## 光标聚焦事件（onfocus，onblur）

当网页中的对象获得聚点时，执行onfocus调用的程序就会被执行。

如下代码, 当将光标移到文本框内时，即焦点在文本框内，触发onfocus 事件，并调用函数message()。



## 文本框内容改变事件（onchange）

通过改变文本框的内容来触发onchange事件，同时执行被调用的程序。

## 内容选中事件（onselect）

选中事件，当文本框或者文本域中的文字被选中时，触发onselect事件，同时调用的程序就会被执行。

## 加载事件（onload）

事件会在页面加载完成后立即发生，同时执行被调用的程序。  
注意：1. 加载页面时，触发onload事件，事件写在<body>标签内。

      2. 此节的加载页面，可理解为打开一个新页面时。  
如下代码,当加载一个新页面时，弹出对话框“加载中，请稍等…”。

## 卸载事件（onunload）

当用户退出页面时（页面关闭、页面刷新等），触发onUnload事件，同时执行被调用的程序。

**注意：不同浏览器对onunload事件支持不同。**

如下代码,当退出页面时，弹出对话框“您确定离开该网页吗？

<script type="text/javascript">

window.onunload = onunload\_message;

function onunload\_message(){

alert("您确定离开该网页吗？");

}

</script>

## 

## window对象

window对象是BOM的核心，window对象指当前的浏览器窗口。

## JavaScript 计时器

在JavaScript中，我们可以在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。  
**计时器类型：**  
一次性计时器：仅在指定的延迟时间之后触发一次。  
间隔性触发计时器：每隔一定的时间间隔就触发一次。  
**计时器方法：**



## 钟表功能

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

## var attime;

## function clock(){

## var time=new Date();

## attime=time.getHours()+":"+time.getMinutes()+":"+time.getSeconds();

## document.getElementById("clock").value = attime;

## }

## setInterval(clock,100);//每隔一秒刷新一下时间

## </script>

## </head>

## <body>

## <form>

## <input type="text" id="clock" size="50" />

## </form>

## 

## History 对象

history对象记录了用户曾经浏览过的页面(URL)，并可以实现浏览器前进与后退相似导航的功能。

**注意:从窗口被打开的那一刻开始记录，每个浏览器窗口、每个标签页乃至每个框架，都有自己的history对象与特定的window对象关联。**

**语法：**

****

var HL = window.history.length;

## 返回前一个浏览的页面

## 

## 

## window.alert(location.href);//当前的URL

## Navigator对象

Navigator 对象包含有关浏览器的信息，通常用于检测浏览器与操作系统的版本。

查看浏览器的名称和版本，**代码如下:**

<script type="text/javascript">

var browser=navigator.appName;

var b\_version=navigator.appVersion;

document.write("Browser name"+browser);

document.write("<br>");

document.write("Browser version"+b\_version);

</script>

## 

## userAgent

返回用户代理头的字符串表示(就是包括浏览器版本信息等的字符串)

**语法**

navigator.userAgent

## 

## 

## 认识DOM

文档对象模型DOM（Document Object Model）定义访问和处理HTML文档的标准方法。DOM 将HTML文档呈现为带有元素、属性和文本的树结构（节点树）。

## 

## 

**HTML文档可以说由节点构成的集合，DOM节点有:**

**1. 元素节点：**上图中<html>、<body>、<p>等都是元素节点，即标签。

**2. 文本节点:**向用户展示的内容，如<li>...</li>中的JavaScript、DOM、CSS等文本。

**3. 属性节点:**元素属性，如<a>标签的链接属性href="http://www.imooc.com"。

## 节点属性:

## 

## 遍历节点树:

## 

## DOM操作:

## 

## getElementsByName()方法

返回带有指定名称的节点对象的集合。

**语法：**

document.getElementsByName(name)

与getElementById() 方法不同的是，通过元素的 name 属性查询元素，而不是通过 id 属性。

**注意:**

1. 因为文档中的 name 属性可能不唯一，所有 getElementsByName() 方法返回的是元素的数组，而不是一个元素。

2. 和数组类似也有length属性，可以和访问数组一样的方法来访问，从0开始。

## getElementsByTagName()方法

返回带有指定标签名的节点对象的集合。返回元素的顺序是它们在文档中的顺序。

**语法:**

getElementsByTagName(Tagname)

