正课: DOM

1. 什么是:

2. DOM Tree:

3. \*\*\*查找

4. 修改:

1. 什么是：

DOM: Document Object Model

什么是: 专门操作网页内容的API标准——W3C

为什么: 统一所有浏览器上操作网页内容的API标准

何时: 只要操作网页内容,都要用DOM API

使用DOM API操作网页，几乎100%兼容所有浏览器

如何: 5件事:

查找->修改->添加/删除

事件绑定

2. DOM Tree

什么是: 网页中所有内容，在内存中都是保存在一棵树形结构中

为什么: 网页中的标签拥有上下级包含关系

何时: 当浏览器获取到一个HTML内容时，会自动将HTML内容转化为内存中的树形结构

如何:

1. 只要读到HTML，立刻创建唯一根节点: document

将来所有网页内容都是document对象的后代节点

2. 顺序读取HTML中的每个标签及其内容

每读取一个标签或内容，就在dom树的对应位置创建新节点对象node。

强调: 网页中一切内容都是节点对象,包括: 标签,文本,属性,注释...

node节点对象: 3个统一的属性:

nodeType: 节点类型

何时: 只要判断节点的类型

返回值: 一个数字

包括: document 9

elem 1

attr 2

text 3

问题: 无法进一步检查元素的标签名

nodeName: 节点名

何时: 只要进一步检查节点的标签名

返回值: 一个字符串

包括: document #document

elem 全大写标签名

attr 属性名

text #text

nodeValue: 节点值

何时: 几乎不用

包括: document null

elem null

attr 属性值

text 文本内容

3. \*\*\*查找: 4种:

0. 不用查找，可直接获得的元素:

<html> document.documentElement

<head> document.head

<body> document.body

<form> document.forms[i/"id"]

1. 按节点间关系查找:

何时: 如果已经获得一个元素，仅在周围找附件的节点时

如何:

节点树: 包含所有网页内容的最完整的树结构

包括: 所有的元素，文本，属性，。。。 。。。

2大类关系:

1. 父子关系:

elem.parentNode 父节点

elem.childNodes 子节点——仅包含直接子节点

返回当前elem下所有直接子节点的集合——类数组对象

elem.firstChild 第一个直接子节点

elem.lastChild 最后一个直接子节点

2. 兄弟关系:

elem.previousSibling 前一个兄弟

elem.nextSibling 后一个兄弟

问题: 受看不见的空字符的干扰！

解决:

元素树: 仅包含元素节点的树结构

2大类关系:

1. 父子关系:

elem.parentElement 父元素

elem.children 子元素——仅包含直接子元素

返回当前elem下所有直接子元素的集合——类数组对象

elem.firstElementChild 第一个直接子元素

elem.lastElementChild 最后一个直接子元素

2. 兄弟关系:

elem.previousElementSibling 前一个兄弟元素

elem.nextElementSibling 后一个兄弟元素

何时: 今后如果只关心元素节点，不关心文本节点时，就用元素树。

问题: IE9+

强调: 元素树不是一棵新树，仅是节点树的子集而已

强调: childNodes和children 都是动态集合

动态集合: live collection 不实际存储属性值，每次访问集合，都要重新查找DOM树。

优: 首次查找速度快!

缺: 反复访问集合，导致重复查找DOM树，效率低

在遍历时:

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

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

2. 遍历查找所有后代元素:

1. 递归: 2步:

1. 定义函数仅遍历直接子节点

2. 对每个直接子元素，调用和父元素完全相同的操作

算法: 深度优先遍历算法: 如果一个元素同时拥有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，才返回遍历兄弟元素。

问题: 效率极低

2. 用循环代替递归: 2步:

1. 创建节点迭代器对象: nodeIterator

nodeIterator: 内置深度优先遍历算法的，依次访问每个子元素的对象

2. 循环调用nodeIterator的nextNode()方法

跳到下一个节点，并返回下一个节点对象

3. 按HTML查找: 4种:

1. 按id查找:

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

返回一个元素，如果找不到，返回null

强调: 1. 只能用document调用

2. 按标签名查找:

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

返回多个元素组成的动态集合, 如果找不到，返回空集合

强调: 1. 可被任意父元素调用，表示，仅在指定父元素下查找。

2. 不仅查找直接子元素，且查找所有后代元素

3. 一律返回集合，即使只找到一个，也是放在集合中返回的。想取出集合中唯一的一个元素，必须加[0]

3. 按name属性查找:

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

返回多个元素组成的动态集合，如果找不到，返回空集合

强调: 1. 只能用document调用

何时: 仅用于查找表单中有name属性的元素

4. 按class属性查找:

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

返回多个元素组成的动态集合，如果找不多，返回空集合

强调: 1. 可在任意父元素上调用，仅查找当前父元素下的后代元素

2. 不仅查找直接子元素，且查找所有后代元素

3. 如果一个元素被多个class修饰，则只用其中一个class就可找到！

问题: 每次只能按1个条件查找。如果条件复杂时，代码会很繁琐

解决: 选择器

4. 用选择器查找: 2个:

1. 只找第一个符合条件的元素

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

返回一个元素对象

何时: 如果已知，只会选中一个元素时

2. 找所有符合条件的元素

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

返回所有符合条件的元素的非动态集合

非动态集合: 实际存储完整属性值，即使反复访问集合，也不会导致反复查找DOM树

优: 反复访问，效率高

缺: 首次查找，慢的

何时: 如果已知，会选中多个元素

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. 受制于当前浏览器对选择器的兼容性要求

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

按选择器查找，返回非动态集合

2. 效率:

按HTML查找，首次返回快，反复访问，慢

按选择器查找，首次返回慢，反复访问，快

3. 易用性: 当条件复杂时

按HTML查找，繁琐

按选择器查找，简洁！

结论: 如果通过一个简单的条件就可找到想要的选择时，首选按HTML查找。

如果查找条件复杂时，首选按选择器查找简化代码！

4. 修改: 3种:

1. 内容: 3种:

1. 获取或设置元素中原始的HTML代码片段:

.innerHTML

2. 获取或设置元素中纯文本内容:

.textContent

强调: 多做两件事:

1. 去掉内嵌标签

2. 将转义字符翻译为正文

3. 获取或设置表单元素的内容:

.value

2. 属性:

3. 样式: 2种:

1. 内联样式: 获取或修改: elem.style.css属性名

强调: 1. css属性名都要去横线变驼峰:

a.style.fontSize

.listStyleType

.backgroundColor

2. 多数数值css属性，有单位，必须先去单位，再参与计算