康康的JavaScript初识

徐振康 整理

目录

[第一章 JavaScript基础 3](#_Toc442129848)

[1.1 浏览器的对象树 3](#_Toc442129849)

[1. window对象常用方法 3](#_Toc442129850)

[2. Form表单对象 3](#_Toc442129851)

[1.2 javaScript定义函数的三种方式 3](#_Toc442129852)

[1.正常方法 3](#_Toc442129853)

[2.构造函数方法 new Function(); 4](#_Toc442129854)

[3.函数直接量定义函数 4](#_Toc442129855)

[第二章 DOM 4](#_Toc442129856)

[2.1 基本概念 4](#_Toc442129857)

[2.2 DOM的结构 5](#_Toc442129858)

[2.3 节点及其类型 5](#_Toc442129859)

[2.4 DOM的属性 9](#_Toc442129860)

[2.5 节点的操作 11](#_Toc442129861)

[第三章 相关案例 17](#_Toc442129862)

[3.1 显示弹出窗口 17](#_Toc442129863)

[附录 20](#_Toc442129864)

# 第一章 JavaScript基础

## 1.1 浏览器的对象树

1. navigator

2. window

**document**

**(文档对象)**

frame

(框架对象)

location

(位置对象)

history

(历史对象)

links 链接对象<a></a>

archors 锚对象

**forms 表单对象**

**images 图片对象**

### 1. window对象常用方法

alert(‘信息’) ：消息框

prompt(‘提示信息’,默认值)：标准输入框

confirm( )：确认框

open( )：打开一个新窗口

close( )：关闭窗口

### 2. Form表单对象

访问表单的方式:

document.forms[n]

document.表单名字

表单对象常用的属性

action <from action=”xxx”> 表单提交的目的地址

method <form method=”xxx”> 表单提交方式

name <form name=”xxx”> 表单名称

## 1.2 javaScript定义函数的三种方式

### 1.正常方法

|  |
| --- |
| function print(msg){  document.write(msg);  } |

对函数进行调用的几种方式：

（1）函数名（传递给函数的参数1，传递给函数的参数2，….）

（2）变量 = 函数名（传递给函数的参数1，传递给函数的参数2，….）

（3）对于有返回值的函数调用，也可以在程序中直接使用返回的结果，例如：alert("sum=” + square(2,3));

（4）不指定任何函数值的函数，返回undefined。

### 2.构造函数方法 new Function();

|  |
| --- |
| //构造函数方式定义javascript函数 注意Function中的F大写  var add=new Function('a','b','return a+b;');  //调用上面定义的add函数  var sum=add(3,4);  alert(sum); |

**注:接受任意多个字符串参数，最后一个参数是函数体。如果只传一个字符串，则其就是函数体。**

### 3.函数直接量定义函数

|  |
| --- |
| //使用函数直接量的方式定义函数  var result=function(a,b){return a+b;}  //调用使用函数直接量定义的函数  var sum=result(7,8);  alert(sum); |

**注:函数直接量是一个表达式，它可以定义匿名函数**

# 第二章 DOM

## 2.1 基本概念

DOM：DOM是Document Object Model文档对象模型的缩写。根据**W3C DOM**规范,DOM是一种与浏览器，平台，语言无关的接口，使得你可以访问页面其他的标准组件。

D：文档 – html 文档 或 xml 文档

O：对象 – document 对象的属性和方法

M：模型

DOM 是针对xml(html)的基于树的API。

DOM树:节点（node）的层次。

DOM 把一个文档表示为一棵家谱树（父，子，兄弟）

DOM定义了Node的接口以及许多种节点类型来表示XML节点的多个方面

## 2.2 DOM的结构

DOM的结构与家谱的结构及其类似。

爷爷

父辈

父辈

老大

老二

老大

DOM的结构

|  |
| --- |
| <html>  <head>  <title>HTML DOM</title>  </head>  <body>  <h1>DOM的结构</h1>  <p><a href=”href”>链接</a></p>  </body>  </html> |

