Java中的正则表达式和Pattern类Matcher类

1. **正则表达式regular expression简写为regex** ：**正则表达式本身就是一个字符串，用于操作字符串数据，通常通过一些特定的符号来体现的，学习正则表达式就是学习这些特定字符的使用。由于是某些规定特殊意义的字符组成了正则表达式，导致其阅读性比较差。**
2. 正则表达式中**组**的概念：
3. **正则表达式中对于字符或者字符串的复用**利用**小括号**进行封装，可以同时封装成多个组；
4. **组号的划分**：从左向右，以左括号为准，第几个左括号就是为第几组；

**如** ((A)(B(C)))[a-z]+ 有5个组，**一个括号**代表**一个组**，整个表达式本身就是**第零组 ；如：**

**((A)(B(C)))[1,3,4] 中** [**\\0代表**](file:///\\0代表) **((A)(B(C)))[1,3,4] 整个表达式；**

**\\1 代表组1： ((A)(B(C))) ；((A)(B(C)))\\2 代表组2： (A)**

**((A)(B(C)))\\3 代表组3：(B(C)) ； ((A)(B(C)))\\4 代表组4： (C)**

1. 正则表达式中组号用**数字符号**表示，为了和字符**[0-9]**相区分，需要对数字进行转义，即[**\\1，表示第1**](file:///\\1，表示第1)组；
2. 在正则表达式外调用正则表达式的组方法是： **$组号。注意：这里的组号不需要转义，如$0、$1、$2等。**

**例子：**

String str = new String("sssshannnndongggggkejiiiiidaaaaaxueeee");

System.out.println(str.replaceAll(**"(.)**

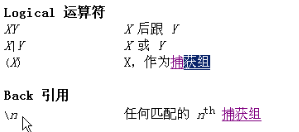
**[\\1+","$1"));](\\\\1+\",\"$1\")); //)**

**[//](\\\\1+\",\"$1\")); //)输出为shandongkejidaxue**

System.out.println(str.replaceAll("(.)\\1+","$0"));

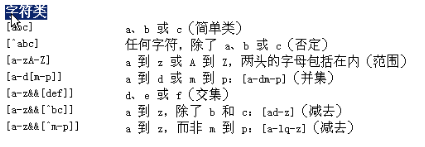
//**输出为sssshannnndongggggkejiiiiidaaaaaxueeee**

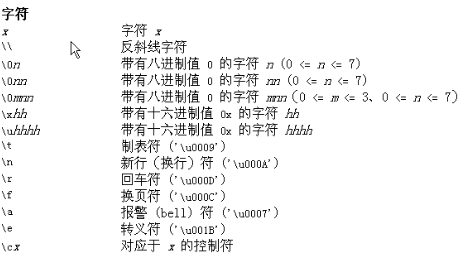
**解析：这里$0代表仍然是整个正则表达式，相当于找出满足的后，然后还是用自己本身替换，等于没有替换。**

1. 
2. **正则表达式的符号集合**：

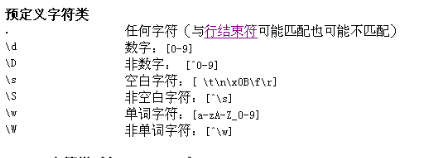
**限定字符的范围用中括号[]括起来，限定字符个数用大括号{}括起来。**

1. **字符类：**





1. **预定义字符类**：
2. **.** 表示**任意字符;**
3. \d 代表**数字[0-9]**，\D代表**非数字[^0-9];**
4. **\w**代表**大小写字母、数字、下划线[a-zA-Z\_0-9];**



