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->day02、day03

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->day02、day03

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->day02

6. 带class样式的表单验证(封装函数)

扩展: 如何单独操作某一个class HTML5 classList

小程序->在线->DOM->day03

2. 选择省份和城市 级联下拉列表

直接在day03.zip中2\_select\_option.html中做即可

作业: 使用navigator.userAgent鉴别浏览器名称和版本号

3. 单词列表: 小程序->单词->DOM->day02、day03

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9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. 修改: 修改样式

2. 添加/删除

3. HTML DOM常用对象  
4. BOM

一. 修改: 3种

1. 内容: 3种

2. 属性: 3种

3. 样式:

对应小程序视频: 小程序->在线->DOM->day02 5 修改样式，计算属性

(1).修改内联样式:

a. 元素.style.css属性名="属性值"

//自动翻译

b. 相当于: html中: <元素 style="... ; css属性:属性值"

c. 坑: .style属性仅代表元素的内联样式，无法获取或修改样式表(内部和外部)中的样式。但是，实际开发中，考虑到可维护性，几乎不会用内联样式！几乎所有样式都是写在样式表中的。所以，用.style属性，很可能只能获得内联样式，无法获得外部的大部分样式。

(2). 获取元素的完整样式:

a. 其实，每个元素都有一个计算后的样式（computedStyle），包含最终应用到这个元素上的所有css属性的集合！

b. 将来，只要获取样式，都应该获取计算后的样式，而不应该用style属性

c. 如何：2步:

1). 先获得这个元素所有计算后的样式的集合对象:

var style=getComputedStyle(元素对象);

2). 从style中获取某一个css属性继续使用:

var 属性值=style.css属性名

d. 强调: 所有通过getComputedStyle()获得的计算后的样式，都是只读的！禁止修改！

原因: 计算后的样式来源不确定，不确定同时有多少元素在使用这个css属性。一旦擅自修改，会牵一发而动全身！

(3). 总结:

a. 只要修改一个css属性，都用: 元素.style.css属性="属性值" —— 只修改内联

b. 只要获取元素的css属性，都用: getComputedStyle(元素对象) —— 获得计算后的完整样式！

(4). 示例: 分别使用style和getComputedStyle()获取和修改元素的css属性

0\_style\_getComputedStyle.html

|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>    <meta charset="UTF-8">    <meta name="viewport" content="width=device-width, initial-scale=1.0">    <title>Document</title>    <style>      h1{        background-color:red;      }    </style>  </head>  <body>    <h1 id="h1" style="color:yellow">欢迎访问tmooc</h1>    <p>welcome to my first web site</p>    <script>      var h1=document.getElementById("h1");      //想获得h1的字体颜色，背景颜色，字号      //使用style:      // console.log(h1.style.color);//yellow      // console.log(h1.style.backgroundColor);//空      // console.log(h1.style.fontSize);//空      //获取计算后样式:      var style=getComputedStyle(h1);      console.log(style.color);//      console.log(style.backgroundColor);//      console.log(style.fontSize);//      //想修改h1的字体为64px      //错误: 修改计算后的样式      //style.fontSize="64px";//报错：Failed to set the 'font-size' property on 'CSSStyleDeclaration': These styles are computed, and therefore the 'font-size' property is read-only.      //正确: 修改内联，仅影响当前元素自己，不影响别人      h1.style.fontSize="64px";    </script>  </body>  </html>  运行结果:  rgb(255, 255, 0)  rgb(255, 0, 0)  32px |

