JavaScript进阶

---JS原型继承

---深入理解JS的继承方式

河北师范大学软件学院

Software College of Hebei Normal University

JavaScript进阶

---JS原型继承

河北师范大学软件学院
Software College of Hebei Normal University

内容提纲

- **JS对象及继承方式综述**
- **JS对象的原型链**
- **基于构造函数实现的原型继承**

JS对象及继承方式综述

- JS对象知识回顾
  - JS对象是若干无序属性的集合（数据属性、访问器属性、内部属性）
  - 生成对象的3种方式：字面量直接生成、Object工场方法、构造函数实例化对象

```
var obj = {
    num:10,
    str:"Hi",
    show:function(){
        console.log(this.str);
    }
}

var subObj = Object.create(obj);
subObj.age = 23;
```

```
function Person(age,name){
    this.age = age;
    this.name = name;
}
Person.prototype.sayHi = function(){
    console.log("Hi,I'm "+this.name);
}
var p1 = new Person(20,"Jame");
p1.sayHi();
```


参见实例demo01 生成对象及对象原型链

JS对象及继承方式综述

- JavaScript语言继承方式
  - JavaScript采用的是原型的继承方式，每个对象都有一个原型对象，最原始的原型是null
  - JavaScript的继承是对象-对象的原型继承，为面向对象提供了动态继承的功能
  - 任何方式创建的对象都有原型对象，可以通过对象的 __proto__ 属性来访问原型对象

```
var obj = { //obj的原型是Object.prototype
    num:10,
    str:"Hi",
    show:function(){
        return this.str;
    }
};
var newObj = Object.create(obj);
newObj.age = 23;
console.log(newObj.__proto__===obj);//true
```

