




文档对象模型

管理科学与工程学科
耿方方



主要内容

- Document对象
- DOM概述
- 访问HTML网页
- 修改HTML网页

Document对象

■ Document对象的属性

属 性	说 明
alinkColor	链接文字被单击时的颜色，对应于<body>标记中的alink属性
all[]	存储HTML标记的一个数组(该属性本身也是一个对象)
anchors[]	存储锚点的一个数组。(该属性本身也是一个对象)
bgColor	文档的背景颜色，对应于<body>标记中的bgcolor属性
cookie	表示cookie的值
fgColor	文档的文本颜色(不包含超链接的文字)对应于<body>标记中的text属性值
forms[]	存储窗体对象的一个数组(该属性本身也是一个对象)
fileCreateDate	创建文档的日期
fileModifiedDate	文档最后修改的日期
fileSize	当前文件的大小
lastModified	文档最后修改的时间
images[]	存储图像对象的一个数组(该属性本身也是一个对象)
linkColor	未被访问的链接文字的颜色，对应于<body>标记中的link属性
links[]	存储link对象的一个数组(该属性本身也是一个对象)
vlinkColor	表示已访问的链接文字的颜色，对应于<body>标记的vlink属性
title	当前文档标题对象
body	当前文档主体对象
readyState	获取某个对象的当前状态
URL	获取或设置URL

Document对象

■ Document对象的方法

方 法	说 明
close	关闭文档的输出流
open	打开一个文档输出流并接收write和writeln方法的创建页面内容
write	向文档中写入HTML或JavaScript语句
writeln	向文档中写入HTML或JavaScript语句，并以换行符结束
createElement	创建一个HTML标记
getElementById	获取指定id的HTML标记

Document对象

- Document对象的应用

- 链接的颜色

- 1、alinkColor属性

该属性用来获取或设置当链接被单击时显示的颜色。

```
document.alinkColor=setColor
```

- 2、linkColor属性

该属性用来获取或设置页面中未单击的链接的颜色。

- 3、vlinkColor属性

该属性用来获取或设置页面中单击过的链接的颜色。

Document对象

- Document对象的应用

- 文档背景色和前景色

- 1、bgColor属性

该属性用来获取或设置页面的背景颜色。

```
document.bgColor=setColor
```

- 2、fgColor属性

该属性用来获取或设置页面的前景颜色，即页面中文字的颜色。

Document对象

- Document对象的应用
- 6-01-01: <body>
 - 背景自动变色
- <script language="javascript">
- var Arraycolor=new Array("#00FF66", "#FFFF99", "#99CCFF", "#FFCCFF", "#FFCC99", "#00FFFF");
- var n=0;
- function changecolors() {
- n++;
- if (n==(Arraycolor.length)) n=0;
- document.bgColor = Arraycolor[n];
- document.fgColor=Arraycolor[n+1];
- setTimeout("changecolors()", 1000);
- }
- changecolors();
- </script>
- </body>

Document对象

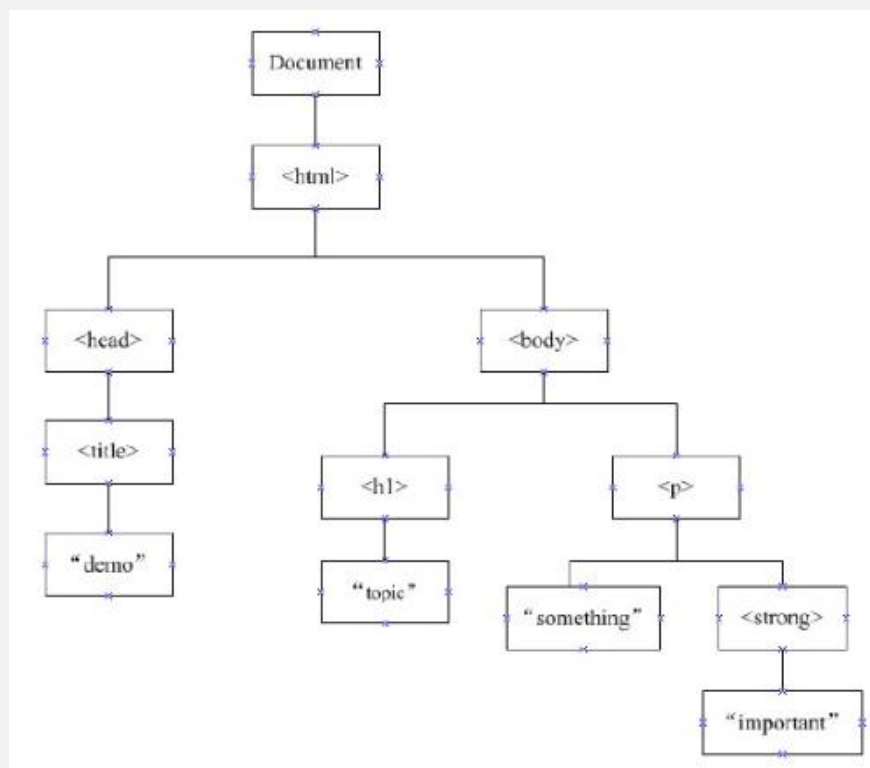
- 查看文档创建时间、修改时间、文档大小以及获取URL
- 1、fileCreateDate属性
- 该属性用来获取文档的创建日期。
- 2、fileModifiedDate属性
- 用来获取文档最后修改日期。
- 3、lastModified属性
- 用来获取文档最后修改的日期。
- 4、fileSize属性
- 用来获取文档的大小。
- 5、URL属性
- document.URL

