[一、基础 2](#_Toc531094545)

[二、字符转义 3](#_Toc531094546)

[三、重复 3](#_Toc531094547)

[四、字符类 3](#_Toc531094548)

[五、分枝条件 4](#_Toc531094549)

[六、分组 4](#_Toc531094550)

[七、反义 5](#_Toc531094551)

[八、后向引用 5](#_Toc531094552)

[8.1零宽断言 6](#_Toc531094553)

[8.2 负向零宽断言 7](#_Toc531094554)

[8.3注释 7](#_Toc531094555)

[九、贪婪与懒惰 8](#_Toc531094556)

[十、附录 9](#_Toc531094557)

## 

## 一、基础

1、精确查找某个单词：\b（你所要查找的单词）\b 示例：\bhi\b

\b：单词的开头或结尾，也就是单词的分界处

2、假如你要找的是hi后面不远处跟着一个Lucy： \bhi\b.\*\bLucy\b

. ：是另一个元字符，匹配除了换行符以外的任意字符

\* ：同样是元字符，不过它代表的不是字符，也不是位置，而是数量，可以为0次

+ ：数量，最少是一次

因此：.\* 连在一起就意味着任意数量的不包含换行的字符

3、以0开头，然后是两个数字，然后是一个连字号“-”，最后是8个数字：0\d\d-\d\d\d\d\d\d\d\d 等于：0\d{2}-\d{8}

\d：是个新的元字符，匹配一位数字(0，或1，或2，或……)

- ：不是元字符，只匹配它本身——连字符(或者减号，或者中横线，或者随你怎么称呼它)。

4、\s匹配任意的空白符，包括空格，制表符(Tab)，换行符，中文全角空格等。

5、\w匹配字母或数字或下划线或汉字等。

6、匹配以字母a开头的单词——先是某个单词开始处(\b)，然后是字母a,然后是任意数量的字母或数字(\w\*)，最后是单词结束处(\b)： \ba\w\*\b

7、匹配1个或更多连续的数字: \d+

8、匹配刚好6个字符的单词：\b\w{6}\b

9、匹配字符串的开始： ^

10、匹配字符串的结束： $

和忽略大小写的选项类似，有些正则表达式处理工具还有一个处理多行的选项。如果选中了这个选项，^和$的意义就变成了匹配行的开始处和结束处。

11、一个网站如果要求你填写的QQ号必须为5位到12位数字时：^\d{5,12}$

{5,12}：则是重复的次数不能少于5次，不能多于12次，否则都不匹配。

## 二、字符转义

如果你想查找元字符本身的话，比如你查找.,或者\*,就出现了问题：你没办法指定它们，因为它们会被解释成别的意思。这时你就得使用\来取消这些字符的特殊意义。因此，你应该使用\.和\\*。当然，要查找\本身，你也得用\\.

例如：deerchao\.net匹配deerchao.net，C:\\Windows匹配C:\Windows。

## 三、重复

\* ：匹配一次或多次

+ ：匹配0次或多次

？：匹配0次或1次

{n}：匹配n次

{n,}：重复n次或更多次

{m,n}：重复m次到n次

举例：Windows\d+：匹配Windows后面跟1个或更多数字

^\w+匹配一行的第一个单词(或整个字符串的第一个单词，具体匹配哪个意思得看选项设置)

## 四、字符类

1、想匹配没有预定义元字符的字符集合(比如元音字母a,e,i,o,u)，需要在方括号里列出它们：

[aeiou]：匹配任何一个英文元音字母

[.?!]：匹配标点符号(.或?或!)

