事件响应

- 鼠标单击事件( `onclick` )
- 鼠标经过事件 ( `onmouseover` )
- 光标聚焦事件 ( `onfocus` )
- 失焦事件 ( `onblur` )
- 内容选中事件 ( `onselect` )
- 文本框内容改变事件 ( `onchange` )
- 加载事件 ( `onload` )
- 卸载事件 ( `onunload` )

对象

- `Date` 日期对象
- `String` 字符串对象
  - `charAt()` 返回指定位置的字符
  - `indexOf()` 返回指定的字符串首次出现的位置
  - `split()` 字符串分割
  - `substring()` 提取字符串
  - `substr()` 提取指定数目的字符
- `Math` 对象
  - `ceil()` 向上取整
  - `floor()` 向下取整
  - `round()` 四舍五入
  - `random()` 随机数
- `Array` 数组对象
  - `concat()` 数组连接
  - `join()` 指定分隔符连接数组元素
  - `reverse()` 颠倒数组元素顺序
  - `slice()` 选定元素
  - `sort()` 数组排序

浏览器对象

- windows对象
- JavaScript计时器
  - `setInterval()` 计时器
  - `clearInterval()` 取消计时器
  - `setTimeout()` 计时器
  - `clearTimeout()` 取消计时器
- history对象
  - `back()` 返回前一个浏览的页面
  - `forward()` 返回下一个浏览的页面
  - `go()` 返回浏览历史中的其他页面
- location对象
- navigator对象
- screen对象
  - `screen.height` / `screen.width` 屏幕分辨率的高和宽
  - `screen.availHeight` / `screen.availWidth` 屏幕可用高和宽度

认识DOM

- `getElementsByName()` 返回带有指定名称的节点对象的集合
- `getElementsByTagName()` 返回带有指定标签名的节点对象的集合
- 区别 `getElementByID` , `getElementsByName` , `getElementsByTagName`
- `getAttribute()` 通过元素节点的属性名称获取属性的值。
- `setAttribute()` 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。

节点属性

访问子节点 `childNodes`

访问子节点的第一 `firstChild` 和最后 `lastChild` 项

访问父节点 `parentNode`

`nextSibling` / `previousSibling` 访问兄弟节点

`appendChild()` 插入节点

`insertBefore()` 插入节点

`removeChild()` 删除节点

`replaceChild()` 替换元素节点

`createElement()` 创建元素节点

`createTextNode()` 创建文本节点

浏览器窗口可视区域大小

`scrollHeight` / `scrollWidth` 获取网页内容高度和宽度

`offsetHeight` / `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)。

网页卷去的距离与偏移量

## 事件响应

JavaScript 创建动态页面。事件是可以被 JavaScript 侦测到的行为。 网页中的每个元素都可以产生某些可以触发 JavaScript 函数或程序的事件。

比如说，当用户单击按钮或者提交表单数据时，就发生一个鼠标单击（ `onclick` ）事件，需要浏览器做出处理，返回给用户一个结果。

主要事件表:

| 事件 | 说明 |
| --- | --- |
| onclick | 鼠标单击事件 |
| onmouseover | 鼠标经过事件 |
| onmouseout | 鼠标移开事件 |
| onchange | 文本框内容改变事件 |
| onselect | 文本框内容被选中事件 |
| onfocus | 光标聚集 |
| onblur | 光标离开 |
| onload | 网页导入 |
| onunload | 关闭网页 |

### 鼠标单击事件( `onclick` )

`onclick` 是鼠标单击事件，当在网页上单击鼠标时，就会发生该事件。同时 `onclick` 事件调用的程序块就会被执行，通常与按钮一起使用。

比如，我们单击按钮时，触发 `onclick` 事件，并调用两个数和的函数 `add2()` 。代码如下：

```
<html>
<head>
   <script type="text/javascript">
      function add2(){
        var numa,numb,sum;
        numa=6;
        numb=8;
        sum=numa+numb;
        document.write("两数和为:"+sum);  }
   </script>
</head>
<body>
   <form>
      <input name="button" type="button" value="点击提交" onclick="add2()" />
   </form>
</body>
</html>
```

注意: 在网页中，如使用事件，就在该元素中设置事件属性。

## 鼠标经过事件（`onmouseover`）

鼠标经过事件，当鼠标移到一个对象上时，该对象就触发 `onmouseover` 事件，并执行 `onmouseover` 事件调用的程序。

现实鼠标经过"确定"按钮时，触发 `onmouseover` 事件，调用函数 `info()` ，弹出消息框，代码如下:

```
<!DOCTYPE HTML>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<title> onmouseout </title>
<script type="text/javascript">
  function info(){
    confirm("请输入姓名后，再单击确定!"); }
</script>
</head>
<body>
  <form>
   密码:<input name="password" type="text">
   <input name="button" type="button" value="确定" onmouseover="info()" >
   <!--当鼠标经过"确定"按钮时，触发onmouseover="info()"-->
  </form>
</body>
</html>
```

## 鼠标移开事件（`onmouseout`）

鼠标移开事件，当鼠标移开当前对象时，执行 `onmouseout` 调用的程序。

当把鼠标移动到"登录"按钮上，然后再移开时，触发 `onmouseout` 事件，调用函数 `message()` ，代码如下:

```
<!DOCTYPE HTML>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<title> onmouseout </title>
<script type="text/javascript">
  function message(){
    confirm("不要离开，只要输入密码，再单击登录，就ok了！"); }
</script>
</head>
<body>
  <form>
   密码:<input name="password" type="text">
   <input name="button" type="button" value="登录" onmouseout="message()" >
   <!--当移开"登录"按钮，触onmouseout="message()"-->
  </form>
</body>
</html>
```

## 光标聚焦事件（onfocus）

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() 。

```
<!DOCTYPE HTML>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<title> onfocus </title>
<script type="text/javascript">
  function message(){
    alert("请在此输入姓名！"); }
</script>
</head>
<body>
  <form>
   姓名:<input name="username" type="text" value="请输入姓名！ " onfocus="message()" >
   <!--当光标在文本框内时(即文本框得到焦点)，调用message()函数-->
  </form>
</body>
</html>
```

## 失焦事件（onblur）

onblur 事件与 onfocus 是相对事件，当光标离开当前获得聚焦对象的时候，触发 onblur 事件，同时执行被调用的程序。

如下代码，网页中有用户和密码两个文本框。当前光标在用户文本框内时（即焦点在文本框），在光标离开该文本框后（即失焦时），触发 onblur 事件，并调用函数 message() 。

```
<!DOCTYPE HTML>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<title> 失焦事件 </title>
<script type="text/javascript">
  function message(){
    alert("请确定已输入用户名后，在离开！"); }
</script>
</head>
<body>
  <form>
   用户:<input name="username" type="text" value="请输入用户名！ " onblur="message()" >
   密码:<input name="password" type="text" value="请输入密码！ " >
  </form>
</body>
</html>
```

## 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。

如下代码,当选中用户文本框内的文字时，触发 `onselect` 事件，并调用函数 `message()` 。

```
<!DOCTYPE HTML>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<title> onselect </title>
<script type="text/javascript">
  function message(){
    alert("您触发了选中事件！"); }
</script>
</head>
<body>
  <form>
   用户：<input name="username" type="text" value="请输入用户名!" onselect="message()">
     <!--当选中用户文本框内的文字时，触发onselect事件。-->
  </form>
</body>
</html>
```

