logo.png

BOM

H5 web前端

[www.bufanui.com](http://www.bufanui.com)

# 一、缓动函数

## 1.1 存在的问题

哪些属性无法获取值或者无法设置值

## 1.2 主要处理两个

### 第一个是透明度

### 第二个是层级

## 1.3 缓动案例

### 旋转木马

### 

### 正转反转的问题

# 二、正则表达式（RegExp）

## 2.1 概述

正则表达式，又称规则表达式。（英语：Regular Expression，在代码中常简写为regex、regexp或RE），计算机科学的一个概念。正则表通常被用来检索、替换那些符合某个模式(规则)的文本。

正则表达式是对字符串操作的一种逻辑公式，就是用事先定义好的一些特定字符、及这些特定字符的组合，组成一个“规则字符串”，这个“规则字符串”用来表达对字符串的一种过滤逻辑。

给定一个正则表达式和另一个字符串，我们可以达到如下的目的：

1. 给定的字符串是否符合正则表达式的过滤逻辑（称作“匹配”）；

2. 可以通过正则表达式，从字符串中获取我们想要的特定部分。

由于正则表达式主要应用对象是文本，因此它在各种文本编辑器场合都有应用。

如：表单验证、高级搜索

（有一定难度，不要求非常熟练，但至少会表单验证）

1. 表单验证



1. 隐藏手机号码： 150\*\*\*\*7654
2. 可以通过正则表达式，从字符串中获取我们想要的特定部分



## 2.2 正则表达式的特点是：

1. 灵活性、逻辑性和功能性非常的强；

2. 可以迅速地用极简单的方式达到字符串的复杂控制。

3. 对于刚接触的人来说，比较晦涩难懂。

4.就算会写，写的过程没问题，写完再看几乎不认识。

比如：

匹配国内电话号码：\d{3}-\d{8}|\d{4}-\d{7}

验证手机号：

/^((13[0-9])|(15[^4,\D])|(18[0,5-9]))\d{8}$/

很难记住。

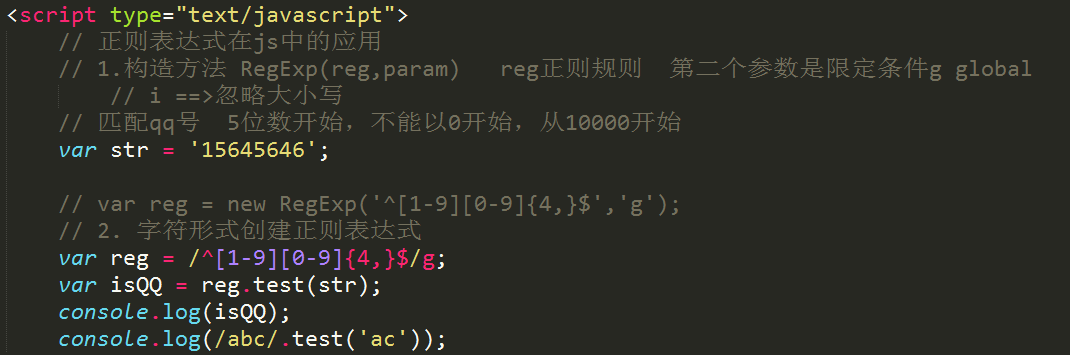
# 三、正则表达式声明

## 3.1 通过构造函数定义

var 变量名= new RegExp(/表达式/);

## 3.2 通过直接量定义

var 变量名= /表达式/;



### 3.3 正则表达式的组成是：

有一些普通字符和元字符组成，普通字符就是字母和数字，元字符具有特殊意义的字符

匹配腾讯QQ号：[1-9][0-9]{4,}

评注：腾讯QQ号从10000开始

比如 \d

预定义类: 表示数字 [0-9]

