1. 什么是DOM:Document Object Model

文档 对象 模型

专门操作网页内容的一套对象和函数的统称

为什么：DOM是唯一能操作网页内容的一套对象和函数

广义js:3部分

(1).ECMAScript(仅仅规定了JS的核心语法，没有规定如何操作网页内容)

(2).DOM.专门操作网页内容的一套对象和函数

(3).BOM.专门操作浏览器窗口的对象和函数

今后只要想操作网页内容，只能用DOM提供的对象和函数

其实今后所有前端js框架：jquery,vue,ng,react底层都是基于DOM实现的

5,DOM标准：

1. .旧的DOM:从有浏览器开始，就有DOM了，但是各个浏览器厂商提供的DOM对象和函数都没有标准，导致各个浏览器在页面的展示不同，被迫使用不同浏览器时，要重新开发不同的对象和函数
2. DOM标准：1998年W3C组织出页面统一的DOM标准，所有浏览器必须遵照执行
3. 之后，使用DOM标准开发网页功能时，几乎所有浏览器100%兼容

但是，IE8没有赶上DOM标准规定，所有IE8对DOM的支持不好

1. DOM可以做：5件事：

增删改查+事件绑定

1. DOM树：
2. 什么是：其实网页中所有HTML内容在内存中都是保存在一棵树形结构上的，内存中这颗保存了所有内容的树形结构，就是DOM树
3. 为什么:因为树形结构是最直观的保存上下级关系的结构，而HTML内容的元素恰好都是上下级包含的关系，所以树形结构保存HTML内容，最一目了然
4. 何时：只有浏览器扫描到一个.html文件时，再将.html内容显示到浏览器窗口给人看，都会在内存中自动创建一颗DOM树来保存html内容的每个细节

4. 如何:

(1). 当浏览器窗口扫描到一个.html文件时，会先在内存中自动创建一个document对象。作为整个DOM树的树根。将来网页中所有的HTML内容，都会成为document树根对象的子节点或后代节点。整个DOM树的树根document对象——也称为根节点对象

(2). 浏览器窗口开始扫描网页内容。每次扫描一项内容，到会在DOm树上对应位置创建一个对象，保存这项扫描到的内容的属性值和其他细节也就是说，网页中的每一项HTML内容(元素或文本)都会成为DOM树上的一个对象——称为节点(node)对象

1. 查找元素：4种情况
2. 不需要查找就可以直接获得的元素：
3. .<html>元素对象 document.documentElement
4. .<head>元素对象 document.head
5. .<body>元素对象 document.body
6. 按节点间关系查找元素：2大类关系，6个属性

(1). 节点树: 旧DOM标准中使用的包含网页中所有内容（元素、文本、注释...）的完整树结构

(2). 包括: 2大类关系，6个属性

a. 父子关系: 4个属性

1). 获得当前节点对象的父级节点对象: 节点对象.parentNode

2). 获得当前节点对象的所有直接子节点对象的集合: 节点对象.childNodes

返回的是一个包含多个直接子节点对象的类数组对象

可用节点对象.childNodes[i]方式，获得某个位置上的子节点

3). 获得当前节点对象的第一个直接子节点对象: 节点对象.firstChild

4). 获得当前节点对象的最后一个直接子节点对象: 节点对象.lastChild

b. 兄弟关系: 2个属性

1). 获得当前节点的前一个兄弟节点对象: 节点对象.previousSibling

前一个 兄弟

2). 获得当前节点的后一个兄弟节点对象: 节点对象.nextSibling

下一个兄弟

(3). 问题: 连看不见的空格换行等空字符，也识别为节点对象，严重干扰了正常的查找秩序。

(4). 解决: 因为程序员普遍只关心元素，不关心其余类型的节点对象，所以新DOM标准中新建了一棵新树——元素树:

(5). 元素树: 只包含元素类型节点对象的树结构，不再包含空格、文本、注释等程序员不关心的节点。

(6). 优点: 不会受到看不见的空字符的干扰。

(7). 何时: 今后在DOM中，只要按节点间关系查找元素对象，都首选元素树。不再使用节点树

(8). 也包括: 2大类关系，6个属性

a. 父子关系: 4个属性

1). 获得当前元素对象的父级元素对象: 元素对象.parentElement

特例: 因为parentElement太长，且parentNode通常不会选中那些非元素节点（因为在HTML中能当爹的只有元素），所以parentNode依然可以代替parentElement继续使用。

