正则表达式 30 分钟入门教程(第二版)

目录

本文目标

30 分钟内让你明白正则表达式是什么，并对它有一些基本的了解，让你可以在自己的程序或网页里使用它。

如何使用本教程

别被下面那些复杂的表达式吓倒，只要跟着我一步一步来，你会发现正则表达式其实并不像你想像中的那么困难。当然，如果你看完了这篇教程之后发现自己明白了 很多，却又几乎什么都记不得，那也是很正常的——其实我认为没接触过正则表达式的人在看完这篇教程后能把提到过的语法记住 80%以上的可能性为零。这里只 是让你明白基本道理，以后你还需要多练习，多查资料，才能熟练掌握正则表达式。

除了作为入门教程之外，本文还试图成为可以在日常工作中使用的正则表达式语法参考手册（就作者本人的经历来说，这个目标还是完成得不错的）。

文本格式约定：**专业术语** 元字符/语法格式 正则表达式 正则表达式中的一部分（用于分析） *用于在其中搜索的字符串* 对正则表达式或其中一部分的说明

什么是正则表达式？

在编写处理字符串的程序或网页时，经常会有查找符合某些复杂规则的字符串的需要。**正则表达式**就是用于描述这些规则的工具。换句话说，正则表达式就是记录文本规则的代码。

很可能你使用过Windows/Dos下用于文件查找的**通配符(wildcard)**，也就是*和?。如果你想查找某个目录下的所有的Word文档的话，你会搜索*.doc。在这里，*会被解释成任意的字符串。和通配符类似，正则表达式也是用来进行文本匹配的工具，只不过比起通配符它能更精确地描述你的需求——当然，代价就是更复杂。比如你可以编写一个正则表达式来查找所有以 0 开头，后面跟着 2-3 个数字，然后是一个连字号“-”，最后是 7 或 8 位数字的字符串(像 *010-12345678* 或 *0376-7654321*)。

正则表达式是用于进行文本匹配的工具，所以本文里多次提到了在字符串里搜索/查找，这种说法的意思是在给定的字符串中，寻找与给定的正则表达式相匹配的部分。有可能字符串里有不止一个部分满足给定的正则表达式，这时每一个这样的部分被称为一个匹配。**匹配**在本文里可能会有三种意思：一种是形容词性的，比如说一个字符串匹配一个表达式；一种是动词性的，比如说在字符串里匹配正则表达式；还有一种是名词性的，就是刚刚说到的“字符串中满足给定的正则表达式的一部分”。

入门

学习正则表达式的最好方法是从例子开始，理解例子之后再自己对例子进行修改，实验。下面给出了不少简单的例子，并对它们作了详细的说明。

假设你在一篇英文小说里查找hi，你可以使用正则正则表达式hi。

这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或标点符号或换行来分隔的，但是\b并不匹配这些单词分隔符中的任何一个，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\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*前边的内容可以重复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。

如果同时使用其它的一些元字符，我们就能构造出功能更强大的正则表达式。比如下面这个例子：

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是一个新的元字符，匹配任意的数字(0，或 1，或 2，或……)。-不是元字符，只匹配它本身——连字号。

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}

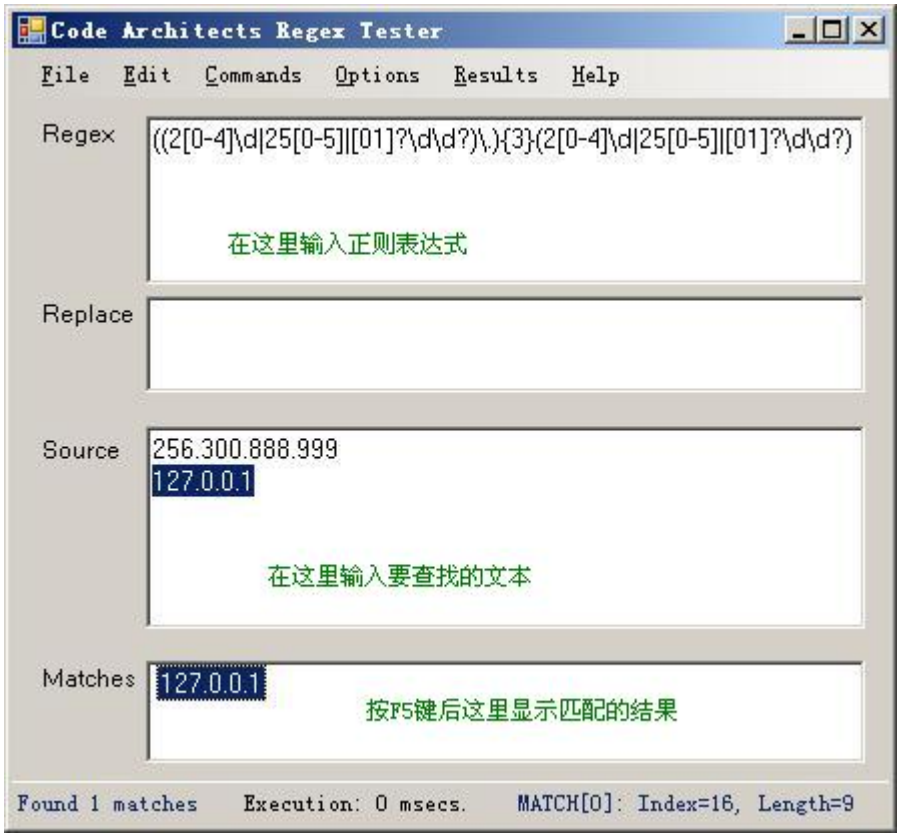
这里\d后面的{2}({8})的意思是前面\d必须连续重复匹配 2 次(8 次)。