## 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。

如下代码,当用户将文本框内的文字改变后，弹出对话框“您改变了文本内容！”。

```
<!DOCTYPE HTML>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<title> onChange </title>
<script type="text/javascript">
  function message(){
    alert("您改变了文本内容！"); }
</script>
</head>
<body>
  <form>
   用户:<input name="username" type="text" value="请输入用户名！" onChange ="message()">
       <!--当改变用户文本框内的文字后，触发onChange事件 。-->
  </form>
</body>
</html>
```

## 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：

加载页面时，触发 `onload` 事件，事件写在 `<body>` 标签内。

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...”。

```
<!DOCTYPE HTML>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<title> onLoad </title>
<script type="text/javascript">
  function message(){
    alert("加载中，请稍等…"); }
</script>
</head>
<body onLoad ="message()">
  欢迎学习JavaScript。
</body>
</html>
```

### 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？”。

```
<!DOCTYPE HTML>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<title> onUnload </title>
<script type="text/javascript">
     window.onunload = onunload_message;
     function onunload_message(){
        alert("您确定离开该网页吗？");
    }
</script>
</head>
<body>
  欢迎学习JavaScript。
</body>
</html>
```

## 对象

JavaScript 中的所有事物都是对象，如:字符串、数值、数组、函数等，每个对象带有属性和方法。

**对象的属性**：反映该对象某些特定的性质的，如：字符串的长度、图像的长宽等；

**对象的方法**：能够在对象上执行的动作。例如，表单的 *提交* (Submit)，时间的 *获取* (getYear)等；

JavaScript 提供多个内建对象，比如 String、Date、Array 等等，使用对象前先定义

如使用string 对象的 toUpperCase() 方法来将文本转换为大写：

```
1   var mystr="Hello world!";//创建一个字符串
2   var request=mystr.toUpperCase(); //使用字符串对象方法
```

以上代码执行后，request的值是： HELLO WORLD!

### Date 日期对象

日期对象可以储存任意一个日期，并且可以精确到毫秒数（1/1000 秒）。

定义一个时间对象：

```
var Udate=new Date();
```

注意:使用关键字 `new` ， `Date()` 的首字母必须大写。

使 Udate 成为日期对象，并且已有初始值：**当前时间(当前电脑系统时间)。**

如果要自定义初始值，可以用以下方法：

```
var d = new Date(2012, 10, 1);  //2012年10月1日
var d = new Date('Oct 1, 2012'); //2012年10月1日
```

我们最好使用下面介绍的method来严格定义时间。

访问方法语法： `<日期对象>.<方法>`

Date对象中处理时间和日期的常用方法：

| 方法名称 | 功能描述 |
| --- | --- |
| get/setDate() | 返回/设置日期 |
| get/setFullYear() | 返回/设置年份，用四位数表示 |
| get/setYear() | 返回/设置年份 |
| get/setMonth() | 返回/设置月份。<br>0:一月...11:十二月，所以加一。 |
| get/setHours() | 返回/设置小时，24 小时制。 |
| get/setMinutes() | 返回/设置分钟数 |
| get/setSeconds() | 返回/设置秒钟数 |
| get/setTime() | 返回/设置时间（毫秒为单位） |

`getDay()` 返回星期

## `String` 字符串对象

在之前的学习中已经使用字符串对象了，定义字符串的方法就是直接赋值。比如：

```
var mystr = "I love JavaScript!"
```

定义mystr字符串后，我们就可以访问它的属性和方法。

**访问字符串对象的属性length:** `var myl=mystr.length;`

**访问字符串对象的方法：** `var mynum=mystr.toUpperCase();`

### `charAt()` 返回指定位置的字符

**语法:**

```
stringObject.charAt(index)
```

**参数说明:**

| 参数 | 描述 |
| --- | --- |
| **index** | 必需。表示字符串中某个位置的数字，即字符在字符串中的下标。 |

注意：

1. 字符串中第一个字符的下标是 0。最后一个字符的下标为字符串长度减一（string.length-1）。
2. 如果参数 index 不在 0 与 string.length-1 之间，该方法将返回一个空字符串。
3. 一个空格也算一个字符。

### `indexOf()` 返回指定的字符串首次出现的位置

**语法**

```
stringObject.indexOf(substring, startpos)
```

**参数说明：**

| 参数 | 描述 |
| --- | --- |
| substring | 必需。规定需检索的字符串值。 |
| startpos | 可选的整数参数。规定在字符串中开始检索的位置。它的合法取值是 0 到 stringObject.length - 1。如省略该参数，则将从字符串的首字符开始检索。 |

说明：

1. 该方法将从头到尾地检索字符串 stringObject，看它是否含有子串 substring。
2. 可选参数，从stringObject的startpos位置开始查找substring，如果没有此参数将从stringObject的开始位置查找。
3. 如果找到一个 substring，则返回 substring 的第一次出现的位置。stringObject 中的字符位置是从 0 开始的。

注意：

1. `indexOf()` 方法**区分大小写**。
2. 如果要检索的字符串值没有出现，则该方法返回 `-1` 。

### `split()` 字符串分割

**语法**

```
stringObject.split(separator,limit)
```

**参数说明：**

| 参数 | 描述 |
| --- | --- |
| separator | 必需。从该参数指定的地方分割 stringObject。 |
| limit | 可选参数，分割的次数，如设置该参数，返回的子串不会多于这个参数指定的数组，如果无此参数为不限制次数 |

**注意：如果把空字符串（`""`）用作 separator，那么 stringObject 中的每个字符之间都会被分割。**

### `substring()` 提取字符串

`substring()` 方法用于提取字符串中介于两个指定下标之间的字符。

**语法**

```
stringObject.substring(startPos,stopPos)
```

**参数说明：**

| 参数 | 描述 |
| --- | --- |
| **startPos** | 必需。一个非负的整数，开始位置。 |
| **stopPos** | 可选。一个非负的整数，结束位置，如果省略该参数，那么返回的子串会一直到字符串对象的结尾。 |

注意：

1. 返回的内容是从 start开始(**包含start位置的字符**)到 stop-1 处的所有字符，其长度为 stop 减 start。
2. 如果参数 start 与 stop 相等，那么该方法返回的就是一个空串（即长度为 0 的字符串）。
3. 如果 start 比 stop 大，那么该方法在提取子串之前会先交换这两个参数。

### `substr()` 提取指定数目的字符

`substr()` 方法从字符串中提取从 startPos位置开始的指定数目的字符串。

**语法**

```
stringObject.substr(startPos,length)
```

**参数说明：**

| 参数 | 描述 |
| --- | --- |
| ***startPos*** | 必需。要提取的子串的起始位置。必须是数值。 |
| ***length*** | 可选。提取字符串的长度。如果省略，返回从 stringObject 的开始位置 *startPos* 到 stringObject 的结尾的字符。 |

注意：

1. 如果参数startPos是负数，从字符串的尾部开始算起的位置。也就是说，-1 指字符串中最后一个字符，-2 指倒数第二个字符，以此类推。
2. 如果startPos为负数且绝对值大于字符串长度，startPos为0。

