

分类：[IT技术](#)

上一篇：[善于说，善于做](#)

下一篇：[Javascript面向对象编程（三](#)

## Javascript面向对象编程（二）：继承

作者：阮一峰

日期：2010年5月23日

上一次的文章，主要介绍了如何["封装"](#)数据和方法，从原型对象生成实例。

今天要介绍的是，多个原型对象之间如何["继承"](#)。

比如，现在有一个"动物"对象，

```
function Animal(){  
  
    this.species = "动物";  
  
}
```

还有一个"猫"对象，

```
function Cat(name,color){  
  
    this.name = name;  
  
    this.color = color;  
  
}
```

怎样才能使"猫"继承"动物"呢？

### 1. 原型对象绑定

最简单的方法，大概就是使用call或apply方法，将父对象绑定在子对象上，也就是在子

对象函数中加一行：

```
function Cat(name,color){  
  
    Animal.apply(this, arguments);  
  
    this.name = name;  
  
    this.color = color;  
  
}  
  
var cat1 = new Cat("大毛","黄色");  
  
alert(cat1.species); // 动物
```

## 2. prototype模式

更常见的做法，则是使用prototype属性。

如果"猫"的prototype对象，指向一个Animal的实例，那么所有"猫"的实例，就能继承Animal了。

```
Cat.prototype = new Animal();  
  
Cat.prototype.constructor = Cat;  
  
var cat1 = new Cat("大毛","黄色");  
  
alert(cat1.species); // 动物
```

代码的第一行，我们将Cat的prototype对象指向一个Animal的实例。

```
Cat.prototype = new Animal();
```

它相当于完全删除了prototype 对象原先的值，然后赋予一个新值。但是，第二行又是什么意思呢？

```
Cat.prototype.constructor = Cat;
```

原来，任何一个prototype对象都有一个constructor属性，指向它的构造函数。也就是说，Cat.prototype 这个对象的constructor属性，是指向Cat的。

我们在前一步已经删除了这个prototype对象原来的值，所以新的prototype对象没有constructor属性，所以我们必须手动加上，否则后面的"继承链"会出问题。这就是第二行的意思。

总之，这是很重要的一点，编程中务必要遵守。下文都遵循这一点，即如果替换了prototype对象，

```
o.prototype = {};
```

那么，下一步必然是为新的prototype对象加上constructor属性，并将这个属性指回原来的构造函数。

```
o.prototype.constructor = o;
```

### 3. 直接继承prototype

由于Animal对象中，不变的属性都可以直接写入Animal.prototype。所以，我们也可以让Cat()跳过Animal()，直接继承Animal.prototype。

现在，我们先将Animal对象改写：

```
function Animal(){ }  
  
Animal.prototype.species = "动物";
```

然后，将Cat的prototype对象，然后指向Animal的prototype对象，这样就完成了继承。

```
Cat.prototype = Animal.prototype;  
  
Cat.prototype.constructor = Cat;  
  
var cat1 = new Cat("大毛","黄色");  
  
alert(cat1.species); // 动物
```

与前一种方法相比，这样做的优点是效率比较高（不用执行和建立Animal的实例了），比较省内存。缺点是 Cat.prototype和Animal.prototype现在指向了同一个对象，那么任何对Cat.prototype的修改，都会反映到Animal.prototype。

所以，上面这一段代码其实是有问题的。请看第二行

```
Cat.prototype.constructor = Cat;
```

这一句实际上把Animal.prototype对象的constructor属性也改掉了！

```
alert(Animal.prototype.constructor); // Cat
```

#### 4. 利用空对象作为中介

由于"直接继承prototype"存在上述的缺点，所以可以利用一个空对象作为中介。

```
var F = function(){};  
  
F.prototype = Animal.prototype;  
  
Cat.prototype = new F();  
  
Cat.prototype.constructor = Cat;
```

