prototype和_proto_

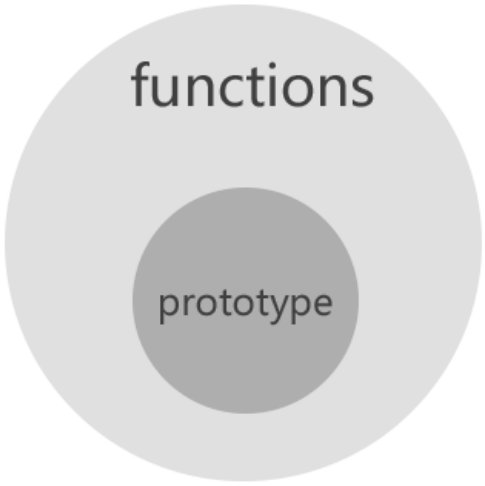
关于对象的prototype和*proto*概念如下:

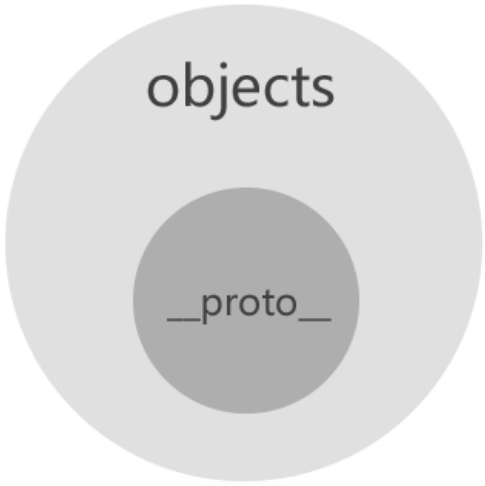
prototype是函数的一个属性（每个函数都有一个prototype属性），这个属性是一个指针，指向一个对象。它是显示修改对象的原型的属性。*_proto_*是一个对象拥有的内置属性（每个对象都有一个*_proto_*属性），是JS内部使用寻找原型链的属性。

这就是为什么实例可以调用到其构造函数原型方法的原因。下面具体来看看：

1、prototype和__proto__的区别

▶ prototype和__proto__的区别





* prototype是函数才有的属性

* __proto__是每个对象都有的属性

* 但__proto__不是一个规范属性，只是部分浏览器实现了此属性，对应的标准属性是[[Prototype]]

注：大多数情况下，__proto__可以理解为“**构造器的原型**”，即：

__proto__ === constructor.prototype

（通过Object.create()创建的对象不适用此等式，图2有说明）

@水乙

```
var a = {};
console.log(a.prototype); //undefined
console.log(a.__proto__);  //Object {}

var b = function(){}
console.log(b.prototype); //b {}
console.log(b.__proto__);  //function() {}
```

