08_Javascript_DOM操作02

特殊的节点

```
// 1. 文档节点
console.log(document);

// 2. html节点
console.log(document.documentElement);

//3. body节点
console.log(document.body);

//4. head节点
console.log(document.head);

//5. 文档声明节点
console.log(document.doctype);
```

通过节点关系找节点

找父节点

parentNode

节点对象.parentNode
返回父节点**元素对象**
如果没有父节点返回null

获取子节点

childNodes

节点对象.childNodes
返回子节点集合,包含**元素节点**、**文本节点**（慎用）

children

节点对象.children
返回子元素节点集合，只包含**元素节点**
非标准用法，但浏览器支持的很好，可以放心使用

获取第一个或最后一个子节点

firstChild

节点对象.firstChild
　返回第一个子节点对象,包含**元素节点**、**文本节点**

lastChild

节点对象.lastChild
　返回最后一个子节点对象,包含**元素节点**、**文本节点**等

firstElementChild

节点对象.firstElementChild
　返回第一个子**元素节点**,IE9以下不支持

lastElementChild

节点对象.lastElementChild
　返回最后一个子**元素节点**,IE9以下不支持

获取兄弟节点

previousSibling

节点对象.previousSibling
　返回上一个兄弟节点，包含**元素节点**、**文本节点**等; 不存在返回null

nextSibling

节点对象.nextSibling
　返回下一个兄弟节点，包含**元素节点**、**文本节点**等; 不存在返回null

previousElementSibling

节点对象.previousElementSibling
　返回上一个兄弟节点，只包含**元素节点**
　IE9以下不支持

nextElementSibling

节点对象.nextElementSibling
　返回下一个兄弟节点，只包含**元素节点**
　IE9以下不支持

示例代码

```
//1. 获取某个节点的所有子节点 childNodes
console.log(document.head.childNodes);

//2. 获取某个节点的所有元素子节点 children
console.log(document.head.children);
```

```
//3. 获取某个节点里的第一个子节点
console.log(document.head.firstChild);

//4. 获取某个节点里的最后一个子节点
console.log(document.head.lastChild);

//5. 获取某个节点里的第一个元素子节点
console.log(document.head.firstElementChild);

//6. 获取某个节点里的最后一个元素子节点
console.log(document.head.lastElementChild);

//7. 获取某个节点的父节点(父节点有且只有一个, document没有父节点)
console.log(document.body.parentNode.parentNode);

//8. 找到某个节点的上一个兄弟节点
console.log(document.body.previousSibling);

//9. 找到某个节点的上一个兄弟元素节点
console.log(document.body.previousElementSibling);

//10. 找到某个节点的下一个兄弟节点
console.log(document.head.nextSibling);

//11. 找到某个节点的下一个兄弟元素节点
console.log(document.head.nextElementSibling);
```

封装一个兼容性强的寻找下一个节点的函数

```
function nextBrother(element) {
        while(element = element.nextSibling){
            if(element.nodeType === 1) {
                return element
            }
        }
        return null;
    }
```

元素节点对象操作

创建节点对象

document.createElement("标签名称")

添加节点对象

父节点对象.appendChild(子节点)
父节点对象.insertBefore(新子节点, 子节点)

克隆节点对象

节点对象.cloneNode()
如果参数不写或为false，则为浅拷贝，只复制节点本身不复制子节点
如果参数为true，则为深拷贝，会复制节点本身及所有子节点

> 如果用=赋值的方式只是赋值了一个节点对象的引用，而内存中还是同一个对象。

删除节点对象

父节点对象.removeChild(子节点对象)

替换子节点对象

父节点对象.replaceChild(新节点,旧节点)

动画函数封装

缓动动画原理
　缓动动画就是让元素运动速度有所变化，最常见的是让速度慢慢停下来
思路：

1. 让盒子每次移动的距离慢慢变小，速度就会慢慢落下来。
2. 核心算法：(目标值 - 现在的位置) / 10　做为每次移动的距离步长
3. 停止的条件是： 让当前盒子位置等于目标位置就停止定时器
4. 注意步长值需要取整

判断步长是正值还是负值
1.如果是正值，则步长往大了取整

2.如果是负值，则步长向小了取整

5.可以添加一个回调函数，执行动画完成之后想要做的事情

```
function animate(element, target, callback = null, speed = 15) {
    clearInterval(element.timer);       //清除原有定时器
    element.timer = setInterval(function() {
        var left = element.offsetLeft;          //原位置
        var step = (target - element.offsetLeft) / 10;    //步长
        step = step > 0 ? Math.ceil(step) : Math.floor(step);       //步长取整
        if (left != target) {
            element.style.left = left + step + 'px';        //改变位置
        } else {
```

```
            clearInterval(element.timer);             //到达目标位置停止动画
            callback && callback();         //处理回调函数
        }
    }, speed);
};
```

效率问题

减少js操作DOM

不管什么时候，只要是查询DOM中的某些元素，浏览器都会搜索整个DOM树，从中查找可能匹配的元素。所以效率很低，我们可以尽量只做一次查找把结果保存在变量中，尤其是对节点集合的遍历操作更加影响效率。

简化DOM

尽量减少文档中的标签数量，简化DOM结构

合并js文件

如果一个html引用多个外部js文件，如果可以合并，尽量合并这些js文件，因为多次引用会造成多次请求数量，浪费带宽。

js引入位置

脚本在html中的引入位置也影响效率。放在head标签中会印象文档内其他静态资源的加载，一般来说，根据HTTP规范，浏览器每次从同一个域名中最多只能同时下载两个文件。而在下载脚本同时，浏览器不会下载其他任何文件，即使是来自不同域名的文件也不会下载，所有其他资源都要等脚本加载完毕后才能下载。

更好的做法是把所有的script标签放在文档末尾< /body>之前。

压缩脚本

删除代码中的空白，回车，注释，可以减少网络传递的流量浪费。