正则表达式教程：30分钟让你精通正则表达式语法

来源：C语言编程网

正则表达式教程：30分钟让你精通正则表达式语法

来源：C语言编程网

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没有你 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——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的原理，以后你还需要多练习，多使用，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——你看，我自己也没能把所有的东西记下来，不是吗

？

本教程文本格式约定： 1、专业术语黑色加粗。例如： **专业术语** 2、元字符、语法格式使用蓝色正常字体。例如： 元字符/语法格式 3、正则表达式使用红色正常字体。例如： 正则表达式 4、正则表达式中的一部分使用绿色正常字体。例如： 正则表达式中的一部分(用于分析) 5、匹配的源字符使用黑色斜体。例如： *对其进行匹配的源字符串* 6、对正则表达式的说明使用黑色正常字体+下划线。例如： 对正则表达式或其中一部分的说明 注意：正文右边灰色背景部分有一些注释，主要是用来提供一些相关信息，或者给没有程序员背景的读者解释一些基本概念，通常可以忽略。

什么是正则表达式

字符是计算机软件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。字符串是0个或更多个字符的序列。文本也就是文字，字符串。说某个字符串匹配某个正则表达式，通常是指这个字符串里有一部分（或几部分分别）能满足表达式给出的条件。

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，用来查找 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*)。

正则表达式入门【开始学习】

1、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(一个是,一个不是或不存在) \w 。
2、换行符就是'\n',ASCII编码为10(十六进制0x0A)的字符。

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，你可以使用正则表达式 hi 。

这几乎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匹配数字字母下划线,不单单是单词

\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，标点符号或者换行来分隔的，但是 \b 并不匹配这些单词分隔字符中的任何一个，它**只匹配一个位置**。

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 * 前边的内容可以连续重复出现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 是个新的元字符，匹配 一位数字(0，或1，或2，或……)。 - 不是元字符，只匹配它本身——连字符或者减号。

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

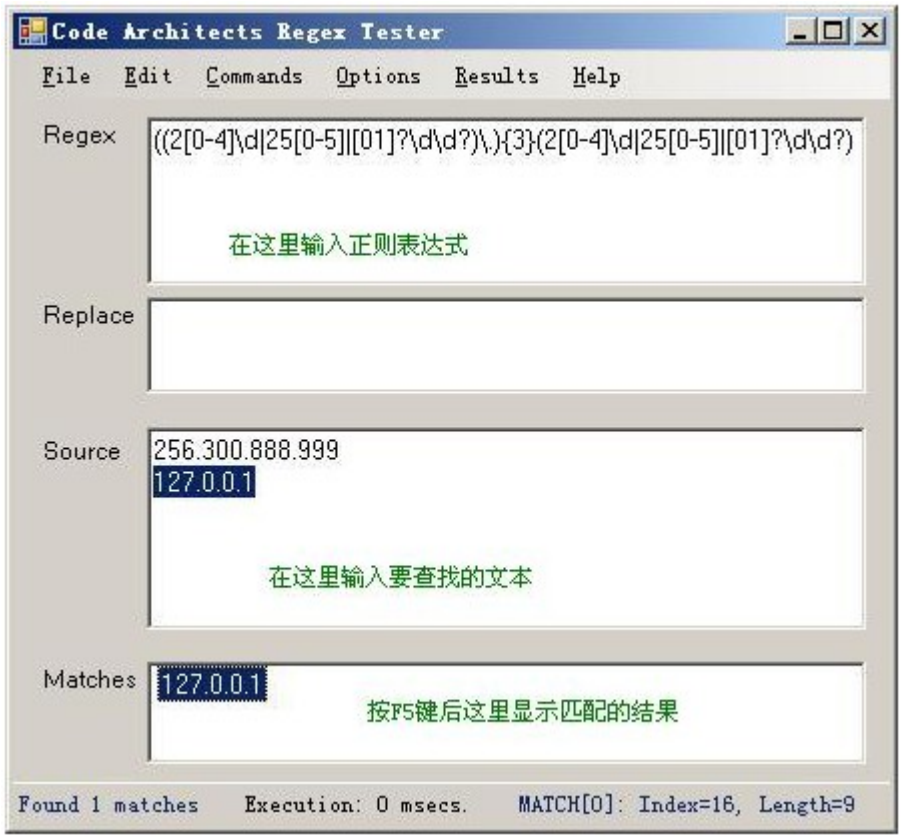
推荐两个在线测试正则表达式的工具:

- RegexBuddy
- Javascript正则表达式在线测试工具

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，所以找一种工具对正则表达式进行测试是很有必要的。

由于在不同的环境下正则表达式的一些细节是不相同的，本教程介绍的是微软 .Net Framework 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。这是个绿色软件，下载完后打开压缩包,直接运行RegexTester.exe就可以了。

下面是Regex Tester运行时的截图：


Code Architects Regex Tester
File Edit Commands Options Results Help
Regex
((2[0-4]\d|25[0-5]|[01]?\d\d?)\.){3}(2[0-4]\d|25[0-5]|[01]?\d\d?)
在这里输入正则表达式
Replace
Source
256.300.888.999
127.0.0.1
在这里输入要查找的文本
Matches
127.0.0.1
按F5键后这里显示匹配的结果
Found 1 matches
Execution: 0 msecs.
MATCH[0]: Index=16, Length=9

正则表达式元字符详解

1、对中文/汉字的特殊处理是由.Net提供的正则表达式引擎支持的，其它环境下的具体情况请查看相关文档。
2、好吧，现在我们说说正则表达式里的单词是什么意思吧：就是多于一个的连续的\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 :)

现在你已经知道几个很有用的元字符了，如\b，.，*，还有\d.正则表达式里还有更多的元字符，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等。

下面来看看更多的例子：

\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)。

\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^（和数字6在同一个键位上的符号）和$都匹配一个位置，这和\b有点类似。^匹配你要用来查找的字符串的开头，$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}则是重复的次数不能少于5次，不能多于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。

正则表达式转义字符详解

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：你没办法指定它们，因为它们会被解释成别的意思。这时你就得使用\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： unibetter\.com 匹配 unibetter.com， C:\\Windows 匹配 C:\Windows。

正则表达式重复匹配详解

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的限定符(指定数量的代码，例如*,{5,12}等)：

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

^\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)

正则表达式字符类详解

“(”和“)”也是元字符，后面的分组节里会提到，所以在这里需要使用转义。

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合的元字符，但是如果你想匹配没有预定义元字符的字符集合(比如元音字母a,e,i,o,u),应该怎么办?

很简单，你只需要在方括号里列出它们就行了，像 [aeiou] 就匹配 任何一个英文元音字母 ， [.?!] 匹配 标点符号(.或?或!) 。

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(\d{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})。

正则表达式分枝条件详解

不幸的是，刚才那个表达式也能匹配*010)12345678*或*(022-87654321*这样的“不正确”的格式。要解决这个问题，我们需要用到**分枝条件**。正则表达式里的**分枝条件**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。你可以试试用分枝条件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：**使用分枝条件时，要注意各个条件的顺序。**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)。原因是匹配分枝条件时，将会从左到右地测试每个条件，如果满足了某个分枝的话，就不会去再管其它的条件了。

正则表达式分组详解

IP地址中每个数字都不能大于255，大家千万不要被《24》第三季的编剧给忽悠了...

