# JavaScript 弹窗

可以在 JavaScript 中创建三种消息框：警告框、确认框、提示框。

## 警告框

警告框经常用于确保用户可以得到某些信息。

当警告框出现后，用户需要点击确定按钮才能继续进行操作。

### 语法

window.alert("*sometext*");

**window.alert()** 方法可以不带上window对象，直接使用**alert()**方法。

## 实例

<!DOCTYPE html>  
<html>  
<head>  
<script>  
function myFunction()  
{  
    alert("你好，我是一个警告框！");  
}  
</script>  
</head>  
<body>  
  
<input type="button" onclick="myFunction()" value="显示警告框">  
  
</body>  
</html>

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=tryjs_alert)

## 确认框

确认框通常用于验证是否接受用户操作。

当确认卡弹出时，用户可以点击 "确认" 或者 "取消" 来确定用户操作。

当你点击 "确认", 确认框返回 true， 如果点击 "取消", 确认框返回 false。

### 语法

window.confirm("*sometext*");

**window.confirm()** 方法可以不带上window对象，直接使用**confirm()**方法。

## 实例

var r=confirm("按下按钮");  
if (r==true)  
{  
    x="你按下了\"确定\"按钮!";  
}  
else  
{  
    x="你按下了\"取消\"按钮!";  
}

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=tryjs_confirm)

## 提示框

提示框经常用于提示用户在进入页面前输入某个值。

当提示框出现后，用户需要输入某个值，然后点击确认或取消按钮才能继续操纵。

如果用户点击确认，那么返回值为输入的值。如果用户点击取消，那么返回值为 null。

### 语法

window.prompt("*sometext*","*defaultvalue*");

**window.prompt()** 方法可以不带上window对象，直接使用**prompt()**方法。

## 实例

var person=prompt("请输入你的名字","Harry Potter");  
if (person!=null && person!="")  
{  
    x="你好 " + person + "! 今天感觉如何?";  
    document.getElementById("demo").innerHTML=x;  
}

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=tryjs_prompt)

## 换行

弹窗使用 反斜杠 + "n"(\n) 来设置换行。

## 实例

alert("Hello\nHow are you?");

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=tryjs_alert2)

**JavaScript-打开新窗口（window.open）**

open() 方法可以查找一个已经存在或者新建的浏览器窗口。

**语法：**

window.open([URL], [窗口名称], [参数字符串])

**参数说明:**

**URL：**可选参数，在窗口中要显示网页的网址或路径。如果省略这个参数，或者它的值是空字符串，那么窗口就不显示任何文档。

**窗口名称：**可选参数，被打开窗口的名称。

1.该名称由字母、数字和下划线字符组成。

2."\_top"、"\_blank"、"\_self"具有特殊意义的名称。

  \_blank：在新窗口显示目标网页

  \_self：在当前窗口显示目标网页

  \_top：框架网页中在上部窗口中显示目标网页

3.相同 name 的窗口只能创建一个，要想创建多个窗口则 name 不能相同。

  4.name 不能包含有空格。

**参数字符串：**可选参数，设置窗口参数，各参数用逗号隔开。

**参数表:**

**[](http://img.mukewang.com/52e3677900013d6a05020261.jpg)**

例如:打开http://www.imooc.com网站，大小为300px \* 200px，无菜单，无工具栏，无状态栏，有滚动条窗口：

<script type="text/javascript"> window.open('http://www.imooc.com','\_blank','width=300,height=200,menubar=no,toolbar=no, status=no,scrollbars=yes')

</script>

**注意：**运行结果考虑浏览器兼容问题。

## JavaScript-关闭窗口（window.close）

close()关闭窗口

**用法：**

window.close(); //关闭本窗口

或

<窗口对象>.close(); //关闭指定的窗口

例如:关闭新建的窗口。

<script type="text/javascript">

var mywin=window.open('http://www.imooc.com'); //将新打的窗口对象，存储在变量mywin中

mywin.close();

</script>

**注意:上面代码在打开新窗口的同时，关闭该窗口，看不到被打开的窗口。**

## 改变 HTML 样式

**这种方式只能改变内联css**

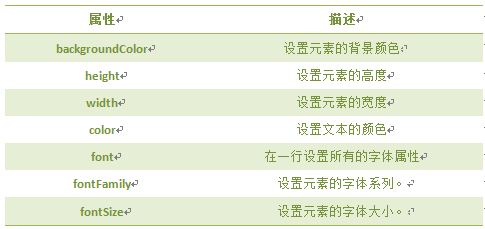
HTML DOM 允许 JavaScript 改变 HTML 元素的样式。如何改变 HTML 元素的样式呢？

**语法:**

**Object.style.property=new style;**

**注意:**Object是获取的元素对象，如通过document.getElementById("id")获取的元素。

**基本属性表（property）:**

**[](http://img.mukewang.com/52e4d4240001dd6c04850229.jpg)**

**注意:**该表只是一小部分CSS样式属性，其它样式也可以通过该方法设置和修改。

**看看下面的代码:**

改变 <p> 元素的样式，将颜色改为红色，字号改为20,背景颜色改为蓝：

<p id="pcon">Hello World!</p>

<script>

var mychar = document.getElementById("pcon");

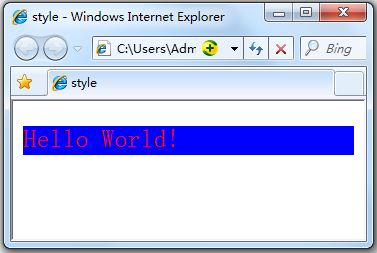
mychar.**style.color**="red";

mychar.**style.fontSize**="20";

mychar.**style.backgroundColor** ="blue";

</script>

**结果:**

[](http://img.mukewang.com/52e4d6ae000177d203770253.jpg)

## 控制类名（className 属性）

className 属性设置或返回元素的class 属性。

**语法：**

**object.className = classname**

**作用:**

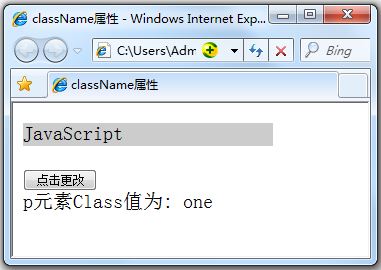
1.获取元素的class 属性

2. 为网页内的某个元素指定一个css样式来更改该元素的外观

**看看下面代码，获得 <p> 元素的 class 属性和改变className：**

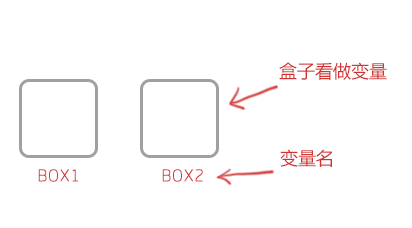
[](http://img.mukewang.com/52e4e28c0001c97f07980838.jpg)

**结果:**

**[](http://img.mukewang.com/52e4e711000135d903810270.jpg)**

## 给变量取个名字(变量命名)

我们为了区分盒子，可以用BOX1,BOX2等名称代表不同盒子，BOX1就是盒子的名字（也就是变量的名字）。

[](http://img.mukewang.com/529c05d40001140c04120228.jpg)

我们赶快给变量取个好名字吧!**变量名字可以任意取，**只不过取名字要遵循一些规则:

1.必须以字母、下划线或美元符号开头，后面可以跟字母、下划线、美元符号和数字。如下:

正确:

mysum

\_mychar

$numa1

错误:

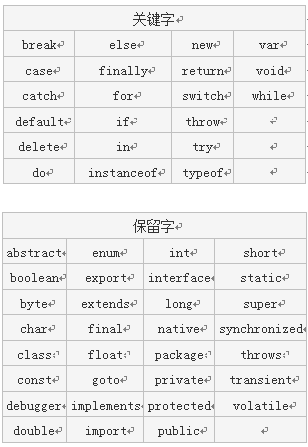
6num //开头不能用数字

%sum //开头不能用除(\_ $)外特殊符号,如(% + /等)

sum+num //开头中间不能使用除(\_ $)外特殊符号，如(% + /等)

2.变量名区分大小写，如:A与a是两个不同变量。

3.不允许使用JavaScript关键字和保留字做变量名。

[](http://img.mukewang.com/529c07c000014f5103080447.jpg)

# JavaScript Array（数组） 对象

数组对象的作用是：使用单独的变量名来存储一系列的值。

Examples

## 在线实例

创建数组, 为其赋值：

## 实例

var mycars = new Array();  
mycars[0] = "Saab";  
mycars[1] = "Volvo";  
mycars[2] = "BMW";

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=tryjs_array)

页面底部你可以找到更多的实例。

## 什么是数组?

数组对象是使用单独的变量名来存储一系列的值。

如果你有一组数据（例如：车名字），存在单独变量如下所示：

var car1="Saab";  
var car2="Volvo";  
var car3="BMW";

然而，如果你想从中找出某一辆车？并且不是3辆，而是300辆呢？这将不是一件容易的事！

最好的方法就是用数组。

数组可以用一个变量名存储所有的值，并且可以用变量名访问任何一个值。

数组中的每个元素都有自己的的ID，以便它可以很容易地被访问到。

## 创建一个数组

创建一个数组，有三种方法。

下面的代码定义了一个名为 myCars的数组对象：

1: 常规方式:

var myCars=new Array();   
myCars[0]="Saab";         
myCars[1]="Volvo";  
myCars[2]="BMW";

2: 简洁方式:

var myCars=new Array("Saab","Volvo","BMW");

3: 字面:

var myCars=["Saab","Volvo","BMW"];

## 访问数组

通过指定数组名以及索引号码，你可以访问某个特定的元素。

以下实例可以访问myCars数组的第一个值：

var name=myCars[0];

以下实例修改了数组 myCars 的第一个元素:

myCars[0]="Opel";

|  |  |
| --- | --- |
| **lamp** | [0] 是数组的第一个元素。[1] 是数组的第二个元素。 |

## 在一个数组中你可以有不同的对象

所有的JavaScript变量都是对象。数组元素是对象。函数是对象。

因此，你可以在数组中有不同的变量类型。

你可以在一个数组中包含对象元素、函数、数组：

myArray[0]=Date.now;  
myArray[1]=myFunction;  
myArray[2]=myCars;

## 数组方法和属性

使用数组对象预定义属性和方法：

var x=myCars.length             // myCars 中元素的数量  
var y=myCars.indexOf("Volvo")   // "Volvo" 值的索引值

## 完整的数组对象参考手册

你可以参考本站关于数组的所有属性和方法的完整参考手册。

参考手册包含了所有属性和方法的描述（和更多的例子）。

[完整数组对象参考手册](http://www.runoob.com/js/jsref-obj-array.html)

## 创建新方法

原型是JavaScript全局构造函数。它可以构建新Javascript对象的属性和方法。

## 实例：创建一个新的方法。

Array.prototype.myUcase=function(){  
    for (i=0;i<this.length;i++){  
        this[i]=this[i].toUpperCase();  
    }  
}

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=tryjs_prototype_array)

上面的例子创建了新的数组方法用于将数组小写字符转为大写字符。

Examples

## 更多实例

[合并两个数组 - concat()](http://www.runoob.com/try/try.php?filename=tryjsref_concat)

[合并三个数组 - concat()](http://www.runoob.com/try/try.php?filename=tryjsref_concat2)

[用数组的元素组成字符串 - join()](http://www.runoob.com/try/try.php?filename=tryjsref_join)

[删除数组的最后一个元素 - pop()](http://www.runoob.com/try/try.php?filename=tryjsref_pop)

[数组的末尾添加新的元素 - push()](http://www.runoob.com/try/try.php?filename=tryjsref_push)

[将一个数组中的元素的顺序反转排序 - reverse()](http://www.runoob.com/try/try.php?filename=tryjsref_reverse)

[删除数组的第一个元素 - shift()](http://www.runoob.com/try/try.php?filename=tryjsref_shift)

[从一个数组中选择元素 - slice()](http://www.runoob.com/try/try.php?filename=tryjsref_slice_array)

[数组排序（按字母顺序升序）- sort()](http://www.runoob.com/try/try.php?filename=tryjsref_sort)

[数字排序（按数字顺序升序）- sort()](http://www.runoob.com/try/try.php?filename=tryjsref_sort2)

[数字排序（按数字顺序降序）- sort()](http://www.runoob.com/try/try.php?filename=tryjsref_sort3)

[在数组的第2位置添加一个元素 - splice()](http://www.runoob.com/try/try.php?filename=tryjsref_splice)

[转换数组到字符串 -toString()](http://www.runoob.com/try/try.php?filename=tryjsref_tostring_array)

[在数组的开头添加新元素 - unshift()](http://www.runoob.com/try/try.php?filename=tryjsref_unshift)

[JavaScript Date（日期）对象](http://www.runoob.com/js/js-obj-date.html)

[JavaScript Boolean（布尔）对象](http://www.runoob.com/js/js-obj-boolean.html)

## 5 篇笔记

1. LongStay

  xuz\*\*\*angzz@vip.qq.com

使用数组对象结合 select 对象实现二级级联菜单。

select1 省级菜单， select2市级菜单

 在省级标签中添加事件onchange="addOption()"

<select id="a1" onchange="addOption()"></select>

<select id="a2"></select>

[**尝试一下 »**](https://c.runoob.com/codedemo/3002)

[LongStay](javascript:;)

   LongStay

  xuz\*\*\*angzz@vip.qq.com

12个月前 (06-14)

1. Mrshang110

  146\*\*\*8812@qq.com

访问数组元素和访问对象属性时,会发现他们的格式很像:

var array = [1, 2, 3];

var item = array[2]; // 访问数组元素

var person = {name: "Mrshang110", age: 24};

var name = person["name"]; // 访问对象元素

数组是特殊的对象。

对象的属性名是字符串类型的,数组可以看做属性名是数字类型的,这样就很好理解了。

数组还针对数组的特性,对对象属性的布局做了优化。

[Mrshang110](javascript:;)

   Mrshang110

  146\*\*\*8812@qq.com

12个月前 (06-14)

1. 子不语

  394\*\*\*469@qq.com

遍历一维数组: 依次访问数组中每个元素，对每个元素执行相同的操作

for(var i=0;i<arr.length;i++){

    arr[i]//当前正在遍历的元素

}

遍历关联数组: for in 循环

for(var key in hash){

    key//只是元素的下标名

    hash[key]//当前元素值

}

其实: js底层，一切对象都是关联数组:

比如: 索引数组，window等

遍历二维数组: 外层循环遍历行，内层循环遍历列

for(var r=0;r<arr.length;r++){

    for(var c=0;c<arr[r].length;c++){

        arr[r][c]//当前元素

    }

}

[子不语](javascript:;)

   子不语

  394\*\*\*469@qq.com

8个月前 (09-28)

1. 生活很黑白

  975\*\*\*049@qq.com

**Js 中为数组提供了多种遍历方式**

const ary = ['a', 'b', 'c'];

// 最基本的方式, 只能遍历下标有序递增的数组

for (let i=0; i<ary.length; i++) {

console.log(i, ary[i]); // 0 a 1 b 2 c

}

// in:每遍历一次数组指针向后移动一位, 并得到当前数组元素值的下标, 可以通过 ary[key] 来访问数组元素值

for (let key in ary) {

console.log(key, ary[key]); // 0 a 1 b 2 c

}

// of:每遍历一次数组指针向后移动一位, 并得到当前数组元素的值, 处理二维数组时极为方便

for (let value of ary) {

console.log(value); // a b c

}

**Js还为数组对象内置了多种迭代方法, 处理数组时极为方便**

/\*\*

\* forEach(function)

\*

\* 该方法接受一个函数作为参数, 该函数拥有两个参数, 分别为数组的值、键

\* 该方法没有返回值

\*/

ary.forEach((value, key) => {

console.log(key, value); // 0 a 1 b 2 c

})

/\*\*

\* map(function)

\*

\* 该方法接受一个函数作为参数, 该函数拥有两个参数, 分别为数组的值、键

\* 可以在传入的函数中返回相应的值,你每次迭代时返回的值会被map方法组装成一个新数组作为返回值返回

\*/

var result = ary.map((value, key) => {

return value + '\_runoob.com';

})

console.log(result); // ['a\_runoob.com', 'b\_runoob.com', 'c\_runoob.com']

[生活很黑白](javascript:;)

   生活很黑白

  975\*\*\*049@qq.com

5个月前 (01-08)

1. tianqixin

  429\*\*\*967@qq.com

## 数字索引与非数字索引

JavaScript 数组只支持数字索引，非数字索引指的是对象的属性了。

JavaScript 的数组可以看做是特殊的对象(object)，对象的属性有两种表示方法。

var a=Object();

a.myattr="runoob";

a["myattr"]="runoob";

**查看以下实例：**

var a=new Array();

a['a1']=1;

a['a2']=2;

a.length // 输出0

实例中由于你没有给数组的元素赋值，所以 a 数组的长度也没有被创建出来，此时 a.length 输出结果为 0。

var a=new Array();

a['a1']=1;

a['a2']=2;

a[2]=3;

a.length // 输出3

在最后一行代码中数组元素 **a[2]** 赋值了，此时 JavaScript 会给数组 a 创建一个长度为 3 的空间，并在第三个元素中赋以3，但是元素1和元素2都没有赋值，所以此时a[0]和a[1]是没有值的。

如果要使用的是一个完全由“字符串下标”组成的数组，那还是将其声明为一个 Object 类型的对象要更好一些。

var a=new Array();

a['a1']=1;

a["100"]=3;

a.length // 输出 101

Javascript 数组下标值的范围为 **0 ~ 232**。对于任意给定的数字下标值，如果不在此范围内，Javascript 会将它转换为一个字符串，并将该下标对应的值作为该数组对象的一个属性值而不是数组元素，例如 **array[-1] = "yes"** 其实就相当于给 array 对象添加了一个名为 -1 的属性，属性值为 yes。如果该下标值在合法范围内，则无论该下标值是数字还是数字字符串，都一律会被转化为数字使用，即 **array["100"] = 0** 和 **array[100] = 0** 执行的是相同的操作。

[tianqixin](javascript:;)

   tianqixin

  429\*\*\*967@qq.com

3个月前 (03-17)

# JavaScript Array 对象

## Array 对象

Array 对象用于在变量中存储多个值:

var cars = ["Saab", "Volvo", "BMW"];

第一个数组元素的索引值为 0，第二个索引值为 1，以此类推。

更多有关 JavaScript Array 内容请参考 [JavaScript Array 对象手册](http://www.runoob.com/js/js-obj-array.html)。

## 数组属性

|  |  |
| --- | --- |
| **属性** | **描述** |
| [constructor](http://www.runoob.com/jsref/jsref-constructor-array.html) | 返回创建数组对象的原型函数。 |
| [length](http://www.runoob.com/jsref/jsref-length-array.html) | 设置或返回数组元素的个数。 |
| [prototype](http://www.runoob.com/jsref/jsref-prototype-array.html) | 允许你向数组对象添加属性或方法。 |

