正课:

1. 添加/删除

2. HTML DOM常用对象

3. BOM

1. 添加/删除

1. 添加: 3步:

1. 创建一个空元素

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

创建 元素

强调: 只能用document调用

问题: <a></a>

2. 为空元素添加关键属性和内容

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

a.innerHTML="go to tmooc"

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

问题: 网页上依然看不见a

因为: 网页的排版和绘制，都是以DOM树为依据。而新创建的元素对象，还没有加载到DOM树上。排版引擎和绘图引擎不知道多了新元素，自然就不会画到页面上  
 3. 将空元素挂载到DOM树

3种:

1. 追加到一个父元素下的所有子元素末尾

父元素.appendChild(新元素)

追加

2. 插入到一个父元素下的某个子元素之前

父元素.insertBefore(新元素, 现有元素)

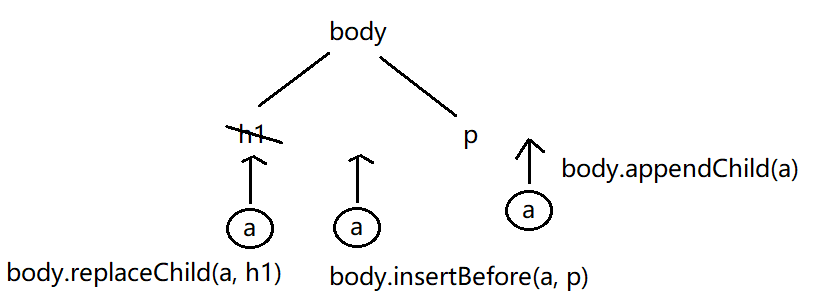
插入 之前

3. 替换父元素下一个现有的旧元素

父元素.replaceChild(新元素，旧元素)

替换

强调: 将新元素挂到DOM树上，都要先找到它的父元素，由父元素调用函数，将新元素添加到自己的子元素中。



2. 优化: 尽量减少操作DOM树的次数

为什么: 只要修改一次DOM树，就会导致重排重绘

如果频繁重排重绘严重: 会闪屏

解决:

1. 如果同时添加父元素和子元素，应该先在内存中将子元素先添加到父元素中，最后再将父元素整体一次性添加到DOM树——只重排重绘一次！

2. 如果父元素已经在页面上了，要添加多个平级子元素。应该找一个临时的爹: 文档片段对象。先将多个平级子元素，在内存中，添加到文档片段对象中，再将文档片段对象整体一次性添加到DOM树上——也只重排重绘一次

什么是文档片段: 内存中临时保存多个平级子元素的虚拟父元素

何时: 只要父元素已经在页面上了，要添加多个平级子元素

如何: 3步:

1. 创建文档片段

var frag=

document.createDocumentFragment();

创建 文档 片段

2. 将子元素添加到文档片段中

frag对象是一个虚拟的父元素，所以，用法和真实父元素完全一样。

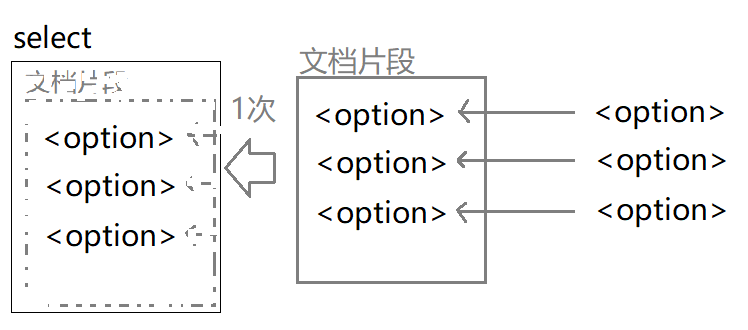
比如: frag.appendChild(新元素)

3. 将文档片段对象一次性添加到DOM树

真实父元素.appendChild(frag)

问题: 文档片段会不会成为页面上实际的元素？

答: 不会，文档片段在将子元素添加到DOM树后，就自动释放。不会成为实际页面元素。



3. 删除: 父元素.removeChild(子元素)

2. HTML DOM常用对象:

HTML DOM对部分常用的复杂的元素对象，提供了一些简写的函数和属性:

1. Image对象：代表页面上一个<img>元素

唯一的简化: 创建<img>元素: var img=new Image()

代替: var img=document.createElement("img");

强调: new Image()只创建<img>元素，不负责挂载。还需要用多一句appendChild()。。。将<img>元素挂载到DOM树上。

2. Select/Option:

1. Select 对象: 代表页面上一个<select>元素

属性:

.selectedIndex 获得当前选中的option在整个select下的位置下标。

.value 获得当前选中的option的value属性值。如果option上没有value属性，则返回option的内容文本代替。

.options 获得当前select下所有option对象的集合。

.options.length 获得当前select下共有几个option对象

.length == .options.length 也可以获得当前select下共有几个option对象

方法:

.add(option) 向select下添加一个option对象

.remove(i) 移除i位置的option对象。

2. Option对象: 代表页面上一个<option>元素

唯一的简化: 创建<option>元素:

var opt=new Option(内容文本, value属性值)

相当于:

var opt=

document.createElement("option")

opt.innerHTML=内容文本;

opt.value=value属性值;

3. Table/...:

Table从上到下完整结构4级: table 行分组 tr td

Table对象采用逐级管理的方式:

Table管着行分组：

添加行分组:

var thead=table.createTHead();

一句话顶两句话

1. var thead=

document.createElement("thead")

2. table.appendChild(thead)

var tbody=table.createTBody();

var tfoot=table.createTFoot();

删除行分组:

table.deleteTHead();

table.deleteTFoot();

获取行分组:

table.tHead

table.tFoot

table.tBodies[0]

|-tBodies:

[

|-tbody

|-tbody

]

因为HTML中规定，一个table下可以有多个tbody！所以，tbody对象都是放在table对象的tBodies集合中管理的，并不直接隶属于table。

行分组管着行：

添加行: var tr=行分组.insertRow(i)

在当前行分组内i位置，添加一个新行

等于以前两句话:

var tr=

document.createElement("tr")

行分组.appendChild(tr)

强调: 如果i位置已经有行，则现有行向后顺移一行。新行插入才当前i位置

固定套路: 2个  
 1. 末尾追加一行:

var tr=行分组.insertRow()

2. 开头插入一行:

var tr=行分组.insertRow(0)

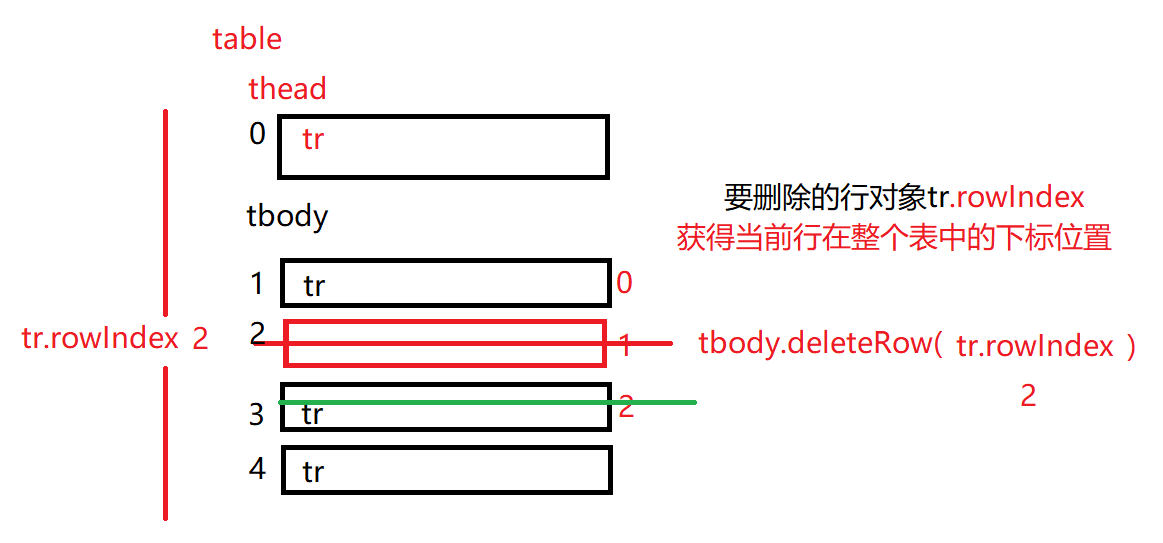
删除行: 行分组.deleteRow(i)

删除行分组中的第i行

问题: i 要求必须是当前行在行分组内的相对下标位置。行在行分组内的相对位置，只能靠肉眼判断，无法用程序自动获得！

其实: 每个行对象tr，都有一个.rowIndex属性，自动获得这一行在整个表中的下标位置

问题2: 行分组.deleteRow()需要一个行分组内的相对下标。而tr.rowIndex自动获得的是行在整个表中的绝对下标位置。很有可能错位！因为表格中很可能有表头行。所有tr.rowIndex不能用在行分组.deleteRow()中



解决: 今后只要删除行:

table.deleteRow(tr.rowIndex)

因为主语换成table后，deleteRow需要的，刚好是行在table内的位置。而tr.rowIndex获得的刚好也是行在table中的位置。两者就配对儿了。



获取行: 行分组.rows[i]

行分组中将所有行对象，都保存在一个rows集合中管理。

行管着格：

添加格: var td=tr.insertCell(i)

格

一句话顶两句话:

var td=

document.createElement("td")

tr.appendChild(td);

固定套路: 行末尾追加一格:

var td=tr.insertCell()

删除格: tr.deleteCell()

获取格: tr.cells[i]

tr将内部所有td元素对象，都集中保存在一个cells集合中，统一管理。

补: 确认框:

什么是确认框: 有两个选项的对话框

如何:

var bool=confirm("消息提示")

