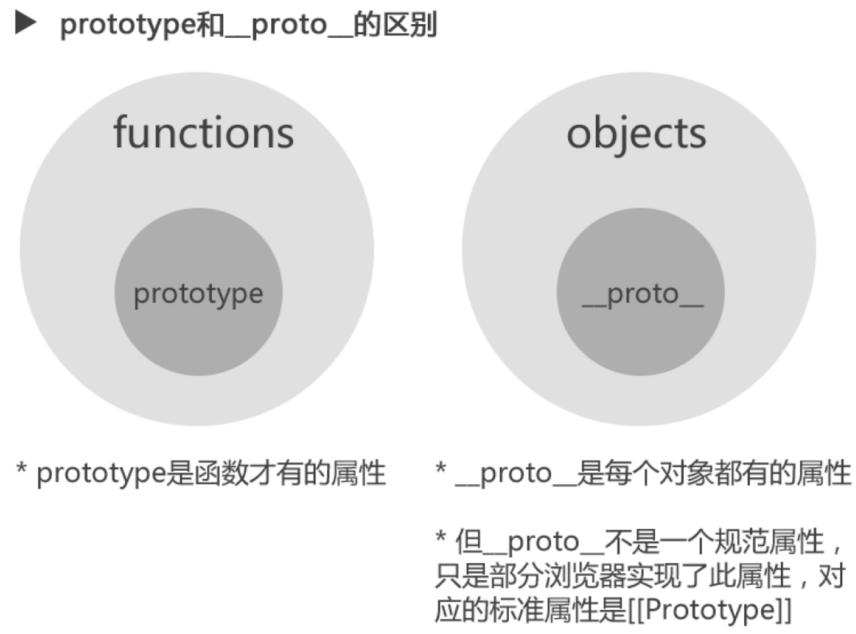
JavaScript 的原型对象与原型链

JavaScript 的原型对象与原型链
- 一、prototype 和 **proto** 的区别:
- 二、例子
- 三、构造函数
- 四、原型规则
- 五、js完整原型链

对于新人来说，JavaScript 的原型是一个很让人头疼的事情，一来 prototype 容易与 **proto** 混淆，

一、prototype 和 **proto** 的区别:




```
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() {}
```

结果：

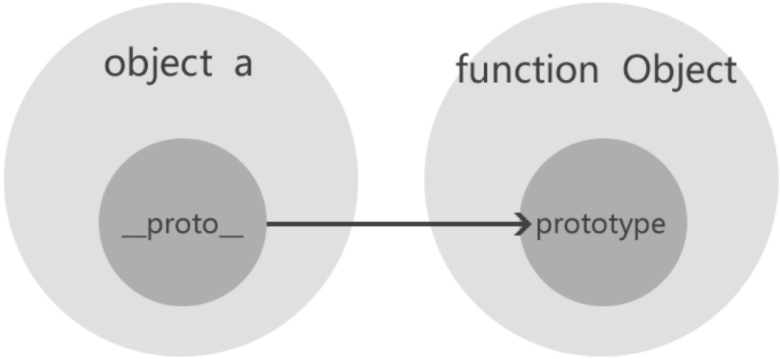

var a = {};
console.log(a.prototype); //undefined
console.log(a.__proto__); //Object {}
var b = function(){}
console.log(b.prototype); //b {}
console.log(b.__proto__); //function() {}
undefined
{constructor: ƒ, __defineGetter__: ƒ, __defineSetter__: ƒ, hasOwnProperty: ƒ, __lookupGetter__: ƒ, …}
{constructor: ƒ}
ƒ () { [native code] }

