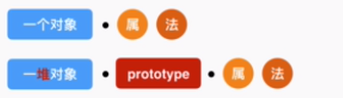
创建对象的三种方式：

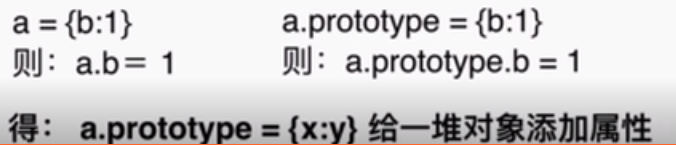


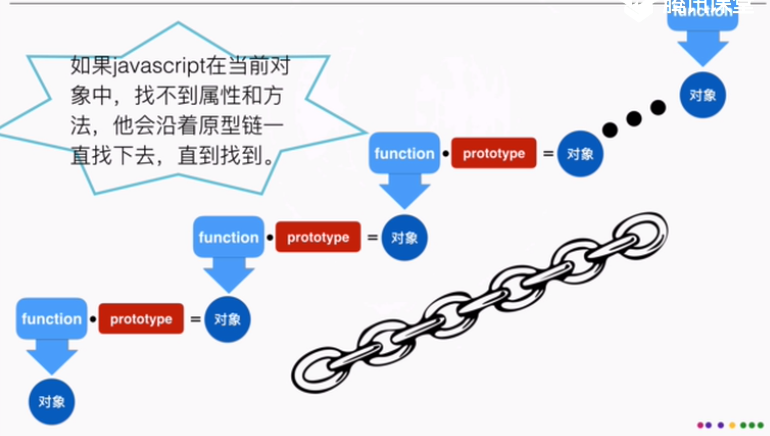


一个对象通过 对象.属性/方法，来添加属性或者方法

一堆对象通过 对象.prototype.属性/方法 来为一堆对象添加属性或方法

原型（prototype）是一次性给一堆对象添加属性和方法的对象





继承：复用其他类的属性和方法，扩展自己的属性和方法



function A(){

this.a = 1

}

function B(){

this.b = 2

}

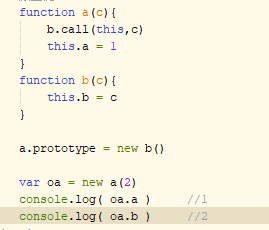
A.prototype = new B() //A类继承自B类

var o = new A() //此对象不仅有A的a属性，还继承了B的b属性

存在的问题？

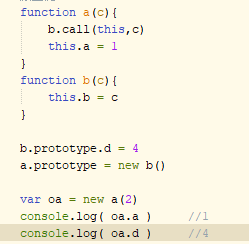
1、无法通过实例化子类对象向父类中传递参数

解决方案： 父类.call(this,参数)