它的DOM结构如下：



## 2.3 节点及其类型

1.节点

（1）由结构图中我们可以看到，整个文档就是一个**文档节点**。

（2）而每一个HMTL标签都是一个**元素节点**。

（3）标签中的文字则是**文本节点**。

（4）标签的属性是**属性节点**。

（5）一切都是节点……

2.节点树

节点树的概念从图中一目了然，最上面的就是“树根”了。节点之间有父子关系，祖先与子孙关系，兄妹关系。这些关系从图中也很好看出来，直接连线的就是父子关系了。而有一个父亲的就是兄妹关系……

3. NODE接口的特性和方法

|  |  |  |
| --- | --- | --- |
| **特性/方法** | **类型/返回类型** | **说    明** |
| nodeName | String | 节点的名字；根据节点的类型而定义 |
| nodeValue | String | 节点的值；根据节点的类型而定义 |
| nodeType | Number | 节点的类型常量值之一 |
| ownerDocument | Document | 指向这个节点所属的文档 |
| firstChild | Node | 指向在childNodes列表中的第一个节点 |
| lastChild | Node | 指向在childNodes列表中的最后一个节点 |
| childNodes | NodeList | 所有子节点的列表 |
| previousSibling | Node | 指向前一个兄弟节点；如果这个节点就是第一个兄弟节点，那么该值为null |
| nextSibling | Node | 指向后一个兄弟节点；如果这个节点就是最后一个兄弟节点，那么该值为null |
| hasChildNodes() | Boolean | 当childNodes包含一个或多个节点时，返回真 |
| attributes | NamedNodeMap | 包含了代表一个元素的特性的Attr对象；仅用于Element节点 |
| appendChild(*node*) | Node | 将*node*添加到childNodes的末尾 |
| removeChild(*node*) | Node | 从childNodes中删除*node* |
| replaceChild(*newnode*, *oldnode*) | Node | 将childNodes中的*oldnode*替换成*newnode* |
| insertBefore(*newnode*, *refnode*) | Node | 在childNodes中的refnode之前插入newnode |

4. 查找元素节点

（1）getElementById()

寻找一个有着给定 id 属性值的元素，返回值是一个有着给定 id 属性值的元素节点。如果不存在这样的元素，它返回 null.

var oElement = document.getElementById ( sID )

该方法只能用于 document 对象

|  |
| --- |
| <input type=”text” value=”康康" id="tid">  function test(){  var usernameElement=document.getElementById(“tid");  //获取元素的值  alert("usernameElement.value: "+usernameElement.value)  //获取元素的类型  alert("usernameElement.type: "+usernameElement.type)  } |

（2）getElementsByName()

寻找有着给定name属性的所有元素，这个方法将返回一个节点集合，这个集合可以当作一个数组来处理。这个集合的 length 属性等于当前文档里有着给定name属性的所有元素的总个数。

|  |
| --- |
| <form name="form1">  <input type="text" name="tname" value="国庆60年\_1" /><br>  <input type="text" name="tname" value="国庆60年\_2" /><br>  <input type="text" name="tname" value="国庆60年\_3" /><br>  <input type="button" name="ok" value="保存" id="ok" onclick="test();">  </form>  function test(){  var tnameArray=document.getElementsByName("tname");  alert(tnameArray.length);  for(var i=0;i<tnameArray.length;i++){  window.alert(tnameArray[i].value);  }  } |

|  |
| --- |
| <input type="text" name="username" value="国庆60年\_1" /><br>  <input type="text" name="username" value="国庆60年\_2" /><br>  <input type="text" name="username" value="国庆60年\_3" /><br>  <input type="button" name="ok" value="保存" id="ok"><br>  <script language="JavaScript">  //该方法返回是数组类型  var usernameElements=document.getElementsByName("username");  for (var i = 0; i < usernameElements.length; i++) {  //获取元素的类型  //alert(usernameElements[i].type)  //获取元素value的值  //alert(usernameElements[i].value);  //采用函数直接量的方法  usernameElements[i].onchange = function(){  alert(this.value);  }  }  </script> |

