JavaScript 进阶第三天

深入面向对象

黑马程序员
www.itheima.com

传智教育旗下
高端IT教育品牌

学习目标

Learning Objectives

1. 理解面向对象思想，掌握函数原型对象
2. 运用面向对象封装确认框对话框功能

目录

Contents

编程思想

- 面向过程介绍
- 面向对象介绍

1.1 面向过程编程

目标：从生活例子了解什么是面向过程编程

- **面向过程**就是分析出解决问题所需要的步骤，然后用函数把这些步骤一步一步实现，使用的时候再一个一个的依次调用就可以了。
- **举个栗子：蛋炒饭**

- 面向过程，就是按照我们分析好了的步骤，按照步骤解决问题。

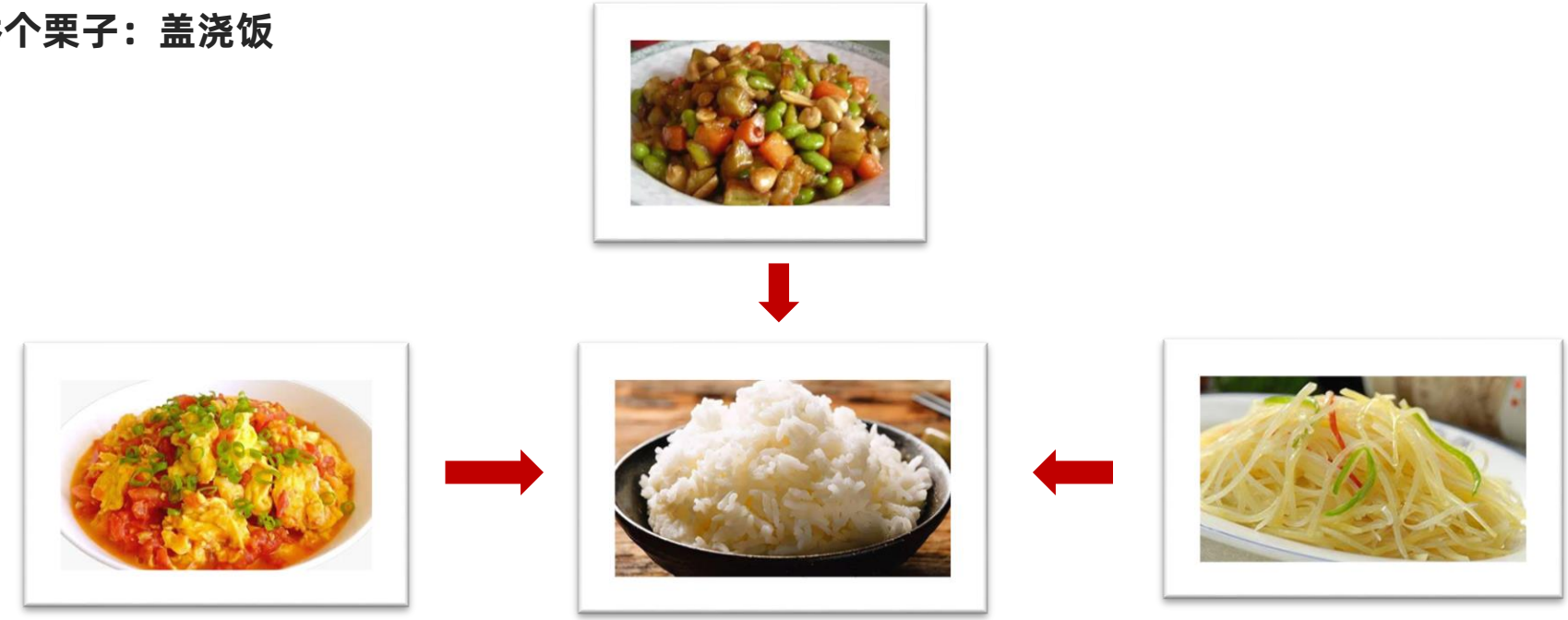
编程思想

- 面向过程介绍
- 面向对象介绍

1.2 面向对象编程 (oop)

目标：从生活例子了解什么是面向对象编程

- **面向对象**是把事务分解成为一个个对象，然后由对象之间分工与合作。
- **举个栗子：盖浇饭**



- 面向对象是以对象功能来划分问题，而不是步骤。

1.2 面向对象编程 (oop)

- 在面向对象程序开发思想中，每一个对象都是功能中心，具有明确分工。
- 面向对象编程具有灵活、代码可复用、容易维护和开发的优点，更适合多人合作的大型软件项目。
- 面向对象的特性：
  - 封装性
  - 继承性
  - 多态性

继承：继承自拖拉机，实现了扫地的接口

封装：无需知道如何运作，开动即可

多态：平时扫地，天热当风扇

重用：没用额外动力，重复利用了发动机能量

多线程：多个扫把同时工作

低耦合：扫把可以换成拖把而无须改动

组件编程：每个配件都是可单独利用的工具

适配器模式：无需造发动机，继承自拖拉机，只取动力方法

代码托管：无需管理垃圾，直接扫到路边即可

1. 编程思想-面向过程和面向对象的对比

面向过程编程

优点：性能比面向对象高，适合跟硬件联系很紧密的东西，例如单片机就采用的面向过程编程。

缺点：没有面向对象易维护、易复用、易扩展

面向对象编程

优点：易维护、易复用、易扩展，由于面向对象有封装、继承、多态性的特性，可以设计出低耦合的系统，使系统 更加灵活、更加易于维护

缺点：性能比面向过程低

生活离不开蛋炒饭，也离不开盖浇饭，选择不同而已，只不过前端不同于其他语言，面向过程更多

目录

Contents

构造函数

2. 构造函数

- 封装是面向对象思想中比较重要的一部分，js面向对象可以通过构造函数实现的封装。
- 同样的将变量和函数组合到了一起并能通过 this 实现数据的共享，所不同的是借助构造函数创建出来的实例对象之间是彼此不影响的

```
function Star(uname, age) {
  this.uname = uname
  this.age = age
  this.sing = function () {
    console.log('我会唱歌')
  }
}
// 实例对像，获得了构造函数中封装的所有逻辑
const ldh = new Star('刘德华', 18)
const zxy = new Star('张学友', 19)
```