确认

如果用户点确定，就返回true

如果用户点取消，就返回false

4. Form/表单元素

Form对象:

获取: document对象已经将当前页面中所有的form对象收集在了forms数组中。

document.forms[i 或 id]

属性:

.elements 获得表单中所有表单元素对象的集合。

强调: .elements中只能获得表单元素

.elements.length 可获得表单中表单元素的个数

.length == .elements.length 获得表单中表单元素的个数

表单元素对象:

获取:

不带name的表单元素:

form.elements[i 或 id 或 name]

有name的表单元素:

form.name名

方法: 表单元素.focus()

让当前表单元素自动获得焦点

3. BOM: Browser Object Model

什么是: 专门操作或访问浏览器软件的一批函数和对象

何时: 只要想获取浏览器软件的信息，或操作浏览器窗口时

如何:

包括:

1个最大的对象: window

3个角色

1. 代替ECMAScript中的global充当全局作用域对象。

所以，所有我们自己声明的全局变量和全局函数默认都是在window中。

2. 包含了所有ECMAScript，DOM和BOM的函数和对象:

比如: window.Array

window.document

window.alert

3. 代表着当前正在打开的浏览器窗口

比如: 获取浏览器窗口的大小:

1. 完整大小:

window.outerWidth

window.outerHeight

2. 文档显示区大小:

window.innerWidth

window.innerHeight

打开和关闭窗口:

1. 打开新窗口: 4种需求:

1. 在当前窗口打开，可后退:

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

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

2. 在当前窗口打开，禁止后退:

js: location.replace("新url")

3. 在新窗口打开，可打开多个:

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

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

4. 在新窗口打开，只能打开一个:

html: <a href="url" target="自定义窗口名">

js: window.open("url", "自定义窗口名")

原理:

1. 在浏览器内存中，每个窗口都有一个name属性，唯一标识一个窗口

2. a的target属性和open()的第二个参数，其实是在给新窗口起名字(name属性)

3. 浏览器规定: 同名的窗口，只能开一个！后打开的同名窗口，会覆盖先打开的同名窗口。

有两个预定义的特殊意义的窗口名：

\_self: 自动获得当前窗口的name名给新窗口——结果: 新窗口覆盖旧窗口

\_blank: 空白，不给新窗口指定窗口名。而是靠浏览器随机生成！——结果: 新窗口的名字，一定不重复！即使打开多个窗口，也不会覆盖。

总结: 优化:

1. 查找:

如果只要一个条件就可以找到元素时:

首先按HTML特征查找: getElementsByXXX

如果查找条件复杂时: 用选择器查找

querySelector()和querySelectorAll()

2. 尽量减少操作DOM树的次数

1. 添加删除:

如果同时添加父元素和子元素，应该先在内存中将子元素添加到父元素中。最后，再将父元素一次性添加到DOM树

如果父元素已经在页面上了，应该先在内存中创建文档片段对象，将所有平级子元素先添加到文档片段中。最后，再将文档片段一次性添加到父元素上

2. 修改时: 能一次修改完成的，就不分两句话修改！

比如: 元素.style.width="200px"

元素.style.height="100px"

其实: 元素.style.cssText=`width:200px; height:100px`;

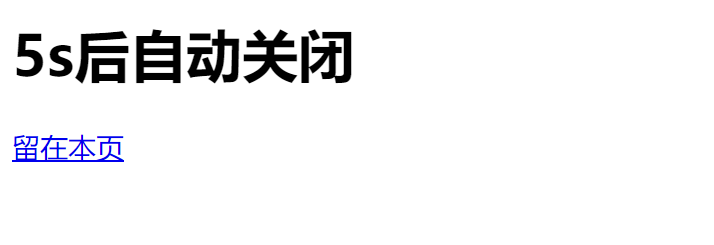
作业:

1. 3个鄙视题视频:

1. f(n,o)\_2\_3.mov

2. event loop in browser.mov和event loop in node js.mov

2. 实现5s后自动关闭页面:



所需知识： window.close()可关闭当前网页

需求:

1. 打开页面时，页面5s可每隔1秒倒计时一次: 5 4 3 2 1

2. 5s后，页面自动关闭

3. 5s内单击留在本页，可停止定时器，不再关闭页面

3. Promise鄙视题:

//假设有一个服务端接口，可根据条件分别查询三种商品的数量:

http://localhost:5050/?type=a,

http://localhost:5050/?type=b,

http://localhost:5050/?type=c

分别返回a类商品的总数，b类商品的总数，c类商品的总数

每个接口，因为数据量很大，所以，可能有3秒的返回延迟。

在day03/homework/文件夹下，运行:

node server.js

可测试以上三个地址:

需求: 在.html网页中如何获得a类, b类, c类商品的总和

考虑程序完成的总时间最少。

//现只有一个已封装的基本的ajax函数，暂时不支持promise。

//步骤:

1. 修改ajax.js中的ajax函数，使其支持Promise

2. 使用自己修改好的支持Promise的ajax函数，实现规定的需求。