正则表达式之道

原著：Steve Mansour
sman@scruznet.com
Revised: June 5, 1999
(copied by jm /at/ jmason.org from http://www.scruz.net/%7esman/regexp.htm, after the original disappeared!)

翻译：Neo Lee
neo.lee@gmail.com
2004年10月16日

英文版原文

译者按：原文因为年代久远，文中很多链接早已过期（主要是关于vi、sed等工具的介绍和手册），本译文中已将此类链接删除，如需检查这些链接可以查看上面链接的原文。除此之外基本照原文直译，括号中有“译者按”的部分是译者补充的说明。如有内容方面的问题请直接和Steve Mansor联系，当然，如果你只写中文，也可以和我联系。

目 录

什么是正则表达式

一个正则表达式，就是用某种模式去匹配一类字符串的一个公式。很多人因为它们看上去比较古怪而且复杂所以不敢去使用——很不幸，这篇文章也不能够改变这一点，不过，经过一点点练习之后我就开始觉得这些复杂的表达式其实写起来还是相当简单的，而且，一旦你弄懂它们，你就能把数小时辛苦而且易错的文本处理工作压缩在几分钟（甚至几秒钟）内完成。正则表达式被各种文本编辑软件、类库（例如Rogue Wave的tools.h++）、脚本工具（像awk/grep/sed）广泛的支持，而且像Microsoft的Visual C++这种交互式IDE也开始支持它了。

我们将在如下的章节中利用一些例子来解释正则表达式的用法，绝大部分的例子是基于**vi**中的文本替换命令和**grep**文件搜索命令来书写的，不过它们都是比较典型的例子，其中的概念可以在sed、awk、perl和其他支持正则表达式的编程语言中使用。你可以看看不同工具中的正则表达式这一节，其中有一些在别的工具中使用正则表达式的例子。还有一个关于vi中文本替换命令（s）的简单说明附在文后供参考。

正则表达式基础

正则表达式由一些普通字符和一些*元字符*（*metacharacters*）组成。普通字符包括大小写的字母和数字，而元字符则具有特殊的含义，我们下面会给予解释。

在最简单的情况下，一个正则表达式看上去就是一个普通的查找串。例如，正则表达式"testing"中没有包含任何元字符，，它可以匹配"testing"和"123testing"等字符串，但是不能匹配"Testing"。

要想真正的用好正则表达式，正确的理解元字符是最重要的事情。下表列出了所有的元字符和对它们的一个简短的描述。

元字符	*描述*
.	匹配任何单个字符。例如正则表达式**r.t**匹配这些字符串：*rat*、*rut*、*r t*，但是不匹配*root*。
$	匹配行结束符。例如正则表达式**weasel$** 能够匹配字符串"*He's a weasel*"的末尾，但是不能匹配字符串"*They are a bunch of weasels.*"。
^	匹配一行的开始。例如正则表达式**^When in**能够匹配字符串"*When in the course of human events*"的开始，但是不能匹配"*What and When in the*"。
*****	匹配0或多个正好在它之前的那个字符。例如正则表达式**.***意味着能够匹配任意数量的任何字符。
****	这是引用府，用来将这里列出的这些元字符当作普通的字符来进行匹配。例如正则表达式**\$**被用来匹配美元符号，而不是行尾，类似的，正则表达式**\.**用来匹配点字符，而不是任何字符的通配符。
[ ] **[c1-c2]** **[^c1-c2]**	匹配括号中的任何一个字符。例如正则表达式**r[aou]t**匹配*rat*、*rot*和*rut*，但是不匹配*ret*。可以在括号中使用连字符-来指定字符的区间，例如正则表达式**[0-9]**可以匹配任何数字字符；还可以制定多个区间，例如正则表达式**[A-Za-z]**可以匹配任何大小写字母。另一个重要的用法是“排除”，要想匹配除了指定区间之外的字符——也就是所谓的补集——在左边的括号和第一个字符之间使用^字符，例如正则表达式**[^269A-Z]** 将匹配除了2、6、9和所有大写字母之外的任何字符。
\< \>	匹配词（*word*）的开始（\<）和结束（\>）。例如正则表达式**\<the**

