

## 05 | 原型链：V8是如何实现对象继承的？

2020-03-26 李兵

《图解 Google V8》

课程介绍 >



讲述：李兵

时长 17:12 大小 11.82M

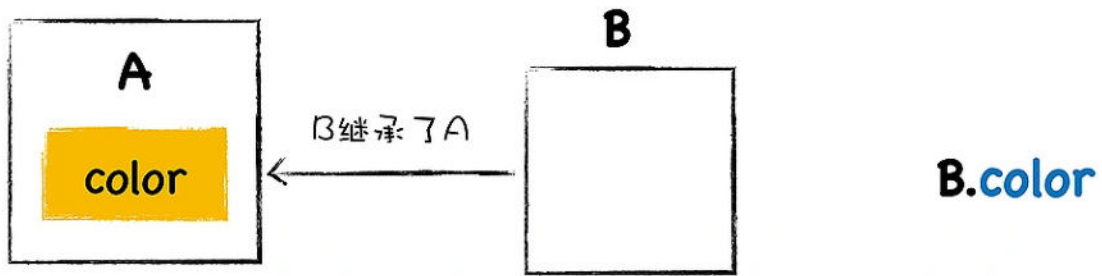


你好，我是李兵。

在前面两节中，我们分析了什么是 JavaScript 中的对象，以及 V8 内部是怎么存储对象的，本节我们继续深入学习对象，一起来聊聊 V8 是如何实现 JavaScript 中对象继承的。

简单地理解，**继承就是一个对象可以访问另外一个对象中的属性和方法**，比如我有一个 B 对象，该对象继承了 A 对象，那么 B 对象便可以直接访问 A 对象中的属性和方法，你可以参考下图：





什么是继承

观察上图，因为 B 继承了 A，那么 B 可以直接使用 A 中的 color 属性，就像这个属性是 B 自带的一样。

不同的语言实现继承的方式是不同的，其中最典型的两种方式是**基于类的设计**和**基于原型继承的设计**。

C++、Java、C# 这些语言都是基于经典的类继承的设计模式，这种模式最大的特点就是提供了非常复杂的规则，并提供了非常多的关键字，诸如 class、friend、protected、private、interface 等，通过组合使用这些关键字，就可以实现继承。

使用基于类的继承时，如果业务复杂，那么你需要创建大量的对象，然后需要维护非常复杂的继承关系，这会导致代码过度复杂和臃肿，另外引入了这么多关键字也给设计带来了更大的复杂度。

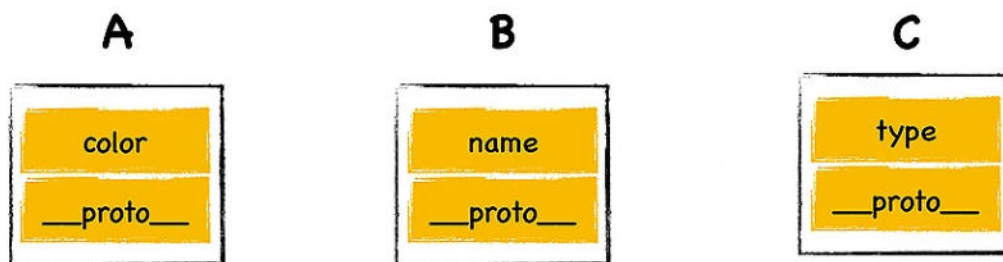
而 JavaScript 的继承方式和其他面向对象的继承方式有着很大差别，JavaScript 本身不提供一个 class 实现。虽然标准委员会在 ES2015/ES6 中引入了 class 关键字，但那只是语法糖，JavaScript 的继承依然和基于类的继承没有一点关系。所以当你看到 JavaScript 出现了 class 关键字时，不要以为 JavaScript 也是面向对象语言了。