## `Math` 对象

使用 `Math` 的属性和方法，代码如下：

```
<script type="text/javascript">
  var mypi=Math.PI;
  var myabs=Math.abs(-15);
  document.write(mypi);
  document.write(myabs);
</script>
```

运行结果：

```
3.141592653589793
15
```

注意：**`Math` 对象是一个固有的对象，无需创建它，直接把 `Math` 作为对象使用就可以调用其所有属性和方法。这是它与Date,String对象的区别。**

**Math 对象属性**

| 属性 | 说明 |
| --- | --- |
| E | 返回算术常里 e，即自然对数的底数（约等于 2.718）。 |
| LN2 | 返回 2 的自然对数（约等于 0.693）。 |
| LN10 | 返回 10 的自然对数（约等于 2.302）。 |
| LOG2E | 返回以 2 为底的 e 的对数（约等于 1.442）。 |
| LOG10E | 返回以 10 为底的 e 的对数（约等于 0.434）。 |
| PI | 返回圆周率（约等于 3.14159）。 |
| SQRT1_2 | 返回返回 2 的平方根的倒数（约等于 0.707）。 |
| SQRT2 | 返回 2 的平方根（约等于 1.414）。 |

**Math 对象方法**

| 方法 | 描述 |
| --- | --- |
| abs(x) | 返回数的绝对值。 |
| acos(x) | 返回数的反余弦值。 |
| asin(x) | 返回数的反正弦值。 |
| atan(x) | 返回数字的反正切值 |
| atan2(y,x) | 返回由 x 轴到点(x,y)的角度(以弧度为单位) |
| ceil(x) | 对数进行上舍入。 |
| cos(x) | 返回数的余弦。 |
| exp(x) | 返回 e 的指数。 |
| floor(x) | 对数进行下舍入。 |
| log(x) | 返回数的自然对数（底为e）。 |
| max(x,y) | 返回 x 和 y 中的最高值。 |
| min(x,y) | 返回 x 和 y 中的最低值。 |
| pow(x,y) | 返回 x 的 y 次幂。 |
| random() | 返回 0～1 之间的随机数。 |
| round(x) | 把数四舍五入为最接近的整数。 |
| sin(x) | 返回数的正弦。 |
| sqrt(x) | 返回数的平方根。 |
| tan(x) | 返回角的正切。 |
| toSource() | 返回该对象的源代码。 |
| valueOf() | 返回 Math 对象的原始值。 |

### `ceil()` 向上取整

`ceil()` 方法可对一个数进行向上取整。

**语法:**

```
Math.ceil(x)
```

**参数说明:**

| 参数 | 描述 |
| --- | --- |
| x | 必需。必须是一个数值。 |

注意：它返回的是**大于或等于x，并且与x最接近的整数。**

### `floor()` 向下取整

`floor()` 方法可对一个数进行向下取整。

**语法:**

```
Math.floor(x)
```

**参数说明:**

| 参数 | 描述 |
| --- | --- |
| x | 必需。必须是一个数值。 |

注意：返回的是**小于或等于x，并且与 x 最接近的整数。**

### `round()` 四舍五入

`round()` 方法可把一个数字四舍五入为最接近的整数。

**语法:**

```
Math.round(x)
```

**参数说明:**

| 参数 | 描述 |
| --- | --- |
| x | 必需。必须是数字。 |

注意：

1. 返回与 x 最接近的整数。
2. 对于 0.5，该方法将进行上舍入。(5.5 将舍入为 6)
3. **如果 x 与两侧整数同等接近，则结果接近 +∞方向的数字值 。(如 -5.5 将舍入为 -5; -5.52 将舍入为 -6)**

### `random()` 随机数

`random()` 方法可返回介于 0~1（大于或等于 0 但小于 1 )之间的一个随机数。

**语法:**

```
Math.random();
```

注意：返回一个大于或等于 0 但小于 1 的符号为正的数字值。

## `Array` 数组对象

**数组对象是一个对象的集合，里边的对象可以是不同类型的。**数组的每一个成员对象都有一个“下标”，用来表示它在数组中的位置，是从零开始的 *类似Python的list*

**数组定义的方法：**

1. 定义了一个空数组: `var 数组名= new Array();`
2. 定义时指定有n个空元素的数组： `var 数组名 =new Array(n);`
3. 定义数组的时候，直接初始化数据： `var 数组名 = [<元素1>, <元素2>, <元素3>...];`

我们定义myArray数组，并赋值，代码如下： `var myArray = [2, 8, 6];`

说明：定义了一个数组 myArray，里边的元素是： `myArray[0] = 2; myArray[1] = 8; myArray[2] = 6` 。

**数组元素使用：** `数组名[下标] = 值;`

**注意:** 数组的下标用方括号括起来，从0开始。

**数组属性：**

length 用法： `<数组对象>.length;` 返回：数组的长度，即数组里有多少个元素。它等于数组里最后一个元素的下标加一。

**数组方法：**

| 方法 | 描述 |
| --- | --- |
| concat() | 连接两个或更多的数组，并返回结果。 |
| join() | 把数组的所有元素放入一个字符串。元素通过指定的分隔符进行分隔。 |
| pop() | 删除并返回数组的最后一个元素 |
| push() | 向数组的末尾添加一个或更多元素，并返回新的长度。 |
| reverse() | 颠倒数组中元素的顺序。 |
| shift() | 删除并返回数组的第一个元素 |
| slice() | 从某个已有的数组返回选定的元素 |
| sort() | 对数组的元素进行排序 |
| splice() | 删除元素，并向数组添加新元素。 |
| toSource() | 返回该对象的源代码。 |
| toString() | 把数组转换为字符串，并返回结果。 |
| toLocaleString() | 把数组转换为本地数组，并返回结果。 |
| unshift() | 向数组的开头添加一个或更多元素，并返回新的长度。 |
| valueOf() | 返回数组对象的原始值 |

**`concat()` 数组连接**

`concat()` 方法用于连接两个或多个数组。此方法返回一个新数组，不改变原来的数组。

**语法:**

```
arrayObject.concat(array1,array2,...,arrayN)
```

**参数说明:**

| 参数 | 说明 |
| --- | --- |
| array1 | 要连接的第一个数组。 |
| ... | ... |
| arrayN | 第 N 个数组。 |

**注意: 该方法不会改变现有的数组，而仅仅会返回被连接数组的一个副本。** *`copy`，可以存在另一个变量里*

### `join()` 指定分隔符连接数组元素

`join()` 方法用于把数组中的所有元素放入一个字符串。元素是通过指定的分隔符进行分隔的。

**语法:**

```
arrayObject.join(分隔符)
```

**参数说明:**

| 参数 | 描述 |
| --- | --- |
| separator | 可选。指定要使用的分隔符。如果省略该参数，则使用逗号作为分隔符。 |

注意：返回一个字符串，该字符串把数组中的各个元素串起来，用<分隔符>置于元素与元素之间。**这个方法不影响数组原本的内容。**

### `reverse()` 颠倒数组元素顺序

`reverse()` 方法用于颠倒数组中元素的顺序。

**语法:**

```
arrayObject.reverse()
```

**注意：该方法会改变原来的数组，而不会创建新的数组。**

### `slice()` 选定元素

`slice()` 方法可从已有的数组中返回选定的元素。

**语法:**

```
arrayObject.slice(start,end)
```

**参数说明:**

| 参数 | 描述 |
| --- | --- |
| start | 必需。规定从何处开始选取。如果是负数，那么它规定从数组尾部开始算起的位置。也就是说，-1 指最后一个元素，-2 指倒数第二个元素，以此类推。 |
| end | 可选。规定从何处结束选取。该参数是数组片断结束处的数组下标。如果没有指定该参数，那么切分的数组包含从 start 到数组结束的所有元素。如果这个参数是负数，那么它规定的是从数组尾部开始算起的元素。 |

