java.util.regex是一个用正则表达式定制的模式来对字符串进行匹配的包.

包括两个类：Pattern和Matcher

一、捕获组的概念

捕获组可以通过从左到右计算其开括号来编号，编号从1开始。例如，在表达式 ((A)(B(C)))中，存在四个这样的组：

1 ((A)(B(C)))

2 (A)

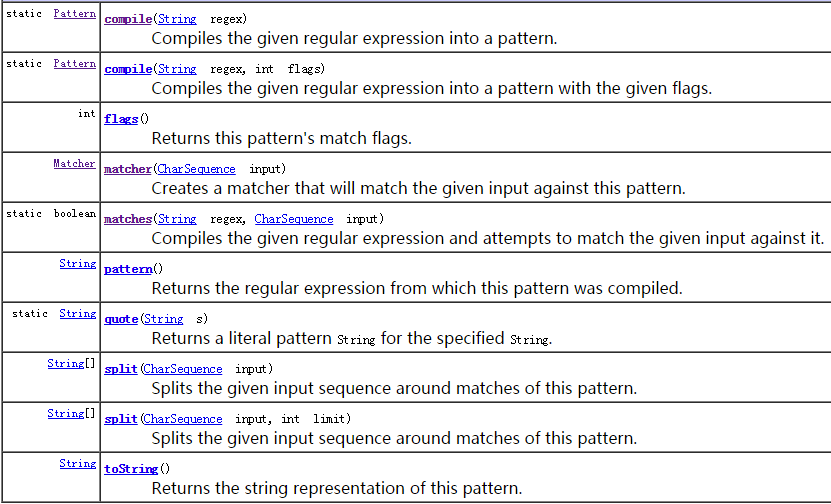
3 (B(C))

4 (C)

组零始终代表整个表达式。以 (?) 开头的组是非捕获组，它不捕获文本，也不针对组合计进行计数。

二、详解Pattern类和Matcher类

java正则表达式通过java.util.regex包下的Pattern类与Matcher类实现。



Pattern类用于创建一个正则表达式匹配模式，它的构造方法是私有的, 不可以直接创建,但可以通过Pattern.complie(String regex) 静态方法创建一个正则表达式模式：

Pattern p = Pattern.compile("\\w+");

p.pattern(); //返回 \w+

pattern() 返回正则表达式的字符串形式，即返回Pattern.complile(String regex)的regex参数

1、split(CharSequence input)

Pattern有一个split(CharSequence input)方法, 用指定的模式分隔字符串, 并返回一个String[], 估计String.split(String regex)就是通过Pattern.split(CharSequence input)来实现的.

// 以数字分隔字符串

Pattern p = Pattern.compile("\\d+");

String[] str=p.split("我的QQ是:456456我的电话是:0532214我的邮箱

是:aaa@aaa.com");

结果:

str[0]="我的QQ是:"

str[1]="我的电话是:"

str[2]="我的邮箱是:aaa@aaa.com"

1. Pattern.matcher(String regex,CharSequence input)

一个静态方法, 检测字符串是否和正则表达式匹配, 该方法需要匹配全部字符串.

Pattern.matches("\\d+","2223"); //返回true

Pattern.matches("\\d+","2223aa"); //返回false,需要匹配到所有字符串才能返回

true, 这里aa不能匹配

3、matcher(CharSequence input)

返回一个Matcher对象.

Matcher类的构造方法也是私有的, 不能随意创建, 只能通过Pattern对象的matcher

(CharSequence input)方法得到该类的实例。Pattern类只能做一些简单的匹配操作, 要想得到更强的正则匹配, 需要Pattern与Matcher一起合作. Matcher类提供了对正则表达式的分组支持, 以及对正则表达式的多次匹配支持.

Pattern p = Pattern.compile("\\d+"); // 将指定的正则表达式编译进Pattern对象

Matcher m = p.matcher("22bb23"); // 用指定的Pattern对象去匹配输入字符串，并返回Matcher对象

m.pattern();//返回p，也就是返回创建该Matcher对象的Pattern对象

4、matches()/ lookingAt()/ find()

Matcher类提供三个匹配操作方法, 三个方法均返回boolean类型, 当匹配到时返回true.

matches()对整个字符串进行匹配, 只有整个字符串都匹配了才返回true

Pattern p = Pattern.compile("\\d+");

Matcher m = p.matcher("22bb23");

m.matches(); //返回false, 因为bb不能被\d+匹配,导致整个字符串匹配未成功.

Matcher m2 = p.matcher("2223");

m2.matches(); //返回true, 因为\d+匹配到了整个字符串

代码Pattern.matcher(String regex,CharSequence input)与下面这段代码等价

Pattern.compile(regex).matcher(input).matches()

lookingAt()对前面的字符串进行匹配, 只有匹配到的字符串在最前面才返回true

Pattern p = Pattern.compile("\\d+");

Matcher m = p.matcher("22bb23");

m.lookingAt(); //返回true,因为\d+匹配到了前面的22

Matcher m2 = p.matcher("aa2223");

m2.lookingAt(); //返回false,因为\d+不能匹配前面的aa

find()对字符串进行匹配, 匹配到的字符串可以在任何位置.

