

版权声明：本文由吴仙杰创作整理，转载请注明出处：  
<https://segmentfault.com/a/1190000009162306>

# 1. 正则表达式

## 1.1 什么是正则表达式

正则表达式

：定义一个搜索模式的字符串。

正则表达式可以用于搜索、编辑和操作文本。

正则对文本的分析或修改过程为：首先正则表达式应用的是文本字符串（text/string），它会以定义的模式从左到右匹配文本，每个源字符只匹配一次。

## 1.2 示例

正则表达式	匹配
this is text	精确匹配字符串 "this is text"
this\s+is\s+text	匹配单词 "this" 后跟一个或多个空格字符，后跟词 "is" 后跟一个或多个空格字符，后跟词 "text"
^\d+(\.\d+)?	^ 定义模式必须匹配字符串的开始，\d+ 匹配一个或多个数字，? 表明小括号内的语句是可选的，\. 匹配 "."，小括号表示分组。例如匹配："5"、"1.5" 和 "2.21"

# 2. 正则表达式的编写规则

## 2.1 常见匹配符号

正则表达式	描述
.	匹配所有单个字符，除了换行符（Linux 中换行是 \n，

	Windows 中换行是 \r\n )
<code>^regex</code>	正则必须匹配字符串开头
<code>regex\$</code>	正则必须匹配字符串结尾
<code>[abc]</code>	复选集定义，匹配字母 a 或 b 或 c
<code>[abc][vz]</code>	复选集定义，匹配字母 a 或 b 或 c，后面跟着 v 或 z
<code>[^abc]</code>	当插入符 ^ 在中括号中以第一个字符开始显示，则表示否定模式。此模式匹配所有字符，除了 a 或 b 或 c
<code>[a-d1-7]</code>	范围匹配，匹配字母 a 到 d 和数字从 1 到 7 之间，但不匹配 d1
<code>xz</code>	匹配 X 后直接跟着 Z
<code>X Z</code>	匹配 X 或 Z

## 2.2 元字符

元字符是一个预定义的字符。

正则表达式	描述
<code>\d</code>	匹配一个数字，是 <code>[0-9]</code> 的简写
<code>\D</code>	匹配一个非数字，是 <code>[^0-9]</code> 的简写
<code>\s</code>	匹配一个空格，是 <code>[\t\n\x0b\r\f]</code> 的简写
<code>\S</code>	匹配一个非空格
<code>\w</code>	匹配一个单词字符（大小写字母、数字、下划线），是

	[a-zA-z_0-9] 的简写
\w	匹配一个非单词字符（除了大小写字母、数字、下划线之外的字符），等同于 [^\w]

## 2.3 限定符

限定符定义了一个元素可以发生的频率。

正则表达式	描述	举例
*	匹配 $\geq 0$ 个，是 {0,} 的简写	x* 表示匹配零个或多个字母 X，.* 表示匹配任何字符串
+	匹配 $\geq 1$ 个，是 {1,} 的简写	x+ 表示匹配一个或多个字母 X
?	匹配 1 个或 0 个，是 {0,1} 的简写	x? 表示匹配 0 个或 1 个字母 X
{X}	只匹配 X 个字符	\d{3} 表示匹配 3 个数字，.{10} 表示匹配任何长度是 10 的字符串
{X,Y}	匹配 $\geq X$ 且 $\leq Y$ 个	\d{1,4} 表示匹配至少 1 个最多 4 个数字
*?	如果 ? 是限定符 * 或 + 或 ? 或 {} 后面	

的第一个 字符，那 么表示 <b>非 贪婪模式</b> (尽可能 少的匹配 字符)， 而不是默 认的 <b>贪婪 模式</b>
--

## 2.4 分组和反向引用

小括号 `()` 可以达到对正则表达式进行分组的效果。

模式分组后会在正则表达式中创建反向引用。反向引用会保存匹配模式分组的字符串片断，这使得我们可以获取并使用这个字符串片断。

在以正则表达式替换字符串的语法中，是通过 `$` 来引用分组的反向引用，`$0` 是匹配完整模式的字符串（注意在 JavaScript 中是用 `&` 表示）；`$1` 是第一个分组的反向引用；`$2` 是第二个分组的反向引用，以此类推。

示例：

```
package com.wuxianjiezhi.demo.regex;public class RegexTest{
    public static void main(String[] args){        // 去除单词与 , 和 . 之间的空格
        String Str = "Hello , World .";        String pattern = "((\\w) (\\s+) ([.,]))";        // $0 匹配 `(\w) (\s+) ([.,])` 结果为 `o空格,` 和 `d空格.`
        // $1 匹配 `(\w)` 结果为 `o` 和 `d`
        // $2 匹配 `(\s+)` 结果为 `空格` 和 `空格`
        // $3 匹配 `([.,])` 结果为 `,` 和 `.`
        System.out.println(Str.replaceAll(pattern, "$1$3")); // Hello, World.
    }
}
```

上面的例子中，我们使用了 `[.]` 来匹配普通字符 `.` 而不需要使用 `[\\.]`。因为正则对于 `[]` 中的 `.`，会自动处理为 `[\\.]`，即普通字符 `.` 进行匹配。

