\?(([A-Za-z0-9-~_=%]++\\&{0,1})+)
```

正则表达式的优化

正则表达式带来的性能问题，给我敲了个警钟，在这里我也希望分享给你一些心得。任何一个细节问题，都有可能导致性能问题，而这背后折射出来的是我们对这项技术的了解不够透彻。所以我鼓励你学习性能调优，要掌握方法论，学会透过现象看本质。下面我就总结几种正则表达式的优化方法给你。

1.少用贪婪模式，多用独占模式

贪婪模式会引起回溯问题，我们可以使用独占模式来避免回溯。前面详解过了，这里我就不再解释了。

2.减少分支选择

分支选择类型“(X|Y|Z)”的正则表达式会降低性能，我们在开发的时候要尽量减少使用。如果一定要用，我们可以通过以下几种方式来优化：

首先，我们需要考虑选择的顺序，将比较常用的选择项放在前面，使它们可以较快地被匹配；

其次，我们可以尝试提取共用模式，例如，将“(abcd|abef)”替换为“ab(cd|ef)”，后者匹配速度较快，因为NFA自动机会尝试匹配ab，如果没有找到，就不会再尝试任何选项；

最后，如果是简单的分支选择类型，我们可以用三次index代替“(X|Y|Z)”，如果测试的话，你就会发现三次index的效率要比“(X|Y|Z)”高出一些。

3.减少捕获嵌套

在讲这个方法之前，我先简单介绍下什么是捕获组和非捕获组。

捕获组是指把正则表达式中，子表达式匹配的内容保存到以数字编号或显式命名的数组中，方便后面引用。一般一个()就是一个捕获组，捕获组可以进行嵌套。

非捕获组则是指参与匹配却不进行分组编号的捕获组，其表达式一般由（?:exp）组成。

在正则表达式中，每个捕获组都有一个编号，编号0代表整个匹配到的内容。我们可以看下面的例子：

```
public static void main( String[] args )
{
 String text = "<input high=\"20\" weight=\"70\">test</input>";
 String reg="(<input.*?>)(.*?)(</input>)";
 Pattern p = Pattern.compile(reg);
 Matcher m = p.matcher(text);
 while(m.find()) {
  System.out.println(m.group(0));//整个匹配到的内容
  System.out.println(m.group(1));//(<input.*?>)
  System.out.println(m.group(2));//(.*?)
  System.out.println(m.group(3));//(</input>)
 }
}
```

运行结果：

```
<input high=\"20\" weight=\"70\">test</input>
<input high=\"20\" weight=\"70\">
test
</input>
```

如果你并不需要获取某一个分组内的文本，那么就使用非捕获分组。例如，使用“(?:X)”代替“(X)”，我们再看下面的例子：

```
public static void main( String[] args )
{
 String text = "<input high=\"20\" weight=\"70\">test</input>";
 String reg="(?:<input.*?>)(.*?)(?:</input>)";
 Pattern p = Pattern.compile(reg);
 Matcher m = p.matcher(text);
 while(m.find()) {
  System.out.println(m.group(0));//整个匹配到的内容
  System.out.println(m.group(1));//(.*?)
 }
}
```

运行结果：