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）；但如果想要重复多个字符又该怎么办？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。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。

正则表达式反义详解

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。

正则表达式后向引用详解

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(也就是此分组捕获的内容)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*，或者 *kitty kitty*。这个表达式首先是 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），这个单词会被捕获到编号为1的分组中，然后是 1个或几个空白符（\s+），最后是 分组1中捕获的内容（也就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名。要指定一个子表达式的组名，请使用这样的语法：(?<Word>\w+)(或者把尖括号换成'也行：(?'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.常用分组语法

分类	代码/语法	说明
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，也不给此分组分配组号
零宽断言	(?=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)	这种类型的分组不对正则表达式的处理产生任何影响，用于提供注释让人阅读

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，也不会拥有组号 。

正则表达式零宽断言详解

1、地球人，是不是觉得这些术语名称太复杂，太难记了？我也和你一样。知道有这么一种东西就行了，它叫什么，随它去吧！“无名，万物之始...”
2、断言用来声明一个应该为真的事实。正则表达式中只有当断言为真时才会继续进行匹配。

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，也就是说它们像 \b , ^ , $ 那样用于指定一个位置，这个位置应该满足一定的条件(即断言)，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 **零宽度正预测先行断言** ，它 断言自身出现的位置的后面能匹配表达式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) ，如查找 *I'm singing while you're dancing.* 时，它会匹配 sing 和 danc 。

(?<=exp) 也叫 **零宽度正回顾后发断言** ，它 断言自身出现的位置的前面能匹配表达式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 ，用它对 *1234567890* 进行查找时结果是 234567890 。

下面这个例子同时使用了这两种断言： (?<=\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)。

正则表达式负向零宽断言详解

请详细分析表达式 (?<=<(\w+)>).*(?=<\/\1>)，这个表达式最能表现零宽断言的真正用途。

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] 总要匹配一个字符，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*。 **负向零宽断言** 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 。

零宽度负预测先行断言 (?!exp)， 断言此位置的后面不能匹配表达式exp。例如： \d{3}(?!\d) 匹配 三位数字，而且这三位数字的后面不能是数字； \b((?!abc)\w)+\b 匹配 不包含连续字符串abc的单词。

同理，我们可以用 (?<!exp) ,**零宽度正回顾后发断言** 来 断言此位置的前面不能匹配表达式exp： (?<![a-z])\d{7} 匹配 前面不是小写字母的七位数字。

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)。

正则表达式的注释详解

小括号的另一种用途是通过语法(?#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：

```
      (?<=     # 断言要匹配的文本的前缀
      <(\w+)> # 查找尖括号括起来的字母或数字(即HTML/XML标签)
      )        # 前缀结束
      .*       # 匹配任意文本
      (?=      # 断言要匹配的文本的后缀
      <\/\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
```

正则表达式的贪婪与懒惰

为什么第一个匹配是aab（第一到第三个字符）而不是ab（第二到第三个字符）？简单地说，因为正则表达式有另一条规则，比懒惰 / 贪婪规则的优先级更高：最先开始的匹配拥有最高的优先权——The match that begins earliest wins。

当正则表达式中包含能接受重复的限定符时，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（第一到第三个字符） 和 ab（第四到第五个字符） 。

表5.懒惰限定符

代码/语法	说明
*?	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

正则表达式的处理选项

在C#中，你可以使用Regex(String, RegexOptions)构造函数来设置正则表达式的处理选项。如：Regex regex = new Regex("\ba\w{6}\b", RegexOptions.IgnoreCase);

上面介绍了几个选项如忽略大小写，处理多行等，这些选项能用来改变处理正则表达式的方式。下面是.Net中常用的正则表达式选项：

表6.常用的处理选项

名称	说明
IgnoreCase(忽略大小写)	匹配时不区分大小写。
Multiline(多行模式)	更改^和$的含义，使它们分别在任意一行的行首和行尾匹配，而不仅仅在整个字符串的开头和结尾匹配。(在此模式下,$的精确含意是:匹配\n之前的位置以及字符串结束前的位置.)
Singleline(单行模式)	更改.的含义，使它与每一个字符匹配（包括换行符\n）。
IgnorePatternWhitespace(忽略空白)	忽略表达式中的非转义空白并启用由#标记的注释。
RightToLeft(从右向左查找)	匹配从右向左而不是从左向右进行。
ExplicitCapture(显式捕获)	仅捕获已被显式命名的组。
ECMAScript(JavaScript兼容模式)	使表达式的行为与它在JavaScript里的行为一致。

一个经常被问到的问题是：是不是只能同时使用多行模式和单行模式中的一种？答案是：不是。这两个选项之间没有任何关系，除了它们的名字比较相似（以至于让人感到疑惑）以外。

正则表达式平衡组和递归匹配详解

1、这里介绍的平衡组语法是由.Net Framework支持的；其它语言 / 库不一定支持这种功能，或者支持此功能但需要使用不同的语法。

2、如 果你不是一个程序员（或者你自称程序员但是不知道堆栈是什么东西），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"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。

有时我们需要匹配像(100 * (50 + 15))这样的可嵌套的层次性结构，这时简单地使用\(.+\)则只会匹配到最左边的左括号和最右边的右括号之间的内容(这里我们讨论的是贪婪模式，懒惰模式也有下面的问题)。假如原来的字符串里的左括号和右括号出现的次数不相等，比如*(5 / (3 + 2)))*，那我们的匹配结果里两者的个数也不会相等。有没有办法在这样的字符串里匹配到最长的，配对的括号之间的内容呢?

为了避免(和\(把你的大脑彻底搞糊涂，我们还是用尖括号代替圆括号吧。现在我们的问题变成了如何把 *xx <aa <bbb> <bbb> aa> yy* 这样的字符串里，最长的配对的尖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(Stack)**
- (?'-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

我们需要做的是每碰到了左括号，就在压入一个"Open",每碰到一个右括号，就弹出一个，到了最后就看看堆栈是否为空 - - 如果不为空那就证明左括号比右括号多，那匹配就应该失败。正则表达式引擎会进行回溯(放弃最前面或最后面的一些字符)，尽量使整个表达式得到匹配。

```
< #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
```

平衡组的一个最常见的应用就是匹配HTML,下面这个例子可以匹配嵌套的 标签： <div[^>]*>[^<>]*(((?'Open'<div[^>]*>)[^<>]*)+((?'-Open')[^<>]*)+)*(?(Open)(?!)) .

正则表达式的其他规则

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7.尚未详细讨论的语法

代码/语法	说明
\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

正则表达式所有元字符及其使用方法一览表

字符	描述
\	将下一个字符标记为一个特殊字符、或一个原义字符、或一个 后向引用、或一个八进制转义符。例如，'n' 匹配字符 "n"。'\n' 匹配一个换行符。序列 '\\' 匹配 "\" 而 "\(" 则匹配 "("。
^	匹配输入字符串的开始位置。如果设置了 **RegExp** 对象的 **Multiline** 属性，^ 也匹配 '\n' 或 '\r' 之后的位置。
$	匹配输入字符串的结束位置。如果设置了**RegExp** 对象的 **Multiline** 属性，$ 也匹配 '\n' 或 '\r' 之前的位置。
*	匹配前面的子表达式零次或多次。例如，zo* 能匹配 "z" 以及 "zoo"。 * 等价于{0,}。
+	匹配前面的子表达式一次或多次。例如，'zo+' 能匹配 "zo" 以及 "zoo"，但不能匹配 "z"。+ 等价于 {1,}。
?	匹配前面的子表达式零次或一次。例如，"do(es)?" 可以匹配 "do" 或 "does" 中的"do" 。? 等价于 {0,1}。
{*n*}	*n* 是一个非负整数。匹配确定的 *n* 次。例如，'o{2}' 不能匹配 "Bob" 中的 'o'，但是能匹配 "food" 中的两个 o。

{*n*,}	*n* 是一个非负整数。至少匹配*n* 次。例如，'o{2,}' 不能匹配 "Bob" 中的 'o'，但能匹配 "fooooood" 中的所有 o。'o{1,}' 等价于 'o+'。'o{0,}' 则等价于 'o*'。
{*n*,*m*}	*m* 和 *n* 均为非负整数，其中*n* <= *m*。最少匹配 *n* 次且最多匹配 *m* 次。刘， "o{1,3}" 将匹配 "fooooood" 中的前三个 o。'o{0,1}' 等价于 'o?'。请注意在逗号和两个数之间不能有空格。
?	当该字符紧跟在任何一个其他限制符 (*, +, ?, {*n*}, {*n*,}, {*n*,*m*}) 后面时，匹配模式是非贪婪的。非贪婪模式尽可能少的匹配所搜索的字符串，而默认的贪婪模式则尽可能多的匹配所搜索的字符串。例如，对于字符串 "oooo"，'o+?' 将匹配单个 "o"，而 'o+' 将匹配所有 'o'。
.	匹配除 "\n" 之外的任何单个字符。要匹配包括 '\n' 在内的任何字符，请使用象 '[.\n]' 的模式。
(*pattern*)	匹配*pattern* 并获取这一匹配。所获取的匹配可以从产生的 Matches 集合得到，在VBScript 中使用 **SubMatches** 集合，在Visual Basic Scripting Edition 中则使用 **$0**…**$9** 属性。要匹配圆括号字符，请使用 '\(' 或 '\)'。
(?:*pattern*)	匹配 *pattern* 但不获取匹配结果，也就是说这是一个非获取匹配，不进行存储供以后使用。这在使用 "或" 字符 (|) 来组合一个模式的各个部分是很有用。例

	如， 'industr(?:y|ies) 就是一个比 'industry|industries' 更简略的表达式。
(?=*pattern*)	正向预查，在任何匹配 *pattern* 的字符串开始处匹配查找字符串。这是一个非获取匹配，也就是说，该匹配不需要获取供以后使用。例如， 'Windows (?=95|98|NT|2000)' 能匹配 "Windows 2000" 中的 "Windows" ，但不能匹配 "Windows 3.1" 中的 "Windows"。预查不消耗字符，也就是说，在一个匹配发生后，在最后一次匹配之后立即开始下一次匹配的搜索，而不是从包含预查的字符之后开始。
(?!*pattern*)	负向预查，在任何不匹配Negative lookahead matches the search string at any point where a string not matching *pattern* 的字符串开始处匹配查找字符串。这是一个非获取匹配，也就是说，该匹配不需要获取供以后使用。例如'Windows (?!95|98|NT|2000)' 能匹配 "Windows 3.1" 中的 "Windows"，但不能匹配 "Windows 2000" 中的 "Windows"。预查不消耗字符，也就是说，在一个匹配发生后，在最后一次匹配之后立即开始下一次匹配的搜索，而不是从包含预查的字符之后开始
x|*y*	匹配 *x* 或 *y*。例如，'z|food' 能匹配 "z" 或 "food"。'(z|f)ood' 则匹配 "zood" 或 "food"。
[*xyz*]	字符集合。匹配所包含的任意一个字符。例如， '[abc]' 可以匹配 "plain" 中的 'a'。

[^*xyz*]	负值字符集合。匹配未包含的任意字符。例如，'[^abc]' 可以匹配 "plain" 中的'p'。
[*a-z*]	字符范围。匹配指定范围内的任意字符。例如，'[a-z]' 可以匹配 'a' 到 'z' 范围内的任意小写字母字符。
[^*a-z*]	负值字符范围。匹配任何不在指定范围内的任意字符。例如，'[^a-z]' 可以匹配任何不在 'a' 到 'z' 范围内的任意字符。
\b	匹配一个单词边界，也就是指单词和空格间的位置。例如， 'er\b' 可以匹配"never" 中的 'er'，但不能匹配 "verb" 中的 'er'。
\B	匹配非单词边界。'er\B' 能匹配 "verb" 中的 'er'，但不能匹配 "never" 中的 'er'。
\c*x*	匹配由*x*指明的控制字符。例如， \cM 匹配一个 Control-M 或回车符。 *x* 的值必须为 A-Z 或 a-z 之一。否则，将 c 视为一个原义的 'c' 字符。
\d	匹配一个数字字符。等价于 [0-9]。
\D	匹配一个非数字字符。等价于 [^0-9]。
\f	匹配一个换页符。等价于 \x0c 和 \cL。
\n	匹配一个换行符。等价于 \x0a 和 \cJ。
\r	匹配一个回车符。等价于 \x0d 和 \cM。
\s	匹配任何空白字符，包括空格、制表符、换页符等等。等价于 [\f\n\r\t\v]。

\S	匹配任何非空白字符。等价于 [^ \f\n\r\t\v]。
\t	匹配一个制表符。等价于 \x09 和 \cI。
\v	匹配一个垂直制表符。等价于 \x0b 和 \cK。
\w	匹配包括下划线的任何单词字符。等价于'[A-Za-z0-9_]'。
\W	匹配任何非单词字符。等价于 '[^A-Za-z0-9_]'。
\x*n*	匹配 *n*，其中 *n* 为十六进制转义值。十六进制转义值必须为确定的两个数字长。例如， '\x41' 匹配 "A"。'\x041' 则等价于 '\x04' & "1"。正则表达式中可以使用 ASCII 编码。.
num	匹配 *num*，其中 *num* 是一个正整数。对所获取的匹配的引用。例如，'(.)\1' 匹配两个连续的相同字符。
n	标识一个八进制转义值或一个后向引用。如果 *n* 之前至少 *n* 个获取的子表达式，则 *n* 为后向引用。否则，如果 *n* 为八进制数字 (0-7)，则 *n* 为一个八进制转义值。
nm	标识一个八进制转义值或一个后向引用。如果 *nm* 之前至少有is preceded by at least *nm* 个获取得子表达式，则 *nm* 为后向引用。如果 *nm* 之前至少有 *n* 个获取，则 *n* 为一个后跟文字 *m* 的后向引用。如果前面的条件都不满足，若 *n* 和 *m* 均为八进制数字 (0-7)，则 *nm* 将匹配八进制转义值 *nm*。
nml	如果 *n* 为八进制数字 (0-3)，且 *m* 和 *l* 均为八进制数

	字 (0-7)，则匹配八进制转义值 *nml*。
\u*n*	匹配 *n*，其中 *n* 是一个用四个十六进制数字表示的 Unicode 字符。例如， \u00A9 匹配版权符号 (?)。

null

C语言正则表达式详解 regcomp() regexec() regfree()详解

标准的C和C++都不支持正则表达式，但有一些函数库可以辅助C/C++程序员完成这一功能，其中最著名的当数Philip Hazel的Perl-Compatible Regular Expression库，许多Linux发行版本都带有这个函数库。

C语言处理正则表达式常用的函数有regcomp()、regexec()、regfree()和regerror()，一般分为三个步骤，如下所示：

C语言中使用正则表达式一般分为三步：

1. 编译正则表达式 regcomp()
2. 匹配正则表达式 regexec()
3. 释放正则表达式 regfree()

下边是对三个函数的详细解释

1、int regcomp (regex_t *compiled, const char *pattern, int cflags)
这个函数把指定的正则表达式pattern编译成一种特定的数据格式compiled，这样可以使匹配更有效。函数regexec 会使用这个数据在目标文本串中进行模式匹配。执行成功返回 0 。

参数说明：
①regex_t 是一个结构体数据类型，用来存放编译后的正则表达式，它的成员re_nsub 用来存储正则表达式中的子正则表达式的个数，子

正则表达式就是用圆括号包起来的部分表达式。
②pattern 是指向我们写好的正则表达式的指针。
③cflags 有如下4个值或者是它们或运算(|)后的值：
REG_EXTENDED 以功能更加强大的扩展正则表达式的方式进行匹配。
REG_ICASE 匹配字母时忽略大小写。
REG_NOSUB 不用存储匹配后的结果。
REG_NEWLINE 识别换行符，这样'$'就可以从行尾开始匹配，'^'就可以从行的开头开始匹配。

2. int regexec (regex_t *compiled, char *string, size_t nmatch, regmatch_t matchptr [], int eflags)
当我们编译好正则表达式后，就可以用regexec 匹配我们的目标文本串了，如果在编译正则表达式的时候没有指定cflags的参数为REG_NEWLINE，则默认情况下是忽略换行符的，也就是把整个文本串当作一个字符串处理。执行成功返回 0 。
regmatch_t 是一个结构体数据类型，在regex.h中定义：

```
typedef struct
{
   regoff_t rm_so;
   regoff_t rm_eo;
} regmatch_t;
```