JavaScript 仅仅在对象中引入了一个原型的属性，就实现了语言的继承机制，基于原型的继承省去了很多基于类继承时的繁文缛节，简洁而优美。

## 原型继承是如何实现的？

那么，基于原型继承是如何实现的呢？我们参看下图：





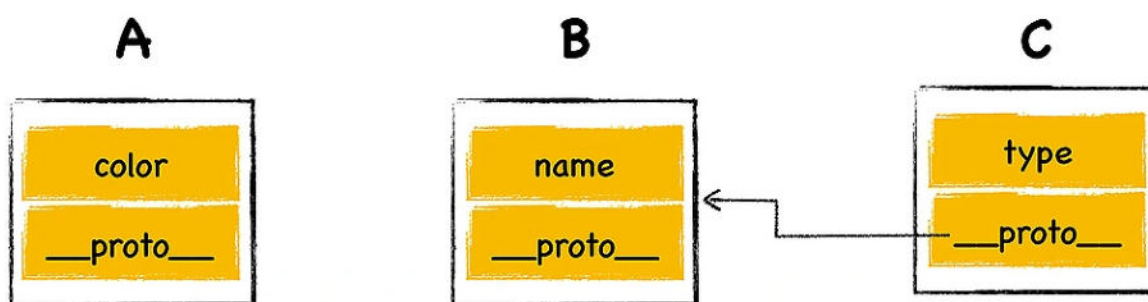
C.type

有一个对象 C，它包含了一个属性“type”，那么对象 C 是可以直接访问它自己的属性 type 的，这点毫无疑问。

怎样让 C 对象像访问自己的属性一样，访问 B 对象呢？

上节我们从 V8 的内存快照看到，JavaScript 的每个对象都包含了一个隐藏属性 \_\_proto\_\_，我们就把该隐藏属性 \_\_proto\_\_ 称之为该**对象的原型 (prototype)**，\_\_proto\_\_ 指向了内存中的另外一个对象，我们就把 \_\_proto\_\_ 指向的对象称为该对象的**原型对象**，那么该对象就可以直接访问其原型对象的方法或者属性。

比如我让 C 对象的原型指向 B 对象，那么便可以利用 C 对象来直接访问 B 对象中的属性或者方法了，最终的效果如下图所示：

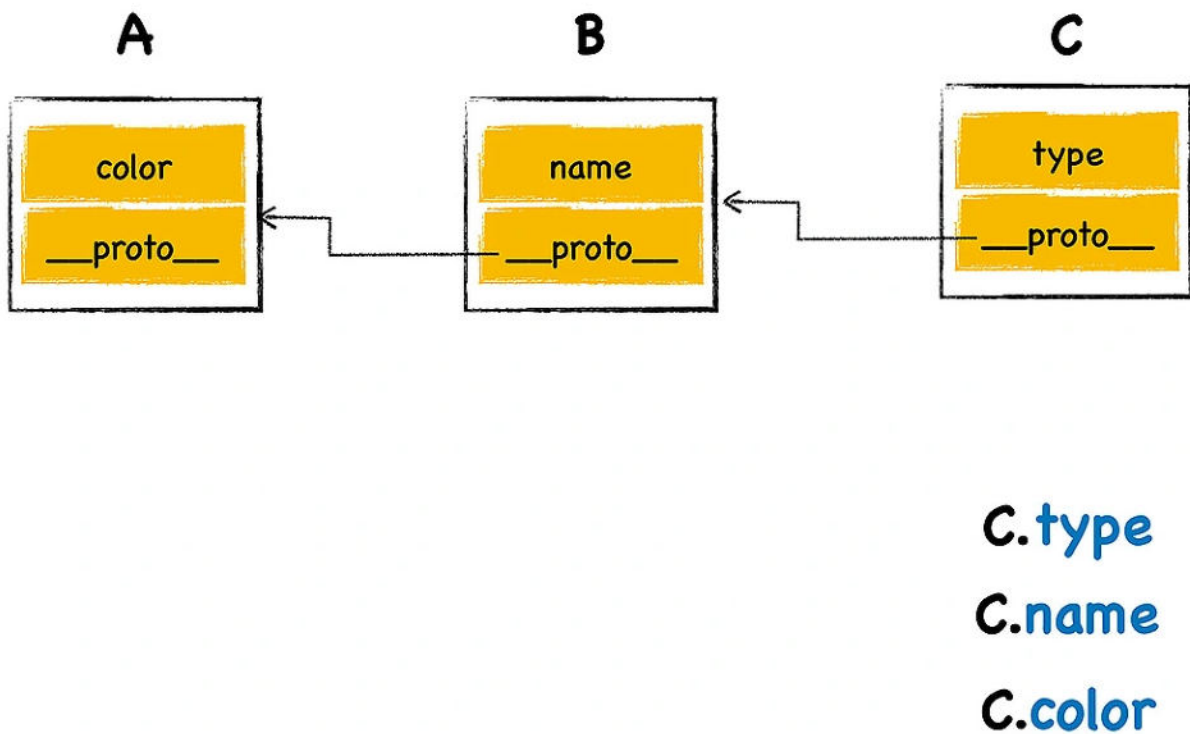


C.type  
C.name



观察上图，当 C 对象将它的 `__proto__` 属性指向了 B 对象后，那么通过对象 C 来访问对象 B 中的 `name` 属性时，V8 会先从对象 C 中查找，但是并没有查找到，接下来 V8 继续在其原型对象 B 中查找，因为对象 B 中包含了 `name` 属性，那么 V8 就直接返回对象 B 中的 `name` 属性值，虽然 C 和 B 是两个不同的对象，但是使用的时候，B 的属性看上去就像是 C 的属性一样。

同样的方式，B 也是一个对象，它也有自己的 `__proto__` 属性，比如它的属性指向了内存中另外一块对象 A，如下图所示：



从图中可以看到，对象 A 有个属性是 `color`，那么通过 `C.color` 访问 `color` 属性时，V8 会先在 C 对象内部查找，但是没有查找到，接着继续在 C 对象的原型对象 B 中查找，但是依然没有查找到，那么继续去对象 B 的原型对象 A 中查找，因为 `color` 在对象 A 中，那么 V8 就返回该属性值。

