目录

1：什么是闭包 1

2：闭包的特点 2

3：闭包的原理 2

4：闭包的应用场景 3

5：使用闭包注意点 3

应用： 3

6：构造函数继承 3

7：call/apply继承 5

8：prototype的概念 6

9：原型链的继承 8

10：混合继承 9

应用： 10

综合应用： 10

## 1：什么是闭包

闭包就是能够读取其他函数内部变量的函数。在JS中，只有函数内部的子函数才能读取局部变量，因此可以把闭包简单理解为”定义在一个函数内部的函数”。

## 2：闭包的特点

1：可以读取函数内部的变量。

2：让这些变量的值始终保存在内存中。

## 3：闭包的原理

理解闭包，首先必须理解JS变量的作用域。变量的作用域无非就是两种：全局变量和局部变量。

JS语言的特殊之处，就在于函数内部可以直接读取全局变量。另一方面，函数外部自然无法读取函数内的局部变量。

**注意：**

1：函数内部声明变量的时候，一定要使用var声明。如果不用的话，你实际上声明了一个全局变量。

2：局部变量的作用域，在函数定义的时候就已经确定下来了。

出于各种原因，我们有时候需要得到函数内部的局部变量。但是正常情况下这是办不到的。只有变通一下才能实现，那就是在函数内部再定义一个函数。外部变量不能访问内部变量，内部变量却能访问外部变量，这正是因为JS特有的”链式作用域”结构(chain scope)，子对象会一级一级地向上寻找所有父对象的变量。所以父对象的所有变量，对子对象都是可见的，反之则不成立。我们只需要把子函数返回出来，我们就可以在外部读取内部变量了。

## 4：闭包的应用场景

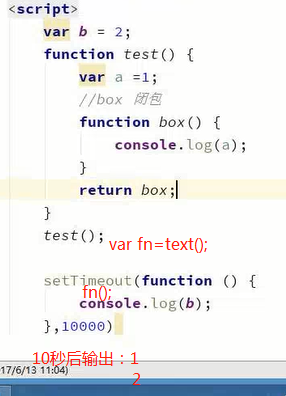
1：函数作为返回值。

2：函数作为参数被传递。

## 5：使用闭包注意点

1：由于闭包会使得函数中的变量都被保存在内存中，内存消耗很大，所以不能滥用闭包。否则会造成网页性能问题，在IE中可能导致内存泄漏。解决方法就是在函数退出之前，将不使用的局部变量删除。

