正则表达式学习笔记

阿左 [1] Nobody [2]

January 12, 2013

[1]感谢档
[2]感谢郭嘉

Contents

List of Figures

List of Tables

Abstract

Regex study note

摘要

Regex study note

Part I

基本概念

Chapter 1

元字符（Metacharacters）

1.1 基本元字符

1.1.1 任意字符

点号“.”匹配任意一个字符。

1.1.2 行开始与结束

脱字符与美元符分别代表行的开始与结束位置。注意这两个元字符只表示两个特殊的位置，位置上是没有字符的。

```
^This is a line.$
```

匹配空白行：

```
^$
```

匹配所有的行（因为所有的行都有一个开头）：

```
^
```

1.1.3 增强锚点

通常来说，锚点“^”、“$”匹配的不是逻辑行的开头与结尾，而是整个字符串的开头与结尾。如果要匹配逻辑行可以切换到增强锚点（enhanced line anchor）模式下。在 Perl 语言中修饰符为“/m”。例如，把空白行替换为 HTML 的段落符“<p>”：

```
$text =~ s/^$/<p>/mg;
```

1.1.4 单词分界符（Word Boundaries）

“\<”与“\>”匹配单词（包括字母和数字）的开始与结束。注意匹配的是位置，而不是字符。

1.2 字符范围（Character Classes）

“[...]”可以定义一个位置上可以出现的字符的范围。“<H1>”、“<H2>”、“<H3>”可以用：“<H[123]>”来表示。

“[^...]”表示排除指定字符。没有列出来的任何字符都可以。

表达式“q[^u]”匹配不了单词“Iraq”，因为表达式的意义不是“q”后面没有u，而是“q”后面要“有”一个字符，这个字符不能是“u”，其他的都行。

可以用连字符来表示连续的字符：“[0-9a-zA-Z_!.?]”；只有在也只有连字符是特殊字符。后面的下划线、问号、点号等都是普通字符。

如果连字符在开头，那也表示普通字符，不表示连续字符。

```
echo '-123456789' | egrep '[a-b]'   # not match
echo '-123456789' | egrep '[-ab]'   # match
```

1.3 选择结构（Alternation）

括号构成子表达式，“|”表示逻辑“或”。

```
Jeffrey|Jeffery
Jeff(re|er)y
```

1.4 注意

比较下面二者的区别：

```
[ \t]*
( *|t*)
```

第一个匹配的内容要么全是空格，要么全是 TAB；第二个可以匹配空格和 TAB 混合。

以下两行是相等的（不过字符范围速度更快）：

```
[ \t]*
( |t)*
```

1.5 重复控制

1.5.1 区间量次（Interval Quantifier）

“{min,max}”规定重复出现的次数：

```
echo '1234567890' | egrep '[0-9]{8,15}'
```

1.5.2 选项元素（Optional Items）

“?”相当于“{0,1}”。“July?”可以匹配“Jul”或“July”。

1.5.3 其他量词：重复出现（Other Quantifier: Repetition）

“*”相当于“{0,n}”。

“+”相当于“{1,n}”；

1.6 括号与反向引用（Parentheses and Backreferences）

在很多版本的正则表达式中，括号中的子表达式能“记住”匹配的内容。“verb|/num|”可以代表第几个子表达式匹配的内容。如，要查找重复的单词：

```
echo 'that that' | egrep '\<([A-Za-z]+) +\1\>'
```

“([a-z])([0-9])\1\2”这个表达式中，“\1”表示第一个表达式“[0-9]”匹配的内容；“\2”表示第二个表达式“[0-9]”匹配的内容。

1.7 转义元字符

反斜线“\”实现元字符的转义。大多数正则工具会把字符范围“[...]”中的“\”作为普通字符。

Chapter 2

拓展

2.1 引用匹配的内容

在 Perl 语言中通过$num 取得匹配的表达式内容：

```
if("-111.222F" =~ m/^([+-]?[0-9]+(\.[0-9]*)?)([CF])$/) {
  print "$1\n"; # -111.222
  print "$2\n"; # .222
  print "$3\n"; # F
}
```

