DOM（文档对象模型）操作网页的

文档：表示整个 HTML 网页文档
对象：表示将杨业中国每个部分都转换成一个对象
模型：使用模型来表示对象之间的关系，方便获取对象
节点：构成 HTML 的最基本的单元
文档节点：整个 HTMl 文档
元素节点：HTML 文档中 HTML 标签
属性节点：元素的属性
文本节点：HTML 标签中的文本内容

	nodeName	nodeType	nodeValue
文档节点	#document	9	null
元素节点	标签名	1	null
属性节点	属性名	2	属性值
文本节点	#text	3	★文本内容

文档加载：一般都在 windown.onLoad = function (){}中写，就可以在页面加载完了再有用，不写在按键里面，会耦合，不方面使用

1.获取对象
获得 ID：var name = document.getElementById（name）;
获得一组 classname：var name = document.getElementsByClassName(name);
不支持 IE8
获得一组标签：var name = document.getElementsByTagName(name);
获得一组名字：var genders = document.getElementsByName("gender");
class 和标签获得的都是类数组，可以拿来当数组用，
例如：

```
var lis = document.getElementsByTagName("li");获得 li 数组
for(var i = 0;i <lis.length ;i++){
        alert(lis[i].innerHTML);
```

2.调用对象的时间，在事件的函数中编写按钮的行为，查文档
设置按钮单击时：**按钮名字.onclick = function () {}响应函数**
设置按钮双击时：**按钮名字.ondlbclick = function () {}**

3 编写内容
对象名字.innerHTML = “要修改的按键上的文字内容”，要双标签才有这个
自结束标签是没有 innerHTML 的，要读属性就元素.属性名，
自结束标签获取名字用 document.getElementsByName
其中 class 名不可以读，要读 className
要读内容：直接使用元素.属性名
innerText 会去除元素，只有文本

DOM 的查询：

操作标签：

```
window.onload = function () { //onload 没有括号
    //查找#bj 节点，打印标签内的内容（innerHTML）
    var btn01 = document.getElementById("btn01");//是获得按键，注意引号
    btn01.onclick = function () { //  给按键添加事件
        var bj = document.getElementById("bj");//获得要查询的名单
        alert(bj.innerHTML);//获得事件的内容，显示出来

    };
}
```

获取元素节点的子节点：

通过具体元素节点调用：

getElementByTagName()

方法，返回当前节点的指定标签后代节点

childNodes：

属性,表示当前节点的所有子节点，包括元素空白文本，注释所有的节点，包括换行，IE8 以及以下只会返回元素

用 child 可以获得我们的元素

firstChild：

属性,表示当前节点的第一个子节点，包括空白

firstElementChild 可以过得元素第一个，但是不兼容 IE8 以下

lastChild：

属性，表示当前节点的最后一个子节点，同上

获取父节点和兄弟节点：

通过具体元素节点调用：

parentNodes：

属性,表示当前节点的父节点

previousSibling：

属性,表示当前节点前一个兄弟节点，也会获取空白节点

nextElementSibling：获取兄弟元素，不兼容 IE8

属性,表示当前节点后一个兄弟节点

文本：

文本框中 value 的值就是文本框中输入的值

修改就换文本框的值

checkbox：

通过多选框 checkbox 的属性，可以来获取或者设置多选框的选中状态

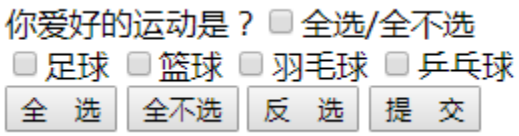
反选：遍历之后，把选中的全部取反就好。谨记是个布尔值

```
items[i].checked = !items[i].checked;
```

提交：遍历 check 是否为 true，然后返回值

```
    for(var i = 0;i < items.length;i++){
            if(items[i].checked == true){//记住是数组.checked == true
                alert(items[i].value); //  记住是输出数组的值.value
}
}
```