2：闭包会在父函数外部，改变父函数内部变量的值。所以不要随便改变父函数内部变量的值。

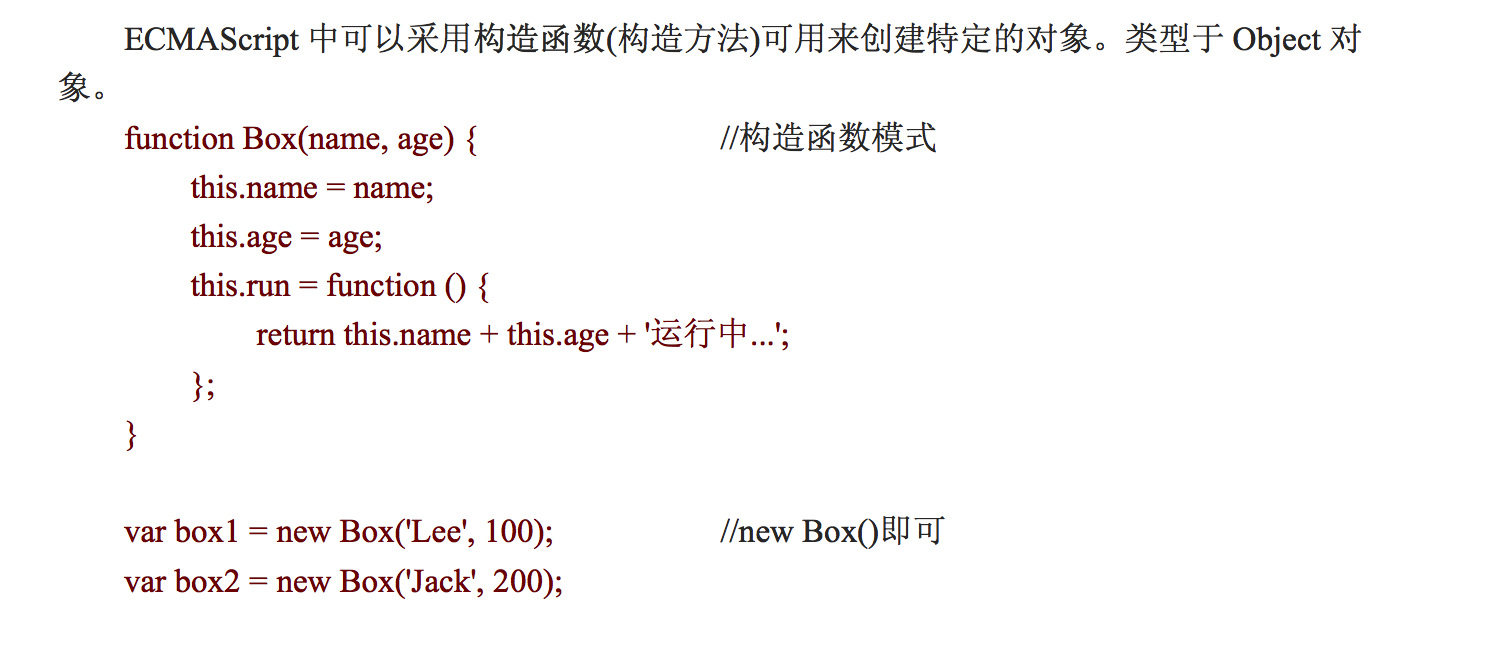


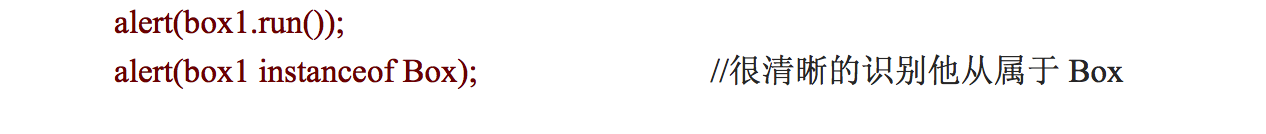
## 应用：

1：掌握闭包的特点和原理。

2：掌握闭包的应用场景。

## 6：构造函数继承（；apply和call继承；原型继承；混合继承:利用构造函数继承属性，用原型链继承方法）







## 7：call/apply继承

call和apply都是为了改变某个函数运行时的context即上下文而存在的，换句话说，就是为了改变函数内部this的指向。

二者作用完全一样，只是接受参数的方式不太一样。

Fn.call(obj, arg1, arg2 [, argN]);

fn,.apply(obj, [arg1, arg2,…, argN]);