对应小程序视频: 小程序->在线->DOM->day02 6. 带class样式的表单验证

(5). 问题: .style属性一句话只能修改一个css属性。但是实际开发中，一个效果的变化，可能同时要修改多个css属性。如果用.style，就会很繁琐。



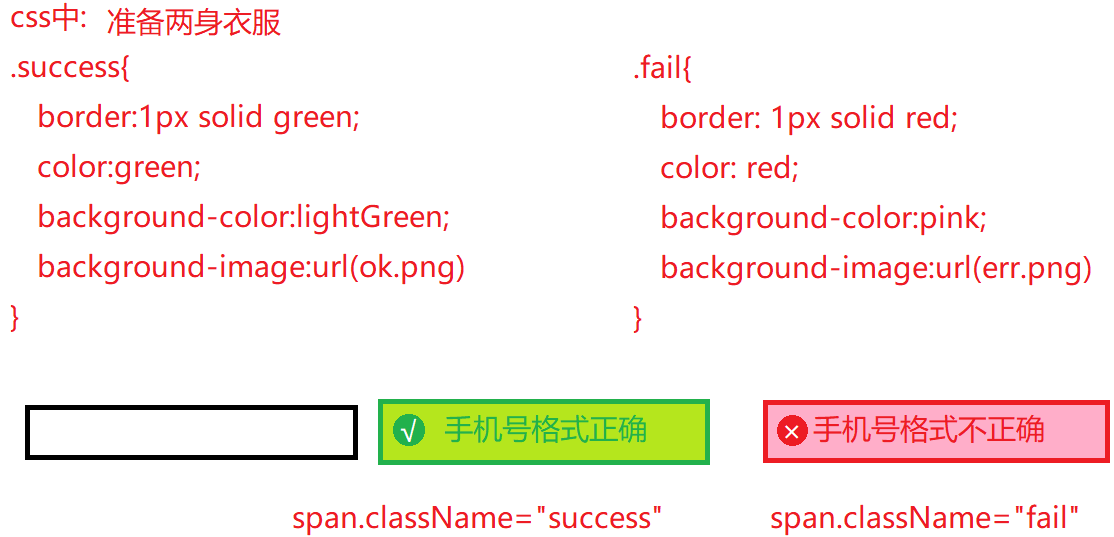
(6). 解决: 今后只要批量修改一个元素的多个css属性，首选class修改

(7). 如何: 2步

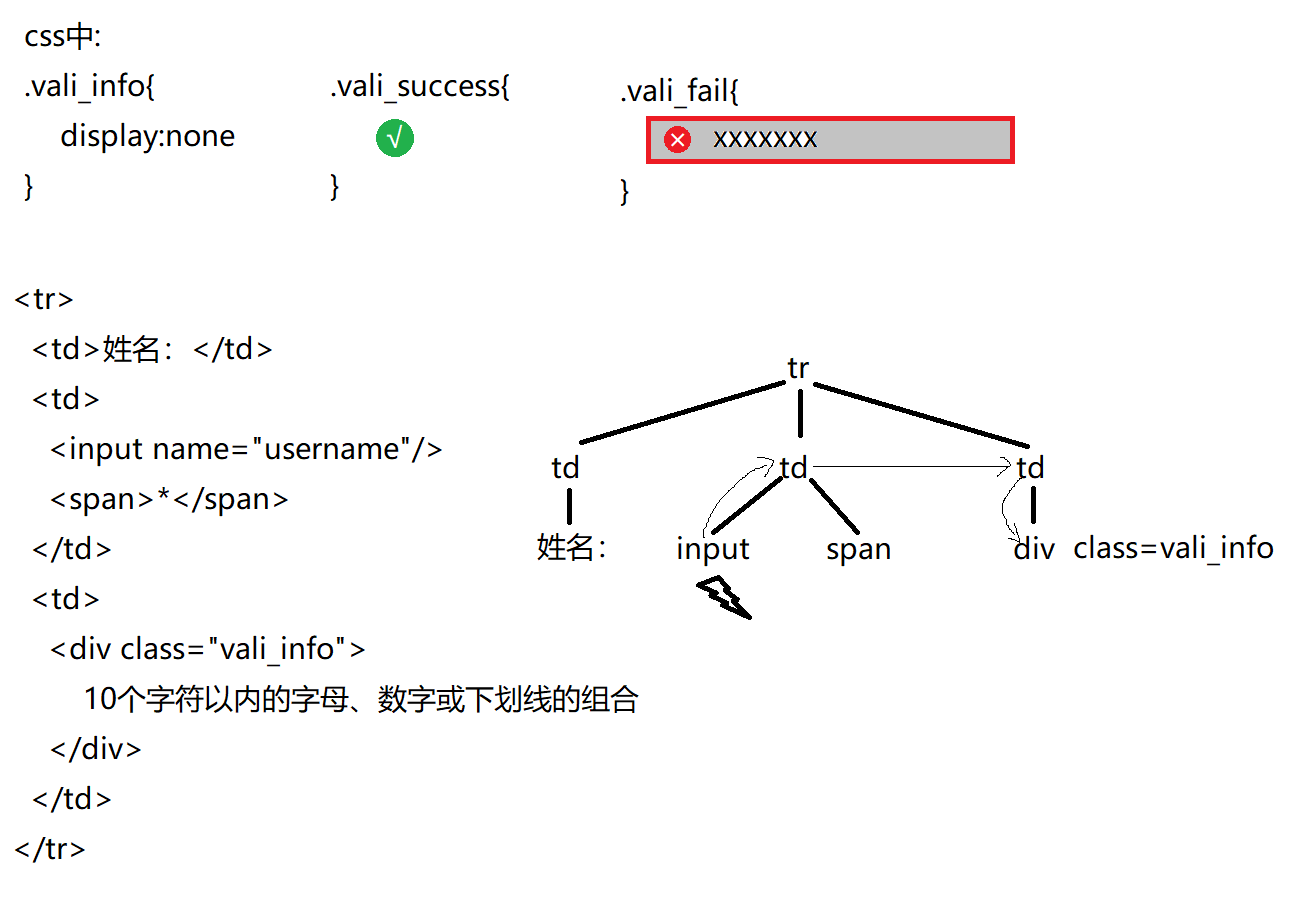
a. 先在css中提前定义好要用的各种备选class样式

b. 在js中根据程序需要，动态修改元素的class为指定的某一种class样式

(8). 无论何时，需要批量修改元素的样式，只需要一句话就够了！



(9). 示例: 带样式的表单验证:



0\_valiWithCss.html

|  |
| --- |
| <!doctype html>  <html>   <head>    <meta charset="UTF-8">    <title>实现带样式的表单验证</title>    <style>      table{width:700px}      td:first-child{width:60px}      td:nth-child(2){width:200px}      td:first-child+td{width:200px}      td span{color:red}      /\*文本框旁边div隐藏的根源\*/      .vali\_info{        display:none;      }      .txt\_focus{        border-top:2px solid black;        border-left:2px solid black;      }      .vali\_success,.vali\_fail{        background-repeat:no-repeat;        background-position:left center;        display:block;      }      /\*文本框验证通过时 div的样式\*/      .vali\_success{          background-image:url("../images/ok.png");        padding-left:20px;        width:0px;height:20px;        overflow:hidden;      }      /\*文本框验证失败时 div的样式\*/      .vali\_fail{          background-image:url("../images/err.png");          border:1px solid red;          background-color:#ddd;          color:Red;          padding-left:30px;      }    </style>   </head>   <body>    <form id="form1">      <h2>增加管理员</h2>      <table>        <tr>          <td>姓名：</td>          <td>            <input name="username"/>            <span>\*</span>          </td>          <td>            <div class="vali\_info base1 base2">              10个字符以内的字母、数字或下划线的组合            </div>          </td>        </tr>        <tr>          <td>密码：</td>          <td>            <input type="password" name="pwd"/>            <span>\*</span>          </td>          <td>            <div class="vali\_info">6位数字</div>          </td>        </tr>        <tr>          <td></td>          <td colspan="2">            <input type="submit" value="保存"/>            <input type="reset" value="重填"/>          </td>        </tr>        </table>    </form>  <script>  //只实现姓名文本框的验证:  //2个需求:  //1. 文本框获得焦点时，显示右边的提示  //2. 文本框失去焦点时，验证文本款中的内容    //如果验证通过，就修改右边的提示为正确的样式    //如果验证不通过，就修改右边的提示为错误的样式  //实现第一个需求: 文本框获得焦点时，显示右边的提示  //DOM 4步:  //1. 查找触发事件的元素  //本例中: name属性=username的文本框获得焦点触发变化  //元素缩写 业务单词  //   ↓    ↓  var txtName=    document.getElementsByName("username")[0];//复习  console.log(txtName);  //2. 绑定事件处理函数  //本例中: 文本框获得焦点时触发变化  txtName.onfocus=function(){    //3. 查找要修改的元素    //本例中: 当前文本框获得焦点，却要修改当前文本框的父元素td的下一个兄弟元素td的第一个孩子元素div的样式    var div=this.parentNode.nextElementSibling.children[0];    //4. 修改元素    //本例中: div因为带有vali\_info class才默认隐藏，所以，我们需要将div的class暂时清除，让div显示出来    div.className="";  }  //实现第二个需求: 当文本框失去焦点时，验证文本框的内容，修改旁边div的样式  //DOM 4步  //1. 查找触发事件的元素  //本例中: 还是姓名文本框失去焦点——上边已经找过了txtName  //2. 绑定事件处理函数  //本例中: 文本框失去焦点时，触发验证和修改  txtName.onblur=function(){    //3. 查找要修改的元素    //本例中: 当前文本框失去焦点，却要修改当前文本框的父元素td的下一个兄弟元素td的第一个孩子元素div的样式    var div=this.parentNode.nextElementSibling.children[0];    //4. 修改元素    //定义正则表达式    var reg=/^\w{1,10}$/;    //如果验证文本框的内容通过    if(reg.test(this.value)==true){      //就修改div为正确的样式      div.className="vali\_success";    }else{//否则如果验证文本框的内容不通过      //就修改div为错误的样式      div.className="vali\_fail";    }  }  </script>   </body>  </html>  运行结果: |

二. 添加/删除

对应小程序视频: 小程序->在线->DOM->day03 1. DOM 添加新元素 3步

1. 添加一个新元素: 3步

(1). 创建一个新的空元素对象:

var 新对象=document.createElement("标签名")

比如: 想创建一个a元素:

var a=document.createElement("a");

结果:<a></a>

(2). 为新元素添加必要属性:

比如: 想将a设置为跳转到tmooc的一个超链接

a.href="http://tmooc.cn";

a.innerHTML="go to tmooc";

结果:

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

(3). 将新元素添加到DOM树

a. 问题: 已经创建出来，且已经设置了关键属性的元素，竟然在网页上看不到！

b. 原因: 浏览器要显示网页内容，必须经过三个阶段！

1). 扫描HTML代码，生成DOM树

2). 根据DOM树中的元素结构，对网页进行排版

3). 根据排版的布局和位置绘制网页中的内容。

而新添加的元素，默认是不在DOM树中的！浏览器根本不知道页面上需要显示新元素！

c. 解决: 今后，只要希望页面上添加一个新元素，都要将新元素先添加到DOM树的指定位置。然后浏览器才能自动侦测到DOM树的变化，重新排版和绘制网页！

d. 如何: 把元素添加到DOM树，3种:

1). 在一个指定父元素下所有直接子元素末尾追加新元素:

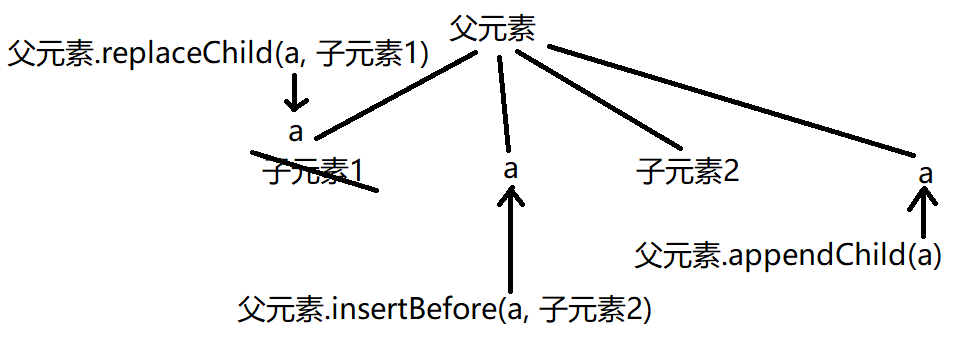
父元素.appendChild(新元素)

2). 在一个指定父元素下的某个子元素之前插入新元素:

父元素.insertBefore(新元素, 现有子元素)

3). 替换一个指定父元素下的某个现有子元素

父元素.replaceChild(新元素, 现有子元素)

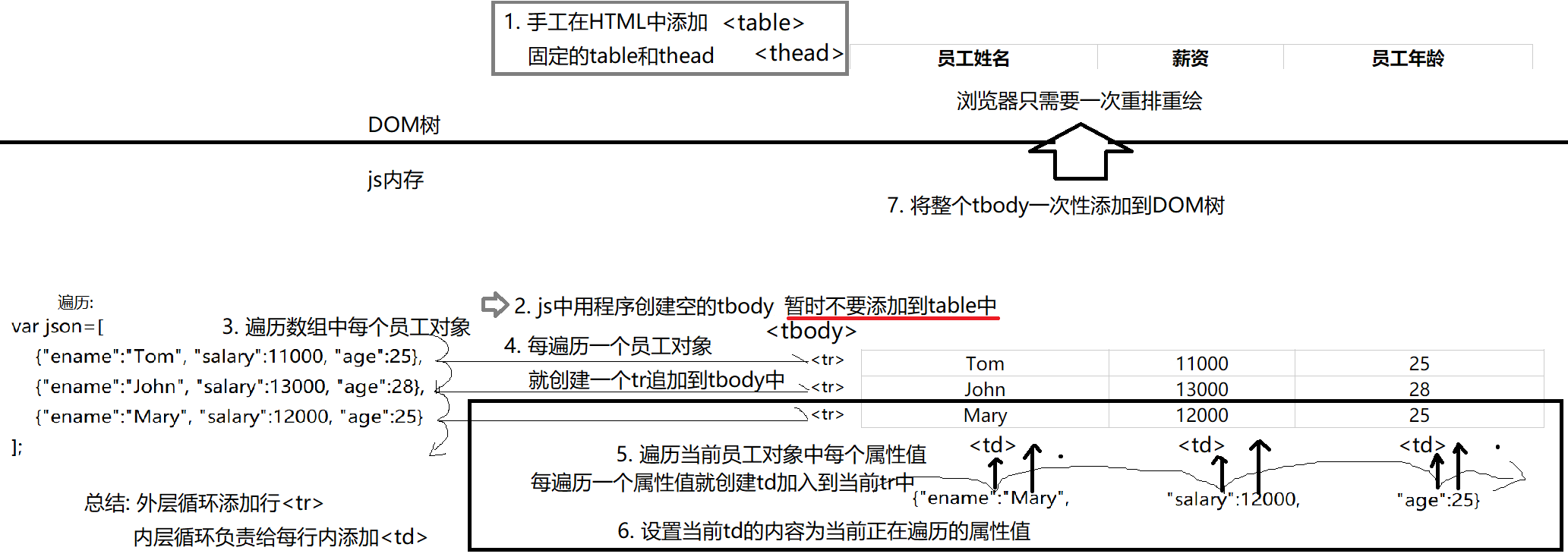


2. 示例: 在页面上添加一个a元素和一个input元素

1\_createElement.html

|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>    <meta charset="UTF-8">    <meta name="viewport" content="width=device-width, initial-scale=1.0">    <title>Document</title>  </head>  <body>    <script>      //想创建a      var a=document.createElement("a");      console.log(a.outerHTML);      a.href="http://tmooc.cn";      a.innerHTML="go to tmooc";      console.log(a);      //想把a追加到body中      document.body.appendChild(a);      //创建一个文本框      var input=document.createElement("input");      //省略第二步      //希望input出现在a之后:      document.body.appendChild(input);      //希望input出现在a之前:      //document.body.insertBefore(input,a);      //希望用input替换a      //document.body.replaceChild(input,a);    </script>  </body>  </html>  运行结果: |

3. 示例: 动态创建表格



1\_createTable.html

|  |
| --- |
| <!DOCTYPE HTML>  <html>  <head>  <title>动态创建表格</title>  <meta charset="utf-8" />  <style>    table{width:600px; border-collapse:collapse;      text-align:center;    }    td,th{border:1px solid #ccc}  </style>  </head>  <body>    <div id="data">    <!-- table>thead>tr>th\*3 -->      <table>        <thead>          <tr>            <th>姓名</th>            <th>薪资</th>            <th>年龄</th>          </tr>        </thead>      </table>    </div>    <script>    var json=[      {"ename":"Tom", "salary":11000, "age":25},      {"ename":"John", "salary":13000, "age":28},      {"ename":"Mary", "salary":12000, "age":25}    ];    //1. 在HTML中手工添加固定不变的table和表头行内容    //查找id为data的div下的table元素    var table=document.querySelector("#data>table")    //2. 创建一个tbody元素    var tbody=document.createElement("tbody");    //暂时不要添加到table中    //table.appendChild(tbody);    //3. 遍历json数组中每个员工对象    for(var emp of json){      //4. 每遍历一个员工对象，就创建一个tr元素，追加到tbody中      var tr=document.createElement("tr");      tbody.appendChild(tr);      //5. 遍历当前员工对象中每个属性      for(var key in emp){        //每遍历一个属性值，就创建td，追加到tr中        var td=document.createElement("td");        tr.appendChild(td);        //6. 设置td的内容为当前属性的属性值        td.innerHTML=emp[key];      }    }    //7. 当tbody元素下所有子内容填充完成之后，再将tbody一次性添加到dom树    table.appendChild(tbody);    </script>  </body>  </html>  运行结果: |