2). 获得当前元素对象的所有直接子元素对象的集合: 元素对象.children

返回的是一个包含多个直接子元素对象的类数组对象

可用元素对象.children[i]方式，获得某个位置上的子元素

3). 获得当前元素对象的第一个直接子元素对象: 元素对象.firstElementChild

第一个元素孩子

4). 获得当前元素对象的最后一个直接子元素对象: 元素对象.lastElementChild

最后一个元素孩子

b. 兄弟关系: 2个属性

1). 获得当前元素的前一个兄弟元素对象: 元素对象.previousElementSibling

前一个 元素 兄弟

2). 获得当前元素的后一个兄弟元素对象: 元素对象.nextElementSibling

下一个 元素 兄弟

(9). 何时使用按节点间关系查找元素？

如果已经获得一个元素对象在手，想找这个元素周围附近的元素时，首选按节点间关系查

1. 按照html特征查找元素：可按4种html特征查找
2. .按照id查找一个元素对象
3. var 一个元素对象=document.getElementById(“id名”);
4. 意为：在整个页面中查找，查找id名为id的一个元素
5. 返回值：
6. .如果找到id的一个元素，就返回这一个元素的对象
7. 如果没找到指定id的元素，就返回null
8. 如果页面汇总包含多个相同id名的元素，只返回一个指定id名的对象
9. 强调:
10. .前的主语必须是document
11. 函数名中的Element是单数，没有s
12. .按照元素的标签名查找元素
13. var 类数组对象=任意父元素.getElementsByTagName(“标签名”);
14. 意为：在指定的父元素范围内，查找多个指定比起签名的多个元素
15. 返回值
16. . 如果找到多个符合指定标签名称的元素，则返回一个类数组对象包含多个找到的元素对象、
17. 如果没找到，返回一个length=0的空的类数组对象：{length:0}
18. 强调：
19. .前的主语是任意元素，用来限制查找范围
20. 函数名中的Element是复制数，有s
21. 不仅查找直接子元素，而且在所有后代元素中查找符合标签名的元素
22. .按class名来查找多个元素
23. var 类数组对象=任意父元素.getElementsByClassName(“class名”);
24. 意为：在一个指定的父元素范围内，查找所有指定的class名的多个元素对象
25. 返回值：
26. . 如果找到多个符合要求的元素，也会返回一个类数组对象，包含找到的多个元素对象
27. . 如果没找到，则返回length=0的空类数组对象：{length=0};
28. 强调：

1） .前的主语是任意元素，用来限制查找范围

2）函数名中的Element是复制数，有s

3）不仅查找直接子元素，而且在所有后代元素中查找符合标签名的元素

4）如果一个元素同时被多个class修饰，则只使用其中一个class名也可以找到该元素

(4).按name名查找多个表单元素的时候

a. var 类数组对象=document.getElementsByName(“name名”);

b. 在整个页面中查找指定name名的多个表单元素

c. 返回值：

1）.如果找到指定的name名的多个表单元素，则返回类数组对象，其中包含多个表单元素对象

2).如果找不到，则返回一个空的类数组对象：{length=0};

d. 强调：

1） .前的主语必须是document，用来限制查找范围

2）函数名中的Element是复制数，有s

|  |
| --- |
| 说明：按标签名、class、name查找的永远会类数组对象。即使只知道一个符合要求的元素，也会讲这个元素放入到类数组对象中返回。  所以想要获得该元素，就要通过类数组对象名[0]来获取该元素 |

1. 按css选择器查找：2个函数
2. . 只查找一个符合选择器条件元素
3. var 一个元素对象=任意父元素.querySelector(“任意选择器”);
4. 意为：在一个指定的父元素内，查找第一个符合选择器的元素对象
5. 返回值
6. . 如果找到，返回找到的DOM元素
7. 没找到，返回null
8. 强调：
9. .前可以是任意元素，限制查找范围
10. 即使多个元素符合选择器条件，也只返回第一个找到的元素
11. . 查找多个符合选择器条件的元素
12. var 类数组对象=任意父元素对象.querySelectorAll(“任意选择器”);   
    b. 意为：在一个指定的父元素内，查找多个符合选择器的元素对象
13. 返回值
14. 如果找到，返回找到的所有DOM元素
15. 没找到，返回一个空的类数组对象：{length=0}

|  |
| --- |
| 强调：按照css选择器查找时，括号里面的任意选择器不要超过.前的任意父元素  因为querySelectorAll和qeurySelector默认是在父元素范围内查找。如果超出父元素范围，可能会报错 |

总结：比较按照HTML查找 和 按照css选择器查找

1. 效率：因为按HTML查找条件简单，效率高

按选择器查找，需要解析选择器，在查找，效率低

1. 易用性：按选择器查找，简单

按HTML查找，繁琐

1. 如何选择：

(1). 如果只靠一个条件就能获得想要的元素时，首选按HTML特征查找。

(2). 如果查找条件复杂(多于1个)，首选按选择器查找。

事件基本概念和用法  
1. 什么是事件: 浏览器自动触发的或用户手动触发的页面内容或状态的改变

2. 什么是事件属性: 每个元素身上都有一系列以"on事件名"方式命名的特殊属性。

元素身上所有以on开头的属性，都是事件属性

3. 事件属性的原理: 每当浏览器中某个元素上发生了事件之后，浏览器都会找到这个元素身上对应事件名称的一个事件属性。尝试自动执行该事件属性上提前保存的一个处理函数。

比如: 当我们单击一个按钮时，就在整个按钮上触发了一个名为"单击(click)"的事件，然后浏览器会自动找到当前按钮身上名为"onclick"的事件属性。尝试自动执行onclick事件属性上提前保存好的一个函数。

