一、基本类型传递方式

由于js中存在**复杂类型**和**基本类型**，对于**基本类型**而言，是**按值传递**的。

```
var a = 1
function test(x) {
 x = 10
 console.log(x)
}
test(a) // 10
console.log(a) // 1
```

虽然在函数 test 中 a 被修改，并没有影响到外部 a 的值，说明基本类型是**按值传递**的。

二、复杂类型按引用传递?

我们将外部 a 作为一个对象传入 test 函数

```
var a = {
 a: 1,
 b: 2
}
function test(x) {
 x.a = 10
 console.log(x)
}
test(a) // {a: 10, b: 2}
 console.log(a) // {a: 10, b: 2}
```

可以看到，在函数体内被修改的 a 对象同时也影响到了外部的 a 对象，可见复杂类型是**按引用传递**的。

再做一个实验：

```
var a = {
 a: 1,
 b: 2
```

```
}
function test(x) {
 x = 10
 console.log(x)
}
test(a) // 10
console.log(a) // {a: 10, b: 2}
```

外部的 a 并没有被修改，如果是按引用传递的话，由于共享同一个堆内存， a 在外部也会表现为 10 才对，此时的复杂类型同时表现出了 按值传递 和 按引用传递 的特性。

三、按共享传递

复杂类型之所以会产生这种特性，原因就是在传递过程中，对象 a 先产生了一个 副本a ，这个 副本a 并不是深克隆得到的 副本a ， 副本a 地址同样指向对象 a 指向的内存地址。

因此在函数体内修改 x = 10 只是修改了 副本a ， a 对象没有变化，但是如果修改了 x.a = 10 是修改了两者指向的同一堆内存，此时对象 a 也会受到影响。

有人将这种特性叫做**传递引用**，也有一种说法叫做**按共享传递**。