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->day03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->day03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->day03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，自己用人话把注释写回来！——才算自己会！

正课:

1. 修改样式

2. 添加/删除

3. HTML DOM常用对象

一. 修改样式: 2种:

1. 获取或修改元素的内联样式: 元素.style.css属性

2. 问题:

(1). 用style.css属性方式，只能修改内联样式，无法修改样式表中的样式

——不是大问题:

a. 因为本来就不应该擅自修改样式表中的样式，因为样式表中的样式都是多个元素共用的，一旦修改可能牵一发而动全身。

b. 当我们想修改一个元素的样式时，通常只希望修改这个元素的样式，不希望影响其他元素，且希望我们这次修改的样式优先级最高！所以，此时修改内联样式反而是好事儿！

(2). 但是style.css属性方式，也只能获得元素的内联样式——真正的大问题！

a. 今后为了便于维护，HTML和CSS都是分离的，所以几乎不会写内联样式！

b. 这就意味着用style.css属性几乎无法获得任何css属性值！只能获得极少数的写在内联样式中的css属性

3. 解决: 只要获取一个元素的完整的css属性都必须获得计算后(computed)的样式。

(1). 什么是计算后的样式: 最终应用到这个元素上的所有样式的总和。(F12->Elements->Computed)

(2). 何时: 今后只要获取元素的css属性值，都必须获得计算后的样式

(3). 如何: 2步:

a. 先获得完整的计算后的样式的集合对象: var style=getComputedStyle(元素对象)

获得 计算后的 样式对象

强调: getComputedStyle()函数是浏览器内置函数，可直接使用，不能改名

b. 从获得的样式集合对象style中获取计算后的某一个css属性值

style.css属性

(4). 计算后的样式，不仅收集这个元素的所有样式到一个对象中集中保存，而且还会将相对单位(em)以及不是数值的属性值，换算回绝对单位和数值方式。

(5). 所有通过计算后的样式获得css属性都是只读的！不能修改！因为计算后的样式来源不确定，一旦修改，很可能牵一发而动全身！

4. 总结:

(1). 今后只要修改一个元素的样式，都用: 元素.style.css属性=”属性值”

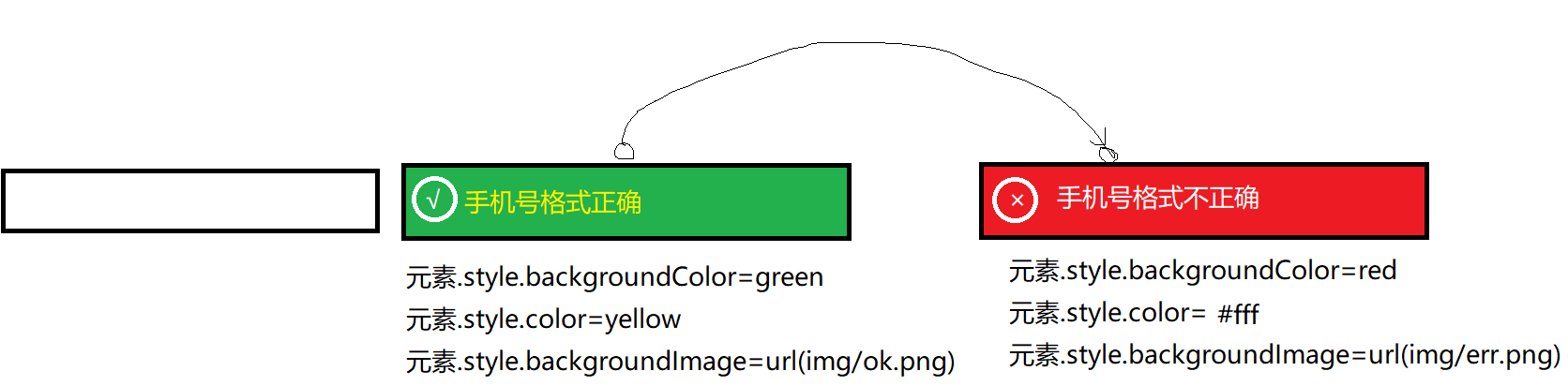
(2). 今后只要获取一个元素的样式，都用: var style=getComputedStyle(元素对象)

style.css属性

5. 示例: 分别使用style和getComputedStyle()操作元素的css属性

|  |
| --- |
| <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 dingding</h1>  <script>  var h1=document.getElementById("h1");  //先获得h1元素对象身上所有计算后的样式的集合  var style=getComputedStyle(h1);  console.log(style);  //想获取h1的内联样式color  //只有知道color是内联样式时，才能这么做  //console.log(h1.style.color);  //任何情况下都可:  console.log(style.color);  //想获得内部样式表中h1的background-color  //错误:  //console.log(h1.style.backgroundColor);  //正确:  console.log(style.backgroundColor);  //想获得浏览器默认样式表中的h1的font-size  //错误:  //console.log(h1.style.fontSize);  //正确:  console.log(style.fontSize);  //修改h1的背景颜色:  //修改h1的内联样式：正确！  h1.style.backgroundColor="green";  //                     ↓  //结果: <h1 style="...;background-color:green"> 优先级最高！  //修改计算后的css属性: 错误！报错！  style.backgroundColor="pink";  </script>  </body> |