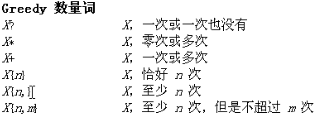
1. **数量词：**

**\*表示任意个；?表示0或1；+表示大于或者等于1；{n}表示n个；{n,} 表示大于等于n个；{n,m} 表示n到m个，闭合区间包括n也包括m。**

**注意：\*和{0，}等价；？和{0,1}等价；+和{1,}等价。**

**如String str = new String("aaaa");**

**System.out.println(str.matches("a{4,4}"));//返回true**

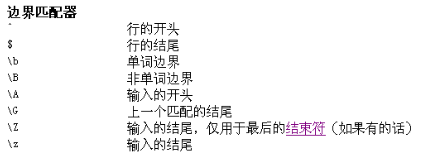


1. **边界匹配器：**

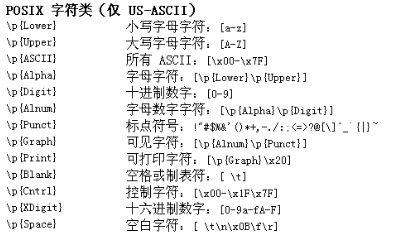
**^ 表示行的开头，$表示行的结尾；**

**\b表示单词边界，\B表示非单词边界；**

\A表示输入的开头，\z表示输入的结尾；



1. POSIX字符类（仅 US-ASCII）



1. **转义的问题**：
2. **正则表达式所需要的 \ ，必须输入两个 \\ 。**

由于系统已经定义了一些转义字符的特殊含义，例如制表符\t 、换行符\t\n 等，直接输出这些转义字符就是表示的对应意义。但是，区分开来，在正则表达式中定义了**新的规定**，正则表达式见到了相应的转义字符后，自己进行翻译，如 遇到**\d** 翻译成数字，但是系统可能已经给\d 赋予了意义，这样只写\d不是表示\和d的组合了，而是表示**赋予的意义**，所以需要对**反斜线**进行转义，让其只是表示**一个特殊的字符\\**，只是个字符，不会对d进行转义，从而对于正则表达式，需要输入[\\d](file:///\\d) 才可以认为是 \d,然后正则的规则在进行翻译成数字意义。对于 **.** 也是一样，. 本身就有任意字符的意义，需要进行转义，正则需要有**\.**；但是\ 需要输入两个 \\ 才可以。

1. 想要输出反斜线需要转义一下，[\\两个反斜线输出的就是一个反斜线](file:///\\两个反斜线输出的就是一个反斜线)。

例如 [0-9]{9} 可以利用 \\d{9} 代替。

理解： \d 相当于反斜线 \ 把d转义成了 **数字** 意义，\ 本身就是一种转义的功能，而想要输出一个 \ ，而后面又带有一些字符会使其成为一个转义字符时，需要利用 \ 把自己给转义了，两个反斜线\\就是代表了一个\。

1. **在字符串切割时，如果想以 . 为标志进行切割**，此时注意了，因为单独的 . 已经赋予任意字符的含义，所以需要读 . 进行转义，怎么转义呢？只需要在 . 前面加上一个反斜线即可，可是字符串中 需要输出两个反斜线才表示一个反斜线。

例如：String str = new String("zhaohong.lisi.zhusan.lihongzhang");

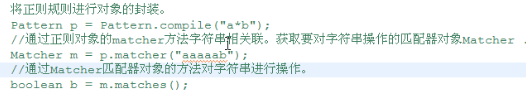
**String[] names = str.split("\\.");**

1. **正则表达式对字符串的常见操作**：
2. **匹配：**

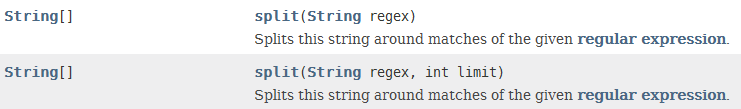
**使用的就是String类的matches方法：返回boolean值。**



String类的matches方法本质上就是调用的是**匹配器中的matches方法**：

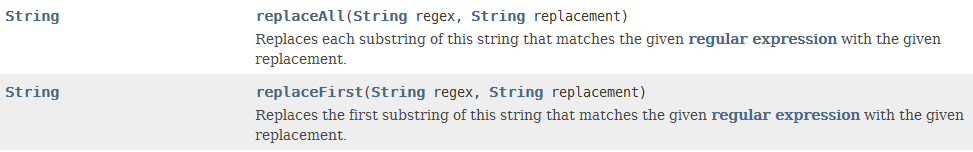


1. **切割：**
   1. **使用的就是String类中的spilt方法：返回字符串数组。**



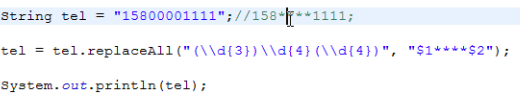
* 1. **利用 组 进行分割**：见上面组的介绍部分。

1. **替换：**
   1. **使用的就是String类中的replace方法：**



* 1. 利用 **$** 可以获取前面指定的正则表达式的**组**。

手机号码：



String str = new String("zhaohongtttttlisieezhusanrrrlihongmmzhangsi");

System.out.println(str.replaceAll("(.)\\1{2}","$1"));**//把重复的字符，变成相应的单个字符**

1. **获取： 需要利用先对正则表达式进行封装成Pattern类的对象，然后调用Pattern中的matcher方法与字符串相关联，返回一个匹配器Matcher对象，再调用Matcher中的group方法进行获取匹配的字符串。如果没有相应匹配的字符，可以返回空字符串。**