## 3.4 方法：

### 3.4.1 test()

正则对象方法，检测测试字符串是否符合该规则，返回true和false，参数（测试字符串）

使用语法：

**Boolean = reg.test(str);**

## 3.5 正则内部类

### 3.5.1 预定义类

### ^如果出现在正则规则第一位 是以\*\*开始的意思

### 如果出现在正则中间，表示取非

**. [^\n\r]** 除了换行和回车之外的任意字符（“”不行）

\d [0-9] 数字字符

\D [^0-9] 非数字字符

\s [ \t\n\x0B\f\r] 空白字符

\S [^ \t\n\x0B\f\r] 非空白字符

\w [a-zA-Z\_0-9] 单词字符 字母数字下划线（注册要求用户名）

\W [^a-zA-Z\_0-9] 非单词字符

### 3.5.2 简单类（正则：//中什么特殊符号都不写，和[]的加入）

1、/abc/.test(“abc”); 一个规则对一个字符

2、/[abc]/.test(“abcd”); 一个[\*\*\*]对一个字符

### 3.5.3 负向类（不能是其中的整体或者一部分）

中括号内，前面加个元字符^进行取反，不是括号里面的字符（一部分也不行）。

console.log(/[^abc]/.test('a'));

### 3.5.4 范围类

比如a-z,0-9

console.log(/[a-z]/.test('1234'));

### 3.5.5 组合类

用中括号匹配不同类型的单个字符。

console.log(/[a-f5-9]/.test("123"))

## 3.5.6 正则边界（重点）

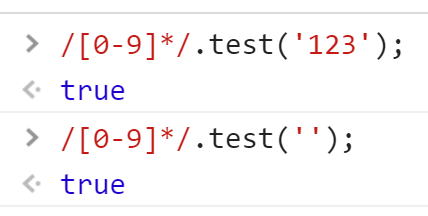
* ^ 以什么开头，$ 以什么结尾
* ^ 会匹配行或者字符串的起始位置
* 注：^在[]中才表示非！这里表示开始
* $ 会匹配行或字符串的结尾位置
* ^$在一起 表示必须是这个（精确匹配）



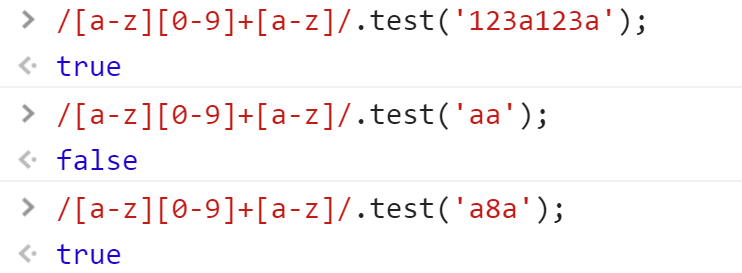
比如：

## 3.5.7 量词

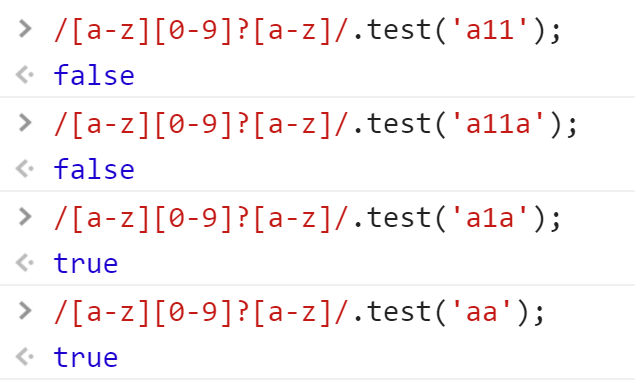
\* (贪婪) 重复零次或更多 (>=0)