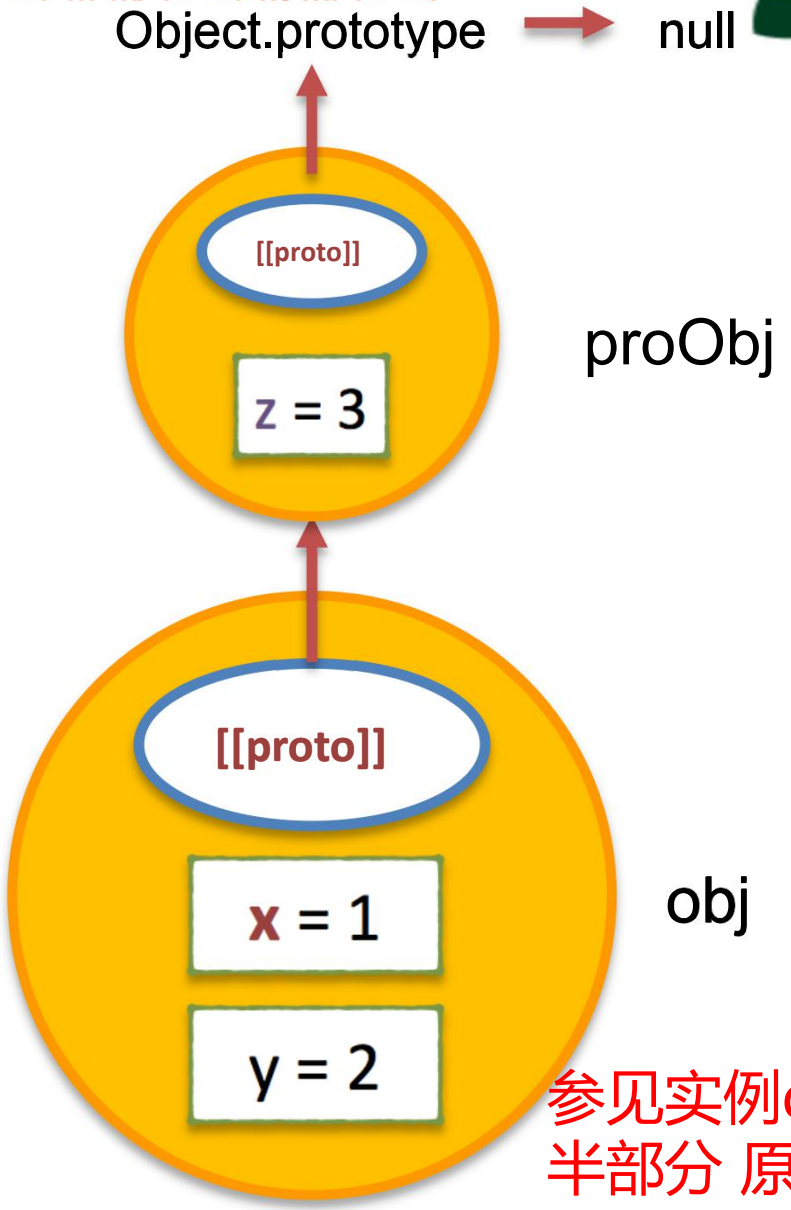

参见实例demo02 __proto__和原型对象

内容提纲

- JS对象及继承方式综述
- JS对象的原型链
- 基于构造函数实现的原型继承

JS对象的属性访问链（自有属性和继承属性）

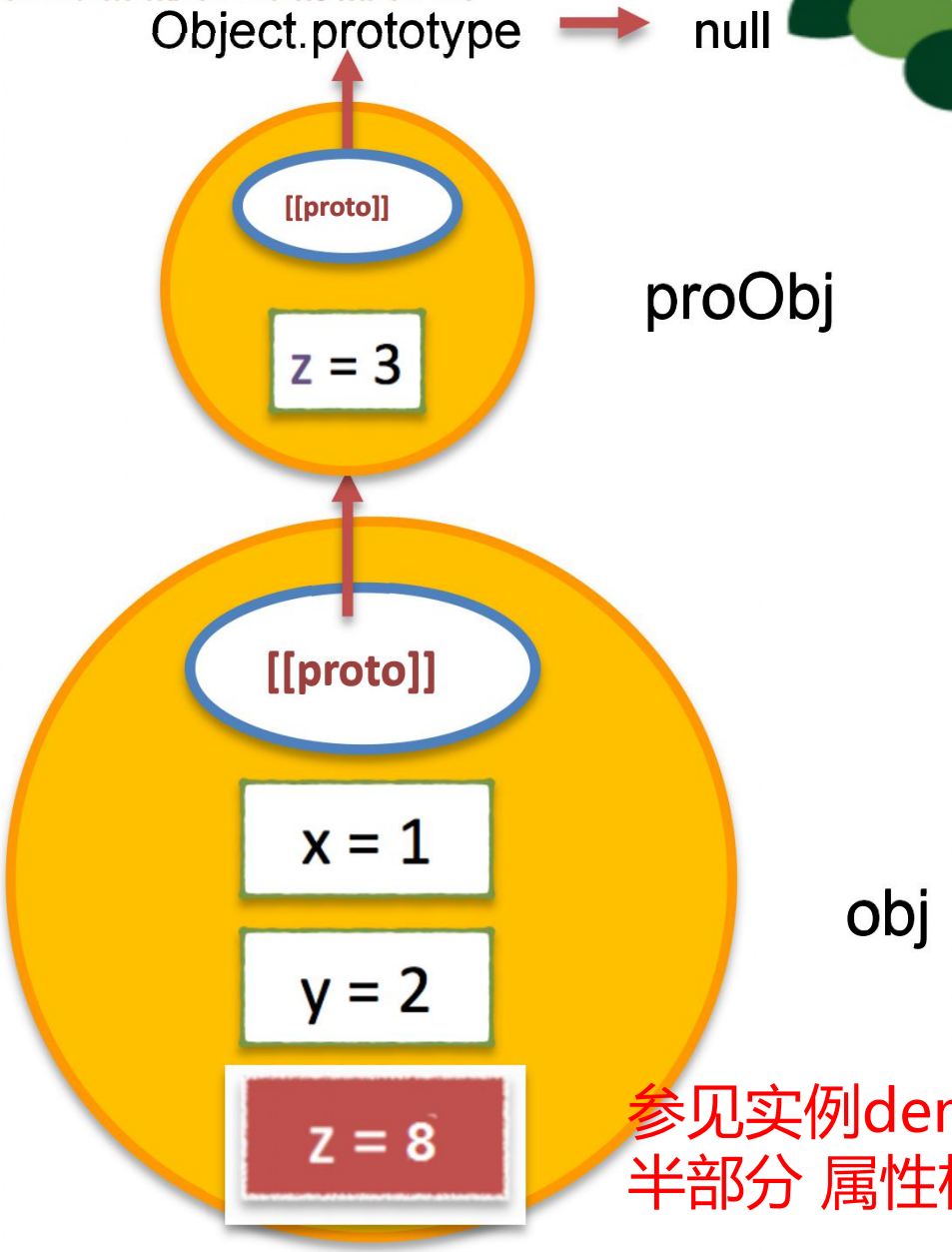
```
var proObj = {
    z:3
};
var obj = Object.create(proObj);
obj.x = 1;
obj.y = 2;

console.log(obj.x); //1
console.log(obj.y); //2
console.log(obj.z); //3

"z" in obj; //true
obj.hasOwnProperty("z"); //false
```



参见实例demo03前半部分 原型链综述

JS对象的原型链-自有属性和继承属性的操作

```
obj.z = 5;

obj.hasOwnProperty('z'); // true
obj.z; // 5
proObj.z; // still 3


obj.z = 8;
obj.z; // 8


delete obj.z; // true
obj.z; // 此时是几?


delete obj.z; // true
obj.z; // still 3!!!
```



参见实例demo03后半部分 属性相关操作

内容提纲

- JS对象及继承方式综述
- JS对象的原型链
- **基于构造函数实现的原型继承**

基于构造函数实现的原型继承

- 通过构造函数来创建对象
  - 当一个函数与new结合，该函数将作为构造函数来使用，用来创建JS对象
  - JS（ES5）中没有其他语言（C++、Java）中的类，JS中通过构造函数来实现类的功能
  - 在JS中构造函数也是对象，有一个重要的属性（原型 prototype），该属性与继承相关

```
function Person(age,name) {
    this.name = name;
    this.age = age;
}
Person.prototype.sayHi = function () {
    console.log("Hi,i'm "+this.name);
};
var p1 = new Person(20,"Jack");
```

