js对象之\_proto\_和prototype的区别和联系

前言：

记得之前面试时候，一位大佬问了我一个问题，说说\_proto\_和prototype的区别，顿时蒙圈了，因为没有深入了解过这两个属性，对对象的认识也不够深刻；今天就分享一下对这两个属性的认识。

引言：

在js中，万物皆对象，咱们写一个js对象，大多数时候是用构造函数或者对象字面量来创建一个对象等等。那么问题来了，通过不同的方式创建的对象有什么区别呢？

我们知道，每个js对象一定对应一个原型对象，并从原型对象继承属性和方法。那么对象是怎么和这个原型对象对应的呢？

先区别一下两者的概念：

**其实说\_proto\_并不准确，确切的说是对象的[[prototype]]属性**，只不过在主流的浏览器中，都用\_proto\_来代表[[prototype]]属性，因为[[prototype]]只是一个标准，而针对这个标准，不同的浏览器有不同的实现方式。在ES5中用Object.getPrototypeOf函数来获得一个对象的[[prototype]]。在ES6中，使用\_proto\_来代表对象的[[prototype]]。

**而prototyp属性是只有函数才特有的属性，**当你创建一个函数的时候，js会自动为这个函数加上prototype属性，值是一个空对象。所以，函数在js中是非常特殊的，是所谓的一等公民。

那么\_proto\_和prototype是怎么联系起来的呢？让我们来看下下面的代码：

Funtion Person(name,age){

This.name = name;

This.age = age;

}

Var person1 = new Person();

当我们new Person()的时候到底发生了什么？

New 一个构造函数，相当于实例化一个对象，这期间其实发生了这三个步骤：

1. 创建对象，设为o,即：var o = {};
2. 上文提到了，每个对象都要\_proto\_属性，该属性指向一个对象，这里，将o对象的\_proto\_指向构造函数Person的原型对象；
3. 将o作为this去调用构造函数Person，从而设置o的属性和方法并初始化。

完成这三步，这个o对象就与构造函数Person再无联系，这个时候即使构造函数Person再添加任何成员，都不再影响已经实例化的o对象了。

此时，o对象具有了name和age属性，同时具有了构造函数Person的原型对象的所有成员，当然，此时该原型对象是没有成员的。

也就是说：js在创建对象的时候，都有一个叫做\_proto\_的内置属性，用于指向创建它的函数对象的原型对象prototype.

创建对象的不同方法解析

1. 对象字面量

比如：

Var Person = {

Name:”hello”,

Age:27

}

其\_proto\_指向Object.prototype。

所以，其实Object是一个函数也不难理解了。Object、Function都是js自带的函数对象。

1. 构造函数

比如：

Function Person(){};

Var person1 = new Person();

这种形式创建的对象的方式就是通过构造函数创建对象的，其\_proto\_指向的是构造函数的prototype属性所指向的对象。

1. Object.create

比如：

Var person1 = {

Name : ‘hahah’,

Age:26

}

Var person2 = Object.create(person1);

这种情况下，person2的\_proto\_指向person1。在没有Object.create函数的时候，人们大多是这样做的：

Object.create = function(p){

Function F(){};

F.prototype = p;

Return new F();  
}

总结：

其实仔细思考下上面提到的三种创建对象的方法，追究其本质，不难发现，最根本的还是利用构造函数再通过new来创建对象。所谓的对象字面量也不过是语法糖而已，本质上还是一样的。所以，函数真不愧是js中的一等公民。