Document对象

- 实例01-02: <script>
- ```
function openWin() {
```
- ```
    var dw;
```
- ```
 dw=window.open("", "", "width=200,height=200");
```
- ```
    dw.document.open();
```
- ```
 dw.document.write("<html><head><title>一个新的窗口</title>");
```
- ```
    dw.document.write("</head>");
```
- ```
 dw.document.write("<body>");
```
- ```
    dw.document.write("<h1>这是一个新窗口</h1><br>");
```
- ```
 dw.document.write("<p>这里有新的内容! </p>");
```
- ```
    dw.document.write("</body></html>");
```
- ```
 dw.document.close();}
```
- 
- ```
</script>
```
- ```
<input type="button" value="打开一个新文档" onclick="openWin();" />
```

- 文档对象模型简称DOM，DOM是一种HTML/XHTML页面的编程接口（API）。借助DOM模型，可以将结构化文档转换成DOM树。实际上，它是把我们的标记方式转换为JavaScript可以理解的格式。简单的说就是DOM就像页面上的所有元素的一个地图。使用它通过名字和元素来找到元素，然后添加、修改或删除元素及其内容。
- 通过DOM技术不仅可以操作HTML页面的内容，包括添加元素、修改元素属性、删除元素等，而且还能操作HTML页面的风格样式。

- 在DOM中，HTML文档的层次结构被表示为一个树形结构。树的根节点是一个表示当前HTML文档的document对象，树的每个子节点表示HTML文档中的不同内容。



- 在DOM树中有很多不同类型的节点。主要包括文档节点、元素节点、属性节点和文本节点。
- 文档节点：HTML中每个元素、属性以及文本都由它自己的DOM节点呈现。这棵树的顶端是文档节点，它呈现为整个页面。当需要访问任何元素、属性和文本节点时，都需要通过文档节点进行导航。
- 元素节点：HTML元素描述了HTML页面的结构。例如<p>表示段落、<ul>表示列表。
- 属性节点：HTML元素的开始标签中可以包含若干属性，这些属性再DOM树中形成属性节点。
- 文本节点：当访问元素节点时，可以访问元素内部的文本，这些文本保存再文本节点中。

- 节点之间的关系

包括三种关系：父子关系、兄弟关系和祖孙关系。

例如：<html>

```
<head>
 <title>节点关系</title>
</head>
<body>
 <ul style= “none” >
 JavaScript
 jQuery

</body>
</html>
```

- W3C在1998年10月标准化了DOM第一级，不仅定义了基本的接口，其中还包含了所有HTML接口。在2000年11月标准化了DOM第二级，在第二级中不但对核心的接口升级，还定义了使用文档事件和CSS样式表的标准的API。
- DOM1：W3C规范。专注于HTML文档和XML文档。
- DOM2：对DOM1增加了样式表对象模型。
- DOM3：对DOM2增加了内容模型（DTD、Schemas）和文档验证。

