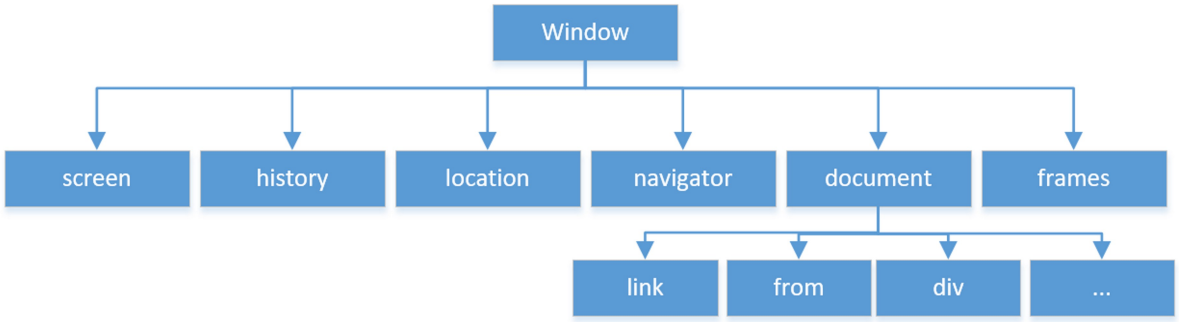
7.BOM和DOM

2021年8月9日　　20:52

1. BOM和DOM模型
   i. BOM模型

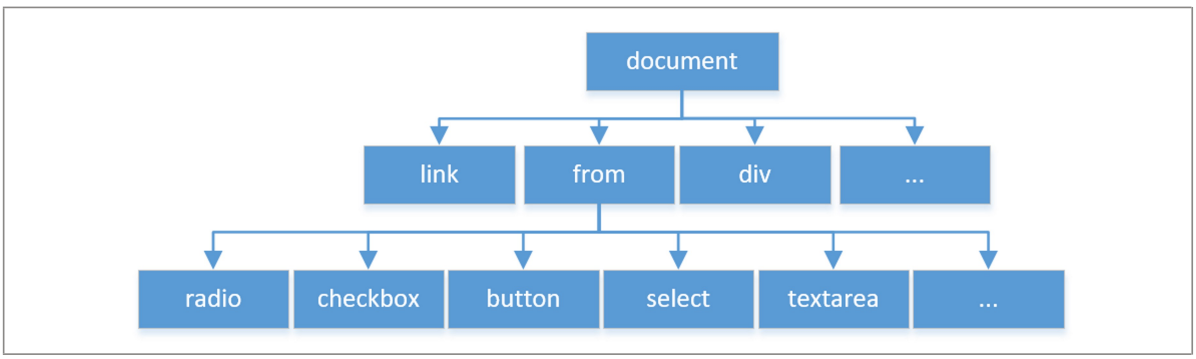
- 浏览器对象模型(Browser Object Model,BOM)定义了JS操作浏览器的接口，提供了与浏览器窗口的交互功能，如获取窗口大小，版本信息，浏览历史记录等。
- 如下图，BOM是用于描述浏览器对象中对象和对象之间层次关系的模型，提供了独立于页面内容并能够与浏览器窗口进行交互的对象结构。
- 其中顶层对象为window，其他对象都是其子对象。当浏览页面时，浏览器会为每个页面自动创建下图中的各个对象



window对象	是BOM模型中的最高一层，通过window对象的属性和方法来实现对浏览器窗口的操作
document	是BOM的核心对象，提供了访问HTML文档对象的属性、方法以及事件处理
location	包含了当前页面的URL地址，如协议，主机名，端口号和路径等信息
navigator	包含与浏览器相关的信息，如浏览器类型和版本等
history	包含历览器历史访问记录，如访问过的URL、访问数量信息等

ii. DOM模型

- 文档对象模型(Document Object Model,DOM)属于BOM的一部分，用于对BOM中核心对象document进行操作，DOM是一种与平台、语言无关的接口，允许程序和脚本动态地访问或更新HTML以及XML文档的内容、结构和样式，且提供了一些列的函数和对象来实现访问、删除、修改和添加等操作
- HTML文档是一种结构化的文档，通过DOM技术不仅可以操作HTML。页面内容，还可以控制HTML页面的风格样式。



iii. 事件机制

- JS采用事件驱动的响应机制，在用户页面上进行交互操作时会触发相应的事件。当事件发生时，系统调用JS中指定的事件处理函数进行处理。事件产生于响应都是由浏览器来完成，HTML中设置那些元素响应哪些事件，JS告诉浏览器如何处理这些响应事件。
- JS事件分为两大类：
  - 操作事件：用户在浏览器中操作所产生的事件。
    - 鼠标事件(Mouse Events)、
    - 键盘事件(Keyboard Events)、
    - 表单及表单元素事件(Form & Element Events)：表单的提交、重置和表单元素的改变、选取、获取/失去焦点
  - 文档（页面）事件：文档本省所产生的事件，如页面加载完毕，页面卸载，关闭页面窗口，
- 对HTML元素的绑定事件的方式包括HTML元素的属性绑定和JS脚本的动态绑定
  - 属性绑定：在HTML标签内使用以on开头的某一事件处理函数进行绑定
  - JS脚本绑定：获取文档中的某一对象，然后通过on口头的事件绑定指定的事件处理函数