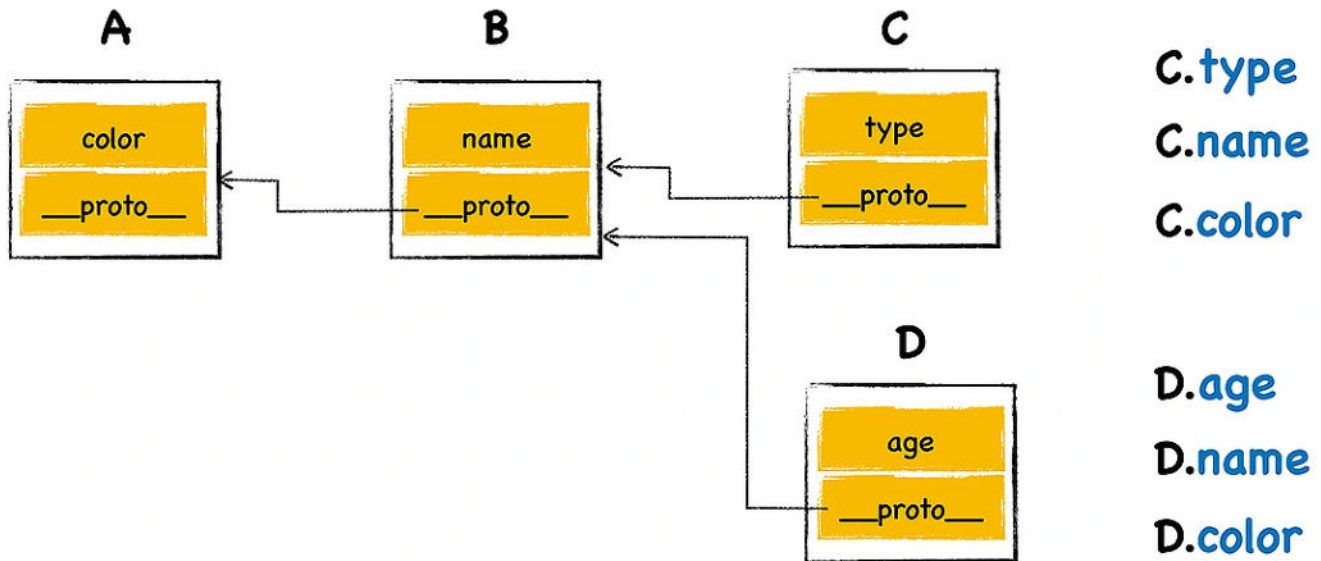
我们看到使用 `C.name` 和 `C.color` 时，给人的感觉属性 `name` 和 `color` 都是对象 C 本身的属性，但实际上这些属性都是位于原型对象上，我们把这个查找属性的路径称为**原型链**，它像一个链条一样，将几个原型链接了起来。

在这里还要注意一点，不要将原型链接和作用域链搞混淆了，作用域链是沿着函数的作用域一级一级来查找变量的，而原型链是沿着对象的原型一级一级来查找属性的，虽然它们的实现方



式是类似的，但是它们的用途是不同的，关于作用域链，我会在《06 | 作用域链：V8 是如何查找变量的？》这节课来介绍。

关于继承，还有一种情况，如果我有另外一个对象 D，它可以和 C 共同拥有同一个原型对象 B，如下图所示：



因为对象 C 和对象 D 的原型都指向了对象 B，所以它们共同拥有同一个原型对象，当我通过 D 去访问 name 属性或者 color 属性时，返回的值和使用对象 C 访问 name 属性和 color 属性是一样的，因为它们是同一个数据。

我们再来回顾下继承的概念：**继承就是一个对象可以访问另外一个对象中的属性和方法，在 JavaScript 中，我们通过原型和原型链的方式来实现了继承特性。**

通过上面的分析，你可以看到在 JavaScript 中的继承非常简洁，就是每个对象都有一个原型属性，该属性指向了原型对象，查找属性时，JavaScript 虚拟机会沿着原型一层一层向上查找，直至找到正确的属性。所以对于 JavaScript 中的原型继承，你不需要把它想得过度复杂。

## 实践：利用 \_\_proto\_\_ 实现继承

了解了 JavaScript 中的原型和原型链继承之后，下面我们就可以通过一个例子，看看原型是怎么应用在 JavaScript 中的，你可以先看下面这段代码：



```
1 var animal = {
2     type: "Default",
3     color: "Default",
4     getInfo: function () {
5         return `Type is: ${this.type}, color is ${this.color}.`
6     }
7 }
8 var dog = {
9     type: "Dog",
10    color: "Black",
11 }
```

在这段代码中，我创建了两个对象 `animal` 和 `dog`，我想让 `dog` 对象继承于 `animal` 对象，那么最直接的方式就是将 `dog` 的原型指向对象 `animal`，应该怎么操作呢？

我们可以通过设置 `dog` 对象中的 `__proto__` 属性，将其指向 `animal`，代码是这样的：

```
1 dog.__proto__ = animal
```

设置之后，我们就可以使用 `dog` 来调用 `animal` 中的 `getInfo` 方法了。

```
1 dog.getInfo()
```

你可以尝试调用下，看看输出的内容。在这里留给你一个关于“this”的小思考题：调用 `dog.getInfo()` 时，`getInfo` 函数中的 `this.type` 和 `this.color` 都是什么值？为什么？

