第十二课 继承

学习目录

- **什么是继承**
- **构造函数进阶应用**
- **继承的方式**

一．什么是继承

许多支持面向对象编程的语言都支持继承，继承分别有两种方式，一种是接口继承，另一种是实现继承，在 js 中只有实现继承，我们主要讨论的也是实现继承，实现继承的方式主要是依靠原型链来做到的。

其实继承简单来理解就是利用原型链让对象 a 拥有对象 b 的属性和方法。

```
var obj1 = {
    a:100,
    getNum:function(){
        console.log(this.a);
    }
};
var obj2 = obj1;

function Fn1(){
    this.num = 100;
```

```
}
Fn1.prototype.getNum = function(){
    console.log(this.num);
}

var f1 = new Fn1();
f1.getNum();

function Fn2(){
    this.job = 'web 培训'
}
Fn2.prototype = new Fn1();
Fn2.prototype.getJob = function(){
    console.log(this.job);
}

var f2 = new Fn2();
f2.getNum();
f2.getJob();
```

1. 每个构造函数都有一个原型对象 Fn1.prototype。
2. 每个原型对象都包含一个指向该构造函数的指针 Fn1.prototype.constructor。
3. 每个 new 出来的实例都包含一个指向原型对象的隐式原型 f1.__proto__

如果让一个构造函数的原型对象等于另一个构造函数的实例，根据原型链的这种关系，那么这个构造函数的 Fn2.prototype = new Fn1()，var f1 = new Fn1()实际上是 Fn2.prototype 根据__proto__找到了 f1.__proto__，f1.__proto 又找到了 Fn1.prototype，也就是 Fn2.prototype.__proto__通过 f1.__proto__隐式原型拥有了 Fn1.prototype 的属性和方法。

```
console.log(Fn1.prototype == f1.__proto__);//true
console.log(Fn2.prototype.__proto__ == f1.__proto__);//true
console.log(Fn2.prototype.__proto__ == Fn1.prototype);//true
```

二．构造函数进阶应用

问题：为什么我们要在构造函数里定义属性，在构造函数的原型对象上写方法？

1. 从实例化的过程来看，每个实例对象的属性大多数情况下都是不同的，所以大多数属性是可变属性，定义在构造函数里的属性优先级较高，所有的实例对象可以快速访问使用构造函数中的属性。

2. 我们可以通过实例对象来访问保存在构造函数原型对象上的值，但是不能通过实例对象来重写原型中的值，如果一个实例对象修改了一个原型上的属性，其他实例对象在去访问这个实例属性的时候这个实例属性就不是以前的属性了。

3. 方法大多数情况下是固定的，因此不必每次实例化的时候都把所有的方法实例化复制一遍。

4. 构造函数的继承是依托于原型链来完成，而在原型链中，构造函数的原型对象大多数情况下都是用来共享方法的，当然也可以共享一些很少改变的共享公用属性。

5. 每次实例化都可以得到一个构造函数实例属性的副本，同时又可以共享着公用的方

法，这样也最大限度的节省了内存，同时构造函数中的属性也支持传递参数来动态修改。

这种组合使用构造函数与原型组合使用的方式，在 js 组件化开发过程中最通用用到的组件化开发方式。

```
function Fn1(name,num){
    this.name = name;
    this.num = num;
}
Fn1.prototype.getTearcher = function(){
    console.log(this.name + this.num);
}

var f1 = new Fn1('阿凯老师',100);
var f2 = new Fn1('小燕老师',200);
f1.getTearcher();
f2.getTearcher();
```

三．继承的方式

在说继承的几种方式之前，我们补充说一下为什么当我们把属性放在构造函数中，把方法放在构造函数的原型对象上，通过 new 操作符实例化一个构造函数的实例对象之后，这个实例对象可以拥有构造函数的属性和构造函数原型对象上的方法呢？

那就要看下 new 运算符实例化一个构造函数之后发生了什么。

var fn1 = new Fn1();

1.首先声明一个空对象，这个空对象目前是还没有任何属性和方法的实例对象。

```
var fn1 = {};
```

2.把实例对象的隐式原型__proto__属性指向构造函数的原型，fn1 因此拥有了构造函数 prototype 原型对象上的属性和方法。

```
fn1.__proto__ =  Fn.prototype;
```

3.把构造函数的 this 指向替换成 fn1 实例对象，在执行构造函数，fn1 因此拥有了构造函数内部的属性和方法。

```
Fn.call(fn1);
```

继承的几种方式

继承的方式有很多，其中每个方式都有自己的有点和不足，因此不能说哪种继承方式好用，哪种继承方式不好用，只能说在实践开发过程中，许多组合在一起的组合继承方式，能够被大多数的编程人员认可和使用，所以我们会主要介绍实用的继承方式。

