1.5 正则表达式

正则表达式，是根据字符串规律按一定法则，简洁表达一组字符串的表达式。正则表达式通常就是从貌似无规律的字符串中发现规律性，进而概括性地表达它们所共有的规律或模式，以方便地操作处理它们，这是真正的**化繁为简，以简御繁**的典范。

几乎所有的高级编程语言都支持正则表达式，正则表达式广泛应用于文本挖掘、数据预处理，例如：

- 检查文本中是否含有指定的特征词
- 找出文本中匹配特征词的位置
- 从文本中提取信息
- 修改文本

正则表达式包括：只能匹配自身的普通字符（如英文字母、数字、标点等）和被转义了的特殊字符（称为‘‘元字符’’）。

1.5.1 基本语法

1. 常用的元字符

符号	描述
.	匹配除换行符“/n”以外的任意字符
\\	转义字符，匹配元字符时，使用“\\元字符”
|	表示或者，即 | 前后的表达式任选一个
^	匹配字符串的开始
$	匹配字符串的结束
( )	提取匹配的字符串，即括号内的看成一个整体，即指定子表达式
[ ]	可匹配方括号内任意一个字符
{}	前面的字符或表达式的重复次数：{n}表示重复 n 次；{n,}重复 n 次到更多次；{n, m}表示重复 n 次到 m 次
*	前面的字符或表达式重复 0 次或更多次
+	前面的字符或表达式重复 1 次或更多次
?	前面的字符或表达式重复 0 次或 1 次

图 1.9: 常用的元字符（表）

其它语言中的转义字符一般是 \\；在多行模式下，^ 和 $ 就表示行的开始和结束。

```
# 创建多行模式的正则表达式
pat = regex("^\\(.+?\\)$", multiline = TRUE)
```

2. 特殊字符类与反义

符号	描述
\\d 与 \\D	匹配 1 位数字字符，匹配非数字字符
\\s 与 \\S	匹配空白符，匹配非空白符
\\w 与 \\W	匹配字母或数字或下划线或汉字，匹配非\w 字符
\\b 与 \\B	匹配单词的开始或结束的位置，匹配非\b 的位置
\\h 与 \\H	匹配水平间隔，匹配非水平间隔
\\v 与 \\V	匹配垂直间隔，匹配非垂直间隔
[^...]	匹配除了...以外的任意字符

图 1.10: 特殊字符类与反义（表）

- \\S+: 匹配不包含空白符的字符串
- \\d: 匹配数字，同 [0-9]
- [a-zA-Z0-9]: 匹配字母和数字
- [\u4e00-\u9fa5] 匹配汉字
- [^aeiou]: 匹配除 aeiou 之外的任意字符，即匹配辅音字母

3. POSIX 字符类

符号	描述
[[:lower:]]	小写字母
[[:upper:]]	大写字母
[[:alpha:]]	大小写字母
[[:digit:]]	数字 0-9
[[:alnum:]]	字母和数字
[[:blank:]]	空白符：空格、制表符、换行符、中文全角空格等
[[:cntrl:]]	控制字符
[[:punct:]]	标点符号：！”＃％＆’（）＊＋－．／：；等
[[:space:]]	空格字符：空格，制表符，垂直制表符，回车，换行符，换页符
[[:xdigit:]]	十六进制数字：0-9 A-F a-f
[[:print:]]	控制字符：[[:alpha:]], [[:punct:]], [[:space:]]
[[:graph:]]	图形化字符：[[:alpha:]], [[:punct:]]

图 1.11: POSIX 字符类（表）

4. 运算优先级

圆括号括起来的表达式最优先，其次是表示重复次数的操作（即 * + { }）；再次是连接运算（即几个字符放在一起，如 abc）；最后是或者运算（|）。

另外，正则表达式还有若干高级用法，常用的有**零宽断言**和**分组捕获**，将在下面实例

中进行演示。

1.5.2 若干实例

以上正则表达式语法组合起来使用，就能产生非常强大的匹配效果，对于匹配到的内容，根据需要可以提取它们，可以替换它们。

正则表达式与 stringr 包连用

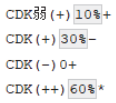
若只是调试和查看正则表达式的匹配效果，可用 `str_view()` 及其 `_all` 后缀版本，将在 RStudio 的 Viewer 窗口显示匹配结果，在原字符向量中高亮显示匹配内容，非常直观。
若要提取正则表达式匹配到的内容，则用 `str_extract()` 及其 `_all` 后缀版本。
若要替换正则表达式匹配到的内容，则用 `str_replace()` 及其 `_all` 后缀版本。

