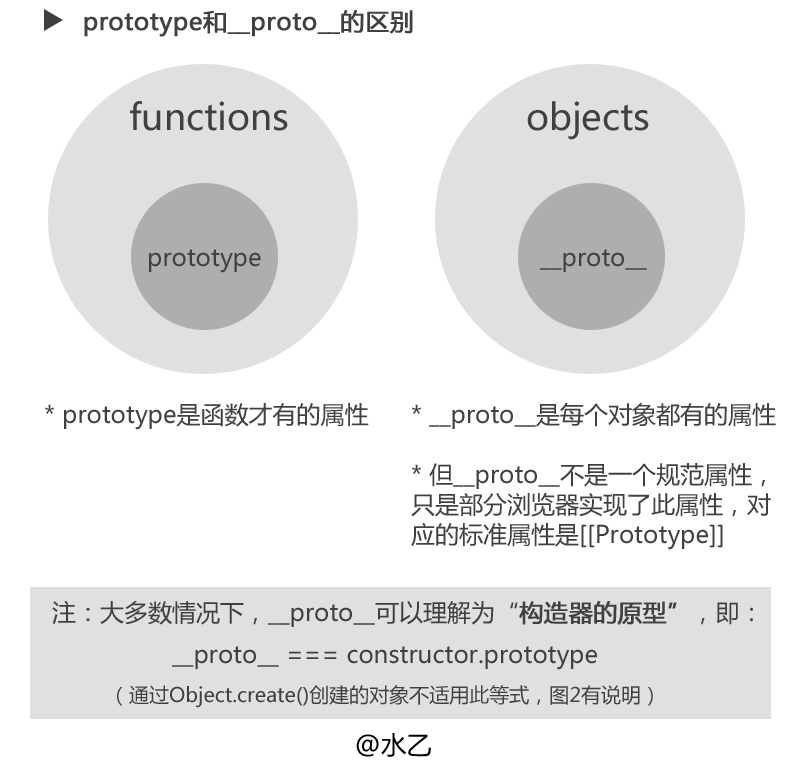
原型

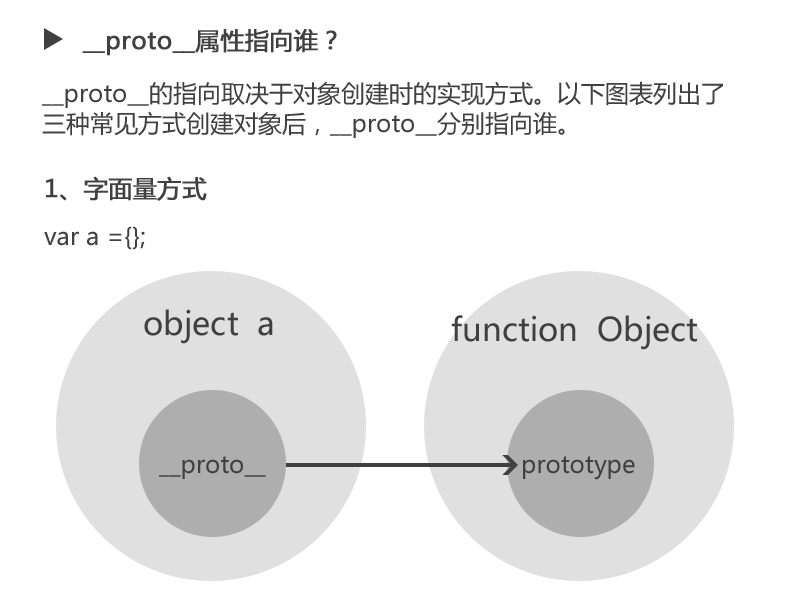
JavaScript 不包含传统的类继承模型，而是使用 prototypal 原型模型。