但是，因为所有元素的所有事件属性值默认都是null，所以，如果不手动为某个事件属性提前保存一个函数，则即使事件发生了，事件属性也找到了，但是什么都不执行。

4. 什么是事件绑定: 提前将一个函数，保存到一个元素的指定的一个事件属性上。

强调: 仅绑定事件，是不会执行事件处理函数的。

结果: 当事件发生在该元素上时，就找到对应的事件属性，才自动执行提前保存在事件属性上的函数。

5. 什么是事件处理函数: 这个提前保存到元素的事件属性上的函数，就称为一个事件处理函数。

6. 如何绑定事件处理函数: 2种:

(1). 在HTML中绑定事件处理函数:

a. <元素 on事件名="js语句">

b. 问题: js语句写在HTML中不符合内容与行为分离的原则，不便于维护。

(2). 今后事件绑定都是在js中集中实现的

a. 元素.on事件名=function(){ ... }

7. 如何在一个元素的事件处理函数中获得当前发生事件的元素本身:

(1). 错误的做法: 使用外部的全局变量当做当前触发事件的元素

(2). 因为: 全局变量很有可能在事件处理函数绑定完之后，被后续程序篡改为其他元素。尤其会发生在使用循环遍历绑定事件处理函数时。循环变量肯定是会变化的！

(3). 正确的做法: 在事件处理函数中，应该用this代表当前正在触发事件的元素

(4). 因为: 当事件发生时，浏览器会找到触发事件的元素，并调用这个元素身上对应的事件属性保存的事件处理函数，浏览器在调用自动执行的语句是:

当前触发事件的元素.onclick()

this

1. 修改：3样东西
2. 内容:3种方式
3. .获取或修改元素开始标签到结束标签之间的原始的HTML内容：元素.innerHTML
4. 获取或修改元素开开始标签到结束标签之间的纯文本内容：元素.textContent

|  |
| --- |
| innerHTML vs textContent  获取元素内容时：  innerHTML直接获取出元素的原始的HTML内容，不做任何加工和删减，直接返回。  textContent多做两件事: 1. 去掉内嵌的标签，2. 特殊符号翻译为正文  修改元素内容时:  innerHTML会将要修改的新内容交给浏览器先解析，再将解析后的效果显示在页面上  textContent不会将要修改的新内容交给浏览器解析，而是努力保持新内用的原样显示，不做任何加工和解析。 |

1. .获取或修改表单元素的值
2. 因为多数表单元素没有结束标记，所以不能用innerHTML和textContent
3. 应该用：表单元素.value

|  |
| --- |
| 强调：凡是从页面上获得内容都是字符串。在进行计算之前，都要先转为数字 |

1. 属性:3中
2. .字符串类型的HTML标准属性
3. 什么是：HYML标准规定的，值为字符串类型的HTML属性
4. 如何获取或修改HTML标准属性：2种方法
5. . 旧DOM标准中规定的4个函数

i. 获取一个元素的指定属性的属性值：

var 属性值=元素.getAttribute("属性名")

获取 属性

ii. 修改一个元素的指定属性的属性值:

元素.setAttribute("属性名","属性值")

修改 属性

iii. 移除一个元素上指定的属性:

元素.removeAttribute("属性名")

移除 属性

iv. 判断元素上是否包含某个属性:

var bool=元素.hasAttribute("属性名")

2). 新DOM提供了简化版的HTML DOM属性:

i. HTML DOM将所有HTML标准属性，提前保存在了内存中的元素对象身上，只不过没有出现在页面上的属性值暂时默认为""而已

ii. 就可以用"元素.属性名"方式来操作HTML标准属性

iii. 如何:

获取一个元素指定属性的属性值: 元素.属性名

修改一个元素指定属性的属性值: 元素.属性名="属性值"

移除一个元素指定属性: 元素.属性名=""

如何判断一个元素上是否包含某个属性: 元素.属性名!=="" 就说明包含，否则就说明不包含

iv. 特例: class属性:

HTML中: <元素 class="btn btn-danger">...</元素>

Js中: 错误！元素.class="btn" 因为class已经是ECMAScript中的关键词，DOM中就不能再用class表示样式类

正确: 今后DOM中只要想操作class，都要被迫改名为className

元素.className="btn"

1. Bool类型的标准属性

a. 什么是: HTML标准中有一种属性只要放在元素的开始标签中就起作用，无需提供属性值，这样的属性就称为Bool类型的HTML标准属性:

b. 比如:

1). <input type="checkbox" checked>

2). <select>

<option>北京</option>

<option selected>天津</option>

<option>河北</option>

3). <button disabled></button>

c. 如何获取或修改:

1). 不能用旧DOM 4个函数修改。因为旧DOM 4个函数只支持字符串类型的属性。

2). 只能用HTML DOM "元素.属性名"方式来获取或修改。且属性值必须是bool类型true或false

d. css中提供了一种伪类选择器，专门匹配处于某种状态的元素:

1). :checked伪类: 专门匹配已选中的checkbox

2). :disabled 伪类: 专门匹配已禁用的表单元素

... ...

