**Class和ID的区别**

id是元素的名称，可以供js或其它脚本程序来访问该元素对象，而class是该元素的css类名。

id的值在整个当前网页中应该是唯一的，即某一个元素定义了id="aaa"，那么这个网页中其它的元素的id就不能定义成aaa，而class则可以。

ID是唯一的并是父级的，CLASS是可以重复的并是子级的。

**JavaScript：写入 HTML 输出**

document.write("<h1>This is a heading</h1>");

document.write("<p>This is a paragraph</p>");

**JavaScript：对事件作出反应**

<button type="button" onclick="alert('Welcome!')">点击这里</button>

**JavaScript：改变 HTML 内容**

x=document.getElementById("demo") //查找元素

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

**JavaScript：改变 HTML 图像**

function changeImage()

{element=document.getElementById('myimage')

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

{element.src="/i/eg\_bulboff.gif"; }

else { element.src="/i/eg\_bulbon.gif"; }

}

## JavaScript：改变 HTML 样式

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

x.style.color="#ff0000"; //改变样式

document.getElementById(id).style.property=new style

## JavaScript：验证输入是否为数字

function myFunction()

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

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

{ alert("Not Numeric"); }

}

**改变 HTML 属性**

document.getElementById(id).attribute=new value

## <head> 或 <body> 中的 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。

## 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>

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

</body>

</html>

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

## 对代码行进行折行

document.write("Hello \

World!");

## JavaScript 变量

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）

## 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：

**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:"Bill", lastname:"Gates", id:5566};

**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。

**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;

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。

**属性和方法**

属性是与对象相关的值。 方法是能够在对象上执行的动作。

举例：汽车就是现实生活中的对象。

汽车的属性：

car.name=Fiat

car.model=500

car.weight=850kg

car.color=white

汽车的方法：

car.start()

car.drive()

car.brake()

**JavaScript 中的对象**

在 JavaScript 中，对象是数据（变量），拥有属性和方法。

当您像这样声明一个 JavaScript 变量时：

var txt = "Hello";

您实际上已经创建了一个 JavaScript 字符串对象。字符串对象拥有内建的属性 length。对于上面的字符串来说，length 的值是 5。

字符串对象同时拥有若干个内建的方法。

属性：

txt.length=5

方法：

txt.indexOf()

txt.replace()

txt.search()

**创建 JavaScript 对象**

JavaScript 中的几乎所有事务都是对象：字符串、数字、数组、日期、函数，等等。 你也可以创建自己的对象。

本例创建名为 "person" 的对象，并为其添加了四个属性：

实例 person=new Object();

person.firstname="Bill";

person.lastname="Gates";

person.age=56;

person.eyecolor="blue";

**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。

function myFunction()

{ var x=5;

return x;

}

上面的函数会返回值 5。

注释：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。

在您仅仅希望退出函数时 ，也可使用 return 语句。

**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 变量来分配值**

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。

**对字符串和数字进行加法运算**

规则是： 如果把数字与字符串相加，结果将成为字符串。

其他类型第一个是什么类型就转变为什么类型（如果没有特定的声明）

**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 greeting=(visitor=="PRES")?"Dear President ":"Dear ";

如果变量 visitor 中的值是 "PRES"，则向变量 greeting 赋值 "Dear President "，否则赋值 "Dear"。

JavaScript 测试和捕捉 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

}

## 必填（或必选）项目

<!DOCTYPE html>

<html>

<head>

<script type="text/javascript">

function validate\_required(field,alerttxt)

{

with (field)

{

if (value==null||value=="")

{alert(alerttxt);return false}

else {return true}

}

}

function validate\_form(thisform)

{

with (thisform)

{

if (validate\_required(email,"Email must be filled out!")==false)

{email.focus();return false}

}

}

</script>

</head>

<body>

<form action="submitpage.htm" onsubmit="return validate\_form(this)" method="post">

