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
```

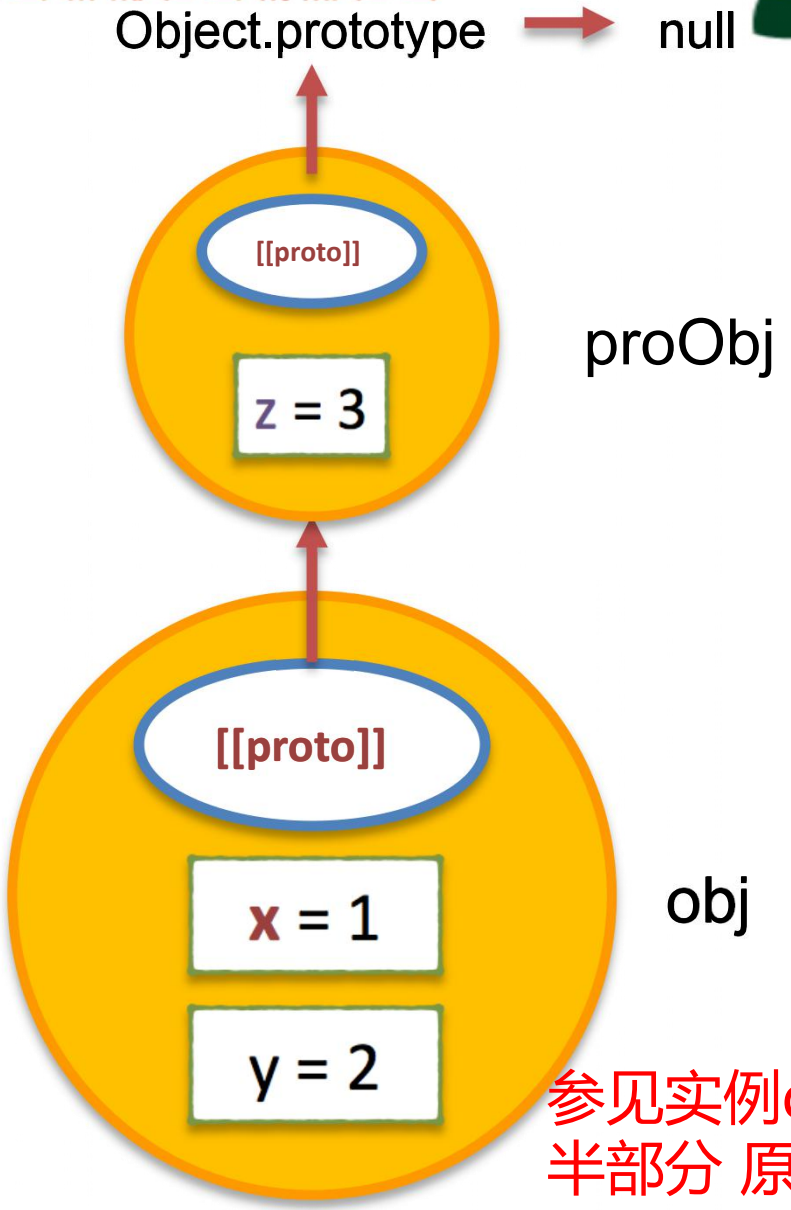

参见实例demo02 __proto__和原型对象

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）

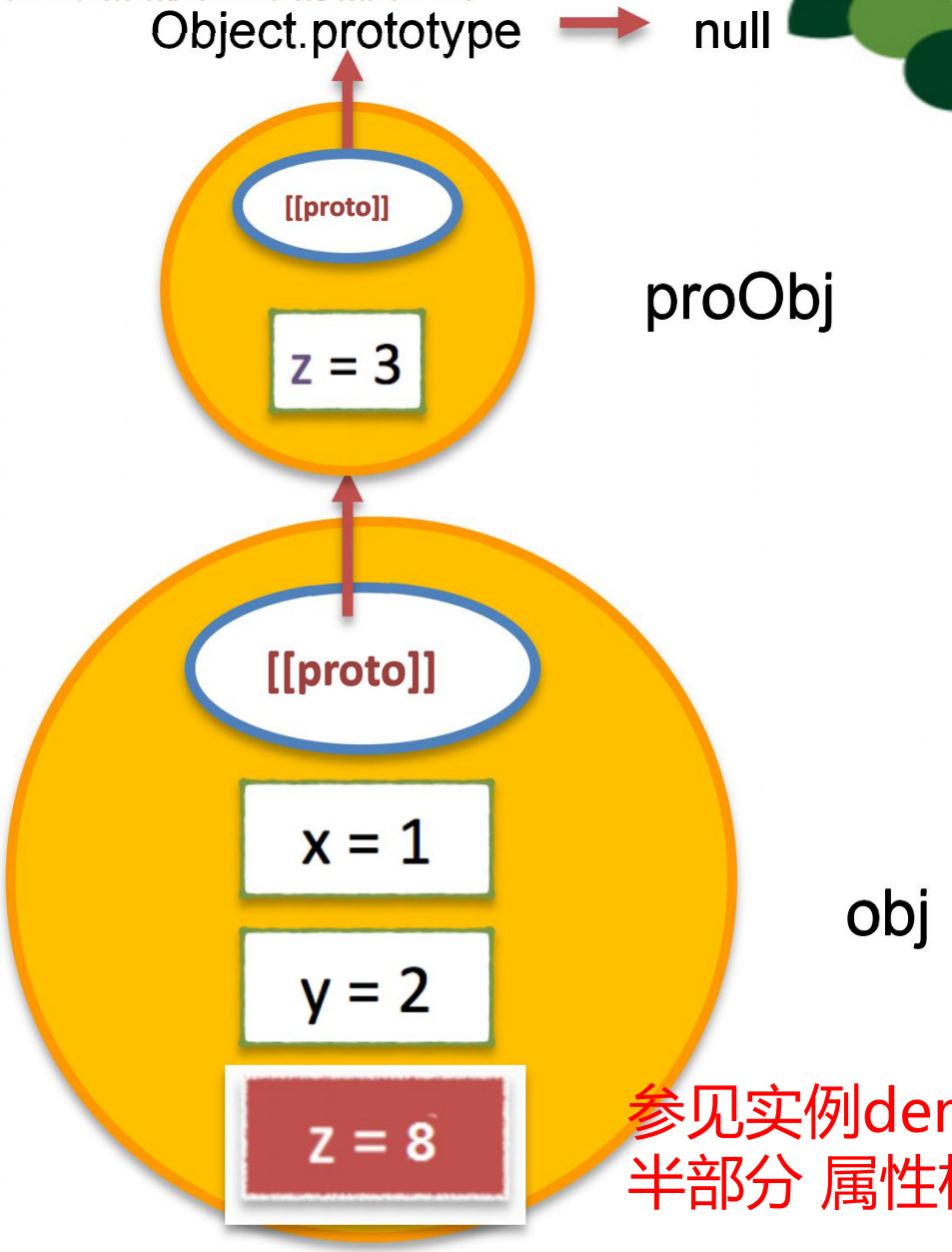
```
var proObj = {
    z:3
};