2. Window对象

i. 常用属性

name	窗口的名字
length	窗口内的框架个数
innerHeight,innerWidth	浏览器窗口内部高度和宽度
self,top	当前窗口，当前框架最顶层窗口
status	状态栏信息
closed	判断窗口是否被关闭
document,frames,history,location	4个下级对象
scrollbars,toolbar,menubar,locationbar	滚动条、工具栏、菜单栏、地址栏

ii. 常用方法

window.open()	打开新窗口
window.close()	关闭当前窗口
window.moveTo()/ window.moveBy()	移动当前窗口
window.resizeTo()/window.resizeBy()	调整当前窗口的尺寸
focus() / blur()	得到焦点 / 失去焦点
alert (message)	在对话框中显示 message 消息
confirm(message)	在有OK和Cancle按钮的对话框中显示 message
prompt(message,response)	在带有文本输入框的窗口中显示 message
setTimeout(expression,time)	在time之后计算expression，time的单位是毫秒
clearTimeout(name)	用名字取消实现暂停
setInterval(expression,time)	每隔time之后计算expression

clearInterval(name)	暂停

iii. Open使用示例

```
window.open ('page.html','newwindow','height=100,width=400,top=0,left=0,
toolbar=no,menubar=no,scrollbars=no, resizable=no,location=no, status=no')
```

page.html将在新窗体newwindow中打开，宽为100，高为400，距屏顶0象素，屏左0象素，无工具条，无菜单条，无滚动条，不可调整大小，无地址栏，无状态栏。请对照。

```
var x = 0;
var y = 0;
var timer;
var myWindow;
/* 打开窗体 */
function winOpen() {
    x = y = 0;
    myWindow = window.open('', 'self', 'width=200,height=100');
    myWindow.document.write('this is my create window');
}
/* 移动窗口 - 间隔1000ms */
function winStartMove() {
    x += 10;
    y += 5;
    myWindow.moveBy(x, y);
    timer = setTimeout('winStartMove()', 1000);
}
/* 停止移动窗体 */
function winStopMove() {
    clearTimeout(timer);
}
/* 关闭窗体 */
function winClose() {
    myWindow.close();
}
```

- window对象的属性：属性示例
- window对象的方法：方法示例

3. **location 和 history 和 navigator对象**：示例演示

```
/* location对象 */
var url = new URL("http://127.0.0.1:5501/01
_Html_Web/ShopFashion/knowledge/Chapter08/8-02windowMethod.html?name=guoqy#myAnchor")
/* var location=Location;
location.href=url; */
document.write("URL:" + url + "对象属性为：<br/>");
document.write("url.protocol: " + url.protocol + "<br/>");
document.write("url.host: " + url.host + "<br/>");
document.write("url.hostname: " + url.hostname + "<br/>");
document.write("url.port: " + url.port + "<br/>");
document.write("url.hash: " + url.hash + "<br/>");
document.write("url.search: " + url.search + "<br/>");
document.write("url.pathname: " + url.pathname + "<br/>");
document.write('<hr />')
/* history对象 */
function goNext() {
```

```
    history.go(1);
}
function goPrevious() {
    history.go(-1);
}
function historyForward() {
    history.forward();
}
function historyBack() {
    history.back();
}
/* navigator */
document.write('浏览器: ' + navigator.appName + "<br>");
document.write('浏览器版本: ' + navigator.appVersion + "<br>");
document.write('浏览器代码: ' + navigator.appCodeName + "<br>");
document.write('浏览器平台: ' + navigator.platform + "<br>");
document.write('浏览器Cookies启用: ' + navigator.cookieEnabled + "<br>");
document.write('浏览器的用户代理报头: ' + navigator.userAgent + "<br>");
function browserType() {
    var isExists;
    var typeInfo = ['MSIE', 'Trident', 'Firefox', 'Chrome', 'safari',
'netscape'];
    var explorer = navigator.userAgent;
    for (let i = 0; i < typeInfo.length; i++) {
        if (explorer.indexOf(typeInfo[i]) >= 0) {
            isExists = typeInfo[i];
            break;
        }
    }
    alert('您的浏览器类型为: ' + isExists);
}
```

4. document对象
   i. document对象的属性

   document对象也是window对象的子对象，是指在浏览器窗口中显示的内容部分，可以通过window.docuement访问。当页面包含框架时，可以通过document.frames[n].document来访问框架中的对象，其中n是当前窗口再框架集中的索引号.document常用的属性有body、title、cookie、URL属性以及all[ ]、forms[ ]、images[ ]等集合属性

属性	描述
body	提供对body元素的直接访问。对于定义了框架集的文档， 该属性引用最外层的frameset元素
cookie	设置和查询当前文档有关的所有cookie
referrer	放回载入当前文档的URL(即上一个页面的URL)
URL	返回当前文档URL
lastModified	返回文档最后被修改的日期和时间
domain	返回下载当前文档的服务器域名
all[]	返回文档中所有HTML元素，all[]已经被document对象的getElementById()等方法代替
forms[]	返回文档中所有form对象集合
images[]	返回文档中所有image对象集合，但不包含<object>标签内定义的图形

ii. 特殊属性说明

- referrer属性：返回加载指定文档的URL地址
- cookie属性：

  是浏览器客户端保存的用户访问服务器时的会话信息，该信息允许服务器端访问。

  cookie的本质是一个字符串。document.cookie=cookieStr；其中使用cookieStr保存cookie值时

  需要注意以下几项：

  - cookie大小在4KB以内
  - cookie:通过键值对构成，需要根据cookieName来检索cookie中的信息，包括expires、path、domain

expires	cookie的过期时间，UTC格式，可以通过Date.toGMTSting()方法生成。 cookie过期就会被删除。默认情况关闭浏览器，cookie即失效
domain	表示域，可以使浏览器确定哪些cookie能够被提交
path	允许访问cookie的路径，只有在此路径下的页面才能读写该cookie， 一般情况下path='/'。同一站点所有页面下均可访问
cookie的编码	当cookie中含有空格、分号、逗号等特殊符号需要escape()进行编码。 读取时再通过unescape()解码

