



分类：[IT技术](#)

上一篇：[HTML5的视频格式之争](#)

下一篇：[不会沉没的海盗湾](#)

Javascript 面向对象编程（一）：封装

作者：阮一峰

日期：2010年5月17日

学习Javascript，最难的地方是什么？

我觉得，Object（对象）最难。因为Javascript的Object模型很独特，和其他语言都不一样，初学者不容易掌握。

下面就是我的学习笔记，希望对大家学习这个部分有所帮助。我主要参考了[Object-Oriented JavaScript](#)和[Professional JavaScript for Web Developers \(2nd Edition\)](#)这两本书。它们都是非常优秀的Javascript读物，推荐阅读。

笔记分成两部分。今天的第一部分是讨论"封装"（Encapsulation），下一次的第二部分讨论"继承"（Inheritance）。

=====

Javascript 面向对象编程（一）：封装

作者：阮一峰

Javascript是一种基于对象（object-based）的语言，你遇到的所有东西几乎都是对象。但是，它又不是一种真正的面向对象编程（OOP）语言，因为它的语法中没有class（类）。

那么，如果我们要把"属性"（property）和"方法"（method），封装成一个对象，甚至要从原型对象生成一个实例对象，我们应该怎么做呢？

1. 生成对象的原始模式

假定我们把猫看成一个对象，它有"名字"和"颜色"两个属性。

```
var Cat = {  
  
  name : "
```

```
    color : "  
  
}
```

现在，我们需要根据这个原型对象，生成两个实例对象。

```
var cat1 = {};  
  
    cat1.name = "大毛";  
  
    cat1.color = "黄色";  
  
var cat2 = {};  
  
    cat2.name = "二毛";  
  
    cat2.color = "黑色";
```

好了，这就是最简单的封装了。但是，这样的写法有两个缺点，一是如果多生成几个实例，写起来就非常麻烦；二是实例与原型之间，没有任何办法，可以看出有什么联系。

2. 原始模式的改进

我们可以写一个函数，解决代码重复的问题。

```
function Cat(name,color){  
  
    return {  
  
        name:name,  
  
        color:color  
  
    }  
  
}
```

然后生成实例对象，就等于是在调用函数：

```
var cat1 = Cat("大毛","黄色");  
  
var cat2 = Cat("二毛","黑色");
```

这种方法的问题依然是，cat1和cat2之间没有内在的联系，不能反映出它们是同一个原型对象的实例。

3. 构造函数模式

为了解决从原型对象生成实例的问题，Javascript提供了一个构造函数（Constructor）模式。

所谓"构造函数"，其实就是一个普通函数，但是内部使用了[this变量](#)。对构造函数使用new运算符，就能生成实例，并且this变量会绑定在实例对象上。

比如，猫的原型对象现在可以这样写，

```
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");  
  
alert(cat1.name); // 大毛  
  
alert(cat1.color); // 黄色
```

这时cat1和cat2会自动含有一个constructor属性，指向它们的构造函数。

```
alert(cat1.constructor == Cat); //true  
  
alert(cat2.constructor == Cat); //true
```

Javascript还提供了一个instanceof运算符，验证原型对象与实例对象之间的关系。

```
alert(cat1 instanceof Cat); //true  
  
alert(cat2 instanceof Cat); //true
```

4. 构造函数模式的问题

构造函数方法很好用，但是存在一个浪费内存的问题。

请看，我们现在为Cat对象添加一个不变的属性"type"（种类），再添加一个方法eat（吃老鼠）。那么，原型对象Cat就变成了下面这样：

```
function Cat(name,color){  
  
    this.name = name;  
  
    this.color = color;  
  
    this.type = "猫科动物";  
  
    this.eat = function(){alert("吃老鼠");};  
  
}
```

还是采用同样的方法，生成实例：

```
var cat1 = new Cat("大毛","黄色");
```

```
var cat2 = new Cat ("二毛","黑色");
```

```
alert(cat1.type); // 猫科动物
```

```
cat1.eat(); // 吃老鼠
```

