提前准备:

1. 今天的历届师兄师姐问过的经典的高频的问题清单都已经整理在小程序->首页->DOM->day01里

2. 今天的课程中所需的单词列表都已经整理在小程序->单词->DOM->day01里

3. 今天的重点视频、扩展高频鄙视题视频已经整理在小程序->在线->DOM->day01里

如果不知道问什么: 谁问的越多！谁培训费花的越值！

1. 笔记中或画图中那句话或者那个位置不明白，立刻问！

2. 案例中哪行代码或注释不明白，立刻问！

3. 课下自己做项目，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！——公众号: 前端大全 定期总结大厂笔试题

5. 看小程序问题清单时，遇到看不懂的问题，立刻问！

今天开始！每个案例三遍:

1. 连注释+代码抄一遍

2. 保留注释，删掉代码，自己试着填回来

3. 删除注释和代码，自己使用用人话把注释写回来！——才算自己会！

正课:

1. 什么是DOM

2. DOM树

3. 查找元素

一. 什么是DOM: Document Object Model

文档 对象 模型

1. 什么是DOM: 一套专门操作网页内容的对象和函数的统称

2. 为什么: ECMAScript标准中，仅规定了js语言核心语法(如何写正确，如何写是错误的)。但是没有规定如何操作网页的内容！

3. 何时: 今后只要想操作网页内容，必须用DOM

4. 什么是DOM标准: 由W3C制定并负责维护一套统一的DOM对象和函数标准！

5. 为什么要有DOM标准: 因为早期的各家浏览器，做同样的事情，DOM的对象和函数各不相同，导致程序员在一种浏览器上实现的效果，换到另一个浏览器就不能用！

6. 1998年W3C统一了DOM标准，规定了统一的对象和函数名，结果，只要使用DOM操作网页内容，几乎所有浏览器100%兼容。

7. 如何: 增删改查+事件绑定

二. DOM树:

1. 什么是DOM树: 其实网页中所有的内容(元素，文本，属性...)，被浏览器加载到内存后，都是保存在一棵树型结构上！

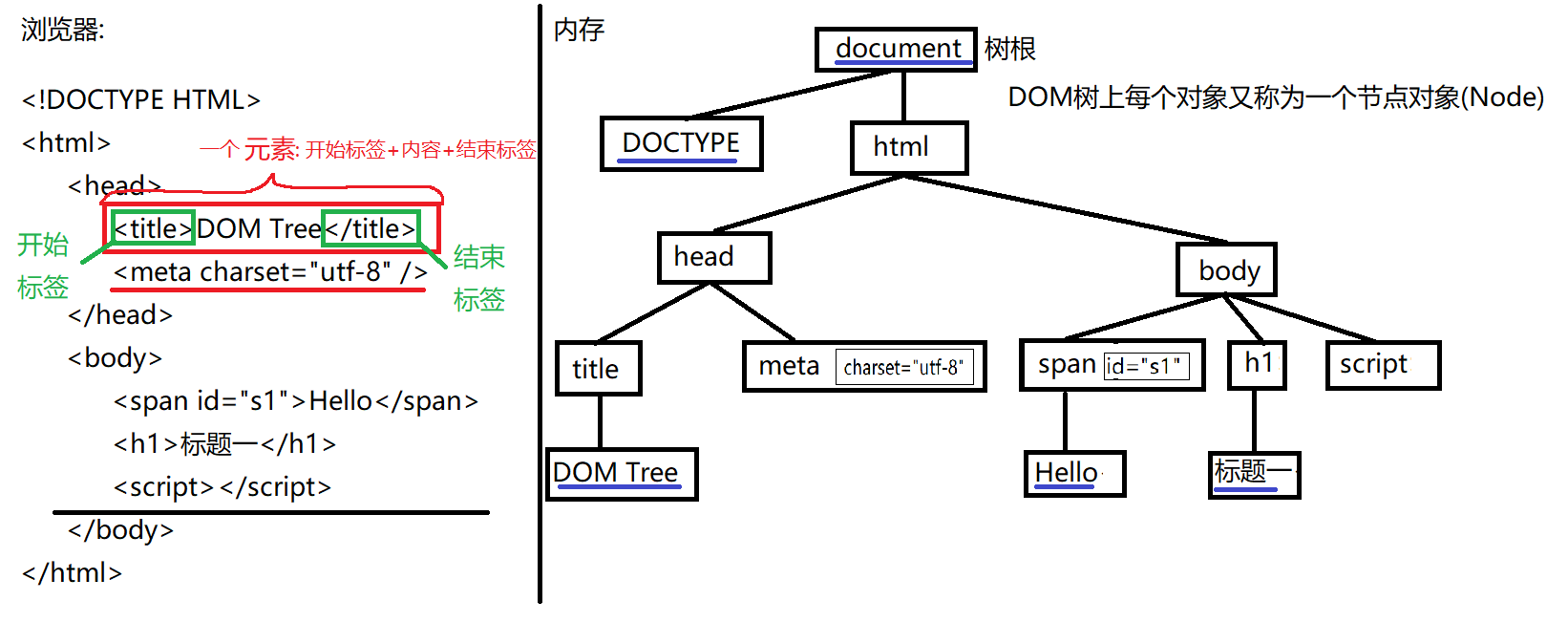
2. 为什么: 因为网页的内容都是上下级包含关系。而树形结构是最直观的展现上下级包含关系的结果。