1.借用构造函数

这种方式，我们之前就用过，其实就是在子类型构造函数的内部调用父类型构造函数。

```
function Fn1(name){
 this.name = name;
 this.num = [100, 200, 300];
}
function Fn2(){
 Fn1.call(this"星星课堂",);  //子类 Fn2 继承了 父类 Fn1 构造函数内部的属性
```

```
this.job = “web 培训”;//同时子类内部也可以在父类调用以后定义自己内部的属性
}
var fn2 = new Fn2();
console.log(fn2.name); //"星星课堂";
console.log(fn2.job); //"web 培训"
```

这种继承方式的优点是子类能够继承父类构造函数中的属性，同时还能给父类构造函数中传递参数，缺点是子类没法继承父类构造函数原型对象上的方法。

2.组合继承

这种继承方式我们也用过，它的原理其实也挺简单好用，它是利用原型链实现对原型属性和方法的继承，同时通过借用构造函数来实现子类对父类构造函数内部属性的继承。

```
function Fn1(name){
 this.name = name;
}
Fn1.prototype.getName= function(){
 console.log(this.name);
};
function Fn2(name, job){
 Fn1.call(this, name); //借用组合构造函数继承属性
 this.job = job;
}
Fn2.prototype = new Fn1(); //使用原型链继承方法
```

```
Fn2.prototype.constructor = Fn2;
Fn2.prototype.getJob= function(){
 console.log(this.job);
};
```

组合继承融合了原型链和借用构造函数的优点，也是比较常用的继承方式之一，通过 instanceof 和 isPrototypeOf()也能够用于识别基于组合继承创建出来的对象是哪种类型的对象。子类如果要修改原型对象上继承来的父类原型对象的方法，父类原型对象上的方法也不会被改写。

```
Fn2.prototype.getName = function(){
    console.log(this.name + '子类不会修改父类原型对象方法');
}
var fn1 = new Fn1('xingxingclassroom');
fn1.getName();//xingxingclassroom
var fn2 = new Fn2('星星课堂','web 培训');
fn2.getName();//星星课堂子类不会修改父类原型对象方法
```

组合继承问题是如果使用组合继承，就会调用两次父类构造函数：一次是在构建子类原型对象的时候，另一次是在子类构造函数内部构建子类属性的时候。

在第一次调用父类构造函数时，子类的原型对象会先得到 name 属性，这个属性是父类的实例属性，只不过现在这个属性在子类的原型对象中。

当调用子类构造函数时，又会调用一次父类构造函数，这一次又在新对象上构建了实例属性 name，于是，这个子类的实例属性就屏蔽了子类原型对象中的 name 同名属性。

3.原型式继承

当我们只想让一个对象与另一个对象保持类似结构的时候，这时候我们没必要构建构造函数来大费周章，因为我们还可以使用一种对象浅拷贝的方式来做到继承，原型式继承可以专门用来做这种继承方式。

```
var obj1 = {
 name: "星星课堂",
 num: [100, 200, 300]
};
var obj2 = Object.create(obj1);
obj2.name = "xingxingclassroom";
obj2.num.push(600);

var obj3 = Object.create(obj1);
obj3.name = "Linda";
obj3.num.push(900);
console.log(obj1.num); //100,200,300,600,900
```

这种继承方式在继承包含引用类型值的属性的时候会共享相应的引用类型的值，也就是说修改其中任何一个对象的引用类型的值，其他对象对应属性的引用类型的值都会改变。

4.寄生式继承

与原型式继承类似，当我们不想考虑用构造函数来做对象继承的时候，也可以用寄生式继承，这种方式是定义一个独立的函数，在这个函数内部以某种方式来增强参数对象，在增强对象之后在返回对象。

```
function enhanceObj(obj){
    var newObj = Object.create(obj); //通过 Object.create()构建一个新对象
    newObj.setJob = function(){ //给这个对象添加新方法来增强这个对象
        console.log('web 培训');
    };
    return newObj; //返回这个对象
}

var obj1 = {
    name: "星星课堂",
    num: [100, 200, 300]
};
var obj2 = enhanceObj(obj1);
obj2.setJob(); //web 培训
```

5.寄生组合式继承

这个继承方式也是比较好用的，它能够解决之前说过的组合继承调用两次父类构造函数的问题，节约了内存开销。

```
function Fn1(name){
```

```
        this.name = name;
    }
    Fn1.prototype.getName = function(){
        console.log(this.name);
    }

    function Fn2(name){
        Fn1.call(this,name);
    }
    enhanceObj(Fn2,Fn1);
    var fn2 = new Fn2('星星课堂');
    fn2.getName();

    function enhanceObj(f2, f1){//f1 代表父类，f2 代表子类
        var prototype = Object.create(f1.prototype); //根据父类构建一个新的原型对象
        prototype.constructor = f2; //修改这个原型对象的 constructor 属性，也就是增强这个原型对象
        f2.prototype = prototype; //把子类的原型对象指定为这个增强之后的原型对象
    }
```

这种寄生组合式继承也有一些其他写法，但是原理都是一样的，利用寄生式继承构建一

个函数来增强对象 ,在使用组合式继承方法先用借用构造函数的形式让子类继承父类构造函数中的属性，在用原型链让子类继承父类原型对象的方法。

谢谢观看！

我是星星课堂老师：周小周