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

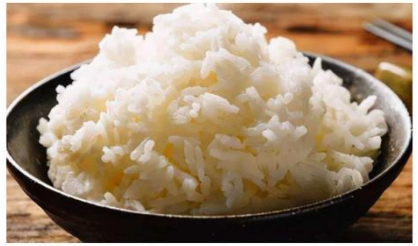
Contents

编程思想

- 面向过程介绍
- 面向对象介绍

1.1 面向过程编程

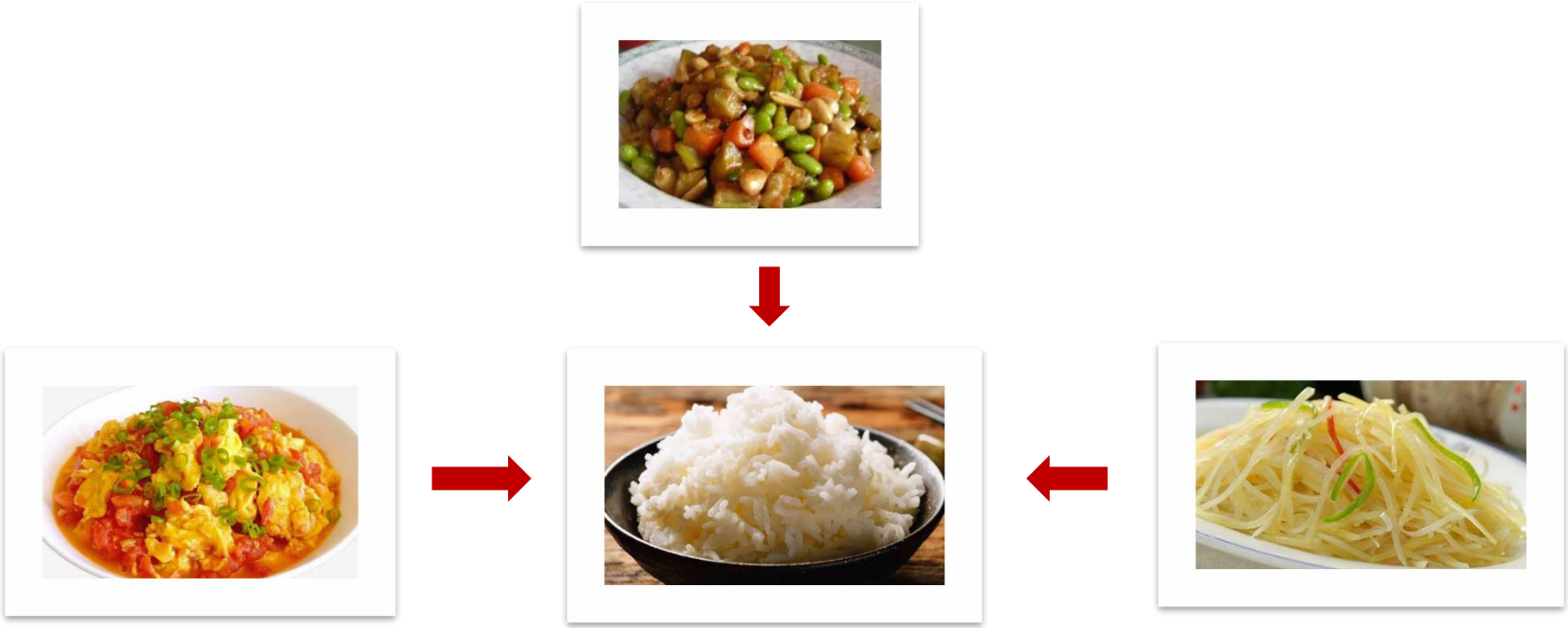
- 面向过程就是分析出解决问题所需要的步骤，然后用函数把这些步骤一步一步实现，使用的时候再一个一个的依次调用就可以了。
- **举个栗子：蛋炒饭**



- 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。

1.2 面向对象编程 (oop)

- 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)

- 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。

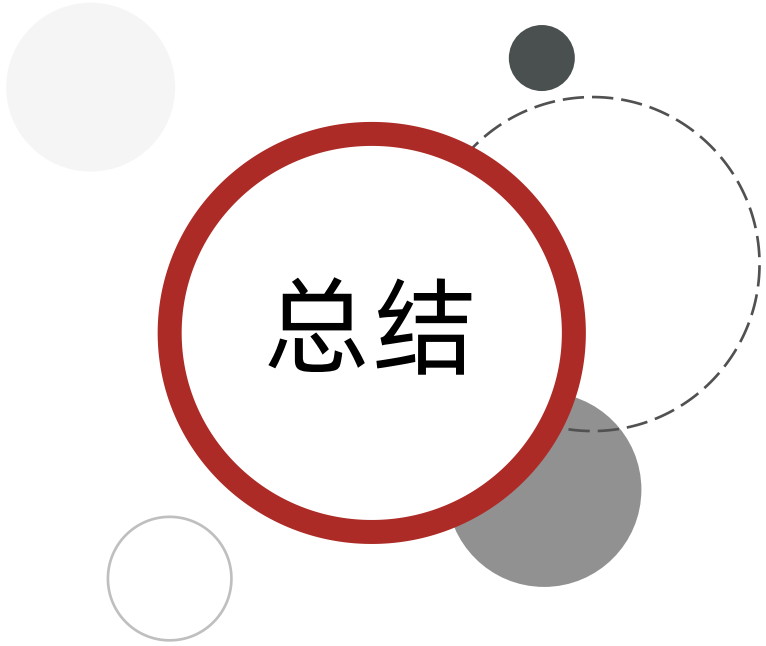
缺点：不灵活、复用性较差

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



1. 什么是面向过程编程？
   - 是按照我们分析好了的步骤，按照步骤一步一步的解决问题
2. 什么是面向对象编程？
   - 是以对象功能来划分为一个个对象，然后由对象之间分工与合作
3. 两种编程思想的特点？
   - 面向过程性能高，但是不灵活，复用性低
   - 面向对象灵活、易扩展、易维护，但是性能比面向过程低

目录

Contents

02 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
- 把公共的属性和方法抽取封装到构造函数里面来实现数据的共享，这样创建的实例对象可以使用这些属性和方法了

```
// 构造函数实现封装
function Person(name, age) {
  this.name = name
  this.age = age
  this.sayHi = function () {
    console.log('hi~')
  }
}
// 实例对象使用属性和方法
const zs = new Person('张三', 18)
const ls = new Person('李四', 19)
```