	能够匹配字符串"*for the wise*"中的"the"，但是不能匹配字符串"*otherwise*"中的"the"。**注意**：这个元字符不是所有的软件都支持的。
\(\)	将 \(和 \) 之间的表达式定义为“组”（*group*），并且将匹配这个表达式的字符保存到一个临时区域（一个正则表达式中最多可以保存9个），它们可以用 **\1** 到**\9** 的符号来引用。
|	将两个匹配条件进行逻辑“或”（*Or*）运算。例如正则表达式 **(him|her)** 匹配"*it belongs to him*"和"*it belongs to her*"，但是不能匹配"*it belongs to them.*"。**注意**：这个元字符不是所有的软件都支持的。
+	匹配1或多个正好在它之前的那个字符。例如正则表达式**9+**匹配9、99、999等。**注意**：这个元字符不是所有的软件都支持的。
?	匹配0或1个正好在它之前的那个字符。**注意**：这个元字符不是所有的软件都支持的。
\{*i*\} **\{*i*,*j*\}**	匹配指定数目的字符，这些字符是在它之前的表达式定义的。例如正则表达式**A[0-9]\{3\}** 能够匹配字符"A"后面跟着正好3个数字字符的串，例如A123、A348等，但是不匹配A1234。而正则表达式**[0-9]\{4,6\}** 匹配连续的任意4个、5个或者6个数字字符。**注意**：这个元字符不是所有的软件都支持的。

最简单的元字符是点，它能够匹配任何单个字符（注意**不**包括新行符）。假定有个文件test.txt包含以下几行内容：

```
he is a rat
he is in a rut
the food is Rotten
I like root beer
```

我们可以使用grep命令来测试我们的正则表达式，grep命令使用正则表达式去尝试匹配指定文件的每一行，并将至少有一处匹配表达式的所有行显示出来。命令

```
grep r.t test.txt
```

在test.txt文件中的每一行中搜索正则表达式**r.t**，并打印输出匹配的行。正则表达式**r.t**匹配一个**r**接着任何一个字符再接着一个**t**。所以它将匹配文件中的**rat**和**rut**，而不能匹配**Rotten**中的**Rot**，因为正则表达式是大小写敏感的。要想同时匹配大写和小写字母，应该使用字符区间元字符（方括号）。正则表达式**[Rr]**能够同时匹配**R**和**r**。所以，要想匹配一个大写或者小写的**r**接着任何一个字符再接着一个**t**就要使用这个表达式：**[Rr].t**。

要想匹配行首的字符要使用抑扬字符（^）——又是也被叫做插入符。例如，想找到text.txt中行首"he"打头的行，你可能会先用简单表达式**he**，但是这会匹配第

三行的**the**，所以要使用正则表达式**^he**，它只匹配在行首出现的**h**。

有时候指定“除了×××都匹配”会比较容易达到目的，当抑扬字符（^）出现在方括号中是，它表示“排除”，例如要匹配**he**，但是排除前面是**t** or **s**的情性（也就是**the**和**she**），可以使用：**[^st]he**。

可以使用方括号来指定多个字符区间。例如正则表达式**[A-Za-z]**匹配任何字母，包括大写和小写的；正则表达式**[A-Za-z][A-Za-z]*** 匹配一个字母后面接着0或者多个字母（大写或者小写）。当然我们也可以用元字符**+**做到同样的事情，也就是：**[A-Za-z]+**，和**[A-Za-z][A-Za-z]***完全等价。但是要注意元字符**+** 并不是所有支持正则表达式的程序都支持的。关于这一点可以参考后面的正则表达式语法支持情况。

要指定特定数量的匹配，要使用大括号（注意必须使用反斜杠来转义）。想匹配所有**100**和**1000**的实例而排除**10**和**10000**，可以使用：**10\{2,3\}**，这个正则表达式匹配数字1后面跟着2或者3个0的模式。在这个元字符的使用中一个有用的变化是忽略第二个数字，例如正则表达式**0\{3,\}** 将匹配至少3个连续的0。

