

JavaScript对象：面向对象还是基于对象？

2019-01-29 winter

《重学前端》

课程介绍 >



讲述：winter

时长 15:23 大小 14.10M



你好，我是 winter。

与其它的语言相比，JavaScript 中的“对象”总是显得不那么合群。

一些新人在学习 JavaScript 面向对象时，往往也会有疑惑：

- 为什么 JavaScript（直到 ES6）有对象的概念，但是却没有像其他的语言那样，有类的概念呢；
- 为什么在 JavaScript 对象里可以自由添加属性，而其他的语言却不能呢？

甚至，在一些争论中，有人强调：JavaScript 并非“面向对象的语言”，而是“基于对象的语言”。这个说法一度流传甚广，而事实上，我至今遇到的持有这一说法的人中，无一能够回答“如何定义面向对象和基于对象”这个问题。



实际上，基于对象和面向对象两个形容词都出现在了 JavaScript 标准的各个版本当中。

我们可以先看看 JavaScript 标准对基于对象的定义，这个定义的具体内容是：“语言和宿主的基础设施由对象来提供，并且 JavaScript 程序即是一系列互相通讯的对象集合”。

这里的意思根本不是表达弱化的面向对象的意思，反而是表达对象对于语言的重要性。

那么，在本篇文章中，我会尝试让你去理解面向对象和 JavaScript 中的面向对象究竟是什么。

什么是面向对象？

我们先来说说什么是对象，因为翻译的原因，中文语境下我们很难理解“对象”的真正含义。事实上，Object（对象）在英文中，是一切事物的总称，这和面向对象编程的抽象思维有互通之处。

中文的“对象”却没有这样的普适性，我们在学习编程的过程中，更多是把它当作一个专业名词来理解。

但不论如何，我们应该认识到，对象并不是计算机领域凭空造出来的概念，它是顺着人类思维模式产生的一种抽象（于是面向对象编程也被认为是：更接近人类思维模式的一种编程范式）。

那么，我们先来看看在人类思维模式下，对象究竟是什么。

对象这一概念在人类的幼儿期形成，这远远早于我们编程逻辑中常用的值、过程等概念。在幼年期，我们总是先认识到某一个苹果能吃（这里的某一个苹果就是一个对象），继而认识到所有的苹果都可以吃（这里的所有苹果，就是一个类），再到后来我们才能意识到三个苹果和三个梨之间的联系，进而产生数字“3”（值）的概念。

在《面向对象分析与设计》这本书中，Grady Booch 替我们做了总结，他认为，从人类的认知角度来说，对象应该是下列事物之一：

1. 一个可以触摸或者可以看见的东西；
2. 人的智力可以理解的东西；



3. 可以指导思考或行动（进行想象或施加动作）的东西。

有了对象的自然定义后，我们就可以描述编程语言中的对象了。在不同的编程语言中，设计者也利用各种不同的语言特性来抽象描述对象，最为成功的流派是使用“类”的方式来描述对象，这诞生了诸如 C++、Java 等流行的编程语言。

而 JavaScript 早年却选择了一个更为冷门的方式：原型（关于原型，我在下一篇文章会重点介绍，这里你留个印象就可以了）。这是我在前面说它不合群的原因之一。

然而很不幸，因为一些公司政治原因，JavaScript 推出之时受管理层之命被要求模仿 Java，所以，JavaScript 创始人 Brendan Eich 在“原型运行时”的基础上引入了 new、this 等语言特性，使之“看起来更像 Java”。

在 ES6 出现之前，大量的 JavaScript 程序员试图在原型体系的基础上，把 JavaScript 变得更像是基于类的编程，进而产生了很多所谓的“框架”，比如 PrototypeJS、Dojo。