表面上好像没什么问题，但是实际上这样做，有一个很大的弊端。那就是对于每一个实例对象，`type`属性和`eat()`方法都是一模一样的内容，每一次生成一个实例，都必须为重复的内容，多占用一些内存。这样既不环保，也缺乏效率。

```
alert(cat1.eat == cat2.eat); //false
```

能不能让`type`属性和`eat()`方法在内存中只生成一次，然后所有实例都指向那个内存地址呢？回答是可以的。

5. Prototype模式

Javascript规定，每一个构造函数都有一个`prototype`属性，指向另一个对象。这个对象的所有属性和方法，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，直接定义在`prototype`对象上。


```
function Cat(name,color){  
  
    this.name = name;  
  
    this.color = color;  
  
}  
  
Cat.prototype.type = "猫科动物";  
  
Cat.prototype.eat = function(){alert("吃老鼠")};
```

然后，生成实例。

```
var cat1 = new Cat("大毛","黄色");  
  
var cat2 = new Cat("二毛","黑色");  
  
alert(cat1.type); // 猫科动物  
  
cat1.eat(); // 吃老鼠
```

这时所有实例的type属性和eat()方法，其实都是一个内存地址，指向prototype对象，因此就提高了运行效率。

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

6. Prototype模式的验证方法

6.1 isPrototypeOf()

这个方法用来判断，某个prototype对象和某个实例之间的关系。

```
alert(Cat.prototype.isPrototypeOf(cat1)); //true
```

```
alert(Cat.prototype.isPrototypeOf(cat2)); //true
```

6.2 hasOwnProperty()

每个实例对象都有一个hasOwnProperty()方法，用来判断某一个属性到底是本地属性，还是继承自prototype对象的属性。

```
alert(cat1.hasOwnProperty("name")); // true
```

```
alert(cat1.hasOwnProperty("type")); // false
```

6.3 in 运算符

in 运算符可以用来判断，某个实例是否含有某个属性，不管是不是本地属性。

```
alert("name" in cat1); // true
```

```
alert("type" in cat1); // true
```

in 运算符还可以用来遍历某个对象的所有属性。

```
for(var prop in cat1) { alert("cat1["+prop+"]="+cat1[prop]); }
```

(完)

- 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_encapsulation.html
- 最后修改时间：2010年5月17日 18:30



相关文章


- **2011.01.14:** [在PHP语言中使用JSON](#)

目前，JSON已经成为最流行的数据交换格式之一，各大网站的API几乎都支持它。

- **2010.12.27:** [PHP最佳实践](#)

今天下午，我在读下面这篇文章。

功能链接

- 前一篇：[HTML5的视频格式之争](#)
- 后一篇：[不会沉没的海盗湾](#)
- 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（21条）

shimu 说：

直白易懂！逐步深入～这篇对于我这种初学者很有用啊！

2010年5月17日 18:24 | [档案](#) | [引用](#)

Astro 说：

崇拜中。。阮兄知识面让我情何以堪啊。。。

2010年5月17日 18:36 | [档案](#) | [引用](#)

ciciはLuckyD 说：

不错的note，最近我在看pro Javascript Design Patterns这本书，三、四章的封装、继承这块讲的非常好

2010年5月17日 21:12 | [档案](#) | [引用](#)

fan 说：

《javascript: the good part》中Dauglas不推荐用new的这种方法构造对象，因为如果忘记加上new，“即没有编译时警告，也没有运行时警告”。他推荐的是函数化的方法，不使用prototype。

2010年5月18日 07:46 | 档案 | 引用

某人 说：

好多年不写程序了，不过还是觉得javascript是挺复杂的。

2010年5月18日 08:44 | 档案 | 引用

laoguo 说：

喜欢这篇。

越来越能看出，今后，这种娓娓道来的知识描述形式，将把至今为止的，逻辑严谨机械正确但却难懂的知识描述方式打入历史的垃圾箱里。

人，是有灵性的，是非线性的，是量子性的。

至今为止所谓的“线性严谨二元逻辑性的学术描述方式”，只不过是一种违反人类本质天性的东西，必将在完成其历史使命之后，退出历史舞台。

2010年5月18日 08:59 | 档案 | 引用

Ruan YiFeng 说：

引用 **fan** 的发言：

Dauglas不推荐用new的这种方法构造对象，因为如果忘记加上new，“即没有编译时警告，也没有运行时警告”。他推荐的是函数化的方法，不使用prototype。

