小程序: WEB问题速查

1. 问题清单: 小程序->首页->DOM->day03

2. 高清无码小视频: 小程序->在线->DOM->day03

今日扩展视频:

小程序->在线->DOM->day03

2. 选择省份和城市 级联下拉列表

所有以"作业"和"扩展"开头的视频

不用今天都看完！！！！！！\* \*\*两个月后\*\*\*鄙视面试前\*\*\*看完就行\*\*\*！

3. 单词列表: 小程序->单词->DOM->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. 拼写错误！自己找！让我帮你找，你需要手抄20遍，给我拍照！

3. 现在正处在照猫画虎的阶段！每个DOM效果至少写3遍:

3.1 注释+代码抄一遍！

3.2 保留注释，删除代码，把代码再填回来！

3.3 删除注释和代码，试着用自己的话把注释写回来，根据自己的注释，把代码填回来

一个DOM效果，只有自己能写出人话注释，才算自己会！——标准！

如何提问:

1. 上课时，案例中，笔记中那句话哪行代码看不懂，立刻微信私信问我！

2. 小程序中问题清单里哪个问题和回答看不懂！立刻微信私信问我！

3. 个人项目中哪个功能不会做！立刻微信私信问我！

4. 刷笔试题时，遇到不会的笔试题！立刻微信私信问我！

下载源代码: 右键->链接另存为！

正课:

1. 修改

2. 添加删除替换

3. HTML DOM常用对象

\*\*\*\*\*BOM\*\*\*\*\*

1. 什么是BOM

2. window

3. 打开新链接4种方式

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->JSCORE->day03: 1 修改样式, 计算后的样式

(1). 修改内联样式:

a. 元素.style.css属性名="属性值"  
 b. 相当于: 修改HTML中<元素>的style属性内的某个css属性的值

<元素 style="css属性名:属性值">

c. 问题: 有些css属性名是带-的，但是js中除了减法计算之外，不允许随便写-。会混淆

d. 解决: DOM标准规定，所有带-的css属性名，一律去横线变驼峰:

比如: z-index -> zIndex

font-size -> fontSize

(2). 修改样式表中的样式:

a. 强烈不建议直接修改样式表中的样式，因为样式表中的样式是共用的，一旦随意修改，会牵一发而动全身！

b. 如果针对一个元素，确实想改变样式表中继承或层叠来的样式，应该也修改内联样式:

1). 因为内联样式优先级最高！可以覆盖样式表中的样式

2). 因为内联样式只影响当前一个元素，不会影响其他元素。

(3). 示例: 修改元素的内联样式和样式表中的样式:

0\_style.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">Welcome</h1>    <script>      var h1=document.getElementById("h1");      //修改:      //想修改字体颜色为白色      h1.style.color="#fff";      //想修改背景样色为绿色      h1.style.backgroundColor="green";    </script>  </body>  </html>  运行结果: |

(4). 获取样式:

a. 不好的做法: 使用.style获取样式。因为.style仅仅代表元素的内联样式，不包含继承和层叠来的样式表中的样式。而实际项目中，很少使用内联样式。结果.style几乎拿不到所有的样式！

b. 好的做法: 今后只要获得css属性值，都要获得计算后的样式

1). 什么是计算后的样式: 网页加载完成后，最终应用到一个元素身上的所有样式的集合。即包含内联，又包含所有样式表中的样式！

2). 如何: 2步:

i. 先获得包含所有计算后的样式的对象:

var style=getComputedStyle(元素对象)

ii. 从style中获取对应的css属性值

style.css属性

3). 坑: getComputedStyle()虽然可以获得完整的样式集合，但是其中的样式，一个也不允许修改！都是只读！

4). 原因: 计算后的样式来源不确定！所以，一旦擅自修改，很可能牵一发而动全身！所以，计算后的样式都是只读的。要先个修改只能用.style通过修改内联样式来覆盖样式表中的样式。

c. 示例: 获取元素的多个样式属性:

0\_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">Welcome</h1>    <script>      var h1=document.getElementById("h1");      //获取:      //不好的做法:      // //想获取h1的字体颜色属性:      // console.log(h1.style.color);//内联      // //想获取h1的背景颜色属性:      // console.log(h1.style.backgroundColor);//样式表      // //想获得h1的字体大小:      // console.log(h1.style.fontSize);//默认样式表      //好的做法:      var style=getComputedStyle(h1);      console.log(style);      //想获取h1的字体颜色属性:      console.log(style.color);      //想获取h1的背景颜色属性:      console.log(style.backgroundColor);      //想获得h1的字体大小:      console.log(style.fontSize);//32px      //想把h1的字体大小改为64px      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.      //不能修改计算后的属性中的font-size属性。因为这些样式是计算后的，所以是只读的。    </script>  </body>  </html>  运行结果： |

