正课:

1. 什么是DOM  
2. \*\*\*查询

3. 修改

1. 什么是DOM

DOM: Document Object Model

专门操作网页内容的API标准——W3C

何时: 只要操作网页内容，都必须用DOM提供的API

分为:

核心DOM: 最初制定的操作一切结构化文档的统一API

优: 万能

缺: 繁琐

HTML DOM: 专门操作HTML文档的API

对核心DOM中部分常用API进行了简化

优: 简洁

缺: 不是万能

总结: 实际开发中不必区分核心DOM和HTML DOM

都是优先使用简单的API

如果简单地实现不了，用复杂的补充

\*\*\*DOM树:

什么是:

网页中一切内容在内存中都是以树型结构存储在一起

每项内容(元素，文本，属性)都是树上的一个节点对象

为什么: 树形结构是保存不确定层级深度的上下级包含关系最好的结构

根节点: 树结构都有唯一的一个树根节点: document节点

所有网页内容，都是document的子节点

节点对象:

什么是: 网页中每一项内容都是一个节点对象

节点对象封装了节点可用的属性和功能

如何: 三大属性:

nodeType: 节点类型:

什么是: 定义了节点的类型

何时: 区分节点的类型时

为什么: 不同类型的节点可用的属性和可执行的操作不一样

包括: 4种:

document 9

element 1

attribute 2

text 3

问题: 无法进一步区分具体的元素名

解决: nodeName

nodeName:

什么是: 保存节点名称的属性

何时: 进一步区分具体的元素名

为什么: 不同的元素拥有的属性和可执行的操作都不一样

包括:

document #document

element 标签名(全大写)

attribute 属性名

text #text

nodeValue: 了解

什么是: 保存节点的值

何时: 几乎不用

包括:

document null

element null

attribute 属性值

text 文本内容

DOM操作的流程: 增删改查+事件处理

查找触发事件的元素->绑定事件处理函数

->查找要修改的元素->修改属性/样式,添加,删除

2. \*\*\*查找: 4种:

不需要查找可直接获得的元素:

document.documentElement -> html

document.head -> head

document.body -> body

1. 按节点间关系查找:

何时: 如果已经获得了一个元素，想找周围元素时

节点树: 包含所有节点的完整树结构

包括:

1. 父子关系:

child.parentNode 获得一个节点的父节点

parent.childNodes 获得父节点下的所有直接子节点

parent.firstChild 获得父节点下的第一个直接子节点

parent.lastChild 获得父节点下的最后一个直接子节点

2. 兄弟关系:

elem.nextSibling 获得一个节点相邻的下一个兄弟节点

elem.previousSibling获得一个节点相邻的前一个兄弟节点

问题: 受看不见的空字符的干扰

解决: 元素树

元素数: 仅包含元素节点的树结构

何时: 只要仅关心元素，不关心其它类型的节点时

如何:

1. 父子关系:

child.parentElement 获得一个节点的父元素

parent.children 获得父元素下的所有直接子元素

parent.firstElementChild 获得父元素下的第一个直接子元素

parent.lastElementChild 获得父元素下的最后一个直接子元素

2. 兄弟关系:

elem.nextElementSibling 获得一个元素相邻的下一个兄弟元素

elem.previousElementSibling获得一个元素相邻的前一个兄弟元素

说明: 元素树不是一棵新树,只是节点树的部分子集

2. 遍历节点:

什么是: 查找一个父元素下所有后代节点

如何: 2步:

1. 定义函数仅遍历直接子节点

2. 对每个碰到的子节点调用和父节点完全相同的操作

算法: 深度优先: 每当同时有子元素和兄弟元素时，总是先遍历子元素。子元素遍历完，才返回遍历兄弟元素

问题: 递归的执行效率极低

解决: 用循环代替

如何: 用节点迭代器对象

什么是节点迭代器对象: NodeIterator

专门按照深度优先遍历的顺序依次访问每个子元素的对象。

如何使用: 2步:

1. 创建NodeIterator:

var iterator=document.createNodeIterator(

parent, NodeFilter.SHOW\_ALL, null, false

SHOW\_ELEMENT

);

2. 用循环反复调用NodeIterator的nextNode()函数

做2件事:

1. 返回当前节点

2. 蹦到下一个节点

到返回null退出

chileNodes和children:

类数组对象: 长得像数组的对象

vs 数组:

相同: 1. 下标, 2. .length, 3. 用for循环遍历

不同: 类型不同, API不通用

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，会造成反复查找DOM树

错误的遍历:

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

解决: 正确的遍历:

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

arguments.callee:

什么是: 专门指代当前正在调用的函数自己

何时: 只要递归，在函数内调用自己，必须用arguments.callee代替

为什么: 避免修改外层函数名后，还要重复修改内部调用的函数名

强调: 因为递归算法效率极低

所以ES6或新版浏览器中已经禁止使用arguments.callee, 意味着不再推荐使用递归算法。

遍历查找的问题: 遍历的步骤非常繁琐

而查找功能又会频繁使用

解决: 使用专门的查找API

3. 按HTML查找: 4个:

1. 按id查找一个元素:

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

强调: 1. 必须用document调用！

2. 只返回一个元素

2. 按标签名查找多个元素

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

强调: 1. 可在任意父元素上调用

表示仅查找当前父元素下的后代元素

2. 不但找直接子元素，且查找所有后代元素

3. 返回多个元素组成的动态集合

3. 按name属性查找: 了解

var elems=document.getElementsByName("name")

何时: 查找拥有name属性的表单元素时

强调: 1. 只能用document调用

2. 返回多个元素组成的动态集合

4. 按class属性查找:

var elems=parent.getElementsByClassName("class")

强调: 1. 可在任意父元素上调用

2. 返回多个元素组成的动态集合

3. 只要包含指定的类名，就选择该元素

不必完整匹配

4. 不但找直接子元素，且找所有后代元素

问题: 每次只能按一种条件查找

如果查找条件复杂，代码会很繁琐

解决: 用选择器查找

4. 按选择器查找: 2个:

1. 只查找一个元素:

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

2. 查找多个元素:

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

强调: 1. 可在任意父元素上调用

2. 返回非动态集合:

非动态集合: 实际存储数据，即使反复访问，也不会导致反复查找DOM树。

遍历: 不用先缓存length

3. 受制于当前浏览器的兼容性要求

总结:

1. 如果已经获得一个元素，找周围元素: 用节点间关系查找

2. 如果通过一个条件就可找到想要的元素: 用按HTML查找

3. 如果查找条件复杂时: 用selector API

鄙视: 按HTML vs 按Selector

1. 返回值: 按HTML查找: 返回动态集合

按Selector查找: 返回非动态集合

2. 效率: 首次查找按HTML效率快！

按selector查找稍慢!

3. 易用: 按selector查找更简洁

按HTML查找更繁琐

2. 修改:

内容:

.innerHTML : 开始标签到结束标签之间的html代码片段

.textContent : 开始标签到结束标签之间的文本内容

去掉了内嵌的标签

将转义字符范围以正文

强调: IE8 不兼容, 用.innerText代替

.value: 专门获得表单元素的内容

属性:

样式: 内联样式: elem.style.css属性名="值"

强调: css属性名都要去横线变驼峰

过渡动画: 不要用display

1. js中: 用支持过渡的css属性修改样式:

width height opacity ....

2. css中:

起始样式

overflow:hidden

transition:all .5s linear

凡是带transition的元素，无论以任何手段修改样式属性值，都自带过渡效果