使用正则表达式关键是，能够从貌似没有规律的字符串中发现规律性，再将规律性用正则表达式语法表示出来。下面看几个正则表达式比较实用的实例。

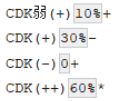
例 1.2 直接匹配

适合想要匹配的内容具有一定规律性，该规律性可用正则表达式表示出来。比如，数据中包含字母、符号、数值，我们想提取其中的数值，按正则表达式语法规则直接把要提取的部分表示出来：

```
x = c("CDK 弱 (+)10%+", "CDK(+)30%-", "CDK(-)0+", "CDK(++)60%*")
str_view(x, "\\d+%")
```




```
str_view(x, "\\d+%?")
```



\\d 表示匹配一位数字，+ 表示前面数字重复 1 次或多次，接着 % 原样匹配 %. 若后面不加 ? 则必须匹配到 % 才会成功，故第 3 个字符串就不能成功匹配；若后面加上 ? 则表示匹配前面的 % 0 次或 1 次，从而能成功匹配第 3 个字符串。

例 1.3 （零宽断言）匹配两个标志之间的内容

适合想要匹配的内容没有规律性，但该内容位于两个有规律性的标志之间，标志也可以是开始和结束。

通常想要匹配的内容不包含两边的“标志”，这就需要用**零宽断言**。简单来说，就是一种引导语法告诉既要匹配到“标志”，但又不包含“标志”。左边标志的引导语法是 (?<= 标志)，右边标志的引导语法是 (?= 标志)，而真正要匹配的内容放在它们中间。

比如，来自问卷星“来自 IP”数据，想要提取 IP、省份。

```
x = c("175.10.237.40(湖南-长沙)", "114.243.12.168(北京-北京)",
      "125.211.78.251(黑龙江-哈尔滨)")
# 提取省份
str_extract(x, "\\(.*-")            # 对比，不用零宽断言
```

```
## [1] "(湖南-"   "(北京-"   "(黑龙江-"
```

```
str_extract(x, "(?<=\\().*(?=-)")  # 用零宽断言
```

```
## [1] "湖南"   "北京"   "黑龙江"
```

```
# 提取 IP
str_extract(x, "\\d.*\\d")          # 直接匹配
```

```
## [1] "175.10.237.40"  "114.243.12.168" "125.211.78.251"
```

```
str_extract(x, "^.*(?=\\()")        # 用零宽断言
```

```
## [1] "175.10.237.40"  "114.243.12.168" "125.211.78.251"
```

省份位于两个标志 “(” 和 “-” 之间，但又不包含该标志，这就需要用到零宽断言。IP 位于两个标志 ‘‘开始” 和 “(” 之间，左边用开始符号 ^，右边用零宽断言。

再比如，用零宽断言提取专业（位于 ‘‘级’’ 和数字之间）：

```
x = c("18 级能源动力工程 2 班", "19 级统计学 1 班")
str_extract(x, "(?<= 级).*?(?=[0-9])")
```

```
## [1] "能源动力工程" "统计学"
```

关于懒惰匹配

正则表达式正常都是贪婪匹配，即重复直到文本中能匹配的最长范围，例如匹配小括号：

```
str_extract("(1st) other (2nd)", "\\(.+\\)")
```

```
## [1] "(1st) other (2nd)"
```

若想只匹配到第 1 个右小括号，则需要懒惰匹配，在重复匹配后面加上 ? 即可：

```
str_extract("(1st) other (2nd)", "\\(.+?\\)")
```

```
## [1] "(1st)"
```

例 1.4 分组捕获

正则表达式中可以用圆括号来分组，作用是

- 确定优先规则
- 组成一个整体
- 拆分出整个匹配中的部分内容（称为捕获）
- 捕获内容供后续引用或者替换。

比如，来自瓜子二手车的数据：若型号是中文，则品牌与型号中间有空格；若型号为英文或数字，则品牌与型号中间没有空格。

若用正则表达式匹配 ‘‘字母或数字” 并分组，然后捕获该分组并用添加空格替换：

```
x = c(" 宝马 X3 2016 款", " 大众 速腾 2017 款", " 宝马 3 系 2012 款")
str_replace(x, "([a-zA-Z0-9])", " \\1")
```

```
## [1] "宝马 X3 2016款"    "大众 速腾  2017款" "宝马 3系 2012款"
```

后续再用空格分割列即可。更多分组的引用还有 \\2, \\3, ...

最后，再推荐一个来自 Github 可以推断正则表达式的包 inferregex：

```
library(inferregex)
infer_regex("abcd-9999-ab9")$regex
```

```
## [1] "^[a-z]{4}-\\d{4}-[a-z]{2}\\d$"
```

本节部分内容参阅 ‘‘正则表达式 30 分钟入门教程’’、(Hadley Wickham 2017)、(李东风2020).