3. 何时: 今后只要查找元素，都会依赖元素在DOM树中的位置，以及不同元素在DOM树上的关系！

4. DOM树如何形成的:

(1). 当浏览器读取到一个网页时，先在内存中创建一个document对象，作为整个树的树根对象

(2). 然后浏览器开始扫描网页中的每一项内容(元素,属性,文本,...)。每扫描到一项内容，浏览器就会自动将这项内容创建为一个对象，来保存这项内容的信息——节点对象



(3). 每个节点对象在DOM树中都不是孤立的，每个节点对象都和它上下左右的其他节点对象有着各种各样的关系！

4. 其实可以查看到DOM树: f12->Elements

三. 查找元素: 4种

1. 不需要查找，就可直接获得的节点对象:

(1). document 根节点

(2). <html> document.documentElement

(3). <head> document.head

(3). <body> document.body

2. 按节点间关系查找:

(1). 如果我们已经获得一个节点对象，只想找它周围附近的节点对象时，就可按节点间关系查找

(2). ~~节点树: 包含所有网页内容(元素, 属性, 文本, ...)的完整树结构:~~

2大类关系，6个属性

a. 父子关系: 4个属性:

1) 获得节点的父级节点对象: 节点对象.parentNode

父 节点

2) 获得一个父节点下的所有直接子节点: 父节点对象.childNodes

子 节点 们

强调: 1. 只获取直接子节点, 2. 返回一个类数组对象包含该父节点下所有直接子节点

3) 获得一个父节点下的第一个直接子节点: 父节点对象.firstChild

4) 获得一个父节点下的最后一个直接子节点: 父节点对象.lastChild

b. 兄弟关系: 2个属性

1) 获得一个节点对象的平级的前一个兄弟: 节点对象.previousSibling

前一个 兄弟

2) 获得一个节点对象的平级的后一个兄弟: 节点对象.nextSibling

下一个兄弟

(3). 节点树有问题: 查找过程中会受看不见的空字符节点对象的干扰！

解决: 较新版本的DOM标准中，推出了一种新树：元素树

(4). 元素树: 仅包含程序员关心的元素节点的树结构，也是2大类关系，6个属性

其实内存中只有一棵完整的节点树。而元素树只不过是在节点树的基础上，新增了一些仅指向元素节点的新属性而已。所以，元素树只是节点树中的一个子集。

a. 父子关系: 4个属性:

1) 获得元素的父级元素对象: 元素对象.parentElement

父 元素

2) 获得一个父元素下的所有直接子元素: 父元素对象.children

孩子们

强调: 1.只获取直接子元素, 2.返回一个类数组对象包含该父元素下所有直接子元素

3) 获得一个父元素下的第一个直接子元素: 父元素对象.firstElementChild

第一个 元素 孩子

4) 获得一个父元素下的最后一个直接子元素: 父元素对象.lastElementChild

最后一个 元素 孩子

b. 兄弟关系: 2个属性

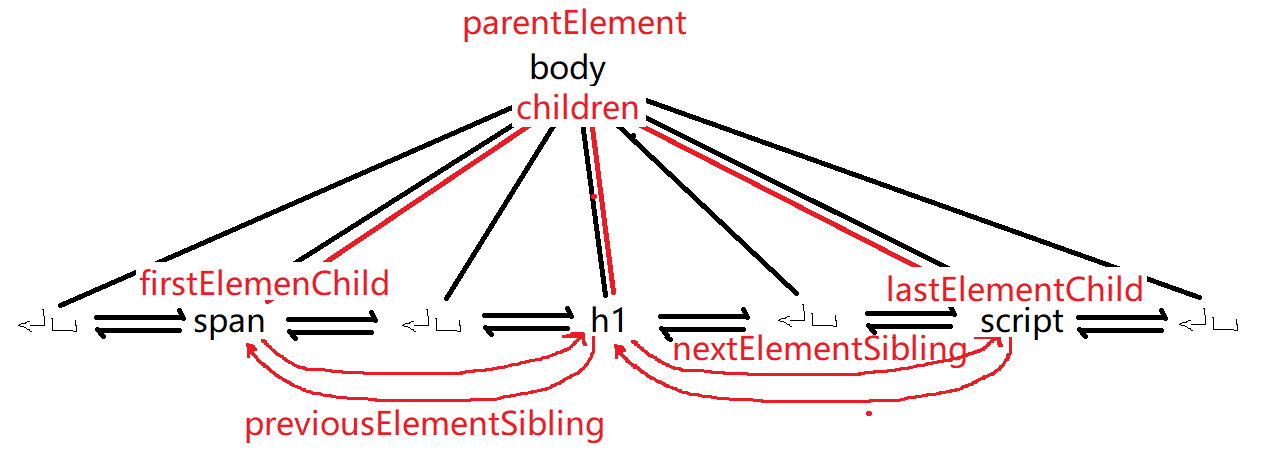
1) 获得一个元素对象的平级的前一个兄弟元素: 元素对象.previousElementSibling

前一个 元素 兄弟

2) 获得一个元素对象的平级的后一个兄弟元素: 元素对象.nextElementSibling

下一个 元素 兄弟

(5). 今后只要按节点间关系查找，都要用元素树的属性代替节点树的属性



(6). 示例: 使用节点间关系查找页面上某个元素

|  |
| --- |
| <body>    <span id="s1">Hello</span>    <h1>标题一</h1>    <script>    // console.log(document);    // console.log(document.documentElement);    // console.log(document.head);    // console.log(document.body);    //先获得body    var body=document.body;    console.log(body);    //再获得body的所有直接子元素    var children=body.children;    console.log(children);//3个    //想获得body下第一个直接子元素    var span=body.firstElementChild;    console.log(span);    //想获得body下最后一个直接子元素:    var script=body.lastElementChild;    console.log(script);    //想获得body下的h1元素: 3种    var h1=children[1];//body下第二个孩子        //span.nextElementSibling//span的下一个兄弟        //script.previousElementSibling//script的前一个兄弟    console.log(h1);    //只有这个例子不要用live server运行，因为live server会悄悄往网页中添加新内容。    //暂时只能去我的电脑文件夹里双击这个.html文件运行    </script>  </body> |

3. 按元素身上的HTML特征查找元素: 4种

(1). 按id查找一个元素

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

在网页中 获取 元素 通过id名

b. 意为: 通过id名查找网页中一个元素对象

c. 强调:

1). 函数只返回一个元素对象

如果不小心两个HTML元素id名相同，则getElementById只能返回第一个找到的元素

2). 必须用document作为.前的主语调用，意为在整个网页范围内查找.