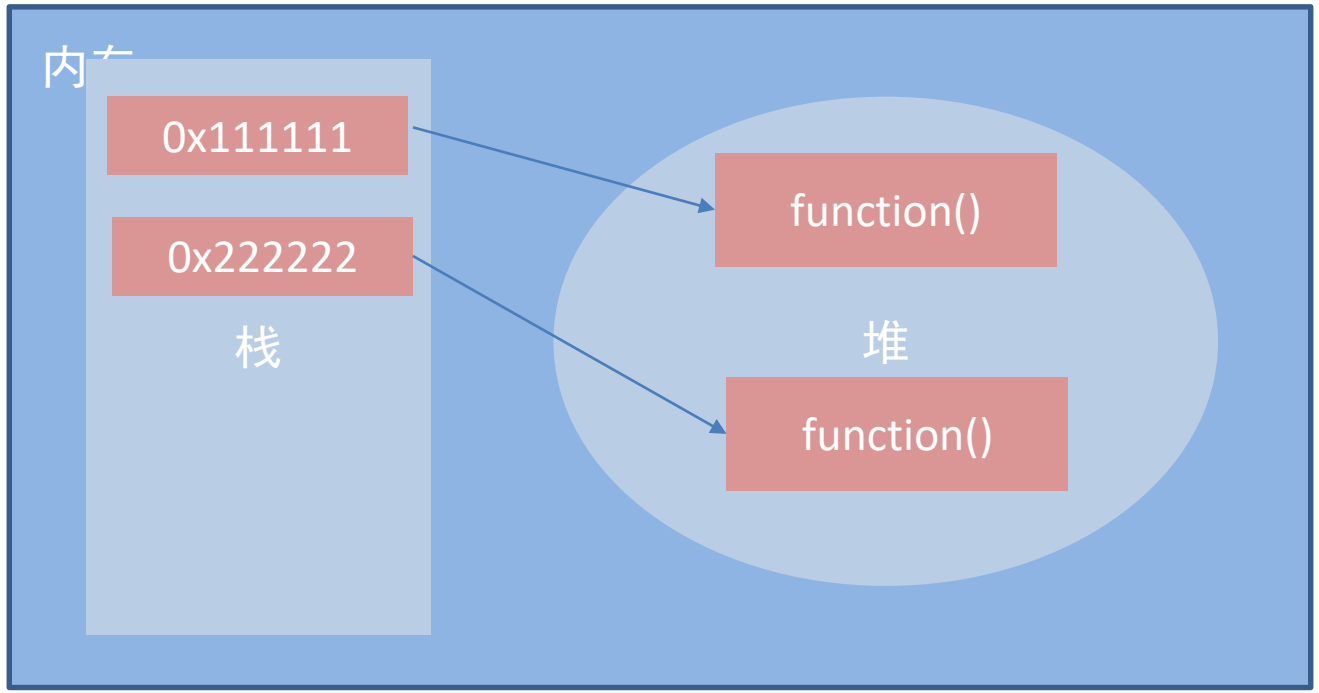
总结：

1. 构造函数体现了面向对象的封装特性
2. 构造函数实例创建的对象彼此独立、互不影响

2. 构造函数

- 封装是面向对象思想中比较重要的一部分，js面向对象可以通过构造函数实现的封装。
- 前面我们学过的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')
    }
}
const ldh = new Star('刘德华', 18)
const zxy = new Star('张学友', 19)
```



2. 构造函数

- 面向对象编程的特性：比如封装性、继承性等，可以借助于构造函数来实现

前面我们学过的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')
    }
}
const ldh = new Star('刘德华', 18)
const zxy = new Star('张学友', 19)
```

```
function Star(uname, age) {
  this.uname = uname;
  this.age = age;
  this.sing = function () {
    console.log('我会唱歌');
  }
}
const ldh = new Star('刘德华', 18)
const zxy = new Star('张学友', 19)
console.log(ldh.sing === zxy.sing)  // 结果是 false 说明俩函数不一样
```

总结

1. Js 实现面向对象需要借助于谁来实现？
   - 构造函数
2. 构造函数存在什么问题？
   - 浪费内存

2. 构造函数

- 面向对象编程的特性：比如封装性、继承性等，可以借助于构造函数来实现

前面我们学过的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')
    }
}
const ldh = new Star('刘德华', 18)
const zxy = new Star('张学友', 19)
```

```
function Star(uname, age) {
  this.uname = uname;
  this.age = age;
  this.sing = function () {
    console.log('我会唱歌');
  }
}
const ldh = new Star('刘德华', 18)
const zxy = new Star('张学友', 19)
console.log(ldh.sing === zxy.sing)  // 结果是 false 说明俩函数不一样
```

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？

目录

Contents

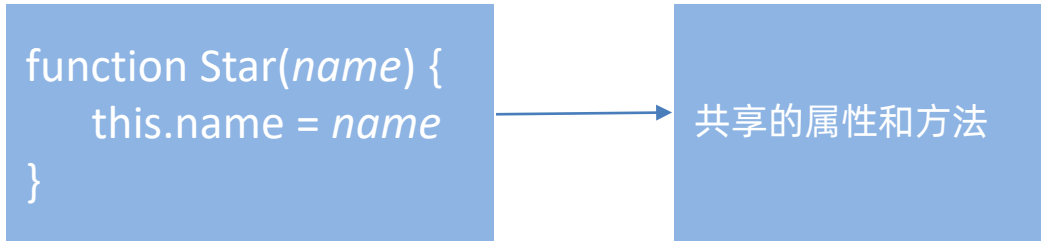
原型

- 原型
- constructor 属性
- 对象原型
- 原型继承
- 原型链

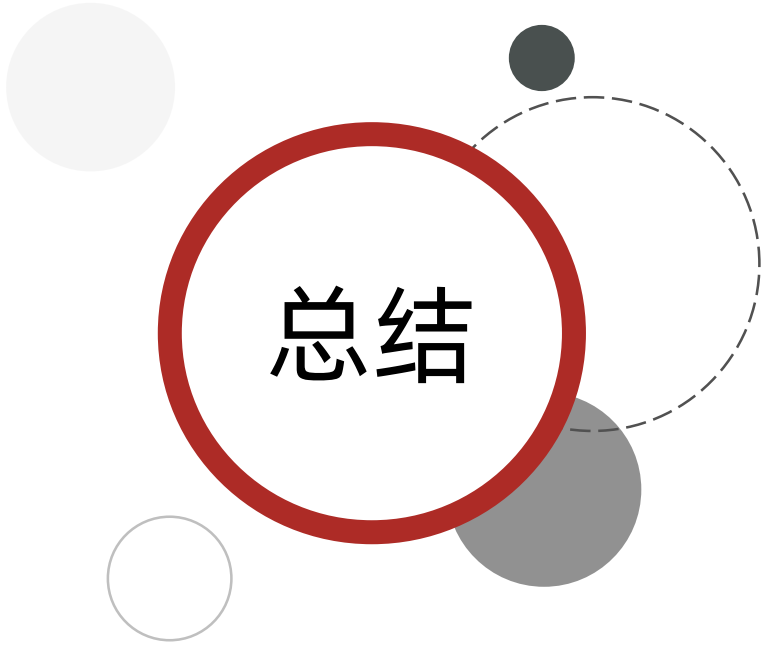
3.1 原型

目标：能够利用原型对象实现方法共享

- 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，所以我们也称为原型对象
- 这个对象可以挂载函数，对象实例化不会多次创建原型上函数，节约内存
- 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。
- 构造函数和原型对象中的this 都指向 实例化的对象