成员rm_so 存放匹配文本串在目标串中的开始位置，rm_eo 存放结束位置。通常我们以数组的形式定义一组这样的结构。因为往往我们的正则表达式中还包含子正则表达式。数组0单元存放主正则表达式位置，后边的单元依次存放子正则表达式位置。

参数说明：
①compiled 是已经用regcomp函数编译好的正则表达式。
②string 是目标文本串。
③nmatch 是regmatch_t结构体数组的长度。
④matchptr regmatch_t类型的结构体数组，存放匹配文本串的位置信息。
⑤eflags 有两个值
REG_NOTBOL 按我的理解是如果指定了这个值，那么'^'就不会从我们的目标串开始匹配。总之我到现在还不是很明白这个参数的意义；
REG_NOTEOL 和上边那个作用差不多，不过这个指定结束end of line。

3. void regfree (regex_t *compiled)
当我们使用完编译好的正则表达式后，或者要重新编译其他正则表达式的时候，我们可以用这个函数清空compiled指向的regex_t结构体的内容，请记住，如果是重新编译的话，一定要先清空regex_t结构体。

4. size_t regerror (int errcode, regex_t *compiled, char *buffer, size_t length)
当执行regcomp 或者regexec 产生错误的时候，就可以调用这个函数而返回一个包含错误信息的字符串。

参数说明：
①errcode 是由regcomp 和 regexec 函数返回的错误代号。
②compiled 是已经用regcomp函数编译好的正则表达式，这个值可以为NULL。
③buffer 指向用来存放错误信息的字符串的内存空间。

④length 指明buffer的长度，如果这个错误信息的长度大于这个值，则regerror 函数会自动截断超出的字符串，但他仍然会返回完整的字符串的长度。所以我们可以用如下的方法先得到错误字符串的长度。

size_t length = regerror (errcode, compiled, NULL, 0);

下边是一个匹配Email例子，按照上面的三步就可以。

```c
#include <stdio.h>
#include <sys/types.h>
#include <regex.h>

int main(int argc, char** argv)
{
    int status, i;
    int cflags = REG_EXTENDED;
    regmatch_t pmatch[1];
    const size_t nmatch = 1;
    regex_t reg;
    const char * pattern = "^\\w+([-+.]\\w+)*@\\w+([-.]\\w+)*.\\w+([-.]\\w+)*$";
    char * buf = "david19842003@gmail.com";

    regcomp(&reg, pattern, cflags);
    status = regexec(&reg, buf, nmatch, pmatch, 0);
    if (status == REG_NOMATCH)
        printf("No Match\n");
    else if(status == 0)
    {
        printf("Match:\n");
        for(i=pmatch[0].rm_so; i<pmatch[0].rm_eo; ++i)
            putchar(buf[i]);
        printf("\n");
    }
    regfree(&reg);
    return 0;
}
```

下面的程序负责从命令行获取正则表达式，然后将其运用于从标准输入得到的每行数据，并打印出匹配结果。

#include <stdio.h>

```
#include <sys/types.h>
#include <regex.h>

/* 取子串的函数 */
static char* substr(const char*str,
unsigned start, unsigned end)
{
  unsigned n = end - start;
  static char stbuf[256];
  strncpy(stbuf, str + start, n);
  stbuf[n] = 0;
  return stbuf;
}

/* 主程序 */
int main(int argc, char** argv)
{
  char * pattern;
  int x, z, lno = 0, cflags = 0;
  char ebuf[128], lbuf[256];
  regex_t reg;
  regmatch_t pm[10];
  const size_t nmatch = 10;
  /* 编译正则表达式*/
  pattern = argv[1];
  z = regcomp(?, pattern, cflags);
  if (z != 0){
```

```
  regerror(z, ?, ebuf, sizeof(ebuf));
  fprintf(stderr, "%s: pattern '%s' \n",ebuf, pattern);
  return 1;
}
/* 逐行处理输入的数据 */
while(fgets(lbuf, sizeof(lbuf), stdin))
{
  ++lno;
  if ((z = strlen(lbuf)) > 0 && lbuf[z-1] == '\n')
  lbuf[z - 1] = 0;
  /* 对每一行应用正则表达式进行匹配 */
  z = regexec(?, lbuf, nmatch, pm, 0);
  if (z == REG_NOMATCH) continue;
  else if (z != 0) {
    regerror(z, ?, ebuf, sizeof(ebuf));
    fprintf(stderr, "%s: regcom('%s')\n", ebuf, lbuf);
    return 2;
  }
  /* 输出处理结果 */
  for (x = 0; x < nmatch && pm[x].rm_so != -1; ++ x)
  {
    if (!x) printf("%04d: %s\n", lno, lbuf);
    printf(" $%d='%s'\n", x, substr(lbuf, pm[x].rm_so, pm[x].rm_eo));
  }
}
/* 释放正则表达式 */
regfree(?);
```

```
  return 0;
}
```

执行下面的命令可以编译并执行该程序：

```
# gcc regexp.c -o regexp
# ./regexp 'regex[a-z]*' < regexp.c
0003: #include <regex.h>
$0='regex'
0027: regex_t reg;
$0='regex'
0054: z = regexec(?, lbuf, nmatch, pm, 0);
$0='regexec'
```

小结：对那些需要进行复杂数据处理的程序来说，正则表达式无疑是一个非常有用的工具。本文重点在于阐述如何在C语言中利用正则表达式来简化字符串处理，以便在数据处理方面能够获得与Perl语言类似的灵活性。

PHP中使用正则表达式详解 preg_match() preg_replace() preg_mat

PHP中嵌入正则表达式常用的函数有四个：

1、preg_match()： preg_match() 函数用于进行正则表达式匹配，成功返回 1，否则返回 0。

语法：int preg_match(string pattern, string subject [, array matches])

参数说明：

参数	说明
pattern	正则表达式
subject	需要匹配检索的对象
matches	可选，存储匹配结果的数组， $matches[0] 将包含与整个模式匹配的文本，$matches[1] 将包含与第一个捕获的括号中的子模式所匹配的文本，以此类推

例子 1：

```
<?php
   if(preg_match("/php/i", "PHP is the web scripting language of choice.", $matches))
   {
```

```
      print "A match was found:". $matches[0];
   }
   else
   {
      print "A match was not found.";
   }
?>
```

浏览器输出：

```
A match was found: PHP
```

在该例子中，由于使用了 i 修正符，因此会不区分大小写去文本中匹配 php 。

提示：preg_match() 第一次匹配成功后就会停止匹配，如果要实现全部结果的匹配，即搜索到subject结尾处，则需使用 preg_match_all() 函数。

例子 2，从一个 URL 中取得主机域名：

```
<?php
   // 从 URL 中取得主机名
   preg_match("/^(http://)?
([^/]+)/i","http://www.5idev.com/index.html", $matches);
   $host = $matches[2]; // 从主机名中取得后面两段
   preg_match("/[^./]+.[^./]+$/", $host, $matches);
   echo "域名为：{$matches[0]}";
?>
```

浏览器输出：

```
域名为：5idev.com
```

2、preg_match_all()：preg_match_all() 函数用于进行正则表达式全局匹配，成功返回整个模式匹配的次数（可能为零），如果出错返回 FALSE 。

语法：int preg_match_all(string pattern, string subject, array matches [, int flags])

参数说明：

参数	说明
pattern	正则表达式
subject	需要匹配检索的对象
matches	存储匹配结果的数组
flags	可选，指定匹配结果放入 matches 中的顺序，可供选择的标记有： 1. PREG_PATTERN_ORDER：默认，对结果排序使 $matches[0] 为全部模式匹配的数组，$matches[1] 为第一个括号中的子模式所匹配的字符串组成的数组，以此类推 2. PREG_SET_ORDER：对结果排序使 $matches[0] 为第一组匹配项的数组，$matches[1] 为第二组匹配项的数组，以此类推 3. PREG_OFFSET_CAPTURE：如果设定本标记，对每个出现的匹配结果也同时返回其附属的字符串

