**该文档需配合 <https://github.com/mqyqingfeng/Blog/issues/10> 食用**

**作者：Jack Lei  
链接：https://www.zhihu.com/question/22473205/answer/156101876  
来源：知乎  
著作权归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。**

**变量赋值分为两种，简单数据类型复制是内存的内容的复制，引用类型的复制，是指针的复制，复制后两个变量指向同一内存地址。**

**我们来看看实例吧！**

**简单数据类型复制：**

**var a = 10;**

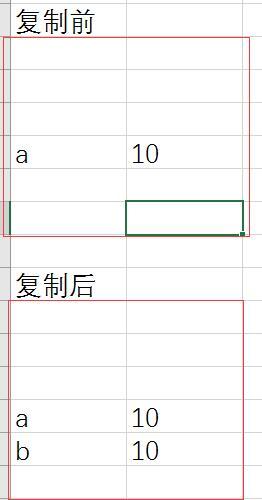
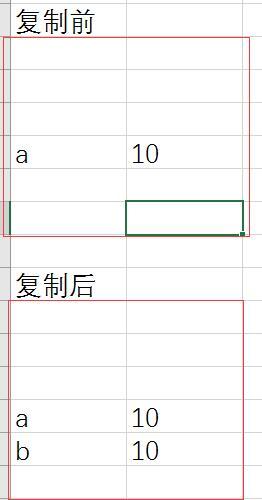
**var b = a; //将a内存的内容就是值填充到 b 的内存中，a b 有各自的内存空间，相互独立**

**console.log(a);//10**

**console.log(b);//10**

**console.log(a == b);//true a,b相等**

**看看复制前后的内存变化**

****

**红框内是内存，简单数据类型的赋值是内存内容复制，变量的内存是独立的，只是内存的内容相同而已。**

**引用数据类型的复制，**

**var obj1 = {name:"jack"};**

**var obj2 = obj1; //对象复制**

**for(var n in obj1){**

**console.log(n); //name**

**}**

**for(var m in obj2){**

**console.log(m); //name**

**}**

**console.log(obj1 === obj2);//true,两个对象相等**

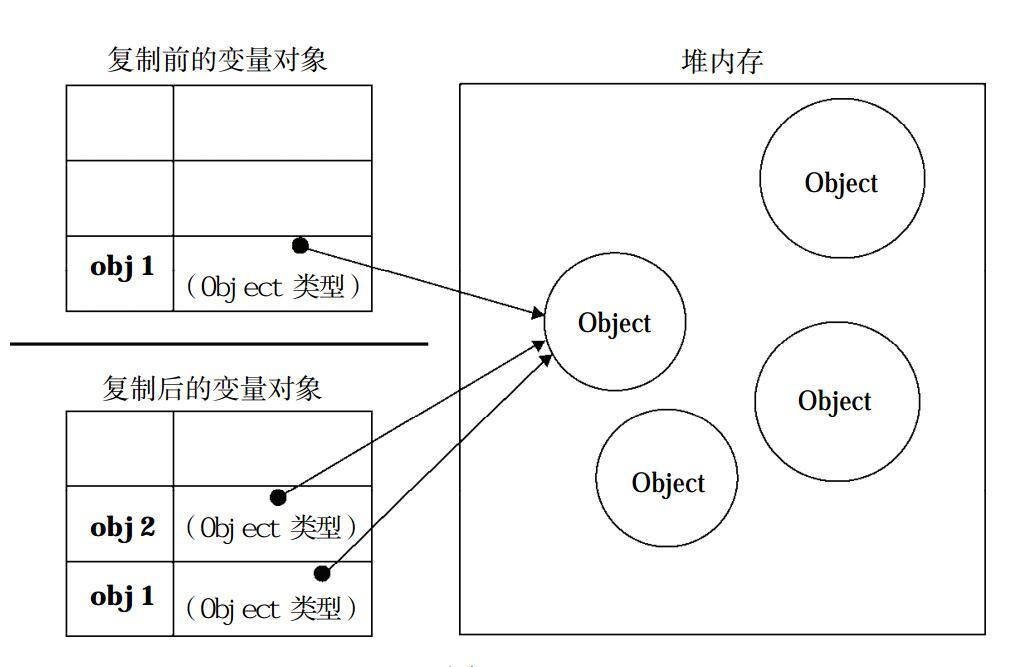
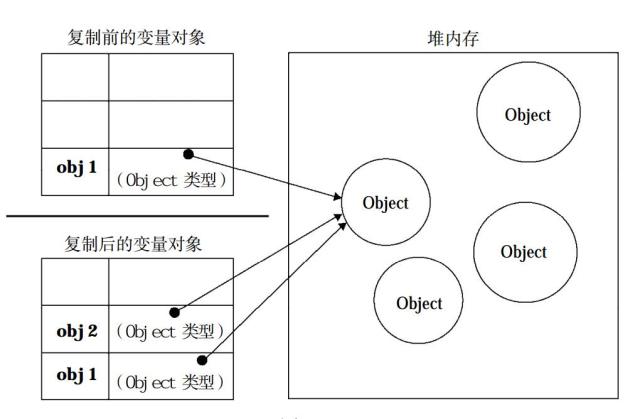
**//修改 obj2 的值，看看 obj1的变化**