```
function Star(uname, age) {
  this.uname = uname
  this.age = age
}
console.log(Star.prototype) // 返回一个对象称为原型对象
Star.prototype.sing = function () {
  console.log('我会唱歌')
}
const ldh = new Star('刘德华', 18)
const zxy = new Star('张学友', 19)
console.log(ldh.sing === zxy.sing)  // 结果是 true 说明俩函数一样，共享
```



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
   - 可以把那些不变的方法，直接定义在 prototype 对象上
3. 构造函数和原型里面的this指向谁？
   - 实例化的对象

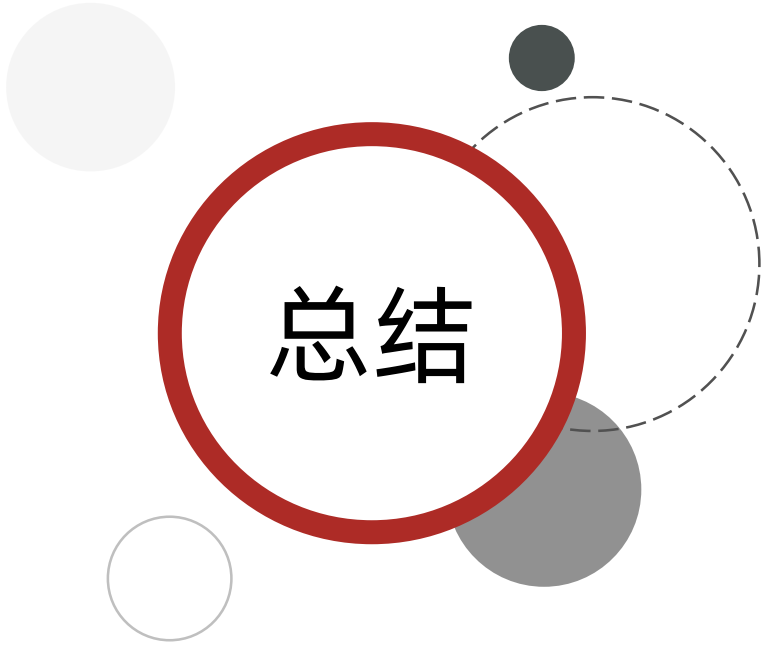
3.1 原型- this指向

目标：能够说出构造函数和原型对象中的this 指向

构造函数和原型对象中的this 都指向 实例化的对象

```
let that
function Person(name) {
  this.name = name
  that = this
}
const o = new Person()
console.log(that === o)  // true
```

```
let that
function Person(name) {
  this.name = name
}
Person.prototype.sing = function () {
  that = this
}
const o = new Person()
o.sing()
console.log(that === o)  // true
```



1. 构造函数和原型里面的this指向谁？
   - 实例化的对象

练习

- **给数组扩展方法**

需求：

①：给数组扩展求最大值方法和求和方法

比如： 以前学过

const arr = [1,2,3]

arr.reverse()　结果是 [3,2,1]

扩展完毕之后：

arr.sum()　返回的结果是 6

练习

- **给数组扩展方法**

需求：

①：给数组扩展求最大值方法和求和方法

```
Array.prototype.max = function () {
  // this指向方法的调用者，就是 数组[1,2,3]
  // console.log(this)
  return Math.max(...this)
}
console.log([1, 2, 3].max())
```

```
Array.prototype.sum = function () {
  return this.reduce((prev, item) => prev + item, 0)
}
console.log([1, 2, 3].sum())
```

03

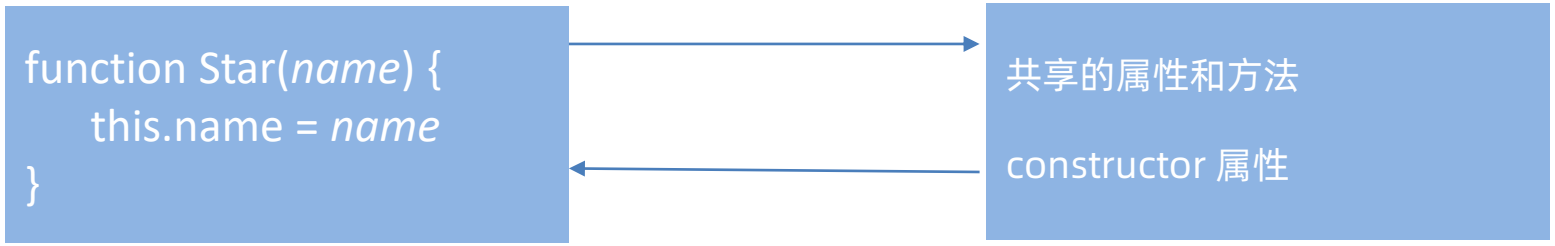
原型

- 原型
- constructor 属性
- 对象原型
- 原型继承
- 原型链

3.2 constructor 属性

在哪里？ 每个原型对象里面都有个constructor 属性（constructor 构造函数）

作用： 该属性指向该原型对象的构造函数， 简单理解，就是指向我的爸爸，我是有爸爸的孩子



3.2 constructor 属性

目标：了解constructor属性的作用

使用场景：

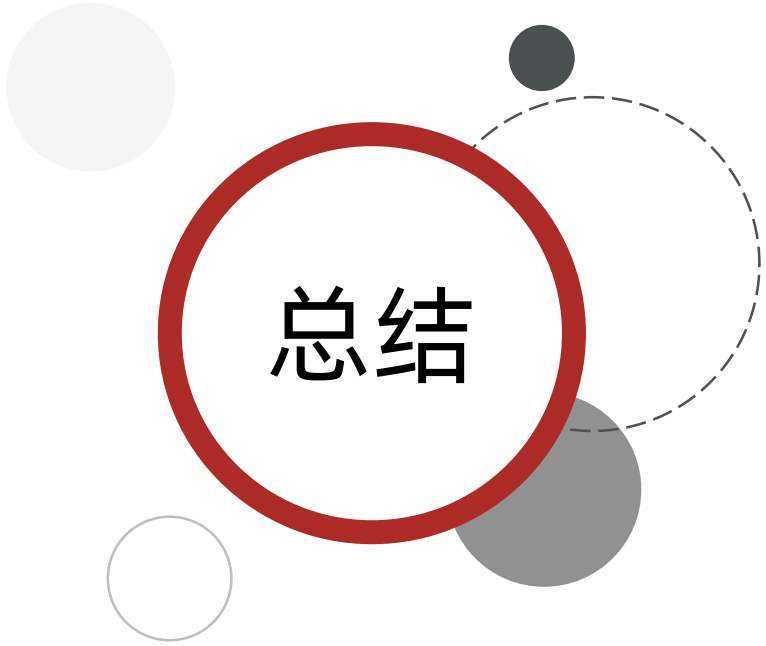
如果有多个对象的方法，我们可以给原型对象采取对象形式赋值.

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。

```
function Star(name) {
  this.name = name
}
Star.prototype = {
  sing: function () { console.log('唱歌') },
  dance: function () { console.log('跳舞') }
}
console.log(Star.prototype.constructor) // 指向 Object
```

