## 正则表达式简介

### 什么是正则表达式 (Regular Expressions)

是指一个用来描述或者匹配一系列符合某个句法规则的字符串的单个字符串。有点拗口，呵呵，说白了就是用来描述字符串的字符串。更直白点说它就是一个字符串，呵呵，够直白吧，那它有什么用呢？ good question。

### 正则表达式的作用

用来对文本也就是字符串进行查找和替换。我估计有点电脑基础的人都知道Ctrl+F，当我们打开一个文件需要查找我们需要的内容时都会想到用Ctrl+F打开一个查找对话框，输入我们要查找的内容，点查找就OK了。但是这种查找方式的能力是非常有限的，我们只能查找一个字，一句话等等。当你觉得这种查找方式无法满足你要求的时候，正则表达式拯救你的时候到了，学习正则表达式的门槛非常低，而且易于学习，一旦你掌握了正则表达式你就会感叹当初没有正则表达式的日子是怎么过的。

### 支持正则表达式的工具

几乎所有的文本编辑器，如：UltraEdit，EmEditor 等。

几乎所有的编程语言，如：Java, VB.NET, C#, PHP, Perl, Python, Tcl 等。

好多基于Linux 的工具，如：grep，egrep，awk，sed，lex 等。

### 一个简单的例子

学习正则表达式我们需要一个运行和测试正则表达式的环境，当然上面提到的工具都可以，文本编辑器对初学者来说更直观一些，所以本人以后的所有例子都通过EmEditor 来讲解，EmEditor是一个非常不错的文本编辑器，它的免费版本就足够我们使用了。当然了，总有些人就是嫌麻烦不想装这些乱七八糟的软件，其实网上有好多在线测试正则表达式的网站，你可以在Google中搜索关键字正则表达式在线测试，这样就可以省去你安装软件的烦恼。

下面我们来看一个简单的例子，比如你想在一个文件中查找张三，很easy，Ctrl+F然后搜索张三就OK了，但是如果你想查找张三或李四，怎么办？一个很自然的想法是分两次进行，先查张三，然后查李四。这样做是可以的，有没有办法一次搞定呢，答案是肯定的，试着搜索一下下面的字符串查看结果：

**张三|李四**

啊？不好使啊？呵呵，别着急，选中搜索对话框中的使用正则表达式(Use Regular Expressions)再试一试。那这个正则表达式是什么意思呢？ 别着急，你很快就会知道。

### 谁适合学习正则表达式

不论你从事什么职业，如果你经常要进行文本操作，大量的重复性劳动你觉得很没意思，那么赶快学习正则表达式吧。如果你是程序员并且还不熟悉正则表达式，那么别人会鄙视你。你还在等什么呢？

## 匹配任何字符：点号(.)

假设现在让你把一个文件中的的英文句号(.)替换为中文句号(。)，是不是很easy， 呵呵，在 EmEditor 中Ctrl+H 调出Replace 对话框，在 Find 中输入英文句号(.)，在 Replace With 中输入中文句号(。)，然后轻轻点击 Replace Al l就搞定了。 我们按 Ctrl+Z 撤销我们刚才的替换，然后我们选中Replace 对话框中的 Use Regular Expressions，再次优雅的点击一下 Replace Al l，奇迹出现了，文件中所有的字都被替换为句号了，这不是我们想要的，为什么会出现这种情况呢？因为点号(.)在正则表达式中具有特殊意义，它可以代表任何字符。我们把像点号这类在正则表达式中具有特殊意义的字符称为元字符(Metacharacter)，正因为有了它们才成就了正则表达式强大的模糊匹配能力。

恭喜你，你已经学会了第一个元字符，我们将它添加到我们的工具箱中，别着急，你很快就会学会更多的元字符，到那时你将到达一个全新的高度。

|  |  |  |
| --- | --- | --- |
| **元字符** | **名称** | **匹配对象** |
| . | 点号(dot) | 单个任意字符 |

