# DOM笔记

## 获取元素

### 获取指定元素

**document.getElementById: 通过id寻找一个元素（找到的是一个元素对象） 该方法只能被document对象调用（同一个文档中id不能重复）**

**一般 克隆 出来的元素 都要注意 ID**

<div id="box"></div>

var box = document.getElementById(“box”);

**document.getElementsByTagName: 通过标签名寻找一类元素（找到的是由元素对象组成的伪数组（只能用length属性）,既可以被document调用，又可以被元素对象调用，被元素对象调时表示在该元素对象内部执行查找**

<div class="cl" id=“cl”>

<div class="cl2"></div>

<div class="cl2"></div>

</div>

<div class="cl"></div>

<div class="cl"></div>

var divs = document.getElementsByTagName("div");// 获取页面上所有的div,divs是一个伪数组

var cl = document.getElementById("cl");// 获取id为cl的元素

var cl2s = cl.getElementsByTagName("div");// 获取cl元素下面所有的div标签,cl2s是一个伪数组

**document.getElementsByClassName通过类名寻找包含这个类名的元素**

<div class="cl main" id=“cl”>

<p class="cl fl"></div>

<span class="cl"></div>

</div>

<a class="cl"></a>

var cls = document.getElementsByClassName("cl");//获取到的是一个伪数组,里面装的是div p span a这四个元素对象(只要标签中的class中有传入的类名就能获取到)

### 获取子元素&兄弟元素&父元素

**element.childNodes: 获取指定元素的子节点,包括文本节点、元素节点等**  
**element.children: 获取指定元素的子元素,只会获取元素节点。**   
**element.nextSibling: 获取指定元素的下一个兄弟节点**  
**element.nextElementSibling: 获取指定元素的下一个兄弟元素**  
**element.previousSibling: 获取指定元素的上一个兄弟节点**  
**element.previousElementSibling: 获取指定元素的上一个兄弟元素**  
**element.parentNode: 获取元素的父节点,父节点肯定是一个元素**

<ul id="list">

<li id="firstlink"><a href="javascript:void(0)">首页</a></li>

<li><a href="javascript:void(0)">播客</a></li>

<li><a href="javascript:void(0)">博客</a></li>

<li><a href="javascript:void(0)">相册</a></li>

<li><a href="javascript:void(0)">关于</a></li>

<li id="lastlink"><a href="javascript:void(0)">帮助</a></li>

</ul>

// 通过id获取元素对象

var ul = document.getElementById("list");

// 通过标签名获取元素数组

var lis = ul.getElementsByTagName("li");

// 不关心层级 只找指定标签

// 缺点:如果内部还有li 也会找到

var nodes = ul.childNodes;

//子节点 只找子级

//缺点:除了我们想要的元素节点 还会获取到其他节点

var children = ul.children;//只获取ul下的子元素

var link = document.getElementById("firstlink");

var nextNodeSibling = link.nextSibling; 下个兄弟节点// 获取到的是换行(文本节点)

var nextElementSibling = link.nextElementSibling;下个兄弟元素//获取到的是下一个li标签

var parentNode = link.parentNode;// 获取到的是ul

### 获取第一&最后一个子元素

**element.firstChild: 获取指定元素下面的第一个子节点**  
**element.firstElementChild: 获取指定元素下面的第一个子元素**  
**element.lastChild: 获取指定元素下面的最后一个子节点**  
**element.lastElementChild: 获取指定元素下面的最后一个子元素**

var firstNode = ul.firstChild;// 获取到的是换行(文本节点)

var firstElement = ul.firstElementChild;// 获取到的是firstlink

var lastNode = ul.lastChild;// 获取到的是换行(文本节点)

var lastElement = ul.lastElementChild;// 获取到的是lastlink

## 节点操作

### 克隆节点 - cloneNode(true/false)

* element.cloneNode(): 复制element节点

参数:布尔值,

true代表深层克隆,把当前节点和内部所有节点都复制一份

false代表浅层克隆,只复制当前节点

<div id="father">

<div id="son"><div/>

</div>

var father = document.getElementById("father");

var son = document.getElementById("son");

var clone = son.cloneNode(true);// 把son这个div复制一份 复制出来的clone和son没有任何关系了

### 添加节点 - appendChild()

father.appendChild(son):将son节点追加到father内部的最后位置

<div id="father">

<div id="son"><div/>

</div>

<div id="demo"></div>

var father = document.getElementById("father");

var demo = document.getElementById("demo");

var clone = demo.cloneNode(true);// 将demo克隆一份

father.appendChild(clone);// 将克隆出来的clone追加到father中

// 此时页面结构应该为

<div id="father">

<div id="son"><div/>

<div id="demo"></div>

</div>

<div id="demo"></div>

#追加克隆节点对原节点不会产生影响

//如果代码如下 则会将demo节点直接移动到father节点下

father.appendChild(demo);// demo是页面上存在的节点

// 此时页面结构应该为

<div id="father">

<div id="son"><div/>

<div id="demo"></div>

</div>

### 插入节点 - insertBefore()

father.inserBefore(son1,son2): 将son1插入到father节点下的son2前面

<div id="father">

<div id="son"><div/>

</div>

<div id="demo"></div>

var father = document.getElementById("father");

