在 JavaScript 中，是一种面向对象的程序设计语言，但是 JS 本身是没有 “类” 的概念，JavaScript 中，和一般的 OOP 语言不同，它没有 '类'的概念，也就没有 '模板' 来创建对象，而是通过字面量或者构造函数的方式直接创建对象。JS 是靠原型和原型链实现对象属性的继承。

历史原因：1994年，网景公司（Netscape）发布了Navigator浏览器0.9版。这是历史上第一个比较成熟的网络浏览器，轰动一时。但是，这个版本的浏览器只能用来浏览，不具备与访问者互动的能力。比如，如果网页上有一栏"用户名"要求填写，浏览器就无法判断访问者是否真的填写了，只有让服务器端判断。如果没有填写，服务器端就返回错误，要求用户重新填写，这太浪费时间和服务器资源了。基于此Javascript被发明。由于Javascript本身的设计是为了完成表单验证等一些辅助功能，所以在设计上并没有考虑Java一样有庞大的类继承机制。但是没有继承机制，那么javascript中所有的对象就不能共享某种属性，需要为某一类对象更改属性时，便需要为所有对象进行更改。基于此，为了达到表达某一类对象共有属性，又不想太过正式的正式的设计下，使用prototype机制。为了就是表示这类对象都属于此prototype, 当修改prototype中的属性时，属于该prototype的对象属性都被更改。

1.

构造函数(构造器)：普通函数加上 new 关键字，并总会返回一个对象。

例如：

// 构造函数 Person ()

function Person (name, gender) {

this.name = name;

this.gender = gender;

}

var person = new Person ("周杰伦", "男"); //Person在JS中不是类，不同于JAVA, Python中的构建机制，是构造函数

// 最后创建出来的对象实例 person

person

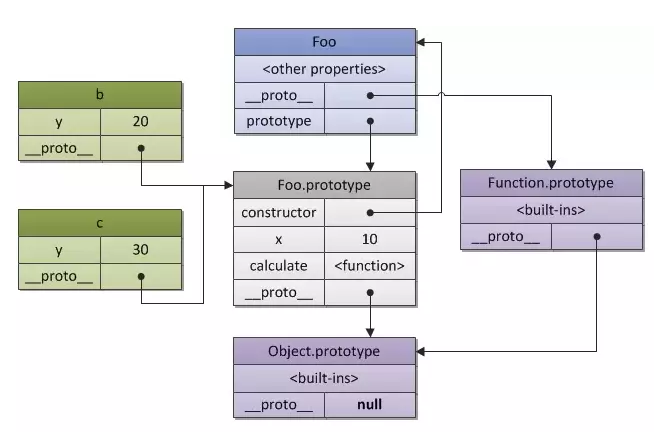
{

name: "周杰伦",

gender: "男"

}

2.



基于上图：

图示分析：b, c 是Foo的实例。Foo是构造函数，也就是b = new Foo(), c = new Foo().

基本原理：

**1. 每个构造函数都有一个原型属性 prototype，该prototype属性指向一个对象，我们称为原型对象。Prototype属性是构造函数才有的。所有由该构造函数构建出的实例，都会获取该原型对象的属性。修改该原型对象，并会修改所有的实例。相当于Python, Java中的类。**

**2. 对象都有一个 \_\_proto\_\_ 属性，指向原型对象。查找对象实例属性时，如果这个对象上没有这个属性，那么 JS 会沿着对象的 \_proto\_链 一层一层地去找，最后如果没找到就返回 undefined. 这样的行为就是Javascript的原型链。**

**3. 由于在JS中，所有的东西都是对象。所以构造函数也是对象，构造函数拥有prototype属性，指向他将构造对象的原型对象。由于构造函数自身也是对象，所以他也有\_proto\_\_属性，这个\_proto\_属性指的是构造构造函数的原型对象。**

**4. 每个原型对象都有一个 constructor 属性指向关联的构造函数。**

**5. 原型链的顶层是是 Object 对象，Object 对象其实也有 \_\_proto\_\_属性，比较特殊的是 Object.\_\_proto\_\_ 指向 null, 也就是空。**

**案例分析：**

**1.对于实例b, c。由于他们是对象，不是构造函数，所以他们有\_\_proto\_\_, 没有prototype属性。\_\_proto\_\_属性指向他们的原型对象Foo.prototype.**

**2. 原型对象Foo.prototype，它是一个对象不是一个构造函数，所以它有\_\_proto\_\_属性，没有prototype属性。它的\_\_proto\_\_属性指向Object.prototype表示它的原型是Object. prototype.**

**3.Foo.prototpye不仅是对象，还是原型对象。所以它有constructor来指向与之相关联的构造函数Foo.**

**4. Foo构造函数由于是构造函数，所以他有prototpye指向它将构建对象的对象原型也就是Foo.prototype. 另外Foo构造函数自身也是对象（JS中所有东西都有是对象）， 所以构造函数自身也有\_\_proto\_\_属性。 所以Foo构造函数也有\_\_proto\_\_属性，指向构造Foo构造函数的原型，也就是Function.prototype. 表示Function.prototype是构造构造函数Foo的原型。**

**5.Function.prototype是对象原型, 它也是对象。它不是构造函数，所以它没有prototype对象，但是它是对象所以有\_\_proto\_\_属性。它的指向应该是一个对象原型构造了Function.prototype, 所以\_\_proto\_\_指向是Object.prototype.**

**6. Object.prototype. 是对象原型, 它也是对象。它不是构造函数，所以它没有prototype对象，但是它是对象所以有\_\_proto\_\_属性。它的指向应该是是一个对象原型构造了Object.prototype, 但是Object已经是顶层，没有什么是构建Object.prototype对象的原型了，所以是null. 同时也是原型链的终点。**

**7.推论，所以如果我们使用Object.prototype.test = “AAA”, 在object.prototype对象是增加一个test属性。那么所有的对象都会通过原型链获取到该属性。**

**实例b, c 他们找寻Foo.prototype, 而Foo.prototype继续通过\_\_proto\_\_找寻它自己的原型Object.prototype找到该属性。**

**对于构造函数Foo, 构造函数Foo也是个对象，它的原型是Function.prototype对象，Function.prototype找不到该属性的时候，继续使用\_\_proto\_\_找寻它的原型，找到Object.prototype。所以构造函数Foo.test也能获取”AAA”**