1. 自定义扩展属性

a. 什么是: HTML标准中没有规定的，程序员根据自身需要，自发添加到元素上的属性

b. 何时: 2种:

1). 代替其他选择器(id、类、元素)，作为查找触发事件的元素的条件

i. 其他选择器当做查找触发事件的元素的条件，有什么问题？

id选择器: 只能查找一个元素

类选择器: 类选择器本职工作是为元素添加css样式。而不是为js交互行为服务。所以，如果用类选择器作为查找触发事件的元素的条件，很可能因为样式的修改，而导致js也无法正常使用！

元素选择器: 实现同一种效果，不同的开发人员可能有多种选择。

比如实现按钮效果，可用: a、input、span、button、...

ii. 今后好的做法: 凡是可能触发事件的元素，都添加自定义扩展属性作为查找条件

比如:

<a class="btn" mybutton>

<input type="button" class="btn" mybutton>

<span class="btn" mybutton>

如何查找三个按钮: 属性选择器

var btns=document.querySelectorAll("[mybutton]")

//具有mybutton属性的元素

iii. 好处: 即使修改标签，修改class样式，都不会影响js。

iv. bootstrap就是这样做的：bootstrap中要想让元素拥有交互行为，都要添加自定义扩展属性:

比如: bootstrap中手风琴效果中每个可点击的标签上都有一个data-toggle="collapse"自定义属性。删除这个属性，不会影响样式。但是点击就没反应了！可见，bootstrap中所有交互行为都是用data-xxx自定义属性添加的。

2). 在网页中临时缓存稍后要用的部分业务所需数据

i. 不缓存这些数据会有什么问题？之后每次使用数据，都要重新请求服务器，加载速度慢，用户体验变差，又增加了服务器端的负担。

ii. 正确做法: 在首次加载时，就把后续可能使用的业务数据多请求一部分过来。临时缓存在网页元素的自定义扩展属性上。

iii. 以放大镜效果为例:

iv. 优点: 之后每次使用数据时，无需重复请求服务端，加载速度极快，用户体验好，且极大的减少了服务器端的负担。

c. 如何使用自定义扩展属性: (标准做法)

1). 在为元素添加自定义属性时:

i. 在html中写死: <元素 data-自定义属性名="属性值">

ii. 用js添加或修改自定义属性: 2种:

不能用HTML DOM"元素.属性名"方式添加。因为js中不允许写"元素.data-属性名"，又会和减法混淆

旧DOM 4个函数之一: 元素.setAttribute("data-自定义属性","属性值")

用HTML5提供的新做法: 元素.dataset.自定义属性="属性值"

说明: 用dataset方式添加，不用自己加data-前缀，当属性被添加到页面元素上时，dataset会自动补充data-前缀

2). 获取自定义属性: 2种

i. 也不能用HTML DOM "元素.data-属性名"。

ii. 旧DOM 4个函数之一: 元素.getAttribute("data-自定义属性")

iii. 用HTML5提供的新方法: 元素.dataset.自定义属性

3). 想用自定义扩展属性当条件查找元素时: 用属性选择器:

i. [data-属性名] 查找带有"data-属性名"的元素

ii. [data-属性名=属性值] 查找带有"data-属性名"，且属性值为指定的属性值的元素

1. 样式
2. .修改元素的内联样式
3. 在HTML中写死：<any style=”css属性：属性值”></any>
4. js中动态修改元素的内联样式：元素.style.属性=”属性值”;
5. 强调：
6. 所有css属性值都必须是字符串，加””;
7. 如果是长度或者大小的相关属性，则属性值必须加单位：如px em rem
8. css属性名带-,则必须改为 去横线变驼峰

比如：<div style="font-size:12px">...</div>

Js:要写为div.style.fontSize="12px";

1. .获取元素的css样式：坑：
2. 问题：用.style只能获得的内联样式，无法获得样式表中的样式，所以用.style获得的样式，无法获得大部分样式
3. 解决：今后只要获取样式都养获得计算后的样式
4. 什么是计算后的样式：最终应用道这个元素的所有样式的总和
5. 如何：2步
6. .先获得这个元素计算后的样式的总和(对象)

var style=getComputedStyle(元素);

1. .从style中获得个别css属性值：style.css属性值
2. getComputedStyle获得的样式是只读的，不可修改

因为计算后的属性不知道是哪来的，不知道有多少元素正在共用这个属性，如果擅自修改，很有可能牵一发而动全身！

(3).用class批量修改元素的样式

a.问题： .style.css属性=”属性值”一句话只能修改一个css属性。但是网页中一个效果的变化，

b. 解决：今后只要批量修改一个元素的多个css属性，都要用class代替.style.

c.如何：2步

1).在css中定义好不同情况的class

2).在js中根据不同情况为元素添加不同的calss： 元素.calssName=”class属性”

1. 添加新元素：3步
2. .创建一个空元素对象
3. var 元素对象=document.createElement(“标签名”);
4. 比如：创建一个a元素

var a=document.createElement(“a”);