var obj = Object.create(proObj);
obj.x = 1;
obj.y = 2;

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");
```

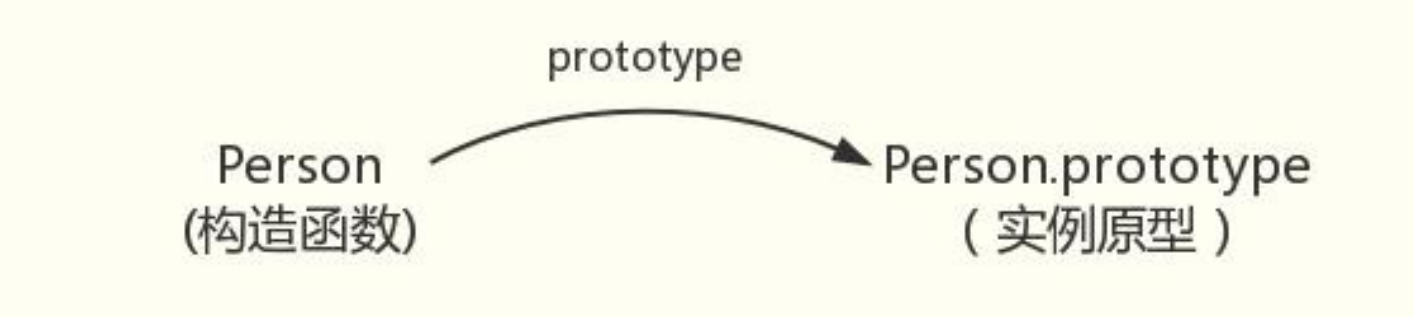
参见实例demo04 构造函数实例化对象

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

- 可通过实例化出来的对象的__proto__属性来确认原型