## Array 对象属性

|  |  |
| --- | --- |
| **方法** | **描述** |
| [concat()](http://www.runoob.com/jsref/jsref-concat-array.html) | 连接两个或更多的数组，并返回结果。 |
| [copyWithin()](http://www.runoob.com/jsref/jsref-copywithin.html) | 从数组的指定位置拷贝元素到数组的另一个指定位置中。 |
| [entries()](http://www.runoob.com/jsref/jsref-entries.html) | 返回数组的可迭代对象。 |
| [every()](http://www.runoob.com/jsref/jsref-every.html) | 检测数值元素的每个元素是否都符合条件。 |
| [fill()](http://www.runoob.com/jsref/jsref-fill.html) | 使用一个固定值来填充数组。 |
| [filter()](http://www.runoob.com/jsref/jsref-filter.html) | 检测数值元素，并返回符合条件所有元素的数组。 |
| [find()](http://www.runoob.com/jsref/jsref-find.html) | 返回符合传入测试（函数）条件的数组元素。 |
|  |  |
| [findIndex()](http://www.runoob.com/jsref/jsref-findindex.html) | 返回符合传入测试（函数）条件的数组元素索引。 |
|  |  |
| [forEach()](http://www.runoob.com/jsref/jsref-foreach.html) | 数组每个元素都执行一次回调函数。 |
| [from()](http://www.runoob.com/jsref/jsref-from.html) | 通过给定的对象中创建一个数组。 |
|  |  |
| [includes()](http://www.runoob.com/jsref/jsref-includes.html) | 判断一个数组是否包含一个指定的值。 |
| [indexOf()](http://www.runoob.com/jsref/jsref-indexof-array.html) | 搜索数组中的元素，并返回它所在的位置。 |
| [isArray()](http://www.runoob.com/jsref/jsref-isarray.html) | 判断对象是否为数组。 |
| [join()](http://www.runoob.com/jsref/jsref-join.html) | 把数组的所有元素放入一个字符串。 |
| [keys()](http://www.runoob.com/jsref/jsref-keys.html) | 返回数组的可迭代对象，包含原始数组的键(key)。 |
| [lastIndexOf()](http://www.runoob.com/jsref/jsref-lastindexof-array.html) | 返回一个指定的字符串值最后出现的位置，在一个字符串中的指定位置从后向前搜索。 |
| [map()](http://www.runoob.com/jsref/jsref-map.html) | 通过指定函数处理数组的每个元素，并返回处理后的数组。 |
| [pop()](http://www.runoob.com/jsref/jsref-pop.html) | 删除数组的最后一个元素并返回删除的元素。 |
| [push()](http://www.runoob.com/jsref/jsref-push.html) | 向数组的末尾添加一个或更多元素，并返回新的长度。 |
|  |  |
| [reduce()](http://www.runoob.com/jsref/jsref-reduce.html) | 将数组元素计算为一个值（从左到右）。 |
|  |  |
| [reduceRight()](http://www.runoob.com/jsref/jsref-reduceright.html) | 将数组元素计算为一个值（从右到左）。 |
| [reverse()](http://www.runoob.com/jsref/jsref-reverse.html) | 反转数组的元素顺序。 |
| [shift()](http://www.runoob.com/jsref/jsref-shift.html) | 删除并返回数组的第一个元素。 |
| [slice()](http://www.runoob.com/jsref/jsref-slice-array.html) | 选取数组的的一部分，并返回一个新数组。 |
| [some()](http://www.runoob.com/jsref/jsref-some.html) | 检测数组元素中是否有元素符合指定条件。 |
| [sort()](http://www.runoob.com/jsref/jsref-sort.html) | 对数组的元素进行排序。 |
| [splice()](http://www.runoob.com/jsref/jsref-splice.html) | 从数组中添加或删除元素。 |
| [toString()](http://www.runoob.com/jsref/jsref-tostring-array.html) | 把数组转换为字符串，并返回结果。 |
| [unshift()](http://www.runoob.com/jsref/jsref-unshift.html) | 向数组的开头添加一个或更多元素，并返回新的长度。 |
| [valueOf()](http://www.runoob.com/jsref/jsref-valueof-array.html) | 返回数组对象的原始值。 |

# JavaScript HTML DOM EventListener

## addEventListener() 方法

## 实例

在用户点击按钮时触发监听事件：

document.getElementById("myBtn").addEventListener("click", displayDate);

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=tryjs_addeventlistener_displaydate)

addEventListener() 方法用于向指定元素添加事件句柄。

addEventListener() 方法添加的事件句柄不会覆盖已存在的事件句柄。

你可以向一个元素添加多个事件句柄。

你可以向同个元素添加多个同类型的事件句柄，如：两个 "click" 事件。

你可以向任何 DOM 对象添加事件监听，不仅仅是 HTML 元素。如： window 对象。

addEventListener() 方法可以更简单的控制事件（冒泡与捕获）。

当你使用 addEventListener() 方法时, JavaScript 从 HTML 标记中分离开来，可读性更强， 在没有控制HTML标记时也可以添加事件监听。

你可以使用 removeEventListener() 方法来移除事件的监听。

## 语法

element.addEventListener(event, function, useCapture);

第一个参数是事件的类型 (如 "click" 或 "mousedown").

第二个参数是事件触发后调用的函数。

第三个参数是个布尔值用于描述事件是冒泡还是捕获。该参数是可选的。

|  |  |
| --- | --- |
| **Note** | 注意:不要使用 "on" 前缀。 例如，使用 "click" ,而不是使用 "onclick"。 |

## 向原元素添加事件句柄

## 实例

当用户点击元素时弹出 "Hello World!" :

element.addEventListener("click", function(){ alert("Hello World!"); });

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=tryjs_addeventlistener_add)

你可以使用函数名，来引用外部函数:

## 实例

当用户点击元素时弹出 "Hello World!" :

element.addEventListener("click", myFunction);  
  
function myFunction() {  
    alert ("Hello World!");  
}

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=tryjs_addeventlistener_add2)

## 向同一个元素中添加多个事件句柄

addEventListener() 方法允许向同一个元素添加多个事件，且不会覆盖已存在的事件：

## 实例

element.addEventListener("click", myFunction);  
element.addEventListener("click", mySecondFunction);

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=tryjs_addeventlistener_add_many)

你可以向同个元素添加不同类型的事件：

## 实例

element.addEventListener("mouseover", myFunction);  
element.addEventListener("click", mySecondFunction);  
element.addEventListener("mouseout", myThirdFunction);

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=tryjs_addeventlistener_add_many2)

## 向 Window 对象添加事件句柄

addEventListener() 方法允许你在 HTML DOM 对象添加事件监听， HTML DOM 对象如： HTML 元素, HTML 文档, window 对象。或者其他支出的事件对象如: xmlHttpRequest 对象。

## 实例

当用户重置窗口大小时添加事件监听：

window.addEventListener("resize", function(){  
    document.getElementById("demo").innerHTML = sometext;  
});

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=tryjs_addeventlistener_dom)

## 传递参数

当传递参数值时，使用"匿名函数"调用带参数的函数：

## 实例

element.addEventListener("click", function(){ myFunction(p1, p2); });

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=tryjs_addeventlistener_parameters)

## 事件冒泡或事件捕获？

事件传递有两种方式：冒泡与捕获。

事件传递定义了元素事件触发的顺序。 如果你将 <p> 元素插入到 <div> 元素中，用户点击 <p> 元素, 哪个元素的 "click" 事件先被触发呢？

在 冒泡 中，内部元素的事件会先被触发，然后再触发外部元素，即： <p> 元素的点击事件先触发，然后会触发 <div> 元素的点击事件。

在 捕获 中，外部元素的事件会先被触发，然后才会触发内部元素的事件，即： <div> 元素的点击事件先触发 ，然后再触发 <p> 元素的点击事件。

addEventListener() 方法可以指定 "useCapture" 参数来设置传递类型：

addEventListener(event, function, **useCapture**);

默认值为 false, 即冒泡传递，当值为 true 时, 事件使用捕获传递。

## 实例

document.getElementById("myDiv").addEventListener("click", myFunction, true);

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=tryjs_addeventlistener_usecapture)

## removeEventListener() 方法

removeEventListener() 方法移除由 addEventListener() 方法添加的事件句柄:

## 实例

element.removeEventListener("mousemove", myFunction);

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=tryjs_addeventlistener_remove)

## 浏览器支持

表格中的数字表示支持该方法的第一个浏览器的版本号。

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 方法 |  |  |  |  |  |
| addEventListener() | 1.0 | 9.0 | 1.0 | 1.0 | 7.0 |
| removeEventListener() | 1.0 | 9.0 | 1.0 | 1.0 | 7.0 |

**注意：** IE 8 及更早 IE 版本，Opera 7.0及其更早版本不支持 addEventListener() 和 removeEventListener() 方法。但是，对于这类浏览器版本可以使用 detachEvent() 方法来移除事件句柄:

element.attachEvent(event, function);element.detachEvent(event, function);

## 实例

跨浏览器解决方法:

var x = document.getElementById("myBtn");  
if (x.addEventListener) {                    // 所有主流浏览器，除了 IE 8 及更早版本  
    x.addEventListener("click", myFunction);  
} else if (x.attachEvent) {                  // IE 8 及更早版本  
    x.attachEvent("onclick", myFunction);  
}

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=tryjs_addeventlistener_crossbrowser)

## HTML DOM 事件对象参考手册

所有 HTML DOM 事件，可以查看我们完整的[HTML DOM Event 对象参考手册](http://www.runoob.com/jsref/dom-obj-event.html)。

# JavaScript 对象

JavaScript 对象是拥有属性和方法的数据。

## 真实生活中的对象，属性和方法

真实生活中，一辆汽车是一个对象。

对象有它的属性，如重量和颜色等，方法有启动停止等:

|  |  |  |
| --- | --- | --- |
| **对象** | **属性** | **方法** |
| http://www.runoob.com/images/objectExplained.gif | car.name = Fiat  car.model = 500  car.weight = 850kg  car.color = white | car.start()  car.drive()  car.brake()   car.stop() |

所有汽车都有这些属性，但是每款车的属性都不尽相同。

所有汽车都拥有这些方法，但是它们被执行的时间都不尽相同。

## JavaScript 对象

在 JavaScript中，几乎所有的事物都是对象。

|  |  |
| --- | --- |
| **Note** | 在 JavaScript 中，对象是非常重要的，当你理解了对象，就可以了解 JavaScript 。 |

你已经学习了 JavaScript 变量的赋值。

以下代码为变量 **car** 设置值为 "Fiat" :

var car = "Fiat";

对象也是一个变量，但对象可以包含多个值（多个变量）。

var car = {type:"Fiat", model:500, color:"white"};

在以上实例中，3 个值 ("Fiat", 500, "white") 赋予变量 car。

在以上实例中，3 个变量 (type, model, color) 赋予变量 car。

|  |  |
| --- | --- |
| **Note** | JavaScript 对象是变量的容器。 |

## 对象定义

你可以使用字符来定义和创建 JavaScript 对象:

## 实例

var person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};

[**尝试一下 »**](http://www.runoob.com/try/tryit.php?filename=tryjs_object_create_1)

定义 JavaScript 对象可以跨越多行，空格跟换行不是必须的：

## 实例

var person = {  
    firstName:"John",  
    lastName:"Doe",  
    age:50,  
    eyeColor:"blue"  
};

[**尝试一下 »**](http://www.runoob.com/try/tryit.php?filename=tryjs_object_create_2)

## 对象属性

可以说 "JavaScript 对象是变量的容器"。

但是，我们通常认为 "JavaScript 对象是键值对的容器"。

键值对通常写法为 **name : value** (键与值以冒号分割)。

键值对在 JavaScript 对象通常称为 **对象属性**。

|  |  |
| --- | --- |
| **Note** | JavaScript 对象是属性变量的容器。 |

对象键值对的写法类似于：

* PHP 中的关联数组
* Python 中的字典
* C 语言中的哈希表
* Java 中的哈希映射
* Ruby 和 Perl 中的哈希表

## 访问对象属性

你可以通过两种方式访问对象属性:

## 实例 1

person.lastName;

[**尝试一下 »**](http://www.runoob.com/try/tryit.php?filename=tryjs_object_properties_1)

## 实例 2

person["lastName"];

[**尝试一下 »**](http://www.runoob.com/try/tryit.php?filename=tryjs_object_properties_2)

## 对象方法

对象的方法定义了一个函数，并作为对象的属性存储。

对象方法通过添加 () 调用 (作为一个函数)。

该实例访问了 person 对象的 fullName() 方法:

## 实例

name = person.fullName();

[**尝试一下 »**](http://www.runoob.com/try/tryit.php?filename=tryjs_object_method)

如果你要访问 person 对象的 fullName 属性，它将作为一个定义函数的字符串返回：

## 实例

name = person.fullName;

[**尝试一下 »**](http://www.runoob.com/try/tryit.php?filename=tryjs_object_function)

|  |  |
| --- | --- |
| **Note** | JavaScript 对象是属性和方法的容器。 |

在随后的教程中你将学习到更多关于函数，属性和方法的知识。

## 访问对象方法

你可以使用以下语法创建对象方法：

*methodName : function() { code lines }*

你可以使用以下语法访问对象方法：

*objectName.methodName()*

通常 fullName() 是作为 person 对象的一个方法， fullName 是作为一个属性。

有多种方式可以创建，使用和修改 JavaScript 对象。

同样也有多种方式用来创建，使用和修改属性和方法。

|  |  |
| --- | --- |
| **Note** | 在随后的教程中，你将学习到更多关于对象的知识。 |

## 更多实例

[创建 JavaScript 对象 I](http://www.runoob.com/try/tryit.php?filename=tryjs_object_create_1)

[创建 JavaScript 对象 II](http://www.runoob.com/try/tryit.php?filename=tryjs_object_create_2)

[访问对象属性 I](http://www.runoob.com/try/tryit.php?filename=tryjs_object_properties_1)

[访问对象属性 II](http://www.runoob.com/try/tryit.php?filename=tryjs_object_properties_2)

[函数属性作为一个方法访问](http://www.runoob.com/try/tryit.php?filename=tryjs_object_method)

[函数属性作为一个属性访问](http://www.runoob.com/try/tryit.php?filename=tryjs_object_function)

# JavaScript 对象

JavaScript 中的所有事物都是对象：字符串、数值、数组、函数...

此外，JavaScript 允许自定义对象。

## 所有事物都是对象

JavaScript 提供多个内建对象，比如 String、Date、Array 等等。 对象只是带有属性和方法的特殊数据类型。

* 布尔型可以是一个对象。
* 数字型可以是一个对象。
* 字符串也可以是一个对象
* 日期是一个对象
* 数学和正则表达式也是对象
* 数组是一个对象
* 甚至函数也可以是对象

## JavaScript 对象

对象只是一种特殊的数据。对象拥有**属性**和**方法**。

## 访问对象的属性

属性是与对象相关的值。

访问对象属性的语法是：

*objectName.propertyName*

这个例子使用了 String 对象的 length 属性来获得字符串的长度：

var message="Hello World!";  
var x=message.length;

在以上代码执行后，x 的值将是：

12

## 访问对象的方法

方法是能够在对象上执行的动作。

您可以通过以下语法来调用方法：

*objectName.methodName()*

这个例子使用了 String 对象的 toUpperCase() 方法来将文本转换为大写：

var message="Hello world!";  
var x=message.toUpperCase();

在以上代码执行后，x 的值将是：

HELLO WORLD!

## 创建 JavaScript 对象

通过 JavaScript，您能够定义并创建自己的对象。

创建新对象有两种不同的方法：

* 定义并创建对象的实例
* 使用函数来定义对象，然后创建新的对象实例

## 创建直接的实例

这个例子创建了对象的一个新实例，并向其添加了四个属性：

## 实例

person=new Object();  
person.firstname="John";  
person.lastname="Doe";  
person.age=50;  
person.eyecolor="blue";

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=tryjs_create_object)

替代语法（使用对象 literals）：

## 实例

person={firstname:"John",lastname:"Doe",age:50,eyecolor:"blue"};

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=tryjs_create_object1)

## 使用对象构造器

本例使用函数来构造对象：

## 实例

function person(firstname,lastname,age,eyecolor) { this.firstname=firstname; this.lastname=lastname; this.age=age; this.eyecolor=eyecolor; }

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=tryjs_create_object2)

在JavaScript中，this通常指向的是我们正在执行的函数本身，或者是指向该函数所属的对象（运行时）

## 创建 JavaScript 对象实例

一旦您有了对象构造器，就可以创建新的对象实例，就像这样：

var myFather=new person("John","Doe",50,"blue");  
var myMother=new person("Sally","Rally",48,"green");

## 把属性添加到 JavaScript 对象

您可以通过为对象赋值，向已有对象添加新属性：

假设 personObj 已存在 - 您可以为其添加这些新属性：firstname、lastname、age 以及 eyecolor：

person.firstname="John";  
person.lastname="Doe";  
person.age=30;  
person.eyecolor="blue";  
  
x=person.firstname;

T在以上代码执行后，x 的值将是：

John

## 把方法添加到 JavaScript 对象

方法只不过是附加在对象上的函数。

在构造器函数内部定义对象的方法：

function person(firstname,lastname,age,eyecolor)

{

    this.firstname=firstname;

    this.lastname=lastname;

    this.age=age;

    this.eyecolor=eyecolor;

    this.changeName=changeName;

    function changeName(name)

    {

        this.lastname=name;

    }

}

changeName() 函数 name 的值赋给 person 的 lastname 属性。

## 现在您可以试一下：

myMother.changeName("Doe");

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=tryjs_create_object3)

## JavaScript 类

JavaScript 是面向对象的语言，但 JavaScript 不使用类。

在 JavaScript 中，不会创建类，也不会通过类来创建对象（就像在其他面向对象的语言中那样）。

JavaScript 基于 prototype，而不是基于类的。

## JavaScript for...in 循环

JavaScript for...in 语句循环遍历对象的属性。

### 语法

for (variable in object)

{

    执行的代码……

}

**注意：**for...in 循环中的代码块将针对每个属性执行一次。

### 实例

循环遍历对象的属性：

## 实例

var person={fname:"John",lname:"Doe",age:25}; for (x in person) { txt=txt + person[x]; }

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=tryjs_object_for_in)

# JavaScript typeof, null, 和 undefined

JavaScript typeof, null, undefined, valueOf()。

## typeof 操作符

你可以使用 typeof 操作符来检测变量的数据类型。

## 实例

typeof "John"                // 返回 string   
typeof 3.14                  // 返回 number  
typeof false                 // 返回 boolean  
typeof [1,2,3,4]             // 返回 object  
typeof {name:'John', age:34} // 返回 object

[**尝试一下 »**](http://www.runoob.com/try/tryit.php?filename=tryjs_typeof_typeof)

|  |  |
| --- | --- |
| **Note** | 在JavaScript中，数组是一种特殊的对象类型。 因此 typeof [1,2,3,4] 返回 object。 |

## null

在 JavaScript 中 null 表示 "什么都没有"。

null是一个只有一个值的特殊类型。表示一个空对象引用。

|  |  |
| --- | --- |
| **Note** | 用 typeof 检测 null 返回是object。 |

你可以设置为 null 来清空对象:

## 实例

var person = null;           // 值为 null(空), 但类型为对象

[**尝试一下 »**](http://www.runoob.com/try/tryit.php?filename=tryjs_typeof_null)

你可以设置为 undefined 来清空对象:

## 实例

var person = undefined;     // 值为 undefined, 类型为 undefined

[**尝试一下 »**](http://www.runoob.com/try/tryit.php?filename=tryjs_typeof_undefined_1)

## undefined

在 JavaScript 中, **undefined** 是一个没有设置值的变量。

**typeof** 一个没有值的变量会返回 **undefined**。

## 实例

var person;                  // 值为 undefined(空), 类型是undefined

[**尝试一下 »**](http://www.runoob.com/try/tryit.php?filename=tryjs_typeof_undefined)

任何变量都可以通过设置值为 **undefined** 来清空。 类型为 **undefined**.