**obj2.name = "zhou";**

**console.log(obj1.name);//zhou,obj1 也修改了**

**console.log(obj2.name);**

**来看看为何会修改：**

****

**对象存在堆内存中（把它想成一种存储方式好了），大小不固定，在栈中有一个指针记录着这个对象在堆中地址，指针的大小是固定的。但我们要复制对象时，是复制对象的指针，复制后，两个变量同时指向一块堆地址，也就是两个地址指向同一对象了，所以指向同一个地址的对象才会相等，那么改变其中一个对象，其实就是改变堆中的对象，所以另一个变量也改变了。**

**从对象复制，我们可以看出，判断两个对象是否相等，要看它们是否指向同一地址，而不是看对象的属性或者方法是否一样。**

**var obj3 = {name:"jack"};**

**var obj4 = {name:"jack"};**

**console.log(obj3 == obj4);//false 两个对象的属性相同，但是不相等**

**函数参数传递**

**是按值传递的。**

**function fn(a,b,c){**

**a=a+10;**

**b.name='jack';**

**c={name:'Tom'};**

**}**

**var n=10;**

**var obj1={name:'zhou'};**

**var obj2={name:'Bob'};**

**fn(n,obj1,obj2);**

**console.log(n);//10,fn 没有改变 n 的值**

**console.log(obj1.name);//zhou,fn 改变了 obj1**

**console.log(obj2.name);//Bob,fn 没改变 obj2 怎么会这样？看下面的分析**

**来说说值传递和引用传递的区别：**

**按值传递，函数参数a，b , c 都会得到新的内存，实参只是把内存中的值复制给形参，改变形参的值不会改变实参的值。是内存内容的复制。如果是传递值，则函数外边的 n ,obj1，obj2 不会改变，但是 obj1 改变了。**

**按引用传递，实参和形参共同指向同一内存地址，改变形参，也会改变实参。如果按引用传递，n 和 obj2 要改变，而实际没有改变。**

**其实是叫 call-by-sharing 的传参方式，中文叫 传递引用的拷贝。**

**fn()中的a, b, c都是一个引用**

**他们的内容是某块内存的地址**

**这个地址的值来自于外部定义的三块内存，是那三块内存地址的一份拷贝**

**同时，在函数内部这三个参数的值是可以直接被修改的，可以指向其他对象（由于JavaScript中没有指针或引用运算符，只能直接修改）**

**function fn(a,b,c){**

**a=a+10; //对 a 赋新值，**

**b.name='jack'; //修改 b 的属性**

**c={name:'Tom'};//改写了 c 的指向，赋新值**

**}**

**call-by-sharing是值传递的一种，所有 JS 传递参数是按值出传递的。**

**那么如何判断改变函数形参，对实参的影响呢？**

**差别就是： 是在函数内对参数赋新值还是改写参数的属性。如果对参数赋新值，则对实参没有影响，改写参数的属性，就会改写实参。明白 obj1 改变而 obj2 没有改变了么？**

**参考链接：**

**<https://stackoverflow.com/questions/518000/is-javascript-a-pass-by-reference-or-pass-by-value-language>**

**It's interesting in JavaScript. Consider this example:**

**function changeStuff(a, b, c){**

**a = a \* 10;**

**b.item = "changed";**

**c = {item: "changed"};**

**}**

**var num = 10;**

**var obj1 = {item: "unchanged"};**

**var obj2 = {item: "unchanged"};**

**changeStuff(num, obj1, obj2);**

**console.log(num); // 10**

**console.log(obj1.item); // changed**

**console.log(obj2.item); // unchanged**

**If obj1 was not a reference at all, then changing obj1.item would have no effect on the obj1 outside of the function.**

**If the argument was a proper reference, then everything would have changed. num would be 100, and obj2.item would read "changed".**

**Instead, the situation is that the item passed in is passed by value. But the item that is passed by value is itself a reference. Technically, this is called **[call-by-sharing](http://en.wikipedia.org/wiki/Evaluation_strategy" \l "Call_by_sharing)**.**

****In practical terms, this means that if you change the parameter itself (as with num and obj2), that won't affect the item that was fed into the parameter. But if you change the INTERNALS of the parameter, that will propagate back up (as with obj1).****

****注意： 按引用传递是传递对象的引用，而按共享传递是传递对象的引用的副本！****

所以修改 o.value，可以通过引用找到原值，但是直接修改 o，并不会修改原值。所以第二个和第三个例子其实都是按共享传递。

****正是因为传递的是引用的副本(拷贝) => 用按值传递来理解，创立了新的内存，保存了引用。这样当改变形参本身时，就不会影响到函数外的值，因为已经开辟了新的内存。****

****如果是简单的按引用传值，那在函数中改变形参就会影响到函数外的原值。****