（3）getElementsByTagName()

寻找有着给定标签名的所有元素，这个方法将返回一个节点集合，这个集合可以当作一个数组来处理。这个集合的 length 属性等于当前文档里有着给定标签名的所有元素的总个数。

var elements = document.getElementsByTagName(tagName);

var elements = element.getElementsByTagName(tagName);

该方法不必非得用在整个文档上。它也可以用来在某个特定元素的子节点当中寻找有着给定标签名的元素。

var container = document.getElementById(“sid”);

var elements = container.getElementsByTagName(“p”);

alert(elements.length);

|  |
| --- |
| // //处理input  // var inputElements=document.getElementsByTagName("input");  // //输出input标签的长度  // //alert(inputElements.length);  // for(var i=0;i<inputElements.length;i++){  // if(inputElements[i].type!='button'){//submit  // alert(inputElements[i].value);  // }  // }  //处理select  // //获取select标签  // var selectElements=document.getElementsByTagName("select");  // //获取select下的子标签  // for(var j=0;j<selectElements.length;j++){  // var optionElements=selectElements[j].getElementsByTagName("option");  // for(var i=0;i<optionElements.length;i++){  // alert(optionElements[i].value);  // }  // }  var textareaElements=document.getElementsByTagName("textarea");  alert(textareaElements[0].value); |

|  |
| --- |
| var inputElements=document.getElementsByTagName("input");  for(var i=0;i<inputElements.length;i++){  if (inputElements.type != 'submit') {  inputElements[i].onchange = function(){  alert(this.value)  };  }  var selectElements=document.getElementsByTagName("select");  for (var i = 0; i < selectElements.length; i++) {  selectElements[i].onchange=function(){  alert(this.value);  }  } |

|  |
| --- |
| var textareaElements=document.getElementsByTagName("textarea");  for (var i = 0; i < textareaElements.length; i++) {  textareaElements[i].onchange = function(){  alert(this.value);  };  } |

5. 查看是否存在子节点

hasChildNodes()

（1）该方法用来检查一个元素是否有子节点，返回值是 true 或 false.

var booleanValue = element.hasChildNodes();

（2）文本节点和属性节点不可能再包含任何子节点，所以对这两类节点使用 hasChildNodes 方法的返回值永远是 false;

（3）如果 hasChildNodes 方法的返回值是 false，则 childNodes,firstChild,lastChild 将是空数组和空字符串。

|  |
| --- |
| var selectElements=document.getElementsByTagName("select");  alert(selectElements[0].hasChildNodes())  var inputElements=document.getElementsByTagName("input");  for(var i=0;i<inputElements.length;i++){  alert(inputElements[i].hasChildNodes());  } |

## 2.4 DOM的属性

1. nodeName

文档里的每个节点都有以下属性。

nodeName:一个字符串，其内容是给定节点的名字。

var name = node.nodeName;

\* 如果节点是元素节点，nodeName返回这个元素的名称

\* 如果是属性节点，nodeName返回这个属性的名称

\* 如果是文本节点，nodeName返回一个内容为#text 的字符串

注:nodeName 是一个只读属性。

2. nodeType

nodeType：返回一个整数，这个数值代表着给定节点的类型。

nodeType 属性返回的整数值对应着 12 种节点类型,常用的有三种：

Node.ELEMENT\_NODE ---1 -- 元素节点

Node.ATTRIBUTE\_NODE ---2 -- 属性节点

Node.TEXT\_NODE ---3 -- 文本节点

**nodeType 是个只读属性**

3. nodeValue

nodeValue：返回给定节点的当前值（字符串）

如果给定节点是一个属性节点，返回值是这个属性的值。

如果给定节点是一个文本节点，返回值是这个文本节点的内容。

如果给定节点是一个元素节点，返回值是 null

nodeValue 是一个 读/写 属性，但不能对元素节点的 nodeValue 属性设置值，

但可以为文本节点的 nodeValue 属性设置一个值。

var li = document.getElementById(“li”);

if(li.firstChild.nodeType == 3)

li.firstChild.nodeValue = “康康康康”;