## 实例

person = undefined;          // 值为 undefined, 类型是undefined

[**尝试一下 »**](http://www.runoob.com/try/tryit.php?filename=tryjs_typeof_undefined_2)

## undefined 和 null 的区别

## 实例

null 和 undefined 的值相等，但类型不等：

typeof undefined             // undefined  
typeof null                  // object  
null === undefined           // false  
null == undefined            // true

[**尝试一下 »**](http://www.runoob.com/try/tryit.php?filename=tryjs_typeof_undefined_3)

**JavaScript 类型转换**

Number() 转换为数字， String() 转换为字符串， Boolean() 转化为布尔值。

**JavaScript 数据类型**

在 JavaScript 中有 5 种不同的数据类型：

* string
* number
* boolean
* object
* function

3 种对象类型：

* Object
* Date
* Array

2 个不包含任何值的数据类型：

* null
* undefined

**typeof 操作符**

你可以使用 **typeof** 操作符来查看 JavaScript 变量的数据类型。

**实例**

typeof "John"                 // 返回 string   
typeof 3.14                   // 返回 number  
typeof NaN                    // 返回 number  
typeof false                  // 返回 boolean  
typeof [1,2,3,4]              // 返回 object  
typeof {name:'John', age:34}  // 返回 object  
typeof new Date()             // 返回 object  
typeof function () {}         // 返回 function  
typeof myCar                  // 返回 undefined (如果 myCar 没有声明)  
typeof null                   // 返回 object

[**尝试一下 »**](http://www.runoob.com/try/tryit.php?filename=tryjs_typeof_all)

请注意：

* NaN 的数据类型是 number
* 数组(Array)的数据类型是 object
* 日期(Date)的数据类型为 object
* null 的数据类型是 object
* 未定义变量的数据类型为 undefined

如果对象是 JavaScript Array 或 JavaScript Date ，我们就无法通过 **typeof** 来判断他们的类型，因为都是 返回 Object。

**constructor 属性**

**constructor** 属性返回所有 JavaScript 变量的构造函数。

**实例**

"John".constructor                 // 返回函数 String()  { [native code] }  
(3.14).constructor                 // 返回函数 Number()  { [native code] }  
false.constructor                  // 返回函数 Boolean() { [native code] }  
[1,2,3,4].constructor              // 返回函数 Array()   { [native code] }  
{name:'John', age:34}.constructor  // 返回函数 Object()  { [native code] }  
new Date().constructor             // 返回函数 Date()    { [native code] }  
function () {}.constructor         // 返回函数 Function(){ [native code] }

[**尝试一下 »**](http://www.runoob.com/try/tryit.php?filename=tryjs_constructor_all)

你可以使用 constructor 属性来查看对象是否为数组 (包含字符串 "Array"):

**实例**

function isArray(myArray) {  
    return myArray.constructor.toString().indexOf("Array") > -1;  
}

[**尝试一下 »**](http://www.runoob.com/try/tryit.php?filename=tryjs_array_isarray)

你可以使用 constructor 属性来查看对象是否为日期 (包含字符串 "Date"):

**实例**

function isDate(myDate) {  
    return myDate.constructor.toString().indexOf("Date") > -1;  
}

[**尝试一下 »**](http://www.runoob.com/try/tryit.php?filename=tryjs_date_isdate)

**JavaScript 类型转换**

JavaScript 变量可以转换为新变量或其他数据类型：

* 通过使用 JavaScript 函数
* 通过 JavaScript 自身自动转换

**将数字转换为字符串**

全局方法 **String()** 可以将数字转换为字符串。

该方法可用于任何类型的数字，字母，变量，表达式：

**实例**

String(x)         // 将变量 x 转换为字符串并返回  
String(123)       // 将数字 123 转换为字符串并返回  
String(100 + 23)  // 将数字表达式转换为字符串并返回

[**尝试一下 »**](http://www.runoob.com/try/tryit.php?filename=tryjs_number_string)

Number 方法 **toString()** 也是有同样的效果。

**实例**

x.toString()  
(123).toString()  
(100 + 23).toString()

[**尝试一下 »**](http://www.runoob.com/try/tryit.php?filename=tryjs_number_tostring)

在 [Number 方法](http://www.runoob.com/jsref/jsref-obj-number.html) 章节中，你可以找到更多数字转换为字符串的方法：

|  |  |
| --- | --- |
| **方法** | **描述** |
| toExponential() | 把对象的值转换为指数计数法。 |
| toFixed() | 把数字转换为字符串，结果的小数点后有指定位数的数字。 |
| toPrecision() | 把数字格式化为指定的长度。 |

**将布尔值转换为字符串**

全局方法 **String()** 可以将布尔值转换为字符串。

String(false)        // 返回 "false"  
String(true)         // 返回 "true"

Boolean 方法 **toString()** 也有相同的效果。

false.toString()     // 返回 "false"  
true.toString()      // 返回 "true"

**将日期转换为字符串**

Date() 返回字符串。

Date()      // 返回 Thu Jul 17 2014 15:38:19 GMT+0200 (W. Europe Daylight Time)

全局方法 String() 可以将日期对象转换为字符串。

String(new Date())      // 返回 Thu Jul 17 2014 15:38:19 GMT+0200 (W. Europe Daylight Time)

Date 方法 **toString()** 也有相同的效果。

**实例**

obj = new Date()  
obj.toString()   // 返回 Thu Jul 17 2014 15:38:19 GMT+0200 (W. Europe Daylight Time)

在 [Date 方法](http://www.runoob.com/jsref/jsref-obj-date.html) 章节中，你可以查看更多关于日期转换为字符串的函数：

|  |  |
| --- | --- |
| **方法** | **描述** |
| getDate() | 从 Date 对象返回一个月中的某一天 (1 ~ 31)。 |
| getDay() | 从 Date 对象返回一周中的某一天 (0 ~ 6)。 |
| getFullYear() | 从 Date 对象以四位数字返回年份。 |
| getHours() | 返回 Date 对象的小时 (0 ~ 23)。 |
| getMilliseconds() | 返回 Date 对象的毫秒(0 ~ 999)。 |
| getMinutes() | 返回 Date 对象的分钟 (0 ~ 59)。 |
| getMonth() | 从 Date 对象返回月份 (0 ~ 11)。 |
| getSeconds() | 返回 Date 对象的秒数 (0 ~ 59)。 |
| getTime() | 返回 1970 年 1 月 1 日至今的毫秒数。 |

**将字符串转换为数字**

全局方法 **Number()** 可以将字符串转换为数字。

字符串包含数字(如 "3.14") 转换为数字 (如 3.14).

空字符串转换为 0。

其他的字符串会转换为 NaN (不是个数字)。

Number("3.14")    // 返回 3.14  
Number(" ")       // 返回 0   
Number("")        // 返回 0  
Number("99 88")   // 返回 NaN

在 [Number 方法](http://www.runoob.com/jsref/jsref-obj-number.html) 章节中，你可以查看到更多关于字符串转为数字的方法：

|  |  |
| --- | --- |
| **方法** | **描述** |
| parseFloat() | 解析一个字符串，并返回一个浮点数。 |
| parseInt() | 解析一个字符串，并返回一个整数。 |

**一元运算符 +**

**Operator +**可用于将变量转换为数字：

**实例**

var y = "5";      // y 是一个字符串  
var x = + y;      // x 是一个数字

[**尝试一下 »**](http://www.runoob.com/try/tryit.php?filename=tryjs_typeof3)

如果变量不能转换，它仍然会是一个数字，但值为 NaN (不是一个数字):

**实例**

var y = "John";   // y 是一个字符串  
var x = + y;      // x 是一个数字 (NaN)

[**尝试一下 »**](http://www.runoob.com/try/tryit.php?filename=tryjs_typeof4)

**将布尔值转换为数字**

全局方法 **Number()** 可将布尔值转换为数字。

Number(false)     // 返回 0  
Number(true)      // 返回 1

**将日期转换为数字**

全局方法 **Number()** 可将日期转换为数字。

d = new Date();  
Number(d)          // 返回 1404568027739

日期方法 **getTime()** 也有相同的效果。

d = new Date();  
d.getTime()        // 返回 1404568027739

**自动转换类型**

当 JavaScript 尝试操作一个 "错误" 的数据类型时，会自动转换为 "正确" 的数据类型。

以下输出结果不是你所期望的：

5 + null    // 返回 5         null 转换为 0  
"5" + null  // 返回"5null"   null 转换为 "null"  
"5" + 1     // 返回 "51"      1 转换为 "1"    
"5" - 1     // 返回 4         "5" 转换为 5

**自动转换为字符串**

当你尝试输出一个对象或一个变量时 JavaScript 会自动调用变量的 toString() 方法：

document.getElementById("demo").innerHTML = myVar;  
  
// if myVar = {name:"Fjohn"}  // toString 转换为 "[object Object]"  
// if myVar = [1,2,3,4]       // toString 转换为 "1,2,3,4"  
// if myVar = new Date()      // toString 转换为 "Fri Jul 18 2014 09:08:55 GMT+0200"

数字和布尔值也经常相互转换：

// if myVar = 123             // toString 转换为 "123"  
// if myVar = true            // toString 转换为 "true"  
// if myVar = false           // toString 转换为 "false"

[JavaScript typeof](http://www.runoob.com/js/js-typeof.html)

[JavaScript 正则表达式](http://www.runoob.com/js/js-regexp.html)

**1 篇笔记**

1. 夜尽

  972\*\*\*483@qq.com

**检测数据类型：typeof 与 instanceof**

**typeof**

typeof 用以获取一个变量或者表达式的类型，typeof 一般只能返回如下几个结果：

number,boolean,string,function（函数）,object（NULL,数组，对象）,undefined。

实例：

document.getElementById("demo").innerHTML =

typeof "john" + "<br>" +

typeof 3.14 + "<br>" +

typeof false + "<br>" +

typeof [1,2,3,4] + "<br>" +

typeof {name:'john', age:34};

[**尝试一下 »**](https://c.runoob.com/codedemo/96)

我们可以使用 typeof 来获取一个变量是否存在，如 **if(typeof a!="undefined"){}**，而不要去使用 if(a) 因为如果 a 不存在（未声明）则会出错。

正因为 typeof 遇到 null,数组,对象时都会返回 object 类型，所以当我们要判断一个对象是否是数组时。

或者判断某个变量是否是某个对象的实例则要选择使用另一个关键语法 **instanceof**

。

**instanceof**

可通过 instanceof 操作符来判断对象的具体类型，语法格式:

var result = objectName instanceof objectType

返回布尔值，如果是指定类型返回 true，否则返回 false：

例：

arr = [1,2,3];

if(arr instanceof Array){

document.write("arr 是一个数组");

} else {

document.write("arr 不是一个数组");

}

[**尝试一下 »**](https://c.runoob.com/codedemo/4505)

[夜尽](javascript:;)

   夜尽

  972\*\*\*483@qq.com

7个月前 (11-16)

# JavaScript 正则表达式

正则表达式（英语：Regular Expression，在代码中常简写为regex、regexp或RE）使用单个字符串来描述、匹配一系列符合某个句法规则的字符串搜索模式。

搜索模式可用于文本搜索和文本替换。

## 什么是正则表达式？

正则表达式是由一个字符序列形成的搜索模式。

当你在文本中搜索数据时，你可以用搜索模式来描述你要查询的内容。

正则表达式可以是一个简单的字符，或一个更复杂的模式。

正则表达式可用于所有文本搜索和文本替换的操作。

## 语法

/正则表达式主体/修饰符(可选)

其中修饰符是可选的。

## 实例：

var patt = /runoob/i

实例解析：

**/runoob/i**  是一个正则表达式。

**runoob**  是一个**正则表达式主体** (用于检索)。

**i**  是一个**修饰符** (搜索不区分大小写)。

## 使用字符串方法

在 JavaScript 中，正则表达式通常用于两个字符串方法 : search() 和 replace()。

**search() 方法** 用于检索字符串中指定的子字符串，或检索与正则表达式相匹配的子字符串，并返回子串的起始位置。

**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。

## search() 方法使用正则表达式

## 实例

使用正则表达式搜索 "Runoob" 字符串，且不区分大小写：

var str = "Visit Runoob!"; var n = str.search(/Runoob/i);

输出结果为：

6

[**尝试一下 »**](http://www.runoob.com/try/tryit.php?filename=tryjs_string_search_regexp)

## search() 方法使用字符串

search 方法可使用字符串作为参数。字符串参数会转换为正则表达式：

## 实例

检索字符串中 "Runoob" 的子串：

var str = "Visit Runoob!"; var n = str.search("Runoob");

[**尝试一下 »**](http://www.runoob.com/try/tryit.php?filename=tryjs_string_search)

## replace() 方法使用正则表达式

## 实例

使用正则表达式且不区分大小写将字符串中的 Microsoft 替换为 Runoob :

var str = document.getElementById("demo").innerHTML; var txt = str.replace(/microsoft/i,"Runoob");

结果输出为:

Visit Runoob!

[**尝试一下 »**](http://www.runoob.com/try/tryit.php?filename=tryjs_string_replace_regexp)

## replace() 方法使用字符串

replace() 方法将接收字符串作为参数：

var str = document.getElementById("demo").innerHTML; var txt = str.replace("Microsoft","Runoob");

[**尝试一下 »**](http://www.runoob.com/try/tryit.php?filename=tryjs_string_replace)

## 你注意到了吗？

|  |  |
| --- | --- |
| **Note** | 正则表达式参数可用在以上方法中 (替代字符串参数)。 正则表达式使得搜索功能更加强大(如实例中不区分大小写)。 |

## 正则表达式修饰符

**修饰符** 可以在全局搜索中不区分大小写:

|  |  |
| --- | --- |
| **修饰符** | **描述** |
| i | 执行对大小写不敏感的匹配。 |
| g | 执行全局匹配（查找所有匹配而非在找到第一个匹配后停止）。 |
| m | 执行多行匹配。 |

## 正则表达式模式

方括号用于查找某个范围内的字符：

|  |  |
| --- | --- |
| **表达式** | **描述** |
| [abc] | 查找方括号之间的任何字符。 |
| [0-9] | 查找任何从 0 至 9 的数字。 |
| (x|y) | 查找任何以 | 分隔的选项。 |

元字符是拥有特殊含义的字符：

|  |  |
| --- | --- |
| **元字符** | **描述** |
| \d | 查找数字。 |
| \s | 查找空白字符。 |
| \b | 匹配单词边界。 |
| \uxxxx | 查找以十六进制数 xxxx 规定的 Unicode 字符。 |

量词:

|  |  |
| --- | --- |
| **量词** | **描述** |
| n+ | 匹配任何包含至少一个 n 的字符串。 |
| n\* | 匹配任何包含零个或多个 n 的字符串。 |
| n? | 匹配任何包含零个或一个 n 的字符串。 |

## 使用 RegExp 对象

在 JavaScript 中，RegExp 对象是一个预定义了属性和方法的正则表达式对象。

## 使用 test()

test() 方法是一个正则表达式方法。

test() 方法用于检测一个字符串是否匹配某个模式，如果字符串中含有匹配的文本，则返回 true，否则返回 false。

以下实例用于搜索字符串中的字符 "e"：

## 实例

var patt = /e/;  
patt.test("The best things in life are free!");

字符串中含有 "e"，所以该实例输出为：

true

[**尝试一下 »**](http://www.runoob.com/try/tryit.php?filename=tryjs_regexp_test)

你可以不用设置正则表达式的变量，以上两行代码可以合并为一行：

/e/.test("The best things in life are free!")

## 使用 exec()

exec() 方法是一个正则表达式方法。

exec() 方法用于检索字符串中的正则表达式的匹配。

该函数返回一个数组，其中存放匹配的结果。如果未找到匹配，则返回值为 null。

以下实例用于搜索字符串中的字母 "e":

## Example 1

/e/.exec("The best things in life are free!");

字符串中含有 "e"，所以该实例输出为:

e

[**尝试一下 »**](http://www.runoob.com/try/tryit.php?filename=tryjs_regexp_exec)

## 更多实例

* [JS 判断输入字符串是否为数字、字母、下划线组成](https://c.runoob.com/codedemo/3527)
* [JS 判断输入字符串是否全部为字母](https://c.runoob.com/codedemo/3526)
* [JS 判断输入字符串是否全部为数字](https://c.runoob.com/codedemo/3525)

## 完整的 RegExp 参考手册

完整的 RegExp 对象参考手册，请参考我们的 [JavaScript RegExp 参考手册](http://www.runoob.com/jsref/jsref-obj-regexp.html)。

该参考手册包含了所有 RegExp 对象的方法和属性。

[JavaScript 类型转换](http://www.runoob.com/js/js-type-conversion.html)

[JavaScript 错误 – Throw、Try 和 Catch](http://www.runoob.com/js/js-errors.html)

## 1 篇笔记

1. qq1056125478

  105\*\*\*5478@qq.com

正则表达式表单验证实例：

/\*是否带有小数\*/

function isDecimal(strValue ) {

var objRegExp= /^\d+\.\d+$/;

return objRegExp.test(strValue);

}

/\*校验是否中文名称组成 \*/

function ischina(str) {

    var reg=/^[\u4E00-\u9FA5]{2,4}$/; /\*定义验证表达式\*/

    return reg.test(str); /\*进行验证\*/

}

/\*校验是否全由8位数字组成 \*/

function isStudentNo(str) {

    var reg=/^[0-9]{8}$/; /\*定义验证表达式\*/

    return reg.test(str); /\*进行验证\*/

}

/\*校验电话码格式 \*/

function isTelCode(str) {

    var reg= /^((0\d{2,3}-\d{7,8})|(1[3584]\d{9}))$/;

    return reg.test(str);

}

/\*校验邮件地址是否合法 \*/

function IsEmail(str) {

    var reg=/^\w+@[a-zA-Z0-9]{2,10}(?:\.[a-z]{2,4}){1,3}$/;

    return reg.test(str);

}

[**尝试一下 »**](https://c.runoob.com/codedemo/2873)

[qq1056125478](javascript:;)

   qq1056125478

  105\*\*\*5478@qq.com

1年前 (2017-04-03)

# JavaScript Date（日期） 对象

日期对象用于处理日期和时间。

Examples

## 在线实例

[如何使用 Date() 方法获得当日的日期。](http://www.runoob.com/try/try.php?filename=tryjsref_date)

[getFullYear()](http://www.runoob.com/try/try.php?filename=tryjsref_getfullyear)  
使用 getFullYear() 获取年份。

[getTime()](http://www.runoob.com/try/try.php?filename=tryjsref_gettime)  
getTime() 返回从 1970 年 1 月 1 日至今的毫秒数。

[setFullYear()](http://www.runoob.com/try/try.php?filename=tryjsref_setfullyear2)  
如何使用 setFullYear() 设置具体的日期。

[toUTCString()](http://www.runoob.com/try/try.php?filename=tryjsref_toutcstring)  
如何使用 toUTCString() 将当日的日期（根据 UTC）转换为字符串。

[getDay()](http://www.runoob.com/try/try.php?filename=tryjsref_date_weekday)  
如何使用 getDay() 和数组来显示星期，而不仅仅是数字。

[Display a clock](http://www.runoob.com/try/try.php?filename=tryjs_timing_clock)  
如何在网页上显示一个钟表。

## 完整的 Date 对象参考手册

我们提供 JavaScript Date 对象参考手册，其中包括所有可用于日期对象的属性和方法。[JavaScript Date 对象参考手册](http://www.runoob.com/js/jsref-obj-date.html)。

该手册包含了对每个属性和方法的详细描述以及相关实例。

## 创建日期

Date 对象用于处理日期和时间。

可以通过 new 关键词来定义 Date 对象。以下代码定义了名为 myDate 的 Date 对象：

有四种方式初始化日期:

new Date() // 当前日期和时间  
new Date(milliseconds) //返回从 1970 年 1 月 1 日至今的毫秒数  
new Date(dateString)  
new Date(year, month, day, hours, minutes, seconds, milliseconds)