iii. 使用Cookie设置expries时间时会出现一些问题。下面提供一调试完成的案例

```
/* 记录cookie */
function saveCookie(cookieName) {
    var userName = document.forms[0].userName.value;
    var userPwd = document.forms[0].userPwd.value;
    var saveTime = +document.forms[0].saveTime.value;
    var expireDate = new Date();
    if (saveTime != "-1") {
        var day = expireDate.getDate() - 0;
        expireDate.setDate(saveTime + day);
        // 此处使用getDay会返回星期数，故需要使用getDate()获取第几天
    }
    document.cookie = cookieSplit + cookieName + '=' +
        escape(userName) + ',' + escape(userPwd) +
        ",expires=" + expireDate.toDateString() + " " +
        expireDate.toLocaleTimeString() + ";\r\n";
    loadCookie(cookieName);
}


/* 读取cookie */
function loadCookie(cookieName) {
    var currentCookie = document.cookie;
    var beginPart = cookieSplit + cookieName + '=';
    var startPosition = currentCookie.indexOf(beginPart);
    var cookieData = '';
    if (startPosition == -1) {
        document.forms[0].userName.value = "";
        document.forms[0].userPwd.value = "";
    } else {
        var endposition = currentCookie.indexOf(';', startPosition);
        if (endposition == -1) {
            endposition = currentCookie.length;
        }
```

```
        cookieData = currentCookie.
              substring(startPosition + (beginPart).length, endposition);
        var datas = cookieData.split(',');
        document.forms[0].userName.value = unescape(datas[0]);
        document.forms[0].userPwd.value = unescape(datas[1]);
    }
    document.getElementsByTagName('p')[0].innerHTML = document.cookie ==
     "" ? "<font color='red'>暂无cookie信息</font>" : document.cookie;
}
```

iv. document对象的方法分为：

对文档流的操作：	write()、writeln()、open()、close()方法
对文档元素的操作：	getElementById()方法 getElementByName()方法 getElementByTagName()方法 getElementByClassName()方法 querySelector()方法 querySelectorAll()方法

方法	描述
open()	打开一个新文档，并擦除当前文档的内容
write()	向闻到那中写入HTML或JS代码
writeln()	在使用<pre>标签是比较有用. 添加的"\n"换行符再HTML中没有有效
close()	关闭一个由open()打开的输出流，并显示选定的数据
getElementById() getElementByName() getElementByTagName() getElementByClassName()	返回一个拥有指定ID的首个对象 返回带有指定名称、指定标签名的对象集合 返回带有指定class属性的对象集合,该方法属于H5 DOM
querySelector()	返回满足条件的单个(首个元素)
querySelectorAll()	返回满足条件的元素集合

```
var phtml = document.getElementsByTagName('p')[0];
(function () {
    var menuDivGet = document.getElementById('menuDiv');
    var baseSpanGet = menuDivGet.getElementsByClassName('baseClass');
    var buySpanGet = menuDivGet.getElementsByClassName('buyClass');
    var menuDivQuery = document.querySelector('#menuDiv');
    var baseSpanQuery = document.querySelectorAll('.baseClass');
     //只返回单个元素,没有length属性
    var filmSpanQuery = document.querySelector('.filmClass');
    // var menuDivjQuery = $('#menuDiv').get(0);
    // var baseSpanjQuery = $('span.baseClass');
    // var buySpanjQuery = $('span.buyClass');
    phtml.innerHTML += "DOM Get:<br/>具有baseClass特征span的标签个数: " +
          baseSpanGet.length + "<br/>具有buyClass特征的span标签个数: " +
          buySpanGet.length + "<br>";
    phtml.innerHTML += "DOM Query:<br/>具有baseClass特征span的标签个数: " +
          baseSpanQuery.length + "<br/>具有buyClass特征的span标签为: " +
          filmSpanQuery.innerHTML + " 元素类型为" + filmSpanQuery.nodeType+"<br>";
    // document.write("DOM jQuery:具有baseClass特征span的标签个数: " +
```

```
                baseSpanjQuery.length + "具有buyClass特征的span标签个数: " +
                buySpanjQuery.length);
})();
function countElement() {
    var userName = document.getElementById('userName');
    var hobby = document.getElementsByName('hobby');
    var inputs = document.getElementsByTagName('input');
    var result = "ID 为userName的元素的值: " + userName.value + '<br>个人爱好';
    for (let i = 0; i < hobby.length; i++) {
        if (hobby[i].checked) {
            result += hobby[i].value + "    ";
        }
    }
    phtml.innerHTML += "<br> name 为hobby的元素个数: " +
          hobby.length + "<br> 标签为inputs的元素个数为: " + inputs.length;
    // document.write(result);
}
```

5. Form对象：应用示例

   i. 语法：访问form对象的属性和方法

document.表单名称.属性	document.formName.length	表单中控件的个数
document.表单名称.方法(参数)	document.formName.submit()	表单提交方法
document.forms[索引].属性	document.formName2.length	第二个表单中控件的个数
document.forms[索引].方法(参数)	document.formName2.submit()	第二个表单的提交方法

   ii. Form对象的常用属性和方法；表单即表单元素事件：应用示例

类型	事件	描述
表单元素事件	onblur	在对象失去焦点时触发事件
	onfocus	在对象获取焦点时触发事件
	onchange	在域的文本内容改变时触发事件
	onselect	在文本框的文本被选中时触发事件
表单事件	onreset	在表单中的重置按钮被单击时触发事件
	onsubmit	在表单中的确认按钮被单击时触发事件