apply以数组的形式传参，call是以列表的形式。

call方法是Function类中的方法

call方法的第一个参数的值赋值给类(即方法)中出现的this

call方法的第二个参数开始依次赋值给类(即方法)所接受的参数

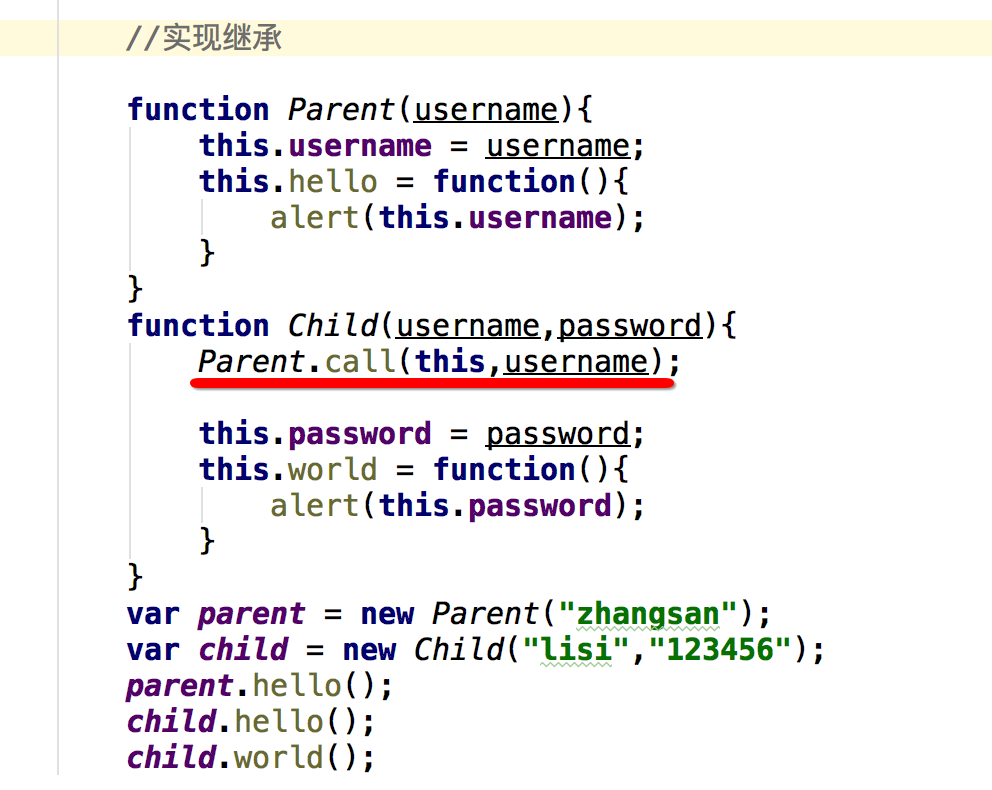
apply方法接受2个参数，

A、第一个参数与call方法的第一个参数一样，即赋值给类(即方法)中出现的this

B、第二个参数为数组类型，这个数组中的每个元素依次赋值给类(即方法)所接受的参数

实现继承：

将父对象的构造函数绑定在子对象上，即在子对象构造函数中加一行：





## 8：prototype的概念

我们创建的每个函数都有一个 prototype(原型)属性，这个属性是一个对象，它的用途是

包含可以由特定类型的所有实例共享的属性和方法。逻辑上可以这么理解: prototype 通过 调用构造函数而创建的那个对象的原型对象 。

function Box() {} //声明一个构造函数

Box.prototype.name = 'Lee'; //在原型里添加属性

Box.prototype.age = 100;

Box.prototype.run = function () { //在原型里添加方法

return this.name + this.age + '运行中...';

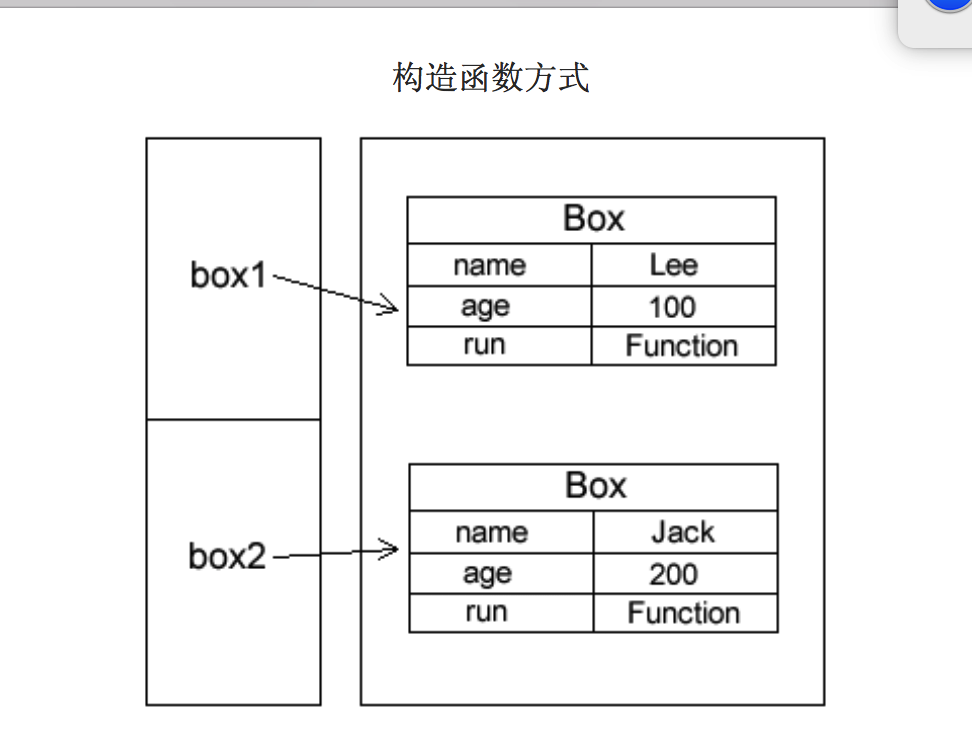
};