  - 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

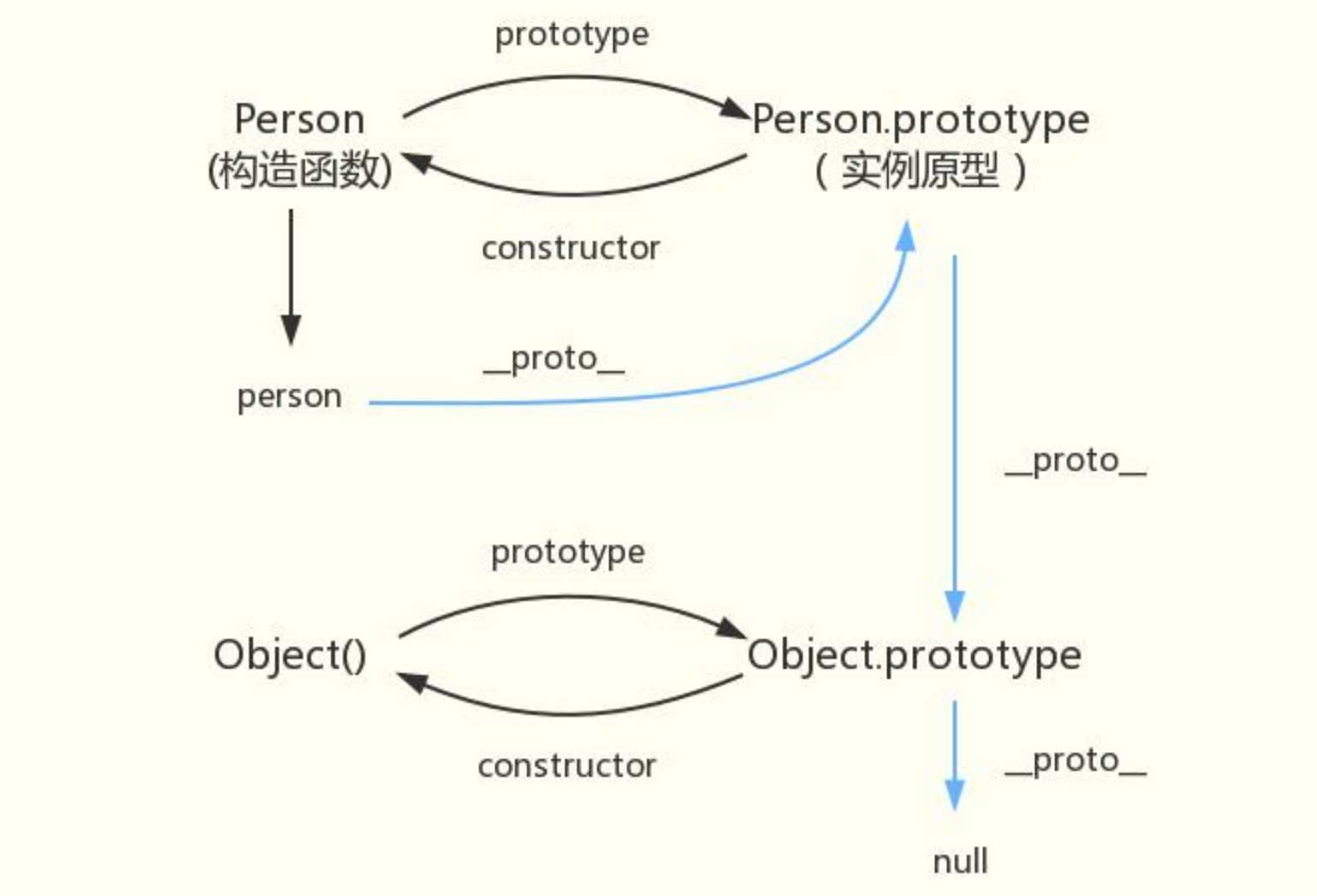
```
function Person(name) {
    this.name = name;
    this.age = 21;
}
Person.prototype.sayHi = function () {console.log(this.name);};

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?