通过$(?:...) 只用来分组，但是不取得匹配内容：

```
if("-111.222F" =~ m/^([+-]?[0-9]+(?:\.[0-9]*)?)([CF])$/) {
  print "$1\n"; # -111.222
  print "$2\n"; # F
}
```

回到温度转换的例子，根据用户输入最后是 C 还是 F 来判断输入的类型：

```
print "Enter a temperature in input(e.g. 32.5F, 10.0C): \n";

$input = <STDIN>;
chomp($input); # remove \n at end of line

if($input =~ m/^([+-]?[0-9]+(\.[0-9]*)?)([CF])$/) {
  $number = $1;
  $type = $3;
  if("C" eq $type){
    $celsius = $number;
```

```
11      $fahrenheit = ($input*9/5)+32;
12    } else {
13      $fahrenheit = $number;
14      $celsius = ($input-32)*5/9;
15    }
16    printf "%.2f C is %.2f F.\n", $celsius, $fahrenheit ;
17  } else {
18    print "Expecting a number, don't understane \"$input\".\n";
19  }
```

2.2 环视功能（lookaround）

环视具体有以下四种：

顺序肯定环视“(?=...)”：某个位置的右边符合子表达式。

顺序否定环视“(?!...)”：某个位置的右边不符合子表达式。

逆序肯定环视“(?<=...)”：某个位置的左边符合子表达式。

逆序否定环视“(?<!...)”：某个位置的左边不符合子表达式。

2.2.1 环视只匹配位置

环视功能只匹配位置，而不匹配具体的字符（就像是行头“^”、字符分界符“\b”）。它匹配的是某一个位置前后的内容是否符合。

例如：表达式“Jeffrey”匹配的是一串文本：

```
by Jeffrey Friedl.
   ^-----^
```

而环视“(?=Jeffrey)”匹配的的两个字符之间的位置：

```
by Jeffrey Friedl.
 ->|<-
```

再看一个例子，“(?=Jeffery)Jeff”和“Jeff(?=rey)”是等价的：

“(?=Jeffery)Jeff”：从“Jeffery”的开头位置开始找“Jeff”。

“Jeff(?=rey)”：找到后面有“rey”的“Jeff”。

2.2.2 利用环视来查找替换

把“Jeffs”替换为“Jeff's”可以有很多种实现：

不用环视 (性能最好)：“s/Jeffs/Jeff's/g”。

单词分界锚点 (同上)：“s/\bJeffs\b/Jeff's/g”。

使用先分组然后再替换：“s/\b(Jeff)(s)\b/$1'$2/g”

通过环视：“s/\bJeff(?=s\b)/Jeff'/g”

在环视的例子中，环视的内容并不在最终匹配的文本中，因为环视只匹配位置而不包括任何字符。更进一步，我们可以把前面的“Jeff”也放入环视：

s/(?<=\bJeff)(?=s\b)/'/g

这样我们只要对应的位置插入了一个字符。

2.2.3 利用环视来格式化数字

以格式化数字“123456789”为“123,456,789”为例，说明环视功能。

算法：左边有数字“\d”，而且右边的数字个数正好是 3 的倍数“(\d\d\d)+$”。

Chapter 3

元字符

3.1 数值传义

3.1.1 通过八进制转义

格式为“\num”，例如：“\015\012”。取值范围一般在“\000”到“\377”之间，而且通常要求以 0 开头。

3.1.2 通过十六进制转义

格式有：“\xnum”、“\x{num}”、“\unum”、“Unum”。

3.2 字符集合

Table 3.1: 常用正则元字符

元字符	作用
\s	空白字符（包括空格、制表符、换行）
\S	除了“\s”以外的任何字符
\w	“[A-Za-z0-9]”
\W	除了“\W”以外的任何字符
\d	“[0-9]”
\D	除了“\d”以外的任何字符