Email: <input type="text" name="email" size="30">

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

</form>

</body>

</html>

**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：

<html>

<head>

<script type="text/javascript">

function validate\_email(field,alerttxt)

{

with (field)

{

apos=value.indexOf("@")

dotpos=value.lastIndexOf(".")

if (apos<1||dotpos-apos<2)

{alert(alerttxt);return false}

else {return true}

}

}

function validate\_form(thisform)

{

with (thisform)

{

if (validate\_email(email,"Not a valid e-mail address!")==false)

{email.focus();return false}

}

}

</script>

</head>

<body>

<form action="submitpage.htm"onsubmit="return validate\_form(this);" method="post">

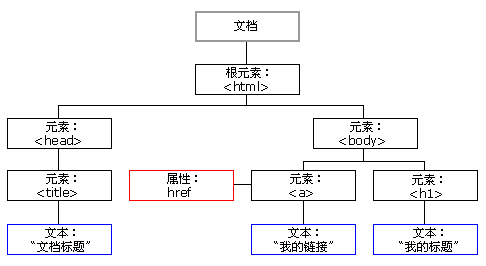
Email: <input type="text" name="email" size="30">

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

</form>

</body>

</html>



**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**

var x=document.getElementById("intro");

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**

实例 本例查找 id="main" 的元素，然后查找 "main" 中的所有 <p> 元素：

var x=document.getElementById("main");

var y=x.getElementsByTagName("p");

查找第一条P元素

**HTML DOM 使 JavaScript 有能力对 HTML 事件做出反应。**

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

向 button 元素分配 onclick 事件：

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

**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。

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。

<input type="text" id="fname" onchange="upperCase()">

输入完成后调用一个函数

## 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。

<!DOCTYPE html>

<html>

<body>

<div onmouseover="mOver(this)" onmouseout="mOut(this)" style="background-color:green;width:120px;height:20px;padding:40px;color:#ffffff;">把鼠标移到上面</div>

<script>

function mOver(obj)

{

obj.innerHTML="谢谢"

}

function mOut(obj)

{

obj.innerHTML="把鼠标移到上面"

}

</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。

<!DOCTYPE html>

<html>

<body>

<div onmousedown="mDown(this)" onmouseup="mUp(this)" style="background-color:green;color:#ffffff;width:90px;height:20px;padding:40px;font-size:12px;">请点击这里</div>

<script>

function mDown(obj)

{

obj.style.backgroundColor="#1ec5e5";

obj.innerHTML="请释放鼠标按钮"

}

function mUp(obj)

{

obj.style.backgroundColor="green";

obj.innerHTML="请按下鼠标按钮"

}

</script>

</body>

</html>

## 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。

<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"); //这段代码创建新的 <p> 元素：

var node=document.createTextNode("这是新段落。"); //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：