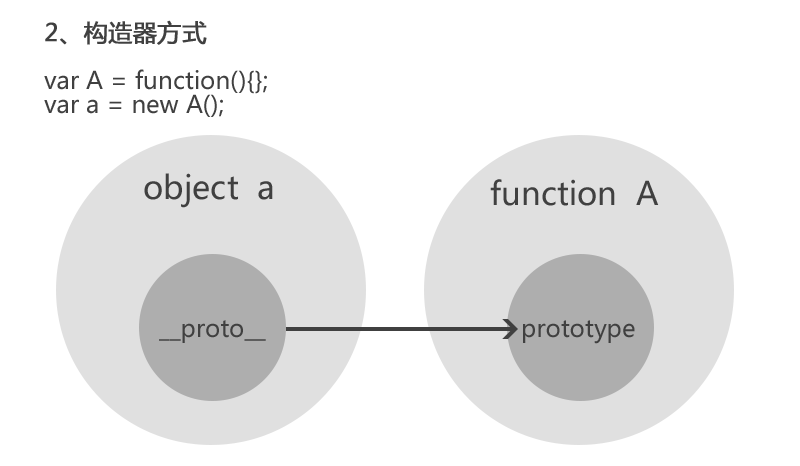
原型即对象，Javascript处处皆对象（普通对象object和函数对象Function），在 JavaScript 中，每当定义一个对象（函数）时候，对象中都会包含一些预定义的属性。其中函数对象的一个属性就是原型对象 prototype。注：普通对象没有prototype,但有\_\_proto\_\_属性。JavaScript的对象中都包含了一个”[[Prototype]]”内部属性，这个属性所对应的就是该对象的原型。“[[Prototype]]”作为对象的内部属性，是不能被直接访问的。所以为了方便查看一个对象的原型，Firefox和Chrome中提供了\_\_proto\_\_这个非标准（不是所有浏览器都支持）的访问器（ECMA引入了标准对象原型访问器”Object.getPrototype(object)”）。在JavaScript的原型对象中，还包含一个”constructor”属性，这个属性对应创建所有指向该原型的实例的构造函数



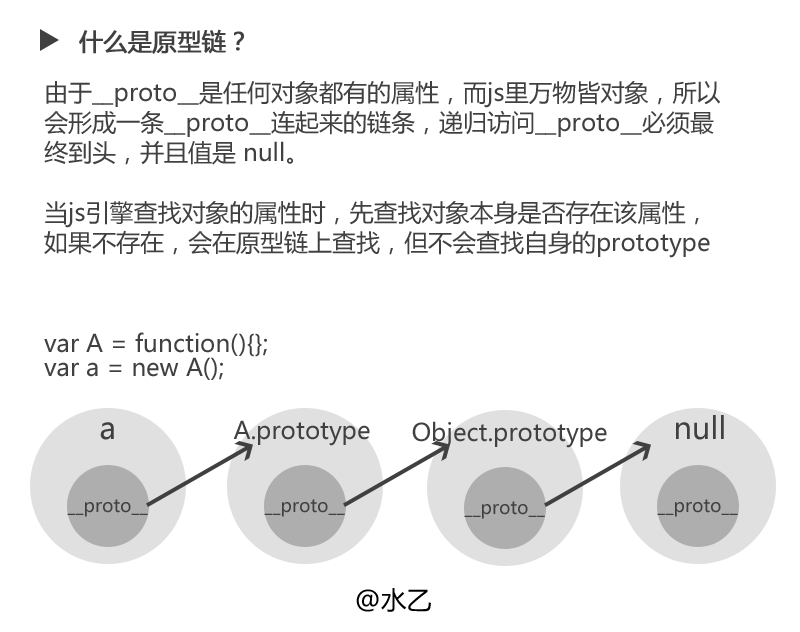
**函数对象和普通对象的区别**

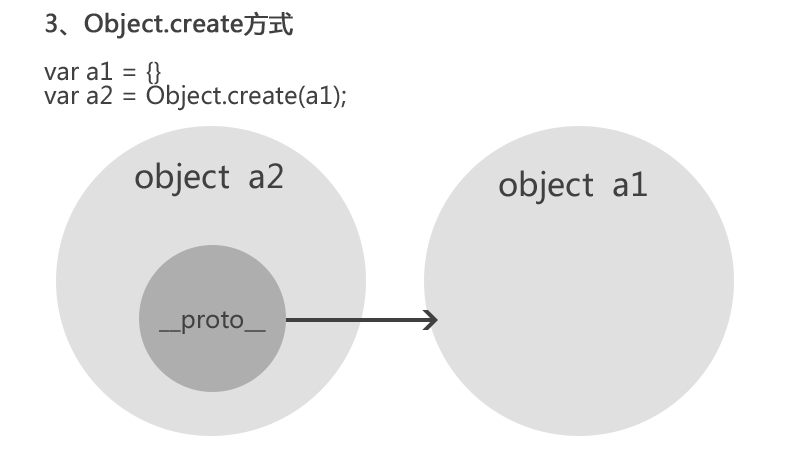
\_\_proto\_\_的指向





**原型链**





原型链代码示例：

var person = function(name){

　　this.name = name

};