事实上，它们成为了某种 JavaScript 的古怪方言，甚至产生了一系列互不相容的社群，显然这样做的收益是远远小于损失的。

如果我们从运行时角度来谈论对象，就是在讨论 JavaScript 实际运行中的模型，这是由于任何代码执行都必定绕不开运行时的对象模型。

不过，幸运的是，从运行时的角度看，可以不必受到这些“基于类的设施”的困扰，这是因为任何语言运行时类的概念都是被弱化的。

首先我们来了解一下 JavaScript 是如何设计对象模型的。

JavaScript 对象的特征

在我看来，不论我们使用什么样的编程语言，我们都先应该去理解对象的本质特征（参考 Grandy Booch《面向对象分析与设计》）。总结来看，对象有如下几个特点。


- 对象具有唯一标识性：即使完全相同的两个对象，也并非同一个对象。
- 对象有状态：对象具有状态，同一对象可能处于不同状态之下。



- 对象具有行为：即对象的状态，可能因为它的行为产生变迁。

我们先来看第一个特征，对象具有唯一标识性。一般而言，各种语言的对象唯一标识性都是用内存地址来体现的，对象具有唯一标识的内存地址，所以具有唯一的标识。

所以，JavaScript 程序员都知道，任何不同的 JavaScript 对象其实是互不相等的，我们可以看下面的代码，o1 和 o2 初看是两个一模一样的对象，但是打印出来的结果却是 false。


 复制代码

```
1    var o1 = { a: 1 };
2    var o2 = { a: 1 };
3    console.log(o1 == o2); // false
```

关于对象的第二个和第三个特征“状态和行为”，不同语言会使用不同的术语来抽象描述它们，比如 C++ 中称它们为“成员变量”和“成员函数”，Java 中则称它们为“属性”和“方法”。

在 JavaScript 中，将状态和行为统一抽象为“属性”，考虑到 JavaScript 中将函数设计成一种特殊对象（关于这点，我会在后面的文章中详细讲解，此处先不用细究），所以 JavaScript 中的行为和状态都能用属性来抽象。

下面这段代码其实就展示了普通属性和函数作为属性的一个例子，其中 o 是对象，d 是一个属性，而函数 f 也是一个属性，尽管写法不太相同，但是对 JavaScript 来说，d 和 f 就是两个普通属性。

 复制代码

```
1    var o = {
2        d: 1,
3        f() {
4            console.log(this.d);
5        }
6    };
```

所以，总结一句话来看，在 JavaScript 中，对象的状态和行为其实都被抽象为了属性。如果你用过 Java，一定不要觉得奇怪，尽管设计思路有一定差别，但是二者都很好地表现了对对象的基本特征：标识性、状态和行为。



在实现了对象基本特征的基础上, 我认为, JavaScript 中对象独有的特色是: 对象具有高度的动态性, 这是因为 JavaScript 赋予了使用者在运行时为对象添改状态和行为的能力。

我来举个例子, 比如, JavaScript 允许运行时向对象添加属性, 这就跟绝大多数基于类的、静态的对象设计完全不同。如果你用过 Java 或者其它别的语言, 肯定会产生跟我一样的感受。

下面这段代码就展示了运行时如何向一个对象添加属性, 一开始我定义了一个对象 o, 定义完成之后, 再添加它的属性 b, 这样操作是完全没问题的。

 复制代码

```
1   var o = { a: 1 };
2   o.b = 2;
3   console.log(o.a, o.b); //1 2
```

为了提高抽象能力, JavaScript 的属性被设计成比别的语言更加复杂的形式, 它提供了数据属性和访问器属性 (getter/setter) 两类。

JavaScript 对象的两类属性

对 JavaScript 来说, 属性并非只是简单的名称和值, JavaScript 用一组特征 (attribute) 来描述属性 (property) 。

先来说第一类属性, 数据属性。它比较接近于其它语言的属性概念。数据属性具有四个特征。

- value: 就是属性的值。
- writable: 决定属性能否被赋值。
- enumerable: 决定 for in 能否枚举该属性。
- configurable: 决定该属性能否被删除或者改变特征值。