上面的参数大多数都是可选的，在不指定的情况下，默认参数是0。

<="" p="" style="color: rgb(51, 51, 51); font-family: "Helvetica Neue", Helvetica, "PingFang SC", "Hiragino Sans GB", "Microsoft YaHei", "Noto Sans CJK SC", "WenQuanYi Micro Hei", Arial, sans-serif; font-size: 12px; font-style: normal; font-variant-ligatures: normal; font-variant-caps: normal; font-weight: 400; letter-spacing: normal; orphans: 2; text-align: start; text-indent: 0px; text-transform: none; white-space: normal; widows: 2; word-spacing: 0px; -webkit-text-stroke-width: 0px; background-color: rgb(255, 255, 255); text-decoration-style: initial; text-decoration-color: initial;">

实例化一个日期的一些例子：

var today = new Date()  
var d1 = new Date("October 13, 1975 11:13:00")  
var d2 = new Date(79,5,24)  
var d3 = new Date(79,5,24,11,33,0)

## 设置日期

通过使用针对日期对象的方法，我们可以很容易地对日期进行操作。

在下面的例子中，我们为日期对象设置了一个特定的日期 (2010 年 1 月 14 日)：

var myDate=new Date();  
myDate.setFullYear(2010,0,14);

在下面的例子中，我们将日期对象设置为 5 天后的日期：

var myDate=new Date();  
myDate.setDate(myDate.getDate()+5);

**注意:** 如果增加天数会改变月份或者年份，那么日期对象会自动完成这种转换。

## 两个日期比较

日期对象也可用于比较两个日期。

下面的代码将当前日期与 2100 年 1 月 14 日做了比较：

var x=new Date();

x.setFullYear(2100,0,14);

var today = new Date();

if (x>today)

{

    alert("今天是2100年1月14日之前");

}

else

{

    alert("今天是2100年1月14日之后");

}

# JavaScript Date 对象

## Date 对象

Date 对象用于处理日期与时间。

创建 Date 对象： **new Date()**

以下四种方法同样可以创建 Date 对象：

var d = new Date();

var d = new Date(milliseconds);

var d = new Date(dateString);

var d = new Date(year, month, day, hours, minutes, seconds, milliseconds);

更完整的日期与实际教程请参照[JavaScript Date 对象教程](http://www.runoob.com/js/js-obj-date.html)。

## Date 对象属性

|  |  |
| --- | --- |
| **属性** | **描述** |
| [constructor](http://www.runoob.com/jsref/jsref-constructor-date.html) | 返回对创建此对象的 Date 函数的引用。 |
| [prototype](http://www.runoob.com/jsref/jsref-prototype-date.html) | 使您有能力向对象添加属性和方法。 |

## Date 对象方法

|  |  |
| --- | --- |
| **方法** | **描述** |
| [getDate()](http://www.runoob.com/jsref/jsref-getdate.html) | 从 Date 对象返回一个月中的某一天 (1 ~ 31)。 |
| [getDay()](http://www.runoob.com/jsref/jsref-getday.html) | 从 Date 对象返回一周中的某一天 (0 ~ 6)。 |
| [getFullYear()](http://www.runoob.com/jsref/jsref-getfullyear.html) | 从 Date 对象以四位数字返回年份。 |
| [getHours()](http://www.runoob.com/jsref/jsref-gethours.html) | 返回 Date 对象的小时 (0 ~ 23)。 |
| [getMilliseconds()](http://www.runoob.com/jsref/jsref-getmilliseconds.html) | 返回 Date 对象的毫秒(0 ~ 999)。 |
| [getMinutes()](http://www.runoob.com/jsref/jsref-getminutes.html) | 返回 Date 对象的分钟 (0 ~ 59)。 |
| [getMonth()](http://www.runoob.com/jsref/jsref-getmonth.html) | 从 Date 对象返回月份 (0 ~ 11)。 |
| [getSeconds()](http://www.runoob.com/jsref/jsref-getseconds.html) | 返回 Date 对象的秒数 (0 ~ 59)。 |
| [getTime()](http://www.runoob.com/jsref/jsref-gettime.html) | 返回 1970 年 1 月 1 日至今的毫秒数。 |
| [getTimezoneOffset()](http://www.runoob.com/jsref/jsref-gettimezoneoffset.html) | 返回本地时间与格林威治标准时间 (GMT) 的分钟差。 |
| [getUTCDate()](http://www.runoob.com/jsref/jsref-getutcdate.html) | 根据世界时从 Date 对象返回月中的一天 (1 ~ 31)。 |
| [getUTCDay()](http://www.runoob.com/jsref/jsref-getutcday.html) | 根据世界时从 Date 对象返回周中的一天 (0 ~ 6)。 |
| [getUTCFullYear()](http://www.runoob.com/jsref/jsref-getutcfullyear.html) | 根据世界时从 Date 对象返回四位数的年份。 |
| [getUTCHours()](http://www.runoob.com/jsref/jsref-getutchours.html) | 根据世界时返回 Date 对象的小时 (0 ~ 23)。 |
| [getUTCMilliseconds()](http://www.runoob.com/jsref/jsref-getutcmilliseconds.html) | 根据世界时返回 Date 对象的毫秒(0 ~ 999)。 |
| [getUTCMinutes()](http://www.runoob.com/jsref/jsref-getutcminutes.html) | 根据世界时返回 Date 对象的分钟 (0 ~ 59)。 |
| [getUTCMonth()](http://www.runoob.com/jsref/jsref-getutcmonth.html) | 根据世界时从 Date 对象返回月份 (0 ~ 11)。 |
| [getUTCSeconds()](http://www.runoob.com/jsref/jsref-getutcseconds.html) | 根据世界时返回 Date 对象的秒钟 (0 ~ 59)。 |
| getYear() | 已废弃。 请使用 getFullYear() 方法代替。 |
| [parse()](http://www.runoob.com/jsref/jsref-parse.html) | 返回1970年1月1日午夜到指定日期（字符串）的毫秒数。 |
| [setDate()](http://www.runoob.com/jsref/jsref-setdate.html) | 设置 Date 对象中月的某一天 (1 ~ 31)。 |
| [setFullYear()](http://www.runoob.com/jsref/jsref-setfullyear.html) | 设置 Date 对象中的年份（四位数字）。 |
| [setHours()](http://www.runoob.com/jsref/jsref-sethours.html) | 设置 Date 对象中的小时 (0 ~ 23)。 |
| [setMilliseconds()](http://www.runoob.com/jsref/jsref-setmilliseconds.html) | 设置 Date 对象中的毫秒 (0 ~ 999)。 |
| [setMinutes()](http://www.runoob.com/jsref/jsref-setminutes.html) | 设置 Date 对象中的分钟 (0 ~ 59)。 |
| [setMonth()](http://www.runoob.com/jsref/jsref-setmonth.html) | 设置 Date 对象中月份 (0 ~ 11)。 |
| [setSeconds()](http://www.runoob.com/jsref/jsref-setseconds.html) | 设置 Date 对象中的秒钟 (0 ~ 59)。 |
| [setTime()](http://www.runoob.com/jsref/jsref-settime.html) | setTime() 方法以毫秒设置 Date 对象。 |
| [setUTCDate()](http://www.runoob.com/jsref/jsref-setutcdate.html) | 根据世界时设置 Date 对象中月份的一天 (1 ~ 31)。 |
| [setUTCFullYear()](http://www.runoob.com/jsref/jsref-setutcfullyear.html) | 根据世界时设置 Date 对象中的年份（四位数字）。 |
| [setUTCHours()](http://www.runoob.com/jsref/jsref-setutchours.html) | 根据世界时设置 Date 对象中的小时 (0 ~ 23)。 |
| [setUTCMilliseconds()](http://www.runoob.com/jsref/jsref-setutcmilliseconds.html) | 根据世界时设置 Date 对象中的毫秒 (0 ~ 999)。 |
| [setUTCMinutes()](http://www.runoob.com/jsref/jsref-setutcminutes.html) | 根据世界时设置 Date 对象中的分钟 (0 ~ 59)。 |
| [setUTCMonth()](http://www.runoob.com/jsref/jsref-setutcmonth.html) | 根据世界时设置 Date 对象中的月份 (0 ~ 11)。 |
| [setUTCSeconds()](http://www.runoob.com/jsref/jsref-setutcseconds.html) | setUTCSeconds() 方法用于根据世界时 (UTC) 设置指定时间的秒字段。 |
| setYear() | 已废弃。请使用 setFullYear() 方法代替。 |
| [toDateString()](http://www.runoob.com/jsref/jsref-todatestring.html) | 把 Date 对象的日期部分转换为字符串。 |
| toGMTString() | 已废弃。请使用 toUTCString() 方法代替。 |
| [toISOString()](http://www.runoob.com/jsref/jsref-toisostring.html) | 使用 ISO 标准返回字符串的日期格式。 |
| [toJSON()](http://www.runoob.com/jsref/jsref-tojson.html) | 以 JSON 数据格式返回日期字符串。 |
| [toLocaleDateString()](http://www.runoob.com/jsref/jsref-tolocaledatestring.html) | 根据本地时间格式，把 Date 对象的日期部分转换为字符串。 |
| [toLocaleTimeString()](http://www.runoob.com/jsref/jsref-tolocaletimestring.html) | 根据本地时间格式，把 Date 对象的时间部分转换为字符串。 |
| [toLocaleString()](http://www.runoob.com/jsref/jsref-tolocalestring.html) | 据本地时间格式，把 Date 对象转换为字符串。 |
| [toString()](http://www.runoob.com/jsref/jsref-tostring-date.html) | 把 Date 对象转换为字符串。 |
| [toTimeString()](http://www.runoob.com/jsref/jsref-totimestring.html) | 把 Date 对象的时间部分转换为字符串。 |
| [toUTCString()](http://www.runoob.com/jsref/jsref-toutcstring.html) | 根据世界时，把 Date 对象转换为字符串。 |
| [UTC()](http://www.runoob.com/jsref/jsref-utc.html) | 根据世界时返回 1970 年 1 月 1 日 到指定日期的毫秒数。 |
| [valueOf()](http://www.runoob.com/jsref/jsref-valueof-date.html) | 返回 Date 对象的原始值。 |

# JavaScript Number 对象

JavaScript 只有一种数字类型。

可以使用也可以不使用小数点来书写数字。

## JavaScript 数字

JavaScript 数字可以使用也可以不使用小数点来书写：

## 实例

var pi=3.14;    // 使用小数点  
var x=34;       // 不使用小数点

极大或极小的数字可通过科学（指数）计数法来写：

## 实例

var y=123e5;    // 12300000  
var z=123e-5;   // 0.00123

## 所有 JavaScript 数字均为 64 位

JavaScript 不是类型语言。与许多其他编程语言不同，JavaScript 不定义不同类型的数字，比如整数、短、长、浮点等等。

在JavaScript中，数字不分为整数类型和浮点型类型，所有的数字都是由 浮点型类型。JavaScript采用IEEE754标准定义的64位浮点格式表示数字，它能表示最大值为±1.7976931348623157 x 10308，最小值为±5 x 10 -324

|  |  |  |
| --- | --- | --- |
| **值 (aka Fraction/Mantissa)** | **指数** | **Sign** |
| 52 bits (0 - 51) | 11 bits (50 - 62) | 1 bit (63) |

## 精度

整数（不使用小数点或指数计数法）最多为 15 位。

## 实例

var x = 999999999999999; // x 为 999999999999999 var y = 9999999999999999; // y 为 10000000000000000

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=tryjs_inaccurate1)

小数的最大位数是 17，但是浮点运算并不总是 100% 准确：

## 实例

var x = 0.2+0.1; // 输出结果为 0.30000000000000004

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=tryjs_inaccurate)

## 八进制和十六进制

如果前缀为 0，则 JavaScript 会把数值常量解释为八进制数，如果前缀为 0 和 "x"，则解释为十六进制数。

## 实例

var y = 0377;   
var z = 0xFF;

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=tryjs_octal_hex)

|  |  |
| --- | --- |
| **lamp** | 绝不要在数字前面写零，除非您需要进行八进制转换。 |

默认情况下，JavaScript 数字为十进制显示。

但是你可以使用 toString() 方法 输出16进制、8进制、2进制。

## 实例

var myNumber=128;  
myNumber.toString(16);   // 返回 80  
myNumber.toString(8);    // 返回 200  
myNumber.toString(2);    // 返回 10000000

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=tryjs_tostring)

## 无穷大（Infinity）

当数字运算结果超过了JavaScript所能表示的数字上限（溢出），结果为一个特殊的无穷大（infinity）值，在JavaScript中以Infinity表示。同样地，当负数的值超过了JavaScript所能表示的负数范围，结果为负无穷大，在JavaScript中以-Infinity表示。无穷大值的行为特性和我们所期望的是一致的：基于它们的加、减、乘和除运算结果还是无穷大（当然还保留它们的正负号）。

## 实例

myNumber=2;  
while (myNumber!=Infinity)  
{  
    myNumber=myNumber\*myNumber; // 重复计算直到 myNumber 等于 Infinity  
}

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=tryjs_infinity)

除以0也产生了无限:

## 实例

var x = 2/0;  
var y = -2/0;

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=tryjs_infinity_zero)

## NaN - 非数字值

NaN 属性是代表非数字值的特殊值。该属性用于指示某个值不是数字。可以把 Number 对象设置为该值，来指示其不是数字值。

你可以使用 isNaN() 全局函数来判断一个值是否是 NaN 值。

## 实例

var x = 1000 / "Apple";  
isNaN(x); // 返回 true  
var y = 100 / "1000";  
isNaN(y); // 返回 false

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=tryjs_isnan_false)

除以0是无穷大，无穷大是一个数字:

## 实例

var x = 1000 / 0;  
isNaN(x); // 返回 false

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=tryjs_isnan_true)

## 数字可以是数字或者对象

数字可以私有数据进行初始化，就像 x = 123;

JavaScript 数字对象初始化数据， var y = new Number(123);

## 实例

var x = 123;  
var y = new Number(123);  
typeof(x) // 返回 Number  
typeof(y) // 返回 Object

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=tryjs_object_number_type)

## 实例

var x = 123;                
var y = new Number(123);  
(x === y) // 为 false，因为 x 是一个数字，y 是一个对象

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=tryjs_object_number)

## 数字属性

* MAX\_VALUE
* MIN\_VALUE
* NEGATIVE\_INFINITY
* POSITIVE\_INFINITY
* NaN
* prototype
* constructor

## 数字方法

* toExponential()
* toFixed()
* toPrecision()
* toString()
* valueOf()

# JavaScript Number 对象

## Number 对象

Number 对象是原始数值的包装对象。

Number 创建方式 new Number()。

## 语法

var num = new Number(value);

**注意：** 如果一个参数值不能转换为一个数字将返回 NaN (非数字值)。

## Number 对象属性

|  |  |
| --- | --- |
| **属性** | **描述** |
| [constructor](http://www.runoob.com/jsref/jsref-constructor-number.html) | 返回对创建此对象的 Number 函数的引用。 |
| [MAX\_VALUE](http://www.runoob.com/jsref/jsref-max-value.html) | 可表示的最大的数。 |
| [MIN\_VALUE](http://www.runoob.com/jsref/jsref-min-value.html) | 可表示的最小的数。 |
| [NEGATIVE\_INFINITY](http://www.runoob.com/jsref/jsref-negative-infinity.html) | 负无穷大，溢出时返回该值。 |
| [NaN](http://www.runoob.com/jsref/jsref-number-nan.html) | 非数字值。 |
| [POSITIVE\_INFINITY](http://www.runoob.com/jsref/jsref-positive-infinity.html) | 正无穷大，溢出时返回该值。 |
| [prototype](http://www.runoob.com/jsref/jsref-prototype-num.html) | 允许您有能力向对象添加属性和方法。 |

## Number 对象方法

|  |  |
| --- | --- |
| **方法** | **描述** |
| [toExponential(x)](http://www.runoob.com/jsref/jsref-toexponential.html) | 把对象的值转换为指数计数法。 |
| [toFixed(x)](http://www.runoob.com/jsref/jsref-tofixed.html) | 把数字转换为字符串，结果的小数点后有指定位数的数字。 |
| [toPrecision(x)](http://www.runoob.com/jsref/jsref-toprecision.html) | 把数字格式化为指定的长度。 |
| [toString()](http://www.runoob.com/jsref/jsref-tostring-number.html) | 把数字转换为字符串，使用指定的基数。 |
| [valueOf()](http://www.runoob.com/jsref/jsref-valueof-number.html) | 返回一个 Number 对象的基本数字值。 |

# JavaScript 字符串（String） 对象

String 对象用于处理已有的字符块。

## JavaScript 字符串

一个字符串用于存储一系列字符就像 "John Doe".

一个字符串可以使用单引号或双引号：

## 实例

var carname="Volvo XC60";  
var carname='Volvo XC60';

你使用位置（索引）可以访问字符串中任何的字符：

## 实例

var character=carname[7];

字符串的索引从零开始, 所以字符串第一字符为 [0],第二个字符为 [1], 等等。

你可以在字符串中使用引号，如下实例：

## 实例

var answer="It's alright";  
var answer="He is called 'Johnny'";  
var answer='He is called "Johnny"';

或者你可以在字符串中使用转义字符(\)使用引号：

## 实例

var answer='It\'s alright';  
var answer="He is called \"Johnny\"";

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=tryjs_datatypes_string)

## 字符串（String）

字符串（String）使用长度属性**length**来计算字符串的长度：

## 实例

var txt="Hello World!";  
document.write(txt.length);  
  
var txt="ABCDEFGHIJKLMNOPQRSTUVWXYZ";  
document.write(txt.length);

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=tryjs_string_length)

## 在字符串中查找字符串

字符串使用 indexOf() 来定位字符串中某一个指定的字符首次出现的位置：

## 实例

var str="Hello world, welcome to the universe.";  
var n=str.indexOf("welcome");

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=tryjs_string_indexof)

如果没找到对应的字符函数返回-1

lastIndexOf() 方法在字符串末尾开始查找字符串出现的位置。

## 内容匹配

**match()**函数用来查找字符串中特定的字符，并且如果找到的话，则返回这个字符。

## 实例

var str="Hello world!";  
document.write(str.match("world") + "<br>");  
document.write(str.match("World") + "<br>");  
document.write(str.match("world!"));

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=tryjs_string_match)

## 替换内容

**replace()** 方法在字符串中用某些字符替换另一些字符。

## 实例

str="Please visit Microsoft!"  
var n=str.replace("Microsoft","Runoob");

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=tryjs_string_replace)

## 字符串大小写转换

字符串大小写转换使用函数 **toUpperCase()**/**toLowerCase()**:

## 实例

var txt="Hello World!";       // String  
var txt1=txt.toUpperCase();   // txt1 文本会转换为大写  
var txt2=txt.toLowerCase();   // txt2 文本会转换为小写

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=tryjs_string_toupper)

## 字符串转为数组

字符串使用**split()**函数转为数组:

## 实例

txt="a,b,c,d,e"   // String  
txt.split(",");   // 使用逗号分隔  
txt.split(" ");   // 使用空格分隔  
txt.split("|");   // 使用竖线分隔

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=tryjs_string_split)

## 特殊字符

Javascript 中可以使用反斜线（\）插入特殊符号，如：撇号,引号等其他特殊符号。

查看如下 JavaScript 代码:

var txt="We are the so-called "Vikings" from the north.";  
document.write(txt);

在JavaScript中，字符串的开始和停止使用单引号或双引号。这意味着，上面的字符串将被切成： We are the so-called

解决以上的问题可以使用反斜线来转义引号：

var txt="We are the so-called \"Vikings\" from the north.";  
document.write(txt);

JavaScript将输出正确的文本字符串：We are the so-called "Vikings" from the north.

