## JS面向对象基础讲解(工厂模式、构造函数模式、原型模式、混合模式、动态原型模式)

**工厂模式**

　　工厂模式是软件工程领域一种广为人知的设计模式，而由于在ECMAScript中无法创建类，因此用函数封装以特定接口创建对象。其实现方法非常简单，也就是在函数内创建一个对象，给对象赋予属性及方法再将对象返回即可。

function createBlog(name, url) {

var o = new Object();

o.name = name;

o.url = url;

o.sayUrl= function() {

alert(this.url);

}

return o;

}

var blog1 = createBlog('wuyuchang', 'http://www.jb51.net/');

可以看到工厂模式的实现方法非常简单，解决了创建多个相似对象的问题，但是工厂模式却无从识别对象的类型，因为全部都是Object，不像Date、Array等，因此出现了构造函数模式。

**构造函数模式**

　　ECMAScript中构造函数可以创建特定类型的对象，类似于Array、Date等原生JS的对象。其实现方法如下：

function Blog(name, url) {

this.name = name;

this.url = url;

this.alertUrl = function() {

alert(this.url);

}

}

var blog = new Blog('wuyuchang', 'http://www.jb51.net/');

console.log(blog instanceof Blog); // true， 判断blog是否是Blog的实例，即解决了工厂模式中不能

这个例子与工厂模式中除了函数名不同以外，细心的童鞋应该发现许多不同之处：

函数名首写字母为大写　　（虽然标准没有严格规定首写字母为大写，但按照惯例，构造函数的首写字母用大写没有显示的创建对象,直接将属性和方法赋值给了this对象,没有return语句,使用new创建对象,能够识别对象（这正是构造函数模式胜于工厂模式的地方）

　　构造函数虽然好用，但也并非没有缺点，使用构造函数的最大的问题在于每次创建实例的时候都要重新创建一次方法（理论上每次创建对象的时候对象的 属性均不同，而对象的方法是相同的），然而创建两次完全相同的方法是没有必要的，因此，我们可以将函数移到对象外面（也许有些童鞋已经看出缺点，嘘！）。

function Blog(name, url) {

this.name = name;

this.url = url;

this.alertUrl = alertUrl;

}

function alertUrl() {

alert(this.url);

}

var blog = new Blog('scjb51', 'http://sc.jb51.net/'),

blog2 = new Blog('jb51', 'http://www.jb51.net/');

blog.alertUrl(); // http://sc.jb51.net/

blog2.alertUrl(); // http://www.jb51.net/

我们将alertUrl设置成全局函数，这样一来blog与blog2访问的都是同一个函数，可是问题又来了，在全局作用域中定义了一个实际只想让 Blog使用的函数，显示让全局作用域有些名副其实，更让人无法接受的是在全局作用域中定义了许多仅供特定对象使用的方法，浪费空间不说，显然失去了面向 对象封装性了，因此可以通过原型来解决此问题。

**原型模式**

　　我们创建的每个函数都有prototype（原型）属性，这个属性是一个指针，指向一个对象，而这个对象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。使用原型对象的好处就是可以让所有对象实例共享它所包含的属性及方法。

function Blog() {

}

Blog.prototype.name = 'wuyuchang';

Blog.prototype.url = 'http://tools.jb51.net/';

Blog.prototype.friend = ['fr1', 'fr2', 'fr3', 'fr4'];

Blog.prototype.alertInfo = function() {

alert(this.name + this.url + this.friend );

}

// 以下为测试代码

var blog = new Blog(),

blog2 = new Blog();

blog.alertInfo(); // wuyuchanghttp://tools.jb51.net/fr1,fr2,fr3,fr4

blog2.alertInfo(); // wuyuchanghttp://tools.jb51.net/fr1,fr2,fr3,fr4

blog.name = 'wyc1';

blog.url = 'http://\*\*\*.com';

blog.friend.pop();

blog2.name = 'wyc2';

blog2.url = 'http://+++.com';

blog.alertInfo(); // wyc1http://\*\*\*.comfr1,fr2,fr3

blog2.alertInfo(); // wyc2http://+++.comfr1,fr2,fr3

原型模式也不是没有缺点，首先，它省略了构造函数传递初始化参数这一环节，结果所有实例在默认情况下都取得了相同的属性值，这样非常不方便，但这还 是不是原型的最大问题，原型模式的最大问题在于共享的本性所导致的，由于共享，因此因此一个实例修改了引用，另一个也随之更改了引用。因此我们通常不单独 使用原型，而是结合原型模式与构造函数模式。

**混合模式（原型模式 + 构造函数模式）**

function Blog(name, url, friend) {

this.name = name;

this.url = url;

this.friend = friend;

}

Blog.prototype.alertInfo = function() {

alert(this.name + this.url + this.friend);

}

var blog = new Blog('wuyuchang', 'http://tools.jb51.net/', ['fr1', 'fr2', 'fr3']),

blog2 = new Blog('wyc', 'http://\*\*.com', ['a', 'b']);

blog.friend.pop();

blog.alertInfo(); // wuyuchanghttp://tools.jb51.net/fr1,fr2

blog2.alertInfo(); // wychttp://\*\*.coma,b

混合模式中构造函数模式用于定义实例属性，而原型模式用于定义方法和共享属性。每个实例都会有自己的一份实例属性，但同时又共享着方法，最大限度的 节省了内存。另外这种模式还支持传递初始参数。优点甚多。这种模式在ECMAScript中是使用最广泛、认同度最高的一种创建自定义对象的方法。

**动态原型模式**

　　动态原型模式将所有信息封装在了构造函数中，而通过构造函数中初始化原型（仅第一个对象实例化时初始化原型），这个可以通过判断该方法是否有效而选择是否需要初始化原型。

function Blog(name, url) {

this.name = name;

this.url = url;

if (typeof this.alertInfo != 'function') {

// 这段代码只执行了一次

alert('exe time');

Blog.prototype.alertInfo = function() {

alert(thia.name + this.url);

}

}

}

var blog = new Blog('wuyuchang', 'http://tools.jb51.net'),

blog2 = new Blog('wyc', 'http:\*\*\*.com');

可以看到上面的例子中只弹出一次窗，'exe time'，即当blog初始化时，这样做blog2就不在需要初始化原型，对于使用这种模式创建对象，可以算是perfect了。