[练习1]

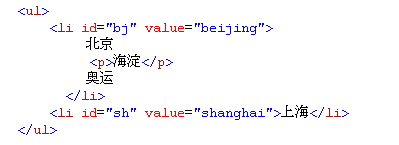


问题:

打印 ”明天休息” (利用两种方法)

提示:使用(firstChild lastChild childNodes)

[练习2]

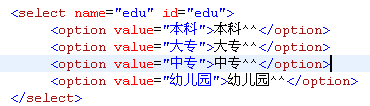


问题:

打印出 id=“bj” 该节点的所有子节点的(nodeName, nodeType, nodeValue)

同时打印文本值 北京 海淀 奥运

[练习3]



问题:

输出所有select元素下的所有option元素中对应的文本内容

例如:<option value="中专">中专^^</option> 输出--->中专^^

元素节点

|  |
| --- |
| //测试元素节点,输出节点名称,节点的类型,节点的值  var liElements=document.getElementsByTagName("li");  for(var i=0;i<liElements.length;i++){  alert(liElements[i].nodeName);  alert(liElements[i].nodeType);  alert(liElements[i].nodeValue);  } |

属性节点

|  |
| --- |
| //测试属性节点,输出属性节点名称,节点的类型,节点的值  var liElements=document.getElementsByTagName("li");  for(var i=0;i<liElements.length;i++){  var attrElement=liElements[i].getAttributeNode("value")  alert("attrElement.nodeName "+attrElement.nodeName);  alert("attrElement.nodeType "+attrElement.nodeType);  alert("attrElement.nodeValue "+liElements[i].getAttribute("value"));  } |

文本节点

|  |
| --- |
| //测试元素节点,输出节点名称,节点的类型,节点的值  var liElements=document.getElementsByTagName("li");  for(var i=0;i<liElements.length;i++){  alert(liElements[i].childNodes[0].nodeName);  alert(liElements[i].childNodes[0].nodeType);  alert(liElements[i].childNodes[0].nodeValue);  liElements[i].childNodes[0].nodeValue="南京";  alert(liElements[i].childNodes[0].nodeValue);  //另一种读取方法  alert(liElements[i].firstChild.nodeName);  alert(liElements[i].firstChild.nodeType);  alert(liElements[i].firstChild.nodeValue);  } |

## 2.5 节点的操作

1. 替换节点

replaceChild()

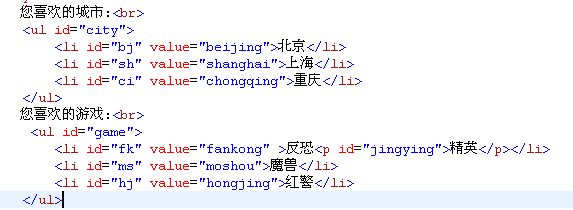
把一个给定父元素里的一个子节点替换为另外一个子节点

var reference = element.replaceChild(newChild,oldChild);

返回值是一个指向已被替换的那个子节点的引用指针。

如果被插入的子节点还有子节点，则那些子节点也被插入到目标节点中。

[练习4]



问题:当单击“北京”这个节点时,北京这个节点被“反恐精英”替换

替换节点例子

|  |
| --- |
| //方法一  // var cityElement=document.getElementById("city");  // var loveElement=document.getElementById("love");  // var cityChildElement=document.getElementById("beijing");  // var loveChildElement=document.getElementById("fankong");  // var oldElement=cityElement.replaceChild(loveChildElement,cityChildElement);  // loveElement.appendChild(oldElement);  // alert(oldElement.getAttribute("id"));  var cityElement=document.getElementById("city");  cityElement.onclick=function(){  var cityChildElement=document.getElementById("beijing");  var loveChildElement=document.getElementById("fankong");  var oldElement=cityElement.replaceChild(loveChildElement,cityChildElement);  loveElement.appendChild(oldElement);  alert(oldElement.getAttribute("id"));  } |

2. 查找属性节点

getAttribute()

返回一个给定元素的一个给定属性节点的值

var attributeValue = element.getAttribute(attributeName);

给定属性的名字必须以字符串的形式传递给该方法。

给定属性的值将以字符串的形式返回，如果给定属性不存在，getAttribute() 将返回一个空字符串.