3). 函数名中的Element是单数，没有s结尾

d. 如果找不到返回: null

e. 按id查找的问题:

1). 按id查找一次只能找一个元素对象，无法同时找多个元素对象

2). 网页中标有id的元素并不多，因为没有那么多英文单词名字可用！

(2). 按元素的标签名查找多个符合条件的元素：

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

在任意父元素下获取多个元素按标签名

b. 意为: 在指定的一个父元素下，查找多个符合指定标签名的元素对象

c. 强调:

1). 因为很可能找到多个元素，所以返回值是一个类数组对象

2). 可用任意父元素作为.前的主语来调用函数。.前的主语是谁，就只在这个父元素内查找。将来查找范围越小，查找效率越高！

3). 因为可能返回多个元素，所以函数名中是Elements复数s结尾

4). getElementsByTagName()不仅查找直接子元素，而是在所有后代中查找符合条件的元素

d. 如果找不到返回: length为0的空类数组对象: { length:0 }

(3). 按class名查找多个元素:

a. 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按class名

b. 意为: 在指定的一个父元素下，查找多个带有指定class名的元素对象

c. 强调:

1). 因为很可能找到多个元素，所以返回值是一个类数组对象

2). 可用任意父元素作为.前的主语来调用函数。.前的主语是谁，就只在这个父元素内查找。将来查找范围越小，查找效率越高！

3). 因为可能返回多个元素，所以函数名中是Elements复数s结尾

4). getElementsByClassName()不仅查找直接子元素，而是在所有后代中查找符合条件的元素

5). 如果一个元素同时被多个class名修饰，则只需要其中任意一个class名，就能找到该元素。

d. 如果找不到，返回: 空类数组对象{ length:0 }

(4). 按name名查找表单元素:

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

在当前网页中查找多个元素按name名

b. 意为: 在当前网页中，按name名查找多个符合条件的表单元素

c. 强调:

1). 因为很可能找到多个元素，所以返回值是一个类数组对象

2). 只能用document对象作为.前的主语！

3). 因为可能返回多个元素，所以函数名中是Elements复数s结尾

d. 如果找不到，返回: 空类数组对象{ length:0 }

e. 示例: 按name名查找表单元素:

|  |
| --- |
| <form>  用户名: <input type="text" name="uname"><br/>  爱好: <label><input type="radio" name="favs" value="swimming">游泳</label>        <label><input type="radio" name="favs" value="running">跑步</label>        <label><input type="radio" name="favs" value="music">音乐</label>  </form>  <script>  //想查找所有name名为favs的元素  var rads=document.getElementsByName("favs");  console.log(rads);  //想查找name名为uname的一个文本框，必须加[0]  var input=document.getElementsByName("uname")[0];  console.log(input);  </script> |

|  |
| --- |
| 如果返回类数组对象的函数，只找到一个元素时，也会返回一个类数组对象。然后在类数组对象中[0]位置放着这个找到的唯一的元素对象。如果想使用这个找到的惟一的元素对象，必须用[0]才能取出这个元素对象使用。 |

4. 按选择器查找元素:

(1). 问题: 元素藏的很深，查找条件很复杂时，使用按节点间关系或按HTML特征查找都会很繁琐！因为按节点间关系和按HTML特征查找，一次只能按一个条件查找！如果查找条件复杂，就需要很多步骤！

(2). 解决: 今后只要元素的查找条件比较复杂，都用按选择器查找！

(3). 为什么: 无论藏的多深的元素，css的选择器永远只需要一句很简洁的话就可找到元素！

(4). 如何: 2个函数

a. 当只想找一个元素时:

var 一个元素=任意父元素.querySelector("任意复杂的合理的css选择器");

b. 当想找多个符合条件的元素时

var 类数组对象=任意父元素.querySelectorAll("任意复杂的合理的css选择器");

总结: 查找函数的返回值:

1. 如果函数返回下标位置i，如果找不到，返回: -1

2. 如果函数返回一个对象或一个数组，如果找不到，返回: null

3. 如果函数返回一个类数组对象，如果找不到，返回: 空类数组对象{ length:0 }

补: 事件绑定：

1. 什么是事件: 浏览器自己触发的或用户手动触发的页面内容或状态的改变

2. 什么是事件处理函数属性: 每个元素上都有一批on开头的特殊属性。这些属性专门负责为元素提前保存触发对应事件时要自动执行的一个处理函数。

3. 何时使用事件处理函数属性: 今后，只要希望元素在发生一件事时，能自动执行一项任务，都要提前为这个元素的指定的事件处理函数属性赋值一个函数。

4. 如何使用事件处理函数属性: 2种:

~~(1). 在HTML中，元素的开始标签中，添加特殊属性: on事件名="js语句"~~

a. ~~<元素 on事件名="js语句">~~

b. 结果: 当这个元素发生指定的事件后，浏览器会自动找到元素上定义的事件属性，并自动执行属性中保存的js语句.

c. 比如: <button onclick="alert('疼！')">

希望将来当单击按钮时，按钮可以喊疼！

d. 结果: 当单击这个按钮时，浏览器会自动找到这个按钮的onclick属性，并执行其中的alert()语句，就能弹出对话框！

e. 问题:

1). 不符合内容与行为分离的元素，不便于今后代码的维护

2). 不能重用：写在一个按钮中的js表达式，只能被当前这一个按钮使用！

f. 所以: 今后不会再HTML中添加事件处理函数属性

(2). 今后只要为元素添加事件处理函数属性，都必须在js中添加。

a. 如何:2步:

1). 查找要添加事件处理函数的元素对象,

2). 元素对象.on事件名=function(){

... 将来事件发生时，希望自动执行的代码 ...

}

b. 比如: 希望给<button id="btn1">绑定单击事件处理函数

var btn1=document.getElementById("btn1");

btn1.onclick=function(){

... ...

}

c. 结果: 当用户单击btn1按钮时，浏览器会自动找到这个按钮对象上的onclick属性，并自动执行这个属性中提前保存的事件处理函数。

d. 优点: 所有js代码集中在js部分编写，及其便于今后代码维护

e. 所以: 今后只要为元素添加事件处理函数属性，都必须在js中添加

f. 强调: 元素对象上可用的事件处理函数属性，都是DOM标准提前预定好的。所以只能在DOM预定的几种事件处理函数属性中选择使用。而不能完全自定义属性名。

如何查看一个元素对象身上都有哪些可用的事件处理函数属性:

console.dir(元素对象)，展开看所有on开头的属性，都是可选的事件处理函数属性。

5. 在事件处理函数中如何获得当前触发事件的元素本身:

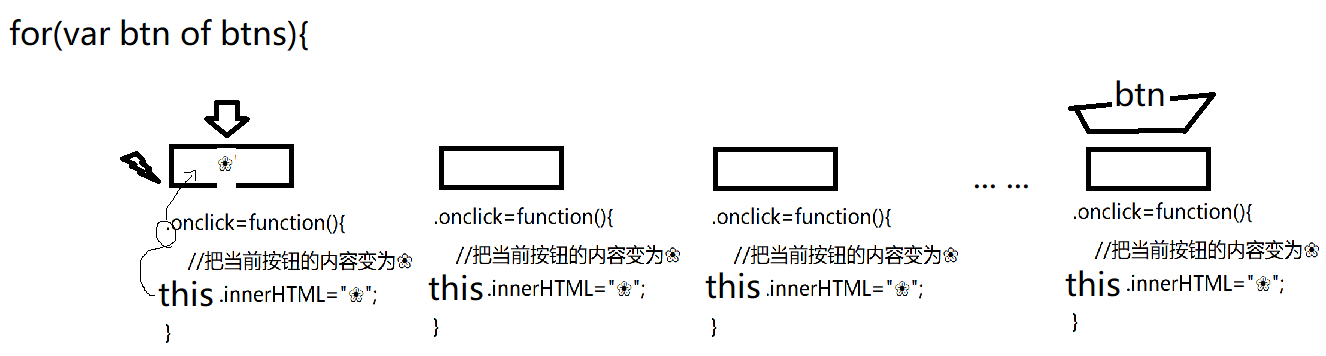
(1). 错误的做法: ~~使用外部的全局变量~~

因为外部的全局变量，很可能在用户触发事件之前就被改变了！早就不是之前所指的对象了！

(2). 正确的做法: this

a. 事件处理函数中的this->当前正在触发事件的这个元素本身！

b. 因为: 当事件发生时，浏览器是找到这个触发事件的元素对象，用这个元素对象本身调用事件处理函数属性: 当前元素.onclick() this->当前正在触发事件的元素！



