
黑马程序员™
www.itheima.com
传智播客旗下
高端IT教育品牌
构造函数和原型

目录 Contents

1. 构造函数和原型

1.1 概述

在典型的 OOP 的语言中（如 Java），都存在类的概念，类就是对象的模板，对象就是类的实例，但在 ES6之前，JS 中并没用引入类的概念。

ES6， 全称 ECMAScript 6.0 ， 2015.06 发版。但是目前浏览器的 JavaScript 是 ES5 版本，大多数高版本的浏览器也支持 ES6，不过只实现了 ES6 的部分特性和功能。

在 ES6之前 ， 对象不是基于类创建的，而是用一种称为构建函数的特殊函数来定义对象和它们的特征。

创建对象可以通过以下三种方式：

1. 对象字面量

2. new Object()

3. 自定义构造函数

1. 构造函数和原型

1.2 构造函数

构造函数是一种特殊的函数，主要用来初始化对象，即为对象成员变量赋初始值，它总与 new 一起使用。我们可以把对象中一些公共的属性和方法抽取出来，然后封装到这个函数里面。

在 JS 中，使用构造函数时要注意以下两点：

1. 构造函数用于创建某一类对象，其首字母要大写
2. 构造函数要和 new 一起使用才有意义

1. 构造函数和原型

1.2 构造函数

构造函数是一种特殊的函数，主要用来初始化对象，即为对象成员变量赋初始值，它总与 new 一起使用。我们可以把对象中一些公共的属性和方法抽取出来，然后封装到这个函数里面。

new 在执行时会做四件事情：

① 在内存中创建一个新的空对象。

② 让 this 指向这个新的对象。

③ 执行构造函数里面的代码，给这个新对象添加属性和方法。

④ 返回这个新对象（所以构造函数里面不需要 return ）。

1. 构造函数和原型

1.2 构造函数

JavaScript 的构造函数中可以添加一些成员，可以在构造函数本身上添加，也可以在构造函数内部的 this 上添加。通过这两种方式添加的成员，就分别称为静态成员和实例成员。

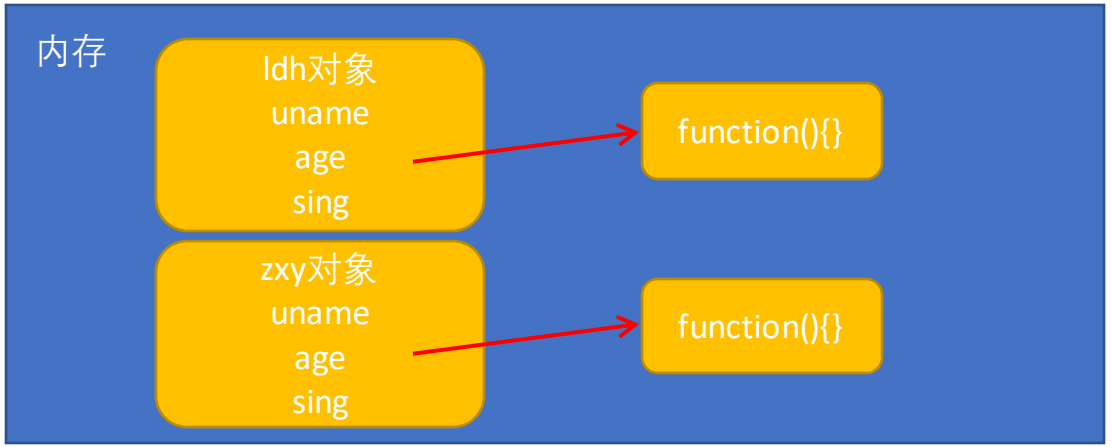
- 静态成员：在构造函数本上添加的成员称为静态成员，只能由构造函数本身来访问
- 实例成员：在构造函数内部创建的对象成员称为实例成员，只能由实例化的对象来访问

1. 构造函数和原型

1.3 构造函数的问题

构造函数方法很好用，但是存在浪费内存的问题。

```
function Star(uname, age){
    this.uname = uname;
    this.age = age;
    this.sing = function() {
        console.log('我会唱歌');
}
}
var ldh = new Star('刘德华', 18);
var zxy = new Star('张学友', 19);
```