1. 返回一个新的数组，包含从 start 到 end （不包括该元素）的 arrayObject 中的元素。
2. **该方法并不会修改数组，而是返回一个子数组。**

注意:

1. **可使用负值从数组的尾部选取元素。**
2. 如果 end 未被规定，那么 slice() 方法会选取从 start 到数组结尾的所有元素。
3. String.slice() 与 Array.slice() 相似。

### `sort()` 数组排序

`sort()` 方法使数组中的元素按照一定的顺序排列。

**语法:**

```
arrayObject.sort(方法函数)
```

**参数说明:**

| 参数 | 描述 |
| --- | --- |
| 方法函数 | 可选。规定排序顺序。必须是函数。 |

1. 如果不指定 `<方法函数>` ，**则按unicode码顺序排列。**
2. 如果指定 `<方法函数>` ，则按 `<方法函数>` 所指定的排序方法排序。

注意:

该函数要比较两个值，然后返回一个用于说明这两个值的相对顺序的数字。比较函数应该具有两个参数 a 和 b，其返回值如下:

1. 若返回值<=-1，则表示 A 在排序后的序列中出现在 B 之前。
2. 若返回值>-1 && <1，则表示 A 和 B 具有相同的排序顺序。
3. 若返回值>=1，则表示 A 在排序后的序列中出现在 B 之后。

# 浏览器对象

## windows对象

window对象指当前的浏览器窗口。

**window对象方法:**

| 方法 | 描述 |
| --- | --- |
| alert() | 显示带有一段消息和一个确认按钮的警告框。 |
| prompt() | 显示可提示用户输入的对话框。 |
| confirm() | 显示带有一段消息以及确认按钮和取消按钮的对话框。 |
| open() | 打开一个新的浏览器窗口或查找一个已命名的窗口。 |
| close() | 关闭浏览器窗口。 |
| print() | 打印当前窗口的内容。 |
| focus() | 把键盘焦点给予一个窗口。 |
| blur() | 把键盘焦点从顶层窗口移开。 |
| moveBy() | 可相对窗口的当前坐标把它移动指定的像素。 |
| moveTo() | 把窗口的左上角移动到一个指定的坐标。 |
| resizeBy() | 按照指定的像素调整窗口的大小。 |
| resizeTo() | 把窗口的大小调整到指定的宽度和高度。 |
| scrollBy() | 按照指定的像素值来滚动内容。 |
| scrollTo() | 把内容滚动到指定的坐标。 |
| setInterval() | 每隔指定的时间执行代码。 |
| setTimeout() | 在指定的延迟时间之后来执行代码。 |
| clearInterval() | 取消 setInterval() 的设置。 |
| clearTimeout() | 取消 setTimeout() 的设置。 |

## 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：**

| 方法 | 说明 |
| --- | --- |
| setTimeout() | 指定的延迟时间之后来执行代码 |
| clearTimeout() | 取消 setTimeout()设置。 |
| setInterval() | 每隔指定的时间执行代码 |
| clearInterval() | 取消 setInterval()设置。 |

### `setInterval()` 计时器

在执行时,从载入页面后每隔指定的时间执行代码。

**语法：**

```
setInterval(代码,交互时间);
```

**参数说明：**

1. 代码：要调用的**函数**或要执行的**代码串**。
2. 交互时间：周期性执行或调用表达式之间的时间间隔，**以毫秒计**（1s=1000ms）。

**返回值:**

一个可以传递给 `clearInterval()` 从而取消对 `代码` 的周期性执行的值。

**调用函数格式**(假设有一个 `clock()` 函数):

```
setInterval("clock()",1000) //或setInterval(clock,1000)
```

我们设置一个计时器，每隔100毫秒调用 `clock()` 函数，并将时间显示出来，代码如下:

```
<!DOCTYPE HTML>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<title>计时器</title>
<script type="text/javascript">
  var int=setInterval(clock, 100)
  function clock(){
    var time=new Date();
    document.getElementById("clock").value = time;
  }
</script>
</head>
<body>
  <form>
    <input type="text" id="clock" size="50"  />
  </form>
</body>
</html>
```

### `clearInterval()` 取消计时器

`clearInterval()` 方法可取消由 `setInterval()` 设置的交互时间。

**语法:**

```
clearInterval(id_of_setInterval)
```

**参数说明:** `id_of_setInterval` : 由 `setInterval()` 返回的 ID 值。

每隔 100 毫秒调用 `clock()` 函数,并显示时间。当点击按钮时，停止时间,代码如下:

```
<!DOCTYPE HTML>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<title>计时器</title>
<script type="text/javascript">
   function clock(){
      var time=new Date();
      document.getElementById("clock").value = time;
   }
// 每隔100毫秒调用clock函数，并将返回值赋值给i
     var i=setInterval("clock()",100);
</script>
</head>
<body>
  <form>
    <input type="text" id="clock" size="50"  />
    <input type="button" value="Stop" onclick="clearInterval(i)"  />
```

```
  </form>
</body>
</html>
```

### `setTimeout()` 计时器

`setTimeout()` 计时器，在载入后延迟指定时间后,去执行一次表达式,仅执行一次。

**语法:**

```
setTimeout(代码,延迟时间);
```

**参数说明:**

1. 要调用的函数或要执行的代码串。
2. 延时时间：在执行代码前需等待的时间，**以毫秒为单位**（1s=1000ms)。

当我们打开网页3秒后，在弹出一个提示框，代码如下:

```
<!DOCTYPE HTML>
<html>
<head>
<script type="text/javascript">
  setTimeout("alert('Hello!')", 3000 ); //这是代码串
</script>
</head>
<body>
</body>
</html>
```

当按钮start被点击时，`setTimeout()` 调用函数，在5秒后弹出一个提示框。

```
<!DOCTYPE HTML>
<html>
<head>
<script type="text/javascript">
function tinfo(){
  var t=setTimeout("alert('Hello!')",5000);
 }
</script>
</head>
<body>
<form>
  <input type="button" value="start" onClick="tinfo()">
</form>
</body>
</html>
```

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

```
<!DOCTYPE HTML>
<html>
<head>
<script type="text/javascript">
var num=0;
function numCount(){
 document.getElementById('txt').value=num;
 num=num+1;
```

```
 setTimeout("numCount()",1000); //有点像recursion
 }
</script>
</head>
<body>
<form>
<input type="text" id="txt" />
<input type="button" value="Start" onClick="numCount()" />
</form>
</body>
</html>
```