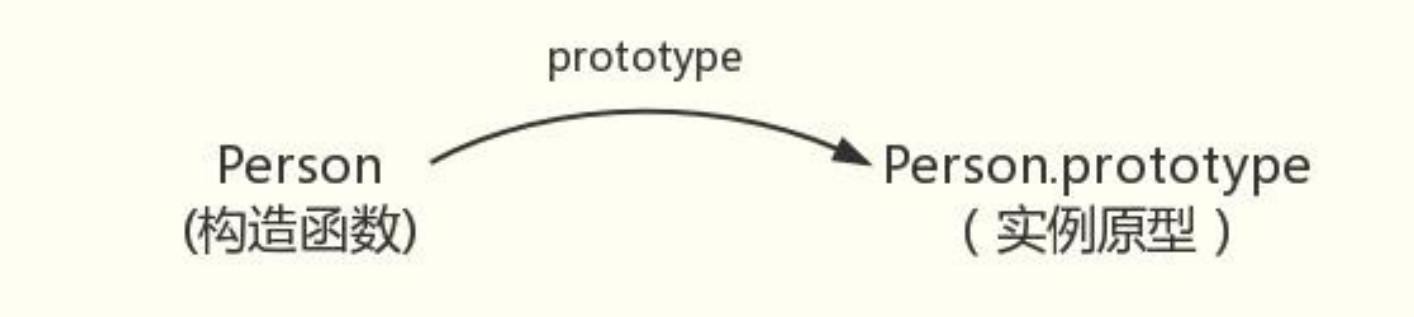
参见实例demo04 构造函数实例化对象

基于构造函数实现的原型继承

- 基于构造函数创建的对象，它的原型是谁呢？
  - 构造函数有一个重要属性（原型 prototype），该属性就是实例化出来的对象的原型
  - 构造函数的这个属性（原型 prototype）是真实对象，实例化的对象通过它实现属性继承

```
function Person(){
}
Person.prototype.name = "Mike";
Person.prototype.age = 21;
Person.prototype.sayName = function(){console.log(this.name);};
```

思考：这样写name和age相当于给谁添加了属性？如果name和age属性写在Person构造函数中会怎样？



基于构造函数实现的原型继承-原型链

- 通过实例化出来的对象的__proto__属性来确认下原型
  - 实例化的这个对象，有一个属性__proto__指向原型
  - 通过判断得知实例化出来的对象的__proto__就是构造函数的prototype属性

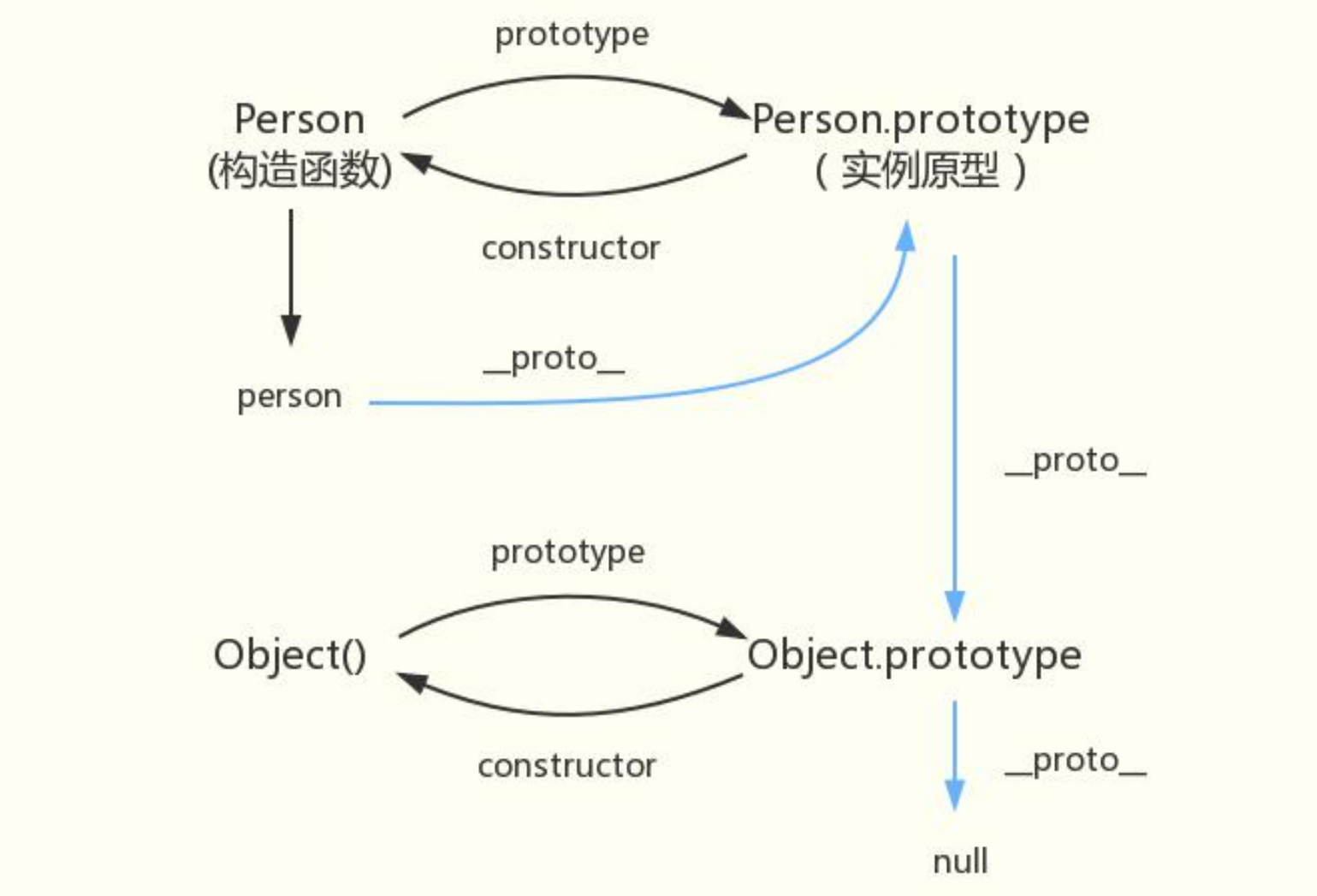
```
function Person(name) {
    this.name = name;
    this.age = 21;
}
Person.prototype.sayHi = function () {console.log(this.name);};

var p1 = new Person("Mike");
p1.sayHi();
console.log(p1.__proto__ === Person.prototype);//true
```

思考：name属性是添加到Person.prototype上了，还是添加到p1上了?