在时间响应函数中，this 是谁给的就指向谁




```
window.onload = function () {
var items = document.getElementsByName("items");
var checkedAllBox = document.getElementById("checkedAllBox")

 //1. 全选按钮
 //- 点击按钮以后，四个多选框全都被选中
var checkedAllBtn = document.getElementById("checkedAllBtn");
checkedAllBtn.onclick = function () {
            for(var i = 0;i < items.length;i++){
            items[i].checked = true; }
                                        //将全选/全部选写成 true
             checkedAllBox.checked = true;
};
// 2. 全不选按钮
//- 点击按钮以后，四个多选框都变成没选中的状态
var checkedNoBtn = document.getElementById("checkedNoBtn")
         checkedNoBtn.onclick = function () {
      for(var i = 0;i < items.length;i++){
             items[i].checked = false;
      }
                                        //将全选/全部选写成 false
             checkedAllBox.checked = false; };
 /*3. 反选按钮
 - 点击按钮以后，选中的变成没选中，没选中的变成选中*/
var checkedRevBtn =document.getElementById("checkedRevBtn")
 checkedRevBtn.onclick = function () {
             checkedAllBox.checked = true;//进来就选中状态
             for(var i = 0;i < items.length;i++){
                items[i].checked = !items[i].checked;
                if (!items[i].checked){//一旦进入判断，则证明不是全选
                        checkedAllBox.checked = false;//
                   }
             }
                        //在反选时，也需要判断四个多选框是不是选中
};
 /*4. 提交按钮
- 点击按钮以后，将所有选中的多选框的 value 属性值弹出*/
        var sendBtn = document.getElementById("sendBtn");
        sendBtn.onclick = function () {
             for(var i = 0;i < items.length;i++){
                 if(items[i].checked == true){
                        alert(items[i].value);
     }
             }};
```

```
/*5. 全选/全不选  多选框
  当它选中时，其余的也选中，当它取消时其余的也取消*/
var checkedAllBox = document.getElementById("checkedAllBox");
  checkedAllBox.onclick = function () {
        for(var i = 0;i < items.length;i++){
           items[i].checked = this.checked
        ;//四个的属性，等于这整个的属性，用 this 可以指代调用者
              }
};
/*6. 自动检查(难)如果四个多选框全都选中，则 checkedAllBox 也应该选中如果四个多选框没都选中，则 checkedAllBox 也不应该选中*/
                                        //为四个多选框都绑定响应函数
for(var i = 0;i < items.length;i++){
        items[i].onclick = function () {
                    checkedAllBox.checked = true;//进来就选中状态
                     for(var j = 0;j < items.length;j++){
                                                  //判断四个时候全选
                                          //只要有体格没选中就不是全选
                            if (!items[j].checked){
                                    //一旦进入判断，则证明不是全选
                                checkedAllBox.checked = false;
                                break;    //只要进入判断就直接跳出

              }}}}}
```

主要思路：

- 先把前面四个按键都完成
- 然后设置全选/全不选的 checkbox 按钮添加 onclick 的响应函数，函数内容主要有让这个按键选中时。其他都选中，取消全取消
- 遍历 items[].checked 完全与这个按钮的 checked 都相同
- 设置每个 checkbox 的响应函数，初始化为选中，当他们其中有一个没选中，就让这个按键也全部选中
- 再检查一次各个按键，全选或者全部选就加一条代码，反选也要判断
- 一下当全部不选的时候，点击反选，就会全部都选

其他的查询方法：

document.body:保存的是 body 的引用

document.documentElement：保存的是 html 根标签

document. all：代表的是页面的所有元素标签

根据元素的 class 属性查询一组元素节点

根据选择器来获取页面元素

document.querySelector(“选择器的元素”)

使用该方法只返回一个值，有很多只会返回第一个

document.querySelectorAll(“选择器”)