F是空对象，所以几乎不占内存。这时，修改Cat的prototype对象，就不会影响到

Animal的prototype对象。

```
alert(Animal.prototype.constructor); // Animal
```

## 5. prototype模式的封装函数

我们将上面的方法，封装成一个函数，便于使用。

```
function extend(Child, Parent) {  
  
    var F = function(){};  
  
    F.prototype = Parent.prototype;  
  
    Child.prototype = new F();  
  
    Child.prototype.constructor = Child;  
  
    Child.uber = Parent.prototype;  
  
}
```



使用的时候，方法如下

```
extend(Cat,Animal);  
  
var cat1 = new Cat("大毛","黄色");  
  
alert(cat1.species); // 动物
```

这个extend函数，就是YUI库如何实现继承的方法。

另外，说明一点。函数体最后一行

```
Child.uber = Parent.prototype;
```

意思是子对象设一个uber属性，这个属性直接指向父对象的prototype属性。这等于是在子对象上打开一条通道，可以直接调用父对象的方法。这一行放在这里，只是为了实现继承的完备性，纯属备用性质。

## 6. 拷贝继承

上面是采用prototype对象，实现继承。我们也可以换一种思路，纯粹采用"拷贝"方法实现继承。简单说，如果把父对象的所有属性和方法，拷贝进子对象，不也能够实现继承吗？

首先，还是把Animal的所有不变属性，都放到它的prototype对象上。

```
function Animal(){  
  
  Animal.prototype.species = "动物";
```

然后，再写一个函数，实现属性拷贝的目的。

```
function extend2(Child, Parent) {  
  
  var p = Parent.prototype;  
  
  var c = Child.prototype;  
  
  for (var i in p) {  
  
    c[i] = p[i];
```

```
    }  
  
    c.uber = p;  
  
}
```

这个函数的作用，就是将父对象的prototype对象中的属性，一一拷贝给Child对象的prototype对象。

使用的时候，这样写：

```
extend2(Cat, Animal);  
  
var cat1 = new Cat("大毛","黄色");  
  
alert(cat1.species); // 动物
```

(完)

## 文档信息

- 版权声明：自由转载-非商用-非衍生-保持署名 | Creative Commons BY-NC-

ND 3.0

- 原文网址：[http://www.ruanyifeng.com/blog/2010/05/object-oriented\\_javascript\\_inheritance.html](http://www.ruanyifeng.com/blog/2010/05/object-oriented_javascript_inheritance.html)
- 最后修改时间：2010年5月23日 21:09



## 相关文章

- **2010.12.27:** [PHP最佳实践](#)


今天下午，我在读下面这篇文章。

- **2010.12.07:** [《黑客英雄》书摘](#)

《黑客与画家》翻译完成后，我还欠出版社一篇译者序。

# 功能链接

---

- 前一篇：[善于说，善于做](#)
- 后一篇：[Javascript面向对象编程（三）：非函数对象的继承](#)
- 更多内容请访问：[首页](#) » [档案](#) » [IT技术](#)
- 站内搜索：
- **Feed**订阅：

# 广告

---

# 留言（14条）

---

朴素少年 说：

关注很久了，第一次沙发，顺便带上自己的链接

2010年5月23日 22:04 | [档案](#) | [引用](#)

---

**RedNax** 说：

我还是不知道为什么要做

`o.prototype.constructor = o;`

是为了维护正确的继承回朔链？来保证形如

`this.constructor.prototype.constructor.prototype....`这类回朔的正确性吗？

那么是不是说如果程序本来就不打算回朔的话其实也就没必要加这个了？

2010年5月23日 22:51 | [档案](#) | [引用](#)

---

**Ruan YiFeng** 说：

引用 **RedNax** 的发言：

```
o.prototype.constructor = o;
```

是为了维护正确的继承回溯链？

基本上就是这个目的，还有就是instanceof运算符能返回正确的结果。

2010年5月24日 00:13 | [档案](#) | [引用](#)