4. 优化:

(1). 问题: 每修改一次DOM树，浏览器都要重新排版，重新绘制。如果频繁修改DOM树，浏览器就被迫重新反复排版，反复绘制！——加载效率极低

(2). 解决: 今后，应该尽量减少操作DOM树的次数！

(3). 如何: 2种情况:

a. 如果同时添加父元素和子元素时，应该现在内存中，将所有子元素添加到父元素中。最后再一次性将父元素添加到DOM树。

b. 父元素已经在页面上了！要添加多个平级子元素。应该借助于一种新的对象——文档片段对象（临时托盘）

1). 什么是文档片段对象： 内存中临时保存多个平级子元素的虚拟父元素对象

2). 何时: 父元素已经在页面上了！要添加多个平级子元素时，都要用文档片段对象。

3). 如何: 3步:

i. 先创建文档片段对象:

var 文档片段=document.createDocumentFragment();

创建 文档 片段

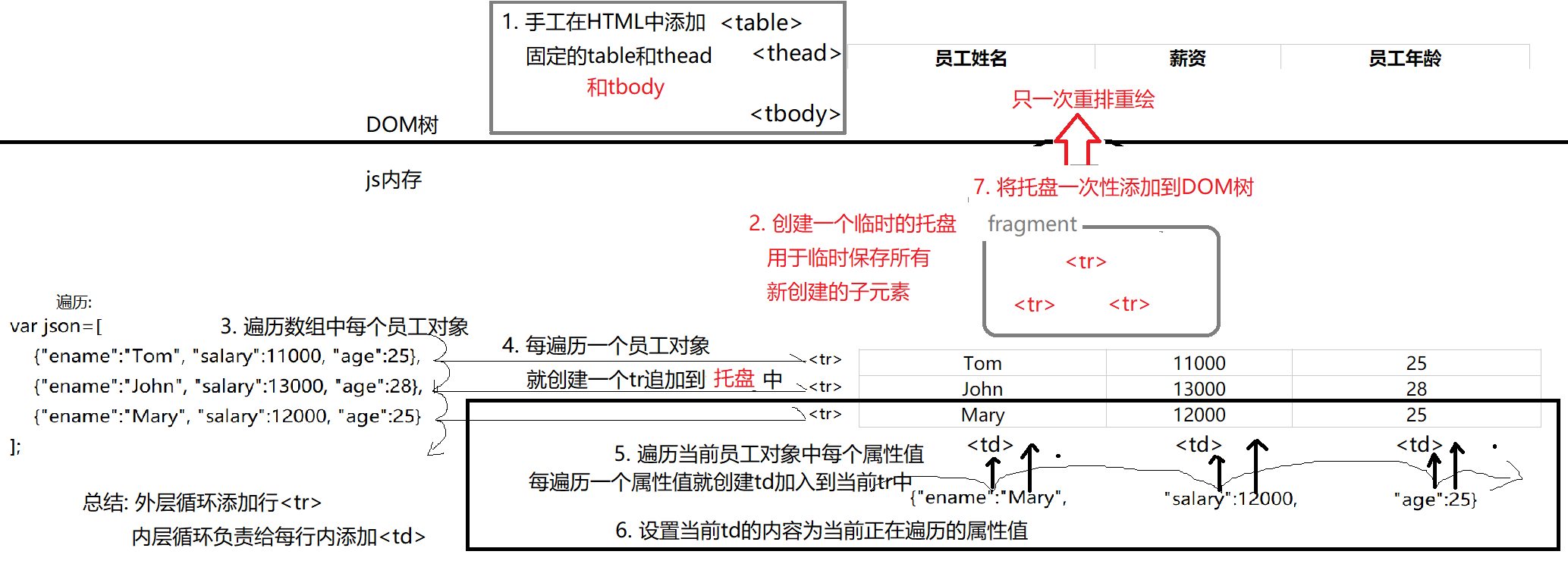
ii. 再将子元素添加到文档片段对象中:

文档片段.appendChild(子元素)

iii. 最后再将整个文档片段一次性添加到DOM树中

父元素.appendChild(文档片段);

4). 强调: 临时创建的文档片段对象，再将子元素追加到他们真正的父元素下之后，文档片段对象就释放了！不会成为页面上真正的元素占用页面空间！



5). 示例: 如果tbody已经在页面上了，如何添加多个tr

1\_createTable2.html

|  |
| --- |
| <!DOCTYPE HTML>  <html>  <head>  <title>动态创建表格</title>  <meta charset="utf-8" />  <style>    table{width:600px; border-collapse:collapse;      text-align:center;    }    td,th{border:1px solid #ccc}  </style>  </head>  <body>    <div id="data">    <!-- table>thead>tr>th\*3 -->      <table>        <thead>          <tr>            <th>姓名</th>            <th>薪资</th>            <th>年龄</th>          </tr>        </thead>        <tbody>        </tbody>      </table>    </div>    <script>    var json=[      {"ename":"Tom", "salary":11000, "age":25},      {"ename":"John", "salary":13000, "age":28},      {"ename":"Mary", "salary":12000, "age":25}    ];    //1. 在HTML中手工添加固定不变的table和表头行内容，并提前添加<tbody>元素    //2. 创建一个文档片段对象    var frag=document.createDocumentFragment();      //3. 遍历json数组中每个员工对象    for(var emp of json){      //4. 每遍历一个员工对象，就创建一个tr元素，追加到文档片段中      var tr=document.createElement("tr");      frag.appendChild(tr);      //5. 遍历当前员工对象中每个属性      for(var key in emp){        //每遍历一个属性值，就创建td，追加到tr中        var td=document.createElement("td");        tr.appendChild(td);        //6. 设置td的内容为当前属性的属性值        td.innerHTML=emp[key];      }    }    //7. 将文档片段一次性添加到table>tbody下    var tbody=      document.querySelector("#data>table>tbody");    tbody.appendChild(frag);    </script>  </body>  </html>  运行结果: |