### 2.4.1 仅分组但无反向引用

当我们在小括号 `()` 内的模式开头加入 `?:`，那么表示这个模式仅分组，但不创建反向引用。

示例：

```
package com.wuxianjiezhi.regex;import java.util.regex.Matcher;import
java.util.regex.Pattern;publicclassRegexTest{
publicstaticvoidmain(String[] args){      String str = "img.jpg";
// 分组且创建反向引用      Pattern pattern = Pattern.compile("
(jpg|png)");      Matcher matcher = pattern.matcher(str);
while (matcher.find()) {
System.out.println(matcher.group());
System.out.println(matcher.group(1));      }      }}
```

运行结果：

jpgjpg

若源码改为：

```
package com.wuxianjiezhi.regex;import java.util.regex.Matcher;import
java.util.regex.Pattern;publicclassRegexTest{
publicstaticvoidmain(String[] args){      String str = "img.jpg";
// 分组但不创建反向引用      Pattern pattern = Pattern.compile("
(?:jpg|png)");      Matcher matcher = pattern.matcher(str);
while (matcher.find()) {
System.out.println(matcher.group());
System.out.println(matcher.group(1));      }      }}
```

运行结果：

jpgException in thread "main" java.lang.IndexOutOfBoundsException: No group 1 at java.util.regex.Matcher.group(Matcher.java:538) at com.wuxianjiezhi.regex.RegexTest.main(RegexTest.java:15)

## 2.4.2 分组的反向引用副本

Java 中可以在小括号中使用 `?` 将小括号中匹配的内容保存为一个名字为 `name` 的副本。

示例：

```
package com.wuxianjiezhi.regex;import java.util.regex.Matcher;import java.util.regex.Pattern;public class RegexTest{public static void main(String[] args){    String str = "@wxj 你好啊";    Pattern pattern = Pattern.compile("@(?\\w+\\s)"); // 保存一个副本    Matcher matcher = pattern.matcher(str);    while (matcher.find()) {        System.out.println(matcher.group());        System.out.println(matcher.group(1));        System.out.println(matcher.group("first"));    }}}
```

运行结果：

```
@wxj wxj wxj
```

## 2.5 否定先行断言 (Negative lookahead)

我们可以创建否定先行断言模式的匹配，即某个字符串后面不包含另一个字符串的匹配模式。

否定先行断言模式通过 `(?!pattern)` 定义。比如，我们匹配后面不是跟着“b”的“a”：

```
a(?!b)
```

## 2.6 指定正则表达式的模式

可以在正则的开头指定模式修饰符。

- `(?i)` 使正则忽略大小写。
- `(?s)` 表示单行模式 ("single line mode") 使正则的 `.` 匹配所有字符，包括换行符。
- `(?m)` 表示多行模式 ("multi-line mode")，使正则的 `^` 和 `$` 匹配字符串中每行的开始和结束。

## 2.7 Java 中的反斜杠

反斜杠 `\` 在 Java 中表示转义字符，这意味着 `\` 在 Java 拥有预定义的含义。

这里例举两个特别重要的用法：

- 在匹配 `.` 或 `{` 或 `[` 或 `(` 或 `?` 或 `$` 或 `^` 或 `*` 这些特殊字符时，需要在前面加上 `\\`，比如匹配 `.` 时，Java 中要写为 `\\.`，但对于正则表达式来说就是 `\.`。
- 在匹配 `\` 时，Java 中要写为 `\\\\`，但对于正则表达式来说就是 `\\`。

注意：Java 中的正则表达式字符串有两层含义，首先 Java 字符串转义出符合正则表达式语法的字符串，然后再由转义后的正则表达式进行模式匹配。

## 2.8 易错点示例

- `[jpg|png]` 代表匹配 `j` 或 `p` 或 `g` 或 `p` 或 `n` 或 `g` 中的任意一个字符。
- `(jpg|png)` 代表匹配 `jpg` 或 `png`。

# 3. 在字符串中使用正则表达式

## 3.1 内置的字符串正则处理方法

在 Java 中有四个内置的运行正则表达式的方法，分别是 `matches()`、`split()`、`replaceFirst()`、`replaceAll()`。注意 `replace()` 方法不支持正则表达式。

方法	描述
<code>s.matches("regex")</code>	当仅当正则匹配整个字符串时返回 <code>true</code>
<code>s.split("regex")</code>	按匹配的正则表达式切片字符串
<code>s.replaceFirst("regex", "replacement")</code>	替换首次匹配的字符串片段
<code>s.replaceAll("regex", "replacement")</code>	替换所有匹配的字符

### 3.2 示例

示例代码:

```
package com.wuxianjiezhi.regex;public class RegexTest{
    public static void main(String[] args){
        System.out.println("wxj".matches("wxj"));          System.out.println("--
        -----");          String[] array = "w x j".split("\\s");          for
        (String item : array) {          System.out.println(item);          }
        System.out.println("-----");          System.out.println("w x
        j".replaceFirst("\\s", "-"));          System.out.println("-----");
        System.out.println("w x j".replaceAll("\\s", "-"));    }}
```

