# 一、正则表达式(regular expression简称res)

## 1、主要用途：

正则表达式被用来匹配一组文字。通常，它有两类用途：

1. 数据有效性验证
2. 查找和替换

## 2、如何来构造正则表达式：

对 JScript 而言，分隔符为一对正斜杠 (/) 字符。

* **构造器函数方法使用方法如下：**new RegExp("pattern", "flags")
* **文本格式： /pattern/flags**

参数说明：

pattern ：一个正则表达式文本

flags ： 如果存在，将是以下值：

g ： global match(全局匹配)

i ：ignore case(忽略大小写)

gi ： both global match and ignore case(匹配所有可能的值，也忽略大小写)

**注意：**文本格式中的参数不要使用引号标记，而构造器函数的参数则要使用引号标记。所以下面的  
表达式建立同样的正则表达式：/ab+c/i 等价于 　　new RegExp("ab+c", "i")

使用 文本格式 文本的长度最大支持128个字符,

**描述：**  
当使用构造函数的时候，必须使用正常的字符串避开规则(在字符串中加入前导字符\ )是必须的。  
例如，下面的两条语句是等价的：  
re = new RegExp("\\w+")  
re = /\w+/

# 二、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** |
| **\** | 转义符 |
| **^** | 匹配输入字符串的开始位置。 |
| **$** | 匹配输入字符串的结束位置。 |
| **\*** | 匹配任意次，等价于{0,} |
| **+** | 匹配至少一次。等价于{1,} |
| **?** | 匹配0次或1次，等价于{0,1} |
| **{*n*}** | 恰好匹配n次 |
| **{*n*,}** | 至少匹配n次 |
| **{*n*,*m*}** | 至少匹配n次，最多匹配m次。 |
| **?** | 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。在JScript 中则使用 **$1**…**$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 |
| **[*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*]** | 处理xyz，可以匹配任意字符。 |
| **[*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。例如:[a-z] [A-Z] [0-9] |
| **[^*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 (?)。 |

# 三、正则表达式的常用方法

|  |  |
| --- | --- |
| **regexp.test(string)** | 用来测试一个字符串是否能够被匹配。它返回ture或false两个值。 |
| **regexp.exec(string)** | 在指定的字符串中执行搜寻一个匹配，匹配的结果是通过一个数组返回。 |

# 四、与正则表达式有关的字符串对象的方法：

|  |  |
| --- | --- |
| **string.replace(pattern,string)** | 替换在正则表达式查找中找到的文本。 |
| **string.search(pattern)** | 通过正则表达式查找相应的字符串，只是判断有无匹配的字符串。如果查找成功，search返回匹配串的位置， 否则返回-1。 |
| **string.match(pattern)** | match方法执行全局查找，查找结果存放在一个数组里。 |

# 五、常用的正则表达式的操作符

|  |  |
| --- | --- |
| Symbol | Function |
| \ | 转义符 |
| (), (?:), (?=), [] | 括号 |
| \*, +, ?, {n}, {n,}, {n,m} | 限定符 |
| ^, $, \anymetacharacter | 定位符 |
| | | 或 |

# 六、常用的正则表达式示例：

1、匹配所有的正数：^[0-9]+$

2、匹配所有的小数：^\-?[0-9]\*\.?[0-9]\*$

3、匹配所有的整数：^\-?[0-9]+$

11、匹配HTML标记的正则表达式：/<(.\*)>.\*<\/\1>|<(.\*) \/>/

12、匹配邮箱：/^([a-zA-Z0-9\_-])+@([a-zA-Z0-9\_-])+(.[a-zA-Z0-9\_-])+/

4、提取信息中的中文字符串： [\u4e00-\u9fa5]\* ;

5、提取信息中的邮件地址：\w+([-+.]\w+)\*@\w+([-.]\w+)\*\.\w+([-.]\w+)\*   
6、提取信息中的中国手机号码：(86)\*0\*13\d{9}   
7、提取信息中的中国固定电话号码：(\(\d{3,4}\)|\d{3,4}-|\s)?\d{8}   
8、提取信息中的中国邮政编码：[1-9]{1}(\d+){5}   
9、提取信息中的中国身份证号码：\d{18}|\d{15}   
10、提取信息中的任何数字：(-?\d\*)(\.\d+)?