**例子：**

**String str = new String("wang101nv,li312nan,zhao233,qian343nv,sun212");**

**//将正则表达式封装成对象**

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

**//与字符串相关联，从而获得匹配器对象**

**Matcher m = p.matcher(str);**

**//获取正则表达式中的组的个数（不包括组0）**

**System.out.println(m.groupCount());**

**//获取每次匹配的相应组的结果**

**while(m.find()) {**

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

**System.out.println(m.group(0));**

**System.out.println(m.group(1));**

**System.out.println(m.group(2));**

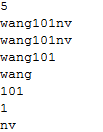
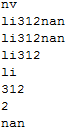
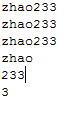
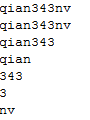
**System.out.println(m.group(3));**

**System.out.println(m.group(4));**

**System.out.println(m.group(5));**

**}**

**结果如下：**

1. **注意：在Matter调用group方法之前，必须执行find方法，否则异常报出。**
2. **上述四个应用：匹配、切割、替换分别利用了String类已经封装好的方法，至于获取这个应用必须使用到正则表达式自己封装的类Pattern。其实，String类封装好的matches、replaceAll、split等方法底层都是依靠Pattern类实现的。**
3. **Pattern类和Matcher类的关系及String类对功能进行了封装：**

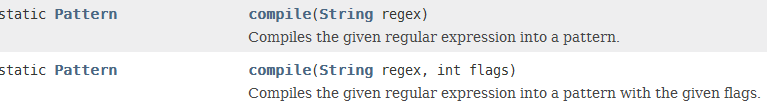
**Pattern类完成的功能主要是实现对正则表达式的对象封装，通过Pattern类的matcher方法获取一个Matcher对象（称为匹配器），Matcher类是本质上就是完成正则表达式对字符串的操作，如提供了find方法、matches方法、lookingAt方法、group方法等方法。对于正则表达式的四大功能，匹配、切割、替换、获取；String类已经对前三个功能进行了封装方法分别为matches、split、replaceAll方法，而要完成获取功能，只能使用Matcher类的find和group方法（一起使用）。Pattern类和Matcher类都采取了静态工厂模式，Pattern对象需要通过自己的静态方法compile方法获取，而Matcher对象通过Pattern对象的matcher方法获取。**

1. **Pattern类：存在于java.util.regex包中，直接继承于java.lang.Object类；**

**Pattern类是正则表达式的编译表现形式。Pattern类是final类，是不可被继承的。对外不提供构造方法，采用静态工厂模式，即单例设计模式。提供静态方法compile方法创建Pattern类对象。Pattern类对象可以利用其matcher方法创建Matcher对象，根据正则表达式，从而这个Matcher对象就可以与任意一个字符串序列匹配。**

**Pattern类只是把正则表达式封装成对象，而具体操作还需要Matcher对象。**

1. **Pattern类的方法：**
2. **静态工厂方法:compile ,参数是正则表达式，返回一个Pattern类对象。**



1. **matches方法：参数为任意一个需要操作的字符序列或字符串，返回的是一个Matcher对象。**

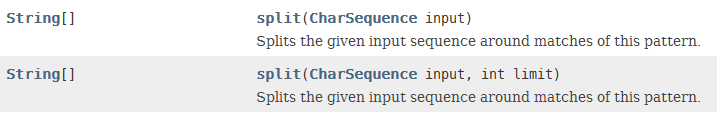


**另外一个，不需要对正则表达式显式地封装成对象，直接利用Pattern类名调用静态方法matches方法，参数为正则表达式和任意字符串，返回boolean值，从而判断是否匹配。这个方法缺点是不能够实现正则表达式Pattern对象的复用。**



**如：System.out.println(Pattern.matches("\\d{11}","15764250250"));//true**

1. **spilt方法：**





1. **pattern（）方法：返回该Pattern类的正则表达式（字符串）。**



1. **quote（String s）方法：静态的，返回一个字符串。**



1. **flags（）方法：**



1. **Matcher类：存在于java.util.regex包中，直接继承于java.lang.Object类。是final修饰的，是不可以被继承的。对外不提供构造方法即不可以通过new的方式创建对象，Pattern类中有方法可以创建Matcher对象，Matcher类自身也有很多方法可以创建Matcher对象，但是不是静态的。**

