正课：

1. HTML DOM常用对象:

Image Select/Option Table/.... Form/...

2. 什么是BOM

3. 打开和关闭窗口

4. \*\*\*\*定时器

5. BOM常用对象:

history location navigator screen

1. HTML DOM常用对象:

Image: 指代页面上一个img元素

创建: var img=new Image();

Select: 指代页面上一个<select>元素

属性: sel.selectedIndex 获得当前选中项的下标位置

sel.value 获得当前选中项的值(value)

如果当前option没有value，则用innerHTML代替

sel.options 获得当前sel下所有option元素的集合

sel.options.length 选项的个数

sel.options.length=0 清除所有选项

sel.length => sel.options.length选项的个数

sel.length=0 清除所有选项

方法: sel.add(option) 向sel末尾追加一个option

强调: 不支持文档片段

sel.remove(i) 移除i位置的选项

事件: sel.onchange 当选中项发生改变时触发

Option: 指代页面上一个option元素

创建: var opt=new Option(text,value)

属性: opt.text代替opt.innerHTML

Tabel: 指代一个table元素:

管着行分组: 创建，删除，获得

创建: var thead=table.createTHead();

var tbody=table.createTBody();

var tfoot=table.createTFoot();

删除: table.deleteTHead();

table.deleteTFoot();

获取: table.tHead

table.tBodies[i]

table.tFoot

行分组: tHead tBody tFoot

管着tr:

创建: var tr=tXXX.insertRow(i);

说明: 如果i位置有行，则原位置的行向后顺移

常用操作: 1. 末尾追加一个新行: tXXX.insertRow();

2. 开头插入: tXXX.insertRow(0)

删除: tXXX.deleteRow(i) 删除当前行分组中下标为i的行

获取: tXXX.rows

行:

管着格:

创建: tr.insertCell(i)

强调: insertCell只能创建td

删除: tr.deleteCell(i)

获取: tr.cells

删除行: 2种:

1. 行分组.deleteRow(i)

i 是行在行分组内的相对位置, 无法直接获得

2. table.deleteRow(i)

i 是行在整个表中的绝对位置，可直接获得:

tr.rowIndex 表示tr在整个表中的位置

总结: 今后凡是删除行: table.deleteRow(tr.rowIndex)

Form: 代表页面上一个form元素

获取: var form=document.forms[i/id]

属性: form.elements 包含所有表单元素: input select button textarea

form.elements.length 获得表单中表单元素的个数

form.length => form.elements.length

方法: form.submit(); //手动提交表单

只有验证通过才提交: 2种:

1. input button 代替input submit

在input button的单击事件中自己调用form.submit();

2. input submit 配合 onsubmit事件 ?

表单元素:

获取: form.elements[i/id/name]

更简写: 如果表单元素有name属性，可直接:

form.name

方法: elem.focus(); 让elem获得焦点

elem.blur(); 让elem失去焦点

2. 什么是BOM: Browser Object Model

BOM: 专门操作浏览器窗口/软件的API

window: 2个角色:

1. 代替ES中的global充当全局作用域对象

2. 封装所有dom和BOM的API

window的功能: 打开和关闭窗口，弹出对话框...

包括:

history: 保存当前窗口打开后，成功访问多的url的历史记录栈。

location: 保存当前窗口正在打开的url的对象

document: 封装页面内容和DOM API的根对象

navigator: 保存浏览器的配置信息

screen: 保存客户端显示设备的信息

event: 定义事件对象

3. 打开和关闭窗口:

打开一个新窗口: /\*window.\*/open("url","name")

其中: name: 新窗口在内存中的名称

浏览器规定: 相同name的窗口只能开1个

后开的会替换先开的。

预定义值: \_self: 用当前窗口的name打开新窗口

结果: 新窗口替换当前窗口

\_blank: 不指定name属性，让浏览器随机分配

结果: 每个窗口的name都不一样

就可打开多个

打开一个超链接: 4种:

1. 在当前窗口打开，可后退:

HTML: <a href="url" target="\_self"></a>

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

2. 在当前窗口打开，不可后退:

当前窗口每打开一个新url，都会将新url保存在history中。

如果新url是追加进history中，则可后退

如果新url将当前url替换掉，则无法后退

js: location.replace(url)

3. 在新窗口打开，可打开多个:

HTML: <a href="url" target="\_blank"></a>

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 "></a>

js: open("url","自定义name")

4. \*\*\*\*定时器：2种:

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次任务

何时: 让程序每隔一段时间间隔反复执行一次任务

如何: 3件事:

1. 任务函数: function task(){...}

定义了定时器每次要执行的任务

2. 启动定时器: var timer=setInterval(task,间隔ms)

强调: task是回调，不用加()

其中: timer是定时器序号, 在内存中唯一标识一个定时器的整数。

3. 停止定时器: clearInterval(timer)

其中: timer是要停止的定时器序号

问题: 停止定时器不会自动清除timer变量中的序号，有可能影响下次启动。

解决: 凡是停止定时器，都要手动清除timer

2种方式:

1. 手动停止: 用户通过操作来停止定时器

2. 自动停止:

在任务函数中判断临界值

只要达到临界值，就自动停止定时器

一次性定时器:

什么是: 程序先等待一段时间后，再自动执行一次任务

何时: 先等待一段时间再自动执行一次

自动执行一次后，自动停止

如何: 3件事:

1. 任务函数:

2. 启动定时器: var timer=setTimeout(task,等待ms)

3. 停止定时器: 在任务执行前，就停止等待

clearTimeout(timer);

原理: 定时器中的回调函数只能在主程序所有语句执行完才能开始执行。

鄙视: var a=10;

function fun(){ a=100; }

setTimeout(fun,0); //fun()

console.log(a);//10 //100

5. BOM常用对象:

history: 保存当前窗口打开后成功访问过的url的历史记录栈

history.go(n) : history.go(0) 刷新

history.go(-1) 后退

history.go(1) 前进

location: 保存当前窗口正在打开的url对象

属性: .href 获取或设置完整url

.protocol 协议

.host 主机名 包含: 主机名+端口号

.hostname 仅主机名

.port端口号

.pathname 相对路径

.hash 获得锚点地址

.search 获得?后的查询字符串

方法: location.assign("url") 在当前页面打开新url

其实: =>location.href="url"

=>location="url"

location.replace("url") 在当前窗口打开，替换history中当前url，实现禁止后退

location.reload(fales/true) 刷新

鄙视: 2种刷新:

1. 默认刷新: 优先使用本地缓存中的文件，除非文件比服务器上的旧，才被迫下载新文件。

F5

history.go(0)

location.reload()

2. 强制刷新: 每次强制跳过浏览器本地缓存，总是从服务器下载新文件。

location.reload(true)

screen: 保存客户端显示设备的信息:

鄙视: 判断屏幕宽

css: 媒体查询

js: screen.width

navigator: 封装了浏览器的配置信息:

cookeEnabled 判断是否启用cookie

plugins 保存所有插件信息

plugins["插件名"]!==undefined 说明装了！