**JavaScript 教程**

JavaScript 是 Web 的编程语言。

所有现代的 HTML 页面都使用 JavaScript。

JavaScript 非常容易学。

本教程将教你学习从初级到高级JavaScript知识。

**JavaScript 在线实例**

本教程包含了大量的 JavaScript 实例， 您可以点击 "尝试一下" 来在线查看实例。

**JavaScript 在线实例**

本教程包含了大量的 JavaScript 实例， 您可以点击 "尝试一下" 来在线查看实例。

**实例**

**我的第一个 JavaScript 程序**

Sat Mar 19 2016 23:35:45 GMT+0800 (ä¸­å½æ åæ¶é´)

显示日期

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<title>菜鸟教程(runoob.com)</title>

<script>

function displayDate()

{

document.getElementById("demo").innerHTML=Date();

}

</script>

</head>

<body>

<h1>我的第一个 JavaScript 程序</h1>

<p id="demo">这是一个段落</p>

<button type="button" onclick="displayDate()">显示日期</button>

</body>

</html>

## 为什么学习 JavaScript?

JavaScript web 开发人员必须学习的 3 门语言中的一门：

1. **HTML** 定义了网页的内容
2. **CSS** 描述了网页的布局
3. **JavaScript** 网页的行为

本教程是关于 JavaScript 及介绍 JavaScript 如何与 HTML 和 CSS 一起工作。

## 谁适合阅读本教程?

1. 如果您想学习 JavaScript，您可以学习本教程：

了解 JavaScript 是如何与 HTML 和 CSS 一起工作的。

2. 如果在此之前您已经使用过 JavaScript，您也可以阅读本教程：

JavaScript 一直在升级，所以我们需要时刻了解 JavaScript 的新技术。

**阅读本教程前，您需要了解的知识：**

阅读本教程，您需要有以下基础：

* HTML 和 CSS 基础

如果您想学习这些基础知识，您可以在我们的首页找到相应的教程[菜鸟教程](http://www.runoob.com/)。

**JavaScript 实例**

学习 100 多个 JavaScript 实例！

在实例页面中，您可以点击 "尝试一下" 来查看 JavaScript 在线实例。

* [JavaScript 实例](http://www.runoob.com/js/js-examples.html)
* [JavaScript 对象实例](http://www.runoob.com/js/js-ex-objects.html)
* [JavaScript 浏览器支持实例](http://www.runoob.com/js/js-ex-browser.html)
* [JavaScript HTML DOM 实例](http://www.runoob.com/js/js-ex-dom.html)

**JavaScript 测验**

在 W3CSchool 中测试您的 JavaScript 技能！

**JavaScript 参考手册**

在 W3CSchool 中，我们为您提供完整的 JavaScript 对象、浏览器对象、HTML DOM 对象参考手册。

以下手册包含了每个对象、属性、方法的实例。

* [JavaScript 内置对象](http://www.runoob.com/jsref/jsref-tutorial.html)
* [Browser 对象](http://www.runoob.com/jsref/jsref-tutorial.html)
* [HTML DOM 对象](http://www.runoob.com/jsref/jsref-tutorial.html)

# JavaScript 简介

JavaScript 是互联网上最流行的脚本语言，这门语言可用于 HTML 和 web，更可广泛用于服务器、PC、笔记本电脑、平板电脑和智能手机等设备。

## JavaScript 是脚本语言

JavaScript 是一种轻量级的编程语言。

JavaScript 是可插入 HTML 页面的编程代码。

JavaScript 插入 HTML 页面后，可由所有的现代浏览器执行。

JavaScript 很容易学习。

## 您将学到什么

下面是您将在本教程中学到的主要内容。

## JavaScript：直接写入 HTML 输出流

## 实例

document.write("<h1>这是一个标题</h1>");  
document.write("<p>这是一个段落。</p>");

<!DOCTYPE html>

<html>

<body>

<p>

JavaScript 能够直接写入 HTML 输出流中：

</p>

<script>

document.write("<h1>这是一个标题</h1>");

document.write("<p>这是一个段落。</p>");

</script>

<p>

您只能在 HTML 输出流中使用 <strong>document.write</strong>。

如果您在文档已加载后使用它（比如在函数中），会覆盖整个文档。

</p>

</body>

</html>

您只能在 HTML 输出中使用 document.write。如果您在文档加载后使用该方法，会覆盖整个文档。

## JavaScript：对事件的反应

## 实例

<button type="button" onclick="alert('欢迎!')">点我!</button>

<!DOCTYPE html>

<html>

<body>

<h1>我的第一个 JavaScript</h1>

<p>

JavaScript 能够对事件作出反应。比如对按钮的点击：

</p>

<button type="button" onclick="alert('欢迎!')">点我!</button>

</body>

</html>

alert() 函数在 JavaScript 中并不常用，但它对于代码测试非常方便。

onclick 事件只是您即将在本教程中学到的众多事件之一。

## JavaScript：改变 HTML 内容

使用 JavaScript 来处理 HTML 内容是非常强大的功能。

## 实例

x=document.getElementById("demo")  //查找元素  
x.innerHTML="Hello JavaScript";    //改变内容

<!DOCTYPE html>

<html>

<body>

<h1>我的第一段 JavaScript</h1>

<p id="demo">

JavaScript 能改变 HTML 元素的内容。

</p>

<script>

function myFunction()

{

x=document.getElementById("demo"); // 找到元素

x.innerHTML="Hello JavaScript!"; // 改变内容

}

</script>

<button type="button" onclick="myFunction()">点击这里</button>

</body>

</html>

您会经常看到 **document.getElementById("*some id*")**。这个方法是 HTML DOM 中定义的。

DOM (**D**ocument **O**bject **M**odel)（文档对象模型）是用于访问 HTML 元素的正式 W3C 标准。

您将在本教程的多个章节中学到有关 HTML DOM 的知识。

## JavaScript：改变 HTML 图像

本例会动态地改变 HTML <image> 的来源（src）：

## 点亮灯泡

<!DOCTYPE html>

<html>

<body>

<script>

function changeImage()

{

element=document.getElementById('myimage')

if (element.src.match("bulbon"))

{

element.src="/images/pic\_bulboff.gif";

}

else

{

element.src="/images/pic\_bulbon.gif";

}

}

</script>

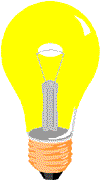
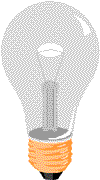
<img id="myimage" onclick="changeImage()"

src="/images/pic\_bulboff.gif" width="100" height="180">

<p>点击灯泡就可以打开或关闭这盏灯</p>

</body>

</html>



## JavaScript 能够改变任意 HTML 元素的大多数属性，而不仅仅是图片。JavaScript：改变 HTML 样式

改变 HTML 元素的样式，属于改变 HTML 属性的变种。

## 实例

x=document.getElementById("demo")  //找到元素   
x.style.color="#ff0000";           //改变样式

<!DOCTYPE html>

<html>

<body>

<h1>我的第一段 JavaScript</h1>

<p id="demo">

JavaScript 能改变 HTML 元素的样式。

</p>

<script>

function myFunction()

{

x=document.getElementById("demo") // 找到元素

x.style.color="#ff0000"; // 改变样式

}

</script>

<button type="button" onclick="myFunction()">点击这里</button>

</body>

</html>

## JavaScript：验证输入

JavaScript 常用于验证用户的输入。

## 实例

if isNaN(x) {alert("不是数字")};

<!DOCTYPE html>

<html>

<body>

<h1>我的第一段 JavaScript</h1>

<p>请输入数字。如果输入值不是数字，浏览器会弹出提示框。</p>

<input id="demo" type="text">

<script>

function myFunction()

{

var x=document.getElementById("demo").value;

if(x==""||isNaN(x))

{

alert("不是数字");

}

}

</script>

<button type="button" onclick="myFunction()">点击这里</button>

</body>

</html>

## 您知道吗？

|  |  |
| --- | --- |
| **lamp** | JavaScript 与 Java 是两种完全不同的语言，无论在概念上还是设计上。 Java（由 Sun 发明）是更复杂的编程语言。  ECMA-262 是 JavaScript 标准的官方名称。  JavaScript 由 Brendan Eich 发明。它于 1995 年出现在 Netscape 中（该浏览器已停止更新），并于 1997 年被 ECMA（一个标准协会）采纳。 |

# JavaScript 用法

HTML 中的脚本必须位于 <script> 与 </script> 标签之间。

脚本可被放置在 HTML 页面的 <body> 和 <head> 部分中。

## <script> 标签

如需在 HTML 页面中插入 JavaScript，请使用 <script> 标签。

<script> 和 </script> 会告诉 JavaScript 在何处开始和结束。

<script> 和 </script> 之间的代码行包含了 JavaScript:

<script>  
alert("我的第一个 JavaScript");  
</script>

您无需理解上面的代码。只需明白，浏览器会解释并执行位于 <script> 和 </script>之间的 JavaScript 代码

|  |  |
| --- | --- |
| **lamp** | 那些老旧的实例可能会在 <script> 标签中使用 type="text/javascript"。现在已经不必这样做了。JavaScript 是所有现代浏览器以及 HTML5 中的默认脚本语言。 |

## <body> 中的 JavaScript

在本例中，JavaScript 会在页面加载时向 HTML 的 <body> 写文本：

<!DOCTYPE html>

<html>

<body>

<p>

JavaScript 能够直接写入 HTML 输出流中：

</p>

<script>

document.write("<h1>这是一个标题</h1>");

document.write("<p>这是一个段落。</p>");

</script>

<p>

您只能在 HTML 输出流中使用 <strong>document.write</strong>。

如果您在文档已加载后使用它（比如在函数中），会覆盖整个文档。

</p>

</body>

</html>

## JavaScript 函数和事件

上面例子中的 JavaScript 语句，会在页面加载时执行。

通常，我们需要在某个事件发生时执行代码，比如当用户点击按钮时。

如果我们把 JavaScript 代码放入函数中，就可以在事件发生时调用该函数。

您将在稍后的章节学到更多有关 JavaScript 函数和事件的知识。

## 在 <head> 或者 <body> 的JavaScript

您可以在 HTML 文档中放入不限数量的脚本。

脚本可位于 HTML 的 <body> 或 <head> 部分中，或者同时存在于两个部分中。

通常的做法是把函数放入 <head> 部分中，或者放在页面底部。这样就可以把它们安置到同一处位置，不会干扰页面的内容。

## <head> 中的 JavaScript 函数

在本例中，我们把一个 JavaScript 函数放置到 HTML 页面的 <head> 部分。

该函数会在点击按钮时被调用：

<!DOCTYPE html>

<html>

<head>

<script>

function myFunction()

{

document.getElementById("demo").innerHTML="我的第一个 JavaScript 函数";

}

</script>

</head>

<body>

<h1>我的 Web 页面</h1>

<p id="demo">一个段落。</p>

<button type="button" onclick="myFunction()">点击这里</button>

</body>

</html>

## <body> 中的 JavaScript 函数

在本例中，我们把一个 JavaScript 函数放置到 HTML 页面的 <body> 部分。

该函数会在点击按钮时被调用：

<!DOCTYPE html>

<html>

<body>

<h1>我的第一个 Web 页面</h1>

<p id="demo">一个段落。</p>

<button type="button" onclick="myFunction()">点击这里</button>

<script>

function myFunction()

{

document.getElementById("demo").innerHTML="我的第一个 JavaScript 函数";

}

</script>

</body>

</html>

## 外部的 JavaScript

也可以把脚本保存到外部文件中。外部文件通常包含被多个网页使用的代码。

外部 JavaScript 文件的文件扩展名是 .js。

如需使用外部文件，请在 <script> 标签的 "src" 属性中设置该 .js 文件：

<!DOCTYPE html>

<html>

<body>

<h1>我的 Web 页面</h1>

<p id="demo">一个段落。</p>

<button type="button" onclick="myFunction()">点击这里</button>

<p><b>注释：</b>myFunction 保存在名为 "myScript.js" 的外部文件中。</p>

<script src="myScript.js"></script>

</body>

</html>

你可以将脚本放置于 <head> 或者 <body>中 实际运行效果与您在 <script> 标签中编写脚本完全一致。

myScript.js 文件代码如下：

function myFunction()

{

document.getElementById("demo").innerHTML="我的第一个 JavaScript 函数";

}

|  |  |
| --- | --- |
| **lamp** | 外部脚本不能包含 <script> 标签。 |

**JavaScript 输出**

JavaScript 没有任何打印或者输出的函数。

**JavaScript 显示数据**

JavaScript 可以通过不同的方式来输出数据：

* 使用 **window.alert()** 弹出警告框。
* 使用 **document.write()** 方法将内容写到 HTML 文档中。
* 使用 **innerHTML** 写入到 HTML 元素。
* 使用 **console.log()** 写入到浏览器的控制台。

## 使用 window.alert()

你可以弹出警告框来显示数据：

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<title>菜鸟教程(runoob.com)</title>

</head>

<body>

<h1>我的第一个页面</h1>

<p>我的第一个段落。</p>

<script>

window.alert(5 + 6);

</script>

</body>

</html>

## 操作 HTML 元素

如需从 JavaScript 访问某个 HTML 元素，您可以使用 document.getElementById(*id*) 方法。

请使用 "id" 属性来标识 HTML 元素，并 innerHTML 来获取或插入元素内容：

## 实例

<!DOCTYPE html>

<html>

<body>

<h1>我的第一个 Web 页面</h1>

<p id="demo">我的第一个段落。</p>

<script>

document.getElementById("demo").innerHTML="段落已修改。";

</script>

</body>

</html>

以上 JavaScript 语句（在 <script> 标签中）可以在 web 浏览器中执行：

**document.getElementById("demo")** 是使用 id 属性来查找 HTML 元素的 JavaScript 代码 。

**innerHTML = "段落已修改。"** 是用于修改元素的 HTML 内容(innerHTML)的 JavaScript 代码。

## 在本教程中

在大多数情况下，在本教程中，我们将使用上面描述的方法来输出：

下面的例子直接把 id="demo" 的 <p> 元素写到 HTML 文档输出中：

## 写到 HTML 文档

出于测试目的，您可以将JavaScript直接写在HTML 文档中：

## 实例

<!DOCTYPE html>

<html>

<body>

<h1>我的第一个 Web 页面</h1>

<p>我的第一个段落。</p>

<script>

document.write(Date());

</script>

</body>

</html>

请使用 document.write() 仅仅向文档输出写内容。

如果在文档已完成加载后执行 document.write，整个 HTML 页面将被覆盖。

<!DOCTYPE html>

<html>

<body>

<h1>我的第一个 Web 页面</h1>

<p>我的第一个段落。</p>

<button onclick="myFunction()">点我</button>

<script>

function myFunction() {

document.write(Date());

}

</script>

</body>

</html>

## 写到控制台

如果您的浏览器支持调试，你可以使用 **console.log()** 方法在浏览器中显示 JavaScript 值。

浏览器中使用 F12 来启用调试模式， 在调试窗口中点击 "Console" 菜单。

## 实例

<!DOCTYPE html>

<html>

<body>

<h1>我的第一个 Web 页面</h1>

<p>

浏览器中(Chrome, IE, Firefox) 使用 F12 来启用调试模式， 在调试窗口中点击 "Console" 菜单。

</p>

<script>

a = 5;

b = 6;

c = a + b;

console.log(c);

</script>

</body>

</html>

实例 console 截图：



## 您知道吗?

|  |  |
| --- | --- |
| **Note** | 程序中调试是测试，查找及减少bug(错误)的过程。 |

# avaScript 语法

JavaScript 是一个程序语言。语法规则定义了语言结构。

## JavaScript 语法

JavaScript 是一个脚本语言。

它是一个轻量级，但功能强大的编程语言。

## JavaScript 字面量

在编程语言中，一个字面量是一个常量，如 3.14。

**数字（Number）字面量** 可以是整数或者是小数，或者是科学计数(e)。

<!DOCTYPE html>

<html>

<body>

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

<script>

document.getElementById("demo").innerHTML = 123e5;

</script>

</body>

</html>

3.14  
  
1001  
  
123e5

**字符串（String）字面量** 可以使用单引号或双引号 be written with double or single quotes:

"John Doe"  
  
'John Doe'

<!DOCTYPE html>

<html>

<body>

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

<script>

document.getElementById("demo").innerHTML = 'John Doe';

</script>

</body>

</html>

**表达式字面量** 用于计算：

5 + 6  
  
5 \* 10

<!DOCTYPE html>

<html>

<body>

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

<script>

document.getElementById("demo").innerHTML = 5 \* 10;

</script>

</body>

</html>

**数组（Array）字面量** 定义一个数组：

[40, 100, 1, 5, 25, 10]

**对象（Object）字面量** 定义一个对象：

{firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"}

**函数（Function）字面量** 定义一个函数：

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

## JavaScript 变量

在编程语言中，变量用于存储数据值。

JavaScript 使用关键字 **var** 来定义变量， 使用等号来为变量赋值：

var x, length  
  
x = 5  
  
length = 6

<!DOCTYPE html>

<html>

<body>

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

<script>

var length;

length = 6;

document.getElementById("demo").innerHTML = length;

</script>

</body>

</html>

变量可以通过变量名访问。在指令式语言中，变量通常是可变的。字面量是一个恒定的值。

|  |  |
| --- | --- |
| **Note** | 变量是一个**名称**。字面量是一个**值**。 |

## JavaScript 操作符

JavaScript使用 **算术运算符** 来计算值:

(5 + 6) \* 10

<!DOCTYPE html>

<html>

<body>

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

<script>

document.getElementById("demo").innerHTML = (5 + 6) \* 10;

</script>

</body>

</html>

JavaScript使用**赋值运算符**给变量赋值：

x = 5  
y = 6  
z = (x + y) \* 10

<!DOCTYPE html>

<html>

<body>

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

<script>

var x, y, z;

x = 5

y = 6;

z = (x + y) \* 10;

document.getElementById("demo").innerHTML = z;

</script>

</body>

</html>

JavaScript语言有多种类型的运算符：

|  |  |  |
| --- | --- | --- |
| **类型** | **实例** | **描述** |
| 赋值，算术和位运算符 | =  +  -  \*  / | 在 JS 运算符中描述 |
| 条件，比较及逻辑运算符 | ==  != <  > | 在 JS 比较运算符中描述 |

## JavaScript 语句

在 HTML 中，JavaScript 语句向浏览器发出的命令。

语句是用分号分隔：

x = 5 + 6;  
y = x \* 10;

## JavaScript 关键词

JavaScript 语句通常于**关键词**为开头。 **var** 关键词告诉浏览器创建一个新的变量：

var x = 5 + 6;  
var y = x \* 10;

## JavaScript 关键字

和其他任何编程语言一样，JavaScript 保留了一些关键字为自己所用。

JavaScript 同样保留了一些关键字，这些关键字在当前的语言版本中并没有使用，但在以后 JavaScript 扩展中会用到。

JavaScript 关键字必须以字母、下划线（\_）或美元符（$）开始。

后续的字符可以是字母、数字、下划线或美元符（数字是不允许作为首字符出现的，以便 JavaScript 可以轻易区分开关键字和数字）。

以下是 JavaScript 中最​​重要的保留字（按字母顺序）：

|  |  |  |  |
| --- | --- | --- | --- |
| abstract | else | instanceof | super |
|  |  |  |  |
| boolean | enum | int | switch |
|  |  |  |  |
| break | export | interface | synchronized |
|  |  |  |  |
| byte | extends | let | this |
|  |  |  |  |
| case | false | long | throw |
|  |  |  |  |
| catch | final | native | throws |
|  |  |  |  |
| char | finally | new | transient |
|  |  |  |  |
| class | float | null | true |
|  |  |  |  |
| const | for | package | try |
|  |  |  |  |
| continue | function | private | typeof |
|  |  |  |  |
| debugger | goto | protected | var |
|  |  |  |  |
| default | if | public | void |
|  |  |  |  |
| delete | implements | return | volatile |
|  |  |  |  |
| do | import | short | while |
|  |  |  |  |
| double | in | static | with |
|  |  |  |  |

## JavaScript 注释

不是所有的 JavaScript 语句都是"命令"。双斜杠 **//** 后的内容将会被浏览器忽略：

// 我不会执行

## JavaScript 数据类型

JavaScript 有多种数据类型：数字，字符串，数组，对象等等：

var length = 16;                                  // Number 通过数字字面量赋值   
var points = x \* 10;                              // Number 通过表达式字面量赋值  
var lastName = "Johnson";                         // String 通过字符串字面量赋值  
var cars = ["Saab", "Volvo", "BMW"];              // Array  通过数组字面量赋值  
var person = {firstName:"John", lastName:"Doe"};  // Object 通过对象字面量赋值

## 数据类型的概念

编程语言中，数据类型是一个非常重要的内容。

为了可以操作变量，了解数据类型的概念非常重要。

如果没有使用数据类型，以下实例将无法执行：

16 + "Volvo"

16 加上 "Volvo" 是如何计算呢? 以上会产生一个错误还是输出以下结果呢？

"16Volvo"

你可以在浏览器尝试执行以上代码查看效果。

在接下来的章节中你将学到更多关于数据类型的知识。

## JavaScript 函数

JavaScript 语句可以写在函数内，函数可以重复引用：

**引用一个函数** = 调用函数(执行函数内的语句)。

function myFunction(a, b) {  
    return a \* b;                                // 返回 a 乘于 b 的结果  
}

## JavaScript 对大小写敏感。

JavaScript 对大小写是敏感的。

当编写 JavaScript 语句时，请留意是否关闭大小写切换键。

函数 **getElementById** 与 **getElementbyID** 是不同的。

同样，变量 **myVariable** 与 **MyVariable** 也是不同的。

## JavaScript 字符集

JavaScript 使用 Unicode 字符集。

Unicode 覆盖了所有的字符，包含标点等字符。

如需进一步了解，请学习我们的 [完整 Unicode 参考手册](http://www.runoob.com/charsets/ref-html-utf8.html)。

## 您知道吗?

|  |  |
| --- | --- |
| **Note** | JavaScript 中，常见的是驼峰法的命名规则，如 lastName (而不是lastname)。 |

# JavaScript 语句

JavaScript 语句向浏览器发出的命令。语句的作用是告诉浏览器该做什么。

## JavaScript 语句

JavaScript 语句是发给浏览器的命令。

这些命令的作用是告诉浏览器要做的事情。

下面的 JavaScript 语句向 id="demo" 的 HTML 元素输出文本 "你好 Dolly" ：

## 实例

document.getElementById("demo").innerHTML = "Hello Dolly.";

<!DOCTYPE html>

<html>

<body>

<h1>我的网页</h1>

<p id="demo">我的第一个段落。</p>

<script>

document.getElementById("demo").innerHTML = "你好 Dolly.";

</script>

</body>

</html>

## 分号 ;

分号用于分隔 JavaScript 语句。

通常我们在每条可执行的语句结尾添加分号。

使用分号的另一用处是在一行中编写多条语句。

Writing:

a = 5;  
b = 6;  
c = a + b;

Is the same as writing:

a = 5; b = 6; c = a + b;

<!DOCTYPE html>

<html>

<body>

<h1>我的网页</h1>

<p id="demo1"></p>

<p id="demo2"></p>

<script>

a = 1;

b = 2;

c = a + b;

document.getElementById("demo1").innerHTML = c;

x = 1; y = 2; z = x + y;

document.getElementById("demo2").innerHTML = z;

</script>

</body>

</html>

|  |  |
| --- | --- |
| **lamp** | 您也可能看到不带有分号的案例。  在 JavaScript 中，用分号来结束语句是可选的。 |

## JavaScript 代码

JavaScript 代码是 JavaScript 语句的序列。

浏览器按照编写顺序依次执行每条语句。

本例向网页输出一个标题和两个段落：

## 实例

document.getElementById("demo").innerHTML="你好 Dolly";  
document.getElementById("myDIV").innerHTML="你最近怎么样?";

<!DOCTYPE html>

<html>

<body>

<h1>我的 Web 页面</h1>

<p id="demo">一个段落。</p>

<div id="myDIV">一个 DIV。</div>

<script>

document.getElementById("demo").innerHTML="你好 Dolly";

document.getElementById("myDIV").innerHTML="你最近怎么样?";

</script>

</body>

</html>

## JavaScript 代码块

JavaScript 可以分批地组合起来。

代码块以左花括号开始，以右花括号结束。

代码块的作用是一并地执行语句序列。

本例向网页输出一个标题和两个段落：

## 实例

function myFunction()  
{  
document.getElementById("demo").innerHTML="你好Dolly";  
document.getElementById("myDIV").innerHTML="你最近怎么样?";  
}

<!DOCTYPE html>

<html>

<body>

<h1>我的 Web 页面</h1>

<p id="myPar">我是一个段落。</p>

<div id="myDiv">我是一个div。</div>

<p>

<button type="button" onclick="myFunction()">点击这里</button>

</p>

<script>

function myFunction()

{

document.getElementById("myPar").innerHTML="你好世界";

document.getElementById("myDiv").innerHTML="你最近怎么样?";

}

</script>

<p>当您点击上面的按钮时，两个元素会改变。</p>

</body>

</html>

您将在稍后的章节学到更多有关函数的知识。

## JavaScript 语句标识符

JavaScript 语句通常以一个 **语句标识符** 为开始，并执行该语句。

语句标识符是保留关键字不能作为变量名使用。

下表列出了 JavaScript 语句标识符 (关键字) ：

|  |  |
| --- | --- |
| **语句** | **描述** |
| break | 用于跳出循环。 |
| catch | 语句块，在 try 语句块执行出错时执行 catch 语句块。 |
| continue | 跳过循环中的一个迭代。 |
| do ... while | 执行一个语句块，在条件语句为 true 时继续执行该语句块。 |
| for | 在条件语句为 true 时，可以将代码块执行指定的次数。 |
| for ... in | 用于遍历数组或者对象的属性（对数组或者对象的属性进行循环操作）。 |
| function | 定义一个函数 |
| if ... else | 用于基于不同的条件来执行不同的动作。 |
| return | 退出函数 |
| switch | 用于基于不同的条件来执行不同的动作。 |
| throw | 抛出（生成）错误 。 |
| try | 实现错误处理，与 catch 一同使用。 |
| var | 声明一个变量。 |
| while | 当条件语句为 true 时，执行语句块。 |

## JavaScript 对大小写敏感。

JavaScript 对大小写是敏感的。

当编写 JavaScript 语句时，请留意是否关闭大小写切换键。

函数 getElementById 与 getElementbyID 是不同的。

同样，变量 myVariable 与 MyVariable 也是不同的。

## 空格

JavaScript 会忽略多余的空格。您可以向脚本添加空格，来提高其可读性。下面的两行代码是等效的：

var person="Hege";  
var person = "Hege";

## 对代码行进行折行

您可以在文本字符串中使用反斜杠对代码行进行换行。下面的例子会正确地显示：

document.write("你好 \  
世界!");

不过，您不能像这样折行：

document.write \   
("你好世界!");

## 您知道吗？

提示：JavaScript 是脚本语言。浏览器会在读取代码时，逐行地执行脚本代码。而对于传统编程来说，会在执行前对所有代码进行编译。

# JavaScript 注释

JavaScript 注释可用于提高代码的可读性。

## JavaScript 注释

JavaScript 不会执行注释。

我们可以添加注释来对 JavaScript 进行解释，或者提高代码的可读性。

单行注释以 // 开头。

本例用单行注释来解释代码：

## 实例

// 输出标题：  
document.getElementById("myH1").innerHTML="欢迎来到我的主页";  
// 输出段落：  
document.getElementById("myP").innerHTML="这是我的第一个段落。";

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=tryjs_comments1)

<!DOCTYPE html>

<html>

<body>

<h1 id="myH1"></h1>

<p id="myP"></p>

<script>

// 输出标题：

document.getElementById("myH1").innerHTML="Welcome to my Homepage";

// 输出段落：

document.getElementById("myP").innerHTML="This is my first paragraph.";

</script>

<p><b>注释：</b>注释不会被执行。</p>

</body>

</html>

## JavaScript 多行注释

多行注释以 /\* 开始，以 \*/ 结尾。

下面的例子使用多行注释来解释代码：

## 实例

/\*  
下面的这些代码会输出  
一个标题和一个段落  
并将代表主页的开始  
\*/  
document.getElementById("myH1").innerHTML="欢迎来到我的主页";  
document.getElementById("myP").innerHTML="这是我的第一个段落。";

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=tryjs_comments2)

<!DOCTYPE html>

<html>

<head>

<title>菜鸟教程测试实例</title>

<meta charset="utf-8">

</head>

<body>

<h1 id="myH1"></h1>

<p id="myP"></p>

<script>

/\*

下面的这些代码会输出

一个标题和一个段落

并将代表主页的开始

\*/

document.getElementById("myH1").innerHTML="欢迎来到菜鸟教程";

document.getElementById("myP").innerHTML="这是一个段落。";

</script>

<p><b>注释：</b>注释块不会被执行。</p>

</body>

</html>

## 使用注释来阻止执行

在下面的例子中，注释用于阻止其中一条代码行的执行（可用于调试）：

## 实例

//document.getElementById("myH1").innerHTML="欢迎来到我的主页";  
document.getElementById("myP").innerHTML="这是我的第一个段落。";

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=tryjs_comments3)

<!DOCTYPE html>

<html>

<body>

<h1 id="myH1"></h1>

<p id="myP"></p>

<script>

//document.getElementById("myH1").innerHTML="Welcome to my Homepage";

document.getElementById("myP").innerHTML="This is my first paragraph.";

</script>

<p><strong>Note:</strong> The comment is not executed.</p>

</body>

</html>

在下面的例子中，注释用于阻止代码块的执行（可用于调试）：

## 实例

/\*  
document.getElementById("myH1").innerHTML="欢迎来到我的主页";  
document.getElementById("myP").innerHTML="这是我的第一个段落。";  
\*/

<!DOCTYPE html>

<html>

<body>

<h1 id="myH1"></h1>

<p id="myP"></p>

<script>

/\*

document.getElementById("myH1").innerHTML="Welcome to my Homepage";

document.getElementById("myP").innerHTML="This is my first paragraph.";

\*/

</script>

<p><strong>Note:</strong> The comment-block is not executed.</p>

</body>

</html>

## 在行末使用注释

在下面的例子中，我们把注释放到代码行的结尾处：

## 实例

var x=5;    // 声明 x 并把 5 赋值给它  
var y=x+2;  // 声明 y 并把 x+2 赋值给它

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=tryjs_comments5)

<!DOCTYPE html>

<html>

<body>

<p id="myP"></p>

<script>

var x=5; // 声明 x 并把 5 赋值给它

var y=x+2; // 声明 y 并把 x+2 赋值给它

document.getElementById("myP").innerHTML=y // 把 y 的值写到 myP

</script>

<p><b>注释：</b>注释不会被执行。</p>

</body>

</html>

# JavaScript 变量

变量是用于存储信息的"容器"。

## 实例

var x=5;  
var y=6;  
var z=x+y;

<!DOCTYPE html>

<html>

<body>

<script>

var x=5;

var y=6;

var z=x+y;

document.write(x + "<br>");

document.write(y + "<br>");

document.write(z + "<br>");

</script>

</body>

</html>

## 就像代数那样

x=5  
y=6  
z=x+y

在代数中，我们使用字母（比如 x）来保存值（比如 5）。

通过上面的表达式 z=x+y，我们能够计算出 z 的值为 11。

在 JavaScript 中，这些字母被称为变量。

|  |  |
| --- | --- |
| **lamp** | 您可以把变量看做存储数据的容器。 |

## JavaScript 变量

与代数一样，JavaScript 变量可用于存放值（比如 x=5）和表达式（比如 z=x+y）。

变量可以使用短名称（比如 x 和 y），也可以使用描述性更好的名称（比如 age, sum, totalvolume）。

* 变量必须以字母开头
* 变量也能以 $ 和 \_ 符号开头（不过我们不推荐这么做）
* 变量名称对大小写敏感（y 和 Y 是不同的变量）

|  |  |
| --- | --- |
| **lamp** | JavaScript 语句和 JavaScript 变量都对大小写敏感。 |

## JavaScript 数据类型

JavaScript 变量还能保存其他数据类型，比如文本值 (name="Bill Gates")。

在 JavaScript 中，类似 "Bill Gates" 这样一条文本被称为字符串。

JavaScript 变量有很多种类型，但是现在，我们只关注数字和字符串。

当您向变量分配文本值时，应该用双引号或单引号包围这个值。

当您向变量赋的值是数值时，不要使用引号。如果您用引号包围数值，该值会被作为文本来处理。

## 实例

var pi=3.14;  
var person="John Doe";  
var answer='Yes I am!';

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=tryjs_data2)

<!DOCTYPE html>

<html>

<body>

<script>

var pi=3.14;

var name="Bill Gates";

var answer='Yes I am!';

document.write(pi + "<br>");

document.write(name + "<br>");

document.write(answer + "<br>");

</script>

</body>

</html>

## 声明（创建） JavaScript 变量

在 JavaScript 中创建变量通常称为"声明"变量。

我们使用 var 关键词来声明变量：

var carname;

变量声明之后，该变量是空的（它没有值）。

如需向变量赋值，请使用等号：

carname="Volvo";

不过，您也可以在声明变量时对其赋值：

var carname="Volvo";

在下面的例子中，我们创建了名为 carname 的变量，并向其赋值 "Volvo"，然后把它放入 id="demo" 的 HTML 段落中：

## 实例

<p id="demo"></p>  
var carname="Volvo";  
document.getElementById("demo").innerHTML=carname;

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=tryjs_variables1)

<!DOCTYPE html>

<html>

<body>

<p>点击这里来创建变量，并显示结果。</p>

<button onclick="myFunction()">点击这里</button>

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

<script>

function myFunction()

{

var carname="Volvo";

document.getElementById("demo").innerHTML=carname;

}

</script>

</body>

</html>

|  |  |
| --- | --- |
| **lamp** | 一个好的编程习惯是，在代码开始处，统一对需要的变量进行声明。 |

## 一条语句，多个变量

您可以在一条语句中声明很多变量。该语句以 var 开头，并使用逗号分隔变量即可：

var lastname="Doe", age=30, job="carpenter";

声明也可横跨多行：

var lastname="Doe",  
age=30,  
job="carpenter";

## Value = undefined

在计算机程序中，经常会声明无值的变量。未使用值来声明的变量，其值实际上是 undefined。

在执行过以下语句后，变量 carname 的值将是 undefined：

var carname;

## 重新声明 JavaScript 变量

如果重新声明 JavaScript 变量，该变量的值不会丢失：

在以下两条语句执行后，变量 carname 的值依然是 "Volvo"：

var carname="Volvo";   
var carname;

## JavaScript 算数

您可以通过 JavaScript 变量来做算数，使用的是 = 和 + 这类运算符：

## 实例

y=5;  
x=y+2;

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

您将在本教程稍后的章节学到更多有关 JavaScript 运算符的知识。

<!DOCTYPE html>

<html>

<body>

<p>假设 y=5，计算 x=y+2，并显示结果。</p>

<button onclick="myFunction()">点击这里</button>

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

<script>

function myFunction()

{

var y=5;

var x=y+2;

var demoP=document.getElementById("demo")

demoP.innerHTML="x=" + x;

}

</script>

</body>

</html>

# JavaScript 数据类型

字符串（String）、数字(Number)、布尔(Boolean)、数组(Array)、对象(Object)、空（Null）、未定义（Undefined）。

## JavaScript 拥有动态类型

JavaScript 拥有动态类型。这意味着相同的变量可用作不同的类型：

## 实例

var x;               // x 为 undefined  
var x = 5;           // 现在 x 为数字  
var x = "John";      // 现在 x 为字符串

## JavaScript 字符串

字符串是存储字符（比如 "Bill Gates"）的变量。

字符串可以是引号中的任意文本。您可以使用单引号或双引号：

## 实例

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

您可以在字符串中使用引号，只要不匹配包围字符串的引号即可：

## 实例

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

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

<!DOCTYPE html>

<html>

<body>

<script>

var carname1="Volvo XC60";

var carname2='Volvo XC60';

var answer1="It's alright";

var answer2="He is called 'Johnny'";

var answer3='He is called "Johnny"';

document.write(carname1 + "<br>")

document.write(carname2 + "<br>")

document.write(answer1 + "<br>")

document.write(answer2 + "<br>")

document.write(answer3 + "<br>")

</script>

</body>

</html>

您将在本教程的高级部分学到更多关于字符串的知识。

## JavaScript 数字

JavaScript 只有一种数字类型。数字可以带小数点，也可以不带：

## 实例

var x1=34.00;      //使用小数点来写  
var x2=34;         // //不使用小数点来写

极大或极小的数字可以通过科学（指数）计数法来书写：

## 实例

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

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

<!DOCTYPE html>

<html>

<body>

<script>

var x1=34.00;

var x2=34;

var y=123e5;

var z=123e-5;

document.write(x1 + "<br>")

document.write(x2 + "<br>")

document.write(y + "<br>")

document.write(z + "<br>")

</script>

</body>

</html>

您将在本教程的高级部分学到更多关于数字的知识。

## JavaScript 布尔

布尔（逻辑）只能有两个值：true 或 false。

var x=true;  
var y=false;

布尔常用在条件测试中。您将在本教程稍后的章节中学到更多关于条件测试的知识。

## JavaScript 数组

下面的代码创建名为 cars 的数组：

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

或者 (condensed array):

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

或者 (literal array):

## 实例

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

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

<!DOCTYPE html>

<html>

<body>

<script>

var i;

var cars = new Array();

cars[0] = "Saab";

cars[1] = "Volvo";

cars[2] = "BMW";

for (i=0;i<cars.length;i++)

{

document.write(cars[i] + "<br>");

}

</script>

</body>

</html>

数组下标是基于零的，所以第一个项目是 [0]，第二个是 [1]，以此类推。

您将在本教程稍后的章节中学到更多关于数组的知识。

## JavaScript 对象

对象由花括号分隔。在括号内部，对象的属性以名称和值对的形式 (name : value) 来定义。属性由逗号分隔：

var person={firstname:"John", lastname:"Doe", id:5566};

上面例子中的对象 (person) 有三个属性：firstname、lastname 以及 id。

空格和折行无关紧要。声明可横跨多行：

var person={  
firstname : "John",  
lastname  : "Doe",  
id        :  5566  
};

对象属性有两种寻址方式：

## 实例

name=person.lastname;  
name=person["lastname"];

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

您将在本教程稍后的章节中学到更多关于对象的知识。

## Undefined 和 Null

Undefined 这个值表示变量不含有值。

可以通过将变量的值设置为 null 来清空变量。

## 实例

cars=null;  
person=null;

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

<!DOCTYPE html>

<html>

<body>

<script>

var person;

var car="Volvo";

document.write(person + "<br>");

document.write(car + "<br>");

var car=null

document.write(car + "<br>");

</script>

</body>

</html>

## 声明变量类型

当您声明新变量时，可以使用关键词 "new" 来声明其类型：

var carname=new String;  
var x=      new Number;  
var y=      new Boolean;  
var cars=   new Array;  
var person= new Object;

|  |  |
| --- | --- |
| **lamp** | JavaScript 变量均为对象。当您声明一个变量时，就创建了一个新的对象。 |

# 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)

<!DOCTYPE html>

<html>

<body>

<p>创建 JavaScript 对象。</p>

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

<script>

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

document.getElementById("demo").innerHTML =

person.firstName + " is " + person.age + " years old.";

</script>

</body>

</html>

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"  
};

<!DOCTYPE html>

<html>

<body>

<p>创建 JavaScript 对象。</p>

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

<script>

var person = {

firstName : "John",

lastName : "Doe",

age : 50,

eyeColor : "blue"

};

document.getElementById("demo").innerHTML =

person.firstName + " is " + person.age + " years old.";

</script>

</body>

</html>

## 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)

<!DOCTYPE html>

<html>

<body>

<p>

有两种方式可以访问对象属性：

</p>

<p>

你可以使用 .property 或 ["property"].