   iii. 理解表单属性和表单子元素属性，不能停留于表面，

   例如select的selectedIndex和options的selected

```
<!-- 布局框架 -->
<div class="panel">
    <h4>练习5-交换多选框内容</h4>
    <div>
        <select name="multiSelect" id="multiSelectLeft" multiple="multiple">
        </select>
        <div class="changeTools">
            <input type="button" id="btnTurnRight" value=">>">
            <input type="button" id="btnTurnLeft" value="<<">
        </div>
        <select name="multiSelect" id="multiSelectRight" multiple="multiple">
```

```
            <option value="0">选项1</option>
            <option value="1">选项2</option>
            <option value="2">选项3</option>
            <option value="3">选项4</option>
        </select>
    </div>
</div>

/* 练习5-交换多选框内容 */
btnTurnRight.onclick = function TurnRight(params) {
    var srcSelect = document.getElementById("multiSelectLeft");
    var desSelect = document.getElementById("multiSelectRight")
    TurnExchange(srcSelect, desSelect);
}
btnTurnLeft.onclick = function TurnLeft(params) {
    var srcSelect = document.getElementById("multiSelectRight");
    var desSelect = document.getElementById("multiSelectLeft");
    TurnExchange(srcSelect, desSelect);
}
/* 交换连个多选下拉列表的选项 */
function TurnExchange(srcSelect, desSelect) {
    console.log(srcSelect.selectedIndex);
    if (srcSelect.selectedIndex == -1) {
        alert('请选择移动项');
    }
    for (let i = 0; i < srcSelect.options.length; i++) {
        if (srcSelect.options[i].selected) {
            var exist = false;
            for (let j = 0; j < desSelect.options.length; j++) {
                if (desSelect.options[j].value == srcSelect.options[i].value){
                    exist = true;
                    break;
                }
            }
            if (!exist) {
                var srcOptions = document.createElement('option');
                srcOptions.value = srcSelect.options[i].value;
                srcOptions.text = srcSelect.options[i].text;
                desSelect.appendChild(srcOptions);
                // srcSelect.options[i]=null;
                // srcSelect.removeChild(srcOptions);
                // srcSelect.removeChild(srcSelect.options[i]);
            }
        }
    }
    var indexArray = [];   // 下标数组
    for (let i = 0; i < desSelect.options.length; i++) {
        for (let j = 0; j < srcSelect.options.length; j++) {
            if (srcSelect.options[j].innerText ==
                    desSelect.options[i].innerText) {
                srcSelect.options[j] = null;  // 将移动到目标项清空删除
                indexArray.push(j);  // 记录下标查看是否
            }
        }
    }
    console.log(indexArray);
}
```

6. Table对象：[应用示例]

i. JS中提供了Table、TableRow、TableCell对象用于表格、行和单元格进行操作

表格对象-属性

对象	属性	属性值
Table	**rows[]**	返货单元格所有行(TableRow)的一个数组集合， 包括分组thead|tfoot|tbody中的行
	cells[]	返回列表中所有单元格(TableCell)的一个数组集合
	border	设置或返回表格框的宽度(像素位单位)
	caption	设置或返回表单的caption元素
	width	设置或返回表格的宽度
	cellPadding	设置或返回单元边框与单元格内容之间的间距
	cellSpacing	设置或返回在表格中的单元格之间的间距
TableRow	**cells[]**	返回当前行所包含的单元格数组
	sectionRowIndex	返回某一行在tbody、thead、tfoot中的位置
	rowIndex	返回某一行在表格的行集合中的位置
	innerHTML	设置或返回表格行的开始和结束的位置
TableCell	width	返回或设置单元格的宽度
	rowSpan	设置或返回单元格的行数
	colSpan	设置或返回单元格横跨的列数
	cellIndex	返回行的单元格集合中单元格的位置
	innerHTML	设置或返回单元格的开始或结束标签之间的HTML内容

表格对象-方法

对象	方法	描述
Table	createCaption()	从表格删除caption元素
	createTFoot()	从表格删除tfoot元素
	createTHead()	从表格删除thead元素
	insertRow(index)	在表格的指定位置插入一行 新插入行在index之前,参数index不可小于0或超出索引
	deleteRow()	从表格中删除指定的行
	deleteCaption()	从表格删除caption元素
	deleteTHead()	从表格删除thead元素
	daleteTFoot()	从表格删除tfoot元素

TableRow		
	insertCell()	在HTML表中一行的指定位置插入一个空的td
	deleteCell()	删除单行行的单元格

ii. 理解表格属性不能停留于表面，要在使用表格时将 **rows 和 cells** 熟练使用

```
/* doucument 对象节点操作 - 以table 为例 */
window.onload = function () {
    // 方法1
    var btnCalcSale = document.getElementById('btnCalcSale');
    btnCalcSale.onclick = function () {
        var sum = 0;
        var nodeTable = document.getElementById('nodeTable');
        var rows = nodeTable.rows;
        for (let i = 0; i < rows.length; i++) {
            var num = parseInt(rows[i].cells[2].innerHTML);
            if (!isNaN(num)) {
                sum += num;
            }
        }
        console.log(sum);
        nodeTable.rows[nodeTable.rows.length - 1].cells[1].innerHTML = sum + "W";
    }
}
// 方法2
function CalcAmount(params) {
    var sum = 0;
    var nodeTable = document.getElementById('nodeTable');
    // 获取表格中tbody节点中的全部信息，包括元素节点 和 tbody标签节点
    var tbodyList = nodeTable.childNodes;
    console.log('tbody的元素长度并非tr的行数,而是: ' + tbodyList.length);
    for (let i = 0; i < tbodyList.length; i++) {
        var tbody = tbodyList.item(i);
        // 筛选出元素节点，过滤掉文本节点
        if (tbody.nodeType == 1) {
            var rowList = tbody.childNodes;
            console.log('row的元素个数并非td的列数，而是:' + rowList.length);
            for (let j = 0; j < rowList.length; j++) {
                var row = rowList.item(j);
                // 筛选出元素节点，过滤掉文本节点
                if (row.nodeType == 1) {
                    var cellList = row.childNodes;
                    console.log("cell的元素个数并不是只有一个内容，而是: " +
                        cellList.length);
                    var lastCell = parseInt(cellList.item(5).innerHTML);
                    if (!isNaN(lastCell)) {
                        var saleAmout = lastCell;
                        sum += saleAmout;
                    }
                }
            }
        }
    }
    console.log(sum);
    // 最后通过遍历节点法，设置金额
    nodeTable.children[3].children[0].children[1].innerHTML = sum + "(W) ￥";
}
```