还有一点我们要注意，通常隐藏属性是不能使用 JavaScript 来直接与之交互的。虽然现代浏览器都开了一个口子，让 JavaScript 可以访问隐藏属性 `_proto_`，但是在实际项目中，我们不应该直接通过 `_proto_` 来访问或者修改该属性，其主要原因有两个：

- 首先，这是隐藏属性，并不是标准定义的；
- 其次，使用该属性会造成严重的性能问题。



我们之所以在课程中使用 `_proto_` 属性，主要是为了方便教学，将其他的一些复杂的概念先抛到一边，这样有利于你循序渐进地掌握我们的课程内容，但是我并不推荐你这么做。那应该怎么去正确地设置对象的原型对象呢？

答案是使用构造函数来创建对象，下面我们就来详细解释这个过程。

## 构造函数是怎么创建对象的？

比如我们要创建一个 `dog` 对象，我可以先创建一个 `DogFactory` 的函数，属性通过参数进行传递，在函数体内，通过 `this` 设置属性值。代码如下所示：

```
1 function DogFactory(type,color){
2     this.type = type
3     this.color = color
4 }
```

 复制代码

然后再结合关键字“`new`”就可以创建对象了，创建对象的代码如下所示：

```
1 var dog = new DogFactory('Dog','Black')
```

 复制代码

通过这种方式，我们就把后面的函数称为构造函数，因为通过执行 `new` 配合一个函数，JavaScript 虚拟机便会返回一个对象。如果你没有详细研究过这个问题，很可能对这种操作感到迷惑，为什么通过 `new` 关键字配合一个函数，就会返回一个对象呢？

关于 JavaScript 为什么要采用这种怪异的写法，我们文章最后再来介绍，先来看看这段代码的深层含义。

其实当 V8 执行上面这段代码时，V8 会在背后悄悄地做了以下几件事情，模拟代码如下所示：

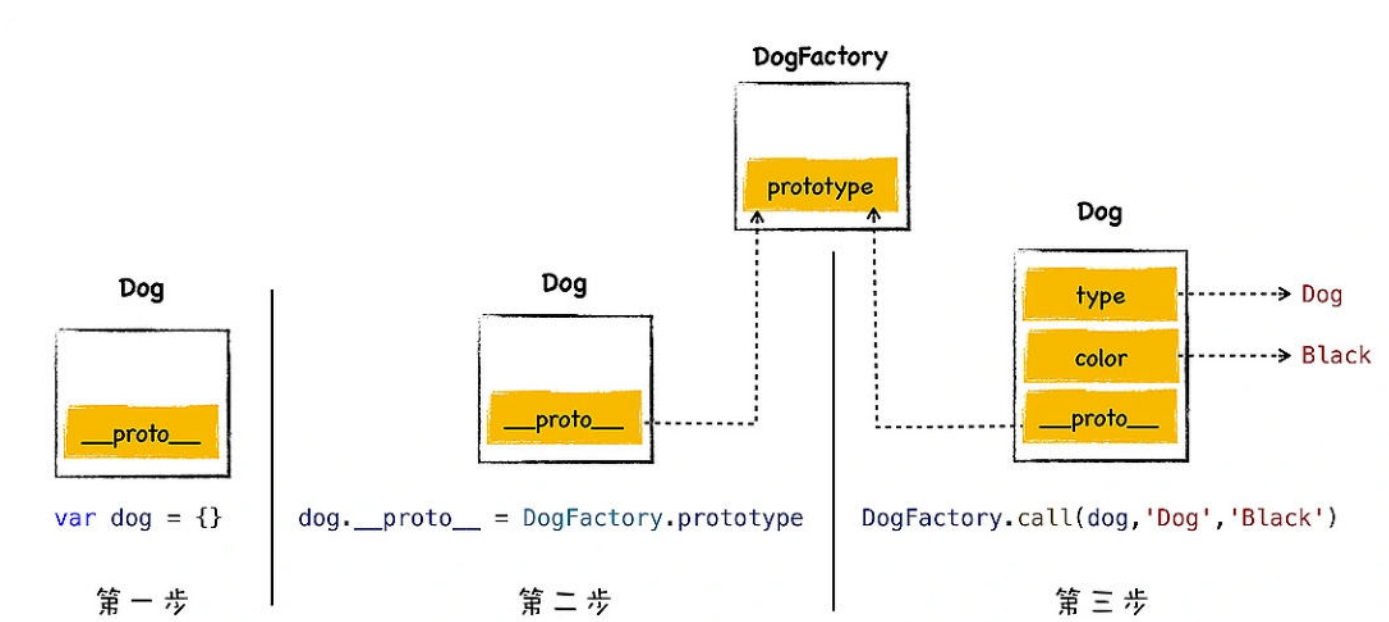
```
1 var dog = {}
2 dog.__proto__ = DogFactory.prototype
3 DogFactory.call(dog,'Dog','Black')
```