</p>

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

<script>

var person = {

firstName: "John",

lastName : "Doe",

id : 5566

};

document.getElementById("demo").innerHTML =

person.firstName + " " + person.lastName;

</script>

</body>

</html>

## 实例 2

person["lastName"];

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

<!DOCTYPE html>

<html>

<body>

<p>

有两种方式可以访问对象属性:

</p>

<p>

你可以使用 .property 或 ["property"]。

</p>

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

<script>

var person = {

firstName: "John",

lastName : "Doe",

id : 5566

};

document.getElementById("demo").innerHTML =

person["firstName"] + " " + person["lastName"];

</script>

</body>

</html>

## 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)

<!DOCTYPE html>

<html>

<body>

<p>创建和使用对象方法。</p>

<p>对象方法作为一个函数定义存储在对象属性中。</p>

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

<script>

var person = {

firstName: "John",

lastName : "Doe",

id : 5566,

fullName : function() {

return this.firstName + " " + this.lastName;

}

};

document.getElementById("demo").innerHTML = person.fullName();

</script>

</body>

</html>

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)

<!DOCTYPE html>

<html>

<body>

<p>创建和使用对象方法。</p>

<p>对象方法是一个函数定义,并作为一个属性值存储。</p>

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

<script>

var person = {

firstName: "John",

lastName : "Doe",

id : 5566,

fullName : function() {

return this.firstName + " " + this.lastName;

}

};

document.getElementById("demo").innerHTML = person.fullName;

</script>

</body>

</html>

|  |  |
| --- | --- |
| **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)

1.

<!DOCTYPE html>

<html>

<body>

<p>创建 JavaScript 对象。</p>

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

<script>

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

document.getElementById("demo").innerHTML =

person.firstName + " is " + person.age + " years old.";

</script>

</body>

</html>

2.

<!DOCTYPE html>

<html>

<body>

<p>创建 JavaScript 对象。</p>

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

<script>

var person = {

firstName : "John",

lastName : "Doe",

age : 50,

eyeColor : "blue"

};

document.getElementById("demo").innerHTML =

person.firstName + " is " + person.age + " years old.";

</script>

</body>

</html>

3.

<!DOCTYPE html>

<html>

<body>

<p>

有两种方式可以访问对象属性：

</p>

<p>

你可以使用 .property 或 ["property"].

</p>

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

<script>

var person = {

firstName: "John",

lastName : "Doe",

id : 5566

};

document.getElementById("demo").innerHTML =

person.firstName + " " + person.lastName;

</script>

</body>

</html>

4.

<!DOCTYPE html>

<html>

<body>

<p>

有两种方式可以访问对象属性:

</p>

<p>

你可以使用 .property 或 ["property"]。

</p>

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

<script>

var person = {

firstName: "John",

lastName : "Doe",

id : 5566

};

document.getElementById("demo").innerHTML =

person["firstName"] + " " + person["lastName"];

</script>

</body>

</html>

5.

<!DOCTYPE html>

<html>

<body>

<p>创建和使用对象方法。</p>

<p>对象方法作为一个函数定义存储在对象属性中。</p>

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

<script>

var person = {

firstName: "John",

lastName : "Doe",

id : 5566,

fullName : function() {

return this.firstName + " " + this.lastName;

}

};

document.getElementById("demo").innerHTML = person.fullName();

</script>

</body>

</html>

6.

<!DOCTYPE html>

<html>

<body>

<p>创建和使用对象方法。</p>

<p>对象方法是一个函数定义,并作为一个属性值存储。</p>

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

<script>

var person = {

firstName: "John",

lastName : "Doe",

id : 5566,

fullName : function() {

return this.firstName + " " + this.lastName;

}

};

document.getElementById("demo").innerHTML = person.fullName;

</script>

</body>

</html>

# JavaScript 函数

函数是由事件驱动的或者当它被调用时执行的可重复使用的代码块。

## 实例

<!DOCTYPE html>  
<html>  
<head>  
<script>  
function myFunction()  
{  
alert("Hello World!");  
}  
</script>  
</head>  
  
<body>  
<button onclick="myFunction()">Try it</button>  
</body>  
</html>

## JavaScript 函数语法

函数就是包裹在花括号中的代码块，前面使用了关键词 function：

function *functionname*()  
{  
*执行代码*  
}

当调用该函数时，会执行函数内的代码。

可以在某事件发生时直接调用函数（比如当用户点击按钮时），并且可由 JavaScript 在任何位置进行调用。

|  |  |
| --- | --- |
| **lamp** | JavaScript 对大小写敏感。关键词 function 必须是小写的，并且必须以与函数名称相同的大小写来调用函数。 |

## 调用带参数的函数

在调用函数时，您可以向其传递值，这些值被称为参数。

这些参数可以在函数中使用。

您可以发送任意多的参数，由逗号 (,) 分隔：

myFunction(*argument1,argument2*)

当您声明函数时，请把参数作为变量来声明：

function myFunction(*var1*,*var2*)  
{  
*代码*  
}

变量和参数必须以一致的顺序出现。第一个变量就是第一个被传递的参数的给定的值，以此类推。

## 实例

<button onclick="myFunction('Harry Potter','Wizard')">Try it</button>  
  
<script>  
function myFunction(name,job)  
{  
alert("Welcome " + name + ", the " + job);  
}  
</script>

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=tryjs_function2)

<!DOCTYPE html>

<html>

<body>

<p>点击这个按钮，来调用带参数的函数。</p>

<button onclick="myFunction('Harry Potter','Wizard')">点击这里</button>

<script>

function myFunction(name,job)

{

alert("Welcome " + name + ", the " + job);

}

</script>

</body>

</html>

上面的函数在按钮被点击时会提示 "Welcome Harry Potter, the Wizard"。

函数很灵活，您可以使用不同的参数来调用该函数，这样就会给出不同的消息：

## 实例

<button onclick="myFunction('Harry Potter','Wizard')">Try it</button>  
<button onclick="myFunction('Bob','Builder')">Try it</button>

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=tryjs_function3)

<!DOCTYPE html>

<html>

<body>

<p>请点击其中的一个按钮，来调用带参数的函数。</p>

<button onclick="myFunction('Harry Potter','Wizard')">点击这里</button>

<button onclick="myFunction('Bob','Builder')">点击这里</button>

<script>

function myFunction(name,job)

{

alert("Welcome " + name + ", the " + job);

}

</script>

</body>

</html>

根据您点击的不同的按钮，上面的例子会提示 "Welcome Harry Potter, the Wizard" 或 "Welcome Bob, the Builder"。

## 带有返回值的函数

有时，我们会希望函数将值返回调用它的地方。

通过使用 return 语句就可以实现。

在使用 return 语句时，函数会停止执行，并返回指定的值。

### 语法

function myFunction()  
{  
var x=5;  
return x;  
}

上面的函数会返回值 5。

**注意：** 整个 JavaScript 并不会停止执行，仅仅是函数。JavaScript 将继续执行代码，从调用函数的地方。

函数调用将被返回值取代：

var myVar=myFunction();

myVar 变量的值是 5，也就是函数 "myFunction()" 所返回的值。

即使不把它保存为变量，您也可以使用返回值：

document.getElementById("demo").innerHTML=myFunction();

"demo" 元素的 innerHTML 将成为 5，也就是函数 "myFunction()" 所返回的值。

您可以使返回值基于传递到函数中的参数：

## 实例

计算两个数字的乘积，并返回结果：

function myFunction(a,b)  
{  
return a\*b;  
}  
  
document.getElementById("demo").innerHTML=myFunction(4,3);

"demo" 元素的 innerHTML 将是：

12

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

<!DOCTYPE html>

<html>

<body>

<p>本例调用的函数会执行一个计算，然后返回结果：</p>

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

<script>

function myFunction(a,b)

{

return a\*b;

}

document.getElementById("demo").innerHTML=myFunction(4,3);

</script>

</body>

</html>

在您仅仅希望退出函数时 ，也可使用 return 语句。返回值是可选的：

function myFunction(a,b)  
{  
if (a>b)  
  {  
  return;  
  }  
x=a+b  
}

如果 a 大于 b，则上面的代码将退出函数，并不会计算 a 和 b 的总和。

## 局部 JavaScript 变量

在 JavaScript 函数内部声明的变量（使用 var）是局部变量，所以只能在函数内部访问它。（该变量的作用域是局部的）。

您可以在不同的函数中使用名称相同的局部变量，因为只有声明过该变量的函数才能识别出该变量。

只要函数运行完毕，本地变量就会被删除。

## 全局 JavaScript 变量

在函数外声明的变量是全局变量，网页上的所有脚本和函数都能访问它。

## JavaScript 变量的生存期

JavaScript 变量的生命期从它们被声明的时间开始。

局部变量会在函数运行以后被删除。

全局变量会在页面关闭后被删除。

## 向未声明的 JavaScript 变量分配值

如果您把值赋给尚未声明的变量，该变量将被自动作为全局变量声明。

这条语句：

carname="Volvo";

将声明一个全局变量 carname，即使它在函数内执行。

# JavaScript 作用域

作用域可访问变量的集合。

## JavaScript 作用域

在 JavaScript 中, 对象和函数同样也是变量。

**在 JavaScript 中, 作用域为可访问变量，对象，函数的集合。**

JavaScript 函数作用域: 作用域在函数内修改。

## JavaScript 局部作用域

变量在函数内声明，变量为局部作用域。

局部变量：只能在函数内部访问。

## 实例

// 此处不能调用 carName 变量  
  
function myFunction() {  
    var carName = "Volvo";  
  
    // 函数内可调用 carName 变量  
  
}

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

<!DOCTYPE html>

<html>

<body>

<p>局部变量在声明的函数内可以访问。</p>

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

<script>

myFunction();

document.getElementById("demo").innerHTML =

"I can display " + typeof carName;

function myFunction() {

var carName = "Volvo";

}

</script>

</body>

</html>

因为局部变量只作用于函数内，所以不同的函数可以使用相同名称的变量。

局部变量在函数开始执行时创建，函数执行完后局部变量会自动销毁。

## JavaScript 全局变量

变量在函数外定义，即为全局变量。

全局变量有 **全局作用域**: 网页中所有脚本和函数均可使用。

## 实例

var carName = " Volvo";  
  
// 此处可调用 carName 变量  
  
function myFunction() {  
  
    // 函数内可调用 carName 变量   
  
}

<!DOCTYPE html>

<html>

<body>

<p>全局变量在任何脚本和函数内均可访问。</p>

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

<script>

var carName = "Volvo";

myFunction();

function myFunction() {

document.getElementById("demo").innerHTML =

"I can display " + carName;

}

</script>

</body>

</html>

如果变量在函数内没有声明（没有使用 var 关键字），该变量为全局变量。

以下实例中 carName 在函数内，但是为全局变量。

## 实例

// 此处可调用 carName 变量  
  
function myFunction() {  
    carName = "Volvo";  
  
    // 此处可调用 carName 变量  
  
}

<!DOCTYPE html>

<html>

<body>

<p>

如果你的变量没有声明，它将自动成为全局变量：

</p>

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

<script>

myFunction();

document.getElementById("demo").innerHTML =

"I can display " + carName;

function myFunction() {

carName = "Volvo";

}

</script>

</body>

</html>

## JavaScript 变量生命周期

JavaScript 变量生命周期在它声明时初始化。

局部变量在函数执行完毕后销毁。

全局变量在页面关闭后销毁。

## 函数参数

函数参数只在函数内起作用，是局部变量。

## HTML 中的全局变量

在 HTML 中, 全局变量是 window 对象: 所有数据变量都属于 window 对象。

## 实例

//此处可使用 window.carName  
  
function myFunction() {  
    carName = "Volvo";  
}

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

<!DOCTYPE html>

<html>

<body>

<p>

在 HTML 中, 所有全局变量都会成为 window 变量。

</p>

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

<script>

myFunction();

document.getElementById("demo").innerHTML =

"I can display " + window.carName;

function myFunction() {

carName = "Volvo";

}

</script>

</body>

</html>

## 你知道吗?

|  |  |
| --- | --- |
| **Note** | 你的全局变量，或者函数，可以覆盖 window 对象的变量或者函数。 局部变量，包括 window 对象可以覆盖全局变量和函数。 |

# JavaScript 事件

HTML 事件是发生在 HTML 元素上的事情。

当在 HTML 页面中使用 JavaScript 时， JavaScript 可以触发这些事件。

## HTML 事件

HTML 事件可以是浏览器行为，也可以是用户行为。

以下是 HTML 事件的实例：

* HTML 页面完成加载
* HTML input 字段改变时
* HTML 按钮被点击

通常，当事件发生时，你可以做些事情。

在事件触发时 JavaScript 可以执行一些代码。

HTML 元素中可以添加事件属性，使用 JavaScript 代码来添加 HTML 元素。

单引号:

<some-HTML-element some-event=**'*some JavaScript*'**>

双引号:

<some-HTML-element some-event=**"*some JavaScript*"**>

在以下实例中，按钮元素中添加了 onclick 属性 (并加上代码):

## 实例

<button onclick='getElementById("demo").innerHTML=Date()'>The time is?</button>

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=tryjs_event_onclick1)

<!DOCTYPE html>

<html>

<body>

<button onclick="getElementById('demo').innerHTML=Date()">现在的时间是?</button>

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

</body>

</html>

以上实例中，JavaScript 代码将修改 id="demo" 元素的内容。

在下一个实例中，代码将修改自身元素的内容 (使用 **this**.innerHTML):

## 实例

<button onclick="this.innerHTML=Date()">The time is?</button>

<!DOCTYPE html>

<html>

<body>

<button onclick="this.innerHTML=Date()">现在的时间是?</button>

</body>

</html>

|  |  |
| --- | --- |
| **Note** | JavaScript代码通常是几行代码。比较常见的是通过事件属性来调用： |

## 实例

<button onclick="displayDate()">The time is?</button>

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=tryjs_events1)

<!DOCTYPE html>

<html>

<body>

<p>点击按钮执行 <em>displayDate()</em> 函数.</p>

<button onclick="displayDate()">点这里</button>

<script>

function displayDate()

{

document.getElementById("demo").innerHTML=Date();

}

</script>

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

</body>

</html>

## 常见的HTML事件

下面是一些常见的HTML事件的列表:

|  |  |
| --- | --- |
| **事件** | **描述** |
| onchange | HTML 元素改变 |
| onclick | 用户点击 HTML 元素 |
| onmouseover | 用户在一个HTML元素上移动鼠标 |
| onmouseout | 用户从一个HTML元素上移开鼠标 |
| onkeydown | 用户按下键盘按键 |
| onload | 浏览器已完成页面的加载 |

更多事件列表: [JavaScript 参考手册 - HTML DOM 事件](http://www.runoob.com/jsref/dom-obj-event.html)。

## JavaScript 可以做什么?

事件可以用于处理表单验证，用户输入，用户行为及浏览器动作:

* 页面加载时触发事件
* 页面关闭时触发事件
* 用户点击按钮执行动作
* 验证用户输入内容的合法性
* 等等 ...

可以使用多种方法来执行 JavaScript 事件代码：

* HTML 事件属性可以直接执行 JavaScript 代码
* HTML 事件属性可以调用 JavaScript 函数
* 你可以为 HTML 元素指定自己的事件处理程序
* 你可以阻止事件的发生。
* 等等 ...

|  |  |
| --- | --- |
| **Note** | 在 HTML DOM 章节中你将会学到更多关于事件及事件处理程序的知识。 |

# JavaScript 字符串

JavaScript 字符串用于存储和处理文本。

## JavaScript 字符串

字符串可以存储一系列字符，如 "John Doe"。

字符串可以是插入到引号中的任何字符。你可以使用单引号或双引号：

## 实例

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

你可以使用索引位置来访问字符串中的每个字符：

## 实例

var character = carname[7];

字符串的索引从 0 开始，这意味着第一个字符索引值为 [0],第二个为 [1], 以此类推。

你可以在字符串中使用引号，字符串中的引号不要与字符串的引号相同:

## 实例

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

你也可以在字符串添加转义字符来使用引号：

## 实例

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

<!DOCTYPE html>

<html>

<body>

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

<script>

var x = 'It\'s alright';

var y = "He is called \"Johnny\"";

document.getElementById("demo").innerHTML = x + "<br>" + y;

</script>

</body>

</html>

## 字符串长度

可以使用内置属性 **length** 来计算字符串的长度：

## 实例

var txt = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";  
var sln = txt.length;

<!DOCTYPE html>

<html>

<body>

<script>

var txt = "Hello World!";

document.write("<p>" + txt.length + "</p>");

var txt="ABCDEFGHIJKLMNOPQRSTUVWXYZ";

document.write("<p>" + txt.length + "</p>");

</script>

</body>

</html>

**特殊字符**

在 JavaScript 中，字符串写在单引号或双引号来中。

因为这样，以下实例 JavaScript 无法解析：x

 "We are the so-called "Vikings" from the north."

字符串 "We are the so-called " 被截断。

如何解决以上的问题呢？可以使用反斜杠 (\) 来转义 "Vikings" 字符串中的双引号，如下:

 "We are the so-called \"Vikings\" from the north."

 反斜杠是一个**转义字符**。 转义字符将特殊字符转换为字符串字符：

转义字符 (\) 可以用于转义撇号，换行，引号，等其他特殊字符。

下表中列举了在字符串中可以使用转义字符转义的特殊字符：

|  |  |
| --- | --- |
| **代码** | **输出** |
| \' | 单引号 |
| \" | 双引号 |
| \\ | 反斜杠 |
| \n | 换行 |
| \r | 回车 |
| \t | tab(制表符) |
| \b | 退格符 |
| \f | 换页符 |

**字符串可以是对象**

通常， JavaScript 字符串是原始值，可以使用字符创建： **var firstName = "John"**

但我们也可以使用 new 关键字将字符串定义为一个对象： **var firstName = new String("John")**

## 实例

var x = "John";  
var y = new String("John");  
typeof x // 返回 String  
typeof y // 返回 Object

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

<!DOCTYPE html>

<html>

<body>

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

<script>

var x = "John"; // x is a string

var y = new String("John"); // y is an object

document.getElementById("demo").innerHTML =

typeof x + " " + typeof y;

</script>

</body>

</html>

|  |  |
| --- | --- |
| **Note** | 不要创建 String 对象。它会拖慢执行速度，并可能产生其他副作用： |

## 实例

var x = "John";                
var y = new String("John");  
(x === y) // 结果为 false，因为是字符串，y 是对象

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

<!DOCTYPE html>

<html>

<body>

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

<script>

var x = "John"; // x 是字符串

var y = new String("John"); // y 是一个对象

document.getElementById("demo").innerHTML = x===y;

</script>

<p>=== 为绝对相等，即数据类型与值都必须相等。</p>

</body>

</html>

<!DOCTYPE html>

<html>

<body>

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

<script>

var x = "John"; // x 是字符串

var y = new String("John"); // y 是一个对象

document.getElementById("demo").innerHTML = x===y;

</script>

<p>=== 为绝对相等，即数据类型与值都必须相等。</p>

</body>

</html>

=== 为绝对相等，即数据类型与值都必须相等。

## 字符串属性和方法

原始值字符串，如 "John", 没有属性和方法(因为他们不是对象)。

原始值可以使用 JavaScript 的属性和方法，因为 JavaScript 在执行方法和属性时可以把原始值当作对象。

**字符串方法我们将在下一章节中介绍。**

## 字符串属性

|  |  |
| --- | --- |
| **属性** | **描述** |
| constructor | 返回创建字符串属性的函数 |
| length | 返回字符串的长度 |
| prototype | 允许您向对象添加属性和方法 |

## 字符串方法

|  |  |
| --- | --- |
| **Method** | **描述** |
| charAt() | 返回指定索引位置的字符 |
| charCodeAt() | 返回指定索引位置字符的 Unicode 值 |
| concat() | 连接两个或多个字符串，返回连接后的字符串 |
| fromCharCode() | 将字符转换为 Unicode 值 |
| indexOf() | 返回字符串中检索指定字符第一次出现的位置 |
| lastIndexOf() | 返回字符串中检索指定字符最后一次出现的位置 |
| localeCompare() | 用本地特定的顺序来比较两个字符串 |
| match() | 找到一个或多个正则表达式的匹配 |
| replace() | 替换与正则表达式匹配的子串 |
| search() | 检索与正则表达式相匹配的值 |
| slice() | 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() | 把字符串分割为子字符串数组 |
| substr() | 从起始索引号提取字符串中指定数目的字符 |
| substring() | 提取字符串中两个指定的索引号之间的字符 |
| toLocaleLowerCase() | 根据主机的语言环境把字符串转换为小写，只有几种语言（如土耳其语）具有地方特有的大小写映射 |
| toLocaleUpperCase() | 根据主机的语言环境把字符串转换为大写，只有几种语言（如土耳其语）具有地方特有的大小写映射 |
| toLowerCase() | 把字符串转换为小写 |
| toString() | 返回字符串对象值 |
| toUpperCase() | 把字符串转换为大写 |
| trim() | 移除字符串首尾空白 |
| valueOf() | 返回某个字符串对象的原始值 |

# JavaScript 运算符

**运算符 = 用于赋值。**

**运算符 + 用于加值。**

运算符 = 用于给 JavaScript 变量赋值。

算术运算符 + 用于把值加起来。

## 实例

指定变量值，并将值相加：

y=5;  
z=2;  
x=y+z;

在以上语句执行后，x 的值是：

7

**<!DOCTYPE html>**

**<html>**

**<body>**

**<p>点击按钮计算 x.</p>**

**<button onclick="myFunction()">点击这里</button>**

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

**<script>**

**function myFunction()**

**{**

**y=5;**

**z=2;**

**x=y+z;**

**document.getElementById("demo").innerHTML=x;**

**}**

**</script>**

**</body>**

**</html>**

## JavaScript 算术运算符

<="" p="" style="color: rgb(51, 51, 51); font-family: 'Open Sans', 'Helvetica Neue', Helvetica, Arial, STHeiti, 'Microsoft Yahei', sans-serif; font-size: 12px; font-style: normal; font-variant: normal; font-weight: normal; letter-spacing: normal; line-height: normal; orphans: auto; text-align: start; text-indent: 0px; text-transform: none; white-space: normal; widows: 1; word-spacing: 0px; -webkit-text-stroke-width: 0px; background-color: rgb(255, 255, 255);"><p给定**y=5，下面的表格解释了这些算术运算符：**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **运算符** | **描述** | **例子** | **x 运算结果** | **y 运算结果** | **在线实例** |
| + | 加法 | x=y+2 | 7 | 5 | [**实例 »**](http://www.runoob.com/try/try.php?filename=tryjs_oper_add) |
| - | 减法 | x=y-2 | 3 | 5 | [**实例 »**](http://www.runoob.com/try/try.php?filename=tryjs_oper_sub) |
| \* | 乘法 | x=y\*2 | 10 | 5 | [**实例 »**](http://www.runoob.com/try/try.php?filename=tryjs_oper_mult) |
| / | 除法 | x=y/2 | 2.5 | 5 | [**实例 »**](http://www.runoob.com/try/try.php?filename=tryjs_oper_div) |
| % | 取模（余数） | x=y%2 | 1 | 5 | [**实例 »**](http://www.runoob.com/try/try.php?filename=tryjs_oper_mod) |
| ++ | 自增 | x=++y | 6 | 6 | [**实例 »**](http://www.runoob.com/try/try.php?filename=tryjs_oper_incr) |
| x=y++ | 5 | 6 | [**实例 »**](http://www.runoob.com/try/try.php?filename=tryjs_oper_incr2) |
| -- | 自减 | x=--y | 4 | 4 | [**实例 »**](http://www.runoob.com/try/try.php?filename=tryjs_oper_decr) |
| x=y-- | 5 | 4 | [**实例 »**](http://www.runoob.com/try/try.php?filename=tryjs_oper_decr2) |

## JavaScript 赋值运算符

**赋值运算符用于给 JavaScript 变量赋值。**

**给定 x=10 和 y=5，下面的表格解释了赋值运算符：**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **运算符** | **例子** | **等同于** | **运算结果** | **在线实例** |
| = | x=y |  | x=5 | [**实例 »**](http://www.runoob.com/try/try.php?filename=tryjs_oper_equal) |
| += | x+=y | x=x+y | x=15 | [**实例 »**](http://www.runoob.com/try/try.php?filename=tryjs_oper_plusequal) |
| -= | x-=y | x=x-y | x=5 | [**实例 »**](http://www.runoob.com/try/try.php?filename=tryjs_oper_minequal) |
| \*= | x\*=y | x=x\*y | x=50 | [**实例 »**](http://www.runoob.com/try/try.php?filename=tryjs_oper_multequal) |
| /= | x/=y | x=x/y | x=2 | [**实例 »**](http://www.runoob.com/try/try.php?filename=tryjs_oper_divequal) |
| %= | x%=y | x=x%y | x=0 | [**实例 »**](http://www.runoob.com/try/try.php?filename=tryjs_oper_modequal) |

## 用于字符串的 + 运算符

**+ 运算符用于把文本值或字符串变量加起来（连接起来）。**

**如需把两个或多个字符串变量连接起来，请使用 + 运算符。**

## 实例

**如需把两个或多个字符串变量连接起来，请使用 + 运算符：**

**txt1="What a very";  
txt2="nice day";  
txt3=txt1+txt2;**

**txt3 运算结果如下:**

**What a verynice day**

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=tryjs_operators2)

**要想在两个字符串之间增加空格，需要把空格插入一个字符串之中：**

## 实例

**txt1="What a very ";  
txt2="nice day";  
txt3=txt1+txt2;**

**在以上语句执行后，变量 txt3包含的值是：**

**What a very nice day**

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=tryjs_operators3)

**或者把空格插入表达式中：:**

## 实例

**txt1="What a very";  
txt2="nice day";  
txt3=txt1+" "+txt2;**

**在以上语句执行后，变量txt3 包含的值是：**

**What a very nice day**

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=tryjs_operators4)

## 对字符串和数字进行加法运算

**两个数字相加，返回数字相加的和，如果数字与字符串相加，返回字符串，如下实例：**

## 实例

**x=5+5;  
y="5"+5;  
z="Hello"+5;**

**x,y, 和 z 输出结果为:**

**10  
55  
Hello5**

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=tryjs_operators5)

**规则:如果把数字与字符串相加，结果将成为字符串！**

# JavaScript 比较 和 逻辑运算符

比较和逻辑运算符用于测试 true 或者 false。

## 比较运算符

比较运算符在逻辑语句中使用，以测定变量或值是否相等。

<p给定**x=5，下面的表格解释了比较运算符：**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **运算符** | **描述** | **比较** | **返回值** | **实例** |
| == | 等于 | x==8 | false | [**实例 »**](http://www.runoob.com/try/try.php?filename=tryjs_comparison1) |
| x==5 | true | [**实例 »**](http://www.runoob.com/try/try.php?filename=tryjs_comparison2) |
| === | 绝对等于（值和类型均相等） | x==="5" | false | [**实例 »**](http://www.runoob.com/try/try.php?filename=tryjs_comparison3) |
| x===5 | true | [**实例 »**](http://www.runoob.com/try/try.php?filename=tryjs_comparison4) |
| != | 不等于 | x!=8 | true | [**实例 »**](http://www.runoob.com/try/try.php?filename=tryjs_comparison5) |
| !== | 绝对不等于（值或类型不相等） | x!=="5" | true | [**实例 »**](http://www.runoob.com/try/try.php?filename=tryjs_comparison6) |
| x!==5 | false | [**实例 »**](http://www.runoob.com/try/try.php?filename=tryjs_comparison7) |
| > | 大于 | x>8 | false | [**实例 »**](http://www.runoob.com/try/try.php?filename=tryjs_comparison8) |
| < | 小于 | x<8 | true | [**实例 »**](http://www.runoob.com/try/try.php?filename=tryjs_comparison9) |
| >= | 大于或等于 | x>=8 | false | [**实例 »**](http://www.runoob.com/try/try.php?filename=tryjs_comparison10) |
| <= | 小于或等于 | x<=8 | true | [**实例 »**](http://www.runoob.com/try/try.php?filename=tryjs_comparison11) |

## 如何使用

可以在条件语句中使用比较运算符对值进行比较，然后根据结果来采取行动：

if (age<18) x="Too young";

您将在本教程的下一节中学习更多有关条件语句的知识。

## 逻辑运算符

逻辑运算符用于测定变量或值之间的逻辑。

给定 x=6 以及 y=3，下表解释了逻辑运算符：

|  |  |  |
| --- | --- | --- |
| **运算符** | **描述** | **例子** |
| && | and | (x < 10 && y > 1) 为 true |
| || | or | (x==5 || y==5) 为 false |
| ! | not | !(x==y) 为 true |

## 条件运算符

JavaScript 还包含了基于某些条件对变量进行赋值的条件运算符。

### 语法

variablename=(condition)?value1:value2

### 例子

## 实例

如果变量 age 中的值小于 18，则向变量 voteable 赋值 "年龄太小"，否则赋值 "年龄已达到"。

voteable=(age<18)?"年龄太小":"年龄已达到";

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

**<!DOCTYPE html>**

**<html>**

**<body>**

**<p>点击按钮检测年龄。</p>**

**年龄:<input id="age" value="18" />**

**<p>是否达到投票年龄?</p>**

**<button onclick="myFunction()">点击按钮</button>**

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

**<script>**

**function myFunction()**

**{**

**var age,voteable;**

**age=document.getElementById("age").value;**

**voteable=(age<18)?"年龄太小":"年龄已达到";**

**document.getElementById("demo").innerHTML=voteable;**

**}**

**</script>**

**</body>**

**</html>**

# JavaScript If...Else 语句

条件语句用于基于不同的条件来执行不同的动作。

## 条件语句

通常在写代码时，您总是需要为不同的决定来执行不同的动作。您可以在代码中使用条件语句来完成该任务。

在 JavaScript 中，我们可使用以下条件语句：

* **if 语句** - 只有当指定条件为 true 时，使用该语句来执行代码
* **if...else 语句** - 当条件为 true 时执行代码，当条件为 false 时执行其他代码
* **if...else if....else 语句**- 使用该语句来选择多个代码块之一来执行
* **switch 语句** - 使用该语句来选择多个代码块之一来执行

## If 语句

只有当指定条件为 true 时，该语句才会执行代码。

### 语法

if (*condition*)  
  {  
*当条件为 true 时执行的代码*  
  }

请使用小写的**if**。使用大写字母（IF）会生成 JavaScript 错误！

## 实例

当时间小于 20:00 时，生成问候 "Good day"：

if (time<20)  
  {  
  x="Good day";  
  }

<p**x 的结果是：**

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

**<!DOCTYPE html>**

**<html>**

**<body>**

**<p>如果时间早于 20:00，会获得问候 "Good day"。</p>**

**<button onclick="myFunction()">点击这里</button>**

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

**<script>**

**function myFunction()**

**{**

**var x="";**

**var time=new Date().getHours();**

**if (time<20)**

**{**

**x="Good day";**

**}**

**document.getElementById("demo").innerHTML=x;**

**}**

**</script>**

**</body>**

**</html>**

请注意，在这个语法中，没有 ..else..。您已经告诉浏览器只有在指定条件为 true 时才执行代码。

## If...else 语句

请使用 if....else 语句在条件为 true 时执行代码，在条件为 false 时执行其他代码。

### 语法

if (*condition*)  
  {  
*当条件为 true 时执行的代码*  
  }  
else  
  {  
*当条件不为 true 时执行的代码*  
  }

## 实例

当时间小于 20:00 时，生成问候 "Good day"，否则生成问候 "Good evening"。

if (time<20)  
  {  
  x="Good day";  
  }  
else  
  {  
  x="Good evening";  
  }

**x**的结果是：

Good evening

**<!DOCTYPE html>**

**<html>**

**<body>**

**<p>点击这个按钮，获得基于时间的问候。</p>**

**<button onclick="myFunction()">点击这里</button>**

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

**<script>**

**function myFunction()**

**{**

**var x="";**

**var time=new Date().getHours();**

**if (time<20)**

**{**

**x="Good day";**

**}**

**else**

**{**

**x="Good evening";**

**}**

**document.getElementById("demo").innerHTML=x;**

**}**

**</script>**

**</body>**

**</html>**

## If...else if...else 语句

使用 if....else if...else 语句来选择多个代码块之一来执行。

### 语法

if (*condition1*)  
  {  
*当条件 1 为 true 时执行的代码*  
  }  
else if (*condition2*)  
  {  
*当条件 2 为 true 时执行的代码*  
  }  
else  
  {  
*当条件 1 和 条件 2 都不为 true 时执行的代码*  
  }

## 实例

如果时间小于 10:00，则生成问候 "Good morning"，如果时间大于 10:00 小于 20:00，则生成问候 "Good day"，否则生成问候 "Good evening"：

if (time<10)  
  {  
  x="Good morning";  
  }  
else if (time>=10 && time<20)  
  {  
  x="Good day";  
  }  
else  
  {  
  x="Good evening";  
  }

**x**的结果是：

Good evening

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

**<html>**

**<body>**

**<script type="text/javascript">**

**var d = new Date();**

**var time = d.getHours();**

**if (time<10)**

**{**

**document.write("<b>早上好</b>");**

**}**

**else if (time>=10 && time<16)**

**{**

**document.write("<b>今天好</b>");**

**}**

**else**

**{**

**document.write("<b>Hello World!</b>");**

**}**

**</script>**

**<p>**

**这个例子演示了 if..else if...else 语句。**

**</p>**

**</body>**

**</html>**

# JavaScript switch 语句

switch 语句用于基于不同的条件来执行不同的动作。

## JavaScript switch 语句

请使用 switch 语句来选择要执行的多个代码块之一。

### 语法

switch(n)  
{  
case 1:  
*执行代码块 1*  
break;  
case 2:  
*执行代码块 2*  
break;  
default:  
*n 与 case 1 和 case 2 不同时执行的代码*  
}

工作原理：首先设置表达式 *n*（通常是一个变量）。随后表达式的值会与结构中的每个 case 的值做比较。如果存在匹配，则与该 case 关联的代码块会被执行。请使用 **break**来阻止代码自动地向下一个 case 运行。

## 实例

显示今天的星期名称。请注意 Sunday=0, Monday=1, Tuesday=2, 等等：

var day=new Date().getDay();  
switch (day)  
{  
case 0:  
  x="Today it's Sunday";  
  break;  
case 1:  
  x="Today it's Monday";  
  break;  
case 2:  
  x="Today it's Tuesday";  
  break;  
case 3:  
  x="Today it's Wednesday";  
  break;  
case 4:  
  x="Today it's Thursday";  
  break;  
case 5:  
  x="Today it's Friday";  
  break;  
case 6:  
  x="Today it's Saturday";  
  break;  
}

x 的运行结果：

Today it's Sunday

**<!DOCTYPE html>**

**<html>**

**<body>**

**<p>点击下面的按钮来显示今天是周几：</p>**

**<button onclick="myFunction()">点击这里</button>**

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

**<script>**

**function myFunction()**

**{**

**var x;**

**var d=new Date().getDay();**

**switch (d)**

**{**

**case 0:**

**x="Today it's Sunday";**

**break;**

**case 1:**

**x="Today it's Monday";**

**break;**

**case 2:**

**x="Today it's Tuesday";**

**break;**

**case 3:**

**x="Today it's Wednesday";**

**break;**

**case 4:**

**x="Today it's Thursday";**

**break;**

**case 5:**

**x="Today it's Friday";**

**break;**

**case 6:**

**x="Today it's Saturday";**

**break;**

**}**

**document.getElementById("demo").innerHTML=x;**

**}**

**</script>**

**</body>**

**</html>**

## default 关键词

请使用 default 关键词来规定匹配不存在时做的事情：

## 实例

如果今天不是星期六或星期日，则会输出默认的消息：

var day=new Date().getDay();  
switch (day)  
{  
case 6:  
  x="Today it's Saturday";  
  break;  
case 0:  
  x="Today it's Sunday";  
  break;  
default:  
  x="Looking forward to the Weekend";  
}

x 的运行结果：

Today it's Sunday

**<html>**

**<body>**

**<p>点击下面的按钮，会显示出基于今日日期的消息：</p>**

**<button onclick="myFunction()">点击这里</button>**

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

**<script>**

**function myFunction()**

**{**

**var x;**

**var d=new Date().getDay();**

**switch (d)**

**{**

**case 6:**

**x="Today it's Saturday";**

**break;**

**case 0:**

**x="Today it's Sunday";**

**break;**

**default:**

**x="Looking forward to the Weekend";**

**}**

**document.getElementById("demo").innerHTML=x;**

**}**

**</script>**

**</body>**

**</html>**

# JavaScript for 循环

循环可以将代码块执行指定的次数。

## JavaScript 循环

如果您希望一遍又一遍地运行相同的代码，并且每次的值都不同，那么使用循环是很方便的。

我们可以这样输出数组的值：

## 一般写法：

document.write(cars[0] + "<br>");   
document.write(cars[1] + "<br>");   
document.write(cars[2] + "<br>");   
document.write(cars[3] + "<br>");   
document.write(cars[4] + "<br>");   
document.write(cars[5] + "<br>");

## 使用for循环

for (var i=0;i<cars.length;i++)  
{   
document.write(cars[i] + "<br>");  
}

**<!DOCTYPE html>**

**<html>**

**<body>**

**<script>**

**cars=["BMW","Volvo","Saab","Ford"];**

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

**{**

**document.write(cars[i] + "<br>");**

**}**

**</script>**

**</body>**

**</html>**

**不同类型的循环**

JavaScript 支持不同类型的循环：

* **for** - 循环代码块一定的次数
* **for/in** - 循环遍历对象的属性
* **while** - 当指定的条件为 true 时循环指定的代码块
* **do/while** - 同样当指定的条件为 true 时循环指定的代码块

**For 循环**

for 循环是您在希望创建循环时常会用到的工具。

下面是 for 循环的语法：

for (*语句 1*;*语句 2*;*语句 3*)  
  {  
*被执行的代码块*  
  }

**语句 1** （代码块）开始前执行 starts.

**语句 2** 定义运行循环（代码块）的条件

**语句 3** 在循环（代码块）已被执行之后执行

**实例**

for (var i=0; i<5; i++)  
  {  
  x=x + "The number is " + i + "<br>";  
  }

**<!DOCTYPE html>**

**<html>**

**<body>**

**<p>点击按钮循环代码5次。</p>**

**<button onclick="myFunction()">点击这里</button>**

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

**<script>**

**function myFunction()**

**{**

**var x="";**

**for (var i=0;i<5;i++)**

**{**

**x=x + "The number is " + i + "<br>";**

**}**

**document.getElementById("demo").innerHTML=x;**

**}**

**</script>**

**</body>**

**</html>**

从上面的例子中，您可以看到：

Statement 1 在循环开始之前设置变量 (var i=0)。

Statement 2 定义循环运行的条件（i 必须小于 5）。

Statement 3 在每次代码块已被执行后增加一个值 (i++)。

## 语句 1

通常我们会使用语句 1 初始化循环中所用的变量 (var i=0)。

语句 1 是可选的，也就是说不使用语句 1 也可以。

您可以在语句 1 中初始化任意（或者多个）值：

## 实例:

for (var i=0,len=cars.length; i<len; i++)  
{   
document.write(cars[i] + "<br>");  
}