2、__proto__属性指向谁

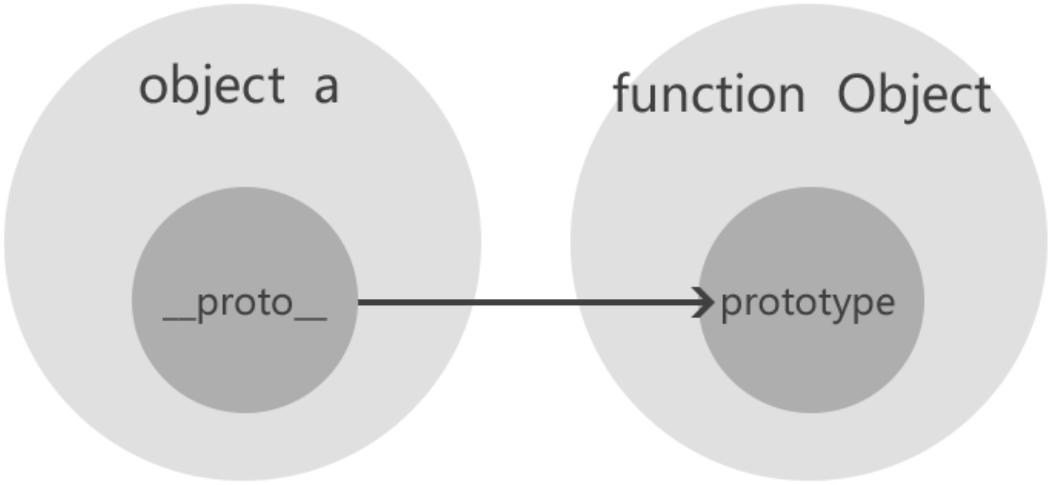
▶ __proto__属性指向谁？

__proto__属性指向谁？

__proto__的指向取决于对象创建时的实现方式。以下图表列出了三种常见方式创建对象后，__proto__分别指向谁。

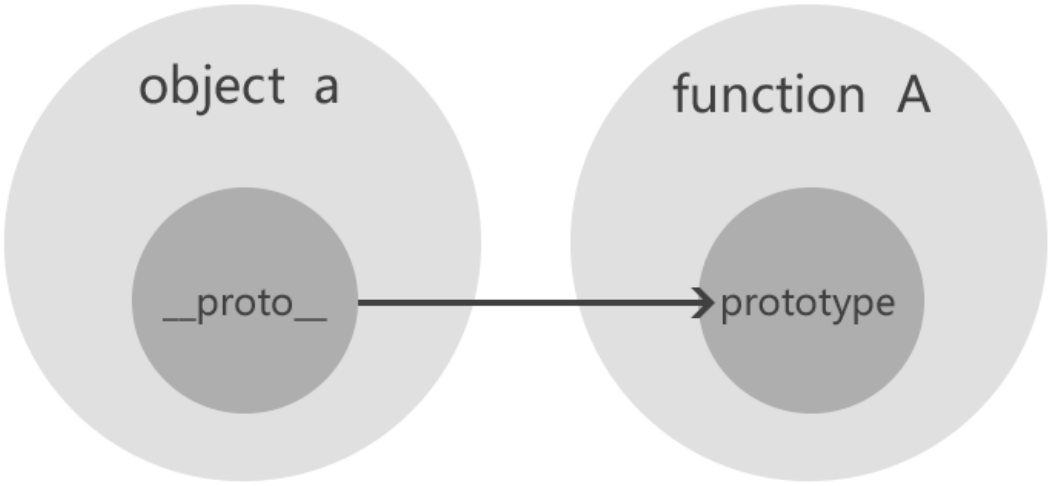
1、字面量方式

```
var a ={};
```



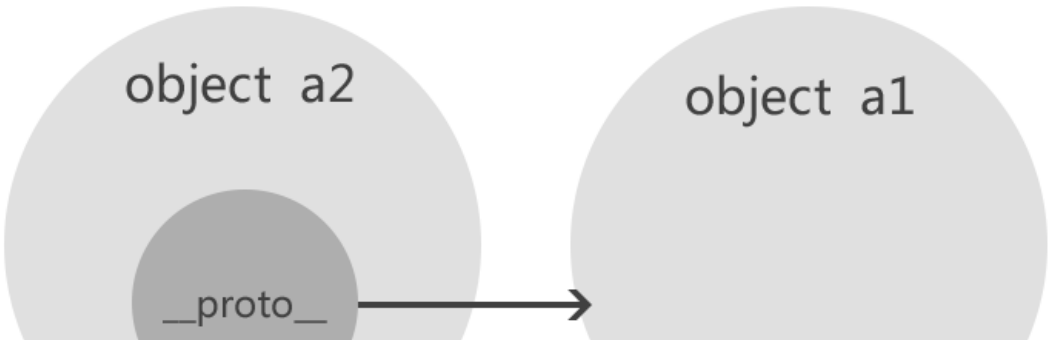
2、构造器方式

```
var A = function(){};
var a = new A();
```



3、Object.create方式

```
var a1 = {}
var a2 = Object.create(a1);
```



@水乙

```
/*1、字面量方式*/
var a = {};
console.log(a.__proto__); //Object {}
console.log(a.__proto__ === a.constructor.prototype); //true

/*2、构造器方式*/
var A = function(){}; var a = new A();
console.log(a.__proto__); //A {}
console.log(a.__proto__ === a.constructor.prototype); //true

/*3、Object.create()方式*/
var a1 = {a:1}
var a2 = Object.create(a1);
console.log(a2.__proto__); //Object {a: 1}
console.log(a2.__proto__ === a2.constructor.prototype); //false（此处即为图1中的例外情况）
```

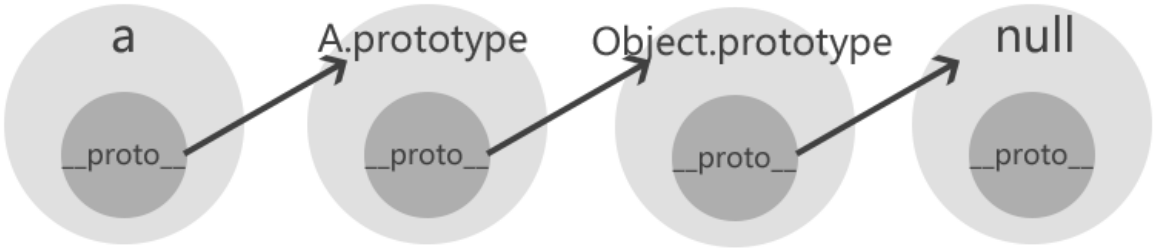
三、什么是原型链

▶ 什么是原型链？

由于__proto__是任何对象都有的属性，而js里万物皆对象，所以会形成一条__proto__连起来的链条，递归访问__proto__必须最终到头，并且值是 null。

当js引擎查找对象的属性时，先查找对象本身是否存在该属性，如果不存在，会在原型链上查找，但不会查找自身的prototype

```
var A = function(){};
var a = new A();
```



@水乙

```
var A = function(){};
var a = new A();
console.log(a.__proto__); //A {}（即构造器function A 的原型对象）
console.log(a.__proto__.__proto__); //Object {}（即构造器function Object 的原型对象）
console.log(a.__proto__.__proto__.__proto__); //null
```