下表列出其他特殊字符，可以使用反斜线转义特殊字符：

|  |  |
| --- | --- |
| **代码** | **输出** |
| \' | 单引号 |
| \" | 双引号 |
| \\ | 斜杆 |
| \n | 换行 |
| \r | 回车 |
| \t | tab |
| \b | 空格 |
| \f | 换页 |

## 字符串属性和方法

属性:

* length
* prototype
* constructor

方法:

* charAt()
* charCodeAt()
* concat()
* fromCharCode()
* indexOf()
* lastIndexOf()
* match()
* replace()
* search()
* slice()
* split()
* substr()
* substring()
* toLowerCase()
* toUpperCase()
* valueOf()

[JavaScript Number 对象](http://www.runoob.com/js/js-obj-number.html)

[JavaScript Date（日期）对象](http://www.runoob.com/js/js-obj-date.html)

## 1 篇笔记

1. xiaoyao

  124\*\*\*4267@qq.com

关于 JavaScript 里的 **trim()/strip()** 方法 在其他语言中常常用 **trim()/strip()** 方法脱去不必要的空格等元素。

JavaScript 本身并不提供 trim() 方法，不过可以用正则表达式，通过给 String 对象增加 trim 方法的方式实现。

如下：

if(typeof(String.prototype.trim) === "undefined")

{

String.prototype.trim = function()

{

return String(this).replace(/^\s+|\s+$/g, '');

};

}

if(" dog ".trim() === "dog") {

document.write("成功去除首尾空格");

}

[**尝试一下 »**](https://c.runoob.com/codedemo/4512)

[xiaoyao](javascript:;)

   xiaoyao

  124\*\*\*4267@qq.com

7个月前 (11-17)

# JavaScript String 对象

## String 对象

String 对象用于处理文本（字符串）。

String 对象创建方法： new String().

## 语法

var txt = new String("*string*");

或者更简单方式：

var txt = "*string*";

了解 String 对象教程，请查看 [JavaScript String 对象教程](http://www.runoob.com/js/js-obj-string.html)。

## String 对象属性

|  |  |
| --- | --- |
| **属性** | **描述** |
| [constructor](http://www.runoob.com/jsref/jsref-constructor-string.html) | 对创建该对象的函数的引用 |
| [length](http://www.runoob.com/jsref/jsref-length-string.html) | 字符串的长度 |
| [prototype](http://www.runoob.com/jsref/jsref-prototype-string.html) | 允许您向对象添加属性和方法 |

## String 对象方法

|  |  |
| --- | --- |
| **方法** | **描述** |
| [charAt()](http://www.runoob.com/jsref/jsref-charat.html) | 返回在指定位置的字符。 |
| [charCodeAt()](http://www.runoob.com/jsref/jsref-charcodeat.html) | 返回在指定的位置的字符的 Unicode 编码。 |
| [concat()](http://www.runoob.com/jsref/jsref-concat-string.html) | 连接两个或更多字符串，并返回新的字符串。 |
| [fromCharCode()](http://www.runoob.com/jsref/jsref-fromcharcode.html) | 将 Unicode 编码转为字符。 |
| [indexOf()](http://www.runoob.com/jsref/jsref-indexof.html) | 返回某个指定的字符串值在字符串中首次出现的位置。 |
| [lastIndexOf()](http://www.runoob.com/jsref/jsref-lastindexof.html) | 从后向前搜索字符串，并从起始位置（0）开始计算返回字符串最后出现的位置。 |
| [match()](http://www.runoob.com/jsref/jsref-match.html) | 查找找到一个或多个正则表达式的匹配。 |
| [replace()](http://www.runoob.com/jsref/jsref-replace.html) | 在字符串中查找匹配的子串， 并替换与正则表达式匹配的子串。 |
| [search()](http://www.runoob.com/jsref/jsref-search.html) | 查找与正则表达式相匹配的值。 |
| [slice()](http://www.runoob.com/jsref/jsref-slice-string.html) | 提取字符串的片断，并在新的字符串中返回被提取的部分。 |
| [split()](http://www.runoob.com/jsref/jsref-split.html) | 把字符串分割为字符串数组。 |
| [substr()](http://www.runoob.com/jsref/jsref-substr.html) | 从起始索引号提取字符串中指定数目的字符。 |
| [substring()](http://www.runoob.com/jsref/jsref-substring.html) | 提取字符串中两个指定的索引号之间的字符。 |
| [toLowerCase()](http://www.runoob.com/jsref/jsref-tolowercase.html) | 把字符串转换为小写。 |
| [toUpperCase()](http://www.runoob.com/jsref/jsref-touppercase.html) | 把字符串转换为大写。 |
| trim() | 去除字符串两边的空白 |
| [valueOf()](http://www.runoob.com/jsref/jsref-valueof-string.html) | 返回某个字符串对象的原始值。 |

## String HTML 包装方法

HTML 包装方法返回加入了适当HTML标签的字符串。

|  |  |
| --- | --- |
| **方法** | **描述** |
| [anchor()](http://www.runoob.com/jsref/jsref-anchor.html) | 创建 HTML 锚。 |
| [big()](http://www.runoob.com/jsref/jsref-big.html) | 用大号字体显示字符串。 |
| [blink()](http://www.runoob.com/jsref/jsref-blink.html) | 显示闪动字符串。 |
| [bold()](http://www.runoob.com/jsref/jsref-bold.html) | 使用粗体显示字符串。 |
| [fixed()](http://www.runoob.com/jsref/jsref-fixed.html) | 以打字机文本显示字符串。 |
| [fontcolor()](http://www.runoob.com/jsref/jsref-fontcolor.html) | 使用指定的颜色来显示字符串。 |
| [fontsize()](http://www.runoob.com/jsref/jsref-fontsize.html) | 使用指定的尺寸来显示字符串。 |
| [italics()](http://www.runoob.com/jsref/jsref-italics.html) | 使用斜体显示字符串。 |
| [link()](http://www.runoob.com/jsref/jsref-link.html) | 将字符串显示为链接。 |
| [small()](http://www.runoob.com/jsref/jsref-small.html) | 使用小字号来显示字符串。 |
| [strike()](http://www.runoob.com/jsref/jsref-strike.html) | 用于显示加删除线的字符串。 |
| [sub()](http://www.runoob.com/jsref/jsref-sub.html) | 把字符串显示为下标。 |
| [sup()](http://www.runoob.com/jsref/jsref-sup.html) | 把字符串显示为上标。 |

# JavaScript RegExp 对象

RegExp：是正则表达式（regular expression）的简写。

## 完整 RegExp 对象参考手册

请查看我们的 [JavaScript RegExp 对象的参考手册](http://www.runoob.com/jsref/jsref-obj-regexp.html)，其中提供了可以与字符串对象一同使用的所有的属性和方法。

这个手册包含的关于每个属性和方法的用法的详细描述和实例。

## 什么是 RegExp？

正则表达式描述了字符的模式对象。

当您检索某个文本时，可以使用一种模式来描述要检索的内容。RegExp 就是这种模式。

简单的模式可以是一个单独的字符。

更复杂的模式包括了更多的字符，并可用于解析、格式检查、替换等等。

您可以规定字符串中的检索位置，以及要检索的字符类型，等等。

## 语法

var patt=new RegExp(pattern,modifiers);  
  
或更简单的方法  
  
var patt=/pattern/modifiers;

* 模式描述了一个表达式模型。
* 修饰符(modifiers)描述了检索是否是全局，区分大小写等。

***注意：****当使用构造函数创造正则对象时，需要常规的字符转义规则（在前面加反斜杠 \）。比如，以下是等价的：*

var re = new RegExp("\\w+");

var re = /\w+/;

## RegExp 修饰符

修饰符用于执行不区分大小写和全文的搜索。

**i** - 修饰符是用来执行不区分大小写的匹配。

**g** - 修饰符是用于执行全文的搜索（而不是在找到第一个就停止查找,而是找到所有的匹配）。

## 实例 1

在字符串中不区分大小写找"runoob"

var str = "Visit RUnoob"; var patt1 = /runoob/i;

以下**标记**的文本是获得的匹配的表达式：

Visit **RUnoob**

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=tryjsref_regexp_i)

## 实例 2

全文查找 "is"

var str="Is this all there is?"; var patt1=/is/g;

以下**标记**的文本是获得的匹配的表达式：

Is th**is** all there **is**?

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=tryjsref_regexp_g)

## 实例 3

全文查找和不区分大小写搜索 "is"

var str="Is this all there is?"; var patt1=/is/gi;

以下 **标记**的文本是获得的匹配的表达式：

**Is** th**is** all there **is**?

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=tryjsref_regexp_g_2)

## test()

test()方法搜索字符串指定的值，根据结果并返回真或假。

下面的示例是从字符串中搜索字符 "e" ：

## 实例

var patt1=new RegExp("e"); document.write(patt1.test("The best things in life are free"));

由于该字符串中存在字母 "e"，以上代码的输出将是：

true

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=tryjs_regexp_test)

当使用构造函数创造正则对象时，需要常规的字符转义规则（在前面加反斜杠 \）

## 实例

var re = new RegExp("\\w+");

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=tryjs_regexp_test1)

## exec()

exec() 方法检索字符串中的指定值。返回值是被找到的值。如果没有发现匹配，则返回 null。

下面的示例是从字符串中搜索字符 "e" ：

## 实例 1

var patt1=new RegExp("e"); document.write(patt1.exec("The best things in life are free"));

由于该字符串中存在字母 "e"，以上代码的输出将是：

e

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=tryjs_regexp_exec)

# JavaScript RegExp 对象

## RegExp 对象

正则表达式是描述字符模式的对象。

正则表达式用于对字符串模式匹配及检索替换，是对字符串执行模式匹配的强大工具。

## 语法

var patt=new RegExp(pattern,modifiers);  
  
或者更简单的方式:  
  
var patt=/pattern/modifiers;

* pattern（模式） 描述了表达式的模式
* modifiers(修饰符) 用于指定全局匹配、区分大小写的匹配和多行匹配

***注意：****当使用构造函数创造正则对象时，需要常规的字符转义规则（在前面加反斜杠 \）。比如，以下是等价的：*

var re = new RegExp("\\w+");

var re = /\w+/;

更多关于 RegExp 对象请阅读我们的 [JavaScript RegExp 对象教程](http://www.runoob.com/js/js-obj-regexp.html)。

## 修饰符

修饰符用于执行区分大小写和全局匹配:

|  |  |
| --- | --- |
| **修饰符** | **描述** |
| [i](http://www.runoob.com/js/jsref-regexp-i.html) | 执行对大小写不敏感的匹配。 |
| [g](http://www.runoob.com/js/jsref-regexp-g.html) | 执行全局匹配（查找所有匹配而非在找到第一个匹配后停止）。 |
| m | 执行多行匹配。 |

## 方括号

方括号用于查找某个范围内的字符：

|  |  |
| --- | --- |
| **表达式** | **描述** |
| [[abc]](http://www.runoob.com/jsref/jsref-regexp-charset.html) | 查找方括号之间的任何字符。 |
| [[^abc]](http://www.runoob.com/jsref/jsref-regexp-charset-not.html) | 查找任何不在方括号之间的字符。 |
| [0-9] | 查找任何从 0 至 9 的数字。 |
| [a-z] | 查找任何从小写 a 到小写 z 的字符。 |
| [A-Z] | 查找任何从大写 A 到大写 Z 的字符。 |
| [A-z] | 查找任何从大写 A 到小写 z 的字符。 |
| [adgk] | 查找给定集合内的任何字符。 |
| [^adgk] | 查找给定集合外的任何字符。 |
| (red|blue|green) | 查找任何指定的选项。 |

## 元字符

元字符（Metacharacter）是拥有特殊含义的字符：

|  |  |
| --- | --- |
| **元字符** | **描述** |
| [.](http://www.runoob.com/jsref/jsref-regexp-dot.html) | 查找单个字符，除了换行和行结束符。 |
| [\w](http://www.runoob.com/jsref/jsref-regexp-wordchar.html) | 查找单词字符。 |
| [\W](http://www.runoob.com/jsref/jsref-regexp-wordchar-non.html) | 查找非单词字符。 |
| [\d](http://www.runoob.com/jsref/jsref-regexp-digit.html) | 查找数字。 |
| [\D](http://www.runoob.com/jsref/jsref-regexp-digit-non.html) | 查找非数字字符。 |
| [\s](http://www.runoob.com/jsref/jsref-regexp-whitespace.html) | 查找空白字符。 |
| [\S](http://www.runoob.com/jsref/jsref-regexp-whitespace-non.html) | 查找非空白字符。 |
| [\b](http://www.runoob.com/jsref/jsref-regexp-begin.html) | 匹配单词边界。 |
| [\B](http://www.runoob.com/jsref/jsref-regexp-begin-not.html) | 匹配非单词边界。 |
| \0 | 查找 NULL 字符。 |
| [\n](http://www.runoob.com/jsref/jsref-regexp-newline.html) | 查找换行符。 |
| \f | 查找换页符。 |
| \r | 查找回车符。 |
| \t | 查找制表符。 |
| \v | 查找垂直制表符。 |
| [\xxx](http://www.runoob.com/jsref/jsref-regexp-octal.html) | 查找以八进制数 xxx 规定的字符。 |
| [\xdd](http://www.runoob.com/jsref/jsref-regexp-hex.html) | 查找以十六进制数 dd 规定的字符。 |
| [\uxxxx](http://www.runoob.com/jsref/jsref-regexp-unicode-hex.html) | 查找以十六进制数 xxxx 规定的 Unicode 字符。 |

## 量词

|  |  |
| --- | --- |
| **量词** | **描述** |
| [n+](http://www.runoob.com/jsref/jsref-regexp-onemore.html) | 匹配任何包含至少一个 n 的字符串。  例如，/a+/ 匹配 "candy" 中的 "a"，"caaaaaaandy" 中所有的 "a"。 |
| [n\*](http://www.runoob.com/jsref/jsref-regexp-zeromore.html) | 匹配任何包含零个或多个 n 的字符串。  例如，/bo\*/ 匹配 "A ghost booooed" 中的 "boooo"，"A bird warbled" 中的 "b"，但是不匹配 "A goat grunted"。 |
| [n?](http://www.runoob.com/jsref/jsref-regexp-zeroone.html) | 匹配任何包含零个或一个 n 的字符串。  例如，/e?le?/ 匹配 "angel" 中的 "el"，"angle" 中的 "le"。 |
| [n{X}](http://www.runoob.com/jsref/jsref-regexp-nx.html) | 匹配包含 X 个 n 的序列的字符串。  例如，/a{2}/ 不匹配 "candy," 中的 "a"，但是匹配 "caandy," 中的两个 "a"，且匹配 "caaandy." 中的前两个 "a"。 |
| [n{X,}](http://www.runoob.com/jsref/jsref-regexp-nxcomma.html) | X 是一个正整数。前面的模式 n 连续出现至少 X 次时匹配。  例如，/a{2,}/ 不匹配 "candy" 中的 "a"，但是匹配 "caandy" 和 "caaaaaaandy." 中所有的 "a"。 |
| [n{X,Y}](http://www.runoob.com/jsref/jsref-regexp-nxy.html) | X 和 Y 为正整数。前面的模式 n 连续出现至少 X 次，至多 Y 次时匹配。  例如，/a{1,3}/ 不匹配 "cndy"，匹配 "candy," 中的 "a"，"caandy," 中的两个 "a"，匹配 "caaaaaaandy" 中的前面三个 "a"。注意，当匹配 "caaaaaaandy" 时，即使原始字符串拥有更多的 "a"，匹配项也是 "aaa"。 |
| [n$](http://www.runoob.com/jsref/jsref-regexp-ndollar.html) | 匹配任何结尾为 n 的字符串。 |
| [^n](http://www.runoob.com/jsref/jsref-regexp-ncaret.html) | 匹配任何开头为 n 的字符串。 |
| [?=n](http://www.runoob.com/jsref/jsref-regexp-nfollow.html) | 匹配任何其后紧接指定字符串 n 的字符串。 |
| [?!n](http://www.runoob.com/jsref/jsref-regexp-nfollow-not.html) | 匹配任何其后没有紧接指定字符串 n 的字符串。 |

## RegExp 对象方法

|  |  |
| --- | --- |
| **方法** | **描述** |
| [compile](http://www.runoob.com/jsref/jsref-regexp-compile.html) | 在 1.5 版本中已废弃。 编译正则表达式。 |
| [exec](http://www.runoob.com/jsref/jsref-exec-regexp.html) | 检索字符串中指定的值。返回找到的值，并确定其位置。 |
| [test](http://www.runoob.com/jsref/jsref-test-regexp.html) | 检索字符串中指定的值。返回 true 或 false。 |
| [toString](http://www.runoob.com/jsref/jsref-regexp-tostring.html) | 返回正则表达式的字符串。 |

## 支持正则表达式的 String 对象的方法

|  |  |  |  |
| --- | --- | --- | --- |
| **方法** | **描述** | **FF** | **IE** |
| [search](http://www.runoob.com/js/jsref-search.html) | 检索与正则表达式相匹配的值。 | 1 | 4 |
| [match](http://www.runoob.com/js/jsref-match.html) | 找到一个或多个正则表达式的匹配。 | 1 | 4 |
| [replace](http://www.runoob.com/js/jsref-replace.html) | 替换与正则表达式匹配的子串。 | 1 | 4 |
| [split](http://www.runoob.com/js/jsref-split.html) | 把字符串分割为字符串数组。 | 1 | 4 |

## RegExp 对象属性

|  |  |
| --- | --- |
| **属性** | **描述** |
| [constructor](http://www.runoob.com/jsref/jsref-regexp-constructor.html) | 返回一个函数，该函数是一个创建 RegExp 对象的原型。 |
| [global](http://www.runoob.com/jsref/jsref-regexp-global.html) | 判断是否设置了 "g" 修饰符 |
| [ignoreCase](http://www.runoob.com/jsref/jsref-regexp-ignorecase.html) | 判断是否设置了 "i" 修饰符 |
| [lastIndex](http://www.runoob.com/jsref/jsref-lastindex-regexp.html) | 用于规定下次匹配的起始位置 |
| [multiline](http://www.runoob.com/jsref/jsref-regexp-multiline.html) | 判断是否设置了 "m" 修饰符 |
| [source](http://www.runoob.com/jsref/jsref-source-regexp.html) | 返回正则表达式的匹配模式 |

# JavaScript Math（算数） 对象

Math（算数）对象的作用是：执行常见的算数任务。

Examples

## 在线实例

[round()](http://www.runoob.com/try/try.php?filename=tryjsref_round)  
如何使用 round()。

[random()](http://www.runoob.com/try/try.php?filename=tryjsref_random)  
如何使用 random() 来返回 0 到 1 之间的随机数。

[max()](http://www.runoob.com/try/try.php?filename=tryjsref_max)  
如何使用 max() 来返回两个给定的数中的较大的数。（在 ECMASCript v3 之前，该方法只有两个参数。）

[min()](http://www.runoob.com/try/try.php?filename=tryjsref_min)  
如何使用 min() 来返回两个给定的数中的较小的数。（在 ECMASCript v3 之前，该方法只有两个参数。）

## 完整的 Math 对象参考手册

我们提供 [JavaScript Math 对象的参考手册](http://www.runoob.com/jsref/jsref-obj-math.html)，其中包括所有可用于算术对象的属性和方法。

