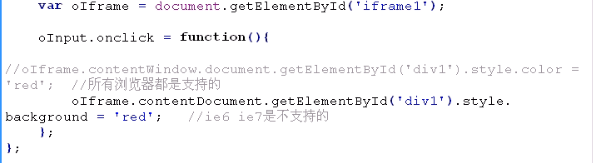
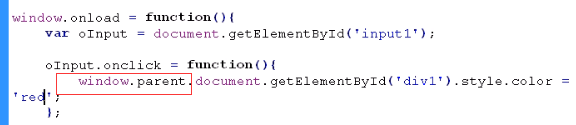
# 操作iframe



iframe内部操作外部：



# 1 cookie

cookie的作用：存储信息，一般同一个网站的所有页面共享一套cookie。

我们使用变量存储数据，那么关闭浏览器时，对象就会被销毁，而cookie则不会。

cookie的一些特点：

1不同的浏览器存放的cookie的位置不一样，且不能通用；

2 cookie的存储是以域名形式进行区分的；

3 cookie的数据可以设置名字：document.cookie = ‘name=zs’;

4 一个域名下存储的cookie个数有限制（不同浏览器不同版本也不一致）；

5 每个cookie存放的内容大小也有限制（不同浏览器存放大小不一致）。

多个cookie的设置和显示

document.cookie = ‘name=zs’;

document.cookie = ‘age=30’;

console.log(document.cookie); //输出： name=zs; age=30

只单单做上述的设置，是不能让cookie进行长时间保存的。我们需要给cookie设置一个过期的时间expires。

var date = new Date();  
date.setDate(date.getDate() + 5); *//根据来访者时间动态增加  
//日期必须是一个字符串*document.cookie = 'name=zs;expires=' + date.toGMTString();  
document.cookie = 'age=30';  
console.log(document.cookie);

注意：当我们存储的数据有一些特殊字符时，比如name=\nzs，会出现粗出数据不对的情况，这里通常使用URI编码处理：

document.cookie = 'msg=' + encodeURI('zs\n你好');  
console.log(decodeURI(document.cookie));

cookie设置、获取、移除的方法封装：

function setCookie(key, value, t) {  
 var oDate = new Date();  
 oDate.setDate( oDate.getDate() + t );  
 document.cookie = key + '=' + value + ';expires=' + oDate.toGMTString();  
}  
  
function getCookie(key) {  
 var arr1 = document.cookie.split('; ');  
 for (var i=0; i<arr1.length; i++) {  
 var arr2 = arr1[i].split('=');  
 if ( arr2[0] == key ) {  
 return decodeURI(arr2[1]);  
 }  
 }  
}  
  
function removeCookie(key) {  
 setCookie(key, '', -1);  
}

# 2 DOM性能优化

## 2.1 JS与DOM分离

浏览器会将DOM与JS分开独立实现，如果使用JS操作DOM就像从一个区域跳转到了另外一个区域，每次跳转都会消耗性能，所以要尽量减少这种操作。

我们使用案例来展示：

var div = document.getElementById('div');  
console.time('hi');  
*//5000次来实现添加html*for(var i = 0; i < 5000; i++){  
 div.innerHTML += 'a';  
}  
console.timeEnd('hi');*//chrome下输出hi: 241.7109375ms*

那么根据刚才所述的提升性能的办法，我们将循环放置在独立的JS中，循环完毕后一次操作DOM：

var div = document.getElementById('div');  
console.time('hi');  
var html = '';  
for(var i = 0; i < 5000; i++){  
 html += 'a';  
}  
div.innerHTML = html;  
console.timeEnd('hi');*//chrome下输出hi: 7.304931640625ms*