2、字符范围

[0-9]：代表的含意与\d就是完全一致的：一位数字；

[a-z0-9A-Z\_]：完全等同于\w（如果只考虑英文的话）。

\(?0\d{2}[) -]?\d{8}：这个表达式可以匹配几种格式的电话号码，像(010)88886666，或022-22334455，或02912345678等。

解释：首先是一个转义字符\(,它能出现0次或1次(?),然后是一个0，后面跟着2个数字(\d{2})，然后是)或-或空格中的一个，它出现1次或不出现(?)，最后是8个数字(\d{8})。

## 五、分枝条件

规则：如果满足其中任意一种规则都应该当成匹配

方法：用 | 把不同的规则分隔开

1、这个表达式能匹配两种以连字号分隔的电话号码：一种是三位区号，8位本地号(如010-12345678)，一种是4位区号，7位本地号(0376-2233445)：0\d{2}-\d{8}|0\d{3}-\d{7}

2、匹配3位区号的电话号码，其中区号可以用小括号括起来，也可以不用，区号与本地号间可以用连字号或空格间隔，也可以没有间隔： \(0\d{2}\)[- ]?\d{8}|0\d{2}[- ]?\d{8}

3、2、匹配3位区号或者4位区号的电话号码，其中区号可以用小括号括起来，也可以不用，区号与本地号间可以用连字号或空格间隔，也可以没有间隔：

\(0\d{2}\)[- ]?\d{8}|0\d{2}[- ]?\d{8}|\(0\d{3}\)[- ]?\d{7}|0\d{3}[- ]?\d{7}

4、美国邮编的规则是5位数字，或者用连字号间隔的9位数字：\d{5}-\d{4}|\d{5}

但是注意不能写成：\d{5}|\d{5}-\d{4}

原因是匹配分枝条件时，将会从左到右地测试每个条件，如果满足了某个分枝的话，就不会去再管其它的条件了。

## 六、分组

重复多个字符

IP地址匹配表达式：(\d{1,3}\.){3}\d{1,3}

用括号括起来你要分组的对象

( (2[0-4]\d | 25[0-5] | [01]?\d\d?)\. ) {3} (2[0-4]\d | 25[0-5] | [01]?\d\d? )

## 七、反义

有时需要查找不属于某个能简单定义的字符类的字符。比如想查找除了数字以外，其它任意字符都行的情况，这时需要用到反义：

（1） \W ：匹配任意不是字母，数字，下划线，汉字的字符

（2） \S ：匹配任意不是空白符的字符

（3） \D ：匹配任意非数字的字符

（4） \B ：匹配不是单词开头或结束的位置

（5） [^x]：匹配除了x以外的任意字符

（6） [^aeiou]：匹配除了aeiou这几个字母以外的任意字符

匹配不包含空白符的字符串：\S+

匹配用尖括号括起来的以a开头的字符串：<a[^>]+>

## 八、后向引用

使用小括号指定一个子表达式后，匹配这个子表达式的文本(也就是此分组捕获的内容)可以在表达式或其它程序中作进一步的处理。 默认情况下，每个分组会自动拥有一个组号，规则是：从左向右，以分组的左括号为标志，第一个出现的分组的组号为1，第二个为2，以此类推。

1、后向引用用于重复搜索前面某个分组匹配的文本。例如，\1代表分组1匹配的文本。

例子：

\b(\w+)\b\s+\1\b ：go go 或者 hello hello这样的单词

2、自己指定子表达式的组名：要指定一个子表达式的组名，请使用这样的语法：(?<Word>\w+)(或者把尖括号换成'也行：(?'Word'\w+)），这样就把\w+的组名指定为Word了。要反向引用这个分组捕获的内容，你可以使用\k<Word>

\b(?<Word>\w+)\b\s+\k<Word>\b 等同于

常用的分组语法：

（1）捕获：

|  |  |
| --- | --- |
| **(exp)** | 匹配exp,并捕获文本到自动命名的组里 |
| **(?<Name>exp)或者(?'Name'exp)** | 匹配exp,并捕获文本到名称为name的组里 |
| **(?:exp)** | 匹配exp,不捕获匹配的文本，也不给此分组分配组号 |