该手册包含了对每个属性和方法的详细描述以及相关实例。

## Math 对象

Math（算数）对象的作用是：执行普通的算数任务。

Math 对象提供多种算数值类型和函数。无需在使用这个对象之前对它进行定义。

**使用Math的属性/方法的语法：**

var x=Math.PI;  
var y=Math.sqrt(16);

**注意：** Math对象无需在使用这个对象之前对它进行定义。

## 算数值

JavaScript 提供 8 种可被 Math 对象访问的算数值：

你可以参考如下Javascript常量使用方法：

Math.E  
Math.PI  
Math.SQRT2  
Math.SQRT1\_2  
Math.LN2  
Math.LN10  
Math.LOG2E  
Math.LOG10E

## 算数方法

除了可被 Math 对象访问的算数值以外，还有几个函数（方法）可以使用。

下面的例子使用了 Math 对象的 round 方法对一个数进行四舍五入。

document.write(Math.round(4.7));

上面的代码输出为：

5

下面的例子使用了 Math 对象的 random() 方法来返回一个介于 0 和 1 之间的随机数：

document.write(Math.random());

上面的代码输出为：

0.8293613976037759

下面的例子使用了 Math 对象的 floor() 方法和 random() 来返回一个介于 0 和 11 之间的随机数：

document.write(Math.floor(Math.random()\*11));

上面的代码输出为：

6

# JavaScript Math 对象

## Math 对象

Math 对象用于执行数学任务。

Math 对象并不像 Date 和 String 那样是对象的类，因此没有构造函数 Math()。

## 语法

var x = Math.PI; // 返回PI  
var y = Math.sqrt(16); // 返回16的平方根

Math 对象教程，请参照本站的[JavaScript Math 对象教程](http://www.runoob.com/js/js-obj-math.html)。

## Math 对象属性

|  |  |
| --- | --- |
| **属性** | **描述** |
| [E](http://www.runoob.com/jsref/jsref-e.html) | 返回算术常量 e，即自然对数的底数（约等于2.718）。 |
| [LN2](http://www.runoob.com/jsref/jsref-ln2.html) | 返回 2 的自然对数（约等于0.693）。 |
| [LN10](http://www.runoob.com/jsref/jsref-ln10.html) | 返回 10 的自然对数（约等于2.302）。 |
| [LOG2E](http://www.runoob.com/jsref/jsref-log2e.html) | 返回以 2 为底的 e 的对数（约等于 1.414）。 |
| [LOG10E](http://www.runoob.com/jsref/jsref-log10e.html) | 返回以 10 为底的 e 的对数（约等于0.434）。 |
| [PI](http://www.runoob.com/jsref/jsref-pi.html) | 返回圆周率（约等于3.14159）。 |
| [SQRT1\_2](http://www.runoob.com/jsref/jsref-sqrt1-2.html) | 返回返回 2 的平方根的倒数（约等于 0.707）。 |
| [SQRT2](http://www.runoob.com/jsref/jsref-sqrt2.html) | 返回 2 的平方根（约等于 1.414）。 |

## Math 对象方法

|  |  |
| --- | --- |
| **方法** | **描述** |
| [abs(x)](http://www.runoob.com/jsref/jsref-abs.html) | 返回 x 的绝对值。 |
| [acos(x)](http://www.runoob.com/jsref/jsref-acos.html) | 返回 x 的反余弦值。 |
| [asin(x)](http://www.runoob.com/jsref/jsref-asin.html) | 返回 x 的反正弦值。 |
| [atan(x)](http://www.runoob.com/jsref/jsref-atan.html) | 以介于 -PI/2 与 PI/2 弧度之间的数值来返回 x 的反正切值。 |
| [atan2(y,x)](http://www.runoob.com/jsref/jsref-atan2.html) | 返回从 x 轴到点 (x,y) 的角度（介于 -PI/2 与 PI/2 弧度之间）。 |
| [ceil(x)](http://www.runoob.com/jsref/jsref-ceil.html) | 对数进行上舍入。 |
| [cos(x)](http://www.runoob.com/jsref/jsref-cos.html) | 返回数的余弦。 |
| [exp(x)](http://www.runoob.com/jsref/jsref-exp.html) | 返回 Ex 的指数。 |
| [floor(x)](http://www.runoob.com/jsref/jsref-floor.html) | 对 x 进行下舍入。 |
| [log(x)](http://www.runoob.com/jsref/jsref-log.html) | 返回数的自然对数（底为e）。 |
| [max(x,y,z,...,n)](http://www.runoob.com/jsref/jsref-max.html) | 返回 x,y,z,...,n 中的最高值。 |
| [min(x,y,z,...,n)](http://www.runoob.com/jsref/jsref-min.html) | 返回 x,y,z,...,n中的最低值。 |
| [pow(x,y)](http://www.runoob.com/jsref/jsref-pow.html) | 返回 x 的 y 次幂。 |
| [random()](http://www.runoob.com/jsref/jsref-random.html) | 返回 0 ~ 1 之间的随机数。 |
| [round(x)](http://www.runoob.com/jsref/jsref-round.html) | 把数四舍五入为最接近的整数。 |
| [sin(x)](http://www.runoob.com/jsref/jsref-sin.html) | 返回数的正弦。 |
| [sqrt(x)](http://www.runoob.com/jsref/jsref-sqrt.html) | 返回数的平方根。 |
| [tan(x)](http://www.runoob.com/jsref/jsref-tan.html) | 返回角的正切。 |

# JavaScript 全局

JavaScript 全局属性和方法可用于创建Javascript对象。

## JavaScript 全局属性

|  |  |
| --- | --- |
| **属性** | **描述** |
| [Infinity](http://www.runoob.com/jsref/jsref-infinity.html) | 代表正的无穷大的数值。 |
| [NaN](http://www.runoob.com/jsref/jsref-nan.html) | 指示某个值是不是数字值。 |
| [undefined](http://www.runoob.com/jsref/jsref-undefined.html) | 指示未定义的值。 |

## JavaScript 全局函数

|  |  |
| --- | --- |
| **函数** | **描述** |
| [decodeURI()](http://www.runoob.com/jsref/jsref-decodeuri.html) | 解码某个编码的 URI。 |
| [decodeURIComponent()](http://www.runoob.com/jsref/jsref-decodeuricomponent.html) | 解码一个编码的 URI 组件。 |
| [encodeURI()](http://www.runoob.com/jsref/jsref-encodeuri.html) | 把字符串编码为 URI。 |
| [encodeURIComponent()](http://www.runoob.com/jsref/jsref-encodeuricomponent.html) | 把字符串编码为 URI 组件。 |
| [escape()](http://www.runoob.com/jsref/jsref-escape.html) | 对字符串进行编码。 |
| [eval()](http://www.runoob.com/jsref/jsref-eval.html) | 计算 JavaScript 字符串，并把它作为脚本代码来执行。 |
| [isFinite()](http://www.runoob.com/jsref/jsref-isfinite.html) | 检查某个值是否为有穷大的数。 |
| [isNaN()](http://www.runoob.com/jsref/jsref-isnan.html) | 检查某个值是否是数字。 |
| [Number()](http://www.runoob.com/jsref/jsref-number.html) | 把对象的值转换为数字。 |
| [parseFloat()](http://www.runoob.com/jsref/jsref-parsefloat.html) | 解析一个字符串并返回一个浮点数。 |
| [parseInt()](http://www.runoob.com/jsref/jsref-parseint.html) | 解析一个字符串并返回一个整数。 |
| [String()](http://www.runoob.com/jsref/jsref-string.html) | 把对象的值转换为字符串。 |
| [unescape()](http://www.runoob.com/jsref/jsref-unescape.html) | 对由 escape() 编码的字符串进行解码。 |

# JavaScript Window - 浏览器对象模型

浏览器对象模型 (BOM) 使 JavaScript 有能力与浏览器"对话"。

## 浏览器对象模型 (BOM)

浏览器对象模型（**B**rowser **O**bject **M**odel (BOM)）尚无正式标准。

由于现代浏览器已经（几乎）实现了 JavaScript 交互性方面的相同方法和属性，因此常被认为是 BOM 的方法和属性。

## Window 对象

所有浏览器都支持 window 对象。它表示浏览器窗口。

所有 JavaScript 全局对象、函数以及变量均自动成为 window 对象的成员。

全局变量是 window 对象的属性。

全局函数是 window 对象的方法。

甚至 HTML DOM 的 document 也是 window 对象的属性之一：

window.document.getElementById("header");

与此相同：

document.getElementById("header");

## Window 尺寸

有三种方法能够确定浏览器窗口的尺寸。

对于Internet Explorer、Chrome、Firefox、Opera 以及 Safari：

* window.innerHeight - 浏览器窗口的内部高度(包括滚动条)
* window.innerWidth - 浏览器窗口的内部宽度(包括滚动条)

对于 Internet Explorer 8、7、6、5：

* document.documentElement.clientHeight
* document.documentElement.clientWidth

或者

* document.body.clientHeight
* document.body.clientWidth

实用的 JavaScript 方案（涵盖所有浏览器）：

## 实例

var w=window.innerWidth  
|| document.documentElement.clientWidth  
|| document.body.clientWidth;  
  
var h=window.innerHeight  
|| document.documentElement.clientHeight  
|| document.body.clientHeight;

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=tryjs_win_inner)

该例显示浏览器窗口的高度和宽度：（不包括工具栏/滚动条）

## 其他 Window 方法

一些其他方法：

* window.open() - 打开新窗口
* window.close() - 关闭当前窗口
* window.moveTo() - 移动当前窗口
* window.resizeTo() - 调整当前窗

# JavaScript Window Screen

window.screen 对象包含有关用户屏幕的信息。

## Window Screen

**window.screen**对象在编写时可以不使用 window 这个前缀。

一些属性：

* screen.availWidth - 可用的屏幕宽度
* screen.availHeight - 可用的屏幕高度

## Window Screen 可用宽度

screen.availWidth 属性返回访问者屏幕的宽度，以像素计，减去界面特性，比如窗口任务栏。

## 实例

返回您的屏幕的可用宽度：

<script>  
document.write("可用宽度: " + screen.availWidth);  
</script>

以上代码输出为：

可用宽度: 1858

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=tryjs_screen_availwidth)

## Window Screen 可用高度

screen.availHeight 属性返回访问者屏幕的高度，以像素计，减去界面特性，比如窗口任务栏。

## 实例

返回您的屏幕的可用高度：

<script>  
  
document.write("可用高度: " + screen.availHeight);  
  
</script>

以上代码将输出：

可用高度: 1080

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=tryjs_screen_availheight)

[所有 screen 属性实例](http://www.runoob.com/try/try.php?filename=tryjs_screen_all)

# JavaScript 计时事件

## JavaScript 计时事件

通过使用 JavaScript，我们有能力做到在一个设定的时间间隔之后来执行代码，而不是在函数被调用后立即执行。我们称之为计时事件。

在 JavaScritp 中使用计时事件是很容易的，两个关键方法是:

* setInterval() - 间隔指定的毫秒数不停地执行指定的代码。
* setTimeout() - 在指定的毫秒数后执行指定代码。

**注意:** setInterval() 和 setTimeout() 是 HTML DOM Window对象的两个方法。

## setInterval() 方法

setInterval() 间隔指定的毫秒数不停地执行指定的代码

### 语法

window.setInterval("*javascript function*",*milliseconds*);

**window.setInterval()** 方法可以不使用window前缀，直接使用函数**setInterval()**。

setInterval() 第一个参数是函数（function）。

第二个参数间隔的毫秒数

**注意:** 1000 毫秒是一秒。

## 实例

每三秒弹出 "hello" ：

setInterval(function(){alert("Hello")},3000);

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=tryjs_setinterval1)

实例展示了如何使用 setInterval() 方法，但是每三秒弹出一次对用户体验并不好。

以下实例将显示当前时间。 setInterval() 方法设置每秒钟执行一次代码，就是手表一样。

## 实例

显示当前时间

var myVar=setInterval(function(){myTimer()},1000); function myTimer() { var d=new Date(); var t=d.toLocaleTimeString(); document.getElementById("demo").innerHTML=t; }

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=tryjs_setinterval2)

## 如何停止执行?

clearInterval() 方法用于停止 setInterval() 方法执行的函数代码。

### 语法

window.clearInterval(intervalVariable)

**window.clearInterval()** 方法可以不使用window前缀，直接使用函数**clearInterval()**。

要使用 clearInterval() 方法, 在创建计时方法时你必须使用全局变量：

myVar=setInterval("*javascript function*",*milliseconds*);

然后你可以使用clearInterval() 方法来停止执行。

## 实例

以下例子,我们添加了 "停止" 按钮：

<p id="demo"></p> <button onclick="myStopFunction()">停止</button> <script> var myVar=setInterval(function(){myTimer()},1000); function myTimer(){ var d=new Date(); var t=d.toLocaleTimeString(); document.getElementById("demo").innerHTML=t; } function myStopFunction(){ clearInterval(myVar); } </script>

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=tryjs_setinterval3)

## setTimeout() 方法

### 语法

myVar= window.setTimeout("*javascript function*",*milliseconds*);

setTimeout() 方法会返回某个值。在上面的语句中，值被储存在名为 myVar 的变量中。假如你希望取消这个 setTimeout()，你可以使用这个变量名来指定它。

setTimeout() 的第一个参数是含有 JavaScript 语句的字符串。这个语句可能诸如 "alert('5 seconds!')"，或者对函数的调用，诸如 alertMsg。

第二个参数指示从当前起多少毫秒后执行第一个参数。

提示：1000 毫秒等于一秒。

## 实例

等待3秒，然后弹出 "Hello":

setTimeout(function(){alert("Hello")},3000);

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=tryjs_timing1)

## 如何停止执行?

clearTimeout() 方法用于停止执行setTimeout()方法的函数代码。

### 语法

window.clearTimeout(timeoutVariable)

**window.clearTimeout()** 方法可以不使用window 前缀。

要使用clearTimeout() 方法, 你必须在创建超时方法中（setTimeout）使用全局变量:

myVar=setTimeout("*javascript function*",*milliseconds*);

如果函数还未被执行，你可以使用 clearTimeout() 方法来停止执行函数代码。

## 实例

以下是同一个实例, 但是添加了 "Stop the alert" 按钮:

var myVar; function myFunction() { myVar=setTimeout(function(){alert("Hello")},3000); } function myStopFunction() { clearTimeout(myVar); }

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=tryjs_settimeout2)

Examples

## 更多实例

[另一个简单的计时](http://www.runoob.com/try/try.php?filename=tryjs_timing2)

# JavaScript Cookie

Cookie 用于存储 web 页面的用户信息。

## 什么是 Cookie？

Cookie 是一些数据, 存储于你电脑上的文本文件中。

当 web 服务器向浏览器发送 web 页面时，在连接关闭后，服务端不会记录用户的信息。

Cookie 的作用就是用于解决 "如何记录客户端的用户信息":

* 当用户访问 web 页面时，他的名字可以记录在 cookie 中。
* 在用户下一次访问该页面时，可以在 cookie 中读取用户访问记录。

Cookie 以名/值对形式存储，如下所示:

username=John Doe

当浏览器从服务器上请求 web 页面时， 属于该页面的 cookie 会被添加到该请求中。服务端通过这种方式来获取用户的信息。

## 使用 JavaScript 创建Cookie

JavaScript 可以使用 **document.cookie** 属性来创建 、读取、及删除 cookie。

JavaScript 中，创建 cookie 如下所示：

document.cookie="username=John Doe";

您还可以为 cookie 添加一个过期时间（以 UTC 或 GMT 时间）。默认情况下，cookie 在浏览器关闭时删除：

document.cookie="username=John Doe; expires=Thu, 18 Dec 2013 12:00:00 GMT";

您可以使用 path 参数告诉浏览器 cookie 的路径。默认情况下，cookie 属于当前页面。

document.cookie="username=John Doe; expires=Thu, 18 Dec 2013 12:00:00 GMT; path=/";

## 使用 JavaScript 读取 Cookie

在 JavaScript 中, 可以使用以下代码来读取 cookie：

var x = document.cookie;

|  |  |
| --- | --- |
| **Note** | document.cookie 将以字符串的方式返回所有的 cookie，类型格式： cookie1=value; cookie2=value; cookie3=value; |

## 使用 JavaScript 修改 Cookie

在 JavaScript 中，修改 cookie 类似于创建 cookie，如下所示：

document.cookie="username=John Smith; expires=Thu, 18 Dec 2013 12:00:00 GMT; path=/";

旧的 cookie 将被覆盖。

## 使用 JavaScript 删除 Cookie

删除 cookie 非常简单。您只需要设置 expires 参数为以前的时间即可，如下所示，设置为 Thu, 01 Jan 1970 00:00:00 GMT:

document.cookie = "username=; expires=Thu, 01 Jan 1970 00:00:00 GMT";

注意，当您删除时不必指定 cookie 的值。

## Cookie 字符串

document.cookie 属性看起来像一个普通的文本字符串，其实它不是。

即使您在 document.cookie 中写入一个完整的 cookie 字符串, 当您重新读取该 cookie 信息时，cookie 信息是以名/值对的形式展示的。

如果您设置了新的 cookie，旧的 cookie 不会被覆盖。 新 cookie 将添加到 document.cookie 中，所以如果您重新读取document.cookie，您将获得如下所示的数据：

cookie1=value; cookie2=value;

显示所有 Cookie  创建 Cookie 1  创建 Cookie 2 删除 Cookie 1  删除 Cookie 2

如果您需要查找一个指定 cookie 值，您必须创建一个JavaScript 函数在 cookie 字符串中查找 cookie 值。

## JavaScript Cookie 实例

在以下实例中，我们将创建 cookie 来存储访问者名称。

首先，访问者访问 web 页面, 他将被要求填写自己的名字。该名字会存储在 cookie 中。

访问者下一次访问页面时，他会看到一个欢迎的消息。

在这个实例中我们会创建 3 个 JavaScript 函数:

1. 设置 cookie 值的函数
2. 获取 cookie 值的函数
3. 检测 cookie 值的函数

## 设置 cookie 值的函数

首先，我们创建一个函数用于存储访问者的名字：

function setCookie(cname,cvalue,exdays)

{

var d = new Date();

d.setTime(d.getTime()+(exdays\*24\*60\*60\*1000));

var expires = "expires="+d.toGMTString();

document.cookie = cname + "=" + cvalue + "; " + expires;

}

**函数解析：**

以上的函数参数中，cookie 的名称为 cname，cookie 的值为 cvalue，并设置了 cookie 的过期时间 expires。

该函数设置了 cookie 名、cookie 值、cookie过期时间。

## 获取 cookie 值的函数

然后，我们创建一个函数用户返回指定 cookie 的值：

function getCookie(cname)

{

var name = cname + "=";

var ca = document.cookie.split(';');

for(var i=0; i<ca.length; i++)

{

var c = ca[i].trim();

if (c.indexOf(name)==0) return c.substring(name.length,c.length);

}

return "";

}

**函数解析：**

cookie 名的参数为 cname。

创建一个文本变量用于检索指定 cookie :cname + "="。

使用分号来分割 document.cookie 字符串，并将分割后的字符串数组赋值给 ca (ca = document.cookie.split(';'))。

循环 ca 数组 (i=0;i<ca.length;i++)，然后读取数组中的每个值，并去除前后空格 (c=ca[i].trim())。

如果找到 cookie(c.indexOf(name) == 0)，返回 cookie 的值 (c.substring(name.length,c.length)。