---

**zhaorui** 说：

看这个系列，我应该可以温习一下javascript

在原型对象绑定中

```
Animal.apply(this, arguments);
```

这一句里面的arguments 好像应该是 Animal？

2010年5月24日 00:15 | [档案](#) | [引用](#)

---

**zhaorui** 说：

另外，我觉得第二种 prototype 的方式看上去比较简洁有效，后面的几种有其他的优点么？

**Micky** 说：

很棒~学习了~

深入浅出，容易理解~

个人倾向于“5. prototype模式的封装函数”这个方法~觉得还蛮优雅的~

2010年5月24日 09:32 | 档案 | 引用

---

**RedNax** 说：

受教了。

第5种方法确实不错，构造函数应该是（根据参数）为实例添加特定成员而存在的。  
new出来作为prototype的话，父类的构造函数就已经跑过一次了，结果子类构造的时候如果必要还要在跑一次，就显得浪费了。

不过debug和用代码回溯的时候可能会比较麻烦，两个prototype中间会隔着一个空类。

2010年5月24日 21:27 | 档案 | 引用

---

**RedNax** 说：

啊，抱歉想错了。



第5种方法不会增加空类的，那个new F()和new Parent()地位一样，只是不跑构造函数而已。  
好方法.....

2010年5月24日 21:32 | 档案 | 引用

---

**dong2590** 说：

```
function extend(Child, Parent) {  
  
    var F = function(){};  
    //这里应该是new Parent()吧  
    F.prototype = Parent.prototype;  
  
    Child.prototype = new F();  
  
    Child.prototype.constructor = Child;  
  
    Child.uber = Parent.prototype;  
  
}
```

2010年5月25日 15:15 | 档案 | 引用

---

**wiky** 说：

不错，支持！

2010年5月29日 11:39 | 档案 | 引用

**55555555** 说：

```
function extend(Child, Parent) {  
  
    var F = function(){};  
    F.prototype = Parent.prototype;  
    Child.prototype = new F();  
    Child.prototype.constructor = Child;  
    Child.uber = Parent.prototype;  
}
```

这个好像parent必须实现prototype方法，

2010年12月27日 14:10 | 档案 | 引用

**1w** 说：

引用 **RedNax** 的发言：

啊，抱歉想错了。

第5种方法不会增加空类的，那个new F()和new Parent()地位一样，只是不跑构造函数而已。

好方法.....

new F() 这里不还是 new 出一个实例么，按作者的意思，这里因为是 new 一个空对象，资源消耗相对较小。

2011年1月12日 11:44 | 档案 | 引用

**1w** 说：

最后一种通过拷贝的方法来实现继承是有问题的。这里博主用的是浅拷贝的方法，c[i]是指向到 p[i]，而非赋值，这样如果改动到 c[i]，会影响到 p[i]。例如：

```
function fn1() {};  
function fn2() {};  
fn2.prototype.arr = [1,2];  
var c = fn1.prototype, p = fn2.prototype;  
for(var i in p) {c[i] = p[i]};  
fn1.prototype.arr.push(3);  
fn2.prototype.arr // [1,2,3]
```

建议用深拷贝的方法来实现:

```
(function(o) {  
  if(typeof o !== "object") return o;  
  var obj = {};  
  for(var p in o) {  
    obj[p] = arguments.callee(o[p]);  
  }  
  return obj;  
})(o);
```

---

2011年1月12日 12:18 | 档案 | 引用

**1w** 说 :

不好意思,上面这样写当遇到数组时是不行的,参考峰兄的代码,修改了一下:

```
var obj = (o.constructor === Array)?[]:();
```

---

2011年1月13日 13:14 | 档案 | 引用

# 我要发表看法

您的留言（HTML标签部分可用）

您的大名：

«-必填

电子邮件：

«-必填，不公开

个人网址：

«-我信任你，不会填写广告链接

记住个人信息？☐

发表

«- 点击按钮

---

联系方式 | [ruanyifeng.com](http://ruanyifeng.com) 2003 - 2010