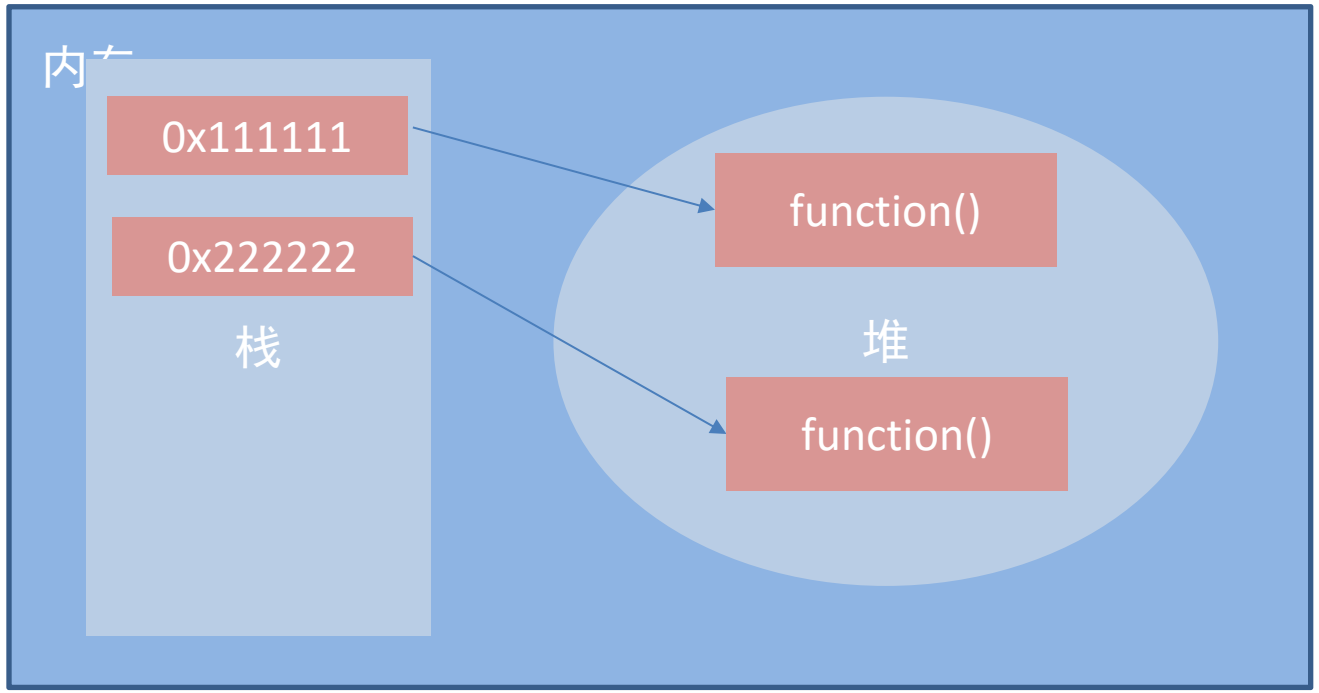
总结：

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

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

```
// 构造函数实现封装
function Person(name, age) {
  this.name = name
  this.age = age
  this.sayHi = function () {
    console.log('hi~')
  }
}
// 实例对象使用属性和方法
const zs = new Person('张三', 18)
const ls = new Person('李四', 19)
```

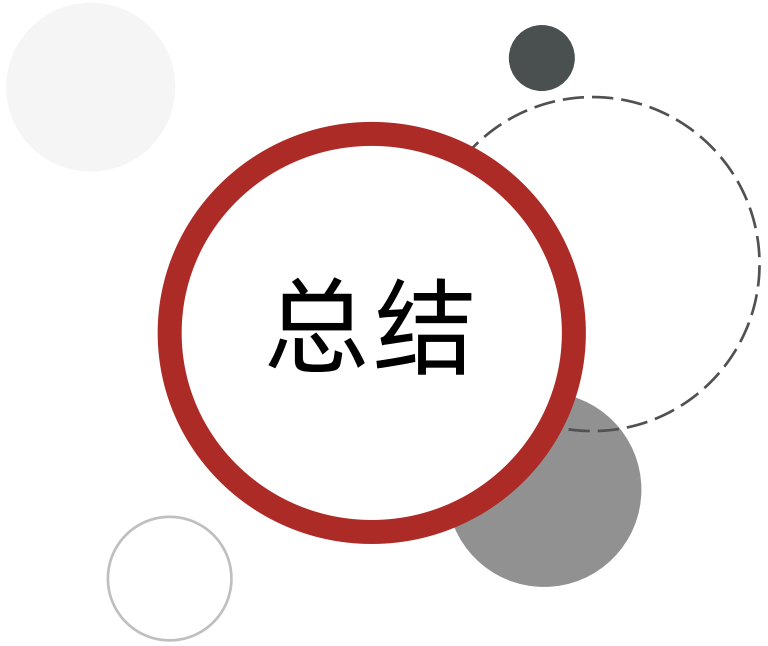


2. 构造函数

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

```
// 构造函数实现封装
function Person(name, age) {
 this.name = name
 this.age = age
 this.sayHi = function () {
  console.log('hi~')
 }
}
// 实例对象使用属性和方法
const zs = new Person('张三', 18)
const ls = new Person('李四', 19)
```

```
// 构造函数实现封装
function Person(name, age) {
  this.name = name
  this.age = age
  this.sayHi = function () {
    console.log('hi~')
  }
}
// 实例对象使用属性和方法
const zs = new Person('张三', 18)
const ls = new Person('李四', 19)
console.log(zs.sayHi === ls.sayHi)  // false 说明函数不一样
```



1. Js 实现面向对象中的封装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
   - 构造函数中函数会多次创建，占用内存

```
// 构造函数实现封装
function Person(name, age) {
  this.name = name
  this.age = age
  this.sayHi = function () {
    console.log('hi~')
  }
}
// 实例对象使用属性和方法
const zs = new Person('张三', 18)
const ls = new Person('李四', 19)
console.log(zs.sayHi === ls.sayHi)  // false 说明函数不一样
```

目录

Contents

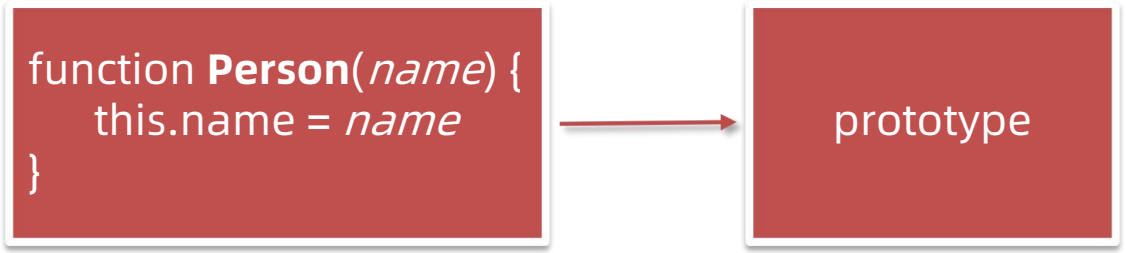
03

原型

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

3.1 原型对象

是什么？ 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



构造函数　　原型对象

```
function Person(name, age) {
  this.name = name
  this.age = age
}
Person.prototype.sayHi = function () {
  console.log('hi~')
}
console.log(zs.sayHi === ls.sayHi)  // true 两个函数一样
```