## 匹配行的起始(^)和结束($)

假设我们有这样一个文件，文件中的每一行包含一个公司名，我们想把这些公司名用双引号括起来，怎么办？用正则表达式解决这个问题很easy，我们分两步来进行：

1. 在 EmEditor 中Ctrl+H 调出 Replace 对话框，在 Find 中输入^，在 Replace With 中输入"，然后轻轻的点击 Replace Al l，我们就每一行的开头添加了双引号。

2. 在 EmEditor 中Ctrl+H 调出 Replace 对话框，在 Find 中输入$，在 Replace With 中输入"，然后优雅的点击 Replace Al l，我们就每一行的结尾添加了双引号。

啊？为什么在我的机器上不好使呢？呵呵，再次提醒，别忘选中 Replace 对话框中的 Use Regular Expressions 哦。

看到这里，如果你有这样的疑问，我们能不能一次搞定呢？ 如果你有这样的想法，你终将成大器。事实上确实可以而且也非常简单，试一试下面这个表达式吧：

**^|$**

竖线代表或的意思，所以上面的表达式的意思是把行头或行尾替换为双引号。

值得注意的是^和$匹配的是一个位置，也就是行首和行尾，并不匹配具体的字符。正则表达式的元字符一般有两类，一类匹配具体字符，一类匹配位置。别着急，你很快就会学会其他更强大的匹配位置的元字符。

恭喜你，你又学会了三个元字符。

|  |  |  |
| --- | --- | --- |
| **元字符** | **名称** | **匹配对象** |
| . | 点号(dot) | 单个任意字符 |
| ^ | 脱字符(caret) | 行的起始位置 |
| $ | 美元符(dollar) | 行的结束位置 |
| | | 竖线(bar) | 匹配分割两边的任意一个表达式 |

最后大家猜一猜下面的正则表达式是什么意思呢？ let me know your idea.

**^$**

## 字符组

我的英语很差，记性又不好，总是搞混 then 和 than，当我写完一篇英语文章，我想搜索一下这两个单词，看看有没有误用的情况，怎么办？虽然到目前为止你只学会了几个正则表达式元字符，但就解决这个问题来说已经足够了，在继续之前，停下来好好想想该用什么表达式来解决这个问题呢？

在 EmEditor 中Ctrl+F 调出 Find 对话框，在 Find 中输入 **then|than**，然后点击 Find Down 就OK了。恭喜你，你已经用正则表达式解决了你的第一个问题，这是一个了不起的进步，学会正则表达式很容易，但想灵活应用是非常难的。你已经有了很好的开始。

事实上，我们还有其他办法解决这个问题，试一试下面这个表达式：

**th[ae]n**

是不是有异曲同工之妙啊。其中的方括号在正在表达式中是一个元字符，它表示**匹配若干字符之一**，所以上面表达式的意思是匹配t，然后是h，然后是a或e，然后是n。怎么样？还好理解吧。

如果我想匹配一个数字，怎么办？好好想想，聪明的人已经想到了，如下：

**[0123456789]**

如果匹配一个数字需要这么复杂，这是不可接受的，事实上我们可以这么写：

**[0-9]**

是不是好多了。如果我想匹配一个小写字母，怎么办？ easy， 如下：

**[a-z]**

大写字母呢？ 还是easy，如下：

**[A-Z]**

如果我想匹配一个数字或小写字母或大写字母，怎么办？呵呵，有人看到**或**字可能要这么写：

**[0-9]|[a-z]|[A-Z]**

这是可以的，还可以怎么办？大胆一点：

**[0-9a-zA-Z]**

呵呵，是不是很简单啊。值得注意的是在字符组内部，只有 出现在开头 ^ 和出现在非开头的 - 才是元字符，其他的任何字符都代表它们自己，如：点号(.)代表任意一个字符，但当它出现在字符组内部时，它只代表它自己。

