day01 pm

A正课:

1. \*\*\*DOM树

遍历:

2. \*\*\*查找:

1. \*\*\*DOM树:

网页中一切内容在内存中都是以树形结构存储的。

今后只要存储上下级关系的数据，也要用树形结构

网页中的每项内容在内存中都是一个节点对象Node

包括:

document: 整个树的根节点

网页中一切内容都是document的子节点

节点对象: Node Node.prototype

三大属性:

\*nodeType: 节点的类型

document 9

\*element 1

attribute 2

\*text 3

问题: 只能区分节点类型，不能进一步区分元素类型

解决:

\*nodeName: 节点的名称

document #document

\*element 标签名 今后只要区分元素，就nodeName

强调: 所有元素的节点名都是全大写！

attribute 属性名

\*text #text

nodeValue: 节点值

document null

element null

attribute 属性值 都用.value

文本 文本内容

节点间关系:

节点树: 特点: 包含所有节点，包括看不见的（空格 回车）文本节点

1. 父子: parentNode childNodes firstChild lastChild

父 直接子

2. 兄弟: previousSibling nextSibling

前一个兄弟 后一个兄弟

问题: 会受到看不见的(空格 换行)文本节点干扰。

解决:

元素树: 只包含元素节点，不包含一切文本节点

1. 父子: parentElement children IE8+

父 直接子元素

firstElementChild lastElementChild

2. 兄弟: previousElementSibling nextElementSibling

前一个兄弟 后一个兄弟

兼容性问题:IE9+

强调: 元素树不是新的树，仅仅是节点树去掉文本后的子集

问题: 不包含一切文本节点 可用.innerHTML

何时: 只要已经站在一个节点，要找周围节点时

重点: childNodes和children返回的都是动态集合(live collection)类数组对象

动态集合: 不实际存储数据，每次访问时都重新查找DOM树

优: 首次查找速度极快。因为不用完全遍历

缺: 反复查找，效率低，会反复查找DOM树

错: for(var i=0;i<children.length;i++)

每遍历一次就会访问children.length反复查找DOM树

解决: 遍历: for(var i=0,len=children.length;i<len;i++)

只在第一次查找DOM树，之后不再反复查找。

没有获得任何节点 在整个页面查找想要的元素

已获得元素 查找旗下符合条件的子元素

遍历: 查找指定父节点下的所有子节点

Step1: 先定义函数，仅遍历直接子节点

Step2: 对每个直接子节点，再调用和父节点相同的函数

强调: 1. 递归: 函数内又调用了函数自己

其实，递归时，不应直接使用函数名

应该用arguments.callee代替

但严格模式下已不推荐使用

效率低（外层函数调用还未结束，内层函数调用已经开始）

解决: 2. 多数递归都可用循环代替。

遍历API: ——了解 2个:

1. nodeIterator节点迭代器:

什么是: 依次访问每个节点对象。每次只能访问一个

何时: 只要用循环方式遍历子节点时

如何: 2步:

Step1. 创建节点迭代器对象:(告知起点)

var iterator=document.createNodeIterator(

parent, NodeFilter.SHOW\_ALL,

.SHOW\_ELEMENT

null,false

);

Step2. 调用iterator的nextNode():

返回当前所在节点，再跳到下一个节点

到头，返回null

固定套路:

var node=null;

while((node=iterator.nextNode())!=null){

node//当前节点对象

}

总结: 递归和遍历API都是用深度优先遍历算法:

优先遍历子节点，子节点遍历完，才返回遍历兄弟

如果想打破深度优先的顺序:

2. TreeWalker: 可任意跳转节点的特殊迭代器

基本用法和iterator完全一样

除了父节点不输出——一开始就站在父节点上

差别: 多了几个调整方向的API:

.parentNode(); //跳到父节点

.first/lastChild(); //跳到第一个/最后一个子节点

.previous/nextSibling();//跳到前一个/后一个兄弟

2. \*\*\*查找:

按HTML查找: 4种:

\*1. 按id属性，精确查找一个元素对象:

var elem=document.getElementById("id")

效率非常高！

强调: getElementById只能用在document上

何时: 只用于精确查找一个元素

问题: 不是所有元素都有id

\*2. 按标签名找:

var elems=parent.getElementsByTagName("tag");