不存在私有属性时：可将属性写在构造函数中，将方法添加到构造函数的prototype属性上，实现方法共享

参见实例demo05 构造函数与原型

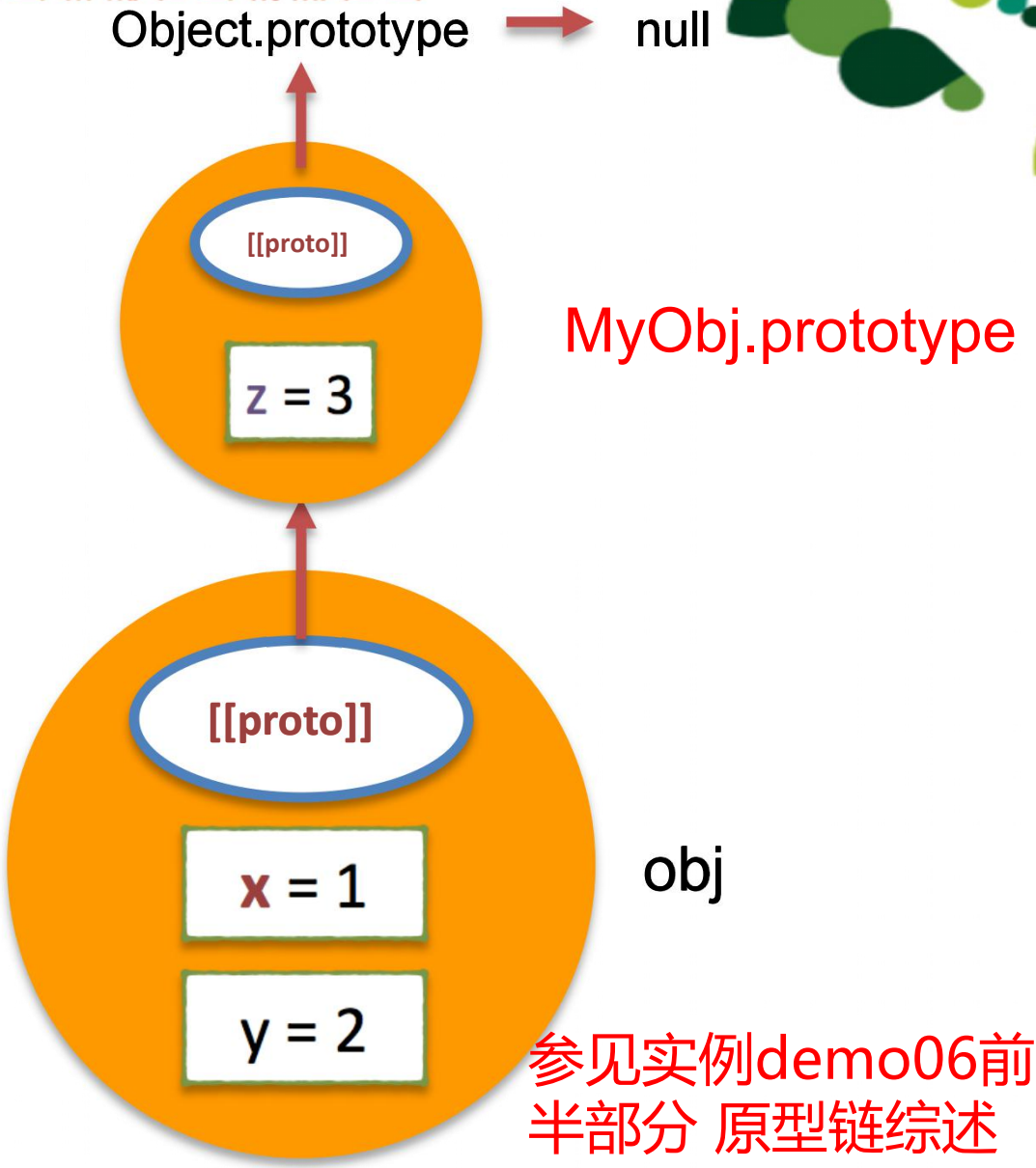
基于构造函数实现的原型继承以及原型链的图解



思考：
person.constructor得到的是什么？

基于构造函数实现的原型继承-属性操作

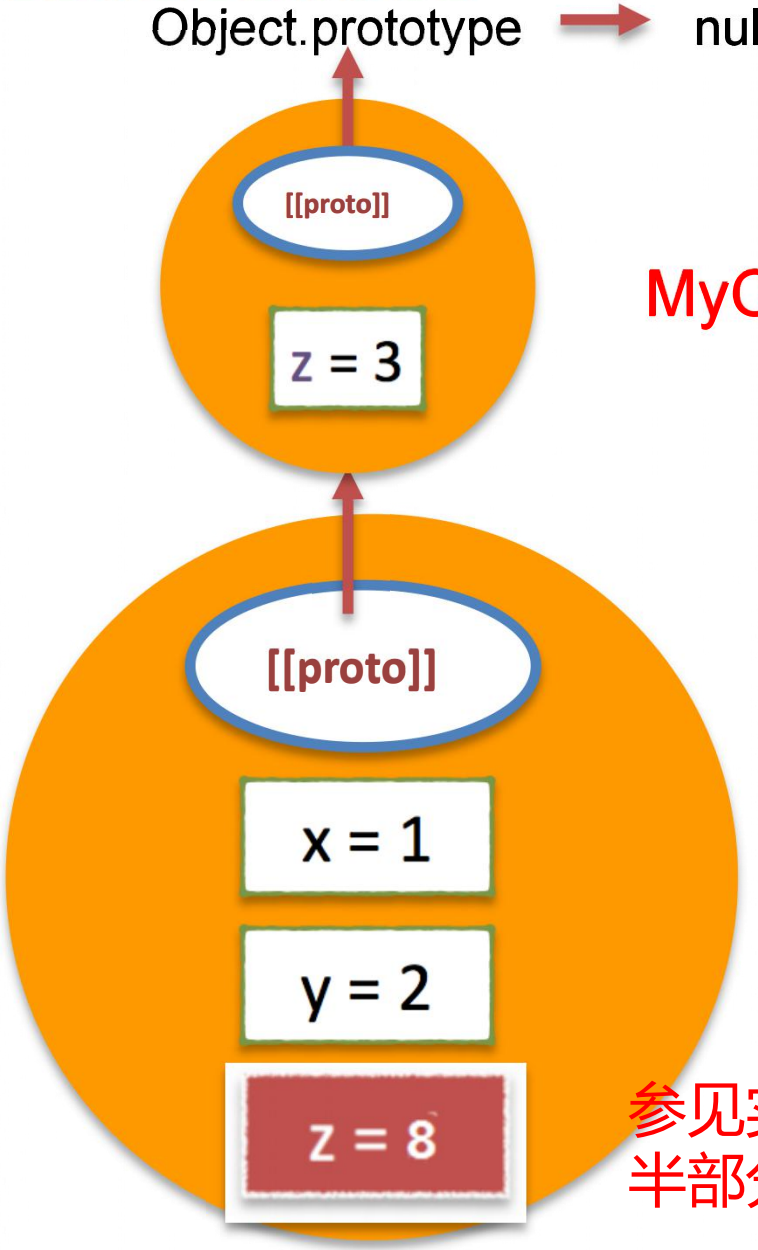
```
function MyObj() { }
MyObj.prototype.z = 3;

var obj = new MyObj();
obj.x = 1;
obj.y = 2;

console.log(obj.x); //1
console.log(obj.y); //2
console.log(obj.z); //3

"z" in obj; //true
obj.hasOwnProperty("z"); //false
```