所以如果你想在字符组内部匹配连字符，那我们必须确保它出现在开头，像下面这样：

**[-0-9a-zA-Z]**

刚才我们提到，在字符组内部还有一个元字符 ^，当它出现在字符组开头的时候，我们把这样的字符组称为排除型字符组，顾名思义，就是匹配任一未列出的字符，如匹配非数字：

**[^0-9]**

是不是很简单啊。恭喜你，你又学会了两个元字符。

|  |  |  |
| --- | --- | --- |
| 元字符 | 名称 | 匹配对象 |
| . | 点号(dot) | 单个任意字符 |
| ^ | 脱字符(caret) | 行的起始位置 |
| $ | 美元符(dollar) | 行的结束位置 |
| | | 竖线(bar) | 匹配分割两边的任意一个表达式 |
| [...] | 字符组(Character Class) | 列出的任意字符 |
| [^...] | 排除型字符组(Negated Character Class) | 未列出的任意字符 |

## 单词分界符

假设你想在一个文件中搜索 **hi**，很不幸 **hisoft** (海辉软件)也被查出来了，可是我只想搜索单词**hi**，怎么办呢？试一试下面的表达式吧：

**\<hi\>**

**\<** 匹配单词的开始，**\>** 匹配单词的结束，其实还有一个元字符也表示单词分界符，下面的表达式和上面是一个意思：

**\bhi\b**

既然有匹配单词边界的需求，当然也会有匹配非单词边界的需求，如果你想查找包含 hi 的单词，试一试下面的表达式吧：

**hi\B**

怎么样，简单吧，注意以上元字符匹配的是一个边界，而不是具体的字符。恭喜你，你又学会了四个元字符。

|  |  |  |
| --- | --- | --- |
| **元字符** | **名称** | **匹配对象** |
| . | 点号(dot) | 单个任意字符 |
| [...] | 字符组(Character Class) | 列出的任意字符 |
| [^...] | 排除型字符组(Negated Character Class) | 未列出的任意字符 |
| ^ | 脱字符(caret) | 行的起始位置 |
| $ | 美元符(dollar) | 行的结束位置 |
| \< |  | 单词的起始位置 |
| \> |  | 单词结束位置 |
| \b | Boundary | 单词边界 |
| \B |  | 非单词边界 |
| | | 竖线(bar) | 匹配分割两边的任意一个表达式 |

## 环视

假设让你在一个文件中查找一下 **u** 后面不是 **x** 的单词，怎么办？ 一个很自然的想法是用下面的正则表达式来查找：

**u[^x]**

不过遗憾的是，你很可能漏掉了部分单词，啊？ 是吗？ 我怎么没看出问题来呢？ 到底会漏掉什么样的单词呢？ 呵呵，如果 you 这个单词出现在一行的末尾，试一试，你还能用这个表达式搜索出来吗？ 记住排除型字符组的意思是**匹配一个未列出的字符**，而不是**不要匹配列出的字符**。

那么正确的做法是什么呢？ 试一试下面的表达式吧。

**u(?!x)**

这个表达式是什么意思呢？它的意思是匹配字母 u， 且 u 的右边不是 x。那我想查找 u **后面是** x 的单词该怎么办呢？试一试下面的表达式吧：

**u(?=x)**

那我想查找 u **前面是** x 的单词该怎么办呢？试一试下面的表达式吧：

**(?<=x)u**

那我想查找 u **前面不是** x 的单词该怎么办呢？试一试下面的表达式吧：

**(?<!x)u**

呵呵，是不是有点糊涂啊。再看一遍。我们把这四个元字符称为**环视(lookaround)**，它们和我们之前学过的单词分界符，行的起始和结束符一样，**它们只匹配位置，而不匹配具体的字符**，了解这一点很重要。恭喜你，你又学会了四个非常强悍的正则表达式元字符。