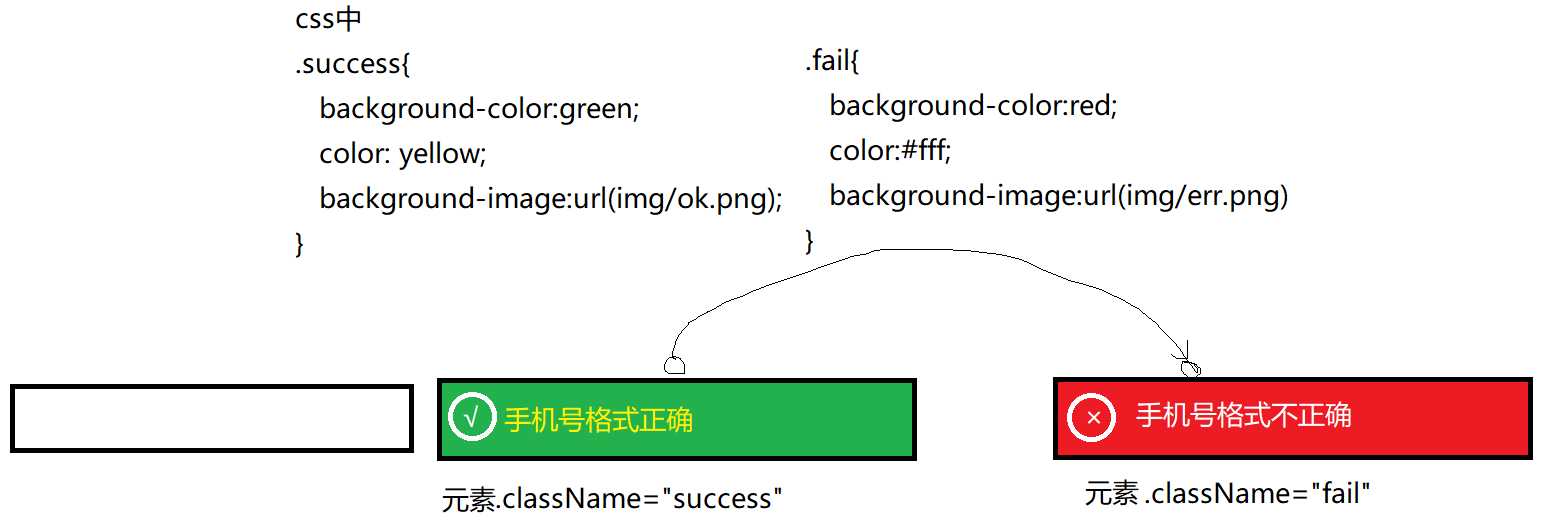
6. 问题: 使用元素.style.css属性一句话只能修改一个css属性，如果在一次效果变化中，需要同时修改这个元素的多个css属性，代码会很繁琐！且效率低！