▶ __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。

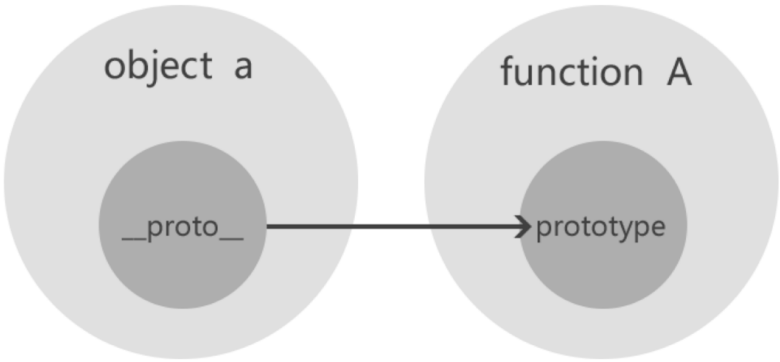
1、字面量方式

var a ={};



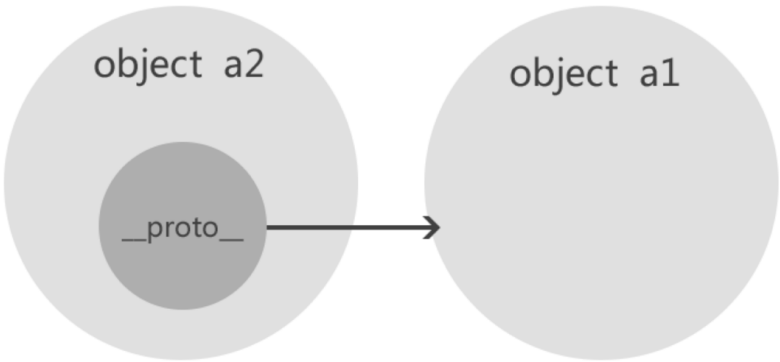
2、构造器方式

```
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__ : ", a.__proto__);  // Object {}
console.log("a.__proto__ === a.constructor.prototype: ",
a.__proto__ === a.constructor.prototype); // true
```

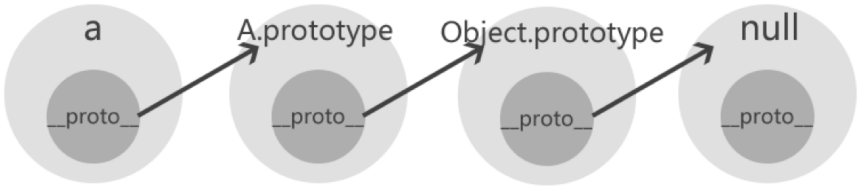
```
/*2、构造器方式*/
var A = function(){};
var a2 = new A();
console.log("a2.__proto__: ", a2.__proto__); // A {}
console.log("a2.__proto__ === a2.constructor.prototype: ",
a2.__proto__ === a2.constructor.prototype); // true

/*3、Object.create()方式*/
var a4 = { a: 1 }
var a3 = Object.create(a4);
console.log("a3.__proto__: ", a3.__proto__); // Object {a:
1}
console.log("a3.__proto__ === a3.constructor.prototype: ",
a3.__proto__ === a3.constructor.prototype); // false（此处
即为图1中的例外情况）
```

▶ 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。

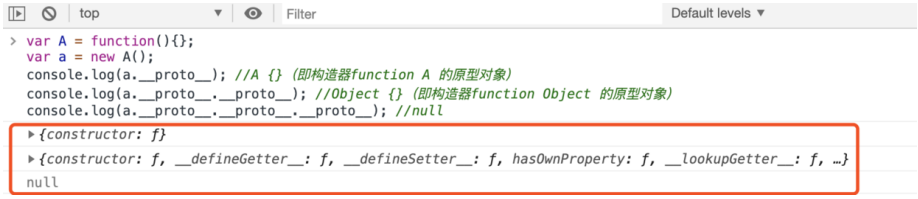
当js引擎查找对象的属性时，先查找对象本身是否存在该属性，如果不存在，会在原型链上查找，但不会查找自身的prototype