3.3 Unicode 属性

Unicode 不仅是字符的映射，还记录了每个字符的属性（是大写还是小写字符、是从右向左读的…… ）。

匹配属性的格式为“\p{Prop}”或“\P{Prop}”。如属性“\p{L}”匹配了字符属性（相对于数字、标点、口音等），还有相等的多字母表示方式“p{Letter}”。还有些系统中在单字符版本中可以省略花括号。还有些系统中可以加上条件“In”或“Is”来限制条件，如“\p{IsL}”。

许多属性还可以进一步加上子属性，如字符可以再一步描述是大写字符还是小写字符。

3.3.1 字母属性

“\p{L}”或“\p{Letter}”字母。

“\p{Ll}”或“\p{Lowercase_Letter}”小写字母。

“\p{Lu}”或“\p{Uppercase_Letter}”大写字母。

“\p{Lt}”或“\p{Titlecase_Letter}”出现在单词开头的字母（某些语言单词组合中会有 ）。

“\p{L&}”包含了“\p{Ll}”“\p{Lu}”“\p{Lt}”三者的集合。

“\p{Lm}”或“\p{Modifier_Letter}”少数的样子像字母，其实是特殊用途的字符。

“\p{Lo}”或“\p{Other_Letter}”没有大小写、也不是修饰符的字母。包括希伯来语、阿拉伯语、日语中的字母。

“\p{M}”或“\p{Mark}”重音符号等修饰符号，不能单独出现。

“\p{Mn}”或“\p{Non_Spacing_Mark}”修饰其他字符的重音符、变音符等。

“\p{Mc}”或“\p{Spacing_Combining_Mark}”会占一定宽度的修饰符，孟加拉语、马来语中有。

“\p{Me}”或“\p{Encolsing_Mark}”可以围住其他字符的标记，圆圈、方框等。

“\p{Z}”或“\p{Separator}”空白的分隔符。

“\p{Zs}”或“\p{Space_Separator}”空格、制表符等。

“\p{Zl}”或“\p{Line_Separator}”LINE SEPAPATOR（U+2028 ）。

“\p{Zp}”或“\p{Paragraph_Separator}”PAPAGRAPH SEPARATOR（U+2029 ）。

“\p{S}”或“\p{Symbol}”图形与符号。

“\p{Sm}”或“\p{Math_Symbol}”数学符号，加减乘除等。

“\p{Sc}”或“\p{Currency_Symbol}”货币符号。

“\p{Sk}”或“\p{Modifier_Symbol}”组合字符，但作为功能完整的字符有自己的意义。

“\p{So}”或“\p{Other_Symbol}”印刷符号、框图等。

“\p{N}”或“\p{Number}”数字。

“\p{Nd}”或“\p{Decimal_Digit_Number}”各种表示 0 到 9 的数字（但不包括中日韩）。

“\p{Nl}”或“\p{Letter_Number}”几乎所有的罗马数字。

“\p{No}”或“\p{Other_Number}”作为加密符号与记数符号，（但不包括中日韩）。

“\p{P}”或“\p{Punctuation}”标点符号。

“\p{Pd}”或“\p{Dash_Punctuation}”各种连字符与短划线。

“\p{Ps}”或“\p{Open_Punctuation}”像是（、《等开符号。

“\p{Pe}”或“\p{Close_Punctuation}”像是)、》等闭符号。

“\p{Pi}”或“\p{Initial_Punctuation}”像是“、< 等。

“\p{Pf}”或“\p{Final_Punctuation}”像是”、> 等。

“\p{Pc}”或“\p{Connector_Punctuation}”少数有特殊语法含义的标点，如下划线。

“\p{Po}”或“\p{Other_Punctuation}”其他标点，句点、感叹号等。

“\p{C}”或“\p{Other}”其他任何字符。

“\p{Cc}”或“\p{Control}”ASCII 和 Latin-1 编码中的控制字符。

“\p{Cf}”或“\p{Format}”表示格式的不可见字符。