7. DOM节点 - **element对象**

i. 在DOM模型中，HTML文档的结构是一种树形结构，HTML中标签和属性可以看作DOM树的节点。节点又分为元素节点、属性节点、文本节点、注释节点、文档节点和文档类型节点，各种节点统称为Node对象，通过Node对象的属性和方法可以遍历整个文档树。

ii. Node对象的nodeType属性用于获取该节点的类型，具体如表

节点类型	nodeType	描述	示例
元素(Element)	1	HTML标签	..
属性(Attribute)	2	HTML标签的属性	Type="text"
文本(Text)	3	文本的内容	Hello HTML !
注释(Comment)	8	HTML注释段	<!--注释-->
文档(Document)	9	HTML文档根节点	<html>
文档类型(DocumentType)	10	文档类型	<!DOCTYPE HTML...>

iii. **Element 对象继承了Node对象，是Node对象的一种，以它为例说明其常用属性和方法如下**：

属性

属性名	描述
attribute	返回指定节点的属性集合
className	**设置或返回元素的class属性（该属性也可获取CSS样式）**
innerHTML	设置或返回元素内部HTML
tagName	返回元素的标签名(始终是大写形式)
childNodes	标准属性，返回直接后代的元素节点和文本节点集合，类型为NodeList
children	非标准属性，返回直接后代的元素节点的集合，类型为Array
firstChild	返回指定节点的首个子节点
lastChild	返回指定节点的最后一个子节点
nextSibling	返回同一父节点的指定节点之后紧跟的节点
previousSibling	返回同一父节点的指定节点之前的一个节点
parentNode	返回指定节点的父节点；当没有父节点时，则返回null
nodeType	返回指定节点的节点类型(数值)
nodeValue	返回或设置指定节点的节点值

方法

方法名	描述
getElementByTagName()	返回具有指定标签名的元素子元素集合, 类型为NdoeList
hasAttribute("属性名称")	指定属性存在时返回true，否则返回false
getAttribute(" 属性名称 ")	返回指定属性对应的属性值
removeAttribute("属性名称")	删除指定的元素
setAttribute("名称","值")	为节点添加属性，当属性存在时，则进行替换
hasChildNodes()	检查元素是否有子节点
removeChild()	删除某个指定的子节点，并返回该节点

	repalceChild()	用新节点替换某个子节点

以上方法在处理 CSS 样式的动态添加、动态删除、判断是否存在中非常常用

```
<h2>JS设置操作CSS样式</h2>
<div class="panel_bg">
    <div class="panel">
        <h4>1. 操作多个P元素</h4>
        <p>文本1文本1文本1文本1文本1文本1文本1文本1文本1文本1</p>
        <p>文本2文本2文本2文本2文本2文本2文本2文本2文本2文本2</p>
        <p class="active test">文本3文本3文本3文本3文本3文本3文本3文本3文本3文本3</p>
        <p>文本4文本4文本4文本4文本4文本4文本4文本4文本4文本4</p>
    </div>
    <div class="panel">
        <h4>2. 操作多个Input元素</h4>
        <div class="colordiv">
            <input type="color" id="colorRed" class="active" value="#FF0000">
            <input type="color" id="colorBlue" value="#467AFC">
            <input type="color" id="colorYellow" value="#FC8C23">
            <input type="color" id="colorGreen" value="#2775B6">
            <input type="color" id="colorBlack" value="#000000">
            <input type="color" id="colorGrey" value="#F0F0F0">
            <input type="color" id="colorWhite" value="#74759B">
        </div>
    </div>
    <div class="panel">
        <h4>3. 操作多个Input radio元素</h4>
        <input type="radio" name="radioGroup" />单选1
        <input type="radio" name="radioGroup" />单选2
        <input type="radio" name="radioGroup" />单选3
        <input type="radio" name="radioGroup" />单选4
        <input type="radio" name="radioGroup" />单选5
        <input type="radio" name="radioGroup" />单选6
    </div>
</div>

<script>
    var plist = document.getElementsByTagName('p');
    for (let i = 0; i < plist.length; i++) {
        const element = plist[i];
        console.log(i + "element.classList获取类样式集合: "+ element.classList);
        console.log(i + "element.className获取类样式名称" + element.className);
        console.log(i + "element.getAttribute('class')" +
        element.getAttribute("class"));
        element.onclick = function () {
            if (this.getAttribute('class') == null) {
                for (let j = 0; j < plist.length; j++) {
                    if (plist[j].className == 'active text') {
                        plist[j].removeAttribute('class');
                    }
                }
                this.setAttribute('class', 'active text');
            }
            else {
                this.removeAttribute('class');
            }
        }
    }
```

```
    var colorInput = document.getElementsByTagName('input');
    for (let i = 0; i < colorInput.length; i++) {
        const element = colorInput[i];
        console.log(i + "element.getAttribute('class')" +
        element.getAttribute("class"));
        // console.log(element.value);
        if (element.getAttribute('type') == 'color') {
            element.onclick = function (event) {
                // this.prevetDefault();
                var target = event.target || event.srcElement;
                if (this.className == 'active') {
                    this.removeAttribute('class');
                } else {
                    for (let j = 0; j < colorInput.length; j++) {
                        if(colorInput[j].className=='active'){
                            colorInput[j].removeAttribute('class');
                        }
                    }
                    this.setAttribute('class', 'active');
                }
            }
        }
    }
    //非闭包
    var Inputs = document.getElementsByTagName('input')
    for (let i = 0; i < Inputs.length; i++) {
        console.log(Inputs[i].getAttribute('type'));
        if (Inputs[i].getAttribute('type') == 'radio') {
            Inputs[i].onclick = function (e) {
                var target = e.target || e.srcElement
                for (let i = 0; i < Inputs.length; ++i) {
                    if (target.type == "radio" && target == Inputs[i]) {
                        alert(++i);
                        break;
                    }
                }
                console.log(target)
            }
        }
    }
<script>
```

