表单及其表单元素操作

一、获取表单元素

1.通过表单 id 属性

document.getElementById("formID");//dom

2.通过表单在表单集合中的索引

document.forms[formIndex]; //bom

document.forms["formName"]; //bom

3.通过表单 name 属性直接访问

document.formName

二、访问表单域（字段）

1.通过表单域 id 属性

document.getElementById("formID");//dom

2. **通过表单域所在的索引**

frm.**elements**[eleIndex]; //可用于遍历表单域

3.通过表单域的 name 属性

frm.elements["name"];//得到名为 name 的表单域

4.通过表单域 id 属性或 name 属性直接访问

frm.eleName;//直接用名称得到表单域

frm[“my name”];//得到 name 属性值有空格的表单域

frm[eleIndex];//得到索引为 eleIndex 的表单元素

三、表单元素常用的属性和方法

disabled:是否可用，true 不可用,false 可用

form:得到表单

blur():使表单域失去焦点。

focus():使表单得到焦点。

onblur 事件：失去焦点时触发，并调用 onblur()函数。

onfocus 事件：得到焦点时触发，并调用 onfocus()函数。

四、列表框、下拉框对象

列表框对象.options，得到所有 option 选项的集合

listbox.options[optIndex].text;/获得文本

listbox.options[optIndex].value;/获得值

listbox.options[optIndex].selected;/获得该项是否被选中

listbox.selected; //选项是否被选中,选中，返回 true,否则，返回 false
multiple=’multiple’设为多选。

五、复选框和单选钮

checked 属性:是否被选中,选中，返回 true,否则，返回 false
click():模仿点击，会触发 click 事件，改变选择状态。
对于复选框，可以进行遍历操作。
通过表单对象.name 值，可返回复选钮的集合(等价于 getElementsByName())

六、掌握表单验证的方式

1、阻止提交 <form onsubmit=”return check()”>
函数 check() 返回 true,表示提交表单，否则，禁止提交表单
2、表单提交 调用表单对象.submit()方法提交

七. 表单常用操作范例

1.获取和设置文本域的内容
例：求和

```
function getSum(){
    //获取表单对象
    var theForm=document.forms["myForm2"];
    //计算和，并赋值给第 3 个文本框
    theForm.elements["sum"].value=
    eval(theForm.elements["x"].value)+eval(theForm.elements["y"].value);
}
```

2.最常见的访问表单字段的方法
最简单常用的访问表单元素的方法自然是 document.getElementById()
例：

```
<input type="text" id="count"  value="" />
var name=document.getElementById("count ").value
```

这种方法无论表单元素处于那个表单中甚至是不在表单中都适用,一般情况下是我们用 JS 访问表单元素的首选.
鉴于 document.getElementById 比较长,你可以用如下函数代替它:

```
function $(id){
    return document.getElementById(id);
}
```

把这个函数放在共有 JS 库中,在 jsp 页面通过如下方法引用它:

```
<head>
<title>添加资源页面</title>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<script src="web/js/check.js" type="text/javascript"></script>
<link rel="stylesheet" rev="stylesheet" href="web/css/style.css" type="text/css" />
</head>
```

此后你就可以直接使用$访问表单元素中的内容:

```
var name=$("name").value;
```

3.获取表单值并将值赋给数组

```
var list=document.getElementsByTagName("input");//对象数组
var strData="";
//对表单中所有的 input 进行遍历
for(var i=0;i<list.length && list[i];i++){
    //判断是否为文本框
    if(list[i].type=="text")  { //type=="text"  表示文本框
        strData +=list[i].value;
        alert(strData);
    }
}
```

4.表单域字段的共性

以下是所有表单字段(除了隐藏字段)

Id 表单域唯一标识

name 表单域名称

disabled 可以用来获取或设置表单控件是否被禁用.

form 特性用来指向字段所在的表单.

blur()方法使表单字段失去焦点.

focus()方法使表单字段获得焦点.

当字段失去焦点是,发生 blur 事件,执行 onblur 事件处理程序.

当字段获取焦点时,发生 focus 事件,执行 onfocus 事件处理函数.

