文章编号：2096-1472(2019)-12-14-04
DOI:10.19644/j.cnki.issn2096-1472.2019.12.004

JavaScript继承机制研究

周 岚

(江苏联合职业技术学院徐州财经分院，江苏 徐州 221008)

摘 要：JavaScript是面向Web的编程语言，其高端、动态以及面向对象的编程风格，使得它已经从一门简单的脚本语言进化成为一门强大的编程语言。JS的核心是支持面向对象的，同时它也提供了强大灵活的面向对象语言的编程能力。本文针对JavaScript继承机制的实现方式进行了总结归纳，深入介绍了基于原型的继承、构造函数方式继承、组合继承、寄生式继承等继承机制，并分析了各自方式的优缺点，便于读者更深层次的理解JavaScript面向对象编程机制。

关键词：原型链；继承；原型对象；构造函数

中图分类号：TP311 **文献标识码**：A

Research on JavaScript Inheritance Mechanism

ZHOU Lan

(*Xuzhou Finance and Economics Branch,Jiangsu Union Technical Institute,Xuzhou* 221008,*China*)

Abstract: Javascript is a web-oriented programming language.Its high-end,dynamic and object-oriented programming style has evolved from a simple scripting language to a powerful programming language.The core of JS is to support object-oriented,and it also provides powerful and flexible programming ability of object-oriented language.This paper summarizes the implementation of JavaScript inheritance mechanism,introduces the inheritance mechanism based on prototype,constructor,combination and parasitism,and analyzes the advantages and disadvantages of each method,which is convenient for readers to understand the JavaScript object-oriented programming mechanism.

Keywords:prototype chain;inheritance;prototype object;constructor

1 引言(Introduction)

JavaScript是面向Web的编程语言，其高端、动态和面向对象的编程风格，使得JavaScript已经从一门简单的脚本语言进化成为一门强大的编程语言[1]。在面向对象(OOP)的程序设计范型中通常强调类的概念，早期的JavaScript中并没有类的概念，JavaScript采用基于原型的继承风格，虽然使用起来非常灵活、高效，但对于初学者，要正确理解和使用原型对象及其继承机制是非常困难的，本文对比类的继承机制，并通过实例深入的讨论了JavaScript特有的原型链继承、构造函数继承、组合继承、寄生组合继承、class extend等多种继承机制。希望能给初学者答疑解惑。

2 基于类的继承(Class-based inheritance)

继承是面向对象程序中最重要的概念之一。继承允许我们根据一个类来定义另一个类，当创建一个类时，不需要完全重新编写新的数据成员和数据函数，只需要设计一个新的类，继承了已有的类的成员即可。这个已有的类被称为基类(父类)，新的类被称为派生类(子类)。实现继承的好处：(1)提高代码重用性高。如果我们新创建的类与已有的类有绝大部分相类似，则没有必要再重新定义这个完整的类。这样做可以实现代码的重用，大大减少了软件开发的成本。(2)继承可以实现面向对象的“多态”特性。程序员可以将子类的对象直接赋值给父类的引用，无须再编写显式的类型[2]。

```
// 基类
public class Person
    {
        string _name;
        public string Name
        {
            get {return _name;}
            set {_name=value;}
        }
        public void Show()
        {
    Console.WriteLine("我是人,早上好！");
```

www.rjszz.com

```
        }
    }
    //派生类
    public class Student:Person
    {
        public void  SayHello()
        {
          base.Show();//调用父类Show方法
          Console.WriteLine("我是学生,早上好！");
          }
    }
```

上例中，class Student从class Person继承而来，子类Student的对象就继承了父类Person的所有非私有化成员。

3 基于原型的继承(Prototype-based inheritance)

3.1 原型

在JavaScript中，只要创建一个新函数，就会根据一组特定规则为该函数创建一个prototype属性，而这个属性指向函数的原型对象。在默认情况下，原型对象会自动获得一个constructor属性，而这个属性包含一个指向prototype属性所在函数对象的指针[3]，如图1所示。当一个函数对象被创建时，function构造器产生的函数对象会运行代码“：this.prototype={constructor：this};”。实例没有prototype属性，但是有__proto__属性。函数同时有prototype和__proto__属性。__proto__属性虽然在ECMAScript6语言规范中标准化，但是不推荐被使用。