虽然这个意见是正确的。但是，Douglas提出的方法，需要自己写一个函数，在函数里再使用prototype，我觉得很不符合直觉。

2010年5月18日 09:24 | 档案 | 引用

resun 说：

很简洁实用的文章，里面有不少重来没有看过的用法,如：alert("name" in cat1);
没想到还能用 in 这样做判断

2010年5月18日 10:18 | 档案 | 引用

牙牙学语 说：

非常实用易懂的文章！

最近看《Object Oriented JavaScript》,看完第六章讲12种对象封装构造的方法，彻底晕菜。。看了您的文章决定继续去读完这本书，的确是本很好的JS参考书啊

2010年5月18日 11:16 | 档案 | 引用

axu 说：

这么一段代码，ruan兄猜猜是啥结果？我觉得这是封装中最容易犯的错误

```
var a=function(){  
  //empty  
}
```

```
a.prototype.var1=[1,2,3];
```

```
var b=new a();
```

```
b.var1.push(4);
```

```
var c=new a();
```

```
alert(c.var1.join(","))
```

2010年5月18日 11:19 | 档案 | 引用

Ruan YiFeng 说：

引用 **axu** 的发言：

这么一段代码，**ruan**兄猜猜是啥结果？我觉得这是封装中最容易犯的错误

所以只能把不变的属性和方法，绑在prototype对象上，而不能把可变属性绑上去。

2010年5月18日 13:31 | 档案 | 引用

清风剑 说：

我始终觉得javascript是一门函数式编程语言，这种OO的封装应该只是一种临时的workaround

2010年5月18日 18:21 | 档案 | 引用

richard 说：

这时所有实例的type属性和eat()方法，其实都是一个内存地址，指向prototype对象，因此就提高了运行效率。

也增大了风险，因为一个地方改变了，其他地方都变了。和他带来的好处相比，风险

更大，不应该推荐。

ruan兄应该是有点“完美主义”的偏执吧

2010年5月19日 10:51 | 档案 | 引用

多志郎 说：

会不会连载，如果连载的话，就跟着你学了！

2010年5月19日 11:51 | 档案 | 引用

ethantsien 说：

引用 **fan** 的发言：

《javascript: the good part》中Douglas不推荐用new的这种方法构造对象，因为如果忘记加上new，“即没有编译时警告，也没有运行时警告”。他推荐的是函数化的方法，不使用prototype。

这个“如果”有点苍白

2010年5月21日 15:51 | 档案 | 引用

netwix 说：

js的水很深很深 特别是用js尝试模仿oo的风格

还是喜欢使用简单的 名称空间+函数

2010年5月21日 17:20 | 档案 | 引用

nil 说：

引用 **ethantsien** 的发言：

这个“如果”有点苍白

苍白与否暂且搁置，为了避免忘记加 new，最好的办法是类名首字母大写。

2010年5月24日 09:50 | 档案 | 引用

lee 说：

不错！是一篇好文章！

textbox 说：

这篇文章绝对要顶，学js多年，像中国唯一一个能把问题讲的这么直白透彻的。

2010年7月 2日 12:59 | 档案 | 引用

bill chen 说：

这篇文章分析得十分仔细清楚，绝对经典啊！！！！

2010年11月11日 10:11 | 档案 | 引用

1w 说：

引用 **axu** 的发言：

这么一段代码，ruan兄猜猜是啥结果？我觉得这是封装中最容易犯的错误

```
var a=function(){  
  //empty  
}
```

```
a.prototype.var1=[1,2,3];
```

```
var b=new a();  
b.var1.push(4);  
  
var c=new a();  
alert(c.var1.join(","))
```

这就涉及到对象的继承机制了. 更直白一点, 涉及到 JS 语言的变量赋值与引用的区分.

2011年1月12日 10:35 | 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

