正课:

1. 添加删除元素

2. HTML DOM常用对象

BOM

1. window

2. history

3. location

4. \*\*\*定时器

5. navigator

1. 添加删除元素

优化: 尽量减少操作DOM树的次数

如何:

1. 如果同时添加父元素和子元素，则应该将子元素先添加到父元素中，最后再将父元素一次性添加到页面

2. 如果同时添加多个平级子元素，则需要借助文档片段

什么是文档片段DocumentFragment: 内存中，临时存储一个DOM子树的虚拟父对象。

何时: 只要同时添加多个平级子元素时

如何: 3步:

1. 创建文档片段

var frag=document.createDocumentFragment();

2. 将子元素临时添加到文档片段中

frag.appendChild(child)

3. 将文档片段整体一次性添加到DOM树

parent.appendChild(frag)

强调: 将子元素送到DOM树后，文档片段自动释放，不占用页面元素

删除: parent.removeChild(child);

child.parentNode.removeChild(child)

2. HTML DOM常用对象:

什么是: 对部分结构复杂的常用的HTML元素提供的简化版API

包括: Image Select/Option Table/... Form/Element

何时: 今后能用HTML DOM简化操作的地方，优先用HTML DOM

如何:

Image: 代表页面上一个<img>元素

创建: var img=new Image();

Select: 代表页面上一个<select>元素

属性: .selectedIndex 获得当前select中选中项的下标位置

.value 获得当前select中选中项的value值

如果没有value，则用innerHTML代替

.options 获得当前select中所有option的集合

.options.length 获得当前select中所有option的个数

.options.length=0;//清空所有option

.length => .options.length

方法: .add(option) => .appendChild(option)

.add不支持frag

.appendChild支持frag

.remove(i) 移除i位置的option

Option: 代表select下一个option元素

创建: var opt=new Option(text,value);

opt=document.createElement("option");

opt.innerHTML=text;

opt.value=value;

Table: 管着行分组

创建行分组: var thead=table.createTHead();

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

table.appendChild(thead);

var tbody=table.createTBody();

var tfoot=table.createTFoot();

删除行分组: table.deleteTHead();

table.deleteTFoot();

获得行分组: table.tHead

table.tBodies[i]

table.tFoot

行分组: 管着行:

添加行: var tr=行分组.insertRow(i)

在表中第i行插入一个新行

如果i位置已经有行，就向后挤压顺移

两个固定套路:

1.末尾追加: var tr=行分组.insertRow()

2. 开头插入: var tr=行分组.insertRow(0)

删除行: 行分组.deleteRow(i)

i: 是在当前行分组内的相对下标位置

问题: 行分组内的相对下标位置，无法自动获得

解决: tr.rowIndex 自动获得当前行在整个表中的下标位置

问题: 行分组.deleteRow(i)要求i是在当前行分组内的下标，和行在整个表中的下标可能不一致

解决: 最好的删除tr的做法:

table.deleteRow(tr.rowIndex)

tr.parentNode.removeChild(tr)

获取行: 行分组.rows[i]

行: 行管着格

添加格: var td=tr.insertCell(i); //一般不加i，表示行尾追加

强调: 只能添加td，不能添加th

删除格: tr.deleteCell(i)

获取格: var td=tr.cells[i]

Form: 代表页面上一个表单元素

获取: var form=document.forms[i/id];

属性: .elements 获得表单中的所有表单元素

只包括: input select textarea button

.elements.length 获得表单中表单元素的个数

.length => .elements.length

何时: 查找表单结尾的不带name属性的按钮

比如: form.elements[form.length-2]

获得表单中倒数第二个提交按钮

方法: form.submit() 代替submit按钮，手动提交表单

Element: 代表表单中一个表单元素

获取: var elem=form.elements[i/id/name]

其实可以更简化: form.name 前提: 必须有name属性

方法: elem.focus() 让当前元素获得焦点

elem.blur() 让当前元素失去焦点

BOM: Browser Object Model

专门操作浏览器的API

问题: 没有标准

BOM模型包括:

window: 2个角色:

1. 代替ES标准中的global，充当全局作用域对象

2. 代表当前正在打开的浏览器窗口

包括:

history

location

document

screen

navigator

event

1. window窗口:

文档显示区: 浏览器窗口中，专门用于显示网页内容的区域

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

打开和关闭窗口:

window.open("url","target")

总结: 4种打开新链接的方式

1. 在当前窗口打开，可后退

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

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

4. 在新窗口打开，只能打开一个

html: <a href="url" target="自定义name">

js: open("url","自定义name")

补充: window.name属性 内存中的，用户看不到

浏览器要求相同name属性的窗口，只能打开1个

后打开的同名窗口会替换先打开的。

window.close()

说明: 其实不加window.也行

2. history: 保存当前窗口打开后，成功访问过的url的历史记录栈

用程序执行前进，后退，刷新

前进: history.go(1)

后退: history.go(-1) history.go(-2)

刷新: history.go(0)

3. location: 地址栏，保存和操作当前窗口正在打开的url的对象

属性:

.href 地址栏中完整的url

.protocol 协议

.host 主机+端口号

.hostname 主机

.port 端口号

.pathname 相对路径

.search ?及其之后的查询字符串参数

.hash #及其之后的锚点地址

方法:

1. 在当前窗口打开新链接:

1. 可后退:

location.assign("url") => location.href="url"

=> location="url";

2. 不可后退:

location.replace("新url");

用新url代替当前url——禁止后退

2. 刷新: 重新加载

2大类:

1. 普通刷新: 优先从缓存中加载资源，缓存过期或缓存中没有，才去服务器下载新的。

F5

history.go(0)

location.reload(/\*false\*/); //不强迫必须从服务器下载新的

force

2. 强制刷新:

location.reload(true) //强制从服务器下载新的

4. \*\*\*定时器: 2种:

1. 周期性定时器:

什么是: 让程序每隔一段时间间隔，反复执行一项任务

何时: 当程序每隔一段时间间隔，反复执行一项任务

如何: 3件事:

1. 定义任务函数:

2. 启动定时器: 将任务函数放入定时器中，并设置时间间隔

var timer=setInterval(task,ms)

3. 停止定时器: clearInterval(timer)

强调: timer不会自动清空,所以必须timer=null

2种:

1. 手动停止:

2. 自动停止:

在任务函数中设置临界值，一旦达到临界值，就自动调用clearInterval;

2. 一次性定时器:

什么是: 让程序先等待一段时间，再执行一次任务。执行后，立刻自动停止定时器。

何时: 只要让程序先等待一段时间，再延迟执行任务时

如何: 3件事:

1. 定义任务函数:

2. 启动定时器: 将任务函数放入定时器中，设置延迟等待的时间

var timer=setTimeout(task,wait\_ms);

3. 停止定时器:

clearTimeout(timer);

定时器原理:

启动定时器时，仅是将任务函数加入到定时器中

到达时间间隔时，定时器将任务函数加入回调队列

回调队列中的回调函数，必须等待主程序执行完，才能执行

结论: 无论何时调用setInterval/setTimeout，则任务函数，只能在主程序之后执行

鄙视: for(var i=0;i<3;i++){

setTimeout(()=>{console.log(i)},50); //0

}//3,3,3

5. navigator:

什么是: 专门保存浏览器配置信息的对象

何时: 只要读取浏览器配置信息

包括:

cookieEnabled: 判断浏览器是否启用cookie

什么是cookie: 客户端永久保存用户私密信息的小文件

为什么: 程序中的数据都是临时的，无法持久保存的

何时: 只要希望在客户端永久保存用户私密信息

plugins

\*\*\*userAgent