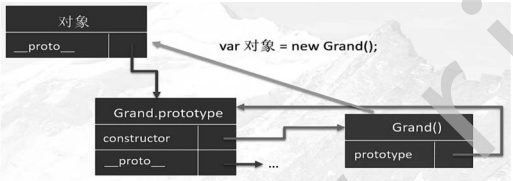

图1 基于原型的继承

Fig.1 Prototype-based inheritance

3.2 原型链

当访问一个对象的属性时，先在对象的本身找，如果找不到就去对象的原型上找，如果还是找不到，就去对象的原型(原型也是对象，也有它自己的原型)的原型上找，如此继续，直到找到为止，或者查找到最顶层的原型对象(原型链的顶端Object类型，Object.prototype.__proto__是原型链的顶端了，指向null)中也没有找到，就结束查找，返回undefined，这条由对象及其原型组成的链就叫作原型链[4]，如图2所示。

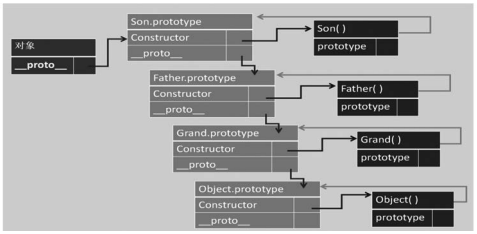

图2 原型链

Fig.2 Prototype chain

继承存在的意义就是属性共享，而原型链存在的意义就是继承：访问对象属性时，在对象本身找不到，就在原型链上一层一层找。说白了就是一个对象可以访问其他对象的属性。如下列所示：

```
<script>
    ////////原型链继承方式
    //汽车
    function automobile(name)
    {
        this.name=name||'automobile';
        this.drive=function() {
            console.log("drive"+this.name);
        }
        this.energy=function() {
            console.log("汽油驱动")
        }
    }
    automobile.prototype.fun=function() {
        console.log('可以载人');
    }
    //公交车
    function bus()
    {
        this.energy=function() {
            console.log("电力驱动")
        }
    }
    bus.prototype=new automobile('bus');
    bus.prototype.constructor=bus;
    var bus1=new bus();
    bus1.drive();
    //如果调用父类的方法
    automobile.call(bus1);
    bus1.energy();
    bus1.fun();
</script>
```

这种继承方式的缺点是子类的实例可以访问父类的私有属性，子类的实例还可以更改该属性，这样不安全。

4 构造函数方式继承(Constructor mode inheritance)

如下例所示，构造函数方式继承,用.call()和.apply()将父类构造函数引入子类函数,父类原型上的方法不会被子类继承。

```
<script>
        function book(name,price) {
        this.name=name;
        this.price=price;
```

www.rjgczz.com

```
        this.mark=function() {
            console.log("在书可以标注");
        }
    }
    book.prototype.read=function() {
        console.log("书可以用来阅读");
    }
    function computerBook(name,price,lang)
    {
        this.lang=lang;
        book.apply(this,[name,price]);
        this.show=function() {
            console.log(this.name+"-"+this.
lang+"-"+this.price);
        }
    }
    var book1=new computerBook('javascript基
础','45','javascript');
    var book2=new computerBook('html5入
门','35','html5');
    book1.show();
    book2.show();
    //book1.read();不能调用
    book2.mark();//可以调用父类的方法
</script>
```

这种方式的优点是，借用构造函数可以解决原型中引用类型值被修改的问题，但是也存在一个问题，那就是，只能继承父对象的实例属性和方法，不能继承父对象原型属性和方法[5]。

5 组合继承(Combinatorial inheritance)

组合继承，就是原型链继承+借用构造函数。它的优点是：(1)能够在实例化子类对象的时候给继承来的属性赋值；(2)能够继承父类的原型对象中的方法。缺点是：继承了两次父类的模板，分别是call绑定和子类原型对象赋值的时候，如果继承来的属性特别多，这会很耗费时间来维护[6]。

```
<script>
function father(job)
  {
    this.gender="male";
    this.job=job;
  }
  father.prototype.getJob=function() {
    return this.job;
  }
  function mother(favorite) {
    this.favorite=favorite;
    this.showFavorite=function() {
      console.log("i like "+this.favorite);
    }
  }
  function son(job,favorite) {
    father.call(this,job);
    mother.call(this,favorite);
  }
  son.prototype=new father();
  son.constructor=son;
  var son1=new son('teacher','swimming');
  console.log(son1.getJob());//调用父类原型方法
  console.log(son1.gender);//父类属性
  son1.showFavorite();//调用父类的原型方法
</script>
```