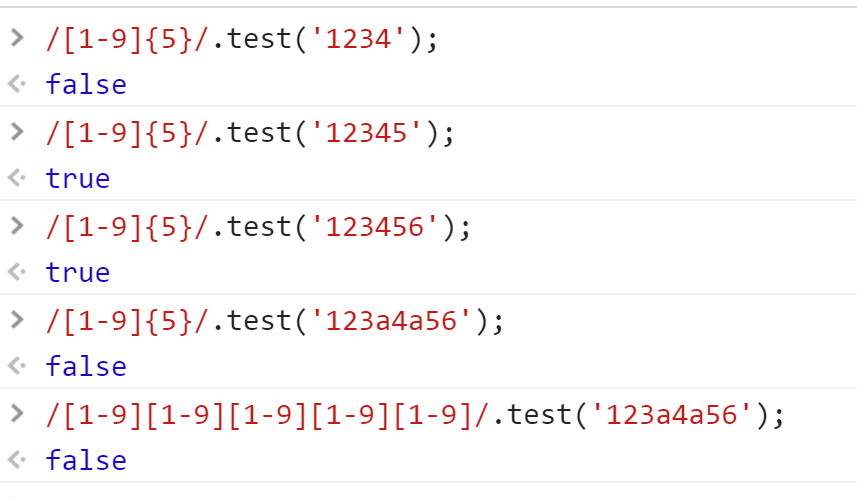
+ (懒惰) 重复一次或更多次 (>=1)



? (占有) 重复零次或一次 （0||1） 要么有 要么没有

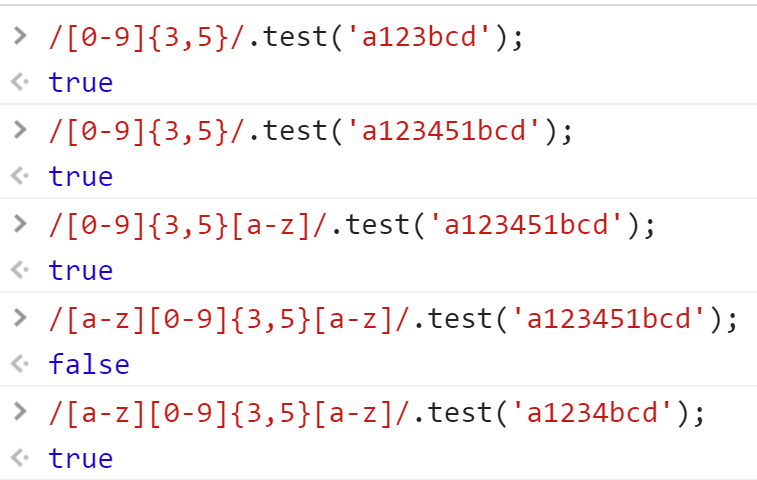


{n} n次 （x=n）



{n,} 重复n次或更多 (x>=n)

{n,m} 重复出现的次数比n多但比m少 (n<=x<=m)



\* {0,}

+ {1,}

? {0,1}

x|y 一个 | x 或者 y（没有&，用的是，代替的）

（）提高权限，有限计算



## 案例：

匹配验证表单（注册QQ）

## 3.6. replace()

replace() 方法用于在字符串中用一些字符替换另一些字符，或替换一个与正则表达式匹配的子串。

语法格式：(返回值是新字符串)

Str.replace（reg,new）;