在大多数情况下, 我们只关心数据属性的值即可。

第二类属性是访问器 (getter/setter) 属性, 它也有四个特征。



- getter: 函数或 undefined, 在取属性值时被调用。
- setter: 函数或 undefined, 在设置属性值时被调用。
- enumerable: 决定 for in 能否枚举该属性。
- configurable: 决定该属性能否被删除或者改变特征值。

访问器属性使得属性在读和写时执行代码, 它允许使用者在写和读属性时, 得到完全不同的值, 它可以视为一种函数的语法糖。

我们通常用于定义属性的代码会产生数据属性, 其中的 writable、enumerable、configurable 都默认为 true。我们可以使用内置函数 `getOwnPropertyDescriptor` 来查看, 如以下代码所示:

 复制代码

```
1  var o = { a: 1 };
2  o.b = 2;
3  //a和b皆为数据属性
4  Object.getOwnPropertyDescriptor(o,"a") // {value: 1, writable: true, enumerab
5  Object.getOwnPropertyDescriptor(o,"b") // {value: 2, writable: true, enumerab
```

我们在这里使用了两种语法来定义属性, 定义完属性后, 我们用 JavaScript 的 API 来查看这个属性, 我们可以发现, 这样定义出来的属性都是数据属性, writable、enumerable、configurable 都是默认值为 true。

如果我们要想改变属性的特征, 或者定义访问器属性, 我们可以使用 `Object.defineProperty`, 示例如下:

 复制代码

```
1  var o = { a: 1 };
2  Object.defineProperty(o, "b", {value: 2, writable: false, enumerable: false,
3  //a和b都是数据属性, 但特征值变化了
4  Object.getOwnPropertyDescriptor(o,"a"); // {value: 1, writable: true, enumerab
5  Object.getOwnPropertyDescriptor(o,"b"); // {value: 2, writable: false, enumerab
6  o.b = 3;
7  console.log(o.b); // 2
```



这里我们使用了 `Object.defineProperty` 来定义属性，这样定义属性可以改变属性的 `writable` 和 `enumerable`。

我们同样用 `Object.getOwnPropertyDescriptor` 来查看，发现确实改变了 `writable` 和 `enumerable` 特征。因为 `writable` 特征为 `false`，所以我们重新对 `b` 赋值，`b` 的值不会发生变化。

在创建对象时，也可以使用 `get` 和 `set` 关键字来创建访问器属性，代码如下所示：

```
1     var o = { get a() { return 1 } };
2
3     console.log(o.a); // 1
```

 复制代码

访问器属性跟数据属性不同，每次访问属性都会执行 `getter` 或者 `setter` 函数。这里我们的 `getter` 函数返回了 `1`，所以 `o.a` 每次都得到 `1`。

这样，我们就理解了，实际上 JavaScript 对象的运行时是一个“属性的集合”，属性以字符串或者 `Symbol` 为 `key`，以数据属性特征值或者访问器属性特征值为 `value`。

对象是一个属性的索引结构（索引结构是一类常见的数据结构，我们可以把它理解为一个能够以比较快的速度用 `key` 来查找 `value` 的字典）。我们以上面的对象 `o` 为例，你可以想象一下“`a`”是 `key`。

`{writable:true,value:1,configurable:true,enumerable:true}` 是 `value`。我们在前面的类型课程中，已经介绍了 `Symbol` 类型，能够以 `Symbol` 为属性名，这是 JavaScript 对象的一个特色。

讲到了这里，如果你理解了对应的特征，也就不难理解我开篇提出来的问题。

你甚至可以理解为什么会有“JavaScript 不是面向对象”这样的说法了。这是由于 JavaScript 的对象设计跟目前主流基于类的面向对象差异非常大。



可事实上，这样的对象系统设计虽然特别，但是 JavaScript 提供了完全运行时的对象系统，这使得它可以模仿多数面向对象编程范式（下一节课我们会给你介绍 JavaScript 中两种面向