“\p{Co}”或“\p{Private_Use}”私人用途，如公司的 Logo 等。

“\p{Cn}”或“\p{Unassigned}”未分配的代码点。

3.3.2 字母表（Scripts）

字母表匹配一个语系中独有的字符，如“\p{Hebrew}”匹配只有希伯莱文才有的字符。

有些字符不属于任何字母表（如句点或空格）而属于通用字母表，用“\p{IsCommon}”匹配。还有一个伪字母表 Inherited 包括从其所属的字母表中基本字符继承而来的组合字符。

3.3.3 区块（Block）

区块代码了一段连续的代码（通常与语言地区相关），如西藏字符在 Perl 与 java.util.regex 中可以用“\p{InTibetan}”来匹配。

3.3.4 字母表与区块

属于某个字母表的字符可能同时包含多个区块，而且字母表和区块很容易混淆。如 Unicode 同时提供了 Tibetan 字母表和 Tibetan 区块。

3.4 字符集合的集合运算

3.4.1 简单的排除运算

.NET 提供了简单的排除（减法）运算，如“[[a-z]-[aeiou]]”就取出了所有的辅音。再看一个排除标点符号中除了书名号括号等成对符号的例子：“[\p{P}-[\p{Ps}\p{Pe}]”。

3.4.2 完整的字符集合运算

Sun 的 Java 正则包提供了完整的字符集合运算（并、交、减）。

并集运算：“[abcxyz]”相同的表示有“[[abc][xyz]]”、“[abc[xyz]]”、“[[abc]xyz]”。

交集运算：“[\p{InThai}&&\p{Cn}]”

排除运算：“[\p{InThai}&&[^\p{Cn}]]”。上一节中.NET 取辅音的例子可以写为：“[[a-z]&&[^aeiou]]”。

Part II

语言与工具

Chapter 4

egrep

4.1 基本使用

在邮件中查找发信人与主题的例子：

```
egrep '^(From|Subject):' ./*
```

4.2 忽略大小写

```
egrep -i '^(From|Subject):' ./*
```

4.3 反向引用

有些版本的 egrep 有个 bug：使用“-i”忽略大小写时会对反向引用无效，即可以查到“the the”但是查不到“The the”。

```
echo 'the the' | egrep '\<([A-Za-z]+) +\1\>'
the the
echo 'the The' | egrep '\<([A-Za-z]+) +\1\>'
```

Chapter 5

Perl

5.1 元字符

5.1.1 空白字符

Table 5.1: Perl 正则元字符

元字符	作用
\t	制表符
\n	换行
\b	一般情况表示下单词分界，但在字符范围中表示退格。
\s	空白字符（包括空格、制表符、换行）
\S	除了“\s”以外的任何字符
\w	“[A-Za-z0-9]”
\W	除了“\W”以外的任何字符
\d	“[0-9]”
\D	除了“\d”以外的任何字符

5.2 基本使用

查找文件中接连重复出现的单词

```
1 $/ = ".\n";
2   while (<>) {
3     next if !s/\b([a-z]+)((?:\s|<[^>]+>)+)(\1\b)/\e[7m$1\e[m$2\e[7
          m$3\e[m/ig;
```

```
4     s/^(?:[^s\e]*\n)+//mg;  # remove the unmarked line
5     s/^/$ARGV: /mg;     # add filename before the line
6     print;
7   }
```

5.2.1 变量的声明与引用

普通变量以美元符开头，而且可以在输出语句中直接使用。

以下是一个转摄氏度为华氏度的例子：

```
1 $celsius = 30;
2 $fahrenheit = ($celsius*9/5)+32;
3 print "$celsius C is $fahrenheit F.\n";
```

