**Matcher类的方法**

索引方法

索引方法提供了有用的索引值，精确表明输入字符串中在哪能找到匹配：

序号 方法及说明

1 public int start()

返回以前匹配的初始索引。

2 public int start(int group)

返回在以前的匹配操作期间，由给定组所捕获的子序列的初始索引

3 public int end()

返回最后匹配字符之后的偏移量。

4 public int end(int group)

返回在以前的匹配操作期间，由给定组所捕获子序列的最后字符之后的偏移量。

查找方法

查找方法用来检查输入字符串并返回一个布尔值，表示是否找到该模式：

序号 方法及说明

1 public boolean lookingAt()

尝试将从区域开头开始的输入序列与该模式匹配。

2 public boolean find()

尝试查找与该模式匹配的输入序列的下一个子序列。

3 public boolean find(int start）

重置此匹配器，然后尝试查找匹配该模式、从指定索引开始的输入序列的下一个子序列。

4 public boolean matches()

尝试将整个区域与模式匹配。

替换方法

替换方法是替换输入字符串里文本的方法：

序号 方法及说明

1 public Matcher appendReplacement(StringBuffer sb, String replacement)

实现非终端添加和替换步骤。

2 public StringBuffer appendTail(StringBuffer sb)

实现终端添加和替换步骤。

3 public String replaceAll(String replacement)

替换模式与给定替换字符串相匹配的输入序列的每个子序列。

4 public String replaceFirst(String replacement)

替换模式与给定替换字符串匹配的输入序列的第一个子序列。

5 public static String quoteReplacement(String s)

返回指定字符串的字面替换字符串。这个方法返回一个字符串，就像传递给Matcher类的appendReplacement 方法一个字面字符串一样工作。

start 和 end 方法

下面是一个对单词 "cat" 出现在输入字符串中出现次数进行计数的例子：

RegexMatches.java 文件代码：

import java.util.regex.Matcher;

import java.util.regex.Pattern;

public class RegexMatches

{

private static final String REGEX = "\\bcat\\b";

private static final String INPUT =

"cat cat cat cattie cat";

public static void main( String[] args ){

Pattern p = Pattern.compile(REGEX);

Matcher m = p.matcher(INPUT); // 获取 matcher 对象

int count = 0;

while(m.find()) {

count++;

System.out.println("Match number "+count);

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

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

}

}

}

以上实例编译运行结果如下：

Match number 1

start(): 0

end(): 3

Match number 2

start(): 4

end(): 7

Match number 3

start(): 8

end(): 11

Match number 4

start(): 19

end(): 22

可以看到这个例子是使用单词边界，以确保字母 "c" "a" "t" 并非仅是一个较长的词的子串。它也提供了一些关于输入字符串中匹配发生位置的有用信息。

Start 方法返回在以前的匹配操作期间，由给定组所捕获的子序列的初始索引，end 方法最后一个匹配字符的索引加 1。

matches 和 lookingAt 方法

matches 和 lookingAt 方法都用来尝试匹配一个输入序列模式。它们的不同是 matches 要求整个序列都匹配，而lookingAt 不要求。

lookingAt 方法虽然不需要整句都匹配，但是需要从第一个字符开始匹配。

这两个方法经常在输入字符串的开始使用。

我们通过下面这个例子，来解释这个功能：

RegexMatches.java 文件代码：

import java.util.regex.Matcher;

import java.util.regex.Pattern;

public class RegexMatches

{

private static final String REGEX = "foo";

private static final String INPUT = "fooooooooooooooooo";

private static final String INPUT2 = "ooooofoooooooooooo";

private static Pattern pattern;

private static Matcher matcher;

private static Matcher matcher2;

public static void main( String[] args ){

pattern = Pattern.compile(REGEX);

matcher = pattern.matcher(INPUT);

matcher2 = pattern.matcher(INPUT2);

System.out.println("Current REGEX is: "+REGEX);

System.out.println("Current INPUT is: "+INPUT);

System.out.println("Current INPUT2 is: "+INPUT2);

System.out.println("lookingAt(): "+matcher.lookingAt());

System.out.println("matches(): "+matcher.matches());

System.out.println("lookingAt(): "+matcher2.lookingAt());

}

}

以上实例编译运行结果如下：

Current REGEX is: foo

Current INPUT is: fooooooooooooooooo

Current INPUT2 is: ooooofoooooooooooo

lookingAt(): true

matches(): false

lookingAt(): false

replaceFirst 和 replaceAll 方法

replaceFirst 和 replaceAll 方法用来替换匹配正则表达式的文本。不同的是，replaceFirst 替换首次匹配，replaceAll 替换所有匹配。

下面的例子来解释这个功能：

RegexMatches.java 文件代码：

import java.util.regex.Matcher;

import java.util.regex.Pattern;

public class RegexMatches

{

private static String REGEX = "dog";

private static String INPUT = "The dog says meow. " +

"All dogs say meow.";

private static String REPLACE = "cat";

public static void main(String[] args) {

Pattern p = Pattern.compile(REGEX);

// get a matcher object

Matcher m = p.matcher(INPUT);

INPUT = m.replaceAll(REPLACE);

System.out.println(INPUT);

}

}

以上实例编译运行结果如下：

The cat says meow. All cats say meow.

appendReplacement 和 appendTail 方法

Matcher 类也提供了appendReplacement 和 appendTail 方法用于文本替换：

看下面的例子来解释这个功能：

RegexMatches.java 文件代码：

import java.util.regex.Matcher;

import java.util.regex.Pattern;

public class RegexMatches

{

private static String REGEX = "a\*b";

private static String INPUT = "aabfooaabfooabfoobkkk";

private static String REPLACE = "-";

public static void main(String[] args) {

Pattern p = Pattern.compile(REGEX);

// 获取 matcher 对象

Matcher m = p.matcher(INPUT);

StringBuffer sb = new StringBuffer();

while(m.find()){

m.appendReplacement(sb,REPLACE);

}

m.appendTail(sb);

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

}

}

以上实例编译运行结果如下：

-foo-foo-foo-kkk

PatternSyntaxException 类的方法

PatternSyntaxException 是一个非强制异常类，它指示一个正则表达式模式中的语法错误。

PatternSyntaxException 类提供了下面的方法来帮助我们查看发生了什么错误。

序号 方法及说明

1 public String getDescription()

获取错误的描述。

2 public int getIndex()

获取错误的索引。

3 public String getPattern()

获取错误的正则表达式模式。

4 public String getMessage()

返回多行字符串，包含语法错误及其索引的描述、错误的正则表达式模式和模式中错误索引的可视化指示。