(5). 问题: 实际开发中修改一个元素的样式，可能同时修改多个css属性。如果用.style一个属性一个属性的修改，代码会很繁琐。



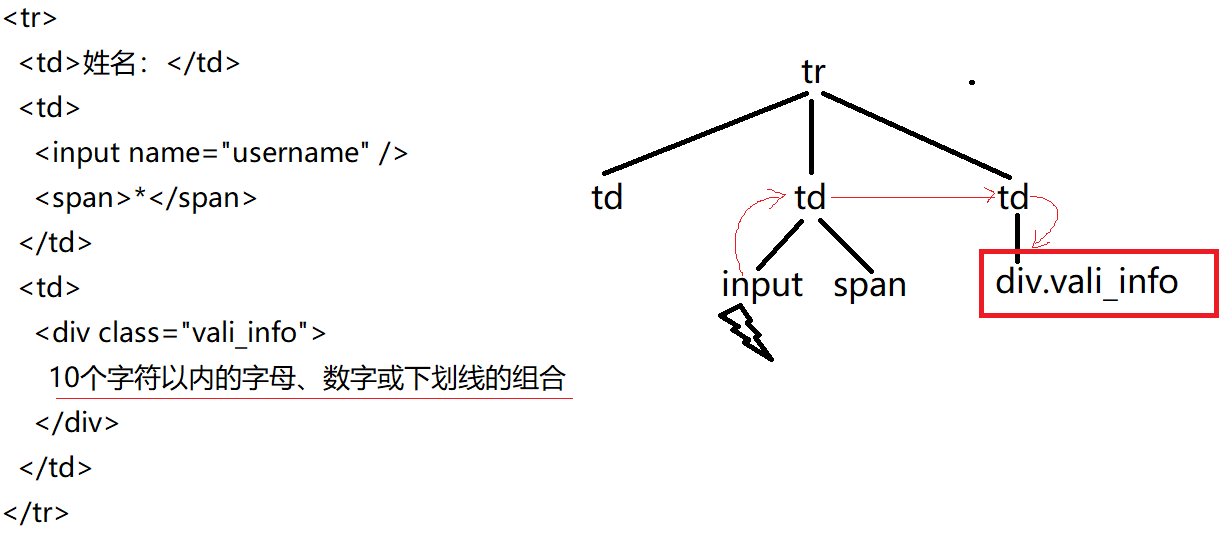
(6). 解决: 通过修改class，批量修改一个元素的多个css属性。——划算

a. 先在css中定义好不同场景下的样式类预案

b. 再在js中只通过`元素.className="样式类名"`一句话就可批量更换元素的多个css属性值。



(7). 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      }      /\*IE\*/      td:first-child+td {        width: 200px      }      td span {        color: red      }      .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;      }      /\*如果验证文本框中的内容通过，则修改文本框旁边的div的class为vali\_success，文本框旁边的div就变成了验证通过的样式！\*/      .vali\_success {        background-image: url("../images/ok.png");        padding-left: 20px;        width: 0px;        height: 20px;        overflow: hidden;      }      /\*当验证文本框的内容没通过，则修改文本框旁边的div的class为vali\_fail，就能让文本框旁边的div变成验证失败的样子\*/      .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>      //DOM 4步      //1. 查找触发事件的元素      //本例中: 姓名文本框失去焦点时触发验证      var inputName=document.getElementsByName("username")[0];//复习第一天查找元素      //2. 绑定事件处理函数      //当姓名文本框失去焦点时，自动执行      inputName.onblur=function(){        //3. 查找要修改的元素        //本例中: 要修改当前文本框的爹的下一个兄弟的第一个孩子        var div=this.parentElement                    .nextElementSibling                    .children[0];        //4. 修改元素        //先定义验证姓名的正则表达式        var reg=/^\w{1,10}$/;        //如果用正则表达式验证当前姓名文本框的内容通过        if(reg.test(this.value)==true){          //就修改div的class为验证通过的样式类名          div.className="vali\_success";        }else{//否则如果验证不通过          //就修改div的class为验证不通过的样式类名          div.className="vali\_fail";        }      }    </script>  </body>  </html>  运行结果: |

二. 添加删除替换元素:

对应小程序视频: 小程序->在线->JSCORE->day03: 2 DOM 添加新元素...

1. 添加新元素: 3步:

(1). 创建一个新的空元素对象:

a. var 新元素=document.createElement("标签名");

创建 元素

b. 意为: 在内存中创建一个新的空元素对象

c. 比如: 想创建一个a元素：

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

结果：

内存中: <a></a>

(2). 为新元素设置必要的属性: 新元素.属性名="属性值"

比如: 想让a变成一个可看见可跳转的超链接元素  
 a.innerHTML="go to tmooc"

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

结果:

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

(3). 将新元素添加到DOM树:

a. 问题: 用createElement创建的新元素对象，暂时是游离在内存中，并没有被加入到DOM树。而浏览器显示哪些元素，完全依照DOM树的内容来生成。所以，新元素不会被浏览器加载到页面上，用户暂时是看不到的！

b. 解决: 今后只要创建新元素，最后一步都是将新元素添加到DOM树上指定的父元素下。

c. 如何: 3种:

1). 在一个指定的父元素下所有直接子元素结尾追加新元素；

父元素.appendChild(新元素)

追加 孩子

2). 插入到一个父元素下的某个现有元素之前:

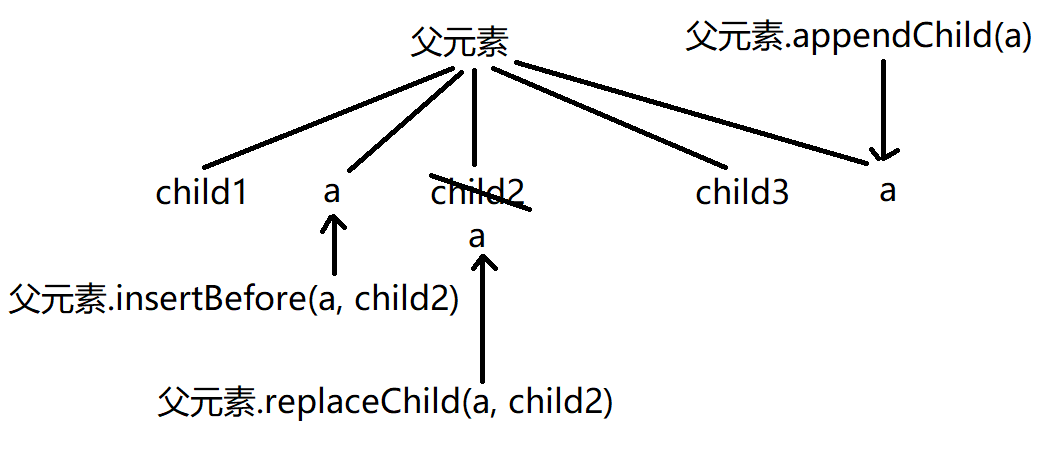
父元素.insertBefore(新元素, 现有元素)

插入(到)之前

3). 用新元素替换指定父元素下的某个现有元素

父元素.replaceChild(新元素, 现有元素)

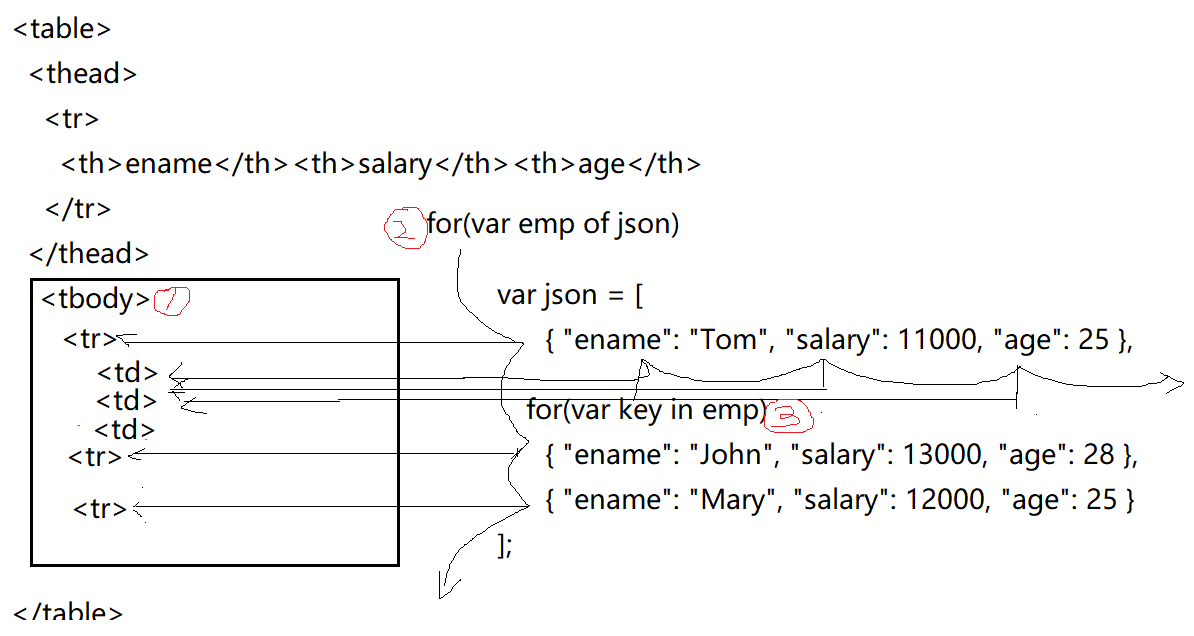
替换 孩子



2. 示例: 动态创建a元素和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>      //想创建一个a元素      var a=document.createElement("a");      //设置a的关键属性      a.innerHTML="go to tmooc";      a.href="http://tmooc.cn";      console.log(a);      //将a追加到body的结尾      document.body.appendChild(a);      //还想创建一个普通的文本框，放在a之前/之后/替换a      var input=document.createElement("input")      // document.body.insertBefore(input,a);      // document.body.appendChild(input);      document.body.replaceChild(input,a);    </script>  </body>  </html>  运行结果: |

3. 示例: 动态生成表格内容:  


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>ename</th>            <th>salary</th>            <th>age</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 }      ];        //向table中添加一个tbody      var tbody=document.createElement("tbody");        //遍历json数组中每个员工对象      for(var emp of json){        //每遍历一个员工对象，就要创建tr，添加到tbody下        var tr=document.createElement("tr");        tbody.appendChild(tr);        //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，就创建一个td，添加到tr下          var td=document.createElement("td");          tr.appendChild(td);          //设置当前td的内容为当前员工对象emp的当前属性值          td.innerHTML=emp[key]; //复习js高级day04->2.3        }      }  //最后在查找table，将tbody一次性装入table中      //只需要一次重绘页面即可！      var table=document.querySelector("#data>table");  table.appendChild(tbody);    </script>  </body>  </html>  运行结果： |

4. 优化:

(1). 原理: 浏览器中显示什么元素，完全根据DOM树中的内容生成。只要程序执行过程中，修改了DOM树的内容，浏览器就被迫重新渲染整个界面。

(2). 问题: 如果频繁修改DOM树的内容，导致浏览器频繁重新渲染整个界面，将极大的降低页面的加载效率！

(3). 解决: 尽量减少操作DOM树的次数！还要保证内容不少！

a. 第一种情况: 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都添加到父元素中。最后再一次性将整个父元素加到DOM树上。

b. 第二种情况: 如果父元素已经在页面上了，想添加多个平级子元素，必须借助文档片段对象来优化。

1). 什么是文档片段对象: 在内存中临时保存多个平级子节点的虚拟父节点对象

2). 如何: 3步:

i. 先创建文档片段(fragment)对象

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

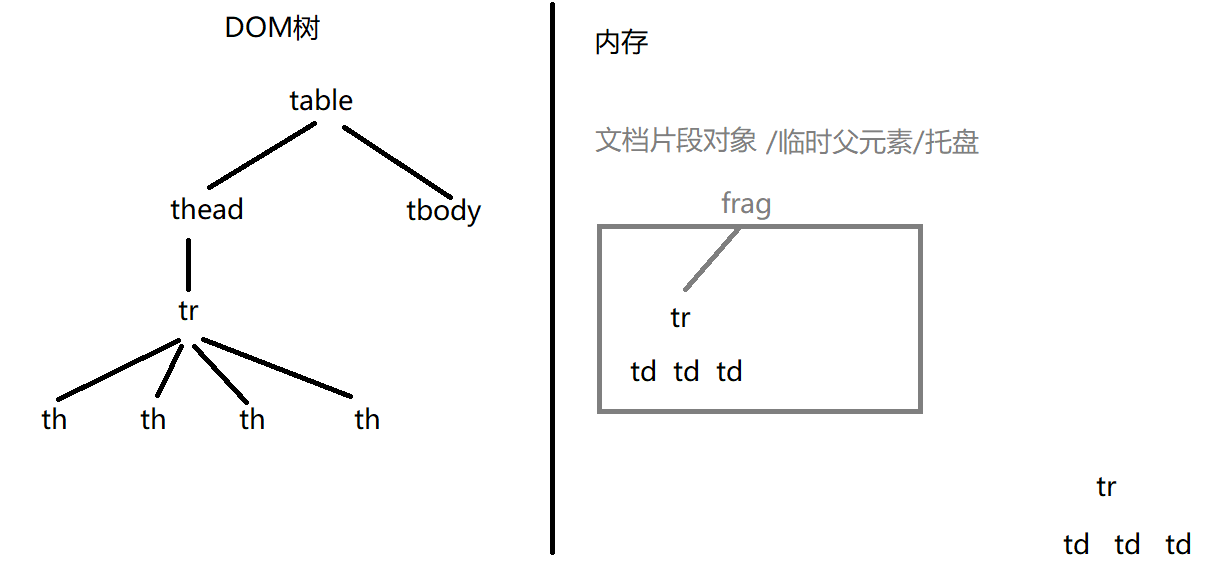
创建 文档 片段

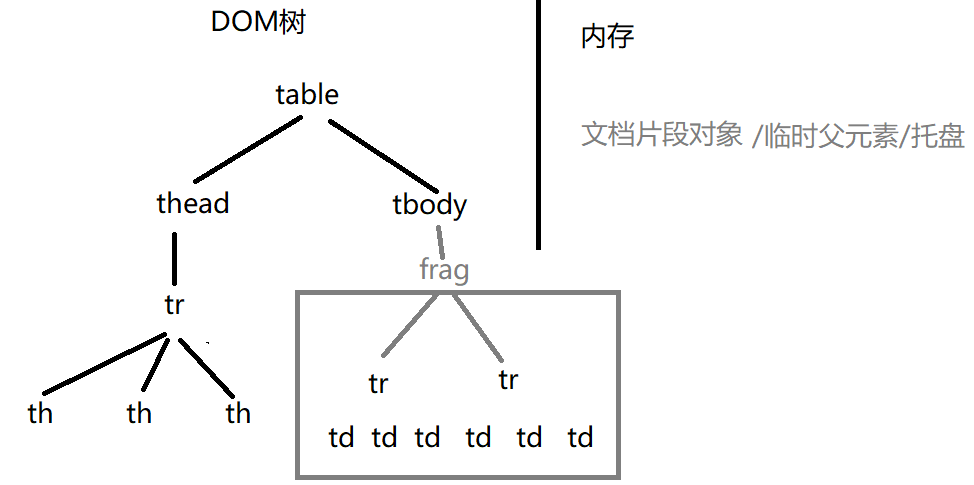
ii. 再将多个平级子元素添加到文档片段对象中

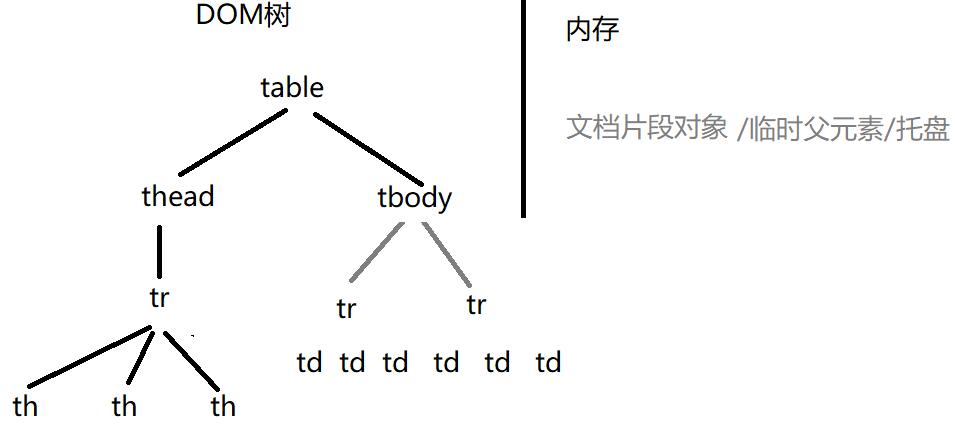
文档片段对象.appendChild(子元素)

iii. 最后将整个文档片段对象一次性添加到DOM树中父元素下

父元素.appendChild(文档片段对象)

