Perl

翻译作品，来自 *perl* 官网

屈庆磊

quqinglei@icloud.com

2013 年 *8* 月 *29* 日

目录

1 说明

文中所有带 [tbd] 字样的地方说明有疑问，还未曾解决！

2 Perl 正则快速入门

2.1 名字

本文讲述 perl 正则表达式如何快速入门，基本采用例子讲解没有完全按照原文进行翻译，主要是使用其中的例子。http://perldoc.perl.org

2.2 描述

本文包含了 Perl 语言的最基本的正则表达式理解、创建、使用

2.3 指导

2.3.1 单词匹配

最简单的正则表达式其实就是一个单词，或者多个单词，由多个字符组成的字符串。含有一个单词的正则表达式可以匹配包含此单词的任意字符串。

```
1 "Hello World" =~ /World/; # 匹配
```

在上面的语句中，= 是匹配操作符，//表示包含的是模式串。可以理解为，字符串 "Hello World" 中是否有模式 World，如果有则他们匹配，如果把以上语句赋值给一个变量，那么如果匹配，变量值为 1，反之为空[1]

下面的语句很简单，就不一一细说了，注意符号!= 为不匹配符。还有第四行和第五行其实就是说明了一个变量替换的道理，也就是说正则也是可以用变量替换表达的。

```
1 print "It matches\n" if "Hello World" =~ /World/;
2 print "It doesn't match\n" if "World Hello" !~ /Hello/;
3
4 $greeting = "Adam";
5 print "It matches\n" if "Hello Adam" =~ /$greeting/;
```

如果你要匹配：$_ ，就不需要：$_ = 了。看下面代码：

```
1 $_ = "Hello World";
2 print "It matches\n" if /World/;
```

模式匹配符 // 也是可以被替换成其他任意的符号，只需要在分隔符号前面加一个：'m' 即可，如下所示：

```
1 "Hello World" =~ m!World!; # 匹配，使用'!' 作为分隔符
2 "Hello World" =~ m{World}; # 匹配，使用'{}' 包含模式匹配
3 "/usr/bin/perl" =~ m"/perl" # '/' 变成了一般符号
```

正则表达式必须精确匹配字符串的一部分，才能保证此语句为真，空格也是字符！

[1]注意，如果不匹配得到的结果并不是 0，而应该是 NULL

```
"Hello World" =~ /world/; # 不匹配，大小写问题
"Hello World" =~ /o W/; # 匹配， ' ' 也是一般字符
"Hello World" =~ /World /; # 不匹配，字符串后面没有 ' '(空格)
```

Perl 会先匹配字符串前面的匹配点，也就是说从字符串的开头开始匹配，一旦匹配就返回真:

```
"Hello World" =~ /o/; # 它匹配的其实是 'Hello' 中的 'o'
"That hat is red" =~ /hat/; # 其实它匹配的是 'That' 中的 'hat' 字符串
```

在 Perl 中，并不是所有的字符就可以直接的匹配，有些字符属于元字符(metacharacters)[2]需要转义后才可以匹配。下面给出了正则表达式中的元字符:

```
{}[]()^$.|*+?\
```

在元字符前面使用反斜杠
转义就可以匹配啦，如下所示:

```
"2+2=4" =~ /2+2/; # 不匹配，原因是'+' 属于元字符
"2+2=4" =~ /2\+2/; # 匹配，我们把'+' 给转义成为了普通字符
'C:\WIN32' =~ /C:\\WIN/; # 匹配，我们使用反斜杠把反斜杠给转义成为了普通字符
"/usr/bin/perl" =~ /\/usr\/bin\/perl/; # 匹配，不解释了，可以参考下面这个语句

# 当我们使用其他字符作为模式匹配符号时，就不需要对 '/' 进行转义了，它只是一个
# 模式匹配字符，并不是元字符
print "matches!\n" if "/usr/bin/perl" =~ m{/usr/bin/perl};
```

不可打印字符可以使用转义序列表示，如:
t 可以表示为制表符
n 可以表示为换行符
r 可以表示为回车。另外任意字节都可以通过八进制或者十六进制数字的转义序列表示，下面一些例子:

```
"1000\t2000" =~ m(0\t2); # 匹配
"cat" =~ /\143\x61\x74/ # 匹配，其中'143' 是 c 的八进制 ASCII，x61、x74 分别为 a、t 的十六进制 ASCII 码
```

可以使用变量替换正则表达式:

[2]所谓元字符就是指那些在正则表达式中具有特殊意义的专用字符，可以用来规定其前导字符（即位于元字符前面的字符）在目标对象中的出现模式

```
$foo = 'house';
'cathouse' =~ /cat$foo/; # 使用 $foo 变量替换，匹配
'housecat' =~ /${house}cat/; # 匹配
```