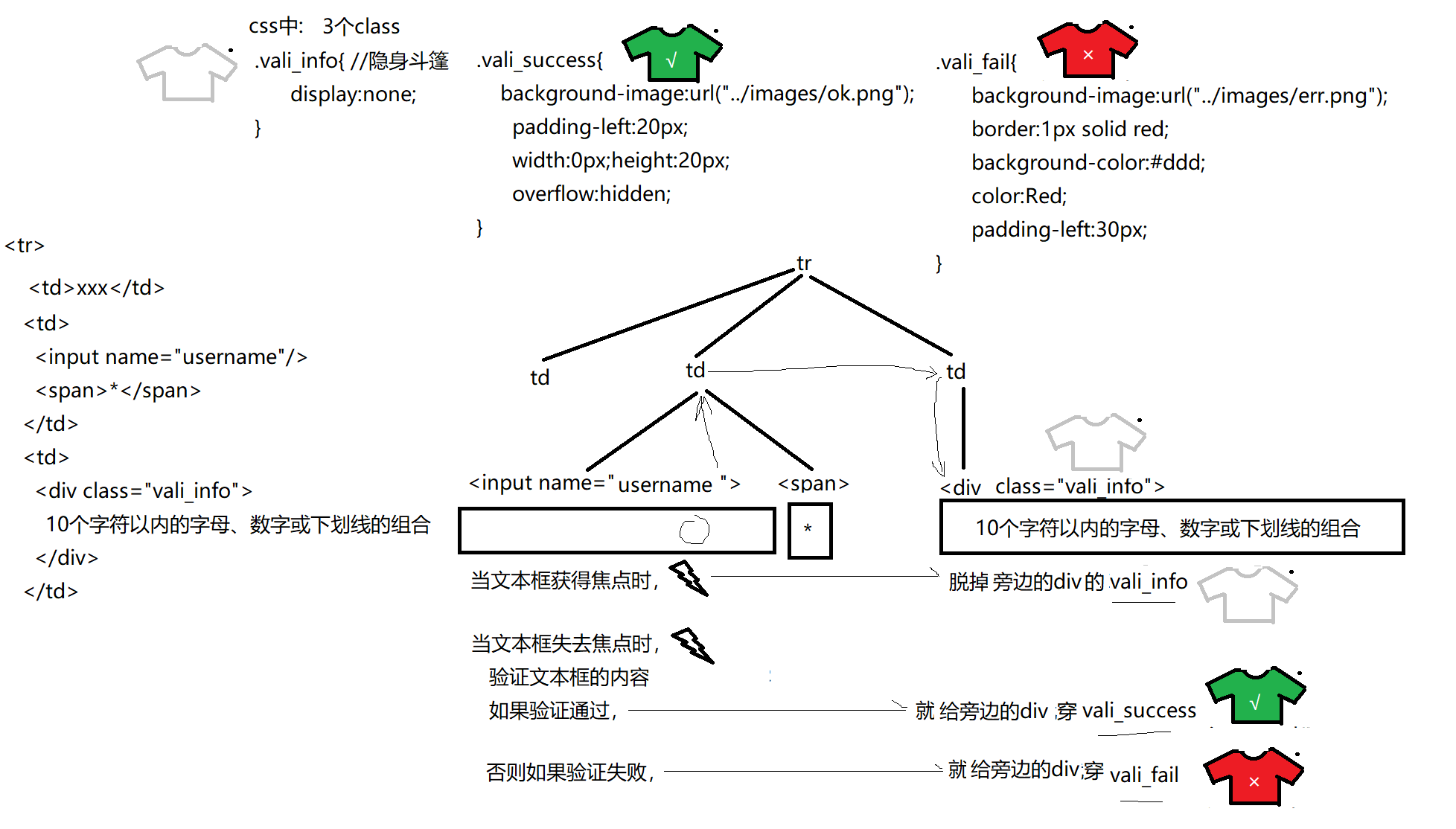
7. 解决: 用操作一个class，代替操作多个css属性

(1). 在css中提前为提示框定义正确时的class和错误时的class

(2). 当要修改元素的样式时，只要修改元素的className属性，就可批量应用一套样式



8. 示例: 带样式的表单验证



|  |
| --- |
| <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}  .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;  }  .vali\_success{      background-image:url("../images/ok.png");    padding-left:20px;    width:0px;height:20px;    overflow:hidden;  }  .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>  //先做文本框获得焦点时，显示提示  //DOM 4步  //1. 查找触发事件的元素  //本例中: 查找name属性=username的一个文本框元素  var txtName=document.getElementsByName("username")[0]  //2. 绑定事件处理函数  //      当获得焦点时  txtName.onfocus=function(){    //this->当前正在触发事件的当前文本框对象    //3. 查找要修改的元素    //本例中: 查找当前文本框的爹的下一个兄弟的第一个孩子    this.parentNode.nextElementSibling.children[0]    //4. 修改元素    //本例中: 脱掉隐身斗篷vali\_info    .className=""  }  //再做文本框失去焦点时，验证文本框内容并修改提示信息  //DOM 4步  //1. 查找触发事件的元素(已找到txtName)  txtName  //2. 绑定事件处理函数  //当失去焦点时  .onblur=function(){    //this->当前正在触发事件的当前文本框对象    //3. 查找要修改的元素    var div=this.parentNode.nextElementSibling.children[0]    //4. 修改元素    //先定义正则表达式描述文本框的规则    var reg=/^\w{1,10}$/;    //先验证当前文本框的内容是否符合规则    var result=reg.test(this.value.trim());    //基础一般的同学:    if(result==true){//如果验证通过      //就修改旁边div的class为vali\_success      div.className="vali\_success";    }else{//否则如果验证不通过      //就修改旁边div的class为vali\_fail      div.className="vali\_fail";    }    //基础好的同学:    //div.className=result?"vali\_success":"vali\_fail";  }  </script>  </body> |

二. 添加删除元素:

1. 添加新元素: 3步:

(1). 先创建一个空元素:

a. var 元素对象=document.createElement(“元素名”);

创建 元素

b. 比如: 想创建一个a元素: var a=document.createElement(“a”)

c. 结果: <a></a>

(2). 为新元素添加关键属性

a. 比如: 一个a要想当做超链接正常使用，至少要加2个属性: href和内容

a.href=”http://tmooc.cn”;

a.innerHTML=”go to tmooc”

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

c. 问题: 这个a只是游离在内存中，没有加入DOM树，浏览器就不知道有这个a，也不知道应该放在那个位置，所以，用户暂时看不到这个a

(3). 将新元素添加到DOM树上：

a. 为什么: 只有将新元素添加到DOM树，浏览器才知道多了一个元素，才会在元素添加的位置，绘制出这个元素，让人看到。

b. 如何: 3种:

1). 追加到一个父元素下所有子元素末尾: 父元素.appendChild(新子元素)