查找指定parent节点下的所有标签为tag的子代节点

强调: 1. 可用在任意父元素上

2. 不仅查直接子节点，而且查所有子代节点

3. 返回一个动态集合

即使只找到一个元素，也返回集合

必须用[0],取出唯一元素

问题: 不但找直接，而且找所有子代。

3. 按name属性找: 专门查找表单中指定name属性的元素

var elems=document.getElementsByName("name");

4. 按class属性找: 查找父元素下指定class属性的元素

var elems=parent.getElemnetsByClassName("class");

有兼容性问题: IE9+

练习: <button onclick="calc(this)"->button对象

\_\_\_传递\_\_|

↓

function calc(btn){btn->this->button对象

this -> window

}

何时: 处理函数中需要快速获得当前元素对象时,就用this

最可靠的API: .parentNode 父元素

.getElementById

.getElementsByTagName

按选择器查找: Selector API

1. 只找一个元素:

var elem=parent.querySelector("selector")

强调: selector支持一切css中选择器

2. 找多个:

var elems=parent.querySelectorAll("selector")

强调: selector API 返回的是非动态集合:

实际存储数据，每次访问不会反复查找DOM树

鄙视vs getXXXX:

3.选择器查找

1. 效率: 首次查找， getXXX 效率高: 不返回全部数据

selectorAPI 效率低: 要准备完整数据

2. 返回值: getXXX 返回动态集合: 可能反复查找DOM树

selectorAPI 返回非动态集合: 不会反复查找DOM树

3. 难易: getXXX繁琐; selector 绝对简单

总结: 如果一次查找即可获得想要的元素时，首选getXXX

如果需要多次查找才能获得想要的元素时, 首选selector

day02

正课:

1. 修改:

内容:

属性:

标准属性

扩展属性(自定义属性)

样式:

1. 内容: 2种:

.innerHTML: 返回开始标签到结束标签之间的html代码片段

.textContent: 返回开始标签到结束标签之间的纯文本内容

去掉所有子标签，翻译转义字符

IE8: innerText

正课:

1. 修改:

属性:

标准属性:

扩展属性(自定义属性)

样式:

内联样式

内部/外部样式表

DOM标准:

核心DOM: 可操作一切结构化文档的API, 包括HTML和XML

万能!

繁琐!

HTML DOM: 专门操作HTML文档的简化版DOM API

仅对常用的复杂的API进行了简化

不是万能!

简单

开发: 先用简单的，再用复杂的补充——以实现效果为目标

1. 属性:

标准属性: HTML标准中规定的属性

核心DOM: 4个操作

1. 读取属性值: 2种:

1. 先获得属性节点对象，再获得节点对象的值:

var attrNode=elem.attributes[i/属性名];

elem.getAttributeNode(属性名)

attrNode.value——属性值

\*2. 直接获得属性值

var value=elem.getAttribute("属性名");

2. 修改属性值:

elem.setAttribute("属性名", value);如果属性不存在 也可以set

3. 判断是否包含指定属性:

var bool=elem.hasAttribute("属性名")

4. 移除属性:

elem.removeAttribute("属性名")只移出开始标签的attribute 不删除property

HTML DOM: 所有标准属性已经成为内存中元素对象的属性

elem.属性名 - 普通对象的属性用法完全一样

elem.class -> elem.className

直接访问内存中的property 还可以操作.checked .disabled .selected

property vs attribute:

attribute: 出现在元素开始标签中的属性

property: 仅在内存中有，没有出现在元素开始标签中

所有标准属性: 即是attribute，也可是property

特例: .checked .disabled .selected

只能用.访问,值为bool类型,不能用get/setAttribute操作

扩展(自定义)属性:

何时: 1. 用于自定义行为的添加

2. 缓存客户端数据

核心DOM: 可以

不能使用HTML DOM:

HTML5: 定义了扩展属性的规范: ——强烈建议

1. 添加扩展属性: 属性名必须是"data-属性名"

2. 凡是加了data-前缀的属性,都可用:

elem.dataset.属性名

课堂练习: 获得焦点和失去焦点事件

txt.onfocus 当获得焦点时自动触发

txt.onblur 当失去焦点时自动触发

2. 样式:

内联样式: elem.style.属性名

强调: 属性名: 去横线，变驼峰

比如: css: background-color => backgroundColor

list-style-type => listStyleType

问题: 一句话只能修改一个属性

解决: 今后都是先用class定义不同场景的元素样式

再用DOM控制何时穿哪个class

问题: elem.style只能获得内联样式！

无法获得从样式表中层叠或继承来的样式

解决:

内部/外部样式表:

\*获取计算后的样式:

var style对象=getComputedStyle(elem)

var value=style.属性名

强调: 仅用作读取，不能修改！想修改: 去修改内联样式

也可直接修改样式表中的样式: 1. 危险; 2. 繁琐

1. 找样式表对象: var sheet=document.styleSheets[i]

2. 找样式表中要修改的样式规则(CSSRule):

var rule=sheet.cssRules[i]

3. 找规则对象的style属性中的css属性:

rule.style.transition="opacity 1s linear"

day03 am

正课:

1. 添加和删除元素

2. \*\*\*HTML DOM常用对象:

Select>option

Table>...

Form>...

1. 添加和删除元素:

添加: 3步:

1. 创建空元素:

var elem=document.createElement("标签名")

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

<a></a>

2. 设置关键属性:

比如: a.innerHTML="go to tmooc"

a.herf="http://tmooc.cn";

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

3. 将元素添加到DOM树: 3种:

1. 在父元素末尾追加:

parent.appendChild(child)

2. 插入到父元素下的现有子元素之前:

parent.insertBefore(child, oldChild)

3. 替换父元素下现有子元素

parent.replaceChild(child, oldChild)

\*\*\*优化: 尽量少的操作DOM树

为什么: 每次修改DOM树，都导致重新layout

1. 如果同时创建父元素和子元素时，建议在内存中先将子元素添加到父元素，再将父元素一次性挂到页面

2. 如果只添加多个平级子元素时, 就要将所有子元素，临时添加到文档片段中。再将文档片段整体添加到页面

文档片段: 内存中，临时保存多个平级子元素的 虚拟父元素

用法和普通父元素完全一样

如何: 3步:

1. 创建片段:

var frag=document.createDocumentFragment();

2. 将子元素临时追加到frag中:

frag.appendChild(child);

3. 将frag追加到页面:

parent.appendChild(frag);

强调: append之后，frag自动释放，不会占用元素

HTML->DOM Tree

↓

Render Tree->\*\*\*layout->paint

↑

CSS->cssRules

练习: select: onchange事件: 当选中项发生改变时

.selectedIndex属性: 当前选中项的下标

day 03pm

正课:

1. \*\*\*HTML DOM常用对象:

Image Select>Option Table>... Form>...

删除元素: parent.removeChild(child对象)

1. Image: 代表img元素

核心DOM: var img=document.createElement("img");

HTML DOM: var img=new Image();

2. Select: 代表一个select元素

创建: 无

属性: 除标准属性外,

.selectedIndex: 当前选中项下标 \*

.options: 包含当前select下所有option对象的集合 \*

.options.length: 获得option元素的个数

.options.length=0: 清空所有option

.value: 获得当前选中项的值:

如果选中项有value属性，则返回选中的opt的value

如果选中项没有value，则返回选中的opt的内容

方法:

添加option:

核心: sel.appendChild(opt)

html DOM: sel.add(opt) \*

删除option:

核心: sel.removeChild(sel.options[i])

HTML DOM: sel.remove(i)

事件:

onchange: 当选中项发生改变时 \*

Option: 代表一个option元素

创建:

核心DOM:

var opt=document.createElement("option");

opt.text=text;

opt.value=value;

HTML DOM: var opt=new Option(text,value);

属性: text, value, index: option在当前select下的位置

练习: 第二种事件绑定方式: 在js中动态绑定事件

elem.on事件名=function(){

this->.前的elem对象

}

3. Table: 表示一个table元素

tHead: 创建: var thead=table.createTHead()

删除: table.deleteTHead();

获取: table.tHead

tr: 创建:

var tr=tHead.insertRow(i);//省略i，默认末尾追加

删除: tHead.deleteRow(i);

获取: tHead.rows

td: 创建: var td=tr.insertCell(i);//省略i，默认末尾追加

强调: insertCell只能创建td, 不能创建th

删除: tr.deleteCell(i);

获取: tr.cells

tBodies:

tBody: 创建: var tbody=table.createTBody();

获取: table.tBodies[i]

tFoot: 创建: var tfoot=table.createTFoot()

删除: table.deleteTFoot();

