# **JavaScript中对象的构造方法**

2009-04-20 23:20

第一种方式： 工厂方法

能创建并返回特定类型的对象的工厂函数（factory function）。

Javascript代码

function createCar(sColor){

  var oTempCar = new Object;

  oTempCar.color = sColor;

  oTempCar.showColor = function (){

    alert(this.color);

  };

  return oTempCar;

}

var oCar1 = createCar();

var oCar2 = createCar();

调用此函数时，将创建新对象，并赋予它所有必要的属性。使用此方法将创建car对象的两个版本（oCar1和oCar2），他们的属性完全一样。

使用此方法存在的问题：

1语义上看起来不像使用带有构造函数的new运算符那么正规。

2使用这种方式必须创建对象的方法。每次调用createCar()，都要创建showColor()，意味着每个对象都有自己的showColor()版本，事实上，每个对象都共享了同一个函数。

有些开发者在工厂函数外定义对象的方法，然后通过属性指向该方法。从而避免这个问题：

Js代码

function createCar(sColor){

  var oTempCar = new Object;

  oTempCar.color = sColor;

  oTempCar.showColor = showColor;

  return oTempCar;

}

function showColor(){

   alert(this.color);

}

在这段重写的代码中，在函数createCar()前定义了函数showColor().在createCar()内部，赋予对象一个已经指向已经存在的showColor()函数的指针。从功能上来讲，这样解决了重复创建对象的问题，但该函数看起来不像对象的方法。

所有这些问题引发了开发者定义的构造函数的出现。

第二种方式：构造函数方式

Js代码

function Car(sColor){

  this.color = sColor;

  this.showColor = function (){

    alert(this.color);

  };

}

var oCar1 = new Car("red");

var oCar2 = new Car("blue");

你可能已经注意到第一个差别了，在构造函数内部无创建对象，而是使用this关键字。使用new运算符调用构造函数时，在执行第一行代码前先创建一个对象，只有用this才能访问该对象。然后可以直接赋予this属性，默认情况下是构造函数的返回值（不必明确使用return运算符）。

这种方式在管理函数方面与工厂方法一样都存在相同的问题。

第三种方式：原型方式

Js代码

function Car(){

}

Car.prototype.color = "blue";

var oCar1 = new Car();

var oCar2 = new Car();

调用new Car()时，原型的所有属性都被立即赋予要创建的对象，意味着所有的Car实例存放的都是指向showColor()函数的指针。从语义上看起来都属于一个对象，因此解决了前面两种方式存在的两个问题。此外使用该方法，还能用instanceof运算符检查给定变量指向的对象类型。因此，下面的代码将输出true：

alert(oCar instanceof Car); //outputs "true"

这个方式看起来很不错，遗憾的是，它并不尽如人意。

1首先这个构造函数没有参数。使用原型方式时，不能给构造函数传递参数初始化属性的值，因为car1和car2的属性都等于&ldquo;red&rdquo;。

2真正的问题出现在属性指向的是对象，而不是函数时。函数共享不会造成任何问题，但是对象却很少被多个实例共享的。

第四种方式：混合的构造函数/原型方式（推荐）

联合使用构造函数和原型方式，就可像用其他程序设计语言一样创建对象。这种概念非常简单，即用构造函数定义对象的所有非函数属性，用原型方式定义对象的函数属性（方法）。

Js代码

function Car(sColor){

  this.color =sColor;

  this.drivers =new Array("Mike","Sue");

}

Car.prototype.showColor = function(){

  alert(this.color);

}

var oCar1 =new Car("red");

var oCar2 =new Car("blue");

oCar1.drivers.push("Matt");

alert(oCar1.drivers);   //outputs "Mike,Sue,Matt"

alert(oCar1.drivers);   //outputs "Mike,Sue"

第五种方式：动态原型方式（推荐）

对于习惯使用其他语言的开发者来说，使用混合的构造函数/原型方式感觉不那么和谐。批评混合的构造函数/原型方式的人认为，在构造函数内找属性，在外部找方法的做法很不合理。所以他们设计了动态原型方式，以提供更友好的编码风格。

动态原型方法的基本想法与混合的构造函数/原型方式相同，即在构造函数内定义非函数属性，而函数属性则利用原型属性定义。唯一的区别是赋予对象方法的位置。下面是使用动态原型方法重写的Car类：

Js代码

function Car(sColor){

  this.color =sColor;

  this.drivers =new Array("Mike","Sue");

  if(typeof Car.\_initialized == "undefined"){

    Car.prototype.showColor = function(){

      alert(this.color);

    }

  }

  Car.\_initialized = true;

}

[原网页](http://www.cnblogs.com/liweiqing/archive/2009/04/20/1440042.html" \t "http://qbview.url.cn/_blank)已由[QQ浏览器](http://mdc.html5.qq.com/d/directdown.jsp?channel_id=10371" \t "http://qbview.url.cn/_blank)云转码