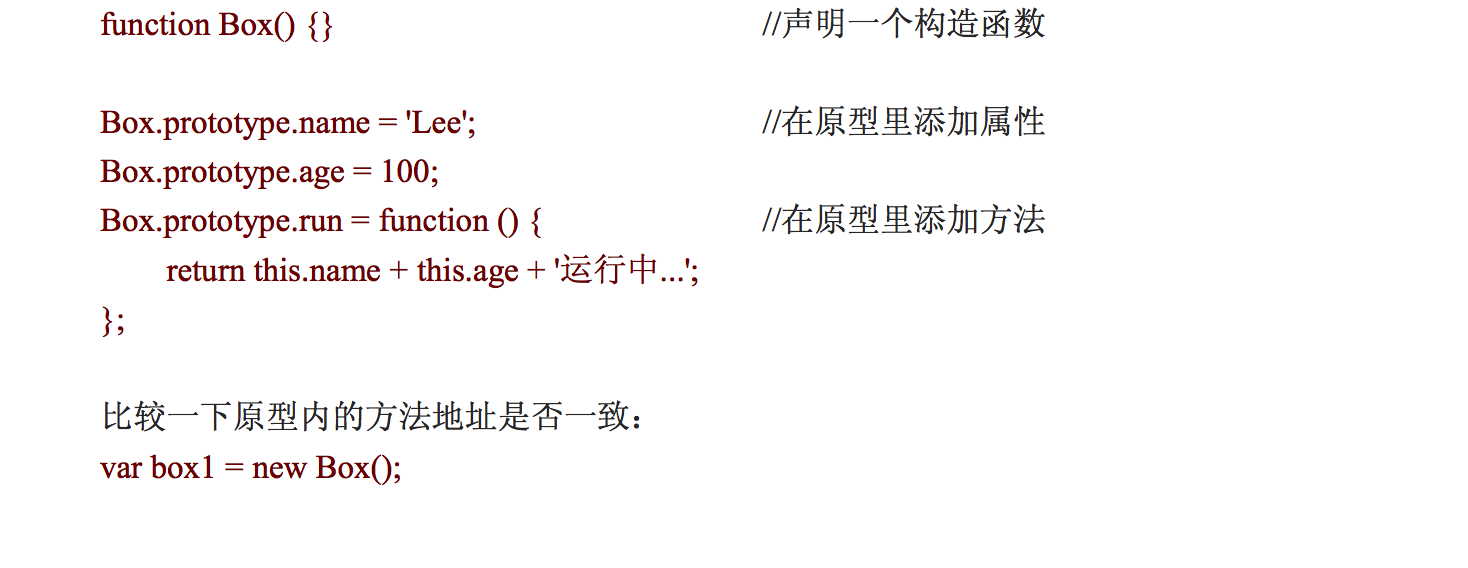
Var box1 = new Box();