使用场景：

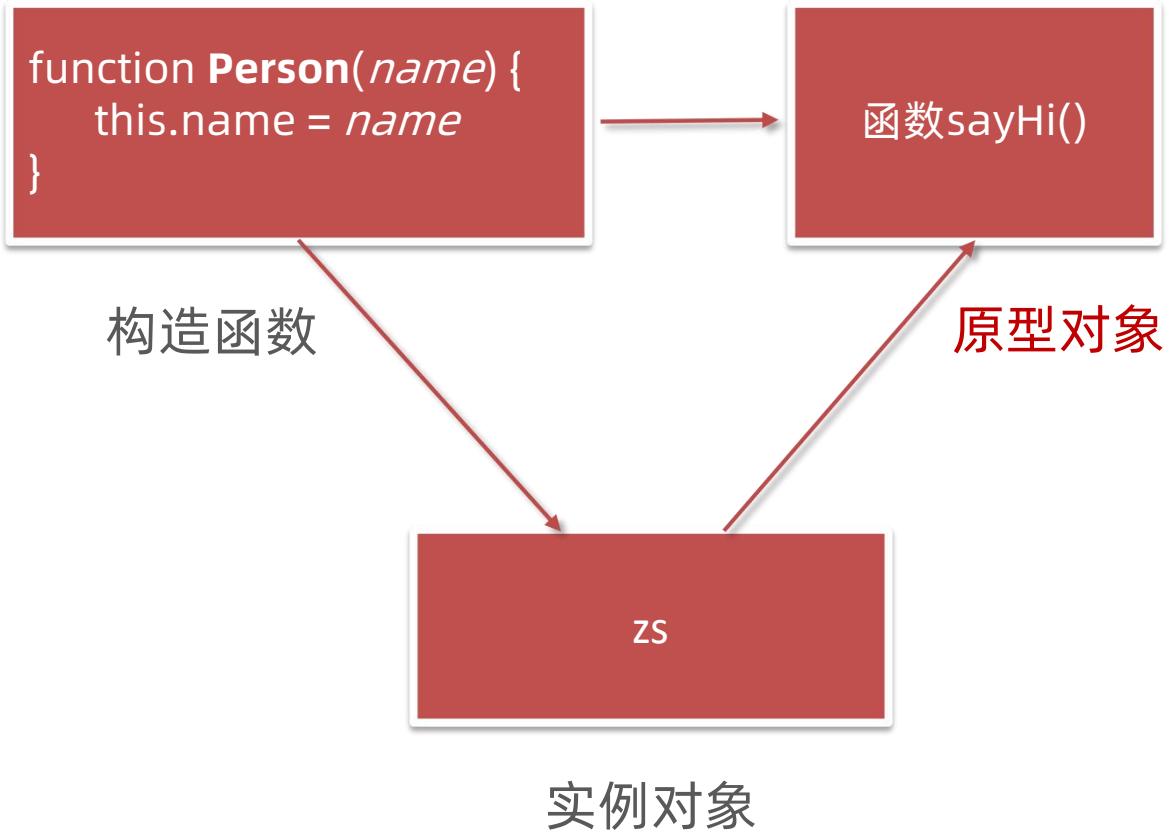
可以解决：构造函数封装时函数（方法）会多次创建，占用内存的问题

原型对象可以挂载函数，对象实例化不会多次创建原型对象里面的函数，节约内存

3.1 原型对象

实例对象可直接访问原型对象中函数

执行过程：先找实例对象属性或函数，找不到会再找原型对象中属性或函数




```
Array.prototype.reverse()
Array.prototype.shift()
Array.prototype.slice()
Array.prototype.some()
Array.prototype.sort()
Array.prototype.splice()
```

```
String.prototype.split()
String.prototype.startsWith()
String.prototype.strike()
String.prototype.sub()
String.prototype.substr()
String.prototype.substring()
```

总结

1. 原型对象是什么？
   - prototype，每个构造函数都有原型对象
2. 原型对象的作用是什么？
   - 可以把那些公共的属性和方法，直接定义在 prototype 对象上
   - 实例对象可直接访问原型对象中属性和方法
   - 这些属性和方法不会多次内存中创建，从而节约内存

3.1 原型对象- 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

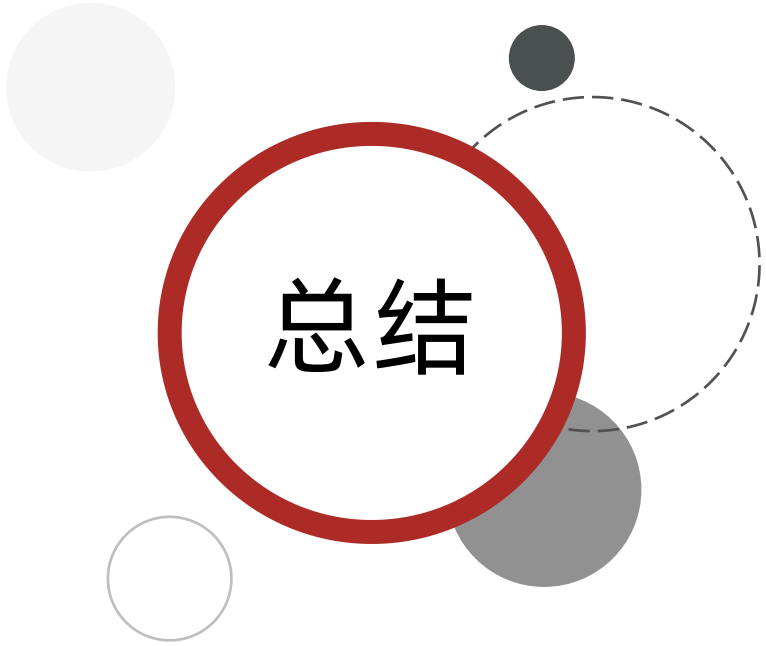
```
// 1. 构造函数this指向 实例对象
function Person(name, age) {
  this.name = name
}

// 2. 原型对象this指向 实例对象
Person.prototype.sayHi = function () {
  console.log('hi~')
  console.log(this)  // 指向实例对象 zs
}

const zs = new Person('张三')
zs.sayHi()
```

注意事项

- 箭头函数不能做构造函数，因为箭头函数里面没有 this
- 原型对象里面的函数如果需要用到this，也不要用箭头函数



1. 构造函数和原型对象里面的this指向谁？
   - 实例化的对象
2. 构造函数能用箭头函数吗？
   - 不能，因为箭头函数没有this

案例

给数组扩展方法

需求：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]