例：当页面载入时将焦点放在第一个字段

在 body 代码中如此书写:

```
<body onload="focusOnFirstElm()">
```

JS 函数如下书写:

```
Fucntion focusOnFirstElm(){
    document.forms[0].elements[0].focus();
}
```

如果第一个字段不是隐藏字段此方法就是凑效的,如果是的话把 elements 的下标改成非隐藏字段的下标即可.

5.控制表单只被提交一次

由于 Web 的响应问题，用户有可能会点击多次提交按钮从而创建重复数据或是导致错误，可以使用 JS 对提交按钮进行设置以让表单只被提交一次。

<input type="submit" value="提交" onclick="this.disabled=true；this.form.submit()"/>

这里在点击提交按钮时执行了两句 JS 代码，一次是 this.disabled=true；这是让提交按钮被禁用；一次是 this.form.submit()这是提交这个按钮所在的表单。

7.检查用户在表单元素中的按键

为控件添加 onkeydown 事件处理，然后在函数查看 keyCode，就能知道用户的按键，代码如下：

```
<input type="text" name="test" value="" onkeydown="testkey(this,event)"/>
function testkey(obj,event){
    alert(event.keyCode);//显示按键代码
}
```

这种技巧在改善用户体验如按回车键提交表单时很常用。

DOM 模型

一、概述

DOM(Document Object Module即文档对象模型)，它**定义了操作文档对象的接口**。在这个模型中，**一个文档被看成是结构化的数据**，对于 XML 就像一颗倒立的树的结构，树中的每个节点对应一个 XML 表记，都是一个对象。

在 Ajax 中，DOM 模型是最核心的结构。是所有 Ajax 开发的基础.如果没有 DOM 模型，就没有办法在客户端改变页面的内容，所有的局部刷新、异步请求也就无从实现，只有熟练掌握了 DOM 模型的相关技术，才算真正掌握了 Ajax 开发精髓

DOM 模型是从 DHTML 发展而来的。它经历了 DOM1,DOM2,DOM3. DOM1 主要针对 HTML 文档，而 DOM2,DOM3 完整定义了对 XML 文档对象的操作.

DHTML 即动态 HTML,它是将 HTML 元素看作对象，并通过定义对象的接口来动态改变这些对象的状态。

DOM 模型是从 DHTML 发展而来的，但它对模型作了根本性的改变。在 DMO2 及其后续版本中，根据 XML 文档结构的性质制定了一系列标准接口，也同样适用于 HTML.然而，**在 IE 浏览器中，一些结构化标记在 DOM 模型中不能很好的支持**，如下拉列表，表格等，需要使用 DOM1 来对它们进行操作。

二、节点

1. 节点概念：

在 DOM 模型中，整个文档是一颗倒立的树。树的根节点是 document 对象，表示整个文档对象，并且它包含一个子节点<html>.例如：

<html>

```
<head>
    <title> new document </title>
</head>
<body>
    Hello Ajax
</body>
</html>
```

在 DOM 模型中，整个文档是由层次不同的多个节点组成的，每个节点都可以看成是一**颗倒立的树，**整个文档是一个递归结构。

2. 节点类型

有三种类型的节点: 元素节点，文本节点和属性节点

在 XML(HTML)文档中，不仅每个闭合的标记是一个节点，而且闭合标记中的文本、标记内的属性也是节点，分别称为元素节点，文本节点和属性节点。例如：

<Label for=”checkBox” >姓名</Label>.

整个内容构成了标记为 Label 的**元素节点**,

for=”checkBox” 为一个**属性节点**.

姓名 为**文本节点**

属性节点和文本节点都是标记为 **Label** 的子节点

三、处理 DOM 中的节点

1. 直接引用节点

1.1. document.getElementById(elementID)

在 HTML 文档中,每个标记都有一个 id 属性,而且具有唯一性,所以可以用该元素(标记)的 id 来获得其引用.

1.2. document.getElementsByTagName(tagName)

获得指定标记元素的集合,返回一个数组

1.3. document.getElementsByName(elementName)

获得指定名称的标记元素的集合,返回一个数组.如访问一组名称为 name=”hoby”的的<input type=”checkBox” name=”hoby”>标记.例如

```
<form action="">
    <input type="checkbox" name="hoby" value="bridge" />桥牌
    <input type="checkbox" name="hoby" value="game" />游戏
    <input type="checkbox" name="hoby" value="swimming" />游泳
    <br />
    input type="button" value="获得爱好" onclick="parseXml()" />
</form>
//遍历 hoby 元素， hobyRef 为一个数组
 var hobyRef=document.getElementsByName("hoby");
  alert(hobyRef.length);//显示元素个数
  for(var i=0;i<hobyRef.length;i++){
    if(hobyRef[i].checked){//如果被选择，则显示
      alert(hobyRef[i].value);
    }
  }
```

2. 间接引用节点

每个节点都有一个 childNodes 集合属性,其类型是数组对象.表示该节点的所有子节点的集合.这些子节点按照其在文档中出现的顺序排列,因此,可以通过索引来访问各个子节点.

2.1. 引用子节点

document.childNodes[index] 引用索引为 index 的子节点

document.firstChild 引用第一个子节点

document.lastChild 引用最后一个子节点

2.2. 引用父节点

element.parantNode

2.3. 引用兄弟节点

element.nextSibling

element.previousSibling

3 获取节点信息

3.1. 使用 nodeType 属性获取节点类型.

node.nodeType 返回值及其意义如下:

1 表示元素节点

2 表示属性节点

3 表示文本节点

3.2. 使用 nodeName 属性获取节点名称

node.nodeName

对于元素节点,返回标记的名称;

对于属性节点,返回属性的名称;

对于文本节点,返回文本的内容

3.3. 使用 nodeValue 属性获取节点的值

node.nodeValue

对于元素节点,返回 null;

对于属性节点,返回 undefined;

对于文本节点,返回文本内容

4. 处理属性节点

4.1. 添加或修改一个属性的值

elemenetNode.setAttribute(attrName,attrValue)

elemenetNode.attrName=Value

4.2. 获得一个属性的值

elemenetNode.getAttribute(attrName)//返回属性 attrName 的值

var value=elemenetNode.attrName /返回属性 attrName 的值

5. 处理文本节点

用 innerHTML 属性 可以获得/设置一个节点内的文本.

例如:

中国人民

上面的 标记,包含了两个子节点：元素节点<img> 和文本节点 ”中国人民”
var div1=document.getElementById(“div1”);
var txtNode=div1.childNodes[1] 引用文本节点 ,进而可以通过 nodeValue 属性读取或改变它的值.

6．创建节点

6.1. 创建元素节点

document.createElement(elementName)
例：创建一个标记 div document.createElement(“div”);

6.2. 创建文本节点

document.createTextNode(textName)
document.createTextNode (“你好”);//创建一个文本节点“你好”
document.createTextNode (“<hello>”);//创建一个文本节点<hello>

例：创建一个<input>标记元素,并设置相关的属性

```
var check=document.createElement("Input");//产生一个标记元素”<input>”
//设置元素的属性
check.type="checkBox";
check.name="IDCheck";//check.id="IDCheck"
check.value="Value "+i;
check.onclick=new Function("checkedClick(this)");//关联响应函数
```

7.添加节点

7.1. 为父节点添加子节点

parentElement.appendChild(childElement) 返回对新节点的引用

```
<dl id=”dl1”>
    <dt>dt1</dt>
    <dd>dd1</dd>
</dl>
<script type=”text/javascript”>
        var dl1=document.getElementById(“dl1”);//获得对”dl1”的引用
        var newDd=document.createElement(“dd”);//创建新元素<dd>
        var newText=document.createTextNode(“dd2”);//创建新文本节点 dd2
        newDd.appendChild(newText);//为 new Dd 添加新文本(作为子节点)
        dl1.appendChild(newDd);//将 newDd 作为 dl1 的子节点
</script>
```