### `clearTimeout()` 取消计时器

`setTimeout()` 和 `clearTimeout()` 一起使用，停止计时器。

**语法:**

```
clearTimeout(id_of_setTimeout)
```

**参数说明:**

`id_of_setTimeout`：由 `setTimeout()` 返回的 ID 值。该值标识要取消的延迟执行代码块。

下面的例子和上节的无穷循环的例子相似。唯一不同是，现在我们添加了一个 "Stop" 按钮来停止这个计数器：

```
<!DOCTYPE HTML>
<html>
<head>
<script type="text/javascript">
  var num=0,i;
  function timedCount(){
    document.getElementById('txt').value=num;
    num=num+1;
    i=setTimeout(timedCount,1000);
  }
    setTimeout(timedCount,1000); //主动触发
  function stopCount(){
    clearTimeout(i);
  }
</script>
</head>
<body>
  <form>
    <input type="text" id="txt">
    <input type="button" value="Stop" onClick="stopCount()">
  </form>
</body>
</html>
```

## 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:**

```
window.history.[属性|方法]
```

**注意：window可以省略。**

**History 对象属性**

| 属性 | 描述 |
| --- | --- |
| length | 返回浏览器历史列表中的 URL 数量。 |

**History 对象方法**

| 方法 | 描述 |
| --- | --- |
| back() | 加载 history 列表中的前一个 URL。 |
| forward() | 加载 history 列表中的下一个 URL。 |
| go() | 加载 history 列表中的某个具体的页面。 |

使用length属性，当前窗口的浏览历史总长度，代码如下：

```
<script type="text/javascript">
  var HL = window.history.length;
  document.write(HL);
</script>
```

**`back()` 返回前一个浏览的页面**

`back()` 方法，加载 history 列表中的前一个 URL。

**语法：**

```
window.history.back();
```

比如，返回前一个浏览的页面，代码如下：

```
window.history.back();
```

**注意：等同于点击浏览器的倒退按钮。**

`back()` 相当于 `go(-1)` ,代码如下:

```
window.history.go(-1);
```

**`forward()` 返回下一个浏览的页面**

`forward()` 方法，加载 history 列表中的下一个 URL。

如果倒退之后，再想回到倒退之前浏览的页面，则可以使用 `forward()` 方法,代码如下:

```
window.history.forward();
```

**注意：等价点击前进按钮。**

`forward()` 相当于 `go(1)` ,代码如下:

```
window.history.go(1);
```

**`go()` 返回浏览历史中的其他页面**

`go()` 方法，根据当前所处的页面，加载 history 列表中的某个具体的页面。

**语法：**

```
window.history.go(number);
```

**参数：**

| number | 参数说明 |
| --- | --- |
| 1 | 前一个，go(1)等价 forward() |
| 0 | 当前页面 |
| -1 | 后一个，go(-1)等价 back() |
| 其它数值 | 要访问的 URL 在 History 的 URL 列表中的相对位置。 |

浏览器中，返回当前页面之前浏览过的第二个历史页面，代码如下：

```
window.history.go(-2);
```

**注意：和在浏览器中单击两次后退按钮操作一样。**

同理，返回当前页面之后浏览过的第三个历史页面，代码如下：

```
window.history.go(3);
```

## location对象

location用于获取或设置窗体的URL，并且可以用于解析URL。

**语法：**

```
location.[属性|方法]
```

**location对象属性图示：**


**location 对象属性：**

| 属性 | 描述 |
| --- | --- |
| hash | 设置或返回从井号 (#) 开始的 URL（锚）。 |
| host | 设置或返回主机名和当前 URL 的端口号。 |
| hostname | 设置或返回当前 URL 的主机名。 |
| href | 设置或返回完整的 URL。 |
| pathname | 设置或返回当前 URL 的路径部分。 |
| port | 设置或返回当前 URL 的端口号。 |
| protocol | 设置或返回当前 URL 的协议。 |
| search | 设置或返回从问号 (?) 开始的 URL（查询部分）。 |

**location 对象方法：**

| 属性 | 描述 |
| --- | --- |
| assign() | 加载新的文档。 |
| reload() | 重新加载当前文档。 |
| replace() | 用新的文档替换当前文档。 |

## 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。

**对象属性:**

| 属性 | 描述 |
| --- | --- |
| appCodeName | 浏览器代码名的字符串表示 |
| appName | 返回浏览器的名称。 |
| appVersion | 返回浏览器的平台和版本信息。 |
| platform | 返回运行浏览器的操作系统平台。 |
| userAgent | 返回由客户机发送服务器的 user-agent 头部的值。 |

`navigator.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)

几种浏览的user_agent.，像360的兼容模式用的是IE、极速模式用的是chrom的内核。

| 浏览器 | userAgent |
| --- | --- |
| chrome | Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/34.0.1847.116 Safari/537.36 |
| firefox | Mozilla/5.0 (Windows NT 6.1; WOW64; rv:24.0) Gecko/20100101 Firefox/24.0 |
| IE 8 | Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 6.1; WOW64; Trident/4.0; SLCC2; .NET CLR 2.0.50727; .NET CLR 3.5.30729; .NET CLR 3.0.30729; .NET4.0C) |

使用userAgent判断使用的是什么浏览器(假设使用的是IE8浏览器),代码如下:

```
function validB(){
  var u_agent = navigator.userAgent;
  var B_name="Failed to identify the browser";
  if(u_agent.indexOf("Firefox")>-1){
      B_name="Firefox";
  }else if(u_agent.indexOf("Chrome")>-1){
      B_name="Chrome";
  }else if(u_agent.indexOf("MSIE")>-1&&u_agent.indexOf("Trident")>-1){
      B_name="IE(8-10)";
  }
    document.write("B_name:"+B_name+"<br>");
    document.write("u_agent:"+u_agent+"<br>");
}
```

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>
```

## screen对象

screen对象用于获取用户的屏幕信息。

**语法:**

```
window.screen.属性
```

**对象属性:**

| 属性 | 描述 |
| --- | --- |
| availHeight | 窗口可以使用的屏幕高度，单位像素 |
| availWidth | 窗口可以使用的屏幕宽度，单位像素 |
| colorDepth | 用户浏览器表示的颜色位数，通常为 32 位(每像素的位数) |
| pixelDepth | 用户浏览器表示的颜色位数，通常为 32 位(每像素的位数)（IE 不支持此属性） |
| height | 屏幕的高度，单位像素 |
| width | 屏幕的宽度，单位像素 |

### `screen.height` / `screen.width` 屏幕分辨率的高和宽

`window.screen` 对象包含有关用户屏幕的信息。

1. `screen.height` 返回屏幕分辨率的高
2. `screen.width` 返回屏幕分辨率的宽

**注意:**

1. **单位以像素计。**
2. `window.screen` 对象在编写时可以不使用 `window` 这个前缀。

我们来获取屏幕的高和宽，代码如下:

```
<script type="text/javascript">
  document.write( "屏幕宽度: "+screen.width+"px<br />" );
  document.write( "屏幕高度: "+screen.height+"px<br />" );
</script>
```

### `screen.availHeight` / `screen.availWidth` 屏幕可用高和宽度

1. `screen.availWidth` 属性返回访问者屏幕的宽度，以像素计，减去界面特性，比如任务栏。
2. `screen.availHeight` 属性返回访问者屏幕的高度，以像素计，减去界面特性，比如任务栏。

**注意:**

不同系统的任务栏默认高度不一样，及任务栏的位置可在屏幕上下左右任何位置，所以有可能可用宽度和高度不一样。

我们来获取屏幕的可用高和宽度，代码如下:

```
<script type="text/javascript">
document.write("可用宽度：" + screen.availWidth);
document.write("可用高度：" + screen.availHeight);
</script>
```

**注意:根据屏幕的不同显示值不同。**

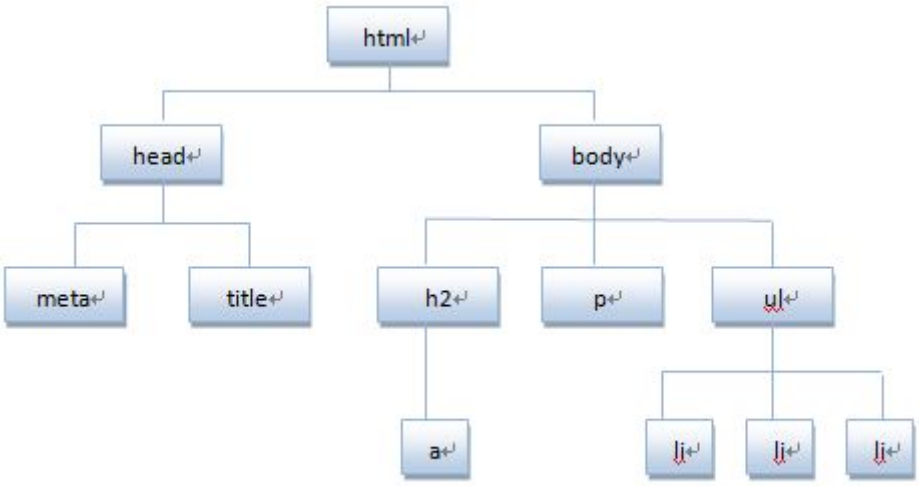
## 认识DOM

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）。

先来看看下面代码:

```
<!DOCTYPE HTML>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312" />
<title>DOM</title>
</head>
<body>
    <h2><a href="http://www.imooc.com">javascript DOM</a></h2>
    <p>对HTML元素进行操作，可添加、改变或移除CSS样式等</p>
    <ul>
        <li>JavaScript</li>
        <li>DOM</li>
        <li>CSS</li>
    </ul>
</body>
</html>
```

将HTML代码分解为DOM节点层次图:




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:**

| 方法 | 说明 |
| --- | --- |
| nodeName | 返回一个字符串，其内容是给定节点的名字。 |
| nodeType | 返回一个整数，这个数值代表给定节点的类型 |
| nodeValue | 返回给定节点的当前值 |

**遍历节点树:**

| 方法 | 说明 |
| --- | --- |
| childNodes | 返回一个数组，这个数组由给定元素节点的子节点构成 |
| firstChild | 返回第一个子节点 |
| lastChild | 返回最后一个子节点 |
| parentNode | 返回一个给定节点的父节点 |
| nextSibling | 返回给定节点的下一个子节点 |
| previousSibling | 返回给定节点的上一个子节点 |

以上图 `ul` 为例，它的父级节点 `body` ,它的子节点3个 `li` ,它的兄弟结点 `h2` 、 `p` 。

DOM操作:

| 方法 | 说明 |
| --- | --- |
| createElement(element) | 创建一个新的元素节点 |
| createTextNode() | 创建一个包含着给定文本的新文本节点 |
| appendChild() | 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。 |
| insertBefore() | 将一个给定节点插入到一个给定元素节点的给定子节点的前面 |
| removeChild() | 从一个给定元素中删除一个子节点 |
| replaceChild() | 把一个给定父元素里的一个子节点替换为另外一个节点 |

注意:前两个是document方法。

## `getElementsByName()` 返回带有指定名称的节点对象的集合

**语法:**

```
1    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。

看看下面的代码:

```
<!DOCTYPE HTML>
<html>
<head>
<script type="text/javascript">
function getElements(){
  var x=document.getElementsByName("alink");
  alert(x.length);
}
</script>
</head>
<body>
  <a name="alink" href="#" />我是链接一</a><br />
  <a name="alink" href="#" />我是链接二</a><br />
  <a name="alink" href="#" />我是链接三</a><br />
  <br />
  <input type="button" onclick="getElements()" value="看看几个链接?" />
</body>
</html>
```

*可以这样记忆，`getElementById`是单数`getElementsByname`是负数，所以`getElementsByName`可能返回一个数组*

## `getElementsByTagName()`返回带有指定标签名的节点对象的集合

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:**

```
document.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。

看看下面代码，通过`getElementsByTagName()`获取节点。

```
<!DOCTYPE HTML>
<html>
    <head>
        <title> JavaScript </title>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
    </head>
    <body>
        <p id="intro">我的课程</p>
        <ul>
            <li>JavaScript</li>
            <li>JQuery</li>
            <li>HTML</li>
            <li>JAVA</li>
            <li>PHP</li>
        </ul>
        <script>
            // 获取所有的li集合
            var list = document.getElementsByTagName('li');
            // 访问无序列表: [0]索引
            li = list[0];
            // 获取list的长度
            document.write(list.length);
            // 弹出li节点对象的内容
            document.write(li.innerHTML);
        </script>
    </body>
</html>
```

## 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。

方法总结如下:

| 方法 | 说明 | 获得 |
| --- | --- | --- |
| getElementById | 通过指定 id 获得元素 | 一个 |
| getElementsByName | 通过元素名称 name 属性获得元素 | 一组 |
| getElementsByTagName | 通过标签名称获得元素 | 一组 |

**注意：方法区分大小写**

通过下面的例子(6个name="hobby"的复选项，两个按钮)来区分三种方法的不同:

```
<input type="checkbox" name="hobby" id="hobby1">  音乐
<input type="checkbox" name="hobby" id="hobby2">  登山
<input type="checkbox" name="hobby" id="hobby3">  游泳
<input type="checkbox" name="hobby" id="hobby4">  阅读
<input type="checkbox" name="hobby" id="hobby5">  打球
<input type="checkbox" name="hobby" id="hobby6">  跑步
<input type="button" value = "全选" id="button1">
<input type="button" value = "全不选" id="button1">
```

1. `document.getElementsByTagName("input")` ，结果为获取所有标签为 `input` 的元素，共8个。
2. `document.getElementsByName("hobby")` ，结果为获取属性 `name="hobby"` 的元素，共6个。
3. `document.getElementById("hobby6")` ，结果为获取属性 `id="hobby6"` 的元素，只有一个，"跑步"这个复选项。

## `getAttribute()` 通过元素节点的属性名称获取属性的值。

通过元素节点的属性名称获取属性的值。

**语法:**

```
elementNode.getAttribute(name)
```

**说明:**

1. `elementNode`：使用 `getElementById()` 、 `getElementsByTagName()` 等方法，获取到的元素节点。
2. `name`：要想查询的元素节点的属性名字

看看下面的代码，获取 `h1` 标签的属性值:

```
<!DOCTYPE HTML>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<title>getAttribute()</title>
</head>
<body>
<h1 id="alink" title="getAttribute()获取标签的属值" onclick="hattr()">点击我，获取标签的属值</h1>

<script type="text/javascript">
 function hattr(){
   var anode=document.getElementById("alink");
   var attr1=anode.getAttribute("id");
   var attr2=anode.getAttribute("title");
   document.write("h1标签的ID : "+attr1+"<br>");
   document.write("h1标签的title : "+attr2);
 }
 </script>