需求扩展完毕之后：

arr.max() 返回结果是 3

arr.sum()　返回的结果是 6

步骤

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

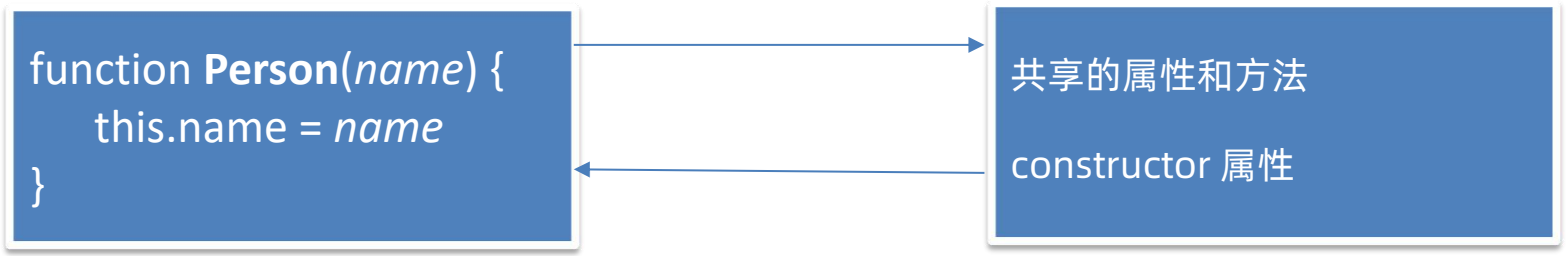
原型

- 原型对象
- constructor 属性
- 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

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。

总结

1. constructor属性的在哪里呀？
   - 原型对象里面
2. constructor属性的作用是什么？
   - 指向该原型对象的构造函数

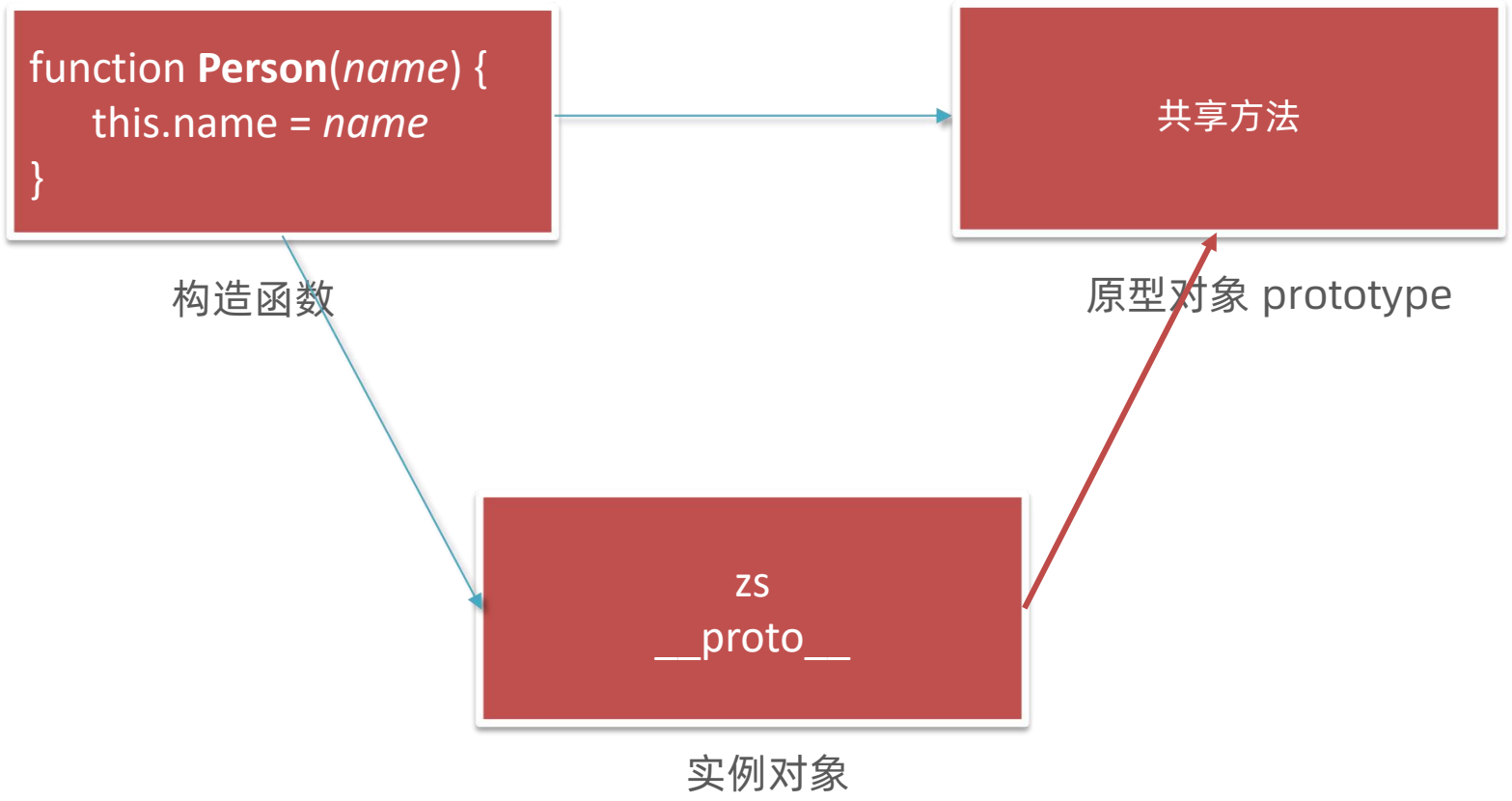
原型

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

3.3 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 原型对象的方法，就是因为对象有 __proto__ 原型的存在

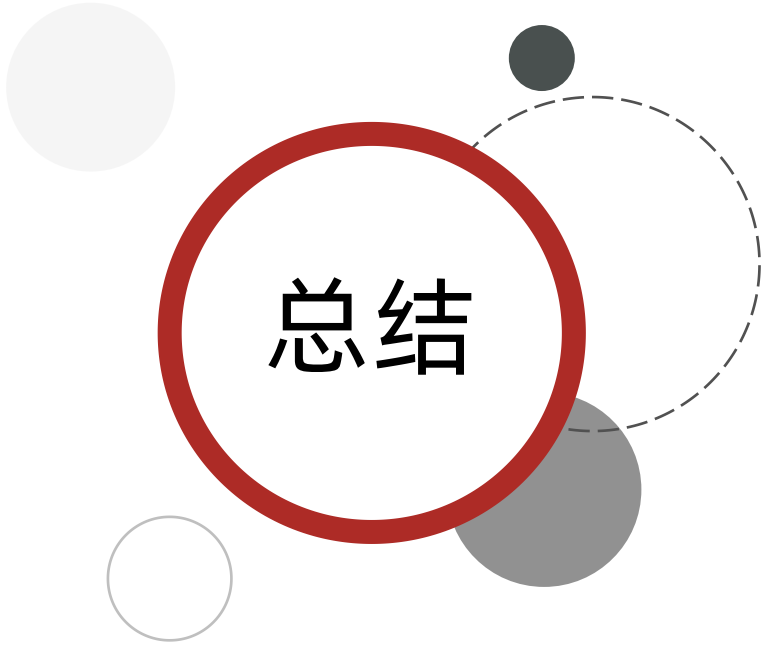


3.3 原型

注意：

- __proto__ 原先是 JS非标准属性，但是 es6 规范中开始标准化， [[prototype]]和__proto__意义相同

- 尽量不要修改这个属性，对性能影响非常严重的
- 约定： prototype 原型对象 而 __proto__ 原型



1. prototype是什么？哪里来的？
   - 原型对象
   - 构造函数都自动有原型对象
2. __proto__属性是什么？在哪里？指向谁？
   - 原型
   - 在实例对象里面
   - 指向 prototype 原型对象，这样实例对象就可以访问原型对象里面的方法了