结果 <a></a>,不能用

1. .为新元素设置必要的属性
2. 比如：a.href=”http://tmooc.cn”;

a.innerHTML=”go to tmooc”;

结果：<a href=”http://tmooc.cn”>go to tmooc</a>

1. 问题：a暂时只保存在内存中。还不知道应该加载到那个页面上。所以，暂时在页面上看不见
2. .将新元素添加到DOM树上指定父元素下3种：
3. 在父元素下所有子元素末尾添加： 父元素.appendChild(新元素);
4. 在父元素下某个指定子元素之前添加：父元素.insertBefore(新元素，某个指定子元素);
5. 替换父元素下某个子元素：父元素.replaceChild(新元素，某个指定子元素);
6. 强调：只有将新元素添加到DOM树上某个父元素下，浏览器才知道在哪添加这个新添加的元素,用户才能看到这个新元素
7. .为新元素设置必要的属性
8. .将新元素添加到DOM树上指定父元素下
9. 优化：
10. .尽量减少操作DOM树的次数
11. 因为：每修改一次DOM数，浏览器都要重新排版，重新绘制网页，二重新排版和重新绘制网页是非常复杂的过程。如果频繁重排重绘，就会导致加载效率降低，甚至闪屏
12. 如何减少操作DOM树的次数：2种

a.如果同时添加父元素和子元素，应该将现有内存中所有子元素加到父元素内，最后在一次性将父元素添加到DOM树上，这样只修改一次DOM树，只需要一次重排重绘即可

b. 如果父元素已经在页面上了，需要同时添加多个平级的子元素，应该借助于"文档片段对象"来减少修改DOM树的次数

1). 什么是文档片段对象: 内存中临时保存多个DOM元素的虚拟父元素

2). 何时使用文档片段对象: 只要同时添加多个平级子元素时，都要用文档片段对象

3). 如何使用: 3步

i. 先创建一个文档片段对象

var 文档片段对象=document.createDocumentFragment()

创建 文档 片段

ii. 将多个平级子元素先暂时添加到文档片段对象中暂存

文档片段对象.appendChild(子元素)

iii. 最后，将整个文档片段对象一次性添加到DOM树上指定父元素下

4). 强调: 文档片段对象将多个子元素运送到DOM树后，就自动释放了。不会占用页面的空间。也不会成为页面上真正的元素。

3.删除元素：父元素.removeChild(子元素);

二．HTML DOM常用对象(了解)

1. 什么是HTML DOM: 对旧DOM的属性和函数提供的一些简写方式。只对部分内容进行了简化。大部分旧DOM内容，还是没有简化。

2. img元素: 只在创建元素时，有简化:

1). var img=new Image();

2). 旧DOM: var img=document.createElement("img")

3. option元素: 在创建元素时，有简化:

1). var option=new Option("内容","值")

2). 结果: <option value="值">内容</option>

3). 旧DOM: 3句话

i. var opt=document.createElement("option")

<option></option>

ii. opt.value="值"

<option value="值"></option>

iii. opt.innerHTML="内容"

<option value="值">内容</option>

3. table元素: HTML DOM中采用逐级管理:

(1). Table管着行分组对象:

a. table可以添加行分组:

1). var thead=table.createTHead(); //做了两件事，相当于之前的两句话

i. 创建一个行分组，相当于var thead=document.createElement("thead")

ii. 自动将新创建的行分组，追加到table下， 相当于table.appendChild(thead)

2). var tbody=table.createTBody();

3). var tfoot=table.createTFoot();

b. table可以删除行分组

1). table.deleteTHead();

2). table.deleteTFoot();

3). 没有table.deleteTBody();

c. table可以获取行分组

1). table.tHead //因为table规定只能有一个thead

2). table.tFoot //因为table规定只能有一个tfoot

3). table.tBodies[i] //但是table规定，可以有多个tbody元素

(2). 行分组对象管着行对象tr

a. 行分组可添加行:

1). var tr=行分组.insertRow(i)

2). 意为: 在行分组中创建一个新行，并插入到行分组下下标为i的位置

3). 相当于旧DOM中2句话:

i. 创建tr: var tr=document.createElement("tr")

ii. 将tr插入到行分组i位置: 行分组.insertBefore(新tr, 行分组.children[i]);

4). 问题: 如果行分组中i位置，已经有一个现有行，则在i位置插入新行，原i位置及其之后的行，被向后挤一位。不会替换

5). 两个位置最常用:

i. 末尾追加一个新行: var tr=行分组.insertRow() //默认追加到末尾

ii. 开头插入一个新行: var tr=行分组.insertRow(0)

b. 行分组可删除行

1). 本来: 行分组.deleteRow(i)

2). 意为: 删除行分组内的第i行

3). i : 要求用行在行分组内的相对下标位置

4). 问题: 将来table中可能有很多行，而程序没有长眼睛，不知道要删除的行是第几行。

5). 解决:

i. 每个行对象tr上都有一个tr.rowIndex属性，记录着这一行在整个表中的下标位置

ii. 新问题: 但是行在整个表中的下标位置，因为要从表头行开始算起，所以比行分组内的相对下标位置通常都大，如果硬用tbody.deleteRow(tr.rowIndex)，其实删除的不是tr这一行，而是tr下方某一行

iii. 最好的删除行的办法: 用table作为.前的主语

table.deleteRow(tr.rowIndex);

