一直认为在JavaScirpt中，数据只有两种访问方式：基本类型数据是按值访问的，而引用类型的值是按引用访问的。  
但是红宝书上还说，传递参数时，不论是什么类型的值，它们都是按值传递的。

按值传递(call by value)是最常用的求值策略：函数的形参是被调用时所传实参的副本。修改形参的值并不会影响实参。

按引用传递(call by reference)时，函数的形参接收实参的隐式引用，而不再是副本。这意味着函数形参的值如果被修改，实参也会被修改。同时两者指向相同的值。

书上举了个例子：

function setName(obj) {  
 obj.name = "Nicholas";  
 obj = new Object();  
 obj.name = "Greg";  
}  
var person = new Object();  
setName(person);  
alert(person.name); //Nicholas

第一次看到这章的时候，我看的云里雾里的，也查了不少资料，最后在旁边写了两行注释：在不为person赋值时，函数内外地址相同，所以操作可以反应在外部；反之，地址不同，则内外联系就断了。

但是总感觉说的还不到位，而且事实上我自己也并不理解具体意思。但是我当时学的不多，也不知道网上什么是对的什么是错的，也就放着了。

今天我又突然想到了这个问题，也搜了不少资料，知道了共享传递这个名词：（接下来是一段引用）

准确的说，JS中的基本类型按值传递，对象类型按共享传递的(call by sharing，也叫按对象传递、按对象共享传递)。最早由Barbara Liskov. 在1974年的GLU语言中提出。该求值策略被用于Python、Java、Ruby、JS等多种语言。

该策略的重点是：调用函数传参时，函数接受对象实参引用的副本(既不是按值传递的对象副本，也不是按引用传递的隐式引用)。 它和按引用传递的不同在于：在共享传递中对函数形参的赋值，不会影响实参的值。

接下来是我的理解。

其实对象A在被赋值到变量B的时候，复制的是一份指向在堆内存的对象Object的指针给B，现在对象A和B都指向对象Object，如果我单纯的给A或者B加上一个属性，比如A.name，那么是不影响他们指针的指向的，他们依然指向Object，那么既然他们还引用同一个对象，那么属性当然共享，所以B也会有name属性。但是现在如果有一句B = {name:Lisa}，这句语句代表什么呢？起始可以拆开来看

var C = {name:Lisa};

B = C;

现在B已经不再指向A指向的那个Object了，他现在指向C的Object1,既然指针都变了，那么改变B，当然对A毫无影响，但是B和C之间的属性变换会相互影响了。

而且在我自己的测试下，我感觉在函数传参是按值传递和按共享传递（与按引用传递无关）这句话并不完整。应该是在JavaScript中，都没有按引用传递这种说法，我们认为的按引用传递，其实都是按共享传递

再来一个简陋的例子

var a = {name:1};  
var b = a;  
console.log(a.name);//1  
console.log(b.name);//1  
b.name = 3;  
console.log(a.name);//3  
a = {value:100};  
console.log(b.name);//3  
console.log(a.name);//undefined

简单的变量赋值也和在函数中参数的表现一样，所以我觉得在JavaScript中，基本类型数据按值传递，引用类型数据（如Object）按共享传递