测试正则表达式

如果你不觉得正则表达式很难读写的话，要么你是一个天才，要么，你不是地球人。正则表达式的语法很令人头疼，即使对经常使用它的人来说也是如此。由于难于读写，容易出错，所以很有必要创建一种工具来测试正则表达式。

由于在不同的环境下正则表达式的一些细节是不相同的，本教程介绍的是 Microsoft .Net 2.0 下正则表达式的行为，所以，我向你介绍一个.Net下的工具Regex Tester。首先你确保已经安装了.Net Framework 2.0，然后下载 Regex Tester，下载完后打开压缩包，直接运行RegexTester.exe。

下面是 Regex Tester 运行时的截图：



元字符

现在你已经知道几个很有用的元字符了，如\b,.,*，还有\d.当然还有更多的元字符，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。

下面来试试更多的例子：

\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)（好吧，现在我们说说这里的单词是什么意思吧：就是几个连续的\w。不错，这与学习英文时要背的成千上万个同名的东西的确关系不大）。

\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 个字母/数字的单词。

表 1. 常用的元字符

代码	说明
.	匹配除换行符以外的任意字符
\w	匹配字母或数字或下划线或汉字

\s 匹配任意的空白符
\d 匹配数字
\b 匹配单词的开始或结束
^ 匹配字符串的开始
$ 匹配字符串的结束

元字符^（和 6 在同一个键位上的符号）以及$和\b 有点类似，都匹配一个位置。^匹配你要用来查找的字符串的开头，$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}则是必须重复最少 5 次，最多 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。

字符转义

如果你想查找元字符本身的话，比如你查找.，或者*，就出现了问题：你没法指定它们，因为它们会被解释成其它的意思。这时你就必须使用\来取消这些字符的特殊意义。因此，你应该使用\.和*。当然，要查找\本身，你也得用\\.

例如：www\.unibetter\.com匹配www.unibetter.com，c:\\windows匹配c:\windows，2\^8 匹配2^8(通常这是 2 的 8 次方的书写方式)。

重复

你已经看过了前面的*，+，{2}，{5,12}这几个匹配重复的方式了。下面是正则表达式中所有指定重复的方式：

表 2. 常用的限定符

代码/语法	说明
*	重复零次或更多次
+	重复一次或更多次
?	重复零次或一次
{n}	重复n次
{n,}	重复n次或更多次

{n,m}　　重复n到m次

下面是一些使用重复的例子：

Windows\d+匹配Windows后面跟 1 个或更多数字

13\d{9}匹配以 13 后面跟 9 个数字(中国的手机号)

^\w+匹配一行的第一个单词(或整个字符串的第一个单词，具体匹配哪个意思得看选项设置)

字符类

要想查找数字，字母或数字，空白是很简单的，因为已经有了对应这些字符集的元字符，但是如果你想匹配没有预定义元字符的字符集比如元音字母(a,e,i,o,u)，怎么办？

很简单，你只需要在中括号里列出它们就行了，像[aeiou]就匹配任何一个元音字母，[.?!]匹配标点符号(.或?或!)(英文语句通常只以这三个标点结束)。注意，我们不需要写成[\.\?!]。

