# [C#正则表达式语法规则详解](http://www.cnblogs.com/greatverve/archive/2011/06/27/csharp-reg.html)

正则表达式通常包含字母文本（Literaltext）和元字符（metacharacter）   
字母文本指的是普通文本如"abcde"可匹配字符串中任何包含"abcde"的字符串。

# 元字符则更加灵活运用通用的表达式匹配所有符合此表达式规律的字符串。 C#正则表达式语法一、

匹配单个字符 []——从中选择一个字符匹配

中间支持的类型：单词字符（[ae]）、非单词字符（[!?,;@#$\*]）、字母范围（[A-Z]）、数字范围（[0]）

eg.正则表达式[ae]ffect  
可匹配字符串 affect,effect

（此例中"[ae]"为元字符，"ffect"为字母文本）

注意：  
1.要在字符类中匹配连字符，那么把连字符号作为第一个字符列出即可。

2.可以在单个正则表达式中包含多个字符类。

eg.[01][0-9]:[0-5][0-9][ap]m可以用来匹配如12:59pm格式的所有时间

^——排除某些字符(在[]中表此意，还可表示字符串的开头)

eg.正则表达式m[^a]t  
可匹配字符串  
不可匹配字符串 met,mit,m&t……mat

C#正则表达式语法二、  
匹配特殊字符 可以使用的特殊字符：

\t——匹配制表符 \r——匹配硬回车符 \f——匹配换页符 \n——匹配换行符 描述表示字符类的元字符：

.——匹配任何除了\n以外的字符（或者在单行模式中的任何字符） \w——匹配任何单词字符（任何字母或数字）   
\W——匹配任何非单词字符（除了字母和数字以外的任何字符） \s——匹配任何空白字符（包括空格、换行、制表符等）   
\S——匹配任何非空白字符（除了空格、换行、制表符等的任何字符） \d——匹配任何数字字符（0～9的数字）   
\D——匹配任何非数字字符（除了0～9以外的任何字符） 表示字符串中字符位置： ^——匹配字符串的开头（或者多行模式下行的开头）。   
$——匹配字符串的结尾，或者是字符串结尾“\n”之前的最后一个字符，或者是多行模式中的行结尾。   
\A——匹配字符串的开头（忽略多行模式） \Z——匹配字符串的结尾或字符串结尾“\n”之前的最后一个字符（忽略多行模式）。   
\z——匹配字符串的结尾。 \G——匹配当前搜索开始的位置。 \b——匹配单词的边界。 \B——匹配单词的非边界。   
注意：

1.句点字符（.）特别有用。可以用它来表示任何一个字符。

eg.正则表达式01.17.84  
可匹配字符串 01/17/84,01-17-84,011784,01.17.84

2.可以使用\b匹配单词的边界

eg.正则表达式  
可匹配字符串 \blet\blet  
不可匹配字符串letter,hamlet

3.\A和\z在确保字符串所包含的是某个表达式，而不是其他内容时很用。

eg.要判断Text控件是否包含单词"sophia"，而不含任何额外的字符、换行符或者空白。

\Asophia\z

4.句点字符(.)具有特殊的含义，若要表示字母字符本身的含义，在前面加一个反斜杠：\.

C#正则表达式语法三、  
匹配二选一的字符序列

|——匹配二选一

eg.正则表达式col(o|ou)r  
可匹配字符串 color,colour

注意：\b(bill|ted)和\bbill|ted是不同的。

后者还可以匹配"malted"因为\b元字符只应用于"bill"。

C#正则表达式语法四、  
用量词匹配 \*——匹配0次或多次 +——匹配1次或多次 ?——匹配0次或1次 {n}——恰好匹配n次 {n,}——至少匹配n次 {n,m}——至少匹配n次，  
但不多于m次   
eg.正则表达式brothers?  
可匹配字符串 brother,brothers

eg.正则表达式\bp\d{3,5}  
可匹配字符串 \b以p开头，且后跟3～5个数字结尾

注意：也可以把量词与（）一起使用，以便把该量词应用到整个字母序列。

eg.正则表达式(The)？schoolisbeautiful.  
可匹配字符串 schoolisbeautiful,Theschoolisbeautiful.

C#正则表达式语法五、  
识别正则表达式和贪婪 有些量词是贪婪的（greedy）.他们会尽可能多的匹配字符。

如量词\*匹配0个或多个字符。假设要匹配字符串中任何HTML标签。你可能会用如下正则表达式：

<.\*>

现有字符串A<i>quantifier</i>canbe<big>greedy</big>

结果<.\*>把<i>quantifier</i>canbe<big>greedy</big>都匹配上了。

要解决该问题，需要与量词一起使用一个特殊的非贪婪字符“？”，因此表达式变化如下：

<.\*？>

这样就可以正确匹配<i>、</i>、<big>、</big>。

?能强制量词尽可能少地匹配字符，？还可以用在以下几个量词中：

\*?——非贪婪的量词\* +?——非贪婪的量词+ ??——非贪婪的量词? {n}?——非贪婪的量词{n} {n,}?——非贪婪的量词  
{n,} {n,m}?——非贪婪的量词{n,m}   
C#正则表达式语法六、  
捕获和反向引用 捕获组（capturegroup）就像是正则表达式中的变量。  
捕获组可以捕获正则表达式中的字符模式，并且由正则表达式后面的编号或名称来引用改模式。

()——用来捕获其中的字符串

\数字——用编号来引用

eg.

正则表达式 (\w)(\w)\2\1  
可匹配字符串abba

注意：  
1.反向引用用来匹配html标签非常有效如<(\w+)></\1>可以匹配<table></table>等类似格式的标签。

2.默认情况下，只要使用圆括号，就会捕获圆括号内所包含的字符，可以使用n选项来禁用这个默认行为（在第7条里会详细介绍），  
或者添加？：到圆括号中。eg.(?:sophia)或(?n:sophia)此时不会捕获sophia。

(?<捕获组名称>)\k<捕获组名称>——用名称来引用

eg.