对象编程的范式：基于类和基于原型），所以它也是正统的面向对象语言。

JavaScript 语言标准也已经明确说明，JavaScript 是一门面向对象的语言，我想标准中能这样说，正是因为 JavaScript 的高度动态性的对象系统。

所以，我们应该在理解其设计思想的基础上充分挖掘它的能力，而不是机械地模仿其它语言。

结语

要想理解 JavaScript 对象，必须清空我们脑子里“基于类的面向对象”相关的知识，回到人类对对象的朴素认知和面向对象的语言无关基础理论，我们就能够理解 JavaScript 面向对象设计的思路。

在这篇文章中，我从对象的基本理论出发，和你理清了关于对象的一些基本概念，分析了 JavaScript 对象的设计思路。接下来又从运行时的角度，介绍了 JavaScript 对象的具体设计：具有高度动态性的属性集合。

很多人在思考 JavaScript 对象时，会带着已有的“对象”观来看问题，最后的结果当然就是“剪不断理还乱”了。

在后面的文章中，我会继续带你探索 JavaScript 对象的一些机制，看 JavaScript 如何基于这样的动态对象模型设计自己的原型系统，以及你熟悉的函数、类等基础设施。

你还知道哪些面向对象语言，它们的面向对象系统是怎样的？请留言告诉我吧！

猜你喜欢

Vue 开发实战

从 0 开始搭建大型 Vue 项目

[戳此试读](#)



唐金州
一点资讯前端技术专家
Ant Design Vue 作者



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 44

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 JavaScript类型：关于类型，有哪些你不知道的细节？

下一篇 JavaScript对象：我们真的需要模拟类吗？

学习推荐

JVM + NIO + Spring

各大厂面试题及知识点详解

限时免费



精选留言 (90)

写留言



风清不扬

2019-01-29

php 是世界上最好的编程语言

共 10 条评论 >

134



37°C^boy

2019-01-29

这篇讲的思路太好了，追本溯源，娓娓道来。在这里不光能学到知识活着重温知识，还有关于学习和讲授的方法lun



👍 94



hhk

2019-01-30

关键点在于是否可以在运行时动态改变对象

结合文章通篇看下来，觉得 JS 的 OO 和他基于类的 OO 不同之处，在于 JS 可以在运行时修改对象，而 class based 的类只能预先全部定义好，我们并不能在运行时动态修改类。在我理解来说，条条大路通罗马，面向对象是罗马，class based 是一条路，prototype based 是另一条路。而且 Symbol 的出现，暴露出了许多内置接口，让 JS 又在这条路上走了更远一些。（以前上学背面向对象的三个特征，封装，继承，多态，现在看一下突然觉得很对。。。)

像我这种年轻一点的前端，很可能就只是相对熟悉 JS 而已，对于其他语言更多都是道听途说，计算机基础也比较薄弱，所以用来比较其实比较难。

只知道 class based 的大概有 Java, C++, C#, Python

另外，好奇 Symbol 是怎么实现的，希望老师以后能大概讲讲啦。暂时只想到这些，其他的还在消化

共 3 条评论 >

👍 54



米斯特菠萝

2019-01-30

好像winter老师没有回答过同学的提问，是我没看见吗？

共 4 条评论 >

👍 39



张汉桂-东莞

2019-01-31

```
var o = { get a() { return 1 } };  
console.log(o.a); // 1
```

看到这段我就感到值了。我目前在用layui框架，根据layui文档的描述，只有执行var form = layui.form;这一句时才会下载form.js这个文件，我一直没能理解。这篇文章解除了我的疑惑，原来调用getter时可以不写括号()。谢谢老师！