para.appendChild(node); //然后您必须向 <p> 元素追加这个文本节点：

var element=document.getElementById("div1"); //这段代码找到一个已有的元素：

element.appendChild(para); // 这段代码向这个已有的元素追加新元素：

</script>

## 删除已有的 HTML 元素

## 如需删除 HTML 元素，您必须首先获得该元素的父元素：

## <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>

## <script>

## var parent=document.getElementById("div1"); //找到 id="div1" 的元素：

## var child=document.getElementById("p1"); //找到 id="p1" 的 <p> 元素：

## parent.removeChild(child); //从父元素中删除子元素：

## </script>

## 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);

## JavaScript 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。

## 使用对象构造器

## function person(firstname,lastname,age,eyecolor)

## {

## this.firstname=firstname;

## this.lastname=lastname;

## this.age=age;

## this.eyecolor=eyecolor;

## }

## 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。

## 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 String（字符串）对象 实例

## 如何为字符串添加样式。

## <html>

## <body>

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

## var txt="Hello World!"

## document.write("<p>Big: " + txt.big() + "</p>")

## document.write("<p>Small: " + txt.small() + "</p>")

## document.write("<p>Bold: " + txt.bold() + "</p>")

## document.write("<p>Italic: " + txt.italics() + "</p>")

## document.write("<p>Blink: " + txt.blink() + " (does not work in IE)</p>")

## document.write("<p>Fixed: " + txt.fixed() + "</p>")

## document.write("<p>Strike: " + txt.strike() + "</p>")

## document.write("<p>Fontcolor: " + txt.fontcolor("Red") + "</p>")

## document.write("<p>Fontsize: " + txt.fontsize(16) + "</p>")

## document.write("<p>Lowercase: " + txt.toLowerCase() + "</p>")

## document.write("<p>Uppercase: " + txt.toUpperCase() + "</p>")

## document.write("<p>Subscript: " + txt.sub() + "</p>")

## document.write("<p>Superscript: " + txt.sup() + "</p>")

## document.write("<p>Link: " + txt.link("http://www.w3school.com.cn") + "</p>")

## </script>

## </body>

## </html>

## indexOf() 方法

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

## var str="Hello world!"

## document.write(str.indexOf("Hello") + "<br />")

## document.write(str.indexOf("World") + "<br />")

## document.write(str.indexOf("world"))

## 有的话输出位置，没有输出-1

## match() 方法

## 如何使用 match() 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。

## var str="Hello world!"

## document.write(str.match("world") + "<br />")

## document.write(str.match("World") + "<br />")

## document.write(str.match("worlld") + "<br />")

## document.write(str.match("world!"))

## 找到输出字符串 找不到输出null

## 如何替换字符串中的字符 - replace()

## 如何使用 replace() 方法在字符串中用某些字符替换另一些字符。

## var str="Visit Microsoft!"

## document.write(str.replace(/Microsoft/,"W3School"))

## 下面的例子使用字符串对象的toUpperCase()方法将字符串转换为大写：

var txt="Hello world!"

document.write(txt.toUpperCase())

## JavaScript Date（日期）对象

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：

var myDate=new Date()

**注释：**Date 对象自动使用当前的日期和时间作为其初始值。

## 操作日期

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 (2008 年 8 月 9 日)：

var myDate=new Date()

myDate.setFullYear(2008,7,9)

**注意：**表示月份的参数介于 0 到 11 之间。也就是说，如果希望把月设置为 8 月，则参数应该是 7。

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。

## 比较日期

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 2008 年 8 月 9 日做了比较：

var myDate=new Date();

myDate.setFullYear(2008,8,9);

var today = new Date();

if (myDate>today)

{

alert("Today is before 9th August 2008");

}

else

{

alert("Today is after 9th August 2008");

}

[For...In 声明](http://www.w3school.com.cn/tiy/t.asp?f=jsrf_array_for_in)

使用 for...in 声明来循环输出数组中的元素。

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

## 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>

[合并两个数组 - concat()](http://www.w3school.com.cn/tiy/t.asp?f=jseg_concat_2)

如何使用 concat() 方法来合并两个数组。

## document.write(arr.concat(arr2))

[用数组的元素组成字符串 - join()](http://www.w3school.com.cn/tiy/t.asp?f=jseg_join)

如何使用 join() 方法将数组的所有元素组成一个字符串。

## var arr = new Array(3);

## arr[0] = "George"

## arr[1] = "John"

## arr[2] = "Thomas"

## document.write(arr.join());

## document.write("<br />");

## document.write(arr.join("."));

[文字数组 - sort()](http://www.w3school.com.cn/tiy/t.asp?f=jseg_sort_1)

如何使用 sort() 方法从字面上对数组进行排序。(文字数组按照首字母排序、数字数组按照从小到大排序)