7.2. 在父节点的子节点中插入新的节点

parentElement.insertBefore(newNode,refrenceNode)
将 newNode 插入到旧节点 refrenceNode 之前，并返回对新节点的引用

8．取代子节点

parentElement.replaceChild(newNode,oldNode)

用 newNode 节点取代 oldNode 节点。

9.复制节点

node.cloneNode(includeChildren)

node 表示要复制的节点。IncludeChildren=true 表示要复制子节点

返回复制到的新节点的引用。

10.删除父节点的子节点

parentElement.removeChild(childNode) 返回被删除节点的引用。

DOM 节点(node)常用属性和方法		
属性/方法	类型/返回类型	说明
nodeName	String	节点名称
nodeValue	String	节点值，由节点类型决定
nodetType	Number	节点类型：1—元素节点，2—属性节点，3—文本节点
firstChild	Node	节点的第一个子节点
lastChild	Node	节点的最后一个子节点
childNodes	NodeList	节点的所有子节点集合
hasChildNodes()	Boolean	本节点有子节点，返回 true,
parentNode	Node	节点的父节点，如果本节点为根节点，则 parentNode 为 null
previousSibling	Node	节点的前一个兄弟节点
nextSibling	Node	节点的后一个兄弟节点
attributes	NameNodeMap	节点的所有属性映射集合
appendChild(node)	Node	在本节点的子节点末尾处添加一个子节点 node
insertBefore(nwNode,oldNode)	Node	在本节点的子节点 oldNode 之前插入一个新节点 newNode
replaceChild(nwNode,oldNode)	Node	将本节点的子节点 oldNode 替换为新节点 newNode
removeChild(node)	Node	在本节点的子节点中删除节点 node

访问页面元素(支持多浏览器)

一、节点的类型：

文档节点（document node ），

元素节点(Element node) ，

文本节点(text node) ，

属性节点(attribute node) 。

文档节点代表了文档本身，也是 DOM 树的根。
元素节点代表了 HTML 文档中任何一个标签。
文本节点代表了一个元素标签内部的文本。
属性节点代表了一个开放的元素标签内部所指定的属性。
每个页面都有一个文档节点，其他的节点都源自于这个节点。通过访问元素的节点、文本节点、属性节点，页面中的所有元素都可以被 Javascript 程序访问。

例：

```
<html>
<head>
    <title>如何访问页面元素</title>
</head>
<body>
    <h1 id='title'>节点的类型</h1>
    <p class="top">文本节点</p>
</body>
</html>
```

文档节点：document
元素节点：document、html 、title、head、body、h1、p
文本节点：如何访问页面元素、节点的类型、文本节点
属性节点：id 、class

二、访问页面元素

在这里举个例子：

```
<ul id='aa'>
<li>    <a href='aasirius.html'>sirius</a></li>
<li>    <a href='aacanopus.html'>canopus</a></li>
<li>    <a href='aaArcturus.html'>arcturus</a></li>
<li>    <a href='aavega.html'>vega</a></li>
</ul>

<ul id='bb'>
<li>     <a href='bbsirius.html'>sirius</a> </li>
<li>    <a href='bbcanopus.html'>canopus</a> </li>
<li>    <a href='bbArcturus.html'>arcturus</a> </li>
<li>    <a href='bbvega.html'>vega</a><br />  </li>
</ul>
```

方法一：通过函数 document.getElementById(id)
例如：document.getElement('aa');

方法二：通过函数 document.getElementsByTagName(tagName)
例如：document.getElementsByTagName("a");

如果要访问 ID='bb' 的 ul 下的所有 a,则可以和方法一组合用

```
var starList = document.getElementById('bb');//引用 ID='bb' 的 ul
var starAnchors = starList.getElementsByTagsName('a');
```

说明 1：如果获得 ID='bb' 的 ul 下的所有直接子元素<li>,则可以如下：

```
var childLI = starList.children;
```

同时可以用 starList.children(i) 访问第 i 个

说明 2：

在早期的浏览器支持：document.body、document.images、document.forms，那么现在浏览器都向 W3C 靠拢，这些方法变得不可靠。现在，我们可以用 var body = bodydocument.getElementsByTagName("body")[0];

方法三：通过 DOM 方法访问

先介绍一下几个常用的方法：

- *node.childNodes 指定节点的所有子节点，包括文本节点和其他所有元素*
- *node.firstChild 指定节点的第一个子节点。*
- *node.lastChild 指定节点的最后一个子节点。*
- *node.parentNode 指定节点的上级节点。*
- *node.nextSibling 指定节点的下一个兄弟节点。*
- *node.perviousSibling 指定节点的上一个兄弟节点。*
- *node.nodeName 指定节点的节点名字*
- *node.nodeType 指定节点的类型。值 = 1 是元素节点。 值 = 3 是文本节点。*

上面有一些方法只对元素节点作用，对文本节点无作用，所以在这里要强调一点，***在做节点操作之前，最好对节点的类型进行一次判断。是否为元素节点。***

对于一些文本描述的 DOM 结构（例如 HTML 文档）,一些浏览器(如 FireFox)会在元素节点之间插入一些空白节点，空白节点实际上就文本节点，不过只是包含一些空格、或者 Tab、制表符。通过上面提到的属性进行节点访问的时候，就要注意到这些空格（文本节点)。有两种办法来辨别一个节点是元素节点还是文本节点？文本节点的属性 nodeName 的值总会是:#text，另外就是通过上面提到的用 nodeType 判断。

例:

要访问 <ul id='aa'> 中的元素节点 a 中的 href 值。

```
var starList = document.getElementById('bb');
var perStarList = starList.previousSibling;//获得前一个兄弟节点<ul>的引用
if(perStarList.nodeType == 3){
    perStarList = perStarList.previousSibling;
}
var perAnchors = perStarList.getElementsByTagName('a');<ul id='aa'>的所有<a>
for(var i = 0; i < perAnchors.length; i ++){
    alert(perAnchors[i].href);
}
```

如果没有以前的判断，则在 FireFox 无法测试成功。因为在两个节点 ul 中插入了一个文本

节点。

三、创建元素节点和文本节点

1.创建元素节点：

createElement(eleName) 就是创建新元素的函数，需要提供的只是一个参数----指明想创建元素的类型(*在某种意义上可理解成是标签*),返回值则指向创建的新元素(也可以省略返回值)。

例如：

var newAnchor = document.createElement("a");//创建元素<a>,注意不带<,>符号

2.创建文本节点：

createTextNode(textContext) 就是创建文本节点的函数。一个元素内部的文本实际上就是文本节点，所以这里的和创建元素的函数不一样。

例如：

var anchorText = document.createTextNode("创建文本节点");

注意：在这里补充一个文本的节点的固有的属性 nodeValue，可以通过这个属性来访问、更改文本节点的文本。

例如：

```
var textNode = document.createTextNode("创建文本节点");
var oldText = textNode.nodeValue;
textNode.nodeValue = "更改文本节点";//修改旧值
```

3.如何插入创建 的节点(元素节点和文本节点)

当节点创建文成，我们就需要把节点插入到文档(HTML 文档)中，那么在这里有三个非常重要函数。

1、appendChild(childNode)：将一个节点添加到*某元素的子节点列表的最后*。所以这个函数的*某元素的一个方法*，参数是新插入的节点。

例：

```
<p id = "starLinks">
<a href="create.html">创建文本</a>
</p>
```

现在对用 DOM 的函数创建和插入一个链接。

```
var anchorText = document.createTextNode('插入节点');

var newAnchor = document.createElement('a');
newAnchor.appendChild(anchorText);//将创建的文本节点，插入到元素节点 a 中

var parent = document.getElementById("starLinks");
parent.appendChild(newAnchor);//将元素节点，插入到 parent 中
```

此时如果要看 DOM 操作的结果，它大概应该是这样的

```
<p id = "starLinks">
    <a href="create.html">创建文本</a><a>插入节点</a>
</p>
```

2、insertBefore：将一个节点添加到*某元素的某个子节点列表的之前。*所以需要两个函数，一个待插入的节点，一个是现有的节点，它还是*某元素*的一个方法。

例子：

```
<p id = "starLinks">
    <a href="create.html">创建元素</a>
    <a href="create.html" id="sirius">创建文本</a>
</p>
```

用 insertBefore 操作

```
var anchorText = document.createTextNode("插入节点");

var newAnchor = document.createElement("a");
newAnchor.appendChild(anchorText);//将创建文本节点，插入到元素节点 a 中

var parent = document.getElementById("starLinks");
var existingAnchor = document.getElementById("sirius");
var newChild = parent.insertBefore(newAnchor,existingAnchor);//注意两个参数的先后顺序
```

操作后的

```
<p id = "starLinks">
    <a href="create.html">创建元素</a>
    <a>插入节点</a>
    <a href="create.html" id="sirius">创建文本</a>
</p>
```

3、replaceChild：替换某个节点，需要的也是两个参数，和 insertBefore 的用法差不多，见例子

在这里还是用上面的例子，下面 dom 操作部分

```
var anchorText = document.createTextNode("插入节点");

var newAnchor = document.createElement("a");
newAnchor.appendChild(anchorText);//将创建的文本节点，插入到元素节点 a 中

var parent = document.getElementById("starLinks");
var existingAnchor = document.getElementById("sirius");
var newChild = parent.replaceChild(newAnchor,existingAnchor);//和面的一样只不过函数变了
```

操作后:

```
<p id = "starLinks">
    <a href="create.html">创建元素</a>
    <a>插入节点</a>
</p>
```

四、改变元素的属性

改变元素的属性没有直接的办法，但是可以利用上面的几个函数，就可以做到。还是举例子说吧！

例子：

```
<p id = "starLinks">
    <a href="create.html">创建元素</a>
   <a href="create.html">插入节点</a>
    <a href="create.html" id="sirius">创建文本</a>
</p>
```

要想把上面的 p 改成 DIV，下面是 DOM 操作

```
var div= document.createElement("div");
var oldAnchor = document.getElementById("starLinks");

for(var i = 0; i < oldAnchor.childNodes.length; i ++){
    var clone = oldAnchor.childNodes[i].cloneNode(true);//cloneNode 函数
    div.appendChild(clone);
}

oldAnchor.replaceChild(div,oldAnchor);
```

cloneNode 函数，*完全创建一样的节点拷贝。参数 true 表明想克隆子节点本身。创建所有子节点的镜像。*

下面介绍另一个更为简洁的办法：

```
var div= document.createElement("div");
var oldAnchor = document.getElementById("starLinks");

while(oldAnchor.childNodes.length > 0){
    div.appendChild(oldAnchor.firstChild);
}

oldAnchor.replaceChild(div,oldAnchor);
```

上面的例子有一个注意点：

注意：当 DOM 中发生变化时，集合中的元素会自动更新，也就是说，在使用 appendChild 函数的时候，已经把子节点，从 oldAnchor 中移出，造成每次循环的时候，oldAnchor 中的子节点逐渐减少。因为每次节点重新定位，所有不能用 for 循环，for 循环过程中是假设集合的内容不会发生变化。

五、删除一个元素节点或者文本节点

这个主要是通过函数 **removeChild** 来实现，返回被删除的对象。这个函数的使用比较简单。

例子：

```
<p>
    <a href="create.html">创建元素</a>
   <a href="create.html">插入节点</a>
    <a href="create.html" id="sirius">创建文本</a>
</p>
```

DOM 删除

```
var anchor = document.getElementById("sirius");
var parent = anchor.parentNode;
var removed = parent.removeChild(anchor);
```

操作后

```
<p>
    <a href="create.html">创建元素</a>
   <a href="create.html">插入节点</a>
</p>
```

如果要删除父类节点，但同时想保留子节点。可以先把子节点先 clone 出来，或者使用 insertBfore 把子节点插入到父类节点之前。

今天到此，明天介绍一下 关于元素属性的读和写。

HTML 元素最为常见的部分就是它的属性，例如 id 、class、href、tittle 或者其他各种属性。Javascript 不仅能读取这些属性值，而且还能写回新值。

六、读写元素的属性

元素的读写主要用到了两个函数，他们分别是

getAttribute(attrName) *可以用于读取一个属性的值，*

setArrtibute(attrName,newValue) *则可以用于写入新值。*

举例子吧！

```
<a id="antares" href="element.html">元素属性介绍</a>
```

利用这两个函数对上面的元素节点进行操作。

```
var anchor = document.getElementById("antares");
var anchorId = anchor.getAttribute('id');
var anchorHref = anchor.getAttribute("href");
document.write(anchorId,anchorHref);
//IE 输出的结果是:antareshttp://127.0.0.1/element.html (IE)
//FireFox 的结果:antareselement.html(firefox)
anchor.setAttribute('id',"newAntares");
anchor.setAttribute("href","newelement.html");
anchor.setAttribute("title","新元素属性");
var newAnchor = document.getElementById("newAntares");
var newAnchorId = newAnchor.getAttribute("id");
var newAnchorHref = newAnchor.getAttribute("href");
var newAnchorTitle = newAnchor.getAttribute("title");
document.write(newAnchorId,newAnchorHref,newAnchorTitle);
//FireFox 和 IE 输出相同 newAntaresnewelement.html 新元素属性
```

通过上面的例子：很容易掌握这两个函数的用法,getAttribute,setAttribute,同时也发现了一些 IE 和 FireFox 的结果不一样.

下面就关于在不同浏览器下读写元素的几个应该注意的点：

1、对元素的读写还有一个更直接的办法，比如说读取 href 属性，直接就 Element.href。Element.title，如果是写入的话 element.href='title.html' 等等，但是这些都是在确定元素属性的基础上进行的操作。如果元素的某个属性不存在，则会出现一些比较困扰的问题。

2、鉴于不同的浏览器，个人建议，*在读取元素的时候用直接读取的办法比较好。就是 element.href 的方法。在写入属性的属性的时候，用函数 setArrtibute 比较好*。

3、因为 style 和 class 对于任何元素都是确定存在的，所以可以直接访问，element.style 和 element.className，不过在使用 getAttribute('class')(firefox)，而且在 IE 中则 getAttribute("className").setAttribute 类似.个人提议，这两个属性请使用直接访问 element.style 和 element.className，来保持不用的浏览器统一。

4、有一个要值得当心的属性，label，它的直接访问形式 element.htmlFor，在使用 getAttribute('for')(firefox)，IE 中使用 getAttribute("htmlFor"),setAttribute 类似.

七、获得拥有特定属性值的所有元素

这是元素读取的一个简单应用，比如说要在 HTML 文档中，查找所有的复选框(即 type = "checkbox")。那么我们就可以这样做：先获得 input 的元素节点，这样可以缩小查找范围。

```
var inputs = document.getElementsByTagName("input");
for(var i = 0; i < inputs.length; i ++){
     if(inputs[i],getAttribute("type") == "checkbox"){
         //处理复选框
     }
     else{
        //处理非复选框
     }
}
```

自定义的函数获得特定属性值的所有元素：

```
function getElementsByAttribute(attribute,attributeValue){
    var elementArray = new Array();
    var matchedArray = new Array();

    if(document.all){
       elementArray = document.all;
    }else{
       elementArray = document.getElementsByTagName("*");
    }

    for(var i = 0 ; i < elementArray.length; i ++){
       if(attribute == "class"){
          var pattern = new RegExp("(^| )"+attributeValue+"( |$");
```

```
          if(pattern.test(elementArray[i].className)){
             matchedArray[matchedArray.length] = elementArray[i];
           }
        }else if(attribute == "for"){
             if(elementArray[i].getArrtibute("htmlFor") ||
elementArray[i].getArrtibute("for")){
                if(elementArray[i].htmlFor == attributeValue){
                    matchedArray[machedArray.length] = elementArray[i];
                }
             }
        }else if(elementArray[i].getAttribute(attribute) == attributeValue){
             matchedArray[matchedArray.length] = elementArray[i];
        }
    }
    return matchedArray;
}
```