我们也可以轻松地指定一个字符**范围**，像[0-9]代表的含意与\d就是完全一致的：一位数字，同理[a-z0-9A-Z_]也完全等同于\w（如果只考虑英文的话）。

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({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，它也能匹配 *010)12345678* 或 *(022-87654321* 这样的“不正确”的格式。要解决这个问题，请在本教程的下面查找答案。

反义

有时需要查找不属于某个能简单定义的字符类的字符。比如想查找除了数字以外，其它任意字符都行的情况，这时需要用到**反义**：

表 3.常用的反义代码

代码/语法	说明
\W	匹配任意不是字母，数字，下划线，汉字的字符
\S	匹配任意不是空白符的字符
\D	匹配任意非数字的字符

\B　　匹配不是单词开头或结束的位置
[^x]　　匹配除了x以外的任意字符
[^aeiou]　匹配除了aeiou这几个字母以外的任意字符

例子：\S+匹配不包含空白符的字符串。

<a[^>]+>匹配用尖括号括起来的以a开头的字符串。

替换

好了，现在终于到了解决 3 位或 4 位区号问题的时间了。正则表达式里的**替换**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。你可以试试用替换|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：**使用替换时，顺序是很重要的**。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)。原因是匹配替换时，将会从左到右地测试每个分枝条件，如果满足了某个分枝的话，就不会去管其它的替换条件了。

Windows98|Windows2000|WindosXP 这个例子是为了告诉你替换不仅仅能用于两种规则，也能用于更多种规则。

分组

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）；但如果想要重复一个字符串又该怎么办？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(IP 地址中每个数字都不能大于 255)。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。

后向引用

使用小括号指定一个子表达式后，**匹配这个子表达式的文本**可以在表达式或其它程序中作进一步的处理。默认情况下，每个分组会自动拥有一个**组号**，规则是：从左向右，以分组的左括号为标志，第一个出现的分组的组号为 1，第二个为 2，以此类推。

后向引用用于重复搜索前面某个分组匹配的文本。例如，\1 代表分组 1 匹配的文本。难以理解？请看示例：

\b(\w+)\b\s+\1\b可以用来匹配重复的单词，像*go go*, *kitty kitty*。首先是一个单词，也就是单词开始处和结束处之间的多于一个的字母或数字(\b(\w+)\b)，然后是1 个或几个空白符(\s+，最后是前面匹配的那个单词(\1)。

你也可以自己指定子表达式的组号或**组名**。要指定一个子表达式的组名，请使用这样的语法：(?<Word>\w+),这样就把\w+的组名指定为 Word 了。要反向引用这个分组**捕获**的内容，你可以使用\k<Word>,所以上一个例子也可以写成这样：
\b(?<Word>\w+)\b\s*\k<Word>\b。

使用小括号的时候，还有很多特定用途的语法。下面列出了最常用的一些：

表 4. 分组语法

	捕获
(exp)	匹配exp,并捕获文本到自动命名的组里
(?<name>exp)	匹配exp,并捕获文本到名称为name的组里，也可以写成(?'name'exp)
(?:exp)	匹配exp,不捕获匹配的文本
	位置指定
(?=exp)	匹配exp前面的位置
(?<=exp)	匹配exp后面的位置
(?!exp)	匹配后面跟的不是exp的位置

(?<!exp)	匹配前面不是exp的位置

注释

(?#comment)	这种类型的组不对正则表达式的处理产生任何影响，只是为了提供让人阅读注释

我们已经讨论了前两种语法。第三个(?:exp)不会改变正则表达式的处理方式，只是这样的组匹配的内容不会像前两种那样被捕获到某个组里面。

位置指定

接下来的四个用于查找在某些内容(但并不包括这些内容)之前或之后的东西，也就是说它们用于指定一个位置，就像\b, ^, $那样，因此它们也被称为**零宽断言**。最好还是拿例子来说明吧：

(?=exp)也叫**零宽先行断言**，它匹配文本中的某些位置，这些位置的后面能匹配给定的后缀exp。比如\b\w+(?=ing\b)，匹配以ing结尾的单词的前面部分(除了ing以外的部分)，如果在查找*I'm singing while you're dancing.*时，它会匹配sing和danc。

(?<=exp)也叫**零宽后行断言**，它匹配文本中的某些位置，这些位置的前面能给定的前缀匹配exp。比如(?<=\bre)\w+\b会匹配以re开头的单词的后半部分(除了re以外的部分)，例如在查找*reading a book*时，它匹配ading。