```
<input high=\"20\" weight=\"70\">test</input>
test
```

综上可知：减少不需要获取的分组，可以提高正则表达式的性能。

总结

正则表达式虽然小，却有着强大的匹配功能。我们经常用到它，比如，注册页面手机号或邮箱的校验。

但很多时候，我们又会因为它小而忽略它的使用规则，测试用例中又没有覆盖到一些特殊用例，不乏上线就中招的情况发生。

综合我以往的经验来看，如果使用正则表达式能使你的代码简洁方便，那么在做好性能排查的前提下，可以去使用；如果不能，那么正则表达式能不用就不用，以此避免造成更多的性能问题。

思考题

除了Split()方法使用到正则表达式，其实Java还有一些方法也使用了正则表达式去实现一些功能，使我们很容易掉入陷阱。现在就请你想一想JDK里面，还有哪些工具方法用到了正则表达式？

期待在留言区看到你的见解。也欢迎你点击“请朋友读”，把今天的内容分享给身边的朋友，邀请他一起学习。

精选留言

陆离 8

老师｛1，3｝的意思不是最少匹配一次，最多匹配三次吗，独占模式那个例子为什么会不匹配呢？

2019-05-28

作者回复

你好，老师这里更正一下独占模式的例子，落了一个字符。ab{1,3}+bc

2019-05-28

Geek_99fab9 6

我没有你们优秀，我就明白以后少用点正则[]

2019-05-28

编辑回复

不一样的优秀～恭喜你学到了精华！

2019-05-28

Liam 5

文中提供的split性能消耗大的例子：

\?(([A-Za-z0-9-~_=%]+)\&{0,1})$"

一个+ 表示量词，至少1个，不是独占模式吧，这里能否详细解释下优化点在哪里

2019-05-29

作者回复

你好，一个+表示匹配一个或多个，表示尽量多的匹配。我们这个再加一个+，\\?(([A-Za-z0-9-~_=%]++\\&{0,1})+)。提供的这个是没有优化的例子。

2019-05-29

没有小名的曲儿

4

老师，那个(X|Y|Z)三次index是什么意思呢

2019-05-28

作者回复

指的是String中的indexof方法

2019-05-28

K

2

\\?(([A-Za-z0-9-~_=%]++\\&{0,1})+)。老师好，麻烦您讲解一下实际您当时是怎么优化的吗？从哪个正则改成了哪个正则，为什么能有这种优化。谢谢老师。

2019-06-01

作者回复

如果是单个+的情况下，是最大匹配规则，遇到特殊字符串时，会出现回溯问题。这里增加了一个+，变成两个++，变成了独占模式，避免回溯。

2019-06-01

ABC

2

看完明白了回溯是什么意思，我总结如下:

回溯就比如，食堂吃饭，你一下拿了3个馒头。吃完两个，发现第三个不是你想吃的口味的时候，又把第三个放回去，这就造成了资源浪费。

避免的办法就是，一开始就只拿两个，觉得需要了再去继续拿，也就是懒惰模式。

2019-05-30

作者回复

理解很到位，懒惰就是有拿到馒头就走，非常懒，还有馒头拿也不要了。

2019-05-30

WL

2

请问一下老师 "NFA 的状态数"这个概念感觉有点抽象我不太理解, 状态数是什么意思, 是NFA可以匹配的字符串的格式枚举吗?

2019-05-28

作者回复

你好 WL，就是不同的匹配格式，例如 ab{1,2}c，则状态数为2, 即 abc abbc。

2019-05-28

胖

1

字符串替换方法

Replace 普通字符替换

Replaceall 正则替换
一直觉得这两个方法的取名很具有迷惑性

2019-05-28

peace

0

老师你好，
String text = "abbc";
String reg1 = "ab{1,3}+bc";
回溯的例子里，不是b{1,3}匹配c的时候发生回溯 而是b匹配的时候吧

优化的正则 可以举个匹配的例子么 哪种url会产生回溯。。
\\?(([A-Za-z0-9-~_=%]++\\&{0,1})+)

2019-06-12

作者回复

由于是最大匹配，所以匹配text中的c时，发现不匹配，这时会继续下一个匹配b和text中的c，发现不匹配，回溯到text中的b中，匹配成功。是在匹配reg1中的b和text中的c时，不匹配时发生回溯。

参数中包含特殊字符串的链接都可能存在回溯。

2019-06-13

ID171

0

还是上边的例子，在字符后面加一个“+”，就可以开启独占模式。

text=“abbc”
regex=“ab{1,3}+bc”

结果是不匹配，结束匹配，不会发生回溯问题。
这里的每一步做了什么，在最大匹配之后又发生了什么

2019-06-11

作者回复

1、匹配regex中的a和text中的a，匹配成功，继续匹配下一个字符；
2、匹配regex中的b{1,3}+，这个时候是最大匹配规则，也就是说text中会尽量多的去匹配b，直到满足3个b字符匹配成功，才会结束b{1,3}的匹配，这里可以直接匹配到text中的abb；
3、由于还没有满足最大3个的匹配需求，会继续匹配text中的c，发现不匹配，这个时候regex会跳到后面这个字符b，拿这个字符继续匹配；
4、regex中的b发现与text中的c不匹配，则进行回溯，回溯到text中的前一个字符b，发现匹配成功；
5、继续regex的下一个字符c与text中的c字符匹配，匹配成功，匹配结束。

2019-06-13

yu

0

老师，\\?(([A-Za-z0-9-~_=%]++\\&{0,1})+)，这个++的独占模式没看明白，+是一次或多次，所以会尽量尝试多次，第二个+是独占模式不匹配返回失败，那结果不是一直都是失败？？？

2019-06-10

作者回复

如果没有特殊字符的情况下，这个是可以分割成功，如果有特殊字符，也会失败，这也证明了这个链接不合法。这样通过split既判断了链接的合法性，又提取了域名。

2019-06-10

肖韬 0

https://stackoverflow.com/questions/4489551/what-is-double-plus-in-regular-expressions

2019-06-09

ROAD 0

replaceall

2019-06-06

Geek_d93d56 0

我的测试代码