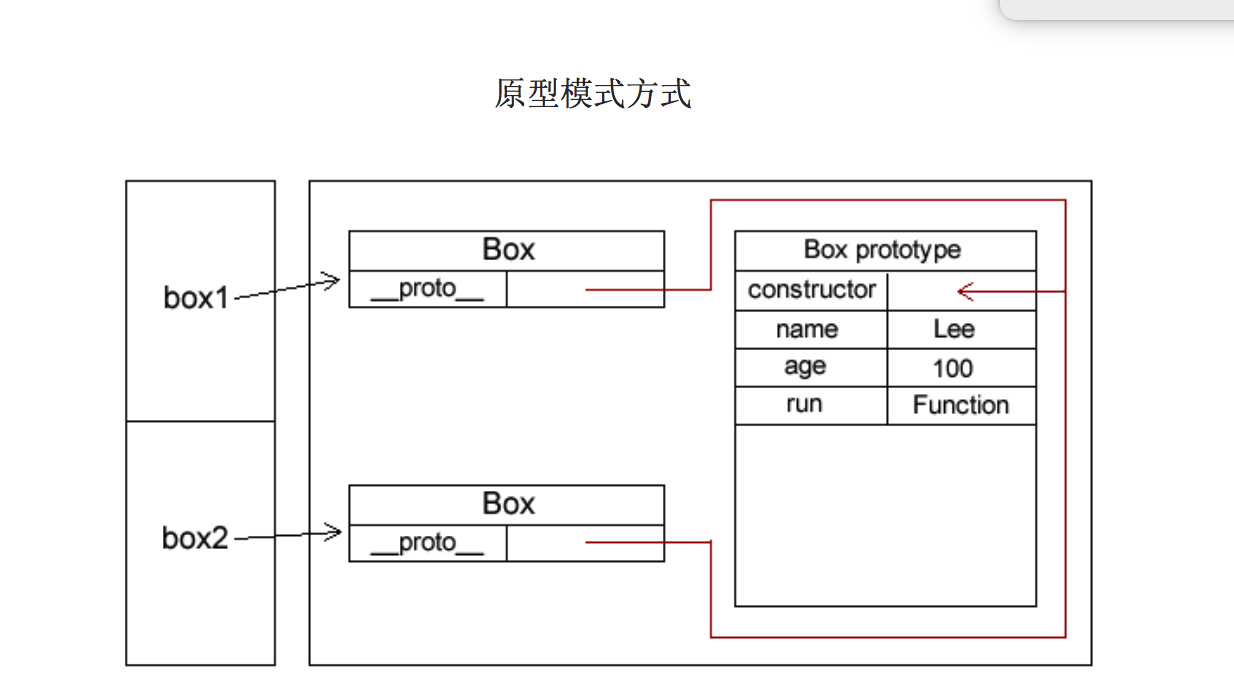
Var box2 = new Box();

Alert(box1.name);

