Javascript继承机制的设计思想

作者： 阮一峰

日期： 2011年6月 5日

我一直很难理解Javascript语言的继承机制。

它没有"子类"和"父类"的概念，也没有"类"（class）和"实例"（instance）的区分，全靠一种很奇特的"原型链"（prototype chain）模式，来实现继承。

我花了很多时间，学习这个部分，还做了很多笔记。但是都属于强行记忆，无法从根本上理解。

直到昨天，我读到法国程序员Vjeux的解释，才恍然大悟，完全明白了Javascript为什么这样设计。

下面，我尝试用自己的语言，来解释它的设计思想。彻底说明白prototype对象到底是怎么回事。其实根本就没那么复杂，真相非常简单。

一、从古代说起

要理解Javascript的设计思想，必须从它的诞生说起。

1994年，网景公司（Netscape）发布了Navigator浏览器0.9版。这是历史上第一个比较成熟的网络浏览器，轰动一时。但是，这个版本的浏览器只能用来浏览，不具备与访问者互动的能力。比如，如果网页上有一栏"用户名"要求填写，浏览器就无法判断访问者是否真的填写了，只有让服务器端判断。如果没有填写，服务器端就返回错误，要求用户重新填写，这太浪费时间和服务器资源了。

因此，网景公司急需一种网页脚本语言，使得浏览器可以与网页互动。工程师Brendan Eich负责开发这种新语言。他觉得，没必要设计得很复杂，这种语言只要能够完成一些简单操作就够了，比如判断用户有没有填写表单。

1994年正是面向对象编程（object-oriented programming）最兴盛的时期，C++是当时最流行的语言，而Java语言的1.0版即将于第二年推出，Sun公司正在大肆造势。

Brendan Eich无疑受到了影响，Javascript里面所有的数据类型都是对象（object），这一点与Java非常相似。但是，他随即就遇到了一个难题，到底要不要设计"继承"机制呢？

二、Brendan Eich的选择

如果真的是一种简易的脚本语言，其实不需要有"继承"机制。但是，Javascript里面都是对象，必须有一种机制，将所有对象联系起来。所以，Brendan Eich最后还是设计了"继承"。

但是，他不打算引入"类"（class）的概念，因为一旦有了"类"，Javascript就是一种完整的面向对象编程语言了，这好像有点太正式了，而且增加了初学者的入门难度。

他考虑到，C++和Java语言都使用new命令，生成实例。

C++的写法是：

```
ClassName *object = new ClassName(param);
```

Java的写法是：

```
Foo foo = new Foo();
```

因此，他就把new命令引入了Javascript，用来从原型对象生成一个实例对象。但是，Javascript没有"类"，怎么来表示原型对象呢？

这时，他想到C++和Java使用new命令时，都会调用"类"的构造函数（constructor）。他就做了一个简化的设计，在Javascript语言中，new命令后面跟的不是类，而是构造函数。

举例来说，现在有一个叫做DOG的构造函数，表示狗对象的原型。

```
function DOG(name){
    this.name = name;
}
```

对这个构造函数使用new，就会生成一个狗对象的实例。

```
var dogA = new DOG('大毛');
alert(dogA.name); // 大毛
```

注意构造函数中的this关键字，它就代表了新创建的实例对象。

三、new运算符的缺点

用构造函数生成实例对象，有一个缺点，那就是无法共享属性和方法。

比如，在DOG对象的构造函数中，设置一个实例对象的共有属性species。

```
function DOG(name){
    this.name = name;
    this.species = '犬科';
}
```

然后，生成两个实例对象：

```
var dogA = new DOG('大毛');
var dogB = new DOG('二毛');
```

这两个对象的species属性是独立的，修改其中一个，不会影响到另一个。

```
dogA.species = '猫科';
alert(dogB.species); // 显示"犬科"，不受dogA的影响
```

每一个实例对象，都有自己的属性和方法的副本。这不仅无法做到数据共享，也是极大的资源浪费。

四、prototype属性的引入

考虑到这一点，Brendan Eich决定为构造函数设置一个prototype属性。

这个属性包含一个对象（以下简称"prototype对象"），所有实例对象需要共享的属性和方法，都放在这个对象里面；那些不需要共享的属性和方法，就放在构造函数里面。

实例对象一旦创建，将自动引用prototype对象的属性和方法。也就是说，实例对象的属性和方法，分成两种，一种是本地的，另一种是引用的。

还是以DOG构造函数为例，现在用prototype属性进行改写：

```
function DOG(name){
    this.name = name;
}
DOG.prototype = { species : '犬科' };

var dogA = new DOG('大毛');
var dogB = new DOG('二毛');

alert(dogA.species); // 犬科
```

```
alert(dogB.species); // 犬科
```

现在，species属性放在prototype对象里，是两个实例对象共享的。只要修改了prototype对象，就会同时影响到两个实例对象。

```
DOG.prototype.species = '猫科';

alert(dogA.species); // 猫科
alert(dogB.species); // 猫科
```

五、总结

由于所有的实例对象共享同一个prototype对象，那么从外界看起来，prototype对象就好像是实例对象的原型，而实例对象则好像"继承"了prototype对象一样。

这就是Javascript继承机制的设计思想。不知道我说清楚了没有，继承机制的具体应用方法，可以参考我写的系列文章：

* 《Javascript面向对象编程（一）：封装》
* 《Javascript面向对象编程（二）：构造函数的继承》
* 《Javascript面向对象编程（三）：非构造函数的继承》

（完）