**对象冒充**

构想原始的 ECMAScript 时，根本没打算设计对象冒充（object masquerading）。它是在开发者开始理解函数的工作方式，尤其是如何在函数环境中使用 this 关键字后才发展出来。

其原理如下：构造函数使用 this 关键字给所有属性和方法赋值（即采用类声明的构造函数方式）。因为构造函数只是一个函数，所以可使 ClassA 构造函数成为 ClassB 的方法，然后调用它。ClassB 就会收到 ClassA 的构造函数中定义的属性和方法。例如，用下面的方式定义 ClassA 和 ClassB：

function ClassA(sColor) {

this.color = sColor;

this.sayColor = function () {

alert(this.color);

};

}

function ClassB(sColor) {

}

还记得吗？关键字 this 引用的是构造函数当前创建的对象。不过在这个方法中，this 指向的所属的对象。这个原理是把 ClassA 作为常规函数来建立继承机制，而不是作为构造函数。如下使用构造函数 ClassB 可以实现继承机制：

function ClassB(sColor) {

this.newMethod = ClassA;

this.newMethod(sColor);

delete this.newMethod;

}

在这段代码中，为 ClassA 赋予了方法 newMethod（请记住，函数名只是指向它的指针）。然后调用该方法，传递给它的是 ClassB 构造函数的参数 sColor。最后一行代码删除了对 ClassA 的引用，这样以后就不能再调用它。

所有新属性和新方法都必须在删除了新方法的代码行后定义。否则，可能会覆盖超类的相关属性和方法：

function ClassB(sColor, sName) {

this.newMethod = ClassA;

this.newMethod(sColor);

delete this.newMethod;

this.name = sName;

this.sayName = function () {

alert(this.name);

};

}

为证明前面的代码有效，可以运行下面的例子：

var objA = new ClassA("blue");

var objB = new ClassB("red", "John");

objA.sayColor(); //输出 "blue"

objB.sayColor(); //输出 "red"

objB.sayName(); //输出 "John"

对象冒充可以实现多重继承

有趣的是，对象冒充可以支持多重继承。也就是说，一个类可以继承多个超类。

继承机制 UML 图示实例

例如，如果存在两个类 ClassX 和 ClassY，ClassZ 想继承这两个类，可以使用下面的代码：

function ClassZ() {

this.newMethod = ClassX;

this.newMethod();

delete this.newMethod;

this.newMethod = ClassY;

this.newMethod();

delete this.newMethod;

}

这里存在一个弊端，如果存在两个类 ClassX 和 ClassY 具有同名的属性或方法，ClassY 具有高优先级。因为它从后面的类继承。除这点小问题之外，用对象冒充实现多重继承机制轻而易举。

由于这种继承方法的流行，ECMAScript 的第三版为 Function 对象加入了两个方法，即 call() 和 apply()。