**Pattern类中创建Matcher对象的方法：**



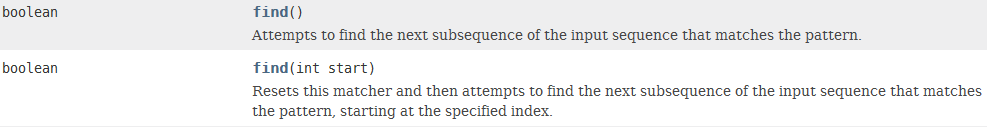
**从创建Matcher对象典例：**

**Pattern p = Pattern.compile("[1-9]{5}");//首先需要创建Pattern对象**

**String str = new String("nihaoma123225hao");**

**Matcher m = p.matcher(str);//利用Pattern类的matcher方法，返回一个Matcher对象**

1. **Matcher类中的方法：**
2. **find（）方法：功能是寻找指定的字符串中是否存在符合正则表达式指定的子字符串，返回boolean值。**



1. **matches()方法：功能是判断是否匹配。String类的matches方法调用的就是此方法。**



1. **lookingAt（）方法:功能是判断字符串的前缀是否符合正则表达式的规则，只要前缀部分符合，就返回true，否则返回false。相当于字符串的前缀matches方法。**





1. **返回正则表达式中组的个数，注意：不包括组0。**



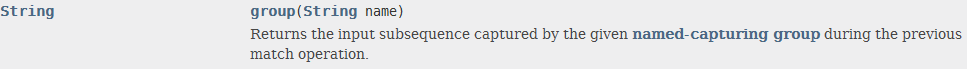
1. **group方法：返回的都是上次已经匹配的字符串或者已经匹配的特定组的字符串，与find方法搭配使用，即可以实现连续获取的功能。连续调用find方法，可以连续判断出匹配的子字符串。如果没有相应匹配的字符，可以返回空字符串。**
2. **首先会检查是否与正则表达式匹配，如果匹配，则返回上次匹配结果中所匹配的字符串；若不匹配则抛出异常IllegalStateException。与find方法搭配使用。**



**b） 首先会检查是否与正则表达式匹配，如果匹配，则返回上次匹配结果中指定组所捕获的子字符串，如果不匹配，抛出异常。与find方法搭配使用。**



**c） 正则表达式中每个组有自己的组名，通过指定组名从而返回上次匹配结果中此组所捕获的子字符串。与find方法搭配使用。**



1. **region（int start ，int end ）方法：返回一个新的匹配器对象；即把原来关联的一部分字符串重新关联成一个新的匹配器对象。**



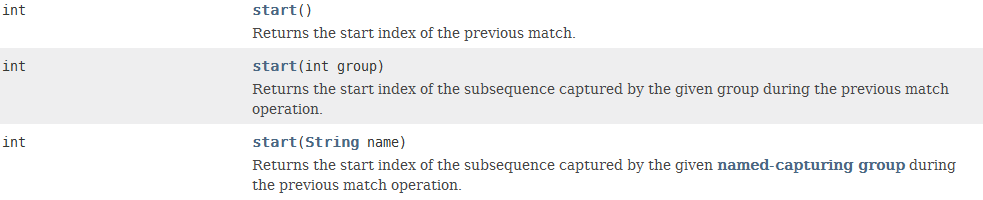
1. **reset（）重置：把关联的字符串取消掉，返回一个新的匹配器对象。**



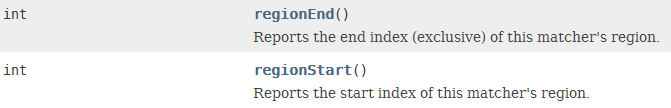
**重新关联新的字符串，返回一个新的匹配器对象。**



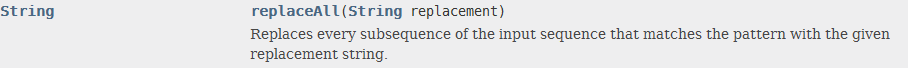
1. **获取位置索引的方法：**
   1. **start方法：返回上次匹配的字符串的索引位置，也就是相应group方法返回字符串的首个字符的索引。**

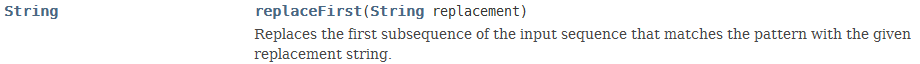


* 1. **regionEnd和regionStart方法：**



1. **replaceAll方法和replaceFirst方法：**





1. **判断方法：是否需要添加字符以达到匹配的需求。**



1. **练习题：**
2. 把**名字**从字符串以**重叠字符**为界切割开来。

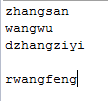
**分析**：切割当然用String的**spilt方法**，重叠字符是任意的，所以只能用**点.**表示，但是又要需要表示出此字符重复出现，只能使用**组**的概念，则正则表达式应该为**"(.)**[**\\1**](file:///\\1)**+"或者"(.)\\1{1,}"**。（注意：**"\\.{2,}"** **这种正则表达式是表示两个及两个以上任意字符出现，没有重复性**）