iv. **NodeList对象是一个节点集合，其item(index)方法用于从集合中返回指定索引的节点，length是用户返回集合节点的数量**

```
// 统计全部金额
function amountProfit() {
    var sum = 0;
    var myTable = document.getElementsByTagName('table')[0];
    var tbodyList = myTable.childNodes;
    for (let i = 0; i < tbodyList.length; i++) {
        var tbody = tbodyList.item(i);
        if (tbody.nodeType == 1) {
            var rowList = tbody.childNodes;
            for (let index = 0; index < rowList.length; index++) {
                var row = rowList[index];
                if (row.nodeType == 1) {
                    var cellList = row.cells;
                    // 3列的表格返回7个元素节点,td元素的前后是换行和制表位
                    var profitCell = cellList[2];
                    if (profitCell != null) {
                        var profit = parseFloat(profitCell.innerHTML);
                        sum += (isNaN(profit)? 0 : profit);
                    }
```

```
                }
            }
        }
    }
    var lastRow=myTable.children[2].children.length-1;
    myTable.children[2].children[lastRow].children[1].innerHTML = sum;
    // 统计方式2
    /* var tableRows=document.getElementsByTagName('tr');
    var lastRow=tableRows.get(tableRows.length-1);
    lastRow.lastChild.previousSibling.innerHTML=sum; */
}
// 向最后一行添加新数据
function insertRearRow() {
    var mytable=document.getElementsByTagName('table')[0];
    var index=mytable.rows.length-1;
    var row=mytable.insertRow(index);
    var userIDCell=row.insertCell(0);
    var userNameCell=row.insertCell(1);
    var profitCell=row.insertCell(2);

    userIDCell.innerHTML=document.getElementById('userID').value;
    userNameCell.innerHTML=document.getElementById('userName').value;
    profitCell.innerHTML=document.getElementById('profit').value;
    amountProfit();
}
```

8. 事件处理
   i. 事件的理解

   JS采用事件驱动的响应机制，用户在界面进行交互操作时会触发相应的事件。

   事件（Event）驱动:指页面中相应用户操作的一种处理方式，

   事件（Event）处理:在响应用户操作时所调用的代码。

   事件的产生和响应都是浏览器来完成的，包括HTML代码中设置哪些元素响应哪些事件，JS告诉浏览器如何处理这些响应

   当浏览器检查到事件(鼠标事件、键盘事件、文档事件、表单、表单元素事件)时，
   调用事先指定的事件处理函数进行处理，在此过程中，事件中需要传递的信息都是通过事件对象来完成的。

   Event对象是JS中非常重要的对象,其中包括当前触发事件的状态,如键盘、鼠标状态或位置、窗口变化、表单填写和提交等。大多数event对象的属性是只读的,因为Event是事件动作的快照。

   ii. 事件绑定的方式（5种）

- HTML的DOM支持onclick、onchange等以on开头的属性，可以直接在属性中编写javascript代码

```
<p class="white" id="funcbinding1"
    onclick="var id=this.id;alert(id); return false;">事件绑定方式1</p>
```

缺点：代码放在字符串里，不能格式化和排版，代码较多时不方便阅读很难看懂。

- 在DOM种通过onClick属性指定函数名，在JS中声明实现函数

```
<h4>2. 行内事件绑定函数名</h4>
<p class="white" id="funcbinding2" onclick="show(this)">事件绑定方式2</p>
<script>
    function show(params) {
        var id = params.id;
        alert(id);
```

```
    }
</script>
```

- 在JavaScript代码中通过DOM元素的onclick、onchange等属性绑定

```
<h4>3. 将事件动态绑定到DOM</h4>
<div class="group">
    <p class="white" id="funcbinding3" name="funcbinding3">事件绑定方式3-1</p>
    <p class="white" id="funcbinding3" name="funcbinding3">事件绑定方式3-2</p>
    <p class="white" id="funcbinding3" name="funcbinding3">事件绑定方式3-3</p>
    <p class="white" id="funcbinding3" name="funcbinding3">事件绑定方式3-4</p>
</div>
<script>
    var plist = document.getElementsByName('funcbinding3');
    for (let i = 0; i < plist.length; i++) {
        plist[i].onclick = function (event) {
            e = event.target || event.srcElement;
            alert((i++) + ":" + e.id);
        }
    }
</script>
```

- 在IE下使用attachEvent/detachEvent 函数进行事件绑定和取消

  不推荐此方法。虽然attachEvent/detachEvent 函数支持IE6~IE11，但是FireFox和Chrome浏览器均不支持该方法，存在兼容性不好的问题。同时它也不是W3C标准