（2）零宽断言：

|  |  |
| --- | --- |
| **(?<=exp)** | 匹配exp后面的位置 |
| **(?!exp)** | 匹配后面跟的不是exp的位置 |
| **(?<!exp)** | 匹配前面不是exp的位置 |

（3）注释:

|  |  |
| --- | --- |
| **(?#comment)** | 这种类型的分组不对正则表达式的处理产生任何影响，用于提供注释让人阅读 |

### 8.1零宽断言

1、匹配以ing结尾的单词的前面部分(除了ing以外的部分)，如查找I'm singing while you're dancing.时，它会匹配sing和danc： \b\w+(?=ing\b)

2、匹配以re开头的单词的后半部分(除了re以外的部分)，例如在查找reading a book时，它匹配ading：(?<=\bre)\w+\b

3、假如你想要给一个很长的数字中每三位间加一个逗号(当然是从右边加起了)，你可以这样查找需要在前面和里面添加逗号的部分：((?<=\d)\d{3})+\b，用它对1234567890进行查找时结果是234567890。

4、匹配以空白符间隔的数字(再次强调，不包括这些空白符)：(?<=\s)\d+(?=\s)

5、(?<=<(\w+)>).\*(?=<\/\1>)：匹配不包含属性的简单HTML标签内里的内容 比如<body>1234</body>匹配的是1234

### 8.2 负向零宽断言

前面我们提到过怎么查找不是某个字符或不在某个字符类里的字符的方法(反义)。但是如果我们只是想要确保某个字符没有出现，但并不想去匹配它:

例如：如果我们想查找这样的单词--它里面出现了字母q,但是q后面跟的不是字母u：\b\w\*q[^u]\w\*\b

但是如果这样写的话，比如Iraq、Benq（报错），Iraq fighting这种字符会将全部匹配

所以我们使用：\b\w\*q(?!u)\w\*\b

1、

(?!exp)：断言此位置的后面不能匹配表达式exp。

\d{3}(?!\d)：匹配三位数字，而且这三位数字的后面不能是数字

\b((?!abc)\w)+\b：匹配不包含连续字符串abc的单词。

2、

(?<!exp)：零宽度负回顾后发断言来断言此位置的前面不能匹配表达式exp

(?<![a-z])\d{7}：匹配前面不是小写字母的七位数字。

### 8.3注释

小括号的另一种用途是通过语法(?#comment)来包含注释。

例如：

2[0-4]\d(?#200-249)|25[0-5](?#250-255)|[01]?\d\d?(?#0-199)。这个正则匹配的数字范围是200-249

要包含注释的话，最好是启用“忽略模式里的空白符”选项，这样在编写表达式时能任意的添加空格，Tab，换行，而实际使用时这些都将被忽略。启用这个选项后，在#后面到这一行结束的所有文本都将被当成注释忽略掉。例如，我们可以前面的一个表达式写成这样：

(?<= # 断言要匹配的文本的前缀

<(\w+)> # 查找尖括号括起来的字母或数字(即HTML/XML标签)

) # 前缀结束

.\* # 匹配任意文本

(?= # 断言要匹配的文本的后缀

<\/\1> # 查找尖括号括起来的内容：前面是一个"/"，后面是先前捕获的标签

) # 后缀结束

## 九、贪婪与懒惰

当正则表达式中包含能接受重复的限定符时，通常的行为是（在使整个表达式能得到匹配的前提下）匹配尽可能多的字符。以这个表达式为例：a.\*b，它将会匹配最长的以a开始，以b结束的字符串。如果用它来搜索aabab的话，它会匹配整个字符串aabab。这被称为贪婪匹配。