我们希望所有的对象使用同一个函数，这样就比较节省内存，那么我们要怎样做呢？

1.4 构造函数原型 prototype

构造函数通过原型分配的函数是所有对象所**共享的**。

JavaScript 规定，每一个构造函数都有一个 prototype 属性，指向另一个对象。注意这个 prototype 就是一个对象，这个对象的所有属性和方法，都会被构造函数所拥有。

我们可以把那些不变的方法，直接定义在 prototype 对象上，这样所有对象的实例就可以共享这些方法。

问答?

1. 原型是什么？

一个对象，我们也称为 prototype 为原型对象。

2. 原型的作用是什么？

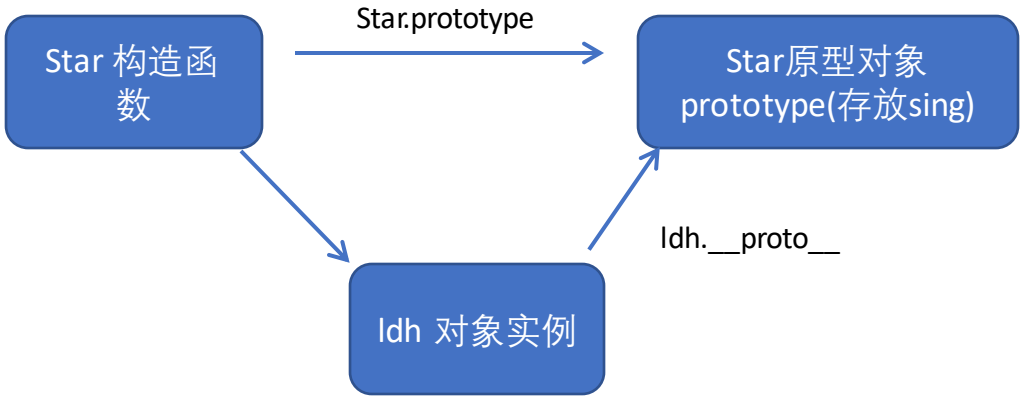
共享方法。

1. 构造函数和原型

1.5 对象原型 __proto__

对象都会有一个属性 __proto__ 指向构造函数的 prototype 原型对象，之所以我们对象可以使用构造函数 prototype 原型对象的属性和方法，就是因为对象有 __proto__ 原型的存在。

- __proto__对象原型和原型对象 prototype 是等价的
- __proto__对象原型的意义就在于为对象的查找机制提供一个方向，或者说一条路线，但是它是一个非标准属性，因此实际开发中，不可以使用这个属性，它只是内部指向原型对象 prototype