5. 删除元素: 父元素.removeChild(子元素)

三. HTML DOM常用对象: 对常用的个别复杂的元素操作提供的简化版访问方式

——了解——鸡肋

1. <img>: 创建img时:

(1). 核心DOM: var img=document.createElement("img")

(2). HTML DOM: var img=new Image();

(3). 强调: 绝不是所有元素都能new，最常用的元素中只有两个元素可以new:

a. var img=new Image()

b. var opt=new Option(文本, 值)

2. <table>: 逐级管理:

(1). table管着行分组:

a. table可以添加行分组:

1). var thead=table.createTHead();

一句话做了两件事: 1. 既创建thead元素，2. 同时又将thead追加到table上

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();

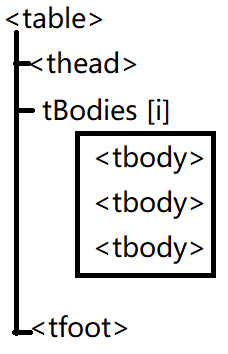
~~真的没有table.deleteTBody()~~

c. table可以获取行分组

1). table.tHead

2). table.tFoot

3). table.tBodies[i]



(2). 行分组管着行:

a. 行分组可添加一行:

1). var tr=行分组.insertRow(i)

2). 意为: 在当前行分组中第i行位置插入一个新行

3). 也是一句话办2件事: 1. 既创建， 2. 又插入

4). 说明: 如果i位置是行分组内的中间位置，i位置已经有旧行了。那么，insertRow(i)，添加的新行会将原i位置的行向后挤，新行占据i位置。

5). 两个位置用的特别多:

开头: 想在开头插入一个新行: var tr=行分组.insertRow(0)

结尾: 想在结尾插入一个新行: var tr=行分组.insertRow() //默认就是末尾追加

b. 行分组可删除一行:

1). 行分组.deleteRow(i)

2). 意为: 删除当前行分组下的第i行

3). 强调: i，应该是要删除的行在行分组内的相对下标位置

4). 问题: 如果用户随便点一行，要删除，又因为页面中行很多，不容易一眼看出要删除的是第几行，行分组.deleteRow(?)中下标应该写几

5). 解决: 每个行对象上都自动有一个.rowIndex属性记录着该行在整个table中的下标位置！

6). 问题: 行分组.deleteRow(i)要求行在行分组内的相对下标。而.rowIndex提供的是行在整个表中的下标。行在整个表中的下标和行在行分组内的相对下标，因为受表头行的影响，几乎都不相等！如果用行分组.deleteRow(tr.rowIndex)删除一行，则实际删除的是tr下方的某一行！

7). 解决: 其实table元素对象上也有一个deleteRow(i)函数，因为调用时.前的主语换成了table，所以需要的参数i，也变成了要删除的行在整个表中的下标位置。刚好和tr.rowIndex配对！

8). 结论: 今后只要在行分组中删除行，都应该用:

table.deleteRow(tr.rowIndex)



c. 行分组可获取一行: var tr=行分组.rows[i]

(3). 行管着格:

a. 行可以添加格: 一行末尾追加格: var td=tr.insertCell()

b. 行可以删除格: tr.deleteCell(i)

c. 行可以获取: tr.cells[i]

(4). 示例: 使用HTML DOM简写实现动态加载表格内容，以及删除行

1\_createTable3\_HTMLDOM.html

|  |
| --- |
| <!DOCTYPE HTML>  <html>  <head>  <title>动态创建表格</title>  <meta charset="utf-8" />  <style>    table{width:600px; border-collapse:collapse;      text-align:center;    }    td,th{border:1px solid #ccc}  </style>  </head>  <body>    <div id="data">    <!-- table>thead>tr>th\*3 -->      <table>        <thead>          <tr>            <th>姓名</th>            <th>薪资</th>            <th>年龄</th>            <th>删除</th>          </tr>        </thead>      </table>    </div>    <script>    var json=[      {"ename":"Tom", "salary":11000, "age":25},      {"ename":"John", "salary":13000, "age":28},      {"ename":"Mary", "salary":12000, "age":25}    ];    //1. 在HTML中手工添加固定不变的table和表头行内容    //查找id为data的div下的table元素    var table=document.querySelector("#data>table")    //2. 创建一个tbody元素    var tbody=document.createElement("tbody");    //3. 遍历json数组中每个员工对象    for(var emp of json){      //4. 每遍历一个员工对象，就创建一个tr元素，追加到tbody中      //旧核心DOM:      // var tr=document.createElement("tr");      // tbody.appendChild(tr);      //HTML DOM:      var tr=tbody.insertRow();      //5. 遍历当前员工对象中每个属性      for(var key in emp){        //每遍历一个属性值，就创建td，追加到tr中        //旧核心DOM:        // var td=document.createElement("td");        // tr.appendChild(td);        //HTML DOM:        var td=tr.insertCell();        //6. 设置td的内容为当前属性的属性值        td.innerHTML=emp[key];      }//内层for循环之后      //在添加完每行的格之后，再额外多添加一个格      var td=tr.insertCell();      //想td中添加一个按钮      var btn=document.createElement("button");      td.appendChild(btn);      btn.innerHTML="×";      btn.onclick=function(){        // alert("疼!");        // confirm("是否继续删除?");        //查找当前按钮所在行        //      btn    td          tr        var tr=this.parentNode.parentNode;        //获得当前行中的员工姓名:        var ename=tr.cells[0].innerHTML;        //希望先弹出确认框，用户点击确认后，才        if(confirm(`是否继续删除 ${ename} 吗?`)==true){          //最终: 删除当前按钮所在的行          table.deleteRow(tr.rowIndex);        }        //否则如果用户点击取消，则我们也不做！      }    }//外层for循环里！    //7. 当tbody元素下所有子内容填充完成之后，再将tbody一次性添加到dom树    table.appendChild(tbody);    </script>  </body>  </html>  运行结果: |