5.2.2 控制结构

```
1 $celsius = 20;
2 while($celsius <= 45) {
3   $fahrenheit = ($celsius*9/5)+32;
4   print "$celsius C is $fahrenheit F.\n";
5   $celsius = $celsius + 5;
6 }
```

运行时可以通过参数“-w”打开编译警告：

```
1 perl -w exp03.pl
```

5.3 用正则匹配文本

“=~”指定正则操作的对象。

“m/.../”表示通过正则进行的操作是匹配操作（可以省略 m，但是加上看起来更加清楚）。

“==”用来比较两个数字是否相等。

“eq”用来比较两个字符串是否相等。

查找是否是数字：

```
if($reply =~ m/^[0-9]+$/) {
  print "Only digits\n";
} else {
  print "Not only digits\n";
}
```

5.4 取得用户输入

增加能够处理小数部分；并通过函数“printf”格式化输出。

```
print "Enter a temperature in Celsius: \n";

$celsius = <STDIN>;
# remove \n at end of line
chomp($celsius);

if($celsius =~ m/^[+-]?[0-9]+(\.[0-9]*)?$/) {
  $fahrenheit = ($celsius*9/5)+32;
  # format output use printf
  printf "%.2f C is %.2f F.\n", $celsius, $fahrenheit ;
} else {
  print "Expecting a number, don't understane \"$celsius\".\n";
}
```

5.5 引用匹配的内容

通过$num 取得匹配的表达式内容：

```
if("-111.222F" =~ m/^([+-]?[0-9]+(\.[0-9]*)?)([CF])$/) {
  print "$1\n"; # -111.222
  print "$2\n"; # .222
  print "$3\n"; # F
}
```

通过$(?:...) 只用来分组，但是不取得匹配内容：

```
if("-111.222F" =~ m/^([+-]?[0-9]+(?:\.[0-9]*)?)([CF])$/) {
  print "$1\n"; # -111.222
  print "$2\n"; # F
}
```

回到温度转换的例子，根据用户输入最后是 C 还是 F 来判断输入的类型：

```
print "Enter a temperature in input(e.g. 32.5F, 10.0C): \n";

$input = <STDIN>;
chomp($input); # remove \n at end of line

if($input =~ m/^([+-]?[0-9]+(\.[0-9]*)?)([CF])$/) {
  $number = $1;
  $type = $3;
  if("C" eq $type){
    $celsius = $number;
    $fahrenheit = ($input*9/5)+32;
  } else {
    $fahrenheit = $number;
    $celsius = ($input-32)*5/9;
  }
  printf "%.2f C is %.2f F.\n", $celsius, $fahrenheit ;
} else {
  print "Expecting a number, don't understane \"$input\".\n";
}
```

5.6 修饰符

5.6.1 忽略大小写

在正则以后加个修饰符“/i”表示忽略大小写。

```
$input =~ m/^aaa$/i;
```

更加完整的温度转换例子：

```
print "Enter a temperature in input(e.g. 32.5F, 10.0C): \n";

$input = <STDIN>;
chomp($input); # remove \n at end of line

if($input =~ m/^([+-]?[0-9]+(\.[0-9]*)?)([CF])$/) {
  $number = $1;
  $type = $3;
  if("C" eq $type){
    $celsius = $number;
```

```
    $fahrenheit = ($input*9/5)+32;
  } else {
    $fahrenheit = $number;
    $celsius = ($input-32)*5/9;
  }
  printf "%.2f C is %.2f F.\n", $celsius, $fahrenheit ;
} else {
  print "Expecting a number, don't understane \"$input\".\n";
}
```