</body>
</html>
```

**运行结果:**

```
h1标签的ID : alink
h1标签的title : getAttribute()获取标签的属值
```

## `setAttribute()` 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。

## 节点属性

在文档对象模型 (DOM) 中，每个节点都是一个对象。DOM 节点有三个重要的属性：

1. `nodeName`：节点的名称
2. `nodeValue` ： 节点的值
3. `nodeType` ： 节点的类型

一、 `nodeName` 属性: 节点的名称，是只读的。

1. 元素节点的 `nodeName` 与标签名相同
2. 属性节点的 `nodeName` 是属性的名称
3. **文本节点的 `nodeName` 永远是 `#text`**
4. **文档节点的 `nodeName` 永远是 `#document`**

二、 `nodeValue` 属性：节点的值

1. 元素节点的 `nodeValue` 是 `undefined` 或 `null`
2. 文本节点的 `nodeValue` 是文本自身
3. 属性节点的 `nodeValue` 是属性的值

三、 `nodeType` 属性: 节点的类型，是**只读**的。以下常用的几种结点类型:

元素类型 节点类型 元素 1 属性 2 文本 3 注释 8 文档 9

**访问子节点 `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。**

**语法:**

```
elementNode.childNodes
```

**注意:**

如果选定的节点没有子节点，则该属性返回不包含节点的 NodeList。

我们来看看下面的代码:

```
<!DOCTYPE HTML>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<title>无标题文档</title>
</head>
<body>
<ul>
  <li>javascript</li>
  <li>jQuery</li>
  <li>PHP</li>
</ul>
<script type="text/javascript">
  var x=document.getElementsByTagName("ul")[0].childNodes;
  document.write("UL子节点个数:"+x.length+"<br />");
  document.write("节点类型:"+x[0].nodeType);
</script>
</body>
</html>
```

**运行结果:**

**IE:**

```
UL子节点个数:3
节点类型:1
```

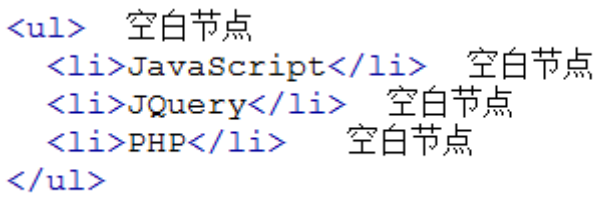
**其它浏览器:**

```
UL子节点个数:7
节点类型:3
```

**注意:**

1. IE全系列、firefox、chrome、opera、safari兼容问题

2. 节点之间的空白符，在firefox、chrome、opera、safari浏览器是文本节点，所以IE是3，其它浏览器是7，如下图所示:




如果把代码改成这样:

```
<ul><li>javascript</li><li>jQuery</li><li>PHP</li></ul>
```

运行结果:（IE和其它浏览器结果是一样的）

```
UL子节点个数:3
节点类型:1
```

*这样的空白符算文本节点，这点可以多做些研究*

**访问子节点的第一 `firstChild` 和最后 `lastChild` 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。

**注意: 上一节中，我们知道Internet Explorer会忽略节点之间生成的空白文本节点，而其它浏览器不会。我们可以通过检测节点类型，过滤子节点。 (以后章节讲解)**

**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。** *父节点只能有一个，所以是 `parentNode` ，子节点能有复数个，所以是 `childNodes`*

看看下面的例子,获取 P 节点的父节点，代码如下:

```
<div id="text">
  <p id="con"> parentNode 获取指点节点的父节点</p>
</div>
<script type="text/javascript">
  var mynode= document.getElementById("con");
  document.write(mynode.parentNode.nodeName);
</script>
```

**运行结果:**

```
parentNode 获取指点节点的父节点
DIV
```

**访问祖节点:**

```
elementNode.parentNode.parentNode
```

看看下面的代码:

```
<div id="text">
  <p>
    parentNode
    <span id="con"> 获取指点节点的父节点</span>
  </p>
</div>
<script type="text/javascript">
  var mynode= document.getElementById("con");
  document.write(mynode.parentNode.parentNode.nodeName);
</script>
```

**运行结果:**

```
parentNode获取指点节点的父节点
DIV
```

**注意: 浏览器兼容问题，chrome、firefox等浏览器标签之间的空白也算是一个文本节点。**

**`nextSibling` / `previousSibling` 访问兄弟节点**

`nextSibling` 属性可返回某个节点之后紧跟的节点（处于同一树层级中）。

**语法:**

```
nodeObject.nextSibling
```

说明：如果无此节点，则该属性返回 `null` 。

`previousSibling` 属性可返回某个节点之前紧跟的节点（处于同一树层级中）。

**语法:**

```
nodeObject.previousSibling
```

说明：如果无此节点，则该属性返回 `null` 。

注意: 两个属性获取的是节点。Internet Explorer 会忽略节点间生成的空白文本节点（例如，换行符号），而其它浏览器不会忽略。

解决问题方法:

判断节点nodeType是否为1, 如是为元素节点，跳过。

```
<!DOCTYPE HTML>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<title>nextSibling</title>
</head>
<body>
<ul id="u1">
            <li id="a">javascript</li>
            <li id="b">jquery</li>
            <li id="c">html</li>
        </ul>
        <ul id="u2">
            <li id="d">css3</li>
            <li id="e">php</li>
            <li id="f">java</li>
        </ul>
<script type="text/javascript">
function get_nextSibling(n)
{
var x=n.nextSibling;
while (x.nodeType!=1)
  {
  x=x.nextSibling;
  }
return x;
}

var x=document.getElementsByTagName("li")[0];
document.write(x.nodeName);
document.write(" = ");
document.write(x.innerHTML);

var y=get_nextSibling(x);

document.write("<br />nextsibling: ");
document.write(y.nodeName);
document.write(" = ");
document.write(y.innerHTML);

</script>
</body>
</html>
```

**运行结果:**

```
LI = javascript
nextsibling: LI = jquery
```

### `appendChild()` 插入节点

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:

```
<div id="test"><p id="x1">HTML</p><p>JavaScript</p></div>

<script type="text/javascript">
  var otest = document.getElementById("test");
  var newnode = document.createElement("p");
  newnode.innerHTML = "This is a new p";
  // appendChild方法添加节点
  otest.appendChild(newnode);
</script>
```

**运行结果:**

```
HTML
JavaScript
This is a new p
```

### `insertBefore()` 插入节点

`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。

我们在来看看下面代码，在指定节点前插入节点。

```
<!DOCTYPE HTML>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<title>无标题文档</title>
</head>
<body>

<div id="div1"><p id="x1">JavaScript</p><p>HTML</p></div>

<script type="text/javascript">

  var otest = document.getElementById("div1");
  var node = document.getElementById("x1");
  var newnode = document.createElement("p");
  newnode.innerHTML = "This is a new p";
  otest.insertBefore(newnode,node);

</script>
</body>
</html>
```

**运行结果:**

```
This is a new p
JavaScript
HTML
```

注意: `otest.insertBefore(newnode,node);` 也可以改为:
`otest.insertBefore(newnode,otest.childNodes[0]);`

### `removeChild()` 删除节点

`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。

我们来看看下面代码，删除子点。