如果没有找到 cookie, 返回 ""。

## 检测 cookie 值的函数

最后，我们可以创建一个检测 cookie 是否创建的函数。

如果设置了 cookie，将显示一个问候信息。

如果没有设置 cookie，将会显示一个弹窗用于询问访问者的名字，并调用 setCookie 函数将访问者的名字存储 365 天：

function checkCookie()

{

var username=getCookie("username");

if (username!="")

{

alert("Welcome again " + username);

}

else

{

username = prompt("Please enter your name:","");

if (username!="" && username!=null)

{

setCookie("username",username,365);

}

}

}

## 完整实例

## 实例

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<title>菜鸟教程(runoob.com)</title>

</head>

<head>

<script>

function setCookie(cname,cvalue,exdays){

var d = new Date();

d.setTime(d.getTime()+(exdays\*24\*60\*60\*1000));

var expires = "expires="+d.toGMTString();

document.cookie = cname+"="+cvalue+"; "+expires;

}

function getCookie(cname){

var name = cname + "=";

var ca = document.cookie.split(';');

for(var i=0; i<ca.length; i++) {

var c = ca[i].trim();

if (c.indexOf(name)==0) { return c.substring(name.length,c.length); }

}

return "";

}

function checkCookie(){

var user=getCookie("username");

if (user!=""){

alert("欢迎 " + user + " 再次访问");

}

else {

user = prompt("请输入你的名字:","");

if (user!="" && user!=null){

setCookie("username",user,30);

}

}

}

</script>

</head>

<body onload="checkCookie()"></body>

</html>  
[**尝试一下 »**](http://www.runoob.com/try/tryit.php?filename=tryjs_cookie_username)

以下实例在页面载入时执行 checkCookie() 函数。

# JavaScript 闭包

JavaScript 变量可以是局部变量或全局变量。

私有变量可以用到闭包。

## 全局变量

函数可以访问由函数内部定义的变量，如：

## 实例

function myFunction() { var a = 4; return a \* a; }

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=tryjs_function_scope2)

函数也可以访问函数外部定义的变量，如：

## 实例

var a = 4; function myFunction() { return a \* a; }

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=tryjs_function_scope1)

后面一个实例中， **a** 是一个 **全局** 变量。

在web页面中全局变量属于 window 对象。

全局变量可应用于页面上的所有脚本。

在第一个实例中， **a** 是一个 **局部** 变量。

局部变量只能用于定义它函数内部。对于其他的函数或脚本代码是不可用的。

全局和局部变量即便名称相同，它们也是两个不同的变量。修改其中一个，不会影响另一个的值。

|  |  |
| --- | --- |
| **Note** | 变量声明时如果不使用 **var** 关键字，那么它就是一个全局变量，即便它在函数内定义。 |

## 变量生命周期

全局变量的作用域是全局性的，即在整个JavaScript程序中，全局变量处处都在。

而在函数内部声明的变量，只在函数内部起作用。这些变量是局部变量，作用域是局部性的；函数的参数也是局部性的，只在函数内部起作用。

## 计数器困境

设想下如果你想统计一些数值，且该计数器在所有函数中都是可用的。

你可以使用全局变量，函数设置计数器递增：

## 实例

var counter = 0; function add() { return counter += 1; } add(); add(); add(); // 计数器现在为 3

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=tryjs_function_counter)

计数器数值在执行 add() 函数时发生变化。

但问题来了，页面上的任何脚本都能改变计数器，即便没有调用 add() 函数。

如果我在函数内声明计数器，如果没有调用函数将无法修改计数器的值：

## 实例

function add() { var counter = 0; return counter += 1; } add(); add(); add(); // 本意是想输出 3, 但事与愿违，输出的都是 1 !

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=tryjs_function_counter2)

以上代码将无法正确输出，每次我调用 add() 函数，计数器都会设置为 1。

**JavaScript 内嵌函数可以解决该问题。**

## JavaScript 内嵌函数

所有函数都能访问全局变量。

实际上，在 JavaScript 中，所有函数都能访问它们上一层的作用域。

JavaScript 支持嵌套函数。嵌套函数可以访问上一层的函数变量。

该实例中，内嵌函数 **plus()** 可以访问父函数的 **counter** 变量：

## 实例

function add() { var counter = 0; function plus() {counter += 1;} plus(); return counter; }

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=tryjs_function_counter1)

如果我们能在外部访问 **plus()** 函数，这样就能解决计数器的困境。

我们同样需要确保 **counter = 0** 只执行一次。

**我们需要闭包。**

## JavaScript 闭包

还记得函数自我调用吗？该函数会做什么？

## 实例

var add = (function () { var counter = 0; return function () {return counter += 1;} })(); add(); add(); add(); // 计数器为 3

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=tryjs_function_counter3)

## 实例解析

变量 **add** 指定了函数自我调用的返回字值。

自我调用函数只执行一次。设置计数器为 0。并返回函数表达式。

add变量可以作为一个函数使用。非常棒的部分是它可以访问函数上一层作用域的计数器。

这个叫作 JavaScript **闭包。**它使得函数拥有私有变量变成可能。

计数器受匿名函数的作用域保护，只能通过 add 方法修改。

|  |  |
| --- | --- |
| **Note** | 闭包是可访问上一层函数作用域里变量的函数，即便上一层函数已经关闭。 |

## 5 篇笔记

1. Mid

  279\*\*\*549@qq.com

我按我的理解总结下JavaScript闭包吧：

闭包就是一个函数引用另一个函数的变量，因为变量被引用着所以不会被回收，因此可以用来封装一个私有变量。这是优点也是缺点，不必要的闭包只会增加内存消耗。

或者说闭包就是子函数可以使用父函数的局部变量，还有父函数的参数。

[Mid](javascript:;)

   Mid

  279\*\*\*549@qq.com

10个月前 (07-21)

1. WosLovesLife

  zha\*\*\*0h00@126.com

我猜想可能很多人看到最后这个计数器问题都会感到困惑, 我也一样。

经过代码验证, 发现其中的奥妙在于：

var add = (function () {

var counter = 0;

return function () {return counter += 1;}

})();

注意: 为什么上面这段代码没有直接写的 function add (){...} 而是把function赋值给了变量add呢?

我们通常会想当然的认为每次调用 add() 都会重走一遍add()中的代码块, 但其实不然。

注意add方法中的return, 它return的并不是1,2,3这样的数值,而是return了一个方法,并且把这个方法赋值给了add变量。

那么在这个function自运行一遍之后,其实最后赋值给add的是return counter += 1 这段代码。

所以后面每次调用add() 其实都是在调用return counter += 1。

再结合文章之前所说的, 闭包会持有父方法的局部变量并且不会随父方法销毁而销毁, 所以这个counter其实就是来自于第一次function执行时创建的变量。

我写了一段demo如下

var tempFunc;

var add = (function () {

    var counter = 0;

    tempFunc = function () { return counter += 1; }

    return tempFunc

})();

function myFunction() {

    console.log("add === tempFunc : " + (add === tempFunc))

    document.getElementById("addOneS").innerHTML = add();

}

[WosLovesLife](javascript:;)

   WosLovesLife

  zha\*\*\*0h00@126.com

6个月前 (12-02)

1. 张玉东

  166\*\*\*9846@qq.com

理解闭包可以将以上代码分解如下：

function outerFunction() {

    var counter = 0;

    function innerFunction(){

        return counter += 1;

    }

    return innerFunction;

    /\*

     注意 typeof innerFunction 是：function；而typeof innerFunction（）是number；

    \*/

}

var add = outerFunction();

/\*

调用 outerFunction()返回的是内部函数innerFucntion,那么调用几次add()将调用几次

内部函数inner Function，内部函数公用了counter，所以能够计数,所以说闭包就是将内部嵌套函数变成外部可调用的。

\*/

add();

add();

add();

[张玉东](javascript:;)

   张玉东

  166\*\*\*9846@qq.com

6个月前 (12-05)

1. MRL

  lwq\*\*\*@163.com

闭包就是在提供了一个在外部访问另一个函数内部局部变量的方式。

var add = (function(){

var count = 0;//外部访问的计数器，局部变量.

var fun = function(){

return ++count;

}

return fun;

})();

//还可以这样写

var add2 = (function(){

var count = 0;//外部访问的计数器，局部变量.

function plus(){

return ++count;

}

return plus;

})();

//还可以这样写

var add3 = (function(){

var count = 0;//外部访问的计数器，局部变量.

return function(){

return ++count;

}

})();

[MRL](javascript:;)

   MRL

  lwq\*\*\*@163.com

4个月前 (01-26)

1. 风之翼

  kof\*\*\*dj@aliyun.com

闭包就是个bug。我相信作者在写javascript的时候都没想到用户会这么用。

闭包其实是利用了一个变量退出作用域的时候，暂时没有被销毁，它的值还在，如果后面有变量也叫这个名字，那这个数据会被重新利用起来。你会发现，后面你使用的这个名字一样的变量是有初值的。下面的例子可以看出问题来。

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<title>菜鸟教程(runoob.com)</title>

</head>

<body>

<p>局部变量计数。</p>

<button type="button" onclick="myFunction()">计数!</button>

<p id="demo">0</p>

<script>

{

var tmp = 2; //理论上在退出语句块后，这个变量要被释放掉的。包括内存可能被回收。但事实并非如此，会影响后面和他同名的变量

}

var add = (function () {

//var counter = 0; //这里注释掉.其实和上面的tmp一样的道理。这里在函数自己执行完后就应该销毁了的。

//return function () {return counter += 1;} //这里的counter已经不是上面的counter了，是一个全局变量。有初值，受上面影响，初值为0

return function () {return tmp += 1;} //这里tmp就是个全局变量。它是有初值的。为上面的2

})();

function myFunction(){

    document.getElementById("demo").innerHTML = add();//3

    document.getElementById("demo").innerHTML = add();//4

    document.getElementById("demo").innerHTML = add(); //5

}

</script>

</body>

</html>

执行上面的代码可以看的很清楚。计数器一样正常工作。

所以，闭包就是利用了个bug: 退出作用域的变量不会立即销毁。会影响后面的同名变量。

但不知道后面的javascript版本会不会修改这种机制。

所以代码这么写是不可靠的。太依赖于javascript的内部实现了。

**其实要实现全局计数器的需求。正常的做法是:**

定义一个全局对象。这个对象定义自己的属性和方法add。

用的时候 **对象.add** 就可以了。这个才正常。

上面用闭包的方式太过巧妙，给人的感觉就是在夹缝中求生存。用起来也不舒服吧。

[风之翼](javascript:;)

   风之翼

  kof\*\*\*dj@aliyun.com

3个月前 (03-14)

# JavaScript 函数定义

JavaScript 使用关键字 **function** 定义函数。

函数可以通过声明定义，也可以是一个表达式。

## 函数声明

在之前的教程中，你已经了解了函数声明的语法 :

function functionName(parameters) {

执行的代码

}

函数声明后不会立即执行，会在我们需要的时候调用到。

## 实例

function myFunction(a, b) {  
    return a \* b;  
}

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=tryjs_function_return)

|  |  |
| --- | --- |
| **Note** | 分号是用来分隔可执行JavaScript语句。  由于函数声明不是一个可执行语句，所以不以分号结束。 |

## 函数表达式

JavaScript 函数可以通过一个表达式定义。

函数表达式可以存储在变量中：

## 实例

var x = function (a, b) {return a \* b};

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=tryjs_function_expression)

在函数表达式存储在变量后，变量也可作为一个函数使用：

## 实例

var x = function (a, b) {return a \* b};  
var z = x(4, 3);

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=tryjs_function_expression_variable)

以上函数实际上是一个 **匿名函数** (函数没有名称)。

函数存储在变量中，不需要函数名称，通常通过变量名来调用。

|  |  |
| --- | --- |
| **Note** | 上述函数以分号结尾，因为它是一个执行语句。 |

## Function() 构造函数

在以上实例中，我们了解到函数通过关键字 **function** 定义。

函数同样可以通过内置的 JavaScript 函数构造器（Function()）定义。

## 实例

var myFunction = new Function("a", "b", "return a \* b");  
  
var x = myFunction(4, 3);

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=tryjs_function_constructor)

实际上，你不必使用构造函数。上面实例可以写成：

## 实例

var myFunction = function (a, b) {return a \* b}  
  
var x = myFunction(4, 3);

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=tryjs_function_constructor2)

|  |  |
| --- | --- |
| **Note** | 在 JavaScript 中，很多时候，你需要避免使用 **new** 关键字。 |

## 函数提升（Hoisting）

在之前的教程中我们已经了解了 "hoisting(提升)"。

提升（Hoisting）是 JavaScript 默认将当前作用域提升到前面去的的行为。

提升（Hoisting）应用在变量的声明与函数的声明。

因此，函数可以在声明之前调用：

myFunction(5);

function myFunction(y) {

return y \* y;

}

使用表达式定义函数时无法提升。

## 自调用函数

函数表达式可以 "自调用"。

自调用表达式会自动调用。

如果表达式后面紧跟 () ，则会自动调用。

不能自调用声明的函数。

通过添加括号，来说明它是一个函数表达式：

## 实例

(function () {  
    var x = "Hello!!";      // 我将调用自己  
})();

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=tryjs_function_expression_self)

以上函数实际上是一个 **匿名自我调用的函数** (没有函数名)。

## 函数可作为一个值使用

JavaScript 函数作为一个值使用：

## 实例

function myFunction(a, b) {  
    return a \* b;  
}  
  
var x = myFunction(4, 3);

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=tryjs_function_value)

JavaScript 函数可作为表达式使用：

## 实例

function myFunction(a, b) {  
    return a \* b;  
}  
  
var x = myFunction(4, 3) \* 2;

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=tryjs_function_value2)

## 函数是对象

在 JavaScript 中使用 **typeof** 操作符判断函数类型将返回 "function" 。

但是JavaScript 函数描述为一个对象更加准确。

JavaScript 函数有 **属性** 和 **方法**。

arguments.length 属性返回函数调用过程接收到的参数个数：

## 实例

function myFunction(a, b) {  
    return arguments.length;  
}

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=tryjs_function_length)

toString() 方法将函数作为一个字符串返回:

## 实例

function myFunction(a, b) {  
    return a \* b;  
}  
  
var txt = myFunction.toString();

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=tryjs_function_tostring)

|  |  |
| --- | --- |
| **Note** | 函数定义作为对象的属性，称之为对象方法。 函数如果用于创建新的对象，称之为对象的构造函数。 |

[JavaScript 代码规范](http://www.runoob.com/js/js-conventions.html)

[JavaScript 函数参数](http://www.runoob.com/js/js-function-parameters.html)

## 1 篇笔记

1. 珍惜

  669\*\*\*60@qq.com

匿名函数自动调用表达式：

(functin(){})()

例如：

<p id="demo"></p>

<script>

(function(){document.getElementById("demo").innerHTML = "hello kity";})()

</script>

[**尝试一下 »**](https://c.runoob.com/codedemo/5358)

# javascript:void(0) 含义

我们经常会使用到 javascript:void(0) 这样的代码，那么在 JavaScript 中 javascript:void(0) 代表的是什么意思呢？

javascript:void(0) 中最关键的是 void 关键字， void 是 JavaScript 中非常重要的关键字，该操作符指定要计算一个表达式但是不返回值。

语法格式如下：

<head>

<script type="text/javascript">

<!--

void func()

javascript:void func()

或者

void(func())

javascript:void(func())

//-->

</script>

</head>

下面的代码创建了一个超级链接，当用户点击以后不会发生任何事。

## 实例

<a href="javascript:void(0)">单击此处什么也不会发生</a>

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=tryjs_void&basepath=0)

当用户链接时，void(0) 计算为 0，但 Javascript 上没有任何效果。

以下实例中，在用户点击链接后显示警告信息：

## 实例

<head>  
<script type="text/javascript">  
<!--  
//-->  
</script>  
</head>  
<body>  
<a href="javascript:void(alert('Warning!!!'))">点我!</a>  
</body>

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=tryjs_void1&basepath=0)

以下实例中参数 a 将返回 undefined :

## 实例

<head>  
<script type="text/javascript">  
<!--  
function getValue(){  
  var a,b,c;  
  a = void ( b = 5, c = 7 );  
  document.write('a = ' + a + ' b = ' + b +' c = ' + c );  
}  
//-->  
</script>  
</head>

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=tryjs_void2&basepath=0)

## href="#"与href="javascript:void(0)"的区别

**#** 包含了一个位置信息，默认的锚是**#top** 也就是网页的上端。

而javascript:void(0), 仅仅表示一个死链接。

在页面很长的时候会使用 **#** 来定位页面的具体位置，格式为：**# + id**。

如果你要定义一个死链接请使用 javascript:void(0) 。

## 实例

<a href="javascript:void(0);">点我没有反应的!</a>  
<a href="#pos">点我定位到指定位置!</a>  
<br>  
...  
<br>  
<p id="pos">尾部定位点</p>

[**尝试一下 »**](http://www.runoob.com/try/demo_source/tryjs_void3.htm)

# JavaScript JSON

JSON 是用于存储和传输数据的格式。

JSON 通常用于服务端向网页传递数据 。

## 什么是 JSON?

* JSON 英文全称 **J**ava**S**cript **O**bject **N**otation
* JSON 是一种轻量级的数据交换格式。
* JSON是独立的语言 **\***
* JSON 易于理解。

|  |  |
| --- | --- |
| **Note** | \* JSON 使用 JavaScript 语法，但是 JSON 格式仅仅是一个文本。 文本可以被任何编程语言读取及作为数据格式传递。 |

## JSON 实例

以下 JSON 语法定义了 sites 对象: 3 条网站信息（对象）的数组:

## JSON 实例

{"sites":[ {"name":"Runoob", "url":"www.runoob.com"}, {"name":"Google", "url":"www.google.com"}, {"name":"Taobao", "url":"www.taobao.com"} ]}

## JSON 格式化后为 JavaScript 对象

JSON 格式在语法上与创建 JavaScript 对象代码是相同的。

由于它们很相似，所以 JavaScript 程序可以很容易的将 JSON 数据转换为 JavaScript 对象。

## JSON 语法规则

* 数据为 键/值 对。
* 数据由逗号分隔。
* 大括号保存对象
* 方括号保存数组

## JSON 数据 - 一个名称对应一个值

JSON 数据格式为 键/值 对，就像 JavaScript 对象属性。

键/值对包括字段名称（在双引号中），后面一个冒号，然后是值：

"name":"Runoob"

## JSON 对象

JSON 对象保存在大括号内。

就像在 JavaScript 中, 对象可以保存多个 键/值 对：

{"name":"Runoob", "url":"www.runoob.com"}

## JSON 数组

JSON 数组保存在中括号内。

就像在 JavaScript 中, 数组可以包含对象：

"sites":[ {"name":"Runoob", "url":"www.runoob.com"}, {"name":"Google", "url":"www.google.com"}, {"name":"Taobao", "url":"www.taobao.com"} ]

在以上实例中，对象 "sites" 是一个数组，包含了三个对象。

每个对象为站点的信息（网站名和网站地址）。

## JSON 字符串转换为 JavaScript 对象

通常我们从服务器中读取 JSON 数据，并在网页中显示数据。

简单起见，我们网页中直接设置 JSON 字符串 (你还可以阅读我们的 [JSON 教程](http://www.runoob.com/json/json-tutorial.html)):

首先，创建 JavaScript 字符串，字符串为 JSON 格式的数据：

var text = '{ "sites" : [' + '{ "name":"Runoob" , "url":"www.runoob.com" },' + '{ "name":"Google" , "url":"www.google.com" },' + '{ "name":"Taobao" , "url":"www.taobao.com" } ]}';