var son = document.getElementById("son");

var demo = document.getElementById("demo");

father.inserBefore(son,demo);//会直接将demo节点移动到father下的son前面

#插入克隆出来的节点也不会对原节点产生影响

### 移除节点 - removeChild()

father.removeChild(son): 将father下的son节点移除

<div id="father">

<div id="son"><div/>

</div>

var father = document.getElementById("father");

var son = document.getElementById("son");

father.removeChild(son);// 直接将son节点删除

## 创建结构

### document.write()

**document.write("内容") 特点:只能被document调用,而且会覆盖页面上原有内容**

document.write("<a href="http://www.baidu.com">百度</a>")

// 可以在页面上创建一个a标签,而且会覆盖页面上原有的所有内容

### element.innerHtml

**element.innerHtml特点:往页面添加html标签,可以限定范围**

<div id="box"></div>

var box = document.getElementById("box");

box.innerHtml = "<a href="http://www.baidu.com">百度</a>";

// 追加后的结构为

<div id="box">

<a href="http://www.baidu.com">百度</a>

</div>

### document.createElement()

**document.createElement("内容") 特点:动态创建标签,添加到页面需要配合appendChild使用**

<div id="box"></div>

var box = document.getElementById("box");

var input = document.createElement("input");

input.type = "text";

box.appendChild(input);

## 标签的自定义属性操作

### setAttribute()

**设置标签属性: setAttribute()**

<div id="box"></div>

var box = document.getElementById("box");

box.setAttribute("id","aaa");// 有规定的属性可以设置

box.setAttribute("bbb","ccc");// 没有规定的属性也可以设置

### getAttribute()

**获取标签属性:getAttribute()**

<div id="box"></div>

var box = document.getElementById("box");

box.getAttribute("id");// 有规定的可以获取

box.getAttribute("aaa"); // 没有规定的也可以获取

### removeAttribute()

**移除标签属性:removeAttribute()**

<div id="box"></div>

var box = document.getElementById("box");

box.removeAttribute("id"); // 有规定的可以删除

box.removeAttribute("aaa"); // 没有规定的也可以删除

## 兼容方法

### innerText兼容写法

#### 获取文本

function getInnerText(element) {

// 能力检测 判断是否有这一属性

if (typeof element.innerText === "string") {

return element.innerText;

} else {

return element.textContent;

}

}

#### 设置文本

function setInnerText(element, content) {

// 能力检测 判断是否有这一属性

if (typeof element.innerText === "string") {

element.innerText = content;

} else {

element.textContent = content;

}

}

### 获取上一个&下一个兄弟元素的兼容写法

#### 获取下一个兄弟元素兼容写法

// 获取下一个兄弟元素

function getNextElement(element){

if(element.nextElementSibling){

//能找到nextElementSibling这个属性 就可以直接使用

return element.nextElementSibling;

}else{

var next = element.nextSibling;// 获取下一个兄弟节点

// 如果next就是想要的下一个兄弟元素 就直接返回 如果不是 就一直找

while(next&&next.nodeType !==1){//next找到了而且不是想要的元素节点就继续找

next = next.nextSibling;

}

return next;

}

}

#### 获取上一个兄弟元素兼容写法

// 获取上一个兄弟元素

function getPreviousElement(element){

if(element.previousElementSibling){

return element.previousElementSibling;

}else{

var previous = element.previousSibling;

while(previous&&previous.nodeType !== 1){

previous = element.previousSibling;

}

return previous;

}

}

### 通过类名获取元素对象的兼容方法

// 通过类名获取元素对象的兼容方法

function getElementByClassName(element,className){

if(element.getElementByClassName){// 可以使用这个属性

return element.getElementByClassName("className");

}else{

// 先找到element里面所有的标签 然后判断是否有需要的类名

// 如果有 就把这些标签放到一个集合中 最后返回这个集合

var elements = element.getElementByTagName("\*"); // 通配符找到所有标签

var filterArr = [];

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

if(elements[i].className.indexOf(className)!== -1){

filter.push(elements[i]);

}

}

return filterArr;

}

}

## 事件

var box = document.getElementById("box");

box.onclick = function(){

// 鼠标单击事件

}

box.dblclick = function(){

// 鼠标双击事件

}

box.onmouseover = function(){

// 鼠标移入事件

}

box.onmouseout = function(){

// 鼠标移出事件

}

box.onfocus = function(){

// 聚焦事件

}

box.onblur = function(){

// 失去焦点事件

}

box.onkeyup = function(){

// 按键弹起事件

}

box.onkeydown = function(){

// 按键按下事件

}

￼