|  |  |  |
| --- | --- | --- |
| 元字符 | 名称 | 匹配对象 |
| . | 点号(dot) | 单个任意字符 |
| [...] | 字符组(Character Class) | 列出的任意字符 |
| [^...] | 排除型字符组(Negated Character Class) | 未列出的任意字符 |
| ^ | 脱字符(caret) | 行的起始位置 |
| $ | 美元符(dollar) | 行的结束位置 |
| \< |  | 单词的起始位置 |
| \> |  | 单词的结束位置 |
| \b | Boundary | 单词边界 |
| \B |  | 非单词边界 |
| (?=...) | 肯定顺序环视(Positive Lookahead) | 成功如果右边能够匹配 |
| (?!...) | 否定顺序环视(Negative Lookahead) | 成功如果右边不能够匹配 |
| (?<=...) | 肯定逆序环视(Positive Lookbehind) | 成功如果左边能够匹配 |
| (?<!...) | 否定逆序环视(Negative Lookbehind) | 成功如果左边不能够匹配 |
| | | 竖线(bar) | 匹配分割两边的任意一个表达式 |

## 字符转译序列

我们已经学习了好多正则表达式元字符，它们在正则表达式中有特殊的意义，而丧失了它们本来的意义。有时候我们只想匹配它本身，该怎么办呢？ 如：我想匹配点号，可是在正则表达式中，点号代表任意单个字符，这时候我们只需要在点号的前面加反斜杠就 OK 了，如下：

**\.**

我们把这种形式称为**转义。我们可以在任何元字符的前面加上反斜杠来匹配它自己。**

大家知道回车和换行的区别吗？ 啊？ 他俩有区别吗？ 如果你没搞清楚它们之间的区别， 去[**这里**](http://blog.csdn.net/shangboerds/article/details/4722059)看看吧。

假设现在有个文件，其中有好多空行，你想把连续多个空行替换成一个空行， 该怎么办？ 在 EmEditor 中**Ctrl+H** 调出**Replace**对话框，在**Find**中输入**\r\n\r\n**，在 **Replace With** 中输入**\r\n**，然后狂点**Replace Al l**就搞定了。其中的**\r** 代表回车， **\n**代表换行， 怎么样，简单吧。

如果你有一个用逗号分隔的文件，想把它导入到 Excel 中看一下，该怎么办呢？ 很简单，我们可以使用Excel自带的数据导入向导。除此之外，我们可以把逗号替换成tab符，然后把文件中的所有内容copy到Excel中就可以了。好，我们来实际操作一下，在 EmEditor 中**Ctrl+H** 调出**Replace**对话框，在**Find**中输入**逗号(,)**，在 **Replace With** 中输入**tab符，**可是当你按tab键的时候发现光标被挪到其他地方了，根本就无法输入，该怎么办呢？一种办法是在文件中输入一个tab符，然后把这个tab符复制粘贴到**Replace With**中，第二种办法就是使用字符转义\t。

恭喜你，你又学会了3个元字符（从严格意义上讲，\r \n \t 并不是正则表达式元字符，而是字符转义，为了便于大家理解，你可以把它们当做元字符看待）。

|  |  |  |
| --- | --- | --- |
| 元字符 | 名称 | 匹配对象 |
| . | 点号(dot) | 单个任意字符 |
| \n | 换行(newline) | 换行 |
| \r | 回车(return) | 回车 |
| \t | 制表符(Tab) | 制表符 |
| [...] | 字符组(Character Class) | 列出的任意字符 |
| [^...] | 排除型字符组(Negated Character Class) | 未列出的任意字符 |
| ^ | 脱字符(caret) | 行的起始位置 |
| $ | 美元符(dollar) | 行的结束位置 |
| \< |  | 单词的起始位置 |
| \> |  | 单词的结束位置 |
| \b | 边界(Boundary) | 单词边界 |
| \B |  | 非单词边界 |
| (?=...) | 肯定顺序环视(Positive Lookahead) | 成功如果右边能够匹配 |
| (?!...) | 否定顺序环视(Negative Lookahead) | 成功如果右边不能够匹配 |
| (?<=...) | 肯定逆序环视(Positive Lookbehind) | 成功如果左边能够匹配 |
| (?<!...) | 否定逆序环视(Negative Lookbehind) | 成功如果左边不能够匹配 |
| | | 竖线(bar) | 匹配分割两边的任意一个表达式 |