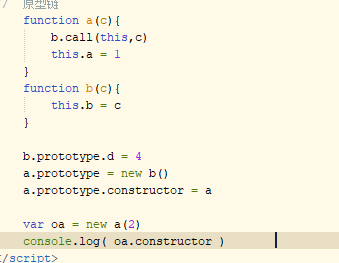
2、如果只是用call方法，则通过原型给父类中添加属性和方法子类无法继承

解决方案：子类.prototype = new 父类

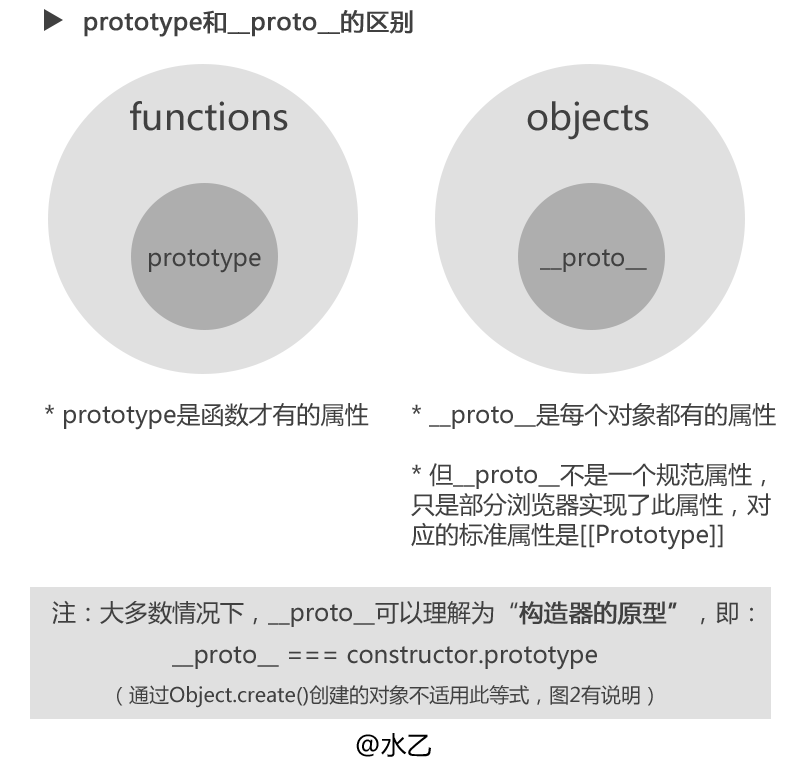


3、子类对象的构造函数指向的都是父类对象的构造函数

解决方案：子类.prototype.constructor = 子类



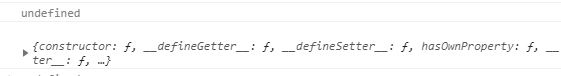
一、prototype和\_\_proto\_\_的区别



var a = {};