正则表达式(？<sophia>\w)abc\k<sophia>  
可匹配字符串 xabcx

注意：在替换模式中使用捕获组的格式略有不同，要用$1、$2等来按数值引用捕获，用${sophia}等名称来按名称引用捕获组

C#正则表达式语法七、  
设置正则表达式的选项

eg.

stringstr="<h4>sophia</h4>"

RegExobjRegEx=newRegEx("<h(d)>(.\*?)</h1>");   
Response.Write(objRegEx.Replace(str,"<fontsize=$1>$2</font>"));

i——所执行的匹配是不区分大小写的（.net中的属性为IgnoreCase） m——指定多行模式（.net中的属性为Multiline）   
n——只捕获显示命名或编号的组（.net中的属性为ExplicitCapture） c——编译正则表达式，这样会产生较快的执行速度，但启动会变慢（.net中的属性为Compiled）   
s——指定单行模式（.net中的属性为SingleLine） x——消除非转义空白字符和注释（.net中的属性为IgnorePatternWhitespace）   
r——搜索从右到左进行（.net中的属性为RightToLeft） -——表示禁用。   
eg.(?im-r:sophia)允许不区分大小写匹配sophia，使用多行模式，但禁用了从右到左的匹配。

注意：  
1.m会影响如何解析起始元字符（^）和结束元字符（$）。  
在默认情况^和$只匹配整个字符串的开头，即使字符串包含多行文本。如果启用了m,那么它们就可以匹配每行文本的开头和结尾。

2.s会影响如何解析句点元字符（.）。通常一个句点能匹配除了换行符以外的所有字符。但在单行模式下，句点也能匹配一个换行符。

**from:**[**http://greatverve.cnblogs.com/archive/2011/06/27/csharp-reg.html**](http://greatverve.cnblogs.com/archive/2011/06/27/csharp-reg.html) **文库：**

# 

# 

# 常用的C#正则表达式！

# "^\d+$" //非负整数（正整数 + 0）  "^[0-9]\*[1-9][0-9]\*$" //正整数  "^((-\d+)|(0+))$" //非正整数（负整数 + 0）  "^-[0-9]\*[1-9][0-9]\*$" //负整数  "^-?\d+$" //整数  "^\d+(\.\d+)?$" //非负浮点数（正浮点数 + 0）  "^(([0-9]+\.[0-9]\*[1-9][0-9]\*)|([0-9]\*[1-9][0-9]\*\.[0-9]+)|([0-9]\*[1-9][0-9]\*))$" //正浮点数  "^((-\d+(\.\d+)?)|(0+(\.0+)?))$" //非正浮点数（负浮点数 + 0）  "^(-(([0-9]+\.[0-9]\*[1-9][0-9]\*)|([0-9]\*[1-9][0-9]\*\.[0-9]+)|([0-9]\*[1-9][0-9]\*)))$" //负浮点数  "^(-?\d+)(\.\d+)?$" //浮点数  "^[A-Za-z]+$" //由26个英文字母组成的字符串  "^[A-Z]+$" //由26个英文字母的大写组成的字符串  "^[a-z]+$" //由26个英文字母的小写组成的字符串  "^[A-Za-z0-9]+$" //由数字和26个英文字母组成的字符串  "^\w+$" //由数字、26个英文字母或者下划线组成的字符串  "^[\w-]+(\.[\w-]+)\*@[\w-]+(\.[\w-]+)+$" //email地址  "^[a-zA-z]+://(\w+(-\w+)\*)(\.(\w+(-\w+)\*))\*(\?\S\*)?$" //url  /^(d{2}|d{4})-((0([1-9]{1}))|(1[1|2]))-(([0-2]([1-9]{1}))|(3[0|1]))$/ // 年-月-日  /^((0([1-9]{1}))|(1[1|2]))/(([0-2]([1-9]{1}))|(3[0|1]))/(d{2}|d{4})$/ // 月/日/年  "^([w-.]+)@(([[0-9]{1,3}.[0-9]{1,3}.[0-9]{1,3}.)|(([w-]+.)+))([a-zA-Z]{2,4}|[0-9]{1,3})(]?)$" //Emil  "(d+-)?(d{4}-?d{7}|d{3}-?d{8}|^d{7,8})(-d+)?" //电话号码  "^(d{1,2}|1dd|2[0-4]d|25[0-5]).(d{1,2}|1dd|2[0-4]d|25[0-5]).(d{1,2}|1dd|2[0-4]d|25[0-5]).(d{1,2}|1dd|2[0-4]d|25[0-5])$" //IP地址  YYYY-MM-DD基本上把闰年和2月等的情况都考虑进去了  ^((((1[6-9]|[2-9]\d)\d{2})-(0?[13578]|1[02])-(0?[1-9]|[12]\d|3[01]))|(((1[6-9]|[2-9]\d)\d{2})-(0?[13456789]|1[012])-(0?[1-9]|[12]\d|30))|(((1[6-9]|[2-9]\d)\d{2})-0?2-(0?[1-9]|1\d|2[0-8]))|(((1[6-9]|[2-9]\d)(0[48]|[2468][048]|[13579][26])|((16|[2468][048]|[3579][26])00))-0?2-29-))$