与上面一样，但是会返回全部符合的，是一个伪数组

文档的增删改：

新建：

首先，新建一个文本内容，再设置一个元素，在找到要放进去的元素，在一个包一个就可以了新建了

- **document.createElement('li')可以创造一个元素节点**
  括号里面是要创造的元素名字，返回的是创建好的对象
  var li = document.createElement('li')引号引号
- **document.createTextNode('文本内容')**
  var gzText = document.createTextNode('广州')
- **.appendChild 然后将子节点放到父节点中，就是将文本放标签中。**
  父节点（标签）.appendChild(文本，子节点)
  li.appendChild(gzText);
- insertBefore:在父节点 city 中，两个子节点的排序，将新的 li 放在 bj 前面

  父节点. insertBefore(新的,新的后面那个);

- **在前面插入：**
  父节点.insertBefore(新，前面)
  city.insertBefore(li,bj);
- **替换插入：**
  父节点.insertBefore(新，旧)
  city.replaceChild(li,bj);
- **在后面插入：**
  前面节点.appendChild(要追加的)
  li.appendChild(gzText);
  city.innerHTML += ‘
广州’
- **删除子节点：**
  父节点.removeChild(要删除的节点)
  city.removeChild(bj);

- **以上的父子节点都可以用要操作的节点. parentNode**
  bj.parentNode. removeChild(bj);
  读取 HTNL 代码，用 innerHTML
  修改 innerHTML = ‘要修改的新内容’也可以增删改

两种方式结合使用:

创建元素：var li = document.creatElement(‘li’);

修改元素里面的文本：li.innerHTML =“li 里面要加的内容”

在后面插入：city.appendChild(li);

Name	Email	Salary	
Tom	tom@tom.com	5000	Delete
Bob	bob@tom.com	10000	Delete
gg	c		Delect

实例的增删改

```
window.onload = function () {
        //这个是删除的函数，方便两个使用
        function del() {//是 as[i]来 onclick
            var tr = this.parentNode.parentNode;
            var  name  =  tr.getElementsByTagName("td")[0].innerHTML;
//this 是循环到这个的时候的这个 a
//要删除的是整个 tr，a 又在 td 里面，所以要删除的是 a 的父节点的父节点，
//得到 tr 后，再在突然的父节点中删除这个 tr
            if(confirm('真的要删除' + name + '吗?')){
                tr.parentNode.removeChild(tr);}
            return false;
        }
```

```
//循环所有的 a 标签，给当前的 a 标签要是点击的事件，调用函数 del，不是 del();这是返回值。
        var as = document.querySelectorAll('a');
        for(var i = 0;i < as.length;i++){
            as[i].onclick = del;
        }

        var btn = document.getElementById('btn');
        btn.onclick = function () {
            //1.获取三个文本框的值
            var empName = document.getElementById('empName').value;
            var email = document.getElementById('email').value;
            var salary = document.getElementById('salary').value;
            //2.创建一个新的 tr,一定要引号引号引号
            var tr = document.createElement("tr");
            //3.创建四个新的 td 元素
            var nameTd = document.createElement('td');
            var emailTd = document.createElement('td');
            var salaryTd = document.createElement('td');
            var aTd = document.createElement('td');
            //4.创建 a 的元素
            var a = document.createElement('a');
            //将文本框的值放进文本框
            var nameText = document.createTextNode(empName);
```

```
//加文本变量不用加引号
 var emailText = document.createTextNode(email);
var salaryText = document.createTextNode(salary);
var aText = document.createTextNode('Delect');
//加字符串的时候就加引号
//将文本框放进 td 元素中
nameTd.appendChild(nameText);
emailTd.appendChild(emailText);
salaryTd.appendChild(salaryText);
aTd.appendChild(aText);
a.appendChild(aText);
aTd.appendChild(a);
//将四个 td 放进 tr 中
tr.appendChild(nameTd);
tr.appendChild(emailTd);
tr.appendChild(salaryTd);
tr.appendChild(aTd);
//给 a 标签添加一个跳转属性
a.href = "javascript:;";
//是加函数不是加返回值
a.onclick = del;
//获得父子标签，在父子的 table 里面添加一个 tr
varemployeeTable = document.getElementById('employeeTable')
var tbody = employeeTable.getElementsByTagName("tbody")[0];
//将 tr 添加到 tbodye 中，这个默认会有一个 tbody 的，所以要加到 tbody 中才不会在外面
        tbody.appendChild(tr);
    }}
```