练习

根据下面代码，请画图（课堂练习）

需求：

①：利用画图工具，画出 构造函数 原型对象 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')
```

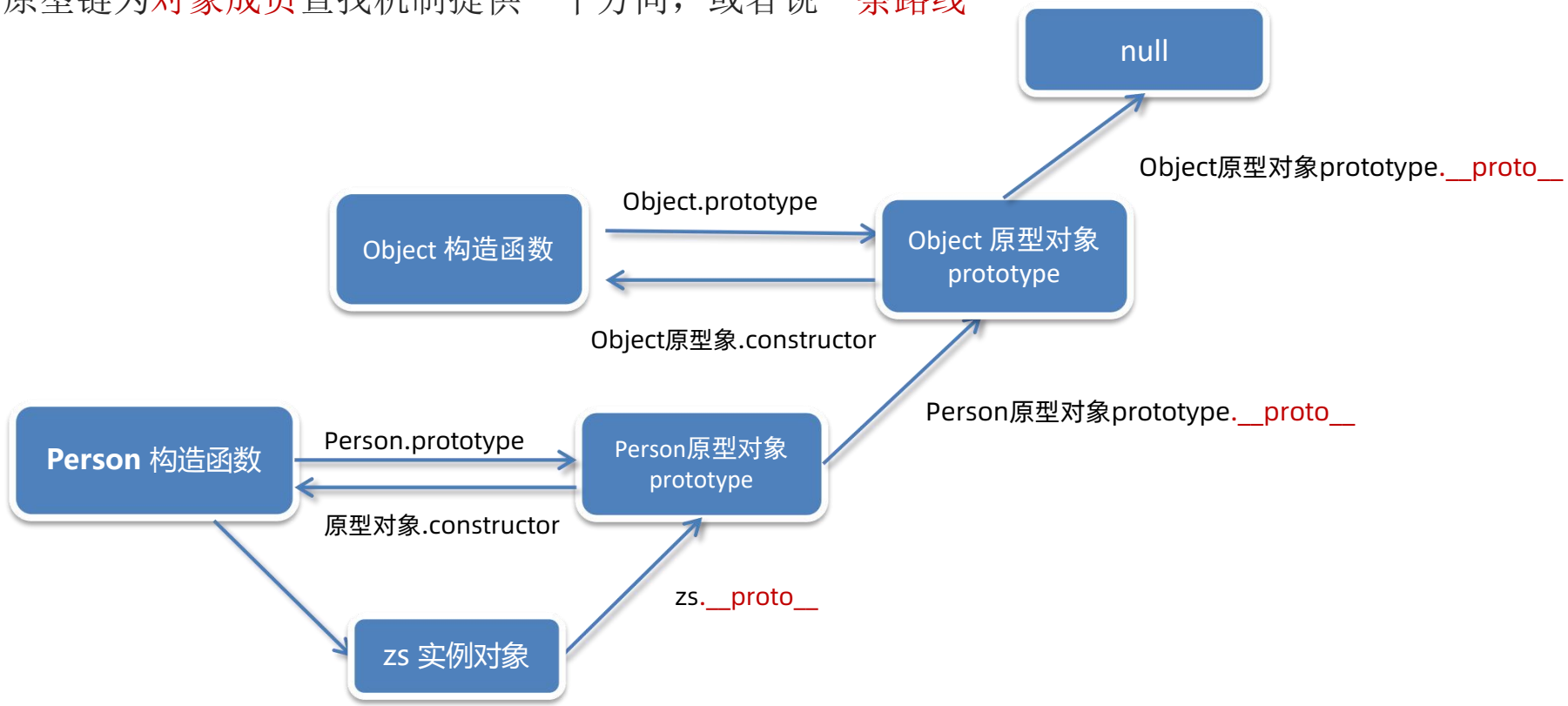
原型

- 原型
- constructor 属性
- 原型
- 原型链
- 原型继承

3.5 原型链

__proto__属性链状结构称为原型链

作用：原型链为对象成员查找机制提供一个方向，或者说一条路线



3.5 原型链-查找规则

① 当访问一个对象成员(属性/方法)时，首先查找这个对象自身有没有该成员(属性/方法)

② 如果没有就查找它的原型对象（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对象（Object的原型对象）

④ 依此类推一直找到 Object 为止（null）