Pattern p=Pattern.compile("\\d+");

Matcher m=p.matcher("22bb23");

m.find(); //返回true

Matcher m2=p.matcher("aa2223");

m2.find(); //返回true

Matcher m3=p.matcher("aa2223bb");

m3.find(); //返回true

Matcher m4=p.matcher("aabb");

m4.find(); //返回false

5、start()/ end()/ group()

当使用matches(), lookingAt(), find( )执行匹配操作后, 就可以利用以上三个方法得到更详细的信息.

start()：返回匹配到的子字符串在字符串中的索引位置.

end()：返回匹配到的子字符串最后一个字符在字符串中的索引位置+ 1

group()：返回匹配到的子字符串

Pattern p = Pattern.compile("\\d+");

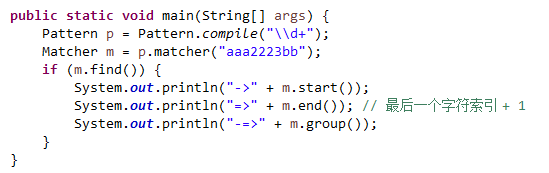
Matcher m = p.matcher("aaa2223bb");

m.find(); //匹配2223

m.start(); //返回3

m.end(); //返回7,返回的是2223后的索引号

m.group(); //返回2223



结果：



Mathcer m2=m.matcher("2223bb");

m.lookingAt(); //匹配2223

m.start(); //返回0, 由于lookingAt()只能匹配前面的字符串,所以当使用lookingAt()

匹配时, start()方法总是返回0

m.end(); //返回4

m.group(); //返回2223

Matcher m3=m.matcher("2223bb");

m.matches(); //匹配整个字符串

m.start(); //返回0,原因相信大家也清楚了

m.end(); //返回6,原因相信大家也清楚了,因为matches()需要匹配所有字符串

m.group(); //返回2223bb

下面说说正则表达式的分组在java中是怎么使用的.

start(), end(), group()均有一个重载方法为：start(int i), end(int i), group(int i)专用于分组操作, Mathcer类还有一个groupCount()用于返回有多少组.

start(int i): 返回第i个分组匹配到的子字符串的索引位置

end(int i): 返回第i个分组匹配到的子字符串最后一个字符的索引位置 + 1

group(int i): 返回第i个分组匹配的子字符串

Pattern p=Pattern.compile("([a-z]+)(\\d+)");

Matcher m=p.matcher("aaa2223bb");

m.find(); //匹配aaa2223

m.groupCount(); //返回2,因为有2组

m.start(1); //返回0 返回第一组匹配到的子字符串在字符串中的索引号

m.start(2); //返回3

m.end(1); //返回3 返回第一组匹配到的子字符串的最后一个字符在字符串中的

索引位置.

m.end(2); //返回7

m.group(1); //返回aaa,返回第一组匹配到的子字符串

m.group(2); //返回2223,返回第二组匹配到的子字符串

现在使用一下稍微高级点的正则匹配操作, 例如有一段文本, 里面有很多数字, 而且这些数字是分开的, 现在要将文本中所有数字都取出来, 利用java的正则操作如下：

Pattern p=Pattern.compile("\\d+");

Matcher m=p.matcher("我的QQ是:456456 我的电话是:0532214 我的邮箱

是:aaa123@aaa.com");

while(m.find()) {

System.out.println(m.group());

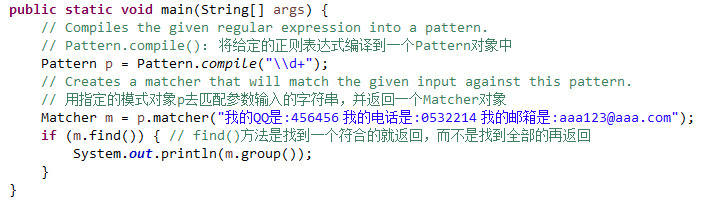
}

输出:

456456

0532214

123



结果：

IMG_256

如将以上while()循环替换成

while(m.find()) {

System.out.println(m.group());

System.out.print("start:"+m.start());

System.out.println(" end:"+m.end());

}

则输出:

456456

start:6 end:12

0532214

start:19 end:26

123

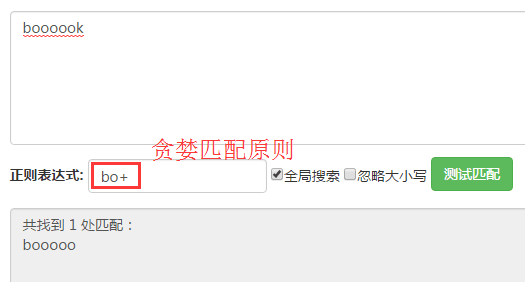
start:36 end:39

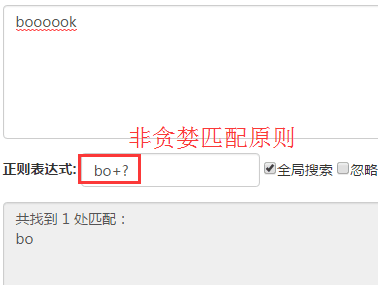
注意:只有当匹配操作成功, 才可以使用start(), end(), group()三个方法,否则会抛出java.lang.IllegalStateException, 也就是当matches(), lookingAt(), find()其中任意一个方法返回true时,才可以使用.

贪婪匹配与非贪婪匹配：

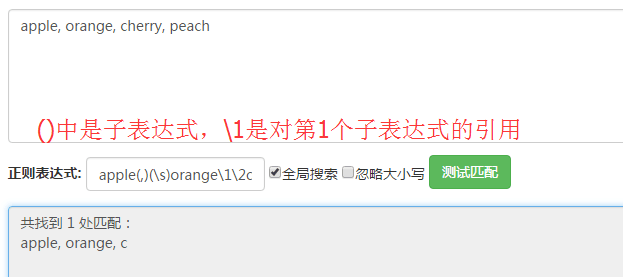
默认情况下，正则表达式使用最长匹配原则，即贪婪匹配原则；

当字符“？”紧跟在任何其他限定符之后时，匹配模式将变成最短匹配原则，即非贪婪匹配原则





子表达式反向引用：



关于Java转义符\:

