Js原型及js继承

说明：凡继承类，本篇例子均采用父级Parent，子级Child；

特别说明：本篇总结为个人心得总结。如格式错乱，请使用正版offic2013.

作者：陶福荣

版权所有，非作者本人同意，不得上传于网络。

目录

[一、Js原型 4](#_Toc521690204)

[1.js原型概念 4](#_Toc521690205)

[2.作用： 4](#_Toc521690206)

[二、原型链 5](#_Toc521690207)

[1.示意图 5](#_Toc521690208)

[2.原理说明 6](#_Toc521690209)

[3.拓展方法 6](#_Toc521690210)

[1）instanceof 6](#_Toc521690211)

[2）for in 遍历 6](#_Toc521690212)

[三、继承 7](#_Toc521690213)

[1.原型链继承 7](#_Toc521690214)

[2.类式继承（借用构造函数继承） 7](#_Toc521690215)

[3.组合继承（原型链及借用构造函数） 8](#_Toc521690216)

[4.原型式继承 8](#_Toc521690217)

[5.寄生式继承 9](#_Toc521690218)

[6.寄生式组合继承 10](#_Toc521690219)

[7.ES6 class类继承 11](#_Toc521690220)

[附加：面向对象 11](#_Toc521690221)

[1.面向对象概念 11](#_Toc521690222)

[2.对象的几个方法总结 12](#_Toc521690223)

[2）对象查询方法 12](#_Toc521690224)

# 一、Js原型

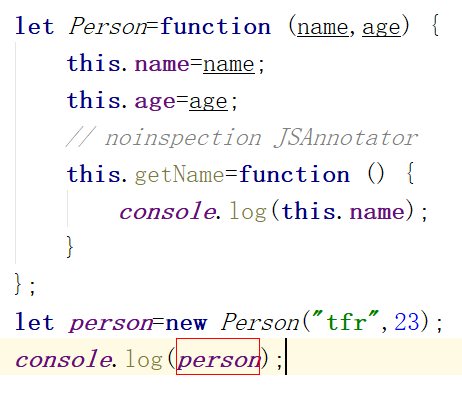
## 1.js原型概念

js原型也是对象，所有的函数都有prototype属性，prototype指向的是一个空对象，这个对象就是当前函数的显示原型。每一个实例对象都有一个\_\_proto\_\_的属性，即隐式原型，该属性指向的是构造函数的显示原型。

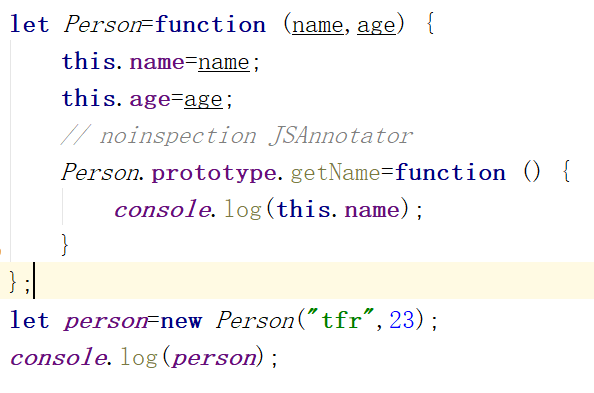
## 2.作用：

继承，简化代码的编写，提高复用性；

Exp:



上图实例说明，在构造函数中直接定义的方法，会在调用其实例化对象时直接被赋予。这样会导致多次调用实例化对象时，方法的重复，导致代码的性能下降。所以我们将方法直接定义在构造函数的显示原型，即prototype当中。代码如下：

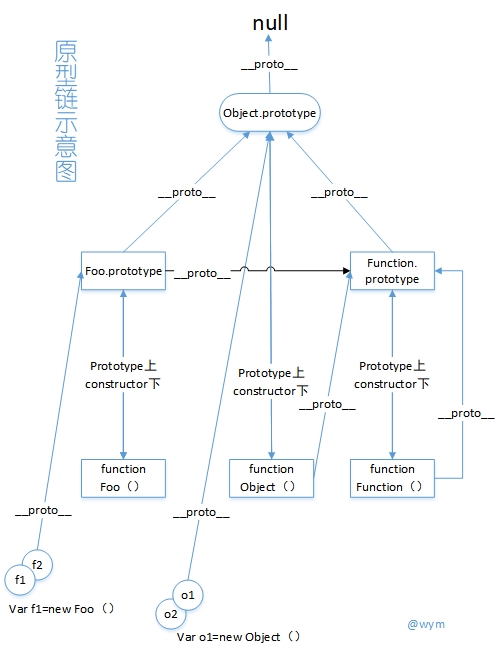


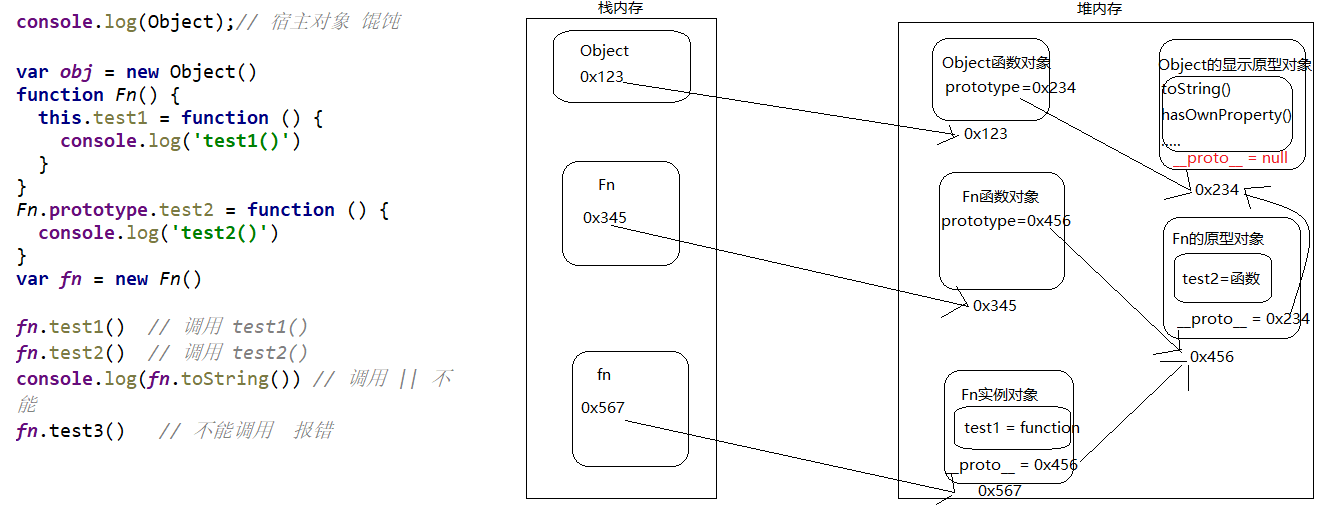


上述图例中的代码，我们将getName方法定义在了实例函数的显示原型里面。（这里我将定义的函数直接放在了构造函数里面，大部分情况实质是放在外面的。这么做的目的提高可阅读性，使代码明确区分成块，无其他作用）。打印的结果我们可以看到，没有带getName方法，这就是原型的好处，这种方法也被成为原型式继承。

# 二、原型链

## 1.示意图





原型链示意图

## 2.原理说明

当使用对象的属性时，先在自身找，如果有就直接使用；如果没有就沿着\_\_proto\_\_这一条链上找，直到找到为Object的原型上，有则返回对应的值，没有则返回underfined。

## 3.拓展方法

### 1）instanceof

A instanceof B, 检查A是否是B的原型；

### 2）for in 遍历

For in 遍历会同时遍历到对象[数组]的属性和对象[数组]原型的属性；为了不使其遍历到原型的属性，则做如下设置；

For(var str in obj){

If(obj.hasOwnproperty(str)){

Console.log(str);

}

};

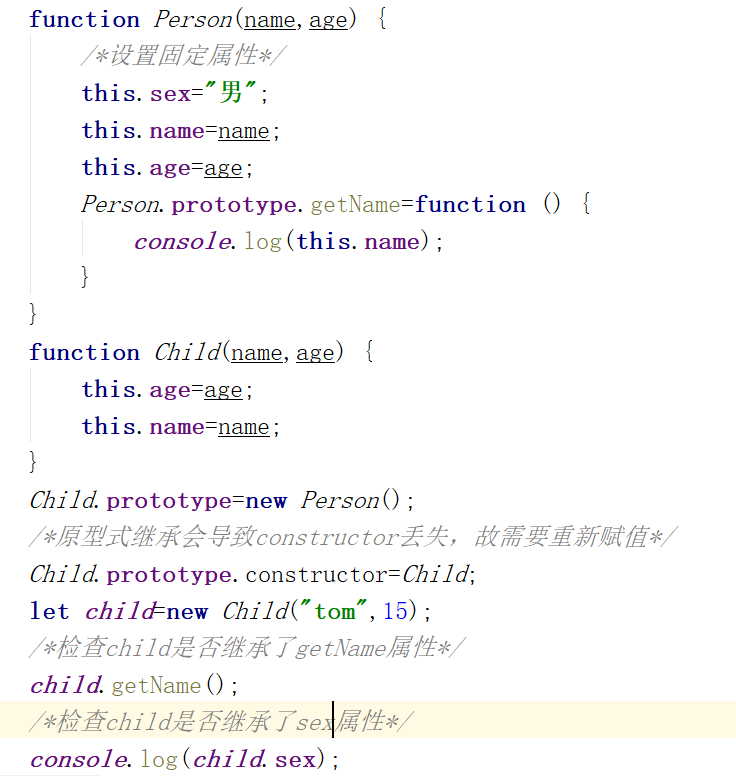
hasOwnProperty：是用来判断一个对象是否有你给出名称的属性或对象。不过需要注意的是，此方法无法检查该对象的原型链中是否具有该属性，当属性是对象本身的一个成员时，返回值为true。

# 三、继承

Js常用的继承方法有两种，原型链继承（对象间的继承）和类继承（函数间的继承）。

## 1.原型链继承

实质就是通过prototype继承；

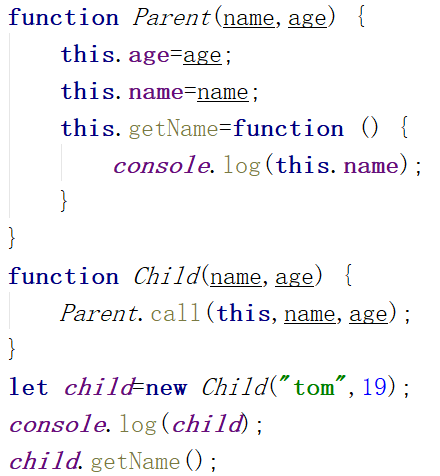




通过上面图示的代码可以看到，可以直接将父级的实例对象赋值赋值给子级，然后子级会继承父级对象的所有属性。但是子级会丢失自己的constructor属性，需要重新赋值。

## 2.类式继承（借用构造函数继承）

借用构造函数就是利用call和apply在子级构造函数中绑定对应的属性和方法（不能继承原型中的方法），使子级构造函数继承父级的属性和方法（不能继承原型中的方法）。





上图代码完美的解释了借用构造函数继承。但是只能继承父级里面的属性和方法，不能继承父级原型里面的属性和方法。

## 3.组合继承（原型链及借用构造函数）

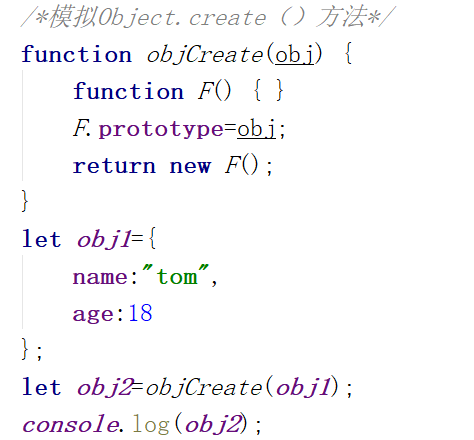
为了方便书写，减少代码量。主要是在Child里面避免写类似this.name=name,这样的代码，只有几个的话体现不出，多的话就会体现出其代码的繁杂了；故使用借用构造函数的方法将子级需要的方法从父级借用过来。但是借用构造函数不能继承父级原型链里面的方法，这不符合我们的将方法写进原型链的方式，所以需要再使用原型链继承。两个组合可进行互补；

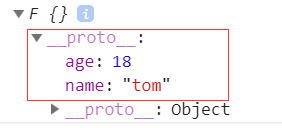
## 4.原型式继承

区分于原型链继承，不能将两者混淆；

原型式继承实质就是通过Object.create(Parent);来实现的；这个方法可以将父级的方法和属性作为子级的原型链存在。并且父级的方法更改后，子级也会受到影响。这个方法创建的子级也可以拥有自己的属性，如果与继承的冲突时，调用的是自己的属性；