获取: table.tFoot

tr.rowIndex: 记录了tr在 整个表 中的下标位置

table.deleteRow(i): 可用tr.rowIndex ——首选

tbody.deleteRow(i): 不能用tr.rowIndow，只能用tr在tbody内的下标。

练习: 三大对话框: alert() confirm() prompt()

确认: var bool=confirm("提示信息");

两个按钮: 确定 true 取消 false

内存泄漏:

原因: 在事件处理函数中，使用外层函数的DOM对象

形成了闭包

后果: 即使DOM树中删除了元素

闭包依然将元素对象保存在内存中

解决: 在事件处理函数中，不要使用外层函数的DOM对象

要随找随用

4. Form对象:

获取form对象: document.forms[i/id/name]

属性: .elements: 保存了表单中所有表单元素的对象

.length: 记录表单中表单元素的个数

方法: .submit() 专门用于用代码提交表单

表单元素:

获取: var elem=form.elements[i/id/name]

form.name(的值) —— 只能访问带name的元素

方法: .focus() 让当前元素获得焦点——

每个页面只能有1个焦点。

当前元素获得焦点同时，原焦点元素自动失去焦点

.blur() 让当前元素失去焦点——很少用

day04 am

正课:

1. BOM:

window

打开关闭窗口

窗口大小和窗口位置

\*\*\*\*定时器:

1. BOM: Browser Object Model

专门操作浏览器窗口的API——没有标准, 有兼容性问题

浏览器对象模型:

window: 代表整个窗口

2个角色: 1. 代替global成为全局作用域对象

2. 封装所有DOM API和BOM API

history: 封装当前窗口打开后，成功访问过的历史url记录

navigator: 封装浏览器配置信息

document: 封装当前正在加载的网页内容

location: 封装了当前窗口正在打开的url地址

screen: 封装了屏幕的信息

event: 定义了网页中的事件机制

打开和关闭窗口:

1. 在当前窗口打开新链接，允许后退:

html: <a href="url"

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

2. 在当前窗口打开新链接，禁止后退:

js:

3. 在新窗口打开新链接，可重复打开多个:

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

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

4. 在新窗口打开新链接，只能打开一个:

html: <a href="url" target="?"

js: open("url","?")

每个窗口在内存中都有一个name属性

(自己不要起变量名为name)

name属性是内存中唯一标识一个窗体的名称

浏览器规定: 相同name的窗口只能打开一个

后打开的会替换先打开的

何时指定name:

a元素的target属性中写的就是新窗口的name属性

open()方法的第二个参数，也是新窗口的name属性

name属性可自定义，也可使用预定义值: \_blank \_self ....

窗口大小和窗口位置:

窗口大小:

获取当前窗口大小:

完整窗口大小: window.outerWidth/outerHeight

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

设置窗口大小:

1. 在打开窗口时:

var pop=window.open("url","name",config);

其中:config="left=?,top=?,width=?,height=?"

文档显示区的大小

返回值: 返回弹出窗口的window对象

比如: pop->就是小窗口的window

2. 修改已经打开的窗口大小:

.resizeTo(width,height): 修改到指定大小

完整大小

.resizeBy(width增量,height增量): 让窗口增大多少

强调: 只能修改在弹出时就已经修改大小的窗口

不能修改默认全屏的当前窗口

窗口位置:

获取窗口相对于屏幕左上角的位置:

window.screenLeft/screenTop

修改已经打开的窗口位置:

moveTo(left,top);

moveBy(left的增量,top的增量)

练习:

屏幕总大小: screen.width/height

可用大小: screen.availWidth/availHeight

=总大小-任务栏的宽/高

day04 pm

正课:

e:

1. \*\*\*\*定时器:

什么是: 让程序按指定时间间隔自动执行任务

1. 周期性定时器:

让程序按指定时间间隔反复自动执行一项任务

何时: 动画

如何: 3件事:

1. \*\*\*任务函数: function task(){...}

2. 启动定时器: 将任务函数放入定时，并设定时间间隔

timer=setInterval(task, ms)

3. 停止定时器:

clearInterval(timer)只是停止定时器 序号不清空; timer=null;

2种方式停止:

1. 手动停止:clearInterval(timer);

timer=null;

2. 自动停止: 在任务函数中设定临界值

如果达到临界值，自动调用clearInterval(timer);

2. 一次性定时器:

让程序先等待一定时间间隔,再执行一次任务。执行后自动停止

何时: 延迟任务的执行

如何: 3件事儿

1. 任务函数: task

2. 启动定时器: timer=setTimeout(task, ms)

