目录

[1:字符分类 1](#_Toc380590889)

[1.1：非打印字符 1](#_Toc380590890)

[1.2:特殊字符 2](#_Toc380590891)

[2:字符集 3](#_Toc380590892)

[2.1:基本概念 4](#_Toc380590893)

[2.2: 在字符集中使用元字符 4](#_Toc380590894)

[2.3:预定义字符集 6](#_Toc380590895)

[3:限定符 7](#_Toc380590896)

[4:定位符 8](#_Toc380590897)

[4.1:基本概念 9](#_Toc380590898)

[4.2:词的边界 9](#_Toc380590899)

[4.3 ^,$和\b的区别 11](#_Toc380590900)

[5: “.”元字符 13](#_Toc380590901)

[6: 用 “|”表示选择 14](#_Toc380590902)

[7: 用 “（）”表示分组 15](#_Toc380590903)

[8: “?”的补充说明 15](#_Toc380590904)

[9: 给正则表达式添加注释 16](#_Toc380590905)

[10: 操作符的运算优先级 16](#_Toc380590906)

[11：综合例子 17](#_Toc380590907)

[1:电话号码 17](#_Toc380590908)

[2:ip地址 18](#_Toc380590909)

# 1:字符分类

## 1.1：非打印字符

|  |  |
| --- | --- |
| **Symbol** | **Description** |
| /cx | 匹配由x指明的控制字符。例如， /cM 匹配一个 Control-M 或回车符。x 的值必须为 A-Z 或 a-z 之一。否则，将 c 视为一个原义的 'c' 字符。 |
| /f | 匹配一个换页符。等价于 /x0c 和 /cL。 |
| /n | 匹配一个换行符。等价于 /x0a 和 /cJ。 |
| /r | 匹配一个回车符。等价于 /x0d 和 /cM。 |
| /s | 匹配任何空白字符，包括空格、制表符、换页符等等。等价于 [ /f/n/r/t/v]。 |
| /S | 匹配任何非空白字符。等价于 [^ /f/n/r/t/v]。 |
| /t | 匹配一个制表符。等价于 /x09 和 /cI。 |
| /v | 匹配一个垂直制表符。等价于 /x0b 和 /cK。 |

## 1.2:特殊字符

特殊字符也叫做**元字符**，保留字符（Metacharactor），在Regex中表示特殊的意义，大部分的意思在不同的上下文中的意义是不同的，这里只列出最普遍的意义。

特殊字符共有11个：

|  |  |
| --- | --- |
| **Symbol** | **Description** |
| $ | 匹配输入字符串的结尾位置。如果设置了 RegExp 对象的 Multiline 属性，则 $ 也匹配 '/n' 或 '/r'。要匹配 $ 字符本身，请使用 /$。 |
| ( ) | 标记一个子表达式的开始和结束位置。子表达式可以获取供以后使用。要匹配这些字符，请使用 /( 和 /)。 |
| \* | 匹配前面的子表达式零次或多次。要匹配 \* 字符，请使用 /\*。 |
| + | 匹配前面的子表达式一次或多次。要匹配 + 字符，请使用 /+。 |
| . | 匹配除换行符 /n之外的任何单字符。要匹配 .，请使用 /。 |
| [ | 标记一个中括号表达式的开始。要匹配 [，请使用 /[。 |
| ? | 匹配前面的子表达式零次或一次，或指明一个非贪婪限定符。要匹配 ? 字符，请使用 /?。 |
| / | 将下一个字符标记为或特殊字符、或原义字符、或反向引用、或八进制转义符。例如， 'n' 匹配字符 'n'。'/n' 匹配换行符。序列 '//' 匹配 "/"，而 '/(' 则匹配 "("。 |
| ^ | 匹配输入字符串的开始位置，除非在方括号表达式中使用，此时它表示不接受该字符集合。要匹配 ^ 字符本身，请使用 /^。 |
| { | 标记限定符表达式的开始。要匹配 {，请使用 /{。 |
| | | 指明两项之间的一个选择。要匹配 |，请使用 /|。 |