然后，使用 JavaScript 内置函数 JSON.parse() 将字符串转换为 JavaScript 对象:

var obj = JSON.parse(text);

最后，在你的页面中使用新的 JavaScript 对象：

## 实例

var text = '{ "sites" : [' + '{ "name":"Runoob" , "url":"www.runoob.com" },' + '{ "name":"Google" , "url":"www.google.com" },' + '{ "name":"Taobao" , "url":"www.taobao.com" } ]}'; obj = JSON.parse(text); document.getElementById("demo").innerHTML = obj.sites[1].name + " " + obj.sites[1].url;

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=tryjs_json_parse)

## 相关函数

|  |  |
| --- | --- |
| **函数** | **描述** |
| [JSON.parse()](http://www.runoob.com/js/javascript-json-parse.html) | 用于将一个 JSON 字符串转换为 JavaScript 对象。 |
| [JSON.stringify()](http://www.runoob.com/js/javascript-json-stringify.html) | 用于将 JavaScript 值转换为 JSON 字符串。 |

更多 JSON 信息，你可以阅读我们的 [JSON 教程](http://www.runoob.com/json/json-tutorial.html)。

# JavaScript 验证 API

## 约束验证 DOM 方法

|  |  |
| --- | --- |
| **Property** | **Description** |
| checkValidity() | 如果 input 元素中的数据是合法的返回 true，否则返回 false。 |
| setCustomValidity() | 设置 input 元素的 validationMessage 属性，用于自定义错误提示信息的方法。  使用 setCustomValidity 设置了自定义提示后，validity.customError 就会变成true，则 checkValidity 总是会返回false。如果要重新判断需要取消自定义提示，方式如下：  setCustomValidity('')  setCustomValidity(null)  setCustomValidity(undefined) |

以下实例如果输入信息不合法，则返回错误信息：

## checkValidity() 方法

<input id="id1" type="number" min="100" max="300" required> <button onclick="myFunction()">验证</button> <p id="demo"></p> <script>

function myFunction() { var inpObj = document.getElementById("id1"); if (inpObj.checkValidity() == false) { document.getElementById("demo").innerHTML = inpObj.validationMessage; } }

</script>

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=tryjs_validation_check)

## 约束验证 DOM 属性

|  |  |
| --- | --- |
| **属性** | **描述** |
| validity | 布尔属性值，返回 input 输入值是否合法 |
| validationMessage | 浏览器错误提示信息 |
| willValidate | 指定 input 是否需要验证 |

## Validity 属性

input 元素的 **validity 属性**包含一系列关于 validity 数据属性:

|  |  |
| --- | --- |
| **属性** | **描述** |
| customError | 设置为 true, 如果设置了自定义的 validity 信息。 |
| patternMismatch | 设置为 true, 如果元素的值不匹配它的模式属性。 |
| rangeOverflow | 设置为 true, 如果元素的值大于设置的最大值。 |
| rangeUnderflow | 设置为 true, 如果元素的值小于它的最小值。 |
| stepMismatch | 设置为 true, 如果元素的值不是按照规定的 step 属性设置。 |
| tooLong | 设置为 true, 如果元素的值超过了 maxLength 属性设置的长度。 |
| typeMismatch | 设置为 true, 如果元素的值不是预期相匹配的类型。 |
| valueMissing | 设置为 true，如果元素 (required 属性) 没有值。 |
| valid | 设置为 true，如果元素的值是合法的。 |

## 实例

如果输入的值大于 100，显示一个信息：

## rangeOverflow 属性

<input id="id1" type="number" max="100"> <button onclick="myFunction()">验证</button> <p id="demo"></p> <script>

function myFunction() { var txt = ""; if (document.getElementById("id1").validity.rangeOverflow) { txt = "输入的值太大了"; } document.getElementById("demo").innerHTML = txt; }

</script>

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=tryjs_validation_rangeOverflow)

如果输入的值小于 100，显示一个信息：

## rangeUnderflow 属性

<input id="id1" type="number" min="100" required> <button onclick="myFunction()">OK</button> <p id="demo"></p> <script>

function myFunction() { var txt = ""; var inpObj = document.getElementById("id1"); if(!isNumeric(inpObj.value)) { txt = "你输入的不是数字"; } else if (inpObj.validity.rangeUnderflow) { txt = "输入的值太小了"; } else { txt = "输入正确"; } document.getElementById("demo").innerHTML = txt; } // 判断输入是否为数字 function isNumeric(n) { return !isNaN(parseFloat(n)) && isFinite(n); }

</script>

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=tryjs_validation_rangeUnderflow)

[JavaScript 表单验证](http://www.runoob.com/js/js-form-validation.html)

[JavaScript 保留关键字](http://www.runoob.com/js/js-reserved.html)

## 1 篇笔记

1. 不爱李磊的韩梅梅

  227\*\*\*1690@qq.com

setCustomValidity 的用法：

var inpObj = document.getElementById("id1");

inpObj.setCustomValidity(''); // 取消自定义提示的方式

if (inpObj.checkValidity() == false) {

    if(inpObj.value==""){

        inpObj.setCustomValidity("不能为空！");

    }else if(inpObj.value<100 || inpObj.value>300){

        inpObj.setCustomValidity("请重新输入数值（100~300之间）!");

    }

document.getElementById("demo").innerHTML = inpObj.validationMessage;

} else {

document.getElementById("demo").innerHTML = "输入正确";

}

[**尝试一下 »**](https://c.runoob.com/codedemo/2918)

[不爱李磊的韩梅梅](javascript:;)

   不爱李磊的韩梅梅

  227\*\*\*1690@qq.com

1年前 (2017-05-24)

# JavaScript 表单

## JavaScript 表单验证

HTML 表单验证可以通过 JavaScript 来完成。

以下实例代码用于判断表单字段(fname)值是否存在，如果存在，则弹出信息，否则阻止表单提交：

## JavaScript 实例

function validateForm() { var x = document.forms["myForm"]["fname"].value; if (x == null || x == "") { alert("需要输入名字。"); return false; } }

以上 JavaScript 代码可以通过 HTML 代码来调用：

## HTML 表单实例

<form name="myForm" action="demo\_form.php" onsubmit="return validateForm()" method="post"> 名字: <input type="text" name="fname"> <input type="submit" value="提交"> </form>

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=tryjs_validation_js)

## JavaScript 验证输入的数字

JavaScript 常用于对输入数字的验证：

请输入 1 到 10 之间的数字：

 提交

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=tryjs_validation_number)

## HTML 表单自动验证

HTML 表单验证也可以通过浏览器来自动完成。

如果表单字段 (fname) 的值为空, **required** 属性会阻止表单提交：

## 实例

<form action="demo\_form.php" method="post"> <input type="text" name="fname" required="required"> <input type="submit" value="提交"> </form>

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=tryjs_validation_html)

*Internet Explorer 9 及更早 IE 浏览器不支持表单自动验证。*

## 数据验证

数据验证用于确保用户输入的数据是有效的。

典型的数据验证有：

* 必需字段是否有输入?
* 用户是否输入了合法的数据?
* 在数字字段是否输入了文本?

大多数情况下，数据验证用于确保用户正确输入数据。

数据验证可以使用不同方法来定义，并通过多种方式来调用。

**服务端数据验证**是在数据提交到服务器上后再验证。

**客户端数据验证 side validation**是在数据发送到服务器前，在浏览器上完成验证。

## HTML 约束验证

HTML5 新增了 HTML 表单的验证方式：约束验证（constraint validation）。

约束验证是表单被提交时浏览器用来实现验证的一种算法。

HTML 约束验证基于：

* **HTML 输入属性**
* **CSS 伪类选择器**
* **DOM 属性和方法**

### 约束验证 HTML 输入属性

|  |  |
| --- | --- |
| **属性** | **描述** |
| disabled | 规定输入的元素不可用 |
| max | 规定输入元素的最大值 |
| min | 规定输入元素的最小值 |
| pattern | 规定输入元素值的模式 |
| required | 规定输入元素字段是必需的 |
| type | 规定输入元素的类型 |

完整列表，请查看 [HTML 输入属性](http://www.runoob.com/html/html5-form-attributes.html)。

### 约束验证 CSS 伪类选择器

|  |  |
| --- | --- |
| **选择器** | **描述** |
| :disabled | 选取属性为 "disabled" 属性的 input 元素 |
| :invalid | 选取无效的 input 元素 |
| :optional | 选择没有"required"属性的 input 元素 |
| :required | 选择有"required"属性的 input 元素 |
| :valid | 选取有效值的 input 元素 |

完整列表，请查看 [CSS 伪类](http://www.runoob.com/css/css-pseudo-classes.html)。

[JavaScript 使用误区](http://www.runoob.com/js/js-mistakes.html)

[JavaScript 表单验证](http://www.runoob.com/js/js-form-validation.html)

## 1 篇笔记

1. 机械挨踢男

  219\*\*\*3799@qq.com

**关于多表单使用同一验证函数问题：**

上面用 js 表单验证判断表单字段(fname)值是否存在，如果想多个表单都使用同一个函数调用，传入参数后功能会失效，希望有大佬能解决这个问题。

现在我在网上找到的替代方法如下：

function validateForm(form) {

var x = form.name.value;

if (x == null || x == "") {

alert("输入不能为空！");

return false;

}

}

所有表单调用时都使用：

onsubmit="return validateForm(this)"

经验证好使。

[机械挨踢男](javascript:;)

   机械挨踢男

  219\*\*\*3799@qq.com

1个月前 (04-21)

# JavaScript 严格模式(use strict)

JavaScript 严格模式（strict mode）即在严格的条件下运行。

## 使用 "use strict" 指令

"use strict" 指令在 JavaScript 1.8.5 (ECMAScript5) 中新增。

它不是一条语句，但是是一个字面量表达式，在 JavaScript 旧版本中会被忽略。

"use strict" 的目的是指定代码在严格条件下执行。

严格模式下你不能使用未声明的变量。

|  |  |
| --- | --- |
| **Note** | 支持严格模式的浏览器: Internet Explorer 10 +、 Firefox 4+ Chrome 13+、 Safari 5.1+、 Opera 12+。 |

## 严格模式声明

严格模式通过在脚本或函数的头部添加 "use strict"; 表达式来声明。

实例中我们可以在浏览器按下 **F12 (或点击"工具>更多工具>开发者工具")** 开启调试模式，查看报错信息。

Gif 图演示如下：



### 实例

"use strict";  
x = 3.14;       // 报错 (x 未定义)

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=tryjs_strict_variable)

### 实例

"use strict";  
myFunction();  
  
function myFunction() {  
    y = 3.14;   // 报错 (y 未定义)  
}

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=tryjs_strict_global)

在函数内部声明是局部作用域 (只在函数内使用严格模式):

### 实例

x = 3.14;       // 不报错   
myFunction();  
  
function myFunction() {  
   "use strict";  
    y = 3.14;   // 报错 (y 未定义)  
}

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=tryjs_strict_local)

为什么使用严格模式:

消除Javascript语法的一些不合理、不严谨之处，减少一些怪异行为;

* 消除代码运行的一些不安全之处，保证代码运行的安全；
* 提高编译器效率，增加运行速度；
* 为未来新版本的Javascript做好铺垫。

"严格模式"体现了Javascript更合理、更安全、更严谨的发展方向，包括IE 10在内的主流浏览器，都已经支持它，许多大项目已经开始全面拥抱它。

另一方面，同样的代码，在"严格模式"中，可能会有不一样的运行结果；一些在"正常模式"下可以运行的语句，在"严格模式"下将不能运行。掌握这些内容，有助于更细致深入地理解Javascript，让你变成一个更好的程序员。

## 严格模式的限制

不允许使用未声明的变量：

"use strict";  
x = 3.14;                // 报错 (x 未定义)

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=tryjs_strict_variable)

|  |  |
| --- | --- |
| **Note** | 对象也是一个变量。 |

"use strict";  
x = {p1:10, p2:20};      // 报错 (x 未定义)

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=tryjs_strict_object)

不允许删除变量或对象。

"use strict";  
var x = 3.14;  
delete x;                // 报错

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=tryjs_strict_delete)

不允许删除函数。

"use strict";  
function x(p1, p2) {};   
delete x;                // 报错

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=tryjs_strict_delete_function)

不允许变量重名:

"use strict";  
function x(p1, p1) {};   // 报错

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=tryjs_strict_duplicate)

不允许使用八进制:

"use strict";  
var x = 010;             // 报错

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=tryjs_strict_octal)

不允许使用转义字符:

"use strict";  
var x = \010;            // 报错

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=tryjs_strict_escape)

不允许对只读属性赋值:

"use strict";  
var obj = {};  
Object.defineProperty(obj, "x", {value:0, writable:false});  
  
obj.x = 3.14;            // 报错

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=tryjs_strict_readonly)

不允许对一个使用getter方法读取的属性进行赋值

"use strict";  
var obj = {get x() {return 0} };  
  
obj.x = 3.14;            // 报错

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=tryjs_strict_getonly)

不允许删除一个不允许删除的属性：

"use strict";  
delete Object.prototype; // 报错

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=tryjs_strict_deleteprop)

变量名不能使用 "eval" 字符串:

"use strict";  
var eval = 3.14;         // 报错

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=tryjs_strict_eval)

变量名不能使用 "arguments" 字符串:

"use strict";  
var arguments = 3.14;    // 报错

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=tryjs_strict_arguments)

不允许使用以下这种语句:

"use strict";  
with (Math){x = cos(2)}; // 报错

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=tryjs_strict_width)

由于一些安全原因，在作用域 eval() 创建的变量不能被调用：

"use strict";  
eval ("var x = 2");  
alert (x);               // 报错

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=tryjs_strict_eval2)

禁止this关键字指向全局对象。

function f(){

    return !this;

}

// 返回false，因为"this"指向全局对象，"!this"就是false

function f(){

    "use strict";

    return !this;

}

// 返回true，因为严格模式下，this的值为undefined，所以"!this"为true。

因此，使用构造函数时，如果忘了加new，this不再指向全局对象，而是报错。

function f(){

    "use strict";

    this.a = 1;

};

f();// 报错，this未定义

## 保留关键字

为了向将来Javascript的新版本过渡，严格模式新增了一些保留关键字：

* implements
* interface
* let
* package
* private
* protected
* public
* static
* yield

"use strict";  
var public = 1500;      // 报错

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=tryjs_strict_names)

|  |  |
| --- | --- |
| **Note** | "use strict" 指令只允许出现在脚本或函数的开头。 |

# JavaScript 调试

在编写 JavaScript 时，如果没有调试工具将是一件很痛苦的事情。

## JavaScript 调试

没有调试工具是很难去编写 JavaScript 程序的。

你的代码可能包含语法错误，逻辑错误，如果没有调试工具，这些错误比较难于发现。

通常，如果 JavaScript 出现错误，是不会有提示信息，这样你就无法找到代码错误的位置。

|  |  |
| --- | --- |
| **Note** | 通常，你在编写一个新的 JavaScript 代码过程中都会发生错误。 |

## JavaScript 调试工具

在程序代码中寻找错误叫做代码调试。

调试很难，但幸运的是，很多浏览器都内置了调试工具。

内置的调试工具可以开始或关闭，严重的错误信息会发送给用户。

有了调试工具，我们就可以设置断点 (代码停止执行的位置), 且可以在代码执行时检测变量。

浏览器启用调试工具一般是按下 F12 键，并在调试菜单中选择 "Console" 。

## console.log() 方法

如果浏览器支持调试，你可以使用 console.log() 方法在调试窗口上打印 JavaScript 值：

## 实例

a = 5; b = 6; c = a + b; console.log(c);

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=tryjs_console)

## 设置断点

在调试窗口中，你可以设置 JavaScript 代码的断点。

在每个断点上，都会停止执行 JavaScript 代码，以便于我们检查 JavaScript 变量的值。

在检查完毕后，可以重新执行代码（如播放按钮）。

## debugger 关键字

**debugger** 关键字用于停止执行 JavaScript，并调用调试函数。

这个关键字与在调试工具中设置断点的效果是一样的。

如果没有调试可用，debugger 语句将无法工作。

开启 debugger ，代码在第三行前停止执行。

## 实例

var x = 15 \* 5; debugger; document.getElementbyId("demo").innerHTML = x;

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=tryjs_debugger)

## 主要浏览器的调试工具

通常，浏览器启用调试工具一般是按下 F12 键，并在调试菜单中选择 "Console" 。

各浏览器的步骤如下:

## Chrome 浏览器

* 打开浏览器。
* 在菜单中选择工具。
* 在工具中选择开发者工具。
* 最后，选择 Console。

## Firefox 浏览器

* 打开浏览器。
* 访问页面:   
  http://www.getfirebug.com。
* 按照说明 :  
  安装 Firebug。

## Internet Explorer 浏览器。

* 打开浏览器。
* 在菜单中选择工具。
* 在工具中选择开发者工具。
* 最后，选择 Console。

## Opera

* 打开浏览器。
* Opera 的内置调试工具为 Dragonfly，详细说明可访问页面:  
  http://www.opera.com/dragonfly/。

## Safari

* 打开浏览器。
* 右击鼠标，选择检查元素。
* 在底部弹出的窗口中选择"控制台"。

# JavaScript 错误 - throw、try 和 catch

**try** 语句测试代码块的错误。

**catch** 语句处理错误。

**throw** 语句创建自定义错误。

## JavaScript 错误

当 JavaScript 引擎执行 JavaScript 代码时，会发生各种错误。

可能是语法错误，通常是程序员造成的编码错误或错别字。

可能是拼写错误或语言中缺少的功能（可能由于浏览器差异）。

可能是由于来自服务器或用户的错误输出而导致的错误。

当然，也可能是由于许多其他不可预知的因素。

## JavaScript 抛出（throw）错误

当错误发生时，当事情出问题时，JavaScript 引擎通常会停止，并生成一个错误消息。

描述这种情况的技术术语是：JavaScript 将**抛出**一个错误。

## JavaScript try 和 catch

**try** 语句允许我们定义在执行时进行错误测试的代码块。

**catch** 语句允许我们定义当 try 代码块发生错误时，所执行的代码块。

JavaScript 语句 **try** 和 **catch** 是成对出现的。

### ****语法****

try { //在这里运行代码 } catch(err) { //在这里处理错误 }

## 实例

在下面的例子中，我们故意在 try 块的代码中写了一个错字。

catch 块会捕捉到 try 块中的错误，并执行代码来处理它。

## 实例

var txt=""; function message() { try { adddlert("Welcome guest!"); } catch(err) { txt="本页有一个错误。\n\n"; txt+="错误描述：" + err.message + "\n\n"; txt+="点击确定继续。\n\n"; alert(txt); } }

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=tryjs_try_catch)

## Throw 语句

throw 语句允许我们创建自定义错误。

正确的技术术语是：创建或**抛出异常**（exception）。

如果把 throw 与 try 和 catch 一起使用，那么您能够控制程序流，并生成自定义的错误消息。

### ****语法****

throw *exception*

异常可以是 JavaScript 字符串、数字、逻辑值或对象。

## 实例

本例检测输入变量的值。如果值是错误的，会抛出一个异常（错误）。catch 会捕捉到这个错误，并显示一段自定义的错误消息：

## 实例

function myFunction() { var message, x; message = document.getElementById("message"); message.innerHTML = ""; x = document.getElementById("demo").value; try { if(x == "") throw "值为空"; if(isNaN(x)) throw "不是数字"; x = Number(x); if(x < 5) throw "太小"; if(x > 10) throw "太大"; } catch(err) { message.innerHTML = "错误: " + err; } }

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=tryjs_throw_error)

请注意，如果 getElementById 函数出错，上面的例子也会抛出一个错误。