5.6.2 全局匹配

修饰符“/g”表示全局匹配。就是在完成了一次匹配以后，再继续匹配剩下的内容。

```
$input =~ m/^aaa$/g;
```

5.6.3 宽松排列表达式

修饰符“/x”表示。

```
$input =~ m/^aaa$/x;
```

5.7 替换文本

“$var =~ s/regex/replacement/”以变量“$var”为对象，把符合正则的内容替换掉。

例：无视大小写，把“peter”替换成“Peter”。

```
$var =~ s/\bpeter\b/Peter/i;
print "$var";
```

5.7.1 使用 perl 自动替换文本

参数“-p”表示对目标文件每一行进行查找和替换；参数“-i”表示替换的结果写回文件；参数“-e”表示后面的字符串就是程序的代码；

```
perl -p -i -e 's/sysread/read/g' filename
```

可以合并参数为：

```
perl -pi -e 's/sysread/read/g' filename
```

5.7.2 生成邮件回复的例子

原始内容在文件“file.in”，通过程序“mkreply.pl”把结果存放在“file.out”。

file.in

```
From elvis Thu Feb 29 11:15 2007
Received: from elvis@localhost by tabloid.org (8.11.3) id KA8CMY
Received: from tabloid.org by gateway.net (8.12.5/2) id N8XBK
To: jfried@regex.info (Jeffrey Friedl)
From: elvis@tabloid.org (The King)
Date: Thu, Feb 29 2007 11:15
Message-Id: <200702239939.KA8CMY@tabloid.org>
Subject: Be seein' ya around
Reply-To: elvis@hh.tabloid.org
X-Mailer: Madam Zelda's Psychic Orb [version 3.7 PL92]

Sorry I haven't been around lately. A few years back I checked into
    that ole heartbreak hotel in the sky, ifyaknowwhatImean.
The Duke says "hi".
                Elvis
```

希望程序能自动生成回复的样式：

```
To: elvis@hh.tabloid.org (The King)
From: jfriedl@regex.info (Jeffery Friedl)
Subjet: Re: Be seein' ya around

On Thu, Feb 29 2007 11:15 The King wrote:
|>  Sorry I haven't been around lately. A few years back i checked
|> into the ole heartbreak hotel in the sky, ifyaknowwhatImaen.
|> The Duke says "hi".
|>        Elvis
```

调用的方法：

```
perl -w mkreply.pl file.in > file.out
```

5.8 从文件读取

Perl 提供了操作符“<>”把每一行读取到变量中，我们通过“^\s*$”检查空行（表示邮件 head 结束）。

```
while($line = <>) {
  # ... deal with $line ...
  if($line =~ m/^\s*$/) {
    last; # jump out of loop
  }
}
```

从邮件头中提取信息的方法，以 Subject 为例：

```
if($line =~ m/^Subject: (.*)/i) {
  $subject = $1;
}
```

分别取得回信地址和昵称：

```
# From: elvis@tabloid.org (The King)
if($line =~ m/^From: (\S+) \(([^()]*)\)/i) {
  $reply_address = $1;
  $from_name = $2;
}
```

就连输出原文引用的部人也可以通过正则来实现：

```
$line =~ s/^/|> /;
print $line
```

5.9 增强锚点

通常来说，锚点“^”、“$”匹配的不是逻辑行的开头与结尾，而是整个字符串的开头与结尾。如果要匹配逻辑行可以切换到增强锚点（enhanced line anchor）模式下。在 Perl 语言中修饰符为“/m”。例如，把空白行替换为 HTML 的段落符“<p>”：

```
$text =~ s/^$/<p>/mg;
```

5.10 格式化

为了增加可读性，修饰符“/x”允许对正则表达式进行排版（还可以用花括号代替斜线）：

```
$text =~ s{
  \b
  # save email address to variable
  {
    username-regex
    \@
    hostname-regex
  }
  \b
}{<a href="mailto$1">$1</a>}gix
```

5.11 重用正则对象

修饰符“qr/.../”表示不应用到字符串，建立一个对象以后再用：

```
$tmpRegex = qr/china/i;

$text =~ s{
  $tmpRegex
}{CHINA}gix
```