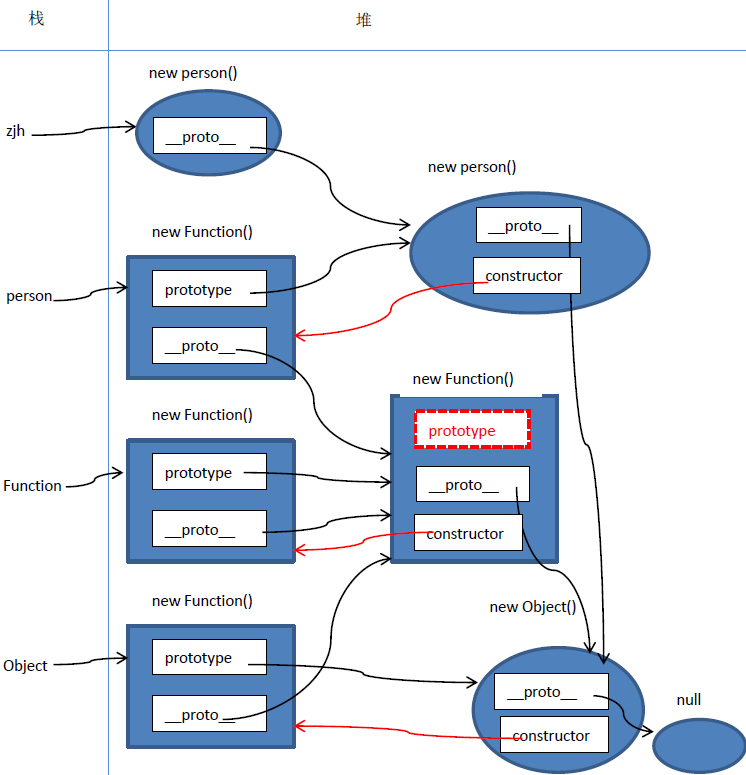
person.prototype.getName = function(){

return this.name;

}

var zjh = new person(‘zhangjiahao');

zjh.getName(); //zhangjiahao



我们把这个有\_\_proto\_\_串起来的直到Object.prototype.\_\_proto\_\_为null的链叫做原型链。

疑点解释：

1、Object.\_\_proto\_\_ === Function.prototype // true

Object是函数对象，是通过new Function()创建，所以Object.\_\_proto\_\_指向Function. prototype。

2、Function.\_\_proto\_\_ === Function.prototype // true

Function 也是对象函数，也是通过new Function()创建，所以Function.\_\_proto\_\_指向Function. prototype。

3、Function.prototype.\_\_proto\_\_ === Object.prototype //true

Function.prototype是个函数对象，理论上他的\_\_proto\_\_应该指向 Function.prototype，就是他自己，自己指向自己，没有意义。

JS一直强调万物皆对象，函数对象也是对象，给他认个祖宗，指向Object. prototype。Object. prototype.\_\_proto\_\_ === null，保证原型链能够正常结束。

**Constructor构造器**

原型对象prototype中都有个预定义的constructor属性，用来引用它的函数对象。这是一种循环引用

person.prototype. constructor === person //true

Function.prototype.constructor === Function //true

Object.prototype.constructor === Object //true

1、注意Object.constructor===Function；//true 本身Object就是Function函数构造出来的

2、如何查找一个对象的constructor，就是在该对象的原型链上寻找碰到的第一个constructor属性所指向的对象

总结

1、原型和原型链是JS实现继承的一种模型。

2、原型链的形成是真正是靠\_\_proto\_\_ 而非prototype

示例：

var animal = function(){};

var dog = function(){};

animal.price = 2000;//

dog.prototype = animal;

var tidy = new dog();

console.log(dog. price) //undefined

console.log(tidy.price) // 2000

