Java 正则表达式详解

本文出自:http://www.computerworld.com.cn 作者:仙人掌工作室 (2002-01-29 20:49:25)

如果你曾经用过 Perl 或任何其他内建正则表达式支持的语言，你一定知道用正则表达式处理文本和匹配模式是多么简单。如果你不熟悉这个术语，那么"正则表达式"（Regular Expression）就是一个字符构成的串，它定义了一个用来搜索匹配字符串的模式。

许多语言，包括 Perl、PHP、Python、JavaScript 和 JScript，都支持用正则表达式处理文本，一些文本编辑器用正则表达式实现高级"搜索-替换"功能。那么 Java 又怎样呢？本文写作时，一个包含了用正则表达式进行文本处理的 Java 规范需求（Specification Request）已经得到认可，你可以期待在 JDK 的下一版本中看到它。

然而，如果现在就需要使用正则表达式，又该怎么办呢？你可以从 Apache.org 下载源代码开放的 Jakarta-ORO 库。本文接下来的内容先简要地介绍正则表达式的入门知识，然后以 Jakarta-ORO API 为例介绍如何使用正则表达式。

一、正则表达式基础知识

我们先从简单的开始。假设你要搜索一个包含字符"cat"的字符串，搜索用的正则表达式就是"cat"。如果搜索对大小写不敏感，单词"catalog"、"Catherine"、"sophisticated"都可以匹配。也就是说：

```
正则表达式: cat
匹配: cat, catalog, Catherine, sophisticated
```

1.1 句点符号

假设你在玩英文拼字游戏，想要找出三个字母的单词，而且这些单词必须以"t"字母开头，以"n"字母结束。另外，假设有一本英文字典，你可以用正则表达式搜索它的全部内容。要构造出这个正则表达式，你可以使用一个通配符——句点符号"."。这样，完整的表达式就是"t.n"，它匹配"tan"、"ten"、"tin"和"ton"，还匹配"t#n"、"tpn"甚至"t n"，还有其他许多无意义的组合。这是因为句点符号匹配所有字符，包括空格、Tab 字符甚至换行符：

```
正则表达式: t.n
匹配: tan, Ten, tin, ton, t n, t#n, tpn, 等
```

1.2 方括号符号

为了解决句点符号匹配范围过于广泛这一问题，你可以在方括号（"[]"）里面指定看来有意义的字符。此时，只有方括号里面指定的字符才参与匹配。也就是说，正则表达式"t[aeio]n"只匹配"tan"、"Ten"、"tin"和"ton"。但"Toon"不匹配，因为在方括号之内你只能匹配单个字符：

```
正则表达式：t[aeio]n
匹配：tan，Ten，tin，ton
```

1.3 “或”符号

如果除了上面匹配的所有单词之外，你还想要匹配“toon”，那么，你可以使用“|”操作符。“|”操作符的基本意义就是“或”运算。要匹配“toon”，使用“t(a|e|i|o|oo)n”正则表达式。这里不能使用方扩号，因为方括号只允许匹配单个字符；这里必须使用圆括号“()”。圆括号还可以用来分组，具体请参见后面介绍。

```
正则表达式：t(a|e|i|o|oo)n
匹配：tan，Ten，tin，ton，toon
```

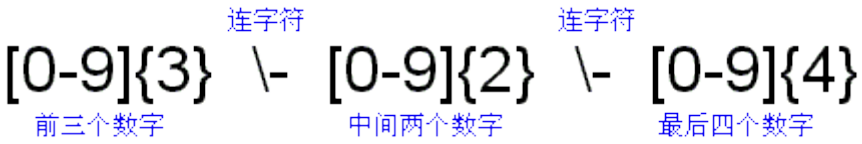
1.4 表示匹配次数的符号

表一显示了表示匹配次数的符号，这些符号用来确定紧靠该符号左边的符号出现的次数：

表一：表示次数的符号

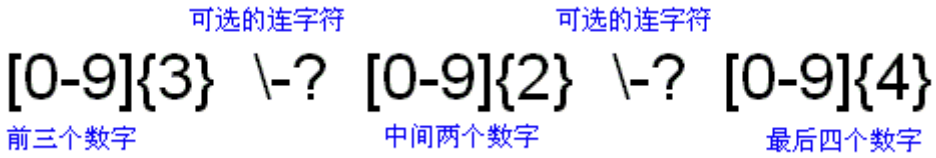
符号	次数
*	0 次或者多次
+	1 次或者多次
?	0 次或者 1 次
{n}	恰好 n 次
{n,m}	从 n 次到 m 次