3). 强调: 虽然加入文档片段的子元素，以appendChild方式添加到文档片段。但是，当文档片段将所有子元素送到DOM树上指定父节点下之后文档片段对象就释放了！不会成为DOM树上一级真实的节点！  






4). 示例: 动态生成表格内容(tbody已经在页面上)

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>ename</th>            <th>salary</th>            <th>age</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 }      ];      //先创建一个文档片段对象      var frag=document.createDocumentFragment();        //遍历json数组中每个员工对象      for(var emp of json){        //每遍历一个员工对象，就要创建tr，添加到文档片段对象下        var tr=document.createElement("tr");        frag.appendChild(tr);        //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，就创建一个td，添加到tr下          var td=document.createElement("td");          tr.appendChild(td);          //设置当前td的内容为当前员工对象emp的当前属性值          td.innerHTML=emp[key]; //复习js高级day04->2.3        }      }      //查找tbody      var tbody=document.querySelector("#data>table>tbody")      //将文档片段对象一次性添加到tbody中      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: 对常用元素的简化操作

1. <img>元素: 唯一的简化，就是创建img时:

(1). var img元素=new Image();

(2). 旧js中: var img元素=document.createElement("img");

(3). 强调: 常用的DOM元素中，只有img元素能用new创建。其余元素都只能用document.createElement()创建。

2. <table>元素: 逐级管理的方式

对应小程序视频: 小程序->在线->JSCORE->day03: 3 HTML DOM table

(1). table管着行分组:

a. table可以创建行分组

1). var thead=table.createTHead(); //相当于旧js两句话

旧js: var thead=document.createElement("thead");//即创建

table.appendChild(thead);//又添加

2). var tbody=table.createTBody(); //相当于旧js两句话

3). var tfoot=table.createTFoot(); //相当于旧js两句话

b. table可以删除行分组

1). table.deleteTHead();

2). table.deleteTFoot();

c. table可以获取行分组

1). table.tHead

2). table.tFoot

3). table.tBodies[i]

因为HTML标准中规定，其实一个table中可以有多个tbody。HTML DOM就将多个tbody集中保存在一个table.tBodies集合中。通过下标方式，访问某一个tBody。正是因为一个table下可以有多个tBody，所以table是没后deleteTBody()函数的，因为不知道该删除哪一个。

(2).行分组管着行:

a. 行分组可以添加行:

1). var tr=行分组.insertRow(i); //一个函数干2件事

2). 意为: 先创建一个新行对象，再将新行对象添加到行分组中i位置。

3). 大概相当于旧js中:

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

行分组.appendChild(tr);

4). 固定套路:

i. 在当前行分组末尾追加新行: var tr=行分组.insertRow()//默认结尾

ii. 在当前行分组开头插入新行: var tr=行分组.insertRow(0)

b. 行分组可以删除行:

1). 行分组.deleteRow(i)