通过属性获取属性节点

getAttributeNode(属性的名称)--Node

|  |
| --- |
| <li name="beijing" id="bj">北京</li>    //通过属性名获取属性的值  **var** bjElement=document.getElementById("bj");  **var** attributeValue=eduElement.getAttribute("name");  alert("attributeValue "+attributeValue);    //通过属性名获取属性的节点  **var**  bjNode=eduElement.getAttributeNode("name");  alert(eduNode.nodeValue);  alert(eduNode.nodeType);  alert(eduNode.nodeName); |

3. 设置属性节点

setAttribute()

将给定元素节点添加一个新的属性值或改变它的现有属性的值。

element.setAttribute(attributeName,attributeValue);

属性的名字和值必须以字符串的形式传递给此方法

如果这个属性已经存在，它的值将被刷新；

如果不存在，setAttribute()方法将先创建它再为其赋值。

|  |
| --- |
| <li id="bj" >北京</li>  //获取元素的引用  var bjElement=document.getElementById("bj");  //设置属性值  bjElement.setAttribute("name","beijing");  //获取设置的属性值  var nameValue=bjElement.getAttribute("name");  alert("nameValue "+nameValue); |

4. 创建新元素节点

createElement()

按照给定的标签名创建一个新的元素节点。方法只有一个参数：将被创建的元素的名字，是一个字符串.

var reference = document.createElement(element);

方法的返回值：是一个指向新建节点的引用指针。返回值是一个元素节点，所以它的 nodeType 属性值等于 1。

新元素节点不会自动添加到文档里，新节点没有 nodeParent 属性，它只是一个存在于 JavaScript 上下文的对象.

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

|  |
| --- |
| //创建一个新的元素  var pElement=document.createElement("li");  //设置属性值  pElement.setAttribute("id","pid");    //获取父元素  var loveElement=document.getElementById("love");  //在父元素中增加子元素  loveElement.appendChild(pElement);  //通过id获取刚创建的元素  var pidElement=document.getElementById("pid");  alert(pidElement.getAttribute("id")); |

5. 创建新文本节点

createTextNode()

创建一个包含着给定文本的新文本节点。这个方法的返回值是一个指向新建文本节点引用指针。

var textNode = document.createTextNode(text);

方法只有一个参数：新建文本节点所包含的文本字符串

方法的返回值：是一个指向新建节点的引用指针。它是一个文本节点，所以它的 nodeType 属性等于 3.

新元素节点不会自动添加到文档里，新节点没有 nodeParent 属性

var pElementText=document.createElement("li");

var textElement=document.createTextNode("南京");

pElementText.appendChild(textElement);

6. 插入节点

（1）appendChild()

为给定元素增加一个子节点：

var newreference = element.appendChild(newChild).

给定子节点 newChild 将成为给定元素节点 element 的最后一个子节点。

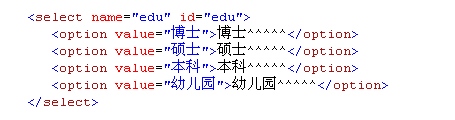
方法的返回值是一个指向新增子节点的引用指针。

该方法通常与 createElement() createTextNode() 配合使用

新节点可以被追加给文档中的任何一个元素

|  |
| --- |
| var newliElement=document.createElement("li");  var textNode=document.createTextNode("北京");  newliElement.appendChild(textNode);  document.body.appendChild(newliElement);    var liElement=document.getElementsByTagName("li");  var textValue=liElement[0].firstChild.nodeValue;  alert(textValue); |

[练习5]



问题: 在id=edu下增加<option value="大专">大专^^^^^</option>

（2）insertBefore()

把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

var reference = element.insertBefore(newNode,targetNode);

节点 newNode 将被插入到元素节点 element 中并出现在节点 targetNode 的前面.

节点 targetNode 必须是 element 元素的一个子节点。

该方法通常与 createElement() 和 createTextNode() 配合使用

|  |
| --- |
| <ul id="city"> <li value="beijing^" id="beijing">北京</li> </ul>  <ul id="city01"> <li value="shanghai^" id="shanghai">上海</li> </ul>  //获取父节点  var parentCityNode=document.getElementById("city");  //获取子节点  var beijingNode=document.getElementById("beijing");  var shanghaiNode=document.getElementById("shanghai");  //插入  parentCityNode.insertBefore(shanghaiNode,beijingNode); |

|  |
| --- |
| <ul id="city">  <li value="beijing^" id="beijing">北京</li>  </ul>  //获取父节点  var parentCityNode=document.getElementById("city");  //获取子节点  var beijingNode=document.getElementById("beijing");  //创建节点newShanghaiNode  var newShanghaiNode=document.createElement("li");  //创建newShanghaiNode节点的文本节点  var newTextNode=document.createTextNode("上海");  //在创建的节点上增加文本  newShanghaiNode.appendChild(newTextNode);  //插入节点  parentCityNode.insertBefore(newShanghaiNode,beijingNode); |

（3）DOM 没有提供 insertAfter() 方法

|  |
| --- |
| function insertAfter(newElement,targetElement){  //获取目标元素的父节点  var parentElement=targetElement.parentNode;  //如果目标元素是最后一个元素,则新元素插入到目标元素的后面  if(parentElement.lastChild==targetElement){  parentElement.appendChild(newElement);  }else{//如果目标元素不是最后一个元素,则新元素插入到目标元素的的  //下一个兄弟节点的前面,即目标元素的后面  parentElement.insertBefore(newElement,targetElement.nextSibling);  }  } |

7. 删除节点

removeChild()

从一个给定元素里删除一个子节点

var reference = element.removeChild(node);

返回值是一个指向已被删除的子节点的引用指针。

某个节点被removeChild()方法删除时,这个节点所包含的所有子节点将同时被删除。

|  |
| --- |
| <ul id="city"> <li value="beijing^" id="beijing">北京</li> </ul>  var ulElement=document.getElementById("city");  var liElement=document.getElementById("beijing");  ulElement.removeChild(liElement); |

如果想删除某个节点，但不知道它的父节点是哪一个，parentNode 属性可以帮忙。

|  |
| --- |
| <ul id="city"> <li value="beijing^" id="beijing">北京</li> </ul>  var liElement=document.getElementById("beijing");  var parentElement=liElement.parentNode;  parentElement.removeChild(liElement); |

8. 遍历节点树

ChildNodes：返回一个数组，这个数组由给定元素节点的子节点构成：

var nodeList = node.childNodes;

文本节点和属性节点都不可能再包含任何子节点，所以它们的 ChildNodes 属性永远会返回一个空数组。

如果想知道某个元素有没有子节点，可以用 hasChildNodes 方法。

如果想知道某个元素有多少个子节点，可以用 childNodes 数组的 length 属性。

childNodes 属性是一个只读属性。

9. 获取第一个子节点

firstChild：该属性返回一个给定元素节点的第一个子节点，返回这个节点对象的指针。

var reference = node.firstChild;