1.6 constructor 构造函数

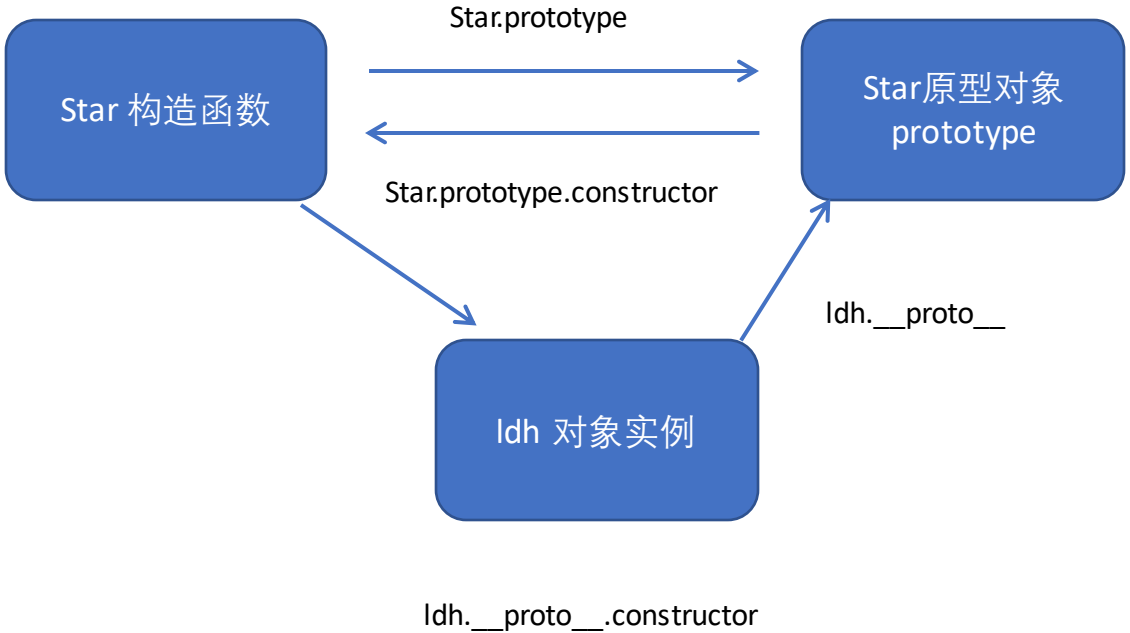
对象原型（__proto__）和构造函数（prototype）原型对象里面都有一个属性 constructor 属性，constructor 我们称为构造函数，因为它指回构造函数本身。

constructor 主要用于记录该对象引用于哪个构造函数，它可以让原型对象重新指向原来的构造函数。

一般情况下，对象的方法都在构造函数的原型对象中设置。如果有多个对象的方法，我们可以给原型对象采取对象形式赋值，但是这样就会覆盖构造函数原型对象原来的内容，这样修改后的原型对象 constructor 就不再指向当前构造函数了。此时，我们可以在修改后的原型对象中，添加一个 constructor 指向原来的构造函数。