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=tryjs_loop_for_om1)

**<!DOCTYPE html>**

**<html>**

**<body>**

**<script>**

**cars=["BMW","Volvo","Saab","Ford"];**

**for (var i=0,l=cars.length; i<l; i++)**

**{**

**document.write(cars[i] + "<br>");**

**}**

**</script>**

**</body>**

**</html>**

同时您还可以省略语句 1（比如在循环开始前已经设置了值时）：

## 实例:

var i=2,len=cars.length;  
for (; i<len; i++)  
{   
document.write(cars[i] + "<br>");  
}

**<!DOCTYPE html>**

**<html>**

**<body>**

**<script>**

**cars=["BMW","Volvo","Saab","Ford"];**

**var i=2,len=cars.length;**

**for (; i<len; i++)**

**{**

**document.write(cars[i] + "<br>");**

**}**

**</script>**

**</body>**

**</html>**

## 语句 2

通常语句 2 用于评估初始变量的条件。

语句 2 同样是可选的。

如果语句 2 返回 true，则循环再次开始，如果返回 false，则循环将结束。

|  |  |
| --- | --- |
| **lamp** | 如果您省略了语句 2，那么必须在循环内提供 break。否则循环就无法停下来。这样有可能令浏览器崩溃。请在本教程稍后的章节阅读有关 break 的内容。 |

## 语句 3

通常语句 3 会增加初始变量的值。

语句 3 也是可选的。

语句 3 有多种用法。增量可以是负数 (i--)，或者更大 (i=i+15)。

语句 3 也可以省略（比如当循环内部有相应的代码时）：

## 实例:

var i=0,len=cars.length;  
for (; i<len; )  
{   
document.write(cars[i] + "<br>");  
i++;  
}

**<!DOCTYPE html>**

**<html>**

**<body>**

**<script>**

**cars=["BMW","Volvo","Saab","Ford"];**

**var i=0,len=cars.length;**

**for (; i<len; )**

**{**

**document.write(cars[i] + "<br>");**

**i++;**

**}**

**</script>**

**</body>**

**</html>**

## For/In 循环

JavaScript for/in 语句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];  
  }

**<!DOCTYPE html>**

**<html>**

**<body>**

**<p>点击下面的按钮，循环遍历对象 "person" 的属性。</p>**

**<button onclick="myFunction()">点击这里</button>**

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

**<script>**

**function myFunction()**

**{**

**var x;**

**var txt="";**

**var person={fname:"Bill",lname:"Gates",age:56};**

**for (x in person)**

**{**

**txt=txt + person[x];**

**}**

**document.getElementById("demo").innerHTML=txt;**

**}**

**</script>**

**</body>**

**</html>**

您将在有关 JavaScript 对象的章节学到更多有关 for / in 循环的知识。

## While 循环

我们将在下一章为您讲解 while 循环和 do/while 循环。

# JavaScript while 循环

只要指定条件为 true，循环就可以一直执行代码块。

## while 循环

while 循环会在指定条件为真时循环执行代码块。

### 语法

while (条件)  
  {  
*需要执行的代码*  
  }

### 实例

本例中的循环将继续运行，只要变量 i 小于 5：

## 实例

while (i<5)  
  {  
  x=x + "The number is " + i + "<br>";  
  i++;  
  }

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

**<!DOCTYPE html>**

**<html>**

**<body>**

**<p>点击下面的按钮，只要 i 小于 5 就一直循环代码块。</p>**

**<button onclick="myFunction()">点击这里</button>**

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

**<script>**

**function myFunction()**

**{**

**var x="",i=0;**

**while (i<5)**

**{**

**x=x + "The number is " + i + "<br>";**

**i++;**

**}**

**document.getElementById("demo").innerHTML=x;**

**}**

**</script>**

**</body>**

**</html>**

|  |  |
| --- | --- |
| **lamp** | 如果您忘记增加条件中所用变量的值，该循环永远不会结束。这可能导致浏览器崩溃。 |

## do/while 循环

do/while 循环是 while 循环的变体。该循环会在检查条件是否为真之前执行一次代码块，然后如果条件为真的话，就会重复这个循环。

### 语法

do  
  {  
*需要执行的代码*}  
while (条件);

### 实例

下面的例子使用 do/while 循环。该循环至少会执行一次，即使条件为 false 它也会执行一次，因为代码块会在条件被测试前执行：

## 实例

do  
  {  
  x=x + "The number is " + i + "<br>";  
  i++;  
  }  
while (i<5);

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

**<!DOCTYPE html>**

**<html>**

**<body>**

**<p>点击下面的按钮，只要 i 小于 5 就一直循环代码块。</p>**

**<button onclick="myFunction()">点击这里</button>**

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

**<script>**

**function myFunction()**

**{**

**var x="",i=0;**

**do**

**{**

**x=x + "The number is " + i + "<br>";**

**i++;**

**}**

**while (i<5)**

**document.getElementById("demo").innerHTML=x;**

**}**

**</script>**

**</body>**

**</html>**

别忘记增加条件中所用变量的值，否则循环永远不会结束！

## 比较 for 和 while

如果您已经阅读了前面那一章关于 for 循环的内容，您会发现 while 循环与 for 循环很像。

本例中的循环使用 **for 循环**来显示 cars 数组中的所有值：

## 实例

cars=["BMW","Volvo","Saab","Ford"];  
var i=0;  
for (;cars[i];)  
{  
document.write(cars[i] + "<br>");  
i++;  
}

**<!DOCTYPE html>**

**<html>**

**<body>**

**<script>**

**cars=["BMW","Volvo","Saab","Ford"];**

**var i=0;**

**for (;cars[i];)**

**{**

**document.write(cars[i] + "<br>");**

**i++;**

**}**

**</script>**

**</body>**

**</html>**

本例中的循环使用 **while 循环**来显示 cars 数组中的所有值：

## 实例

cars=["BMW","Volvo","Saab","Ford"];  
var i=0;  
while (cars[i])  
{  
document.write(cars[i] + "<br>");  
i++;  
}

**<!DOCTYPE html>**

**<html>**

**<body>**

**<script>**

**cars=["BMW","Volvo","Saab","Ford"];**

**var i=0;**

**while (cars[i])**

**{**

**document.write(cars[i] + "<br>");**

**i++;**

**}**

**</script>**

**</body>**

**</html>**

# JavaScript Break 和 Continue 语句

break 语句用于跳出循环。

continue 用于跳过循环中的一个迭代。

## Break 语句

我们已经在本教程之前的章节中见到过 break 语句。它用于跳出 switch() 语句。

break 语句可用于跳出循环。

break 语句跳出循环后，会继续执行该循环之后的代码（如果有的话）：

## 实例

for (i=0;i<10;i++)  
  {  
  if (i==3)  
    {  
    break;  
    }  
  x=x + "The number is " + i + "<br>";  
  }

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

**<!DOCTYPE html>**

**<html>**

**<body>**

**<p>点击按钮，测试带有 break 语句的循环。</p>**

**<button onclick="myFunction()">点击这里</button>**

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

**<script>**

**function myFunction()**

**{**

**var x="",i=0;**

**for (i=0;i<10;i++)**

**{**

**if (i==3)**

**{**

**break;**

**}**

**x=x + "The number is " + i + "<br>";**

**}**

**document.getElementById("demo").innerHTML=x;**

**}**

**</script>**

**</body>**

**</html>**

由于这个 if 语句只有一行代码，所以可以省略花括号：

for (i=0;i<10;i++)  
  {  
  if (i==3) break;  
  x=x + "The number is " + i + "<br>";  
  }

## Continue 语句

**continue 语句**中断循环中的迭代，如果出现了指定的条件，然后继续循环中的下一个迭代。 该例子跳过了值 3：

## 实例

for (i=0;i<=10;i++)  
 {  
 if (i==3) continue;  
  x=x + "The number is " + i + "<br>";  
  }

**<!DOCTYPE html>**

**<html>**

**<body>**

**<p>点击下面的按钮来执行循环，该循环会跳过 i=3 的步进。</p>**

**<button onclick="myFunction()">点击这里</button>**

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

**<script>**

**function myFunction()**

**{**

**var x="",i=0;**

**for (i=0;i<10;i++)**

**{**

**if (i==3)**

**{**

**continue;**

**}**

**x=x + "The number is " + i + "<br>";**

**}**

**document.getElementById("demo").innerHTML=x;**

**}**

**</script>**

**</body>**

**</html>**

## JavaScript 标签

正如您在 switch 语句那一章中看到的，可以对 JavaScript 语句进行标记。

如需标记 JavaScript 语句，请在语句之前加上冒号：

label:  
statements

break 和 continue 语句仅仅是能够跳出代码块的语句。

语法:

break labelname;   
  
continue labelname;

continue 语句（带有或不带标签引用）只能用在循环中。

break 语句（不带标签引用），只能用在循环或 switch 中。

通过标签引用，break 语句可用于跳出任何 JavaScript 代码块：

## 实例

cars=["BMW","Volvo","Saab","Ford"];  
list:   
{  
document.write(cars[0] + "<br>");   
document.write(cars[1] + "<br>");   
document.write(cars[2] + "<br>");   
break list;  
document.write(cars[3] + "<br>");   
document.write(cars[4] + "<br>");   
document.write(cars[5] + "<br>");   
}

**<!DOCTYPE html>**

**<html>**

**<body>**

**<script>**

**cars=["BMW","Volvo","Saab","Ford"];**

**list:**

**{**

**document.write(cars[0] + "<br>");**

**document.write(cars[1] + "<br>");**

**document.write(cars[2] + "<br>");**

**break list;**

**document.write(cars[3] + "<br>");**

**document.write(cars[4] + "<br>");**

**document.write(cars[5] + "<br>");**

**}**

**</script>**

**</body>**

**</html>**

# 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)

**<!DOCTYPE html>**

**<html>**

**<body>**

**<p> typeof 操作符返回变量或表达式的类型。</p>**

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

**<script>**

**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};**

**</script>**

**</body>**

**</html>**

|  |  |
| --- | --- |
| **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;           // Value is null, but type is still an object

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

**<!DOCTYPE html>**

**<html>**

**<body>**

**<p>对象可以通过设置为 <b>null</b> 来清空。</p>**

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

**<script>**

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

**var person = null;**

**document.getElementById("demo").innerHTML = typeof person;**

**</script>**

**</body>**

**</html>**

你可以设置为 undefined 来清空对象:

## 实例

var person = undefined;     // 值为 undefined, type is undefined

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

**<!DOCTYPE html>**

**<html>**

**<body>**

**<p>对象可以设置为 <b>undefined</b> 来清空。</p>**

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

**<script>**

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

**var person = undefined;**

**document.getElementById("demo").innerHTML = typeof person;**

**</script>**

**</body>**

**</html>**

## 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;                  // Value is undefined, type is undefined

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

**<!DOCTYPE html>**

**<html>**

**<body>**

**<p>变量的值如果不存在则该变量值为 <b>undefined</b>。</p>**

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

**<script>**

**var person;**

**document.getElementById("demo").innerHTML =**

**person + "<br>" + typeof person;**

**</script>**

**</body>**

**</html>**

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, type is undefined

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

**<!DOCTYPE html>**

**<html>**

**<body>**

**<p>变量可以通过设置 <b>undefined</b> 来清空。</p>**

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

**<script>**

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

**var person = undefined;**

**document.getElementById("demo").innerHTML =**

**person + "<br>" + typeof person;**

**</script>**

**</body>**

**</html>**

## Undefined 和 Null 的区别

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)

**<!DOCTYPE html>**

**<html>**

**<body>**

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

**<script>**

**document.getElementById("demo").innerHTML =**

**typeof undefined + "<br>" +**

**typeof null + "<br>" +**

**(null === undefined) + "<br>" +**

**(null == undefined);**

**</script>**

**</body>**

**</html>**

# 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 中不同的数据类型：

* 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 (if myCar is not declared)  
typeof null                   // 返回 object

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

**<!DOCTYPE html>**

**<html>**

**<body>**

**<p> typeof 操作符返回变量、对象、函数、表达式的类型。</p>**

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

**<script>**

**document.getElementById("demo").innerHTML =**

**typeof "john" + "<br>" +**

**typeof 3.14 + "<br>" +**

**typeof NaN + "<br>" +**

**typeof false + "<br>" +**

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

**typeof {name:'john', age:34} + "<br>" +**

**typeof new Date() + "<br>" +**

**typeof function () {} + "<br>" +**

**typeof myCar + "<br>" +**

**typeof null;**

**</script>**

**</body>**

**</html>**

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)

**<!DOCTYPE html>**

**<html>**

**<body>**

**<p> constructor 属性返回变量或对象的构造函数。</p>**

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

**<script>**

**document.getElementById("demo").innerHTML =**

**"john".constructor + "<br>" +**

**(3.14).constructor + "<br>" +**

**false.constructor + "<br>" +**

**[1,2,3,4].constructor + "<br>" +**

**{name:'john', age:34}.constructor + "<br>" +**

**new Date().constructor + "<br>" +**

**function () {}.constructor;**

**</script>**

**</body>**

**</html>**

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

## 实例

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

**<!DOCTYPE html>**

**<html>**

**<body>**

**<!DOCTYPE html>**

**<html>**

**<body>**

**<p>判断是否为数组。</p>**

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

**<script>**

**var fruits = ["Banana", "Orange", "Apple", "Mango"];**

**document.getElementById("demo").innerHTML = isArray(fruits);**

**function isArray(myArray) {**

**return myArray.constructor.toString().indexOf("Array") > -1;**

**}**

**</script>**

**</body>**

**</html>**

你可以使用 constructor 属性来查看是对象是否为日期 (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)

**<!DOCTYPE html>**

**<html>**

**<body>**

**<p>判断是否为日期。</p>**

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

**<script>**

**var myDate = new Date();**

**document.getElementById("demo").innerHTML = isDate(myDate);**

**function isDate(myDate) {**

**return myDate.constructor.toString().indexOf("Date") > -1;**

**}**

**</script>**

**</body>**

**</html>**

## 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)

**<!DOCTYPE html>**

**<html>**

**<body>**

**<p> String() 方法可以将数字转换为字符串。</p>**

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

**<script>**

**var x = 123;**

**document.getElementById("demo").innerHTML =**

**String(x) + "<br>" +**

**String(123) + "<br>" +**

**String(100 + 23);**

**</script>**

**</body>**

**</html>**

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)

**<!DOCTYPE html>**

**<html>**

**<body>**

**<p>toString() 方法将数字转换为字符串。</p>**

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

**<script>**

**var x = 123;**

**document.getElementById("demo").innerHTML =**

**x.toString() + "<br>" +**

**(123).toString() + "<br>" +**

**(100 + 23).toString();**

**</script>**

**</body>**

**</html>**

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

全局方法 **String()** 可以将日期转换为字符串。

String(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

Date().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)

**<!DOCTYPE html>**

**<html>**

**<body>**

**<p> typeof 操作符返回变量或表达式的类型。</p>**

**<button onclick="myFunction()">点我</button>**

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

**<script>**

**function myFunction() {**

**var y = "5";**

**var x = + y;**

**document.getElementById("demo").innerHTML =**

**typeof y + "<br>" + typeof x;**

**}**

**</script>**

**</body>**

**</html>**

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)

**<!DOCTYPE html>**

**<html>**

**<body>**

**<p> typeof 操作符返回变量或表达式的类型。</p>**

**<button onclick="myFunction()">点我</button>**

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

**<script>**

**function myFunction() {**

**var y = "John";**

**var x = + y;**

**document.getElementById("demo").innerHTML =**

**typeof x + "<br>" + x;**

**}**

**</script>**

**</body>**

**</html>**

## 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 Type Conversion

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         because null is converted to 0  
"5" + null  // 返回"5null"   because null is converted to "null"  
"5" + 1     // 返回 "51"      because 1 is converted to "1"    
"5" - 1     // 返回 4         because "5" is converted to 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 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

/pattern/modifiers;

## 实例：

var patt = /w3cschool/i

实例解析：

**/w3cschool/i**  是一个正则表达式。

**w3cschool**  是一个模式 (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 "w3cschool" 字符串，且不区分大小写：

var str = "Visit w3cschool";  
var n = str.search(/w3cschool/i);

输出结果为：

6

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

**<!DOCTYPE html>**

**<html>**

**<body>**

**<p>搜索字符串 "w3cSchool", 并显示匹配的起始位置：</p>**

**<button onclick="myFunction()">点我</button>**

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

**<script>**

**function myFunction() {**

**var str = "Visit W3cSchool!";**

**var n = str.search(/w3cSchool/i);**

**document.getElementById("demo").innerHTML = n;**

**}**

**</script>**

**</body>**

**</html>**

## 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 "w3cschool" 的子串：

var str = "Visit w3cschool!";  
var n = str.search("w3cschool");

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

**<!DOCTYPE html>**

**<html>**

**<body>**

**<p>搜索字符串 "W3cSchool", 并显示匹配的起始位置：</p>**

**<button onclick="myFunction()">点我</button>**

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

**<script>**

**function myFunction() {**

**var str = "Visit W3cSchool!";**

**var n = str.search("W3cSchool");**

**document.getElementById("demo").innerHTML = n;**

**}**

**</script>**

**</body>**

**</html>**

## 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 w3cschool :

var str = "Visit Microsoft!";  
var res = str.replace(/microsoft/i, "w3cschool");

结果输出为:

Visit w3cschool!

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

**<!DOCTYPE html>**

**<html>**

**<body>**

**<p>替换 "microsoft" 为 "W3cSchool" :</p>**

**<button onclick="myFunction()">点我</button>**

**<p id="demo">Please visit Microsoft!</p>**

**<script>**

**function myFunction() {**

**var str = document.getElementById("demo").innerHTML;**

**var txt = str.replace(/microsoft/i,"W3cSchool");**

**document.getElementById("demo").innerHTML = txt;**

**}**

**</script>**

**</body>**

</html>

## replace() 方法使用字符串

replace() 方法将接收字符串作为参数：

var str = "Visit Microsoft!";  
var res = str.replace("Microsoft", "w3cschool");

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

**<!DOCTYPE html>**

**<html>**

**<body>**

**<p>替换 "Microsoft" 为 "W3cSchool" :</p>**

**<button onclick="myFunction()">点我</button>**

**<p id="demo">Please visit Microsoft!</p>**

**<script>**

**function myFunction() {**

**var str = document.getElementById("demo").innerHTML;**

**var txt = str.replace("Microsoft","W3cSchool");**

**document.getElementById("demo").innerHTML = txt;**

**}**

**</script>**

**</body>**

**</html>**

## 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)

**<!DOCTYPE html>**

**<html>**

**<body>**

**<script>**

**var patt1=new RegExp("e");**

**document.write(patt1.test("The best things in life are free"));**

**</script>**

**</body>**

**</html>**

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)

**<!DOCTYPE html>**

**<html>**

**<body>**

**<script>**

**var patt1=new RegExp("e");**

**document.write(patt1.exec("The best things in life are free"));**

**</script>**

**</body>**

**</html>**

## 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 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

<!DOCTYPE html>  
<html>  
<head>  
<script>  
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);  
  }  
}  
</script>  
</head>  
  
<body>  
<input type="button" value="查看消息" onclick="message()">  
</body>  
  
</html>

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

**<!DOCTYPE html>**

**<html>**

**<head>**

**<script>**

**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);**

**}**

**}**

**</script>**

**</head>**

**<body>**

**<input type="button" value="查看消息" onclick="message()" />**

**</body>**

**</html>**

## 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

<script>  
function myFunction()  
{  
try  
  {   
  var x=document.getElementById("demo").value;  
  if(x=="")    throw "empty";  
  if(isNaN(x)) throw "not a number";  
  if(x>10)     throw "too high";  
  if(x<5)      throw "too low";  
  }  
catch(err)  
  {  
  var y=document.getElementById("mess");  
  y.innerHTML="Error: " + err + ".";  
  }  
}  
</script>  
  
<h1>My First JavaScript</h1>  
<p>Please input a number between 5 and 10:</p>  
<input id="demo" type="text">  
<button type="button" onclick="myFunction()">Test Input</button>  
<p id="mess"></p>

[**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。

**<!DOCTYPE html>**

**<html>**

**<body>**

**<script>**

**function myFunction()**

**{**

**try**

**{**

**var x=document.getElementById("demo").value;**

**if(x=="") throw "值为空";**

**if(isNaN(x)) throw "不是数字";**

**if(x>10) throw "太大";**

**if(x<5) throw "太小";**

**}**

**catch(err)**

**{**

**var y=document.getElementById("mess");**

**y.innerHTML="错误：" + err + "。";**

**}**

**}**

**</script>**

**<h1>我的第一个 JavaScript 程序</h1>**

**<p>请输入 5 到 10 之间的数字：</p>**

**<input id="demo" type="text">**

**<button type="button" onclick="myFunction()">测试输入值</button>**

**<p id="mess"></p>**

**</body>**

**</html>**

# 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

<!DOCTYPE html>  
<html>  
<body>  
  
<h1>My First Web Page</h1>  
  
<script>  
a = 5;  
b = 6;  
c = a + b;  
console.log(c);  
</script>  
  
</body>  
</html>

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

<!DOCTYPE html>

<html>

<body>

<h1>我的第一个 Web 页面</h1>

<p>

浏览器中(Chrome, IE, Firefox) 使用 F12 来启用调试模式， 在调试窗口中点击 "Console" 菜单。

</p>

<script>

a = 5;

b = 6;

c = a + b;

console.log(c);

</script>

</body>

</html>

## 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)

<!DOCTYPE html>

<html>

<head>

</head>

<body>

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

<p>开启调试工具，在代码执行到第三行前会停止执行。</p>

<script>

var x = 15 \* 5;

debugger;

document.getElementById("demo").innerHTML = x;

</script>

</body>

</html>

## 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。
* 访问页面:  
  http://extentions.apple.com。
* 按说明操作：  
  install Firebug Lite。

**JavaScript 表单验证**

**JavaScript 表单验证**

JavaScript 可用来在数据被送往服务器前对 HTML 表单中的这些输入数据进行验证。

表单数据经常需要使用 JavaScript 来验证其正确性：

<!DOCTYPE html>

<html>

<head>

<script>

function validateForm()

{

var x=document.forms["myForm"]["fname"].value;

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

{

alert("姓必须填写");

return false;

}

}

</script>

</head>

<body>

<form name="myForm" action="demo-form.php" onsubmit="return validateForm()" method="post">

姓: <input type="text" name="fname">

<input type="submit" value="提交">

</form>

</body>

</html>

* 验证表单数据是否为空？

* 验证输入是否是一个正确的email地址？

* 验证日期是否输入正确？

* 验证表单输入内容是否为数字型？

**必填（或必选）项目**

下面的函数用来检查用户是否已填写表单中的必填（或必选）项目。假如必填或必选项为空，那么警告框会弹出，并且函数的返回值为 false，否则函数的返回值则为 true（意味着数据没有问题）：

function validateForm()  
{  
var x=document.forms["myForm"]["fname"].value;  
if (x==null || x=="")  
  {  
  alert("First name must be filled out");  
  return false;  
  }  
}

以上函数在 form 表单提交时被调用:

**实例**

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

## E-mail 验证

下面的函数检查输入的数据是否符合电子邮件地址的基本语法。

意思就是说，输入的数据必须包含 @ 符号和点号(.)。同时，@ 不可以是邮件地址的首字符，并且 @ 之后需有至少一个点号：

function validateForm()  
{  
var x=document.forms["myForm"]["email"].value;  
var atpos=x.indexOf("@");  
var dotpos=x.lastIndexOf(".");  
if (atpos<1 || dotpos<atpos+2 || dotpos+2>=x.length)  
  {  
  alert("Not a valid e-mail address");  
  return false;  
  }  
}

下面是连同 HTML 表单的完整代码：

## 实例

<form name="myForm" action="demo\_form.asp" onsubmit="return validateForm();" method="post">  
Email: <input type="text" name="email">  
<input type="submit" value="Submit">  
</form>

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

**<!DOCTYPE html>**

**<html>**

**<head>**

**<script>**

**function validateForm()**

**{**

**var x=document.forms["myForm"]["email"].value;**

**var atpos=x.indexOf("@");**

**var dotpos=x.lastIndexOf(".");**

**if (atpos<1 || dotpos<atpos+2 || dotpos+2>=x.length)**

**{**

**alert("Not a valid e-mail address");**

**return false;**

**}**

**}**

**</script>**

**</head>**

**<body>**

**<form name="myForm" action="demo-form.php" onsubmit="return validateForm();" method="post">**

**Email: <input type="text" name="email">**

**<input type="submit" value="Submit">**

**</form>**

**</body>**

**</html>**

# JavaScript 保留关键字

在 JavaScript 中，一些标识符是保留关键字，不能用作变量名或函数名。

## JavaScript 标准

所有的现代浏览器完全支持 ECMAScript 3（ES3，JavaScript 的第三版，从 1999 年开始）。

ECMAScript 4（ES4）未通过。

ECMAScript 5（ES5，2009 年发布），是 JavaScript 最新的官方版本。

随着时间的推移，我们开始看到，所有的现代浏览器已经完全支持 ES5。

## JavaScript 保留关键字

Javascript 的保留关键字不可以用作变量、标签或者函数名。有些保留关键字是作为 Javascript 以后扩展使用。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| abstract | arguments | boolean | break | byte |
| case | catch | char | class\* | const |
| continue | debugger | default | delete | do |
| double | else | enum\* | eval | export\* |
| extends\* | false | final | finally | float |
| for | function | goto | if | implements |
| import\* | in | instanceof | int | interface |
| let | long | native | new | null |
| package | private | protected | public | return |
| short | static | super\* | switch | synchronized |
| this | throw | throws | transient | true |
| try | typeof | var | void | volatile |
| while | with | yield |  |  |

\* 标记的关键字是 ECMAScript5 中新添加的。

## JavaScript 对象、属性和方法

您也应该避免使用 JavaScript 内置的对象、属性和方法的名称作为 Javascript 的变量或函数名：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Array | Date | eval | function | hasOwnProperty |
| Infinity | isFinite | isNaN | isPrototypeOf | length |
| Math | NaN | name | Number | Object |
| prototype | String | toString | undefined | valueOf |

## Java 保留关键字

JavaScript 经常与 Java 一起使用。您应该避免使用一些 Java 对象和属性作为 JavaScript 标识符：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| getClass | java | JavaArray | javaClass | JavaObject | JavaPackage |

## Windows 保留关键字

JavaScript 可以在 HTML 外部使用。它可在许多其他应用程序中作为编程语言使用。

在 HTML 中，您必须（为了可移植性，您也应该这么做）避免使用 HTML 和 Windows 对象和属性的名称作为 Javascript 的变量及函数名：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| alert | all | anchor | anchors | area |
| assign | blur | button | checkbox | clearInterval |
| clearTimeout | clientInformation | close | closed | confirm |
| constructor | crypto | decodeURI | decodeURIComponent | defaultStatus |
| document | element | elements | embed | embeds |
| encodeURI | encodeURIComponent | escape | event | fileUpload |
| focus | form | forms | frame | innerHeight |
| innerWidth | layer | layers | link | location |
| mimeTypes | navigate | navigator | frames | frameRate |
| hidden | history | image | images | offscreenBuffering |
| open | opener | option | outerHeight | outerWidth |
| packages | pageXOffset | pageYOffset | parent | parseFloat |
| parseInt | password | pkcs11 | plugin | prompt |
| propertyIsEnum | radio | reset | screenX | screenY |
| scroll | secure | select | self | setInterval |
| setTimeout | status | submit | taint | text |
| textarea | top | unescape | untaint | window |

## HTML 事件句柄

除此之外，您还应该避免使用 HTML 事件句柄的名称作为 Javascript 的变量及函数名。

实例：

|  |  |  |  |
| --- | --- | --- | --- |
| onblur | onclick | onerror | onfocus |
| onkeydown | onkeypress | onkeyup | onmouseover |
| onload | onmouseup | onmousedown | onsubmit |
|  |  |  |  |

## 非标准 JavaScript

除了保留关键字，在 JavaScript 实现中也有一些非标准的关键字。

一个实例是 **const** 关键字，用于定义变量。 一些 JavaScript 引擎把 const 当作 var 的同义词。另一些引擎则把 const 当作只读变量的定义。

Const 是 JavaScript 的扩展。JavaScript 引擎支持它用在 Firefox 和 Chrome 中。但是它并不是 JavaScript 标准 ES3 或 ES5 的组成部分。**建议：不要使用它**。

# 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 employees 对象: 3 条员工记录（对象）的数组:

## JSON Example

{"employees":[  
    {"firstName":"John", "lastName":"Doe"},   
    {"firstName":"Anna", "lastName":"Smith"},  
    {"firstName":"Peter", "lastName":"Jones"}  
]}

## 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：

"firstName":"John"

## 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：

{"firstName":"John", "lastName":"Doe"}

## 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：

"employees":[  
    {"firstName":"John", "lastName":"Doe"},   
    {"firstName":"Anna", "lastName":"Smith"},   
    {"firstName":"Peter", "lastName":"Jones"}  
]

在以上实例中，对象 "employees" 是一个数组。包含了三个对象。

每个为个对象为员工的记录（姓和名）。

## 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 = '{ "employees" : [' +  
'{ "firstName":"John" , "lastName":"Doe" },' +  
'{ "firstName":"Anna" , "lastName":"Smith" },' +  
'{ "firstName":"Peter" , "lastName":"Jones" } ]}';

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

<p id="demo"></p>  
  
<script>  
document.getElementById("demo").innerHTML =  
obj.employees[1].firstName + " " + obj.employees[1].lastName;  
</script>

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

<!DOCTYPE html>

<html>

<body>

<h2>为 JSON 字符串创建对象</h2>

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

<script>

var text = '{"employees":[' +

'{"firstName":"John","lastName":"Doe" },' +

'{"firstName":"Anna","lastName":"Smith" },' +

'{"firstName":"Peter","lastName":"Jones" }]}';

obj = JSON.parse(text);

document.getElementById("demo").innerHTML =

obj.employees[1].firstName + " " + obj.employees[1].lastName;

</script>

</body>

</html>

## 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: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)

**<html>**

**<head>**

**</head>**

**<body>**

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

**</body>**

**</html>**

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)

**<html>**

**<head>**

**<script type="text/javascript">**

**<!--**

**//-->**

**</script>**

**</head>**

**<body>**

**<p>点击以下链接查看结果：</p>**

**<a href="javascript:void(alert('Warning!!!'))">点我!</a>**

**</body>**

**</html>**

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>

**<html>**

**<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>**

**<body>**

**<p>点击以下按钮查看结果：</p>**

**<form>**

**<input type="button" value="点我" onclick="getValue();" />**

**</body>**

**</html>**

## 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><br> <p id="pos">尾部定位点</p>

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=tryjs_void3)

**<html>**

**<head>**

**<script type="text/javascript">**

**<!--**

**//-->**

**</script>**

**</head>**

**<body>**

**<p>点击以下链接查看不同效果：</p>**

**<a href="javascript:void(0);">点我没有反应的!</a>**

**<br>**

**<a href="#pos">点我定位到指定位置!</a>**

**<br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br>**

**<p id="pos">尾部定位点</p>**

**</body>**

**</html>**

# JavaScript 代码规范

所有的 JavaScript 项目适用同一种规范。

## JavaScript 代码规范

代码规范通常包括以下几个方面:

* 变量和函数的命名规则
* 空格，缩进，注释的使用规则。
* 其他常用规范……

规范的代码可以更易于阅读与维护。

代码规范一般在开发前规定，可以跟你的团队成员来协商设置。

## 变量名

变量名推荐使用驼峰法来命名(**camelCase**):

firstName = "John";  
lastName = "Doe";  
  
price = 19.90;  
tax = 0.20;  
  
fullPrice = price + (price \* tax);

## 空格与运算符

通常运算符 ( = + - \* / ) 前后需要添加空格:

### 实例:

var x = y + z;  
var values = ["Volvo", "Saab", "Fiat"];

## 代码缩进

通常使用 4 个空格符号来缩进代码块：

### 函数:

function toCelsius(fahrenheit) {  
    return (5 / 9) \* (fahrenheit - 32);  
}

|  |  |
| --- | --- |
| **Note** | 不推荐使用 TAB 键来缩进，因为不同编辑器 TAB 键的解析不一样。 |

## 语句规则

简单语句的通用规则:

* 一条语句通常以符号作为结束符。

### 实例:

var values = ["Volvo", "Saab", "Fiat"];  
  
var person = {  
    firstName: "John",  
    lastName: "Doe",  
    age: 50,  
    eyeColor: "blue"  
};

复杂语句的通用规则:

* 将左花括号放在第一行的结尾。
* 左花括号前添加一空格。
* 将右花括号独立放在一行。
* 不要以分号结束一个复杂的声明。

### 函数:

function toCelsius(fahrenheit) {  
    return (5 / 9) \* (fahrenheit - 32);  
}

### 循环:

for (i = 0; i < 5; i++) {  
    x += i;  
}

### 条件语句:

if (time < 20) {  
    greeting = "Good day";  
} else {  
    greeting = "Good evening";  
}

## 对象规则

对象定义的规则:

* 将左花括号与类名放在同一行。
* 冒号与属性值间有个空格。
* 字符串使用双引号，数字不需要。
* 最好一个属性-值对后面不要添加逗号。
* 将右花括号独立放在一行，并以符号作为结束符号。

### 实例:

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

短的对象代码可以直接写成一行:

### 实例:

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

## 每行代码字符小于 80

为了便于阅读每行字符建议小于数 80 个。

如果一个 JavaScript 语句超过了 80 个字符，建议在 运算符或者逗号后换行。

### 实例:

document.getElementById("demo").innerHTML =  
    "Hello Runoob.";

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

## 命名规则

一般很多代码语言的命名规则都是类似的，例如:

* 变量很函数为驼峰法（ **camelCase**）
* 全局变量为大写 (**UPPERCASE**)
* 常量 (如 PI) 为大写 (**UPPERCASE**)

变量命名你是否使用这几种规则： **hyp-hens**, **camelCase**, 或 **under\_scores** ?