## var arr = new Array(6)

## arr[0] = "George"

## arr[1] = "John"

## arr[2] = "Thomas"

## arr[3] = "James"

## arr[4] = "Adrew"

## arr[5] = "Martin"

## document.write(arr + "<br />")

## document.write(arr.sort())

Boolean（逻辑）对象用于将非逻辑值转换为逻辑值（true 或者 false）。

0 是逻辑的 false  
1 是逻辑的 true  
空字符串是逻辑的 false  
null 是逻辑的 false  
NaN 是逻辑的 false  
字符串 'false' 是逻辑的 true

### 创建 Boolean 对象

使用关键词 new 来定义 Boolean 对象。下面的代码定义了一个名为 myBoolean 的逻辑对象：

var myBoolean=new Boolean()

下面的所有的代码行均会创建初始值为 false 的 Boolean 对象。

var myBoolean=new Boolean();

var myBoolean=new Boolean(0);

var myBoolean=new Boolean(null);

var myBoolean=new Boolean("");

var myBoolean=new Boolean(false);

var myBoolean=new Boolean(NaN);

下面的所有的代码行均会创初始值为 true 的 Boolean 对象：

var myBoolean=new Boolean(1);

var myBoolean=new Boolean(true);

var myBoolean=new Boolean("true");

var myBoolean=new Boolean("false");

var myBoolean=new Boolean("Bill Gates");

Math.floor(Math.random()\*11)

Floor限制数字在0-10之间的整数（random（）\*11到不了11）

## 什么是 RegExp？

RegExp 是正则表达式的缩写。

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。

## 定义 RegExp

RegExp 对象用于存储检索模式。

通过 new 关键词来定义 RegExp 对象。以下代码定义了名为 patt1 的 RegExp 对象，其模式是 "e"：

var patt1=new RegExp("e");

当您使用该 RegExp 对象在一个字符串中检索时，将寻找的是字符 "e"。

## egExp 对象的方法

RegExp 对象有 3 个方法：test()、exec() 以及 compile()。

## test()

test() 方法检索字符串中的指定值。返回值是 true 或 false。

### 例子：

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

## 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。

### 例子 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

您可以向 RegExp 对象添加第二个参数，以设定检索。例如，如果需要找到所有某个字符的所有存在，则可以使用 "g" 参数 ("global")。

如需关于如何修改搜索模式的完整信息，请访问我们的 [RegExp 对象参考手册](http://www.w3school.com.cn/jsref/jsref_obj_regexp.asp)。

在使用 "g" 参数时，exec() 的工作原理如下：

* 找到第一个 "e"，并存储其位置
* 如果再次运行 exec()，则从存储的位置开始检索，并找到下一个 "e"，并存储其位置

var patt1=new RegExp("e","g");

do

{

result=patt1.exec("The best things in life are free");

document.write(result);

}

while (result!=null)

由于这个字符串中 6 个 "e" 字母，代码的输出将是：

eeeeeenull

## compile()

compile() 方法用于改变 RegExp。

compile() 既可以改变检索模式，也可以添加或删除第二个参数。

### 例子：

var patt1=new RegExp("e");

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

patt1.compile("d"); //把检索参数改成d

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

由于字符串中存在 "e"，而没有 "d"，以上代码的输出是：

truefalse

## 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。

## 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;

## 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

## Window Screen

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

一些属性：

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

## Window Screen 可用宽度

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

### 实例

返回您的屏幕的可用宽度：

<script>

document.write("可用宽度：" + screen.availWidth);

</script>

以上代码输出为：

可用宽度：1920

## Window Screen 可用高度

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

### 实例

返回您的屏幕的可用高度：

<script>

document.write("可用高度：" + screen.availHeight);

</script>

以上代码输出为：

可用高度：1032

## Window Location

**window.location** 对象在编写时可不使用 window 这个前缀。

一些例子：

* 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.w3school.com.cn/js/js\_window\_location.asp

## 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.asp

## 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.w3school.com.cn")

}

</script>

</head>

<body>

<input type="button" value="加载新文档" onclick="newDoc()">

</body>

</html>

## 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：

转到上一页

## 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：

转到下一页

## 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：

alert("文本")

