Javascript面向对象

Javascript面向对象中所有事物都是对象：字符串、数值、数组、函数…

此外，Javascript允许自定义对象。

Javascript对象：

Javascript提供多种内置对象，比如String、Date、Array对象，对象只是带有属性和方法的特殊数据类型。

访问对象的属性：

objectName.propertyName

下面这个例子使用了String对象的Length属性来获得字符串的长度。

var message =”Hello World!”;

var x= message.Length;

以上代码执行之后，x将会是12

访问对象的方法：

方法是能够在Javascript对象上执行的动作。

objectName.methodName()

下面例子使用了String对象的toUpperCase()方法将文本转换为大写：

var message =”hello world!”;

var x=message.toUpperCase();

执行以上代码之后

x= “HELLO WORLD!”;

创建Javascript对象：

创建自定义Javascript对象有两种方式：

1. 定义并创建对象的实例
2. 使用函数来定义对象，然后创建新的对象实例

创建直接的实例：下面这个例子创建对象的新实例，并添加四个成员属性值

<!DOCTYPE html>

<html>

<body>

<script>

person =new Object();

person.firstName=”Jim”;

person.LastName=”Green”;

person.Age=23;

person.eyeColor=”blue”;

document.write(person.firstname+person.Lastname+”is”+person.Age);

</script>

</body>

</html>

替代语法：

person ={firatName:”jim”, lastName:”Green”,Age:23,eyeColor:”blue”};

Javascript面向对象：

Javascript是面向对象的语言，但javascript不使用类。

Javascript是基于prototype，而不是基于类的。

一： 生成对象的原始模式：

假如我们把猫看做一个对象，有名字个颜色两个属性：

var Cat ={

name :’ ’;

color: ‘ ’

}

现在，我们根据这个原型对象的规格生成两个实例。

var cat1 ={}；

cat1.name =”大毛”;

cat1.color =”黄色”;

var cat2 ={}；

cat2.name =”二毛”;

cat2.color= “黑色”;

这是最简单的封装，如果多生成几个实例就会非常麻烦，而且实例和原型之间没有

看不出任何关联。

二．原始模式的改进

function Cat(name,color)

{

return {

name:name,

color:color

}

}

然后生成实例对象，就等于在调用函数。

var cat1 =Cat(“大毛”,”黑色”);

var cat2 =Cat(“二毛”，”黄色”);

cat1 ，cat2没有任何联系，看不出原型。

三． 构造函数模式

javascript提出了一个构造函数模式，其实就是一个普通的javascript函数，但是内部使用了this变量。对这个“普通”函数使用new运算符，就能生成实例，并且this会绑定到实例对象上。

猫的原型可以这样

function Cat (name,color)

{

this.name=name;

this.color=color;  
}

我们现在就可以生成实例对象了。

var cat1 =new Cat（”大毛”，”黄色”）；

var cat2 =new Cat（”二毛”, ”黑色”）；

这时候 cat1 ，cat2会自动含有一个constructor属性，指向他们的构造函数（也就是这个普通函数）

alert(cat1.constructor ==Cat) // true

alert(cat2.constructor ==Cat) // true

四．构造函数模式的问题

构造函数很好用，但是存在一个浪费内存的问题。

如果我为猫对象添加

this.type =”猫科”;

this.eat =function() { alert(‘吃老鼠’);};

使用上面的new 构造函数构造几个实例对象，看起来没有什么问题，但是type和 eat函数重复了几次，不环保。

考虑能不能让type和eat函数在内存中只有一份，所有生成实例指向这个地址。

引出prototype 模式。

五．prototype模式

Javascript规定，每个构造函数都有一个prototype属性，指向另一个对象，这个对象所有属性和方法，都会被构造函数的实例继承。这意味着我们可以把那些不变的属性和方法，直接定义到prototype对象上。

function Cat(name,color)

{

this.name =name;

this.color =color;

}

Cat.prototype.type =”猫科”;

Cat.prototype.eat = function() { alert(‘吃老鼠’) ;};

之后所有实例的type属性和eat函数，其实都指向一个地址，指向prototype对象，因此提高了效率。

alert (cat1.eat ==cat2.eat); // true