分析：属性和方法定义在构造函数中和写在prototype上这两种情况有什么不同（没有私有属性时，常将方法添加到构造函数的prototype属性上，实现方法共享，而属性根据情况来确定）

参见实例demo05 构造函数与原型

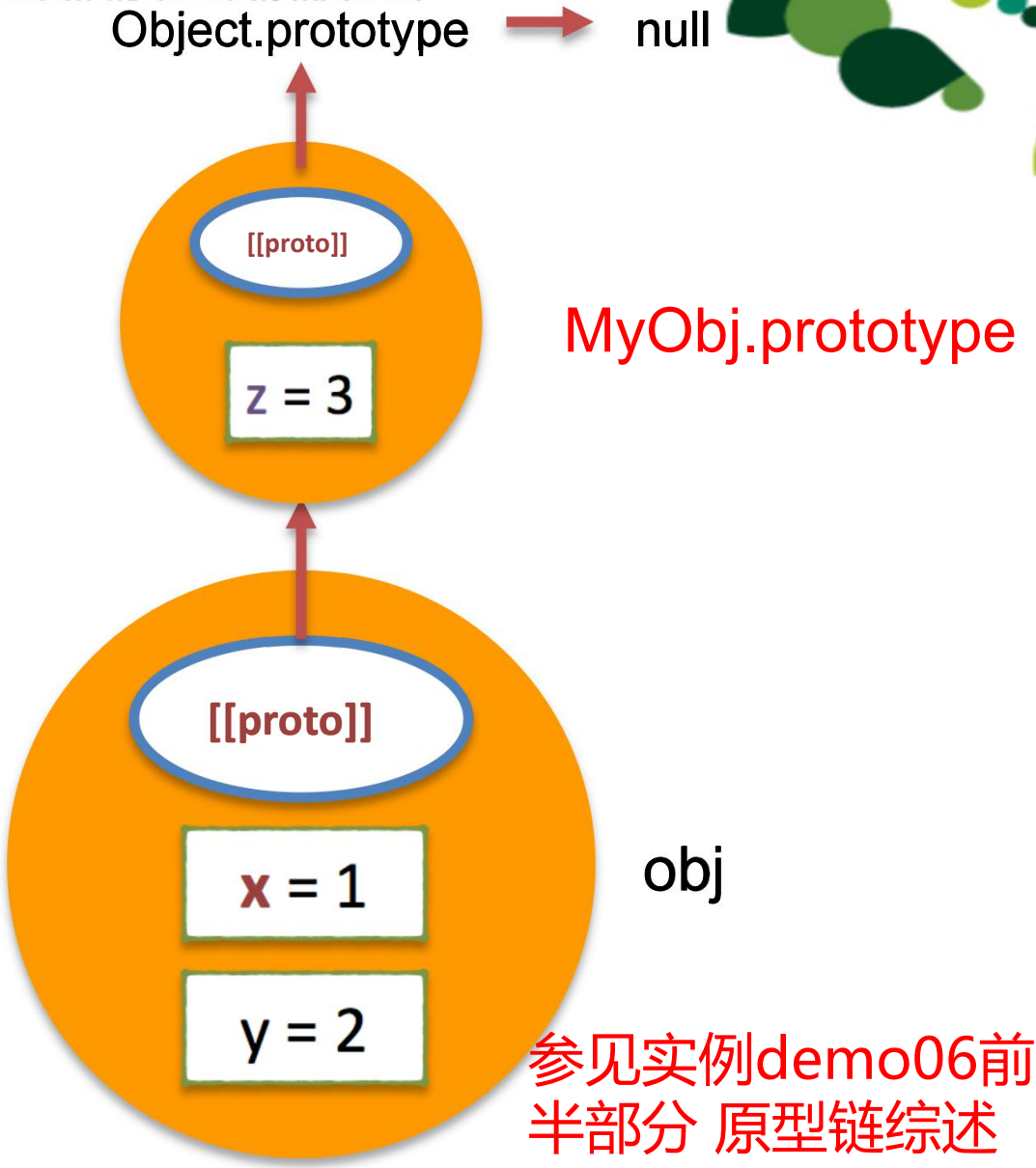
基于构造函数实现的原型继承以及原型链的图解



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