# C#正则表达式  图片 src[^>]\*[^/].(?:jpg|bmp|gif)(?:\"|\')  中文 ^([\u4e00-\u9fa5]+|[a-zA-Z0-9]+)$  网址 "\<a.+?href=['""](?!http\:\/\/)(?!mailto\:)(?>foundAnchor>[^'"">]+?)[^>]\*?\>"  匹配中文字符的正则表达式： [\u4e00-\u9fa5]  匹配双字节字符(包括汉字在内)：[^\x00-\xff]  匹配空行的正则表达式：\n[\s| ]\*\r  匹配HTML标记的正则表达式：/<(.\*)>.\*<\/\1>|<(.\*) \/>/  匹配首尾空格的正则表达式：(^\s\*)|(\s\*$)（像vbscript那样的trim函数）  匹配Email地址的正则表达式：\w+([-+.]\w+)\*@\w+([-.]\w+)\*\.\w+([-.]\w+)\*  匹配网址URL的正则表达式：[http://](http://www.cnblogs.com/qyz123/archive/2007/05/12/743537.html" \t "_blank)([\w-]+\.)+[\w-]+(/[\w- ./?%&=]\*)?  ---------------------------------------------------------------------------  以下是例子：  利用正则表达式限制网页表单里的文本框输入内容：  用正则表达式限制只能输入中文：onkeyup="value=value.replace(/[^\u4E00-\u9FA5]/g,'')" onbeforepaste="clipboardData.setData('text',clipboardData.getData('text').replace(/[^\u4E00-\u9FA5]/g,''))"  1.用正则表达式限制只能输入全角字符： onkeyup="value=value.replace(/[^\uFF00-\uFFFF]/g,'')" onbeforepaste="clipboardData.setData('text',clipboardData.getData('text').replace(/[^\uFF00-\uFFFF]/g,''))"  2.用正则表达式限制只能输入数字：onkeyup="value=value.replace(/[^\d]/g,'') "onbeforepaste="clipboardData.setData('text',clipboardData.getData('text').replace(/[^\d]/g,''))"  3.用正则表达式限制只能输入数字和英文：onkeyup="value=value.replace(/[\W]/g,'') "onbeforepaste="clipboardData.setData('text',clipboardData.getData('text').replace(/[^\d]/g,''))"  4.计算字符串的长度（一个双字节字符长度计2，ASCII字符计1）  String.prototype.len=function(){return this.replace([^\x00-\xff]/g,"aa").length;}  5.javascript中没有像vbscript那样的trim函数，我们就可以利用这个表达式来实现，如下：  String.prototype.trim = function()  {  return this.replace(/(^\s\*)|(\s\*$)/g, "");  }  利用正则表达式分解和转换IP地址：  6.下面是利用正则表达式匹配IP地址，并将IP地址转换成对应数值的Javascript程序：  function IP2V(ip)  {  re=/(\d+)\.(\d+)\.(\d+)\.(\d+)/g //匹配IP地址的正则表达式  if(re.test(ip))  {  return RegExp.$1\*Math.pow(255,3))+RegExp.$2\*Math.pow(255,2))+RegExp.$3\*255+RegExp.$4\*1  }  else  {  throw new Error("不是一个正确的IP地址!")  }  }  不过上面的程序如果不用正则表达式，而直接用split函数来分解可能更简单，程序如下：  var ip="10.100.20.168"  ip=ip.split(".")  alert("IP值是："+(ip[0]\*255\*255\*255+ip[1]\*255\*255+ip[2]\*255+ip[3]\*1))  (?<=>)[^>]\*(?=<)

**正则表达式基础知识**

# 一个正则表达式就是由普通字符（例如字符 a 到 z）以及特殊字符（称为元字符）组成的文字模式。该模式描述在查找文字主体时待匹配的一个或多个字符串。正则表达式作为一个模板，将某个字符模式与所搜索的字符串进行匹配。如：

|  |  |  |
| --- | --- | --- |
| JScript | **VBScript** | **匹配** |
| /^\[ \t]\*$/ | "^\[ \t]\*$" | 匹配一个空白行。 |
| /\d{2}-\d{5}/ | "\d{2}-\d{5}" | 验证一个ID 号码是否由一个2位数字，一个连字符以及一个5位数字组成。 |
| /<(.\*)>.\*<\/\1>/ | "<(.\*)>.\*<\/\1>" | 匹配一个 HTML 标记。 |

# 下表是元字符及其在正则表达式上下文中的行为的一个完整列表：

|  |  |
| --- | --- |
| 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 向后引用、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 "foooood" 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。例如，"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JScript 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 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'。 |
| \cx | 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\_]'。 |
| \xn | 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向后引用。如果 \n 之前至少 n 个获取的子表达式，则 n 为向后引用。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向后引用。如果 \nm 之前至少有 nm 个获得子表达式，则 nm 为向后引用。如果 \nm 之前至少有 n 个获取，则 n 为一个后跟文字 m 的向后引用。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。 |
| \un | 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 (©)。 |

下面看几个例子：  
"^The"：表示所有以"The"开始的字符串（"There"，"The cat"等）；   
"of despair$"：表示所以以"of despair"结尾的字符串；   
"^abc$"：表示开始和结尾都是"abc"的字符串——呵呵，只有"abc"自己了；   
"notice"：表示任何包含"notice"的字符串。   
  
'\*'，'+'和'?'这三个符号，表示一个或一序列字符重复出现的次数。它们分别表示“没有或   
更多”，“一次或更多”还有“没有或一次”。下面是几个例子：

"ab\*"：表示一个字符串有一个a后面跟着零个或若干个b。（"a", "ab", "abbb",……）；   
"ab+"：表示一个字符串有一个a后面跟着至少一个b或者更多；   
"ab?"：表示一个字符串有一个a后面跟着零个或者一个b；   
"a?b+$"：表示在字符串的末尾有零个或一个a跟着一个或几个b。

也可以使用范围，用大括号括起，用以表示重复次数的范围。

"ab{2}"：表示一个字符串有一个a跟着2个b（"abb"）；   
"ab{2,}"：表示一个字符串有一个a跟着至少2个b；   
"ab{3,5}"：表示一个字符串有一个a跟着3到5个b。

请注意，你必须指定范围的下限（如："{0,2}"而不是"{,2}"）。还有，你可能注意到了，'\*'，'+'和   
'?'相当于"{0,}"，"{1,}"和"{0,1}"。   
还有一个'¦'，表示“或”操作：

"hi¦hello"：表示一个字符串里有"hi"或者"hello"；   
"(b¦cd)ef"：表示"bef"或"cdef"；   
"(a¦b)\*c"：表示一串"a""b"混合的字符串后面跟一个"c"；