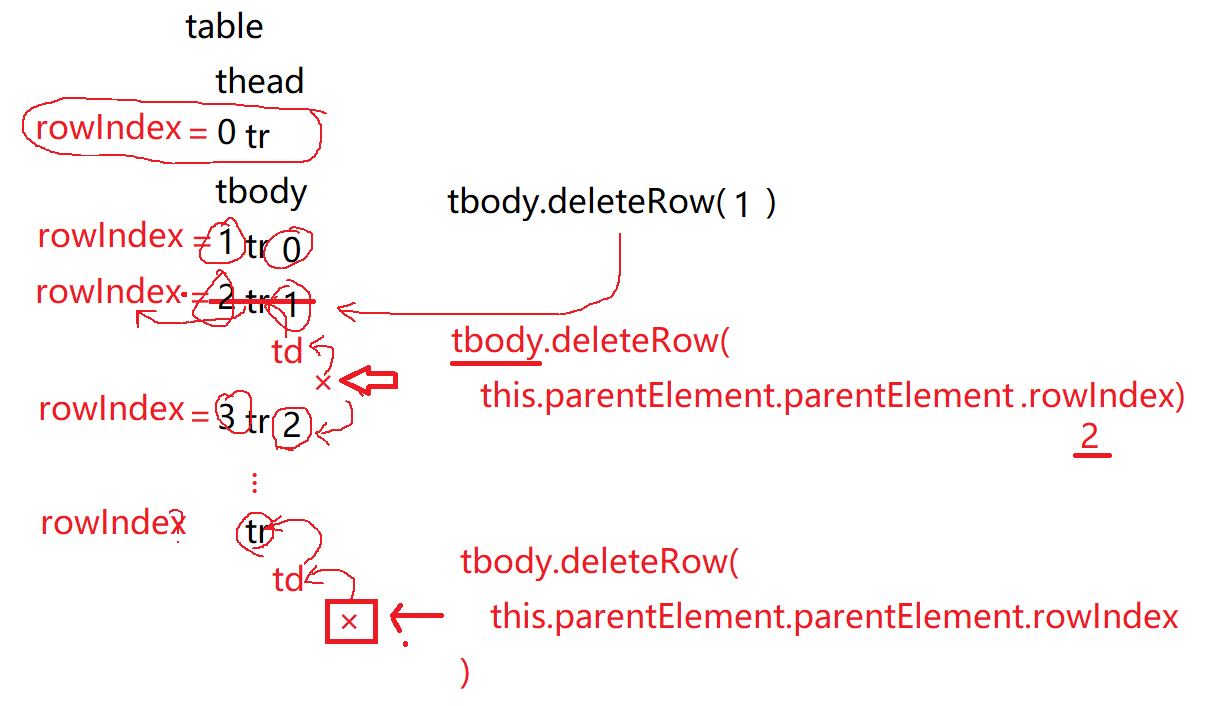
2). 意为: 删除当前行分组内的第i行

3). 坑: 如果deleteRow.前使用行分组删除行时，则下标必须使用行在行分组内的相对下标位置！

4). 问题: 如果用户随便点击一行中的删除按钮，我们虽然可以根据按钮逐级向上找到父元素tr对象。但是，依然不知道用户点的这一行在整个行分组内是第几行。如果不知道下标，就无法删除当前行。

5). 解决: 其实每个行对象身上都自动就有一个属性rowIndex。记录着当前行对象在整个表中的下标位置。

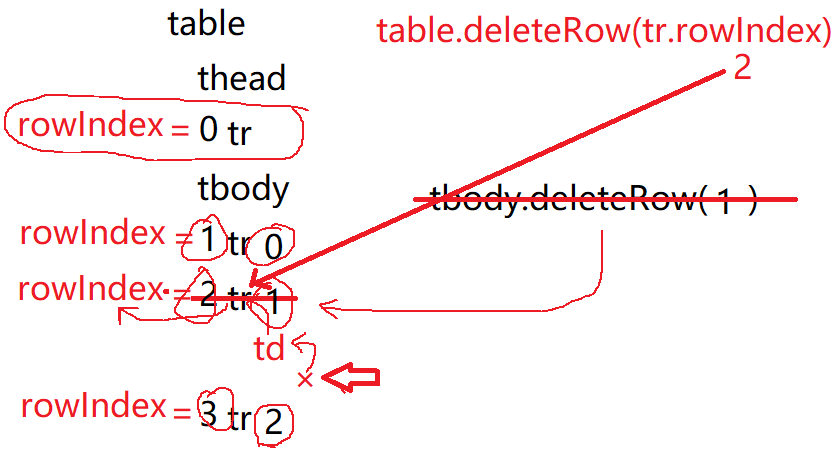
6). 坑: 行对象的rowIndex属性记录的是行在整个表中的下标位置，不是在行分组内的相对位置。而受到表头行的影响，表体中的行的rowIndex值比行在行分组内的相对位置都要大！如果使用行分组作为主语，使用rowIndex作为删除行的参数值，结果实际删除的行一定是原本要删除行的下一行！



7). 解决: 今后要想删除任意一行，不要用行分组作为主语，而应该用table作为主语:

table.deleteRow(tr.rowIndex)

因为主语变成了table，所以参数值刚好也是行在整个table中的下标位置，参数与主语配套了！



c. 行分组可以获取行

行分组.rows[i] //行分组下的所有行对象都集中保存在名为rows的集合中

(3). 行管着格

a. 行可以添加格:

1). var 格对象=行.insertCell()//末尾插入！

插入 格

2). 意为: 先创建一个td对象，再将td对象追加到行中

b. 行可以删除格:

行.deleteCell(i) //删除行中下标为i的一个单元格

c. 行可以获取格:

行.cells[i] //行中的所有格对象已经被提前保存在了名为cells的集合中

(4). 示例: 使用HTML DOM简化动态生成表格，并实现删除行

1\_createTable3\_delete.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>ename</th>            <th>salary</th>            <th>age</th>            <th></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 }      ];      var table=document.querySelector("#data>table");      //创建tbody      var tbody=document.createElement("tbody");        //遍历json数组中每个员工对象      for(var emp of json){        //每遍历一个员工对象，就要创建tr，添加到tbody下        //每次都在末尾追加新行        var tr=tbody.insertRow();        //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，就创建一个td，添加到tr下          // var td=document.createElement("td");          // tr.appendChild(td);          var td=tr.insertCell();          //设置当前td的内容为当前员工对象emp的当前属性值          td.innerHTML=emp[key]; //复习js高级day04->2.3        }        //除了遍历当前对象的每个属性添加的数据格之外，额外再多添加一个格        // var td=document.createElement("td");        // tr.appendChild(td);        var td=tr.insertCell();        //每个td中要再添加一个按钮        var btn=document.createElement("button");        btn.innerHTML="×";        td.appendChild(btn);        //为每个删除按钮绑定单击事件        btn.onclick=function(){          //根据当前点的按钮逐级找到当期要删除的行          var tr=this.parentElement //td                     .parentElement //tr          //获取当前行中第一个td的内容          // var ename=tr.children[0].innerHTML;          var ename=tr.cells[0].innerHTML;          //先跟用户确认是否删除          var result=confirm(`是否继续删除 ${ ename } 吗？`);          //只有用户点确定，确认继续删除时，才执行删除操作          if(result==true){            //删除table中一行的标准做法            table.deleteRow(tr.rowIndex);          }//否则如果用户点取消，说明用于反悔了！就什么也不做！        }      }      //最后在查找table，将tbody一次性装入table中      //只需要一次重绘页面即可！        table.appendChild(tbody);    </script>  </body>  </html>  运行结果: |

3. <form>元素:

对应小程序视频: 小程序->在线->JSCORE->day03: 作业: 带class样式的表单验证valiWithCss

(1). 获取<form>时: document已经将网页中的所有<form>元素，都集中保存在了document.forms集合中，可通过下标方式访问网页中某一个<form>

document.forms[i]

(2). 还可以进一步简化获取<form>内的各种表单元素:

a. 标准: form对象已经将表单中所有的表单元素都集中保存在了名为elements的集合中。

var 一个表单元素=form对象.elements[i/name/id]

b. 更简写: 如果要找的表单元素有name属性，可以直接

form对象.name名

(3). 自动获得焦点: 表单元素.focus()

(4). 示例: 使用HTML DOM简写带样式的表单验证:

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      }      /\*IE\*/      td:first-child+td {        width: 200px      }      td span {        color: red      }      .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;      }      /\*如果验证文本框中的内容通过，则修改文本框旁边的div的class为vali\_success，文本框旁边的div就变成了验证通过的样式！\*/      .vali\_success {        background-image: url("../images/ok.png");        padding-left: 20px;        width: 0px;        height: 20px;        overflow: hidden;      }      /\*当验证文本框的内容没通过，则修改文本框旁边的div的class为vali\_fail，就能让文本框旁边的div变成验证失败的样子\*/      .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>      //想获得网页中唯一的一个form      //旧DOM:      //var form=document.getElementById("form1");      //HTML DOM:      var form=document.forms[0];      console.log(form);      //DOM 4步      //1. 查找触发事件的元素      //本例中: 姓名文本框失去焦点时触发验证      // var inputName=document.getElementsByName("username")[0];//复习第一天查找元素      // var inputName=form.elements["username"];      var inputName=form.username;      //希望页面一加载，姓名文本框就自动获得焦点！      inputName.focus();      //2. 绑定事件处理函数      //当姓名文本框失去焦点时，自动执行      inputName.onblur=function(){        //3. 查找要修改的元素        //本例中: 要修改当前文本框的爹的下一个兄弟的第一个孩子        var div=this.parentElement                    .nextElementSibling                    .children[0];        //4. 修改元素        //先定义验证姓名的正则表达式        var reg=/^\w{1,10}$/;        //如果用正则表达式验证当前姓名文本框的内容通过        if(reg.test(this.value)==true){          //就修改div的class为验证通过的样式类名          div.className="vali\_success";        }else{//否则如果验证不通过          //就修改div的class为验证不通过的样式类名          div.className="vali\_fail";        }      }    </script>  </body>  </html>  运行结果: |

\*\*\*\*\*BOM\*\*\*\*\*

一. 什么是BOM: Browser Object Model

浏览器 对象 模型

1. 什么是BOM: 专门操作和访问浏览器窗口和软件信息的一组对象和函数

2. 何时: 今后只要想获取浏览器软件和窗口相关的信息，都要用BOM

3. 问题: BOM没有标准！BOM都有非常严重的兼容性问题，用的越来越少了！

4. 包括:

window history location document navigator screen event

已学过

二. window对象: 在浏览器中有三个角色

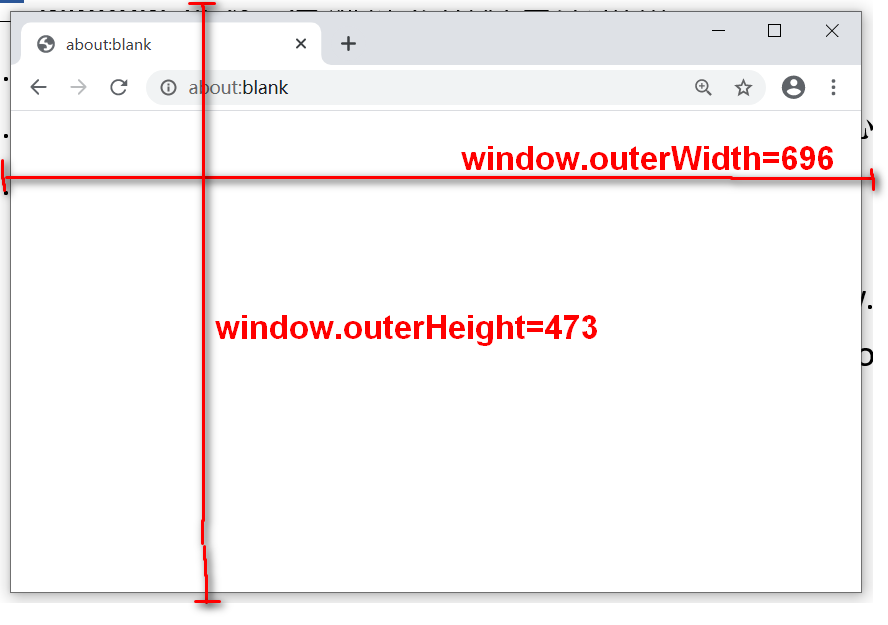
1. 代替ES标准中的global充当全局作用域

2. 包含了所有原生的js对象和函数: ECMAScript(核心语法)+DOM+BOM

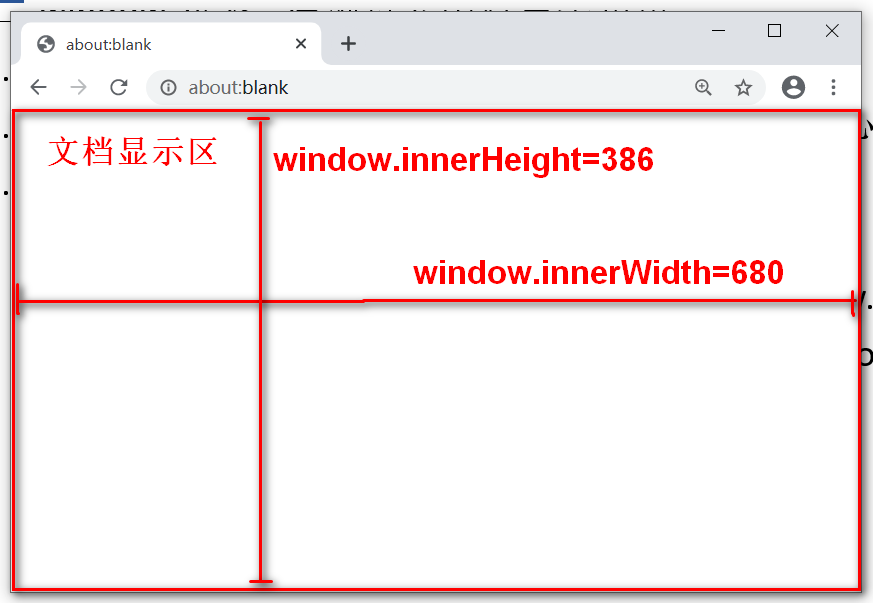
3. 代表当前正在打开的这个浏览器窗口:

(1). 获得窗口大小: 2组:

a. 完整窗口大小: window.outerWidth window.outerHeight



b. 文档显示区大小: window.innerWidth window.innerHeight



(2). window还可以打开和关闭窗口:

a. 打开新窗口: window.open()

b. 关闭当前窗口: window.close();

补: 浏览器三大对话框

1. 输入框: var 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=confirm("提示信息")

总结:

2. 修改元素: 3种东西可修改

(1). 修改内容: 3种内容可修改:

a. 获取或修改元素的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:

表单元素.value

(2). 修改属性: 3种

a. 字符串类型的HTML标准属性: 2种:

1). 旧核心DOM: 4个函数

i. 元素.getAttribute("属性名");

ii. 元素.setAttribute("属性名", "属性值")

iii. var bool=元素.hasAttribute("属性名")

iv. 元素.removeAttribute("属性名")

优点: 万能, 缺点: 繁琐

2). 新HTML DOM:

i. 元素.属性名

ii. 元素.属性名="属性值"

iii. 元素.属性名!==""

iv. 元素.属性名=""

优点: 简单, 缺点: 不万能

b. bool类型的HTML标准属性：

1). 不能用旧核心DOM4个函数修改

2). 只能用HTML DOM的"元素.属性名"方式获取或修改，且值为bool类型

c. 自定义扩展属性:

1). 何时: 2种

i. 代替id、class、元素等选择器作为查找触发事件的元素的条件

ii. 在客户端元素上临时缓存业务数据

2) HTML中: <元素 data-自定义属性名="属性值">

3). js中: 2种: （不能用.访问）

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

计算后的样式都是只读的

c. 批量修改元素的样式时，都用class:

元素.className="class名"

总结: 不要背英文名字！反而应该记中文能做哪些事儿！

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，2种:

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()

总结:

BOM:

1. window:

(1). 获得窗口大小:

a. 获得完整窗口大小:

window.outerWidth和window.outerHeight

b. 获得文档显示区大小:

window.innerWidth和window.innerHeight

(2). 打开和关闭窗口:

window.open()和window.close()

2. 打开新链接4种方式:

(1). 在当前窗口打开，可后退

a. html: <a href="url" target="\_self">

b. js: window.open("url", "\_self");

(2). 在当前窗口打开，禁止后退

a. js: location.replace("新url")

(3). 在新窗口打开，可同时打开多个

a. html: <a href="url" target="\_blank">

b. js: window.open("url", "\_blank");

(4). 在新窗口打开，只能打开一个

a. html: <a href="url" target="自定义窗口名">

b. js: window.open("url", "自定义窗口名")

3. history:

(1). 前进: history.go(n)

(2). 后退: history.go(-n)

(3). 刷新: history.go(0)

4. location:

(1). 属性: 分段获得url中各个部分:

a. location.href 完整url

b. location.protocol 协议

c. location.host 主机名+端口号

d. location.hostname 主机名

e. location.port 端口号

f. location.pathname 相对路径

g. location.search ?及其之后的查询字符串参数列表

h. location.hash #锚点地址

(2). 方法:

a. 在当前窗口打开，可后退：

location.assign("新url") 或 location.href="新url"

b. 在当前窗口打开，禁止后退:

location.replace("新url")

c. 刷新: location.reload();

5. navigator

(1). 查看浏览器的名称和版本号: navigator.userAgent

(2). 查看浏览器中安装的插件列表: navigator.plugins

今日对应小程序视频列表:

小程序->在线->JSCORE->day03:

1 修改样式, 计算后的样式

2 DOM 添加新元素...

3 HTML DOM table 删除行 确认框

作业: 带class样式的表单验证valiWithCss

4. 打开新链接4种方式:

作业:

1. 复习今日小程序问题清单: 小程序->首页-> DOM ->day03

2. 看小程序视频实现倒计时关闭窗口

小程序->在线->DOM->day03 作业: 倒计时关闭窗口

3. 看小程序视频实现选择省份和城市，级联下拉列表

小程序->在线->DOM->day03 作业； 选择省份和城市，级联下拉列表

4. 预习明日小程序视频: 小程序->在线->DOM->day04