**说明:**

1. Tagname是标签的名称，如p、a、img等标签名。

2. 和数组类似也有length属性，可以和访问数组一样的方法来访问，所以从0开始。

## 区别getElementByID,getElementsByName,getElementsByTagName

**以人来举例说明，人有能标识身份的身份证，有姓名，有类别(大人、小孩、老人)等。**

1. ID 是一个人的身份证号码，是唯一的。所以通过getElementById获取的是指定的一个人。

2. Name 是他的名字，可以重复。所以通过getElementsByName获取名字相同的人集合。

3. TagName可看似某类，getElementsByTagName获取相同类的人集合。如获取小孩这类人，getElementsByTagName("小孩")。

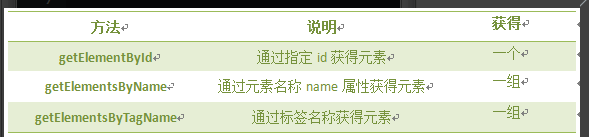
**把上面的例子转换到HTML中，如下:**

**<input type="checkbox" name="hobby" id="hobby1"> 音乐**

input标签就像人的类别。

name属性就像人的姓名。

id属性就像人的身份证。



## getAttribute()方法

通过元素节点的属性名称获取属性的值。

获取标签内的值

## setAttribute()方法

setAttribute() 方法增加一个指定名称和值的新属性，或者把一个现有的属性设定为指定的值。

**语法：**

elementNode.setAttribute(name,value)

**说明：**

1.name: 要设置的属性名。

2.value: 要设置的属性值。

**注意：**

1.把指定的属性设置为指定的值。如果不存在具有指定名称的属性，该方法将创建一个新属性。

2.类似于getAttribute()方法，setAttribute()方法只能通过元素节点对象调用的函数。

## 访问子结点childNodes

访问选定元素节点下的所有子节点的列表，返回的值可以看作是一个数组，他具有length属性。

**运行结果:**

**IE:**

UL子节点个数:3

节点类型:1

**其它浏览器:**

UL子节点个数:7

节点类型:3

## 

## 访问子结点的第一和最后项

一、firstChild 属性返回‘childNodes’数组的第一个子节点。如果选定的节点没有子节点，则该属性返回 NULL。

**语法：**

node.firstChild

**说明：**与elementNode.childNodes[0]是同样的效果。

二、 lastChild 属性返回‘childNodes’数组的最后一个子节点。如果选定的节点没有子节点，则该属性返回 NULL。

**语法：**

node.lastChild

**说明：**与elementNode.childNodes[elementNode.childNodes.length-1]是同样的效果。

## 访问父节点parentNode

获取指定节点的父节点

**语法：**

elementNode.parentNode

**注意:父节点只能有一个。**

## 插入节点appendChild()

在指定节点的最后一个子节点列表之后添加一个新的子节点。

**语法:**

appendChild(newnode)

**参数:**

newnode：指定追加的节点。

## 我们来看看，div标签内创建一个新的 P 标签，代码如下:

## 

**运行结果:**

HTML

JavaScript

This is a new p

## 插入节点insertBefore()

insertBefore() 方法可在已有的子节点前插入一个新的子节点。

**语法:**

insertBefore(newnode,node);

**参数:**

newnode: 要插入的新节点。

node: 指定此节点前插入节点。

## 删除节点removeChild()

removeChild() 方法从子节点列表中删除某个节点。如删除成功，此方法可返回被删除的节点，如失败，则返回 NULL。

**语法:**

nodeObject.removeChild(node)

**参数:**

node ：必需，指定需要删除的节点。

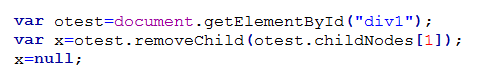
**运行结果:**

HTML

删除节点的内容: javascript

**注意:** 把删除的子节点赋值给 x，这个子节点不在DOM树中，但是还存在内存中，可通过 x 操作。

如果要完全删除对象，给 x 赋 null 值，代码如下:

[](http://img.mukewang.com/539975a800017c8e04790082.jpg)

## 替换元素节点replaceChild()

replaceChild 实现子节点(对象)的替换。返回被替换对象的引用。

**语法：**

node.replaceChild (newnode,oldnew )

**参数：**

newnode : 必需，用于替换 oldnew 的对象。   
oldnew : 必需，被 newnode 替换的对象。

## 创建元素节点createElement

createElement()方法可创建元素节点。此方法可返回一个 Element 对象。

**语法：**

document.createElement(tagName)