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
```



二、例子

- 写一个原型链继承的例子

```
//动物
function Animal() {
    this.food = 'something'
}
Animal.prototype.eat = function() {
    console.log('animals eat' + this.food);
}
//dog
function Dog() {
    this.bark = function() {
        console.log('dog bark');
    }
}
Dog.prototype = new Animal();
//哈士奇
let hashiqi = new Dog();
```

- 封装 DOM 查询: 封装的应用

```
//封装
function Elem(id) {
    this.elem = document.getElementById(id);
}
Elem.prototype.html = function(val) {
    let elem = this.elem;
    if(val) {
        elem.innerHTML = val;
        return this; //用于修改html内容后链式操作
    } else {
        return elem.innerHTML;
    }
};
Elem.prototype.on = function(type,fn) {
    let elem = this.elem;
    elem.addEventListener(type,fn);
    return this;  //链式操作
};
let div = new Elem('did');
console.log(div.html());
```

链式操作:

```
div.html('<p>click</p>').on('click',function() {
    alert('click');
}).html('<span>hehe</span>');
```

- 描述new一个对象的过程

```
function Foo(name,age) {
    this.name = name;
    this.age = age;
    this.class = '3-1'
}
let f1 = new Foo('Tom',18);
//1、创建一个新对象；
//2、this指向这个新对象；
//3、执行代码，即对this赋值；
//4、返回this；
```

***new** 的过程：*

- var f1 = new Foo('Tom',18) 将参数传进去，函数中的 this 会变成空对象；
- this.name = name;this.age = age;this.class = '3-1' 为赋值；return this 为实际的运行机制；
- return 之后赋值给 f1，f1 具备了 f1.name = Tom、f1.age = 18、f1.class = '3-1'；

三、构造函数

- *var a = {} 其实是 var a = new Object() 的语法糖 (a 的构造函数是 Object 函数)*
- *var a = [] 其实是 var a = new Array() 的语法糖 (a 的构造函数是 Array 函数)*
- *function Foo(){...} 其实是 var Foo = new Function(...) (Foo 的构造函数是 Function 函数)*
- 使用 instanceof 判断一个函数是否是一个引用类型变量的构造函数 (判断一个变量是否为“数组” variables instanceof Array)

```
var arr = [];
arr instanceof Array;    //true
//instanceof 可以在继承关系中用来判断一个实例是否属于它的父类型。在多层继承关系中，instanceof 运算符同样适用
//用法: variables instanceof Father
```

四、原型规则

- 所有的 *引用类型*（数组、对象、函数）都具有对象特性，即可自由扩展属性（除了“null”）；

```
var obj = {};
obj.a = 10;
var arr = [1,2];
arr.a = 100;
function fo() {};
fo.a = 100;
```

- 所有的 *引用类型*（数组、对象、函数）都有一个 `__proto__` 属性(隐式原型），属性值是一个普通的对象；

- 所有的 *函数*，都有一个 `prototype` 属性(显式原型），属性值也是一个普通的对象
- 所有的 *引用类型*（数组、对象、函数），`__proto__` 属性值(隐式原型属性）指向它的构造函数的 `prototype` 属性值;
- js引擎查找对象属性时，先从自身开始找，如果没有，会通过原型向上一级原型中查找，以此类推（即在原型链中查找）;
- 循环对象自身的属性:

```
var selfProp;
for (selfProp in f) {
    //虽然高级浏览器已经在 for in 中屏蔽来自原型的属性；
    //但还是建议加上这个判断，保证代码壮硕行；
    if (f.hasOwnProperty(selfProp)) {
        console.log(selfProp);
    }
}
```

*if (f.hasOwnProperty(item)) 中遍历 f 时，判断遍历中的 item, 是否可以通过 hasOwnProperty 验证，通过则表明它是 f **自身的属性**，未通过则表明 是 f 通过原型获得的属性;*

五、js完整原型链

JavaScript Object Layout [Hursh Jain/mollypages.org]