'.'可以替代任何字符：

"a.[0-9]"：表示一个字符串有一个"a"后面跟着一个任意字符和一个数字；   
"^.{3}$"：表示有任意三个字符的字符串（长度为3个字符）；

方括号表示某些字符允许在一个字符串中的某一特定位置出现：

"[ab]"：表示一个字符串有一个"a"或"b"（相当于"a¦b"）；   
"[a-d]"：表示一个字符串包含小写的'a'到'd'中的一个（相当于"a¦b¦c¦d"或者"[abcd]"）；   
"^[a-zA-Z]"：表示一个以字母开头的字符串；   
"[0-9]%"：表示一个百分号前有一位的数字；   
",[a-zA-Z0-9]$"：表示一个字符串以一个逗号后面跟着一个字母或数字结束。

你也可以在方括号里用'^'表示不希望出现的字符，'^'应在方括号里的第一位。（如："%[^a-zA-Z]%"表   
示两个百分号中不应该出现字母）。

为了逐字表达，必须在"^.$()¦\*+?{\"这些字符前加上转移字符'\'。

请注意在方括号中，不需要转义字符。

# 全面剖析C#正则表达式        到目前为止，许多的编程语言和工具都包含对正则表达式的支持，当然.NET也不例外，.NET基础类库中包含有一个名称空间和一系列可以充分发挥规则表达式威力的类。          正则表达式的知识可能是不少编程人员最烦恼的事儿了。如果你还没有规则表达式方面的知识的话，建议从正则表达式的基础知识入手。。          下面就来研究C#中的正则表达式，C#中的正则表达式包含在.NET基础雷库的一个名称空间下，这个名称空间就是System.Text.RegularExpressions。该名称空间包括8个类，1个枚举，1个委托。他们分别是：                      Capture: 包含一次匹配的结果；                       CaptureCollection: Capture的序列；                       Group: 一次组记录的结果，由Capture继承而来；                       GroupCollection：表示捕获组的集合                      Match: 一次表达式的匹配结果，由Group继承而来；                       MatchCollection: Match的一个序列；                       MatchEvaluator: 执行替换操作时使用的委托；                       Regex：编译后的表达式的实例。                       RegexCompilationInfo：提供编译器用于将正则表达式编译为独立程序集的信息                      RegexOptions 提供用于设置正则表达式的枚举值 Regex类中还包含一些静态的方法：                      Escape: 对字符串中的regex中的转义符进行转义；                      IsMatch: 如果表达式在字符串中匹配，该方法返回一个布尔值；                      Match: 返回Match的实例；                      Matches: 返回一系列的Match的方法；                      Replace: 用替换字符串替换匹配的表达式；                      Split: 返回一系列由表达式决定的字符串；                      Unescape:不对字符串中的转义字符转义。 下面介绍他们的用途：         先看一个简单的匹配例子，我们首先从使用Regex、Match类的简单表达式开始学习。　Match m = Regex.Match("abracadabra", "(a|b|r)+"); 我们现在有了一个可以用于测试的Match类的实例，例如：if (m.Success){}，如果想使用匹配的字符串，可以把它转换成一个字符串： 　　MesaageBox.Show("Match="+m.ToString()); 这个例子可以得到如下的输出: Match=abra。这就是匹配的字符串了。         Regex 类表示只读正则表达式类。它还包含各种静态方法（在下面的实例中将逐一介绍），允许在不显式创建其他类的实例的情况下使用其他正则表达式类。         以下代码示例创建了 Regex 类的实例并在初始化对象时定义一个简单的正则表达式。声明一个Regex对象变量：Regex objAlphaPatt;，接着创建Regex对象的一个实例，并定义其规则：objAlphaPatt=new Regex("[^a-zA-Z]");         IsMatch方法指示 Regex 构造函数中指定的正则表达式在输入字符串中是否找到匹配项。这是我们使用C#正则表达式时最常用的方法之一。下面的例子说明了IsMatch方法的使用： if( !objAlphaPatt.IsMatch("testisMatchMethod"))  lblMsg.Text = "匹配成功"; else  lblMsg.Text = "匹配不成功"; 这段代码执行的结果是“匹配成功” if( ! objAlphaPatt.IsMatch("testisMatchMethod7654298"))  lblMsg.Text = "匹配成功"; else  lblMsg.Text = "匹配不成功"; 这段代码执行的结果是“匹配不成功”          Escape方法表示把转义字符作为字符本身使用，而不再具有转义作用，最小的元字符集（\、\*、+、?、|、{、[、(、)、^、$、.、# 和空白）。Replace方法则是用指定的替换字符串替换由正则表达式定义的字符模式的所有匹配项。看下面的例子，还是使用上面定义的Regex对 象：objAlphaPatt.Replace("this [test] \*\* replace and escape" ,Regex.Escape("()"));他的返回结果是：this\(\)\(\)test\(\)\(\)\(\)\(\)\(\)replace \(\)and\(\)escape，如果不是Escape的话，则返回结果是：this()()test()()()() ()replace()and()escape，Unescape 反转由 Escape 执行的转换，但是，Escape 无法完全反转 Unescape。         Split方法是把由正则表达式匹配项定义的位置将输入字符串拆分为一个子字符串数组。例如： Regex r = new Regex("-"); // Split on hyphens. string[] s = r.Split("first-second-third"); for(int i=0;i<s.Length;i++) {  Response.Write(s[i]+"<br>"); }

执行的结果是：  
First  
Second  
Third

        看上去和String的Split方法一样，但string的Split方法在由正则表达式而不是一组字符确定的分隔符处拆分字符串。  
  
        Match方法是在输入字符串中搜索正则表达式的匹配项，并Regex 类的 Match 方法返回 Match 对象，Match 类表示正则表达式匹配操作的结果。下面的例子演示Match方法的使用，并利用Match对象的Group属性返回Group对象：  
  
string text = @"public string testMatchObj string s string  match ";  
string pat = @"(\w+)\s+(string)";  
// Compile the regular expression.  
Regex r = new Regex(pat, RegexOptions.IgnoreCase);  
// Match the regular expression pattern against a text string.  
Match m = r.Match(text);  
int matchCount = 0;  
while (m.Success)   
{  
 Response.Write("Match"+ (++matchCount) + "<br>");  
 for (int i = 1; i <= 2; i++)   
 {  
  Group g = m.Groups[i];  
  Response.Write("Group"+i+"='" + g + "'"  + "<br>");  
  CaptureCollection cc = g.Captures;  
  for (int j = 0; j < cc.Count; j++)   
  {  
   Capture c = cc[j];  
   Response.Write("Capture"+j+"='" + c + "', Position="+c.Index + "<br>");  
  }  
 }  
 m = m.NextMatch();  
}