参见实例demo06前半部分 原型链综述

基于构造函数实现的原型继承-属性操作

```
obj.z = 5;

obj.hasOwnProperty('z'); // true
obj.z; // 5
MyObj.prototype.z; // still 3


obj.z = 8;
obj.z; // 8


delete obj.z; // true
obj.z; // 此时是几?


delete obj.z; // true
obj.z; // still 3!!!
```



参见实例demo06后半部分 属性相关操作

Thank You!

JavaScript进阶

---深入理解JS的继承方式

河北师范大学软件学院
Software College of Hebei Normal University

内容提纲

- JS对象-对象原型继承
- 通过构造函数模拟类-类的继承
- JS继承补充部分

JS原型继承的方式及其优缺点

- JavaScript的原型继承是对象-对象的继承
  - 每个对象都有一个原型对象（可动态的指定原型，来改变继承关系，最原始的原型是null）
  - 思考并回答三种方式创建的对象的原型都是什么？
  - 多个对象继承于一个原型时，存在原型共享（节省内存如共享方法，但也带来了共享问题）

```
var superObj = {
    x:1,
    y:2
};
var subObj_First = Object.create(superObj);
var subObj_Second = Object.create(superObj);
subObj_First.__proto__.x = 5;
console.log(subObj_Second.x);
```

思考：若此行写为subObj_First.x = 5;
结果又是如何？

参见实例demo07

JS原型继承的方式及其优缺点

- 构造函数实现的对象-对象的原型继承的原型共享问题

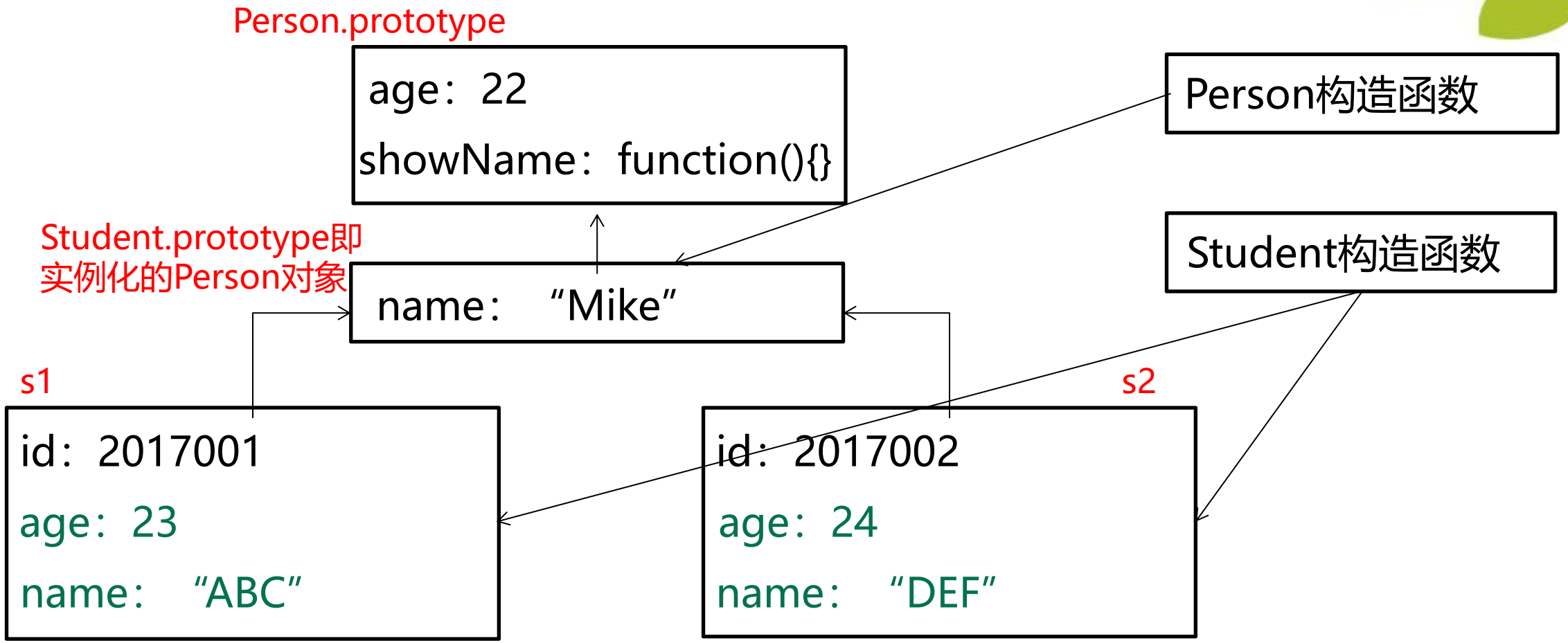
```
function Person(name){
    this.name = name;
};
Person.prototype.age = 22;
Person.prototype.showName = function(){console.log(this.name);};
function Student(id){
    this.id = id;
}
Student.prototype = new Person("Mike");
var s1 = new Student(2017001);
var s2 = new Student(2017002);
//测试如下代码，思考为什么，这样的继承有什么弊端
console.log(s1.name,s1.age,s1.id);
console.log(s2.name,s2.age,s2.id);
s1.__proto__.name = "Jack";
console.log(s2.name);
s2.__proto__.__proto__.age = 99;
console.log(s2.age);
```