假设我们要在文本文件中搜索美国的社会安全号码。这个号码的格式是 999-99-9999。用来匹配它的正则表达式如图一所示。在正则表达式中，连字符（“-”）有着特殊的意义，它表示一个范围，比如从 0 到 9。因此，匹配社会安全号码中的连字符号时，它的前面要加上一个转义字符“\”。



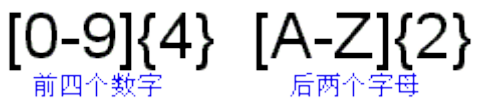
图一：匹配所有 123-12-1234 形式的社会安全号码

假设进行搜索的时候，你希望连字符号可以出现，也可以不出现——即，999-99-9999 和 999999999 都属于正确的格式。这时，你可以在连字符号后面加上“？”数量限定符号，如图二所示：



图二：匹配所有 123-12-1234 和 123121234 形式的社会安全号码

下面我们再来看另外一个例子。美国汽车牌照的一种格式是四个数字加上二个字母。它的正则表达式前面是数字部分"[0-9]{4}"，再加上字母部分"[A-Z]{2}"。图三显示了完整的正则表达式。



图三：匹配典型的美国汽车牌照号码，如 8836KV

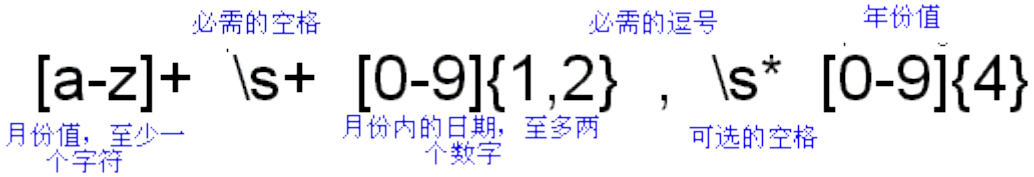
1.5 "否"符号

"^"符号称为"否"符号。如果用在方括号内，"^"表示不想要匹配的字符。例如，图四的正则表达式匹配所有单词，但以"X"字母开头的单词除外。

图四：匹配所有单词，但"X"开头的除外

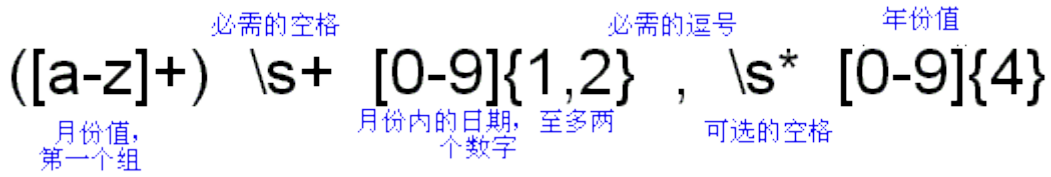
1.6 圆括号和空白符号

假设要从格式为"June 26, 1951"的生日日期中提取出月份部分，用来匹配该日期的正则表达式可以如图五所示：



图五：匹配所有 Moth DD,YYYY 格式的日期

新出现的"\s"符号是空白符号，匹配所有的空白字符，包括 Tab 字符。如果字符串正确匹配，接下来如何提取出月份部分呢？只需在月份周围加上一个圆括号创建一个组，然后用 ORO API（本文后面详细讨论）提取出它的值。修改后的正则表达式如图六所示：



图六：匹配所有 Month DD,YYYY 格式的日期，定义月份值为第一个组

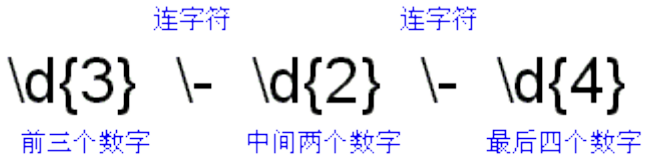
1.7 其它符号

为简便起见，你可以使用一些为常见正则表达式创建的快捷符号。如表二所示：

表二：常用符号

表二：常用符号	
符号	等价的正则表达式
\d	[0-9]
\D	[^0-9]
\w	[A-Z0-9]
\W	[^A-Z0-9]
\s	[\t\n\r\f]
\S	[^ \t\n\r\f]