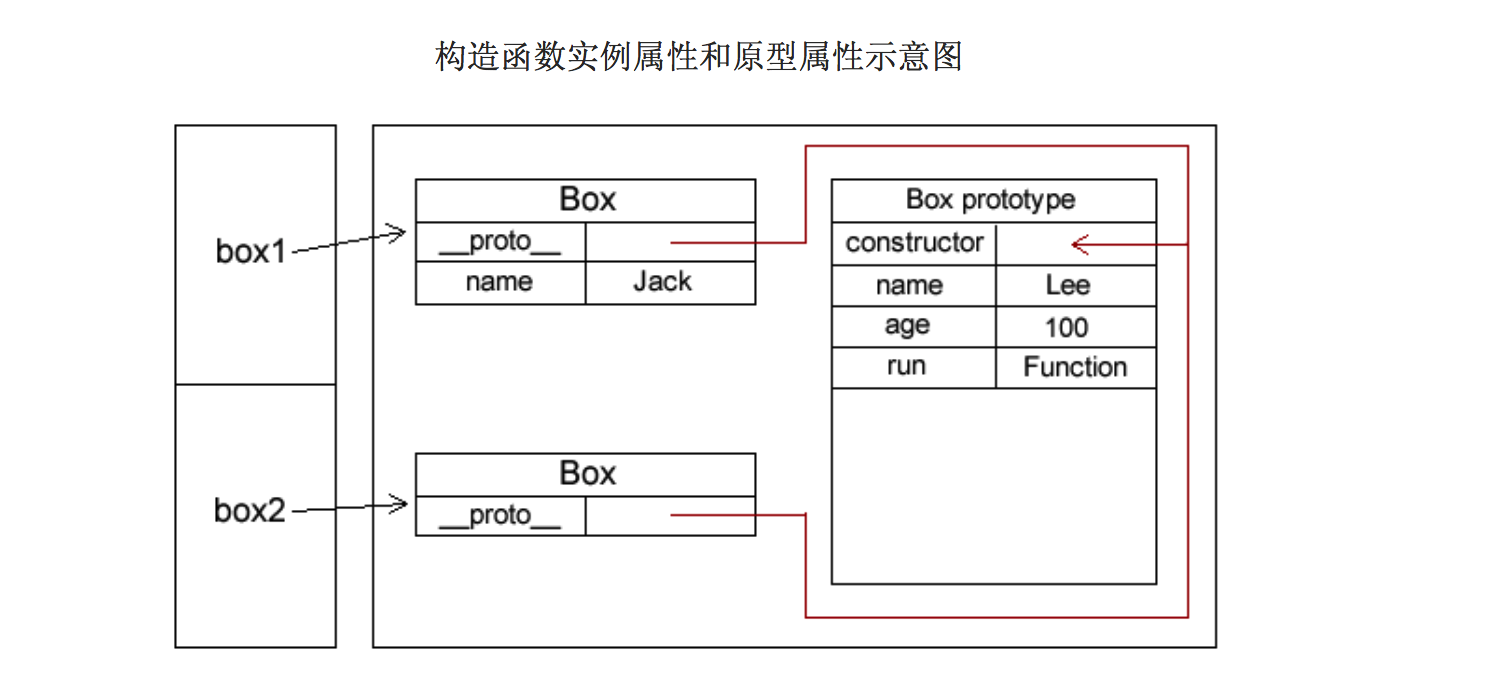
Alert(box2.name);//两个对象的name属性是一样的







在原型模式声明中，多了两个属性，这两个属性都是创建对象时自动生成的 。\_\_proto\_\_ 属性是实例指向原型对象的一个指针，它的作用就是指向构造函数的原型属性， constructor-》构造函数 。 通过这两个属性，就可以访问到原型里的属性和方法了。



如何判断属性是在构造函数的实例里，还是在原型里? 可以使用 hasOwnProperty()函数 来验证:

alert(box.hasOwnProperty('name')); //实例里有返回 true，否则返回 false

in 操作符会在通过对象能够访问给定属性时返回 true，无论该属性存在于实例中还是原 型中。

alert('name' in box); //true，存在实例中或原型中

原型链：

原型模式的执行流程: 1.先查找构造函数实例里的属性或方法，如果有，立刻返回; 2.如果构造函数实例里没有，则去它的原型对象里找，如果有，就返回;

JS在创建对象的时候，都有一个叫做\_\_proto\_\_的内置属性，用于指向创建它的函数对象的原型对象prototype。  
同样，box1.prototype对象也有\_\_proto\_\_属性，它指向创建它的函数对象（Object）的prototype。Object.prototype对象也有\_\_proto\_\_属性，但它比较特殊，为null。我们把这个由\_\_proto\_\_串起来的直到Object.prototype.\_\_proto\_\_为null的链叫做原型链

## 9：原型链的继承

将一个构造函数的原型指向另一个构造函数的实例对象来实现继承。

function Box() { //Box构造

this.name = 'Lee';

}

function Desk() { //Desk构造

this.age = 100;

}

Desk.prototype = new Box(); //Desc继承了Box，通过原型，形成链条

var desk = new Desk();

alert(desk.age);

alert(desk.name); //得到被继承的属性

function Table() { //Table构造

this.level = 'AAAAA';

}

Table.prototype = new Desk(); //继续原型链继承

var table = new Table();

alert(table.name); //继承了Box和Desk

## 10：混合继承

组合继承就是结合使用原型链与构造函数的优点，组合而成的一个模式。实现也很简单，既然是结合，那当然结合了两方的优点，

**即原型链继承方法，而在构造函数继承属性**。

function Box(age) {

this.name = ['Lee', 'Jack', 'Hello']

this.age = age;

}

Box.prototype.run = function () {

return this.name + this.age;

};

function Desk(age) {

Box.call(this, age); //对象冒充 继承了 name 和age 属性

}

Desk.prototype = new Box(); //原型链继承 run方法

var desk = new Desk(100);

alert(desk.run());

## 应用：

1：掌握继承的原理。

## 综合应用：

1：分析微信飞机大战游戏的继承关系。