## 字符组转译序列

还记得我们在[**字符组**](http://blog.csdn.net/shangboerds/article/details/7482765)这一节中是如何匹配数字的吗？ 对了， 是**[0-9]**， 事实上如果你这样去匹配一个数字，别人可能要笑话你 out 了， 更常用的是**\d**，而且 \d 比前者更强大， 因为如果工具支持 Unicode，它还可以匹配中文全角数字，在 EmEditor 中试一试吧。

有匹配数字的需求，当然肯定会有匹配**非**数字的需求，试一试 **\D** 吧。当然你也可以用 **[^0-9]**或**[^\d]**。

如果你想匹配一个组成单词的字符，该怎么办呢？ 试一试 **[a-zA-Z0-9\_]** 吧，当然更常用的是 **\w**，而且 \w 比前者更强大，因为如果工具支持 Unicode， 它还可以匹配汉字，在 EmEditor 中亲自操练一下吧。

当然，如果让你匹配一个**非**组成单词的字符，该怎么办呢？不知道？ 好好想想。看看上面的非数字，你猜对了，是 **\W**，当然也可以这样**[^\w]**。

大家知道什么叫空白字符吗? 像空格、tab等这类字符我们称之为空白字符，因为我们无法看到它们。我们可以用 **[ \t\v\r\n\f]**来匹配空白字符，当然更常用的是**\s**，那如何匹配非空白字符呢？ 试一试 **\S**吧，当然你也可以用**[^\s]**。

恭喜你，你又学会了6个元字符。

|  |  |  |
| --- | --- | --- |
| 元字符 | 名称 | 匹配对象 |
| . | 点号(dot) | 单个任意字符 |
| \n | 换行(newline) | 换行 |
| \r | 回车(return) | 回车 |
| \t | 制表符(Tab) | 制表符 |
| [...] | 字符组(Character Class) | 列出的任意字符 |
| [^...] | 排除型字符组(Negated Character Class) | 未列出的任意字符 |
| \d | 数字(digit) | 数字 |
| \D |  | 非数字 |
| \w | 单词(word) | 单词字符 |
| \W |  | 非单词字符 |
| \s | 空白字符(whitespace) | 空白字符 |
| \S |  | 非空白字符 |
| ^ | 脱字符(caret) | 行的起始位置 |
| $ | 美元符(dollar) | 行的结束位置 |
| \< |  | 单词的起始位置 |
| \> |  | 单词的结束位置 |
| \b | 边界(Boundary) | 单词边界 |
| \B |  | 非单词边界 |
| (?=...) | 肯定顺序环视(Positive Lookahead) | 成功如果右边能够匹配 |
| (?!...) | 否定顺序环视(Negative Lookahead) | 成功如果右边不能够匹配 |
| (?<=...) | 肯定逆序环视(Positive Lookbehind) | 成功如果左边能够匹配 |
| (?<!...) | 否定逆序环视(Negative Lookbehind) | 成功如果左边不能够匹配 |
| | | 竖线(bar) | 匹配分割两边的任意一个表达式 |

## 量词

我们已经学会了使用 \d 来匹配数字，如果我要匹配三个数字， 我可以写成 \d\d\d， 那如果我要匹配三十个数字，是不是要写三十个 \d 呢？ 呵呵， 当然不用。试一试下面的表达式吧。

**\d{30}**

有时候，我们并不能确定有多少个数字，甚至有时候我们连有没有数字也无法确定，该怎么匹配呢？看看下面这个表格吧，我们把这些元字符称为**量词**(quantifiers).

|  |  |  |  |
| --- | --- | --- | --- |
| 元字符 | 次数下限 | 次数上限 | 含义 |
| X? | 无 | 1 | X 可以不出现，也可以只出现一次 |
| X+ | 1 | 无 | X 可以出现无数次，但至少要出现一次 |
| X\* | 无 | 无 | X 可以出现无数次，也可以不出现 |
| X{n} | n | n | X 精确出现n次 |
| X{n,} | n | 无 | X 可以出现无数次，但至少要出现n次 |
| X{n,m} | n | m | X 至少要出现n次，至多出现m次 |

我们知道 July 和 Jul 都可以表示七月，我们想在文件中查找这个单词，该怎么办呢？easy， 很显然我们之前做过一个类似的例子，用下面的正则表达式。

July|Jul

事实上，我们还可以这样

July?