删除：

1. 先消除 a 标签会跳转的默认样式，在函数中加一个 return false
2. 循环的获取每行的 a 标签放数组，然后就可以遍历的获得
3. 定义一个 tr 来获取当前的 tr，便于删除，在遍历函数时，用当前的 a 的 this 的父节点来调用当前的 tr
4. 用当前的 tr.父节点.removeChild 来删除 tr。

优化：

添加一个提示框，在提示框中确认是否删除，将以上 remove 的代码放进 if 的判断中，要、true 就执行，false 就不执行。

增加：

1. 先获得当前文本框的值。get。。。（）.value
2. 创造文本节点，里面 append 获取的文本
3. 创建 td 元素节点和 a 标签，将文本节点放进元素里
4. 创建 tr 标签，将 td 标签逐个放进 tr 标签
5. 给 a 添加一个属性，a.href = "javascript:;";让她不跳转
6. 将删除功能变成一个函数，实现删除功能。
7. 获取 tr 标签的父节点，在这里要获取 tbody，放进去 tr

注意：

1. a 默认跳转，form 默认跳转，加一个 return false
2. 添加时注意系统是否都自动放在了 tbody 里面，添加注意父节点
3. 可以将重复的代码写成函数，注意调用直接 del，而不是返回值 del（）；
4. 给 a 添加属性，用于跳转时

写备注的时候，尽量精简，尽量用于描述过程

操作文档中的样式

通过 JS 修改元素的样式
语法：元素.style.样式名 = 样式值
注意：如果 css 的样式名含有—，把-去掉，首字母大写
background-color 改成 backgroudColor 就好
通过 style 是修改内联样式的表格。而内联样式有较高的优先级
获取元素当前的显示样式：
元素.currentStyle.样式名
它可以用来读取当前元素正在显示的样式
如果当前没有设置该样式，便返回默认样式
✋ 只要 IE 支持其他不支持
其他浏览器可以用 getComputedStyle（）的方法来直接使用
需要两个参数
第一个：要获取样式的元素
第二个：可以传递一个伪数组，一般都传 null
该方法会返回对象，对象中封装了当前的元素对应的样式
✋ 不支持 IE8

```
getComputedStyle(asd,null).width
```

可以通过对象来读取样式，没有设置会获得真实的值
都兼容：自己写一个//用 window.它就是一个对象的属性

```
function getStyle(obj,name) {
    if(window.getComputedStyle ){
        return getComputedStyle(obj,null)[name];
    }else{ return obj.currentStyle[name]; }
     alert(getStyle(asd,'width')); }
```

其他样式的相关属性

/*这些属性只能读，不能改，只可通过 style 来改

可见长宽：

clientHeight---clientWidth 这两个元素可以获得我们的可见宽度和可见高度
返回值不带 PX 的，返回都是数字，可以直接计算，包括所有的盒子的内容区和内边距
有滚动条的时候返回的值是，返回值=设置 div 长度-滚动条的长度
offsetHeight----- offsetWidth
是获取元素整个宽度宽度和高度，包括内容区，内边距和边框

偏移量要有父元素：

offsetParent 可以获取当前元素的定位父元素
会获取到离去、当前元素最近的开启了定位的祖先元素
所有的祖先都没开 就返回 body
offsetLeft
当前元素相对于其定位父元素的水平偏移量，就是相差多远
offsetTop
当前元素相对于其定位父元素的垂直偏移量

滚动条：

scrollHeight 里面滚动的高度，就是 div 的高度
scrollWidth 横着的宽度，里面 div 的宽

scrollLeft 滚动条水平距离原点移动的距离

scrollTop 滚动条垂直距离原点移动的距离

当满足 scrollHeight-scrollTop ==clientHeight 说明垂直滚动条到末尾了

定时器：

setInterval();
定时调用，每隔一段时间执行一次
参数：
var timer = setInterval(function(),毫秒数)
var timer 要定义在全局变量中，这样在别的单击事件里面可以直接调用
1.回调函数：该函数会每隔一段时间被调用一次
2.每次调用间隔时间：单位是毫秒
返回值，返回一个 number 类型的数据
这个数字作为定时器的唯一标示
claerInterval(timer)。
可以接受任意参数
如果有效就会停止，要不就不会有反应
每点击一次就会生成一个定时器，但是消除只能消除最后一个定时器
在开始定时器之前要吧上一个定时器关闭
为了防止错乱，在写单击函数的时候，先把定时器关一次，有谁就关谁
第一句加： clearInterval(timer);
延时调用：
开启一个定时器，setTimeout
不马上执行，隔一段时间再执行，只会执行一次
setInterval(function(),毫秒数)
关闭也是一样的
延时调用和定时调用都是可以互相调用的

移动 div：

先获得 div 的 left，变整数之后再加载变成字符串，在修改样式来移动当前。

```
var box = document.querySelector("#box1");
var btn = document.querySelector('#btn01');
var timer = null;//定时器放外面，好调用
   btn.onclick = function () {
   clearTimeout(timer);//每进来一次都要消除之前的，以防乱
    timer = setInterval(function () {
    var left = parseInt(getStyle(box,'left'));//调用这个兼容函数，取得当前属性值，得到的是 111px，要取整数
    left += 200;//每次移动非步数
    box.style.left = left + 'px';//这个修改要接收 111px，所有又把 px 加上去。
        },1000) };
```

这个是获得当前的指定的属性值。

```
function getStyle(obj,name) {
      if (window.getComputedStyle) {
            return getComputedStyle(obj, null)[name];
      } else {
            return obj.currentStyle[name];
      }
}
```

移动 div 到某个地方停止

```
btn.onclick = function () {
        clearTimeout(timer);
        timer = setInterval(function () {
        var left    =    parseInt(getStyle(box,'left'));
              left += 20 ;//向右移动，每次+20
              if(left > 800){
    left = 800; //如果长度大于 800，就让她停在 800，再按就无效
                        }
              box.style.left = left + 'px';
              if(left == 800){//如果长度等于 800 了，就停止行动
                 clearTimeout(timer)
                  } },50) };


    var btn3 = document.querySelector('#btn03');
    btn3.onclick = function () {
     clearTimeout(timer);
     timer = setInterval(function () {
                  var left = parseInt(getStyle(box,'left'));
                  left -= 20 ;
                  if(left < 0){//如果小于 0 就停在 0，再按无效
                 left = 0;    }
          box.style.left = left + 'px';
          if(left == 0){//到了 0 的时候  就停止。
                clearTimeout(timer)
          } },50) };
```

```
function move(obj,attr,speed,target,callback) {
                clearTimeout( obj.timer);
                var current =    parseInt(getStyle(obj,attr));
                if(target < current){
                      speed = -speed; }

                obj.timer = setInterval(function () {
                      var oldValue =    parseInt(getStyle(obj,attr));
                      var newValue = oldValue+speed ;
                      //左移动，判断 left 是否小于 target
                      //右移动，判断 left 是否大于 target
if((speed < 0 && newValue < target) || (speed > 0 && newValue > target)){
                            newValue = target;
                      }
                      obj.style[attr] = newValue + 'px';
                      if(newValue == target){
                            //达到目标，关闭定时器
                            clearInterval( obj.timer);
                            callback();
                      }

                },50);
          };
```

obj：要操作移动的对象
attr：想要改变的属性：”top”, ”width”, ”left”等，要记得引号
speed：要移动的速度
target：在这个方向的到达的目标
callback：回调函数，用于所有动作执行完后的菜执行的动作