废话就不翻译了，我们可以使用锚点来确定匹配位置，比如使用:^ 来表示匹配字符串开头，使用 $ 来匹配字符串结尾。如下代码所示:

```
"housekeeper" =~ /keeper/; # 匹配
"housekeeper" =~ /^keeper/; # 不匹配
"housekeeper" =~ /keeper$/; # 匹配
"housekeeper" =~ /^housekeeper/; # 匹配
```

2.3.2 使用字符类

字符类可以表示一类字符，我们还是看例子吧，这样比较清楚

```
/cat/; # 匹配 cat
/[bcr]at/; # 匹配 bat cat rat
"abc" =~ /[cab]/; 先匹配到 a
```

在上面代码中的最后一样，匹配时最先匹配到的其实是'a' 虽然说它在正则表达式中的第二个，我们以为先匹配到的应该是 c，但是，perl 在处理此问题的时候对 [ ] 里面的字符进行了排序。

```
/[yY][eE][sS]/; # 不区分大小写匹配 'yes'
/yes/i; # 正则后面多了个 i 就可以代替上面一行代码的功能了！
```

在上面代码中最后一行，我们使用了一个字符i 它命令正则不区分大小写进行匹配。

```
/[\]c]def/; # 匹配 ']def' 或者 'cdef'
$x = 'bcr';
/[$x]at/; # 匹配 'bat', 'cat', 'rat'
/[\$x]at/; # 匹配 '$at', 'xat'
/[\\$x]at/; # 匹配 '\at', '$at', 'bat', 'cat', 'rat'
```

以上代码的目的：注意转义，正则可替换为变量，[] 里面的字符是或的关系。

特殊字符：- 可以称之为范围操作符，看例子就明白了:

```
/item[0-9]/; # 匹配'item0', 'item1', 'item2' ... 'item9'
/[0-9a-fA-F]/; # 匹配十六进制字符
```

在 [] 号里面括住的，注意符号 ，^ 她在不同的位置表示不同的意思，看代码：

```
/[^a]at/; # 不能匹配 'aat', 'at' 但可以匹配 ' at', 'bat', 'cat', 'dat', 'Oat', 等
/[^-9]/; # 匹配非数字字符
/[a^]at/; # 注意此时 ^ 只是一个单纯的普通字符，此表达式匹配 'aat', '^at'
```

Perl 还有几个缩写的字符类，下面是介绍，为方便我们用'<=>' 作为等价符号来介绍，她并不是 Perl 里面的符号：在以下代码中我提到的变量，其实意思是在语言中可以用来作为变量的字符

```
[0-9] <=> \d # \d 用来表示任意数字字符
[\ \t\r\n\f] <=> \s # \s 表示空格、制表符或者换行符、换页符，\f 已经不常用了，可以查看 ASCII 码表
[0-9a-zA-Z_] <=> \w # \w 可以这么理解，它代表可以作为变量的所有字符
\D <=> [^0-9] # 它表示非数字字符
\S <=> [^\s] # 她匹配所有非空格，制表符，换行符，换页符之外的字符
\W <=> [^\w] # 匹配任意非变量字符
```

. 号可以匹配除换行符以外任意字符

```
/\d\d:\d\d:\d\d/; # 匹配时间 hh:mm:ss
/[\d\s]/; # 匹配数字和空格字符，包含换行符、制表符、空格、换页符
/\w\W\w/; # 匹配一个变量字符中间一个非变量字符，然后后面是一个单词字符，如：a&b

/..rt/; # 匹配以 rt 结尾开头为任意两个字符的串
/end\./; # 匹配 'end.'
/end[.]/; # 匹配 'end.'
```

单词边界符号
b 这个不大好理解，看例子就明白啦：

```
$x = "Housecat catenates house and cat";
$x =~ /\bcat/; # 在 'catenates' 单词中匹配到 'cat'
$x =~ /cat\b/; # 在 'housecat' 单词中匹配到 'cat'
$x =~ /\bcat\b/; # 在最后匹配到 'cat'
```

2.3.3 匹配中的或操作

我们可以使用或元字符来匹配模式中的任意一个，下面例子可以说明这一点：

```
"cats and dogs" =~ /cat|dog|bird/; # 匹配 "cat"
"cats and dogs" =~ /dog|cat|bird/; # 匹配 "cat"
```

虽然在上面代码中的第二行，dog 在正则中的开头，但还是先匹配的'cat' 这是因为 Perl 在处理正则表达式的时候对其进行了字典排序。

```
"cats" =~ /c|ca|cat|cats/; # 匹配到 "c"
"cats" =~ /cats|cat|ca|c/; # 匹配到 "cats"
```

2.3.4 分组分层匹配

```
/(a|b)b/; # 匹配 'ab' 或者 'bb'
/(^a|b)c/; # 匹配以 a 开头的 'ac' 或者任意位置的 'bc'
/house(cat|)/; # 匹配 'housecat' 或者 'house'
"20" =~ /(19|20)\d\d/; # 无法匹配到'20' 所以匹配结果是 NULL
print "$1\n"; # 得到的结果是空
```

2.3.5 提取匹配字符串