## 确认框

确认框用于使用户可以验证或者接受某些信息。

当确认框出现后，用户需要点击确定或者取消按钮才能继续进行操作。

如果用户点击确认，那么返回值为 true。如果用户点击取消，那么返回值为 false。

### 语法：

confirm("文本")

## 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：

prompt("文本","默认值")

**setTimeout()**

未来的某时执行代码

**clearTimeout()**

取消setTimeout()

## setTimeout()

### 语法

var t=setTimeout("javascript语句",毫秒) // javascript语句例如alert（“。。。。”）

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 - 无穷循环

要创建一个运行于无穷循环中的计时器，我们需要编写一个函数来调用其自身。在下面的例子中，当按钮被点击后，输入域便从 0 开始计数。

<html>

<head>

<script type="text/javascript">

var c=0

var t

function timedCount()

{

document.getElementById('txt').value=c

c=c+1

t=setTimeout("timedCount()",1000)

}

</script>

</head>

<body>

<form>

<input type="button" value="Start count!" onClick="timedCount()">

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

</form>

</body>

</html>

### 引用 jQuery

<script src="https://ajax.googleapis.com/ajax/libs/jquery/1.8.3/jquery.min.js">

</script>

## 请看下面的 HTML 片段：

<html>

<head>

<title>DOM 教程</title>

</head>

<body>

<h1>DOM 第一课</h1>

<p>Hello world!</p>

</body>

</html>

从上面的 HTML 中：

* <html> 节点没有父节点；它是根节点
* <head> 和 <body> 的父节点是 <html> 节点
* 文本节点 "Hello world!" 的父节点是 <p> 节点

并且：

* <html> 节点拥有两个子节点：<head> 和 <body>
* <head> 节点拥有一个子节点：<title> 节点
* <title> 节点也拥有一个子节点：文本节点 "DOM 教程"
* <h1> 和 <p> 节点是同胞节点，同时也是 <body> 的子节点

并且：

* <head> 元素是 <html> 元素的首个子节点
* <body> 元素是 <html> 元素的最后一个子节点
* <h1> 元素是 <body> 元素的首个子节点
* <p> 元素是 <body> 元素的最后一个子节点

## 警告！

DOM 处理中的常见错误是希望元素节点包含文本。

在本例中：**<title>DOM 教程</title>**，元素节点 <title>，包含值为 "DOM 教程" 的**文本节点**。

可通过节点的 **innerHTML** 属性来访问文本节点的值。

这里提供一些您将在本教程中学到的常用方法：

|  |  |
| --- | --- |
| **方法** | **描述** |
| getElementById() | 返回带有指定 ID 的元素。 |
| getElementsByTagName() | 返回包含带有指定标签名称的所有元素的节点列表（集合/节点数组）。 |
| getElementsByClassName() | 返回包含带有指定类名的所有元素的节点列表。 |
| appendChild() | 把新的子节点添加到指定节点。 |
| removeChild() | 删除子节点。 |
| replaceChild() | 替换子节点。 |
| insertBefore() | 在指定的子节点前面插入新的子节点。 |
| createAttribute() | 创建属性节点。 |
| createElement() | 创建元素节点。 |
| createTextNode() | 创建文本节点。 |
| getAttribute() | 返回指定的属性值。 |
| setAttribute() | 把指定属性设置或修改为指定的值。 |

# HTML DOM 属性

## innerHTML 属性

获取元素内容的最简单方法是使用 innerHTML 属性。

innerHTML 属性对于获取或替换 HTML 元素的内容很有用。

## nodeName 属性

nodeName 属性规定节点的名称。

* nodeName 是只读的
* 元素节点的 nodeName 与标签名相同
* 属性节点的 nodeName 与属性名相同
* 文本节点的 nodeName 始终是 #text
* 文档节点的 nodeName 始终是 #document