	偏移量

下面的例子演示了将文本中所有 <pre></pre> 标签内的关键字（php）显示为红色。

```
<?php
   $str = "<pre>学习php是一件快乐的事。</pre><pre>所有的phper需要共同努力！</pre>";
   $kw = "php"; preg_match_all('/<pre>([sS]*?)</pre>/',$str,$mat);
   for($i=0;$i<count($mat[0]);$i++)
   {
      $mat[0][$i] = $mat[1][$i];
      $mat[0][$i] = str_replace($kw, '<span style="color:#ff0000">'.$kw.'</span>', $mat[0][$i]);
      $str = str_replace($mat[1][$i], $mat[0][$i], $str);
   }
   echo $str;
?>
```

3、preg_replace()：字符串比对解析并取代。

语法：mixed preg_replace(mixed pattern, mixed replacement, mixed subject);

返回值：混合类型资料

内容说明：本函数以 pattern 的规则来解析比对字符串 subject，欲取而代之的字符串为参数 replacement。返回值为混合类型资料，为取代后的字符串结果。

范例：下例返回值为 $startDate = 6/19/1969

```
$patterns = array("/(19|20\d{2})-(\d{1,2})-(\d{1,2})/", "/^\s*{(\w+)}\s*=/");

$replace = array("\\3/\\4/\\1", "$\\1 =");

print preg_replace($patterns, $replace, "{startDate} = 1969-6-
```

19");

4、preg_split()：将字符串依指定的规则切开。

语法：array preg_split(string pattern, string subject, int [limit]);

返回值：数组

内容说明：本函数可将字符串依指定的规则分开。切开后的返回值为数组变量。参数 pattern 为指定的规则字符串、参数 subject 则为待处理的字符串、参数 limit 可省略，表示欲处理的最多合乎值。

正则表达式匹配汉字或中文

正则匹配中文汉字根据页面编码不同而略有区别：

GBK/GB2312编码：[x80-xff>]+ 或 [xa1-xff]+

UTF-8编码：[x{4e00}-x{9fa5}]+/u

以下以PHP为例进行匹配：

```
<?php
   $str = "学习php是一件快乐的事。";
   preg_match_all("/[x80-xff]+/", $str, $match);
   //UTF-8 使用：
   //preg_match_all("/[x{4e00}-x{9fa5}]+/u", $str, $match);
   print_r($match);
?>
```

输出：

```
Array
(
    [0] => Array
        (
            [0] => 学习
            [1] => 是一件快乐的事。
        )

)
```

正则表达式匹配身份证号码

身份证为15位或者18位，15位的全为数字，18位的前17位为数字，最后一位为数字或者大写字母”X“。

与之匹配的正则表达式：(^\d{15}$)|(^\d{17}([0-9]|X)$)

下面以Javascript为例进行说明：

```
function isIdCardNo(num)
{
  num = num.toUpperCase();
    //身份证号码为15位或者18位，15位时全为数字，18位前17位为数字，最后一位是校验位，可能为数字或字符X。
  if (!(/(^\d{15}$)|(^\d{17}([0-9]|X)$)/.test(num)))
  {
    alert('输入的身份证号长度不对，或者号码不符合规定！\n15位号码应全为数字，18位号码末位可以为数字或X。');
    return false;
  }
```

正则表达式匹配电子邮箱或者电子邮件地址

正则匹配表达式：/^[a-z]([a-z0-9]*[-_]?[a-z0-9]+)*@([a-z0-9]*[-_]?[a-z0-9]+)+[\.][a-z]{2,3}([\.][a-z]{2})?$/i

国际域名格式如下：域名由各国文字的特定字符集、英文字母、数字及“-”(即连字符或减号)任意组合而成, 但开头及结尾均不能含有“-”，“-”不能连续出现 。 域名中字母不分大小写。域名最长可达60个字节(包括后缀.com、.net、.org等)。

说明：
①/内容/i 构成一个不区分大小写的正则表达式；^ 匹配开始；$ 匹配结束。

②[a-z] E-Mail前缀必需是一个英文字母开头

③([a-z0-9]*[-_]?[a-z0-9]+)* 和_a_2、aaa11、_1_a_2匹配，和a1_、aaff_33a_、a__aa不匹配，如果是空字符，也是匹配的，*表示0个或者多个。

④*表示0个或多个前面的字符.

⑤[a-z0-9]* 匹配0个或多个英文字母或者数字；[-_]? 匹配0个或1“-”，因为“-”不能连续出现。

⑥[a-z0-9]+ 匹配1个或多个英文字母或者数字，因为“-”不能做为结尾

⑦@ 必需有个有@

⑧([a-z0-9]*[-_]?[a-z0-9]+)+ 见上面([a-z0-9]*[-_]?[a-z0-9]+)*解释，但是不能为空，+表示一个或者为多个。

⑨[\.] 将特殊字符(.)当成普通字符；[a-z]{2,3} 匹配2个至3个英文字母，一般为com或者net等。
⑩([\.][a-z]{2})? 匹配0个或者1个[\.][a-z]{2}(比如.cn等) 我不知道一般.com.cn最后部份是不是都是两位的,如果不是请修改{2}为{起始字数,结束字数}

下面以PHP为例进行说明：

```
< ?php
if (ereg(“/^[a-z]([a-z0-9]*[-_\.]?[a-z0-9]+)*@([a-z0-9]*[-_]?[a-z0-9]+)+[\.][a-z]{2,3}([\.][a-z]{2})?$/i; ”,$email))
{
   echo “Your email address is correct!”;
}
else
{
   echo “Please try again!”;
}
?>
```

正则表达式匹配用户密码

密码形式可以根据开发需求自己设定，下面是几个匹配密码的例子：

正则表达式	^[a-zA-Z]\w{5,17}$
匹配	以字母开头，长度在6~18之间，只能包含字符、数字和下划线
不匹配	

正则表达式	^(?=.*\d)(?=.*[a-z])(?=.*[A-Z])(?!.*\s).{4,8}$
匹配	1agdA*$# \| 1agdA*$# | 1agdA*$#
不匹配	wyrn%@*&$# f \| mbndkfh782 \| BNfhjdhfjd&*)%#$)

正则表达式	[^A-Za-z0-9]
匹配	!@#$ | %^&* | '><?.,"
不匹配	ABC123abc | abc123ABC | abc0132ABC

正则表达式	Password="(\{.+\}[0-9a-zA-Z]+[=]*|[0-9a-zA-Z]+)"
匹配	!@#$ | %^&* | '><?.,"
不匹配	ABC123abc | abc123ABC | abc0132ABC

正则表达式	(?-i)(?=^.{8,}$)((?!.*\s)(?=.*[A-Z])(?=.*[a-z]))(?=(1)(?=.*\d)\|.*[^A-Za-z0-9])^.*$
匹配	a3dAbed. | P@ssword1 | aB_1bbbb | myPassw0rd!
不匹配	password | password12 | password__12 | p@ssw0rd

正则表达式匹配日期时间

正则表达式	(?n:^(?=\d)((?<day>31(?!(.0?[2469]|11))|30(?!.0?2)|29(?(.0?2)(?=.{3,4}(1[6-9]|[2-9]\d)(0[48]|[2468][048]|[13579][26])|(16|[2468][048]|[3579][26])00))|0?[1-9]|1\d|2[0-8])(?<sep>[/.-])(?<month>0?[1-9]|1[012])\2(?<year>(1[6-9]|[2-9]\d)\d{2})(?:(?=\x20\d)\x20|$))?(?<time>((0?[1-9]\|1[012])(:[0-5]\d){0,2}(?i:\ [AP]M))\|([01]\d\|2[0-3])(:[0-5]\d){1,2})?$)
匹配	31/12/2003 | 29/2/2004 4:50 PM | 23:59:59
不匹配	12/31/2003 | 29/2/2003 | 4:00