 复制代码





为了加深你的理解，我画了上面这段代码的执行流程图：



观察上图，我们可以看到执行流程分为三步：

- 首先，创建了一个空白对象 `dog`；
- 然后，将 `DogFactory` 的 `prototype` 属性设置为 `dog` 的原型对象，这就是给 `dog` 对象设置原型对象的关键一步，我们后面来介绍；
- 最后，再使用 `dog` 来调用 `DogFactory`，这时候 `DogFactory` 函数中的 `this` 就指向了对象 `dog`，然后在 `DogFactory` 函数中，利用 `this` 对对象 `dog` 执行属性填充操作，最终就创建了对象 `dog`。

## 构造函数怎么实现继承？

好了，现在我们可以通过构造函数来创建对象了，接下来我们就看看构造函数是如何实现继承的？你可以先看下面这段代码：

```
1 function DogFactory(type,color){
2     this.type = type
3     this.color = color
4     //Mammalia
5     //恒温
6     this.constant_temperature = 1
7 }
```

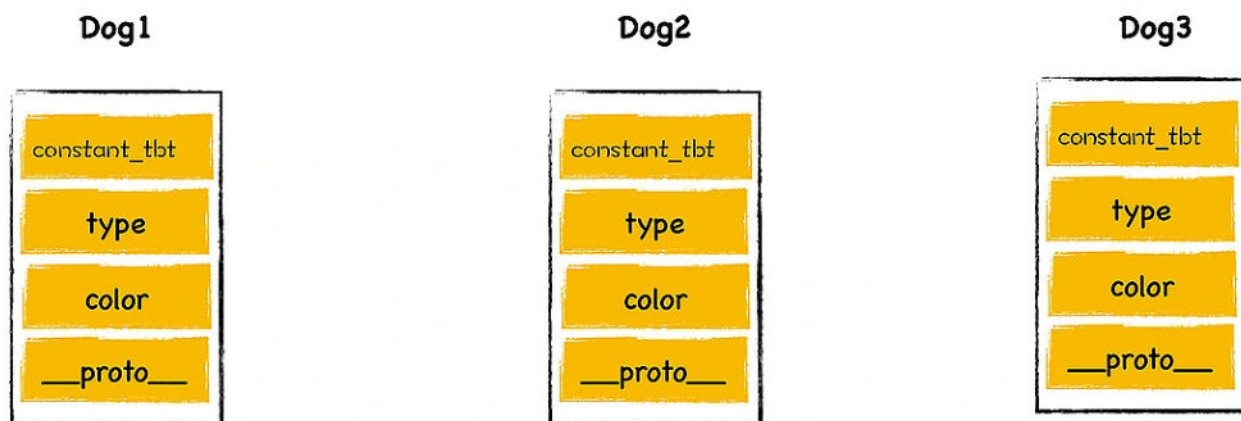
复制代码

☆



```
8 var dog1 = new DogFactory('Dog', 'Black')
9 var dog2 = new DogFactory('Dog', 'Black')
10 var dog3 = new DogFactory('Dog', 'Black')
```

我利用上面这段代码创建了三个 dog 对象，每个对象都占用了一块空间，占用空间示意图如下所示：



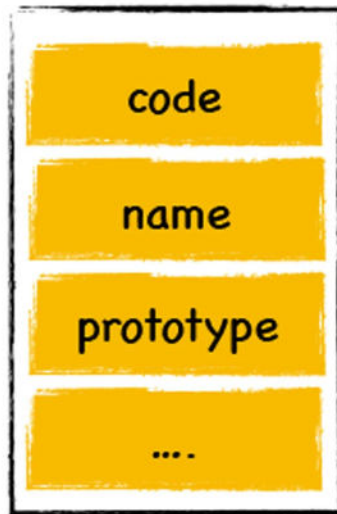
从图中可以看出来，对象 dog1 到 dog3 中的 constant\_temperature 属性都占用了一块空间，但是这是一个通用的属性，表示所有的 dog 对象都是恒温动物，所以没有必要在每个对象中都为该属性分配一块空间，我们可以将该属性设置公用的。

怎么设置呢？

还记得我们介绍函数时提到关于函数有两个隐藏属性吗？这两个隐藏属性就是 name 和 code，其实函数还有另外一个隐藏属性，那就是 prototype，刚才介绍构造函数时我们也提到过。一个函数有以下几个隐藏属性：

