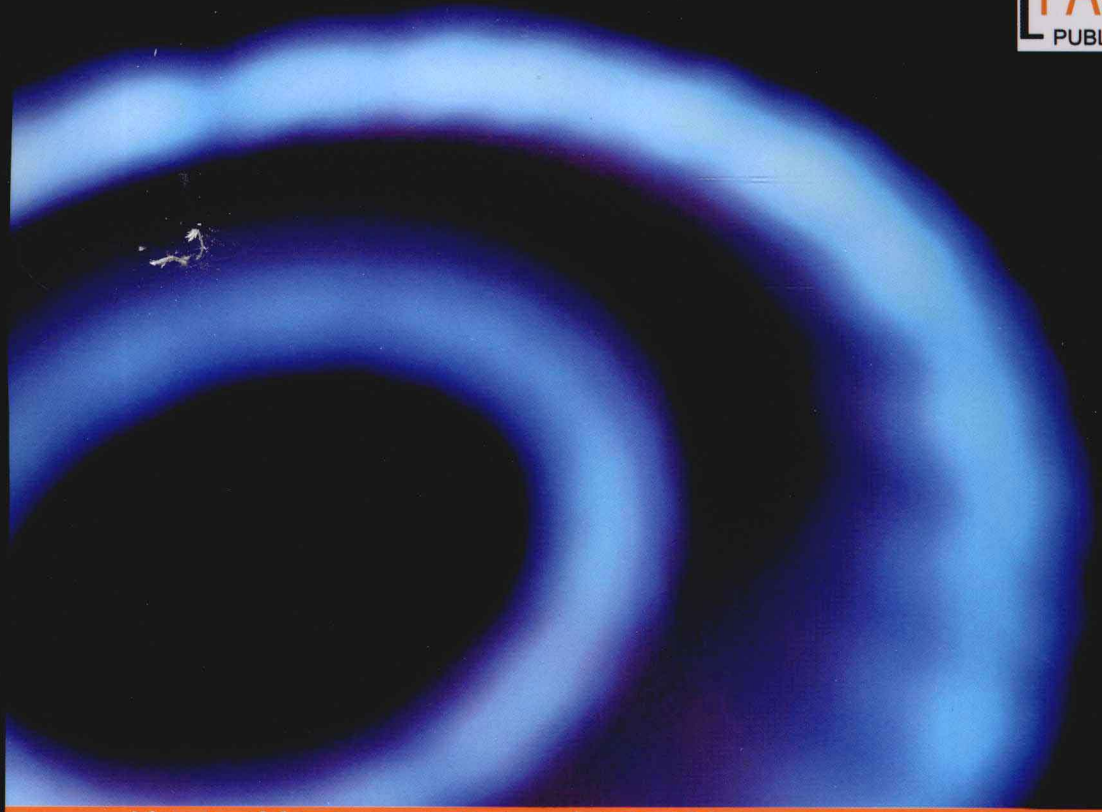
PACKT PUBLISHING



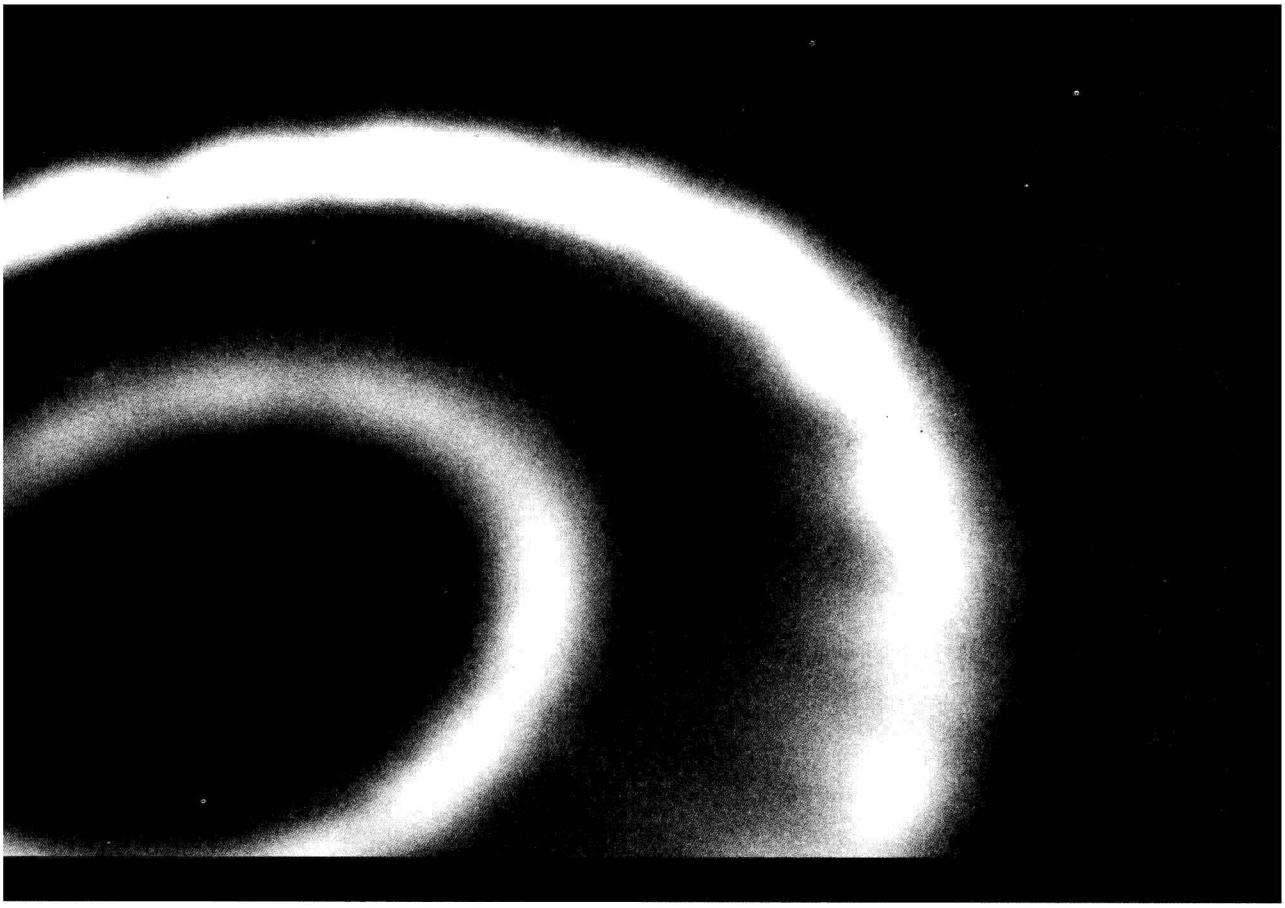
编写可扩展、可重用、高质量的JavaScript应用程序和库

JavaScript 面向对象编程指南

Obj[redacted]d JavaScript

[加] Stoyan Stefanov 著
凌杰 译

人民邮电出版社
POSTS & TELECOM PRESS



JavaScript 面向对象编程指南

[加] Stoyan Stefanov 著
凌杰 译

人民邮电出版社
北京

图书在版编目（CIP）数据

JavaScript面向对象编程指南 / (加) 斯托扬(Stefanov,S.) 著 ; 凌杰译. -- 北京 : 人民邮电出版社, 2013.3
ISBN 978-7-115-30904-4

Ⅰ. ①J… Ⅱ. ①斯… ②凌… Ⅲ. ①JAVA语言－程序设计－指南 Ⅳ. ①TP312-62

中国版本图书馆CIP数据核字(2013)第016139号

版权声明

JavaScript 面向对象编程指南

◆ 著 [加]Stoyan Stefanov
译 凌 杰
责任编辑 陈冀康

◆ 人民邮电出版社出版发行 北京市崇文区夕照寺街 14 号
邮编 100061 电子邮件 315@ptpress.com.cn
网址 http://www.ptpress.com.cn
三河市海波印务有限公司印刷