正则表达式	^(?:(?:31(\/|-|\.)(?:0?[13578]|1[02]))\1|(?:(?:29|30)(\/|-|\.)(?:0?[1,3-9]|1[0-2])\2))(?:(?:1[6-9]|[2-9]\d)?\d{2})$\|^(?:29(\/\|-\|\.)0?2\3(?:(?:(?:1[6-9]\|[2-9]\d)?(?:0[48]\|[2468][048]\|[13579][26])\|(?:(?:16\|[2468][048]\|[3579][26])00))))$|^(?:0?[1-9]|1\d|2[0-8])(\/|-|\.)(?:(?:0?[1-9])|(?:1[0-2]))\4(?:(?:1[6-9]|[2-9]\d)?\d{2})$
匹配	29/02/1972 | 5-9-98 | 10-11-2002
不匹配	29/02/2003 | 12/13/2002 | 1-1-1500

正则表达式	^(?:(((Jan(uary)?|Ma(r(ch)?|y)|Jul(y)?|Aug(ust)?|Oct(ober)?|Dec(ember)?)\ 31)|((Jan(uary)?|Ma(r(ch)?|y)|Apr(il)?|Ju((ly?)|(ne?))|Aug(ust)?|Oct(ober)?|(Sept|Nov|Dec)(ember)?)\ (0?[1-9]|([12]\d)|30))|(Feb(ruary)?\ (0?[1-9]|1\d|2[0-8]|(29(?=,\ ((1[6-9]|[2-9]\d)(0[48]|[2468][048]|[13579][26])|((16|[2468][048]|[3579][26])00)))))))\,\ ((1[6-9]|[2-9]\d)\d{2}))
匹配	Jan 1, 2003 | February 29, 2004 | November 02, 3202
不匹配	Feb 29, 2003 | Apr 31, 1978 | jan 33,3333

正则表达式	^(([0-1]?[0-9])|([2][0-3])):([0-5]?[0-9])(:([0-5]?[0-9]))?$
匹配	12:15 | 10:26:59 | 22:01:15
不匹配	24:10:25 | 13:2:60

正则表达式	(?=\d)^(?:(?!(?:10\D(?:0?[5-9]|1[0-4])\D(?:1582))|(?:0?9\D(?:0?[3-9]|1[0-3])\D(?:1752)))((?:0?[13578]|1[02])|(?:0?[469]|11)(?!\/31)(?!-31)(?!\.31)|(?:0?2(?=.?(?:(?:29.(?!000[04]|(?:(?:1[^0-6]|[2468][^048]|[3579][^26])00))(?:(?:(?:\d\d)(?:[02468][048]|

	[13579][26])(?!\x20BC))| (?:00(?:42|3[0369]|2[147]|1[258]|09)\x20BC))))))| (?:0?2(?=.(?:(?:\d\D)|(?:[01]\d)|(?:2[0-8])))))([-.\/])(0? [1-9]|[12]\d|3[01])\2(?!0000)((?=(?:00(?:4[0-5]|[0-3]? \d)\x20BC)|(?:\d{4}(?!\x20BC)))\d{4}(?:\x20BC)?) (?:$\|(?=\x20\d)\x20))?((?:(?:0?[1-9]\|1[012])(?::[0-5]\d) {0,2}(?:\x20[aApP][mM]))\|(?:[01]\d\|2[0-3])(?::[0-5]\d) {1,2})?$
匹配	11/24/0004 11:59 PM | 2.29.2008 | 02:50:10
不匹配	12/33/1020 | 2/29/2005 | 13:00 AM

正则表达式	^([0]?[1-9]|[1|2][0-9]|[3][0|1])[./-]([0]?[1-9]|[1][0-2])[./-] ([0-9]{4}|[0-9]{2})$
匹配	10/03/1979 | 1-1-02 | 01.1.2003
不匹配	10/03/197 | 09--02--2004 | 01 02 03

正则表达式	^((0?[13578]|10|12)(-|\/)(([1-9])|(0[1-9])|([12])([0-9]?)| (3[01]?))(-|\/)((19)([2-9])(\d{1})|(20)([01])(\d{1})| ([8901])(\d{1}))|(0?[2469]|11)(-|\/)(([1-9])|(0[1-9])| ([12])([0-9]?)|(3[0]?))(-|\/)((19)([2-9])(\d{1})|(20)([01]) (\d{1})|([8901])(\d{1})))$

匹配	1/2/03 | 02/30/1999 | 3/04/00
不匹配	3/4/2020 | 3/4/1919 | 4/31/2000

正则表达式	^([2-9]\d{3}((0[1-9]|1[012])(0[1-9]|1\d|2[0-8])|(0[13456789]|1[012])(29|30)|(0[13578]|1[02])31)|(([2-9]\d)(0[48]|[2468][048]|[13579][26])|(([2468][048]|[3579][26])00))0229)$
匹配	20000101 | 20051231 | 20040229
不匹配	19990101 | 20053112 | 20050229

正则表达式	^([1-9]|1[0-2]|0[1-9]){1}(:[0-5][0-9][aApP][mM]){1}$
匹配	08:00AM | 10:00am | 7:00pm
不匹配	13:00pm | 12:65am

正则表达式	^(([1-9])|(0[1-9])|(1[0-2]))\/(([0-9])|([0-2][0-9])|(3[0-1]))\/(([0-9][0-9])|([1-2][0,9][0-9][0-9]))$
匹配	01/01/2001 | 1/1/1999 | 10/20/2080
不匹配	13/01/2001 | 1/1/1800 | 10/32/2080

正则表达式	^\d{1,2}\/\d{1,2}\/\d{4}$
匹配	4/1/2001 | 12/12/2001 | 55/5/3434
不匹配	1/1/01 | 12 Jan 01 | 1-1-2001

正则表达式	(\d{2}|\d{4})(?:\-)?([0]{1}\d{1}|[1]{1}[0-2]{1})(?:\-)?([0-2]{1}\d{1}|[3]{1}[0-1]{1})(?:\s)?([0-1]{1}\d{1}|[2]{1}[0-3]{1})(?::)?([0-5]{1}\d{1})(?::)?([0-5]{1}\d{1})
匹配	00-00-00 00:00:00 | 0000-00-00 00:00:00 | 09-05-22 08:16:00 | 1970-00-00 00:00:00 | 20090522081600
不匹配	2009-13:01 00:00:00 | 2009-12-32 00:00:00 | 2002-12-31 24:00:00 | 2002-12-31 23:60:00 | 02-12-31 23:00:60

正则表达式	^(?=\d)(?:(?!(?:1582(?:\.|-|\/)10(?:\.|-|\/)(?:0?[5-9]|1[0-4]))|(?:1752(?:\.|-|\/)0?9(?:\.|-|\/)(?:0?[3-9]|1[0-3])))(?=(?:(?!000[04]|(?:(?:1[^0-6]|[2468][^048]|[3579][^26])00))(?:(?:\d\d)(?:[02468][048]|[13579][26]))\D0?2\D29)|(?:\d{4}\D(?!(?:0?[2469]|11)\D31)(?!0?2(?:\.|-

	|\/)(?:29|30))))(\d{4})([-\/.])(0?\d|1[012])\2((?!00)[012]?\d|3[01])(?:$\|(?=\x20\d)\x20))?((?:(?:0?[1-9]\|1[012])(?::[0-5]\d){0,2}(?:\x20[aApP][mM]))\|(?:[01]\d\|2[0-3])(?::[0-5]\d){1,2})?$
匹配	0008-02-29 | 2:34:59 PM | 9999/12/31 11:59 PM
不匹配	04/04/04 | 1:00 | 1999/1/32

正则表达式	^(([1-9]{1})|([0-1][0-9])|([1-2][0-3])):([0-5][0-9])$
匹配	00:00 | 23:59 | 10:10
不匹配	24:00 | 00:60 | 25:61