```
function Star(name) {
  this.name = name
}
Star.prototype = {
  // 手动利用constructor 指回 Star构造函数
  constructor: Star,
  sing: function () { console.log('唱歌') },
  dance: function () { console.log('跳舞') }
}
console.log(Star.prototype.constructor) // 指向 Star
```

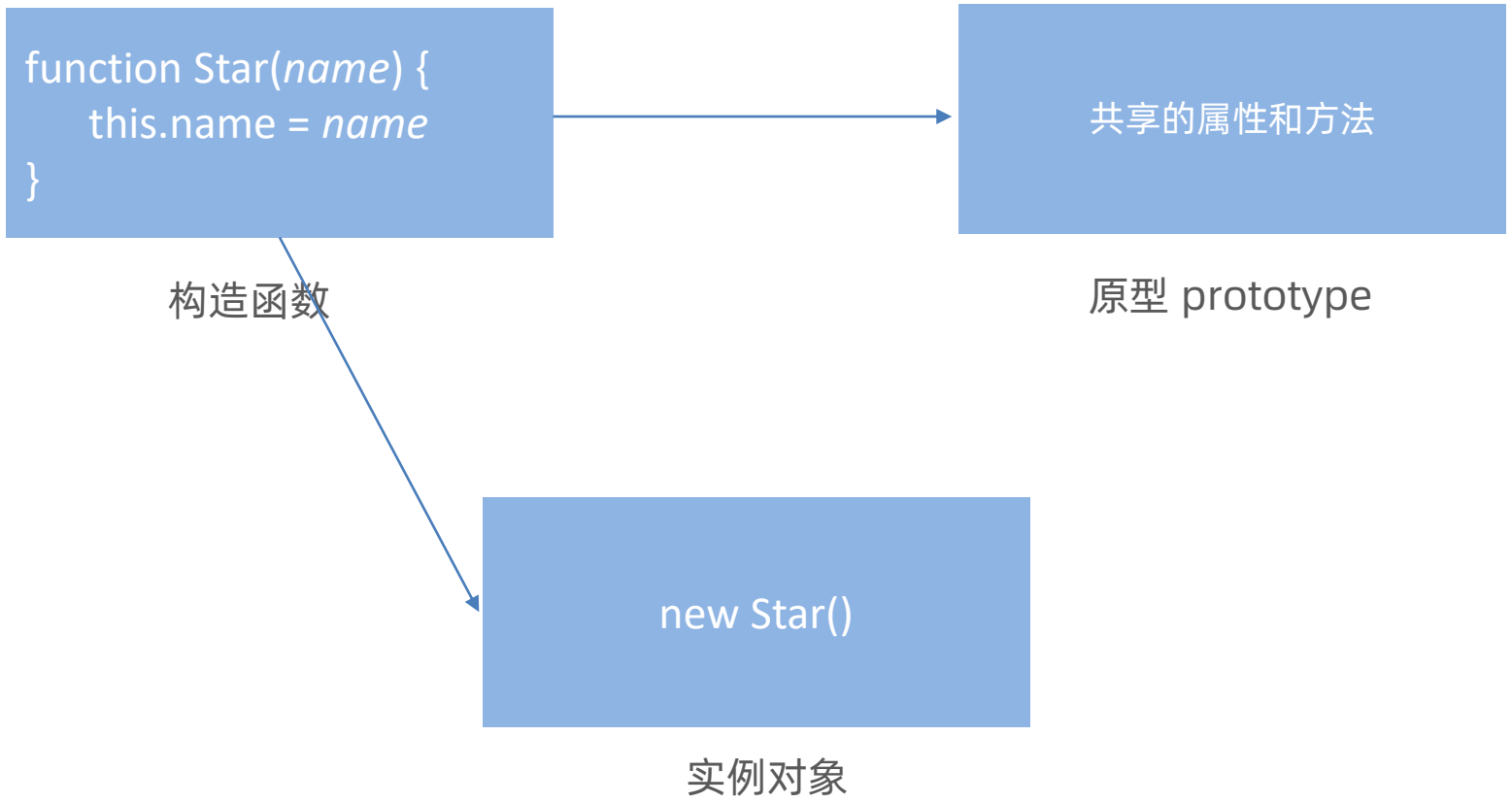


1. constructor属性的作用是什么？
   - 指向该原型对象的构造函数

思考

构造函数可以创建实例对象，构造函数还有一个原型对象，一些公共的属性或者方法放到这个原型对象身上

但是 为啥实例对象可以访问原型对象里面的属性和方法呢？

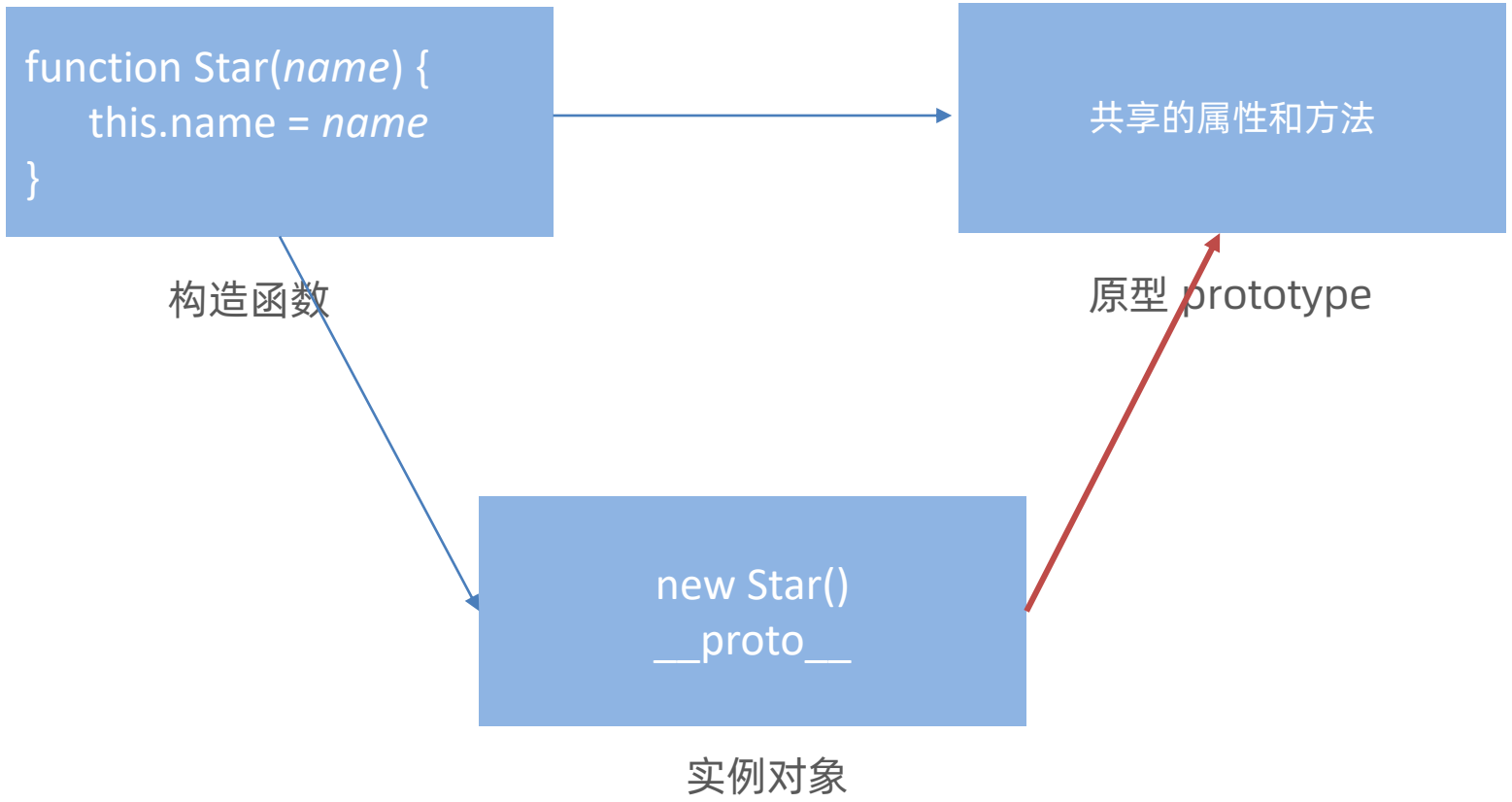


原型

- 原型
- constructor 属性
- 对象原型
- 原型继承
- 原型链

3.3 对象原型

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。