该事例运行结果是：  
Match1  
Group1='public'  
Capture0='public', Position=0  
Group2='string'  
Capture0='string', Position=7  
Match2  
Group1='testMatchObj'  
Capture0='testMatchObj', Position=14  
Group2='string'  
Capture0='string', Position=27  
Match3  
Group1='s'  
Capture0='s', Position=34  
Group2='string'  
Capture0='string', Position=36  
  
        MatchCollection 类表示成功的非重叠匹配的只读的集合，MatchCollection 的实例是由 Regex.Matches 属性返回的，下面的实例说明了通过在输入字符串中找到所有与Regex中指定的匹配并填充 MatchCollection。  
  
MatchCollection mc;  
Regex r = new Regex("match");   
mc = r.Matches("matchcollectionregexmatchs");  
for (int i = 0; i < mc.Count; i++)   
{  
 Response.Write( mc[i].Value + " POS:" + mc[i].Index.ToString() + "<br>");  
}  
该实例运行的结果是：  
match POS:0  
match POS:20

[**正则表达式基本语法**](http://www.cnblogs.com/light169/archive/2006/10/18/532347.html)

1.正则表达式基本语法

两个特殊的符号'^'和'$'。他们的作用是分别指出一个字符串的开始和结束。例子如下：

"^The"：表示所有以"The"开始的字符串（"There"，"The cat"等）；  
"of despair$"：表示所以以"of despair"结尾的字符串；  
"^abc$"：表示开始和结尾都是"abc"的字符串——呵呵，只有"abc"自己了；  
"notice"：表示任何包含"notice"的字符串。

象最后那个例子，如果你不使用两个特殊字符，你就在表示要查找的串在被查找串的任意部分——你并  
不把它定位在某一个顶端。

其它还有'\*'，'+'和'?'这三个符号，表示一个或一序列字符重复出现的次数。它们分别表示“没有或  
更多”，“一次或更多”还有“没有或一次”。下面是几个例子：

"ab\*"：表示一个字符串有一个a后面跟着零个或若干个b。（"a", "ab", "abbb",……）；  
"ab+"：表示一个字符串有一个a后面跟着至少一个b或者更多；  
"ab?"：表示一个字符串有一个a后面跟着零个或者一个b；  
"a?b+$"：表示在字符串的末尾有零个或一个a跟着一个或几个b。

你也可以使用范围，用大括号括起，用以表示重复次数的范围。

"ab{2}"：表示一个字符串有一个a跟着2个b（"abb"）；  
"ab{2,}"：表示一个字符串有一个a跟着至少2个b；  
"ab{3,5}"：表示一个字符串有一个a跟着3到5个b。

请注意，你必须指定范围的下限（如："{0,2}"而不是"{,2}"）。还有，你可能注意到了，'\*'，'+'和  
'?'相当于"{0,}"，"{1,}"和"{0,1}"。  
还有一个'¦'，表示“或”操作：

"hi¦hello"：表示一个字符串里有"hi"或者"hello"；  
"(b¦cd)ef"：表示"bef"或"cdef"；  
"(a¦b)\*c"：表示一串"a""b"混合的字符串后面跟一个"c"；

'.'可以替代任何字符：

"a.[0-9]"：表示一个字符串有一个"a"后面跟着一个任意字符和一个数字；  
"^.{3}$"：表示有任意三个字符的字符串（长度为3个字符）；

方括号表示某些字符允许在一个字符串中的某一特定位置出现：

"[ab]"：表示一个字符串有一个"a"或"b"（相当于"a¦b"）；  
"[a-d]"：表示一个字符串包含小写的'a'到'd'中的一个（相当于"a¦b¦c¦d"或者"[abcd]"）；  
"^[a-zA-Z]"：表示一个以字母开头的字符串；  
"[0-9]%"：表示一个百分号前有一位的数字；  
",[a-zA-Z0-9]$"：表示一个字符串以一个逗号后面跟着一个字母或数字结束。

你也可以在方括号里用'^'表示不希望出现的字符，'^'应在方括号里的第一位。（如："%[^a-zA-Z]%"表  
示两个百分号中不应该出现字母）。

为了逐字表达，你必须在"^.$()¦\*+?{\"这些字符前加上转移字符'\'。

请注意在方括号中，不需要转义字符。  
  
  
  
2.正则表达式验证控制文本框的输入字符类型  
**1.只能输入数字和英文的：**   
<input onkeyup="value=value.replace(/[\W]/g,'') " onbeforepaste="clipboardData.setData('text',clipboardData.getData('text').replace(/[^\d]/g,''))" ID="Text1" NAME="Text1">  
  
**2.只能输入数字的：**   
<input onkeyup="value=value.replace(/[^\d]/g,'') " onbeforepaste="clipboardData.setData('text',clipboardData.getData('text').replace(/[^\d]/g,''))" ID="Text2" NAME="Text2">  
  
**3.只能输入全角的：**   
<input onkeyup="value=value.replace(/[^\uFF00-\uFFFF]/g,'')" onbeforepaste="clipboardData.setData('text',clipboardData.getData('text').replace(/[^\uFF00-\uFFFF]/g,''))" ID="Text3" NAME="Text3">  
  
**4.只能输入汉字的：**   
<input onkeyup="value=value.replace(/[^\u4E00-\u9FA5]/g,'')" onbeforepaste="clipboardData.setData('text',clipboardData.getData('text').replace(/[^\u4E00-\u9FA5]/g,''))" ID="Text4" NAME="Text4">  
  