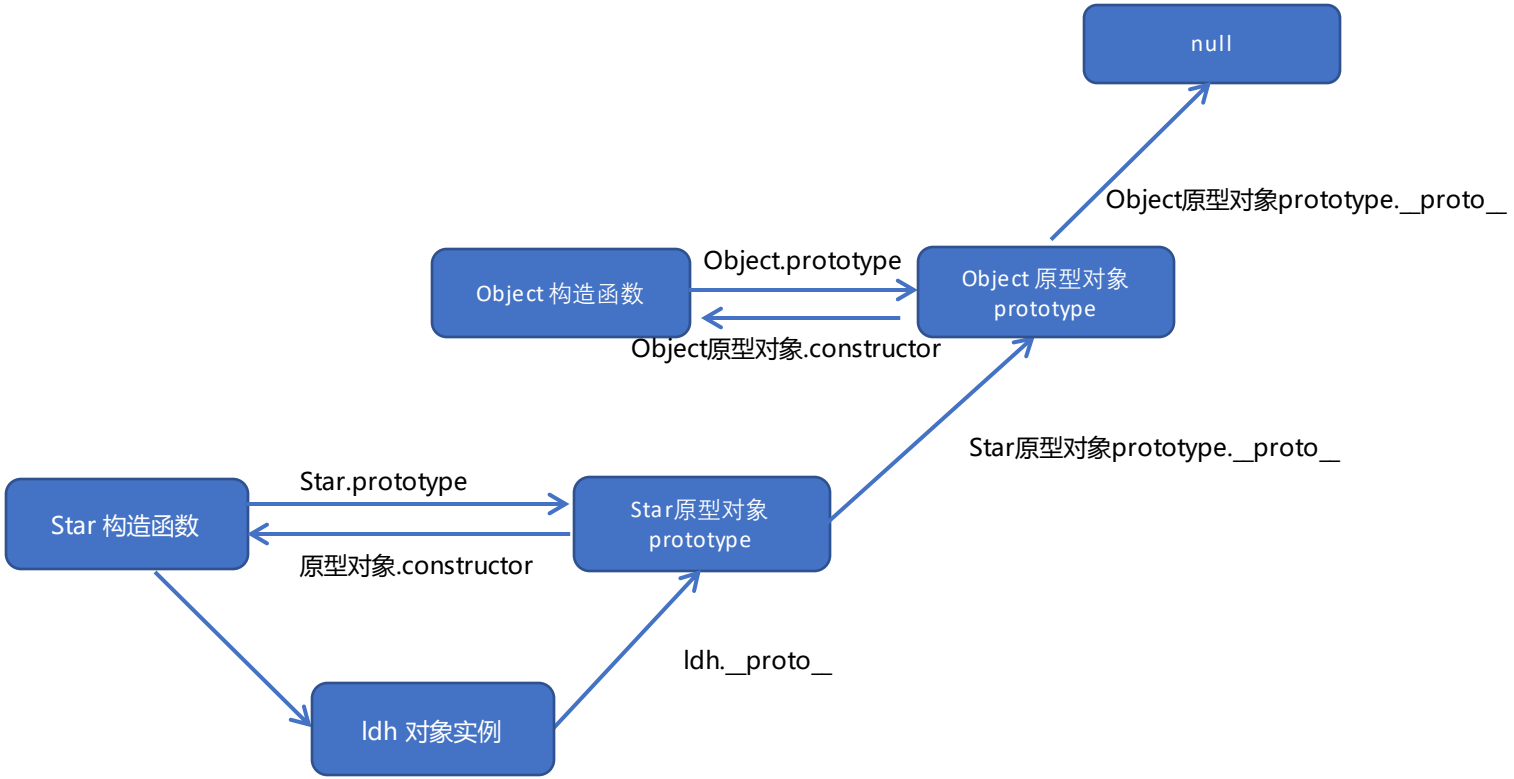
1. 构造函数和原型

1.7 构造函数、实例、原型对象三者之间的关系



1. 构造函数和原型

1.8 原型链



1. 构造函数和原型

1.9 JavaScript 的成员查找机制(规则)

① 当访问一个对象的属性（包括方法）时，首先查找这个对象自身有没有该属性。

② 如果没有就查找它的原型（也就是 __proto__指向的 prototype 原型对象）。

③ 如果还没有就查找原型对象的原型（Object的原型对象）。

④ 依此类推一直找到 Object 为止（null）。

⑤ __proto__对象原型的意义就在于为对象成员查找机制提供一个方向，或者说一条路线。

1. 构造函数和原型

1.10 原型对象this指向

构造函数中的this 指向我们实例对象.

原型对象里面放的是方法，这个方法里面的this 指向的是 这个方法的调用者, 也就是这个实例对象.

1. 构造函数和原型

1.11 扩展内置对象

可以通过原型对象，对原来的内置对象进行扩展自定义的方法。比如给数组增加自定义求偶数和的功能。

注意：数组和字符串内置对象不能给原型对象覆盖操作 Array.prototype = {} ，只能是 Array.prototype.xxx = function(){} 的方式。

目录 Contents

2. 继承

ES6之前并没有给我们提供 extends 继承。我们可以通过构造函数+原型对象模拟实现继承，被称为组合继承。

2.1 call()

调用这个函数, 并且修改函数运行时的 this 指向

```
fun.call(thisArg, arg1, arg2, ...)
```

- thisArg ： 当前调用函数 this 的指向对象
- arg1，arg2：传递的其他参数

2. 继承

ES6之前并没有给我们提供 extends 继承。我们可以通过构造函数+原型对象模拟实现继承，被称为组合继承。

2.2 借用构造函数继承父类型属性

核心原理： 通过 call() 把父类型的 this 指向子类型的 this ， 这样就可以实现子类型继承父类型的属性。

```
// 父类
function Person(name, age, sex) {
  this.name = name;
  this.age = age;
  this.sex = sex;
}
// 子类
function Student(name, age, sex, score) {
  Person.call(this, name, age, sex);  // 此时父类的 this 指向子类的 this，同时调用这个函数
  this.score = score;
}
var s1 = new Student('zs', 18, '男', 100);
console.dir(s1);
```

2. 继承

ES6之前并没有给我们提供 extends 继承。我们可以通过构造函数+原型对象模拟实现继承，被称为组合继承。

2.3 借用原型对象继承父类型方法

一般情况下，对象的方法都在构造函数的原型对象中设置，通过构造函数无法继承父类方法。

核心原理：

① 将子类所共享的方法提取出来，让子类的 prototype 原型对象 = new 父类()

② 本质：子类原型对象等于是实例化父类，因为父类实例化之后另外开辟空间，就不会影响原来父类原型对象

③ 将子类的 constructor 从新指向子类的构造函数

3. 类的本质

1. class本质还是function.