在元字符前加 / 转义符，可以把特殊字符当作普通字符来使用。

比如：如要要匹配 1+1=2 ，正确的正则表达式应该为1/+1=2。否则， + 会被当作特殊字符对待。

除了特殊字符，所有的其他字符都不应该加 / 。因为 / 也是一个特殊字符。/ 和普通字符组合在一起也可以创造一种特殊的意义。比如 /d 表示匹配所有的数字。

作为程序员，单引号和双引号不是特殊字符会也许让我们感到很惊讶。但这是正确的。因为我们在编程的时候，编程语言会知道引号之间的哪些字符表示特殊意义，编译器在把字符串x传递给regex解释引擎之前，会把它们处理成regex。比如，在C#中，如果我们要匹配 1/+1=2 ，在程序中我们要这样写： “1//+1=2” ，C#编译器会把 “//” ，处理为一个“/” 。同样，如果要匹配 C:/Temp ，首先，正则表达式要这样写 C://Temp，然后在程序中我们应该这样写：“ C:////temp”。

# 2:字符集

## 2.1:基本概念

字符集描述了一组字符，Regex解释器会认为匹配字符集中的一个字符就可以认为匹配成功。

字符集用[ ]括起来即可。比如gr[ae]y就可以匹配gray或者grey。

字符集只能匹配一个字符，gr[ae]y就不能和graey匹配。字符集中的字符顺序是任意的，得到的结果都是唯一的。

可以在字符集中用连字符“-”来表示一个范围。[0-9]的结果和[0123456789]的匹配结果都是相同的。字符集中的范围可以有多种。比如[0-9a-fA-F]表示匹配所有的16进制，包括大小写。也可以把范围和单个字符组合在一起用,[0-9a-fxA-FX]表示匹配所有的16进制或者一个字符X。字符集的顺序不会影响结果。

在字符集的开始标志“[”后面加上一个“^”符号，表示否定，表示匹配除字符集中定义的字符以外的所有字符。包括非打印字符和行结束符。

**注意：**字符集匹配的一个字符，而不是一个位置。所以。q[^u]的意义不是“q后面的字符不是u”。而是“q后面的字符可以是除了u以外的所有字符”。

q[^u]不会和Iraq匹配。

但是会和Iraq is a country匹配，因为q后面的空格字符是一个“不是u的字符”。

### 2.2: 在字符集中使用元字符

字符集中的元字符只能是 ‘]’, ‘/’, ‘^’, 和 ‘-‘ 。

其他元字符在字符集中都失去了特殊意义，表示的只是一个普通字符。也不需要用加“/”。

比如：

匹配一个“\*”或者“+”，用[\*+]就足够了。即使给他们加上“/”，regex解释器也会把他们忽略掉。

**四种特殊字符的处理：**

在字符集中要表示“]”，“^”和“-”需要在后面加上转义符“/”，来表示它们代表的分别是普通字符“]”，“^”和“-”。

**也可以把它们放在一个不能表示特殊意义的位置，后一个方法比较好**，因为他们不会影响可读性。

* “^”

要想匹配一个“^”，可以把它放在除了紧跟“[”的任意一个位置。

|  |  |  |
| --- | --- | --- |
| Regex | String | Description |
| [x^] | A string with x and ^. | 匹配x或者“^” |

* “]”

可以把“]”放在紧跟着“[”的位置，或者使用否定字符集。

|  |  |  |
| --- | --- | --- |
| Regex | String | Description |
| **[]x]** | **A string with x and ]** | 匹配x或者“]” |
| [^]x] | A string with x and ] | 匹配除了x和”] ”以外的所有字符 |

* “/”

要想把“/”当作一个普通字符来匹配，而不是一个特殊字符，必须把“/”再用一个“/”括起来。

|  |  |  |
| --- | --- | --- |
| Regex | String | Description |
| [//x] | A string with x and / | 匹配x或者“/” |

* “-”

连字符可以放在紧跟着“[”的后面，或者正好“]”的前面，或者紧跟着“^”的后面。

|  |  |  |
| --- | --- | --- |
| Regex | String | Description |
| **[-x]** | **A string with x and -** | 匹配x或者“-” |
| [x-] | A string with x and - | 匹配x或者“-” |

## 2.3:预定义字符集

|  |  |  |
| --- | --- | --- |
| Regex | Meaning | Description |
| /d | [0-9] | 所有数字 |
| /w | [a-zA-Z] | 表示所有的字符，和文化字体有关 |
| /s | [ /t/r/n] | 空格，回车和tab。和文化字体有关 |

同样，在预定义字符集前面加一个“^”符号表示否定。它们也有预先定义好的表示：

|  |  |  |
| --- | --- | --- |
| Regex | Meaning | Description |
| /D | [^/d] | 非数字 |
| /W | [^/w] | 非字符，和文化字体有关 |
| /S | [^/s] | 非空格，回车和tab。和文化字体有关 |

