小程序: WEB问题速查

1. 问题清单: 历届师兄师姐在今天课程中集中问过的高频的重点的问题！

那些自己发现不了问题，又迫切想知道其它同学都问了什么问题，的同学

小程序->首页->DOM->day01

当老师在回答其它同学的问题时，你不要闲着！你可以打开问题清单浏览别人曾经问过的经典问题！

2. 高清无码小视频: 小程序->在线->DOM->day01

2部分: 1. 课上重点知识和案例的回放！2. 重点高频笔试面试题讲解扩展视频

万一家里或中心网络断了！暂时看不了直播了！

因为某些事儿请假了，错过了当天的直播！

课下复习时，觉得TMOOC视频太大，浏览不方便或TMOOC不能看了！

——打开小程序中"在线"，就可下载观看当前所讲内容和当天所讲全部内容的回放。

基础比较好的，确实学有余力的同学，想拿高薪的同学，在我答疑时，不要闲着:

要看今天扩展视频: 小程序->在线->DOM->day01

4. DOM 优化 查找元素 。。。

作业: 高频笔试题 动态集合与非动态几何

作业: 高频鄙视题 多个按钮 ... ...

作业: 高频笔试题 递归遍历一个父元素下所有后代元素

是否高频: 看我的朋友圈！历届拿到高薪的师兄师姐，反馈回来的鄙视面试重点内容！

有些有心的同学，已经把朋友圈里反复出现的高频词总结出来！

3. 单词列表: 小程序->单词->DOM->day01

第三阶段上课约定:

1. 上课千万不要跟着记笔记和写代码！

2. 凡是拼写错误，自己懒得找！我可以帮你找，但是找到后，错误的单词你要手抄20遍，并拍照！

出错: 先开f12，console，看错误提示！

3. 每个带注释的案例至少写三遍:

(1). 注释+代码抄一遍

(2). 保留注释，删除代码，自己试着把代码翻译回来！

(3). 删除注释和代码，自己试着用自己的话把注释写回来！

一个例子只有自己可以把人话的注释写回来！才算自己会！

问题: 一听就会一做就废！原因: 你自己无法用人话精确描述自己想干什么！

如何提问: （vip待遇的剩余时间11天）

1. 上课时，案例中，笔记中那句话哪行代码看不懂，立刻问！

2. 小程序中问题清单里哪个问题和回答看不懂！立刻问！

3. 个人项目中哪个功能不会做！立刻问！

4. 刷笔试题时，遇到不会的笔试题！立刻问！

谁问的问题越多！谁的培训费交的越值！

正课:

1. 什么是DOM

2. DOM树  
3. 查找元素

4. 事件绑定基础

5. 示例: 购物车

一. 什么是DOM: Document Object Model

文档 对象 模型

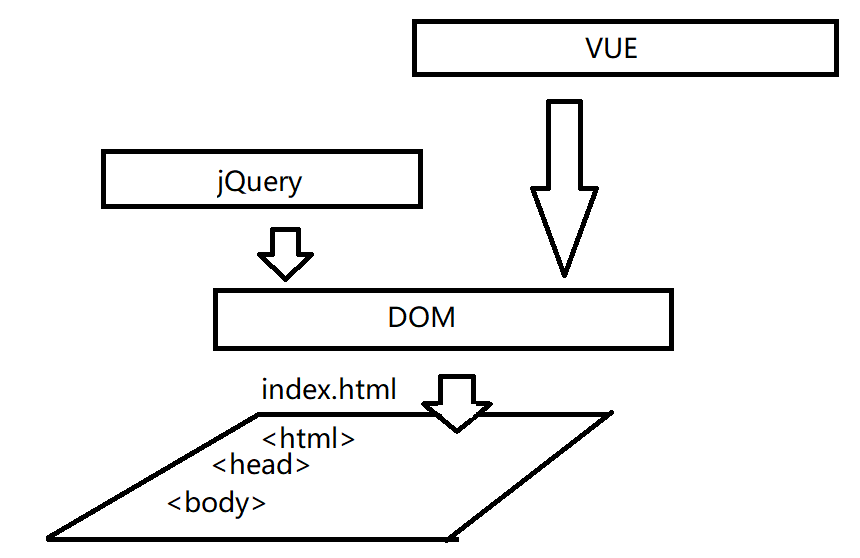
1. 什么是: 一套专门操作网页内容的对象、函数和属性的整体！

2. 为什么: ECMAScript仅规定了js语言的核心语法（怎么写对，怎么写不对）。但是，ES标准没有规定如何使用js操作网页内容！

3. 何时: 将来只要操作网页内容，都必须使用DOM提供的对象、函数和属性。

4. 问题: DOM提供的对象、函数和属性比较繁琐，不太好用！

5. 解决: 后人在DOM提供的对象、函数和属性基础上，又封装了简化的jQuery和Vue。



6. 问题: 因为ECMAScript标准不包括DOM的对象和函数。所以早期DOM没有标准，每种浏览器操作网页的方法各不相同！——严重的兼容性问题

7. 解决: W3C组织作为第三方机构又制定了一个DOM标准，规定了所有浏览器操作网页内容的标准方式——今后使用DOM标准操作网页，几乎所有浏览器100%兼容。

8. 学了DOM可以做哪些事儿？5件事: 增删改查+事件绑定

二. DOM树:

对应小程序视频: 小程序->在线->DOM->day01 0 DOM Tree 树

1. 什么是: 一个html网页中所有元素、内容和属性，在内存中都是被保存在一棵树形结构上！

2. 为什么: 因为HTML网页中的元素和内容都有明显的复杂的上下级包含关系，而树形结构是最直观的保存和反应上下级包含关系的结构！

3. 何时: 当浏览器窗口加载了一个HTML网页文件后，浏览器都会扫描HTML文件的内容，并在内存中创建DOM树以及DOM树上的节点对象来保存HTML中每项内容。

4. 节点: 树形结构或网状结构中多条线交汇的点，就称为节点！

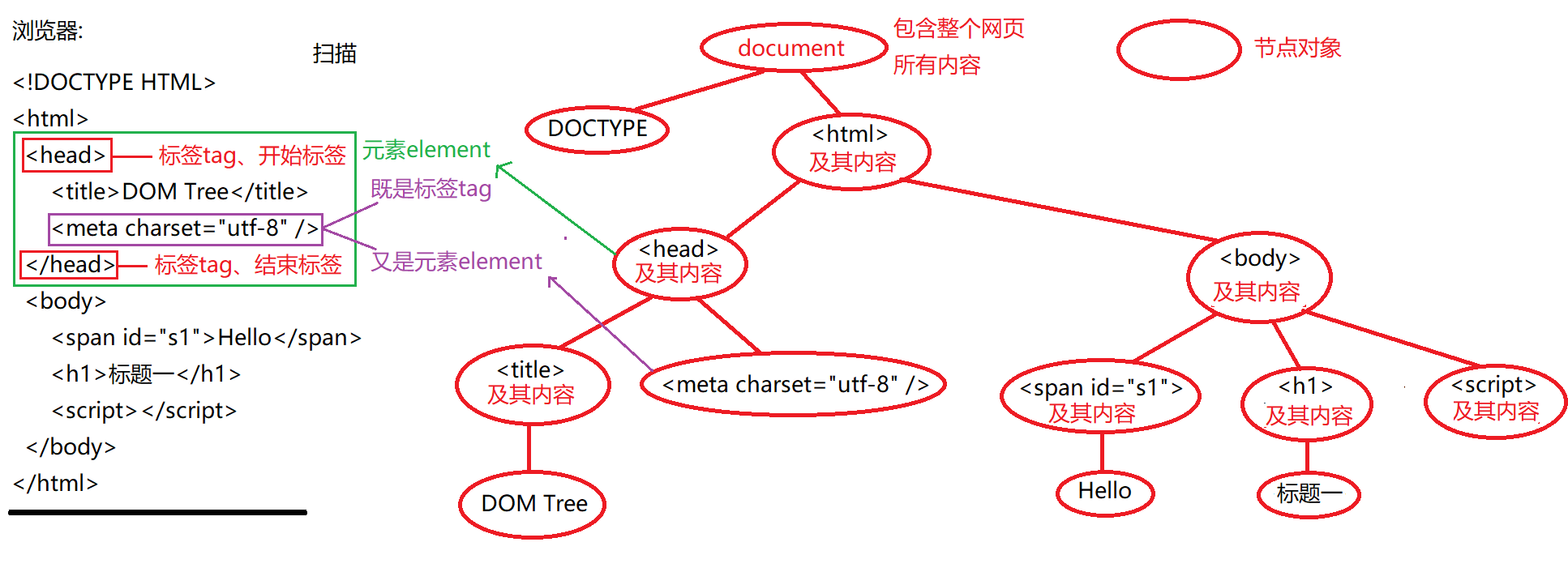
5. 如何:

(1). 先在内存中创建一个唯一的树根节点对象: document

(2). 浏览器扫描HTML文件内容

(3). 每扫描到一项内容(元素、文本、属性)，浏览器就会在树根节点document下对应位置，创建一个节点对象，保存当前扫描到的一项内容

(4). 当扫描整个HTML文件结束，内存中就形成了一棵属性结构



三. 查找元素: 4种:

对应小程序视频: 小程序->在线->DOM->day01 1 按节点关系查找

1. 不需要查找就可直接获得的元素: 4种

(1). document对象:  **document** —— 代表整个网页所有内容

(2). <html>元素对象: **document.documentElement**

(3). <head>元素对象: **document.head**

(4). <body>元素对象:  **document.body**

2. 按节点间关系查找: 2大类关系，6个属性:

(1). 其实有两种DOM树: 节点树和元素树

a. 节点树: 包含所有节点对象(元素、文本等)的完整树结构

b. 元素树: 仅包含元素节点，不包含文本等其他类型节点的简化版树结构

(2). 节点树上的关系:

a. 父子关系: 4个属性:

1). 获得一个节点对象的父节点: 节点对象.parentNode

父 节点

2).获得一个节点对象下的所有直接子节点对象: 节点对象.childNodes

孩子节点们

3).获得一个节点对象下的第一个直接子节点: 节点对象.firstChild

第一个孩子

4).获得一个节点对象下的最后一个直接子节点: 节点对象.lastChild

最后一个孩子

b. 兄弟关系: 2个属性：

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

前一个兄弟

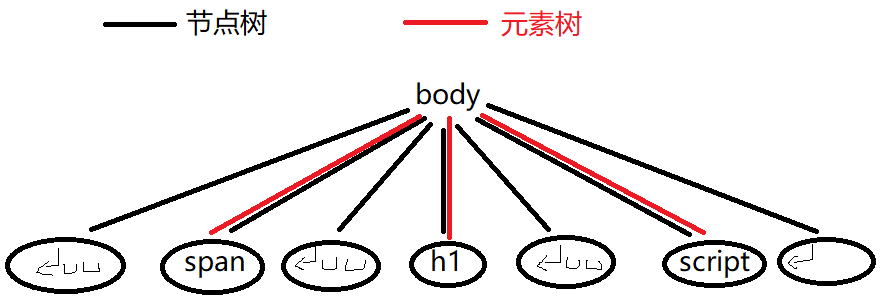
2). 获得当前节点对象相邻的下一个兄弟节点: 节点对象.nextSibling

下一个兄弟

(3). 节点树的问题: 节点树认为连看不见的换行和空格，也是文本类型的节点对象。也会成为子节点和兄弟节点。——严重干扰我们的查找结果！

(4). 解决: 新DOM标准中在原有完整DOM树基础上规定了一棵新的DOM树——元素树。元素树上的关系仅指向元素类型的节点。不再指向其他类型的节点。——好处，查找结果不会受到看不见的空字符的干扰！

(5). 强调: 元素树，不是一棵新树。仅仅是原完整节点树中的部分元素节点的一个子集。



(6). 总结: 今后按节点间关系查找时，都用元素树，而不用节点树

a. 父子关系: 4个属性:

1). 获得一个元素对象的父元素: 元素对象.**parentElement**

父 元素

2).获得一个元素对象下的所有直接子元素: 元素对象.**children**

孩子们

因为一个元素可能包含多个子元素，所以children属性返回一个类数组对象，其中包含找到的所有直接子元素对象。下标从0开始！

3).获得一个元素对象下的第一个直接子元素: 元素对象.**firstElementChild**

第一个 元素 孩子

4).获得一个元素对象下的最后一个直接子元素: 元素对象.**lastElementChild**

最后一个 元素 孩子

b. 兄弟关系: 2个属性：

1). 获得当前元素对象相邻的前一个兄弟元素:

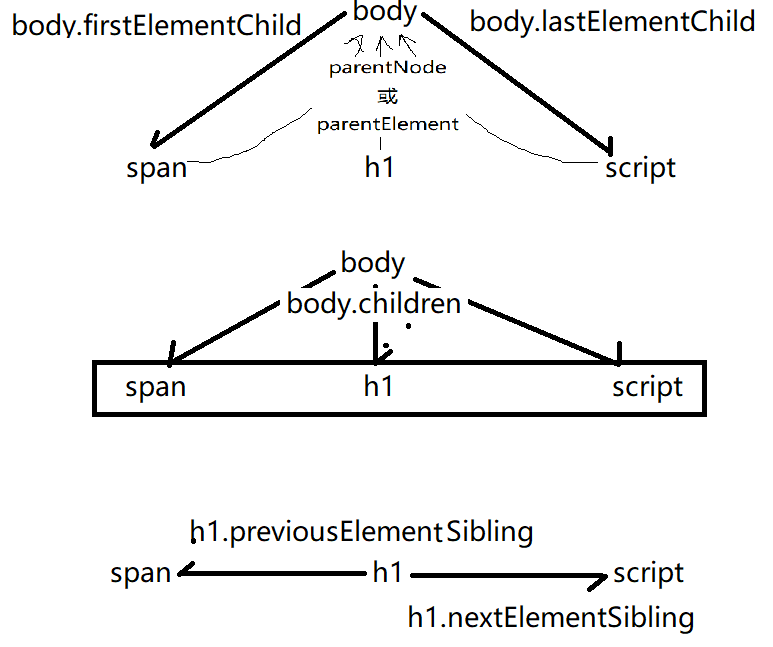
元素对象**.previousElementSibling**

前一个 元素 兄弟

2). 获得当前元素对象相邻的下一个兄弟元素:

元素对象.**nextElementSibling**

下一个 元素 兄弟



(7). 示例: 使用节点间关系，查找body下的元素

1\_domTree.html

|  |
| --- |
| <!DOCTYPE HTML>  <html>  <head>    <title>DOM Tree</title>    <meta charset="utf-8" />  </head>  <body>    <span id="s1">Hello</span>    <h1>标题一</h1>    <script>      //想输出document对象      console.log(document);      //想输出html元素对象      console.log(document.documentElement);      //想输出head元素对象      console.log(document.head);      //想输出body元素对象      console.log(document.body);      //本例暂时不用live server运行      //因为live server会自动插入一个多余的script，干扰我们的查找。      //应该直接在硬盘上的文件夹中找到这个.html文件，右键，打开方式，用chrome打开      //想获得body的父节点: 应该是<html>      //既可以用parentNode，又可以用parentElement      var html=document.body.parentNode;//首选      //var html=document.body.parentElement;      console.log(html);//正确      //因为在网页中能当爹，包含其它子内容的，只可能是元素。<开始标签>子内容</结束标签>      //想获得body下的所有直接子元素: 应该是3个      //错误做法:      //var childNodes=document.body.childNodes;      //console.log(childNodes);//7个      //正确做法:      var children=document.body.children;      console.log(children);//3个      //想获得body下的第一个直接子元素: 应该是span      //错误做法:      //var span=document.body.firstChild;      //正确做法:      var span=document.body.firstElementChild;      console.log(span);      //想实现body中最后一个直接子元素: 应该是script      var script=document.body.lastElementChild;      console.log(script);      //想获得body中第二个孩子：应该是h1      var h1=document.body.children[1];      console.log(h1);      //想获得h1，还可以通过span的下一个兄弟元素获得      var h1=span.nextElementSibling;      console.log(h1);      //想获得h1，还可以通过script的前一个兄弟元素获得      var h1=script.previousElementSibling;      console.log(h1);    </script>  </body>  </html>  运行结果: |

(8). 何时: 今后如果已经获得一个DOM元素，想找它周围附近的其它DOM元素时，就用按节点间关系查找。

3. 按HTML特征查找: 4个方法:

对应小程序视频: 小程序->在线->DOM->day01 2 按HTML特征查找

(1). 何时: 只要还没有获得任何DOM元素，就要查找元素，或要查找的元素离当前获得的元素很远，就选择按HTML特征查找。

(2). 4大特征:

a. 按**id**名查找一个元素:

1). var 一个元素对象=document.getElementById("id名")

在整个网页中获取 元素 以id为条件

2). 返回值:

i. 如果找到符合条件的一个元素，就返回一个**元素对象**

ii. 如果没找到符合条件的元素，就返回null

3). 强调:

i. **必须用document.作为开头**

ii. 因为按id查找，只能找到一个元素，所以Element不带s结尾，是单数形式

iii. 如果网页中确实有多个相同的id，则getElementById()永远只能找第一个符合条件的。

b. **按标签名**查找多个元素:

1). var 类数组对象=任意父元素.getElementsByTagName("标签名")

在指定父元素下获取多个元素以标签名为查找条件

2). 返回值:

i. 如果找到符合条件的多个元素，则多个元素放在一个类数组对象中返回

ii. 如果找不到符合条件的元素，则返回空类数组对象: { length:0 }

3). 强调:

i. **不一定必须用document作为开头。**可以用任意父元素作为开头！仅在当前父元素下查找符合条件的元素。——好处，限制查找范围，提高查找效率

ii. 因为可能返回多个符合要求的子元素，所以方法名中Elements是s结尾，复数形式。

iii. 不但查找直接子元素，而且在所有后代中查找符合要求的元素

iv. 问题: getElementsByTagName()注定会返回一个类数组对象。即使只找到一个元素，也会放在类数组对象中返回！但是，我们希望使用的绝不是类数组对象，我们通常希望使用的是这一个DOM元素对象

解决: 只要在查找结果基础上继续加[0]，从类数组对象中0位置，取出找到的唯一的一个DOM元素对象。

比如: body中只有一个span

错误:

var span=document.body.getElementsByTagName("span");

//返回的span不是DOM元素对象，而是一个类数组对象，其中0位置放着找到的唯一一个DOM元素对象span： { 0:span对象, length:0 }

正确:

var span=document.body.getElementsByTagName("span")[0];

//返回的span就是HTML中的DOM元素对象。[0]是从查找结果类数组对象中0位置取出唯一一个DOM元素对象的意思。

c. 按**class**名查找多个元素:

1). var 类数组对象=任意父元素.getElementsByClassName("class名")

在任意父元素下获取多个元素以className为条件

2). 返回值:

i. 如果找到符合条件的多个元素，则多个元素放在一个类数组对象中返回

ii. 如果找不到符合条件的元素，则返回空类数组对象: { length:0 }

3). 强调:

i.**可以用任意父元素作为开头！**仅在当前父元素下查找符合条件的元素。

——好处，限制查找范围，提高查找效率

ii. 方法名中Elements是s结尾，复数形式。

iii.在所有后代中查找符合要求的元素

iv. 如果只找到一个元素，也必须加[0]，才能取出这唯一的DOM元素对象

v. 如果一个元素上同时被多个class修饰，那么只用其中一个class名就可找到该元素。

d. 按**name**属性查找多个表单元素:

1). var 类数组对象=document.getElementsByName("name名")

在整个页面中获取多个元素以name名作为条件

2). 返回值:

i. 如果找到符合条件的多个元素，则多个元素放在一个类数组对象中返回

ii. 如果找不到符合条件的元素，则返回空类数组对象: { length:0 }

3). 强调:

i.**必须用document.作为开头！**不能以任意父元素开头！

ii. 方法名中Elements是s结尾，复数形式。

iii. 如果只找到一个元素，也必须加[0]，才能取出这唯一的DOM元素对象

(3). 示例: 使用按HTML特征查找四个函数查找想要的元素

2\_iterator.html

|  |
| --- |
| <!DOCTYPE HTML>  <html>    <head>      <title>遍历节点树</title>      <meta charset="utf-8"/>    </head>    <body>      <span>Hello World !</span>      <ul id="nav">        <li class="parent">电影</li>        <li class="parent">综艺          <ul>            <li class="child active">跑男</li>            <li class="child">爸爸</li>            <li class="child">极限</li>          </ul>        </li>        <li class="parent">剧集</li>      </ul>      用户名: <input type="text" name="uname"><br/>      性别:      <label>        <input type="radio" name="sex" value="1">男      </label>      <label>        <input type="radio" name="sex" value="0">女      </label>      <script>        //想查找id为nav的ul        var ul=document.getElementById("nav");        console.log(ul);        //强调: 所有元素不查找，是不能直接使用！        //要想使用某个元素，都必须先查找再使用        //想查找ul下的所有li        var lis=ul.getElementsByTagName("li");        console.log(lis);        //想在body下找span        //只可能找到一个span时        var span=          //document.body.getElementsByTagName("span")          //返回类数组对象，其中包含一个span元素对象          document.body.getElementsByTagName("span")[0]          //直接返回类数组对象中0位置的span元素对象。        console.log(span);//        //想查找ul下所有class为parent的li        var parents=ul.getElementsByClassName("parent");        console.log(parents);        //想查找ul下所有class为child的li        var children=ul.getElementsByClassName("child");        console.log(children);        //想查找ul下class为active的一个li        var liActive=ul.getElementsByClassName("active")[0];        console.log(liActive);        //想查找name为sex的两个元素        var radios=document.getElementsByName("sex");        console.log(radios);        //想查找name为uname的一个文本框        var txtUname=document.getElementsByName("uname")[0];        console.log(txtUname);      </script>    </body>  </html>  运行结果: |

4. 按选择器查找: 2个函数:

今后，只要元素藏的很深，查找条件很复杂时，都要选择按选择器查找

(1). 只查找一个符合条件的元素:

var 一个元素对象=任意父元素.querySelector("任意选择器");

(2). 查找多个符合条件的元素:

var 类数组对象=任意父元素.querySelectorAll("任意选择器");

对应小程序视频: 小程序->在线->DOM->day01 3. 购物车 shoppingcart

四. 事件绑定基础:

1. 什么是事件: 浏览器自己触发的或用户手工触发的页面中元素内容和状态的改变

2. 什么是事件处理函数: 当事件发生时，希望浏览器自动调用的函数。

3. 什么是绑定事件: 提前在元素的事件属性上保存一个事件处理函数

结果: 等到事件发生时，自动执行该事件处理函数。

4. 为什么绑定事件: 网页中所有元素，默认即使发生了事件，也什么都不做。

5. 何时需要事件绑定: 如果希望一个元素触发事件时可以自动执行一个操作，都要提前为元素绑定事件处理函数。

6. 如何绑定事件处理函数:

(1). 在HTML中绑定: —— 不好，几乎不用！

a. HTML中: <元素 on事件名="函数名()" >

b. js中: function 函数名(){ ... }

c. 问题:

1). 不符合js程序与HTML内容分离的原则，不便于维护！

2). 开始标签中写死的事件属性，新生成的元素，无法自动获得。

d. 结论: 在DOM和jQuery中绝对不会使用HTML绑定事件处理函数

(2). 在js中用赋值方式绑定:

a. 查找到要绑定事件的元素对象

b. 每个元素对象上，都包含很多on开头的特殊事件属性

c. 当这个元素上发生事件时，浏览器会自动找到这个元素上对应的onxxx事件属性，并执行onxxx事件属性上提前绑定的事件处理函数。

d. 所以，我们需要提前为这个元素对象的某个onxxx事件属性赋值一个事件处理函数备用！

**元素对象.on事件名=function(){**

//当事件发生时，才执行的代码

**}**

e. 强调: 这里的function，仅赋值并保存在on事件名属性中，暂不执行！

f. 优点:

1). 可集中写在js中，便于维护

2). 是js语句，想什么时候执行，就什么时候执行！想给谁绑定，就给谁绑定！——灵活！

g. 示例: 只为第一个按钮绑定单击事件处理函数

3\_shoppingcart.html

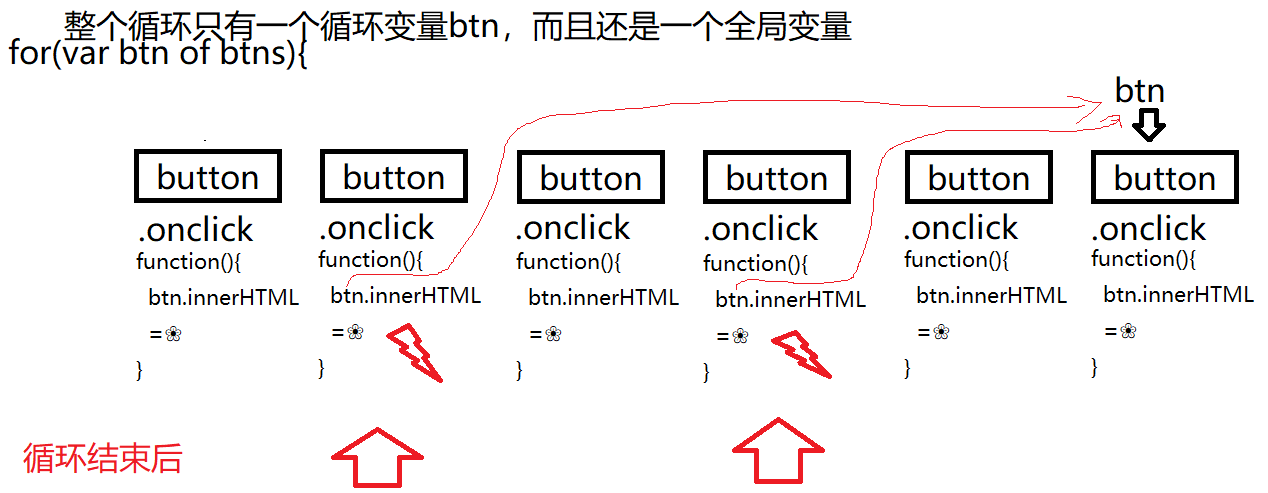
|  |
| --- |
| <!DOCTYPE HTML>  <html>  <head>  <title>使用Selector API实现购物车客户端计算</title>  <meta charset="utf-8" />  <style>    table{width:600px; text-align:center;      border-collapse:collapse;    }    td,th{border:1px solid black}    td[colspan="3"]{text-align:right;}  </style>  </head>  <body>    <table id="data">      <thead>        <tr>          <th>商品名称</th>          <th>单价</th>          <th>数量</th>          <th>小计</th>        </tr>      </thead>      <tbody>        <tr>          <td>iPhone6</td>          <td>¥4488.00</td>          <td>            <button>-</button>            <span>1</span>            <button>+</button>          </td>          <td>¥4488.00</td>        </tr>        <tr>          <td>iPhone6 plus</td>          <td>¥5288.00</td>          <td>            <button>-</button>            <span>1</span>            <button>+</button>          </td>          <td>¥5288.00</td>        </tr>        <tr>          <td>iPad Air 2</td>          <td>¥4288.00</td>          <td>            <button>-</button>            <span>1</span>            <button>+</button>          </td>          <td>¥4288.00</td>        </tr>      </tbody>      <tfoot>        <tr>          <td colspan="3">Total: </td>          <td>¥14064.00</td>        </tr>      </tfoot>    </table>    <script>      //说人话      //实现第一个功能: 点按钮改数量      //DOM 4步      //1. 查找触发事件的元素      //本例中:将来用户点table中每个button元素都会触发变化？      //所以，应该查找table下所有button元素，2步:      //1.1 先查找table元素: 查找id为data的table元素      var table=document.getElementById("data");      console.log(table);      //1.2 再在table下查找所有的button元素      var btns=table.getElementsByTagName("button");      console.log(btns);      //2. 绑定事件处理函数      //实验: 只给第一个按钮绑定单击事件，当单击第一个按钮时，它可以喊疼！      var btn1=btns[0];      //想看元素的属性，不能用console.log()输出      //console.log(btn1);      //应该用console.dir()输出，dir专门输出内存中的对象存储结构      console.dir(btn1);      //为btn1的onclick事件属性提前赋值一个函数，会喊疼      btn1.onclick=function(){ alert("疼！") }        //3. 查找要修改的元素        //4. 修改元素    </script>  </body>  </html>  运行结果: 当单击第一个按钮时 |

7. 示例: 为所有找到按钮都绑定单击事件处理函数:

3\_shoppingcart.html

|  |
| --- |
| <!DOCTYPE HTML>  <html>  <head>  <title>使用Selector API实现购物车客户端计算</title>  <meta charset="utf-8" />  <style>    table{width:600px; text-align:center;      border-collapse:collapse;    }    td,th{border:1px solid black}    td[colspan="3"]{text-align:right;}  </style>  </head>  <body>    <table id="data">      <thead>        <tr>          <th>商品名称</th>          <th>单价</th>          <th>数量</th>          <th>小计</th>        </tr>      </thead>      <tbody>        <tr>          <td>iPhone6</td>          <td>¥4488.00</td>          <td>            <button>-</button>            <span>1</span>            <button>+</button>          </td>          <td>¥4488.00</td>        </tr>        <tr>          <td>iPhone6 plus</td>          <td>¥5288.00</td>          <td>            <button>-</button>            <span>1</span>            <button>+</button>          </td>          <td>¥5288.00</td>        </tr>        <tr>          <td>iPad Air 2</td>          <td>¥4288.00</td>          <td>            <button>-</button>            <span>1</span>            <button>+</button>          </td>          <td>¥4288.00</td>        </tr>      </tbody>      <tfoot>        <tr>          <td colspan="3">Total: </td>          <td>¥14064.00</td>        </tr>      </tfoot>    </table>    <script>      //说人话      //实现第一个功能: 点按钮改数量      //DOM 4步      //1. 查找触发事件的元素      //本例中:将来用户点table中每个button元素都会触发变化？      //所以，应该查找table下所有button元素，2步:      //1.1 先查找table元素: 查找id为data的table元素      var table=document.getElementById("data");      console.log(table);      //1.2 再在table下查找所有的button元素      var btns=table.getElementsByTagName("button");      console.log(btns);      //2. 绑定事件处理函数      //实验: 只给第一个按钮绑定单击事件，当单击第一个按钮时，它可以喊疼！      //var btn1=btns[0];      //想看元素的属性，不能用console.log()输出      //console.log(btn1);      //应该用console.dir()输出，dir专门输出内存中的对象存储结构      //console.dir(btn1);      //为btn1的onclick事件属性提前赋值一个函数，会喊疼      //btn1.onclick=function(){ alert("疼！") }        //本例中: 所有button都可以点击。所以，要为每个button的onclick提前赋值一个事件处理函数！      //遍历btns类数组对象中找到的所有按钮对象      for(var btn of btns){        //每遍历一个按钮对象，就为这个按钮对象提前保存一个事件处理函数        btn.onclick=function(){          alert("疼！");        }      }          //3. 查找要修改的元素        //4. 修改元素    </script>  </body>  </html>  运行结果: |

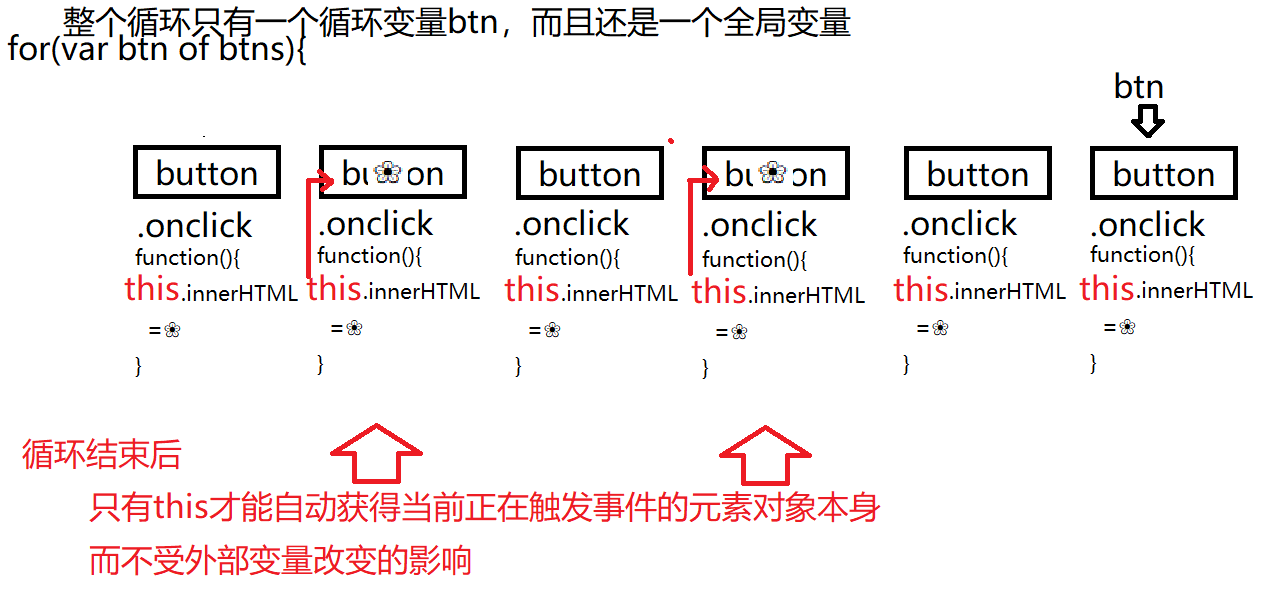
8. 问题: 通过遍历为每个按钮都绑定一个单击事件处理函数，在事件处理函数中如果用循环变量代表“当前按钮”，执行后续操作，其实操作的都是最后一个按钮！



原因: 整个循环中只有一个循环变量btn，该循环变量在循环结束后，会停在最后一个按钮元素上。所以，当用户点击按钮，执行事件处理函数时，永远找唯一的btn变量，而这个变量永远指最后一个按钮！

9. 解决: 在事件处理函数中，如何获得当前正在触发事件的元素，而不受外部变量的影响！

**事件处理函数中的this->当前正在触发事件的元素**



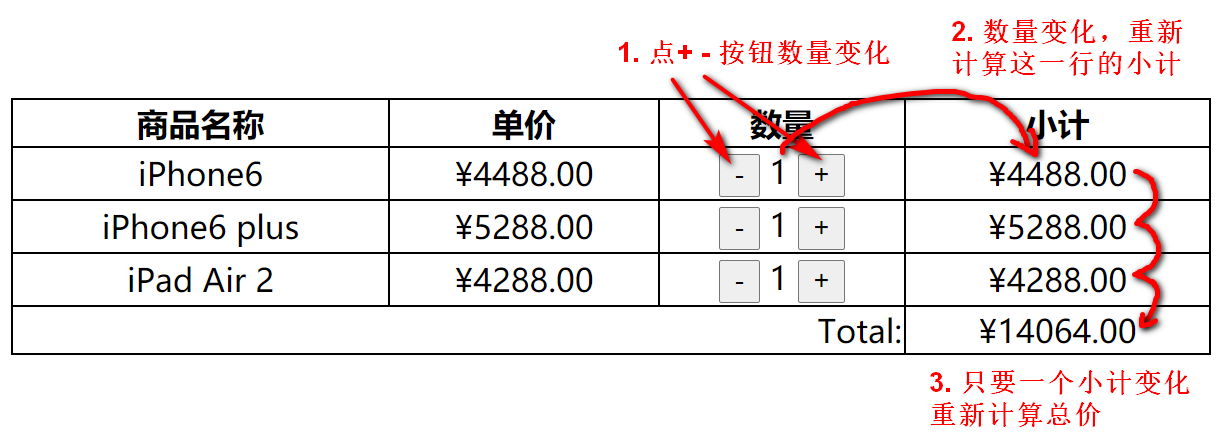
10. 示例: 点那个按钮，让那个按钮变成小花:

3\_shoppingcart.html

|  |
| --- |
| <!DOCTYPE HTML>  <html>  <head>  <title>使用Selector API实现购物车客户端计算</title>  <meta charset="utf-8" />  <style>    table{width:600px; text-align:center;      border-collapse:collapse;    }    td,th{border:1px solid black}    td[colspan="3"]{text-align:right;}  </style>  </head>  <body>    <table id="data">      <thead>        <tr>          <th>商品名称</th>          <th>单价</th>          <th>数量</th>          <th>小计</th>        </tr>      </thead>      <tbody>        <tr>          <td>iPhone6</td>          <td>¥4488.00</td>          <td>            <button>-</button>            <span>1</span>            <button>+</button>          </td>          <td>¥4488.00</td>        </tr>        <tr>          <td>iPhone6 plus</td>          <td>¥5288.00</td>          <td>            <button>-</button>            <span>1</span>            <button>+</button>          </td>          <td>¥5288.00</td>        </tr>        <tr>          <td>iPad Air 2</td>          <td>¥4288.00</td>          <td>            <button>-</button>            <span>1</span>            <button>+</button>          </td>          <td>¥4288.00</td>        </tr>      </tbody>      <tfoot>        <tr>          <td colspan="3">Total: </td>          <td>¥14064.00</td>        </tr>      </tfoot>    </table>    <script>      //说人话      //实现第一个功能: 点按钮改数量      //DOM 4步      //1. 查找触发事件的元素      //本例中:将来用户点table中每个button元素都会触发变化？      //所以，应该查找table下所有button元素，2步:      //1.1 先查找table元素: 查找id为data的table元素      var table=document.getElementById("data");      console.log(table);      //1.2 再在table下查找所有的button元素      var btns=table.getElementsByTagName("button");      console.log(btns);      //2. 绑定事件处理函数      //实验: 只给第一个按钮绑定单击事件，当单击第一个按钮时，它可以喊疼！      //var btn1=btns[0];      //想看元素的属性，不能用console.log()输出      //console.log(btn1);      //应该用console.dir()输出，dir专门输出内存中的对象存储结构      //console.dir(btn1);      //为btn1的onclick事件属性提前赋值一个函数，会喊疼      //btn1.onclick=function(){ alert("疼！") }        //本例中: 所有button都可以点击。所以，要为每个button的onclick提前赋值一个事件处理函数！      //遍历btns类数组对象中找到的所有按钮对象      for(var btn of btns){        //每遍历一个按钮对象，就为这个按钮对象提前保存一个事件处理函数        btn.onclick=function(){          //alert("疼！");          //新需求: 点那个按钮，让那个按钮的内容变成❀          //错误的解决: btn是唯一的全局变量，也是循环变量。在触发事件前，btn就移动到最后一个按钮上了          //btn.innerHTML="❀";              //内容          //正确的解决:          console.log(this);          this.innerHTML="❀";        }        //结果：        //btns[0].onclick=function(){alert("疼")}        //btns[1].onclick=function(){alert("疼")}        //btns[2].onclick=function(){alert("疼")}        //btns[3].onclick=function(){alert("疼")}        //btns[4].onclick=function(){alert("疼")}        //btns[5].onclick=function(){alert("疼")}      }          //3. 查找要修改的元素        //4. 修改元素    </script>  </body>  </html>  运行结果: |

五. 示例: 购物车: 3大功能:

1. 需求:



2. 需求一: 先实现点击按钮，让按钮旁边的span数量+1或-1

(1). 需求:

a. 每行的按钮都能点击

b. 如果点击+，则找到+前边的span，将其内容中的数字+1

c. 如果点击-，则找到-后边的span，将其内容中的数字-1，但是，购物车中商品数量不能<1.

(2). 今后几乎所有DOM操作的标准4步:

a. 先**查找**可能触发事件的元素

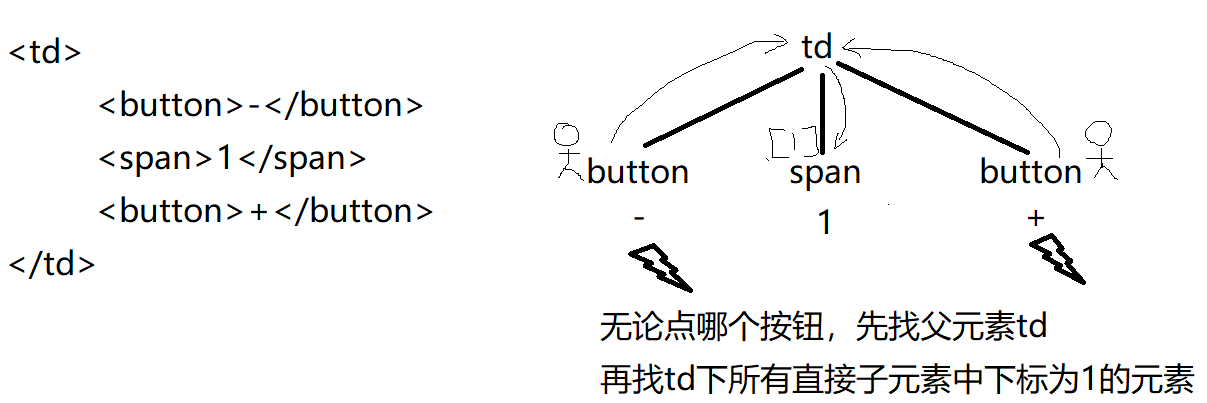
本例中: 查找table下所有button元素

b. 再为元素**绑定**事件处理函数

本例中: 为每个button绑定onclick单击事件处理函数

c. 当事件发生时，**查找**要修改的元素

本例中: 先查找当前按钮的父元素td，然后再找td下所有直接子元素中下标为1的span元素。



d. **修改**元素

本例中: 先获得span元素的内容，转为整数后，+1或-1，再放回span的内容中

(3). 示例: 实现单击按钮，修改数量:

3\_shoppingcart2.html

|  |
| --- |
| <!DOCTYPE HTML>  <html>  <head>  <title>使用Selector API实现购物车客户端计算</title>  <meta charset="utf-8" />  <style>    table{width:600px; text-align:center;      border-collapse:collapse;    }    td,th{border:1px solid black}    td[colspan="3"]{text-align:right;}  </style>  </head>  <body>    <table id="data">      <thead>        <tr>          <th>商品名称</th>          <th>单价</th>          <th>数量</th>          <th>小计</th>        </tr>      </thead>      <tbody>        <tr>          <td>iPhone6</td>          <td>¥4488.00</td>          <td>            <button>-</button>            <span>1</span>            <button>+</button>          </td>          <td>¥4488.00</td>        </tr>        <tr>          <td>iPhone6 plus</td>          <td>¥5288.00</td>          <td>            <button>-</button>            <span>1</span>            <button>+</button>          </td>          <td>¥5288.00</td>        </tr>        <tr>          <td>iPad Air 2</td>          <td>¥4288.00</td>          <td>            <button>-</button>            <span>1</span>            <button>+</button>          </td>          <td>¥4288.00</td>        </tr>      </tbody>      <tfoot>        <tr>          <td colspan="3">Total: </td>          <td>¥14064.00</td>        </tr>      </tfoot>    </table>    <script>      //说人话      //实现第一个功能: 点按钮改数量      //DOM 4步      //1. 查找触发事件的元素      //本例中:应该查找table下所有button元素，2步:      //1.1 先查找table元素: 查找id为data的table元素      var table=document.getElementById("data");      //1.2 再在table下查找所有的button元素      var btns=table.getElementsByTagName("button");      //2. 绑定事件处理函数      //本例中: 要为每个button的onclick提前赋值一个事件处理函数！      //遍历btns类数组对象中找到的所有按钮对象      for(var btn of btns){        //每遍历一个按钮对象，就为这个按钮对象提前保存一个事件处理函数        btn.onclick=function(){          //this->才可获得当前正在单击的按钮按钮对象          //不要用外部变量btn，因为btn会变！不靠谱！          //3. 查找要修改的元素          //本例中: 查找当前按钮的爹的所有直接子元素中下标为1的span元素          var span=this.parentNode.children[1];          //测试: 让当前span的背景色变为红色          //span.style.backgroundColor="red";          //4. 修改元素          //span:<span>1</span>          //强调：一定不能直接对元素对象+1或-1，而应该对元素的内容+1或-1          //本例中: 先获得span元素的内容，+1或-1后，再放回span中          //4.1 先获得span的内容，保存在变量n中          //坑: 凡是从页面元素上获得的一切，都是字符串类型！          //解决: 今后凡是从页面上获得的一切，在进行算数计算前，必须都要先转为number类型的数字。          var n=parseInt(span.innerHTML);          //4.2 如果当前按钮的内容是+，就n+1，否则如果当前按钮的内容是-，且n的值>1时，才允许n-1          if(this.innerHTML=="+"){            n++;          }else if(n>1){//如果程序可以执行到这里，暗含着innerHTML肯定等于"-"，因为除了+和-，没有其它情况            n--;          }          //坑: 从页面上获取内容或值，拿到的都是副本！在程序中修改变量，原值或内容，不改变的！          //解决:          //4.3 必须将修改后的n变量的值，重新赋值回页面上span元素对象的内容中，页面上才能看到变化！          span.innerHTML=n;        }        }    </script>  </body>  </html>  运行结果: |