正则表达式	^[0-9]{4}-(((0[13578]|(10|12))-(0[1-9]|[1-2][0-9]|3[0-1]))|(02-(0[1-9]|[1-2][0-9]))|((0[469]|11)-(0[1-9]|[1-2][0-9]|30)))$
匹配	2004-04-30 | 2004-02-29
不匹配	2004-04-31 | 2004-02-30

正则表达式匹配URL或者网址

正则表达式	(http|ftp|https):\/\/[\w\-_]+(\.[\w\-_]+)+([\w\-\.,@?^=%&:/~\+#]*[\w\-\@?^=%&/~\+#])?
匹配	http://regxlib.com/Default.aspx | http://electronics.cnet.com/electronics/0-6342366-8-8994967-1.html
不匹配	www.yahoo.com

正则表达式	^\\{2}[\w-]+\\(([\w-][\w-\s]*[\w-]+[$$]?$)\|([\w-][$$]?$))
匹配	\\server\service | \\server\my service | \\serv_001\service$
不匹配	\\my server\service | \\server\ service | \\server$\service

正则表达式	^(http|https|ftp)\://([a-zA-Z0-9\.\-]+(\:[a-zA-Z0-9\.&%\$\-]+)*@)?((25[0-5]|2[0-4][0-9]|[0-1]{1}[0-9]{2}|[1-9]{1}[0-9]{1}|[1-9])\.(25[0-5]|2[0-4][0-9]|[0-1]{1}[0-9]{2}|[1-9]{1}[0-9]{1}|[1-9]|0)\.(25[0-5]|2[0-4][0-9]|[0-1]{1}[0-9]{2}|[1-9]{1}[0-9]{1}|[1-9]|0)\.(25[0-5]|2[0-4][0-9]|[0-1]{1}[0-9]{2}|[1-9]{1}[0-9]{1}|[0-9])|([a-zA-Z0-9\-]+\.)*[a-zA-Z0-9\-]+\.[a-zA-Z]{2,4})(\:[0-9]+)?(/[^/][a-zA-Z0-9\.\,\?\'\\/\+&%\$#\=~_\-@]*)*$
匹配	http://www.sysrage.net | https://64.81.85.161/site/file.php?cow=moo's |ftp://user:pass@host.com:123
不匹配	sysrage.net

正则表达式	^([a-zA-Z]\:|\\\\[^\/\\:*?"<>|]+\\[^\/\\:*?"<>|]+)(\\[^\/\\:*?"<>|]+)+(\.[^\/\\:*?"<>|]+)$
匹配	c:\Test.txt | \\server\shared\Test.txt | \\server\shared\Test.t
不匹配	c:\Test | \\server\shared | \\server\shared\Test.?

正则表达式	^(http|https|ftp)\://([a-zA-Z0-9\.\-]+(\:[a-zA-Z0-9\.&%\$\-]+)*@)*((25[0-5]|2[0-4][0-9]|[0-1]{1}[0-9]{2}|[1-9]{1}[0-9]{1}|[1-9])\.(25[0-5]|2[0-4][0-9]|[0-1]{1}[0-9]{2}|[1-9]{1}[0-9]{1}|[1-9]|0)\.(25[0-5]|2[0-4][0-9]|[0-1]{1}[0-9]{2}|[1-9]{1}[0-9]{1}|[1-9]|0)\.(25[0-5]|2[0-4][0-9]|[0-1]{1}[0-9]{2}|[1-9]{1}[0-9]{1}|[0-9])|localhost|([a-zA-Z0-9\-]+\.)*[a-zA-Z0-9\-]+\.(com|edu|gov|int|mil|net|org|biz|arpa|info|name|pro|aero|coop|museum|[a-zA-Z]{2}))(\:[0-9]+)*(/($\|[a-zA-Z0-9\.\,\?\'\\\+&%\$#\=~_\-]+))*$
匹配	http://site.com/dir/file.php?var=moo | https://localhost |ftp://user:pass@site.com:21/file/dir
不匹配	site.com | http://site.com/dir//

正则表达式	^([a-zA-Z]\:)(\\[^\\/:*?<>"|]*(?<![ ]))*(\.[a-zA-Z]{2,6})$
匹配	C:\di___r\fi_sysle.txt | c:\dir\filename.txt
不匹配	c:\dir\file?name.txt

正则表达式	^([a-zA-Z0-9]([a-zA-Z0-9\-]{0,61}[a-zA-Z0-9])?\.)+[a-zA-Z]{2,6}$
匹配	regexlib.com | this.is.a.museum | 3com.com
不匹配	notadomain-.com | helloworld.c | .oops.org

正则表达式	^(((ht|f)tp(s?))\://)?(www.|[a-zA-Z].)[a-zA-Z0-9\-\.]+\.(com|edu|gov|mil|net|org|biz|info|name|museum|us|ca|uk)(\:[0-9]+)*(/($\|[a-zA-Z0-9\.\,\;\?\'\\\+&%\$#\=~_\-]+))*$
匹配	www.blah.com:8103 | www.blah.com/blah.asp?sort=ASC |www.blah.com/blah.htm#blah
不匹配	www.state.ga | http://www.blah.ru

正则表达式	\b(([\w-]+://?|www[.])[^\s()<>]+(?:\([\w\d]+\)|([^[:punct:]\s]|/)))
匹配	http://foo.com/blah_blah | http://foo.com/blah_blah/ | (Something like http://foo.com/blah_blah) | htt http://foo.com/blah_blah_(wikipedia)) | http://foo.com/blah_blah. |http://foo.com/blah_blah/. | <http: p=364. | http://?df.ws/123 | rdar://1234 | rdar:/1234 | http://userid:password@example.com:8080 |h
不匹配	no_ws.example.com | no_proto_or_ws.com | /relative_resource.php

正则表达式匹配电话号码和手机号码

正则表达式	\d{3}-\d{8}|\d{4}-\d{7}
匹配	0511-4405222 | 021-87888822
不匹配	02-552255 12345-784787

正则表达式	(^\([0]\d{2}|\d{4}\))(\d{6,7}$)
匹配	(021)1234567 | (0411)123456 | (000)000000
不匹配	(123)1234567 | 025123456 | 0252345678

正则表达式	^(?<national>\+?(?:86)?)(?<separator>\s?-?)(?<phone>(?<vender>(13|15|18)[0-9])(?<area>\d{4})(?<id>\d{4}))$
匹配	手机号 +8613012345678 | 86 13012345678 | 13245679087
不匹配	+86130123456781231434352 | 13560012513 | ++8613012345678

正则表达式匹配IP地址

正则表达式	^(25[0-5]|2[0-4][0-9]|[0-1]{1}[0-9]{2}|[1-9]{1}[0-9]{1}|[1-9])\.(25[0-5]|2[0-4][0-9]|[0-1]{1}[0-9]{2}|[1-9]{1}[0-9]{1}|[1-9]|0)\.(25[0-5]|2[0-4][0-9]|[0-1]{1}[0-9]{2}|[1-9]{1}[0-9]{1}|[1-9]|0)\.(25[0-5]|2[0-4][0-9]|[0-1]{1}[0-9]{2}|[1-9]{1}[0-9]{1}|[0-9])$
匹配	127.0.0.1 | 255.255.255.0 | 192.168.0.1
不匹配	1200.5.4.3 | abc.def.ghi.jkl | 255.foo.bar.1

正则表达式	^((0|1[0-9]{0,2}|2[0-9]{0,1}|2[0-4][0-9]|25[0-5]|[3-9][0-9]{0,1})\.){3}(0|1[0-9]{0,2}|2[0-9]{0,1}|2[0-4][0-9]|25[0-5]|[3-9][0-9]{0,1})(?(\/)\/([0-9]|[1-2][0-9]|3[0-2])|)$
匹配	192.168.0.1 | 192.168.0.1/32 | 255.255.0.0/1
不匹配	010.0.0.0 | 192.168.0.1/33 | 256.0.1.55