2. 类的所有方法都定义在类的prototype属性上

3. 类创建的实例,里面也有__proto__ 指向类的prototype原型对象

4.所以ES6的类它的绝大部分功能，ES5都可以做到，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。

5.所以ES6的类其实就是语法糖.

6. 语法糖:语法糖就是一种便捷写法. 简单理解, 有两种方法可以实现同样的功能, 但是一种写法更加清晰、方便,那么这个方法就是语法糖

目录 Contents

3. ES5 中的新增方法

3.1 ES5 新增方法概述

ES5 中给我们新增了一些方法，可以很方便的操作数组或者字符串，这些方法主要包括：

- 数组方法
- 字符串方法
- 对象方法

3. ES5 中的新增方法

3.2 数组方法

迭代(遍历)方法：forEach()、map()、filter()、some()、every()；

```
array.forEach(function(currentValue, index, arr))
```

- currentValue：数组当前项的值
- index：数组当前项的索引
- arr：数组对象本身

3. ES5 中的新增方法

3.2 数组方法

迭代(遍历)方法：forEach()、map()、filter()、some()、every()；

```
array.filter(function(currentValue, index, arr))
```

- filter() 方法创建一个新的数组，新数组中的元素是通过检查指定数组中符合条件的所有元素,主要用于筛选数组
- 注意它直接返回一个新数组
- currentValue: 数组当前项的值
- index：数组当前项的索引
- arr：数组对象本身

3. ES5 中的新增方法

3.2 数组方法

迭代(遍历)方法：forEach()、map()、filter()、some()、every()；

```
array.some(function(currentValue, index, arr))
```

- some() 方法用于检测数组中的元素是否满足指定条件. 通俗点 查找数组中是否有满足条件的元素
- 注意它返回值是布尔值, 如果查找到这个元素, 就返回true ， 如果查找不到就返回false.
- 如果找到第一个满足条件的元素,则终止循环. 不在继续查找.
- currentValue: 数组当前项的值
- index：数组当前项的索引
- arr：数组对象本身

3. ES5 中的新增方法

3.2 数组方法

查询商品案例

1. 把数据渲染到页面中 (forEach)
2. 根据价格显示数据 (filter)
3. 根据商品名称显示数据

3. ES5 中的新增方法

3.3 字符串方法

trim() 方法会从一个字符串的两端删除空白字符。

```
str.trim()
```

trim() 方法并不影响原字符串本身，它返回的是一个新的字符串。

3. ES5 中的新增方法

3.4 对象方法

1. Object.keys() 用于获取对象自身所有的属性

```
Object.keys(obj)
```

- 效果类似 for...in
- 返回一个由属性名组成的数组

3. ES5 中的新增方法

3.4 对象方法

2. Object.defineProperty() 定义对象中新属性或修改原有的属性。(了解)

```
Object.defineProperty(obj, prop, descriptor)
```

- obj：必需。目标对象
- prop：必需。需定义或修改的属性的名字
- descriptor：必需。目标属性所拥有的特性

3. ES5 中的新增方法

3.4 对象方法

2. Object.defineProperty() 定义新属性或修改原有的属性。

```
Object.defineProperty(obj, prop, descriptor)
```

Object.defineProperty() 第三个参数 descriptor 说明： 以对象形式 { } 书写

- value: 设置属性的值 默认为undefined
- writable: 值是否可以重写。true | false 默认为false
- enumerable: 目标属性是否可以被枚举。true | false 默认为 false
- configurable: 目标属性是否可以被删除或是否可以再次修改特性 true | false 默认为false

传智播客旗下高端IT教育品牌