- 使用W3C标准提供的 addEventListener 和 removeEventListener

  这两个函数作为W3C标准被FireFox和Chrome所支持，虽然早期的IE6/IE7/IE8，并未支持两个API，但从IE9之后均可正常使用

```
<h4>5. addEventListener/removeEventListener 绑定方式</h4>
<!--
    addEventListener(type,listener,useCapture)
        type: 事件类型，此时不含"on"，如click、mousemover、keydown
        listener: 事件处理函数
        useCapture：代表事件冒泡类型，是事件冒泡还是事件捕获：
                    false 代表事件冒泡，true代表事件捕获
    同一个事件处理函数可以绑定2次，但必须是一次捕获一次冒泡。
    如果绑定的事件冒泡类型相同，则只执行一次
    -->
<div class="funcbinding5 group">
    <span class="">A
        <span class="">B
            <span class="">C
                <span class="">D</span>
            </span>
        </span>
    </span>
</div>
<script>
    window.onload = function () {
        var spanlist = document.getElementsByTagName('span');
        for (let i = 0; i < spanlist.length; i++) {
            const element = spanlist[i];
            element.addEventListener('click', function show() {
                alert(i + ":" + this.innerText);
            }, true);
```

```
        }
    }
</script>
```

iii. 事件流和事件对象

- DOM事件流：也称事件传播过程，亦即，在树形的DOM结构中，其中一个HTML元素产生一个事件时，事件会在元素节点与根节点之间按特定顺序传播，路径经过的节点都会收到该事件。
- 事件流顺序的两种类型：
  - 事件冒泡(Event Bubbling):从叶子节点沿祖先节点一直传递到根节点。事件从特点的目标到最不特点的事件目标传播.
  - 事件捕获(Event Capturing)：从顶层元素一直传递到最精确的元素
- Event对象提供：
  - prevetDefault()方法，用以通知浏览器不再执行与事件关联的默认动作，
  - stopPropagation()用于终止事件的进一步传播

iv. Event对象的常用属性：**经常使用 e=event||window.event**

属性	描述
(e.screenX,e.screenY)	**返回事件发生时鼠标指针相对屏幕的水平、垂直坐标**
(e.clientX,e.clientY)	**返回事件发生时鼠标指针相对当前窗口的水平、垂直坐标**
e.button	返回指示 是哪个鼠标按键点击时 触发了当前事件，包括： 0（左键）、1（中键）、2（右键）、4（IE浏览器中的中键）
e.altKey e.shiftKey e.ctrlKey	返回一个布尔值，判断当前alt、ctrl、shift键 是否一直被按住
e.type	返回发生事件的类型，如submit、click、load
e.target	返回触发事件的目标元素

- 对于IE浏览器还额外支持如下的对象属性

属性名	描述
e.cancelBubble	若事件句柄取消事件传播到包含的对象，必须将该书香设置为true
e.keyCode	对于keypress事件：返回被敲击键盘的uncode字符码 对于keyup,keydown事件：指定被敲击键的虚拟键盘码， 但虚拟码可能与键布局相关
e.offsetX,e.offsetY	发生事件的对象，在事件源元素的坐标系统中的X和Y坐标
e.X,e.Y	事件发生位置的X和Y坐标, 他们对应于用CSS动态定位的最外层包容元素
e.returnValue	如果设置了该属性，它的值比事件句柄的返回值优先级高, 将该属性设置为false,可以取消发生事件源元素的默认动作
e.srcElement	对生成事件的window对象、document对象或element对象的引用
e.toElement	对于mouseover和mouseout事件，该属性代之移入鼠标的元素

```
function enlargre(e) {
    /* 获取图片 */
    var imge=document.getElementsByTagName('img')[0];
    var e = e || window.event;
    var x = e.clientX - 30;
```

```
    /* imge.offsetTop表示距离屏幕右上角的高度偏移，imge.offsetLeft表示左侧偏移 */
    var y = e.clientY - imge.offsetTop - 30;
    context.drawImage(img, x, y, 60, 60, 0, 0, canvas.width, canvas.height);
}
```

```
/* 显示鼠标移动时的坐标信息 */
this.onmousemove = function (event) {
    var mouseX = event.screenX;
    var mouseY = event.screenY;
    showMouseInfo.value = mouseX + ',' + mouseY;
};
```

```
/* 全选商品 */
function checkAll(event) {
    if (event.srcElement.checked) {
        //如果是顶部操作栏是选中动作，则进行全选操作
        var inputList = document.getElementsByTagName('input');
        for (let i = 0; i < inputList.length; i++) {
            if (inputList[i].type == "checkbox") {
                inputList[i].checked = true;
            }
        }
    } else {
        //如果是顶部操作栏是取消选中动作，则进行全不选操作
        var inputList = document.getElementsByTagName('input');
        for (let i = 0; i < inputList.length; i++) {
            if (inputList[i].type == "checkbox") {
                inputList[i].checked = false;
            }
        }
    }
    //最后将页面的数据收集后计算
    calSumMoney();
}
/* 全选店铺 */
function checkBrand(event) {
    if (event.srcElement.checked) {
        var checboxlist = event.srcElement.form.goods;
        //如果当前店铺的操作栏多选框被选中，则将该店铺的商品全选
        for (let i = 0; i < checboxlist.length; i++) {
            checboxlist[i].checked = true;
        }
    } else {
        //如果当前店铺的多选框被取消选中，则将该店铺的商品全不选
        var checboxlist = event.srcElement.form.goods;
        for (let i = 0; i < checboxlist.length; i++) {
            checboxlist[i].checked = false;
        }
    }
    // 检查以下全选复选框的状态是否可以变成全选
    selectAllState();
    //最后将页面的数据收集后计算
    calSumMoney();
}
/* 选中当前 */
function checkthis(event) {
```

```
    /* 选中当前本来可以不进行额外操作，但是因为商铺全选的存在，
    在选中当前后要根据店铺选中的情况改变，当前商铺的全选的状态 */
    if (event.srcElement.checked) {
        var j = 0;
        var checkboxList = event.srcElement.form.goods;
        for (let i = 0; i < checkboxList.length; i++) {
            if (checkboxList[i].checked) {
                j++;
            } else {
                break;
            }
        }
        if (j == checkboxList.length) {
            event.srcElement.form.checkshop.checked = true;
        }
    } else {
        event.srcElement.form.checkshop.checked = false;
    }
    // 检查以下全选复选框的状态是否可以变成全选
    selectAllState();
    //最后将页面的数据收集后计算
    calSumMoney();
}
```