## 2.2 appendChild与innerHTML

给DOM添加元素，有appendChild和innerHTML两种方式，但是在不同内核下，他们的效率却不同：

webkit内核：操作DOM性能比使用innerHTML性能好

其他内核：操作innerHTML性能更好

## 2.3 减少DOM操作

### 2.3.1 节点克隆

方式一：性能低

console.time('hi');  
var oUl = document.getElementById('ul');  
for(var i = 0; i< 5000; i++){  
 var oLi = document.createElement('li');  
 oLi.innerHTML = 'li';  
 oUl.appendChild(oLi);  
}  
  
console.timeEnd('hi');*//chrome下输出hi: 28ms*

方式二：使用clone

console.time('hi');  
var oUl = document.getElementById('ul');  
var oLi = document.createElement('li');  
oLi.innerHTML = 'li';  
for(var i = 0; i< 5000; i++){  
 var newLi = oLi.cloneNode(true);  
 oUl.appendChild(newLi);  
}  
  
console.timeEnd('hi');*//chrome下输出hi: 8ms*

clone性能更好的原因：clone不是新创建节点。

### 2.3.2 局部变量访问元素集合

比如我们使用一个循环 for(var i = 0; i < arr.length; i++)

这里我们每次进入循环，都要使用一个arr.length ，会严重影响性能，我们可以顶一个变量存储这个长度，使用变量进行循环，对性能会有所提升。

其他示范：比如我们经常使用到了 document.get\*\*，这里反复的使用document都会影响性能，同样我们也可以定义一个变量保存document，使用这个便刘昂来提升性能。