思考共享的弊端，如何给每个Studnent对象添加自有的name属性，s1.name =”Jack”，和原型的name属性什么关系，思考这样的话是否造成内存的浪费，具体参见下页图解

参见实例demo08

JS原型继承的方式及其优缺点

- 上页代码图解（原型共享问题）



参见实例demo08 原型继承的原型共享问题

内容提纲

- JS对象-对象原型继承
- **通过构造函数模拟类-类的继承**
- JS继承补充部分

模拟类-类继承的形式一（避免原型共享）

```
function Person(name,age){
    this.name = name;
    this.age = age;
};
Person.prototype.showName = function(){
    console.log(this.name);
};
function Student(name,age,id){
    Person.call(this,name,age);
    this.id = id;
}
Student.prototype.__proto__ = Person.prototype;
var s1 = new Student("xxx",22,2017001);
var s2 = new Student("www",23,2017002);
```

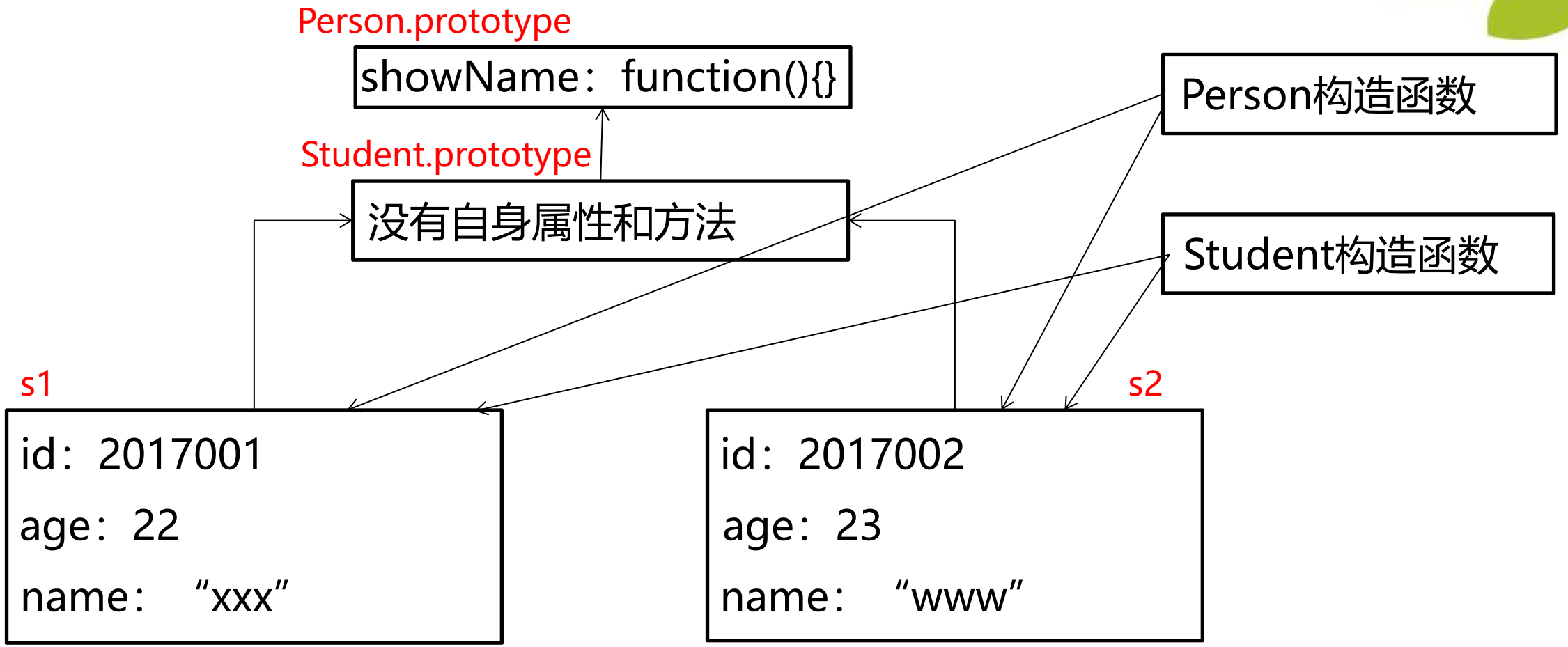
思考：name属性添加到哪个对象上了？Person.prototype、Student.prototype还是实例化的对象上？

推荐：将方法添加到对象的原型上（即构造函数的prototype上）便于共享，节省内存

参见实例demo09 Part1 JS面向对象继承

模拟类-类继承的形式-图解

- 构造函数实现的对象-对象的原型继承的原型共享问题



参见实例demo08 原型继承的原型共享问题

模拟类-类继承的形式二 （避免原型共享）

```
function Person(name,age){
    this.name = name;
    this.age = age;
};
Person.prototype.showName = function(){
    console.log(this.name);
};
function Student(name,age,id){
    Person.call(this,name,age);
    this.id = id;
}
Student.prototype = Object.create(Person.prototype);
Student.prototype.constructor = Student;
var s1 = new Student("xxx",22,2017001);
var s2 = new Student("www",23,2017002);
```

如果不把
Student.prototype.constructor
指回Student，那它将指向谁?

