正则表达式 – 匹配规则

正则表达式 - 示例

简单表达式

正则表达式的最简单形式是在搜索字符串中匹配其本身的单个普通字符。例如，单字符模式，如 A，不论出现在搜索字符串中的何处，它总是匹配字母 A。下面是一些单字符正则表达式模式的示例：

```
/a/
/7/
/M/
```

可以将许多单字符组合起来以形成大的表达式。例如，以下正则表达式组合了单字符表达式：a、7 和 M。

```
/a7M/
```

请注意，没有串联运算符。只须在一个字符后面键入另一个字符。

字符匹配

句点 (.) 匹配字符串中的各种打印或非打印字符，只有一个字符例外。这个例外就是换行符 (\n)。下面的正则表达式匹配 aac、abc、acc、adc 等等，以及 a1c、a2c、a-c 和 a#c：

```
/a.c/
```

若要匹配包含文件名的字符串，而句点 (.) 是输入字符串的组成部分，请在正则表达式中的句点前面加反斜杠 (\) 字符。举例来说明，下面的正则表达式匹配 filename.ext：

```
/filename\.ext/
```

这些表达式只让您匹配"任何"单个字符。可能需要匹配列表中的特定字符组。例如，可能需要查找用数字表示的章节标题（Chapter 1、Chapter 2 等等）。

中括号表达式

若要创建匹配字符组的一个列表，请在方括号（[和]）内放置一个或更多单个字符。当字符括在中括号内时，该列表称为"中括号表达式"。与在任何别的位置一样，普通字符在中括号内表示其本身，即，它在输入文本中匹配一次其本身。大多数特殊字符在中括号表达式内出现时失去它们的意义。不过也有一些例外，如：

- 如果] 字符不是第一项，它结束一个列表。若要匹配列表中的] 字符，请将它放在第一位，紧跟在开始 [后面。
- \ 字符继续作为转义符。若要匹配 \ 字符，请使用 \\。

括在中括号表达式中的字符只匹配处于正则表达式中该位置的单个字符。以下正则表达式匹配 Chapter 1、Chapter 2、Chapter 3、Chapter 4 和 Chapter 5：

```
/Chapter [12345]/
```

请注意，单词 Chapter 和后面的空格的位置相对于中括号内的字符是固定的。中括号表达式指定的只是匹配紧跟在单词 Chapter 和空格后面的单个字符位置的字符集。这是第九个字符位置。
若要使用范围代替字符本身来表示匹配字符组，请使用连字符 (-) 将范围中的开始字符和结束字符分开。单个字符的字符值确定范围内的相对顺序。下面的正则表达式包含范围表达式，该范围表达式等效于上面显示的中括号中的列表。

```
/Chapter [1-5]/
```

当以这种方式指定范围时，开始值和结束值两者都包括在范围内。注意，还有一点很重要，按 Unicode 排序顺序，开始值必须在结束值的前面。
若要在中括号表达式中包括连字符，请采用下列方法之一：

- 用反斜杠将它转义：

```
[\-]
```

- 将连字符放在中括号列表的开始或结尾。下面的表达式匹配所有小写字母和连字符：

```
[-a-z]
[a-z-]
```

- 创建一个范围，在该范围中，开始字符值小于连字符，而结束字符值等于或大于连字符。下面的两个正则表达式都满足这一要求：

```
[!--]
[!-~]
```

若要查找不在列表或范围内的所有字符，请将插入符号 (^) 放在列表的开头。如果插入字符出现在列表中的其他任何位置，则它匹配其本身。下面的正则表达式匹配1、2、3、4 或 5 之外的任何数字和字符：

```
/Chapter [^12345]/
```

在上面的示例中，表达式在第九个位置匹配 1、2、3、4 或 5 之外的任何数字和字符。这样，例如，Chapter 7 就是一个匹配项，Chapter 9 也是一个匹配项。
上面的表达式可以使用连字符 (-) 来表示：

```
/Chapter [^1-5]/
```