### 2.3.3 只获取元素的节点的方法

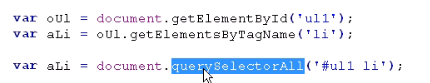
我们获取元素的节点有很多方式，比如下列两种：

childNodes： 获取元素的节点和文本节点

children： 只获取元素节点

这里可以发现children消耗的性能低一点。同理还有：firstElementChild比firsChild性能高。

### 2.3.4 利用query选择器



后者的写法性能要高，但是IE8以及以下版本不支持新的选择器。

### 2.3.5 重排与重绘-appendChild

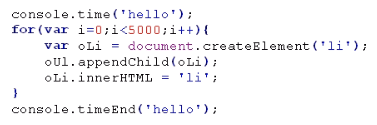
浏览器重排：通过DOM操作改变了页面元素的位置、形状。

浏览器重绘：重排后的内容显示出来的过程叫做重绘。

注意：改变背景颜色不会触发重排，只会触发重绘。

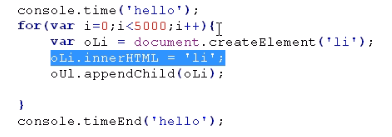
重排与重绘都会影响性能。

我们需要尽量在appendChild钱添加重排、重绘操作。

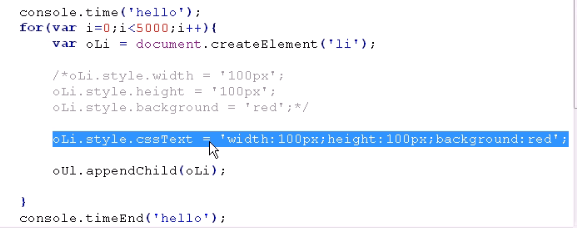


运行时间在：750ms

调换位置后的运行时间：440ms

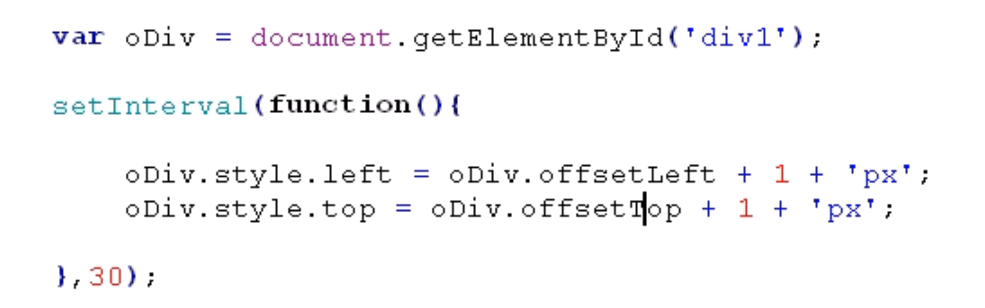


### 2.3.6 重排与重绘-合并css

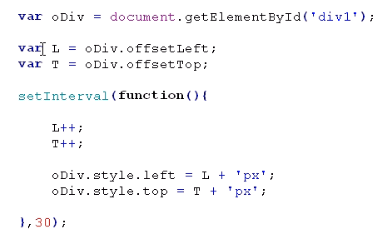


把三句CSS操作合并为一句，使用cssText属性进行添加，可以优化性能。

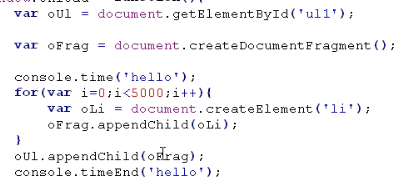
### 2.3.7 重排与重绘-缓存布局信息

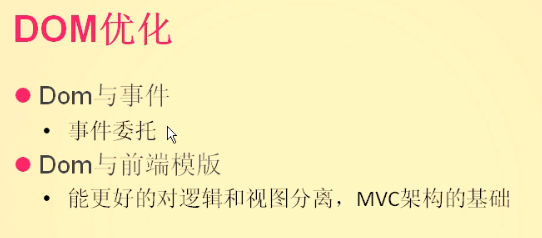


布局卸载了setInterval中，每次执行都会获取offsetLeft这样的DOM元素，我们可以将其用一个变量来记录。



### 2.3.8 重排与重绘-文档碎片



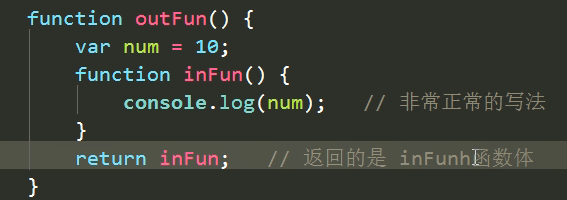


# 3 闭包

### 1.1 闭包简介

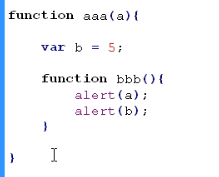
闭包，是指语法域位于某个特定的区域，具有持续参照（读写）位于该区域内自身范围之外的执行域上的非持久型变量值能力的段落。这些外部执行域的非持久型变量将保留他们在闭包最初定义时创建的值。

即：一个函数 访问另一个函数内部的变量方式。



### 1.2 闭包的优缺点

闭包的缺点：闭包中的数据会常驻内存，不用时需要手动删除；



参数a和变量b都在闭包bbb中被引用了，不会被JS的垃圾回收机制收回

案例代码：

function aaa() {  
 let a = 5;  
 function bbb() {  
 console.log(a);  
 }  
 return bbb;  
}  
  
let c = aaa();  
c(); *//这里仍然会输出5*

闭包的优点：不产生全局变量，实现属性私有化（避免全局变量污染）；

let a = 1;  
function aaa() {  
 a++;  
 console.log(a);  
}  
aaa();  
console.log(a); *//a的值改变了*

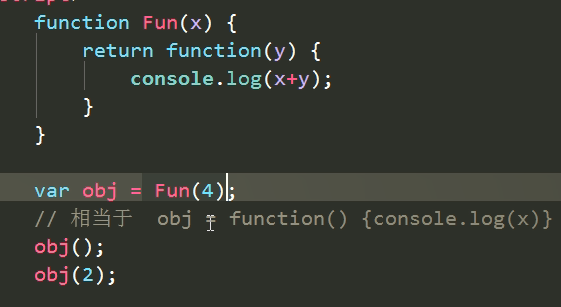
使用局部变量后：

function aaa() {  
 let a = 1;  
 a++;  
 console.log(a);  
}  
aaa(); *//每次调用都不变*aaa(); *//每次调用都不变*

但是这样我们无法获取内部的变量值，需要如下写法：

function aaa() {  
 let a = 1;  
 return function () {  
 a++;  
 console.log(a);  
 }  
}  
let b = aaa();  
b(); *//输出2*b(); *//输出3*

