最近要在项目中做HTML解析,本想使用NekoHTML来解析,但看了下NekoHTML有点复杂,故采用了一种便捷,笨的方式来完成项目的要求. 正则表达式此时发挥了她巨大的威力,为我排忧解难,很快便完成了HTML解析工作.

在解析期间,研究了很多java正则表达式使用方法,颇得一些经验,不敢独享,愿与大家共享.

java正则表达式通过java.util.regex包下的Pattern类与Matcher类实现(建议在阅读本文时,打开java API文档,当介绍到哪个方法时,查看java API中的方法说明,效果会更佳).

**Pattern类**

用于创建一个正则表达式,也可以说创建一个匹配模式,它的构造方法是私有的,不可以直接创建,但可以

**1、通过Pattern.complie(String regex)简单工厂方法创建一个正则表达式**

Java代码示例:

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

p.pattern();//返回 /w+

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

1.Pattern.split(CharSequence input)

Pattern有一个split(CharSequence input)方法,用于分隔字符串,并返回一个String[],我猜String.split(String regex)就是通过Pattern.split(CharSequence input)来实现的.

Java代码示例:

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"

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

一个静态方法,用于快速匹配字符串,该方法适合用于只匹配一次,且匹配全部字符串.

Java代码示例:

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不能匹配到

Pattern.matches("//d+","22bb23");//返回false,需要匹配到所有字符串才能返回true,这里bb不能匹配到

3.Pattern.matcher(CharSequence input)

**Matcher类**

**Pattern.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.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,要想得到更强更便捷的正则匹配操作,那就需要将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.

Java代码示例:

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

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

m.pattern();//返回p 也就是返回该Matcher对象是由哪个Pattern对象的创建的

4.Matcher.matches() / Matcher.lookingAt() / Matcher.find()

Matcher类提供三个方法

三个匹配操作方法：均返回boolean类型,当匹配到时返回true,没匹配到则返回false

**1、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**

**Java代码示例:**

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

我们现在回头看一下Pattern.matcher(String regex,CharSequence input),它与下面这段代码等价

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

**2、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**

Java代码示例:

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

**3、find()对字符串进行匹配,匹配到的字符串可以在任何位置**

Java代码示例:

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

**4.Mathcer.start() / Matcher.end() / Matcher.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的最后一个字符在字符串中的索引位置.

**group()**返回匹配到的子字符串

**Java代码示例:**

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

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

start(),end(),group()均有一个重载方法：start(int i),end(int i),group(int i)专用于分组操作

**groupCount()**：用于返回有多少组**（Mathcer类）**

**Java代码示例:**

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

现在我们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,我们现在要将文本中所有数字都取出来,利用java的正则操作是那么的简单.

**Java代码示例:**

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

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：**

**1、**每次执行匹配操作后start(),end(),group()三个方法的值都会改变,改变成匹配到的子字符串的信息,以及它们的重载方法,也会改变成相应的信息.

**2、**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.

**5.Matcher.replaceAll(String replacement) / Matcher.replaceFirst(String replacement)**

String.replaceAll()和String.replaceFirst()两个方法的功能

与Matcher.replaceAll()和Matcher.replaceFirst()的功能是一样的,只不过是使用方式不一样

例如我要将某文本中的所有数字变成\*

使用String完成该要求

**Java代码示例:**

String str="我的QQ是:456456 我的电话是:0532214 我的邮箱是:aaa123@aaa.com";

System.out.println(str.replaceAll("//d","\*"));

输出: 我的QQ是:\*\*\*\*\*\* 我的电话是:\*\*\*\*\*\*\* 我的邮箱是:aaa\*\*\*@aaa.com

现在我们用Matcher完成该要求

Java代码示例:

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");

System.out.println(m.replaceAll("\*"));

输出: 我的QQ是:\*\*\*\*\*\* 我的电话是:\*\*\*\*\*\*\* 我的邮箱是:aaa\*\*\*@aaa.com

String.replaceAll()应该是调用了Matcher.replaceAll(),String.replaceAll()与下面这段代码等价

Pattern.compile(regex).matcher(str).replaceAll(replacement)

至于Matcher.replaceFirst()也很简单,它与String.replaceFirst()功能一样,我就不多说了.

str.replaceFirst(regex, replacement)与下面这段代码等价

Pattern.compile(regex).matcher(str).replaceFirst(replacement)

\*\*\*\*public String replaceFirst(String regex,String replacement)使用给定的 replacement 替换此字符串匹配给定的正则表达式的第一个子字符串\*\*\*\*

**6.Matcher.appendReplacement(StringBuffer sb, String replacement)**

**/ Matcher.appendTail(StringBuffer sb)**

将当前匹配子串替换为指定字符串，并且将替换后的子串以及其之前到上次匹配子串之后的字符串段添加到一个StringBuffer对象里，而appendTai­l(StringBuffer sb) 方法则将最后一次匹配工作后剩余的字符串添加到一个StringBuffer对象里．看例子：

**Java代码示例:**

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");

StringBuffer sb=new StringBuffer();

m.find(); //匹配到456456

m.appendReplacement(sb,"\*"); //将456456之前的字符串追加到sb,再将456456替换为\*,并追加到sb

System.out.println(sb.toString());

m.appendTail(sb); //将前面替换过的内容连接后面未替换过的内容,并放入sb

System.out.println(sb.toString());