例如，在前面社会安全号码的例子中，所有出现"[0-9]"的地方我们都可以使用"\d"。修改后的正则表达式如图七所示：



图七：匹配所有 123-12-1234 格式的社会安全号码

二、Jakarta-ORO 库

有许多源代码开放的正则表达式库可供 Java 程序员使用，而且它们中的许多支持 Perl 5 兼容的正则表达式语法。我在这里选用的是 Jakarta-ORO 正则表达式库，它是最全面的正则表达式 API 之一，而且它与 Perl 5 正则表达式完全兼容。另外，它也是优化得最好的 API 之一。

Jakarta-ORO 库以前叫做 OROMatcher，Daniel Savarese 大方地把它赠送给了 Jakarta Project。你可以按照本文最后参考资源的说明下载它。

我首先将简要介绍使用 Jakarta-ORO 库时你必须创建和访问的对象，然后介绍如何使用 Jakarta-ORO API。

▲ PatternCompiler 对象

首先，创建一个 Perl5Compiler 类的实例，并把它赋值给 PatternCompiler 接口对象。Perl5Compiler 是 PatternCompiler 接口的一个实现，允许你把正则表达式编译成用来匹配的 Pattern 对象。

```
PatternCompiler compiler=new Perl5Compiler();
```

▲ Pattern 对象

要把正则表达式编译成 Pattern 对象，调用 compiler 对象的 compile()方法，并在调用参数中指定正则表达式。例如，你可以按照下面这种方式编译正则表达式"t[aeio]n"：

```
Pattern pattern=null;
try {
  pattern=compiler.compile("t[aeio]n");
} catch (MalformedPatternException e) {
  e.printStackTrace();
}
```

默认情况下，编译器创建一个大小写敏感的模式（pattern）。因此，上面代码编译得到的模式只匹配"tin"、"tan"、 "ten"和"ton"，但不匹配"Tin"和"taN"。要创建一个大小写不敏感的模式，你应该在调用编译器的时候指定一个额外的参数：

```
pattern=compiler.compile("t[aeio]n",Perl5Compiler.CASE_INSENSITIVE_MASK);
```

创建好 Pattern 对象之后，你就可以通过 PatternMatcher 类用该 Pattern 对象进行模式匹配。

▲ PatternMatcher 对象

PatternMatcher 对象根据 Pattern 对象和字符串进行匹配检查。你要实例化一个 Perl5Matcher 类并把结果赋值给 PatternMatcher 接口。Perl5Matcher 类是 PatternMatcher 接口的一个实现，它根据 Perl 5 正则表达式语法进行模式匹配：

```
PatternMatcher matcher=new Perl5Matcher();
```

使用 PatternMatcher 对象，你可以用多个方法进行匹配操作，这些方法的第一个参数都是需要根据正则表达式进行匹配的字符串：

?boolean matches(String input, Pattern pattern)：当输入字符串和正则表达式要精确匹配时使用。换句话说，正则表达式必须完整地描述输入字符串。

?boolean matchesPrefix(String input, Pattern pattern)：当正则表达式匹配输入字符串起始部分时使用。

?boolean contains(String input, Pattern pattern)：当正则表达式要匹配输入字符串的一部分时使用（即，它必须是一个子串）。

另外，在上面三个方法调用中，你还可以用 PatternMatcherInput 对象作为参数替代 String 对象；这时，你可以从字符串中最后一次匹配的位置开始继续进行匹配。当字符串可能有多个子串匹配给定的正则表达式时，用 PatternMatcherInput 对象作为参数就很有用了。用 PatternMatcherInput 对象作为参数替代 String 时，上述三个方法的语法如下：

?boolean matches(PatternMatcherInput input, Pattern pattern)

?boolean matchesPrefix(PatternMatcherInput input, Pattern pattern)

?boolean contains(PatternMatcherInput input, Pattern pattern)

三、应用实例

下面我们来看看 Jakarta-ORO 库的一些应用实例。

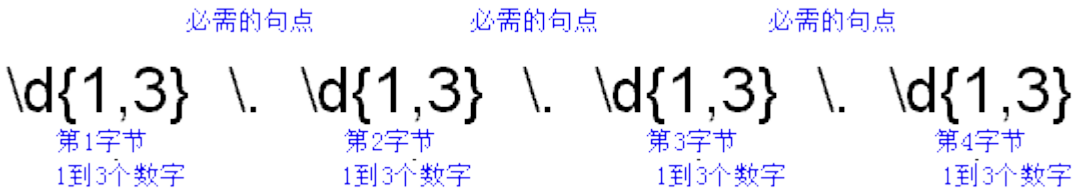
3.1 日志文件处理

任务：分析一个 Web 服务器日志文件，确定每一个用户花在网站上的时间。在典型的 BEA WebLogic 日志文件中，日志记录的格式如下：

```
172.26.155.241 - - [26/Feb/2001:10:56:03 -0500]
    "GET /IsAlive.htm HTTP/1.0" 200 15
```