因为.前主语变成整个table了，所以deleteRow()需要的下标，也就变成行在整个table中的下标位置。刚好和tr.rowIndex所表示的意义相同！

c. 行分组可获取一行: 行分组.rows[i]

(3). 行对象tr管着格对象td

a. 行可以添加一格: var td=tr.insertCell(i) //通常不写i，默认在行尾追加新格

b. 行可以删除一格: tr.deleteCell(i)

c. 行可以获取一格: tr.cells[i]

4.form元素

(1).获取form元素时：

a.var form=document.forms[i或者id或者name名];

b.因为在document对象中已经把网页中的所有form元素集中存储在了forms集合中

c.比如，如果网页中只有一个form元素: var form=document.forms[0]

(2). 获取form内的表单元素时:

a. 其实form内的所有表单元素（input、textarea、select、button），都被集中保存在了form.elements集合中

b. 获取form内的表单元素: 2种:

1). var 表单元素=form.elements[i或id或name]

2). 如果表单元素有name属性，则可以: form.name名

(3). form.length 统计表单内所有表单元素的个数

比如: 获得表单中倒数第二个提交按钮: var btn=form.elements[form.length-2]

(4). 让表单元素自动获得焦点: 表单元素.focus();

最大的用处，就是减少用户一步点击操作！

补: 浏览器三大对话框:

1. prompt: 输入框: 带一个可输入内容的文本框的弹出框

2. alert: 警告框: 只有一个"确定"按钮的提示框，用户没有选择的余地

3. confirm: 确认框: 包含两个按钮"确认"和"取消"的提示框，允许用户选择。

如果用户点确认按钮，则确认框返回true，说明用户同意继续

如果用户点取消按钮，则确认框返回false，说明用户不同意继续

三大对话框因为有致命的缺点:

1. 样子很丑，还改不了！因为对话框是用浏览器低层代码写死的。用普通的css无法修改。

2. 兼容性问题，不同浏览器中对话框样子各部相同！

所以今后几乎不再使用了。

今后所有对话框都是用div+css+定位做的。

小程序->在线->DOM->day04 扩展: DOM版学子商城 / videos/day01 1、2、3、4、5

下载并解压code/中public压缩包，其中包含学子商城静态页面，可作为起步。

然后，跟着视频中的步骤，直接在public文件夹内实现学子商城首页商品动态加载。

大部分同学后端不熟练或者MySQL环境有问题，可完全不需要管后端和mysql，只要将视频中ajax()中的/xxx相对端口地址换成http://xzserver.applinzi.com/xxx地址就可以用我的新浪云当服务器端。

实现过程中有问题可参考code/中public\_finished压缩包中的完成版本的代码纠错。

code/中xzserver\_start是带服务器端的起步项目源代码，基础比较好的同学可以用。

xzserver\_finish是带服务器端的起步项目做完首页和详情页的源代码，基础比较好可以参考。

三．BOM Browser Object Model

1.什么是BOM: 一套专门操作浏览器窗口的对象和函数的集合

2.何时：只要想操作浏览器窗口相关的功能，都用BOM

比如：打开和关闭窗口，窗口滚动相关操作

3.BOM由一系列对象掌握一摊组成的：

window 对象 囊括一切

history 对象 控制前进和后退

location 对象 页面跳转

document 对象 包括一切dom相关操作

screen 对象 包括当前显示器的信息

navigator 对象 包含浏览器的配置信息

event 对象 包含所有和事件相关的信息和操作

1. BOM最大的问题：没有标准！浏览器兼容性极差
2. window 对象：3个角色
3. . window 代替了ECMAScript中的global,充当全局作用域对象，保存全局变量和全局函数
4. window包含了所有原生的对象和函数：ECMAScrip+DOM+BOM
5. Window代表了当前正在打开的浏览器窗口，比如：
6. 获得当前窗口大小:2组大小
7. 当前浏览器的完整窗口大小：window.outerWidth window.outerHeight
8. 仅用于显示网页的文档显示区部分大小：

window.innerWidth window.innerHeight

b.window还可关闭当前窗口：window.close();

6.打开新链接的四种情况

(1). 在当前窗口打开新链接，可后退

a. HTML中: <a href="新url" target="\_self">

b. js: window.open( "新url", "\_self")或者location.href="url";

(2). 在当前窗口打开新链接，禁止后退

a. HTML中: 做不了

b. js: location.replace("新url") //新url会代替history中之前旧url，来阻止后退。

(3). 在新窗口打开新链接，允许同时打开多个

a. HTML中: <a href="新url" target="\_blank">

b. js: window.open("新url", "\_blank")

(4). 在新窗口打开新链接，只允许打开一个

a. HTML中: <a href="新url" target="自定义窗口名">

b. js: window.open("新url", "自定义窗口名")