输出：

我的QQ是:\*

我的QQ是:\* 我的电话是:0532214 我的邮箱是:aaa123@aaa.com

再看一个例子

**Java代码示例:**

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");

StringBuffer sb=new StringBuffer();

while(m.find()) {

m.appendReplacement(sb,"\*");

System.out.println(sb.toString());

}

m.appendTail(sb);

System.out.println("使用appendTail()的最终内容是："+sb.toString());

输出：

我的QQ是:\*

我的QQ是:\* 我的电话是:\*

我的QQ是:\* 我的电话是:\* 我的邮箱是:aaa\*

使用appendTail()的最终内容是：我的QQ是:\* 我的电话是:\* 我的邮箱是:aaa\*@aaa.com

关于这两个方法就介绍到这,如果不明白的话,还需要自己动下手,认真体会一下其内涵.

**7.Matcher.region(int start, int end) / Matcher.regionEnd() / Matcher.regionStart()**

我们在做匹配操作时,默认去匹配的是整个字符串,例如有一字符串"aabbcc",使用"//d+"去find()时,是从第一个a开始匹配,也就是索引号为0的位置,开始去匹配,当索引号为0的位置没有匹配到时,就去下一个位置去匹配...直到匹配到子字符串或匹配完最后一个字符索引号才结束,很显然"//d+"不能匹配"aabbcc",当它匹配完最后一个c时,结束本次匹配,宣告匹配失败,也就是说它会去匹配完整个字符串,能不能不去匹配完整个字符串呢,答案是可以的.

**region(int start,int end)**就是用来设置此匹配器的区域限制。

先来看一个例子.

**Java代码示例:**

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

String content="aaabb2233cc";

Matcher m=p.matcher(content);

System.out.println(m);

输出: java.util.regex.Matcher[pattern=/d+ region=0,11 lastmatch=]

可以看到region=0,11 表示start=0,end=11,更通俗的说就是当去匹配字符串,先从索引号为0的位置去匹配,如果匹配到了子字符串就返回,如果没有匹配到则到下一个位置去匹配,一直匹配到索引号为11的字符就结束匹配.

为什么是11呢,因为content.length()==11

现在你应该明白了它的作用,来看一个例子.

**Java代码示例:**

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

String content="aaabb2233cc";

Matcher m=p.matcher(content);

m.find(); //匹配到2223,返回true

Matcher m2=p.matcher(content);

m2.region(0,5);

m2.find(); //返回false,只去匹配索引号0至5-1的字符,没有匹配到

Matcher m3=p.matcher(content);

m2.region(3,8);

m2.find(); //返回true

m2.group(); //返回223,为什么,请数一下索引号就知道了.

**Matcher.regionStart()**返回region(int start,int end)中的start值,默认为0

**Matcher.regionEnd()**返回region(int start,int end)中的end值,默认为去匹配字符串的length()值

**8.Matcher.reset() / Matcher.reset(CharSequence input)**

用于重置匹配器。看示例

Java代码示例:

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

String content="aaabb2233cc";

Matcher m=p.matcher(content); //此时m刚创建出来,为最初状态

m.find();

m.group(); //返回aaabb

m.find();

m.group(); //返回cc

Matcher m2=p.matcher(content); //此时m2刚创建出来,为最初状态

m.find();

m.group(); //返回aaabb

m.reset(); //恢复到了最初状态,此时相当于m2刚创建出来

m.find();

m.group(); //返回aaabb,相信大家应该知道了吧

Matcher.reset(CharSequence input)

恢复到最初状态,并将匹配字符串换成input,以后执行匹配操作时,就来匹配input,而不匹配原来的字符串了

**9.Matcher.toMatchResult()**

**（**用于保存某次匹配后的信息,待以后再使用.**）**

大家查看一下java API 对Matcher类的说明,会发现它实现了MatchResult 接口,这个接口只有以下几个方法

groupCount()

group() / group(int i)

start() / start(int i)

end() / end(int i)

至于这几个方法的功能前面已经介绍过,现在我们来看一下toMatchResult() 是如何使用的

**Java代码示例:**

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");

List list=new ArrayList();

while(m.find()) {

list.add(m.toMatchResult());

}

MatchResult matchResult=null;

Iterator it=list.iterator();

int i=1;

while(it.hasNext()) {

matchResult=(MatchResult)it.next();

System.out.print("第"+(i++)+"次匹配到的信息: ");

System.out.println(matchResult.group()+"/t/t"+matchResult.start()+"/t"+matchResult.end());

}

输出:

第1次匹配到的信息: 456456 6 12

第2次匹配到的信息: 0532214 19 26

第3次匹配到的信息: 123 36 39

**toMatchResult()**用于保存某次匹配后的信息,待以后再使用.

方法使用就说到这里,现在再介绍一个实例

有这样一个需求,有一个HTML文件,需要将其中的内容抽取出来,并不带HTML标签,如果使用正则表达式,这是一件很容易的事情. 前提是这个HTML文件只保留了<body></body>标签以内的内容.

Java代码示例:

String html="<div><font color='red'>example1</font></div>"; //可以是任何html文件源代码,但格式一定要正确

Pattern p=Pattern.compile("<[^>]\*>");

Matcher m=p.matcher(html);

String result=m.replaceAll("");

System.out.println(result);

输出:example1

参考资料：

java.util.regex的API文档

陈广佳的JAVA正则表达式--Pattern和Matcher