|  |
| --- |
| 浏览器三大对话框:  1. 输入框: var 用户输入的内容=prompt("提示信息")  2. 警告框: alert("警告信息")    3. 确认框: var bool是否同意继续=confirm("提示信息")  true false |

3. <form>元素及其内部表单元素的简写:

(1). 获取<form>元素时:

a. document已经将页面中所有<form>元素都集中存储在document.forms集合中。可通过下标，方式来获取

b. var form=document.forms[i]

(2). 获取<form>元素内的表单元素时

a. 标准: <form>元素内的所有表单元素(input, textarea, button, select)都已经被收集到了form对象的elements集合中。

var 表单元素=form.elements[i或id或name]

b. 简写: 如果这个表单元素带有name属性，可直接:

var 表单元素=form.name名

(3). 让表单元素自动获得焦点: 表单元素.focus()

(4). 示例: 使用HTMLDOM简化带样式的表单验证，并让姓名文本框自动获得焦点

0\_valiWithCss\_HTMLDOM.html

|  |
| --- |
| <!doctype html>  <html>   <head>    <meta charset="UTF-8">    <title>实现带样式的表单验证</title>    <style>      table{width:700px}      td:first-child{width:60px}      td:nth-child(2){width:200px}      td:first-child+td{width:200px}      td span{color:red}      /\*文本框旁边div隐藏的根源\*/      .vali\_info{        display:none;      }      .txt\_focus{        border-top:2px solid black;        border-left:2px solid black;      }      .vali\_success,.vali\_fail{        background-repeat:no-repeat;        background-position:left center;        display:block;      }      /\*文本框验证通过时 div的样式\*/      .vali\_success{          background-image:url("../images/ok.png");        padding-left:20px;        width:0px;height:20px;        overflow:hidden;      }      /\*文本框验证失败时 div的样式\*/      .vali\_fail{          background-image:url("../images/err.png");          border:1px solid red;          background-color:#ddd;          color:Red;          padding-left:30px;      }    </style>   </head>   <body>    <form id="form1">      <h2>增加管理员</h2>      <table>        <tr>          <td>姓名：</td>          <td>            <input name="username"/>            <span>\*</span>          </td>          <td>            <div class="vali\_info">              10个字符以内的字母、数字或下划线的组合            </div>          </td>        </tr>        <tr>          <td>密码：</td>          <td>            <input type="password" name="pwd"/>            <span>\*</span>          </td>          <td>            <div class="vali\_info">6位数字</div>          </td>        </tr>        <tr>          <td></td>          <td colspan="2">            <input type="submit" value="保存"/>            <input type="reset" value="重填"/>          </td>        </tr>        </table>    </form>  <script>  //只实现姓名文本框的验证:  //2个需求:  //1. 文本框获得焦点时，显示右边的提示  //2. 文本框失去焦点时，验证文本款中的内容    //如果验证通过，就修改右边的提示为正确的样式    //如果验证不通过，就修改右边的提示为错误的样式  //实现第一个需求: 文本框获得焦点时，显示右边的提示  //DOM 4步:  //1. 查找触发事件的元素  //本例中: name属性=username的文本框获得焦点触发变化  //先找到form元素对象:  var form=document.forms[0];  //元素缩写 业务单词  //   ↓    ↓  var txtName=form.username;            //form.elements["username"];            //form.elements[0];  //希望一加载网页，就让姓名文本框默认获得焦点！  txtName.focus();  //2. 绑定事件处理函数  //本例中: 文本框获得焦点时触发变化  txtName.onfocus=function(){    //3. 查找要修改的元素    //本例中: 当前文本框获得焦点，却要修改当前文本框的父元素td的下一个兄弟元素td的第一个孩子元素div的样式    var div=this.parentNode.nextElementSibling.children[0];    //4. 修改元素    //本例中: div因为带有vali\_info class才默认隐藏，所以，我们需要将div的class暂时清除，让div显示出来    div.className="";  }  //实现第二个需求: 当文本框失去焦点时，验证文本框的内容，修改旁边div的样式  //DOM 4步  //1. 查找触发事件的元素  //本例中: 还是姓名文本框失去焦点——上边已经找过了txtName  //2. 绑定事件处理函数  //本例中: 文本框失去焦点时，触发验证和修改  txtName.onblur=function(){    //3. 查找要修改的元素    //本例中: 当前文本框失去焦点，却要修改当前文本框的父元素td的下一个兄弟元素td的第一个孩子元素div的样式    var div=this.parentNode.nextElementSibling.children[0];    //4. 修改元素    //定义正则表达式    var reg=/^\w{1,10}$/;    //如果验证文本框的内容通过    if(reg.test(this.value)==true){      //就修改div为正确的样式      div.className="vali\_success";    }else{//否则如果验证文本框的内容不通过      //就修改div为错误的样式      div.className="vali\_fail";    }  }  </script>   </body>  </html>  运行结果: |

