**Js原型链：**

在ES6之前，JS没有类和继承的概念，JS是通过原型来实现继承的，在JS中一个构造函数默认带有一个**prototype属性，这个的属性值是一个对象**，同时这个**prototype对象自带有一个constructor属性，这个属性指向这个构造函数**，同时**每一个实例都会有一个\_proto\_属性指向这个prototype对象**，我们可以把这个叫做隐式原型，我们在使用一个实例的方法的时候，会先检查这个实例中是否有这个方法，没有的话就会检查这个prototype对象是否有这个方法，

基于这个规则，如果让原型对象指向另一个类型的实例，即constructor1.protoytpe=instance2，这时候如果试图引用constructor1构造的实例instance1的某个属性p1,

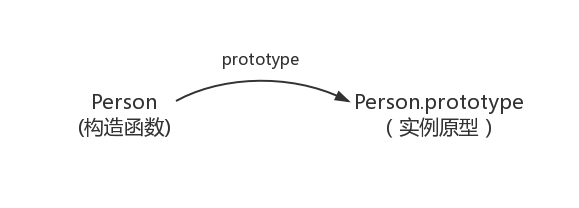
首先会在instance1内部属性中找一遍，

接着会在instance1.\_proto\_（constructor1.prototype）即是instance2中寻找p1

搜寻轨迹：instance1->instance2->constructor2.prototype……->Object.prototype;这即是原型链，**原型链顶端是Object.prototype**

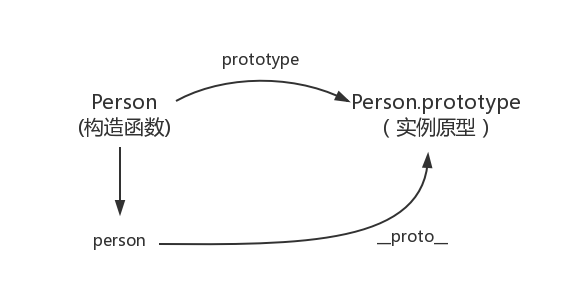
补充学习：

每个函数都有一个prototype属性，这个属性指向了一个对象，这个对象正是调用该函数而创建的实例的原型，那么什么是原型呢，可以这样理解，每一个JavaScript对象在创建的时候就会预制管理另一个对象，这个对象就是我们所说的原型，每一个对象都会从原型继承属性，如图：



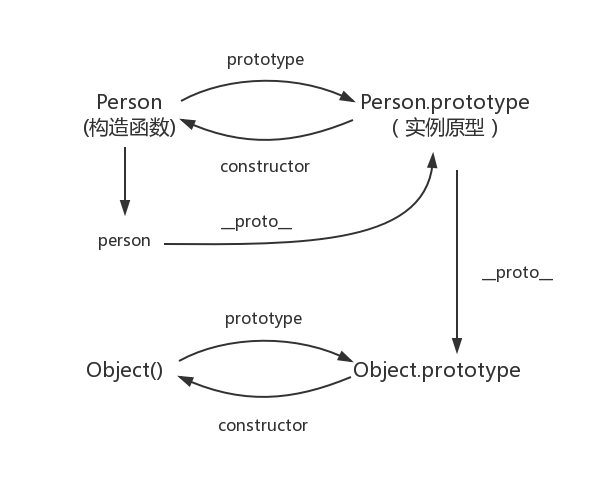
那么怎么表示实例与实例原型的关系呢，这时候就要用到第二个属性\_proto\_

这是每一个JS对象都会有的一个属性，指向这个对象的原型，如图：



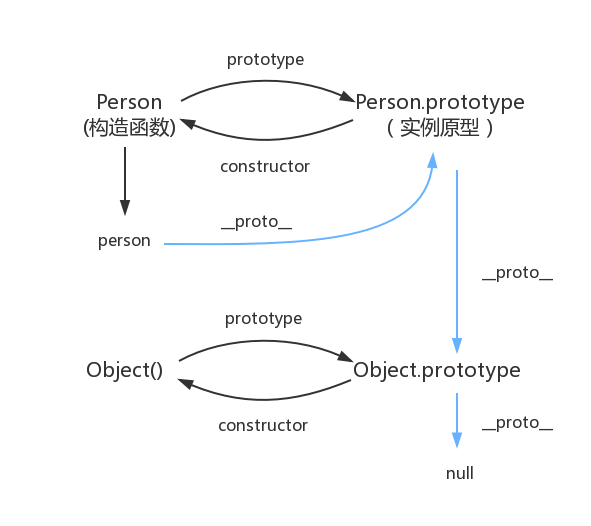
既然实例对象和构造函数都可以指向原型，那么原型是否有属性指向构造函数或者实例呢，指向实例是没有的，因为一个构造函数可以生成多个实例，但是原型有属性可以直接指向构造函数，通过constructor即可

接下来讲解实例和原型的关系：

当读取实例的属性时，如果找不到，就会查找与对象相关的原型中的属性，如果还查不到，就去找原型的原型，一直找到最顶层，那么原型的原型是什么呢，首先，原型也是一个对象，既然是对象，我们就可以通过构造函数的方式创建它，所以原型对象就是通过Object构造函数生成的，如图：  


那么Object.prototype的原型呢，我们可以打印console.log(Object.prototype.\_\_proto\_\_ === null)，返回true

null表示没有对象，即该处不应有值，所以Object.prototype没有原型，如图：



图中这条蓝色的线即是原型链，

最后补充三点：

constructor：

function Person(){

}

var person = new Person();

console.log(Person === person.constructor);

原本person中没有constructor属性，当不能读取到constructor属性时，会从person的原型中读取，所以指向构造函数Person

\_\_proto\_\_：

绝大部分浏览器支持这个非标准的方法访问原型，然而它并不存在与Person.prototype中，实际上它来自Object.prototype，当使用obj.\_\_proto\_\_时，可以理解为返回来Object.getPrototype(obj)

继承：

前面说到，每个对象都会从原型继承属性，但是引用《你不知道的JS》中的话，继承意味着复制操作，然而JS默认不会复制对象的属性，相反，JS只是在两个对象之间创建一个关联，这样子一个对象就可以通过委托访问另一个对象的属性和函数，所以与其叫继承，叫委托更合适，