 在“[]”使用否定预订一字符集时要特别心。[/D/S]不等于[/^d/s]。[/^d/s]会匹配除了数字和空白符以外的所有字符。而[/D/S]会匹配要么不是一个数字，要么是空白符。因为数字不是空白符，空白符也不是数字，所以[/D/S]会匹配任意的字符。

# 3:限定符

限定符提供了一种简单方法，用于指定在模式中允许特定字符或字符集自身重复出现的次数。限定符始终引用限定符前（左边）的模式，通常是单个字符，除非使用括号创建模式组。

限定符有\*或+或?或{n}或{n,}或{n,m}共6种。

|  |  |  |
| --- | --- | --- |
| Symbol | Description | Description |
| ? | 0次获1次 |  |
| \* | 0次或n次 |  |
| + | 1次或n次 |  |
| {min, max} | 最少min次，最多max次 | Max必须大于等于min。 |
| {min，<不指定> } | 最少min次，或者n次 |  |
| {min} | 精确的重复min次 |  |

在字符集后面使用 “?”，”\*”，”+”，表示重复。会重复整个的字符集，而不是重复匹配的那个字符。

|  |  |  |
| --- | --- | --- |
| Regex | String | 意义 |
| **[0-9]+** | **846，111** | 匹配数字 |
| ([0-9))+ | 846，111 | 匹配数字相同的数字 |

[0-9]+会匹配846，也会匹配111。

如果想要重复的只是匹配的那个字符，而不是整个字符集，则必须使用**“反向引用”**。

([0-9])/1+ 只会匹配111，而不会匹配846。

0\d\d-\d\d\d\d\d\d\d\d匹配这样的字符串：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(也就是中国的电话号码。当然，这个例子只能匹配区号为3位的情形)。

这里的\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)。

为了避免那么多烦人的重复，我们也可以这样写这个表达式：0\d{2}-\d{8}。这里\d后面的{2}({8})的意思是前面\d必须连续重复匹配2次(8次)。

# 4:定位符

## 4.1:基本概念

到现在为止，我们已经熟悉了普通字符、特殊字符（元字符）和字符集。在这两种情况下，Regex匹配的都是一个字符。

定位符是另外一种，它不匹配字符，相反，它匹配的是一个位置。

定位符有几种：

|  |  |  |
| --- | --- | --- |
| Regex | Function | Description |
| ^ | 第一个字符之前的位置 | 包含换行符 |
| $ | 最后一个字符后面的位置 | 包含换行符 |
| /A | 总是匹配string的第一个位置 | 不包含换行符 |
| /Z | 总是匹配string的最后一个位置 | 不包含换行符 |

|  |  |  |
| --- | --- | --- |
| Regex | String | 意义 |
| **^** | **Abc** | 匹配A之前的位置 |
| **$** | **Abc** | 匹配c后面的位置 |
| **^A** | **Abc** | 匹配A |
| **^b** | **Abc** | **不能匹配** |
| **c$** | **Abc** | 匹配c |
| A$ | Abc | **不能匹配** |

## 4.2:词的边界

还有一种定位符匹配的是一个词（word）的边界。用**/b**表示。

词（word）是由可以组成词的字符组成（“word characters”），“word characters”就是可以组成词的字符，不包括非打印字符和回车换行。

有四种不同的位置被认为是词的边界：

1. 第一个字符之前的位置，如果第一个字符是一个“word character”。
2. 最后一个字符之后的位置，如果最后一个字符是一个“word character”。
3. 介于词和非词之间的紧跟着词的位置。
4. 介于非词和词之间的紧跟着非词的位置

所有的word characters都可以用/w来表示。

所有的non-word characters都可以用/W来表示。

 /b匹配一个词的边界。

/B表示一个非词的边界的位置，它能匹配任意一个不是次的边界的位置。

假设你在一篇英文小说里查找hi，你可以使用正则表达式hi。

这几乎是最简单的正则表达式了，它可以精确匹配这样的字符串：由两个字符组成，前一个字符是h,后一个是i。通常，处理正则表达式的工具会提供一个忽略大小写的选项，如果选中了这个选项，它可以匹配hi,HI,Hi,hI这四种情况中的任意一种。

不幸的是，很多单词里包含hi这两个连续的字符，比如him,history,high等等。用hi来查找的话，这里边的hi也会被找出来。如果要精确地查找hi这个单词的话，我们应该使用\bhi\b。