### 1.3 闭包的传参



# 4 对象

基本数据类型：string number boolean null undefined

对象数据类型：Array 对象

对象数据类型就是带有属性和方法的数据类型。

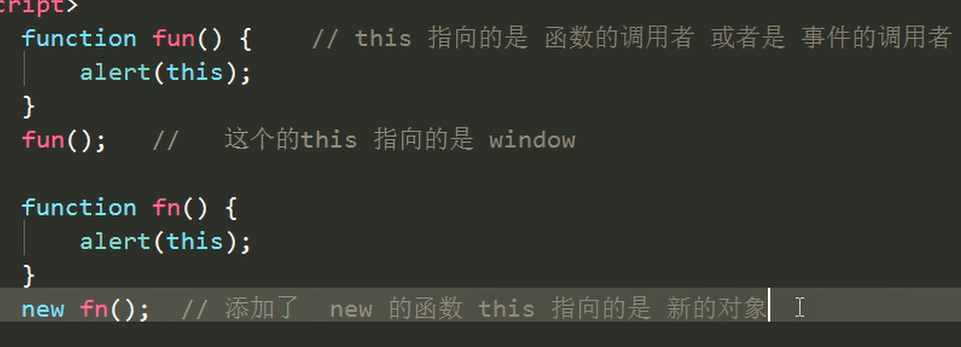
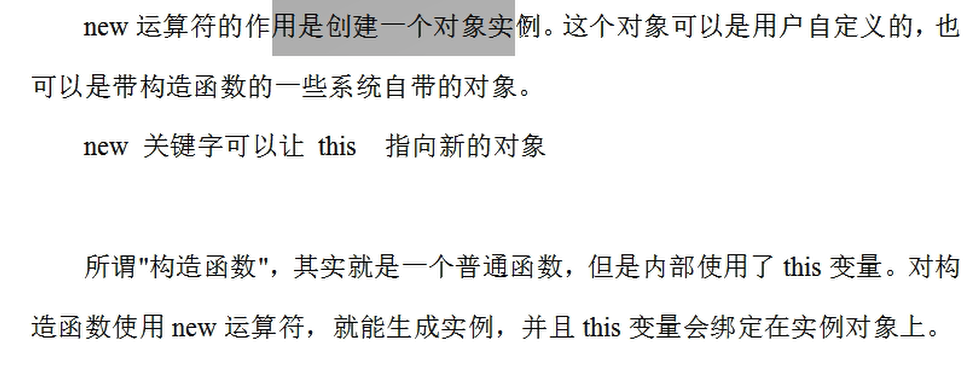
声明对象的两种方式

var obj = new Object();

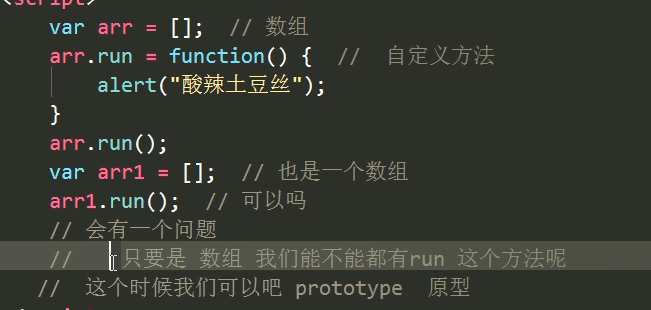
我们提倡使用第二种方式，使用字面量声明：

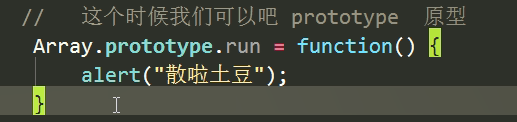
var obj = {};

## this



## 原型的用处





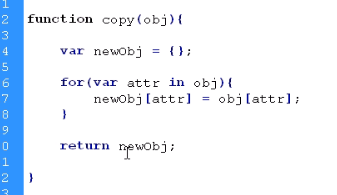
同理，构造函数也类似于Array，使用原型方式添加方法：



# 事件委托：

如果点击ul中的li，会触发事件，那么我们可以给这个li来添加事件，也可以利用冒泡机制直接给ul添加事件，这种情况很类似事件委托，也提升了性能。

浅拷贝：只是复制了一个新对象



深拷贝：拷贝了这个对象的值和引用地址

需要利用递归解决对象内部还有对象~