**注释：**nodeName 始终包含 HTML 元素的大写字母标签名。

## nodeValue 属性

nodeValue 属性规定节点的值。

* 元素节点的 nodeValue 是 undefined 或 null
* 文本节点的 nodeValue 是文本本身
* 属性节点的 nodeValue 是属性值

<html>

<body>

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

<script type="text/javascript">

x=document.getElementById("intro");

document.write(x.firstChild.nodeValue);

</script>

</body>

</html>

返回hello，world

**nodeType 属性**

nodeType 属性返回节点的类型。nodeType 是只读的。

比较重要的节点类型有：

|  |  |
| --- | --- |
| **元素类型** | **NodeType** |
| 元素 | 1 |
| 属性 | 2 |
| 文本 | 3 |
| 注释 | 8 |
| 文档 | 9 |

## 访问 HTML 元素（节点）

访问 HTML 元素等同于访问节点

您能够以不同的方式来访问 HTML 元素：

* 通过使用 getElementById() 方法
* 通过使用 getElementsByTagName() 方法
* 通过使用 getElementsByClassName() 方法

## getElementById() 方法

getElementById() 方法返回带有指定 ID 的元素：

### 语法

node.getElementById("id");

下面的例子获取 id="intro" 的元素：

### 实例

document.getElementById("intro");

[亲自试一试](http://www.w3school.com.cn/tiy/t.asp?f=dom_getelementbyid)

## getElementsByTagName() 方法

getElementsByTagName() 返回带有指定标签名的所有元素。

### 语法

node.getElementsByTagName("tagname");

下面的例子返回包含文档中所有 <p> 元素的列表：

### 实例 1

document.getElementsByTagName("p");

<p>Hello World!</p>

<p>DOM 很有用！</p>

<p>本例演示 <b>getElementsByTagName</b> 方法。</p>

<script>

x=document.getElementsByTagName("p");

document.write("第一段的文本: " + x[0].innerHTML); //多个标签等同于数组

</script>

下面的例子返回包含文档中所有 <p> 元素的列表，并且这些 <p> 元素应该是 id="main" 的元素的后代（子、孙等等）：

### 实例 2

document.getElementById("main").getElementsByTagName("p");

## getElementsByClassName() 方法

如果您希望查找带有相同类名的所有 HTML 元素，请使用这个方法：

document.getElementsByClassName("intro");

上面的例子返回包含 class="intro" 的所有元素的一个列表：

**注释：**getElementsByClassName() 在 Internet Explorer 5,6,7,8 中无效。

修改 HTML = 改变元素、属性、样式和事件

## 创建 HTML 内容

改变元素内容的最简答的方法是使用 innerHTML 属性。

下面的例子改变一个 <p> 元素的 HTML 内容：

### 实例

<html>

<body>

<p id="p1">Hello World!</p>

<script>

document.getElementById("p1").innerHTML="New text!";

</script>

</body>

</html>

## 改变 HTML 样式

通过 HTML DOM，您能够访问 HTML 元素的样式对象。

下面的例子改变一个段落的 HTML 样式：

### 实例

<html>

<body>

<p id="p2">Hello world!</p>

<script>

document.getElementById("p2").style.color="blue";

</script>

</body>

</html>

## 创建新的 HTML 元素

如需向 HTML DOM 添加新元素，您首先必须创建该元素（元素节点），然后把它追加到已有的元素上。

### 实例

<div id="d1">

<p id="p1">This is a paragraph.</p>

<p id="p2">This is another paragraph.</p>

</div>

<script>

var para=document.createElement("p");

var node=document.createTextNode("This is new.");

para.appendChild(node);

var element=document.getElementById("d1");

element.appendChild(para);