**参数:**

tagName：字符串值，这个字符串用来指明创建元素的类型。

**注意：要与appendChild() 或 insertBefore()方法联合使用，将元素显示在页面中。**

**我们来**创建一个按钮**，代码如下：**

<script type="text/javascript">

var body = document.body;

var input = document.createElement("input");

input.type = "button";

input.value = "创建一个按钮";

body.appendChild(input);

</script>

效果：在HTML文档中，创建一个按钮。

我们也可以使用setAttribute来设置属性，代码如下：

<script type="text/javascript">

var body= document.body;

var btn = document.createElement("input");

btn.setAttribute("type", "text");

btn.setAttribute("name", "q");

btn.setAttribute("value", "使用setAttribute");

btn.setAttribute("onclick", "javascript:alert('This is a text!');");

body.appendChild(btn);

</script>

## 创建文本节点createTextNode

createTextNode() 方法创建新的文本节点，返回新创建的 Text 节点。

**语法：**

document.createTextNode(data)

**参数：**

data : 字符串值，可规定此节点的文本。

我们来创建一个<div>元素并向其中添加一条消息，代码如下：

## 

## 

## 在代码编辑器<script>标签中，创建一个P标签，设置className属性，使用createTextNode创建文本节点"I love JavaScript!"。

<script type="text/javascript">

   var element = document.createElement("p");  
   element.className = "message";  
   var textNode = document.createTextNode("I love JavaScript!");  
   element.appendChild(textNode);  
   document.body.appendChild(element);  
          
</script>

## 浏览器窗口可视区域大小

获得浏览器窗口的尺寸（浏览器的视口，不包括工具栏和滚动条）的方法:

**一、对于IE9+、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

**二、对于 Internet Explorer 8、7、6、5：**

•  document.documentElement.clientHeight表示HTML文档所在窗口的当前高度。

•  document.documentElement.clientWidth表示HTML文档所在窗口的当前宽度。

或者

Document对象的body属性对应HTML文档的<body>标签

•  document.body.clientHeight

•  document.body.clientWidth

## **在不同浏览器都实用的**窗口可视区域大小 **JavaScript 方案：**

var w= document.documentElement.clientWidth

|| document.body.clientWidth;

var h= document.documentElement.clientHeight

|| document.body.clientHeight;

## 网页尺寸scrollHeight

scrollHeight和scrollWidth，获取网页内容高度和宽度(不包括滚动条)。

**一、针对IE、Opera:**

scrollHeight 是网页内容实际高度，可以小于 clientHeight。

**二、针对NS、FF:**

scrollHeight 是网页内容高度，不过最小值是 clientHeight。也就是说网页内容实际高度小于 clientHeight 时，scrollHeight 返回 clientHeight

**三、**scrollHeight浏览器兼容性写法

var w=document.documentElement.scrollWidth

|| document.body.scrollWidth;

var h=document.documentElement.scrollHeight

|| document.body.scrollHeight;

## 网页尺寸offsetHeight

offsetHeight和offsetWidth，获取网页内容高度和宽度(包括滚动条等边线，会随窗口的显示大小改变)。

**一、值**

offsetHeight = clientHeight + 滚动条 + 边框。

## **二、**网页尺寸offsetHeigh浏览器兼容性

var w= document.documentElement.offsetWidth

|| document.body.offsetWidth;

var h= document.documentElement.offsetHeight

|| document.body.offsetHeight;

## 网页卷去的距离与偏移量

## 

## 区别在于 是否包含滚动条。

**scrollLeft:**设置或获取位于给定对象左边界与窗口中目前可见内容的最左端之间的距离 ，即左边灰色的内容。

**scrollTop:**设置或获取位于对象最顶端与窗口中可见内容的最顶端之间的距离 ，即上边灰色的内容。

**offsetLeft:**获取指定对象相对于版面或由 offsetParent 属性指定的父坐标的计算左侧位置 。

**offsetTop:**获取指定对象相对于版面或由 offsetParent 属性指定的父坐标的计算顶端位置 。

**注意:**

**1. 区分大小写**

**2. offsetParent：布局中设置postion属性(Relative、Absolute、fixed)的父容器，从最近的父节点开始，一层层向上找，直到HTML的body。**