复习:

1. 将类数组对象复制为数组:

var arr2=Array.prototype.slice.call(arguments)

将类数组对象复制为数组，并选取指定位置的剩余元素

var arr2= Array.prototype.slice.call(arguments,starti)

相当于arguments.slice(starti)

其实更简单的: var arr2= [].slice.call(arguments,starti)

2. promise中的错误处理:

其实: new Promise(可接收2件事)

.then( ) .catch( )

new Promise((正常函数,出错函数)=>{

如果顺利执行:

调用正常()

否则

调用出错()

})

3. 等待多个任务完成

前提: 每个任务都必须都返回Promise

如何: Promise.all([

task1(), task2(),...

]).then(()=>{所有任务完成后才执行的任务})

正课:

1. 什么是DOM

2. DOM Tree

3. 查找

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: 统一不同浏览器操作网页内容的API标准

优点: 几乎所有浏览器100%兼容

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

网页中每项内容(元素,文本,属性,注释...)，都是树上的一个节点对象。

唯一的树根节点: document

为什么: 树形结构是最好的保存上下级包含关系的结构

节点对象: Node

网页中每项内容都是DOM树上的一个节点对象:

所有节点都有的三个属性:

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: 无法进一步区分元素的标签名

nodeName: 节点名称

何时: 只要进一步判断元素的标签名时

——可代替nodeType

包括:

document #document

element 全大写标签名

attribute 属性名——不常用!

text #text

nodeValue: 节点值 ——不常用

document null

element null

attribute 属性值

text 文本内容

3. \*\*\*查找: 4种:

1. 不需要查找可直接获得节点:

document.documentElement html

document.head head

document.body body

document.forms[id/i] form

2. 按节点间关系查找:

何时: 如果已经获得一个节点了。想找周围节点时。

包括: 2种树:

1. 节点树: 包含网页中所有内容的完整树结构

2大类关系:

1. 父子: 4种:

elem.parentNode elem的父节点

elem.childNodes elem的直接子节点

elem.firstChild elem下的第一个直接子节点

elem.lastChild elem下的最后一个直接子节点

2. 兄弟: 2种:

elem.previousSibling elem的前一个兄弟元素

elem.nextSibling elem的后一个兄弟元素

问题: 受看不见的空字符的干扰!

2. 元素树: 仅包含元素节点的树结构

2大类关系:

1. 父子: 4种:

elem.parentElement elem的父元素

elem.children elem的直接子元素

elem.firstElementChild elem下的第一个直接子元素

elem.lastElementChild elem下的最后一个直接子元素

2. 兄弟: 2种:

elem.previousElementSibling elem的前一个兄弟元素

elem.nextElementSibling elem的后一个兄弟元素

今后只要用DOM操作网页内容，都用元素树

说明: 元素树不是一棵新树，其实只是节点树的一个子集

childNodes和children: 动态集合(live collection)

什么是: 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，效率高! 因为不用返回完整的属性。

缺: 每次访问集合，都会重新查找DOM树，降低效率

遍历:

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

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

遍历指定父元素下的所有后代元素: 2种:

1. 递归: 2步:

1. 定义函数仅遍历指定父元素下的所有直接子元素

2. 对每个直接子节点调用和父节点完全相同的操作

深度优先: 当一个节点同时拥有子节点和兄弟节点时，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。

2. 循环: 2步:

1. 定义迭代器:

迭代器: 可以依次获得每个后代元素节点的 专门对象

如何:

创建: var iterator=document.createNodeIterator(

parent, NodeFilter.SHOW\_ELEMENT, null, false

);

2. 循环调用迭代器，获得下一个节点对象:

var curr=iterator.nextNode()

内置深度优先遍历的算法

如果curr返回null，说明遍历结束

3. 按HTML查找: 4种:

1. 按id查找:

var elem=document.getElementById("id")

返回值: 一个元素

如果找不到返回null!

强调: 1. 只能在document上调用

2. 按标签名查找:

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

返回值: 多个元素的集合

如果找不到返回空集合

强调: 1. 可在任意父元素上调用

2. 不但找直接子元素，且在所有后代中查找

3. 按name查找:

var elems=document.getElementsByName("name")

返回值: 多个元素的集合

如果找不到返回空集合

强调: 只能在document上调用

4. 按class查找:

var elems=parent.getElementsByClassName("class")

返回值: 多个元素的集合

如果找不到返回空集合

强调: 1. 可在任意父元素上调用

2. 不但找直接子元素，且在所有后代中查找

3. 只要元素的一个class名匹配，就能找到该元素

强调: 返回的集合都是动态集合

问题: 每次只能按一个条件查找

当查找条件复杂时，步骤很繁琐

解决: 用选择器查找:

4. 用选择器查找:

1. 仅查找一个符合条件的:

var elem=parent.querySelector("选择器")

返回值: 一个元素

如果找不到,返回null

2. 查找多个符合条件的元素:

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

返回值: 多个元素的集合

如果找不到返回空集合

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

强调: 1. 可在任何父元素上调用

2. 选择器只要相对于当前父元素内部即可

3. 选择器的兼容性，受制于当前浏览器的兼容性

鄙视: 按HTML查找和按选择器查找的差别:

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

按选择器查找效率低

3. 易用性: 按HTML查找繁琐

按选择器查找简单

总结: 今后，只要一个条件即可找到想要的元素，首选按HTML查找

只要查找条件复杂，都选按选择器查找

jQuery中就是这么选择的

4. 修改:

内容: 3种:

1. html代码片段: elem.innerHTML 没有兼容性问题

2. 纯文本内容: elem.textContent 有兼容性问题

1. 去掉内嵌的标签

2. 翻译转义字符为正文

3. 表单元素的值: elem.value

属性:

样式:

DOM操作基本步骤:

0. 构建DOM树

1. 查找触发事件的元素

2. 绑定事件

3. 在事件处理函数中，查找要修改的元素

4. 修改元素(内容,属性,样式)