⑤ 原型链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

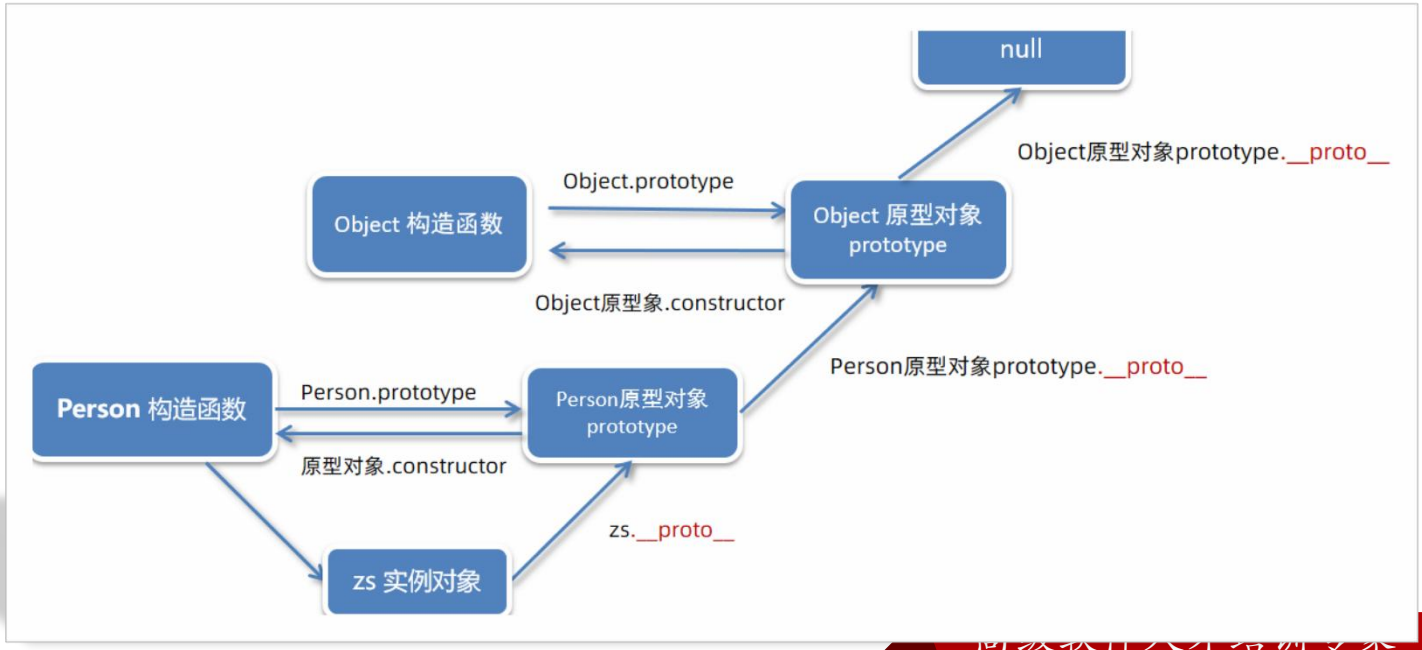
3.5 原型链- instanceof 运算符

语法：

实例对象 instanceof 构造函数

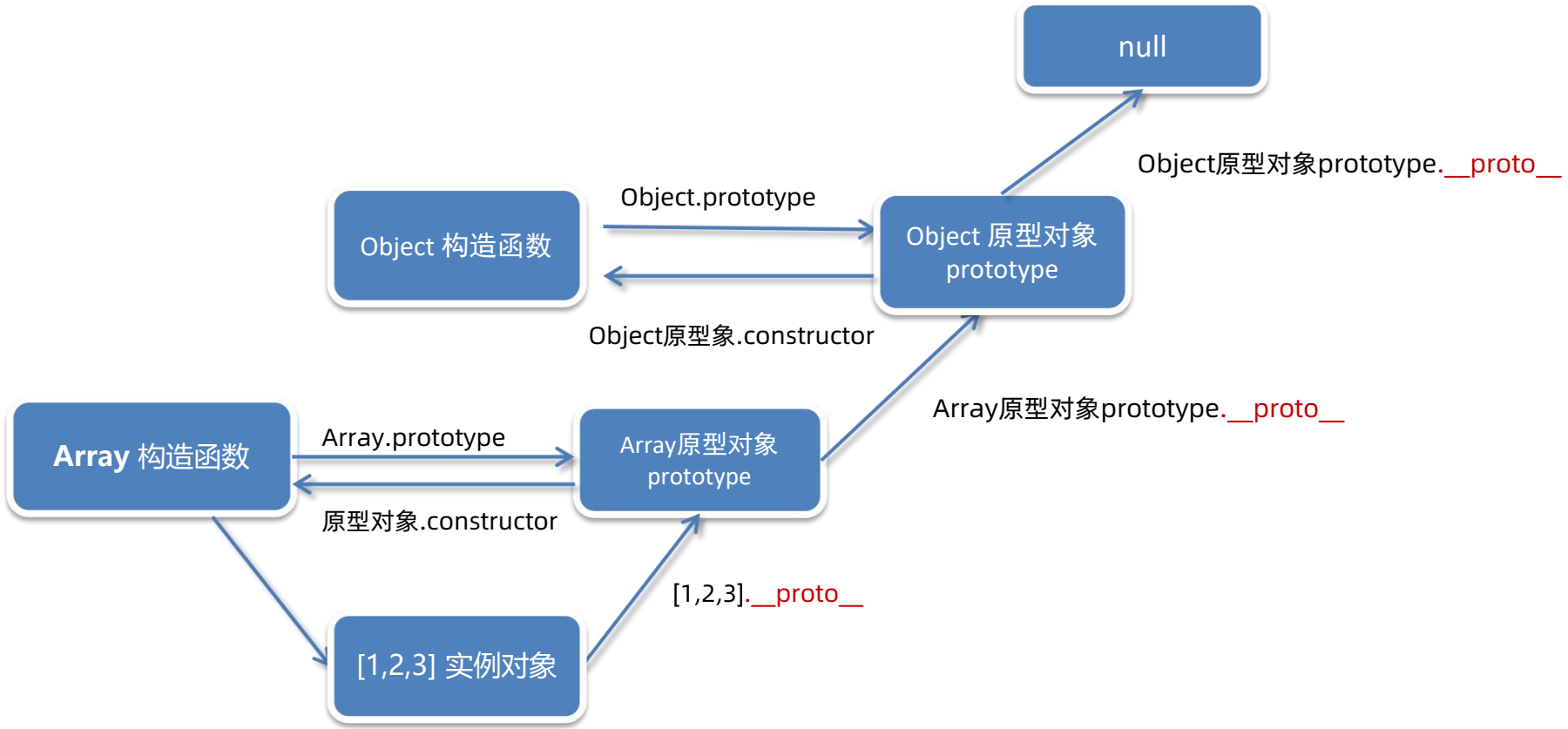
作用：

用来检测构造函数.prototype 是否存在于实例对象的原型链上



3.5 原型链- instanceof 运算符

实例对象 instanceof 构造函数



03

原型

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

3.5 原型继承

继承是面向对象编程的另一个特征。龙生龙、凤生凤、老鼠的儿子会打洞描述的正是继承的含义

有些公共的属性和方法可以写到父级身上，子级通过继承也可以使用这些属性和方法

JavaScript 中大多是借助原型对象实现继承的特性

我们来看个代码：