```
<!DOCTYPE HTML>
<html>
  <head>
    <title>Untitled</title>
  </head>
  <body>
    <div id="div1"><h1>HTML</h1><h2>javascript</h2></div>

    <script type="text/javascript">
      var otest=document.getElementById("div1");
      var x=otest.removeChild(otest.childNodes[1]);
      document.write("删除节点的内容:"+x.innerHTML);
    </script>

  </body>
</html>
```

**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:

```
var otest=document.getElementById("div1");
var x=otest.removeChild(otest.childNodes[1]);
x=null;
```

### `replaceChild()` 替换元素节点

`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。

我们来看看下面的代码:

```
<!DOCTYPE HTML>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<title>无标题文档</title>
</head>
<body>
  <script type="text/javascript">
      function replaceMessage(){
           var newnode=document.createElement("p");
           var newnodeText=document.createTextNode("JavaScript");
           newnode.appendChild(newnodeText);
           var oldNode=document.getElementById("oldnode");
           oldNode.parentNode.replaceChild(newnode,oldNode);
       }
  </script>

  <h1 id="oldnode">Java</h1>
  <a href="javascript:replaceMessage()"> "Java" 替换 "JavaScript"</a>
 </body>
</html>
```

效果: 将文档中的 Java 改为 JavaScript。

注意:

1. 当 `oldnode` 被替换时，所有与之相关的属性内容都将被移除。
2. `newnode` 必须先被建立。

### `createElement()` 创建元素节点

`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!');");//还能这样用

   body.appendChild(btn);
</script>
```

效果：在HTML文档中，创建一个文本框，使用 `setAttribute` 设置属性值。 当点击这个文本框时，会弹出对话框“This is a text!”。

### `createTextNode()` 创建文本节点

`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：

```
<!DOCTYPE HTML>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<title>无标题文档</title>
<style type="text/css">

.message{
    width:200px;
    height:100px;
    background-color:#CCC;}

</style>
</head>
<body>
<script type="text/javascript">

   var element = document.createElement("div");
   element.className = "message";
   var textNode = document.createTextNode("Hello world!");
   element.appendChild(textNode);
   document.body.appendChild(element);

</script>

</body>
</html>
```

## 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：**

1. `window.innerHeight` - 浏览器窗口的内部高度
2. `window.innerWidth` - 浏览器窗口的内部宽度

**二、对于 Internet Explorer 8、7、6、5：**

1. `document.documentElement.clientHeight` 表示HTML文档所在窗口的当前高度。
2. `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

1. `document.body.clientHeight`
2. `document.body.clientWidth`

在不同浏览器都实用的 JavaScript 方案：

```
var w= document.documentElement.clientWidth
      || document.body.clientWidth;
var h= document.documentElement.clientHeight
      || document.body.clientHeight;
```

## `scrollHeight` / `scrollWidth` 获取网页内容高度和宽度

**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、浏览器兼容性**

```
var w=document.documentElement.scrollWidth
   || document.body.scrollWidth;
var h=document.documentElement.scrollHeight
   || document.body.scrollHeight;
```

**注意:**

1. 区分大小写
2. `scrollHeight` 和 `scrollWidth` 还可获取Dom元素中内容实际占用的高度和宽度。

## `offsetHeight` / `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)。

`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**

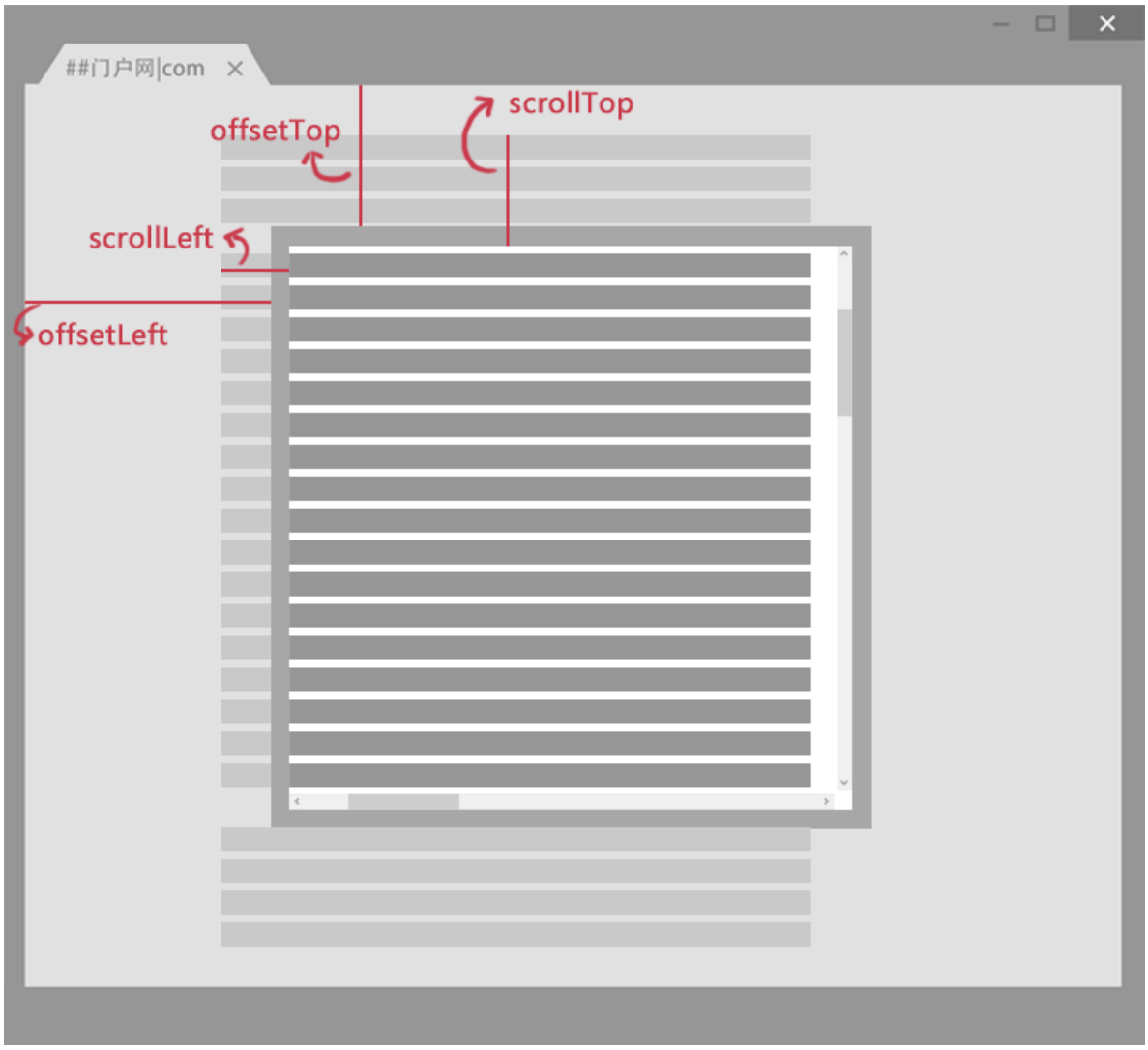
`offsetHeight` = `clientHeight` + 滚动条 + 边框。

**二、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

**我们先来看看下面的图:**




`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。