String str = new String("zhangsanttwangwudddzhangziyirrrrrwangfeng");

String[] names1 = str.split("(.)\\1");//表示一个字符出现两次，[\\1表示调用组1](file:///\\1表示调用组1)一次

String[] names2 = str.split("(.)\\1+");//表示一个字符出现**两次之上**，[\\1表示调用组1](file:///\\1表示调用组1)，+号表示组1次数大于等于1次，加上前面的括号组，相当于出现两次之上

names1结果为：



names2结果为：



1. 把**重叠出现的字符**替换成**单个此字符**。

思路：替换且是全部替换，故选用String类的replaceAll方法。既然是替换，首先是寻找出重叠字符，由于重叠字符是任意的，且个数是大于等于2即可，故只能使用**组的概念**。正则表达式为**"(.)**[**\\1**](file:///\\1)**+"或"(.)\\1{1,}" 。**

**代码如下：**

**String str = new String("sssshannnndongggggkejiiiiidaaaaaxueeee");**

**System.*out*.println(str.replaceAll("(.)\\1+","$1"));**

**输出为：**

1. 治口吃：

思路：首先需要把点去掉，利用replaceAll方法，用空字符串替换；

其次把剩下的重叠字符换成单个字符即可。

String str = new

String("我我......我我...要要....学学学学学.....习....编编...程！");

str = str.replaceAll("\\.","");//去除.

str = str.replaceAll("(.)\\1+","$1");**// 我要学习编程**！

1. 对**邮件地址**进行校验，返回boolean值即可。

分析：掌握一些预定义字符的使用，如**\w**等，还有非什么字符之外的使用方法，及组的灵活使用。

String str = "zhaohong@163.com.cn";

boolean b = str.matches("\\w+@[^\_]+(\\.[^1-9\_]{1,3})\\1\*");

1. 对一连串**IP地址**进行排序：
2. 首先把字符串切割成一个个单独的IP地址：注意首先把字符串的首尾空格去除利用trim方法，其次根据空格（任意多个）切割字符串；
3. 对IP地址的每个字段补2个0，利用替换的方法；
4. 去除每个字段多于3个数字的多于前缀0，利用替换的方法；
5. 把这些个IP地址**字符串存放到TreeSet集合中，使其自动排序；**
6. **然后把这些字符串取出；**
7. **把字段的前缀0去除，利用替换方法；**
8. **最后合成一个字符串。**

代码如下：

String ip\_str = new String(" 192.168.1.101 172.16.1.19 20.50.0.2 0.1.3.9 ");

ip\_str = ip\_str.trim();

System.out.println(ip\_str);

String[] ips = ip\_str.split(" +");

TreeSet<String> ts = new TreeSet<String>();

for (int i = 0; i < ips.length; i++) {

ips[i] = ips[i].replaceAll("\\d+","00$0");

ips[i] = ips[i].replaceAll("0+(\\d{3})","$1");

ts.add(ips[i]);

}

for (int i = 0; i < ips.length; i++) {

System.out.println(ips[i]);

}

StringBuffer sb = new StringBuffer();

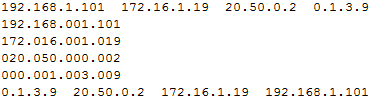
for (String ip : ts) {

sb.append(ip.replaceAll("(0\*(\\d+))","$2")+" ");

}

System.out.println(sb);

**输出结果**：



1. 把手机号**中间四位**遮挡住。

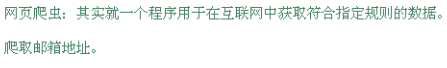
**String num = new String("15764250250");**

**String numex = num.replaceAll("((1[358]**[**\\d)(\\d{4})(\\d{4}))","$2\*\*\*\*$4")**](file:///\\d)(\\d%7b4%7d)(\\d%7b4%7d))%22,%22$2****$4%22))**;**

**结果为：157\*\*\*\*0250**

1. **网页爬虫**：

网络爬虫（又被称为网页蜘蛛，网络机器人，在[FOAF](http://baike.baidu.com/view/271451.htm)社区中间，更经常的称为网页追逐者），是一种按照一定的规则，自动地抓取[万维网](http://baike.baidu.com/view/7833.htm)信息的程序或者脚本。另外一些不常使用的名字还有蚂蚁、自动索引、模拟程序或者[蠕虫](http://baike.baidu.com/view/2596.htm)







从网络上读取数据：改变方法：