参见实例demo09 Part2 JS面向对象继承

内容提纲

- JS对象-对象原型继承
- 通过构造函数模拟类-类的继承
- JS继承补充部分

JS继承补充部分

- 静态方法与原型方法的区别
  - 静态方法是构造器函数对象（类）的属性，原型方法是实例化对象（对象）的原型的属性
  - 使用形式有什么不同，区别在哪里？（属性共享）
  - 思考Object.getPrototypeOf(...)与Object.prototype.isPrototypeOf(...)

```
var BaseClass = function() {};
BaseClass.prototype.f2 = function () {
    console.log("This is a prototype method ");
};
BaseClass.f1 = function(){//定义静态方法
    console.log("This is a static method ");
};
BaseClass.f1();//This is a static method
var instance1 = new BaseClass();
instance1.f2();//This is a prototype method
```

参见实例demo10 静态方法与原型方法

JS继承补充部分

- 再谈对象原型的constructor属性
  - 因为对象实例从原型中继承了constructor，所以可以通过constructor得到实例的构造函数
  - 确定对象的构造函数名、创建相似对象、constructor可用于指定构造函数

```
function Foo() {}
var f = new Foo();
console.log(f.constructor.name);//Foo

function Constr(name) {this.name = name;}
var x = new Constr("Jack");
var y = new x.constructor("Mike");
console.log(y,y instanceof Constr);
```

```
▶ Constr {name: "Mike"} true
```


参见实例demo11 Part1 constructor应用

JS继承补充部分

- 对象的公有属性、私有属性（回顾闭包）

```
function A(id) {
    this.publicId = id;
    var privateId = 456;
    this.getId = function () {
        console.log(this.publicId,privateId);
    };
}
var a = new A(123);
console.log(a.publicId);//123
// console.log(a.privateId);
a.getId();//123 456
```

涉及到访问私有属性时，需将间接访问私有变量的函数定义在构造函数中

参见实例demo11 Part2 公有属性、私有属性

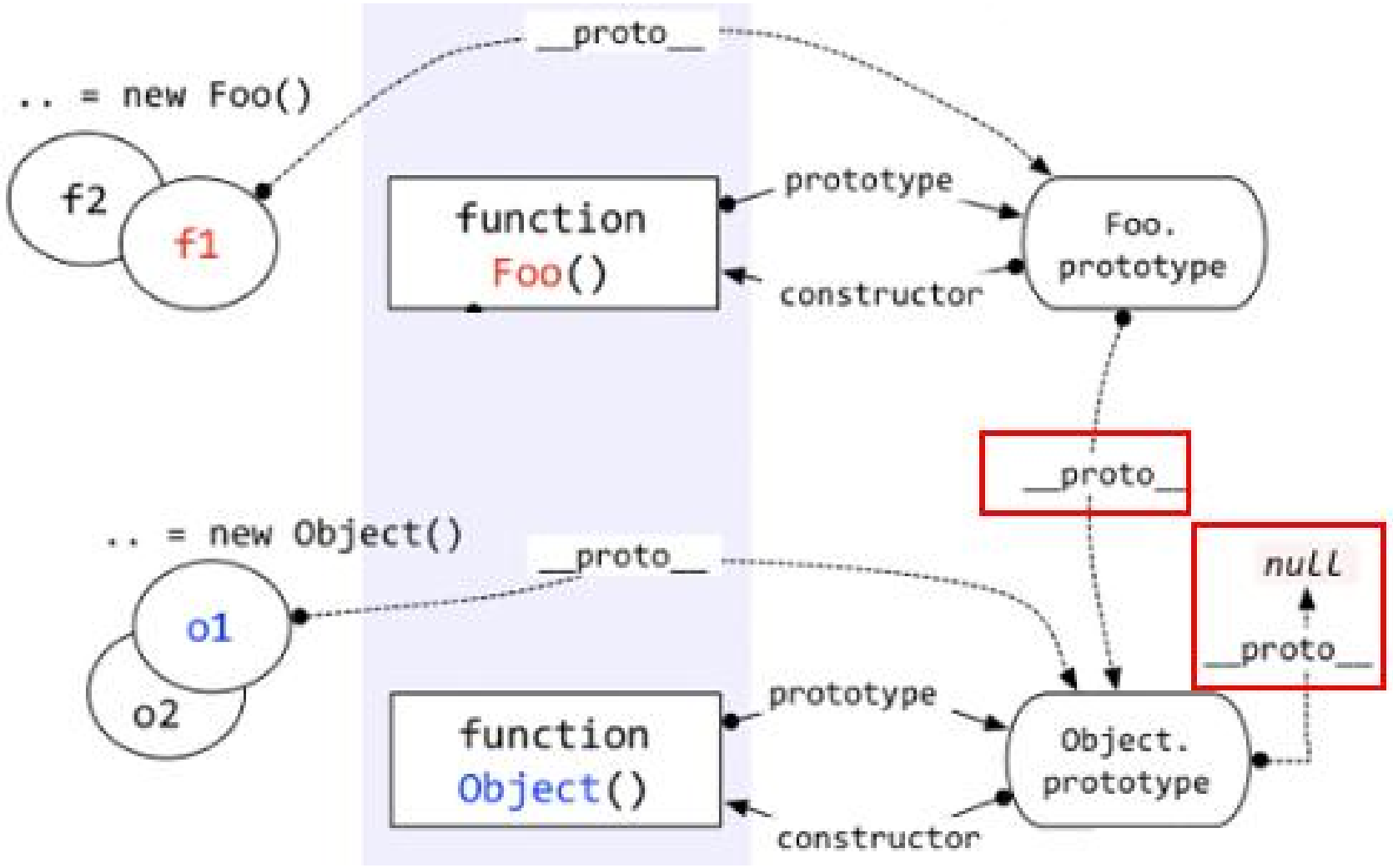
Thank You!