正则表达式的匹配模式支持的2个标志

g：表示全局模式（global），即模式将被应用于所有字符串而非发现一个而停止

i：表示不区分大小写（ease-insensitive）模式，在确定匹配想时忽略模式与字符串的大小写

### 封装自己的trim 函数

function trim(str) {

return str.replace(/（^\s+）|（\s+$）/g,""); // 去掉前面和后面的空格

}

# 附件

http://www.w3school.com.cn/jsref/jsref\_obj\_regexp.asp

|  |  |
| --- | --- |
| 元字符 | 描述 |
| \ | 将下一个字符标记符、或一个向后引用、或一个八进制转义符。例如，“\\n”匹配\n。“\n”匹配换行符。序列“\\”匹配“\”而“\(”则匹配“(”。即相当于多种编程语言中都有的“转义字符”的概念。 |
| ^ | 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任意次。例如，zo\*能匹配“z”，也能匹配“zo”以及“zoo”。\*等价于o{0,} |
| + | 匹配前面的子表达式一次或多次(大于等于1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为一组，后三个o为一组。“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+”将尽可能多的匹配“o”，得到结果[“oooo”]，而“o+?”将尽可能少的匹配“o”，得到结果 ['o', 'o', 'o', 'o'] |
| .点 | 匹配除“\r\n”之外的任何单个字符。要匹配包括“\r\n”在内的任何字符，请使用像“[\s\S]”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) | 非获取匹配，匹配pattern但不获取匹配结果，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时很有用。例如“industr(?:y|ies)”就是一个比“industry|industries”更简略的表达式。 |
| (?=pattern) | 非获取匹配，正向肯定预查，在任何匹配pattern的字符串开始处匹配查找字符串，该匹配不需要获取供以后使用。例如，“Windows(?=95|98|NT|2000)”能匹配“Windows2000”中的“Windows”，但不能匹配“Windows3.1”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) | 非获取匹配，正向否定预查，在任何不匹配pattern的字符串开始处匹配查找字符串，该匹配不需要获取供以后使用。例如“Windows(?!95|98|NT|2000)”能匹配“Windows3.1”中的“Windows”，但不能匹配“Windows2000”中的“Windows”。 |
| (?<=pattern) | 非获取匹配，反向肯定预查，与正向肯定预查类似，只是方向相反。例如，“(?<=95|98|NT|2000)Windows”能匹配“2000Windows”中的“Windows”，但不能匹配“3.1Windows”中的“Windows”。 |
| (?<!pattern) | 非获取匹配，反向否定预查，与正向否定预查类似，只是方向相反。例如“(?<!95|98|NT|2000)Windows”能匹配“3.1Windows”中的“Windows”，但不能匹配“2000Windows”中的“Windows”。这个地方不正确，有问题  此处用或任意一项都不能超过2位，如“(?<!95|98|NT|20)Windows正确，“(?<!95|980|NT|20)Windows 报错，若是单独使用则无限制，如(?<!2000)Windows 正确匹配 |
| x|y | 匹配x或y。例如，“z|food”能匹配“z”或“food”(此处请谨慎)。“[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“plin”。 |
| [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。  注意:只有连字符在字符组内部时,并且出现在两个字符之间时,才能表示字符的范围; 如果出字符组的开头,则只能表示连字符本身. |
| [^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（即正则表达式的“匹配”有两种概念，一种是匹配字符，一种是匹配位置，这里的\b就是匹配位置的）。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]。grep 要加上-P，perl正则支持 |
| \D | 匹配一个非数字字符。等价于[^0-9]。grep要加上-P，perl正则支持 |
| \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 | 匹配任何不可见字符，包括空格、制表符、换页符等等。等价于[ \f\n\r\t\v]。 |
| \S | 匹配任何可见字符。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。类似但不等价于“[A-Za-z0-9\_]”，这里的"单词"字符使用Unicode字符集。 |
| \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（0-7），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（&copy;）。 |
| \p{P} | 小写 p 是 property 的意思，表示 Unicode 属性，用于 Unicode 正表达式的前缀。中括号内的“P”表示Unicode 字符集七个字符属性之一：标点字符。  其他六个属性：  L：字母；  M：标记符号（一般不会单独出现）；  Z：分隔符（比如空格、换行等）；  S：符号（比如数学符号、货币符号等）；  N：数字（比如阿拉伯数字、罗马数字等）；  C：其他字符。  *\*注：此语法部分语言不支持，例：javascript。* |
| \<  \> | 匹配词（word）的开始（\<）和结束（\>）。例如正则表达式\<the\>能够匹配字符串"for the wise"中的"the"，但是不能匹配字符串"otherwise"中的"the"。注意：这个元字符不是所有的软件都支持的。 |
| ( ) | 将( 和 ) 之间的表达式定义为“组”（group），并且将匹配这个表达式的字符保存到一个临时区域（一个正则表达式中最多可以保存9个），它们可以用 \1 到\9 的符号来引用。 |
| | | 将两个匹配条件进行逻辑“或”（Or）运算。例如正则表达式(him|her) 匹配"it belongs to him"和"it belongs to her"，但是不能匹配"it belongs to them."。注意：这个元字符不是所有的软件都支持的。 |