- 这个动作执行之前要关闭自己上次执行的定时器
- 获得当前的要改变的属性的位置，getStyle()来调用，转成数字
- 得到之后，与目标位置相比较，如果大于目标函数，就设置速度是反方向
- 设置定时器，在定时器里面编写函数
- 先得到要操作的元素的属性
- 然后将旧的属性+speed=新的属性
- 如果速度小于 0 并且要移动的目标还小于目标值，或者，速度大于 0 并且要移动的目标已经大于目标函数，就把目标函数给新值，让她不能再动了
- 这时就给目标值加上属性
- 当新值已经达到了目标位置，就关闭定时器，让她按键也不能再动了。
- 在这里面再加上一个回调函数 来定义已经执行完之后要执行的函数。

事件的响应函数

事件对象 event：

当事件的响应函数触发时，浏览器每次都会将一个事件对象作为实参传递进响应函数在事件对象中封装了当前事件相关的一切信息，比如；鼠标的坐标，按键,就是获取这个时间的信息

onmousemove 事件当鼠标移动时就触发

在 IE8 中不好使用，响应函数被触发时，浏览器不会传递信息，要加一个 window，但是加 windown 在火狐不兼容

加一个浏览器判断器

两种写法都可以：

```
if(!event){    event = window.event;    }
event = event ||window.event;
```

clientX 可以获取鼠标指针的水平坐标

clientY 可以获取鼠标指针的垂直坐标

clientX client 相对于当前的可见窗口的坐标，就是滚条滚下去了，也是现在的原点，然后就会和 DIV 有一定距离

pageX

pageY

可以获取鼠标相对于当前页面坐标，有滚动条也一样，不会脱离，在 IE8 中不能使用

移动 div 练习

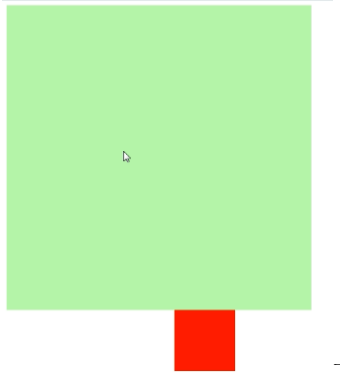
```
var big    = document.querySelector("#big");
    //把它的移动属性给 document，这样在全局都可以动
document.onmousemove = function (event) {
            event = event || window.event;

//解决 scroll.top 和 left 在谷歌和其他浏览器的兼容
var st=document.body.scrollTop||document.documentElement.scrolTop;
var sl = document.body.scrollLeft ||document.documentElement.scrollLeft;

// clientX clientY 是相对于当前的可见窗口的坐标，就是滚条滚下去了，也是现在的原点，然后就会和 DIV 有一定距离
//所以解决这个就把滚动条一定的就距离加上文档移动的距离
            var left = event.clientX;
            var top = event.clientY;
            big.style.left =left + st+sl+'px';
            big.style.top =top +st+'px';//想移动一定要开启绝对定位
      }}
    /
```

事件的冒泡：

指的是事件的向上传导，当后代元素上的事件被触发时，祖先元素的事件也会被触发在开发中，冒泡是有用的，就是爸爸有的儿子也有，大多数是有用的

取消冒泡：可以将事件冒泡的取消

```
元素.onclick = function (event) {
        event =event ||window.event;//有 event 就有它
        event.cancelBubble = true;
}
```



不给红色进入绿色里面

事件的委派：

- 指将事件统一绑定给元素的共同的祖先元素，这样当后代元素上的事件触发时，会一直冒泡到祖先元素

从而通过祖先元素的响应函数来处理事件。

- 事件委派是利用了冒泡，通过委派可以减少事件绑定的次数，提高程序的性能

```
u1.onclick = function(event){
event = event || window.event;

//target：- event 中的 target 表示的触发事件的对象
//alert(event.target);
//如果触发事件的对象是我们期望的元素，则执行否则不执行

if(event.target.className == "link"){
        alert("我是 ul 的单击响应函数");
  }
}
```