看懂了吗？ 注意问号修饰的是字母y 。表示 y 可以不出现，也可以只出现一次。更简单点说，y 是可有可无的。怎么样？简单吧。之前我们学习了如何匹配空行，还记得吗？ 对了，用下面的表达式：

^$

事实上，这个表达式并不精确，因为空行中可以包括空白字符，当然，也可以没有，所以，正确的应该是:

^\s\*$

恭喜你，你又学会了六个元字符。

|  |  |  |
| --- | --- | --- |
| 元字符 | 名称 | 匹配对象 |
| . | 点号(dot) | 单个任意字符 |
| \n | 换行(newline) | 换行 |
| \r | 回车(return) | 回车 |
| \t | 制表符(Tab) | 制表符 |
| [...] | 字符组(Character Class) | 列出的任意字符 |
| [^...] | 排除型字符组(Negated Character Class) | 未列出的任意字符 |
| \d | 数字(digit) | 数字 |
| \D |  | 非数字 |
| \w | 单词(word) | 单词字符 |
| \W |  | 非单词字符 |
| \s | 空白字符(whitespace) | 空白字符 |
| \S |  | 非空白字符 |
| ^ | 脱字符(caret) | 行的起始位置 |
| $ | 美元符(dollar) | 行的结束位置 |
| \< |  | 单词的起始位置 |
| \> |  | 单词的结束位置 |
| \b | 边界(Boundary) | 单词边界 |
| \B |  | 非单词边界 |
| (?=...) | 肯定顺序环视(Positive Lookahead) | 成功如果右边能够匹配 |
| (?!...) | 否定顺序环视(Negative Lookahead) | 成功如果右边不能够匹配 |
| (?<=...) | 肯定逆序环视(Positive Lookbehind) | 成功如果左边能够匹配 |
| (?<!...) | 否定逆序环视(Negative Lookbehind) | 成功如果左边不能够匹配 |
| ? |  | 它之前的字符可以不出现，也可以只出现一次 |
| + |  | 它之前的字符可以出现无数次，但至少要出现一次 |
| \* |  | 它之前的字符可以出现无数次，也可以不出现 |
| {n} |  | 它之前的字符精确匹配n次 |
| {n,} |  | 它之前的字符可以出现无数次，但至少要出现n次 |
| {n,m} |  | 它之前的字符至少要出现n次，至多出现m次 |
| | | 竖线(bar) | 匹配分割两边的任意一个表达式 |

## 括号及后向引用

假设你有一个文件，其中包括两列，第一列是姓名，第二列是身份证号，中间用逗号分隔。现在让你把这两列调换一下顺序，也就是第一列是身份证号，第二列是姓名，该怎么办呢？

我们可以先把逗号替换成 Tab 符， 然后把整个文件中的内容 copy 到 excel 中， 在 excel 中调换顺序后把内容 copy 回文件。 当然这种方法可行，只是有点笨了，试一试下面这种方法吧。

在 EmEditor 中**Ctrl+H** 调出**Replace**对话框，在**Find**中输入**(.\*),(.\*)**，在 **Replace With** 中输入**\2,\1**，然后优雅的点击**Replace Al l**就搞定了。

就这么简单？ 是的，就这么简单。在正则表达式中**括号**有个神奇的功能，它能够记住它包含的子表达式匹配的文本，**\1**表示第一个括号中的内容，**\2**表示第二个括号中的内容。我们把这种神奇的功能称为**后向引用（backreference）**。不过遗憾的 EmEditor 最多能引用**九个**括号，也就是说支持从 \1 - \9。

事实上，在很多时候我们使用括号并不是想要它的后向引用功能，而是将若干字符组成一个单元。还记得在[**量词**](http://blog.csdn.net/shangboerds/article/details/7515592)一节中，我们是如何匹配七月的吗？对了， 用下面的表达式：

**July?**

问号的作用对象是 y，表示 y 可有可无。如果现在让你匹配一月 (january 或 jan) 该怎么办呢？答案是使用括号，如下：

**jan(uary)?**

尽管在这里我们并不希望括号能够记住它包含的内容，但是它还是记住了。这样会使我们的正则表达式效率低下，为此正则表达式提供了另一种形式的括号，我们可以将上面的表达式改写成下面这样：

**jan(?:uary)?**

这样正则表达式的运行效率会更高。

恭喜你，你有学会了三个元字符。

|  |  |  |
| --- | --- | --- |
| 元字符 | 名称 | 匹配对象 |
| . | 点号(dot) | 单个任意字符 |
| \n | 换行(newline) | 换行 |
| \r | 回车(return) | 回车 |
| \t | 制表符(Tab) | 制表符 |
| [...] | 字符组(Character Class) | 列出的任意字符 |
| [^...] | 排除型字符组(Negated Character Class) | 未列出的任意字符 |
| \d | 数字(digit) | 数字 |
| \D |  | 非数字 |
| \w | 单词(word) | 单词字符 |
| \W |  | 非单词字符 |
| \s | 空白字符(whitespace) | 空白字符 |
| \S |  | 非空白字符 |
| ^ | 脱字符(caret) | 行的起始位置 |
| $ | 美元符(dollar) | 行的结束位置 |
| \< |  | 单词的起始位置 |
| \> |  | 单词的结束位置 |
| \b | 边界(Boundary) | 单词边界 |
| \B |  | 非单词边界 |
| (?=...) | 肯定顺序环视(Positive Lookahead) | 成功如果右边能够匹配 |
| (?!...) | 否定顺序环视(Negative Lookahead) | 成功如果右边不能够匹配 |
| (?<=...) | 肯定逆序环视(Positive Lookbehind) | 成功如果左边能够匹配 |
| (?<!...) | 否定逆序环视(Negative Lookbehind) | 成功如果左边不能够匹配 |
| ? |  | 它之前的字符可以不出现，也可以只出现一次 |
| + |  | 它之前的字符可以出现无数次，但至少要出现一次 |
| \* |  | 它之前的字符可以出现无数次，也可以不出现 |
| {n} |  | 它之前的字符精确匹配n次 |
| {n,} |  | 它之前的字符可以出现无数次，但至少要出现n次 |
| {n,m} |  | 它之前的字符至少要出现n次，至多出现m次 |
| (...) | 圆括号(parenthese) | 分组子表达式和记录它包含的字表达匹配的内容 |
| (?:...) |  | 分组子表达式 |
| \1 - \9 |  | 后向引用 |
| | | 竖线(bar) | 匹配分割两边的任意一个表达式 |

## 非贪婪匹配

我们接着上节的例子，你有一个文件，其中包括两列，第一列是姓名，第二列是身份证号，中间用逗号分隔。不同的是，姓名和身份证号都用双引号引起来。像下面这样：

"尚波","1234567890"

假设你想查找姓名这一列，用下面的表达式：

**^".\*"**

然而结果并非像我们期望的那样匹配 **"尚波"**，而是匹配了整行 **"尚波","1234567890"**，这不是我们想要的。我们把星号(\*)这种匹配方式称为贪婪（greedy）匹配。我们到底该怎么办呢？ 试一试下面的表达式吧。

**^".\*?"**

呵呵，怎么样？简单吧，仅仅在星号之后加上问号我们就改变成了非贪婪匹配。其他量词同星号一样。

贪婪匹配：\*, +, ?, {n}, {n,}, {n,m}  
非贪婪匹配：\*?, +?, ??, {n}?, {n,}?, {n,m}?