分析这个日志记录，可以发现，要从这个日志文件提取的内容有两项：IP 地址和页面访问时间。你可以用分组符号（圆括号）从日志记录提取出 IP 地址和时间标记。

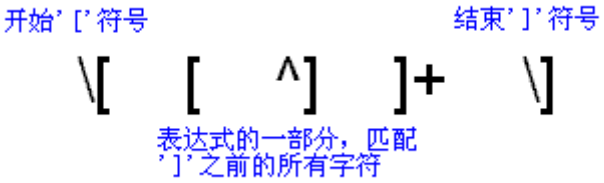
首先我们来看看 IP 地址。IP 地址有 4 个字节构成，每一个字节的值在 0 到 255 之间，各个字节通过一个句点分隔。因此，IP 地址中的每一个字节有至少一个、最多三个数字。图八显示了为 IP 地址编写的正则表达式：



图八：匹配 IP 地址

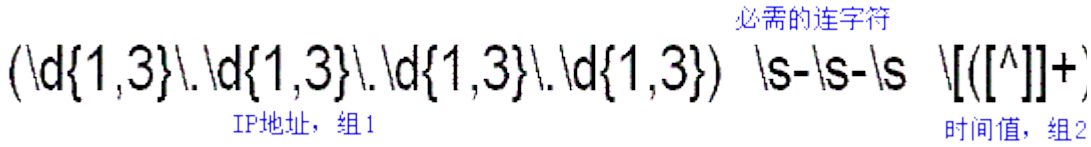
IP 地址中的句点字符必须进行转义处理（前面加上"\"），因为 IP 地址中的句点具有它本来的含义，而不是采用正则表达式语法中的特殊含义。句点在正则表达式中的特殊含义本文前面已经介绍。

日志记录的时间部分由一对方括号包围。你可以按照如下思路提取出方括号里面的所有内容：首先搜索起始方括号字符（"["），提取出所有不超过结束方括号字符（"]"）的内容，向前寻找直至找到结束方括号字符。图九显示了这部分的正则表达式。



图九：匹配至少一个字符，直至找到"]"

现在，把上述两个正则表达式加上分组符号（圆括号）后合并成单个表达式，这样就可以从日志记录提取出 IP 地址和时间。注意，为了匹配"- -"（但不提取它），正则表达式中间加入了"\s-\s-\s"。完整的正则表达式如图十所示。



图十：匹配 IP 地址和时间标记

现在正则表达式已经编写完毕，接下来可以编写使用正则表达式库的 Java 代码了。

为使用 Jakarta-ORO 库，首先创建正则表达式字符串和待分析的日志记录字符串：

```
String logEntry="172.26.155.241 - - [26/Feb/2001:10:56:03 -0500]
        \"GET /IsAlive.htm HTTP/1.0\" 200 15 ";
String
regexp="([0-9]{1,3}\\.[0-9]{1,3}\\.[0-9]{1,3}\\.[0-9]{1,3})
        \\s-\\s-\\s\\[([^\\]]+)\\]";
```

这里使用的正则表达式与图十的正则表达式差不多完全相同，但有一点例外：在 Java 中，你必须对每一个向前的斜杠（"\"）进行转义处理。图十不是 Java 的表示形式，所以我们要在每个"\"前面加上一个"\"以免出现编译错误。遗憾的是，转义处理过程很容易出现错误，所以应该小心谨慎。你可以首先输入未经转义处理的正则表达式，然后从左到右依次把每一个"\"替换成"\\"。如果要复检，你可以试着把它输出到屏幕上。

初始化字符串之后，实例化 PatternCompiler 对象，用 PatternCompiler 编译正则表达式创建一个 Pattern 对象：

```
PatternCompiler compiler=new Perl5Compiler();
Pattern pattern=compiler.compile(regexp);
```

现在，创建 PatternMatcher 对象，调用 PatternMatcher 接口的 contain()方法检查匹配情况：

```
PatternMatcher matcher=new Perl5Matcher();
  if (matcher.contains(logEntry,pattern)) {
  MatchResult result=matcher.getMatch();
  System.out.println("IP: "+result.group(1));
  System.out.println("Timestamp: "+result.group(2));
  }
```