中括号表达式的典型用途是指定任何大写或小写字母或任何数字的匹配。下面的表达式指定这样的匹配：

```
/[A-Za-z0-9]/
```

替换和分组

替换使用 | 字符来允许在两个或多个替换选项之间进行选择。例如，可以扩展章节标题正则表达式，以返回比章标题范围更广的匹配项。但是，这并不象您可能认为的那样简单。替换匹配 | 字符任一侧最大的表达式。
您可能认为，下面的表达式匹配出现在行首和行尾、后面跟一个或两个数字的 Chapter 或 Section：

```
/^Chapter|Section [1-9][0-9]{0,1}$/
```

很遗憾，上面的正则表达式要么匹配行首的单词 Chapter，要么匹配行尾的单词 Section 及跟在其后的任何数字。如果输入字符串是 Chapter 22，那么上面的表达式只匹配单词 Chapter。如果输入字符串是 Section 22，那么该表达式匹配 Section 22。
若要使正则表达式更易于控制，可以使用括号来限制替换的范围，即，确保它只应用于两个单词 Chapter 和 Section。但是，括号也用于创建子表达式，并可能捕获它们以供以后使用，这一点在有关反向引用的那一节讲述。通过在上面的正则表达式的适当位置添加括号，就可以使该正则表达式匹配 Chapter 1 或 Section 3。
下面的正则表达式使用括号来组合 Chapter 和 Section，以便表达式正确地起作用：

```
/^(Chapter|Section) [1-9][0-9]{0,1}$/
```

尽管这些表达式正常工作，但 Chapter|Section 周围的括号还将捕获两个匹配字中的任一个供以后使用。由于在上面的表达式中只有一组括号，因此，只有一个被捕获的"子匹配项"。
在上面的示例中，您只需要使用括号来组合单词 Chapter 和 Section 之间的选择。若要防止匹配被保存以备将来使用，请在括号内正则表达式模式之前放置 ?:。下面的修改提供相同的能力而不保存子匹配项：

```
/^(?:Chapter|Section) [1-9][0-9]{0,1}$/
```

除 ?: 元字符外，两个其他非捕获元字符创建被称为"预测先行"匹配的某些内容。正向预测先行使用 ?= 指定，它匹配处于括号中匹配正则表达式模式的起始点的搜索字符串。反向预测先行使用 ?! 指定，它匹配处于与正则表达式模式不匹配的字符串的起始点的搜索字符串。
例如，假设您有一个文档，该文档包含指向 Windows 3.1、Windows 95、Windows 98 和 Windows NT 的引用。再进一步假设，您需要更新该文档，将指向 Windows 95、Windows 98 和 Windows NT 的所有引用更改为 Windows 2000。下面的正则表达式（这是一个正向预测先行的示例）匹配 Windows 95、Windows 98 和 Windows NT：

```
/Windows(?=95 |98 |NT )/
```

找到一处匹配后，紧接着就在匹配的文本（不包括预测先行中的字符）之后搜索下一处匹配。例如，如果上面的表达式匹配 Windows 98，将在 Windows 之后而不是在 98 之后继续搜索。

其他示例

下面列出一些正则表达式示例：

正则表达式	描述
/\b([a-z]+) \1\b/gi	一个单词连续出现的位置。
/(\w+):\/\/([^/:]+)(:\d*)?([^#]*)/	将一个URL解析为协议、域、端口及相对路径。
/^(?:Chapter|Section) [1-9][0-9]{0,1}$/	定位章节的位置。
/[-a-z]/	a至z共26个字母再加一个-号。
/ter\b/	可匹配chapter，而不能匹配terminal。
/\Bapt/	可匹配chapter，而不能匹配aptitude。
/Windows(?=95 |98 |NT)/	可匹配Windows95或Windows98或WindowsNT，当找到一个匹配后，从Windows后面开始进行下一次的检索匹配。
/^\s*$/	匹配空行。
/\d{2}-\d{5}/	验证由两位数字、一个连字符再加 5 位数字组成的 ID 号。
/<\s*(\S+)(\s[^>]*)?>[\s\S]*<\s*\/\1\s*>/	匹配 HTML 标记。

← 正则表达式 – 匹配规则