```
long l1 = System.currentTimeMillis();
for (int i = 0; i < 10000000; i++) {
"abc".matches("ab{1,3}c");
}
System.out.println("1111: " + (System.currentTimeMillis()-l1));
long l2 = System.currentTimeMillis();
for (int i = 0; i < 10000000; i++) {
"abc".matches("ab{1,3}?c");
}
System.out.println("2222: " + (System.currentTimeMillis()-l2));
```

输出结果：

1111: 6348

2222: 6329

消耗时间差不多，没有什么差别，请问是正常的吗

2019-06-04

作者回复

因为你的这个例子发生回溯非常少，几乎可以忽略了，可以写个回溯比较长的。

2019-06-05

门窗小二 0

老师，正则表达式中+可以表示量词也可以表达独占，文中出现两个++表示独占，如果正则没有量词需要表达独占该如何写？谢谢

2019-06-02

作者回复

一般* + {n,m}三种会去匹配更多内容，引起回溯，所以一般使用+修饰这三种情况，就会变成独

占模式。所以没有量词的情况下，一般不会引起贪婪匹配更多的情况出现。

2019-06-02

行者 0

老师，我想到Spring框架中在匹配url地址的时候就使用到了正则表达式~

2019-06-01

猫头鹰爱拿铁 0

又复习了一遍正则表达式 对这个理解更加清晰了 贪婪模式会引起回溯 懒惰模式是最小匹配 我理解懒惰模式就是我匹配到了最小的数量我就先匹配下下个字符 所以ab{1,3}?bc匹配字符串abbc 匹配到了第一个b就开始匹配c了 独占模式不会回溯的最大匹配 对于ab{1,3}+bc会先匹配两个b再去匹配c 这个时候发现已经用掉两个b了 而c与b不匹配 所以独占匹配失败

2019-05-31

Fever 0

老师，关于独占我还有点疑惑：

String text = "abbc";

String regex = "ab{1,3}+c";

这里应该也会拉着c一起匹配，最后结果也是true，这个情况和贪婪有什么区别呢？

2019-05-31

作者回复

这个后面解释更正错误了，落了一个字母b，已经更正过了。

String regex ="ab{1,3}+bc"

2019-05-31

goingao 0

@陆离的问题，老师的回答是少了一个字符，看了和原文一样呀？所以少了哪一个呢？

2019-05-31

编辑回复

你好同学，少了的字符已经加上去了，感谢关注！

2019-05-31

Coder4 0

regex=“ab{1,3}?c”

这个是懒惰模式？

2019-05-31

作者回复

对的

2019-05-31