正课:

1. 什么是DOM

2. \*\*\*DOM Tree

3. \*\*\*查找

4. 修改

1. 什么是DOM: Document Object Model

专门操作网页内容的API

W3C制定的标准，所有浏览器100%兼容

为什么: 以前操作网页内容的API，不同浏览器，各不相同

存在严重的兼容性问题

何时: 只要操作网页内容，都用DOM API

能够对网页内容做哪些操作:

查找触发事件的元素

->绑定事件

->查找要修改的元素

->修改, 删除, 添加

2. \*\*\*DOM Tree

什么是: 内存中，存储所有网页内容的树型结构

为什么: 树形结构最适合表达上下级包含关系

如何:

1. DOM Tree只有唯一树根节点: document 代表整个网页

2. 所有网页内容，都是document的子节点

3. 每项网页内容(元素，文本，属性...)，都是树上的一个节点对象:node

节点对象三大属性:

nodeType: 节点类型:

返回一个数字: 包括:

document 9

elem 1

attr 2

text 3

何时: 只要鉴别节点对象的类型时

为什么: 不同类型的节点，可执行的操作和属性都不一样

问题: 无法进一步鉴别元素的名称

nodeName: 节点名称:

返回一个字符串: 包括:

document #document

elem 全大写的标签名

attr 属性名 —— 可忽略 ——属性很少当节点用

text #text

何时: 只要进一步鉴别元素的名称时

其实, 可代替nodeType鉴别节点的类型

nodeValue: 代表节点的值——了解

　　包括:

document null

elem null

attr 属性值

text 文本的内容 —— 可用innerHTML代替

3. \*\*\*查找: 4种:

1. 不用查找，即可直接获得的节点: 4个

document

document.documentElement html

document.head

document.body

2. 按节点间关系查找:

何时: 如果先获得了一个节点，找周围有关系的节点时

如何:

节点树: 包含所有网页内容(元素，文本，属性)的树结构

包含2大类关系:

1. 父子: 4个

elem.parentNode 父节点

elem.childNodes 所有\*直接\*子节点

elem.firstChild 第一个直接子节点

elem.lastChild 最后一个直接子节点

2. 兄弟:

elem.previousSibling 前一个兄弟节点

elem.nextSibling 后一个兄弟节点

问题: 受看不见的空字符的干扰！

解决: 元素树

元素树: 仅包含元素节点的树结构

强调: 元素树不是一颗新树，而是节点树的子集而已

为什么: 节点树包含所有文本内容，而通常我们只关心元素节点。

何时: 只要仅访问元素节点时，不希望受看不见的空字符干扰时

包含2大类关系:

1. 父子: 4个

elem.parentElement 父元素

elem.children 所有\*直接\*子元素

elem.firstElementChild 第一个直接子元素

elem.lastElementChild 最后一个直接子元素

2. 兄弟:

elem.previousElementSibling 前一个兄弟元素

elem.nextElementSibling 后一个兄弟元素

问题: IE9+

鄙视: 递归遍历所有后代节点:

为什么: childNodes,children，都只能遍历直接子节点/元素

何时: 只要希望在所有后代中查找节点时

如何: 2步:

1. 先定义函数，仅遍历直接子节点:

2. 对每个直接子节点，调用和父节点完全相同的函数

什么是递归: 函数内，又调用了自己

何时: 只要希望对子元素，执行和父元素相同的操作时，就要用递归。

算法: 深度优先遍历: 当同时遇到兄弟节点和子节点时，总是优先遍历子节点，所有子节点遍历完，才返回遍历兄弟节点。

问题1: childNodes和children的返回值:

并不是数组,而是类数组对象！

类数组对象: 长得像数组的对象:

相同: 1. 下标, 2. .length, 3. for循环遍历

不同: 类型不同, API不通用！

不但是类数组对象，而是是一种动态集合！

动态集合: 不实际存储数据，每次访问都需要重新查找DOM树

缺点: 反复访问，会导致频繁查找DOM树——效率低

解决: 遍历时，先缓存数量，再遍历:

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

正确:

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

问题2: 递归调用时，内部函数调用不能写死函数名

解决: 用arguments.callee 代替写死的函数名

arguments.callee 就指代当前正在调用的函数本身

与函数名无关

问题3: 递归算法效率本身极低,今后尽量少用递归！

解决: 几乎所有递归，都可用循环代替！

节点迭代器: NodeIterator

什么是: 按照深度优先的顺序，依次遍历父元素下所有后代节点的小对象.

何时: 只要希望遍历父节点下所有后代节点时，都要用节点迭代器

如何: 2步:

1. 用父节点创建节点迭代器对象:

2. 用循环反复调用迭代器的nextNode()方法，跳向下一个节点！

3. 按HTML属性查找: 4个

1. 按id精确查找一个元素:

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

返回值: 返回一个元素，找不到，返回null!

强调: 1. 只能通过document调用

2. 按标签名查找多个元素:

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

返回值: 包含所有找到元素的类数组对象

如果找不到，返回.length为0的空集合

强调: 1. 用任意父元素调用

2. 不仅查找直接子元素，且查找所有后代元素

3. 返回动态集合！

3. 按Name属性查找:

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

返回值: 包含所有找到的元素的类数组对象

如果找不到，返回.length为0的空集合

强调: 1. 必须用document调用

2. 返回动态集合

4. 按class属性查找:

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

返回值: 包含所有找到的元素的类数组对象

如果找不到，返回.length为0的空集合

强调: 1. 可在任意父元素上调用

2. 返回动态集合！

3. 在所有后代元素中查找

4. 只要包含要找的class名即可，不必完全相同

问题: 一次只能按1个条件查找

如果条件复杂, 代码会很繁琐！

解决: 选择器

4. 用选择器查找:

何时: 只要查找条件复杂时，都用选择器查找:

为什么: 选择器可用最简单的代码，实现最精确的查找！

如何: 2种:

1. 找一个: var elem=parent.querySelector("selector");

返回值: 仅返回一个找到的元素, 找不到，返回null

2. 找多个: var elems=parent.querySelectorAll("selector");

返回值: 非动态集合！找不到，返回空集合

非动态集合: 实际存储属性值，即使反复访问，也不会反复查找DOM树

强调:1. 可在任意父元素上调用!

2. 受制于浏览器对选择器的兼容性

鄙视: 按HTML vs 按选择器

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: 首次查找: 按HTML查找，效率高

因为无需准备好所有数据，就可立刻返回

按选择器查找，效率低

因为必须准备好完整数据，才能返回！

3. 易用性: 如果条件复杂: 按HTML查找，繁琐

按选择器查找，简洁

总结: 如果只靠一个条件就可找到元素时，首先按HTML查找

如果查找条件复杂时，首选按选择器查找

4. 修改:

内容: 3种:

1. html代码片段: .innerHTML

2. 纯文本内容: .textContent

多做两件事: 1. 去标签，2. 转义字符翻译为正文

3. 表单元素的值: .value

属性:

样式: 2种:

1. 获取或修改内联样式:

elem.style.css属性名="值";

强调: css属性名必须去横线变驼峰