\b是正则表达式规定的一个特殊代码（好吧，某些人叫它元字符，metacharacter），代表着单词的开头或结尾，也就是单词的分界处。虽然通常英文的单词是由空格，标点符号或者换行来分隔的，但是\b并不匹配这些单词分隔字符中的任何一个，它**只匹配一个位置**。

如果需要更精确的说法，\b匹配这样的位置：它的前一个字符和后一个字符不全是(一个是,一个不是或不存在)\w。

假如你要找的是hi后面不远处跟着一个Lucy，你应该用\bhi\b.\*\bLucy\b。

这里，.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——它指定\*前边的内容可以连续重复使用任意次以使整个表达式得到匹配。因此，.\*连在一起就意味着任意数量的不包含换行的字符。现在\bhi\b.\*\bLucy\b的意思就很明显了：先是一个单词hi,然后是任意个任意字符(但不能是换行)，最后是Lucy这个单词。

## 4.3 ^,$和\b的区别

^,$用来匹配字符，适用于对一些注册信息的验证，小而短的文本，而\b用来匹配在一大段文本中某一个单词

现在你已经知道几个很有用的元字符了，如\b,.,\*，还有\d.正则表达式里还有更多的元字符，比如\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。\w匹配字母或数字或下划线或汉字等。

下面来看看更多的例子：

\ba\w\*\b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)。

\d+匹配1个或更多连续的数字。这里的+是和\*类似的元字符，不同的是\*匹配重复任意次(可能是0次)，而+则匹配重复1次或更多次。

\b\w{6}\b 匹配刚好6个字符的单词。

元字符^（和数字6在同一个键位上的符号）和$都匹配一个位置，这和\b有点类似。^匹配你要用来查找的字符串的开头，$匹配结尾。这两个代码在验证输入的内容时非常有用，比如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，可以使用：^\d{5,12}$。

这里的{5,12}和前面介绍过的{2}是类似的，只不过{2}匹配只能不多不少重复2次，{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。

因为使用了^和$，所以输入的整个字符串都要用来和\d{5,12}来匹配，也就是说整个输入必须是5到12个数字，因此如果输入的QQ号能匹配这个正则表达式的话，那就符合要求了。

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。

# 5: “.”元字符

在正则表达式中，“.”是用的最多的一个元字符，同时，它也是最容易用错的一个。所以我们单独来讲。

**“.”**几乎匹配任何字符。唯一的一个例外是**换行符。**

这个例外存在是有历史原因的。第一个用正则表达式的工具是基于换行符的。它从文件中读取一行字符，然后用去匹配。因为在这些工具中，string中永远不可能有换行符，所以**“.”**也永远不会和换行符匹配。

现代的工具可以用正则表达式去和很大的一个string甚至整个文件去匹配。所以现在的Regex解释器都含有一个选项，激活以后就可以让**“.”**去匹配所有的字符，包括换行符。

**“.”**是一个非常强大的元字符。它可以让我们偷懒。但是我们要慎重的使用。我们看一个例子：

我们要匹配一个mm/dd/yy的格式的日期。但是我们可以让用户指定日期的分割符。一个简单的Regex是：/d/d./d/d./d/d这个看起来可以实现。它会很好的匹配04/09/07。问题是： 04409407也会被匹配。因为第三个4和第五个4都会被“.”匹配。这不是我们想要得结果。

/d/d[-/.]/d/d[-/.]/d/d是一个比上面的好的一个方法，用户可以指定“-”,“.”,“/”作为日期的分割符。因为“.”在字符集中不表示一个特殊字符，所以我们不需要在“.”之前加”/”。

但这个方法还不完美，它会匹配 99/99/99 , [0-1]/d[-/.][0-3]/d[-/.]/d/d也许会好一些。虽然它仍旧会匹配19/39/99, 。方法够用就好了，不必追求完美，如果这个是用来验证用户需求，可能还需要改进，如果只是用来分析一段code,或许已经足够了。

如果我们要匹配一段带双引号的字符串。听起来很容易，我们可以在两个双引号之间放任意多个任意字符。Regex可能会这么写：“.\*”，这个会匹配put a “string” between double quotes.结果是对的，但是如果去匹配“"string one” and “string two””则得到的结果会是　“string one” and “string two” 。这不是我们要的结果。

所以这里我们可以用否定字符集来代替“[^”/r/n]\*”

# 6: 用 “|”表示选择

前面已经讲过，用字符集可以匹配很多字符其中的一个，替换的作用稍有不同。

如果需要匹配 cat 或者 dog，可以这样写：cat|dog，也可以添加很多：cat|dog|mouse|fish。

但是注意：“|”是正则表达式中优先级最低的操作符。Regex解释器在匹配的时候，要么匹配“|”左边的所有，要么匹配“|”右边的所有。

# 7: 用 “（）”表示分组

可以使用圆括号来限制选择的范围。

上面的例子，如果想要限制替换，可以使用“（）”符号。

比如： 如果我们要匹配整个词而不是一个词的一部分。Regex可以这样写：/b(cat|dog)/b。

这告诉regex解释器先去寻找一个边界，然后要么是cat，要么是dog，然后在去寻找一个边界。如果忽略掉括号，regex解释器会这样来匹配：要么是cat跟在一个边界的后面，要么是dog后面有一个边界。

# 8: “?”的补充说明

**“？”除了表示重复之外，还表示可选。**

例如：colou?r，会匹配color和colour。

用括号括起来的表示这組是一个可选的项目。

例如：Nov(ember)?会匹配Nov和November。

用“？”标记起来，等于告诉regex解释器有两种选项：要么匹配括起来的，要么不匹配。但是，**Regex解释器总会首先去匹配括起来的部分**，只有当这个失败了，才会当做忽略处理。

效果就是，如果用Feb 23(rd)?去匹配Today is Feb 23rd, 2004，结果总是Feb 23rd，而不是Feb 23。

**“？”也称作“懒元字符”,因为它总是尽可能的少的去匹配。**

# 9: 给正则表达式添加注释

可以这样给正则表达式添加注释：

（?#comment here）

# 10: 操作符的运算优先级

|  |  |  |
| --- | --- | --- |
| Symbol | Function | Memo |
| / | 转义符 |  |
| (), (?:), (?=), [] | 括号 |  |
| \*, +, ?, {n}, {n,}, {n,m} | 限定符 |  |
| ^, $, /anymetacharacter | 定位符 |  |
| | | 或 |  |

# 11：综合例子

## 1:电话号码

下面是一个更复杂的表达式：\(?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。我们对它进行一些分析吧：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})。

不幸的是，刚才那个表达式也能匹配010)12345678或(022-87654321这样的“不正确”的格式。要解决这个问题，我们需要用到分枝条件。正则表达式里的分枝条件指的是有几种规则，如果满足其中任意一种规则都应该当成匹配，具体方法是用|把不同的规则分隔开。听不明白？没关系，看例子：

0\d{2}-\d{8}|0\d{3}-\d{7}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{2}[- ]?\d{8}这个表达式匹配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。你可以试试用分枝条件把这个表达式扩展成也支持4位区号的。

\d{5}-\d{4}|\d{5}这个表达式用于匹配美国的邮政编码。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。之所以要给出这个例子是因为它能说明一个问题：**使用分枝条件时，要注意各个条件的顺序**。如果你把它改成\d{5}|\d{5}-\d{4}的话，那么就只会匹配5位的邮编(以及9位邮编的前5位)。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。

## 2:ip地址

我们已经提到了怎么重复单个字符（直接在字符后面加上限定符就行了）；但如果想要重复多个字符又该怎么办？你可以用小括号来指定子表达式(也叫做分组)，然后你就可以指定这个子表达式的重复次数了，你也可以对子表达式进行其它一些操作(后面会有介绍)。

(\d{1,3}\.){3}\d{1,3}是一个简单的IP地址匹配表达式。要理解这个表达式，请按下列顺序分析它：\d{1,3}匹配1到3位的数字，(\d{1,3}\.){3}匹配三位数字加上一个英文句号(这个整体也就是这个分组)重复3次，最后再加上一个一到三位的数字(\d{1,3})。

不幸的是，它也将匹配256.300.888.999这种不可能存在的IP地址。如果能使用算术比较的话，或许能简单地解决这个问题，但是正则表达式中并不提供关于数学的任何功能，所以只能使用冗长的分组，选择，字符类来描述一个正确的IP地址：((2[0-4]\d|25[0-5]|[01]?\d\d?)\.){3}(2[0-4]\d|25[0-5]|[01]?\d\d?)。

理解这个表达式的关键是理解2[0-4]\d|25[0-5]|[01]?\d\d?，这里我就不细说了，你自己应该能分析得出来它的意义。

（IP地址中每个数字都不能大于255. 经常有人问我, 01.02.03.04 这样前面带有0的数字, 是不是正确的IP地址呢? 答案是: 是的, IP 地址里的数字可以包含有前导 0 (leading zeroes).）