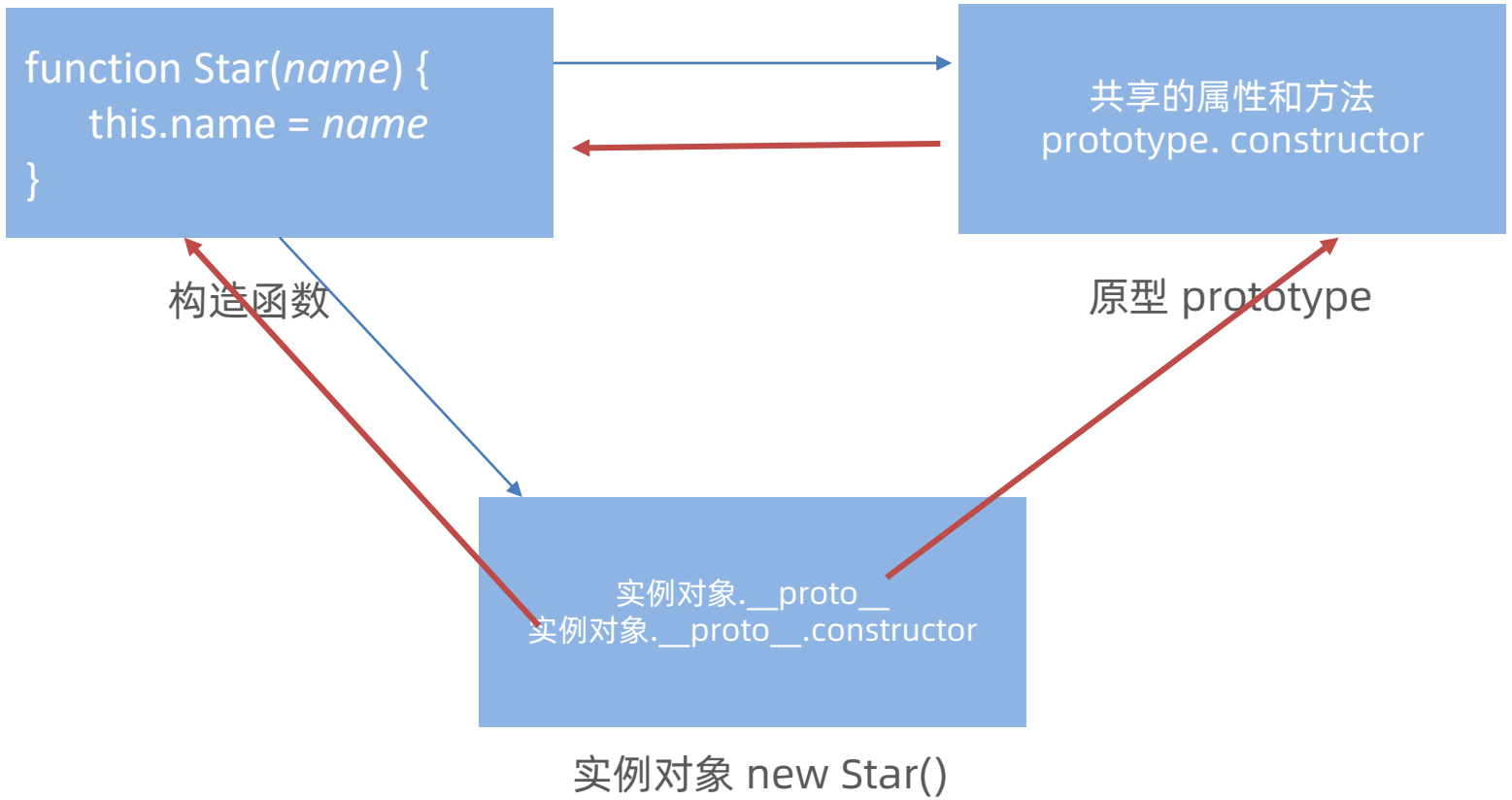
3.3 对象原型

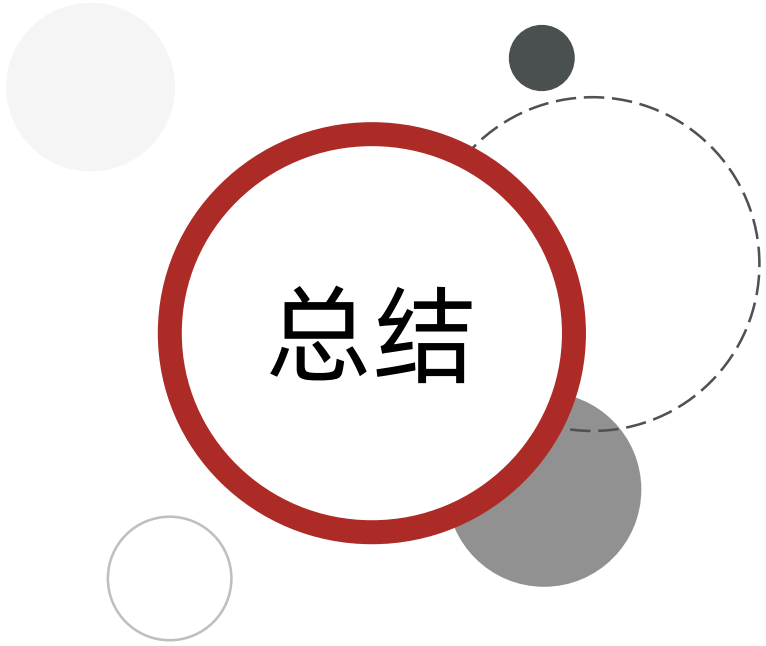
注意：

- __proto__ 是JS非标准属性
- [[prototype]]和__proto__意义相同
- 用来表明当前实例对象指向哪个原型对象prototype
- __proto__对象原型里面也有一个 constructor属性，指向创建该实例对象的构造函数

3.3 对象原型

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。





1. prototype是什么？哪里来的？
   - 原型（原型对象）
   - 构造函数都自动有原型
2. constructor属性在哪里？作用干啥的？
   - prototype原型和对象原型__proto__里面都有
   - 都指向创建实例对象/原型的构造函数
3. __proto__属性在哪里？指向谁？
   - 在实例对象里面
   - 指向原型 prototype

练习

- **根据下面代码，请画图**

需求：

①：利用画图工具，画出 构造函数 原型 实例对象 三者的关系

②：要求里面有__proto__ 和 constructor 的指向

③：并在代码打印验证指向正确