- getElementById
- DOM提供了一个名为getElementById的方法，这个方法将返回一个与哪个有着给定id属性值的元素节点对应的对象。
- getElementById是document对象特有的函数。在脚本代码里，函数名的后面必须跟有一对圆括号，这对圆括号包含着函数的参数。getElementById方法只有一个参数：想要获得的那个元素的id属性的值，这个id属性值必须放在单引号或者双引号里，其使用方法如下所示：

```
document.getElementById("purchases");
```

# 访问HTML网页

- 案例1:通过id获取元素, 主要代码

```
▪ <body>
▪ <div id="page">
▪ <h1 id="header">List King</h1>
▪ <h2>Buy groceries</h2>
▪
▪ <li id="one" class="hot">fresh
figs
▪ <li id="two" class="hot">pine nuts
▪
▪ </div>
▪ <script>
▪ var el = document.getElementById('one');
▪ el.className = 'cool';
▪ </script>
▪ </body>
▪ </html>
```

```
.hot {
```

```
background-color: #d7666b;
color: #fff;
text-shadow: 2px 2px 1px #914141;
border-top: 1px solid #e99295;
border-bottom: 1px solid #914141;}
```

```
.cool {
```

```
background-color: #6cc0ac;
color: #fff;
text-shadow: 2px 2px 1px #3b6a5e;
border-top: 1px solid #7ee0c9;
border-bottom: 1px solid #3b6a5e;}
```



# 访问HTML网页

- 以上是返回单一元素的方法,使用下面方式可以返回多个元素,即返回一个NodeList。
- NodeList是一组元素的集合。每个节点都有索引编号(从0开始编号),元素节点出现的顺序和它们在HTML页面中出现的顺序相同。当DOM查询返回一个NodeList时,可以;

从NodeList选择元素,遍历NodeList中的每个元素,然后针对每个元素节点执行相同的语句。

- **length**属性表示NodeList中一共有多少项,**item ()**方法返回NodeList中特定的节点,需要在小括号中制定所需要的索引编号。还可以像数组那样,获取NodeList中的一项。

- `getElementsByName`
- `getElementsByName`方法返回一个对象数组，每个对象分别对应着文档里有着给定标签的一个元素。类似于`getElementById`，这个方法也是只有一个参数的函数，它的参数是标签的名字，其使用方法如下所示：

```
element.getElementsByTagName(tag);
```

- `getElementsByClassName`
- HTML5 DOM中新增了一个方法：`getElementsByClassName`。这个方法能够使我们通过`class`属性中的类名来访问元素。不过由于这是一个新增方法，某些DOM实现中可能还未支持此方法的解析，其在Internet Explorer 5, 6, 7, 8中无效，因此在使用时要注意其兼容性：
- `getElementsByClassName`方法与`getElementsByTagName`方法相似，也只接受一个参数，就是类名，其使用方法如下所示：

```
element.getElementsByClassName(class);
```

# 访问HTML网页

- item () 方法:

```
var elements=document.getElementsByClassName ('hot') ;
 if(elements.length>=1) {
 var firstItem=elements.item(0);
 }
}
```

## 数组方法

```
var elements=document.getElementsByClassName ('hot') ;
 if(elements.length>=1) {
 var firstItem=elements[0];
 }
}
```

- `querySelector()` 方法返回文档中匹配指定 CSS 选择器的一个元素。
- 语法：`document.querySelector(CSS selectors)`
- 例如：`document.querySelector("p");`
- `document.querySelector(".example");`
- `document.querySelector("#example");`
- `document.querySelector("p.example");`
- `document.querySelector("a[target]");`

- `querySelector()` 方法返回文档中匹配指定 CSS 选择器的多个元素。
- 语法：`document.querySelectorAll(CSS selectors)`
- 遍历NodeList使其具备相同的操作。
- 例如：`var hotItems=document.querySelectorAll('li.hot');`

```
 if(hotItems.length>0) {
```

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

```
 hotItems[i].className='cool';
```

```
 }
```

```
 }
```

- 得到一个元素节点后，可以使用如下5个属性来找到其他相关的元素，这种方式被称为遍历DOM。
- parentNode: 该属性在HTML中找到包含该元素的元素节点（或其父元素节点）。
- previousSibling和nextSibling这两个属性找到当前节点的前一个或后一个兄弟节点。
- firstChild和lastChild这两个属性找到当前元素的第一个或最后一个子节点。
- 使用这些属性可能会遇到一些麻烦，例如有些浏览器会在元素之间添加一个文本节点，不管它们之间是不是真的有空白。解决这个问题最受欢迎的方法就是使用像jQuery这样的JavaScript库，可以解决因为浏览器之间的不一致性。

