1.单例模式: var single = {}; 就是最简单的单例模式;

2.可以多次new的单件模式,方案是用全局变量来保存这个唯一的实例，构造器看起来像：

function Logger(){

    if(typeof global\_log === "undefined"){

            global\_log = this;

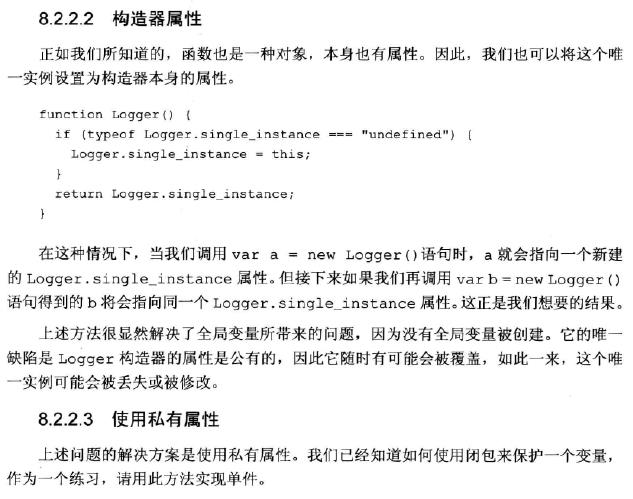
     }

    return global\_log;

}

var a = new Logger();var b= new Logger(); alert(a === b);//true

这样做的缺点是使用了全局变量,它在任何时候都有可能被覆盖掉，从而导致这个实例被丢失，它也有可能覆盖掉其它的对象。



构造函数用于定义实例属性,原型模型用于定义方法和共享的属性。

function Person(){}

var person = new Person();

Person.prototype = {constructor:Person,name:'aa'....}

尽管可以随时为原型添加属性和方法，并且修改能够立即在所有对象实例中反映出来，但如果是重写整个原型对象，那么情况就不一样了。我们知道，调用构造函数时会为实例添加一个指向最初原型的\_proto\_指针，而把原型修改为另一个对象就等于切断了构造函数与最初原型之间的联系。

包含引用类型值的原型属性会被所有实例共享。

在通过原型链实现继承时，不能使用对象字面量创建原型方法。因为这样做就会重写原型链。如

function SuperType(){...}

function SubType(){...}

SubType.prototype = new SuperTyper();

SubType.prototype = {....} //使用字面量添加方法,会导致上一行代码无效。上一行刚刚把SuperType的实例赋值给原型，紧接着又将原型替换成一个对象字面量而导致的问题，现在的原型包含的是一个Object的实例，而非SuperType的实例，所以原型链被切断了。SuperType和SubType没有关系了。

在创建子类类型的实例时，不能向超类型的构造函数中传递参数（如果有超类中有引用类型的话，会被影响）。

上次讲到的原型链中，原型链存在一个问题就是不能向超类型的构造函数传递参数。那么这次就是要实现如何向超类型构造函数传递参数。

这种方法我们称之为**借用构造函数（constructor stealing）**

这里的实现方法是使用js的原生方法apply()或all()。那么先温习下apply()和all()函数的知识。

**call方法:**

语法：call([thisObj[,arg1[, arg2[,   [,.argN]]]]])

定义：调用一个对象的一个方法，以另一个对象替换当前对象。

说明：

call 方法可以用来代替另一个对象调用一个方法。call 方法可将一个函数的对象上下文从初始的上下文改变为由 thisObj 指定的新对象。

如果没有提供 thisObj 参数，那么 Global 对象被用作 thisObj。

**apply方法：**

语法：apply([thisObj[,argArray]])

定义：应用某一对象的一个方法，用另一个对象替换当前对象。

说明：

如果 argArray 不是一个有效的数组或者不是 arguments 对象，那么将导致一个 TypeError。

如果没有提供 argArray 和 thisObj 任何一个参数，那么 Global 对象将被用作 thisObj， 并且无法被传递任何参数。

call和apply方法通常被用来实现类似继承一样的功能，以达到代码复用的功效。它们的区别主要体现在参数上。

看例子：

copycode.gif

复制代码

function SuperType() {
this.colors = ["red", "green", "blue"];
}
function SubType() {
//继承了SuperType
SuperType.call(this);
}
var instance1 = new SubType();
instance1.colors.push("black");
alert(instance1.colors); //red,green,blue,black
var instance2 = new SubType();
alert(instance2.colors); //red,green,blue

copycode.gif

复制代码

通过call方法继承了SuperType。我们实际上是在新创建的SubType实例的环境下调用了SuperType构造函数。即执行了SuperType函数中定义的所有对象的初始化代码。

如何给超类型构造函数传递参数：

copycode.gif

复制代码

function SuperType(name) {
this.name = name;
}
function SubType() {
//继承了SuperType
SuperType.call(this, "yangjinjin");
this.age = 27;
}
var obj = new SubType();
alert(obj.name); //yangjinjin
alert(obj.age); //27

copycode.gif

复制代码

以上代码，SuperType构造函数接受一个name参数，直接赋给this.name属性。在调用SubType构造函数内部调用SuperType构造函数时，实际上是为SubType的实例设置name属性。

共用构造函数的问题：

方法都在构造函数内部定义，函数的复用性就无从谈起了。在超类型的原型中定义的方法，对子类型而言是不可见的。考虑这些问题，借用构造函数也是很少单独使用