(5). 原理:

a. 其实浏览器打开的每个窗口在内存中低层都有一个唯一的窗口名

b. 浏览器规定，相同名称的窗口只能打开一个，后打开的同名窗口，会自动将之前打开的同名窗口覆盖，只保留一个最新的同名窗口

c. a元素的target属性和window.open()的第二个参数，都是在为新窗口指定内存中低层的窗口名。

d. 新窗口中可用window.name查看自定义的新窗口的名字

强调: 今后不允许使用"name"作为全局变量名，会和BOM中的window.name冲突

e. 两个预定窗口名:

1). \_self: 不指定窗口名，自动获得当前窗口自己的名字，作为新窗口的名称。——自杀

2). \_blank: 不指定任何窗口名，但是浏览器低层决不会让窗口名空着，浏览器会自动随机分配窗口名——永不冲突，所以，可以打开多个新窗口

7. history对象:

(1). 什么是: 专门保存当前窗口打开后，成功访问过的url的历史记录数组

(2). 强调: history中的内容，没有对我们开放，看不见的。

(3). 原理:

a. 当浏览器窗口打开时，window中会自动创建一个history对象

b. 只要当前窗口成功访问过一个url，浏览器就会自动把这个url以push()方式压入history中保存。

c. 结果: 先访问的url在下层压着，后访问的url在上层

(4). History中有个指针，默认指向最新压入的新url。可通过前进后退按钮控制指针上下移动。

(5). 如果当前指针位置下方有旧url，则可以后退，否则不能后退

如果当前指针位置上方有新url，则可以前进，否则不能前进

(6). 如何使用history对象:

a. 前进一步: history.go(1)

b. 后退一步: history.go(-1)

如果后退一步不管用，可以history.go(-2) 后退两步

c. 刷新: history.go(0)

8. location对象:

(1). 什么是: 专门保存当前窗口中地址栏中url信息的对象，并提供了页面跳转相关的功能

(2). 何时: 2种情况:

a. 分段获得url中各个部分的值时

b. 执行个别跳转操作时

(3). 如何:

a. 分段获得url中各个部分的值:

1). location.href 获得的是完整的url地址

2). location.protocol 获得的是协议部分，比如: http: https: ...

3). location.host 获得的是主机名+端口号，比如: localhost:3000

4). location.hostname 仅获得主机名，比如: localhost

5). location.port 仅获得端口号, 比如: 3000

6). location.pathname 获得相对路径, 比如: /users/login.html

7). location.search 获得?以及之后的查询字符串参数，比如: ?uname=dingding&upwd=123456

8). location.hash 获得的是#及其之后的锚点地址，比如: #collapse

b. 执行个别跳转操作：

1). 也可以实现在当前窗口打开新链接，禁止后退: location.href="新url"

2). 独有的: 在当前窗口打开新链接，禁止后退: location.replace("新url")

3). 也可以刷新页面: location.reload()

9. navigator对象:

(1). 什么是: 专门保存浏览器配置信息的对象

(2). 何时: 如果想获得浏览器软件相关的配置信息时

(3). 如何: 2个最常用的属性:

a. navigator.plugins: 获得当前浏览器已经安装的插件列表

b. navigator.userAgent: 获得当前浏览器的名称和版本号

1. 事件(重要)
2. 回顾事件的概念
3. 什么是：浏览器自动触发的或者延后胡搜到能够触发的页面中元素状态或内容的改变
4. 什么是事件属性：每个元素身上都有一系列 on事件名 格式的属性，
5. 事件属性的作用：当某个元素发生事件时，浏览器就会自动查找这个元素身上的事件属性行提前保存好的事件处理函数

4. 什么是事件绑定: 提前将事件发生时希望自动执行的函数，提前保存在元素的事件属性上。

5. 什么是事件处理函数: 提前保存在事件属性上，事件发生时能够自动执行的函数，称为事件处理函数。

二. 事件绑定: 3种:

1. 在HTML中:

(1). HTML中: <元素 on事件名="js语句">

(2). 问题: 不符合内容与行为分离的原则，不便于维护

2. 在js中，用赋值方式绑定:

(1). 元素.on事件名=function(){ ... }

(2). 问题: 同一个元素的同一个事件，只能绑定一个处理函数。

3. 在js中，用添加事件监听对象方式:

(1). 元素.addEventListener("事件名", 事件处理函数)

添加 事件监听对象

(2). 原理:

a. 除了每个元素上有事件属性用于保存事件处理函数之外，在浏览器内存中还有一个巨大的事件队列，也可用于为元素保存事件处理函数。

b. 每执行一次addEventListener()，就会自动创建一个事件监听对象，对象中保存三个属性{ 哪个元素, 哪个事件名, 处理函数}，并自动将事件监听对象添加到浏览器的事件队列中保存

c. 当某个元素发生事件时，浏览器会先查找当前元素自己身上的事件属性上的处理函数，自动执行。然后浏览器还会继续去事件队列里查找，是否有当前元素的当前事件匹配的事件监听对象。只要找到匹配的事件监听对象，就可以自动执行，事件监听对象中保存的处理函数。找到几个就执行几个。