### ■ 案例2:

```
<li id="one" class="hot">fresh figs<li id="two" class="hot">pine nuts<li id="three" class="hot">honey<li id="four">balsamic vinegar
```

```
var startItem = document.getElementById('two');
var prevItem = startItem.previousSibling;
var nextItem = startItem.nextSibling;
```

```
// Change the values of the siblings' class attributes
prevItem.className = 'complete';
nextItem.className = 'cool';
```



- 可以使用如下方法来操作元素的内容:
- 导航到文本节点。这种方式在元素只包含文本、不包含其他元素时最好用。在一个元素导航到它的文本节点后，使用`nodeValue`访问文字。
- 使用包含元素。这种方法可以让你同时获取到其文本节点和子元素。当操作一个元素节点时，这个元素可能包含标签。可以选择需要获取/更新标签，或是获取/更新文本。

`innerHTML`: 获取/设置文本和标签

`textContent`: 仅获取/设置文本

`innerText`: 仅获取/设置文本

- 例如nodeValue的使用:

```
<li id="one" >freshfigs
```

```
document.getElementById("one").firstChild.nextSibling.noValue
```

### ■ 案例3: 使用nodeValue更新文本节点内容:

```
■ <body>
■ <div id="page">
■ <h1 id="header">List</h1>
■ <h2>Buy groceries</h2>
■
■ <li id="one" class="hot">fresh
figs
■ <li id="two" class="hot">pine nuts
■ <li id="three" class="hot">honey
■ <li id="four">balsamic vinegar
■
■ </div>
■ </body>
```

```
<script>
 var itemTwo = document.getElementById('two');
 var elText = itemTwo.firstChild.nodeValue;
 elText = elText.replace('pine nuts', 'kale');
 itemTwo.firstChild.nodeValue = elText;
</script>
```

### ■ 案例4: innerText与TextContent:

- `<body>`
- `<div id="page">`
- `<h1 id="header">List</h1>`
- `<h2>Buy groceries</h2>`
- `<ul>`
- `<li id="one" class="hot"><em>fresh</em>`  
figs</li>
- `<li id="two" class="hot">pine nuts</li>`
- `<li id="three" class="hot">honey</li>`
- `<li id="four">balsamic vinegar</li>`
- `</ul>`
- `<div id="scriptResults"></div>`
- `</div>`
- `</body>`

```
<script>
 var firstItem = document.getElementById('one');
 var showTextContent = firstItem.textContent;
 var showInnerText = firstItem.innerText;
 var msg = '<p>textContent: ' + showTextContent + '</p>';
 msg += '<p>innerText: ' + showInnerText + '</p>';
 var el = document.getElementById('scriptResults');
 el.innerHTML = msg;

 firstItem.textContent = 'sourdough bread';
</script>
```

- `innerText`与`textContent`的区别:
- `innerText`:支持情况,虽然大多数浏览器厂商都接受这个属性,不过Firefox不支持它,因为`innerText`不属于任何标准。遵从CSS,它不会返回任何被CSS隐藏的内容。性能,因为`innerText`属性需要考虑到布局规则来判断元素的可见性,它在获取文本内容时的速度比`textContent`要慢。
- 在IE8或更早地IE中, `textContent`属性不起作用。

- 有两种不同的方法来添加和移除DOM树种的内容：innerHTML属性和DOM操作。
- innerHTML属性可以获取或修改元素的内容，包括其中的所有子节点。
- 例如<li id="one"><em>fresh</em>figs</li>

获取内容：

```
var elContent=document.getElementById("one").innerHTML
```

则elContent变量应包含如下字符串：<em>fresh</em> figs

设置内容：

```
document.getElementById("one").innerHTML=elContent
```

注意：当使用innerHTML添加新内容时，需要注意如果缺失关闭标签的话，可能会影响整个页面的设计。更坏的情况是，如果使用innerHTML把用户提供的内容添加到一个页面上，他们可能会添加恶意内容。

- DOM操作-添加元素

如果我们需要向HTML中添加新元素，那么我们首先便需要创建该元素，然后向已存在的元素追加创建的新元素。具体涉及以下三个步骤：

- 1、创建元素createElement ( )
- 2、设置内容createTextNode ( )
- 3、把它添加到DOM中appendChild ( )

### ■ DOM操作-添加元素

- 首先创建一个新的元素，比如<p>，其代码如下所示：

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

- 如果需要向<p>元素中添加文本内容，必须先创建一个文本节点，如下代码所示：

```
var node=document.createTextNode("这是创建的新段落。");
```

- 然后将该文本节点追加到刚开始创建的<p>元素中，代码如下所示：

```
para.appendChild(node);
```

- 最后必须向一个已有的元素追加这个新建的元素，其实现代码如下所示：

```
var element=document.getElementById("div1");
element.appendChild(para);
```



### ■ 案例5:

```
<div id="page">

 <h1 id="header">List</h1>

 <h2>Buy groceries</h2>

 <ul id="todo"><li id="one"
class="hot">fresh
figs<li id="two"
class="hot">pine nuts<li
id="three" class="hot">honey<li
id="four">balsamic vinegar

</div>

</body>
```

```
<script>
 var newEl = document.createElement('li');
 var newText = document.createTextNode('quinoa');
 newEl.appendChild(newText);

 var position = document.getElementsByTagName('ul')[0];
 position.appendChild(newEl);
</script>
```

- 创建多个节点
- 创建多个节点使用循环语句，利用createElement()方法和createTextNode()方法生成新元素并生成文本节点，最常用使用appendChild()方法将创建的新节点添加到页面上。
- 例如5-01:
- `<body onload="dc()">`
- `<script>`
- `function dc() {`
- `var aText=["第一个节点内容","第二个节点内容","第三个节点内容","第四个节点内容"];`
- `for (i=0;i<aText.length;i++){`
- `var ce=document.createElement("p");`
- `var cText=document.createTextNode(aText[i]);`
- `ce.appendChild(cText);`
- `document.body.appendChild(ce);`
- `}`
- `}`
- `</script>`
- `</body>`

- 创建多个节点
- 由于每次添加新的节点时都会刷新页面，通过循环语句添加节点的方法会使浏览器显得十分缓慢。这里可以通过使用`createDocumentFragment()`方法来解决这个问题。
- 例如5-02: `<body onload="dc()">`

```
<script>
 function dc() {
 var aText=["第一个节点内容","第二个节点内容","第三个节点内容","第四个节点内容"];
 var cdf=document.createDocumentFragment();//创建文件碎片节点
 for(var i=0;i<aText.length;i++){//遍历节点
 var ce=document.createElement("b");
 var cb=document.createElement("br");
 var cText=document.createTextNode(aText[i]);
 ce.appendChild(cText);
 cdf.appendChild(ce);
 cdf.appendChild(cb);
 }
 document.body.appendChild(cdf);
 }
</script>
</body>
```

- 节点的插入和追加
- 插入节点通过使用 `insertBefore()` 方法来实现。 `insertBefore()` 方法将新的子节点添加到当前节点的前面。
- 语法：
  - `Obj.insertBefore(new, ref)`
  - `New`: 表示新的子节点；
  - `Ref`: 指定一个节点，在这个节点前插入新的节点。

- 节点的复制
- 复制节点可以使用`cloneNode()`方法来实现。
- 语法：
- `Obj.cloneNode(deep)`

- 案例：5-03：
- `<script language="javascript">`
- `<!--`
- `function AddRow(b1) {`
- `var sel=document.getElementById("sexType");//访问节点`
- `var newSelect=sel.cloneNode(b1); //复制节点`
- `var b=document.createElement("br");//创建节点元素`
- `var dq=document.getElementById("di");`
- `dq.appendChild(newSelect); //将新节点添加到当前节点的未`
- `尾`
- `dq.appendChild(b);`
- `}`
- `-->`
- `</script>`

- DOM操作-删除元素
- 如果需要在HTML中删除元素，那么我们首先便需要获得该元素，然后得到该元素的父元素，最后通过removeChild方法删除该元素，其实现流程如下所示：
- 获得该元素，比如要获得id属性值为div1的元素，其代码如下所示：

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

- 获得该元素的父元素，代码如下所示：

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

- 从父元素中删除该元素

```
parent.removeChild(child);
```

- 案例6:

```
<body>

 <div id="page">

 <h1 id="header">List</h1>

 <h2>Buy groceries</h2>

 <li id="one" class="hot">fresh figs

 <li id="two" class="hot">pine nuts

 <li id="three" class="hot">honey

 <li id="four">balsamic vinegar

 </div>

</body>
```

```
var removeEl = document.getElementsByTagName('li')[3];

var containerEl = document.getElementsByTagName('ul')[0];

containerEl.removeChild(removeEl);
```



