1.prototype是函数的一个属性，是函数的原型对象，只能被函数调用

只要创建一个函数，这个函数就自动获得了prototype属性

//

const obj = {}

console.log(obj.prototype) // undefined

//

function fun() {}

console.log(fun.prototype) // [Object Object]--返回的是对象--这也能看出来prototype是 //函数的原型对象

console.log(JSON.stringify(fun.prototype))//{}--这也能看出来prototype是函数的原型对象

fun.prototype.name = 'prototype是函数的属性，本质是函数的原型对象'

console.log(JSON.stringify(fun.prototype)) // {"name": "prototype是函数的属性，本质是函 //数的原型对象"}

2.可以通过new来实例化函数的对象即fun.prototype--所有原型属性是实例化对象所共享的

//

function fun(){}

fun.prototype.get = function(value){// 这样get就是fun的属性，是函数属性

return value

}

怎么调用fun的get属性呢？

属性需要通过对象来调用，那么就可以通过new构造出fun的实例化对象，通过该对象就可以调用fun的get函数

那么就可以这样理解：prototype只能被函数调用用来表示或其实质就是函数的原型对象

var objFun1 = new fun()

alert(objFun1.get('hello,prototype原型对象')) // hello,prototype原型对象

var objFun2 = new fun()

alert(objFun2.get('依然调用了fun的函数属性get')) //依然调用了fun的函数属性get

在次利用new构造出了fun的实例化对象objFun2，且这个对象依然能调用get这个属性函数；

这就说明prototype赋予的属性函数get能够被fun的所有实例化对象调用

所以说再次说明了prototype是函数的原型对象

3.原型链：

//

function Fn(){}

Fn.prototype.name = 'zhouqi'

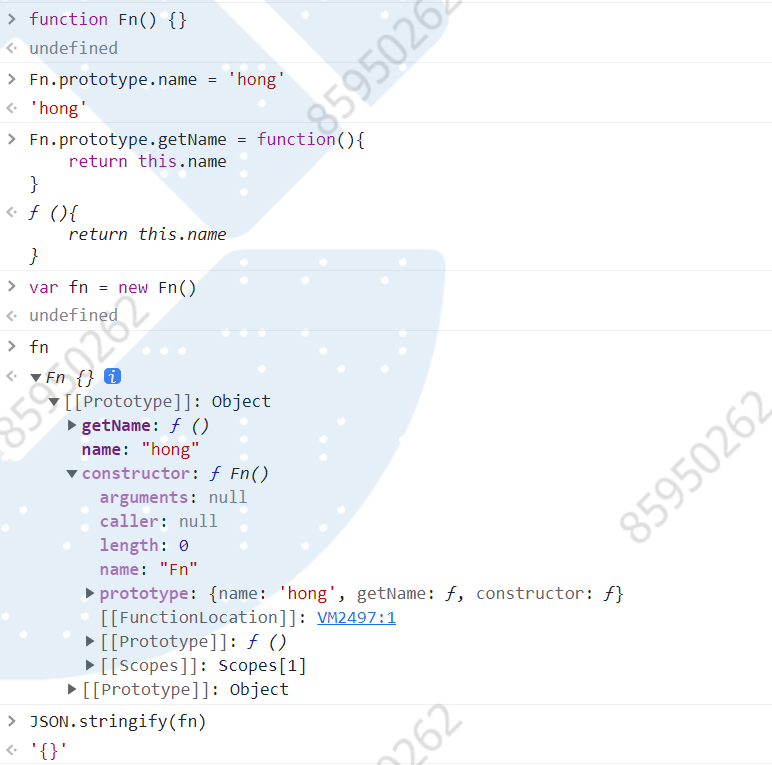
Fn.prototype.getName = function(){

return this.name

}

var fn = new Fn()

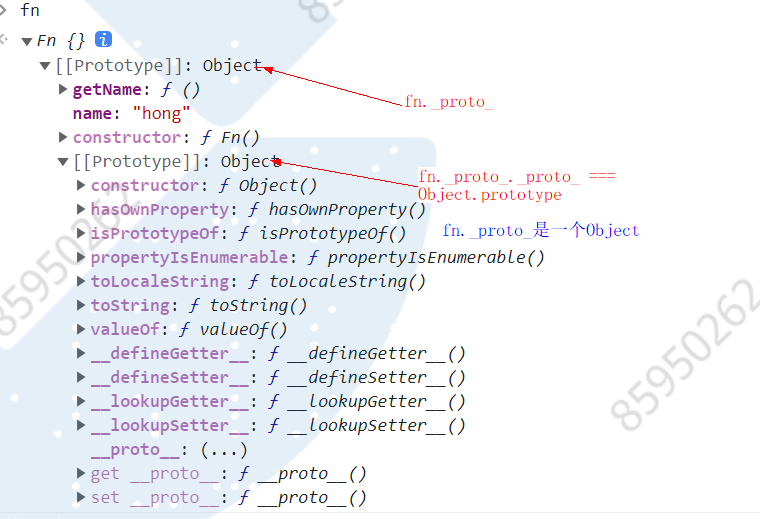
console.log(fn)



这个时候对象fn.constructor.prototype = Fn.prototype

但fn是一个空对象

//

console.log(fn.name) // ‘hong’--虽然fn是空对象，但是仍然打印出了name属性，这就 //用到了原型链--当访问实例对象fn自身没有的属性和方法时，就会 //沿着这个对象的\_proto\_（下图红色箭头）（也从图里看出指向的是 //一个对象）一直去找我们要的属性或方法直到找到

//Object.prototype

//如果找到就返回,没有找到则报错;

// 即fn.\_proto\_ = Fn.prototype;fn.\_proto\_.\_proto\_ = Object.prototype;

//(但是Object.\_proto\_()一般不使用，使用Object.getPrototypeOf方法)

fn.name = ‘name’

console.log(fn)



4.特别指出：

　Array.prototype是一个数组

　String.prototype是一个字符串

　Object.prototype是一个对象

这三种prototype的例子不像构造函数的prototype一样