正则表达式	^(25[0-5]|2[0-4][0-9]|1[0-9][0-9]|[0-9]{1,2})(\.(25[0-5]|2[0-4][0-9]|1[0-9][0-9]|[0-9]{1,2})){3}$
匹配	97.67.44.20 | 199.154.37.214 | 127.0.0.1
不匹配	63.125.94.287 | 140.370.a.187 | 94.923.1

正则表达式	/^(([01]?\d?\d|2[0-4]\d|25[0-5])\.){3}([01]?\d?\d|2[0-4]\d|25[0-5])\/(\d{1}|[0-2]{1}\d{1}|3[0-2])$/
匹配	192.168.100.1/24 | 0.0.0.0/0

不匹配	192.168.100.1/33 | 0.0.0.0/90

正则表达式	\d+\.\d+\.\d+\.\d+
匹配	127.0.0.1 | 255.255.255.0 | 192.168.0.1
不匹配	@#.5.4.3 | abc.def.ghi.jkl | 255.foo.bar.1

正则表达式	^((\d|\d\d|[0-1]\d\d|2[0-4]\d|25[0-5])\.(\d|\d\d|[0-1]\d\d|2[0-4]\d|25[0-5])\.(\d|\d\d|[0-1]\d\d|2[0-4]\d|25[0-5])\.(\d|\d\d|[0-1]\d\d|2[0-4]\d|25[0-5]))$
匹配	1.198.0.1 | 100.10.0.1 | 200.200.123.123
不匹配	..12.23 | a.23.345 | 400.500.300.300

正则表达式	^(\d{1,2}|1\d\d|2[0-4]\d|25[0-5])\.(\d{1,2}|1\d\d|2[0-4]\d|25[0-5])\.(\d{1,2}|1\d\d|2[0-4]\d|25[0-5])\.(\d{1,2}|1\d\d|2[0-4]\d|25[0-5])$
匹配	0.0.0.0 | 255.255.255.02 | 192.168.0.136
不匹配	256.1.3.4 | 023.44.33.22 | 10.57.98.23.

正则表	^(http|https|ftp)\://(((25[0-5]|2[0-4][0-9]|1[0-9][0-9]|[1-9][0-9]|[0-9])\.){3}(25[0-5]|2[0-4][0-9]|1[0-9][0-9]|[1-9][0-9]|[0-9])|([a-zA-Z0-9_\-\.])+\.(com|net|org|edu|int|mil|gov|arpa|biz|aero|name|coop|info|pro|museum|uk|me))((:[a-zA-Z0-9]*)?/?([a-zA-Z0-9\-\._\?\,\'/\\\+&%\$#\=~])*)$

达式	
匹配	http://www.allkins.com | http://255.255.255.255 | http://allkins.com/page.asp?action=1
不匹配	http://test.testing

正则表达式	^(([0-2]*[0-9]+[0-9]+)\.([0-2]*[0-9]+[0-9]+)\.([0-2]*[0-9]+[0-9]+)\.([0-2]*[0-9]+[0-9]+))$
匹配	113.173.40.255 | 171.132.248.57 | 79.93.28.178
不匹配	189.57.135 | 14.190.193999 | A.N.D.233

正则表达式	\b(([01]?\d?\d|2[0-4]\d|25[0-5])\.){3}([01]?\d?\d|2[0-4]\d|25[0-5])\b
匹配	217.6.9.89 | 0.0.0.0 | 255.255.255.255
不匹配	256.0.0.0 | 0978.3.3.3 | 65.4t.54.3

正则表达式匹配字母 大写字母 小写字母 大小写字母 数字和字母组

匹配特定字符串：

只能输入长度为3的字符："^.{3}$"。

只能输入由26个英文字母组成的字符串："^[A-Za-z]+$"。

只能输入由26个大写英文字母组成的字符串："^[A-Z]+$"。

只能输入由26个小写英文字母组成的字符串："^[a-z]+$"。

只能输入由数字和26个英文字母组成的字符串："^[A-Za-z0-9]+$"。

只能输入由数字、26个英文字母或者下划线组成的字符串："^\w+$"。

正则表达式匹配HTML标签或标记

正则表达式	<(\S*?) [^>]*>.*?</\1>|<.*? />
匹配	<html>hello</html>|<a>abcd
不匹配	abc|123|<html>ddd

正则表达式	^[^<>`~!/@\#}$%:;)(_^{&*=|'+]+$
匹配	This is a test
不匹配	<href =	 	That's it

正则表达式	<!--.*?-->
匹配	<!-- this text has been removed -->	<!-- yada -->
不匹配	this text has not been removed

正则表达式	(\[(\w+)\s*(([\w]*)=('|")?([a-zA-Z0-9|:|\/|=|-|.|\?|&]*)(\5)?)*\])([a-zA-Z0-9|:|\/|=|-|.|\?|&|\s]+)(\[\/\2\])
匹配	[link url="http://www.domain.com/file.extension?getvar=value&secondvar=value"]Link[/li
不匹配	[a]whatever[/b]	[a var1=something var2=somethingelse]whatever[/a]	[a]whatever[a]

正则表达式	href=[\"\'](http:\/\/|\.\/|\/)?\w+(\.\w+)*(\/\w+(\.\w+)?)*(\/|\?\w*=\w*(&\w*=\w*)*)?[\"\']
匹配	href="www.yahoo.com"	href="http://localhost/blah/"	href="eek"
不匹配	href=""	href=eek	href="bad example"

正则表达式	"([^"](?:\\.|[^\\"]*)*)"
匹配	"This is a \"string\"."
不匹配	"This is a \"string\".

正则表达式	(?i:on(blur|c(hange|lick)|dblclick|focus|keypress|(key|mouse)(down|up)|(un)?load|mouse(move|o(ut|ver))|reset|s(elect|ubmit)))
匹配	onclick | onsubmit | onmouseover
不匹配	click | onandon | mickeymouse

正则表达式	(?s)/*.**/
匹配	/* ..... */ | /* imagine lots of lines here */
不匹配	*/ malformed opening tag */ | /* malformed closing tag /*

正则表达式	<(\S*?) [^>]*>.*?</\1>|<.*? />
匹配	<html>hello</html>|<a>abcd
不匹配	abc|123|<html>ddd

正则表达式	\xA9
匹配	©
不匹配	anything

正	src[^>]*[^/].(?:jpg|bmp|gif)(?:\"|\')

则表达式	
匹配	src="../images/image.jpg" | src="http://domain.com/images/image.jpg" | src=
不匹配	src="../images/image.tif" | src="cid:value"

正则表达式	/*[\d\D]*?*/
匹配	/* my comment */ | /* my multiline comment */ | /* my nested comment */
不匹配	*/ anything here /* | anything between 2 seperate comments | * *\

正则表达式	<[a-zA-Z]+(\s+[a-zA-Z]+\s*=\s*("([^"]*)"|'([^']*'))*\s*/>
匹配	
不匹配	<img src="test.gif"> |

常用正则表达式大全(匹配空格、替换等)

完整版Word文档下载：http://115.com/file/e74xgria

部分匹配规则预览：
中国电话号码验证
匹配形式如:0511-4405222 或者021-87888822 或者 021-44055520-555 或者 (0511)4405222
正则表达式 "((d{3,4})|d{3,4}-)?d{7,8}(-d{3})*"

中国邮政编码验证
匹配形式如:215421
正则表达式 "d{6}"

电子邮件验证
匹配形式如:justali@justdn.com
正则表达式 "w+([-+.]w+)*@w+([-.]w+)*.w+([-.]w+)*"

身份证验证
匹配形式如:15位或者18位身份证
正则表达式 "d{18}|d{15}"

常用数字验证
正则表达式
"d{n}" n为规定长度
"d{n,m}" n到m的长度范围

非法字符验证
匹配非法字符如:< > & / ' |
正则表达式 [^<>&/|'\]+

日期验证
匹配形式如:20030718,030718
范围:1900--2099
正则表达式((((19){1}|(20){1})d{2})|d{2})[01]{1}d{1}[0-3]{1}d{1}
正则表达式是一个好东西，但是一般情况下，我们需要验证的内容少之又少。