- DOM操作-属性操作
- 得到一个元素节点后,可以在这个元素上使用其他一些对象属性和方法来获取和修改它的HTML属性。
- `getAttribute`方法就是专门用来获取元素属性的,相应的我们也可以使用`setAttribute`方法来更改元素节点的值。
- `getAttribute`是一个函数,它只有一个参数即:查询的属性的名称,其使用方法如下所示:

```
object.getAttribute(attribute);
```

- DOM操作-属性操作
- `hasAttribute()` 检查元素节点是否包含特定属性。
- `setAttribute()` 方法是用来进行设置属性的，它允许我们对属性节点的值做出修改，与`getAttribute`方法一样，它也是只能用于元素节点，其使用方法如下所示：

```
object.setAttribute(attribute, value);
```

- `removeAttribute()` 从元素移除属性。
- 属性：
- `className` 获取或设置`class`属性的值。
- `id` 获取或设置`id`属性的值。

- 案例7: 获取属性

```
<div id="page">

 <h1 id="header">List</h1>

 <h2>Buy groceries</h2>

 <li id="one" class="hot">fresh figs

 <li id="two" class="hot">pine
nuts

 <li id="three" class="hot">honey

 <li id="four">balsamic
vinegar

 <div id="scriptResults"></div>

</div>
```

```
var firstItem = document.getElementById('one');
if (firstItem.hasAttribute('class')) {
 var attr = firstItem.getAttribute('class');
 var el = document.getElementById('scriptResults');
 el.innerHTML = '<p>The first item has a class name: ' +
 attr + '</p>';
}
```

### ■ 案例8：更新和创建属性

```


 <li id="one" class="hot">fresh
 figs

 <li id="two" class="hot">pine nuts

 <li id="three" class="hot">honey
 <li id="four">balsamic vinegar

<style>
 .hot {color:red;}
 .cool {color:blue;}
 .complete {color:gray;}
</style>

var firstItem = document.getElementById('one');
// Get the first item
firstItem.className = 'complete';
// Change its class attribute

var fourthItem = document.getElementsByTagName('li').item(3);
// Get fourth item
// NOTE: The following line should say fourthItem (not el2)
fourthItem.setAttribute('class', 'cool');
```

### ■ 案例9：综合示例

#### ■ html:

```
<body>

 <div id="page">

 <h1 id="header">List</h1>

 <h2>Buy groceries</h2>

 <li id="one">fresh
 figs

 <li id="two">pine nuts

 <li id="three">honey

 <li id="four">balsamic
 vinegar

 </div>

</body>
```

CSS:

```
<style>

 .cool{
 color:blue;
 }

 span{
 color:red;
 }

</style>
```

### ■ 案例9：综合示例

#### ■ JS:

```
var list = document.getElementsByTagName('ul')[0];

var newItemLast = document.createElement('li');

var newTextLast = document.createTextNode('cream');
newItemLast.appendChild(newTextLast);
list.appendChild(newItemLast);
// Add element end of list

var newItemFirst = document.createElement('li');
// Create element

var newTextFirst = document.createTextNode('kale');
// Create text node

newItemFirst.appendChild(newTextFirst);
// Add text node to element

list.insertBefore(newItemFirst, list.firstChild);
// Add element to list

var listItems = document.querySelectorAll('li');
// All elements
```

```
// ADD A CLASS OF COOL TO ALL LIST ITEMS
var i; //
Counter variable
for (i = 0; i < listItems.length; i++)
{listItems[i].className = 'cool';}
// ADD NUMBER OF ITEMS IN THE LIST TO THE
HEADING
var heading = document.querySelector('h2');
// h2 element
var headingText = heading.firstChild.nodeValue;
// h2 text
var totalItems = listItems.length;
// No. of elements
var newHeading = headingText + '' +
totalItems + ''; // Content
heading.innerHTML = newHeading;
```

# 想一想？

- 1、总结获取网页元素的方法，以及网页元素的操作方法；
- 2、使用DOM编程，在网页上显示一个列表，例如：

```
星期一
```

```
星期二
```

```
星期三
```

```
星期四
```

```
星期五
```

```
星期六
```

```
星期日
```

统计项目数，也显示至网页。