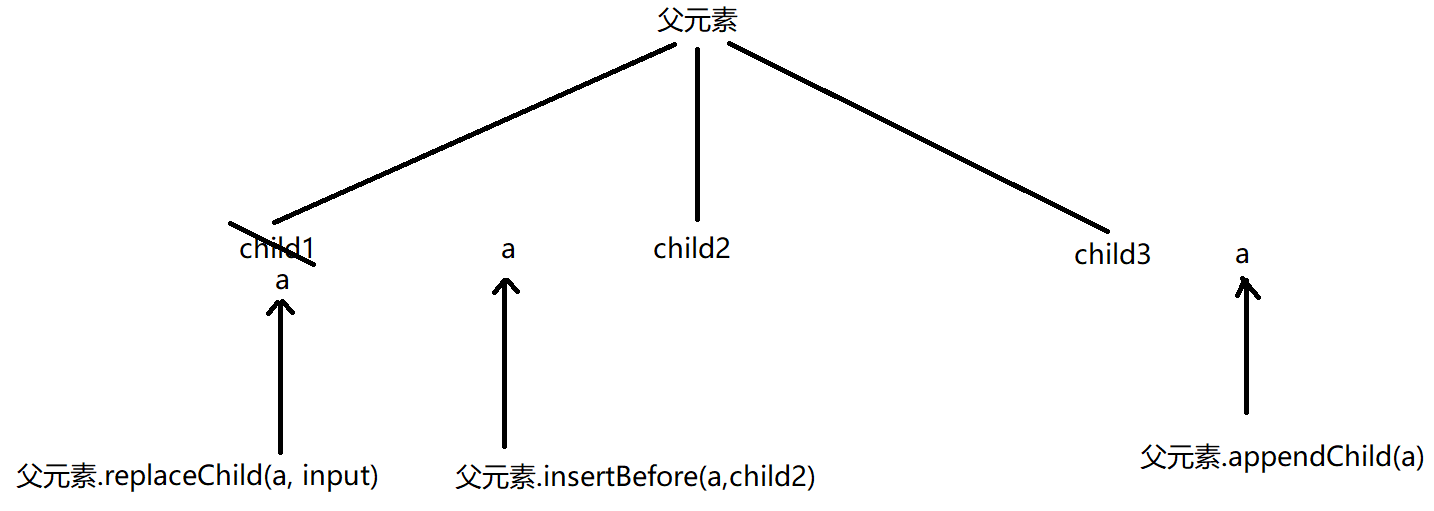
追加 孩子

2). 插入到一个父元素下的现有的一个子元素之前: 父元素.insertBefore(新子元素, 现有子元素)

插入(到)之前

3). 替换一个父元素下现有的一个子元素: 父元素.replaceChild(新子元素, 现有子元素)

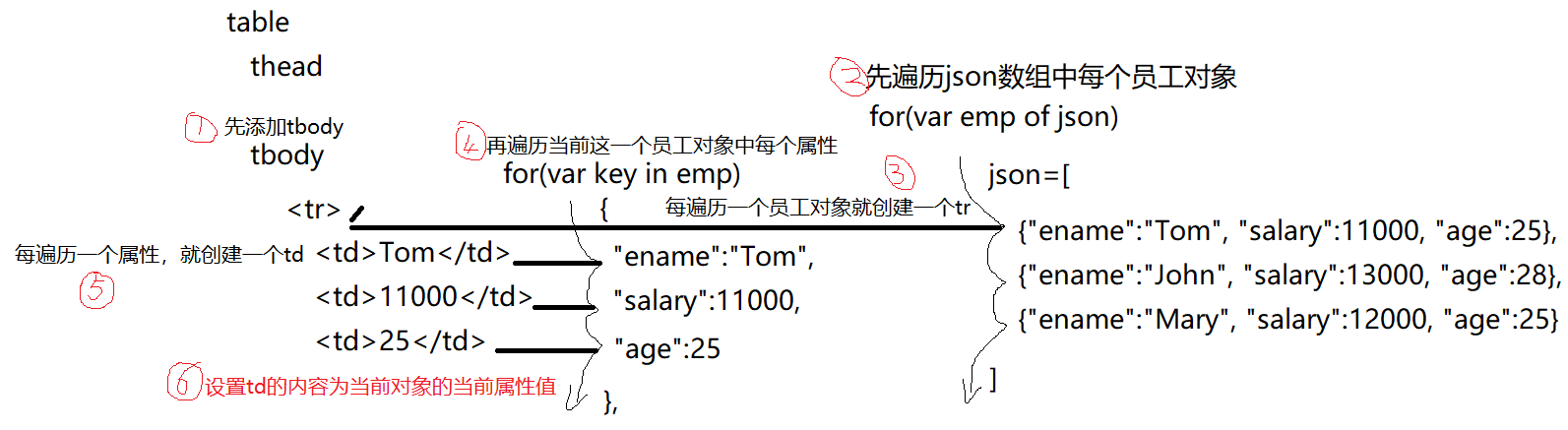
替换 孩子



2. 示例: 尝试添加一个a元素和一个input元素

|  |
| --- |
| <body>  <script>  //想创建一个a元素，添加到body中  //1. 创建一个空元素a  var a=document.createElement("a");  //2. 为新的元素添加必要属性  a.href="http://tmooc.cn";  a.innerHTML="go to tmooc";  console.log(a);  //3. 追加到DOM树上指定父元素下  document.body.appendChild(a);  //又想创建一个文本框:  var input=document.createElement("input");  //希望文本框出现在a之前  //document.body.insertBefore(input,a);  //希望文本框出现在a之后  //document.body.appendChild(input)  //希望文本框替换a  //document.body.replaceChild(input,a)  </script>  </body> |