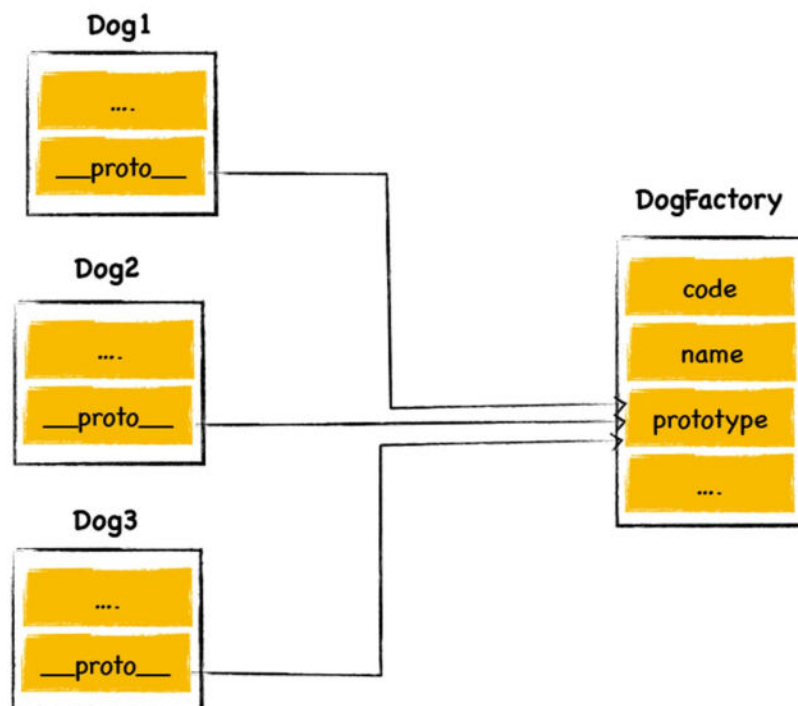


## Function




每个函数对象中都有一个公开的 prototype 属性，当你将这个函数作为构造函数来创建一个新的对象时，新创建对象的原型对象就指向了该函数的 prototype 属性。当然了，如果你只是正常调用该函数，那么 prototype 属性将不起作用。

现在我们知道了新对象的原型对象指向了构造函数的 prototype 属性，当你通过一个构造函数创建多个对象的时候，这几个对象的原型都指向了该函数的 prototype 属性，如下图所示：



这时候我们可以将 `constant_temperature` 属性添加到 `DogFactory` 的 `prototype` 属性上，代码如下所示：

 复制代码

```
1 function DogFactory(type,color){
2     this.type = type
3     this.color = color
4     //Mammalia
5 }
6 DogFactory.prototype.constant_temperature = 1
7 var dog1 = new DogFactory('Dog','Black')
8 var dog2 = new DogFactory('Dog','Black')
9 var dog3 = new DogFactory('Dog','Black')
```

这样我们三个 `dog` 对象的原型对象都指向了 `prototype`，而 `prototype` 又包含了 `constant_temperature` 属性，这就是我们实现继承的正确方式。

## 一段关于 `new` 的历史

现在我们知道 `new` 关键字结合构造函数，就能生成一个对象，不过这种方式很怪异，为什么要这样呢？要了解这背后的原因，我们需要了解一段关于 JavaScript 的历史。

JavaScript 是 Brendan Eich 发明的，那是个“战乱”的时代，各种大公司相互争霸，有 Sun、微软、网景、甲骨文等公司，它们都有推出自己的语言，其中最炙手可热的编程语言是 Sun 的 Java，而 JavaScript 就是这个时候诞生的。当时创造 JavaScript 的目的仅仅是为了让浏览器页面可以动起来，所以尽可能采用简化的方式来设计 JavaScript，所以本质上来说，Java 和 JavaScript 的关系就像雷锋和雷峰塔的关系。

那么之所以叫 JavaScript 是出于市场原因考量的，因为一门新的语言需要吸引新的开发者，而当时最大的开发者群体就是 Java，于是 JavaScript 就蹭了 Java 的热度，事后，这一招被证明的确有效果。

虽然叫 JavaScript，但是其编程方式和 Java 比起来，依然存在着非常大的差异，其中 Java 中使用最频繁的代码就是创建一个对象，如下所示：



 复制代码

```
1 CreateInstance instance = new CreateInstance();
```

当时 JavaScript 并没有使用这种方式来创建对象，因为 JavaScript 中的对象和 Java 中的对象是完全不一样的，因此，完全没有必要使用关键字 `new` 来创建一个新对象的，但是为了进一步吸引 Java 程序员，依然需要在语法层面去蹭 Java 热点，所以 JavaScript 中就被硬生生地强制加入了非常不协调的关键字 `new`，然后使用 `new` 来创造对象就变成这样了：

```
1 var bar = new Foo()
```

 复制代码

Java 程序员看到这段代码时，当然会感到倍感亲切，觉得 Java 和 JavaScript 非常相似，那么使用 JavaScript 也就天经地义了。不过代码形式只是表象，其背后原理是完全不同的。

了解了这段历史之后，我们知道 JavaScript 的 `new` 关键字设计并不合理，但是站在市场角度来说，它的出现又是非常成功的，成功地推广了 JavaScript。

## 总结

好了，今天的主要内容就介绍到这里，下面我们来回顾下。

今天我们的主要目的是介绍清楚 JavaScript 中的继承机制，这涉及到了原型继承机制，虽然基于原型的继承机制本身比较简单，但是在 JavaScript 中，这是通过关键字 `new` 加上构造函数来体现的。这种方式非常绕，且不符合人的直觉，如果直接上来就介绍 `new` 加构造函数是怎么工作的，可能会把你给绕晕了。

于是我先通过每个对象中都有的隐含属性 `__proto__`，来介绍了什么是原型和原型链。V8 为每个对象都设置了一个 `__proto__` 属性，该属性直接指向了该对象的原型对象，原型对象也有自己的 `__proto__` 属性，这些属性串连在一起就成了原型链。

不过在 JavaScript 中，并不建议直接使用 `__proto__` 属性，主要有两个原因。

- 一，这是隐藏属性，并不是标准定义的；
- 二，使用该属性会造成严重的性能问题。

所以，在 JavaScript 中，是使用 `new` 加上构造函数的这种组合来创建对象和实现对象的继承。不过使用这种方式隐含的语义过于隐晦，所以理解起来有点难度。



为什么 JavaScript 中要使用这种怪异的方式来创建对象？为了理解这个问题，我们回顾了一段 JavaScript 的历史。由于当前的 Java 非常流行，基于市场推广的考虑，JavaScript 采取了蹭 Java 热度的策略，在语言命名上使用了 Java 字样，在语法形式上也模仿了 Java。事实上通过这些策略，确实为 JavaScript 带来了市场上的成功。不过你依然要记住，JavaScript 和 Java 是完全两种不同的语言。


## 思考题

我们知道函数也是一个对象，所以函数也有自己的 `__proto__` 属性，那么今天留给你的思考题是：DogFactory 是一个函数，那么“DogFactory.prototype”和“DogFactory.\_\_proto\_\_”这两个属性之间有关联吗？欢迎你在留言区与我分享讨论。

感谢你的阅读，如果你觉得这一讲的内容对你有所启发，也欢迎把它分享给你的朋友。

分享给需要的人，Ta订阅超级会员，你将得 50 元

Ta单独购买本课程，你将得 20 元

 生成海报并分享

 赞 11  提建议

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 04 | 函数表达式：涉及大量概念，函数表达式到底该怎么学？

下一篇 06 | 作用域链：V8是如何查找变量的？



# JVM + NIO + Spring

各大厂面试题及知识点详解

限时免费



## 精选留言 (37)

写留言



张青天

2020-03-26

DogFactory 是 Function 构造函数的一个实例，所以 `DogFactory.__proto__ === Function.prototype`

DogFactory.prototype 是调用 Object 构造函数的一个实例，所以 `DogFactory.prototype.__proto__ === Object.prototype`

因此 `DogFactory.__proto__` 和 `DogFactory.prototype` 没有直接关系

作者回复: 没问题

共 5 条评论 >

91



搞学习

2020-03-26

推荐一篇挺好的文章，结合老师讲的一起看有奇效

<https://juejin.im/post/5cc99fdfe51d453b440236c3>

编辑回复: 优秀





20



sugar

2020-03-26

老师，这几节课看了有关对象，函数这些东西在v8的实现，感觉还不过瘾，想问下老师能否把文中提到的一些v8的实现思路，在文末增加一个链接直接跳转到v8的c++源代码里 具体到文件和行号？

作者回复: 这个专栏定位还是给前端工程师的，所以根本没打算讲源码，源码比想象的复杂太多，光一个原型的实现就做了很多复杂的优化！比如通过隐藏类优化了很多原有的对象结构，所以通过直接修改\_\_proto\_\_会直接破坏现有已经优化的结构，造成严重的性能问题！

另外比如讲作用域的C++实现我觉得也没太大意义，有能力看代码的人结合文档和流程就可以直接去看代码了！

比如编译流程，代码的文档结构 在v8.dev中都有介绍。

共 7 条评论 >



10



mfist

2020-03-26

DogFactory.prototype 是Dog工厂函数实例对象的原型链（`dog = new DogFactory()`），dog实例上面没有属性或方法会去原型链上面寻找。

DogFactory.\_\_proto\_\_ 是函数对象的原型链，`function DogFactory(){}` 另外一种类似实现是 `DogFactory = new Function([arg1, arg2] functionBody)` 所以它应该指向Function.prototype。引用MDN一句话：Function对象继承自Function.prototype属性，它是不能被修改的。`Function.prototype.toString()` 得到 `"function () { [native code] }"`

所以两者是有本质区别的。要说有啥关联性的话，就是`DogFactory.prototype.constructor === DogFactory // true` DogFactory.prototype上面构造函数就是 DogFactory

今日总结

1. 普通对象上面有一个隐藏的\_\_proto\_\_对象，指向自己的原型。当在对象上面访问属性的时候会先在当前对象寻找，如果找不到再去原型链上面寻找。
2. javascript为了蹭到当时java的热度和迎合java程序员，起名为javascript，和模仿了 new Foo() 创建对象的语法（虽然和面向对象创建实例的底层逻辑完全不一样）

...

```
function Foo(){
  this.name = 'foo'
  this.label = 'function'
}
```





```
const foo = new Foo()  
// new Foo执行的内部逻辑如下  
let obj = {}  
obj.__proto__ = Foo.prototype  
let args = [...arguments]  
let result= Foo.call(obj, args)  
if (typeof result === 'object'){  
    return result  
}  
return obj  
...
```