```
function Person() {
  this.name = name
}
const peppa = new Person('佩奇')
```

原型

- 原型
- constructor 属性
- 对象原型
- 原型继承
- 原型链

3.4 原型继承

继承是面向对象编程的另一个特征，通过继承进一步提升代码封装的程度，JavaScript 中大多是借助原型对象实现继承的特性。

龙生龙、凤生凤、老鼠的儿子会打洞描述的正是继承的含义。

我们来看个代码：

```
function Man() {
  this.head = 1;
  this.eyes = 2;
  this.legs = 2;
  this.say = function () {},
  this.eat = function () {}
}
const pink = new Man()
```

```
function Woman() {
  this.head = 1;
  this.eyes = 2;
  this.legs = 2;
  this.say = function () { },
  this.eat = function () { },
  this.baby = function () { }
}
const red = new Woman()
```

3.4 原型继承

1. 封装-抽取公共部分

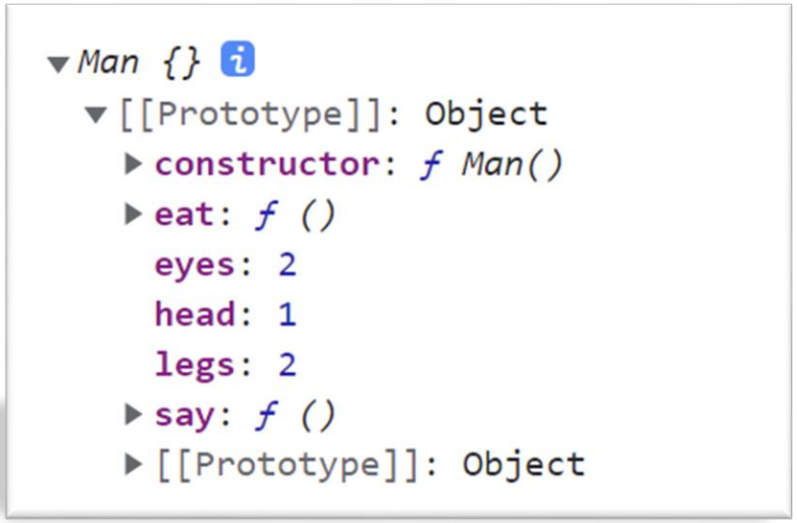
把男人和女人公共的部分抽取出来放到人类里面

```
// 人类
const People = {
  head: 1,
  eyes: 2,
  legs: 2,
  say: function () { },
  eat: function () { }
}
// 男人
function Man() {
}
// 女人
function Woman() {
  this.baby = function () { }
}
```

3.4 原型继承

2. 继承-让男人和女人都能继承人类的一些属性和方法

- 把男人女人公共的属性和方法抽取出来 People
- 然后赋值给Man的原型对象，可以共享这些属性和方法
- 注意让constructor指回Man这个构造函数



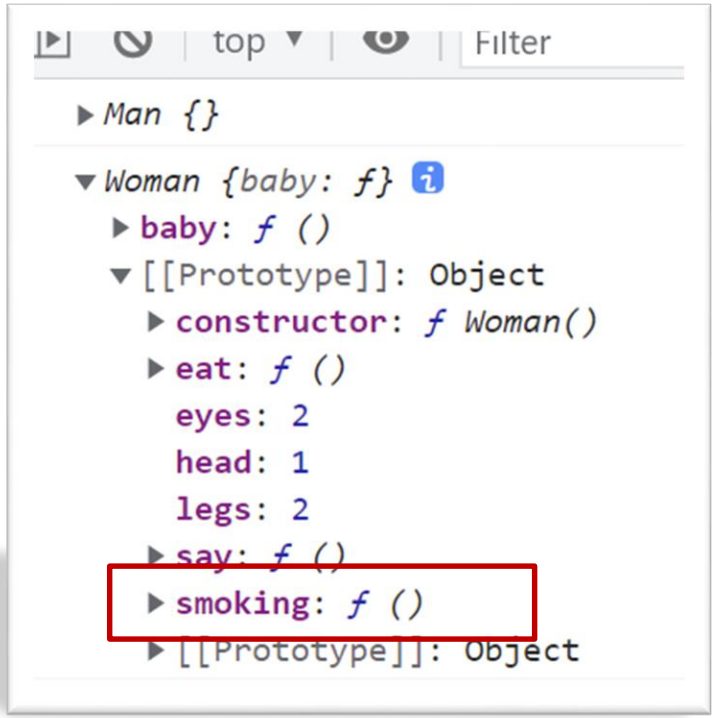