总结:

2. 修改元素: 3 种

(1). 修改内容: 3种

a. 获取或修改元素的原始HTML内容: 元素.innerHTML

b. 获取或修改元素的纯文本内容: 元素.textContent

c. 获取或修改表单元素的值: 元素.value

(2). 修改属性: 3种:

a. 字符串类型的HTML标准属性: 2种方式  
 1). 核心DOM 4个函数:

i. 获取属性值: var 属性值=元素.getAttribute("属性名")

ii. 修改属性值: 元素.setAttribute("属性名","属性值")

iii. 判断是否包含某个属性: var bool=元素.hasAttribute("属性名")

iv. 移除属性: 元素.removeAttribute("属性名")

2). HTML DOM简写:

i. 获取属性值:元素.属性名

ii. 修改属性值: 元素.属性名="属性值"

iii. 判断是否包含某个属性:元素.属性名!==""

iv. 移除属性: 元素.属性名=""

b. bool类型HTML标准属性: 只能用"元素.属性名"，且值为bool类型

c. 自定义扩展属性:

1) HTML中: <元素 data-自定义属性名="属性值">

2). js中: 2种:

i. 核心DOM:

var 属性值=元素.getAttribute("data-自定义属性名")

元素.setAttribute("data-自定义属性名","属性值")

ii. HTML5标准: 元素.dataset.自定义属性名

(3). 修改样式:

a. 修改元素的内联样式: 元素.style.css属性="属性值"

b. 获取元素的完整样式:

var style=getComputedStyle(元素对象);

style.css属性

c. 批量修改元素的样式时，都用class:

元素.className="class名"

3. 添加/删除元素:

(1). 只添加一个新元素: 3步

a. 创建一个新元素:

var 新元素=document.createElement("标签名")

b. 为元素设置关键属性:

新元素.属性名="属性值";

c. 将新元素添加到DOM树: 3种:

1). 末尾追加:

父元素.appendChild(新元素)

2). 在某个元素前插入:

父元素.insertBefore(新元素, 现有元素)

3). 替换某个元素:

父元素.replaceChild(新元素, 现有元素)

(2). 优化: 尽量减少操作DOM树的次数

a. 如果同时添加父元素和子元素，应该先将子元素添加到父元素，最后再将父元素一次性添加到DOM树

b. 如果父元素已经在页面上，要添加多个平级子元素。应该利用文档片段对象

1). 创建文档片段对象:

var frag=document.createDocumentFragment()

2). 将子元素添加到文档片段对象中:

frag.appendChild(子元素)

3). 最后将文档片段对象一次性添加到DOM树上父元素下  
 父元素.appendChild(frag);

(3). 删除元素: 父元素.removeChild(子元素)

4. HTML DOM常用对象:（了解即可）

(1). var img=new Image()

(2). table  
a. table管着行分组:

1). 添加行分组:

var thead=table.createTHead()

var tbody=table.createTBody()

var tfoot=table.createTFoot()

2) 删除行分组:

table.deleteTHead(); table.deleteTFoot()

3). 获取行分组:

table.tHead table. tFoot table.tBodies[i]

b. 行分组管着行:

1). 添加行:

i. 任意行插入新行: var tr=行分组.insertRow(i);

ii. 开头插入新行: var tr=行分组.insertRow(0)

iii. 末尾追加新行: var tr=行分组.insertRow()

2). 删除行: table.deleteRow(tr.rowIndex)

3). 获取行: 行分组.rows[i]

c. 行管着格:

1). 添加格: var td=tr.insertCell()

2). 删除格: tr.deleteCell(i)

3). 获取格: tr.cells[i]

(3). form:

a. 获取form元素: document.forms[i]

b. 获取form中的表单元素:

1). 标准: form.elements[i或id或name名]

2). 简写: 如果有name属性: form.name名

c. 让表单元素自动获得焦点: 表单元素.focus()

今日对应小程序视频:

小程序->在线->DOM->day02 5. 修改样式，计算属性

6. 带class样式的表单验证

小程序->在线->DOM->day03 1. DOM 添加新元素 3步 动态创建表格

作业:

1. 复习今日问题清单: 小程序->在线->DOM->day02、day03

2. 看小程序视频实现: 倒计时自动关闭窗口

小程序->在线->DOM->day03 作业: 倒计时关闭窗口

3. 看小程序视频学习DOM版学子商城项目

小程序->在线->DOM->day04 扩展: DOM版学子商城...

家里环境xampp和nodejs能用，且第一阶段学的好的同学:

下载网盘中xzserver\_start.zip，看视频中day01，跟着做

家里环境xampp和nodejs不能用，或者第一阶段学的不好的同学：

只下载网盘中public.zip，看视频中day01，跟着做

然后，所有网页用live server运行

视频中所有ajax请求的服务器端接口地址，都改为我的新浪云服务器地址，就不需要你再运行服务器端了。也不需要你了解任何服务器端知识了:

/index 改为 http://xzserver.applinzi.com/index

/details 改为 http://xzserver.applinzi.com/details?lid=xxx