共 3 条评论 >

👍 34



王小宏music

2019-03-19

```
var o = { get a() { return 1 } }
```

console.log(o.a); // 1

肯定有同学对这里有疑问，解释一下吧，这里边应用到了ES6的getter,setter属性，为啥o.a，没写小括号呢？因为每次访问get，函数返回为1，作为一个value返回的，而非Obj中，调用某个方法，所以才没写成Obj.fun()的方式，另外老师下边有一句总结，很容易遗漏，每次访问，访问器属性，都会执行get,set方法

共 1 条评论 >

👍 24



bitmxy

2019-01-29

JS的設計者原本是個Lisp程序員而且不怎麼喜歡Java面向對象，所以採用了原型。在當時基於原型比基於類的做法要靈活很多。

共 2 条评论 >

👍 23



如斯

2019-02-13

有个疑惑哈，讲道理symbolObj对象也是对象。也可以调用symbolObj.toString方法（symbolObj.toString() // "Symbol(a)"）。

但为什么会 symbolObj+" 会报错呢。

Uncaught TypeError: Cannot convert a Symbol value to a string at <anonymous>:1:10

作者回复: 这个问题问的很好，是这个东西在作怪：

```
typeof Object(Symbol("a"))[Symbol.toPrimitive]()
```

共 5 条评论 >

👍 22



咩啊

2019-02-12

请问“运行时”是指什么？一开始我以为是指“程序执行的时候”这一时间状态，但是在正文倒数第三段又有“但是 JavaScript 提供了完全运行时的对象系统，这使得它可以模仿多数面向对象编程范式”这一句，这里的“完全运行时”是什么意思？我上网查了一下，好像没有比较符合的解析。

共 4 条评论 >

👍 15



(;3」 ∠) 🏆

2019-02-02

太难了(눈_눈)

完全没看懂面向对象，有没有更加数学一点更加精确一点的定义啊。

(◡ _ ◡)



**庖丁**

2019-01-29

我们应该在理解其设计思想的基础上充分挖掘它的能力，而不是机械地模仿其它语言。



12

**next_one**

2019-10-09

我理解的，重学前端专栏的意义是，从语言使用者的角度，转到语言实现者的角度，来看待语言的发展，通过对比其他语言，来阐述js语言本身的特性。重学的意义在于，多数开发者是语言使用者，而没有从语言实现者的角度，对语言本身有思考。有一种“不识庐山真面目，只缘身在此山中”的感觉。简单说，就是大多数人就是用了，没有想过（或者没有能力）去了解语言本身的来龙去脉。



10

**CC**

2019-01-29

我暂时接触的编程有限，JavaScript 是我接触的第一个面向对象语言。

由于缺少对其他语言的了解，winter 老师在文中的横向对比，感觉能让我更容易理解 JavaScript 的设计思路，以及 Object 这么设计的原因。

关于ECMAScript 2015 加入的“类”，其实它并不是 JavaScript 新增的面向对象模式，它主要是语法糖的作用，只是一种特殊的函数，背后仍然是基于原型的设计思路。



10

**费马**

2019-02-01

这才理解数值属性和访问器属性！赞



9

**朋友**

2019-02-22

getter setter实际应用的例子有哪些？ vue的数据，视图双向绑定算吗？

作者回复: vue2.0确实用到了这个。





Scorpio
2019-01-29

感觉js出es6后，和java更像了。。

共 1 条评论 >



7



白嗣
2019-01-30

老师是否按照犀牛书的顺序讲解😁



6



Smallfly
2019-01-29

基于类的面向对象使用的是继承，而 Javascript 更像是组合。



6



Mirror
2020-06-30

做了几年的前端开发，算是老学生不算优等生。越来越讨厌JS目前的状态，被赋予了太多的责任和设计思想，借鉴了太多其他语言，新特性、新玩法层出不穷。前端开发者太累了，真的太累了。一门被拿来玩玩的脚本语言承受了太多它不该承受的...

共 1 条评论 >



2



宋捷
2020-06-28

什么时候Symbol作为属性的键去使用呢？实际的应用场景还比较模糊想不到，有同学和老师简单提示下吗？

共 2 条评论 >



2