```
// 人类 公共的属性和方法
const People = {
  head: 1,
  eyes: 2,
  legs: 2,
  say: function () { },
  eat: function () { }
}
// 男人
function Man() {
}
// 把公共的属性和方法给原型，这样就可以共享了
Man.prototype = People
// 注意让原型里面的constructor从新指回Man找自己的爸爸
Man.prototype.constructor = Man
const pink = new Man()
console.log(pink)
```

3.4 原型继承

3. 问题：

如果我们给男人添加了一个吸烟的方法，发现女人自动也添加这个方法

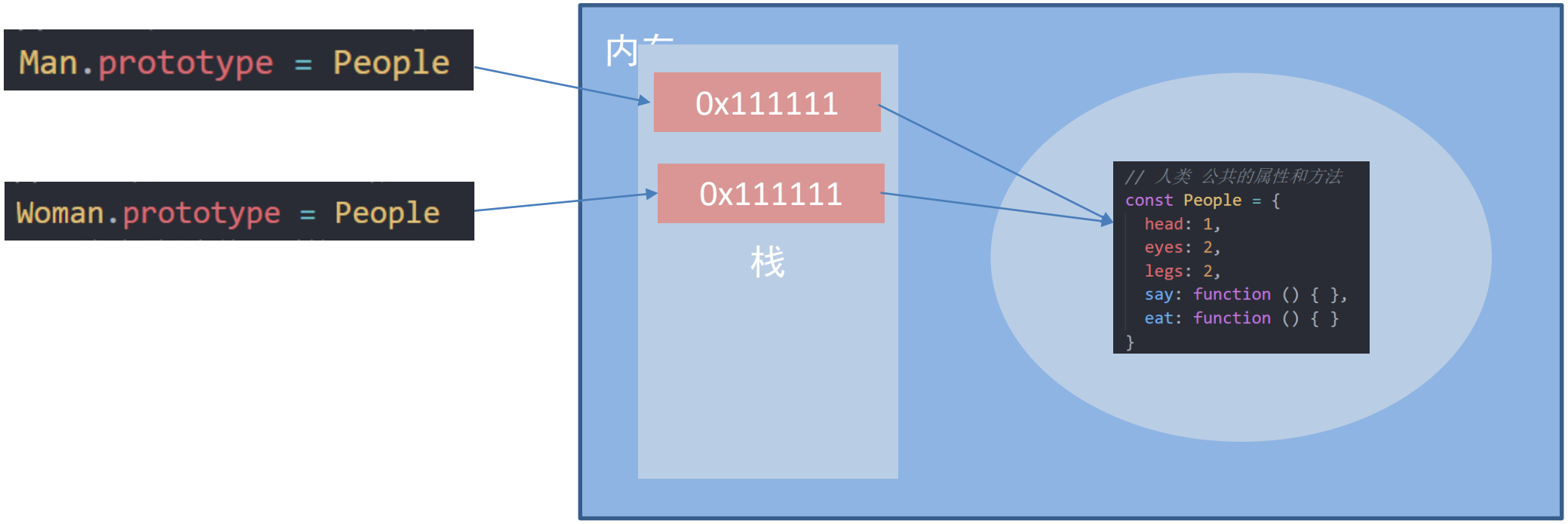
```
// 把公共的属性和方法给原型，这样就可以共享了
Man.prototype = People
// 注意让原型里面的constructor从新指回Man找自己的爸爸
Man.prototype.constructor = Man
const pink = new Man()
Man.prototype.smoking = function () { }
console.log(pink)
// 女人
function Woman() {
  this.baby = function () { }
}
// 把公共的属性和方法给原型，这样就可以共享了
Woman.prototype = People
// 注意让原型里面的constructor从新指回Man找自己的爸爸
Woman.prototype.constructor = Woman
const red = new Woman()
console.log(red)
```



3.4 原型继承

3. 问题：--原因

男人和女人都同时使用了同一个对象，根据引用类型的特点，他们指向同一个对象，修改一个就会都影响



3.4 原型继承

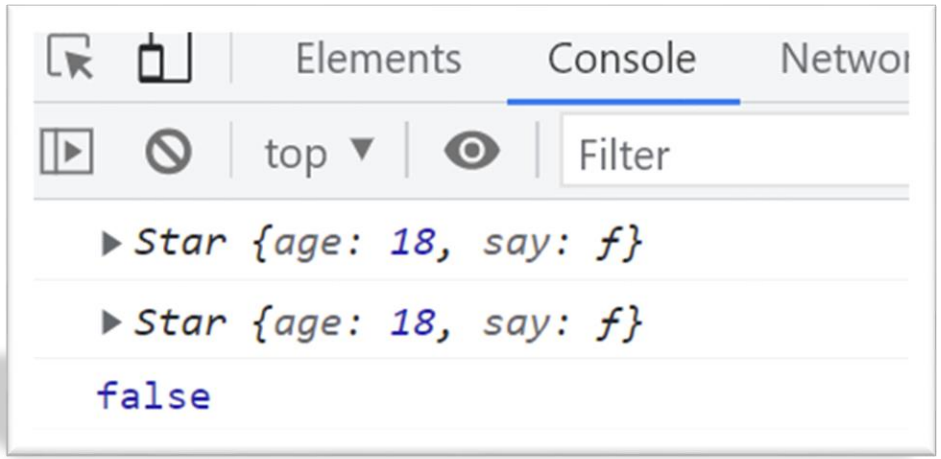
4. 解决：

需求：男人和女人不要使用同一个对象，但是不同对象里面包含相同的属性和方法

答案：构造函数

new 每次都会创建一个新的对象

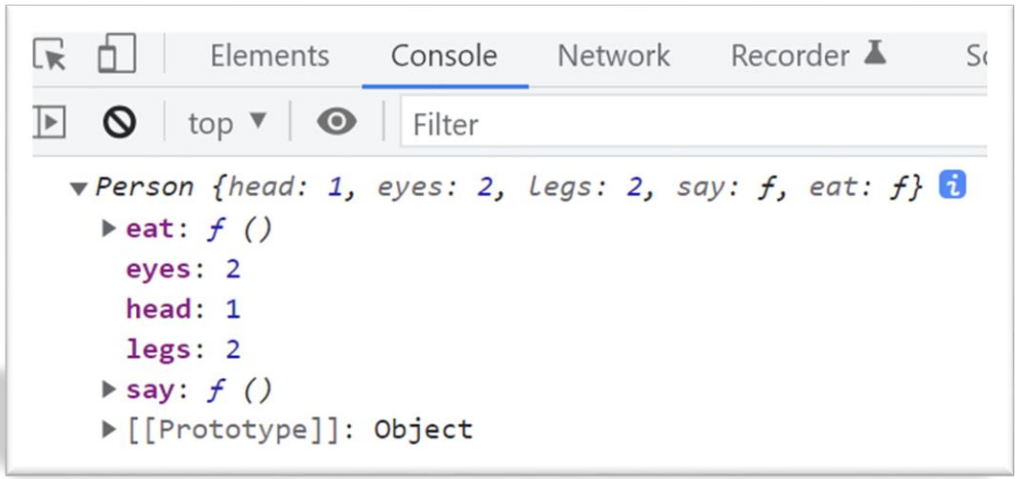
```
function Star() {
  this.age = 18
  this.say = function () { }
}
const ldh = new Star()
const zxy = new Star()
console.log(ldh)
console.log(zxy)
console.log(ldh === zxy)  // false 每个实例对象都不一样
```



3.4 原型继承

5. 继承写法完善

```
// 人类 公共的属性和方法
function Person() {
  this.head = 1
  this.eyes = 2
  this.legs = 2
  this.say = function () { }
  this.eat = function () { }
}
console.log(new Person()) // 就是输出刚才的对象
```




```
const People = {
  head: 1,
  eyes: 2,
  legs: 2,
  say: function () { },
  eat: function () { }
}
```

3.4 原型继承

5. 继承写法完善

```
function Man() {
}
// 用 new Person() 替换刚才的固定对象
Man.prototype = new Person()
// 注意让原型里面的constructor从新指回Man找自己的爸爸
Man.prototype.constructor = Man
const pink = new Man()
Man.prototype.smoking = function () { }
console.log(pink)
// 女人
function Woman() {
  this.baby = function () { }
}
// 用 new Person() 替换刚才的固定对象
Woman.prototype = new Person()
// 注意让原型里面的constructor从新指回Man找自己的爸爸
Woman.prototype.constructor = Woman
const red = new Woman()
console.log(red)
```