(3). 所以，今后只要在事件处理函数中希望获得当前元素本身，都用this！

示例: 购物车

|  |
| --- |
| <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tfoot下tr下最后一个td背景变为黄色！\*/    tfoot>tr>td:last-child{      background-color:yellow    }    /\*想让tbody下每个tr中最后一个td背景变为粉色\*/    tbody>tr>td:last-child{      background-color:pink    }  </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 class="subtitle">¥4488.00</td>      </tr>      <tr>        <td>iPhone6 plus</td>        <td>¥5288.00</td>        <td>          <button>-</button>          <span>1</span>          <button>+</button>        </td>        <td class="subtitle">¥5288.00</td>      </tr>      <tr>        <td>iPad Air 2</td>        <td>¥4288.00</td>        <td>          <button>-</button>          <span>1</span>          <button>+</button>        </td>        <td class="subtitle">¥4288.00</td>      </tr>    </tbody>    <tfoot>      <tr>        <td colspan="3">Total: </td>        <td>¥14064.00</td>      </tr>    </tfoot>  </table>  <script>  //功能1: 点击+/-按钮，数量变化  //今后做任何一种DOM效果的固定套路:  //DOM 4步  //1. 先查找触发事件的元素  //本例中: 因为用户点击按钮，修改数量，触发后续一系列变化，所以应该先找到table中所有按钮  //查找id为data的table元素  var table=document.getElementById("data");  console.log(table);  //查找table下所有的button元素  var btns=table.getElementsByTagName("button");  console.log(btns);  //查看第一个按钮对象身上有哪些可用的on开头的事件处理函数属性  console.dir(btns[0]);  //2. 提前为元素绑定事件处理函数  //本例中: 遍历找到的每个按钮对象，每遍历一个按钮对象，就为当前按钮对象绑定一个事件处理函数:  for(var btn of btns){    btn.onclick=function(){      //把当前按钮的内容变为❀      //this.innerHTML="❀";      //点按钮改数量        //3. 查找要修改的元素      //本例中: 点按钮要修改按钮旁边的span      //所以应该找按钮的爹的所有孩子中第二个孩子      var span=this.parentNode.children[1];      //4. 修改元素      //本例中:      //4.1 先取出span中现在的值，转为整数(凡是页面上的一切都是字符串！)      var n=parseInt(span.innerHTML);      //4.2 如果点的是+，就span的值+1, 否则就span的值-1      if(this.innerHTML=="+"){        n++;      }else if(n>1){//只有当n>1时，才允许-1，因为购物车中不可能出现数量为0的商品        n--;      }      //4.3 把修改后的新值，手动放回到span内容中(因为从页面上拿到的一切，都是副本！按值传递来的！所以，修改副本，不影响页面显示。想要修改页面显示，必须手动放回去！)      span.innerHTML=n;      //数量改小计也改        //3. 查找要修改的元素      //本例中: 查找当前按钮的父元素的下一个兄弟      var subTd=this.parentNode.nextElementSibling;      //4. 修改元素: 小计=数量n\*单价      //本例中: 先获得当前商品的单价:      //其实就是找按钮的爹的前一个兄弟的内容，转为小数      var price=parseFloat(          this.parentNode              .previousElementSibling              .innerHTML              .slice(1)      );      //计算小计      var subtotal=price\*n;      //再放回最后一个td中      subTd.innerHTML=`¥${subtotal.toFixed(2)}`;      //小计改总计也改        //3. 查找要修改的元素      //本例中:总计最终要放入tfoot中tr下最后一个td中      var totalTd=table.querySelector("tfoot>tr>td:last-child");      console.log(totalTd);      //4. 修改元素      //本例中: 找到所有的小计，遍历并求和，然后放入总计td中      //找到所有的小计: tbody下的每行中最后一个td      var subTds=table.querySelectorAll(        "tbody>tr>td:last-child"      );      console.log(subTds);      var total=0;      //遍历每个td      for(var td of subTds){        //每遍历一个td就取出内容，去掉¥，转为小数，累加到变量total中        total+=parseFloat(td.innerHTML.slice(1))      }      //将计算好的总价，放回总价td中      totalTd.innerHTML=`¥${total.toFixed(2)}`    }  }    //查找table下所有class为subtitle的td元素  var subTds=table.getElementsByClassName("subtitle");  console.log(subTds);  </script>  </body> |