1. .移除事件监听对象
2. 元素.removeRvenListener(“事件名”，原事件处理函数);
3. 坑：
4. .移除事件监听时，如果原事件处理函数只是写的和绑定时的处理函数一样，但是依然使用匿名函数，是无法移除绑定的事件处理函数的

因为，匿名函数的function是new Function的意思，是有创建了一个新的函数，即使写的一样，也不是原函数

2).正确做法：如果一个事件有可能被移除，则绑定时要就必须使用有名称的函数，移除时，才能使用函数名，找到原函数对象，才能移除。

3). 但是事件队列中有限制: 元素对象，事件名，函数对象完全相同的事件监听监听对象，只能添加一个！不能重复添加！（暂时不讨论获得多个跟踪导弹的情况）

3.事件模型

1.什么是：从事件触发开始到所有处理函数执行完，所经历的过程

2.包括：3个阶段

(1).捕获(capture)阶段：从document节点开始，有内向外，逐级遍利当前触发事件的的元素的各级父级元素上绑定的事件处理函数——只记录有哪些函数，暂不触发！，

(2). 目标触发阶段: 浏览器总是优先触发目标元素上绑定的事件处理函数

a. 什么是目标(target)元素: 最初实际想点的那个元素

(3). 冒泡执行: 当目标元素上的处理函数执行完之后，浏览器会从目标元素开始，由内向外，按照捕获阶段时记录的反向顺序依次触发各级父元素上绑定的事件处理函数

4.事件对象

（1）.什么是：事件发生时，浏览器自动创建的保存事件信息的对象

（2）.何时：2种：

a.如果想获得事件的信息时，比如事件发生时鼠标的位置

b.如果想改变事件默认的行为时，比如不想冒泡

（3）.如何：2步：

a.获取事件对象：其实事件对象总是作为事件处理函数的第一个实参值自动传入

元素.on事件名=function(e){

// 当事件发生时，形参e就自动获得了事件对象 event

}

b.就可以用e来访问事件对象中的属性，调用事件对象提供的方法

（4）.取消冒泡/停止蔓延：e.stopPropagation();

强调：e.stopProgation();放的是父级元素上的处理函数，而不是自身元素上的，所以e.stopProgation();写前写后都可以，

5.事件委托/利用冒泡:

(1).优化：尽量减少事件监听的个数

(2).为什么：应为浏览器触发事件，是通过遍历事件监听队列的凡是符合查找条件的监听对象触发的。因为如果监听对象过多，遍历就会很慢，导致事件响应变慢

(3).如何：事件委托/利用冒泡

a.如果多个平级子元素都要绑定相同的事件时，应该只给父元素绑定一次事件，所有子元素可用过冒泡机制共用父元素上的一个事件处理函数----称为事件委托/代理

b.问题1：如果事件绑定在父元素上，则事件发生时，this指向父元素，不在指向实际点击的子元素了。 。所以，事件委托中，this无法用于获取当前点击的子元素了

1).解决：今后如果使用事件委托，事件处理函数中必须使用e.target代替this获得实际点击的子元素

2).e.target:专门用于保存最初点击的目标元素的特殊属性，一旦保存住最初点击的目标元素，就不会随冒泡而改变！

c.问题2：用户可能点在父元素的内部的无关紧要的地方，或者直接点在父元素上，也会触发事件，但是，此时我们不想让他触发事件

1).解决：如果使用事件委托开优化代码，都要先判断用户点击的目标元素e.target是不是想要的。只有e.target是符合要求的，就继续执行代码，否则，就神魔也不干

2).如何判断目标元素是不是自己想要的，一切可以将目标元素和不想要元素分开的一切特征(nodeName、className ....).都可以作为判断条件

|  |
| --- |
| 总结: 事件委托3步  1. 事件只绑定在父元素上一份即可  2. e.target代替this  3. 判断e.target是否是想要的！ |

1. 阻止默认行为
2. .个别元素身上自带的默认的操作
3. 比如;<a href=”#top”>click me</a> 会在地址栏最后添加#top，对以后程序有害
4. 如果元素自带的默认行为不想要，可以用事件对象来阻止默认行为
5. 如何：e.preventDefault(); 阻止 ，默认
6. 强调：e.preventDefault()只阻止默认行为，不会阻止自己写的事件处理函数
7. 鼠标坐标：当事件发生时，事件对象会保存住鼠标所在位置的3组x,y坐标
8. .当前鼠标位置相对于屏幕左上角的坐标：e.screenX e.screenY
9. .当前鼠标位置相对于文档显示区左上角的坐标：e.clientX e.clientY
10. .当前鼠标位置相对于当前事件的元素的左上角的坐标： e.offsetX e.offsetY
11. 窗口滚动事件
12. .其实窗口滚动条每次滚动时，都会自动触发一个事件：

window.onscroll=function(){

1. .获得页面滚动过的距离：

var scrollTop=document.bodyscrollTop

}