```
▼Man {} i
  ▼[[Prototype]]: Person
    ▶constructor: ƒ Man()
    ▶eat: ƒ ()
      eyes: 2
      head: 1
      legs: 2
    ▶say: ƒ ()
    ▶smoking: ƒ ()
    ▶[[Prototype]]: Object
▼Woman {baby: ƒ} i
  ▶baby: ƒ ()
  ▼[[Prototype]]: Person
    ▶constructor: ƒ Woman()
    ▶eat: ƒ ()
      eyes: 2
      head: 1
      legs: 2
    ▶say: ƒ ()
    ▶[[Prototype]]: Object
```

此处Woman么有smoking方法了

思路

真正做这个案例，我们的思路应该是先考虑大的，后考虑小的

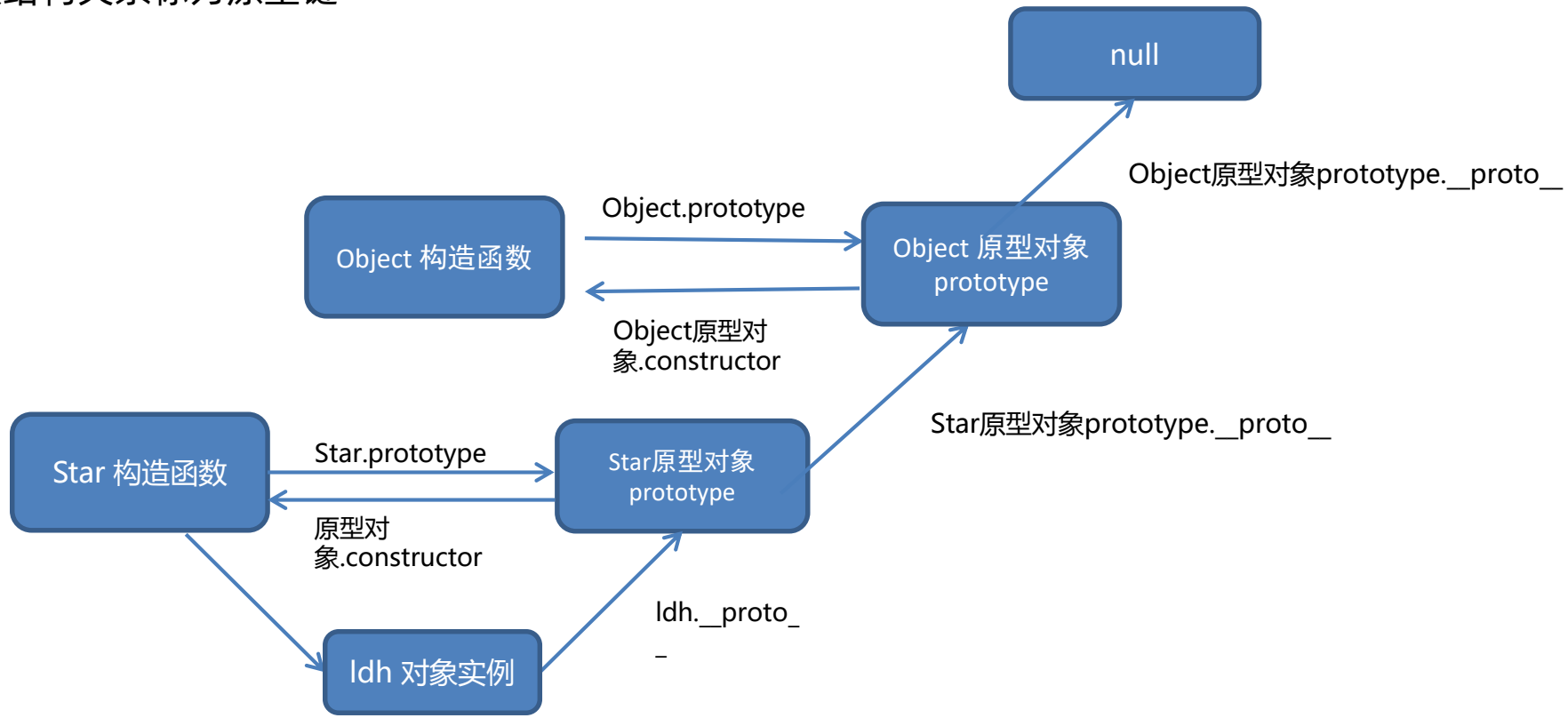
1. 人类共有的属性和方法有那些，然后做个构造函数，进行封装，一般公共属性写到构造函数内部，公共方法，挂载到构造函数原型身上。
2. 男人继承人类的属性和方法，之后创建自己独有的属性和方法
3. 女人同理

原型

- 原型
- constructor 属性
- 对象原型
- 原型继承
- 原型链

3.5 原型链

基于原型对象的继承使得不同构造函数的原型对象关联在一起，并且这种关联的关系是一种链状的结构，我们将原型对象的链状结构关系称为原型链



3.5 原型链-查找规则

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

⑥ 可以使用 instanceof 运算符用于检测构造函数的 prototype 属性是否出现在某个实例对象的原型链上

目录

Contents

案例

消息提示对象封装

目的： 练习面向对象写插件（模态框）

需求：

案例

消息提示对象封装

分析需求：

1. 定义模态框 Modal 构造函数，用来创建对象
2. 模态框具备 打开功能 open 方法 （按钮点击可以打开模态框）
3. 模态框 具备关闭功能 close 方法

问：

open 和 close 方法 写到哪里？

构造函数的原型对象上，共享方法

所以可以分为三个模块， 构造函数， open方法， close方法

步骤

• 消息提示对象封装

步骤：

①：Modal 构造函数 制作

- 需要的公共属性： 标题（title）、提示信息内容（message） 可以设置默认参数
- 在页面中创建模态框

(1) 创建div标签可以命名为：modalBox

(2) div标签的类名为 modal

(3) 标签内部添加 基本结构，并填入相关数据

```
<div class="modal">
  <div class="header">温馨提示 <i>x</i></div>
  <div class="body">您没有删除权限操作</div>
</div>
```

步骤

- **消息提示对象封装**

步骤：

①：open方法

- 写到构造函数的原型对象身上
- 把刚才创建的modalBox 添加到 页面 body 标签中
- open 打开的本质就是 把创建标签添加到页面中
- 点击按钮， 实例化对象，传入对应的参数，并执行 open 方法

步骤

- **消息提示对象封装**

步骤：

①：close方法

- 写到构造函数的原型对象身上
- 把刚才创建的modalBox 从页面 body 标签中 删除
- 需要注意，x 删除按钮绑定事件，要写到open里面添加

  因为open是往页面中添加元素，同时顺便绑定事件

今日作业

1. 整理笔记
2. 画图的方式画出原型链
3. 开始做测试题： PC端地址： https://ks.wjx.top/vj/rTd1xoS.aspx
4. 预习第四天内容

传智教育旗下高端IT教育品牌