正课:

1. 什么是DOM

2. \*\*\*查找

3. 修改

1. 什么是DOM

DOM: Document Object Model

什么是: 专门操作网页内容的API

W3C指定的标准，所有浏览器厂商遵照实现

何时: 今后只要操作网页内容，就必须用DOM

为什么: 为了统一操作网页内容的API

用DOM标准操作网页内容几乎100%兼容

分为: 核心DOM和HTML DOM

核心DOM: 万能，繁琐

HTML DOM: 对核心DOM部分常用API的简化

不是万能，但简化

\*\*\*DOM树:

网页中的一切内容在内存中都是以树形结构存储

所有内容(元素，属性，文本)在树上，都是一个节点对象

DOM有唯一的一个根节点document

所有内容节点都是document的后代节点

\*\*\*节点对象:

网页中的所有内容，都是DOM树上的节点对象

类型: Node

三大属性:

nodeType: 表示节点的类型

值是整数, 4个:

document 9

element 1

attribute 2

text 3

何时: 只要区分节点类型时

为什么: 不同类型的节点可执行的操作不同

问题: 无法进一步区分不同元素

解决:

nodeName: 节点名(元素的标签名)

何时: 只要进一步细致区分元素种类时

包括: document #document

element 元素的标签名(全大写)

attribute 属性名

text #text

其实, nodeName可代替nodeType来鉴别节点类型

nodeValue: 节点值——了解

包括: document null

element null

attribute 属性值

text 文本内容

DOM操作:

查找触发事件的元素->绑定事件

->查找要操作的元素-->修改/添加/删除

2. \*\*\*查找: 4种:

1. 不需要查找就可直接得到:

document

document.documentElement —— html

document.head

document.body

2. 按节点间关系查找:

何时: 只要已经获得一个节点，要找周围节点时

包括: 2大类关系:

节点树: 包含一切内容节点的树结构

父子: elem.parentNode

elem.childNodes 返回\*直接\*子节点的集合

elem.firstChild

elem.lastChild

兄弟: elem.previousSibling

elem.nextSibling

问题: 包括看不见的空字符文本节点的干扰

元素树: 只包含元素节点的树结构

父子: elem.parentElement

elem.children 返回\*直接\*子元素的集合

elem.firstElementChild

elem.lastElementChild

兄弟: elem.previousElementSibling

elem.nextElementSibling

优: 不受空文本的干扰

缺: IE8不兼容

说明: 元素树不是一棵新树，仅是节点树的一个子集

强调: childNodes和children返回的不是数组，而是类数组对象。

类数组对象: 长的像数组的对象

vs 数组: 相同: 1.下标, 2. .length

不同: 类型不同，API不通用

简称: 集合

其实,childNodes和children都返回动态集合

动态集合: 不实际存储数据，每次访问集合，都重新查找DOM树。

问题: 反复访问集合，会导致反复查找DOM树

遍历动态集合:

错误: for(var i=0;i<childNodes.length;i++){...}

正确: for(var i=0,len=childNodes.length;i<len;i++)

鄙视: 用节点间关系，遍历指定父节点下所有后代节点

1. 递归遍历: 2步:

Step1: 先仅查找直接子节点

Step2: 对每个直接子节点，调用和父节点完全相同的函数

算法: 深度优先遍历: 每个节点都优先遍历子节点，子节点遍历完，才遍历兄弟节点。

问题: 递归时，函数内的函数名不能写死

解决: 用arguments.callee指代当前函数自己。

2. 用循环代替递归:

节点迭代器对象: 依次遍历，并返回每个节点对象的小工具——内置深度优先算法

如何: 2步:

1. 创建节点迭代器对象:

var iterator=document.createNodeIterator(

parent, NodeFilter.SHOW\_ALL, null, false

SHOW\_ELEMENT

);

2. 循环调用迭代器的nextNode()，调到下一个节点。直到返回null结束。

do{

var node=iterator.nextNode();

if(node!=null){

//输出node

}else break;

}while(true);

3. 按HTML查找: 4种:

1. 按id查找: var elem=document.getElementById("id")

强调: 1. 必须用document调用！

2. 按标签名查找:

var elems=

parent.getElementsByTagName("标签名");

强调: 1. 可在任意父元素上调用，仅找当前父元素下的指定元素。

2. 返回动态集合

3. 不但找直接子元素，而且找所有后代元素

3. 按name属性查找: ——了解

何时: 在表单中查找有name属性的表单元素时

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

强调: 1. 只能用document调用

2. 返回动态集合

4. 按class属性查找:

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

强调: 1. 在任意父元素上调用

2. 返回动态集合

3. 不但找直接子元素，而且在所有后代中查找

4. 不要求完整匹配class，只要包含就找到

问题: 每次只能按一个条件查找，如果条件复杂，代码会很繁琐

解决: 当查找条件复杂时，要用选择器查找

4. 按选择器查找: Selector API: 2个:

1. 只找一个元素:

var elem=parent.querySelector("selector");

2. 找多个元素:

var elems=parent.querySelectorAll("selector");

强调: 1. 可在任意父元素上调用

2. 不仅查找直接子元素，且查找所有后代元素

3. 返回非动态集合

非动态集合: 数据直接保存在集合本地，无序反复查找。

4. 选择器兼容性，受制于当前浏览器

鄙视: 按HTML查找 vs 按选择器查找

1. 返回值: 按HTML查找: 返回动态集合:

优: 效率高！只需返回需要的数据即可，不需要返回完整数据

缺: 造成反复查找

按选择器查找: 返回非动态集合

优: 不会反复查找

缺: 首次执行效率低

2. 易用性: 当查找条件复杂时

按HTML查找: 繁琐

按选择器查找: 简洁

总结:

如果根据一个条件就可获得想要的元素:

首选按HTML查找

如果查找条件复杂时

首选按选择器查找

总结返回值:

1. 凡是返回一个元素的API，如果没找到，都返回null

2. 凡是返回多个元素的API，如果没找到，都返回空元素的集合

练习: 事件绑定:

//当事件发生时，自动执行保存的事件处理函数

elem.on事件名=function(){

this //自动获得触发事件的.前的当前元素

}

3. 修改:

内容:

innerHTML: 获取或设置元素开始标签到结束标签之间的HTML代码片段。

textContent: 获取或设置元素开始标签到结束标签之间的纯文字内容。

2件事: 1. 去掉文本中内嵌的标签

2. 将所有转义字符翻译为正文

IE8: innerText

.value: 获取或设置表单元素的内容

属性:

样式: