

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Grady 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。

```
var o1 = { a: 1 };
var o2 = { a: 1 };
```

```
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。

```
var o = {  
  d: 1,  
  f() {  
    console.log(this.d);  
  }  
};
```

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。

```
var o = { a: 1 };  
o.b = 2;  
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：

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

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：

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

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

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

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 Grady 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。

```
var o1 = { a: 1 };
var o2 = { a: 1 };
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。

```
var o = {
  d: 1,
  f() {
    console.log(this.d);
  }
};
```

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。

```
var o = { a: 1 };
o.b = 2;
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：

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

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：

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

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

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

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，如果你理解了对对象的特征，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

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标准对于基于对象的定义，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Grady 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。

```
var o1 = { a: 1 };
var o2 = { a: 1 };
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。

```
var o = {
  d: 1,
  f() {
    console.log(this.d);
  }
};
```

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。

```
var o = { a: 1 };
o.b = 2;
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：

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

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：

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

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

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

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



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标准对于基于对象的定义，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。

```
var o1 = { a: 1 };
var o2 = { a: 1 };
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。

```
var o = {
  d: 1,
  f() {
    console.log(this.d);
  }
};
```

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。

```
var o = { a: 1 };
o.b = 2;
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，如下代码所示：

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

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：

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

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

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

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



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标准对于基于对象的定义，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。

```
var o1 = { a: 1 };
var o2 = { a: 1 };
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。

```
var o = {
  d: 1,
  f() {
    console.log(this.d);
  }
};
```

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。

```
var o = { a: 1 };
o.b = 2;
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：

```
var o = { a: 1 };
o.b = 2;
//a和b皆为数据属性
```

```
Object.getOwnPropertyDescriptor(o,"a") // {value: 1, writable: true, enumerable: true, configurable: true}
Object.getOwnPropertyDescriptor(o,"b") // {value: 2, writable: true, enumerable: true, configurable: true}
```

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：

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

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

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

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



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标准对于基于对象的定义，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。

```
var o1 = { a: 1 };
var o2 = { a: 1 };
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。

```
var o = {
  d: 1,
  f() {
    console.log(this.d);
  }
};
```

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。

```
var o = { a: 1 };
o.b = 2;
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，如下代码所示：

```
var o = { a: 1 };
o.b = 2;
//a和b皆为数据属性
```

```
Object.getOwnPropertyDescriptor(o,"a") // {value: 1, writable: true, enumerable: true, configurable: true}
Object.getOwnPropertyDescriptor(o,"b") // {value: 2, writable: true, enumerable: true, configurable: true}
```

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：

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

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

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

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



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标准对于基于对象的定义，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。

```
var o1 = { a: 1 };
var o2 = { a: 1 };
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。

```
var o = {
  d: 1,
  f() {
    console.log(this.d);
  }
};
```

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。

```
var o = { a: 1 };
o.b = 2;
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：

```
var o = { a: 1 };
o.b = 2;
//a和b皆为数据属性
```

```
Object.getOwnPropertyDescriptor(o,"a") // {value: 1, writable: true, enumerable: true, configurable: true}
Object.getOwnPropertyDescriptor(o,"b") // {value: 2, writable: true, enumerable: true, configurable: true}
```

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：

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

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

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

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



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标准对于基于对象的定义，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。

```
var o1 = { a: 1 };
var o2 = { a: 1 };
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。

```
var o = {
  d: 1,
  f() {
    console.log(this.d);
  }
};
```

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。

```
var o = { a: 1 };
o.b = 2;
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：

```
var o = { a: 1 };
o.b = 2;
//a和b皆为数据属性
```



```
Object.getOwnPropertyDescriptor(o,"a") // {value: 1, writable: true, enumerable: true, configurable: true}
Object.getOwnPropertyDescriptor(o,"b") // {value: 2, writable: true, enumerable: true, configurable: true}
```

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：

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

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

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

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