运行结果:

```
true-----wxj-----w-x j-----w-x-j
```

### 4. 模式和匹配



Java 中使用正则表达式需要用到两个类，分别为 `java.util.regex.Pattern` 和 `java.util.regex.Matcher`。

第一步，通过正则表达式创建模式对象 `Pattern`。

第二步，通过模式对象 `Pattern`，根据指定字符串创建匹配对象 `Matcher`。

第三步，通过匹配对象 `Matcher`，根据正则表达式操作字符串。

来个例子，加深理解：

```
package com.wuxianjiezhang.regex;import java.util.regex.Matcher;import
java.util.regex.Pattern;public class RegexTest{
    public static void main(String[] args){        String text = "Hello
    Regex!";        Pattern pattern = Pattern.compile("\\w+");        //
    Java 中忽略大小写，有两种写法：// Pattern pattern = Pattern.compile("\\w+",
    Pattern.CASE_INSENSITIVE);// Pattern pattern = Pattern.compile("(?
    i)\\w+"); // 推荐写法        Matcher matcher = pattern.matcher(text);
    // 遍历所有匹配的序列while (matcher.find()) {
        System.out.print("Start index: " + matcher.start());
        System.out.print(" End index: " + matcher.end() + " ");
        System.out.println(matcher.group());        }        // 创建第二个模式，将
    空格替换为 tab        Pattern replace = Pattern.compile("\\s+");
        Matcher matcher2 = replace.matcher(text);
        System.out.println(matcher2.replaceAll("\\t"));    }}
```

运行结果：

```
Start index: 0End index: 5 HelloStart index: 6End index: 11 RegexHello
Regex!
```

## 5. 若干个常用例子

## 5.1 中文的匹配

`[\u4e00-\u9fa5]+` 代表匹配中文字。

```
package com.wuxianjiezhang.regex;import java.util.regex.Matcher;import java.util.regex.Pattern;public class RegexTest{public static void main(String[] args){    String str = "闲人到人间";    Pattern pattern = Pattern.compile("[\\u4e00-\\u9fa5]+");    Matcher matcher = pattern.matcher(str);    while (matcher.find()) {        System.out.println(matcher.group());    }}}
```

运行结果：

闲人到人间

## 5.2 数字范围的匹配

比如，匹配 1990 到 2017。

注意：这里有个新手易犯的错误，就是正则 `[1990-2017]`，实际这个正则只匹配 0 或 1 或 2 或 7 或 9 中的任一个字符。

正则表达式匹配数字范围时，首先要确定最大值与最小值，最后写中间值。

正确的匹配方式：

```
package com.wuxianjiezhang.regex;import java.util.regex.Matcher;import java.util.regex.Pattern;public class RegexTest{public static void main(String[] args){    String str = "1990\n2010\n2017";    // 这里应用了 (?m) 的多行匹配模式，只为方便我们测试    输出// "^1990$|^199[1-9]$|^20[0-1][0-6]$|^2017$" 为判断 1990-2017 正确的正则表达式    Pattern pattern = Pattern.compile("(?m)^1990$|^199[1-9]$|^20[0-1][0-6]$|^2017$");    Matcher matcher =
```



```
pattern.matcher(str);          while (matcher.find()) {
System.out.println(matcher.group());      }    }}
```

运行结果：

199020102017

## 5.3 img 标签的匹配

比如，获取图片文件内容，这里我们考虑了一些不规范的 img 标签写法：

```
package com.wuxianjiezhang.regex;import java.util.regex.Matcher;import
java.util.regex.Pattern;publicclassRegexTest{
publicstaticvoidmain(String[] args){          String str = "

" +
"

";          // 这里我们考虑了一些不规范的 img 标签写法，比如：空格、引号
Pattern pattern = Pattern.compile("\\w+.(jpg|png)(?:['\"])?\\s*/>");
Matcher matcher = pattern.matcher(str);          while (matcher.find()) {
System.out.println(matcher.group("src"));          }    }}
```

运行结果：

aaa.jpgbbb.pngccc.png

## 5.4 贪婪与非贪婪模式的匹配

比如，获取 div 标签中的文本内容：

```
package com.wuxianjiezhang.regex;import java.util.regex.Matcher;import
java.util.regex.Pattern;publicclassRegexTest{
```

```

public static void main(String[] args) {
    String str = "

文章标题

发布时间


";
    // 贪婪模式
    Pattern pattern = Pattern.compile("

(?)

");
    Matcher matcher = pattern.matcher(str);
    while (matcher.find()) {
        System.out.println(matcher.group("title"));
    }
    System.out.println("-----");
    // 非贪婪模式
    pattern = Pattern.compile("

(?)

");
    matcher = pattern.matcher(str);
    while (matcher.find()) {
        System.out.println(matcher.group("title"));
    }
}

```

运行结果：

文章标题

发布时间-----文章标题发布时间

## 工具

- JavaScript、Python 等的在线表达式工具：<https://regex101.com/>
- Java 在线表达式工具：  
<http://www.regexplanet.com/advanced/java/index.html>

## 6. 推荐两个在线正则

## 7. 参考

[Java Regex - Tutorial](#)