简单的例子

这里有一些有代表性的、比较简单的例子。

vi 命令	*作用*
:%s/ */ /g	把一个或者多个空格替换为一个空格。
:%s/ *$//	去掉行尾的所有空格。
:%s/^/ /	在每一行头上加入一个空格。
:%s/^[0-9][0-9]* //	去掉行首的所有数字字符。
:%s/b[aeio]g/bug/g	将所有的*bag*、*beg*、*big*和*bog*改为*bug*。
:%s/t\([aou]\)g/h\1t/g	将所有*tag*、*tog*和*tug*分别改为*hat*、*hot*和*hug*（注意用group的用法和使用\1引用前面被匹配的字符）。

中级的例子（神奇的咒语）

例1

将所有方法foo(*a*,*b*,*c*)的实例改为foo(*b*,*a*,*c*)。这里a、b和c可以是任何提供给方法foo()的参数。也就是说我们要实现这样的转换：

之前 之后

```
foo(10,7,2)              foo(7,10,2)
foo(x+13,y-2,10)         foo(y-2,x+13,10)
foo( bar(8), x+y+z, 5)   foo( x+y+z, bar(8), 5)
```

下面这条替换命令能够实现这一魔法：

```
:%s/foo(\([^,]*\),\([^,]*\),\([^)]*\))/foo(\2,\1,\3)/g
```

现在让我们把它打散来加以分析。写出这个表达式的基本思路是找出foo()和它的括号中的三个参数的位置。第一个参数是用这个表达式来识别的：**\([^,]*\)**，我们可以从里向外来分析它：

[^,]	除了逗号之外的任何字符
[^,]*	0或者多个非逗号字符
\([^,]*\)	将这些非逗号字符标记为**\1**，这样可以在之后的替换模式表达式中引用它
\([^,]*\),	我们必须找到0或者多个非逗号字符后面跟着一个逗号，并且非逗号字符那部分要标记出来以备后用。

现在正是指出一个使用正则表达式常见错误的最佳时机。为什么我们要使用**[^,]***这样的一个表达式，而不是更加简单直接的写法，例如：**.***，来匹配第一个参数呢？设想我们使用模式**.***来匹配字符串"10,7,2"，它应该匹配"10,"还是"10,7,"？为了解决这个两义性（ambiguity），正则表达式规定一律按照最长的串来，在上面的例子中就是"10,7,"，显然这样就找出了两个参数而不是我们期望的一个。所以，我们要使用**[^,]***来强制取出第一个逗号之前的部分。

这个表达式我们已经分析到了：**foo(\([^,]*\)**，这一段可以简单的翻译为“当你找到**foo(**就把其后直到第一个逗号之前的部分标记为**\1**”。然后我们使用同样的办法标记第二个参数为**\2**。对第三个参数的标记方法也是一样，只是我们要搜索所有的字符直到右括号。我们并没有必要去搜索第三个参数，因为我们不需要调整它的位置，但是这样的模式能够保证我们只去替换那些有三个参数的foo()方法调用，在foo()是一个重载（overoading）方法时这种明确的模式往往是比较保险的。然后，在替换部分，我们找到foo()的对应实例，然后利用标记好的部分进行替换，是的第一和第二个参数交换位置。

例2

假设有一个CSV（comma separated value）文件，里面有一些我们需要的信息，但是格式却有问题，目前数据的列顺序是：姓名，公司名，州名缩写，邮政编码，现在我们希望讲这些数据重新组织，以便在我们的某个软件中使用，需要的格式为：姓名，州名缩写-邮政编码，公司名。也就是说，我们要调整列顺序，还要合并两个列来构成一个新列。另外，我们的软件不能接受逗号前后面有任何空格（包括空格和制表符）所以我们还必须要去掉逗号前后的所有空格。

这里有几行我们现在的数据：

```
Bill Jones,     HI-TEK Corporation ,  CA, 95011
Sharon Lee Smith,  Design Works Incorporated,  CA, 95012
B. Amos   ,  Hill Street Cafe,  CA, 95013
Alexander Weatherworth,  The Crafts Store,  CA, 95014
...
```

我们希望把它变成这个样子：

```
Bill Jones,CA 95011,HI-TEK Corporation
Sharon Lee Smith,CA 95012,Design Works Incorporated
B. Amos,CA 95013,Hill Street Cafe
Alexander Weatherworth,CA 95014,The Crafts Store
...
```