3. 示例: 动态生成表格



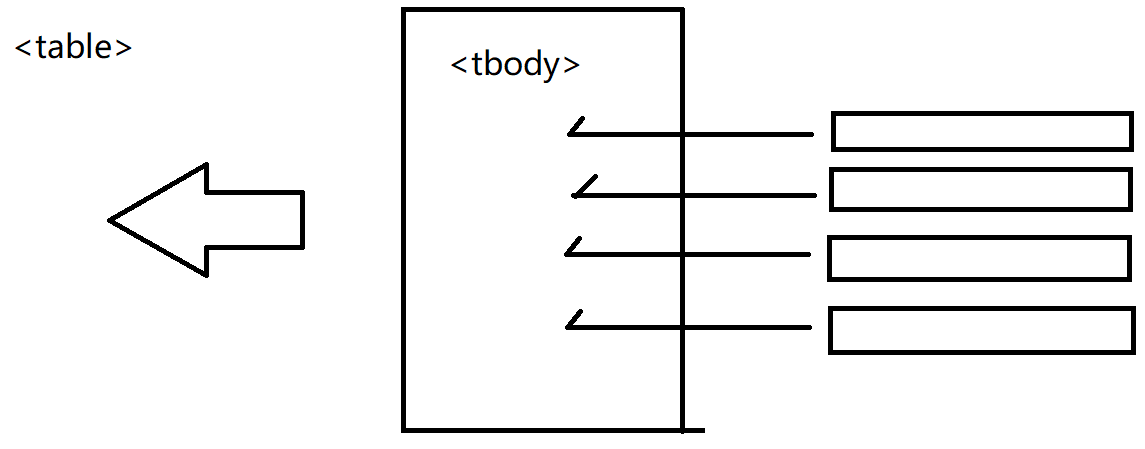
|  |
| --- |
| <head>  <title>动态创建表格</title>  <meta charset="utf-8" />  <style>    table{width:600px; border-collapse:collapse;      text-align:center;    }    td,th{border:1px solid #ccc}  </style>  </head>  <body>  <div id="data">    <!-- table>thead>tr>th\*3 -->    <table>      <thead>        <tr>          <th>员工姓名</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}  ];  //先找到table  var table=document.querySelector("#data>table");  //先动态创建tbody  var tbody=document.createElement("tbody");  //暂时不将tbody添加到table中  //再遍历json数组中每个员工对象  for(var emp of json){    //每遍历一个员工对象，就创建一个tr，并添加到tbody中    var tr=document.createElement("tr");    tbody.appendChild(tr);    //遍历当前员工对象中每属性    for(var key in emp){      //每遍历一个属性就创建一个td加入tr中      var td=document.createElement("td");      tr.appendChild(td);      //设置当前td的内容为当前员工对象的当前属性的属性值      td.innerHTML=emp[key];    }  }  //当整个表格内容填充完成后，最后再将tbody添加到table中——浏览器只需要重绘一次！  table.appendChild(tbody);  </script>  </body> |

4. 优化: 尽量减少操作DOM树的次数

(1). 为什么: 因为每修改一次DOM树，浏览器都会重新绘制一次页面。如果频繁修改DOM树，浏览器只能频繁绘制页面，导致页面加载效率降低。

(2). 如何:

a. 如果同时添加父元素和子元素时，应该先在内存中将所有子元素添加到父元素中，然后再将父元素一次性添加到DOM树上



b. 如果父元素已经在页面上了，我们希望同时添加多个平级子元素，应该借助于文档片段对象

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

2). 何时: 父元素已经在页面上了，希望同时添加多个平级子元素，都要用文档片段

3). 如何: 3步:

i. 先创建一个文档片段对象

var frag=document.createDocumentFragment();