6 寄生式继承(Parasitic inheritance)

寄生式继承是通过Object.create()将子类的原型继承到父类的原型上。寄生式继承其实就是对原型继承的第二次封装，在封装过程中对继承的对象进行了扩展，也存在原型继承的缺点，这种思想的作用也是为了寄生组合式继承模式的实现[7]。

```
<script>
  function book(name,price)
  {
    this.name=name;
    this.price=price;
  }
  function inheritObject(o) {
    var F=function() {
    }
    F.prototype=o;
    return F();
  }
  function createBook(o){
    // 通过原型继承方式创建对象
    var object=new inheritObject(o);
    // 拓展新对象
    o.show=function() {
console.log("书名:"+this.name+"单价:"+this.price)
    }
    // 返回扩展后的对象
    return o;
  }
  var book1=createBook(book);
  book1.name="javascript";
  book1.price="52";
```

www.rjgcz.com

```
    book1.show();
</script>
```

7 ES6新增的Class(类)(ES6 new class)

ES6新增的Class(类)，给我们编程带来了极大方便，可以通过class声明一个类，通过extends关键字来实现继承关系。新的class写法只是让对象原型的写法更加清晰、更像面向对象编程的语法而已，ES6中的类，实际上也是函数，写法更加面向对象，原理还是原型链[8]。

```
<script>
 class Person{
        constructor(name,age)
        {
            this.name=name;
            this.age=age;
        }
        say() {
            console.log("hello");
        }
        static run()
        {
            console.log("i can run");
        }
    }
     class Teacher extends Person{
        constructor(name,age,professional)
        {
            super(name,age);
            this.professional=professional;
            super.say();
        }
        teach() {
            console.log("i can teach");
        }
        say() {
            super.say();//调用父类方法
            console.log("hello everyone");
        }
    }
     class student extends Person{
        constructor(name,age,stuid)
        {
            super(name,age);
            this.stuid=stuid;
        }
        study() {
            console.log("I'm studying");
        }
        show() {
            console.log("姓名："+this.name+"；年龄"+this.age+";学号："+this.stuid);
        }
    }
    let student1=new student('zhangsan',16,'N0001');
        student1.show();
        student.run();//调用静态方法
        let teacher1=new Teacher('wangyong',35,'计算机');
        teacher1.teach();
        teacher1.say();
</script>
```

8 结论(Conclusion)

传统基于类的面向对象思维在一定程度上妨碍了大家对JavaScript面向对象特性的理解，本文通过比较分析及举例说明的研究方法深入讨论了JavaScript的面向对象的继承的特性。它的弱类型，简单易用性、解释性和跨平台性，使其在Web的应用开发中可以说是无处不在，但它的无类动态对象、原型链继承、组合继承、寄生式继承等特性使其具有更高的灵活性[9]。所以，只有深入理解JavaScript中实现继承的原理，实现继承机制才可以变得游刃有余。

参考文献(References)

[1] Douglas Crockford.Private Members in JavaScript[EB/OL].http://javascript.crockford.com/zh/private.html,2010-01-01.

[2] JohnResig.Simple JavaScript Inheritance[EB/OL].http://ejohn.org/blog/simple-javascript-inheritance,2011-01-01.

[3] 吕远.基于JavaScript原型链的继承机制研究[J].江阴工学院学报,2017(10):13-18.

[4] 石正喜.MySQL数据库实用教程[M].北京:北京师范大学出版社,2014.

[5] 姜承尧.高性能网站MySQL数据库实践[J].程序员,2013(9):49-53.

[6] 黄华林,宋阳秋.JavaScript面向对象特性浅析与范例[J].安庆师范学院学报:自然科学版,2005(4):85-88.

[7] Zakas N C.JavaScript高级程序设计(第3版)[M].北京:人民邮电出版社,2015:147-161.

[8] Nicholas C,Zakas.JavaScript面向对象精要[M].北京:人民邮电出版社,2014:53-60.

[9] Stoyan Stefanov,Kumar Chetan Sharma.JavaScript面向对象编程指南(第2版)[M].北京:人民邮电出版社,2015:4-6.

作者简介：

周 岚(1977-)，女，硕士，副教授.研究领域：程序设计，软件开发与数据库.

www.rjgcz.com