文本节点和属性节点都不可能包含任何子节点，所以它们的 firstChild 属性永远会返回 null。

某个元素的 firstChild 属性等价于这个元素的 childNodes 节点集合中的第一个节点，即：

var reference = node.ChildNodes[0];

firstChild 属性是一个只读属性。

10. 获取最后一个子节点

lastChild：对应 firstChild 的一个属性。

nextSibling: 返回一个给定节点的下一个兄弟节点。

parentNode：返回一个给定节点的父节点。

parentNode 属性返回的节点永远是一个元素节点，因为只有元素节点才有可能包含子节点。

document 节点的没有父节点。

previousSibling：返回一个给定节点的上一个兄弟节点

11. innerHTML属性

浏览器几乎都支持该属性，但不是 DOM 标准的组成部分。

innerHTML 属性可以用来读，写某给定元素里的 HTML 内容。

|  |
| --- |
| <div id="city"></div>  var divElement=document.getElementById("city");  divElement.innerHTML="<li value='beijing^'' id='beijing'>北京</li>"; |

# 第三章 相关案例

## 3.1 显示弹出窗口

方案一

语法：

vReturnValue = window . showModalDialog ( sURL, vArguments , sFeatures )

参数说明：

sURL--必选参数,用来指定对话框要显示的文档的URL。

vArguments--可选参数，用来向对话框传递参数。传递的参数类型不限，包括数组等。对话框通过window.dialogArguments来取得传递进来的参数。

sFeatures--可选参数，用来描述对话框的外观等信息，可以使用一个或几个，用分号“;”隔开。

方案二

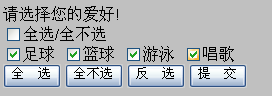
window.opener 的用法

window.opener 返回的是创建当前窗口的那个窗口的引用，比如点击了a.htm上的一个链接而打开了b.htm，然后我们打算在b.htm上输入一个值然后赋予a.htm上的一个id为“name”的text中，就可以写为：   
    window.opener.document.getElementById("name").value = "输入的数据";

[练习6]

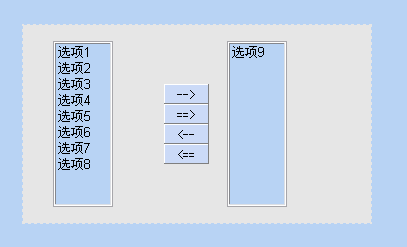


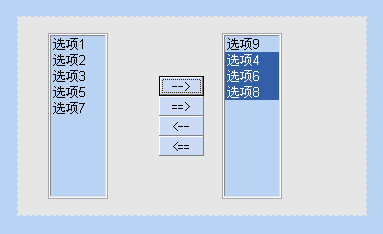
[练习7]



当 checkbox 节点有 checked 属性时, 在 HTML 页面上即表现为 "被选中"；利用 setAttribute("checked", "checked"); 实现选中,setAttribute("checked",null); 实现不被选中；也可以利用 document.getElementById("checkedAll\_2").checked = "checked";实现选中, 利用 document.getElementById("checkedAll\_2").checked = null; 实现不被选中。

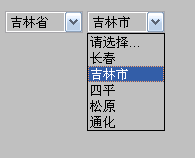
[练习8]

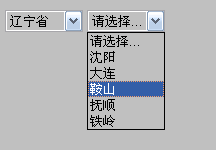




[练习9]





|  |
| --- |
| function parseXML(filename){  var xmlDoc;  try { //Internet Explorer  xmlDoc = new ActiveXObject("Microsoft.XMLDOM");  } catch (e) {  try { //Firefox, Mozilla, Opera, etc.  xmlDoc = document.implementation.createDocument("", "", null);  } catch (e) { }  }  //关闭异步加载，这样确保在文档完全加载之前解析器不会继续脚本的执行。  xmlDoc.async=false;  //解析器加载名为 "xxx.xml" 的 XML 文档、  xmlDoc.load(filename);  return xmlDoc;  } |

# 附录

在逻辑运算中，0、""、false、null、 undefined、NaN均表示false

null，对象不存在 当前页面不存在id对象

var ii= document.getElementById("id");

alert(ii); //null

undefined 声明的变量没有初始化 　或者对象属性，方法不存在

var i;

alert(i);

if(i==undefined){

alert("dddddddddddddd");

}

对象属性,方法不存在

alert(document.oiji);　　 // undefined