```
function Man() {
  this.eyes = 2
  this.eat = function () {
    console.log('我会吃饭')
  }
}
```

```
function Woman() {
  this.eyes = 2
  this.eat = function () {
    console.log('我会吃饭')
  }
}
```

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person对象里面

```
const person = {
  eyes: 2,
  eat() {
    console.log('我会吃饭')
  }
}
function Man() {
}
function Woman() {
}
```

2. 继承- 通过原型对象实现继承

把男人和女人原型对象赋值为 person对象

```
Man.prototype = person
Woman.prototype = person
```

3.4 原型继承

3. 问题：

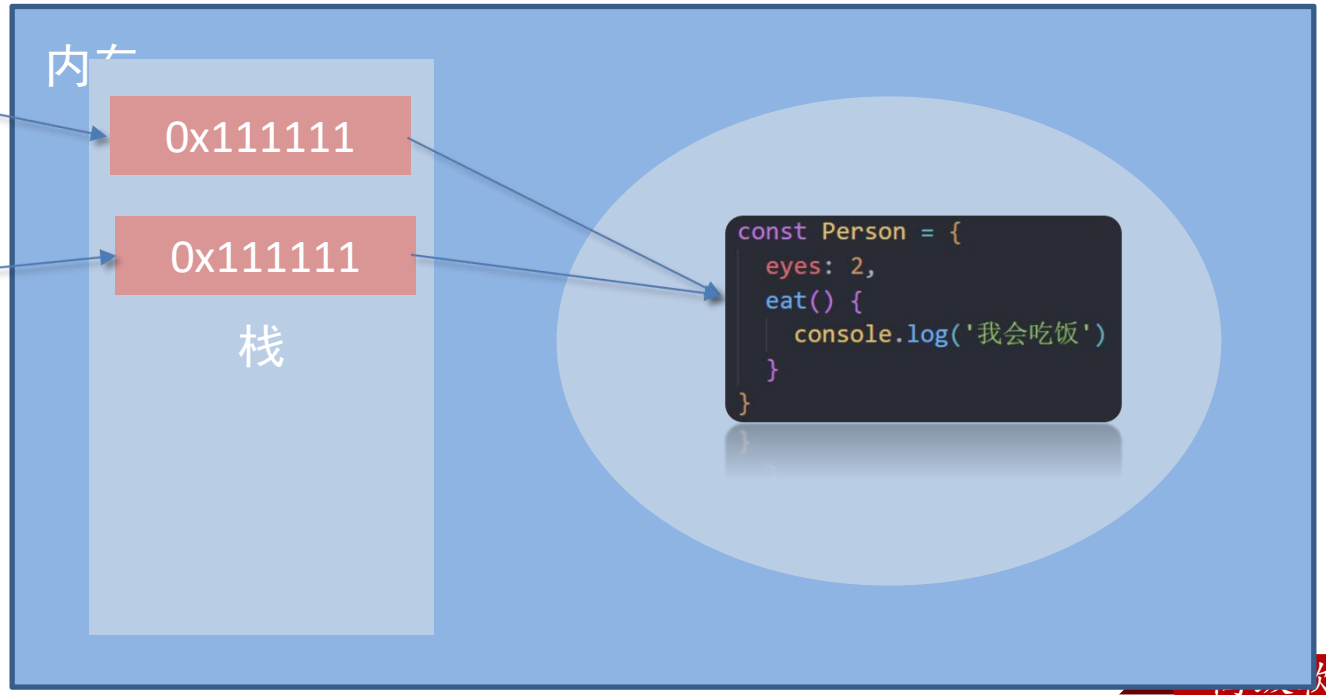
如果给女人添加一个baby方法，男人也会自动增加

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（对象不独立）

```
Man.prototype = person
```

```
Woman.prototype = person
```



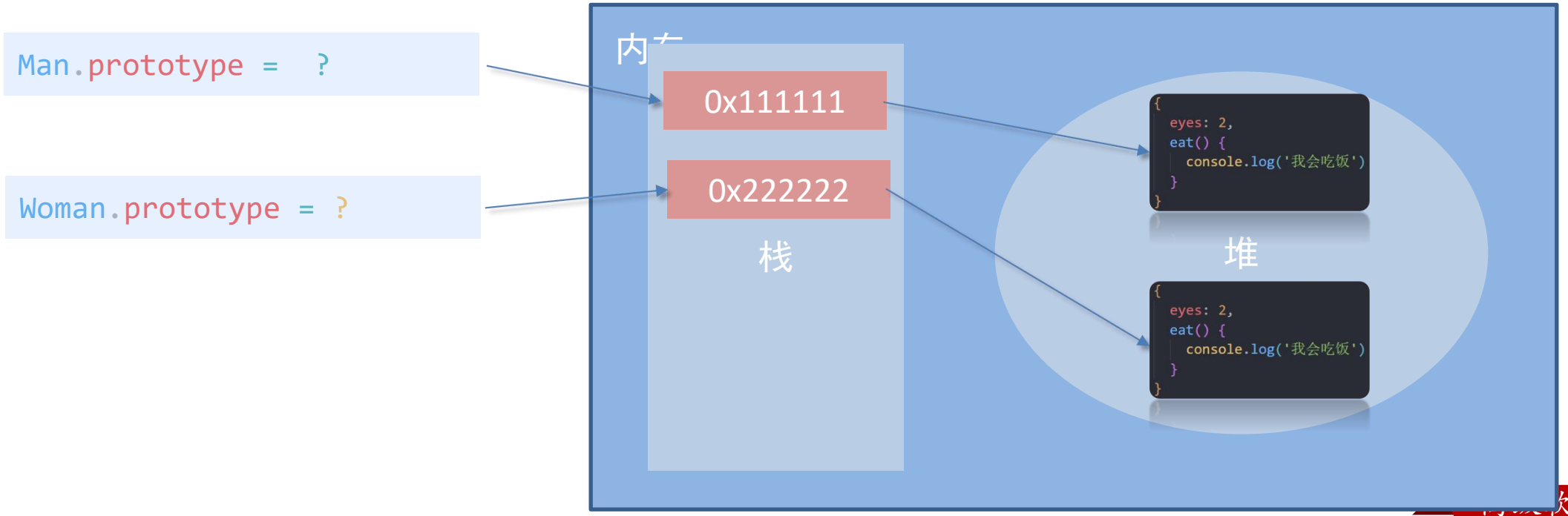
总结

1. JavaScript实现继承借助于谁呢？
   - 借助原型对象实现继承的特性
2. 用字面量对象继承有什么缺点？
   - 使用同一个对象，修改任何一个都会影响其他
   - 对象不独立的问题

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



3.4 原型继承

4. 解决：

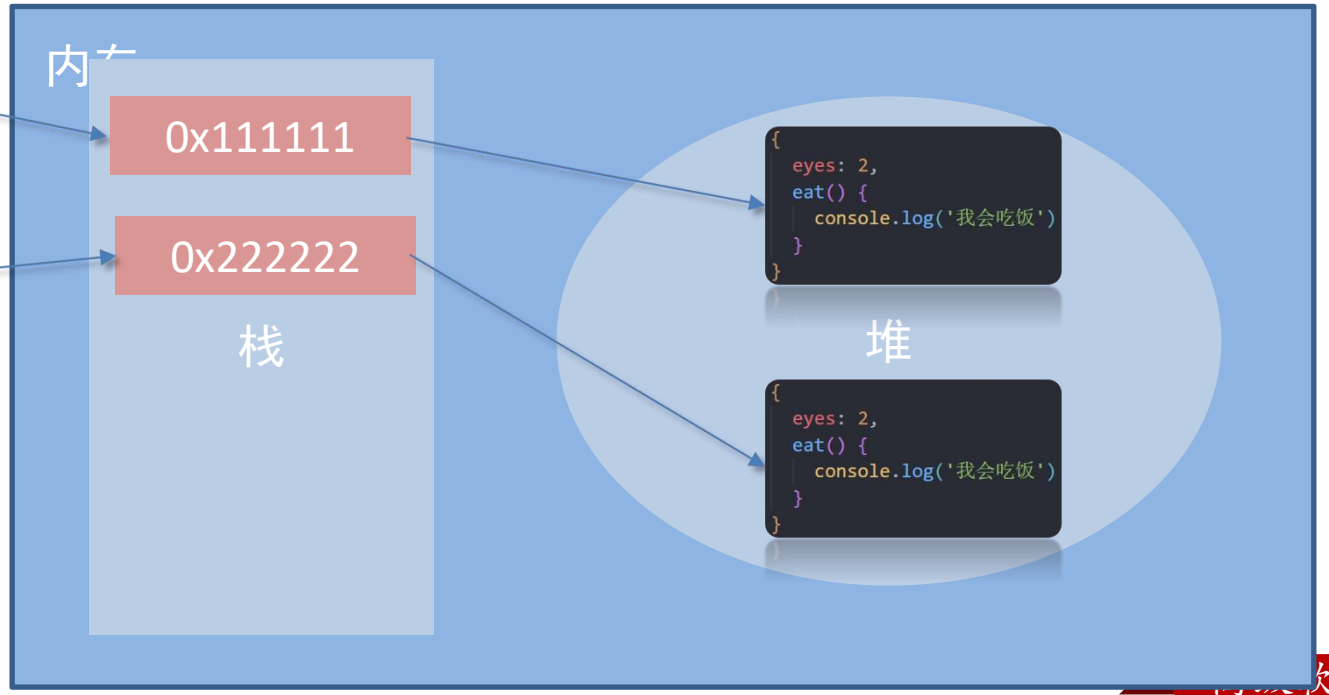
需求：男人和女人不要使用同一个对象，但是不同对象里面包含相同的属性和方法

答案：构造函数实例化对象继承，因为 new 每次都会创建一个新的对象