假如你想要给一个很长的数字中每三位间加一个逗号(当然是从右边加起了)，你可以这样查找需要在前面和里面添加逗号的部分：((?<=\d)\d{3})*\b。请仔细分析这个表达式，它可能不像你第一眼看出来的那么简单。

下面这个例子同时使用了前缀和后缀：(?<=\s)\d+(?=\s)匹配以空白符间隔的数字(再次强调，不包括这些空白符)。

负向位置指定

前面我们提到过怎么查找**不是某个字符或不在某个字符类里**的字符的方法(反义)。但是如果我们只是想要**确保某个字符没有出现，但并不想去匹配它**时怎么办？例如，如果我们想查找这样的单词--它里面出现了字母 q,但是 q 后面跟的不是字母 u,我们可以尝试这样：

\b\w*q[^u]\w*\b匹配包含**后面不是字母u的字母q**的单词。但是如果多做测试(或者你思维足够敏锐，直接就观察出来了)，你会发现，如果q出现在单词的结尾的话，像Iraq,Benq，这个表达式就会出错。这是因为[^u]总是匹配一个字符，所以如果q是单词的最后一个字符的话，后面的[^u]将会匹配q后面的单词分隔符(可能是空格，或者是句号或其它的什么)，后面的\w+\b将会匹配下一个单词，于是\b\w*q[^u]\w*\b就能匹配整个*Iraq fighting*。**负向位置指定**能解决这样的

问题，因为它只匹配一个位置，并不**消费**任何字符。现在，我们可以这样来解决这个问题：\b\w*q(?!u)\w*\b。

零宽负向先行断言(?!exp)，只会匹配**后缀**exp不存在的位置。\d{3}(?!\d)匹配三位数字，而且这三位数字的后面不能是数字。

同理，我们可以用(?<!exp)，**零宽负向后行断言**来查找前缀exp不存在的位置：(?<![a-z])\d{7}匹配前面不是小写字母的七位数字(实验时发现错误？注意你的“区分大小写”先项是否选中)。

一个更复杂的例子：(?<=<(\w+)>).*(?=<\/\1>)匹配不包含属性的简单HTML标签内里的内容。(<?(\w+)>)指定了这样的前缀：被尖括号括起来的单词(比如可能是)，然后是.*(任意的字符串)，最后是一个后缀(?=<\/\1>)。注意后缀里的\/，它用到了前面提过的字符转义；\1 则是一个反向引用，引用的正是捕获的第一组，前面的(\w+)匹配的内容，这样如果前缀实际上是的话，后缀就是了。整个表达式匹配的是和之间的内容(再次提醒，不包括前缀和后缀本身)。

注释

小括号的另一种用途是能过语法(?#comment)来包含注释。例如：2[0-4]\d(?#200-249)|25[0-5](?#250-255)|[01]?\d\d?(?#0-199)。

要包含注释的话，最好是启用“忽略模式里的空白符”选项，这样在编写表达式时能任意的添加空格，Tab，换行，而实际使用时这些都将被忽略。启用这个选项后，在#后面到这一行结束的所有文本都将被当成注释忽略掉。

例如，我们可以前面的一个表达式写成这样：

```
        (?<=     # 查找前缀，但不包含它
        <(\w+)>  # 查找尖括号括起来的字母或数字(标签)
        )        # 前缀结束
        .*       # 匹配任意文本
        (?=      # 查找后缀，但不包含它
        <\/\1>   # 查找尖括号括起来的内容：前面是一个"/"，后面是先前捕获的标签
        )        # 后缀结束
```

贪婪与懒惰

当正则表达式中包含能接受重复的限定符(指定数量的代码，例如*,{5,12}等)时，通常的行为是(在使整个表达式能得到匹配的前提下)匹配**尽可能多**的字符。

考虑这个表达式：a.*b，它将会匹配最长的以a开始，以b结束的字符串。如果用它来搜索*aabab*的话，它会匹配整个字符串aabab。这被称为**贪婪**匹配。

有时，我们更需要**懒惰**匹配，也就是匹配**尽可能少**的字符。前面给出的限定符都可以被转化为懒惰匹配模式，只要在它后面加上一个问号?。这样.*?就意味着匹配任意数量的重复，但是在能使整个匹配成功的前提下使用最少的重复。现在看看懒惰版的例子吧：

a.*?b匹配最短的，以a开始，以b结束的字符串。如果把它应用于*aabab*的话，它会匹配aab和ab（为什么第一个匹配是aab而不是ab？简单地说，最先开始的区配最有最大的优先权——The Match That Begins Earliest Wins）。