# 常用表达式

1、匹配中文字符的正则表达式： [\u4e00-\u9fa5]

* 评注：匹配中文还真是个头疼的事，有了这个表达式就好办了

2、匹配双字节字符(包括汉字在 内)：[^\x00-\xff]

* 评注：可以用来计算字符串的长度（一个双字节字符长度计2，ASCII字符计1）

3、匹配空白行的正 则表达式：\n\s\*\r

* 评注：可以用来删除空白行

4、匹配HTML标记的正则表达式：<(\S\*?) [^>]\*>.\*?</\1>|<.\*? />

* 评注：网上流传的版本太糟糕，上面这个也仅仅能匹配部分，对 于复杂的嵌套标记依旧无能为力

5、匹配首尾空白字符的正则表达式：^\s\*|\s\*$

* 评注：可以用来删除行首行尾的空白字符(包括空 格、制表符、换页符等等)，非常有用的表达式

6、匹配Email地址的正则表达式：\w+([-+.]\w+)\*@\w+([-.] \w+)\*\.\w+([-.]\w+)\*

* 评注：表单验证时很实用

7、匹配网址URL的正则表达式：[a-zA- z]+://[^\s]\*

* 评注：网上流传的版本功能很有限，上面这个基本可以满足需求

8、匹配帐号是否合法(字母开头，允许5-16 字节，允许字母数字下划线)：^[a-zA-Z][a-zA-Z0-9\_]{4,15}

* 评注：表单验证时很实用

9、匹配国内电话号 码：\d{3}-\d{8}|\d{4}-\d{7}

* 评注：匹配形式如 0511-4405222 或 021-87888822

10、匹 配腾讯QQ号：[1-9][0-9]{4,}

* 评注：腾讯QQ号从10000开始

11、匹配中国邮政编码：[1-9]\d{5}(?! \d)

* 评注：中国邮政编码为6位数字

12、匹配身份证：\d{15}|\d{18}

* 评注：中国的身份证为15位或18位

13、匹 配ip地址：\d+\.\d+\.\d+\.\d+

* 评注：提取ip地址时有用

14、匹配特定数字：

* ^[1-9]\d\*$　 　  //匹配正整数
* ^-[1-9]\d\*$ 　 //匹配负整数
* ^-?[1-9]\d\*$　　 //匹配整数
* ^[1-9]\d\*|0$ 　 //匹配非负整数（正整数 + 0）
* ^-[1-9]\d\*|0$　　 //匹配非正整数（负整数 + 0）
* ^[1-9]\d\*\.\d\*|0\.\d\*[1-9]\d\*$ 　　 //匹配正浮点数
* ^-([1-9]\d\*\.\d\*|0\.\d\*[1-9]\d\*)$　 //匹配负浮点数
* ^-?([1-9]\d\*\.\d\*|0\.\d\*[1-9]\d\*|0?\.0+|0)$ 　 //匹配浮点数
* ^[1-9]\d\*\.\d\*|0\.\d\*[1-9]\d\*|0?\.0+|0$　　 //匹配非负浮点数（正浮点 数 + 0）
* ^(-([1-9]\d\*\.\d\*|0\.\d\*[1-9]\d\*))|0?\.0+|0$　　//匹配非正浮点数（负浮点 数 + 0）
* 评注：处理大量数据时有用，具体应用时注意修正

1. 匹配特定字符串：

* ^[A-Za-z]+$　　//匹配由26 个英文字母组成的字符串  
  ^[A-Z]+$　　//匹配由26个英文字母的大写组成的字符串  
  ^[a-z]+$　　//匹配由26个英文字母 的小写组成的字符串  
  ^[A-Za-z0-9]+$　　//匹配由数字和26个英文字母组成的字符串  
  ^\w+$　　//匹配由数字、26个 英文字母或者下划线组成的字符串  
  评注：最基本也是最常用的一些表达式

1. 加了时间验证的

^((((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-)) (20|21|22|23|[0-1]?\d):[0-5]?\d:[0-5]?\d$