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标准——W3C

为什么: 最初的DOM是没有标准的，存在严重的兼容性问题。DOM就是为了统一操作网页内容的API

何时: 只要操作网页内容，都用DOM API

只要用DOM标准API操作网页内容，几乎100%兼容所有浏览器。

原生js: ECMAScript(核心语法)

+DOM(操作网页内容)

+BOM(操作浏览器窗口)

2. DOM Tree:

什么是: 所有网页内容，在内存中，都是在一棵树形结构中存储的

为什么: 树型结构是最好的展现层次关系的结构

何时: 当浏览器读到网页内容时，就会自动在内存中创建一棵DOM树

内容:

DOM树都有一个唯一的树根节点对象: document

网页中每项内容，都是树上的一个节点对象: node

每项内容: 元素，文本，属性

节点对象: 所有节点都是 Node 类型的节点对象

三大属性:

nodeType: 节点类型

何时: 只要判断节点的类型时

包括: document 9

element 1

attribute 2

text 3

问题: 只能判断是不是元素，无法进一步判断元素的标签名

nodeName: 节点名

何时: 只要进一步判断节点的标签名时

包括: document #document

element 全大写的标签名

attribute 属性名

text #text

nodeValue: 节点值

何时: 几乎不用

包括: document null

element null

attribute 属性值

text 文本内容

3. \*\*\*查找: 4种:

1. VIP: 不需要查找，就可直接获得的元素:

document.documentElement -> html

document.head -> head

document.body -> body

document.forms[i/id/name] -> form

2. 按节点间关系查找:

节点树: 包含一切网页(元素,属性,文本)内容的完整树结构

2大类:

1. 父子关系:

elem.parentNode 父节点

elem.childNodes 所有\*直接\*子节点

返回所有直接子节点对象组成的类数组对象

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 前一个兄弟节点

elem.nextSibling 下一个兄弟节点

问题: 受看不见的空字符的干扰! 通常我们只关心元素，不关心字儿。

解决:

元素树: 仅包含元素节点的树结构

不是一棵新树，只是节点树的子集

2大类:

1. 父子关系:

elem.parentElement 父元素

.parentNode

elem.children 所有\*直接\*子元素

返回所有直接子元素对象组成的类数组对象

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 下一个兄弟元素

childNodes和children: 返回动态集合: live collection

什么是动态集合: 不实际存储属性值，每次访问集合，都重新查找DOM树。

优: 首次查找快!

缺: 反复访问集合，会导致反复查找DOM树，效率低

强调: 遍历

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

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

鄙视: 查找一个父元素下所有后代元素

方法一: 递归(深度优先遍历)

2步:

1. 先定义函数，仅遍历直接子元素

2. 对每个直接子节点，调用和父节点完全相同的操作

方法二: 循环

节点迭代器: 可按深度优先遍历的顺序，每次遍历下一个节点对象

如何: 2步:

1. 创建节点迭代器对象:

var iterator=document.createNodeIterator(

parent, NodeFilter.SHOW\_ELEMENT, null, false

.SHOW\_ALL

);

2. 循环调用iterator的nextNode()方法:

var currNode=iterator.nextNode();

1. 返回当前节点

2. 跳到下一个节点

如果没有下一个节点了，则返回null

3. 按HTML查找: 4个

1. 按id查找一个元素:

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

返回值: 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(tag name)查找多个元素

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

返回值: 返回包含多个元素的动态集合(类数组对象)

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. 不仅查找直接子元素，且查找所有后代元素

3. 按name属性查找:

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

返回值: 返回包含多个元素的动态集合(类数组对象)

找不到,返回length为0的空集合

强调: 1. 只能在document上调用

4. 按class查找:

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

返回值: 返回包含多个元素的动态集合(类数组对象)

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. 不仅查找直接子元素，且查找所有后代

3. 只要元素的class属性中，有一个class匹配，就选中该元素

4. 按选择器查找:

为什么: 按HTML查找，每次只能按一个条件查找。

如果查找条件复杂，步骤会很繁琐

何时: 只要查找条件复杂时，都用按选择器查找。

如何:

1. 如果只查找一个元素:

var elem=parent.querySelector("选择器");

返回值: 一个元素对象

没找到，返回null

2. 如果查找多个元素:

var elems=parent.querySelectorAll("选择器");

返回值: 包含所有符合条件元素的非动态集合

找不到，返回length为0的空集合

非动态集合: 实际存储元素和属性值，即使反复访问，不会反复查找DOM树

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

3. 受制于浏览器的兼容性

鄙视: getElementsByTagName vs querySelectorAll

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查找快

按选择器查找慢

3. 易用性:

按选择器查找绝对简单

按HTML查找繁琐

总结: 如果只要一个条件就可获得想要的元素时，首选按HTML查找

如果查找条件复杂时，首选按选择器查找

4. 修改: 3种东西可以修改

1. 内容: 3种:

1. 获取或修改HTML内容:elem.innerHTML

2. 获取或修改纯文本内容: elem.textContent

1. 去掉内嵌的标签

2. 将转义字符翻译为原文

说明: textContent IE9+

3. 获取或修改表单元素的内容: elem.value

2. 属性:

3. 样式:

所有DOM操作，都遵循4步:

1. 查找触发事件的元素

2. 绑定事件处理函数

3. 查找要修改的元素

4. 修改其内容，属性，样式