我们将用两个正则表达式来解决这个问题。第一个移动列和合并列，第二个用来去掉空格。

下面就是第一个替换命令：

```
:%s/\([^,]*\),\([^,]*\),\([^,]*\),\(.*\)/\1,\3 \4,\2/
```

这里的方法跟例1基本一样，第一个列（姓名）用这个表达式来匹配：**\([^,]*\)**，即第一个逗号之前的所有字符，而姓名内容被用**\1**标记下来。公司名和州名缩写字段用同样的方法标记为**\2**和**\3**，而最后一个字段用**\(.*\)**来匹配（"匹配所有字符直到行末"）。替换部分则引用上面标记的那些内容来进行构造。

下面这个替换命令则用来去除空格：

```
:%s/[ \t]*,[ \t]*/,/g
```

我们还是分解来看：**[\t]**匹配空格/制表符，**[\t]*** 匹配0或多个空格/制表符，**[\t]***,匹配0或多个空格/制表符后面再加一个逗号，最后，**[\t]*,[\t]***匹配0或多个空格/制表符接着一个逗号再接着0或多个空格/制表符。在替换部分，我们简单的我们找到的所有东西替换成一个逗号。这里我们使用了结尾的可选的**g**参数，这表示在每行中对所有匹配的串执行替换（而不是缺省的只替换第一个匹配串）。

例3

假设有一个多字符的片断重复出现，例如：

```
Billy tried really hard
Sally tried really really hard
Timmy tried really really really hard
Johnny tried really really really really hard
```

而你想把"really"、"really really"，以及任意数量连续出现的"really"字符串换成一个简单的"very"（simple is good!），那么以下命令：

```
:%s/\(really \)\(really \)*/very /
```

就会把上述的文本变成：

```
Billy tried very hard
Sally tried very hard
Timmy tried very hard
Johnny tried very hard
```

表达式**\(really \)***匹配0或多个连续的"really "（注意结尾有个空格），而**\(really \)\(really \)*** 匹配1个或多个连续的"really "实例。

困难的例子（不可思议的象形文字）

Coming soon.

不同工具中的正则表达式

OK，你已经准备使用RE（regular expressions，正则表达式），但是你并准备使用vi。所以，在这里我们给出一些在其他工具中使用RE的例子。另外，我还会总结一下你在不同程序之间使用RE可能发现的区别。

当然，你也可以在Visual C++编辑器中使用RE。选择Edit->Replace，然后选择"Regular expression"选择框，Find What输入框对应上面介绍的vi命令**:%s/pat1/pat2/g**中的pat1部分，而Replace输入框对应pat2部分。但是，为了得到vi的执行范围和**g**选项，你要使用Replace All或者适当的手工Find Next and Replace（译者按：知道为啥有人骂微软弱智了吧，虽然VC中可以选中一个范围的文本，然后在其中执行替换，但是总之不够vi那么灵活和典雅）。

sed

Sed是**S**tream **ED**itor的缩写，是Unix下常用的基于文件和管道的编辑工具，可以在手册中得到关于sed的详细信息。

这里是一些有趣的sed脚本，假定我们正在处理一个叫做price.txt的文件。注意这些编辑并不会改变源文件，sed只是处理源文件的每一行并把结果显示在标准输出中（当然很容易使用重定向来定制）：

sed脚本	*描述*

`sed 's/^$/d' price.txt`	删除所有空行
`sed 's/^[ \t]*$/d' price.txt`	删除所有只包含空格或者制表符的行
`sed 's/"//g' price.txt`	删除所有引号

awk

awk是一种编程语言，可以用来对文本数据进行复杂的分析和处理。可以在手册中得到关于awk的详细信息。这个古怪的名字是它作者们的姓的缩写（Aho，Weinberger和Kernighan）。