  
**3.正则表达式的应用实例通俗说明**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**//校验是否全由数字组成**

**/^[0-9]{1,20}$/**

**^ 表示打头的字符要匹配紧跟^后面的规则**

**$ 表示打头的字符要匹配紧靠$前面的规则**

**[ ] 中的内容是可选字符集**

**[0-9] 表示要求字符范围在0-9之间**

**{1,20}表示数字字符串长度合法为1到20，即为[0-9]中的字符出现次数的范围是1到20次。**

**/^ 和 $/成对使用应该是表示要求整个字符串完全匹配定义的规则，而不是只匹配字符串中的一个子串。**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**//校验登录名：只能输入5-20个以字母开头、可带数字、“\_”、“.”的字串**

**/^[a-zA-Z]{1}([a-zA-Z0-9]|[.\_]){4,19}$/**

**^[a-zA-Z]{1} 表示第一个字符要求是字母。**

**([a-zA-Z0-9]|[.\_]){4,19} 表示从第二位开始（因为它紧跟在上个表达式后面）的一个长度为4到9位的字符串，它要求是由大小写字母、数字或者特殊字符集[.\_]组成。**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**//校验用户姓名：只能输入1-30个以字母开头的字串**

**/^[a-zA-Z]{1,30}$/**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**//校验密码：只能输入6-20个字母、数字、下划线**

**/^(\w){6,20}$/**

**\w：用于匹配字母，数字或下划线字符**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**//校验普通电话、传真号码：可以“+”或数字开头，可含有“-” 和 “ ”**

**/^[+]{0,1}(\d){1,3}[ ]?([-]?((\d)|[ ]){1,12})+$/**

**\d：用于匹配从0到9的数字；**

**“?”元字符规定其前导对象必须在目标对象中连续出现零次或一次**

**可以匹配的字符串如：+123 -999 999 ； +123-999 999 ；123 999 999 ；+123 999999等**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**//校验URL**

**/^http[s]{0,1}:\/\/.+$/ 或 /^http[s]{0,1}:\/\/.{1,n}$/ (表示url串的长度为length(“https://”) + n )**

**\ / ：表示字符“/”。**

**. 表示所有字符的集**

**+ 等同于{1,}，就是1到正无穷吧。**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**//校验纯中文字符**

**/^[\u4E00-\u9FA5]+$/**

**[\u4E00-\u9FA5] ：估计是中文字符集的范围吧**

**以上表达式均在下面的javascript中测试通过**

**<html>**

**<script language="JavaScript">**

**<!--**

**function regx(r,s)**

**{**

**if (r == null || r == ""){**

**return false;**

**}**

**var patrn= new RegExp(r);**

**if (patrn.exec(s))**

**return true**

**return false**

**}**

**-->**

**</script>**

**<body>**

**<form>**

**规则表达式 ： <input type="input" name="regxStr" value="" > (填写/ /之间的表达式)**

**<br>**

**校验字符串 ： <input type="input" name="str" value="" >**

**<input type="button" name="match" value="匹配" onClick="alert(regx(regxStr.value,str.value));">**

**</form>**

**</body>**

**</html>  
  
4.正則表達式應用**

**"^\d+$"　　//非负整数（正整数 + 0）   
"^[0-9]\*[1-9][0-9]\*$"　　//正整数   
"^((-\d+)|(0+))$"　　//非正整数（负整数 + 0）   
"^-[0-9]\*[1-9][0-9]\*$"　　//负整数   
"^-?\d+$"　　　　//整数   
"^\d+(\.\d+)?$"　　//非负浮点数（正浮点数 + 0）   
"^(([0-9]+\.[0-9]\*[1-9][0-9]\*)|([0-9]\*[1-9][0-9]\*\.[0-9]+)|([0-9]\*[1-9][0-9]\*))$"　　//正浮点数   
"^((-\d+(\.\d+)?)|(0+(\.0+)?))$"　　//非正浮点数（负浮点数 + 0）   
"^(-(([0-9]+\.[0-9]\*[1-9][0-9]\*)|([0-9]\*[1-9][0-9]\*\.[0-9]+)|([0-9]\*[1-9][0-9]\*)))$"　　//负浮点数   
"^(-?\d+)(\.\d+)?$"　　//浮点数   
"^[A-Za-z]+$"　　//由26个英文字母组成的字符串   
"^[A-Z]+$"　　//由26个英文字母的大写组成的字符串   
"^[a-z]+$"　　//由26个英文字母的小写组成的字符串   
"^[A-Za-z0-9]+$"　　//由数字和26个英文字母组成的字符串   
"^\w+$"　　//由数字、26个英文字母或者下划线组成的字符串   
"^[\w-]+(\.[\w-]+)\*@[\w-]+(\.[\w-]+)+$"　　　　//email地址   
"^[a-zA-z]+://(\w+(-\w+)\*)(\.(\w+(-\w+)\*))\*(\?\S\*)?$"　　//url  
/^(d{2}|d{4})-((0([1-9]{1}))|(1[1|2]))-(([0-2]([1-9]{1}))|(3[0|1]))$/   //  年-月-日  
/^((0([1-9]{1}))|(1[1|2]))/(([0-2]([1-9]{1}))|(3[0|1]))/(d{2}|d{4})$/   // 月/日/年  
"^([w-.]+)@(([[0-9]{1,3}.[0-9]{1,3}.[0-9]{1,3}.)|(([w-]+.)+))([a-zA-Z]{2,4}|[0-9]{1,3})(]?)$"   //Emil  
"(d+-)?(d{4}-?d{7}|d{3}-?d{8}|^d{7,8})(-d+)?"     //电话号码  
"^(d{1,2}|1dd|2[0-4]d|25[0-5]).(d{1,2}|1dd|2[0-4]d|25[0-5]).(d{1,2}|1dd|2[0-4]d|25[0-5]).(d{1,2}|1dd|2[0-4]d|25[0-5])$"   //IP地址**

**^([0-9A-F]{2})(-[0-9A-F]{2}){5}$   //MAC地址的正则表达式  
^[-+]?\d+(\.\d+)?$  //值类型正则表达式  
  
  
5.**javascript正则表达式检验  
//校验是否全由数字组成  
function isDigit(s)  
{  
var patrn=/^[0-9]{1,20}$/;  
if (!patrn.exec(s)) return false  
return true  
}