有时，我们更需要懒惰匹配，也就是匹配尽可能少的字符。前面给出的限定符都可以被转化为懒惰匹配模式，只要在它后面加上一个问号?。这样.\*?就意味着匹配任意数量的重复，但是在能使整个匹配成功的前提下使用最少的重复。现在看看懒惰版的例子吧：

a.\*?b匹配最短的，以a开始，以b结束的字符串。如果把它应用于aabab的话，它会匹配aab（第一到第三个字符）和ab（第四到第五个字符）。

懒惰限定符：

|  |  |
| --- | --- |
| \*? | 重复任意次，但尽可能少重复 |
| +? | 重复1次或更多次，但尽可能少重复 |
| ?? | 重复0次或1次，但尽可能少重复 |
| {n,m}? | 重复n到m次，但尽可能少重复 |
| {n,}? | 重复n次以上，但尽可能少重复 |

## 十、附录

| 代码/语法 | 说明 |
| --- | --- |
| \a | 报警字符(打印它的效果是电脑嘀一声) |
| \b | 通常是单词分界位置，但如果在字符类里使用代表退格 |
| \t | 制表符，Tab |
| \r | 回车 |
| \v | 竖向制表符 |
| \f | 换页符 |
| \n | 换行符 |
| \e | Escape |
| \0nn | ASCII代码中八进制代码为nn的字符 |
| \xnn | ASCII代码中十六进制代码为nn的字符 |
| \unnnn | Unicode代码中十六进制代码为nnnn的字符 |
| \cN | ASCII控制字符。比如\cC代表Ctrl+C |
| \A | 字符串开头(类似^，但不受处理多行选项的影响) |
| \Z | 字符串结尾或行尾(不受处理多行选项的影响) |
| \z | 字符串结尾(类似$，但不受处理多行选项的影响) |
| \G | 当前搜索的开头 |
| \p{name} | Unicode中命名为name的字符类，例如\p{IsGreek} |
| (?>exp) | 贪婪子表达式 |
| (?<x>-<y>exp) | 平衡组 |
| (?im-nsx:exp) | 在子表达式exp中改变处理选项 |
| (?im-nsx) | 为表达式后面的部分改变处理选项 |
| (?(exp)yes|no) | 把exp当作零宽正向先行断言，如果在这个位置能匹配，使用yes作为此组的表达式；否则使用no |
| (?(exp)yes) | 同上，只是使用空表达式作为no |
| (?(name)yes|no) | 如果命名为name的组捕获到了内容，使用yes作为表达式；否则使用no |
| (?(name)yes) | 同上，只是使用空表达式作为no |

java匹配正则：

获取最后一行的日期：

(?<=出生日期)(?<jltzssj>**[**\s\S**]**\*)

**public** **static** **final** Pattern *XWDD* = Pattern.*compile*("(?:地点[\\s]\*)(?<XWDD>.\*?)(?:[\\s]\*(讯问|询问))");

/\*\*

\* 保留方法

\* 讯问地点不合法

\* **@param** zhzjDocument

\* **@param** zhzjDocFileList

\* **@param** map

\* **@return**

\*/

**public** Boolean isXwd(ZhzjDocument zhzjDocument, List<ZhzjDocFile> zhzjDocFileList, Map<String, Object> map) {

String xwdd = "";

**if** (xwbltype.equals(zhzjDocument.getDocType())) {

// 获取当前询问笔录询问时间

**for** (ZhzjDocFile zhzjDocFile : zhzjDocFileList) {

String str = zhzjDocFile.getContent();

Matcher XwddMatcher = *XWDD*.matcher(str);

**if** (XwddMatcher.find()) { 正则里面按照XWDD分组

xwdd = XwddMatcher.group("XWDD").replaceAll("[\\s]", StringUtils.*EMPTY*);

**for** (**int** i = 0; i < GJZ.length; i++) {

**if** (xwdd.contains(GJZ[i])) {

**return** Consts.*CHECKPOINT\_NOTPOINT*;

}

}

} **else** {

**return** Consts.*CHECKPOINT\_NOTPOINT*;

}

}

}

**return** Consts.*CHECKPOINT\_ISPOINT*;

}