作业

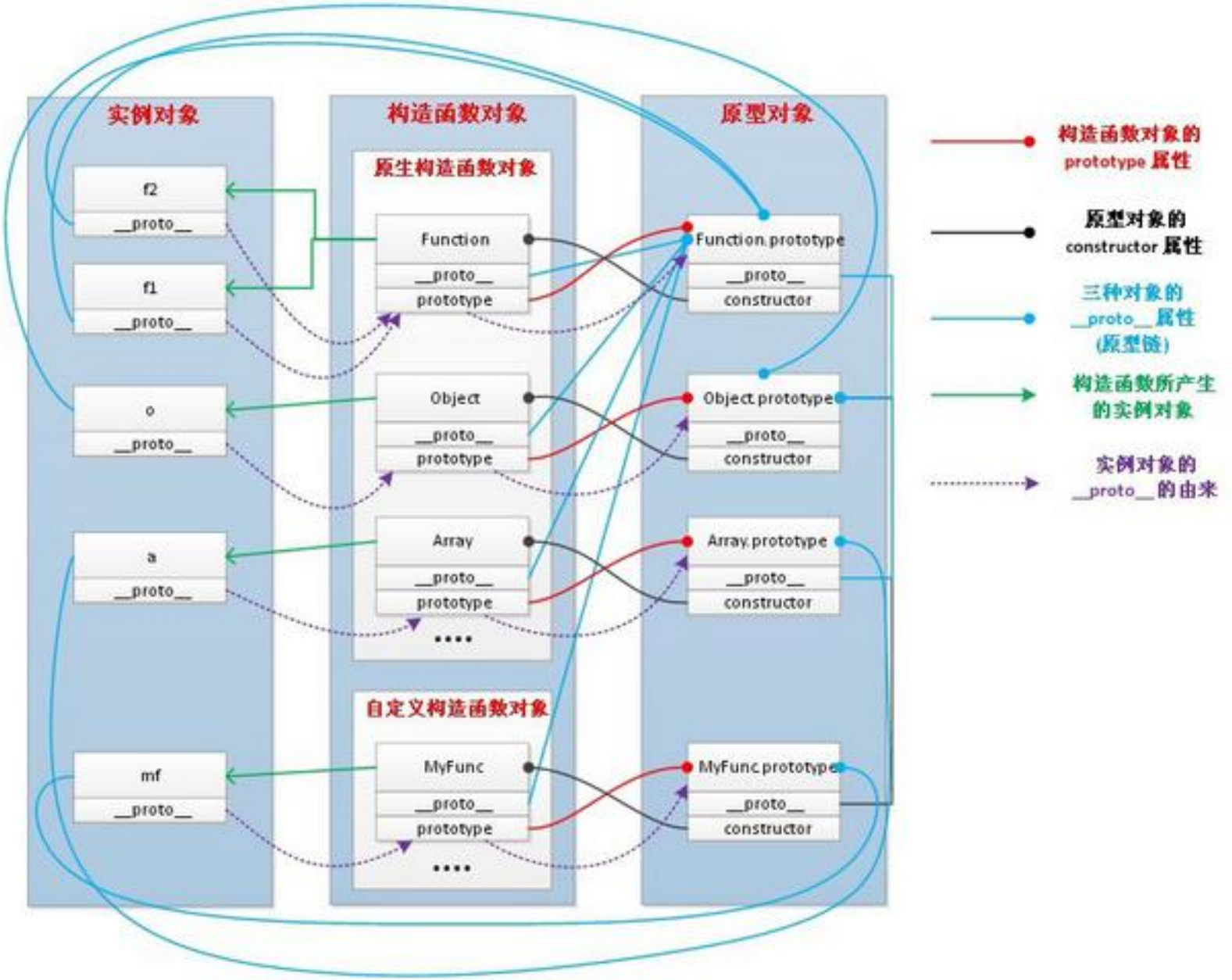
- 阅读《深入理解JavaScript》的第17章
- 学习并重写FlappyBird案例

  http://pan.baidu.com/s/1ge3H8YJ

原型链图解


__proto__
.. = new Foo()
f2
f1
function
Foo()
prototype
constructor
Foo.
prototype
__proto__
.. = new Object()
__proto__
o1
o2
null
__proto__
prototype
function
Object()
Object.
prototype
constructor

原型链图解


实例对象
构造函数对象
原型对象
原生构造函数对象
f2
__proto__
f1
__proto__
o
__proto__
a
__proto__
mf
__proto__
Function
__proto__
prototype
Object
__proto__
prototype
Array
__proto__
prototype
....
自定义构造函数对象
MyFunc
__proto__
prototype
....
Function.prototype
__proto__
constructor
Object.prototype
__proto__
constructor
Array.prototype
__proto__
constructor
MyFunc.prototype
__proto__
constructor
构造函数对象的 prototype 属性
原型对象的 constructor 属性
三种对象的 __proto__ 属性 (原型链)
构造函数所产生的实例对象
实例对象的 __proto__ 的由来