**HTML 和 CSS 的横杆(-)字符:**

HTML5 属性可以以 data- (如：data-quantity, data-price) 作为前缀。

CSS 使用 - 来连接属性名 (font-size)。

|  |  |
| --- | --- |
| **Note** | - 通常在 JavaScript 中被认为是减法，所以不允许使用。 |

**下划线:**

很多程序员比较喜欢使用下划线(如：date\_of\_birth), 特别是在 SQL 数据库中。

PHP 语言通常都使用下划线。

**帕斯卡拼写法(PascalCase):**

帕斯卡拼写法(PascalCase) 在 C 语言中语言较多。

驼峰法：

JavaScript 中通常推荐使用驼峰法，jQuery 及其他 JavaScript 库都使用驼峰法。

|  |  |
| --- | --- |
| **Note** | 变量名不要以 $ 作为开始标记，会与很多 JavaScript 库冲突。 |

## HTML 载入外部 JavaScript 文件

使用简洁的格式载入 JavaScript 文件 ( type 属性不是必须的):

<script src="myscript.js">

## 使用 JavaScript 访问 HTML 元素

一个糟糕的 HTML 格式可能会导致 JavaScript 执行错误。

以下两个 JavaScript 语句会输出不同结果:

### 实例

var obj = getElementById("Demo")  
  
var obj = getElementById("demo")

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

HTML 与 JavaScript 尽量使用相同的命名规则。

[访问 HTML(5) 代码规范](http://www.runoob.com/html/html5-syntax.html)。

## 文件扩展名

HTML 文件后缀可以是 **.html** (或r **.htm**)。

CSS 文件后缀是 **.css** 。

JavaScript 文件后缀是 **.js** 。

## 使用小写文件名

大多 Web 服务器 (Apache, Unix) 对大小写敏感： london.jpg 不能通过 London.jpg 访问。

其他 Web 服务器 (Microsoft, IIS) 对大小写不敏感： london.jpg 可以通过 London.jpg 或 london.jpg 访问。

你必须保持统一的风格，我们建议统一使用小写的文件名。

**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;  
}

**<!DOCTYPE html>**

**<html>**

**<body>**

**<p>本例调用的函数会执行一个计算，然后返回结果：</p>**

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

**<script>**

**function myFunction(a,b)**

**{**

**return a\*b;**

**}**

**document.getElementById("demo").innerHTML=myFunction(4,3);**

**</script>**

**</body>**

**</html>**

|  |  |
| --- | --- |
| **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)

**<!DOCTYPE html>**

**<html>**

**<body>**

**<p>函数可以存储在变量中:</p>**

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

**<script>**

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

**document.getElementById("demo").innerHTML = x;**

**</script>**

**</body>**

**</html>**

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)

**<!DOCTYPE html>**

**<html>**

**<body>**

**<p>JavaScrip 内置构造函数。</p>**

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

**<script>**

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

**document.getElementById("demo").innerHTML = myFunction(4, 3);**

**</script>**

**</body>**

**</html>**

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);

**<!DOCTYPE html>**

**<html>**

**<body>**

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

**<script>**

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

**document.getElementById("demo").innerHTML = myFunction(4, 3);**

**</script>**

**</body>**

**</html>**

|  |  |
| --- | --- |
| **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)

**<!DOCTYPE html>**

**<html>**

**<body>**

**<p>函数可以自动调用：</p>**

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

**<script>**

**(function () {**

**document.getElementById("demo").innerHTML = "Hello! 我是自己调用的";**

**})();**

**</script>**

**</body>**

**</html>**

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)

**<!DOCTYPE html>**

**<html>**

**<body>**

**<p>函数可作为一个值：</p>**

**<p>x = myFunction(4,3) 或 x = 12</p>**

**<p>两种情况下，x 的值都为 12。</p>**

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

**<script>**

**function myFunction(a, b) {**

**return a \* b;**

**}**

**var x = myFunction(4, 3);**

**document.getElementById("demo").innerHTML = x;**

**</script>**

**</body>**

**</html>**

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)

**<!DOCTYPE html>**

**<html>**

**<body>**

**<p>函数可作为一个表达式使用。</p>**

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

**<script>**

**function myFunction(a, b) {**

**return a \* b;**

**}**

**var x = myFunction(4, 3) \* 2;**

**document.getElementById("demo").innerHTML = x;**

**</script>**

**</body>**

**</html>**

## 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)

**<!DOCTYPE html>**

**<html>**

**<body>**

**<p> arguments.length 属性返回函数接收到参数的个数：</p>**

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

**<script>**

**function myFunction(a, b) {**

**return arguments.length;**

**}**

**document.getElementById("demo").innerHTML = myFunction(4, 3);**

**</script>**

**</body>**

**</html>**

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)

**<!DOCTYPE html>**

**<html>**

**<body>**

**<p> toString() 将函数作为一个字符串返回：</p>**

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

**<script>**

**function myFunction(a, b) {**

**return a \* b;**

**}**

**document.getElementById("demo").innerHTML = myFunction.toString();**

**</script>**

**</body>**

**</html>**

|  |  |
| --- | --- |
| **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 函数参数

JavaScript 函数对参数的值(arguments)没有进行任何的检查。

## 函数显式参数与隐藏参数(arguments)

在先前的教程中，我们已经学习了函数的显式参数:

functionName(parameter1, parameter2, parameter3) {  
    code to be executed  
}

函数显式参数在函数定义时列出。

函数隐藏参数(arguments)在函数调用时传递给函数真正的值。

## 参数规则

JavaScript 函数定义时参数没有指定数据类型。

JavaScript 函数对隐藏参数(arguments)没有进行检测。

JavaScript 函数对隐藏参数(arguments)的个数没有进行检测。

## 默认参数

如果函数在调用时缺少参数，参数会默认设置为： **undefined**

有时这是可以接受的，但是建议最好为参数设置一个默认值：

## 实例

function myFunction(x, y) {  
    if (y === undefined) {  
          y = 0;  
    }   
}

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

**<!DOCTYPE html>**

**<html>**

**<body>**

**<p>设置参数参数的默认值。</p>**

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

**<script>**

**function myFunction(x, y) {**

**if (y === undefined) {**

**y = 0;**

**}**

**return x \* y;**

**}**

**document.getElementById("demo").innerHTML = myFunction(4);**

**</script>**

**</body>**

**</html>**

或者，更简单的方式：

**实例**

function myFunction(x, y) {  
    y = y || 0;  
}

**<!DOCTYPE html>**

**<html>**

**<body>**

**<p>设置函数参数默认值。</p>**

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

**<script>**

**function myFunction(x, y) {**

**y = y || 0;**

**return x \* y;**

**}**

**document.getElementById("demo").innerHTML = myFunction(4);**

**</script>**

**</body>**

**</html>**

|  |  |
| --- | --- |
| **Note** | 如果y已经定义 ， y || 返回 y, 因为 y 是 true, 否则返回 0, 因为 undefined 为 false。 |

如果函数调用时设置了过多的参数，参数将无法被引用，因为无法找到对应的参数名。 只能使用 arguments 对象来调用。

## Arguments 对象

JavaScript 函数有个内置的对象 arguments 对象.

argument 对象包含了函数调用的参数数组。

通过这种方式你可以很方便的找到最后一个参数的值：

## 实例

x = findMax(1, 123, 500, 115, 44, 88);  
  
function findMax() {  
    var i, max = 0;  
    for (i = 0; i < arguments.length; i++) {  
        if (arguments[i] > max) {  
            max = arguments[i];  
        }  
    }  
    return max;  
}

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

**<!DOCTYPE html>**

**<html>**

**<body>**

**<p>查找最大的数。</p>**

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

**<script>**

**function findMax() {**

**var i, max = 0;**

**for(i = 0; i < arguments.length; i++) {**

**if (arguments[i] > max) {**

**max = arguments[i];**

**}**

**}**

**return max;**

**}**

**document.getElementById("demo").innerHTML = findMax(4, 5, 6);**

**</script>**

**</body>**

**</html>**

或者创建一个函数用来统计所有数值的和：

## 实例

x = sumAll(1, 123, 500, 115, 44, 88);  
  
function sumAll() {  
    var i, sum = 0;  
    for (i = 0; i < arguments.length; i++) {  
        sum += arguments[i];  
    }  
    return sum;  
}

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

**<!DOCTYPE html>**

**<html>**

**<body>**

**<p>计算所有参数之和：</p>**

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

**<script>**

**function sumAll() {**

**var i, sum = 0;**

**for(i = 0; i < arguments.length; i++) {**

**sum += arguments[i];**

**}**

**return sum;**

**}**

**document.getElementById("demo").innerHTML =**

**sumAll(1, 123, 500, 115, 44, 88);**

**</script>**

**</body>**

**</html>**

**通过值传递参数**

在函数中调用的参数是函数的参数。

如果函数修改参数的值，将不会修改参数的初始值（在函数外定义）。

函数参数的改变不会影响函数外部的变量（局部变量）。

**通过对象传递参数**

在JavaScript中，可以引用对象的值。

因此我们在函数内部修改对象的属性就会修改其初始的值。

修改对象属性可作用于函数外部（全局变量）。

# JavaScript 函数调用

JavaScript 函数有 4 种调用方式。

每种方式的不同方式在于 **this** 的初始化。

## *****this***** 关键字

一般而言，在Javascript中，this指向函数执行时的当前对象。

|  |  |
| --- | --- |
| **Note** | 注意 **this** 是保留关键字，你不能修改 **this** 的值。 |

## 调用 JavaScript 函数

在之前的章节中我们已经学会了如何创建函数。

函数中的代码在函数被调用后执行。

## 作为一个函数调用

## 实例

function myFunction(a, b) {  
    return a \* b;  
}  
myFunction(10, 2);           // myFunction(10, 2) 返回 20

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

**<!DOCTYPE html>**

**<html>**

**<body>**

**<p>**

**全局函数 (myFunction) 返回参数参数相乘的结果：**

**</p>**

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

**<script>**

**function myFunction(a, b) {**

**return a \* b;**

**}**

**document.getElementById("demo").innerHTML = myFunction(10, 2);**

**</script>**

**</body>**

**</html>**

以上函数不属于任何对象。但是在 JavaScript 中它始终是默认的全局对象。

在 HTML 中默认的全局对象是 HTML 页面本身，所以函数是属于 HTML 页面。

在浏览器中的页面对象是浏览器窗口(window 对象)。以上函数会自动变为 window 对象的函数。

myFunction() 和 window.myFunction() 是一样的：

## 实例

function myFunction(a, b) {  
    return a \* b;  
}  
window.myFunction(10, 2);    // window.myFunction(10, 2) 返回 20

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=tryjs_invoke_function_2)

**<!DOCTYPE html>**

**<html>**

**<body>**

**<p>全局函数 myFunction() 会自动成为 window 对象的方法。</p>**

**<p>myFunction() is the same as window.myFunction().</p>**

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

**<script>**

**function myFunction(a, b) {**

**return a \* b;**

**}**

**document.getElementById("demo").innerHTML = window.myFunction(10, 2);**

**</script>**

**</body>**

**</html>**

|  |  |
| --- | --- |
| **Note** | 这是调用 JavaScript 函数常用的方法， 但不是良好的编程习惯  全局变量，方法或函数容易造成命名冲突的bug。 |

## 全局对象

当函数没有被自身的对象调用时， **this** 的值就会变成全局对象。

在 web 浏览器中全局对象是浏览器窗口（window 对象）。

该实例返回 **this** 的值是 window 对象:

## 实例

function myFunction() {  
    return this;  
}  
myFunction();                // 返回 window 对象

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=tryjs_invoke_function_3)

**<!DOCTYPE html>**

**<html>**

**<body>**

**<p>在 HTML 中 <b>this</b> 的值, 在全局函数是一个 window 对象。</p>**

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

**<script>**

**function myFunction() {**

**return this;**

**}**

**document.getElementById("demo").innerHTML = myFunction();**

**</script>**

**</body>**

**</html>**

|  |  |
| --- | --- |
| **Note** | 函数作为全局对象调用，会使 **this** 的值成为全局对象。 使用 window 对象作为一个变量容易造成程序崩溃。 |

## 函数作为方法调用

在 JavaScript 中你可以将函数定义为对象的方法。

以下实例创建了一个对象 (**myObject**), 对象有两个属性 (**firstName** 和 **lastName**), 及一个方法 (**fullName**):

## 实例

var myObject = {  
    firstName:"John",  
    lastName: "Doe",  
    fullName: function () {  
        return this.firstName + " " + this.lastName;  
    }  
}  
myObject.fullName();         // 返回 "John Doe"

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

**<!DOCTYPE html>**

**<html>**

**<body>**

**<p>myObject.fullName() 返回 John Doe:</p>**

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

**<script>**

**var myObject = {**

**firstName:"John",**

**lastName: "Doe",**

**fullName: function() {**

**return this.firstName + " " + this.lastName;**

**}**

**}**

**document.getElementById("demo").innerHTML = myObject.fullName();**

**</script>**

**</body>**

**</html>**

**fullName** 方法是一个函数。函数属于对象。 **myObject** 是函数的所有者。

**this**对象，拥有 JavaScript 代码。实例中 **this** 的值为 **myObject** 对象。

测试以下！修改 **fullName** 方法并返回 **this** 值:

## 实例

var myObject = {  
    firstName:"John",  
    lastName: "Doe",  
    fullName: function () {  
        return this;  
    }  
}  
myObject.fullName();          // 返回 [object Object] (所有者对象)

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=tryjs_invoke_method_2)

**<!DOCTYPE html>**

**<html>**

**<body>**

**<p>在一个对象方法中 ，<b>this</b> 的值是对象本身。</p>**

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

**<script>**

**var myObject = {**

**firstName:"John",**

**lastName: "Doe",**

**fullName: function() {**

**return this;**

**}**

**}**

**document.getElementById("demo").innerHTML = myObject.fullName();**

**</script>**

**</body>**

**</html>**

|  |  |
| --- | --- |
| **Note** | 函数作为对象方法调用，会使得 **this** 的值成为对象本身。 |

## 使用构造函数调用函数

如果函数调用前使用了 **new** 关键字, 则是调用了构造函数。

这看起来就像创建了新的函数，但实际上 JavaScript 函数是重新创建的对象：

## 实例

// 构造函数:  
function myFunction(arg1, arg2) {  
    this.firstName = arg1;  
    this.lastName  = arg2;  
}  
  
// This creates a new object  
var x = new myFunction("John","Doe");  
x.firstName;                             // 返回 "John"

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

**<!DOCTYPE html>**

**<html>**

**<body>**

**<p>该实例中， myFunction 是函数构造函数:</p>**

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

**<script>**

**function myFunction(arg1, arg2) {**

**this.firstName = arg1;**

**this.lastName = arg2;**

**}**

**var x = new myFunction("John","Doe")**

**document.getElementById("demo").innerHTML = x.firstName;**

**</script>**

**</body>**

**</html>**

构造函数的调用会创建一个新的对象。新对象会继承构造函数的属性和方法。

|  |  |
| --- | --- |
| **Note** | 构造函数中 **this** 关键字没有任何的值。 **this** 的值在函数调用时实例化对象(new object)时创建。 |

**作为函数方法调用函数**

在 JavaScript 中, 函数是对象。JavaScript 函数有它的属性和方法。

**call()** 和 **apply()** 是预定义的函数方法。 两个方法可用于调用函数，两个方法的第一个参数必须是对象本身。

**实例**

function myFunction(a, b) {  
    return a \* b;  
}  
myFunction.call(myObject, 10, 2);      // 返回 20

**实例**

function myFunction(a, b) {  
    return a \* b;  
}  
myArray = [10,2];  
myFunction.apply(myObject, myArray);   // 返回 20

两个方法都使用了对象本身作为第一个参数。 两者的区别在于第二个参数： apply传入的是一个参数数组，也就是将多个参数组合成为一个数组传入，而call则作为call的参数传入（从第二个参数开始）。

在 JavaScript 严格模式(strict mode)下, 在调用函数时第一个参数会成为 **this** 的值， 即使该参数不是一个对象。

在 JavaScript 非严格模式(non-strict mode)下, 如果第一个参数的值是 null 或 undefined, 它将使用全局对象替代。

|  |  |
| --- | --- |
| **Note** | 通过 call() 或 apply() 方法你可以设置 **this** 的值, 且作为已存在对象的新方法调用。 |

# 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)

**<!DOCTYPE html>**

**<html>**

**<body>**

**<p>函数可以访问函数内部定义的变量:</p>**

**<button type="button" onclick="myFunction()">点我</button>**

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

**<script>**

**function myFunction() {**

**var a = 4;**

**document.getElementById("demo").innerHTML = a \* a;**

**}**

**</script>**

**</body>**

**</html>**

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)

**<!DOCTYPE html>**

**<html>**

**<body>**

**<p>函数可以访问定义在函数外的变量：</p>**

**<button type="button" onclick="myFunction()">点我</button>**

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

**<script>**

**var a = 4;**

**function myFunction() {**

**document.getElementById("demo").innerHTML = a \* a;**

**}**

**</script>**

**</body>**

**</html>**

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** | 变量声明是如果不使用 **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() {  
    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)

**<!DOCTYPE html>**

**<html>**

**<body>**

**<p>全局变量计数。</p>**

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

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

**<script>**

**var counter = 0;**

**function add() {**

**return counter += 1;**

**}**

**function myFunction(){**

**document.getElementById("demo").innerHTML = add();**

**}**

**</script>**

**</body>**

**</html>**

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;  
    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)

**<!DOCTYPE html>**

**<html>**

**<body>**

**<p>局部变量计数。</p>**

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

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

**<script>**

**function add() {**

**var counter = 0;**

**return counter += 1;**

**}**

**function myFunction(){**

**document.getElementById("demo").innerHTML = add();**

**}**

**</script>**

**</body>**

**</html>**

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)

**<!DOCTYPE html>**

**<html>**

**<body>**

**<p>局部变量计数。</p>**

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

**<script>**

**document.getElementById("demo").innerHTML = add();**

**function add() {**

**var counter = 0;**

**function plus() {counter += 1;}**

**plus();**

**return counter;**

**}**

**</script>**

**</body>**

**</html>**

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)

**<!DOCTYPE html>**

**<html>**

**<body>**

**<p>局部变量计数。</p>**

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

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

**<script>**

**var add = (function () {**

**var counter = 0;**

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

**})();**

**function myFunction(){**

**document.getElementById("demo").innerHTML = add();**

**}**

**</script>**

**</body>**

**</html>**

**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。 |

**JavaScript HTML DOM**

通过 HTML DOM，可访问 JavaScript HTML 文档的所有元素。

**HTML DOM (文档对象模型)**

当网页被加载时，浏览器会创建页面的文档对象模型（Document Object Model）。

**HTML DOM** 模型被构造为**对象**的树：

**HTML DOM 树**



通过可编程的对象模型，JavaScript 获得了足够的能力来创建动态的 HTML。

* JavaScript 能够改变页面中的所有 HTML 元素
* JavaScript 能够改变页面中的所有 HTML 属性
* JavaScript 能够改变页面中的所有 CSS 样式
* JavaScript 能够对页面中的所有事件做出反应

## 查找 HTML 元素

通常，通过 JavaScript，您需要操作 HTML 元素。

为了做到这件事情，您必须首先找到该元素。有三种方法来做这件事：

* 通过 id 找到 HTML 元素
* 通过标签名找到 HTML 元素
* 通过类名找到 HTML 元素

## 通过 id 查找 HTML 元素

在 DOM 中查找 HTML 元素的最简单的方法，是通过使用元素的 id。

本例查找 id="intro" 元素：

## 实例

var x=document.getElementById("intro");

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

**<!DOCTYPE html>**

**<html>**

**<body>**

**<p id="intro">你好世界!</p>**

**<p>该实例展示了 <b>getElementById</b> 方法!</p>**

**<script>**

**x=document.getElementById("intro");**

**document.write("<p>文本来自 id 为 intro 段落: " + x.innerHTML + "</p>");**

**</script>**

**</body>**

**</html>**

如果找到该元素，则该方法将以对象（在 x 中）的形式返回该元素。

如果未找到该元素，则 x 将包含 null。

## 通过标签名查找 HTML 元素

本例查找 id="main" 的元素，然后查找 id="main" 元素中的所有 <p> 元素：

## 实例

var x=document.getElementById("main");  
var y=x.getElementsByTagName("p");

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

**<!DOCTYPE html>**

**<html>**

**<body>**

**<p>你好世界!</p>**

**<div id="main">**

**<p> DOM 是非常有用的。</p>**

**<p>该实例展示了 <b>getElementsByTagName</b> 方法</p>**

**</div>**

**<script>**

**var x=document.getElementById("main");**

**var y=x.getElementsByTagName("p");**

**document.write('id="main"元素中的第一个段落为：' + y[0].innerHTML);**

**</script>**

**</body>**

**</html>**

## 通过类名找到 HTML 元素

本例通过 [**getElementsByClassName**](http://www.runoob.com/jsref/met-document-getelementsbyclassname.html) 函数来查找 class="intro" 的元素：

## 实例

var x=document.getElementsByClassName("intro");

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

**<!DOCTYPE html>**

**<html>**

**<body>**

**<p class="intro">你好世界!</p>**

**<p>该实例展示了 <b>getElementsByClassName</b> 方法!</p>**

**<script>**

**x=document.getElementsByClassName("intro");**

**document.write("<p>文本来自 class 为 intro 段落: " + x[0].innerHTML + "</p>");**

**</script>**

**<p><b>注意：</b>Internet Explorer 8 及更早 IE 版本不支持 getElementsByClassName() 方法。</p>**

**</body>**

**</html>**

## HTML DOM 教程

在本教程接下来的篇幅中，您将学到：

* 如何改变 HTML 元素的内容 (innerHTML)
* 如何改变 HTML 元素的样式 (CSS)
* 如何对 HTML DOM 事件对出反应
* 如何添加或删除 HTML 元素

# JavaScript HTML DOM - 改变 HTML

HTML DOM 允许 JavaScript 改变 HTML 元素的内容。

## 改变 HTML 输出流

JavaScript 能够创建动态的 HTML 内容：

**今天的日期是： Sun Mar 20 2016 22:13:25 GMT+0800 (ä¸­å½æ åæ¶é´)**

在 JavaScript 中，document.write() 可用于直接向 HTML 输出流写内容。

## 实例

<!DOCTYPE html>  
<html>  
<body>  
  
<script>  
document.write(Date());  
</script>  
  
</body>  
</html>

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

**<!DOCTYPE html>**

**<html>**

**<body>**

**<script>**

**document.write(Date());**

**</script>**

**</body>**

**</html>**

|  |  |
| --- | --- |
| **lamp** | 绝对不要在文档加载完成之后使用 document.write()。这会覆盖该文档。 |

## 改变 HTML 内容

修改 HTML 内容的最简单的方法时使用 innerHTML 属性。

如需改变 HTML 元素的内容，请使用这个语法：

document.getElementById(*id*).innerHTML=*new HTML*

本例改变了 <p>元素的内容：

## 实例

<html>  
<body>  
  
<p id="p1">Hello World!</p>  
  
<script>  
document.getElementById("p1").innerHTML="New text!";  
</script>  
  
</body>  
</html>

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

**<!DOCTYPE html>**

**<html>**

**<body>**

**<p id="p1">Hello World!</p>**

**<script>**

**document.getElementById("p1").innerHTML="新文本!";**

**</script>**

**<p>以上段落通过脚本修改文本。</p>**

**</body>**

**</html>**

本例改变了 <h1> 元素的内容：

**实例**

<!DOCTYPE html>  
<html>  
<body>  
  
<h1 id="header">Old Header</h1>  
  
<script>  
var element=document.getElementById("header");  
element.innerHTML="New Header";  
</script>  
  
</body>  
</html>

**<!DOCTYPE html>**

**<html>**

**<body>**

**<h1 id="header">旧标题</h1>**

**<script>**

**var element=document.getElementById("header");**

**element.innerHTML="新标题";**

**</script>**

**<p>"旧标题" 被 "新标题" 取代。</p>**

**</body>**

**</html>**

实例讲解：

* 上面的 HTML 文档含有 id="header" 的 <h1> 元素

* 我们使用 HTML DOM 来获得 id="header" 的元素

* JavaScript 更改此元素的内容 (innerHTML)

## 改变 HTML 属性

如需改变 HTML 元素的属性，请使用这个语法：

document.getElementById(*id*).*attribute=new value*

本例改变了 <img> 元素的 src 属性：

## 实例

<!DOCTYPE html>  
<html>  
<body>  
  
<img id="image" src="smiley.gif">  
  
<script>  
document.getElementById("image").src="landscape.jpg";  
</script>  
  
</body>  
</html>

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

<!DOCTYPE html>

<html>

<body>

<img id="image" src="smiley.gif" width="160" height="120">

<script>

document.getElementById("image").src="landscape.jpg";

</script>

<p>原图片为 smiley.gif,脚本将图片修改为 landscape.jpg</p>

</body>

</html>

实例讲解：

* 上面的 HTML 文档含有 id="image" 的 <img> 元素
* 我们使用 HTML DOM 来获得 id="image" 的元素
* JavaScript 更改此元素的属性（把 "smiley.gif" 改为 "landscape.jpg"）
* **JavaScript HTML DOM - 改变CSS**
* HTML DOM 允许 JavaScript 改变 HTML 元素的样式。
* **改变 HTML 样式**
* 如需改变 HTML 元素的样式，请使用这个语法：
* document.getElementById(*id*).style.*property*=*new style*
* 下面的例子会改变 <p> 元素的样式：
* **实例**
* <html>  
  <body>  
    
  <p id="p2">Hello World!</p>  
    
  <script>  
  document.getElementById("p2").style.color="blue";  
  </script>  
    
  <p>以上段落通过脚本修改。</p>  
    
  </body>  
  </html>

**<!DOCTYPE html>**

**<html>**

**<body>**

**<p id="p1">Hello World!</p>**

**<p id="p2">Hello World!</p>**

**<script>**

**document.getElementById("p2").style.color="blue";**

**document.getElementById("p2").style.fontFamily="Arial";**

**document.getElementById("p2").style.fontSize="larger";**

**</script>**

**<p>以上段落通过脚本修改。</p>**

**</body>**

**</html>**

## 使用事件

HTML DOM 允许我们通过触发事件来执行代码。

比如以下事件：

* 元素被点击。
* 页面加载完成。
* 输入框被修改。
* ……

在接下来的章节，你会学到更多关于事件的知识。

本例改变了 id="id1" 的 HTML 元素的样式，当用户点击按钮时：

## 实例

<!DOCTYPE html>  
<html>  
<body>  
  
<h1 id="id1">My Heading 1</h1>  
<button type="button"   
onclick="document.getElementById('id1').style.color='red'">  
Click Me!</button>  
  
</body>  
</html>

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=trydhtml_dom_color2)

**<!DOCTYPE html>**

**<html>**

**<body>**

**<h1 id="id1">我的标题 1</h1>**

**<button type="button"**

**onclick="document.getElementById('id1').style.color='red'">**

**点我!</button>**

**</body>**

**</html>**

**<!DOCTYPE html>**

**<html>**

**<body>**

**<p id="p1">这是一个文本。 这是一个文本。 这是一个文本。 这是一个文本。 这是一个文本。 这是一个文本。 这是一个文本。</p>**

**<input type="button" value="隐藏文本" onclick="document.getElementById('p1').style.visibility='hidden'" />**

**<input type="button" value="显示文本" onclick="document.getElementById('p1').style.visibility='visible'" />**

**</body>**

**</html>**

# JavaScript HTML DOM 事件

HTML DOM 使 JavaScript 有能力对 HTML 事件做出反应。

## 实例

**Mouse Over Me**

## 对事件做出反应

我们可以在事件发生时执行 JavaScript，比如当用户在 HTML 元素上点击时。

如需在用户点击某个元素时执行代码，请向一个 HTML 事件属性添加 JavaScript 代码：

onclick=*JavaScript*

HTML 事件的例子：

* 当用户点击鼠标时
* 当网页已加载时
* 当图像已加载时
* 当鼠标移动到元素上时
* 当输入字段被改变时
* 当提交 HTML 表单时
* 当用户触发按键时

在本例中，当用户在 <h1> 元素上点击时，会改变其内容：

## 实例

<!DOCTYPE html>  
<html>  
<body>  
<h1 onclick="this.innerHTML='Ooops!'">点击文本!</h1>  
</body>  
</html>

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=trydhtml_event_onclick2)

**<!DOCTYPE html>**

**<html>**

**<body>**

**<h1 onclick="this.innerHTML='Ooops!'">点击文本!</h1>**

**</body>**

**</html>**

本例从事件处理器调用一个函数：

## 实例

<!DOCTYPE html>  
<html>  
<head>  
<script>  
function changetext(id)  
{  
id.innerHTML="Ooops!";  
}  
</script>  
</head>  
<body>  
<h1 onclick="changetext(this)">Click on this text!</h1>  
</body>  
</html>

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=trydhtml_event_onclick3)

**<!DOCTYPE html>**

**<html>**

**<head>**

**<script>**

**function changetext(id)**

**{**

**id.innerHTML="Ooops!";**

**}**

**</script>**

**</head>**

**<body>**

**<h1 onclick="changetext(this)">点击文本!</h1>**

**</body>**

**</html>**

## HTML 事件属性

如需向 HTML 元素分配 事件，您可以使用事件属性。

## 实例

向 button 元素分配 onclick 事件：

<button onclick="displayDate()">Try it</button>

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=tryjs_events1)

**<!DOCTYPE html>**

**<html>**

**<body>**

**<p>点击按钮执行 <em>displayDate()</em> 函数.</p>**

**<button onclick="displayDate()">点这里</button>**

**<script>**

**function displayDate()**

**{**

**document.getElementById("demo").innerHTML=Date();**

**}**

**</script>**

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

**</body>**

**</html>**

在上面的例子中，名为 displayDate 的函数将在按钮被点击时执行。

## 使用 HTML DOM 来分配事件

HTML DOM 允许您使用 JavaScript 来向 HTML 元素分配事件：

## 实例

向 button 元素分配 onclick 事件：

<script>  
document.getElementById("myBtn").onclick=function(){displayDate()};  
</script>

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=tryjs_events2)

**<!DOCTYPE html>**

**<html>**

**<head>**

**</head>**

**<body>**

**<p>点击按钮执行 <em>displayDate()</em> 函数.</p>**

**<button id="myBtn">点这里</button>**

**<script>**

**document.getElementById("myBtn").onclick=function(){displayDate()};**

**function displayDate()**

**{**

**document.getElementById("demo").innerHTML=Date();**

**}**

**</script>**

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

**</body>**

**</html>**

在上面的例子中，名为 displayDate 的函数被分配给 id=myButn" 的 HTML 元素。

按钮点击时Javascript函数将会被执行。

## onload 和 onunload 事件

onload 和 onunload 事件会在用户进入或离开页面时被触发。

onload 事件可用于检测访问者的浏览器类型和浏览器版本，并基于这些信息来加载网页的正确版本。

onload 和 onunload 事件可用于处理 cookie。

## 实例

<body onload="checkCookies()">

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

**<!DOCTYPE html>**

**<html>**

**<body onload="checkCookies()">**

**<script>**

**function checkCookies()**

**{**

**if (navigator.cookieEnabled==true)**

**{**

**alert("Cookies 可用")**

**}**

**else**

**{**

**alert("Cookies 不可用")**

**}**

**}**

**</script>**

**<p>弹窗-提示浏览器cookie是否可用。</p>**

**</body>**

**</html>**

## onchange 事件

onchange 事件常结合对输入字段的验证来使用。

下面是一个如何使用 onchange 的例子。当用户改变输入字段的内容时，会调用 upperCase() 函数。

## 实例

<input type="text" id="fname" onchange="upperCase()">

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

**<!DOCTYPE html>**

**<html>**

**<head>**

**<script>**

**function myFunction()**

**{**

**var x=document.getElementById("fname");**

**x.value=x.value.toUpperCase();**

**}**

**</script>**

**</head>**

**<body>**

**输入你的名字: <input type="text" id="fname" onchange="myFunction()">**

**<p>当你离开输入框后，函数将被触发，将小写字母转为大写字母。</p>**

**</body>**

**</html>**

## onmouseover 和 onmouseout 事件

onmouseover 和 onmouseout 事件可用于在用户的鼠标移至 HTML 元素上方或移出元素时触发函数。

## 实例

一个简单的 onmouseover-onmouseout 实例：

**Mouse Over Me**

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

**<!DOCTYPE html>**

**<html>**

**<body>**

**<div onmouseover="mOver(this)" onmouseout="mOut(this)" style="background-color:#D94A38;width:120px;height:20px;padding:40px;">Mouse Over Me</div>**

**<script>**

**function mOver(obj)**

**{**

**obj.innerHTML="Thank You"**

**}**

**function mOut(obj)**

**{**

**obj.innerHTML="Mouse Over Me"**

**}**

**</script>**

**</body>**

**</html>**

**onmousedown、onmouseup 以及 onclick 事件**

onmousedown, onmouseup 以及 onclick 构成了鼠标点击事件的所有部分。首先当点击鼠标按钮时，会触发 onmousedown 事件，当释放鼠标按钮时，会触发 onmouseup 事件，最后，当完成鼠标点击时，会触发 onclick 事件。

**实例**

一个简单的 onmousedown-onmouseup 实例：

**Thank You**

**<!DOCTYPE html>**

**<html>**

**<head>**

**<script>**

**function lighton()**

**{**

**document.getElementById('myimage').src="bulbon.gif";**

**}**

**function lightoff()**

**{**

**document.getElementById('myimage').src="bulboff.gif";**

**}**

**</script>**

**</head>**

**<body>**

**<img id="myimage" onmousedown="lighton()" onmouseup="lightoff()" src="bulboff.gif" width="100" height="180" />**

**<p>点击不释放鼠标灯将一直亮着!</p>**

**</body>**

**</html>**

**<!DOCTYPE html>**

**<html>**

**<head>**

**<script>**

**function mymessage()**

**{**

**alert("消息在 onload 事件触发后弹出。");**

**}**

**</script>**

**</head>**

**<body onload="mymessage()">**

**</body>**

**</html>**

**<!DOCTYPE html>**

**<html>**

**<head>**

**<script>**

**function myFunction(x)**

**{**

**x.style.background="yellow";**

**}**

**</script>**

**</head>**

**<body>**

**输入你的名字: <input type="text" onfocus="myFunction(this)">**

**<p>当输入框获取焦点时，修改背景色（background-color属性） 将被触发。</p>**

**</body>**

**</html>**

**<!DOCTYPE html>**

**<html>**

**<body>**

**<h1 onmouseover="style.color='red'"**

**onmouseout="style.color='black'">**

**将鼠标移至文部上</h1>**

**</body>**

**</html>**

# JavaScript HTML DOM EventListener

## addEventListener() 方法

## 实例

点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)

**<!DOCTYPE html>**

**<html>**

**<body>**

**<p>该实例使用 addEventListener() 方法在按钮中添加点击事件。 </p>**

**<button id="myBtn">点我</button>**

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

**<script>**

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

**function displayDate() {**

**document.getElementById("demo").innerHTML = Date();**

**}**

**</script>**

**</body>**

**</html>**

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)

**<!DOCTYPE html>**

**<html>**

**<body>**

**<p>该实例使用 addEventListener() 方法在按钮中添加点击事件。 </p>**

**<button id="myBtn">点我</button>**

**<script>**

**document.getElementById("myBtn").addEventListener("click", function(){**

**alert("Hello World!");**

**});**

**</script>**

**</body>**

**</html>**

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)

**<!DOCTYPE html>**

**<html>**

**<body>**

**<p>该实例使用 addEventListener() 方法在用户点击按钮时执行函数。</p>**

**<button id="myBtn">点我</button>**

**<script>**

**document.getElementById("myBtn").addEventListener("click", myFunction);**

**function myFunction() {**

**alert ("Hello World!");**

**}**

**</script>**

**</body>**

**</html>**

## 向同一个元素中添加多个事件句柄

addEventListener() 方法允许向同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)

**<!DOCTYPE html>**

**<html>**

**<body>**

**<p>该实例使用 addEventListener() 方法向同个按钮中添加两个点击事件。</p>**

**<button id="myBtn">点我</button>**

**<script>**

**var x = document.getElementById("myBtn");**

**x.addEventListener("click", myFunction);**

**x.addEventListener("click", someOtherFunction);**

**function myFunction() {**

**alert ("Hello World!")**

**}**

**function someOtherFunction() {**

**alert ("函数已执行!")**

**}**

**</script>**

**</body>**

**</html>**

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)

**<!DOCTYPE html>**

**<html>**

**<body>**

**<p>实例使用 addEventListener() 方法在同一个按钮中添加多个事件。</p>**

**<button id="myBtn">点我</button>**

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

**<script>**

**var x = document.getElementById("myBtn");**

**x.addEventListener("mouseover", myFunction);**

**x.addEventListener("click", mySecondFunction);**

**x.addEventListener("mouseout", myThirdFunction);**

**function myFunction() {**

**document.getElementById("demo").innerHTML += "Moused over!<br>"**

**}**

**function mySecondFunction() {**

**document.getElementById("demo").innerHTML += "Clicked!<br>"**

**}**

**function myThirdFunction() {**

**document.getElementById("demo").innerHTML += "Moused out!<br>"**

**}**

**</script>**

**</body>**

**</html>**

## 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)

**<!DOCTYPE html>**

**<html>**

**<body>**

**<p>实例在window对象中使用 addEventListener() 方法。</p>**

**<p>尝试重置浏览器的窗口触发 "resize" 事件句柄。</p>**

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

**<script>**

**window.addEventListener("resize", function(){**

**document.getElementById("demo").innerHTML = Math.random();**

**});**

**</script>**

**</body>**

**</html>**

## 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)

**<!DOCTYPE html>**

**<html>**

**<body>**

**<p>实例演示了在使用 addEventListener() 方法时如何传递参数。</p>**

**<p>点击按钮执行计算。</p>**

**<button id="myBtn">点我</button>**

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

**<script>**

**var p1 = 5;**

**var p2 = 7;**

**document.getElementById("myBtn").addEventListener("click", function() {**

**myFunction(p1, p2);**

**});**

**function myFunction(a, b) {**

**var result = a \* b;**

**document.getElementById("demo").innerHTML = result;**

**}**

**</script>**

**</body>**

**</html>**

## 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)

**<!DOCTYPE html>**

**<html>**

**<head>**

**<style>**

**div {**

**background-color: coral;**

**border: 1px solid;**

**padding: 50px;**

**}**

**</style>**

**</head>**

**<body>**

**<p>实例演示了在添加不同事件监听时，冒泡与捕获的不同。</p>**

**<div id="myDiv">**

**<p id="myP">点击段落，我是冒泡。</p>**

**</div><br>**

**<div id="myDiv2">**

**<p id="myP2">点击段落，我是捕获。 </p>**

**</div>**

**<script>**

**document.getElementById("myP").addEventListener("click", function() {**

**alert("你点击了 P 元素!");**

**}, false);**

**document.getElementById("myDiv").addEventListener("click", function() {**

**alert(" 你点击了 DIV 元素 !");**

**}, false);**

**document.getElementById("myP2").addEventListener("click", function() {**

**alert("你点击了 P 元素!");**

**}, true);**

**document.getElementById("myDiv2").addEventListener("click", function() {**

**alert("你点击了 DIV 元素 !");**

**}, true);**

**</script>**

**</body>**

**</html>**

## 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)

**<!DOCTYPE html>**

**<html>**

**<head>**

**<style>**

**#myDIV {**

**background-color: coral;**

**border: 1px solid;**

**padding: 50px;**

**color: white;**

**}**

**</style>**

**</head>**

**<body>**

**<div id="myDIV"> div 元素添加了 onmousemove 事件句柄，鼠标在桔红色的框内移动时会显示随机数。**

**<p>点击按钮移除 DIV 的事件句柄。</p>**

**<button onclick="removeHandler()" id="myBtn">点我</button>**

**</div>**

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

**<script>**

**document.getElementById("myDIV").addEventListener("mousemove", myFunction);**

**function myFunction() {**

**document.getElementById("demo").innerHTML = Math.random();**

**}**

**function removeHandler() {**

**document.getElementById("myDIV").removeEventListener("mousemove", myFunction);**

**}**

**</script>**

**</body>**

**</html>**

## 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)

<!DOCTYPE html>

<html>

<body>

<p> Internet Explorer 8 及更早IE版本不支持addEventListener() 方法。</p>

<p>该实例演示了所有浏览器兼容的解决方法。</p>

<button id="myBtn">点我</button>

<script>

var x = document.getElementById("myBtn");

if (x.addEventListener) {

x.addEventListener("click", myFunction);

} else if (x.attachEvent) {

x.attachEvent("onclick", myFunction);

}

function myFunction() {

alert("Hello World!");

}

</script>

</body>

</html>

## 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 HTML DOM 元素(节点)

创建新的 HTML 元素

## 创建新的 HTML 元素

如需向 HTML DOM 添加新元素，您必须首先创建该元素（元素节点），然后向一个已存在的元素追加该元素。

## 实例

<div id="div1">  
<p id="p1">这是一个段落。</p>  
<p id="p2">这是另一个段落。</p>  
</div>  
  
<script>  
var para=document.createElement("p");  
var node=document.createTextNode("这是一个新段落。");  
para.appendChild(node);  
  
var element=document.getElementById("div1");  
element.appendChild(para);  
</script>

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

**<!DOCTYPE html>**

**<html>**

**<body>**

**<div id="div1">**

**<p id="p1">这是一个段落。</p>**

**<p id="p2">这是另一个段落。</p>**

**</div>**

**<script>**

**var para=document.createElement("p");**

**var node=document.createTextNode("这是一个新段落。");**

**para.appendChild(node);**

**var element=document.getElementById("div1");**

**element.appendChild(para);**

**</script>**

**</body>**

**</html>**

## 例子解析：

这段代码创建新的<p> 元素：

var para=document.createElement("p");

如需向 <p> 元素添加文本，您必须首先创建文本节点。这段代码创建了一个文本节点：

var node=document.createTextNode("这是一个新段落。");

然后您必须向 <p> 元素追加这个文本节点：

para.appendChild(node);

最后您必须向一个已有的元素追加这个新元素。

这段代码找到一个已有的元素：

var element=document.getElementById("div1");

以下代码在已存在的元素后添加新元素：

element.appendChild(para);

## 删除已有的 HTML 元素

以下代码演示了如何删除元素：

## 实例

<div id="div1">  
<p id="p1">这是一个段落。</p>  
<p id="p2">这是另一个段落。</p>  
</div>

<script>  
var parent=document.getElementById("div1");  
var child=document.getElementById("p1");  
parent.removeChild(child);  
</script>

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

**<!DOCTYPE html>**

**<html>**

**<body>**

**<div id="div1">**

**<p id="p1">这是一个段落。</p>**

**<p id="p2">这是另一个段落。</p>**

**</div>**

**<script>**

**var parent=document.getElementById("div1");**

**var child=document.getElementById("p1");**

**parent.removeChild(child);**

**</script>**

**</body>**

**</html>**

## 实例解析

这个 HTML 文档含有拥有两个子节点（两个 <p> 元素）的 <div> 元素：

<div id="div1">  
<p id="p1">这是一个段落。</p>  
<p id="p2">这是另一个段落。</p>  
</div>

找到 id="div1" 的元素：

var parent=document.getElementById("div1");

找到 id="p1" 的 <p> 元素：

var child=document.getElementById("p1");

从父元素中删除子元素：

parent.removeChild(child);

|  |  |
| --- | --- |
| **lamp** | 如果能够在不引用父元素的情况下删除某个元素，就太好了。 不过很遗憾。DOM 需要清楚您需要删除的元素，以及它的父元素。 |

这是常用的解决方案：找到您希望删除的子元素，然后使用其 parentNode 属性来找到父元素：

var child=document.getElementById("p1");  
child.parentNode.removeChild(child);

## HTML DOM 教程

在我们的 JavaScript 教程的 HTML DOM 部分，您已经学到了：

* 如何改变 HTML 元素的内容 (innerHTML)
* 如何改变 HTML 元素的样式 (CSS)
* 如何对 HTML DOM 事件作出反应
* 如何添加或删除 HTML 元素

如果您希望学到更多有关使用 JavaScript 访问 HTML DOM 的知识，请访问我们完整的 [HTML DOM 教程](http://www.runoob.com/htmldom/htmldom-tutorial.html)。

# 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)

<!DOCTYPE html>

<html>

<body>

<script>

var person=new Object();

person.firstname="John";

person.lastname="Doe";

person.age=50;

person.eyecolor="blue";

document.write(person.firstname + " is " + person.age + " years old.");

</script>

</body>

</html>

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)

<!DOCTYPE html>

<html>

<body>

<script>

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

document.write(person.firstname + " is " + person.age + " years old.");

</script>

</body>

</html>

## 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)

<!DOCTYPE html>

<html>

<body>

<script>

function person(firstname,lastname,age,eyecolor)

{

this.firstname=firstname;

this.lastname=lastname;

this.age=age;

this.eyecolor=eyecolor;

}

myFather=new person("John","Doe",50,"blue");

document.write(myFather.firstname + " is " + myFather.age + " years old.");

</script>

</body>

</html>

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)

<!DOCTYPE html>

<html>

<body>

<script>

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;

}

}

myMother=new person("Sally","Rally",48,"green");

myMother.changeName("Doe");

document.write(myMother.lastname);

</script>

</body>

</html>

## 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*)  
  {  
*code to be executed*  
  }

**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)

<!DOCTYPE html>

<html>

<body>

<p>点击下面的按钮，循环遍历对象 "person" 的属性。</p>

<button onclick="myFunction()">点击这里</button>

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

<script>

function myFunction()

{

var x;

var txt="";

var person={fname:"Bill",lname:"Gates",age:56};

for (x in person)

{

txt=txt + person[x];

}

document.getElementById("demo").innerHTML=txt;

}

</script>

</body>

</html>

# 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。

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; // result will be 0.30000000000000004

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

<!DOCTYPE html>

<html>

<body>

<script>

var x;

document.write("<p>Only 17 digits: ");

x=12345678901234567890;

document.write(x + "</p>");

document.write("<p>0.2 + 0.1 = ");

x=0.2+0.1;

document.write(x + "</p>");

document.write("<p>It helps multiplying and dividing by 10: ");

x=(0.2\*10+0.1\*10)/10;

document.write(x +"</p>");

</script>

</body>

</html>

## 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)

<!DOCTYPE html>

<html>

<body>

<script>

var y = 0377;

var z = 0xFF;

document.write(y + "<br>");

document.write(z + "<br>");

</script>

</body>

</html>

|  |  |
| --- | --- |
| **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);   // returns 80  
myNumber.toString(8);    // returns 200  
myNumber.toString(2);    // returns 10000000

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

<!DOCTYPE html>

<html>

<body>

<script>

var myNumber = 128;

document.write(myNumber + ' decimal<br>');

document.write(myNumber.toString(16) + ' hex<br>');

document.write(myNumber.toString(8) + ' octal<br>');

document.write(myNumber.toString(2) + ' binary<br>');

</script>

</body>

</html>

## 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; // Calculate until Infinity  
}

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

<!DOCTYPE html>

<html>

<body>

<script>

myNumber=2;

while (myNumber!=Infinity)

{

myNumber=myNumber\*myNumber;

document.write(myNumber +'<BR>');

}

</script>

</body>

</html>

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)

<!DOCTYPE html>

<html>

<body>

<script>

var x = 2/0;

var y = -2/0;

document.write(x + "<br>");

document.write(y + "<br>");

</script>

</body>

</html>

## 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); // returns true  
var y = 100 / "1000";  
isNaN(y); // returns false

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

<!DOCTYPE html>

<html>

<body>

<p>A number divided by a string is not a number</p>

<p>A number divided by a numeric string is a number</p>

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

<script>

var x = 1000 / "Apple";

var y = 1000 / "1000";

document.getElementById("demo").innerHTML = isNaN(x) + "<br>" + isNaN(y);

</script>

</body>

</html>

除以0是无穷大，无穷大是一个数字:

## 实例

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

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

<!DOCTYPE html>

<html>

<body>

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

<script>

var y = 1000 / 0;

document.getElementById("demo").innerHTML = isNaN(y);

</script>

</body>

</html>

## 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) // returns Number  
typeof(y) // returns Object

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

<!DOCTYPE html>

<html>

<body>

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

<script>

var x = 123; // x is a number

var y = new Number(123); // y is an object

var txt = typeof(x) + " " + typeof(y);

document.getElementById("demo").innerHTML=txt;

</script>

</body>

</html>

## 实例

var x = 123;                
var y = new Number(123);  
(x === y) // is false because x is a number and y is an object.

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

<!DOCTYPE html>

<html>

<body>

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

<script>

var x = 123; // x is a number

var y = new Number(123); // y is an object

document.getElementById("demo").innerHTML = x===y;

</script>

</body>

</html>

**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 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)

<!DOCTYPE html>

<html>

<body>

<script>

var carname1="Volvo XC60";

var carname2='Volvo XC60';

var answer1="It's alright";

var answer2="He is called 'Johnny'";

var answer3='He is called "Johnny"';

document.write(carname1 + "<br>")

document.write(carname2 + "<br>")

document.write(answer1 + "<br>")

document.write(answer2 + "<br>")

document.write(answer3 + "<br>")

</script>

</body>

</html>

## 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)

<!DOCTYPE html>

<html>

<body>

<script>

var txt = "Hello World!";

document.write("<p>" + txt.length + "</p>");

var txt="ABCDEFGHIJKLMNOPQRSTUVWXYZ";

document.write("<p>" + txt.length + "</p>");

</script>

</body>

</html>

## 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)

<!DOCTYPE html>

<html>

<body>

<p id="p1">Click the button to locate where "locate" first occurs.</p>

<p id="p2">0</p>

<button onclick="myFunction()">Try it</button>

<script>

function myFunction()

{

var str=document.getElementById("p1").innerHTML;

var n=str.indexOf("locate");

document.getElementById("p2").innerHTML=n+1;

}

</script>

</body>

</html>

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)

<!DOCTYPE html>

<html>

<body>

<script>

var str="Hello world!";

document.write(str.match("world") + "<br>");

document.write(str.match("World") + "<br>");

document.write(str.match("world!"));

</script>

</body>

</html>

## 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","w3cschool");

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

<!DOCTYPE html>

<html>

<body>

<p>替换 "Microsoft" 为 "W3cSchool" :</p>

<button onclick="myFunction()">点我</button>

<p id="demo">Please visit Microsoft!</p>

<script>

function myFunction() {

var str = document.getElementById("demo").innerHTML;

var txt = str.replace("Microsoft","W3cSchool");

document.getElementById("demo").innerHTML = txt;

}

</script>

</body>

</html>

## 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 is txt converted to upper  
var txt2=txt.toLowerCase();   // txt2 is txt converted to lower

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

<!DOCTYPE html>

<html>

<body>

<script>

var txt="Hello World!";

document.write("<p>" + txt.toUpperCase() + "</p>");

document.write("<p>" + txt.toLowerCase() + "</p>");

document.write("<p>" + txt + "</p>");

</script>

<p>

The methods returns a new string.

The original string is not changed.

</p>

</body>

</html>

## 字符串转为数组

字符串使用strong>split()函数转为数组:

## 实例

txt="a,b,c,d,e"   // String  
txt.split(",");   // Split on commas  
txt.split(" ");   // Split on spaces  
txt.split("|");   // Split on pipe

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

<!DOCTYPE html>

<html>

<body>

<p id="demo">Click the button to display the array values after the split.</p>

<button onclick="myFunction()">Try it</button>

<script>

function myFunction()

{

var str="a,b,c,d,e,f";

var n=str.split(",");

document.getElementById("demo").innerHTML=n[0];

}

</script>

</body>

</html>

**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 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)

<!DOCTYPE html>

<html>

<body>

<p id="demo">Click the button to display the full year of todays date.</p>

<button onclick="myFunction()">Try it</button>

<script>

function myFunction()

{

var d = new Date();

var x = document.getElementById("demo");

x.innerHTML=d.getFullYear();

}

</script>

</body>

</html>

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。

<!DOCTYPE html>

<html>

<body>

<p id="demo">Click the button to display the number of milliseconds since midnight, January 1, 1970.</p>

<button onclick="myFunction()">Try it</button>

<script>

function myFunction()

{

var d = new Date();

var x = document.getElementById("demo");

x.innerHTML=d.getTime();

}

</script>

</body>

</html>

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

<!DOCTYPE html>

<html>

<body>

<p id="demo">Click the button to display a date after changing the year, month, and day.</p>

<button onclick="myFunction()">Try it</button>

<script>

function myFunction()

{

var d = new Date();

d.setFullYear(2020,10,3);

var x = document.getElementById("demo");

x.innerHTML=d;

}

</script>

<p>Remember that JavaScript counts months from 0 to 11.

Month 10 is November.</p>

</body>

</html>

[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。

<!DOCTYPE html>

<html>

<body>

<p id="demo">Click the button to display the UTC date and time as a string.</p>

<button onclick="myFunction()">Try it</button>

<script>

function myFunction()

{

var d = new Date();

var x = document.getElementById("demo");

x.innerHTML=d.toUTCString();

}

</script>

</body>

</html>

[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。

<!DOCTYPE html>

<html>

<body>

<p id="demo">Click the button to display todays day of the week.</p>

<button onclick="myFunction()">Try it</button>

<script>

function myFunction()

{

var d = new Date();

var weekday=new Array(7);

weekday[0]="Sunday";

weekday[1]="Monday";

weekday[2]="Tuesday";

weekday[3]="Wednesday";

weekday[4]="Thursday";

weekday[5]="Friday";

weekday[6]="Saturday";

var x = document.getElementById("demo");

x.innerHTML=weekday[d.getDay()];

}

</script>

</body>

</html>

[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。

<!DOCTYPE html>

<html>

<head>

<script>

function startTime()

{

var today=new Date();

var h=today.getHours();

var m=today.getMinutes();

var s=today.getSeconds();

// add a zero in front of numbers<10

m=checkTime(m);

s=checkTime(s);

document.getElementById('txt').innerHTML=h+":"+m+":"+s;

t=setTimeout(function(){startTime()},500);

}

function checkTime(i)

{

if (i<10)

{

i="0" + i;

}

return i;

}

</script>

</head>

<body onload="startTime()">

<div id="txt"></div>

</body>

</html>

## 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: 'Open Sans', 'Helvetica Neue', Helvetica, Arial, STHeiti, 'Microsoft Yahei', sans-serif; font-size: 12px; font-style: normal; font-variant: normal; font-weight: normal; letter-spacing: normal; line-height: normal; orphans: auto; text-align: start; text-indent: 0px; text-transform: none; white-space: normal; widows: 1; word-spacing: 0px; -webkit-text-stroke-width: 0px; background-color: rgb(255, 255, 255);">

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("Today is before 14th January 2100");  
  }  
else  
  {  
  alert("Today is after 14th January 2100");  
  }

# 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)

<!DOCTYPE html>

<html>

<body>

<script>

var i;

var mycars = new Array();

mycars[0] = "Saab";

mycars[1] = "Volvo";

mycars[2] = "BMW";

for (i=0;i<mycars.length;i++)

{

document.write(mycars[i] + "<br>");

}

</script>

</body>

</html>

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             // the number of elements in myCars  
var y=myCars.indexOf("Volvo")   // the index position of "Volvo"

## 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.ucase=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。

<!DOCTYPE html>

<html>

<body>

<p id="demo">Click the button to create an array, call the new ucase() method, and display the result.</p>

<button onclick="myFunction()">Try it</button>

<script>

Array.prototype.myUcase=function()

{

for (i=0;i<this.length;i++)

{

this[i]=this[i].toUpperCase();

}

}

function myFunction()

{

var fruits = ["Banana", "Orange", "Apple", "Mango"];

fruits.myUcase();

var x=document.getElementById("demo");

x.innerHTML=fruits;

}

</script>

</body>

</html>

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 Boolean（布尔） 对象

TBoolean（布尔）对象用于将非布尔值转换为布尔值（true 或者 false）。

Examples

## 在线实例

[检查布尔值](http://www.runoob.com/try/try.php?filename=tryjs_boolean)  
检查布尔对象是 true 还是 false。

<!DOCTYPE html>

<html>

<body>

<script>

var b1=new Boolean(0);

var b2=new Boolean(1);

var b3=new Boolean("");

var b4=new Boolean(null);

var b5=new Boolean(NaN);

var b6=new Boolean("false");

document.write("0 is boolean "+ b1 +"<br>");

document.write("1 is boolean "+ b2 +"<br>");

document.write("An empty string is boolean "+ b3 + "<br>");

document.write("null is boolean "+ b4+ "<br>");

document.write("NaN is boolean "+ b5 +"<br>");

document.write("The string 'false' is boolean "+ b6 +"<br>");

</script>

</body>

</html>

## 完整的 Boolean（布尔） 对象参考手册

我们提供 [JavaScript Boolean 对象参考手册](http://www.runoob.com/jsref/jsref-obj-boolean.html)，其中包括所有可用于布尔对象的属性和方法。

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。

## 创建 Boolean 对象

Boolean 对象代表两个值:"true" 或者 "false"

下面的代码定义了一个名为 myBoolean 的布尔对象：

var myBoolean=new Boolean();

如果布尔对象无初始值或者其值为:

* 0
* -0
* null
* ""
* false
* undefined
* NaN

那么对象的值为 false。否则，其值为 true（即使当自变量为字符串 "false" 时）！

# 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()。

<!DOCTYPE html>

<html>

<body>

<p id="demo">Click the button to round the number 2.5 to its nearest integer.</p>

<button onclick="myFunction()">Try it</button>

<script>

function myFunction()

{

document.getElementById("demo").innerHTML=Math.round(2.5);

}

</script>

</body>

</html>

[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。

<!DOCTYPE html>

<html>

<body>

<p id="demo">Click the button to display a random number.</p>

<button onclick="myFunction()">Try it</button>

<script>

function myFunction()

{

document.getElementById("demo").innerHTML=Math.random();

}

</script>

</body>

</html>

[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。）

<!DOCTYPE html>

<html>

<body>

<p id="demo">Click the button to return the lowest number of 5 and 10.</p>

<p id="demo1">Click the button to return the lowest number of 5 and 10.</p>

<button onclick="myFunction()">Try it</button>

<script>

function myFunction()

{

document.getElementById("demo").innerHTML=Math.min(5,10);

document.getElementById("demo1").innerHTML=Math.max(5,10);

}

</script>

</body>

</html>

## 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：

0.8665666756177051

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：

10

# 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"W3CSchool"

var str="Visit W3CSchool";  
var patt1=/w3cschool/i;

以下标记的文本是获得的匹配的表达式：

Visit W3CSchool

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

<!DOCTYPE html>

<html>

<body>

<script>

var str = "Visit W3CSchool";

var patt1 = /w3cschool/i;

document.write(str.match(patt1));

</script>

</body>

</html>

## 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 this all there is?

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

<!DOCTYPE html>

<html>

<body>

<script>

var str="Is this all there is?";

var patt1=/is/g;

document.write(str.match(patt1));

</script>

</body>

</html>

## 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 this all there is?

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

<!DOCTYPE html>

<html>

<body>

<script>

var str="Is this all there is?";

var patt1=/is/gi;

document.write(str.match(patt1));

</script>

</body>

</html>

## 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)

<!DOCTYPE html>

<html>

<body>

<script>

var patt1=new RegExp("e");

document.write(patt1.test("The best things in life are free"));

</script>

</body>

</html>

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)

<!DOCTYPE html>

<html>

<body>

<script>

var str = 'runoob';

var patt1 = new RegExp('\\w', 'g'); // 有转义作为正则表达式处理

var patt2 = new RegExp('\w', 'g'); // 无转义作为字符串处理

var patt3 =/\w+/g; // 与 patt1 效果相同

document.write(patt1.test(str)) //输出 true

document.write("<br>")

document.write(patt2.test(str)) //输出 false

document.write("<br>")

document.write(patt3.test(str)) //输出 true

</script>

</body>

</html>

## 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)

<!DOCTYPE html>

<html>

<body>

<script>

var patt1=new RegExp("e");

document.write(patt1.exec("The best things in life are free"));

</script>

</body>

</html>

# 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（浏览器的视口，不包括工具栏和滚动条）。

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

* window.innerHeight - 浏览器窗口的内部高度
* window.innerWidth - 浏览器窗口的内部宽度

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）

<!DOCTYPE html>

<html>

<body>

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

<script>

var w=window.innerWidth

|| document.documentElement.clientWidth

|| document.body.clientWidth;

var h=window.innerHeight

|| document.documentElement.clientHeight

|| document.body.clientHeight;

x=document.getElementById("demo");

x.innerHTML="Browser inner window width: " + w + ", height: " + h + "."

</script>

</body>

</html>

**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() - 调整当前窗口的尺寸

# 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("Available Width: " + screen.availWidth);  
  
</script>

以上代码输出为：

Available Width: 1366

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

<!DOCTYPE html>

<html>

<body>

<script>

document.write("Available Width: " + screen.availWidth);

</script>

</body>

</html>

## 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("Available Height: " + screen.availHeight);  
  
</script>

以上代码将输出：

Available Height: 728

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

<!DOCTYPE html>

<html>

<body>

<script>

document.write("Available Height: " + screen.availHeight);

</script>

</body>

</html>

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

<!DOCTYPE html>

<html>

<body>

<h3>Your Screen:</h3>

<script>

document.write("Total width/height: ");

document.write(screen.width + "\*" + screen.height);

document.write("<br>");

document.write("Available width/height: ");

document.write(screen.availWidth + "\*" + screen.availHeight);

document.write("<br>");

document.write("Color depth: ");

document.write(screen.colorDepth);

document.write("<br>");

document.write("Color resolution: ");

document.write(screen.pixelDepth);

</script>

</body>

</html>

# JavaScript Window Location

window.location 对象用于获得当前页面的地址 (URL)，并把浏览器重定向到新的页面。

## Window Location

**window.location** 对象在编写时可不使用 window 这个前缀。 一些例子：

一些实例:

* location.hostname 返回 web 主机的域名
* location.pathname 返回当前页面的路径和文件名
* location.port 返回 web 主机的端口 （80 或 443）
* location.protocol 返回所使用的 web 协议（http:// 或 https://）

## Window Location Href

location.href 属性返回当前页面的 URL。

## 实例

返回（当前页面的）整个 URL：

<script>  
  
document.write(location.href);  
  
</script>

以上代码输出为：

http://www.runoob.com/js/js-window-location.html

## Window Location Pathname

location.pathname 属性返回 URL 的路径名。

## 实例

返回当前 URL 的路径名：

<script>  
  
document.write(location.pathname);  
  
</script>

以上代码输出为：

/js/js-window-location.html

## Window Location Assign

location.assign() 方法加载新的文档。

## 实例

加载一个新的文档：

<html>  
<head>  
<script>  
function newDoc()  
  {  
  window.location.assign("http://www.w3cschool.cc")  
  }  
</script>  
</head>  
<body>  
  
<input type="button" value="Load new document" onclick="newDoc()">  
  
</body>  
</html>

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

<!DOCTYPE html>

<html>

<head>

<script>

function newDoc()

{

window.location.assign("http://www.w3cschool.cc")

}

</script>

</head>

<body>

<input type="button" value="加载新文档" onclick="newDoc()">

</body>

</html>

**JavaScript Window History**

window.history 对象包含浏览器的历史。

**Window History**

**window.history**对象在编写时可不使用 window 这个前缀。

为了保护用户隐私，对 JavaScript 访问该对象的方法做出了限制。

一些方法：

* history.back() - 与在浏览器点击后退按钮相同
* history.forward() - 与在浏览器中点击按钮向前相同

**Window History Back**

history.back() 方法加载历史列表中的前一个 URL。

这与在浏览器中点击后退按钮是相同的：

**实例**

在页面上创建后退按钮：

<html>  
<head>  
<script>  
function goBack()  
  {  
  window.history.back()  
  }  
</script>  
</head>  
<body>  
  
<input type="button" value="Back" onclick="goBack()">  
  
</body>  
</html>

以上代码输出为：

**Window History Forward**

history forward() 方法加载历史列表中的下一个 URL。

这与在浏览器中点击前进按钮是相同的：

**实例**

在页面上创建一个向前的按钮：

<html>  
<head>  
<script>  
function goForward()  
  {  
  window.history.forward()  
  }  
</script>  
</head>  
<body>  
  
<input type="button" value="Forward" onclick="goForward()">  
  
</body>  
</html>

以上代码输出为：

# JavaScript Window Navigator

window.navigator 对象包含有关访问者浏览器的信息。

## Window Navigator

**window.navigator** 对象在编写时可不使用 window 这个前缀。

## 实例

<div id="example"></div>  
  
<script>  
  
txt = "<p>Browser CodeName: " + navigator.appCodeName + "</p>";  
txt+= "<p>Browser Name: " + navigator.appName + "</p>";  
txt+= "<p>Browser Version: " + navigator.appVersion + "</p>";  
txt+= "<p>Cookies Enabled: " + navigator.cookieEnabled + "</p>";  
txt+= "<p>Platform: " + navigator.platform + "</p>";  
txt+= "<p>User-agent header: " + navigator.userAgent + "</p>";  
txt+= "<p>User-agent language: " + navigator.systemLanguage + "</p>";  
  
document.getElementById("example").innerHTML=txt;  
  
</script>

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

<!DOCTYPE html>

<html>

<body>

<div id="example"></div>

<script>

txt = "<p>Browser CodeName: " + navigator.appCodeName + "</p>";

txt+= "<p>Browser Name: " + navigator.appName + "</p>";

txt+= "<p>Browser Version: " + navigator.appVersion + "</p>";

txt+= "<p>Cookies Enabled: " + navigator.cookieEnabled + "</p>";

txt+= "<p>Platform: " + navigator.platform + "</p>";

txt+= "<p>User-agent header: " + navigator.userAgent + "</p>";

txt+= "<p>User-agent language: " + navigator.systemLanguage + "</p>";

document.getElementById("example").innerHTML=txt;

</script>

</body>

</html>

**警告!!!**

来自 navigator 对象的信息具有误导性，不应该被用于检测浏览器版本，这是因为：

* navigator 数据可被浏览器使用者更改
* 一些浏览器对测试站点会识别错误
* 浏览器无法报告晚于浏览器发布的新操作系统

**浏览器检测**

由于 navigator 可误导浏览器检测，使用对象检测可用来嗅探不同的浏览器。

由于不同的浏览器支持不同的对象，您可以使用对象来检测浏览器。例如，由于只有 Opera 支持属性 "window.opera"，您可以据此识别出 Opera。

例子：if (window.opera) {...some action...}

# 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)

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<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>

## 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)

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

</head>

<body>

<p>点击按钮，显示确认框。</p>

<button onclick="myFunction()">点我</button>

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

<script>

function myFunction()

{

var x;

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!";

}

document.getElementById("demo").innerHTML=x;

}

</script>

</body>

</html>

## 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)

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

</head>

<body>

<p>点击按钮查看输入的对话框。</p>

<button onclick="myFunction()">点我</button>

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

<script>

function myFunction()

{

var x;

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;

}

}

</script>

</body>

</html>

## 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)

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

</head>

<body>

<p>点击按钮在弹窗总使用换行。</p>

<button onclick="myFunction()">点我</button>

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

<script>

function myFunction()

{

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

}

</script>

</body>

</html>

**JavaScript 计时事件**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | JavaScript 一个设定的时间间隔之后来执行代码  我们称之为计时事件 |

**JavaScript 计时事件**

通过使用 JavaScript，我们有能力作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() - 暂停指定的毫秒数后执行指定的代码

**Note:** 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)

<!DOCTYPE html>

<html>

<body>

<p>Click the button to wait 3 seconds, then alert "Hello".</p>

<p>After clicking away the alert box, an new alert box will appear in 3 seconds. This goes on forever...</p>

<button onclick="myFunction()">Try it</button>

<script>

function myFunction()

{

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

}

</script>

</body>

</html>

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)

<!DOCTYPE html>

<html>

<body>

<p>A script on this page starts this clock:</p>

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

<script>

var myVar=setInterval(function(){myTimer()},1000);

function myTimer()

{

var d=new Date();

var t=d.toLocaleTimeString();

document.getElementById("demo").innerHTML=t;

}

</script>

</body>

</html>

## 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 "Stop time" 按钮：

<p id="demo"></p>  
<button onclick="myStopFunction()">Stop time</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)

<!DOCTYPE html>

<html>

<body>

<p>A script on this page starts this clock:</p>

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

<button onclick="myStopFunction()">Stop time</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>

</body>

</html>

## setTimeout() 方法

### 语法

window.setTimeout("*javascript 函数*",*毫秒数*);

setTimeout() 方法会返回某个值。在上面的语句中，值被储存在名为 t 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)

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<title>菜鸟教程(runoob.com)</title>

</head>

<body>

<p>点击按钮，在等待 3 秒后弹出 "Hello"。</p>

<button onclick="myFunction()">点我</button>

<script>

function myFunction()

{

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

}

</script>

</body>

</html>

## 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)

<!DOCTYPE html>

<html>

<body>

<p>Click the first button alert "Hello" after waiting 3 seconds.</p>

<p>Click the second button to prevent the first function to execute. (You must click it before the 3 seconds are up.)</p>

<button onclick="myFunction()">Try it</button>

<button onclick="myStopFunction()">Stop the alert</button>

<script>

var myVar;

function myFunction()

{

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

}

function myStopFunction()

{

clearTimeout(myVar);

}

</script>

</body>

</html>

Javascript 教程

[JavaScript 教程](http://www.runoob.com/js/js-tutorial.html)[JavaScript 简介](http://www.runoob.com/js/js-intro.html)[JavaScript 用法](http://www.runoob.com/js/js-howto.html)[JavaScript 输出](http://www.runoob.com/js/js-output.html)[JavaScript 语法](http://www.runoob.com/js/js-syntax.html)[JavaScript 语句](http://www.runoob.com/js/js-statements.html)[JavaScript 注释](http://www.runoob.com/js/js-comments.html)[JavaScript 变量](http://www.runoob.com/js/js-variables.html)[JavaScript 数据类型](http://www.runoob.com/js/js-datatypes.html)[JavaScript 对象](http://www.runoob.com/js/js-obj-intro.html)[JavaScript 函数](http://www.runoob.com/js/js-functions.html)[JavaScript 作用域](http://www.runoob.com/js/js-scope.html)[JavaScript 事件](http://www.runoob.com/js/js-events.html)[JavaScript 字符串](http://www.runoob.com/js/js-strings.html)[JavaScript 运算符](http://www.runoob.com/js/js-operators.html)[JavaScript 比较](http://www.runoob.com/js/js-comparisons.html)[JavaScript 条件语句](http://www.runoob.com/js/js-if-else.html)[JavaScript switch 语句](http://www.runoob.com/js/js-switch.html)[JavaScript for 循环](http://www.runoob.com/js/js-loop-for.html)[JavaScript while 循环](http://www.runoob.com/js/js-loop-while.html)[JavaScript Break 和 Continue 语句](http://www.runoob.com/js/js-break.html)[JavaScript typeof](http://www.runoob.com/js/js-typeof.html)[JavaScript 类型转换](http://www.runoob.com/js/js-type-conversion.html)[JavaScript 正则表达式](http://www.runoob.com/js/js-regexp.html)[JavaScript 错误](http://www.runoob.com/js/js-errors.html)[JavaScript 调试](http://www.runoob.com/js/js-debugging.html)[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)[JavaScript JSON](http://www.runoob.com/js/js-json.html)[JavaScript void](http://www.runoob.com/js/js-void.html)[JavaScript 代码规范](http://www.runoob.com/js/js-conventions.html)

## JS 函数

[JavaScript 函数定义](http://www.runoob.com/js/js-function-definition.html)[JavaScript 函数参数](http://www.runoob.com/js/js-function-parameters.html)[JavaScript 函数调用](http://www.runoob.com/js/js-function-invocation.html)[JavaScript 闭包](http://www.runoob.com/js/js-function-closures.html)

## JS HTML DOM

[DOM 简介](http://www.runoob.com/js/js-htmldom.html)[DOM HTML](http://www.runoob.com/js/js-htmldom-html.html)[DOM CSS](http://www.runoob.com/js/js-htmldom-css.html)[DOM 事件](http://www.runoob.com/js/js-htmldom-events.html)[DOM EventListener](http://www.runoob.com/js/js-htmldom-eventlistener.html)[DOM 元素](http://www.runoob.com/js/js-htmldom-elements.html)

## JS 高级教程

[JavaScript 对象](http://www.runoob.com/js/js-objects.html)[JavaScript Number 对象](http://www.runoob.com/js/js-obj-number.html)[JavaScript String](http://www.runoob.com/js/js-obj-string.html)[JavaScript Date（日期）](http://www.runoob.com/js/js-obj-date.html)[JavaScript Array（数组）](http://www.runoob.com/js/js-obj-array.html)[JavaScript Boolean（布尔）](http://www.runoob.com/js/js-obj-boolean.html)[JavaScript Math（算数）](http://www.runoob.com/js/js-obj-math.html)[JavaScript RegExp 对象](http://www.runoob.com/js/js-obj-regexp.html)

## JS 浏览器BOM

[JavaScript Window](http://www.runoob.com/js/js-window.html)[JavaScript Window Screen](http://www.runoob.com/js/js-window-screen.html)[JavaScript Window Location](http://www.runoob.com/js/js-window-location.html)[JavaScript Window History](http://www.runoob.com/js/js-window-history.html)[JavaScript Navigator](http://www.runoob.com/js/js-window-navigator.html)[JavaScript 弹窗](http://www.runoob.com/js/js-popup.html)[**JavaScript 计时事件**](http://www.runoob.com/js/js-timing.html)[JavaScript Cookies](http://www.runoob.com/js/js-cookies.html)

## JS 库

[JavaScript 库](http://www.runoob.com/js/js-libraries.html)[JavaScript 测试 jQuery](http://www.runoob.com/js/js-lib-jquery.html)[JavaScript 测试 Prototype](http://www.runoob.com/js/js-lib-prototype.html)

## JS 实例

[JavaScript 实例](http://www.runoob.com/js/js-examples.html)[JavaScript 对象实例](http://www.runoob.com/js/js-ex-objects.html)[JavaScript 浏览器对象实例](http://www.runoob.com/js/js-ex-browser.html)[JavaScript HTML DOM 实例](http://www.runoob.com/js/js-ex-dom.html)[JavaScript 总结](http://www.runoob.com/js/js-summary.html)

## JS 参考手册

[JavaScript 对象](http://www.runoob.com/jsref/jsref-tutorial.html)[HTML DOM 对象](http://www.runoob.com/jsref/dom-obj-document.html)

← [JavaScript 弹窗](http://www.runoob.com/js/js-popup.html)

[JavaScript Cookies](http://www.runoob.com/js/js-cookies.html) →

# JavaScript 计时事件

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | JavaScript 一个设定的时间间隔之后来执行代码  我们称之为计时事件 |

## JavaScript 计时事件

通过使用 JavaScript，我们有能力作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() - 暂停指定的毫秒数后执行指定的代码

**Note:** 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 "Stop time" 按钮：

<p id="demo"></p>  
<button onclick="myStopFunction()">Stop time</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

window.setTimeout("*javascript 函数*",*毫秒数*);

setTimeout() 方法会返回某个值。在上面的语句中，值被储存在名为 t 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)

<!DOCTYPE html>

<html>

<head>

<script>

function timedText()

{

var x=document.getElementById('txt');

var t1=setTimeout(function(){x.value="2 seconds"},2000);

var t2=setTimeout(function(){x.value="4 seconds"},4000);

var t3=setTimeout(function(){x.value="6 seconds"},6000);

}

</script>

</head>

<body>

<form>

<input type="button" value="Display timed text!" onclick="timedText()" />

<input type="text" id="txt" />

</form>

<p>Click on the button above. The input field will tell you when two, four, and six seconds have passed.</p>

</body>

</html>

# JavaScript Cookies

Cookies 用于存储 web 页面的用户信息。

## 什么是 Cookies？

Cookies 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。

Cookies 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。

Cookies 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 cookies 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 cookies。

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 cookies：

var x = document.cookie;

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

## 使用 JavaScript 修改 Cookie

在 JavaScript 中，修改 cookies 类似于创建 cookies，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

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("Welcome again " + user);  
  }  
else   
  {  
  user = prompt("Please enter your name:","");  
  if (user!="" && user!=null)  
    {  
    setCookie("username",user,365);  
    }  
  }  
}

[**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。

<!DOCTYPE html>

<html>

<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("Welcome again " + user);

}

else

{

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

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

{

setCookie("username",user,30);

}

}

}

</script>

</head>

<body onload="checkCookie()">

</body>

</html>

# JavaScript 库

JavaScript 库 - jQuery、Prototype、MooTools。

## JavaScript 框架（库）

JavaScript 高级程序设计（特别是对浏览器差异的复杂处理），通常很困难也很耗时。

为了应对这些调整，许多的 **JavaScript (helper)** 库应运而生。

这些 JavaScript 库常被称为 **JavaScript 框架**。

在本教程中，我们将了解到一些广受欢迎的 JavaScript 框架：

* jQuery
* Prototype
* MooTools

所有这些框架都提供针对常见 JavaScript 任务的函数，包括动画、DOM 操作以及 Ajax 处理。

在本教程中，您将学习到如何开始使用它们，来使得 JavaScript 编程更容易、更安全且更有乐趣。

## jQuery

jQuery 是目前最受欢迎的 JavaScript 框架。

它使用 CSS 选择器来访问和操作网页上的 HTML 元素（DOM 对象）。

jQuery 同时提供 companion UI（用户界面）和插件。

许多大公司在网站上使用 jQuery：

* Google
* Microsoft
* IBM
* Netflix

如需更深入地学习 jQuery，请访问我们的 [jQuery 教程](http://www.runoob.com/jquery/jquery-tutorial.html)。

## Prototype

**Prototype** 是一种库，提供用于执行常见 web 任务的简单 API。

**API** 是应用程序编程接口（Application Programming Interface）的缩写。它是包含属性和方法的库，用于操作 HTML DOM。

Prototype 通过提供类和继承，实现了对 JavaScript 的增强。

## MooTools

**MooTools** 也是一个框架，提供了可使常见的 JavaScript 编程更为简单的 API。

MooTools 也含有一些轻量级的效果和动画函数。

## 其他框架

下面是其他一些在上面未涉及的框架：

**YUI** - Yahoo! User Interface Framework，涵盖大量函数的大型库，从简单的 JavaScript 功能到完整的 internet widget。

**Ext JS** - 可定制的 widget，用于构建富因特网应用程序（rich Internet applications）。

**Dojo** - 用于 DOM 操作、事件、widget 等的工具包。

**script.aculo.us** - 开源的 JavaScript 框架，针对可视效果和界面行为。

**UIZE** - Widget、AJAX、DOM、模板等等。

## CDN -内容分发网络

您总是希望网页可以尽可能地快。您希望页面的容量尽可能地小，同时您希望浏览器尽可能多地进行缓存。

如果许多不同的网站使用相同的 JavaScript 框架，那么把框架库存放在一个通用的位置供每个网页分享就变得很有意义了。

CDN (Content Delivery Network) 解决了这个问题。CDN 是包含可分享代码库的服务器网络。

Google 为一系列 JavaScript 库提供了免费的 CDN，包括：

* jQuery
* Prototype
* MooTools
* Dojo
* Yahoo! YUI

但是由于 Google 在中国经常被GFW（防火长城，英文名称Great Firewall of China，简写为Great Firewall，缩写GFW）屏蔽，造成访问不稳定，所以建议使用百度静态资源公共库:<http://cdn.code.baidu.com/>。

如需在您的网页中使用 JavaScript 框架库，只需在 <script> 标签中引用该库即可：

## 引用 jQuery

<script src="http://apps.bdimg.com/libs/jquery/2.1.1/jquery.min.js">  
</script>

## 使用框架

在您决定为网页使用 JavaScript 框架之前，首先对框架进行测试是明智的。

JavaScript 框架很容易进行测试。您无需在计算机上安装它们，同时也没有安装程序。

通常您只需从网页中引用一个库文件。

在本教程的下一章，我们会为您完整地讲解 jQuery 的测试过程。

# JavaScript - 测试 jQuery

测试 JavaScript 框架库 - jQuery

## 引用 jQuery

如需测试 JavaScript 库，您需要在网页中引用它。

为了引用某个库，请使用 <script> 标签，其 src 属性设置为库的 URL：

## 引用 jQuery

<!DOCTYPE html>  
<html>  
<head>  
<script src="http://apps.bdimg.com/libs/jquery/2.1.1/jquery.min.js">  
</script>  
</head>  
<body>  
</body>  
</html>

## jQuery 描述

主要的 jQuery 函数是 $() 函数（jQuery 函数）。如果您向该函数传递 DOM 对象，它会返回 jQuery 对象，带有向其添加的 jQuery 功能。

jQuery 允许您通过 CSS 选择器来选取元素。

在 JavaScript 中，您可以分配一个函数以处理窗口加载事件：

## JavaScript 方式：

function myFunction()  
{  
var obj=document.getElementById("h01");  
obj.innerHTML="Hello jQuery";  
}  
onload=myFunction;

等价的 jQuery 是不同的：

## jQuery 方式：

function myFunction()  
{  
$("#h01").html("Hello jQuery");  
}  
$(document).ready(myFunction);

上面代码的最后一行，HTML DOM 文档对象被传递到 jQuery ：$(document)。

当您向 jQuery 传递 DOM 对象时，jQuery 会返回以 HTML DOM 对象包装的 jQuery 对象。

jQuery 函数会返回新的 jQuery 对象，其中的 ready() 是一个方法。

由于在 JavaScript 中函数就是变量，因此可以把 myFunction 作为变量传递给 jQuery 的 ready 方法。

|  |  |
| --- | --- |
| **lamp** | jQuery 返回 jQuery 对象，与已传递的 DOM 对象不同。 jQuery 对象拥有的属性和方法，与 DOM 对象的不同。 您不能在 jQuery 对象上使用 HTML DOM 的属性和方法。 |

## 测试 jQuery

请试一下下面这个例子：

## 实例

<!DOCTYPE html>  
<html>  
<head>  
<script src="http://apps.bdimg.com/libs/jquery/2.1.1/jquery.min.js">  
</script>  
<script>  
function myFunction()  
{  
$("#h01").html("Hello jQuery")  
}  
$(document).ready(myFunction);  
</script>  
</head>  
<body>  
<h1 id="h01"></h1>  
</body>  
</html>

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

<!DOCTYPE html>

<html>

<head>

<script

src="http://libs.baidu.com/jquery/1.8.3/jquery.min.js">

</script>

<script>

function myFunction()

{

$("#h01").html("Hello jQuery")

}

$(document).ready(myFunction);

</script>

</head>

<body>

<h1 id="h01"></h1>

</body>

</html>

请再试一下这个例子：

## 实例

<!DOCTYPE html>  
<html>  
<head>  
<script src="http://apps.bdimg.com/libs/jquery/2.1.1/jquery.min.js">  
</script>  
<script>  
function myFunction()  
{  
$("#h01").attr("style","color:red").html("Hello jQuery")  
}  
$(document).ready(myFunction);  
</script>  
</head>  
<body>  
<h1 id="h01"></h1>  
</body>  
</html>

[**尝试一下 »**](http://www.runoob.com/try/try.php?filename=tryjs_lib_jquery2)

<!DOCTYPE html>

<html>

<head>

<script

src="http://libs.baidu.com/jquery/1.8.3/jquery.min.js">

</script>

<script>

function myFunction()

{

$("#h01").attr("style","color:red").html("Hello jQuery")

}

$(document).ready(myFunction);

</script>

</head>

<body>

<h1 id="h01"></h1>

</body>

</html>

正如您在上面的例子中看到的，jQuery 允许链接（链式语法）。

链接（Chaining）是一种在同一对象上执行多个任务的便捷方法。

需要学习更多内容吗？W3School 为您提供了非常棒的 [jQuery 教程](http://www.runoob.com/jquery/)。

# JavaScript - 测试 Prototype

测试 JavaScript 框架库 - Prototype

## 引用 Prototype

如需测试 JavaScript 库，您需要在网页中引用它。

为了引用某个库，请使用 <script> 标签，其 src 属性设置为库的 URL：

## 引用 Prototype

<!DOCTYPE html>  
<html>  
<head>  
<script  
src="http://apps.bdimg.com/libs/prototype/1.7.1.0/prototype.js">  
</script>  
</head>  
<body>  
</body>  
</html>