表 5.懒惰限定符

*?	重复任意次，但尽可能少重复
+?	重复 1 次或更多次，但尽可能少重复
??	重复 0 次或 1 次，但尽可能少重复
{n,m}?	重复n到m次，但尽可能少重复
{n,}?	重复n次以上，但尽可能少重复

平衡组

如果想要匹配可嵌套的层次性结构的话，就得使用平衡组了。举个例子吧，如何把“xx〈aa〈bbb〉〈bbb〉aa〉yy”这样的字符串里，最长的括号内的内容捕获出来？

这里需要用到以下的语法构造：

- (?<group>) 把捕获的内容命名为 group,并压入堆栈
- (?<-group>) 从堆栈上弹出最后压入堆栈的名为 group 的捕获内容,如果堆栈本来为空，则本分组的匹配失败
- (?(group)yes|no) 如果堆栈上存在以名为 group 的捕获内容的话，继续匹配 yes 部分的表达式，否则继续匹配 no 部分
- (?!) 零宽负向先行断言，由于没有后缀表达式，试图匹配总是失败

如果你不是一个程序员（或者你是一个对堆栈的概念不熟的程序员），你就这样理解上面的三种语法吧：第一个就是在黑板上写一个（或再写一个）"group"，第二个就是从黑板上擦掉一个"group"，第三个就是看黑板上写的还有没有"group"，如果有就继续匹配 yes 部分，否则就匹配 no 部分。

我们需要做的是每碰到了左括号，就在黑板上写一个"group"，每碰到一个右括号，就擦掉一个，到了最后就看看黑板上还有没有一如果有那就证明左括号比右括号多,那匹配就应该失败(为了能看得更清楚一点,我用了(?'group')的语法)：

```
<                    #最外层的左括号
```

```
    [^<>]*                  #最外层的左括号后面的不是括号的内容
    (
        (
            (?'Open'<)      #碰到了左括号，在黑板上写一个"Open"
            [^<>>]*         #匹配左括号后面的不是括号的内容
        )+
        (
            (?'-Open'>)     #碰到了右括号，擦掉一个"Open"
            [^<>]*          #匹配右括号后面不是括号的内容
        )+
    )*
    (?(Open)(?!))           #在遇到最外层的右括号前面，判断黑板上还有没有没擦掉的"Open"；如果还有，则匹配失败
>                           #最外层的右括号
```

还有些什么东西没提到

我已经描述了构造正则表达式的大量元素，还有一些我没有提到的东西。下面是未提到的元素的列表，包含语法和简单的说明。你可以在网上找到更详细的参考资料 来学习它们--当你需要用到它们的时候。如果你安装了 MSDN Library,你也可以在里面找到关于.net 下正则表达式详细的文档。

表 6.尚未详细讨论的语法

\a	报警字符(打印它的效果是电脑嘀一声)
\b	通常是单词分界位置，但如果在字符类里使用代表退格
\t	制表符，Tab
\r	回车
\v	竖向制表符
\f	换页符
\n	换行符
\e	Escape
\0nn	ASCII代码中八进制代码为nn的字符
\xnn	ASCII代码中十六进制代码为nn的字符
\unnnn	Unicode代码中十六进制代码为nnnn的字符
\cN	ASCII控制字符。比如\cC代表Ctrl+C
\A	字符串开头(类似^，但不受处理多行选项的影响)
\Z	字符串结尾或行尾(不受处理多行选项的影响)
\z	字符串结尾(类似$，但不受处理多行选项的影响)
\G	当前搜索的开头
\p{name}	Unicode中命名为name的字符类，例如\p{IsGreek}
(?>exp)	贪婪子表达式
(?<x>-<y>exp)	平衡组

(?im-nsx:exp)	在子表达式exp中改变处理选项
(?im-nsx)	为表达式后面的部分改变处理选项
(?(exp)yes|no)	把exp当作零宽正向先行断言，如果在这个位置能匹配，使用yes作为此组的表达式；否则使用no
(?(exp)yes)	同上，只是使用空表达式作为no
(?(name)yes|no)	如果命名为name的组捕获到了内容，使用yes作为表达式；否则使用no
(?(name)yes)	同上，只是使用空表达式作为no

一些我认为你可能已经知道的术语的参考

字符

程序处理文字时最基本的单位，可能是字母，数字，标点符号，空格，换行符，汉字等等。

字符串

0 个或更多个字符的序列。

文本

文字，字符串。

匹配

符合规则，检验是否符合规则，符合规则的部分。