//校验登录名：只能输入5-20个以字母开头、可带数字、“\_”、“.”的字串  
function isRegisterUserName(s)  
{  
var patrn=/^[a-zA-Z]{1}([a-zA-Z0-9]|[.\_]){4,19}$/;  
if (!patrn.exec(s)) return false  
return true  
}

//校验用户姓名：只能输入1-30个以字母开头的字串  
function isTrueName(s)  
{  
var patrn=/^[a-zA-Z]{1,30}$/;  
if (!patrn.exec(s)) return false  
return true  
}

//校验密码：只能输入6-20个字母、数字、下划线  
function isPasswd(s)  
{  
var patrn=/^(\w){6,20}$/;  
if (!patrn.exec(s)) return false  
return true  
}

//校验普通电话、传真号码：可以“+”开头，除数字外，可含有“-”  
function isTel(s)  
{  
//var patrn=/^[+]{0,1}(\d){1,3}[ ]?([-]?(\d){1,12})+$/;  
var patrn=/^[+]{0,1}(\d){1,3}[ ]?([-]?((\d)|[ ]){1,12})+$/;  
if (!patrn.exec(s)) return false  
return true  
}

//校验手机号码：必须以数字开头，除数字外，可含有“-”  
function isMobil(s)  
{  
var patrn=/^[+]{0,1}(\d){1,3}[ ]?([-]?((\d)|[ ]){1,12})+$/;  
if (!patrn.exec(s)) return false  
return true  
}

//校验邮政编码  
function isPostalCode(s)  
{  
//var patrn=/^[a-zA-Z0-9]{3,12}$/;  
var patrn=/^[a-zA-Z0-9 ]{3,12}$/;  
if (!patrn.exec(s)) return false  
return true  
}

//校验搜索关键字  
function isSearch(s)  
{  
var patrn=/^[^`~!@#$%^&\*()+=|\\\][\]\{\}:;\'\,.<>/?]{1}[^`~!@$%^&()+=|\\\][\]\{\}:;\'\,.<>?]{0,19}$/;  
if (!patrn.exec(s)) return false  
return true  
}

function isIP(s) //by zergling  
{  
var patrn=/^[0-9.]{1,20}$/;  
if (!patrn.exec(s)) return false  
return true  
}  
   
   
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
\* FUNCTION: isBetween  
\* PARAMETERS: val AS any value  
\* lo AS Lower limit to check  
\* hi AS Higher limit to check  
\* CALLS: NOTHING  
\* RETURNS: TRUE if val is between lo and hi both inclusive, otherwise false.  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
function isBetween (val, lo, hi) {  
if ((val < lo) || (val > hi)) { return(false); }  
else { return(true); }  
}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
\* FUNCTION: isDate checks a valid date  
\* PARAMETERS: theStr AS String  
\* CALLS: isBetween, isInt  
\* RETURNS: TRUE if theStr is a valid date otherwise false.  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
function isDate (theStr) {  
var the1st = theStr.indexOf('-');  
var the2nd = theStr.lastIndexOf('-');

if (the1st == the2nd) { return(false); }  
else {  
var y = theStr.substring(0,the1st);  
var m = theStr.substring(the1st+1,the2nd);  
var d = theStr.substring(the2nd+1,theStr.length);  
var maxDays = 31;

if (isInt(m)==false || isInt(d)==false || isInt(y)==false) {  
return(false); }  
else if (y.length < 4) { return(false); }  
else if (!isBetween (m, 1, 12)) { return(false); }  
else if (m==4 || m==6 || m==9 || m==11) maxDays = 30;  
else if (m==2) {  
if (y % 4 > 0) maxDays = 28;  
else if (y % 100 == 0 && y % 400 > 0) maxDays = 28;  
else maxDays = 29;  
}  
if (isBetween(d, 1, maxDays) == false) { return(false); }  
else { return(true); }  
}  
}  
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
\* FUNCTION: isEuDate checks a valid date in British format  
\* PARAMETERS: theStr AS String  
\* CALLS: isBetween, isInt  
\* RETURNS: TRUE if theStr is a valid date otherwise false.  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
function isEuDate (theStr) {  
if (isBetween(theStr.length, 8, 10) == false) { return(false); }  
else {  
var the1st = theStr.indexOf('/');  
var the2nd = theStr.lastIndexOf('/');

if (the1st == the2nd) { return(false); }  
else {  
var m = theStr.substring(the1st+1,the2nd);  
var d = theStr.substring(0,the1st);  
var y = theStr.substring(the2nd+1,theStr.length);  
var maxDays = 31;

if (isInt(m)==false || isInt(d)==false || isInt(y)==false) {  
return(false); }  
else if (y.length < 4) { return(false); }  
else if (isBetween (m, 1, 12) == false) { return(false); }  
else if (m==4 || m==6 || m==9 || m==11) maxDays = 30;  
else if (m==2) {  
if (y % 4 > 0) maxDays = 28;  
else if (y % 100 == 0 && y % 400 > 0) maxDays = 28;  
else maxDays = 29;  
}

if (isBetween(d, 1, maxDays) == false) { return(false); }  
else { return(true); }  
}  
}

}  
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
\* FUNCTION: Compare Date! Which is the latest!  
\* PARAMETERS: lessDate,moreDate AS String  
\* CALLS: isDate,isBetween  
\* RETURNS: TRUE if lessDate<moreDate  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
function isComdate (lessDate , moreDate)  
{  
if (!isDate(lessDate)) { return(false);}  
if (!isDate(moreDate)) { return(false);}  
var less1st = lessDate.indexOf('-');  
var less2nd = lessDate.lastIndexOf('-');  
var more1st = moreDate.indexOf('-');  
var more2nd = moreDate.lastIndexOf('-');  
var lessy = lessDate.substring(0,less1st);  
var lessm = lessDate.substring(less1st+1,less2nd);  
var lessd = lessDate.substring(less2nd+1,lessDate.length);  
var morey = moreDate.substring(0,more1st);  
var morem = moreDate.substring(more1st+1,more2nd);  
var mored = moreDate.substring(more2nd+1,moreDate.length);  
var Date1 = new Date(lessy,lessm,lessd);   
var Date2 = new Date(morey,morem,mored);   
if (Date1>Date2) { return(false);}  
return(true);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
\* FUNCTION isEmpty checks if the parameter is empty or null  
\* PARAMETER str AS String  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
function isEmpty (str) {  
if ((str==null)||(str.length==0)) return true;  
else return(false);  
}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
\* FUNCTION: isInt  
\* PARAMETER: theStr AS String   
\* RETURNS: TRUE if the passed parameter is an integer, otherwise FALSE  
\* CALLS: isDigit  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
function isInt (theStr) {  
var flag = true;