在Aho，Weinberger和Kernighan的书The AWK Programming Language中有很多很好的awk的例子，请不要让下面这些微不足道的脚本例子限制你对awk强大能力的理解。我们同样假定我们针对price.txt文件进行处理，跟sed一样，awk也只是把结果显示在终端上。

awk脚本	*描述*
`awk '$0 !~ /^$/' price.txt`	删除所有空行
`awk 'NF > 0' price.txt`	awk中一个更好的删除所有行的办法
`awk '$2 ~ /^[JT]/ {print $3}' price.txt`	打印所有第二个字段是'J'或者'T'打头的行中的第三个字段
`awk '$2 !~ /[Mm]isc/ {print $3 + $4}' price.txt`	针对所有第二个字段不包含'Misc'或者'misc'的行，打印第3和第4列的和（假定为数字）
`awk '$3 !~ /^[0-9]+\.[0-9]*$/ {print $0}' price.txt`	打印所有第三个字段不是数字的行，这里数字是指`d.d`或者`d`这样的形式，其中`d`是0到9的任何数字
`awk '$2 ~ /John|Fred/ {print $0}' price.txt`	如果第二个字段包含'John'或者'Fred'则打印整行

grep

grep是一个用来在一个或者多个文件或者输入流中使用RE进行查找的程序。它的name编程语言可以用来针对文件和管道进行处理。可以在手册中得到关于grep的完整信息。这个同样古怪的名字来源于vi的一个命令，**g/*re*/p**，意思是**g**lobal **r**egular **e**xpression **p**rint。

下面的例子中我们假定在文件phone.txt中包含以下的文本，——其格式是姓加一个逗号，然后是名，然后是一个制表符，然后是电话号码：

```
Francis, John           5-3871
Wong, Fred              4-4123
Jones, Thomas           1-4122
Salazar, Richard        5-2522
```

grep命令	*描述*
grep '\t5-...1' phone.txt	把所有电话号码以5开头以1结束的行打印出来，注意制表符是用**\t**表示的
grep '^S[^]* R' phone.txt	打印所有姓以S打头和名以R打头的行
grep '^[JW]' phone.txt	打印所有姓开头是J或者W的行
grep ', ...\t' phone.txt	打印所有姓是4个字符的行，注意制表符是用**\t**表示的
grep -v '^[JW]' phone.txt	打印所有不以J或者W开头的行
grep '^[M-Z]' phone.txt	打印所有姓的开头是M到Z之间任一字符的行
grep '^[M-Z].*[12]' phone.txt	打印所有姓的开头是M到Z之间任一字符，并且点号号码结尾是1或者2的行

egrep

egrep是grep的一个扩展版本，它在它的正则表达式中支持更多的元字符。下面的例子中我们假定在文件phone.txt中包含以下的文本，——其格式是姓加一个逗号，然后是名，然后是一个制表符，然后是电话号码：

```
Francis, John           5-3871
Wong, Fred              4-4123
Jones, Thomas           1-4122
Salazar, Richard        5-2522
```

egrep command	*Description*
egrep '(John|Fred)' phone.txt	打印所有包含名字*John*或者*Fred*的行
egrep 'John|22$|^W' phone.txt	打印所有包含*John* 或者以22结束或者以*W*的行
egrep 'net(work)?s' report.txt	从report.txt中找到所有包含*networks*或者*nets*的行

正则表达式语法支持情况

命令或环境	.	[ ]	^	$	\(\)	\{ \}	?	+	|	( )
vi	X	X	X	X	X					
Visual C++	X	X	X	X	X					
awk	X	X	X	X			X	X	X	X
sed	X	X	X	X	X	X				
Tcl	X	X	X	X	X		X	X	X	X
ex	X	X	X	X	X	X				
grep	X	X	X	X	X	X				
egrep	X	X	X	X	X		X	X	X	X
fgrep	X	X	X	X	X					
perl	X	X	X	X	X		X	X	X	X

---

vi替换命令简介

Vi的替换命令：

:*range***s/***pat1***/***pat2***/g**

其中

: 这是Vi的命令执行界面。

range 是命令执行范围的指定，可以使用百分号（%）表示所有行，使用点（.）表示当前行，使用美元符号（$）表示最后一行。你还可以使用行号，例如**10,20**表示第10到20行，**.,$**表示当前行到最后一行，**.+2,$-5**表示当前行后两行直到全文的倒数第五行，等等。

s 表示其后是一个替换命令。

pat1 这是要查找的一个正则表达式，这篇文章中有一大堆例子。

pat2 这是希望把匹配串变成的模式的正则表达式，这篇文章中有一大堆例子。

g 可选标志，带这个标志表示替换将针对行中每个匹配的串进行，否则则只替换行中第一个匹配串。

网上有很多vi的在线手册，你可以访问他们以获得更加完整的信息。

[回到主页]