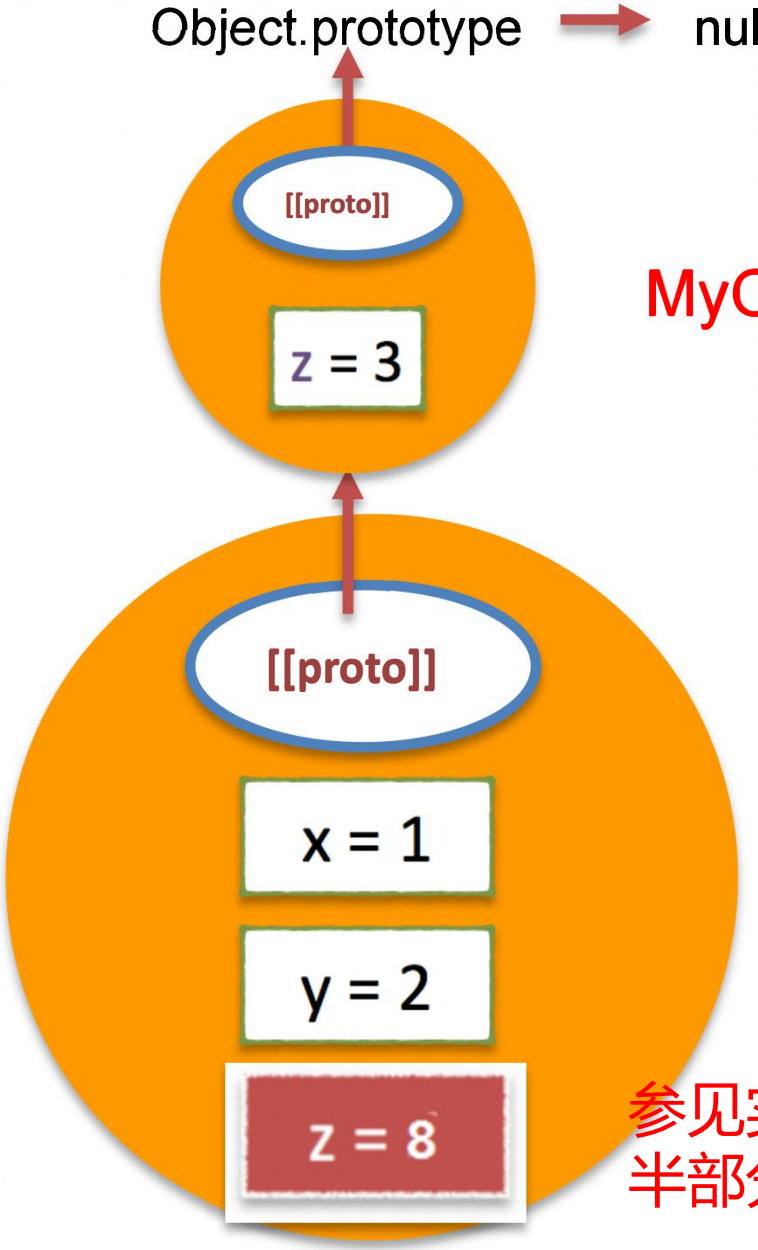
```
function MyObj() { }
MyObj.prototype.z = 3;

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!

作业

- codefordream网站上JavaScript中级
- 复习本章内容及练习
- freecodecamp在线任务