```
function Person() {
  this.eyes = 2
}
Person.prototype.eat = function () {
  console.log('我会吃饭')
}
```

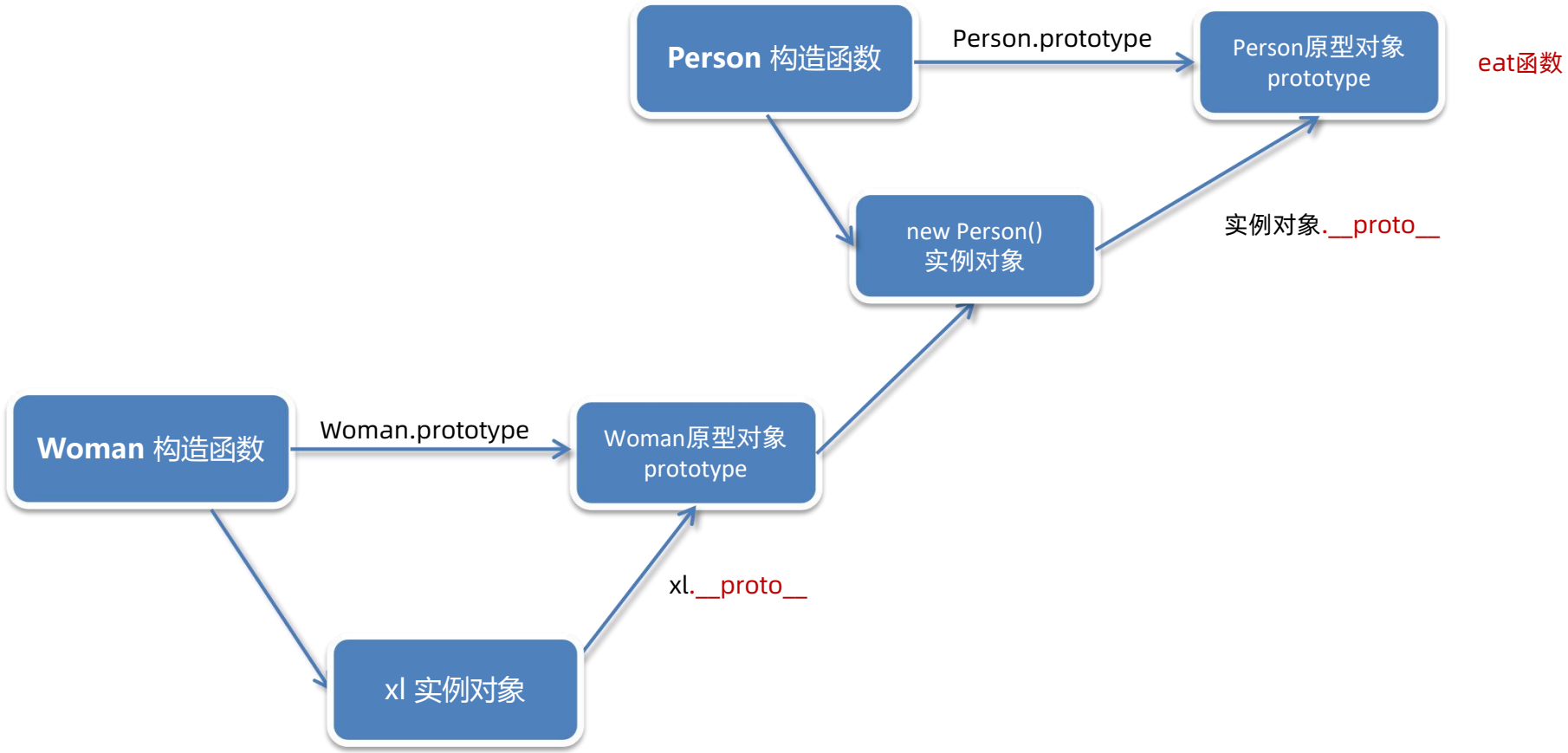
```
Man.prototype = new Person()
```

```
Man.prototype = new Person()
```



3.4 原型继承

5. 完整过程



思路

真正做这个案例，我们的思路应该是先考虑父级，后考虑子级的

1. 父级人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对象身上
2. 子级
   - 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
   - 子级女人同理

注意： 因为对象覆盖了原型对象，所以在把constructor指回当前构造函数

```
Man.prototype = new Person()
Man.prototype.constructor = Man
```

```
function Person() {
  this.eyes = 2
}
Person.prototype.eat = function () {
  console.log('我会吃饭')
}
```

```
// 2. 继承-借助于原型对象
Man.prototype = new Person()
const zs = new Man()
// console.log(zs)
```

目录

Contents

案例

模态框封装

目的： 练习面向对象写插件（模态框）

需求：

案例

模态框封装

分析需求：

1. 多个模态框一样的，而且每次点击都会出来一个，怎么做呢？

- 构造函数。把模态框封装一个构造函数 Modal，每次new 都会产出一个模态框，所以点击不同的按钮就是在做 new 模态框，实例化。

2. 模态框有什么功能呢？ 打开功能（显示），关闭功能，而且每个模态框都包含着2个功能

- open功能
- close功能

问：

open 和 close 方法 写到哪里？

构造函数Modal的原型对象上，共享方法

案例 **模态框封装**

- 模态框Modal业务
- 打开方法open
- 关闭方法close

步骤

模态框封装

模态框Modal业务

业务①：Modal 构造函数 制作

- 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

- 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


```
<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>
```

案例 **模态框封装**

- 模态框Modal业务
- 打开方法open
- 关闭方法close

步骤 **模态框封装**

打开方法open

业务②：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

案例 **模态框封装**

- 模态框Modal业务
- 打开方法open
- 关闭方法close

步骤 **模态框封装**

关闭方法close

业务③：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 关闭按钮是在模态框里面，

  所以应该是页面显示这个模态框就要绑定事件

  页面显示模态框是在 open里面，所以绑定关闭事件也写到 open方法里面

步骤 **模态框封装**

小bug处理

BUG：

多次点击会显示很多的模态框

解决：

准备open显示的时候，先判断页面中有没有 modal盒子，有就移除，没有就添加

传智教育旗下高端IT教育品牌