总结:

1. 查找元素: 4种

(1). 不需要查找就可直接获得的元素: 4种

a. document

b. document.documentElement

c. document.head

d. document.body

(2). 按节点间关系查找: 2大类关系，6个属性

a. 父子关系

1). 获得当前元素的父元素: 元素.parentNode或 元素.parentElement

2). 获得当前元素所有直接子元素: 元素.children

3). 获得当前元素第一个直接子元素: 元素.firstElementChild

4). 获得当前元素最后一个直接子元素: 元素.lastElementChild

b. 兄弟关系

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

2). 获得当前元素的下一个兄弟元素: 元素.nextElementSibling

(3). 按HTML特征查找: 4个函数

a. 按id查找一个元素: var 一个元素=document.getElementById("id名")

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

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

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

(4). 按选择器查找:

a. 如果只想找一个元素: var 一个元素=任意父元素.querySelector("任意选择器")

b. 如果想找多个元素: var 类数组对象=任意父元素.querySelectorAll("任意父元素")

总结: this

1. obj.fun() this->obj

2. fun() 或 (function(){ ... })() 或 回调函数 this->window

3. new Fun() this->new将来要创建的那个子对象

4. 构造函数.prototype.方法=function(){ ... } this->将来调用这个方法的.前的某个子对象

5. 访问器属性中的this: this->当前访问器属性所在的对象本身

Object.defineProperties(obj,{

eage:{

get(){ return this.\_eage },

set(value){ this.\_eage=value}

}

})

6. DOM中事件处理函数中的this->当前触发事件的元素。

总结: es6

1. 拼接字符串: `...${js表达式}...` 模板字符串

2. 代替var声明变量: let

3. 简化function、保持函数内外this一致: => 箭头函数

4. 简化普通for循环遍历数组、类数组对象、字符串: for of

5. 希望一个形参变量，将来不传值，也有默认值使用时: function fun(形参=默认值)

6. 代替arguments接受不确定个数的实参值: function fun(...arr){ ... }  
7. 希望先打散数组为多个值，再传入函数时: fun(...数组)

8. 希望复制一个数组或对象: var arr2=[...arr] 或 var obj2={...obj}

9. 希望合并两个数组或对象: var arr3=[...arr1,...arr2] 或 var obj3={...obj1, ...obj2};

10. 希望提取大的数组或对象中个别成员出来单独使用: var [变量1, 变量2, ...]=数组 或 var {属性名1, 属性名2,...}=对象

11. 如果一个函数多个形参不确定给不给值，但又要求实参值必须传给规定的形参: 参数解构:

定义函数时: function fun({ 形参1=默认值, 形参2=默认值,... }){ ... }

调用函数时: fun({形参1: 实参值, 形参2:实参值 })

12. 代替构造函数和原型对象方法，定义一种类型: class

class 类型名{

constructor(){

}

原型对象方法(){

}

}

13. 为class添加访问器属性:

class 类型名{

constructor(){

}

get eage(){ return this.\_eage }

set eage(value){ this.\_eage=value }

}

14. 一个class继承另一个class:

class 子类型 extends 父类型{

constructor(){

super()

}

}

15. 希望两个异步函数必须顺序执行: Promise

定义函数时:

function fun(){

return new Promise(

function(open){

异步任务

在异步任务最后一句话执行之后调用open()开门，通知下一项任务可以继续执行

}

)

}

调用函数时: fun().then(下一项任务函数)

强调: 下一项任务函数一定不要加()，因为不是立刻调用！只是暂时保存起来！

作业:

1. (必须)复习css中所有选择器！

2. (必须)看小程序视频学习DOM优化:

小程序->在线->DOM->day01: 4. DOM优化 查找元素 getElementsByXXX vs querySelectorAll

3. (必须)看小程序视频学习高频笔试题:

小程序->在线->DOM->day01:

作业: 高频笔试题: 动态集合与非动态集合差别

作业: 高频笔试题: 多个按钮，点哪个按钮就提示点的是第几个按钮

作业: 高频鄙视提: 递归遍历一个父元素下所有后代元素