console.log(a.prototype) //undefined

console.log(a.\_\_proto\_\_) //Object {}

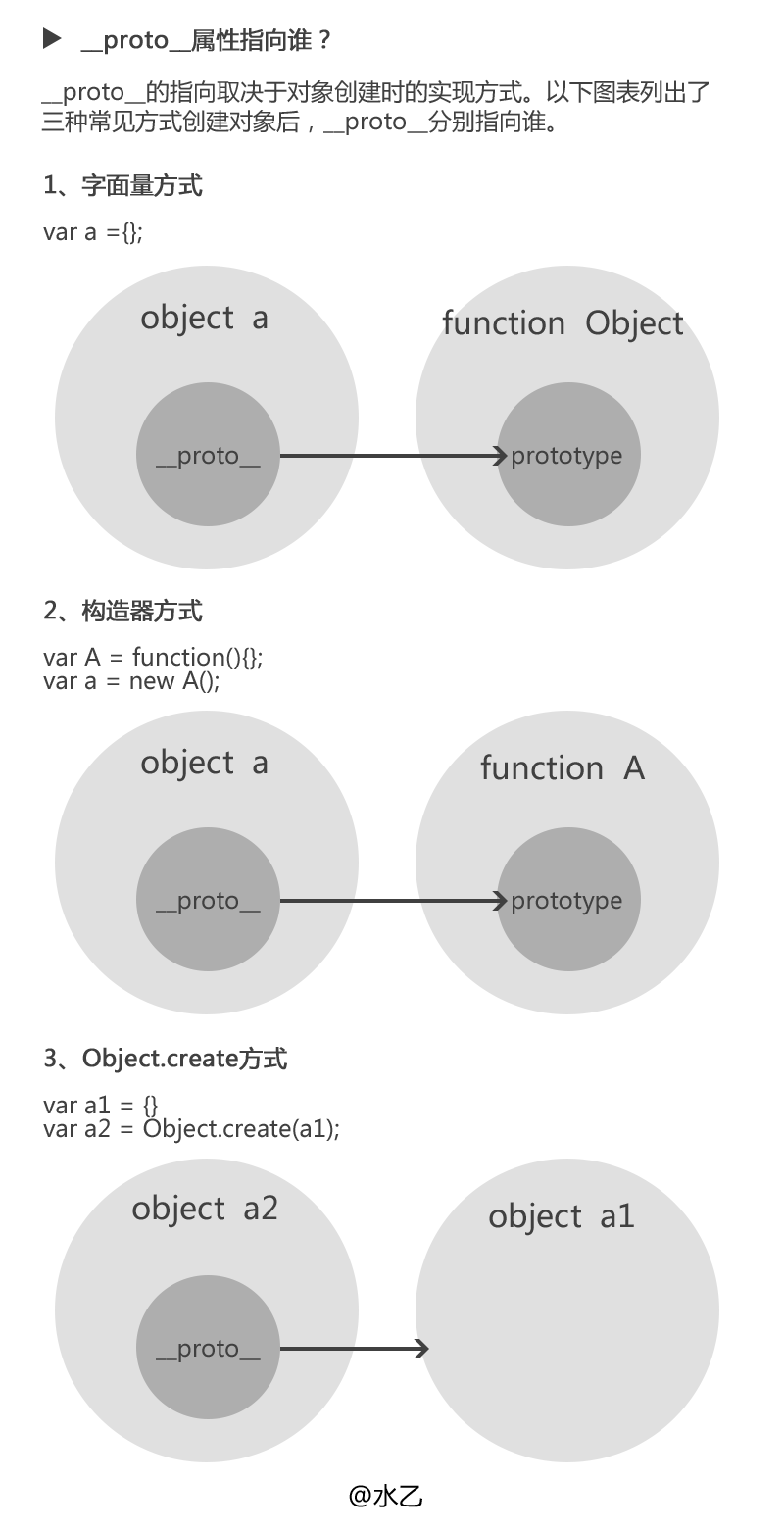


var b = function(){}

console.log(b.prototype) //b {}

console.log(b.\_\_proto\_\_) // function(){}

clipboard.png



/\*1、字面量方式\*/

var a = {};

console.log(a.\_\_proto\_\_); //Object {}

console.log(a.\_\_proto\_\_ === a.constructor.prototype); //true

/\*2、构造器方式\*/

var A = function(){};

var a = new A();

console.log(a.\_\_proto\_\_); //A {}

console.log(a.\_\_proto\_\_ === a.constructor.prototype); //true

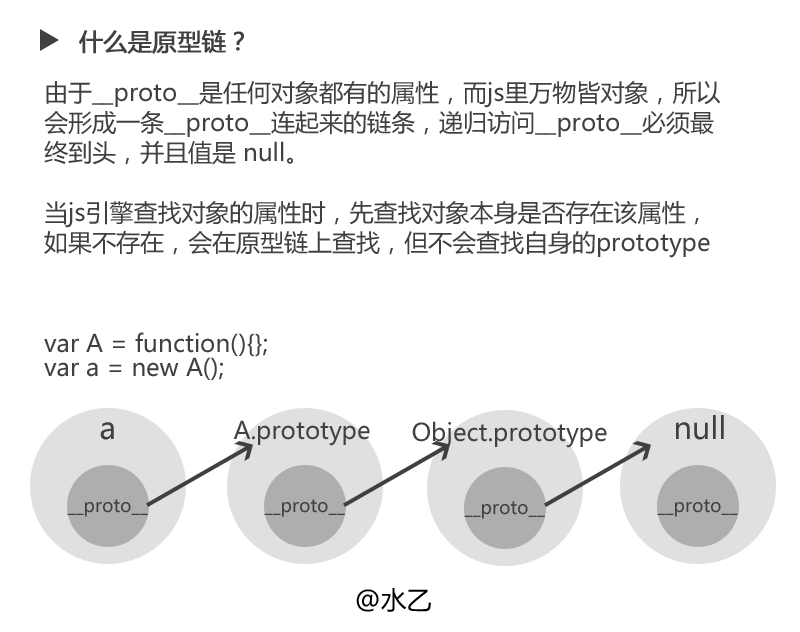
/\*3、Object.create()方式\*/

var a1 = {a:1}

var a2 = Object.create(a1);

console.log(a2.\_\_proto\_\_); //Object {a: 1}

console.log(a.\_\_proto\_\_ === a.constructor.prototype); //false（此处即为图1中的例外情况）



var A = function(){};
var a = new A();
console.log(a.\_\_proto\_\_); //A {}（即构造器function A 的原型对象）
console.log(a.\_\_proto\_\_.\_\_proto\_\_); //Object {}（即构造器function Object 的原型对象）
console.log(a.\_\_proto\_\_.\_\_proto\_\_.\_\_proto\_\_); //null