if (isEmpty(theStr)) { flag=false; }  
else  
{ for (var i=0; i<theStr.length; i++) {  
if (isDigit(theStr.substring(i,i+1)) == false) {  
flag = false; break;  
}  
}  
}  
return(flag);  
}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
\* FUNCTION: isReal  
\* PARAMETER: heStr AS String   
decLen AS Integer (how many digits after period)  
\* RETURNS: TRUE if theStr is a float, otherwise FALSE  
\* CALLS: isInt  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
function isReal (theStr, decLen) {  
var dot1st = theStr.indexOf('.');  
var dot2nd = theStr.lastIndexOf('.');  
var OK = true;

if (isEmpty(theStr)) return false;

if (dot1st == -1) {  
if (!isInt(theStr)) return(false);  
else return(true);  
}

else if (dot1st != dot2nd) return (false);  
else if (dot1st==0) return (false);  
else {  
var intPart = theStr.substring(0, dot1st);  
var decPart = theStr.substring(dot2nd+1);

if (decPart.length > decLen) return(false);  
else if (!isInt(intPart) || !isInt(decPart)) return (false);  
else if (isEmpty(decPart)) return (false);  
else return(true);  
}  
}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
\* FUNCTION: isEmail  
\* PARAMETER: String (Email Address)  
\* RETURNS: TRUE if the String is a valid Email address  
\* FALSE if the passed string is not a valid Email Address  
\* EMAIL FORMAT: [AnyName@EmailServer](mailto:AnyName@EmailServer) e.g; [webmaster@hotmail.com](mailto:webmaster@hotmail.com)  
\* @ sign can appear only once in the email address.  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
function isEmail (theStr) {  
var atIndex = theStr.indexOf(['@'](mailto:%27@%27));  
var dotIndex = theStr.indexOf('.', atIndex);  
var flag = true;  
theSub = theStr.substring(0, dotIndex+1)

if ((atIndex < 1)||(atIndex != theStr.lastIndexOf(['@'))||(dotIndex](mailto:%27@%27%29%29%7C%7C%28dotIndex) < atIndex + 2)||(theStr.length <= theSub.length))   
{ return(false); }  
else { return(true); }  
}  
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
\* FUNCTION: newWindow  
\* PARAMETERS: doc -> Document to open in the new window  
hite -> Height of the new window  
wide -> Width of the new window  
bars -> 1-Scroll bars = YES 0-Scroll Bars = NO  
resize -> 1-Resizable = YES 0-Resizable = NO  
\* CALLS: NONE  
\* RETURNS: New window instance  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
function newWindow (doc, hite, wide, bars, resize) {  
var winNew="\_blank";  
var opt="toolbar=0,location=0,directories=0,status=0,menubar=0,";  
opt+=("scrollbars="+bars+",");  
opt+=("resizable="+resize+",");  
opt+=("width="+wide+",");  
opt+=("height="+hite);  
winHandle=window.open(doc,winNew,opt);  
return;  
}  
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
\* FUNCTION: DecimalFormat  
\* PARAMETERS: paramValue -> Field value  
\* CALLS: NONE  
\* RETURNS: Formated string  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
function DecimalFormat (paramValue) {  
var intPart = parseInt(paramValue);  
var decPart =parseFloat(paramValue) - intPart;

str = "";  
if ((decPart == 0) || (decPart == null)) str += (intPart + ".00");  
else str += (intPart + decPart);

return (str);  
}

"^\\d+$"　　//非负整数（正整数 + 0）   
"^[0-9]\*[1-9][0-9]\*$"　　//正整数   
"^((-\\d+)|(0+))$"　　//非正整数（负整数 + 0）   
"^-[0-9]\*[1-9][0-9]\*$"　　//负整数   
"^-?\\d+$"　　　　//整数   
"^\\d+([\\.\\d+)?$](file:///\\.\d+%29?$)"　　//非负浮点数（正浮点数 + 0）   
"^(([0-9]+\\.[0-9]\*[1-9][0-9]\*)|([0-9]\*[1-9][0-9]\*\\.[0-9]+)|([0-9]\*[1-9][0-9]\*))$"　　//正浮点数   
"^((-\\d+([\\.\\d+)?)|(0+(\\.0+)?))$](file:///\\.\d+%29?%29%7C%280+%28//.0+%29?%29%29$)"　　//非正浮点数（负浮点数 + 0）   
"^(-(([0-9]+\\.[0-9]\*[1-9][0-9]\*)|([0-9]\*[1-9][0-9]\*\\.[0-9]+)|([0-9]\*[1-9][0-9]\*)))$"　　//负浮点数   
"^(-?\\d+)([\\.\\d+)?$](file:///\\.\d+%29?$)"　　//浮点数   
"^[A-Za-z]+$"　　//由26个英文字母组成的字符串   
"^[A-Z]+$"　　//由26个英文字母的大写组成的字符串   
"^[a-z]+$"　　//由26个英文字母的小写组成的字符串   
"^[A-Za-z0-9]+$"　　//由数字和26个英文字母组成的字符串   
"^\\w+$"　　//由数字、26个英文字母或者下划线组成的字符串   
"^[\\w-]+(\\.[\\w-]+)\*@[\\w-]+(\\.[\\w-]+)+$"　　　　//email地址   
"^[a-zA-z]+://(\\w+(-\\w+)\*)(\\.(\\w+(-\\w+)\*))\*(\\?\\S\*)?$"　　//url