接下来，利用 PatternMatcher 接口返回的 MatchResult 对象，输出匹配的组。由于 logEntry 字符串包含匹配的内容，你可以看到类如下面的输出：

```
IP: 172.26.155.241
Timestamp: 26/Feb/2001:10:56:03 -0500
```

3.2 HTML 处理实例一

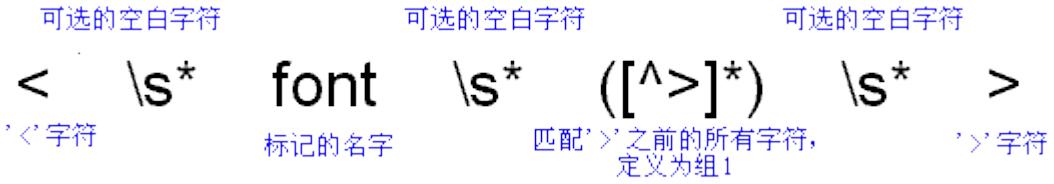
下面一个任务是分析 HTML 页面内 FONT 标记的所有属性。HTML 页面内典型的 FONT 标记如下所示：

```
<font face="Arial, Serif" size="+2" color="red">
```

程序将按照如下形式，输出每一个 FONT 标记的属性：

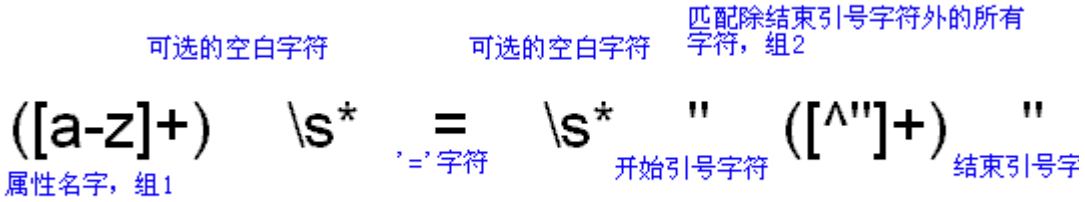
```
face=Arial, Serif
size=+2
color=red
```

在这种情况下，我建议你使用两个正则表达式。第一个如图十一所示，它从字体标记提取出""face="Arial, Serif" size="+2" color="red"""。



图十一：匹配 FONT 标记的所有属性

第二个正则表达式如图十二所示，它把各个属性分割成名字-值对。



图十二：匹配单个属性，并把它分割成名字-值对

分割结果为：

```
face Arial, Serif
size +2
color red
```

现在我们来看看完成这个任务的 Java 代码。首先创建两个正则表达式字符串，用 Perl5Compiler 把它们编译成 Pattern 对象。编译正则表达式的时候，指定 Perl5Compiler.CASE_INSENSITIVE_MASK 选项，使得匹配操作不区分大小写。

接下来，创建一个执行匹配操作的 Perl5Matcher 对象。

```
String regexpForFontTag="<\\s*font\\s+([^>]*)\\s*>";
String regexpForFontAttrib="([a-z]+)\\s*=\\s*\"([^\"]+)\"";

PatternCompiler compiler=new Perl5Compiler();
Pattern patternForFontTag=compiler.compile(regexpForFontTag,
      Perl5Compiler.CASE_INSENSITIVE_MASK);
Pattern
patternForFontAttrib=compiler.compile(regexpForFontAttrib,
      Perl5Compiler.CASE_INSENSITIVE_MASK);

PatternMatcher matcher=new Perl5Matcher();
```

假设有一个 String 类型的变量 html，它代表了 HTML 文件中的一行内容。如果 html 字符串包含 FONT 标记，匹配器将返回 true。此时，你可以用匹配器对象返回的 MatchResult 对象获得第一个组，它包含了 FONT 的所有属性：

```
  if (matcher.contains(html,patternForFontTag)) {
      MatchResult result=matcher.getMatch();
      String attribs=result.group(1);

      PatternMatcherInput input=new PatternMatcherInput(attribs);
      while (matcher.contains(input,patternForFontAttrib)) {
          result=matcher.getMatch();
          System.out.println(result.group(1)+":
"+result.group(2));
      }
  }
```

接下来创建一个 PatternMatcherInput 对象。这个对象允许你从最后一次匹配的位置开始继续进行匹配操作，因此，它很适合于提取 FONT 标记内属性的名字-值对。创建 PatternMatcherInput 对象，以参数形式传入待匹配的字符串。然后，用匹配器实例提取出每一个 FONT 的属性。这通过指定 PatternMatcherInput 对象（而不是字符串对象）为参数，反复地调用 PatternMatcher 对象的 contains()方法完成。PatternMatcherInput 对象之中的每一次迭代将把它内部的指针向前移动，下一次检测将从前一次匹配位置的后面开始。