作者回复: 赞



👍 8



若川

2020-05-31

关于思考题，我以前写了一篇文章《面试官问：JS的继承》画了一张图可以很好的回答这个问题。

<https://user-gold-cdn.xitu.io/2019/2/18/169014cf74620047?imageslim>

（极客时间评论不支持图片。只好放个图片链接了）

文章链接 <https://juejin.im/post/5c433e216fb9a049c15f841b>

作者回复: 很赞



👍 6



tomision

2020-03-26

直接使用 \_\_proto\_\_ 属性，会有严重的性能问题。这个点可以详细说说嘛？

作者回复: 隐藏类的优化措施优化过了对象，修改了proto的属性指向，相当于要重建整个隐藏类，必然会影响性能



👍 3



墨灵

2020-03-29

...

```
const someFactory = (key) => {  
  this.key = key  
}  
...
```

试问，someFactory能否成为一个构造函数？

答案是不能，箭头函数在js里也是一个比较特殊的存在，根本没是prototype的属性，自然也没有constructor

作者回复: 赞



👍 2



盖世英雄

2020-03-26

每个函数对象中都有一个公开的 prototype 属性，当你将这个函数作为构造函数来创建一个新的对象时，

这句话中： 都有一个 公开的 prototype属性，‘公开的‘ 是不是写错了？

上文还提到 prototype属性是隐藏的呢？ 还是我理解的不对呢？

作者回复: —proto—是隐藏属性，prototype可是标准定义的



👍 2



罗乾林

2020-03-26

个人理解：new DogFactory 的所有dog实例的\_\_proto\_\_指向 DogFactory.prototype这个对象，DogFactory.\_\_proto\_\_指向函数DogFactory的原型对象，两者之间没直接关系。

望老师指正

作者回复: 这样理解没问题



👍 2



行问

2020-08-01

不建议去使用 prototype 和 \_\_proto\_\_  
(左右都是2个下划线)

可以使用



Object.setPrototypeOf()  
Object.getPrototypeOf()

又或者使用 ES6 的 extends 实现



1



伪装

2020-05-25

Null 设计的初衷是什么 它具体担任了什么样的角色

作者回复: 最初NULL就代表是空, 比如Number (null) , 就会返回一个0, 可以把null看成是c中或者java中的null。

可以根据一个值是否是null, 来判断做什么事情。

但是javascript同时支持原生类型和对象类型, null是一个对象, 那么发明者认为, 对象和原生类型进行默认转换, 会造成很多误解, 并且不容易发现错误, 那么又设计了一个undefined, 用来表示未使用的原始值, 转换为数值时为NaN!

总的来说, 这个设计糟糕的一塌糊涂, 但是我们依然得使用它们

还需要一个类型来表示原生类型的



1



不二

2020-05-19

```
function Animal (name, gender) {  
  this.name = name;  
  this.gender = gender;  
}  
  
let dog = new Animal('狗', '男');  
console.log(dog.name);  
console.log(dog.gender);  
console.log(dog.__proto__ === Animal.prototype);  
console.log(Animal.__proto__ === Function.prototype);  
console.log(Animal.prototype.__proto__ === Object.prototype);  
console.log(Function.prototype.__proto__ === Object.prototype);  
console.log(Object.prototype.__proto__);
```





1

**安辰**

2020-04-16

请问：

`__proto__` 也是隐藏属性吗

除了name、code、prototype还有哪些隐藏属性？

作者回复: `__proto__` 不是隐藏属性，但是它不是标准中定义的属性

共 3 条评论 &gt;



1

**七月有风**

2020-04-08

class 中的方法为什么要放在构造函数prototype上。而不是放在构造函数中？

作者回复: 放在prototype属于全局的，只要继承了prototype的类都可以共同拥有该方法，放在构造函数中就属于当前对象了，具体放在什么地方要看具体需求了



1

**天天**

2020-04-03

文中老师说到，js不是通过类的去实现继承的，便认为js不是面向对象的语言，这有点懵，不是说js万物皆对象(或复合对象类型)，对象皆源于原型咩，还请大家帮我捋一捋这一概念

共 6 条评论 &gt;



1

**luckyone**

2020-03-27

这节课对我来说很简单，以前用lua实现过面向对象，虚表啥的

作者回复: 你是高手



1

**杨越**

2020-03-26

老师你开发的时候用let还是var？我看PyCharm写js用var会被建议用let替代

作者回复: var能不用就不用了，历史的产物，用let和const



共 3 条评论 >



1



伏枫

2020-03-26

DogFactory.prototype”和“DogFactory.\_proto\_”这两个属性，按我的理解，它们是同一个东西，指向的是同一个对象。

作者回复: 不是一个东西，函数作为对象他得拥有一个proto

函数作为一个构造函数，它得拥有一个prototype，

这两个属性的用途是不同的

共 4 条评论 >



1



哎哟迪奥

2021-11-24

对象的\_\_proto\_\_ 等于他的构造函数的 prototype，函数是个对象也是个函数，所以它同时拥有 \_\_proto\_\_ 和 prototype，而纯对象只有\_\_proto\_\_。\_\_proto\_\_表示当前对象的构造函数的 prototype，prototype 表示是另一个对象。



Yvan

2021-10-21

老师太牛了，请收下我的膝盖