## Prototype 描述

Prototype 提供的函数可使 HTML DOM 编程更容易。

与 jQuery 类似，Prototype 也有自己的 $() 函数。

$() 函数接受 HTML DOM 元素的 id 值（或 DOM 元素），并会向 DOM 对象添加新的功能。

与 jQuery 不同，Prototype 没有用以取代 window.onload() 的 ready() 方法。相反，Prototype 会向浏览器及 HTML DOM 添加扩展。

在 JavaScript 中，您可以分配一个函数以处理窗口加载事件：

## JavaScript 方式：

function myFunction()  
{  
var obj=document.getElementById("h01");  
obj.innerHTML="Hello Prototype";  
}  
onload=myFunction;

等价的 Prototype 是不同的：

## Prototype 方式：

function myFunction()  
{  
$("h01").insert("Hello Prototype!");  
}  
Event.observe(window,"load",myFunction);

Event.observe() 接受三个参数：

* 您希望处理的 HTML DOM 或 BOM（浏览器对象模型）对象
* 您希望处理的事件
* 您希望调用的函数

## 测试 Prototype

请试一下下面这个例子：

## Example

<!DOCTYPE html>  
<html>  
<script  
src="http://apps.bdimg.com/libs/prototype/1.7.1.0/prototype.js">  
</script>  
<script>  
function myFunction()  
{  
$("h01").insert("Hello Prototype!");  
}  
Event.observe(window,"load",myFunction);  
</script>  
</head>  
<body>  
<h1 id="h01"></h1>  
</body>  
</html>

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

<!DOCTYPE html>

<html>

<script

src="http://apps.bdimg.com/libs/prototype/1.7.1.0/prototype.js">

</script>

<script>

function myFunction()

{

$("h01").insert("Hello Prototype!");

}

Event.observe(window,"load",myFunction);

</script>

</head>

<body>

<h1 id="h01"></h1>

</body>

</html>

请再试一下这个例子：

## Example

<!DOCTYPE html>  
<html>  
<script  
src="http://apps.bdimg.com/libs/prototype/1.7.1.0/prototype.js">  
</script>  
<script>  
function myFunction()  
{  
$("h01").writeAttribute("style","color:red").insert("Hello Prototype!");  
}  
Event.observe(window,"load",myFunction);  
</script>  
</head>  
<body>  
<h1 id="h01"></h1>  
</body>  
</html>

[**测试一下 »**](http://www.runoob.com/try/try.php?filename=tryjs_lib_prototype2)

<!DOCTYPE html>

<html>

<script

src="http://apps.bdimg.com/libs/prototype/1.7.1.0/prototype.js">

</script>

<script>

function myFunction()

{

$("h01").writeAttribute("style","color:red").insert("Hello Prototype!");

}

Event.observe(window,"load",myFunction);

</script>

</head>

<body>

<h1 id="h01"></h1>

</body>

</html>

正如您在上面的例子中看到的，与 jQuery 相同，Prototype 允许链式语法。

链接（Chaining）是一种在同一对象上执行多个任务的便捷方法。

# 现在您已经学习了 JavaScript， 接下来该学习什么呢？

## JavaScript 总结

本教程中我们向您讲授了如何向 html 页面添加 JavaScript，使得网站的动态性和交互性更强。

你已经学习了如何创建对事件的响应，验证表单，以及如何根据不同的情况运行不同的脚本。

你也学到了如何创建和使用对象，以及如何使用 JavaScript 的内置对象。

如需更多关于 JavaScript 的信息和知识，请参阅我们的 [JavaScript 实例](http://www.runoob.com/js/js-examples.html) 和 [JavaScript 参考手册](http://www.runoob.com/jsref/)。

## 现在已经你已经学习了 JavaScript，接下来该学习什么呢？

下一步应该学习 HTML DOM 和 DHTML。

如果你希望学习关于服务器端脚本的知识，那么下一步应该学习 ASP,PHP, .Net。

**HTML DOM**

HTML DOM 定义了访问和操作 HTML 文档的标准方法。 HTML DOM 独立于平台和语言，可被任何编程语言使用，比如 Java、JavaScript 和 VBscript。 假如希望了解更多关于 DOM 的知识，请访问我们的 [HTML DOM 教程](http://www.runoob.com/htmldom/)。

**jQuery**

jQuery 是一个 JavaScript 库。

jQuery 极大地简化了 JavaScript 编程。

jQuery 很容易学习。

假如希望了解更多关于 jQuery 的知识，请访问我们的 [jQuery 教程](http://www.runoob.com/jquery/)。

**AJAX**

AJAX = 异步 JavaScript 和 XML。

AJAX 不是一种新的编程语言，而是一种使用现有标准的新方法。

通过与服务器进行数据交换，AJAX 可以在不重新加载整个网页的情况下，对网页的某部分进行更新。

有很多使用 AJAX 的应用程序案例：新浪微博、Google 地图、开心网等等。

假如您希望学习更多关于 AJAX 的知识，请访问我们的 [AJAX 教程](http://www.runoob.com/ajax/)。

**ASP / PHP / .NET**

和 HTML 文档中的脚本运行于客户端（浏览器）不同，ASP/PHP 文件中的脚本在服务器上运行。

使用 ASP，你可以动态地编辑、改变或者添加网站内容，对由 HTML 表单提交而来的数据进行响应，访问数据或者数据库并向浏览器返回结果，或者定制对不同的用户来说更有帮助的网页。

由于 ASP/PHP 文件返回的是纯粹的 HTML，因此可显示在任何浏览器中。

如果希望学习更多关于 ASP 的知识，请访问我们的 [ASP 教程](http://www.runoob.com/asp/)。

如果希望学习更多关于 PHP 的知识，请访问我们的 [PHP 教程](http://www.runoob.com/php/)。

如果希望学习更多关于 .NET 的知识，请访问我们的[.NET 实例/教程](http://www.runoob.com/aspnet/)

# JavaScript 和 HTML DOM 参考手册

<="" p="" style="color: rgb(51, 51, 51); font-family: 'Open Sans', 'Helvetica Neue', Helvetica, Arial, STHeiti, 'Microsoft Yahei', sans-serif; font-size: 12px; font-style: normal; font-variant: normal; font-weight: normal; letter-spacing: normal; line-height: normal; orphans: auto; text-align: start; text-indent: 0px; text-transform: none; white-space: normal; widows: 1; word-spacing: 0px; -webkit-text-stroke-width: 0px; background-color: rgb(255, 255, 255);">

所有内置的JavaScript对象

所有浏览器对象

所有HTML DOM对象

## JavaScript Objects 参考手册

参考手册描述了每个对象的属性和方法，并提供了在线实例。

* [Array 对象](http://www.runoob.com/jsref/jsref-obj-array.html)
* [Boolean 对象](http://www.runoob.com/jsref/jsref-obj-boolean.html)
* [Date 对象](http://www.runoob.com/jsref/jsref-obj-date.html)
* [Math 对象](http://www.runoob.com/jsref/jsref-obj-math.html)
* [Number 对象](http://www.runoob.com/jsref/jsref-obj-number.html)
* [String 对象](http://www.runoob.com/jsref/jsref-obj-string.html)
* [RegExp 对象](http://www.runoob.com/jsref/jsref-obj-regexp.html)
* [全局属性和函数](http://www.runoob.com/jsref/jsref-obj-global.html)

## Browser Objects 参考手册

参考手册描述了每个对象的属性和方法，并提供了在线实例。

* [Window 对象](http://www.runoob.com/jsref/obj-window.html)
* [Navigator 对象](http://www.runoob.com/jsref/obj-navigator.html)
* [Screen 对象](http://www.runoob.com/jsref/obj-screen.html)
* [History 对象](http://www.runoob.com/jsref/obj-history.html)
* [Location 对象](http://www.runoob.com/jsref/obj-location.html)

## HTML DOM 参考手册

参考手册描述了 HTML DOM 的属性和方法，并提供在线实例。

* [HTML Document](http://www.runoob.com/jsref/dom-obj-document.html)
* [HTML Element](http://www.runoob.com/jsref/dom-obj-all.html)
* [HTML Attributes](http://www.runoob.com/jsref/dom-obj-attributes.html)
* [HTML Events](http://www.runoob.com/jsref/dom-obj-event.html)

## HTML DOM Element Objects 参考手册

参考手册描述了每个对象的属性和方法，并提供了在线实例。

* [Anchor 对象](http://www.runoob.com/jsref/dom-obj-anchor.html)
* [Area 对象](http://www.runoob.com/jsref/dom-obj-area.html)
* [Base 对象](http://www.runoob.com/jsref/dom-obj-base.html)
* [Body 对象](http://www.runoob.com/jsref/dom-obj-body.html)
* [Button 对象](http://www.runoob.com/jsref/dom-obj-pushbutton.html)
* [Form 对象](http://www.runoob.com/jsref/dom-obj-form.html)
* [Frame/IFrame 对象](http://www.runoob.com/jsref/dom-obj-frame.html)
* [Frameset 对象](http://www.runoob.com/jsref/dom-obj-frameset.html)
* [Image 对象](http://www.runoob.com/jsref/dom-obj-image.html)
* [Input Button 对象](http://www.runoob.com/jsref/dom-obj-button.html)
* [Input Checkbox 对象](http://www.runoob.com/jsref/dom-obj-checkbox.html)
* [Input File 对象](http://www.runoob.com/jsref/dom-obj-fileupload.html)
* [Input Hidden 对象](http://www.runoob.com/jsref/dom-obj-hidden.html)
* [Input Password 对象](http://www.runoob.com/jsref/dom-obj-password.html)
* [Input Radio 对象](http://www.runoob.com/jsref/dom-obj-radio.html)
* [Input Reset 对象](http://www.runoob.com/jsref/dom-obj-reset.html)
* [Input Submit 对象](http://www.runoob.com/jsref/dom-obj-submit.html)
* [Input Text 对象](http://www.runoob.com/jsref/dom-obj-text.html)
* [Link 对象](http://www.runoob.com/jsref/dom-obj-link.html)
* [Meta 对象](http://www.runoob.com/jsref/dom-obj-meta.html)
* [Object 对象](http://www.runoob.com/jsref/dom-obj-object.html)
* [Option 对象](http://www.runoob.com/jsref/dom-obj-option.html)
* [Select 对象](http://www.runoob.com/jsref/dom-obj-select.html)
* [Style 对象](http://www.runoob.com/jsref/dom-obj-style.html)
* [Table 对象](http://www.runoob.com/jsref/dom-obj-table.html)
* [td / th 对象](http://www.runoob.com/jsref/dom-obj-tabledata.html)
* [tr 对象](http://www.runoob.com/jsref/dom-obj-tablerow.html)
* [Textarea 对象](http://www.runoob.com/jsref/dom-obj-textarea.html)

# 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参考手册请参考 [JavaScript Array 对象手册](http://www.runoob.com/js/js-obj-array.html)。

## 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。 |
| [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。 |
| [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。 |
| [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。 |
| [join()](http://www.runoob.com/jsref/jsref-join.html) | 把数组的所有元素放入一个字符串。 |
| [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。 |
| [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 Boolean 对象

## Boolean 对象

Boolean 对象用于转换一个不是 Boolean 类型的值转换为 Boolean 类型值 (true 或者false).

Boolean 对象完整教程, 请访问 [JavaScript Boolean 对象教程](http://www.runoob.com/js/js-obj-boolean.html)。

## Boolean 对象属性

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

## Boolean 对象方法

|  |  |
| --- | --- |
| **方法** | **描述** |
| [toString()](http://www.runoob.com/jsref/jsref-tostring-boolean.html) | 把布尔值转换为字符串，并返回结果。 |
| [valueOf()](http://www.runoob.com/jsref/jsref-valueof-boolean.html) | 返回 Boolean 对象的原始值。 |

# JavaScript Date 对象

## Date 对象

Date 对象用于处理日期与实际。

创建 Date 对象： new Date().

以上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 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 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 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。 |
| [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) | 使用小字号来显示字符串。 |
| [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 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 NUL 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。 |
| [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。 |
| [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。 |
| [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。 |
| [n{X,Y}](http://www.runoob.com/jsref/jsref-regexp-nxy.html) | 匹配包含 X 或 Y 个 n 的序列的字符串。 |
| [n{X,}](http://www.runoob.com/jsref/jsref-regexp-nxcomma.html) | 匹配包含至少 X 个 n 的序列的字符串。 |
| [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** | **FF** | **IE** |
| [compile](http://www.runoob.com/jsref/jsref-regexp-compile.html) | 编译正则表达式。 | 1 | 4 |
| [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。 | 1 | 4 |
| [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。 | 1 | 4 |

## 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 |

# 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 运算符

JavaScript 运算符用于赋值，比较值，执行算术运算等。

## JavaScript 算术运算符

算术运算符用于执行两个变量或值的运算。

赋值 **y = 5**, 以下表格将向你说明算术运算符的使用：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **运算符** | **描述** | **例子** | **y 值** | **x 值** | **在线实例** |
| + | 加法 | x = y + 2 | y = 5 | x = 7 | [**实例 »**](http://www.runoob.com/try/try.php?filename=tryjsref_oper_add) |
| - | 减法 | x = y - 2 | y = 5 | x = 3 | [**实例 »**](http://www.runoob.com/try/try.php?filename=tryjsref_oper_sub) |
| \* | 乘法 | x = y \* 2 | y = 5 | x = 10 | [**实例 »**](http://www.runoob.com/try/try.php?filename=tryjsref_oper_mult) |
| / | 除法 | x = y / 2 | y = 5 | x = 2.5 | [**实例 »**](http://www.runoob.com/try/try.php?filename=tryjsref_oper_div) |
| % | 余数 | x = y % 2 | y = 5 | x = 1 | [**实例 »**](http://www.runoob.com/try/try.php?filename=tryjsref_oper_mod) |
| ++ | 自增 | x = ++y | y = 6 | x = 6 | [**实例 »**](http://www.runoob.com/try/try.php?filename=tryjsref_oper_incr) |
| x = y++ | y = 6 | x = 5 | [**实例 »**](http://www.runoob.com/try/try.php?filename=tryjsref_oper_incr2) |
| -- | 自减 | x = --y | y = 4 | x = 4 | [**实例 »**](http://www.runoob.com/try/try.php?filename=tryjsref_oper_decr) |
| x = y-- | y = 4 | x = 5 | [**实例 »**](http://www.runoob.com/try/try.php?filename=tryjsref_oper_decr2) |

关于算术运算符，你可以阅读我们的 [JavaScript 运算符教程](http://www.runoob.com/js/js-operators.html)。

## JavaScript 赋值运算符

赋值运算符用于给 JavaScript 变量赋值。

给定 **x=10**和**y=5**，下面的表格解释了赋值运算符：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **运算符** | **例子** | **Same As** | **x 值** | **在线实例** |
| = | x = y | x = y | x = 5 | [**实例 »**](http://www.runoob.com/try/try.php?filename=tryjsref_oper_equal) |
| += | x += y | x = x + y | x = 15 | [**实例 »**](http://www.runoob.com/try/try.php?filename=tryjsref_oper_plusequal) |
| -= | x -= y | x = x - y | x = 5 | [**实例 »**](http://www.runoob.com/try/try.php?filename=tryjsref_oper_minequal) |
| \*= | x \*= y | x = x \* y | x = 50 | [**实例 »**](http://www.runoob.com/try/try.php?filename=tryjsref_oper_multequal) |
| /= | x /= y | x = x / y | x = 2 | [**实例 »**](http://www.runoob.com/try/try.php?filename=tryjsref_oper_divequal) |
| %= | x %= y | x = x % y | x = 0 | [**实例 »**](http://www.runoob.com/try/try.php?filename=tryjsref_oper_modequal) |

关于赋值运算符，你可以阅读我们的 [JavaScript 运算符教程](http://www.runoob.com/js/js-operators.html)。

## JavaScript 字符串运算符

+ 运算符， += 运算符可用于连接字符串。

给定 **text1 = "Good "**, **text2 = "Morning"**, **及 text3 = ""**, 下面的表格解释了字符串运算符的使用：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **运算符** | **例子** | **text1** | **text2** | **text3** | **在线实例** |
| + | text3 = text1 + text2 | "Good " | "Morning" | "Good Morning" | [**实例 »**](http://www.runoob.com/try/try.php?filename=tryjsref_oper_string1) |
| += | text1 += text2 | "Good Morning" | "Morning" | "" | [**实例 »**](http://www.runoob.com/try/try.php?filename=tryjsref_oper_string2) |

## 比较运算符

比较运算符用于逻辑语句的判断，从而确定给定的两个值或变量是否相等。

给定 **x=5**, 下表展示了比较运算符的使用：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **运算符** | **描述** | **比较** | **结果** | **在线实例** |
| == | 等于 | x == 8 | false | [**实例 »**](http://www.runoob.com/try/try.php?filename=tryjsref_comparison1) |
| x == 5 | true | [**实例 »**](http://www.runoob.com/try/try.php?filename=tryjsref_comparison2) |
| === | 值及类型均相等（恒等于） | x === "5" | false | [**实例 »**](http://www.runoob.com/try/try.php?filename=tryjsref_comparison3) |
| x === 5 | true | [**实例 »**](http://www.runoob.com/try/try.php?filename=tryjsref_comparison4) |
| != | 不等于 | x != 8 | true | [**实例 »**](http://www.runoob.com/try/try.php?filename=tryjsref_comparison5) |
| !== | 值与类型均不等（不恒等于） | x !== "5" | true | [**实例 »**](http://www.runoob.com/try/try.php?filename=tryjsref_comparison6) |
| x !== 5 | false | [**实例 »**](http://www.runoob.com/try/try.php?filename=tryjsref_comparison7) |
| > | 大于 | x > 8 | false | [**实例 »**](http://www.runoob.com/try/try.php?filename=tryjsref_comparison8) |
| < | 小于 | x < 8 | true | [**实例 »**](http://www.runoob.com/try/try.php?filename=tryjsref_comparison9) |
| >= | 大于或等于 | x >= 8 | false | [**实例 »**](http://www.runoob.com/try/try.php?filename=tryjsref_comparison10) |
| <= | 小于或等于 | x <= 8 | true | [**实例 »**](http://www.runoob.com/try/try.php?filename=tryjsref_comparison11) |

关于比较运算符，你可以阅读我们的 [JavaScript 比较运算符教程](http://www.runoob.com/js/js-comparisons.html)。

## 条件运算符

条件运算符用于基于条件的赋值运算。

给定 **x=6 and y=3**, 下表演示了条件运算符的运算：

|  |  |  |
| --- | --- | --- |
| **语法** | **例子** | **在线实例** |
| 变量= (条件) ?值1:值2 | voteable = (age & 18) ? "Too young" : "Old enough" | [**实例 »**](http://www.runoob.com/try/try.php?filename=tryjsref_comparison) |

## 逻辑运算符

逻辑运算符用来确定变量或值之间的逻辑关系。

给定 **x=6 and y=3**, 以下实例演示了逻辑运算符的使用：

|  |  |  |
| --- | --- | --- |
| **运算符** | **描述** | **例子** |
| && | 和 | (x < 10 && y > 1) 为 true |
| || | 或 | (x == 5 || y == 5) 为 false |
| ! | 非 | !(x == y) 为 true |

## JavaScript 位运算符

位运算符工作于32位的数字上。任何数字操作都将转换为32位。结果会转换为 JavaScript 数字。

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **运算符** | **描述** | **例子** | **类似于** | **结果** | **十进制** |
| & | AND | x = 5 & 1 | 0101 & 0001 | 0001 | 1 |
| | | OR | x = 5 | 1 | 0101 | 0001 | 0101 | 5 |
| ~ | 取反 | x = ~ 5 | ~0101 | 1010 | 10 |
| ^ | 异或 | x = 5 ^ 1 | 0101 ^ 0001 | 0100 | 4 |
| << | 左移/td> | x = 5 << 1 | 0101 << 1 | 1010 | 10 |
| >> | 右移 | x = 5 >> 1 | 0101 >> 1 | 0010 | 2 |

# Window 对象

## Window 对象

Window 对象表示浏览器中打开的窗口。

如果文档包含框架（<frame> 或 <iframe> 标签），浏览器会为 HTML 文档创建一个 window 对象，并为每个框架创建一个额外的 window 对象。

Note**注意：** 没有应用于 window 对象的公开标准，不过所有浏览器都支持该对象。

## Window 对象属性

|  |  |
| --- | --- |
| **属性** | **描述** |
| [closed](http://www.runoob.com/jsref/prop-win-closed.html) | 返回窗口是否已被关闭。 |
| [defaultStatus](http://www.runoob.com/jsref/prop-win-defaultstatus.html) | 设置或返回窗口状态栏中的默认文本。 |
| [document](http://www.runoob.com/jsref/dom-obj-document.html) | 对 Document 对象的只读引用。(请参阅[对象](http://www.runoob.com/jsref/dom-obj-document.html)) |
| [frames](http://www.runoob.com/jsref/prop-win-frames.html) | 返回窗口中所有命名的框架。该集合是 Window 对象的数组，每个 Window 对象在窗口中含有一个框架。 |
| [history](http://www.runoob.com/jsref/obj-history.html) | 对 History 对象的只读引用。请参数 [History 对象](http://www.runoob.com/jsref/obj-history.html)。 |
| [innerHeight](http://www.runoob.com/jsref/prop-win-innerheight.html) | 返回窗口的文档显示区的高度。 |
| [innerWidth](http://www.runoob.com/jsref/prop-win-innerheight.html) | 返回窗口的文档显示区的宽度。 |
| [length](http://www.runoob.com/jsref/prop-win-length.html) | 设置或返回窗口中的框架数量。 |
| [location](http://www.runoob.com/jsref/obj-location.html) | 用于窗口或框架的 Location 对象。请参阅 [Location 对象](http://www.runoob.com/jsref/obj-location.html)。 |
| [name](http://www.runoob.com/jsref/prop-win-name.html) | 设置或返回窗口的名称。 |
| [navigator](http://www.runoob.com/jsref/obj-navigator.html) | 对 Navigator 对象的只读引用。请参数 [Navigator 对象](http://www.runoob.com/jsref/obj-navigator.html)。 |
| [opener](http://www.runoob.com/jsref/prop-win-opener.html) | 返回对创建此窗口的窗口的引用。 |
| [outerHeight](http://www.runoob.com/jsref/prop-win-outerheight.html) | 返回窗口的外部高度，包含工具条与滚动条。 |
| [outerWidth](http://www.runoob.com/jsref/prop-win-outerheight.html) | 返回窗口的外部宽度，包含工具条与滚动条。 |
| [pageXOffset](http://www.runoob.com/jsref/prop-win-pagexoffset.html) | 设置或返回当前页面相对于窗口显示区左上角的 X 位置。 |
| [pageYOffset](http://www.runoob.com/jsref/prop-win-pagexoffset.html) | 设置或返回当前页面相对于窗口显示区左上角的 Y 位置。 |
| [parent](http://www.runoob.com/jsref/prop-win-parent.html) | 返回父窗口。 |
| [screen](http://www.runoob.com/jsref/obj-screen.html) | 对 Screen 对象的只读引用。请参数 [Screen 对象](http://www.runoob.com/jsref/obj-screen.html)。 |
| [screenLeft](http://www.runoob.com/jsref/prop-win-screenleft.html) | 返回相对于屏幕窗口的x坐标 |
| [screenTop](http://www.runoob.com/jsref/prop-win-screenleft.html) | 返回相对于屏幕窗口的y坐标 |
| [screenX](http://www.runoob.com/jsref/prop-win-screenx.html) | 返回相对于屏幕窗口的x坐标 |
| [screenY](http://www.runoob.com/jsref/prop-win-screenx.html) | 返回相对于屏幕窗口的y坐标 |
| [self](http://www.runoob.com/jsref/prop-win-self.html) | 返回对当前窗口的引用。等价于 Window 属性。 |
| [status](http://www.runoob.com/jsref/prop-win-status.html) | 设置窗口状态栏的文本。 |
| [top](http://www.runoob.com/jsref/prop-win-top.html) | 返回最顶层的父窗口。 |

## Window 对象方法

|  |  |
| --- | --- |
| **方法** | **描述** |
| [alert()](http://www.runoob.com/jsref/met-win-alert.html) | 显示带有一段消息和一个确认按钮的警告框。 |
| [blur()](http://www.runoob.com/jsref/met-win-blur.html) | 把键盘焦点从顶层窗口移开。 |
| [clearInterval()](http://www.runoob.com/jsref/met-win-clearinterval.html) | 取消由 setInterval() 设置的 timeout。 |
| [clearTimeout()](http://www.runoob.com/jsref/met-win-cleartimeout.html) | 取消由 setTimeout() 方法设置的 timeout。 |
| [close()](http://www.runoob.com/jsref/met-win-close.html) | 关闭浏览器窗口。 |
| [confirm()](http://www.runoob.com/jsref/met-win-confirm.html) | 显示带有一段消息以及确认按钮和取消按钮的对话框。 |
| [createPopup()](http://www.runoob.com/jsref/met-win-createpopup.html) | 创建一个 pop-up 窗口。 |
| [focus()](http://www.runoob.com/jsref/met-win-focus.html) | 把键盘焦点给予一个窗口。 |
| [moveBy()](http://www.runoob.com/jsref/met-win-moveby.html) | 可相对窗口的当前坐标把它移动指定的像素。 |
| [moveTo()](http://www.runoob.com/jsref/met-win-moveto.html) | 把窗口的左上角移动到一个指定的坐标。 |
| [open()](http://www.runoob.com/jsref/met-win-open.html) | 打开一个新的浏览器窗口或查找一个已命名的窗口。 |
| [print()](http://www.runoob.com/jsref/met-win-print.html) | 打印当前窗口的内容。 |
| [prompt()](http://www.runoob.com/jsref/met-win-prompt.html) | 显示可提示用户输入的对话框。 |
| [resizeBy()](http://www.runoob.com/jsref/met-win-resizeby.html) | 按照指定的像素调整窗口的大小。 |
| [resizeTo()](http://www.runoob.com/jsref/met-win-resizeto.html) | 把窗口的大小调整到指定的宽度和高度。 |
| scroll() |  |
| [scrollBy()](http://www.runoob.com/jsref/met-win-scrollby.html) | 按照指定的像素值来滚动内容。 |
| [scrollTo()](http://www.runoob.com/jsref/met-win-scrollto.html) | 把内容滚动到指定的坐标。 |
| [setInterval()](http://www.runoob.com/jsref/met-win-setinterval.html) | 按照指定的周期（以毫秒计）来调用函数或计算表达式。 |
| [setTimeout()](http://www.runoob.com/jsref/met-win-settimeout.html) | 在指定的毫秒数后调用函数或计算表达式。 |

# Navigator 对象

## Navigator 对象

Navigator 对象包含有关浏览器的信息。

Note**注意：** 没有应用于 navigator 对象的公开标准，不过所有浏览器都支持该对象。

## Navigator 对象属性

|  |  |
| --- | --- |
| **属性** | **说明** |
| [appCodeName](http://www.runoob.com/jsref/prop-nav-appcodename.html) | 返回浏览器的代码名 |
| [appName](http://www.runoob.com/jsref/prop-nav-appname.html) | 返回浏览器的名称 |
| [appVersion](http://www.runoob.com/jsref/prop-nav-appversion.html) | 返回浏览器的平台和版本信息 |
| [cookieEnabled](http://www.runoob.com/jsref/prop-nav-cookieenabled.html) | 返回指明浏览器中是否启用 cookie 的布尔值 |
| [platform](http://www.runoob.com/jsref/prop-nav-platform.html) | 返回运行浏览器的操作系统平台 |
| [userAgent](http://www.runoob.com/jsref/prop-nav-useragent.html) | 返回由客户机发送服务器的user-agent 头部的值 |

## Navigator 对象方法

|  |  |
| --- | --- |
| **方法** | **描述** |
| [javaEnabled()](http://www.runoob.com/jsref/met-nav-javaenabled.html) | 指定是否在浏览器中启用Java |
| [taintEnabled()](http://www.runoob.com/jsref/met-nav-taintenabled.html) | 规定浏览器是否启用数据污点(data tainting) |

# Screen 对象

## Screen 对象

Screen 对象包含有关客户端显示屏幕的信息。

Note**注意：** 没有应用于 screen 对象的公开标准，不过所有浏览器都支持该对象。

## Screen 对象属性

|  |  |
| --- | --- |
| **属性** | **说明** |
| [availHeight](http://www.runoob.com/jsref/prop-screen-availheight.html) | 返回屏幕的高度（不包括Windows任务栏） |
| [availWidth](http://www.runoob.com/jsref/prop-screen-availwidth.html) | 返回屏幕的宽度（不包括Windows任务栏） |
| [colorDepth](http://www.runoob.com/jsref/prop-screen-colordepth.html) | 返回目标设备或缓冲器上的调色板的比特深度 |
| [height](http://www.runoob.com/jsref/prop-screen-height.html) | 返回屏幕的总高度 |
| [pixelDepth](http://www.runoob.com/jsref/prop-screen-pixeldepth.html) | 返回屏幕的颜色分辨率（每象素的位数） |
| [width](http://www.runoob.com/jsref/prop-screen-width.html) | 返回屏幕的总宽度 |

# The History 对象

## History 对象

History 对象包含用户（在浏览器窗口中）访问过的 URL。

History 对象是 window 对象的一部分，可通过 window.history 属性对其进行访问。

Note**注意：** 没有应用于History对象的公开标准，不过所有浏览器都支持该对象。

## History 对象属性

|  |  |
| --- | --- |
| **属性** | **说明** |
| [length](http://www.runoob.com/jsref/prop-his-length.html) | 返回历史列表中的网址数 |

## History 对象方法

|  |  |
| --- | --- |
| **方法** | **说明** |
| [back()](http://www.runoob.com/jsref/met-his-back.html) | 加载 history 列表中的前一个 URL |
| [forward()](http://www.runoob.com/jsref/met-his-forward.html) | 加载 history 列表中的下一个 URL |
| [go()](http://www.runoob.com/jsref/met-his-go.html) | 加载 history 列表中的某个具体页面 |

# Location 对象

## Location 对象

Location 对象包含有关当前 URL 的信息。

Location 对象是 window 对象的一部分，可通过 window.Location 属性对其进行访问。

Note**注意：** 没有应用于Location对象的公开标准，不过所有浏览器都支持该对象。

## Location 对象属性

|  |  |
| --- | --- |
| **属性** | **描述** |
| [hash](http://www.runoob.com/jsref/prop-loc-hash.html) | 返回一个URL的锚部分 |
| [host](http://www.runoob.com/jsref/prop-loc-host.html) | 返回一个URL的主机名和端口 |
| [hostname](http://www.runoob.com/jsref/prop-loc-hostname.html) | 返回URL的主机名 |
| [href](http://www.runoob.com/jsref/prop-loc-href.html) | 返回完整的URL |
| [pathname](http://www.runoob.com/jsref/prop-loc-pathname.html) | 返回的URL路径名。 |
| [port](http://www.runoob.com/jsref/prop-loc-port.html) | 返回一个URL服务器使用的端口号 |
| [protocol](http://www.runoob.com/jsref/prop-loc-protocol.html) | 返回一个URL协议 |
| [search](http://www.runoob.com/jsref/prop-loc-search.html) | 返回一个URL的查询部分 |

## Location 对象方法

|  |  |
| --- | --- |
| **方法** | **说明** |
| [assign()](http://www.runoob.com/jsref/met-loc-assign.html) | 载入一个新的文档 |
| [reload()](http://www.runoob.com/jsref/met-loc-reload.html) | 重新载入当前文档 |
| [replace()](http://www.runoob.com/jsref/met-loc-replace.html) | 用新的文档替换当前文档 |

# HTML DOM Document 对象

## HTML DOM 节点

在 HTML DOM (Document Object Model) 中 , 每一个元素都是 **节点**:

* 文档是一个文档。
* 所有的HTML元素都是元素节点。
* 所有 HTML 属性都是属性节点。
* 文本插入到 HTML 元素是文本节点。are text nodes。
* 注释是注释节点。

## Document 对象

当浏览器载入 HTML 文档, 它就会成为 **document 对象**。

document 对象是HTML文档的根节点与所有其他节点（元素节点，文本节点，属性节点, 注释节点）。

Document 对象使我们可以从脚本中对 HTML 页面中的所有元素进行访问。

**提示：**Document 对象是 Window 对象的一部分，可通过 window.document 属性对其进行访问。

## 浏览器支持

Internet ExplorerFirefoxOperaGoogle ChromeSafari

所有主要浏览器都支持 Document 对象。

## Document 对象属性和方法

HTML文档中可以使用以上属性和方法:

|  |  |
| --- | --- |
| **属性 / 方法** | **描述** |
| [document.activeElement](http://www.runoob.com/jsref/prop-document-activeelement.html) | 返回当前获取焦点元素 |
| [document.addEventListener()](http://www.runoob.com/jsref/met-document-addeventlistener.html) | 向文档添加句柄 |
| document.adoptNode(node) | 从另外一个文档返回 adapded 节点到当前文档。 |
| [document.anchors](http://www.runoob.com/jsref/coll-doc-anchors.html) | 返回对文档中所有 Anchor 对象的引用。 |
| document.applets | 返回对文档中所有 Applet 对象的引用。 |
| [document.baseURI](http://www.runoob.com/jsref/prop-doc-baseuri.html) | 返回文档的绝对基础 URI |
| document.body | 返回文档的body元素 |
| [document.close()](http://www.runoob.com/jsref/met-doc-close.html) | 关闭用 document.open() 方法打开的输出流，并显示选定的数据。 |
| [document.cookie](http://www.runoob.com/jsref/prop-doc-cookie.html) | 设置或返回与当前文档有关的所有 cookie。 |
| [document.createAttribute()](http://www.runoob.com/jsref/met-document-createattribute.html) | 创建一个属性节点 |
| [document.createComment()](http://www.runoob.com/jsref/met-document-createcomment.html) | createComment() 方法可创建注释节点。 |
| [document.createDocumentFragment()](http://www.runoob.com/jsref/met-document-createdocumentfragment.html) | 创建空的 DocumentFragment 对象，并返回此对象。 |
| [document.createElement()](http://www.runoob.com/jsref/met-document-createelement.html) | 创建元素节点。 |
| [document.createTextNode()](http://www.runoob.com/jsref/met-document-createtextnode.html) | 创建文本节点。 |
| [document.doctype](http://www.runoob.com/jsref/prop-document-doctype.html) | 返回与文档相关的文档类型声明 (DTD)。 |
| [document.documentElement](http://www.runoob.com/jsref/prop-document-documentelement.html) | 返回文档的根节点 |
| [document.documentMode](http://www.runoob.com/jsref/prop-doc-documentmode.html) | 返回用于通过浏览器渲染文档的模式 |
| [document.documentURI](http://www.runoob.com/jsref/prop-document-documenturi.html) | 设置或返回文档的位置 |
| [document.domain](http://www.runoob.com/jsref/prop-doc-domain.html) | 返回当前文档的域名。 |
| document.domConfig | 返回normalizeDocument()被调用时所使用的配置 |
| document.embeds | 返回文档中所有嵌入的内容（embed）集合 |
| [document.forms](http://www.runoob.com/jsref/coll-doc-forms.html) | 返回对文档中所有 Form 对象引用。 |
| [document. getElementsByClassName()](http://www.runoob.com/jsref/met-document-getelementsbyclassname.html) | 返回文档中所有指定类名的元素集合，作为 NodeList 对象。 |
| [document.getElementById()](http://www.runoob.com/jsref/met-document-getelementbyid.html) | 返回对拥有指定 id 的第一个对象的引用。 |
| [document.getElementsByName()](http://www.runoob.com/jsref/met-doc-getelementsbyname.html) | 返回带有指定名称的对象集合。 |
| [document.getElementsByTagName()](http://www.runoob.com/jsref/met-document-getelementsbytagname.html) | 返回带有指定标签名的对象集合。 |
| [document.images](http://www.runoob.com/jsref/coll-doc-images.html) | 返回对文档中所有 Image 对象引用。 |
| [document.implementation](http://www.runoob.com/jsref/prop-document-implementation.html) | 返回处理该文档的 DOMImplementation 对象。 |
| [document.importNode()](http://www.runoob.com/jsref/met-document-importnode.html) | 把一个节点从另一个文档复制到该文档以便应用。 |
| [document.inputEncoding](http://www.runoob.com/jsref/prop-document-inputencoding.html) | 返回用于文档的编码方式（在解析时）。 |
| [document.lastModified](http://www.runoob.com/jsref/prop-doc-lastmodified.html) | 返回文档被最后修改的日期和时间。 |
| [document.links](http://www.runoob.com/jsref/coll-doc-links.html) | 返回对文档中所有 Area 和 Link 对象引用。 |
| [document.normalize()](http://www.runoob.com/jsref/met-document-normalize.html) | 删除空文本节点，并连接相邻节点 |
| [document.normalizeDocument()](http://www.runoob.com/jsref/met-document-normalizedocument.html) | 删除空文本节点，并连接相邻节点的 |
| [document.open()](http://www.runoob.com/jsref/met-doc-open.html) | 打开一个流，以收集来自任何 document.write() 或 document.writeln() 方法的输出。 |
| [document.querySelector()](http://www.runoob.com/jsref/met-document-queryselector.html) | 返回文档中匹配指定的CSS选择器的第一元素 |
| document.querySelectorAll() | document.querySelectorAll() 是 HTML5中引入的新方法，返回文档中匹配的CSS选择器的所有元素节点列表 |
| [document.readyState](http://www.runoob.com/jsref/prop-doc-readystate.html) | 返回文档状态 (载入中……) |
| [document.referrer](http://www.runoob.com/jsref/prop-doc-referrer.html) | 返回载入当前文档的文档的 URL。 |
| [document.removeEventListener()](http://www.runoob.com/jsref/met-document-removeeventlistener.html) | 移除文档中的事件句柄(由 addEventListener() 方法添加) |
| [document.renameNode()](http://www.runoob.com/jsref/met-document-renamenode.html) | 重命名元素或者属性节点。 |
| document.scripts | 返回页面中所有脚本的集合。 |
| [document.strictErrorChecking](http://www.runoob.com/jsref/prop-document-stricterrorchecking.html) | 设置或返回是否强制进行错误检查。 |
| [document.title](http://www.runoob.com/jsref/prop-doc-title.html) | 返回当前文档的标题。 |
| [document.URL](http://www.runoob.com/jsref/prop-doc-url.html) | 返回文档完整的URL |
| [document.write()](http://www.runoob.com/jsref/met-doc-write.html) | 向文档写 HTML 表达式 或 JavaScript 代码。 |
| [document.writeln()](http://www.runoob.com/jsref/met-doc-writeln.html) | 等同于 write() 方法，不同的是在每个表达式之后写一个换行符。 |

## 警告 !!!

在 W3C DOM核心，文档对象 继承节点对象的所有属性和方法。

很多属性和方法在文档中是没有意义的。

**HTML 文档对象可以避免使用这些节点对象和属性：**

|  |  |
| --- | --- |
| **属性 / 方法** | **避免的原因** |
| document.attributes | 文档没有该属性 |
| document.hasAttributes() | 文档没有该属性 |
| document.nextSibling | 文档没有下一节点 |
| document.nodeName | 这个通常是 #document |
| document.nodeType | 这个通常是 9(DOCUMENT\_NODE) |
| document.nodeValue | 文档没有一个节点值 |
| document.ownerDocument | 文档没有主文档 |
| document.ownerElement | 文档没有自己的节点 |
| document.parentNode | 文档没有父节点 |
| document.previousSibling | 文档没有兄弟节点 |
| document.textContent | 文档没有文本节点 |

# HTML DOM 元素对象

## HTML DOM 节点

在 HTML DOM (Document Object Model) 中, 每个东西都是 **节点**:

* 文档本身就是一个文档对象
* 所有 HTML 元素都是元素节点
* 所有 HTML 属性都是属性节点
* 插入到 HTML 元素文本是文本节点
* 注释是注释节点

## 元素对象

在 HTML DOM 中, **元素对象**代表着一个 HTML 元素。

元素对象 的 **子节点**可以是, 可以是元素节点，文本节点，注释节点。

**NodeList 对象** 代表了节点列表，类似于 HTML元素的子节点集合。

元素可以有属性。属性属于属性节点（查看下一章节）。

## 浏览器支持

Internet ExplorerFirefoxOperaGoogle ChromeSafari

所有主流浏览器都支持 元素对象 和 NodeList 对象。.

