Javascript，万事万物皆对象；

但是呢，对象也是有区别滴，它分为普通对象和函数对象；

也就是object和function；

什么是原型对象？

在javascript中，每当定义一个对象的时候，对象都会包含一些自带的属性，其中，每一个函数对象都有一个属性：prototype属性，这个属性，就是指向函数的原型对象。

在JS里面，每一个对象，都有\_proto\_属性，但是只有函数对象，才有prototype属性；

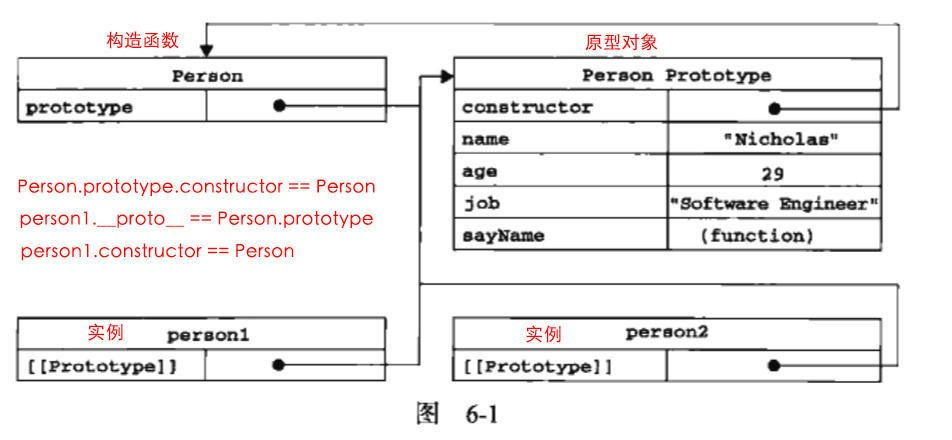
原型对象和普通对象，没啥区别，只要记住原型对象就是：

构造函数.prototype

也就是说，原型对象（person.prototype）就是构造函数（person）的一个实例

## \_proto\_

JS在创建对象的时候，都有一个叫做\_proto\_的内置属性，用于指向创建他的构造函数的原型对象；



通过这个连接图，我们可以看到

Person.prototype.constructor = Person；//构造函数的指针指向他自己

person1.\_proto\_ == Person.prototype；//实例对象的\_proto\_就是构造函数的原型对象

person1.constructor == Person;//实例对象的指针，是指向构造函数；

那么，这个连接链，它的重点是：他是存在于实例和构造函数的原型之间，而不是存在于实例和构造函数之间；

在javascript里面，创建对象有两种方法：

Var obj = {};字面量

Var obj = new Object();和上面完全相同

那么obj就是构造函数的一个实例，所以

obj.constructor ===Object;

obj.\_proto\_ ===Object.prototype;

也就是说，我们实例化的obj变量（实例化对象），他的指针，指向object这个构造函数

他的\_proto\_指向Object的原型对象；

我们用来创建对象的构造函数又称之为构造器；

原型链：function Person(name,age){

this.name = name;

this.age = age;

}

var p1 = new Person('吴泽权',19);

//p1.\_proto\_是什么东西？

//是不是Person.prototype；

/\*因为实例对象.\_proto\_等于构造函数的原型对象\*/

/\*也就是说，p1.\_proto\_等于Person.prototype\*/

/\*\_\_proto\_\_每一边有两个下划线\*/

console.log(p1.\_\_proto\_\_ === Person.prototype)//true

//Person.\_\_proto\_\_又是什么？

//前面说过，实例对象的\_\_proto\_\_等于构造函数的prototype

//因为Person是通过function来写出来的

//也就是说，Person是通过new Function来构造出来的实例对象；

//Person.\_\_proto\_\_===Function.prototype;

console.log(Person.\_\_proto\_\_==Function.prototype);

//Person.prototype.\_\_proto\_\_又是什么呢？

//前面说过，对象分为函数对象和普通对象，prototype他不是函数对象，他只是函数对象衍生的普通对象

//普通对象就是由Object构造的

//那么Person.prototype.\_\_proto\_\_就是等于Object.prototype

console.log(Person.prototype.\_\_proto\_\_==Object.prototype)

//Object.\_\_proto\_\_又是什么？

//和第二题一样，不论是Object还是Function，都是通过函数构造的

console.log(Object.\_\_proto\_\_==Function.prototype);

//Object.prototype.\_\_proto\_\_又是什么呢？

//记住，Object对象的原型对象，已经是处于原型链的顶端了，顶端没有东西了啊，所以是null

console.log(Object.prototype.\_\_proto\_\_==null)

前面说过，所有的函数对象的\_\_proto\_\_(原型)都指向Function.prototype

在JS里面，有十多个构造器：例如Number、Object、Array、Date;

他们这些构造器的原型都是Function.prototype

而Math、JSON是以对象的形式存在，不需要new就可以声明，所以他们的原型（\_\_proto\_\_）就是Object.prototy；

也就是说：

所有的构造器都来自Function.prototype，甚至包括根构造器Object以及Function自身，也就是说，所有的构造器都继承了Function.prototype的属性以及方法，比如说length、call、apply

也就是说，构造器的\_\_proto\_\_就是Function.prototype，那么Function.prototype的原型（\_\_proto\_\_）又是谁呢？

在Javascript里面，函数是一等公民

也就是说，所有的构造器也都是一个普通的JS对象，同时它也继承了Object.prototype上面的所有方法：toString,valueOf

Object.protot又是谁呢？

已经到顶了，就是NULL

在ECMAscript核心定义的全部属性中，prototype是最耐人寻味的，对于ECMAscript中的引用类型而言，prototype是保存他们所有实例方法的真正所在，例如toSring、、、都是存在prototype里面，只不过是各自对象的访问实例罢了；

之前学习Js，可以使用很多内置方法

parseInt()、toString()，都可以使用，为什么可以使用呢？

因为，当我们创建一个函数的时候，

var peoson = new Object()；

person是Object的实例，所以继承了Object.prototype的所有方法

例如，当我们创建一个数组的时候，是不是数组就继承了Array.prototype上面的所有方法，所以我们之所以能用那么多的方法，就是因为继承

console.log(Date.\_\_proto\_\_==Function.prototype);

console.log(Math.\_\_proto\_\_==Object.prototype);

console.log(Function.prototype.\_\_proto\_\_==Object.prototype);

console.log(Object.prototype)

console.log(Array.prototype)

var num = [1,2,3,]

console.log(num.hasOwnProperty())

/\*虽然Array.prototype没有hasOwnProperty()这个方法，但是他有原型，他的原型是不是\_\_proto\_\_\*/

//Array()是继承自Object，那么Array.\_\_proto\_\_==Object.prototype

/\*所以,当你使用num.hasOwnProperty()的时候,JS会先找到他的构造函数(Array)的原型对象Array.prototype,看下他有没有hasOwnProperty()

没有的话,一路按照原型对象找下去,下一步就是找Arry.prototype的原型对象,那么就是找Array.prototype.\_\_proto\_\_\*/

总结：原型和原型链是JS实现继承的一种模型

原型链的形成，真正是靠的\_\_proto\_\_,而不是prototype；