正课:

1. 什么是DOM

2. \*DOM Tree

3. \*查询: 4种

1. 什么是DOM: Document Object Model

什么是: 专门操作网页内容的统一的API标准——W3C

为什么: 为了统一操作网页内容的API标准

用DOM操作网页内容，几乎兼容所有浏览器

何时: 只要操作网页内容，都用DOM

如何:

2. DOM树:

什么是: 内存中，存储网页中所有内容的树型结构

为什么: 树型结构最适合保存上下级关系，且可无限向下延伸。

何时: 只要存储不确定层级深度的上下级关系，都用树型结构

如何: 自动创建，自动维护

1. 当浏览器读取到HTML文档时，开始创建

2. 首先创建根节点document

document对象是整棵DOM树的树根

所有网页内容，都是document节点的后代节点

3. 依次读取网页中每个元素，属性，文本

网页中每项内容(元素，属性，文本)都是DOM树上的一个节点对象。

节点对象: Node

三大属性:

nodeType: 表示节点的类型

值是一个整数,包括:

document 9

elem 1

attr 2

text 3

何时: 只要鉴别节点的类型时

问题: 无法细致区分元素的名称

nodeName: 表示节点名称

何时: 只要进一步区分元素的名称时

包括:

document #document

elem 全大写的标签名

attr 属性名

text #text

nodeValue: 表示节点的值——了解

document null

elem null

attr 属性值

text 文本内容

DOM: \*查询，修改，添加，删除，事件绑定

3. 查找: 4种:

0. 不需要查找就可直接获得元素:

html document.documentElement

head document.head

body document.body

1. 按节点间关系查找:

什么是: DOM树中任何节点都不是孤立的。

一个节点和父级，子级，兄弟之间都建立了联系

何时: 如果已经获得一个节点，找周围附近的有关系的节点时

如何: 2大类关系:

节点树: 包含所有网页内容的完整树结构

1. 父子关系:

elem.parentNode 获得elem的父节点

返回值: 唯一的一个父节点对象

elem.childNodes 获得elem的所有\*直接\*子节点

返回值: 所有直接子节点组成的类数组对象

elem.firstChild 获得elem下的第一个子节点

elem.lastChild 获得elem下的最后一个子节点

2. 兄弟关系:

elem.previousSibling 获得elem的前一个兄弟节点

elem.nextSibling 获得elem的后一个兄弟节点

问题: 一切文本都是节点对象，包括看不见的空字符，也是节点对象(tab，空格，换行)

解决: 元素树

元素树: 仅包含元素节点的树结构

优: 不受看不见的空字符的干扰

1. 父子关系:

elem.parentElement 获得elem的父元素

elem.children 获得elem的所有\*直接\*子元素

返回值: 所有直接子元素组成的类数组对象

elem.firstElementChild 获得elem下的第一个子元素

elem.lastElementChild 获得elem下的最后一个子元素

2. 兄弟关系:

elem.previousElementSibling 获得elem的前一个兄弟元素

elem.nextElementSibling 获得elem的后一个兄弟元素

元素树不是一颗新树，只是节点树的一个子级

问题: 1. IE9+

2. 遍历指定父节点下的所有后代节点: ——鄙视

问题: children和childNodes只能查找直接子节点，无法查找更深层次!

解决: 递归遍历:

如何: 2步:

1. 先定义函数，仅遍历直接子节点

2. 对每个子元素，调用和父元素完全相同的方法

算法: 深度优先遍历:

什么是: 每次同时碰到子元素和兄弟元素时，总是优先遍历子元素。所有子元素遍历完，才返回遍历兄弟。

问题: children和childNodes返回动态集合

什么是: 不实际存储数据，每次访问集合，都重新查找DOM树.

优: 首次查找，不需要返回完整数据，效率高!

缺: 反复访问集合，导致反复查找DOM树，效率低!

错误: for(var i=0;i<children.length;i++)

解决: 遍历时，提前缓存length

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

问题: 递归效率低，避免使用

解决: 用循环代替:

节点迭代器: NodeIterator:

什么是: 封装深度优先遍历算法，可以依次获取每个后代节点的对象。

如何: 2步:

1. 用父元素创建节点迭代器对象:

var iterator=document.createNodeIterator(

parent, NodeFilter.SHOW\_ALL , null, false

.SHOW\_ELEMENT

);

2. 反复调用迭代器的nextNode()方法获取下一个节点对象

nextNode()2件事:

1. 返回当前节点，2. 跳到下一个节点

如果没有后续节点，返回null

do{

var node=iterator.nextNode()

if(node!=null)

console.log(node.nodeName);

else break;

}while(true)

3. 按HTML查找: 4种:

1. 按id查找:

var elem=document.getElementById("id");

强调: 只能用document调用！

2. 按标签名查找:

var elems=parent.getElementsByTagName("标签名");

返回动态集合:

强调: 1. 可在任意父元素上调用！仅在当前父元素下查找

2. 不仅找直接子元素，且查找所有后代中的元素

3. 按name属性查找:

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

返回动态集合:

强调: 只能用document调用

4. 按class属性查找:

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

返回动态集合:

强调: 1. 可在任意父元素上调用

2. 不仅找直接子元素，且在所有后代中查找

3. 如果一个元素同时被多个class修饰时，只要按其中一个class，就可找到该元素

问题: 每次只能按一种条件查找，当查找条件复杂时，步骤就非常繁琐。

解决: 用选择器查找

4. 用选择器查找:

1. 仅查找一个元素:

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

2. 查找多个元素:

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

强调: 1. 可在任何父元素上调用

2. 返回非动态集合:

非动态集合: 实际存储完整数据，即使反复访问集合，也不会反复查找DOM树

3. selector受制于浏览器的兼容性限制