## 属性和方法

以上属性和方法可适用于所有 HTML 元素：

|  |  |
| --- | --- |
| **属性 / 方法** | **描述** |
| [*element*.accessKey](http://www.runoob.com/jsref/prop-html-accesskey.html) | 设置或返回accesskey一个元素 |
| [*element*.addEventListener()](http://www.runoob.com/jsref/met-element-addeventlistener.html) | 向指定元素添加事件句柄 |
| [*element*.appendChild()](http://www.runoob.com/jsref/met-node-appendchild.html) | 为元素添加一个新的子元素 |
| [*element*.attributes](http://www.runoob.com/jsref/prop-node-attributes.html) | 返回一个元素的属性数组 |
| [*element*.childNodes](http://www.runoob.com/jsref/prop-node-childnodes.html) | 返回元素的一个子节点的数组 |
| [*element*.classlist](http://www.runoob.com/jsref/prop-element-classlist.html) | 返回元素的类名，作为 DOMTokenList 对象。 |
| [*element*.className](http://www.runoob.com/jsref/prop-html-classname.html) | 设置或返回元素的class属性 |
| *element*.clientHeight | 在页面上返回内容的可视高度（不包括边框，边距或滚动条） |
| *element*.clientWidth | 在页面上返回内容的可视宽度（不包括边框，边距或滚动条） |
| [*element*.cloneNode()](http://www.runoob.com/jsref/met-node-clonenode.html) | 克隆某个元素 |
| [*element*.compareDocumentPosition()](http://www.runoob.com/jsref/met-node-comparedocumentposition.html) | 比较两个元素的文档位置。 |
| [*element*.contentEditable](http://www.runoob.com/jsref/prop-html-contenteditable.html) | 设置或返回元素的内容是否可编辑 |
| [*element*.dir](http://www.runoob.com/jsref/prop-html-dir.html) | 设置或返回一个元素中的文本方向 |
| [*element*.firstChild](http://www.runoob.com/jsref/prop-node-firstchild.html) | 返回元素的第一个子节点 |
| [*element*.focus()](http://www.runoob.com/jsref/met-html-focus.html) | 设置文档或元素获取焦点 |
| [*element*.getAttribute()](http://www.runoob.com/jsref/met-element-getattribute.html) | 返回指定元素的属性值 |
| [*element*.getAttributeNode()](http://www.runoob.com/jsref/met-element-getattributenode.html) | 返回指定属性节点 |
| [*element*.getElementsByTagName()](http://www.runoob.com/jsref/met-element-getelementsbytagname.html) | 返回指定标签名的所有子元素集合。 |
| [*element*. getElementsByClassName()](http://www.runoob.com/jsref/met-element-getelementsbyclassname.html) | 返回文档中所有指定类名的元素集合，作为 NodeList 对象。 |
| *element*.getFeature() | 返回指定特征的执行APIs对象。 |
| *element*.getUserData() | 返回一个元素中关联键值的对象。 |
| [*element*.hasAttribute()](http://www.runoob.com/jsref/met-element-hasattribute.html) | 如果元素中存在指定的属性返回 true，否则返回false。 |
| [*element*.hasAttributes()](http://www.runoob.com/jsref/met-node-hasattributes.html) | 如果元素有任何属性返回true，否则返回false。 |
| [*element*.hasChildNodes()](http://www.runoob.com/jsref/met-node-haschildnodes.html) | 返回一个元素是否具有任何子元素 |
| [*element*.hasfocus()](http://www.runoob.com/jsref/met-document-hasfocus.html) | 返回布尔值，检测文档或元素是否获取焦点 |
| [*element*.id](http://www.runoob.com/jsref/prop-html-id.html) | 设置或者返回元素的 id。 |
| [*element*.innerHTML](http://www.runoob.com/jsref/prop-html-innerhtml.html) | 设置或者返回元素的内容。 |
| [*element*.insertBefore()](http://www.runoob.com/jsref/met-node-insertbefore.html) | 现有的子元素之前插入一个新的子元素 |
| [*element*.isContentEditable](http://www.runoob.com/jsref/prop-html-iscontenteditable.html) | 如果元素内容可编辑返回 true，否则返回false |
| [*element*.isDefaultNamespace()](http://www.runoob.com/jsref/met-node-isdefaultnamespace.html) | 如果指定了namespaceURI 返回 true，否则返回 false。 |
| [*element*.isEqualNode()](http://www.runoob.com/jsref/met-node-isequalnode.html) | 检查两个元素是否相等 |
| [*element*.isSameNode()](http://www.runoob.com/jsref/met-node-issamenode.html) | 检查两个元素所有有相同节点。 |
| [*element*.isSupported()](http://www.runoob.com/jsref/met-node-issupported.html) | 如果在元素中支持指定特征返回 true。 |
| [*element*.lang](http://www.runoob.com/jsref/prop-html-lang.html) | 设置或者返回一个元素的语言。 |
| [*element*.lastChild](http://www.runoob.com/jsref/prop-node-lastchild.html) | 返回的最后一个子元素 |
| [*element*.namespaceURI](http://www.runoob.com/jsref/prop-node-namespaceuri.html) | 返回命名空间的 URI。 |
| [*element*.nextSibling](http://www.runoob.com/jsref/prop-node-nextsibling.html) | 返回该元素紧跟的一个元素 |
| [*element*.nodeName](http://www.runoob.com/jsref/prop-node-nodename.html) | 返回元素的标记名（大写） |
| [*element*.nodeType](http://www.runoob.com/jsref/prop-node-nodetype.html) | Returns the node type of an element |
| [*element*.nodeValue](http://www.runoob.com/jsref/prop-node-nodevalue.html) | 返回元素的类型 |
| [*element*.normalize()](http://www.runoob.com/jsref/met-node-normalize.html) | 使得此成为一个"normal"的形式，其中只有结构（如元素，注释，处理指令，CDATA节和实体引用）隔开Text节点，即元素（包括属性）下面的所有文本节点，既没有相邻的文本节点也没有空的文本节点 |
| *element*.offsetHeight | 返回，任何一个元素的高度包括边框和填充，但不是边距 |
| *element*.offsetWidth | 返回元素的宽度，包括边框和填充，但不是边距 |
| *element*.offsetLeft | 返回当前元素的相对水平偏移位置的偏移容器 |
| *element*.offsetParent | 返回元素的偏移容器 |
| *element*.offsetTop | 返回当前元素的相对垂直偏移位置的偏移容器 |
| [*element*.ownerDocument](http://www.runoob.com/jsref/prop-node-ownerdocument.html) | 返回元素的根元素（文档对象） |
| [*element*.parentNode](http://www.runoob.com/jsref/prop-node-parentnode.html) | 返回元素的父节点 |
| [*element*.previousSibling](http://www.runoob.com/jsref/prop-node-previoussibling.html) | 返回某个元素紧接之前元素 |
| [*element*.querySelector()](http://www.runoob.com/jsref/met-element-queryselector.html) | 返回匹配指定 CSS 选择器元素的第一个子元素 |
| document.querySelectorAll() | 返回匹配指定 CSS 选择器元素的所有子元素节点列表 |
| [*element*.removeAttribute()](http://www.runoob.com/jsref/met-element-removeattribute.html) | 从元素中删除指定的属性 |
| [*element*.removeAttributeNode()](http://www.runoob.com/jsref/met-element-removeattributenode.html) | 删除指定属性节点并返回移除后的节点。 |
| [*element*.removeChild()](http://www.runoob.com/jsref/met-node-removechild.html) | 删除一个子元素 |
| [*element*.removeEventListener()](http://www.runoob.com/jsref/met-element-removeeventlistener.html) | 移除由 addEventListener() 方法添加的事件句柄 |
| [*element*.replaceChild()](http://www.runoob.com/jsref/met-node-replacechild.html) | 替换一个子元素 |
| *element*.scrollHeight | 返回整个元素的高度（包括带滚动条的隐蔽的地方） |
| *element*.scrollLeft | 返回当前视图中的实际元素的左边缘和左边缘之间的距离 |
| *element*.scrollTop | 返回当前视图中的实际元素的顶部边缘和顶部边缘之间的距离 |
| *element*.scrollWidth | 返回元素的整个宽度（包括带滚动条的隐蔽的地方） |
| [*element*.setAttribute()](http://www.runoob.com/jsref/met-element-setattribute.html) | 设置或者改变指定属性并指定值。 |
| [*element*.setAttributeNode()](http://www.runoob.com/jsref/met-element-setattributenode.html) | 设置或者改变指定属性节点。 |
| *element*.setIdAttribute() |  |
| *element*.setIdAttributeNode() |  |
| *element*.setUserData() | 在元素中为指定键值关联对象。 |
| *element*.style | 设置或返回元素的样式属性 |
| [*element*.tabIndex](http://www.runoob.com/jsref/prop-html-tabindex.html) | 设置或返回元素的标签顺序。 |
| [*element*.tagName](http://www.runoob.com/jsref/prop-element-tagname.html) | 作为一个字符串返回某个元素的标记名（大写） |
| [*element*.textContent](http://www.runoob.com/jsref/prop-node-textcontent.html) | 设置或返回一个节点和它的文本内容 |
| [*element*.title](http://www.runoob.com/jsref/prop-html-title.html) | 设置或返回元素的title属性 |
| *element*.toString() | 一个元素转换成字符串 |
|  |  |
| [nodelist.item()](http://www.runoob.com/jsref/met-nodelist-item.html) | 返回某个元素基于文档树的索引 |
| [nodelist.length](http://www.runoob.com/jsref/prop-nodelist-length.html) | 返回节点列表的节点数目。 |

# The HTML DOM 属性 对象

## HTML DOM 节点

在 HTML DOM (Document Object Model) 中, 所有的都是 **节点**：

* 文档是文档节点
* 所有 HTML 元素是元素节点
* 所有 HTML 属性都是属性节点
* 插入到 HTML 元素中的文本为文本节点
* 注释是注释节点

## Attr 对象

在 HTML DOM 中, **Attr 对象** 代表一个 HTML 属性。

HTML属性总是属于HTML元素。

## NamedNodeMap 对象

在 HTML DOM 中, the **NamedNodeMap 对象** 表示一个无顺序的节点列表。

我们可通过节点名称来访问 NamedNodeMap 中的节点。

## 浏览器支持

Internet ExplorerFirefoxOperaGoogle ChromeSafari

所有主流浏览器都支持 Attr 对象和 NamedNodeMap 对象。

<h2属性和方法< h2="">

|  |  |
| --- | --- |
| **属性 / 方法** | **描述** |
| [attr.isId](http://www.runoob.com/jsref/prop-attr-isid.html) | Returns true if the attribute is of type Id, otherwise it returns false |
| [attr.name](http://www.runoob.com/jsref/prop-attr-name.html) | 返回属性名称 |
| [attr.value](http://www.runoob.com/jsref/prop-attr-value.html) | 设置或者返回属性值 |
| [attr.specified](http://www.runoob.com/jsref/prop-attr-specified.html) | 如果属性被指定返回 true ，否则返回 false |
|  |  |
| [nodemap.getNamedItem()](http://www.runoob.com/jsref/met-namednodemap-getnameditem.html) | 从节点列表中返回的指定属性节点。Returns a specified attribute node from a NamedNodeMap. |
| [nodemap.item()](http://www.runoob.com/jsref/met-namednodemap-item.html) | 返回节点列表中处于指定索引号的节点。 |
| [nodemap.length](http://www.runoob.com/jsref/prop-namednodemap-length.html) | 返回节点列表的节点数目。 |
| [nodemap.removeNamedItem()](http://www.runoob.com/jsref/met-namednodemap-removenameditem.html) | 删除指定属性节点 |
| [nodemap.setNamedItem()](http://www.runoob.com/jsref/met-namednodemap-setnameditem.html) | 设置指定属性节点(通过名称) |

## DOM 4 警告 !!!

在 W3C DOM 内核中, Attr (属性) 对象继承节点对象的所有属性和方法 。

在 DOM 4 中, Attr (属性) 对象不再从节点对象中继承。

**从长远的代码质量来考虑，在属性对象中你需要避免使用节点对象属性和方法:**

|  |  |
| --- | --- |
| **属性 / 方法** | **避免原因** |
| *attr*.appendChild() | 属性没有子节点 |
| *attr*.attributes | 属性没有属性 |
| *attr*.baseURI | 使用 document.baseURI 替代 |
| *attr*.childNodes | 属性没有子节点 |
| *attr*.cloneNode() | 使用 attr.value 替代 |
| *attr*.firstChild | 属性没有子节点 |
| *attr*.hasAttributes() | 属性没有属性Attributes don't have attributes |
| *attr*.hasChildNodes | 属性没有子节点Attributes don't have child nodes |
| *attr*.insertBefore() | 属性没有子节点 |
| *attr*.isEqualNode() | 没有意义 |
| *attr*.isSameNode() | 没有意义 |
| *attr*.isSupported() | 通常为 true |
| *attr*.lastChild | 属性没有子节点 |
| *attr*.nextSibling | 属性没有兄弟节点 |
| *attr*.nodeName | 使用 attr.name 替代 |
| *attr*.nodeType | 通常为 2 (ATTRIBUTE-NODE) |
| *attr*.nodeValue | 使用 attr.value 替代 |
| *attr*.normalize() | 属性没有规范 |
| *attr*.ownerDocument | 通常为你的 HTML 文档 |
| *attr*.ownerElement | 你用来访问属性的 HTML 元素 |
| *attr*.parentNode | 你用来访问属性的 HTML 元素 |
| *attr*.previousSibling | 属性没有兄弟节点 |
| *attr*.removeChild | 属性没有子节点 |
| *attr*.replaceChild | 属性没有子节点 |
| *attr*.textContent | 使用 *attr*.value 替代 |

</h2属性和方法<>

# HTML DOM 事件

## HTML DOM 事件

HTML DOM 事件允许Javascript在HTML文档元素中注册不同事件处理程序。

事件通常与函数结合使用，函数不会在事件发生前被执行！ (如用户点击按钮)。

**提示：** 在 W3C 2 级 DOM 事件中规范了事件模型。

## HTML DOM 事件

**DOM：** 指明使用的 DOM 属性级别。

## 鼠标事件

|  |  |  |
| --- | --- | --- |
| **属性** | **描述** | **DOM** |
| [onclick](http://www.runoob.com/jsref/event-onclick.html) | 当用户点击某个对象时调用的事件句柄。 | 2 |
| [oncontextmenu](http://www.runoob.com/jsref/event-oncontextmenu.html) | 在用户点击鼠标右键打开上下文菜单时触发 |  |
| [ondblclick](http://www.runoob.com/jsref/event-ondblclick.html) | 当用户双击某个对象时调用的事件句柄。 | 2 |
| [onmousedown](http://www.runoob.com/jsref/event-onmousedown.html) | 鼠标按钮被按下。 | 2 |
| [onmouseenter](http://www.runoob.com/jsref/event-onmouseenter.html) | 当鼠标指针移动到元素上时触发。 | 2 |
| [onmouseleave](http://www.runoob.com/jsref/event-onmouseleave.html) | 当鼠标指针移出元素时触发 | 2 |
| [onmousemove](http://www.runoob.com/jsref/event-onmousemove.html) | 鼠标被移动。 | 2 |
| [onmouseover](http://www.runoob.com/jsref/event-onmouseover.html) | 鼠标移到某元素之上。 | 2 |
| [onmouseout](http://www.runoob.com/jsref/event-onmouseout.html) | 鼠标从某元素移开。 | 2 |
| [onmouseup](http://www.runoob.com/jsref/event-onmouseup.html) | 鼠标按键被松开。 | 2 |

## 键盘事件

|  |  |  |
| --- | --- | --- |
| **属性** | **描述** | **DOM** |
| [onkeydown](http://www.runoob.com/jsref/event-onkeydown.html) | 某个键盘按键被按下。 | 2 |
| [onkeypress](http://www.runoob.com/jsref/event-onkeypress.html) | 某个键盘按键被按下并松开。 | 2 |
| [onkeyup](http://www.runoob.com/jsref/event-onkeyup.html) | 某个键盘按键被松开。 | 2 |

## 框架/对象（Frame/Object）事件

|  |  |  |
| --- | --- | --- |
| **属性** | **描述** | **DOM** |
| [onabort](http://www.runoob.com/jsref/event-onabort.html) | 图像的加载被中断。 ( <object>) | 2 |
| [onbeforeunload](http://www.runoob.com/jsref/event-onbeforeunload.html) | 该事件在即将离开页面（刷新或关闭）时触发 | 2 |
| [onerror](http://www.runoob.com/jsref/event-onerror.html) | 在加载文档或图像时发生错误。 ( <object>, <body>和 <frameset>) |  |
| [onhashchange](http://www.runoob.com/jsref/event-onhashchange.html) | 该事件在当前 URL 的锚部分发生修改时触发。 |  |
| [onload](http://www.runoob.com/jsref/event-onload.html) | 一张页面或一幅图像完成加载。 | 2 |
| [onpageshow](http://www.runoob.com/jsref/event-onpageshow.html) | 该事件在用户访问页面时触发 |  |
| [onpagehide](http://www.runoob.com/jsref/event-onpagehide.html) | 该事件在用户离开当前网页跳转到另外一个页面时触发 |  |
| [onresize](http://www.runoob.com/jsref/event-onresize.html) | 窗口或框架被重新调整大小。 | 2 |
| [onscroll](http://www.runoob.com/jsref/event-onscroll.html) | 当文档被滚动时发生的事件。 | 2 |
| [onunload](http://www.runoob.com/jsref/event-onunload.html) | 用户退出页面。 ( <body> 和 <frameset>) | 2 |

## 表单事件

|  |  |  |
| --- | --- | --- |
| **属性** | **描述** | **DOM** |
| [onblur](http://www.runoob.com/jsref/event-onblur.html) | 元素失去焦点时触发 | 2 |
| [onchange](http://www.runoob.com/jsref/event-onchange.html) | 该事件在表单元素的内容改变时触发( <input>, <keygen>, <select>, 和 <textarea>) | 2 |
| [onfocus](http://www.runoob.com/jsref/event-onfocus.html) | 元素获取焦点时触发 | 2 |
| [onfocusin](http://www.runoob.com/jsref/event-onfocusin.html) | 元素即将获取焦点时触发 | 2 |
| [onfocusout](http://www.runoob.com/jsref/event-onfocusout.html) | 元素即将失去焦点时触发 | 2 |
| [oninput](http://www.runoob.com/jsref/event-oninput.html) | 元素获取用户输入时触发 | 3 |
| [onreset](http://www.runoob.com/jsref/event-onreset.html) | 表单重置时触发 | 2 |
| [onsearch](http://www.runoob.com/jsref/event-onsearch.html) | 用户向搜索域输入文本时触发 ( <input="search">) |  |
| [onselect](http://www.runoob.com/jsref/event-onselect.html) | 用户选取文本时触发 ( <input> 和 <textarea>) | 2 |
| [onsubmit](http://www.runoob.com/jsref/event-onsubmit.html) | 表单提交时触发 | 2 |

## 剪贴板事件

|  |  |  |
| --- | --- | --- |
| **属性** | **描述** | **DOM** |
| [oncopy](http://www.runoob.com/jsref/event-oncopy.html) | 该事件在用户拷贝元素内容时触发 |  |
| [oncut](http://www.runoob.com/jsref/event-oncut.html) | 该事件在用户剪切元素内容时触发 |  |
| [onpaste](http://www.runoob.com/jsref/event-onpaste.html) | 该事件在用户粘贴元素内容时触发 |  |

## 打印事件

|  |  |  |
| --- | --- | --- |
| **属性** | **描述** | **DOM** |
| [onafterprint](http://www.runoob.com/jsref/event-onafterprint.html) | 该事件在页面已经开始打印，或者打印窗口已经关闭时触发 |  |
| [onbeforeprint](http://www.runoob.com/jsref/event-onbeforeprint.html) | 该事件在页面即将开始打印时触发 |  |

## 拖动事件

|  |  |  |
| --- | --- | --- |
| **事件** | **描述** | **DOM** |
| [ondrag](http://www.runoob.com/jsref/event-ondrag.html) | 该事件在元素正在拖动时触发 |  |
| [ondragend](http://www.runoob.com/jsref/event-ondragend.html) | 该事件在用户完成元素的拖动时触发 |  |
| [ondragenter](http://www.runoob.com/jsref/event-ondragenter.html) | 该事件在拖动的元素进入放置目标时触发 |  |
| [ondragleave](http://www.runoob.com/jsref/event-ondragleave.html) | 该事件在拖动元素离开放置目标时触发 |  |
| [ondragover](http://www.runoob.com/jsref/event-ondragover.html) | 该事件在拖动元素在放置目标上时触发 |  |
| [ondragstart](http://www.runoob.com/jsref/event-ondragstart.html) | 该事件在用户开始拖动元素时触发 |  |
| [ondrop](http://www.runoob.com/jsref/event-ondrop.html) | 该事件在拖动元素放置在目标区域时触发 |  |

## 多媒体（Media）事件

|  |  |  |
| --- | --- | --- |
| **事件** | **描述** | **DOM** |
| [onabort](http://www.runoob.com/jsref/event-onabort-media.html) | 事件在视频/音频（audio/video）终止加载时触发。 |  |
| [oncanplay](http://www.runoob.com/jsref/event-oncanplay.html) | 事件在用户可以开始播放视频/音频（audio/video）时触发。 |  |
| [oncanplaythrough](http://www.runoob.com/jsref/event-oncanplaythrough.html) | 事件在视频/音频（audio/video）可以正常播放且无需停顿和缓冲时触发。 |  |
| [ondurationchange](http://www.runoob.com/jsref/event-ondurationchange.html) | 事件在视频/音频（audio/video）的时长发生变化时触发。 |  |
| onemptied | 当期播放列表为空时触发 |  |
| [onended](http://www.runoob.com/jsref/event-onended.html) | 事件在视频/音频（audio/video）播放结束时触发。 |  |
| [onerror](http://www.runoob.com/jsref/event-onerror-media.html) | 事件在视频/音频（audio/video）数据加载期间发生错误时触发。 |  |
| [onloadeddata](http://www.runoob.com/jsref/event-onloadeddata.html) | 事件在浏览器加载视频/音频（audio/video）当前帧时触发触发。 |  |
| [onloadedmetadata](http://www.runoob.com/jsref/event-onloadedmetadata.html) | 事件在指定视频/音频（audio/video）的元数据加载后触发。 |  |
| [onloadstart](http://www.runoob.com/jsref/event-onloadstart.html) | 事件在浏览器开始寻找指定视频/音频（audio/video）触发。 |  |
| [onpause](http://www.runoob.com/jsref/event-onpause.html) | 事件在视频/音频（audio/video）暂停时触发。 |  |
| [onplay](http://www.runoob.com/jsref/event-onplay.html) | 事件在视频/音频（audio/video）开始播放时触发。 |  |
| [onplaying](http://www.runoob.com/jsref/event-onplaying.html) | 事件在视频/音频（audio/video）暂停或者在缓冲后准备重新开始播放时触发。 |  |
| [onprogress](http://www.runoob.com/jsref/event-onprogress.html) | 事件在浏览器下载指定的视频/音频（audio/video）时触发。 |  |
| [onratechange](http://www.runoob.com/jsref/event-onratechange.html) | 事件在视频/音频（audio/video）的播放速度发送改变时触发。 |  |
| [onseeked](http://www.runoob.com/jsref/event-onseeked.html) | 事件在用户重新定位视频/音频（audio/video）的播放位置后触发。 |  |
| [onseeking](http://www.runoob.com/jsref/event-onseeking.html) | 事件在用户开始重新定位视频/音频（audio/video）时触发。 |  |
| [onstalled](http://www.runoob.com/jsref/event-onstalled.html) | 事件在浏览器获取媒体数据，但媒体数据不可用时触发。 |  |
| [onsuspend](http://www.runoob.com/jsref/event-onsuspend.html) | 事件在浏览器读取媒体数据中止时触发。 |  |
| [ontimeupdate](http://www.runoob.com/jsref/event-ontimeupdate.html) | 事件在当前的播放位置发送改变时触发。 |  |
| [onvolumechange](http://www.runoob.com/jsref/event-onvolumechange.html) | 事件在音量发生改变时触发。 |  |
| [onwaiting](http://www.runoob.com/jsref/event-onwaiting.html) | 事件在视频由于要播放下一帧而需要缓冲时触发。 |  |

## 动画事件

|  |  |  |
| --- | --- | --- |
| **事件** | **描述** | **DOM** |
| [animationend](http://www.runoob.com/jsref/event-animationend.html) | 该事件在 CSS 动画结束播放时触发 |  |
| [animationiteration](http://www.runoob.com/jsref/event-animationiteration.html) | 该事件在 CSS 动画重复播放时触发 |  |
| [animationstart](http://www.runoob.com/jsref/event-animationstart.html) | 该事件在 CSS 动画开始播放时触发 |  |

## 过渡事件

|  |  |  |
| --- | --- | --- |
| **事件** | **描述** | **DOM** |
| [transitionend](http://www.runoob.com/jsref/event-transitionend.html) | 该事件在 CSS 完成过渡后触发。 |  |

## 其他事件

|  |  |  |
| --- | --- | --- |
| **事件** | **描述** | **DOM** |
| onmessage | 该事件通过或者从对象(WebSocket, Web Worker, Event Source 或者子 frame 或父窗口)接收到消息时触发 |  |
| onmousewheel | 已废弃。 使用 [onwheel](http://www.runoob.com/jsref/event-onwheel.html) 事件替代 |  |
| [ononline](http://www.runoob.com/jsref/event-ononline.html) | 该事件在浏览器开始在线工作时触发。 |  |
| [onoffline](http://www.runoob.com/jsref/event-onoffline.html) | 该事件在浏览器开始离线工作时触发。 |  |
| onpopstate | 该事件在窗口的浏览历史（history 对象）发生改变时触发。 |  |
| [onshow](http://www.runoob.com/jsref/event-onshow.html) | 该事件当 <menu> 元素在上下文菜单显示时触发 |  |
| onstorage | 该事件在 Web Storage(HTML 5 Web 存储)更新时触发 |  |
| [ontoggle](http://www.runoob.com/jsref/event-ontoggle.html) | 该事件在用户打开或关闭 <details> 元素时触发 |  |
| [onwheel](http://www.runoob.com/jsref/event-onwheel.html) | 该事件在鼠标滚轮在元素上下滚动时触发 |  |

## 事件对象

### 常量

|  |  |  |
| --- | --- | --- |
| **静态变量** | **描述** | **DOM** |
| CAPTURING-PHASE | 当前事件阶段为捕获阶段(3) | 1 |
| AT-TARGET | 当前事件是目标阶段,在评估目标事件(1) | 2 |
| BUBBLING-PHASE | 当前的事件为冒泡阶段 (2) | 3 |

### 属性

|  |  |  |
| --- | --- | --- |
| **属性** | **描述** | **DOM** |
| [bubbles](http://www.runoob.com/jsref/event-bubbles.html) | 返回布尔值，指示事件是否是起泡事件类型。 | 2 |
| [cancelable](http://www.runoob.com/jsref/event-cancelable.html) | 返回布尔值，指示事件是否可拥可取消的默认动作。 | 2 |
| [currentTarget](http://www.runoob.com/jsref/event-currenttarget.html) | 返回其事件监听器触发该事件的元素。 | 2 |
| eventPhase | 返回事件传播的当前阶段。 | 2 |
| [target](http://www.runoob.com/jsref/event-target.html) | 返回触发此事件的元素（事件的目标节点）。 | 2 |
| [timeStamp](http://www.runoob.com/jsref/event-timestamp.html) | 返回事件生成的日期和时间。 | 2 |
| [type](http://www.runoob.com/jsref/event-type.html) | 返回当前 Event 对象表示的事件的名称。 | 2 |

### 方法

|  |  |  |
| --- | --- | --- |
| **方法** | **描述** | **DOM** |
| initEvent() | 初始化新创建的 Event 对象的属性。 | 2 |
| preventDefault() | 通知浏览器不要执行与事件关联的默认动作。 | 2 |
| stopPropagation() | 不再派发事件。 | 2 |

## 目标事件对象

### 方法

|  |  |  |
| --- | --- | --- |
| **方法** | **描述** | **DOM** |
| addEventListener() | 允许在目标事件中注册监听事件(IE8 = attachEvent()) | 2 |
| dispatchEvent() | 允许发送事件到监听器上 (IE8 = fireEvent()) | 2 |
| removeEventListener() | 运行一次注册在事件目标上的监听事件(IE8 = detachEvent()) | 2 |

## 事件监听对象

### 方法

|  |  |  |
| --- | --- | --- |
| **方法** | **描述** | **DOM** |
| handleEvent() | 把任意对象注册为事件处理程序 | 2 |

## 文档事件对象

### 方法

|  |  |  |
| --- | --- | --- |
| **方法** | **描述** | **DOM** |
| createEvent() |  | 2 |

## 鼠标/键盘事件对象

### 属性

|  |  |  |
| --- | --- | --- |
| **属性** | **描述** | **DOM** |
| [altKey](http://www.runoob.com/jsref/event-altkey.html) | 返回当事件被触发时，"ALT" 是否被按下。 | 2 |
| [button](http://www.runoob.com/jsref/event-button.html) | 返回当事件被触发时，哪个鼠标按钮被点击。 | 2 |
| [clientX](http://www.runoob.com/jsref/event-clientx.html) | 返回当事件被触发时，鼠标指针的水平坐标。 | 2 |
| [clientY](http://www.runoob.com/jsref/event-clienty.html) | 返回当事件被触发时，鼠标指针的垂直坐标。 | 2 |
| [ctrlKey](http://www.runoob.com/jsref/event-ctrlkey.html) | 返回当事件被触发时，"CTRL" 键是否被按下。 | 2 |
| [Location](http://www.runoob.com/jsref/event-key-location.html) | 返回按键在设备上的位置 | 3 |
| [charCode](http://www.runoob.com/jsref/event-key-charcode.html) | 返回onkeypress事件触发键值的字母代码。 | 2 |
| [key](http://www.runoob.com/jsref/event-key-key.html) | 在按下按键时返回按键的标识符。 | 3 |
| [keyCode](http://www.runoob.com/jsref/event-key-keycode.html) | 返回onkeypress事件触发的键的值的字符代码，或者 onkeydown 或 onkeyup 事件的键的代码。 | 2 |
| [which](http://www.runoob.com/jsref/event-key-which.html) | 返回onkeypress事件触发的键的值的字符代码，或者 onkeydown 或 onkeyup 事件的键的代码。 | 2 |
| [metaKey](http://www.runoob.com/jsref/event-metakey.html) | 返回当事件被触发时，"meta" 键是否被按下。 | 2 |
| [relatedTarget](http://www.runoob.com/jsref/event-relatedtarget.html) | 返回与事件的目标节点相关的节点。 | 2 |
| [screenX](http://www.runoob.com/jsref/event-screenx.html) | 返回当某个事件被触发时，鼠标指针的水平坐标。 | 2 |
| [screenY](http://www.runoob.com/jsref/event-screeny.html) | 返回当某个事件被触发时，鼠标指针的垂直坐标。 | 2 |
| [shiftKey](http://www.runoob.com/jsref/event-shiftkey.html) | 返回当事件被触发时，"SHIFT" 键是否被按下。 | 2 |

### 方法

|  |  |  |
| --- | --- | --- |
| **方法** | **描述** | **W3C** |
| initMouseEvent() | 初始化鼠标事件对象的值 | 2 |
| initKeyboardEvent() | 初始化键盘事件对象的值 | 3 |

# HTML DOM Anchor 对象

## Anchor 对象

Anchor 对象表示 HTML 超链接。

在 HTML 文档中 <a> 标签每出现一次，就会创建 Anchor 对象。

锚可用于创建指向另一个文档的链接（通过 href 属性），或者创建文档内的书签（通过 name 属性）。

您可以通过搜索 Document 对象中的 anchors[] 数组来访问锚，或者使用 document.getElementById()。

## Anchor 对象属性

**W3C：** W3C 标准。

|  |  |  |
| --- | --- | --- |
| **属性** | **描述** | **W3C** |
| [charset](http://www.runoob.com/jsref/prop-anchor-charset.html) | 设置或返回被链接资源的字符集。 | Yes |
| [href](http://www.runoob.com/jsref/prop-anchor-href.html) | 设置或返回被链接资源的 URL。 | Yes |
| [hreflang](http://www.runoob.com/jsref/prop-anchor-hreflang.html) | 设置或返回被链接资源的语言代码。 | Yes |
| [name](http://www.runoob.com/jsref/prop-anchor-name.html) | 设置或返回一个链接的名称。 | Yes |
| [rel](http://www.runoob.com/jsref/prop-anchor-rel.html) | 设置或返回当前文档与目标 URL 之间的关系。 | Yes |
| [rev](http://www.runoob.com/jsref/prop-anchor-rel.html) | 设置或返回目标 URL 与之间当前文档的关系。 | Yes |
| [target](http://www.runoob.com/jsref/prop-anchor-target.html) | 设置或返回在何处打开链接。 | Yes |
| [type](http://www.runoob.com/jsref/prop-anchor-type.html) | 设置或返回被链接资源的 MIME 类型。 | Yes |

## 标准属性和事件

Anchor 对象同样支持标准[属性](http://www.runoob.com/jsref/dom-obj-all.html) 和 [事件](http://www.runoob.com/jsref/dom-obj-event.html)。

# HTML DOM Area 对象

## Area 对象

Area 对象代表图像映射的一个区域（图像映射指的是带有可点击区域的图像）

在 HTML 文档中 <area> 标签每出现一次，就会创建一个 Area 对象。

## Area 对象属性

**W3C：** W3C 标准：

|  |  |  |
| --- | --- | --- |
| **属性** | **描述** | **W3C** |
| [alt](http://www.runoob.com/jsref/prop-area-alt.html) | 设置或返回当浏览器无法显示某个区域时的替换文字。 | Yes |
| [coords](http://www.runoob.com/jsref/prop-area-coords.html) | 设置或返回图像映射中可点击区域的坐标。 | Yes |
| [hash](http://www.runoob.com/jsref/prop-area-hash.html) | 设置或返回某个区域中 URL 的锚部分。 | Yes |
| [host](http://www.runoob.com/jsref/prop-area-host.html) | 设置或返回某个区域中 URL 的主机名和端口。 | Yes |
| [hostname](http://www.runoob.com/jsref/prop-area-hostname.html) | 设置或返回href属性值得主机部分。 | Yes |
| [href](http://www.runoob.com/jsref/prop-area-href.html) | 设置或返回某个区域中href属性值 | Yes |
| [noHref](http://www.runoob.com/jsref/prop-area-nohref.html) | 设置或者返回某个区域的 nohref 属性值。 | Yes |
| [pathname](http://www.runoob.com/jsref/prop-area-pathname.html) | 设置或者返回某个区域 href 属性值的路径名部分。 | Yes |
| [port](http://www.runoob.com/jsref/prop-area-port.html) | 设置或者返回某个区域 href 属性值的端口部分。 | Yes |
| [protocol](http://www.runoob.com/jsref/prop-area-protocol.html) | 设置或者返回某个区域 href 属性值的协议部分。 | Yes |
| [search](http://www.runoob.com/jsref/prop-area-search.html) | 设置或者返回某个区域 href 属性值的查询字符串部分。 | Yes |
| [shape](http://www.runoob.com/jsref/prop-area-shape.html) | 设置或者返回某个区域 shape属性值。 | Yes |
| [target](http://www.runoob.com/jsref/prop-area-target.html) | 设置或者返回某个区域 target 属性值。 | Yes |

## 标准属性和事件

Area 对象同样支持标准的 [属性](http://www.runoob.com/jsref/dom-obj-all.html) 和 [事件](http://www.runoob.com/jsref/dom-obj-event.html).

# HTML DOM Audio 对象

## Audio 对象

Audio 对象sHTML5中新增的。。

Audio 对象代表着 HTML <audio> 元素。

### 访问 Audio 对象

你可以使用getElementById()来访问 <audio> 元素：

var x = document.getElementById("myAudio");[**尝试一下**](http://www.runoob.com/try/try.php?filename=tryjsref_audio_get)

### 创建 Audio 对象

你可以使用document.createElement()方法创建 <audio> 元素:

var x = document.createElement("AUDIO");[**尝试一下**](http://www.runoob.com/try/try.php?filename=tryjsref_audio_create)

## Audio 对象属性

|  |  |
| --- | --- |
| **属性** | **描述** |
| [audioTracks](http://www.runoob.com/jsref/prop-audio-audiotracks.html) | 返回表示可用音频轨道的 AudioTrackList 对象 |
| [autoplay](http://www.runoob.com/jsref/prop-audio-autoplay.html) | 设置或返回是否在加载完成后随即播放音频 |
| [buffered](http://www.runoob.com/jsref/prop-audio-buffered.html) | 返回表示音频已缓冲部分的 TimeRanges 对象 |
| [controller](http://www.runoob.com/jsref/prop-audio-controller.html) | 返回表示音频当前媒体控制器的 MediaController 对象 |
| [controls](http://www.runoob.com/jsref/prop-audio-controls.html) | 设置或返回音频是否显示控件（比如播放/暂停等） |
| crossOrigin | 设置或返回音频的 CORS 设置 |
| [currentSrc](http://www.runoob.com/jsref/prop-audio-currentsrc.html) | 回当前音频的 URL |
| [currentTime](http://www.runoob.com/jsref/prop-audio-currenttime.html) | 设置或返回音频中的当前播放位置（以秒计） |
| [defaultMuted](http://www.runoob.com/jsref/prop-audio-defaultmuted.html) | 设置或返回音频默认是否静音 |
| [defaultPlaybackRate](http://www.runoob.com/jsref/prop-audio-defaultplaybackrate.html) | 设置或返回音频的默认播放速度 |
| [duration](http://www.runoob.com/jsref/prop-audio-duration.html) | 返回当前音频的长度（以秒计） |
| [ended](http://www.runoob.com/jsref/prop-audio-ended.html) | 返回音频的播放是否已结束 |
| [error](http://www.runoob.com/jsref/prop-audio-error.html) | 返回表示音频错误状态的 MediaError 对象 |
| [loop](http://www.runoob.com/jsref/prop-audio-loop.html) | 设置或返回音频是否应在结束时重新播放 |
| [mediaGroup](http://www.runoob.com/jsref/prop-audio-mediagroup.html) | 设置或返回音频所属的组合（用于连接多个音频元素） |
| [muted](http://www.runoob.com/jsref/prop-audio-muted.html) | 设置或返回音频是否静音 |
| [networkState](http://www.runoob.com/jsref/prop-audio-networkstate.html) | 返回音频的当前网络状态 |
| [paused](http://www.runoob.com/jsref/prop-audio-paused.html) | 设置或返回音频是否暂停 |
| [playbackRate](http://www.runoob.com/jsref/prop-audio-playbackrate.html) | 设置或返回音频播放的速度 |
| [played](http://www.runoob.com/jsref/prop-audio-played.html) | 返回表示音频已播放部分的 TimeRanges 对象 |
| [preload](http://www.runoob.com/jsref/prop-audio-preload.html) | 设置或返回音频是否应该在页面加载后进行加载 |
| [readyState](http://www.runoob.com/jsref/prop-audio-readystate.html) | 返回音频当前的就绪状态 |
| [seekable](http://www.runoob.com/jsref/prop-audio-seekable.html) | 返回表示音频可寻址部分的 TimeRanges 对象 |
| [seeking](http://www.runoob.com/jsref/prop-audio-seeking.html) | 返回用户是否正在音频中进行查找 |
| [src](http://www.runoob.com/jsref/prop-audio-src.html) | 设置或返回音频元素的当前来源 |
| [textTracks](http://www.runoob.com/jsref/prop-audio-texttracks.html) | 返回表示可用文本轨道的 TextTrackList 对象 |
| [volume](http://www.runoob.com/jsref/prop-audio-volume.html) | 设置或返回音频的音量 |

## Audio 对象方法

|  |  |
| --- | --- |
| **方法** | **描述** |
| [addTextTrack()](http://www.runoob.com/jsref/met_audio_addtexttrack.html) | 在音频中添加一个新的文本轨道 |
| canPlayType() | 检查浏览器是否可以播放指定的音频类型 |
| fastSeek() | 在音频播放器中指定播放时间。 |
| getStartDate() | 返回一个新的Date对象，表示当前时间轴偏移量 |
| load() | 重新加载音频元素 |
| [play()](http://www.runoob.com/jsref/met_audio_play.html) | 开始播放音频 |
| [pause()](http://www.runoob.com/jsref/met_audio_pause.html) | 暂停当前播放的音频 |

## 标准属性和事件

Canvas 对象同样支持标准 [属性](http://www.runoob.com/jsref/dom-obj-all.html) 和 [事件](http://www.runoob.com/jsref/dom-obj-event.html)。

## 相关文章

HTML 教程: [HTML5 audio](http://www.runoob.com/html/html5-audio.html)

HTML 参考手册: [HTML <audio> 标签](http://www.runoob.com/tags/tag-audio.html)

HTML 音频/视频 DOM 参考手册: [HTML 音频/视频](http://www.runoob.com/tags/ref-av-dom.html)

# HTML DOM 对象 Object

## Base Object

Base 对象代表 HTML 的 base 元素。

base 元素适用页面中于指定的默认地址或者所有链接的目标地址。

在 HTML 文档中 <base> 每出现一次，Base 对象就会被创建。

## Base 对象属性

**W3C：** W3C 标准。

|  |  |  |
| --- | --- | --- |
| **属性** | **描述** | **W3C** |
| [href](http://www.runoob.com/jsref/prop-base-href.html) | 设置或返回针对页面中所有链接的基准 URL。 | Yes |
| [target](http://www.runoob.com/jsref/prop-base-target.html) | 设置或返回针对页面中所有链接的默认目标框架。 | Yes |

## 标准属性和事件

Base 对象同样支持标准的 [属性](http://www.runoob.com/jsref/dom-obj-all.html) 和 [事件](http://www.runoob.com/jsref/dom-obj-event.html)。

# HTML DOM Blockquote 对象

## Blockquote 对象

Blockquote 对象代表着一个 HTML 引用(blockquote)块元素。

<blockquote> 标签定义摘自另一个源的块引用。

一个 <blockquote> 元素的内容通常展现为一个左、右两边进行缩进的段落。

在 HTML 文档中的每一个 <blockquote> 标签创建时，都会创建一个 Blockquote 对象。

## Blockquote 对象属性

|  |  |
| --- | --- |
| **属性** | **描述** |
| [cite](http://www.runoob.com/jsref/prop-blockquote-cite.html) | 设置或返回一个引用的cite属性值 |

## 标准属性和事件

Blockquote 对象支持标准 [属性](http://www.runoob.com/jsref/dom-obj-all.html) 和[事件](http://www.runoob.com/jsref/dom-obj-event.html)。

## 相关文章

HTML 参考手册: [HTML <blockquote> 标签](http://www.runoob.com/tags/tag-blockquote.html)

JavaScript reference: [HTML DOM Quote 对象](http://www.runoob.com/jsref/dom-obj-quote.html)

# 引用 cite 属性

[[引用对象参考手册](http://www.runoob.com/jsref/dom-obj-quote.html)引用对象](http://www.runoob.com/jsref/dom-obj-quote.html)

## 实例

返回一个引用来源:

var x = document.getElementById("myQuote").cite;

x 输出结果为:

http://www.wwf.org

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

## 定义与使用

cite 属性用于设置或者返回一个引用的cite属性值 。

cite 属性描述了引用的URL源地址。

**注意：** cite 属性在普通的 Web 浏览器中没有视觉效果，但可以通过屏幕阅读器使用。

## Browser Support

Internet Explorer Firefox Opera Google Chrome Safari

所有主流浏览器都支持 cite 属性。

## 语法

返回 cite 属性：

quoteObject.cite

设置 cite 属性：

quoteObject.cite=*URL*

## 属性值

|  |  |
| --- | --- |
| **值** | **描述** |
| *URL* | 引用的来源的 URL。  可能的值：   * 绝对 URL - 指向另一个网站（比如 cite="http://www.example.com"） * 相对 URL - 指向网站内的一个页面（比如 cite="example.html"） |

## 技术描述

|  |  |
| --- | --- |
| **返回值:** | 一个字符串, 源文档的URL地址。 |

## 更多实例

## 实例

修改引用的 cite 属性值:

document.getElementById("myQuote").cite="http://www.cnn.com/";

[**尝试一下 »**](http://www.runoob.com/try/tryit.php?filename=tryjsref_quote_cite2)

## 相关页面

HTML 参考手册: [HTML <q> cite 属性](http://www.runoob.com/tags/att-q-cite.html)

[[引用对象参考手册](http://www.runoob.com/jsref/dom-obj-quote.html)引用对象](http://www.runoob.com/jsref/dom-obj-quote.html)

# HTML DOM 引用( <q>) 对象

## 引用对象

引用对象定义了一个同内联元素的HTML引用。

<q> 标签定义短的引用。

<q> 元素经常在引用的内容周围添加引号。

HTML 文档中的每一个 <q> 标签 , 都会创建一个引用对象。

## 引用对象属性

|  |  |
| --- | --- |
| **属性** | **描述** |
| [cite](http://www.runoob.com/jsref/prop-quote-cite.html) | 设置或返回一个引用的cite属性值 |

## 标准属性和事件

引用对象支持标准 [属性](http://www.runoob.com/jsref/dom-obj-all.html) 和[事件](http://www.runoob.com/jsref/dom-obj-event.html)。

## 相关文章

HTML 参考手册: [HTML <q> 标签](http://www.runoob.com/tags/tag-q.html)

JavaScript 参考手册: [HTML DOM Blockquote 对象](http://www.runoob.com/jsref/dom-obj-blockquote.html)

# Columngroup span 属性

[[Columngroup 对象参考手册](http://www.runoob.com/jsref/dom-obj-columngroup.html) Columngroup 对象](http://www.runoob.com/jsref/dom-obj-columngroup.html)

## 实例

返回 <colgroup> 元素横跨的列数:

var x = document.getElementById("myColgroup").span;

x 输出结果为:

2

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

## 定义和使用

span 属性设置或者返回列组span属性的值。

span 属性定义 <colgroup> 元素横跨的列数。

**提示：** 描述 <colgroup>中不同列元素的属性, 请使用 [Column](http://www.runoob.com/jsref/dom-obj-col.html) 对象。

## 浏览器支持

Internet Explorer Firefox Opera Google Chrome Safari

所有主流浏览器都支持 span 属性。

## 语法

返回 span 属性:

columngroupObject.span

设置 formMethod 属性:

columngroupObject.span=number

## 属性值

|  |  |
| --- | --- |
| **值** | **描述** |
| number | 描述了<colgroup> 元素中横跨的列数。 |

## 技术描述

|  |  |
| --- | --- |
| **返回值:** | 返回列数 |

## 更多实例

## 实例

使用 span 和 backgroundColor 属性设置前两列的背景颜色为红色：

document.getElementById("myColgroup").span="2";  
document.getElementById("myColgroup").style.backgroundColor="red";

[**尝试一下 »**](http://www.runoob.com/try/tryit.php?filename=tryjsref_colgroup_span2)

## 相关页面

HTML 参考手册: [HTML <colgroup> span 属性](http://www.runoob.com/tags/att-colgroup-span.html)

[[Columngroup 对象参考手册](http://www.runoob.com/jsref/dom-obj-columngroup.html) Columngroup 对象](http://www.runoob.com/jsref/dom-obj-columngroup.html)

# HTML DOM Columngroup 对象

## Columngroup 对象

Columngroup 对象代表着HTML表格（table）的一组列。

在HTML文档中每个 <colgroup> 标签，都会创建一个 Columngroup 对象。

如需对全部列应用样式，<colgroup> 标签很有用，这样就不需要对各个单元和各行重复应用样式了。<colgroup> 标签只能在 table 元素中使用。

## Columngroup 对象属性

|  |  |
| --- | --- |
| **属性** | **描述** |
| [span](http://www.runoob.com/jsref/prop-colgroup-span.html) | 设置或者返回列组中span属性的值。 |

## 标准属性和事件

Columngroup 对象同样支持标准 [属性](http://www.runoob.com/jsref/dom-obj-all.html) 和 [事件](http://www.runoob.com/jsref/dom-obj-event.html)。

## Related Pages

HTML tutorial: [HTML 表格](http://www.runoob.com/html/html-tables.html)

HTML reference: [HTML <colgroup> 标签](http://www.runoob.com/tags/tag-colgroup.html)

JavaScript reference: [HTML DOM Column 对象](http://www.runoob.com/jsref/dom-obj-col.html)

# HTML DOM Datalist 对象

## Datalist 对象

The Datalist 对象是HTML5新增的。

Datalist 对象代表 HTML <datalist> 元素。

### 访问 Datalist 对象

你可以使用 getElementById() 函数来访问 <datalist> 元素：

var x = document.getElementById("myDatalist");[**尝试一下**](http://www.runoob.com/try/tryit.php?filename=tryjsref_datalist_get)

### 创建 Datalist 对象

你可以使用 document.createElement() 方法来创建 <datalist> 元素:

var x = document.createElement("DATALIST");[**尝试一下**](http://www.runoob.com/try/tryit.php?filename=tryjsref_datalist_create)

## Datalist 对象集合

|  |  |
| --- | --- |
| **集合** | **描述** |
| [options](http://www.runoob.com/jsref/coll-datalist-options.html) | 返回 datalist 中所有选项的集合 |

## 标准属性和事件

Datalist 对象同样支持标准 [属性](http://www.runoob.com/jsref/dom-obj-all.html) 和 [事件](http://www.runoob.com/jsref/dom-obj-event.html)。

## 相关页面

HTML 教程: [HTML 表单](http://www.runoob.com/html/html-forms.html)

HTML 参考手册: [HTML <datalist> 标签](http://www.runoob.com/tags/tag-datalist.html)

# HTML DOM del 对象

## del 对象

del 对象代表了HTML文档中已被删除的文本。

<del> 元素定义文档中已被删除的文本。

<del> 元素经常与 <ins> 元素一起使用，来展示不同风格的旧内容与新内容。 浏览器上会在两个标签上的内容加上横线及下划线来区别内容。

在 HTML 文档中每个 <del> 标签都能创建 del 对象。

## del 对象属性

|  |  |
| --- | --- |
| **属性** | **描述** |
| [cite](http://www.runoob.com/jsref/prop-del-cite.html) | 设置或者返回删除文本的cite属性值 |
| [dateTime](http://www.runoob.com/jsref/prop-del-datetime.html) | 设置或者返回删除文本的dateTime属性值 |

## 标准属性和事件

del 对象同样支持标准 [属性](http://www.runoob.com/jsref/dom-obj-all.html) 和 [事件](http://www.runoob.com/jsref/dom-obj-event.html)。

## 相关页面

HTML 教程: [HTML 文本格式化](http://www.runoob.com/html/html-formatting.html)

HTML 参考手册: [HTML <del> 标签](http://www.runoob.com/tags/tag-del.html)

JavaScript 参考手册: [HTML DOM ins 对象](http://www.runoob.com/jsref/dom-obj-ins.html)

# HTML DOM ins 对象

## ins 对象

ins 对象代表了一个已经被插入HTML文档中的文本。

<ins> 元素定义已经被插入文档中的文本。

**提示：**<del> 和 <ins> 一起使用，描述文档中的更新和修正。浏览器通常会在已删除文本上添加一条删除线，在新插入文本下添加一条下划线。

在HTML文档中每个 <ins> 标签都能创建 ins 对象。

## ins 对象属性

|  |  |
| --- | --- |
| **属性** | **描述** |
| [cite](http://www.runoob.com/jsref/prop-ins-cite.html) | 设置或者返回插入文本的cite属性值。 |
| [dateTime](http://www.runoob.com/jsref/prop-ins-datetime.html) | 设置或者返回删除文本的插入属性值 |

## 标准属性和事件

ins 对象同样支持标准 [属性](http://www.runoob.com/jsref/dom-obj-all.html) 和 [事件](http://www.runoob.com/jsref/dom-obj-event.html)。

## 相关页面

HTML 教程: [HTML 文本格式化](http://www.runoob.com/html/html-formatting.html)

HTML 参考手册: [HTML <ins> 标签](http://www.runoob.com/tags/tag-ins.html)

JavaScript 参考手册: [HTML DOM del 对象](http://www.runoob.com/jsref/dom-obj-del.html)

# ins dateTime 属性

[[ins 对象参考手册](http://www.runoob.com/jsref/dom-obj-ins.html) ins 对象](http://www.runoob.com/jsref/dom-obj-ins.html)

## 实例

返回文本被插入的日期和时间：

var x = document.getElementById("myIns").dateTime;

x 输出结果为:

2012-09-15T22:55:03Z

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

## 定义和使用

dateTime 属性用于设置或返回插入文本的 datetime 属性值。

datetime 属性定义了被插入或修改文本的日期和时间。

**注意:** dateTime 属性在普通的网页浏览器中没有视觉效果，但可以通过屏幕阅读器使用。

## 浏览器支持

Internet Explorer Firefox Opera Google Chrome Safari

所有主流浏览器都支持 dateTime 属性。

## 语法

返回 dateTime 属性:

insObject.dateTime

设置 dateTime 属性:

insObject.dateTime=*YYYY-MM-DDThh:mm:ssTZD*

## 属性值

|  |  |
| --- | --- |
| **值** | **描述** |
| *YYYY-MM-DDThh:mm:ssTZD* | 日期或时间。下面解释了其中的成分：   * YYYY - 年 (例如 2011) * MM - 月 (例如 01 表示 January) * DD - 天 (例如 08) * T - 必需的分隔符，若规定时间的话 * hh - 时 (例如 22 表示 10.00pm) * mm - 分 (例如 55) * ss - 秒 (例如 03) * TZD - 时区标识符 (Z 表示祖鲁，也称为格林威治时间) |

## 技术描述

|  |  |
| --- | --- |
| **返回值:** | 字符串, 代表一个插入文本的日期和时间 |

## 更多实例

## 实例

修改 datetime 属性值:

document.getElementById("myIns").dateTime="2013-11-15T21:40:07Z";

[**尝试一下 »**](http://www.runoob.com/try/tryit.php?filename=tryjsref_ins_datetime2)

## 相关文章

HTML 参考手册: [HTML <ins> datetime 属性](http://www.runoob.com/tags/att-ins-datetime.html)

[[ins 对象参考手册](http://www.runoob.com/jsref/dom-obj-ins.html) ins 对象](http://www.runoob.com/jsref/dom-obj-ins.html)

# ins cite 属性

[[ins 对象参考手册](http://www.runoob.com/jsref/dom-obj-ins.html) ins 对象](http://www.runoob.com/jsref/dom-obj-ins.html)

## 实例

返回解释文本为什么被插入的URL文档地址：

var x = document.getElementById("myIns").cite;

x 输出结果为:

http://www.example.com/example/why\_inserted.htm

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

## 定义和使用

cite 属性用于设置或者返回删除文本的cite属性值。

cite 属性指向另外一个文档的 URL，此文档可解释文本被插入或修改的原因。

**注意:** cite 属性在普通的 Web 浏览器中没有视觉效果，但可以通过屏幕阅读器使用。

## 浏览器支持

Internet Explorer Firefox Opera Google Chrome Safari

所有主流浏览器都支持 cite 属性。

## 语法

返回 cite 属性:

insObject.cite

设置 cite 属性:

insObject.cite=*URL*

## 属性值

|  |  |
| --- | --- |
| **值** | **描述** |
| *URL* | 定义了解释文本为什么被插入或修改的URL文档地址。  Possible values:   * 绝对 URL - 指向另一个网站（比如 cite="http://www.example.com"） * 相对 URL - 指向网站内的一个页面（比如 cite="example.html"） |

## 技术描述

|  |  |
| --- | --- |
| **Return Value:** | 字符串, 代表着源文档的URL地址 |

## 更多实例

## 实例

Change the value of the cite attribute:

document.getElementById("myIns").cite="http://www.example.com/whyweinsertedsometext.htm";

[**尝试一下 »**](http://www.runoob.com/try/tryit.php?filename=tryjsref_ins_cite2)

## 相关文章

HTML 参考手册: [HTML <ins> cite 属性](http://www.runoob.com/tags/att-ins-cite.html)

[[ins 对象参考手册](http://www.runoob.com/jsref/dom-obj-ins.html) ins 对象](http://www.runoob.com/jsref/dom-obj-ins.html)

# HTML DOM Keygen 对象

## Keygen 对象

Keygen 对象代表着HTML form表单的 keygen 字段。

该对象提供了一个安全的方式来验证用户。

当提交表单时，私钥存储在本地，公钥发送到服务器。

在 HTML 文档中的每个 <keygen> 标签都能创建一个 Keygen 对象。

你可以通过form 表单的elements[]数组来搜索 keygen 字段，或者使用 document.getElementById()。

## Keygen 对象属性

= HTML5新增属性。

|  |  |
| --- | --- |
| **属性** | **描述** |
| [autofocus](http://www.runoob.com/jsref/prop-keygen-autofocus.html) | 设置或者返回页面加载时是否自动获得焦点。 |
| challenge | 设置或者返回keygen字段的challenge属性值。 |
| [disabled](http://www.runoob.com/jsref/prop-keygen-disabled.html) | 设置或者返回是否用 keytag 字段。 |
| [form](http://www.runoob.com/jsref/prop-keygen-form.html) | 返回包含该 keygen 字段的表单。 |
| [keytype](http://www.runoob.com/jsref/prop-keygen-keytype.html) | 设置或者返回keygen字段的keytype属性值。 |
| [name](http://www.runoob.com/jsref/prop-keygen-name.html) | 设置或者返回keygen字段name属性的值。 |
| [type](http://www.runoob.com/jsref/prop-keygen-type.html) | 返回keygen字段是哪种表单元素类型。 |

## 标准属性和事件

keygen 对象同样支持标准 [属性](http://www.runoob.com/jsref/dom-obj-all.html) 和 [事件](http://www.runoob.com/jsref/dom-obj-event.html)。

## 相关页面

HTML 参考手册: [HTML <keygen> 标签](http://www.runoob.com/tags/tag-keygen.html)

# Keygen autofocus 属性

[[Keygen 对象参考手册](http://www.runoob.com/jsref/dom-obj-keygen.html) Keygen 对象](http://www.runoob.com/jsref/dom-obj-keygen.html)

## 实例

查看 keygen 字段在页面加载时是否自动获取焦点：

var x = document.getElementById("myKeygen").autofocus;

x 输出结果为:

true

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

## 定义和使用

autofocus 属性用于设置或者返回 keygen 字段在页面加载时是否自动获得焦点。

该属性为 HTML autofocus属性。

## 浏览器支持

Internet Explorer Firefox Opera Google Chrome Safari

所有主流浏览器都支持 autofocus 属性，除了 Internet Explorer 。

**注意:** <keygen> 元素中的 HTML autofocus 属性不支持 Firefox。

## 语法

返回 autofocus 属性:

keygenObject.autofocus

设置 autofocus 属性:

keygenObject.autofocus=true|false

## 属性值

|  |  |
| --- | --- |
| **值** | **描述** |
| true|false | 描述了 keygen 字段在页面加载时是否获取焦点。   * true - keygen 字段获取焦点 * false - 默认。 keygen 字段不获取焦点 |

## 技术描述

|  |  |
| --- | --- |
| **返回值:** | 布尔值, 如果keygen字段在页面加载时自动获取焦点返回tru，否则返回false |

## 相关页面

HTML 参考手册: [HTML <keygen> autofocus 属性](http://www.runoob.com/tags/att-keygen-autofocus.html)

[[Keygen 对象参考手册](http://www.runoob.com/jsref/dom-obj-keygen.html) Keygen 对象](http://www.runoob.com/jsref/dom-obj-keygen.html)

# Keygen disabled 属性

[[Keygen 对象参考手册](http://www.runoob.com/jsref/dom-obj-keygen.html) Keygen 对象](http://www.runoob.com/jsref/dom-obj-keygen.html)

## 实例

查看 keygen 字段是否可用:

var x = document.getElementById("myKeygen").disabled;

x 输出结果为:

true

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

## 定义与使用

disabled 属性用于设置或者返回 keygen 字段是否可用。

禁用元素是不能使用的和点击的。禁用元素再浏览器默认呈现为灰色。

该属性体现了 HTML disabled 属性的功能。

## 浏览器支持

Internet Explorer Firefox Opera Google Chrome Safari

所有主流浏览器都支持 disabled 属性，除了Internet Explorer。

## 语法

返回 disabled 属性:

keygenObject.disabled

设置 disabled 属性:

keygenObject.disabled=true|false

## 属性值

|  |  |
| --- | --- |
| **值** | **描述** |
| true|false | 描述了 keygen 字段是否可用   * true - 禁用文本域 * false - 默认。文本域是可用的。 |

## 技术描述

|  |  |
| --- | --- |
| **返回值:** | 布尔值, keygen 字段如果被禁用则返回true，否则返回false |

## 更多实例

## 实例

禁用一个 keygen 字段:

document.getElementById("myKeygen").disabled=true;

输出结果为:

[**尝试一下 »**](http://www.runoob.com/try/tryit.php?filename=tryjsref_keygen_disabled2)

## 实例

禁用及不禁用 keygen 字段:

function disableKey()  
  {  
  document.getElementById("myKeygen").disabled=true;  
  }  
function undisableKey()  
  {  
  document.getElementById("myKeygen").disabled=false;  
  }

[**尝试一下 »**](http://www.runoob.com/try/tryit.php?filename=tryjsref_keygen_disabled3)

## 相关页面

HTML 参考手册: [HTML <keygen> disabled 属性](http://www.runoob.com/tags/att-keygen-disabled.html)

[[Keygen 对象参考手册](http://www.runoob.com/jsref/dom-obj-keygen.html) Keygen 对象](http://www.runoob.com/jsref/dom-obj-keygen.html)

# Keygen form 属性

[[Keygen 对象参考手册](http://www.runoob.com/jsref/dom-obj-keygen.html) Keygen 对象](http://www.runoob.com/jsref/dom-obj-keygen.html)

## 实例

返回包含 <keygen> 元素的表单id:

var x = document.getElementById("myKeygen").form.id;

x 输出结果为:

myForm

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

## 定义与使用

form 属性返回包含 keygen 字段的表单。

成功时form属性返回表单form对象。

**注意:** 该属性是只读的。

## 浏览器支持

Internet Explorer Firefox Opera Google Chrome Safari

所有主流浏览器都支持 form 属性，除了Internet Explorer。

## 语法

keygenObject.form

## 技术描述

|  |  |
| --- | --- |
| **返回值:** | 包含 keygen 字段的表单form元素。如果不包含 keygen 字段，则返回null。 |

[[Keygen 对象参考手册](http://www.runoob.com/jsref/dom-obj-keygen.html) Keygen 对象](http://www.runoob.com/jsref/dom-obj-keygen.html)

# Keygen keytype 属性

[[Keygen 对象参考手册](http://www.runoob.com/jsref/dom-obj-keygen.html) Keygen 对象](http://www.runoob.com/jsref/dom-obj-keygen.html)

## 实例

返回 keygen 字段的 keytype 属性值：

var x = document.getElementById("myKeygen").keytype;

x 输出结果：

rsa

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

## 定义与使用

ketype 属性用于设置或者返回 keygen 字段 keytype 属性的值。

keytype 属性描述了使用的密钥类型。

## 浏览器支持

Internet Explorer Firefox Opera Google Chrome Safari

所有主流浏览器都支持 disabled 属性，除了Internet Explorer 和 Safari。

## 语法

返回 keytype 属性:

keygenObject.keytype

设置 keytype 属性:

keygenObject.keytype="rsa|dsa|ec"

## 属性值

|  |  |
| --- | --- |
| **值** | **描述** |
| rsa | 默认。指定一个RSA加密算法。 |
| dsa | 指定一个DSA加密算法。 |
| ec | 指定EC加密算法。 |

## 技术描述

|  |  |
| --- | --- |
| **返回值:** | 字符串, 代表着密钥类型。 |

## 相关页面

HTML 参考手册: [HTML <keygen> keytype 属性](http://www.runoob.com/tags/att-keygen-keytype.html)

[[Keygen 对象参考手册](http://www.runoob.com/jsref/dom-obj-keygen.html) Keygen 对象](http://www.runoob.com/jsref/dom-obj-keygen.html)

# Keygen name 属性

[[Keygen 对象参考手册](http://www.runoob.com/jsref/dom-obj-keygen.html) Keygen 对象](http://www.runoob.com/jsref/dom-obj-keygen.html)

## 实例

返回 keygen 字段的 name 属性值:

var x = document.getElementById("myKeygen").name;

x 输出结果:

security

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

## 定义与使用

name 属性用于设置或者返回 keygen 字段的 name 属性值。

name 属性描述了 keygen 字段的 name 属性值， 用于在表单数据已被提交后对数据进行引用, 或者在javascript脚本中引用。

**注意:** 只有在表单元素中使用name属性才能在提交表单时将数据传递给后台服务器。

## 浏览器支持

Internet Explorer Firefox Opera Google Chrome Safari

所有主流浏览器都支持 disabled 属性，除了Internet Explorer。

## 语法

返回 name 属性:

keygenObject.name

设置 name 属性:

keygenObject.name=name

## 属性值

|  |  |
| --- | --- |
| **Value** | **Description** |
| name | 描述了 keygen字段的名称。 |

## 技术描述

|  |  |
| --- | --- |
| **返回值:** | 字符串, 代表着 keygen 字段的名称。 |

## 更多实例

## 实例

修改 keygen 字段的 name 属性值:

document.getElementById("myKeygen").name="newName";

[**尝试一下 »**](http://www.runoob.com/try/tryit.php?filename=tryjsref_keygen_name2)

## 相关页面

HTML 参考手册: [HTML <keygen> name 属性](http://www.runoob.com/tags/att-keygen-name.html)

[[Keygen 对象参考手册](http://www.runoob.com/jsref/dom-obj-keygen.html) Keygen 对象](http://www.runoob.com/jsref/dom-obj-keygen.html)

# Keygen type 属性

[[Keygen 对象参考手册](http://www.runoob.com/jsref/dom-obj-keygen.html) Keygen 对象](http://www.runoob.com/jsref/dom-obj-keygen.html)

## 实例

返回form元素keygen字段的类型：

var x = document.getElementById("myKeygen").type;

x 输出结果:

keygen

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

## 定义与使用

type 属性返回form元素的field字段类型。

对于 keygen 字段,属性总是返回 "keygen".

**注意:** 该属性是只读的。

## 浏览器支持

Internet Explorer Firefox Opera Google Chrome Safari

所有主流浏览器都支持 disabled 属性，除了Internet Explorer。

## 语法

keygenObject.type

## 技术细节

|  |  |
| --- | --- |
| **返回值:** | 字符串, 代表着form元素keygen字段的类型。 |

[[Keygen 对象参考手册](http://www.runoob.com/jsref/dom-obj-keygen.html) Keygen 对象](http://www.runoob.com/jsref/dom-obj-keygen.html)

# itle text 属性

[[Title 对象参考手册](http://www.runoob.com/jsref/dom-obj-title.html) Title 对象](http://www.runoob.com/jsref/dom-obj-title.html)

## 实例

返回文档title标签的文本内容:

var x = document.getElementsByTagName("TITLE")[0].text;

x 输出结果为:

HTML DOM Objects

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

## 定义和使用

text 属性用于设置或者返回文档title元素的文本内容。

## 浏览器支持

Internet Explorer Firefox Opera Google Chrome Safari

所有主流浏览器都支持 text 属性。

## 语法

返回 text 属性:

titleObject.text

设置 text 属性:

titleObject.text=*text*

## 属性值

|  |  |
| --- | --- |
| **值** | **描述** |
| *text* | 描述了文档的title元素的文本内容。 |

## 技术细节

|  |  |
| --- | --- |
| **返回值:** | 字符串，返回了文档title元素的文本内容。 |

## 更多实例

## 实例

修改文档中title元素的文本内容：

document.getElementsByTagName("TITLE")[0].text="A new title text..";

[**尝试一下 »**](http://www.runoob.com/try/tryit.php?filename=tryjsref_title_text2)

# HTML DOM Title 对象

## Title 对象

Title 对象代表了一个HTML <title> 元素。

### 访问 Title 对象

你可以使用getElementsByTagName()来访问 <title> 元素：

var x = document.getElementsByTagName("TITLE")[0];[**尝试一下**](http://www.runoob.com/try/try.php?filename=tryjsref_title_get)

### 创建 Title 对象

你可以使用document.createElement()函数来创建 <title> 元素:

var x = document.createElement("TITLE");[**尝试一下**](http://www.runoob.com/try/try.php?filename=tryjsref_title_create)

## Title 对象属性

|  |  |
| --- | --- |
| **属性** | **Description** |
| [text](http://www.runoob.com/jsref/prop-title-text.html) | 设置或返回文档的title元素文本信息 |

## 标准属性和事件

Title 对象支持标准 [属性](http://www.runoob.com/jsref/dom-obj-all.html) 和 [事件](http://www.runoob.com/jsref/dom-obj-event.html)。

## 相关文章

HTML tutorial: [HTML 头部](http://www.runoob.com/html/html-head.html)

HTML reference: [HTML <title> 标签](http://www.runoob.com/tags/tag-title.html)

# HTML DOM Time 对象

## Time 对象

Time 对象是 HTML5 中新增的。

Time 对象表示一个 HTML <time> 元素。

### 访问 Time 对象

您可以使用 getElementById() 来访问 <time> 元素：

var x = document.getElementById("myTime");[**尝试一下**](http://www.runoob.com/try/try.php?filename=tryjsref_time_get)

### 创建 Time 对象

您可以使用 document.createElement() 方法来创建 <time> 元素：

var x = document.createElement("TIME");[**尝试一下**](http://www.runoob.com/try/try.php?filename=tryjsref_time_create)

## Time 对象属性

|  |  |
| --- | --- |
| **属性** | **描述** |
| [dateTime](http://www.runoob.com/jsref/prop-time-datetime.html) | 设置或返回 <time> 元素中 datetime 属性的值。 |

## 标准属性和事件

Time 对象同样支持标准 [属性](http://www.runoob.com/jsref/dom-obj-all.html) 和 [事件](http://www.runoob.com/jsref/dom-obj-event.html)。

## 相关文章

HTML 参考手册：[HTML <time> 标签](http://www.runoob.com/tags/tag-time.html)

# HTML DOM Track 对象

## Track 对象

Track 对象是 HTML5 新增的。

Track 对象表示 HTML <track> 元素。

### 访问 Track 对象

你可以通过使用 getElementById() 来访问 <track> 元素：

var x = document.getElementById("myTrack");

### 创建 Track 对象

你可以使用 document.createElement() 方法来创建 <track> 元素：

var x = document.createElement("TRACK");

## Track 对象属性

|  |  |
| --- | --- |
| **属性** | **描述** |
| [default](http://www.runoob.com/jsref/prop-track-default.html) | 设置或返回轨道的默认状态 |
| [kind](http://www.runoob.com/jsref/prop-track-kind.html) | 设置或返回轨道的 kind 属性值 |
| [label](http://www.runoob.com/jsref/prop-track-label.html) | 设置或返回轨道的 label 属性值 |
| [readyState](http://www.runoob.com/jsref/prop-track-readystate.html) | 返回当前轨道资源的状态 |
| [src](http://www.runoob.com/jsref/prop-track-src.html) | 设置或返回轨道的src属性值 |
| [srclang](http://www.runoob.com/jsref/prop-track-srclang.html) | 设置或返回轨道的 srclang 属性值 |
| track | 返回 TextTrack 对象，表示 track 元素的轨道文本数据 |

## 标准属性和事件

Track 对象同样支持标准 [属性](http://www.runoob.com/jsref/dom-obj-all.html) 和 [事件](http://www.runoob.com/jsref/dom-obj-event.html)。

## 相关文章

HTML 参考手册: [HTML <track> 标签](http://www.runoob.com/tags/tag-track.html)

# HTML DOM Video 对象

## Video 对象

Video 对象是 HTML5 中新增的。

Video 对象表示 HTML <video> 元素。

### 访问 Video 对象t

您可以通过使用 getElementById() 来访问 <video> 元素：

var x = document.getElementById("myVideo");[**尝试一下**](http://www.runoob.com/try/try.php?filename=tryjsref_video_get)

### 创建 Video 对象

您可以通过使用 document.createElement() 方法来创建 <video> 元素：

var x = document.createElement("VIDEO");[**尝试一下**](http://www.runoob.com/try/try.php?filename=tryjsref_video_create)

## Video 对象属性

|  |  |
| --- | --- |
| **属性** | **描述** |
| [audioTracks](http://www.runoob.com/jsref/prop-video-audiotracks.html) | 返回表示可用音频轨道的 AudioTrackList 对象。 |
| [autoplay](http://www.runoob.com/jsref/prop-video-autoplay.html) | 设置或返回是否在就绪（加载完成）后随即播放视频。 |
| [buffered](http://www.runoob.com/jsref/prop-video-buffered.html) | 返回表示视频已缓冲部分的 TimeRanges 对象。 |
| [controller](http://www.runoob.com/jsref/prop-video-controller.html) | 返回表示视频当前媒体控制器的 MediaController 对象。 |
| [controls](http://www.runoob.com/jsref/prop-video-controls.html) | 设置或返回视频是否应该显示控件（比如播放/暂停等）。 |
| crossOrigin | 设置或返回视频的 CORS 设置。 |
| [currentSrc](http://www.runoob.com/jsref/prop-video-currentsrc.html) | 返回当前视频的 URL。 |
| [currentTime](http://www.runoob.com/jsref/prop-video-currenttime.html) | 设置或返回视频中的当前播放位置（以秒计）。 |
| [defaultMuted](http://www.runoob.com/jsref/prop-video-defaultmuted.html) | 设置或返回视频默认是否静音。 |
| [defaultPlaybackRate](http://www.runoob.com/jsref/prop-video-defaultplaybackrate.html) | 设置或返回视频的默认播放速度。 |
| [duration](http://www.runoob.com/jsref/prop-video-duration.html) | 返回视频的长度（以秒计）。 |
| [ended](http://www.runoob.com/jsref/prop-video-ended.html) | 返回视频的播放是否已结束。 |
| [error](http://www.runoob.com/jsref/prop-video-error.html) | 返回表示视频错误状态的 MediaError 对象。 |
| [height](http://www.runoob.com/jsref/prop-video-height.html) | 设置或返回视频的 height 属性的值。 |
| [loop](http://www.runoob.com/jsref/prop-video-loop.html) | 设置或返回视频是否应在结束时再次播放。 |
| [mediaGroup](http://www.runoob.com/jsref/prop-video-mediagroup.html) | 设置或返回视频所属媒介组合的名称。 |
| [muted](http://www.runoob.com/jsref/prop-video-muted.html) | 设置或返回是否关闭声音。 |
| [networkState](http://www.runoob.com/jsref/prop-video-networkstate.html) | 返回视频的当前网络状态。 |
| [paused](http://www.runoob.com/jsref/prop-video-paused.html) | 设置或返回视频是否暂停。 |
| [playbackRate](http://www.runoob.com/jsref/prop-video-playbackrate.html) | 设置或返回视频播放的速度。 |
| [played](http://www.runoob.com/jsref/prop-video-played.html) | 返回表示视频已播放部分的 TimeRanges 对象。 |
| [poster](http://www.runoob.com/jsref/prop-video-poster.html) | 设置或返回视频的 poster 属性的值。 |
| [preload](http://www.runoob.com/jsref/prop-video-preload.html) | 设置或返回视频的 preload 属性的值。 |
| [readyState](http://www.runoob.com/jsref/prop-video-readystate.html) | 返回视频当前的就绪状态。 |
| [seekable](http://www.runoob.com/jsref/prop-video-seekable.html) | 返回表示视频可寻址部分的 TimeRanges 对象。 |
| [seeking](http://www.runoob.com/jsref/prop-video-seeking.html) | 返回用户当前是否正在视频中进行查找。 |
| [src](http://www.runoob.com/jsref/prop-video-src.html) | 设置或返回视频的 src 属性的值。 |
| [startDate](http://www.runoob.com/jsref/prop-video-startdate.html) | 返回表示当前时间偏移的 Date 对象。 |
| [textTracks](http://www.runoob.com/jsref/prop-video-texttracks.html) | 返回表示可用文本轨道的 TextTrackList 对象。 |
| [videoTracks](http://www.runoob.com/jsref/prop-video-videotracks.html) | 返回表示可用视频轨道的 VideoTrackList 对象。 |
| [volume](http://www.runoob.com/jsref/prop-video-volume.html) | 设置或返回视频的音量。 |
| [width](http://www.runoob.com/jsref/prop-video-width.html) | 设置或返回视频的 width 属性的值。 |

## Video 对象方法

|  |  |
| --- | --- |
| **Method** | **描述** |
| [addTextTrack()](http://www.runoob.com/jsref/met-video-addtexttrack.html) | 向视频添加新的文本轨道。 |
| [canPlayType()](http://www.runoob.com/jsref/met-video-canplaytype.html) | 检查浏览器是否能够播放指定的视频类型。 |
| [load()](http://www.runoob.com/jsref/met-video-load.html) | 重新加载视频元素。 |
| [play()](http://www.runoob.com/jsref/met-video-play.html) | 开始播放视频。 |
| [pause()](http://www.runoob.com/jsref/met-video-pause.html) | 暂停当前播放的视频。 |

## 标准属性和事件

Video 对象同样支持标准的 [属性](http://www.runoob.com/jsref/dom-obj-all.html) 和 [事件](http://www.runoob.com/jsref/dom-obj-event.html)。

## 相关文章

HTML 教程: [HTML5 视频](http://www.runoob.com/html/html5-video.html)

HTML 参考手册: [HTML <video> 标签](http://www.runoob.com/tags/tag-video.html)