</script>

## 创建新的 HTML 元素 - insertBefore()

上一个例子中的 appendChild() 方法，将新元素作为父元素的最后一个子元素进行添加。

如果不希望如此，您可以使用 insertBefore() 方法：

### 实例

<div id="div1">

<p id="p1">This is a paragraph.</p>

<p id="p2">This is another paragraph.</p>

</div>

<script>

var para=document.createElement("p");

var node=document.createTextNode("This is new.");

para.appendChild(node);

var element=document.getElementById("div1");

var child=document.getElementById("p1");

element.insertBefore(para,child);

</script>

放到第一个

## 删除已有的 HTML 元素

如需删除 HTML 元素，您必须清楚该元素的父元素：

### 实例

<div id="div1">

<p id="p1">This is a paragraph.</p>

<p id="p2">This is another paragraph.</p>

</div>

<script>

var parent=document.getElementById("div1");

var child=document.getElementById("p1");

parent.removeChild(child);

</script>

如需替换 HTML DOM 中的元素，请使用 replaceChild() 方法：

### 实例

<div id="div1">

<p id="p1">This is a paragraph.</p>

<p id="p2">This is another paragraph.</p>

</div>

<script>

var para=document.createElement("p");

var node=document.createTextNode("This is new.");

para.appendChild(node);

var parent=document.getElementById("div1");

var child=document.getElementById("p1");

parent.replaceChild(para,child);

</script>

## 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 元素分配一个 onclick 事件：

<button onclick="displayDate()">试一试</button>

[亲自试一试](http://www.w3school.com.cn/tiy/t.asp?f=dom_event)

在上面的例子中，当点击按钮时，会执行名为 displayDate 的函数。

## 使用 HTML DOM 来分配事件

HTML DOM 允许您使用 JavaScript 向 HTML 元素分配事件：

### 实例

为 button 元素分配 onclick 事件：

<script>

document.getElementById("myBtn").onclick=function(){displayDate()};

</script>

## onchange 事件

onchange 事件常用于输入字段的验证。

下面的例子展示了如何使用 onchange。当用户改变输入字段的内容时，将调用 upperCase() 函数。

### 实例

<input type="text" id="fname" onchange="upperCase()">

## onmouseover 和 onmouseout 事件

onmouseover 和 onmouseout 事件可用于在鼠标指针移动到或离开元素时触发函数。

<div

onmouseover="mOver(this)"

onmouseout="mOut(this)"

style="background-color:#D94A38;width:200px;height:50px;padding-top:25px;text-align:center;">

Mouse Over Me

</div>

<script>

function mOver(obj)

{

obj.innerHTML="谢谢你"

}

function mOut(obj)

{

obj.innerHTML="把鼠标指针移动到上面"

}

</script>

## onmousedown、onmouseup 以及 onclick 事件

onmousedown、onmouseup 以及 onclick 事件是鼠标点击的全部过程。首先当某个鼠标按钮被点击时，触发 onmousedown 事件，然后，当鼠标按钮被松开时，会触发 onmouseup 事件，最后，当鼠标点击完成时，触发 onclick 事件。

### 实例

一个简单的 onmousedown-onmouseup 实例：

<div

onmousedown="mDown(this)"

onmouseup="mUp(this)"

style="background-color:#D94A38;width:200px;height:50px;padding-top:25px;text-align:center;">

点击这里

</div>

<script>

function mDown(obj)

{

obj.style.backgroundColor="#1ec5e5";

obj.innerHTML="松开鼠标"

}

function mUp(obj)

{

obj.style.backgroundColor="#D94A38";

obj.innerHTML="谢谢你"

}

</script>

## DOM 根节点

这里有两个特殊的属性，可以访问全部文档：

* document.documentElement - 全部文档
* document.body - 文档的主体

## childNodes 和 nodeValue

除了 innerHTML 属性，您也可以使用 childNodes 和 nodeValue 属性来获取元素的内容。

下面的代码获取 id="intro" 的 <p> 元素的值：

### 实例

<html>

<body>

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

<script>

var txt=document.getElementById("intro").childNodes[0].nodeValue;

document.write(txt);

</script>

</body>

</html>