3. 需求2: 数量变化，重新计算本行的小计

(1). DOM 4步:

a. 查找触发事件的元素: 需求一已完成，和需求一共用同一个单击事件即可

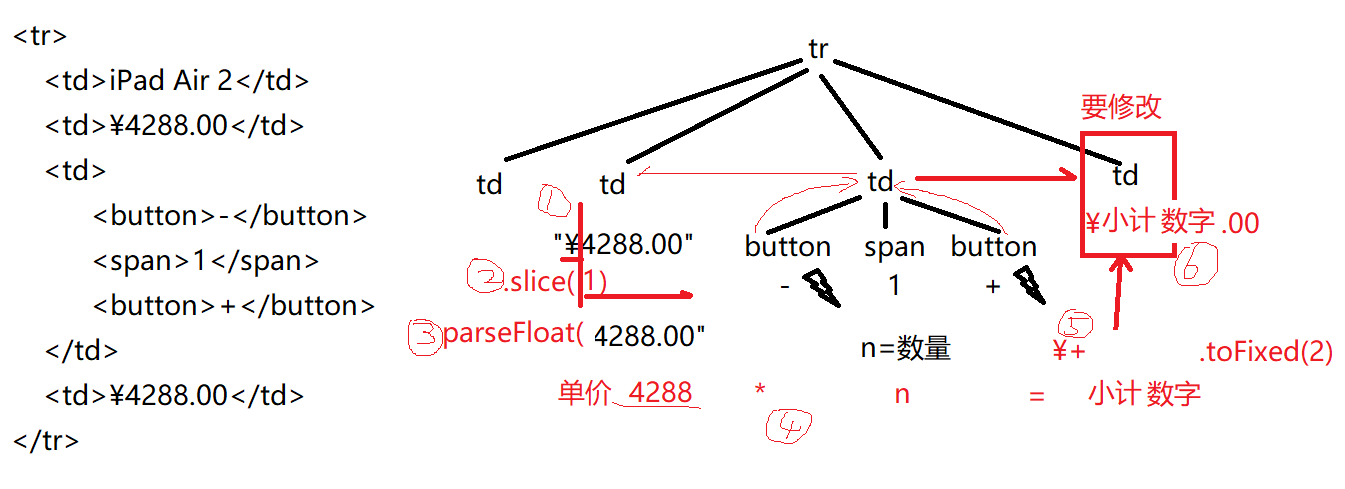
b. 绑定事件处理函数: 需求一已完成，和需求一共用同一个单击事件即可

c. 查找要修改的元素:

本例中: 查找当前按钮的父元素td的下一个兄弟元素td

d. 修改元素:

本例中: 获得当前按钮的父元素td的前一个兄弟的内容，去掉¥，转为浮点数，\*数量n。算出的小计，再拼上¥，并按两位小数四舍五入。最后放入小计td中.



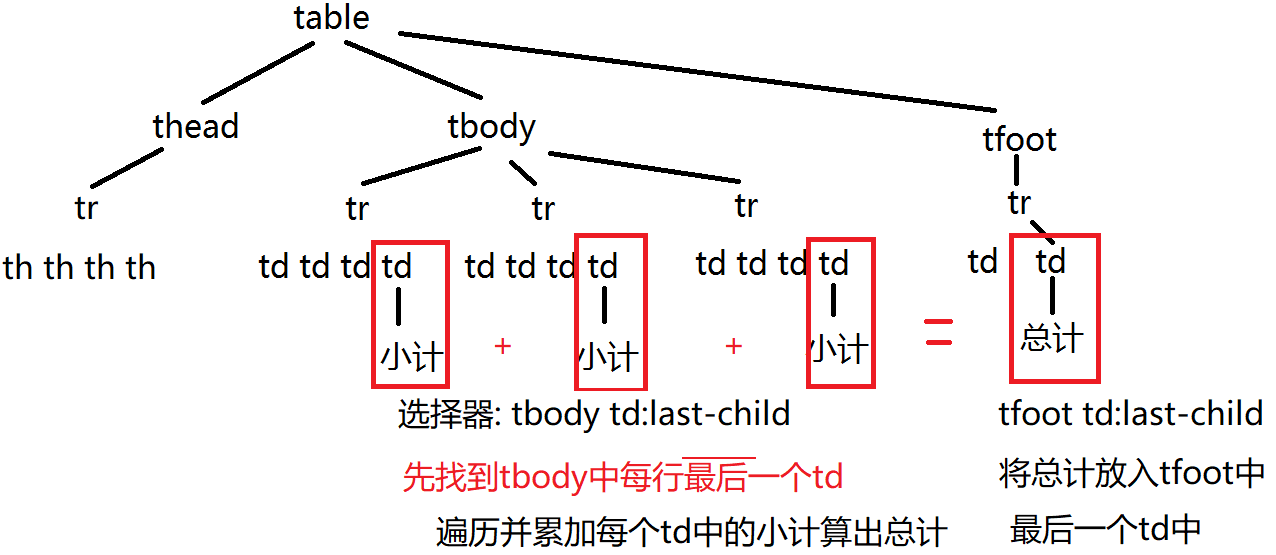
(2). 示例: 数量变化，修改本行小计

3\_shoppingcart.html

|  |
| --- |
|  |

4. 需求3: 小计变化，重新计算总计

(1). 需求:



(2). 示例: 实现小计变化，自动修改总计:

3\_shoppingcart2.html

|  |
| --- |
| <!DOCTYPE HTML>  <html>  <head>  <title>使用Selector API实现购物车客户端计算</title>  <meta charset="utf-8" />  <style>    table{width:600px; text-align:center;      border-collapse:collapse;    }    td,th{border:1px solid black}    td[colspan="3"]{text-align:right;}    /\*想让tbody中每行最后一个td背景变为粉色\*/    tbody td:last-child{      background-color:pink;    }    /\*想让tfoot中最后一个td背景变为黄色\*/    /\* tfoot td:last-child{      background-color:yellow;    } \*/  </style>  </head>  <body>    <table id="data">      <thead>        <tr>          <th>商品名称</th>          <th>单价</th>          <th>数量</th>          <th>小计</th>        </tr>      </thead>      <tbody>        <tr>          <td>iPhone6</td>          <td>¥4488.00</td>          <td>            <button>-</button>            <span>1</span>            <button>+</button>          </td>          <td>¥4488.00</td>        </tr>        <tr>          <td>iPhone6 plus</td>          <td>¥5288.00</td>          <td>            <button>-</button>            <span>1</span>            <button>+</button>          </td>          <td>¥5288.00</td>        </tr>        <tr>          <td>iPad Air 2</td>          <td>¥4288.00</td>          <td>            <button>-</button>            <span>1</span>            <button>+</button>          </td>          <td>¥4288.00</td>        </tr>      </tbody>      <tfoot>        <tr>          <td colspan="3">Total: </td>          <td>¥14064.00</td>        </tr>      </tfoot>    </table>    <script>      //说人话      //实现第一个功能: 点按钮改数量      //DOM 4步      //1. 查找触发事件的元素      //本例中:应该查找table下所有button元素，2步:      //1.1 先查找table元素: 查找id为data的table元素      var table=document.getElementById("data");      //1.2 再在table下查找所有的button元素      var btns=table.getElementsByTagName("button");      //2. 绑定事件处理函数      //本例中: 要为每个button的onclick提前赋值一个事件处理函数！      //遍历btns类数组对象中找到的所有按钮对象      for(var btn of btns){        //每遍历一个按钮对象，就为这个按钮对象提前保存一个事件处理函数        btn.onclick=function(){          //this->才可获得当前正在单击的按钮按钮对象          //不要用外部变量btn，因为btn会变！不靠谱！          //3. 查找要修改的元素          //本例中: 查找当前按钮的爹的所有直接子元素中下标为1的span元素          var span=this.parentNode.children[1];          //测试: 让当前span的背景色变为红色          //span.style.backgroundColor="red";          //4. 修改元素          //span:<span>1</span>          //强调：一定不能直接对元素对象+1或-1，而应该对元素的内容+1或-1          //本例中: 先获得span元素的内容，+1或-1后，再放回span中          //4.1 先获得span的内容，保存在变量n中          //坑: 凡是从页面元素上获得的一切，都是字符串类型！          //解决: 今后凡是从页面上获得的一切，在进行算数计算前，必须都要先转为number类型的数字。          var n=parseInt(span.innerHTML);          //4.2 如果当前按钮的内容是+，就n+1，否则如果当前按钮的内容是-，且n的值>1时，才允许n-1          if(this.innerHTML=="+"){            n++;          }else if(n>1){//如果程序可以执行到这里，暗含着innerHTML肯定等于"-"，因为除了+和-，没有其它情况            n--;          }          //坑: 从页面上获取内容或值，拿到的都是副本！在程序中修改变量，原值或内容，不改变的！          //解决:          //4.3 必须将修改后的n变量的值，重新赋值回页面上span元素对象的内容中，页面上才能看到变化！          span.innerHTML=n;          //实现需求二: 数量修改，自动修改本行的小计          //3. 查找要修改的元素:          //本例中: 当前按钮的父元素td的下一个兄弟元素td          var tdSub=this.parentNode.nextElementSibling;          //4. 修改元素:          //本例中: 获得单价，\*数量n。算出的小计,最后放入小计td中.          //4.1 获得单价: 获得当前按钮的父元素td的前一个兄弟td的内容，去掉¥，转为浮点数          var price=parseFloat(            this.parentNode //按钮所在td                .previousElementSibling //按钮所在td的前一个兄弟td                .innerHTML //前一个兄弟td的内容: "¥单价.00"                .slice(1) //"单价.00"          )//单价          //4.2 单价\*数量=小计          var sub=price\*n;          //4.3 将小计拼接¥，按两位小数四舍五入，再放回小计td的内容中          tdSub.innerHTML=`¥${sub.toFixed(2)}`;          //实现需求三: 总计          //3. 查找要修改的元素:          //本例中: 修改tfoot中最后一个td          //因为下边的选择器肯定只能找到一个td，所以不加all          var lastTd=table.querySelector("tfoot td:last-child");          //测试:          //lastTd.style.backgroundColor="yellow";          //4. 修改元素:          //本例中: 获得tbody中每行最后一个td中的小计，累加后算出总计，放入tfoot中最后一个td中          //4.1 获得tbody中每行最后一个td          var tds=table.querySelectorAll("tbody td:last-child");          //先定义一个变量total=0，用于累加每行的小计          var total=0;          //4.2 遍历找到的每个td          for(var td of tds){            //每遍历一个td，就要获取td的内容，去掉¥，转为浮点数，累加到变量total上            total+=parseFloat(td.innerHTML.slice(1))          }          //4.3 将算出总价拼上¥符号，按2位小数四舍五入，最后放入tfoot中最后一个td中          lastTd.innerHTML=`¥${total.toFixed(2)}`;        }      }    </script>  </body>  </html>  运行结果: |

|  |
| --- |
| 总结: 查找方法的返回值规律: 3种  1. 如果原方法返回下标位置i，则如果找不到返回-1  2. 如果原方法返回一个数组或对象，则如果找不到返回null  3. 如果原方法返回类数组对象，则如果找不到返回空类数组对象: { length:0 } |

总结: DOM 4步:

1. 先查找可能触发事件的元素

2. 再为元素绑定事件处理函数

3. 当事件发生时，查找要修改的元素

4. 修改元素

总结: DOM 5件事: 增删改查+事件绑定:

1. 查找元素:

(1). 不需要查找就可直接获得的节点对象: 4种:

document

document.documentElement —— <html>

document.head —— <head>

document.body —— <body>

(2). 如果已经获得一个元素对象，找周围附近的元素对象时，就用按节点间关系查找：2大类关系，6个属性

父子关系: 4种

元素的父元素: 元素.parentNode或元素.parentElement

元素下的所有直接子元素: 元素.children

元素下的第一个直接子元素: 元素.firstElementChild

元素下的最后一个直接子元素: 元素.lastElementChild

兄弟关系: 2种

元素的前一个兄弟: 元素.previousElementSibling

元素的后一个兄弟: 元素.nextElementSibling

(3). 如果用一个特征就能找到想要的元素，就用按HTML特征查找：4个方法:

按id查找: var 一个元素对象=document.getElementById("id名");

按标签名查找: var 类数组对象=任意父元素.getElementsByTagName("标签名")、

按class名查找: var 类数组对象=任意父元素.getElementsByClassName("class名")

按name名查找表单元素: var 类数组对象=document.getElementsByName("name名")

(4). 如果通过复杂的查找条件，才能找到想要的元素时，就用按选择器查找: 2个方法

只查找一个符合条件的元素: var 一个元素=任意父元素.querySelector("任意选择器")

查找多个符合条件的元素: var 类数组对象=任意父元素.querySelectorAll("任意选择器")

2. 修改元素: 3种东西可修改

(1). 修改内容: 3种内容可修改:

获取或修改元素的HTML内容: 元素.innerHTML

(2). 修改属性: (待续...)

(3). 修改样式:

修改元素的内联样式: 元素.style.css属性="属性值"

3. 添加/删除元素:(待续...)

4. 事件绑定: 3种:

(1). 在HTML中绑定: DOM和jquery中几乎不用

HTML中: <元素 on事件名="事件处理函数()">

JS中: function 事件处理函数(){ ... }

(2). 使用赋值方式绑定: 元素对象.on事件名=function(){ ... }

(3). (待续...)

|  |
| --- |
| 总结: this 5种: 判断this，~~一定不要看定义在哪儿！~~只看调用时！  1. obj.fun() this->obj  2. fun() 或 (function(){ ... })() 或 多数回调函数 this->window  3. new Fun() this->new正在创建的新对象  4. 类型名.prototype.共有方法=function(){ ... } this->将来谁调用指谁，同第一种情况  5. 事件处理函数中的this->当前正在触发事件的DOM元素对象 |

今日对应小程序视频列表:

小程序->在线->DOM->day01 0 DOM Tree 树

1. 按节点间关系查找

2. 按HTML特征查找

3. 购物车 shoppingcart

作业:

1. 复习今日问题清单: 小程序->首页->DOM->day01

2. 看小程序视频完成购物车效果:

小程序->在线->DOM->day01 3.购物车 shoppingcart

3. 看小程序视频，预习修改元素:

小程序->在线->DOM->day01 0.1 修改元素的内容

0.2 修改元素的属性 字符串类型

0.3 修改元素的属性 bool类型

4.1 自定义扩展属性

5. 修改样式