本例的输出结果如下：

```
face:Arial, Serif
size:+2
color:red
```

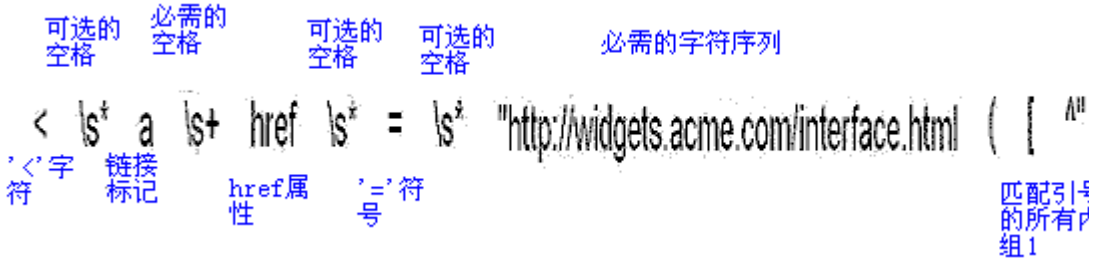
3.3 HTML 处理实例二

下面我们来看看另一个处理 HTML 的例子。这一次，我们假定 Web 服务器从 widgets.acme.com 移到了 newserver.acme.com。现在你要修改一些页面中的链接：

```
从:
<a href="http://widgets.acme.com/interface.html#How_To_Buy">
<a href="http://widgets.acme.com/interface.html#How_To_Sell">

改成:
<a href="http://newserver.acme.com/interface.html#How_To_Buy">
<a href="http://newserver.acme.com/interface.html#How_To_Sell">
```

执行这个搜索的正则表达式如图十三所示：



图十三：匹配修改前的链接

如果能够匹配这个正则表达式，你可以用下面的内容替换图十三的链接：

```
<a href="http://newserver.acme.com/interface.html#$1">
```

注意#字符的后面加上了$1。Perl 正则表达式语法用$1、$2 等表示已经匹配且提取出来的组。图十三的表达式把所有作为一个组匹配和提取出来的内容附加到链接的后面。

现在，返回 Java。就象前面我们所做的那样，你必须创建测试字符串，创建把正则表达式编译到 Pattern 对象所必需的对象，以及创建一个 PatternMatcher 对象：

```
String link="<a href=\"http://widgets.acme.com/interface.html#How_To_Trade\">";
String
regexpForLink="<\\s*a\\s+href\\s*=\\s*\"http://widgets.acme.com/interface.html#([^\"]+)\">

PatternCompiler compiler=new Perl5Compiler();
Pattern patternForLink=compiler.compile(regexpForLink,
          Perl5Compiler.CASE_INSENSITIVE_MASK);

PatternMatcher matcher=new Perl5Matcher();
```

接下来，用 com.oroinc.text.regex 包 Util 类的 substitute()静态方法进行替换，输出结果字符串：

```
String result=Util.substitute(matcher,
     patternForLink,
     new Perl5Substitution(
     "<a href=\"http://newserver.acme.com/interface.html#$1\">"),
     link,
     Util.SUBSTITUTE_ALL);
System.out.println(result);
```

Util.substitute()方法的语法如下：

```
public static String substitute(PatternMatcher matcher,
  Pattern pattern,
  Substitution sub,
  String input,
  int numSubs)
```

这个调用的前两个参数是以前创建的 PatternMatcher 和 Pattern 对象。第三个参数是一个 Substiution 对象，它决定了替换操作如何进行。本例使用的是 Perl5Substitution 对象，它能够进行 Perl5 风格的替换。第四个参数是想要进行替换操作的字符串，最后一个参数允许指定是否替换模式的所有匹配子串（Util.SUBSTITUTE_ALL），或只替换指定的次数。

【结束语】在这篇文章中，我为你介绍了正则表达式的强大功能。只要正确运用，正则表达式能够在字符串提取和文本修改中起到很大的作用。另外，我还介绍了如何在 Java 程序中通过 Jakarta-ORO 库利用正则表达式。至于最终采用老式的字符串处理方式（使用 StringTokenizer，charAt，和 substring），还是采用正则表达式，这就有待你自己决定了。