创建 文档 片段

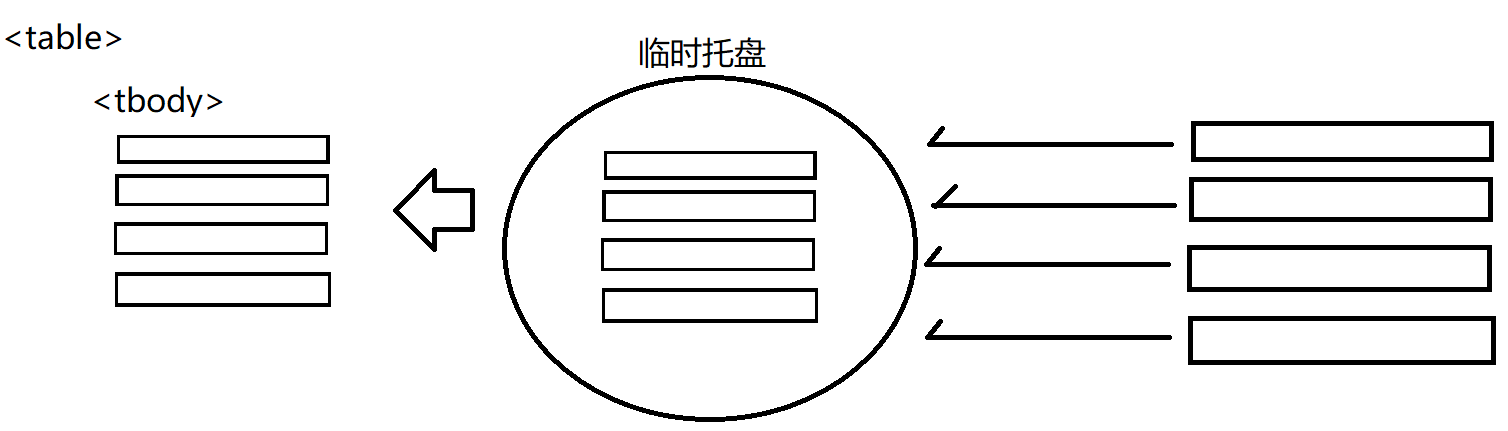
ii. 将子元素添加到文档片段对象中

frag.appendChild(新子元素)

iii. 将文档片段对象一次性添加到DOM树

父元素.appendChild(frag);

4). 文档片段对象添加到DOM树后，自动释放，不会成为页面上真正的一级元素。不影响原来的子元素和父元素的关系！



(3). 示例: 使用文档片段动态生成表格:

|  |
| --- |
| <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>员工姓名</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}  ];  //先创建一个文档片段对象  var frag=document.createDocumentFragment();  //再遍历json数组中每个员工对象  for(var emp of json){    //每遍历一个员工对象，就创建一个tr，并临时添加到frag中    var tr=document.createElement("tr");    frag.appendChild(tr);    //遍历当前员工对象中每属性    for(var key in emp){      //每遍历一个属性就创建一个td加入tr中      var td=document.createElement("td");      tr.appendChild(td);      //设置当前td的内容为当前员工对象的当前属性的属性值      td.innerHTML=emp[key];    }  }  //查找table中的tbody  var tbody=document.querySelector("#data>table>tbody");  //将文档片段一次性添加到tbody中  tbody.appendChild(frag);  </script>  </body> |

5. 删除元素: 父元素.removeChild(子元素对象)

三. HTML DOM常用对象: (了解)

|  |
| --- |
| HTML DOM不仅对访问属性进行了简化，而且还对个别比较常用的复杂的HTML元素提供了一批简化的函数和属性。 |

1. <img>元素: 创建img元素时有简写: （下个阶段）

(1). 旧DOM: var img=document.createElement(“img”)

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

强调: new在DOM中不可推广。常用的只有img元素和option元素才能new！别的元素老老实实用createElement()去！

2. <table>元素: 逐级管理

(1)table: 管着行分组: 3件事

a. 添加行分组:

1). 旧DOM中: 2步

var thead=document.createElement(“thead”);

table.appendChild(thead);

2). HTML DOM中: 1步

var thead=table.createTHead(); //做了2件事，既创建，又添加

3). 其实还包括:

var thead=table.createTHead();

var tbody=table.createTBody();

var tfoot=table.createTFoot();

b. 删除行分组:

table.deleteTHead()

table.deleteTFoot()

~~//没有table.deleteTBody()~~

c. 获取行分组

table.tHead //thead直接隶属于table

table.tFoot //tfoot直接隶属于table

table.tBodies[i] //说明: 1. 一个table中可以有多个tbody，2. 所有tbody都是放在数组中，再保存在table里，并不是直接隶属于table。

(2)行分组: 管着行

a. 添加行:

1). 旧DOM中: 2句

var tr=document.createElement(“tr”);

tbody.appendChild(tr);

2). HTML DOM中: 1句

var tr=tbody.insertRow() //不写参数，默认表示末尾追加

3). 其实insertRow(i)可以有一个下标i参数，i是几，就在第几行添加新行。

如果i位置已经有行，则新行添加到第i行，原第i行向后顺移一行，变为i+1行

4). 固定套路: 如果在开头插入一行: tbody.insertRow(0);

b. 删除行:

1).原本删除行: tbody.deleteRow(i) 删除tbody中i位置的行

2). 强调: 如果.前的主语是tbody，则删除行时必须使用tbody内的相对下标位置，不能用行在整个表中的下标位置。

3). 问题: 通常要删除的行，不确定是第几行！如何自动获得要删除的行的行下标呢？

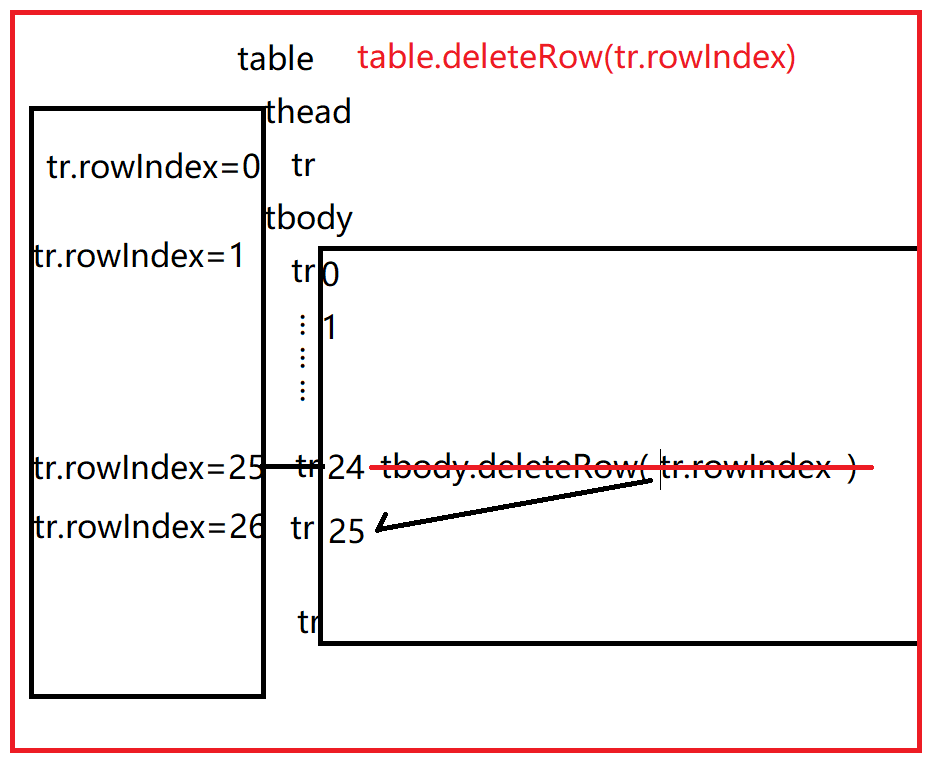
4). 解决: 其实每个行对象身上自动带着一个”rowIndex”属性，该属性自动记录该行在整个表中的下标位置

5). 问题: 但是tbody.deleteRow(i)需要的不是行在整个表中的下标，而是行在自己内部的相对下标位置。因为有表头行的干扰，所以一个行对象在整个表中的下标(tr.rowIndex)肯定都会比这一行在tbody内的相对下标位置大！结果，tbody.deleteRow()中配tr.rowIndex，删除的绝对不是tr行。而是tr的下方某一行！

6). 解决: 其实table对象也能越级直接删除行！且table做为.前的主语时，刚好需要的就是行在整个表中的下标位置，也就是tr.rowIndex

7). 总结: 今后如果想删除table中的一行！都应该用:

table.deleteRow(tr.rowIndex)



c. 获取行: 行分组.rows[i]

(3)行: 管着格

a. 添加格: var td=tr.insertCell() //末尾追加一个新格——常用

插入 格

//也可以加下标i参数，但是很少用。因为行中间插入一个格，会导致行中的右侧剩余格向右移动，列就错误

b. 删除格: tr.deleteCell() //用的也很少，因为行中间删除一个格，会导致行中的右侧剩余格向左移动，列也错位！

c. 获取格: tr.cells

|  |
| --- |
| 补充: 浏览器三大对话框之确认框  1. 浏览器三大对话框: alert prompt confirm  2. confirm(“提示消息”) 可弹出一个确认框  确认 3. 警告 vs 确认:  (1). 警告: 就是告知，且没有选择！知道也得知道！不知道也得知道！  所以，警告框只有一个”确定”按钮，没的选择，必须点！  (2). 确认: 询问，可以选择！  所以，确认框有两个按钮: 确认和取消  4. 如何知道用户点的是确认？还是取消呢？  var bool= confirm(“提示消息”);  当用户点确认时，返回true，当用户点取消时返回false！  5. 以上三大对话框因为兼容性问题和不可修改的界面，今后用的越来越少了！甚至绝迹了！  6. 今后我们所有的对话框一律用div+css自己弹出(控制显示隐藏) |

3. 表单元素:

(1). 获取<form>元素时: document已经将网页中所有<form>元素集中保存在了一个forms数组中，可通过下标方式访问forms中的一个表单元素:

var form=document.forms[i] 如果网页中只有一个<form>元素，则document.forms[0];

(2). 获取<form>内的表单元素时也有简化:

a. 旧DOM中: document.getElementsByName(“name名”)

或document.getElementsByTagName(“标签名”)

都不好用

b. HTML DOM中: form对象，已经提前把表单内所有的表单元素都集中保存在了elements集合中！

1). form.elements 包含这个表单中所有的表单元素（input textarea button select）

2). 进一步获得具体的表单元素:

i. 标准做法: var 一个表单元素=form.elements[i或name或id]

ii. 简化写法: 如果这个表单元素有name属性，可简写为: form.name名

(3). 让一个表单元素自动获得焦点: 表单元素.focus() ——今后用的多！

(4). 示例: 使用HTML简化表单操作:

|  |
| --- |
| <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}  .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;  }  .vali\_success{      background-image:url("../images/ok.png");    padding-left:20px;    width:0px;height:20px;    overflow:hidden;  }  .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="">            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>  //先获得form,页面中只有一个<form>元素  var form=document.forms[0];  //想输出form中有哪些表单元素  console.log(form.elements);  //DOM 4步  //1. 查找触发事件的元素  //本例中: 查找name属性=username的一个文本框元素  var txtName=//form.elements["username"];              form.username;  console.log(txtName);  //再做文本框失去焦点时，验证文本框内容并修改提示信息  //2. 绑定事件处理函数  //当失去焦点时  txtName.onblur=function(){    //this->当前正在触发事件的当前文本框对象    //3. 查找要修改的元素    var div=this.parentNode.nextElementSibling.children[0]    //4. 修改元素    //先定义正则表达式描述文本框的规则    var reg=/^\w{1,10}$/;    //先验证当前文本框的内容是否符合规则    var result=reg.test(this.value.trim());    //基础一般的同学:    if(result==true){//如果验证通过      //就修改旁边div的class为vali\_success      div.className="vali\_success";    }else{//否则如果验证不通过      //就修改旁边div的class为vali\_fail      div.className="vali\_fail";      //希望出错的文本框再次重新获得焦点！      this.focus();    }    //基础好的同学:    //div.className=result?"vali\_success":"vali\_fail";  }  </script>  </body> |

总结:

1. 查找元素: 4种:

(1). 不需要查找就可直接获得的对象：5个

a. document

b. <html> document.documentElement

c. <head> document.head

d. <body> document.body

e. <form> document.forms[i]

(2). 按节点间关系查找: 2大类关系 6个属性

何时: 如果已经获得一个元素，想找它周围附近的元素时

a. 父子关系:

1). 获得当前元素的父元素: 元素.parentNode

2). 获得当前元素的直接子元素: 元素.children

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

4). 获得当前元素下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.previousElementSibling

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

(3). 按HTML特征查找: 4个函数

何时: 只需要一个条件，就可找到想要的元素时

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=任意父元素.getElementsByName(“name名”)

(4). 按选择器查找: 2个函数

何时: 查找条件复杂时

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

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

|  |
| --- |
| 总结: 返回值:  1. 如果原函数返回下标位置i，找不到返回-1  2. 如果原函数返回一个对象或数组，找不到返回null  3. 如果原函数返回一个类数组对象，找不到返回空类数组对象{ length:0 } |

2. 修改元素: 3种

(1). 内容: 3种

a. 原始HTML内容: 元素.innerHTML

b. 纯文本内容: 元素.textContent

c. 表单元素的值: 表单元素.value

(2). 属性:3种:

a. 字符串类型的HTML标准属性: 2种:

1). 核心DOM 4个函数:

i. 获取属性值: 元素.getAttribute(“属性名”)

ii. 修改属性值: 元素.setAttribute(“属性名”,”新属性值”)

iii. 移除属性值: 元素.removeAttribute(“属性名”)

iv. 判断是否包含某个属性: 元素.hasAttribute(“属性名”)

2). HTML DOM: 元素.属性名

b. bool类型的HTML标准属性: 只能用: HTML DOM: 元素.属性名

c. 自定义扩展属性:

1). 何时: 2种:

i. 代替其他选择器查找专门触发事件的元素——避免HTML和CSS的修改影响js程序

ii. 在客户端页面中元素上临时缓存一些稍后要用的数据——减少向服务器发送请求的次数

2). 如何: 2种:

i. 核心DOM 4个函数: 元素.getAttribute()/setAttribute()/removeAttribute()/has Attribute()

ii. HTML5: 元素.dataset.属性名

说明: 只有带有data-前缀的自定义扩展属性才能用dataset操作

(3). 样式: 2种:

a. 修改内联样式: 元素.style.css属性

b. 获取元素css属性值时: var style=getComputedStyle(元素对象); style.css属性

c. 批量修改一个元素的多个css属性: 元素.className

3. 添加元素: 2种情况:

(1). 直接添加一个新元素: 3步

a. 创建一个新元素对象: var 新元素=document.createElement(“元素名”)

b. 为新元素添加关键属性: 新元素.属性名=属性值

c. 将新元素添加到DOM树: 3种:

1). 追加到一个父元素下所有子元素之后: 父元素.appendChild(新元素)

2). 插入到一个父元素下的一个现有子元素之前: 父元素.insertBefore(新元素, 现有元素)

3). 替换一个父元素下的一个现有子元素: 父元素.replaceChild(新元素, 现有元素)

(2). 同时添加多个平级子元素: 3步:

a. 先创建文档片段对象: var frag=document.createDocumentFragment();

b. 将子元素添加到文档片段对象中: frag.appendChild(子元素)

c. 将文档片段一次性添加到DOM树中指定父元素下: 父元素.appendChild(frag)

4. 删除元素: 父元素.removeChild(子元素)

总结: this

1. obj.fun() this->obj

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

3. new Fun() this->new将来创建的某个子对象

4. 原型对象.prototype.fun=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->当前访问器属性所在的对象

Object.defineProperties(eric,{

eage:{

get(){ return this.\_eage },

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

}

})

6. 元素.onclick=function(){ … } this->当前正在触发事件的元素

作业:

1. (必须) 看小程序视频学习重排重绘及其优化:

小程序->在线->DOM->day02: 5. 修改样式 计算属性 computed 重排重绘

2. (必须) 看小程序视频继续完成带样式的表单验证

小程序->在线->DOM->day02: 作业: 带class样式的表单验证

3. (扩展)看小程序视频学习实现选择省份和城市的级联下拉列表

小程序->在线->DOM->day03: 2. 选择省份和城市的级联下拉列表