**学习目标：**

1、基本类型和引用类型的值

2、动态的属性

3、复制变量值

4、传递参数

5、检测类型

**一、基本类型和引用类型的值**

基本类型：Undefined 、 Null 、 Boolean 、 Number 和 String 。这 5 种基本数据类型是按值访问的，因为可以操作保存在变量中的实际的值。

引用类型的值是保存在内存中的对象，在操作对象时，实际上是在操作对象的引用而不是实际的对象。为此，引用类型的值是按引用访问的。

**二、动态的属性**

如1：

var person = new Object();

person.name = "八维";

alert(person.name);

如2：

var name = "八维";

name.age = 20;

alert(name.age); //undefined

结论：只能给引用类型值动态地添加属性

**三、复制变量值**

基本类型的复制：会在变量对象上创建一个新值，然后把该值复制到为新变量分配的位置上。

如1:

var num1 = 5;

var num2 = num1;

说明：num2中的 5 与 num1 中的 5 是完全独立的



引用类型的复制：当从一个变量向另一个变量复制引用类型的值时，同样也会将存储在变量对象中的值复制一份放到为新变量分配的空间中。不同的是，这个值的副本实际上是一个指针，而这个指针指向存储在堆中的一个对象。复制操作结束后，两个变量实际上将引用同一个对象。因此，改变其中一个变量，就会影响另一个变量

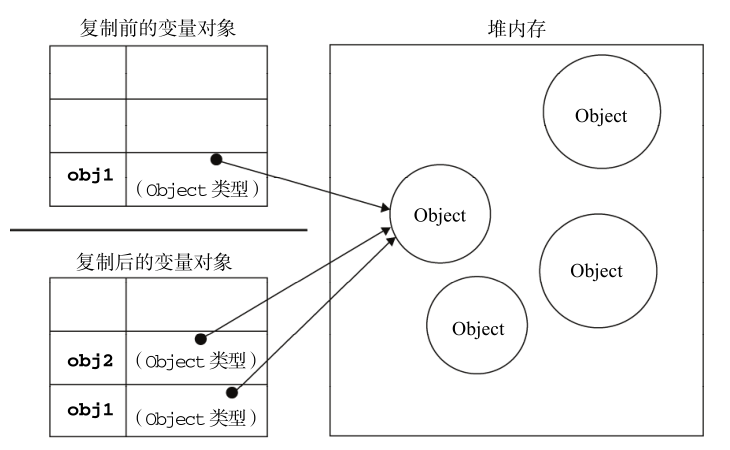
如：

var obj1 = new Object();

var obj2 = obj1;

obj1.name = "八维";

alert(obj2.name); //"八维"



**四、传递参数**

ECMAScript 中所有函数的参数都是按值传递的。

把函数外部的值复制给函数内部的参数，就和把值从一个变量复制到另一个变量一样。

基本类型值的传递如同基本类型变量的复制一样

引用类型值的传递，则如同引用类型变量的复制一样。

如1：

function addTen(num) {

num += 10;

return num;

}

var count = 20;

var result = addTen(count);

alert(count); //20，没有变化

alert(result); //30

如2：

function setName(obj) {

obj.name = "八维";

}

var person = new Object();

setName(person);

alert(person.name); //"八维"

**五、检测类型**

typeof：确定一个变量是字符串、数值、布尔值，还是 undefined 的最佳工具。

在检测function时，返回function

在检测null和对象时，返回的是Object

结论：在检测基本数据类型时 typeof 是非常得力的助手，但在检测引用类型的值时，这个操作符的用处不大。

instanceof：运算符

格式：对象 instanceof 构造函数

返回值：如果变量是给定引用类型，则返回true

alert(person instanceof Object); // 变量 person 是 Object 吗？

alert(colors instanceof Array); // 变量 colors 是 Array 吗？

alert(pattern instanceof RegExp); // 变量 pattern 是 RegExp 吗？

说明：所有引用类型的值都是 Object 的实例，因此，在检测一个引用类型值和 Object 构造函数时， instanceof 操作符始终会返回 true 。

instanceof 操作符检测基本类型的值，则该操作符始终会返回 false ，因为基本类型不是对象。

例：

var box = [1,2,3];

alert(box instanceof Array); //是否是数组

var box2 = {};

alert(box2 instanceof Object); //是否是对象

var box3 = /g/;

alert(box3 instanceof RegExp); //是否是正则表达式

var box4 = new String('Lee');

alert(box4 instanceof String); //是否是字符串对象

PS：当使用instanceof检查基本类型的值时，它会返回false。