v. 鼠标事件：应用示例

单机	onclick	双击	ondblclick	移出	onmousedown
按下	onmouseover	松开	onmouseout	移动	onmousemove
悬停	onmouseup				

```
<h2>8-12 鼠标事件</h2>
<div class="dl_bg">
    <dl>
        <dt>
        <img src="img/card/01.jpg" alt=""
             onclick="this.src='img/card/10.jpg'">
        </dt>
        <dd>单击事件</dd>
    </dl>
    <dl>
        <dt><img src="img/card/01.jpg" alt=""
            onclick="this.src='img/card/11.jpg';
            alert('双击了图片')"></dt>
        <dd>双击事件</dd>
    </dl>
    <dl>
        <dt><img src="img/card/01.jpg" alt=""
                onmouseover="this.src='img/card/12.jpg'"
                onmouseout="this.src='img/card/01.jpg'"></dt>
        <dd>移动悬停</dd>
    </dl>
    <dl>
        <dt><img src="img/card/01.jpg" alt=""
                onmousedown="this.src='img/card/13.jpg'"
                onmouseup="this.src='img/card/01.jpg'"></dt>
        <dd>按下松下</dd>
    </dl>
    <dl>
```

```
        <dt><img src="img/card/01.jpg" alt=""
             onmouseover="changeImage(event)"></dt>
        <dd>鼠标移动四角</dd>
    </dl>
</div>
<script type="text/javascript">
    function changeImage(e) {
        e = e || window.event;
        var myImageList = document.getElementsByTagName('img');
        var myImage = myImageList[myImageList.length - 1];
        var x = e.clientX - myImage.offsetLeft;
        var y = e.clientY - myImage.offsetTop;
        if (x < myImage.width / 2 && y < myImage.height / 2) {
            myImage.src = 'img/card/11.jpg';
        } else if (x < myImage.width / 2 && y > myImage.height / 2) {
            myImage.src = 'img/card/12.jpg';
        } else if (x > myImage.width / 2 && y < myImage.height / 2) {
            myImage.src = 'img/card/13.jpg';
        } else if (x > myImage.width / 2 && y > myImage.height / 2) {
            myImage.src = 'img/card/14.jpg';
        }
    }
</script>
```

vi. 键盘事件：应用示例

js键盘按钮keyCode及示例大全 - 我的过去 - 博客园 (cnblogs.com)

window.onkeypress	在键盘按键被按下并释放一个键时触发事件
window.onkeydown	在用户按下一个按键时触发
window.onkeyup	在键盘按钮送开始触发

```
/* DOM 键盘事件示例：  */
var rightImages = ["r0.png", "r1.png", "r2.png"];
var leftImages = ["l0.png", "l1.png", "l2.png"];
var downImages = ["d0.png", "d1.png", "d2.png"];
var upImages = ["u0.png", "u1.png", "u2.png"];
var images = rightImages;
var n = 0;
/* 根据键盘信息操作dom对象 */
function doKeyDown(e) {
    var walkDiv = document.getElementById('walkingDiv')
    var walkPerson = document.getElementById('walkPerson');
    var imgWidth = walkDiv.offsetWidth;
    var imgHeight = walkDiv.offsetWidth;
    var x = dealPx(walkDiv.style.left);
    var y = dealPx(walkDiv.style.top);
    var e = e || window.e;
    var keyID = e.keyCode ? e.keyCode : e.which;
    switch (keyID) {
        case 39:
        case 68:
            if (x + imgWidth < window.innerWidth) {
                x += 10;
            }
            images = rightImages;
            break;
        case 37:
        case 65:
            if (x - 10 >= 0) {
```

```
                x -= 10;
            }
            images = leftImages;
            break;
        case 38:
        case 87:
            if (y - 10 >= 0) {
                y -= 10;
            }
            images = upImages;
            break;
        case 40:
        case 83:
            if (y + imgHeight < window.innerHeight) {
                y += 10;
            }
            images = downImages;
            break;
        default: break;
    }
    if (n >= rightImages.length) {
        n = 0;
    }
    walkPerson.src = './img/walker/' + images[n];
    walkDiv.style.left = x + 'px';
    walkDiv.style.top = y + 'px';
    n++;
}
/* px单位设置 */
function dealPx(pixeStr) {
    var pixel = pixeStr.substring(0, pixeStr.indexOf('px'));
    if (pixel == "") {
        return 0;
    } else {
        return parseInt(pixel);
    }
}
```

键盘按键对应的键值

数字键盘上的键的键码值(keyCode)				功能键键码值(keyCode)			
按键	键码	按键	键码	按键	键码	按键	键码
0	96	8	104	F1	112	F7	118
1	97	9	105	F2	113	F8	119
2	98	*	106	F3	114	F9	120
3	99	+	107	F4	115	F10	121
4	100	Enter	108	F5	116	F11	122
5	101	-	109	F6	117	F12	123
6	102	.	110				
7	103	/	111				

控制键键码值(keyCode)							
按键	键码	按键	键码	按键	键码	按键	键码
BackSpace	8	Esc	27	Right Arrow	39	-_	189
Tab	9	Spacebar	32	Down Arrow	40	.>	190
Clear	12	Page Up	33	Insert	45	/?	191
Enter	13	Page Down	34	Delete	46	`~	192
Shift	16	End	35	Num Lock	144	[{	219
Control	17	Home	36	;:	186	/|	220
Alt	18	Left Arrow	37	=+	187	]}	221
Cape Lock	20	Up A					

vii. 文档事件：[应用示例]

window.onload	在页面或图像加载完成后立即触发事件
window.onunload	在用户推出页面时触发事件
window.onresize	在窗口和框架被调整大小时触发事件