当然有版本较老的浏览器并不支持上述方法，我们可以用js代码模拟出来一个。



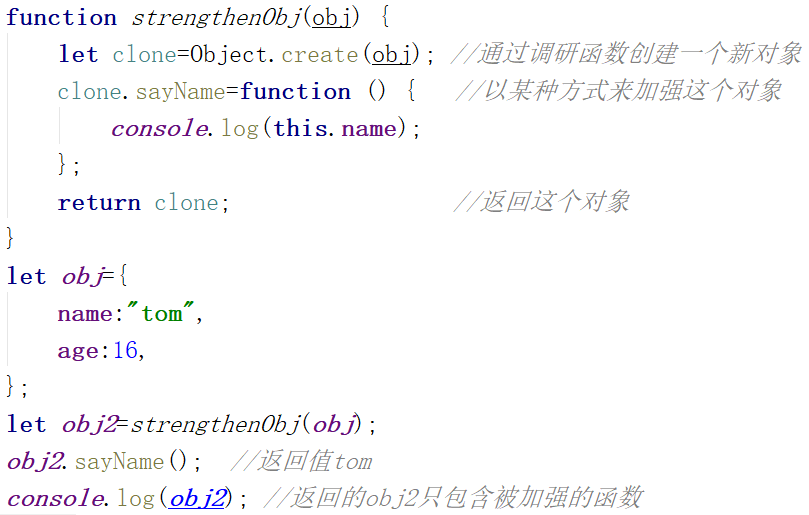


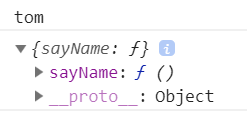
如上图代码所示，当然使用Object.create()是一样的效果。区别于Object.assign(); 这个方法时直接将父级的属性添加到子级本身上，这里不做代码演示。

还有一种继承是子级的隐式原型链继承。Ex:obj2.\_\_proto\_\_=obj1;这种方法也可以将父级的属性写进子级的原型链当中；

## 5.寄生式继承

寄生式继承实质就是通过一个函数，来增强某个对象，然后返回这个对象。





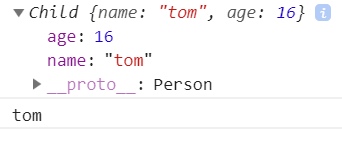
通过上面途中的代码可以看到，最后通过使用寄生式继承方法，子级被加强了，拥有了sayName的函数，而且可以调用到父级的属性，但是本身却不具有父级的属性；

## 6.寄生式组合继承

组合继承是js常用的继承模式，最大的问题就是无论在什么情况下，都会调用两次构造函数；一次是在创建子类原型时，另一次是在子类构造函数内部。

寄生式组合继承使用借用构造函数简化代码，将父级的属性直接复制到自己身上；同时利用寄生式继承，将父级的方法直接赋到自己的原型中，即将父级作为自己的原型属性；可以通过原型链查找父级原型中的方法为自己使用。





从上图中的代码不难看出，寄生式组合实质是将原型链继承封装成了一个函数，同时又使用了借用构造函数继承。同时将寄生式继承，原型链继承，借用构造函数结合在一起，完美发挥了他们各自的优势。

## 7.ES6 class类继承





ES6的class类创建的方法简便，且其定义的方法直接存在于原型中，但是只支持较新版本的浏览器。其大大地简化了代码的写法，且简化了继承，不用额外增设函数，一个super搞定一切。

# 附加：面向对象

## 1.面向对象概念

prototype是javascript实现与管理继承的一种机制，也是面向对象的设计思想。构造函数的原型存储着引用对象的一个指针，该指针指向一个原型对象，对象内部存储着函数的原始属性和方法；我们可以借助prototype属性，可以访问原型内部的属性和方法。

当构造函数被实列化后，所有的实例对象都可以访问构造函数的原型成员，如果在原型中声明一个成员，所有的实列方法都可以共享它。我们可以利用原型及原型链的特性及方法，将属性和方法放在构造函数的原型中，实现面向对象开发。

具体示例请访问后面的网址：<https://www.cnblogs.com/tugenhua0707/p/5068449.html>;

## 2.对象的几个方法总结

1）Object.defineProperty

### 2）对象查询方法

1）Object.getOwnPropertyNames(obj4)

获取对象下所有属性名，以数组的形式返回，包括不可枚举的  
2）Object.getOwnPropertyDescriptor(obj4,"icon")

返回对象中属性的设置对象,实质为defineproperty，里面的设置值；  
3）obj.hasOwnProperty("name");

检查对象中是否含有该属性  
4）Object.getPrototypeOf(person1)==Person.prototype;

检查person返回的对象是否是构造函数的原型  
5）Person.prototype.isPrototypeOf(person1);

判断person是否是Person的实例对象  
6）var keys=Object.keys(Person.prototype);

此方法可以将对的属性名全部返回并放到一个数组当中

【全文完】