3. 停止: clearTimeout(timer)

如何选择:

只要反复执行，就用周期性

只要只执行一次，就用一次性

\*\*\*\*定时器的原理:

定时器中的回调函数，必须在所有主程序执行完，也就是主程序最后一句话之后，才能执行！

day05 am

正课:

1. \*\*\*BOM常用对象: history location navigator event

1. history: 封装当前窗口打开后成功访问过的url的历史记录栈

无法读取和修改其内容

如何: history.go(1): 前进

history.go(-1): 后退

history.go(0): 刷新

history.go(n): 前进n次

2. location: 封装当前窗口正在打开的url地址

属性:

href: 完整url

protocol: 协议

host: 主机名+端口号:

hostname: 主机名

port: 端口号

pathname: 相对路径名称

hash: #锚点名

search: ?查询字符串

在当前窗口打开新链接:

location.href="url"

location.assign("url")

location="url";

禁止后退: location.replace("url");

刷新: location.reload(false/true)

force: 是否强制从服务器硬盘重新获取html文件

false: 优先从服务器缓存中快速获取html

true: 强制从服务器硬盘获取html文件

3. navigator: 封装浏览器配置信息的对象

.cookieEnabled: 检测浏览器是否启用了cookie

.plugins: 包含浏览器安装的所有插件的集合

.userAgent: 记录浏览器内核，版本号，名称的字符串

day05pm

正课:

1. Event:

事件绑定: 3种:

\*\*\*事件模型:

冒泡

取消冒泡

利用冒泡

取消事件:

事件坐标:

页面滚动:

cookie: 读写

1. 事件绑定: 3种:

1. 在html中绑定:

<button onclick="calc(this)"

问题: 无法对动态生成的元素绑定事件

不符合内容与行为分离的原则

2. 在js中动态绑定: 2种:

1. btn.onclick方式

btn.on事件名=function(){

this -> btn;

}

触发事件时，btn.onclick();——事件处理函数

优: 可对任何元素动态绑定事件

内容与行为分离

缺: 无法为一个元素的一个事件同时绑定多个事件处理函数

2. btn.addEventListener("事件名",function(){...})

优: 为一个元素的一个事件同时绑定多个事件处理函数

2. \*\*\*事件模型:

目标元素: 实际点击的元素

DOM: 3个阶段:

1. 捕获阶段(capture): 由外向内, 到目标元素结束

依次记录每级元素上绑定的相同事件

2. 目标触发: 默认，首先执行目标元素上的事件处理函数

3. 冒泡执行: 由内向外，按照捕获顺序的反向，依次触发各级父元素上绑定的事件

btn.addEventListener("事件名",function(){...},capture)

capture: false: 不在捕获阶段执行——默认

true: 在捕获阶段提前执行！

如果事件处理函数在捕获阶段提前出发，那么冒泡阶段不再重复触发。

IE8: 2个阶段: 没有捕获阶段

btn.attachEvent("on事件名",function(){...})

事件对象:

在事件发生时，自动创建的，保存事件相关信息的对象。

如何获得:

DOM: 事件对象，默认会作为事件处理函数的第一个参数传入

IE8: window.event

e=e||window.event; //利用短路逻辑解决兼容性问题

取消冒泡:

DOM: e.stopPropagation();

IE8: e.cancelBubble=true;

if(e.stopPropagation){

e.stopPropagation();

}else{

e.cancelBubble=true;

}

利用冒泡:

优化: 尽量减少事件监听对象的个数

解决: 只在父元素上添加一次事件监听，所有子元素共用！

问题: 如何区分目标元素: e.target||e.srcElement

DOM IE8

取消事件: e.preventDefault();——阻止默认行为

IE8: e.returnValue=false;

if(e.preventDefault){

e.preventDefault();

}else{

e.returnValue=false;

}

练习: \*form\*的onsubmit事件:

只要提交表单，正式提交前都会自动触发onsubmit

事件坐标:

1. 相对于整个屏幕的坐标: e.screenX|screenY

2. 相对于文档显示区的坐标: e.clientX||e.x

e.clientY||e.y

3. 相对于相对定位的父元素的坐标: e.offsetX|e.offsetY

页面滚动:

onscroll: 当页面滚动时触发

scrollTop: 页面滚动，超出文档显示区顶部的高度