事实上，我们还有其他办法解决这个问题，试一试下面的表达式吧。

**^"[^"]\*"**

你看懂了吗？

## 忽略大小写匹配

我们在之前的例子中并没有考虑大小写的问题，事实上正则表达式本身是区分大小写的，为了匹配不区分大小写的 a，我们需要用正则表达式 [aA]，这样就大大增加了正则表达式的复杂度，幸运的是每一种支持正则表达式的工具都提供了忽略大小写的匹配模式，如 EmEditor 默认的模式就是忽略大小写的匹配模式，你可以选中 Find 对话框中的 Match Case 来进行大小写匹配。

## 方言

我们的祖国那么大，每个地方都有自己的方言，虽然它们都称为汉语，但是和普通话还是有一定的差别的。事实上，正则表达式也有方言，在[**正则表达式简介**](http://blog.csdn.net/shangboerds/article/details/7462569)一节中我们了解了一些支持正则表达式的工具，虽然它们都支持正则表达式，但是每样工具支持的元字符和其他特性是有细微差别的。在使用一样工具之前，你最好查看一下它的文档，了解一下它支持哪些元字符。如果你使用的是 EmEditor，打开 Help --> Search Help Topics，搜索 Regular Expression 来了解更多的细节。

## 总结

恭喜你，你已经学完了基本篇的全部知识，下面是我们本篇学到的全部正则表达式元字符。你肯定对正则表达式有了一定的理解，遇到问题时，你也会想到用正则表达式能够解决这个问题，但是真正让你写正则表达式的时候，我想对你来说还是有困难的，如果你有这种感觉，那么正则表达式练习篇的系列文章正是为你准备的。你还在等什么呢？

|  |  |  |
| --- | --- | --- |
| 元字符 | 名称 | 匹配对象 |
| . | 点号(dot) | 单个任意字符 |
| \n | 换行(newline) | 换行 |
| \r | 回车(return) | 回车 |
| \t | 制表符(Tab) | 制表符 |
| [...] | 字符组(Character Class) | 列出的任意字符 |
| [^...] | 排除型字符组(Negated Character Class) | 未列出的任意字符 |
| \d | 数字(digit) | 数字 |
| \D |  | 非数字 |
| \w | 单词(word) | 单词字符 |
| \W |  | 非单词字符 |
| \s | 空白字符(whitespace) | 空白字符 |
| \S |  | 非空白字符 |
|  |  |  |
| ^ | 脱字符(caret) | 行的起始位置 |
| $ | 美元符(dollar) | 行的结束位置 |
| \< |  | 单词的起始位置 |
| \> |  | 单词的结束位置 |
| \b | 边界(Boundary) | 单词边界 |
| \B |  | 非单词边界 |
| (?=...) | 肯定顺序环视(Positive Lookahead) | 成功如果右边能够匹配 |
| (?!...) | 否定顺序环视(Negative Lookahead) | 成功如果右边不能够匹配 |
| (?<=...) | 肯定逆序环视(Positive Lookbehind) | 成功如果左边能够匹配 |
| (?<!...) | 否定逆序环视(Negative Lookbehind) | 成功如果左边不能够匹配 |
|  |  |  |
| X? |  | 匹配X 0次或1次 |
| X\* |  | 匹配X 0次或无数次 |
| X+ |  | 匹配X 1次或无数次 |
| X{n} |  | 匹配X n次 |
| X{n,} |  | 匹配X 至少n次 |
| X{n,m} |  | 匹配X 至少n次至多m次 |
| X?? |  | 匹配X 0次或1次 |
| X\*? |  | 匹配X 0次或无数次 |
| X+? |  | 匹配X 1次或无数次 |
| X{n}? |  | 匹配X n次 |
| X{n,}? |  | 匹配X 至少n次 |
| X{n,m}? |  | 匹配X 至少n次至多m次 |
|  |  |  |
| (...) | 圆括号(parenthese) | 分组或捕获 |
| (?:...) |  | 分组 |
| \1 - \9 |  | 后向引用 |
| | | 竖线(bar) | 匹配分割两边的任意一个表达式 |