◆ 开本：800×1000 1/16
印张：20.75
字数：404 千字 2013 年 3 月第 1 版
印数：1 – 3 000 册 2013 年 3 月河北第 1 次印刷

著作权合同登记号 图字：01-2012-4600 号
ISBN 978-7-115-30904-4

定价：59.00 元

读者服务热线：(010)67132692 印装质量热线：(010)67129223
反盗版热线：(010)67171154

目录

第 1 章
引言

众所周知，时下所流行的这些 Web 应用，例如 Yahoo! Maps、Google Maps、Yahoo! Mail、My Yahoo!、Gmail、Digg 以及 YouTube 等都有一些明显的共同特征，即：它们都是 Web2.0 时代的应用程序，都有非常丰富的人性化交互界面，而这往往意味着大量的 JavaScript 应用。事实上，JavaScript 最初也只不过是一种内嵌于 HTML 语句中的单行式脚本语言。但如今已经今非昔比了，对于它今天所拥有的面向对象特性来说，无论是在可重用性方面，还是在可扩展性方面都已经足以支持我们去实现任何网站项目中的行为逻辑了。毕竟，对于今天的标准来说，任何一个符合规范的 Web 页面都应该包含以下三个要素：内容（HTML）、外观（CSS）和行为（JavaScript）。

通常来说，JavaScript 程序的运行必须要依赖于某个宿主环境。其中最常见的当然就是我们的 Web 浏览器了，但请注意，浏览器并不是 JavaScript 代码唯一的宿主环境。事实上，我们可以利用 JavaScript 来创建各种类型的插件工具、应用扩展以及其他形式的组件。总之，学习 JavaScript 语言是一件一举多得的事情，我们可以通过学习这种语言，来编写各种不同的应用程序。

这本书将着重于介绍 JavaScript 语言本身，特别是其中的面向对象特性。我们会从零开始讲解这些内容，也就是说，读这本书无需具备任何的程序设计基础。另外，除了有一章内容是专门为 Web 浏览器环境而写的以外，本书其余部分介绍的都是 JavaScript 的一般特性，适用于任何支持该语言的执行环境。

现在，让我们进入第 1 章的学习吧。首先，我们需要先来了解一下 JavaScript 背后的发展历程，而后我们才能逐步引入面向对象编程方面的基本概念。

1.1 回顾历史

起初，Web 站点事实上只不过是一个静态的 HTML 文档集，这些文档之间仅依靠一些

简单的超链接（hyperlinks）绑定在一起。但很快，随着 Web 业务的快速普及和增长，网站管理者越来越希望自己所创建的 Web 页面能处理更多的事情。例如，他们希望网站具有更丰富的用户交互能力，以便能完成一些简单的任务（比如验证表单之类），加强与服务器端的信息交互。那时候，他们有两种选择：Java applets（后来被证明失败了）和 LiveScript。其中，LiveScript 就是 1995 年由 Netscape 公司开发的程序设计语言。Netscape 2.0 之后，它正式被更名为 JavaScript。

不久，这种对 Web 页面中静态元素进行扩展的方式就在业界大放异彩，令其他的浏览器厂商也纷纷效仿，推出了自己的类似产品。例如，Microsoft 公司随后就发布了支持 JScript 的 Internet Explorer（IE）3.0。该语言在 JavaScript 的基础上引入了一些 IE 独有的特性。最终，为了使语言的实现更趋向于标准化，一个叫做 ECMAScript（欧洲计算机制造商协会）的组织应运而生了。这才有了我们今天所看到的这份被叫做 ECMA-262 的标准文档。目前在业界广为流行的 JavaScript 也只是遵守该标准的一种实现而已。

无论结果是好是坏，JavaScript 在随后爆发的第一次浏览器大战（大约是在 1996 年到 2001 年间）中得到了迅速的普及。那时正值互联网发展的第一波繁荣期，主要由 Netscape 和 Microsoft 这两大浏览器厂商在争夺市场份额。在此过程中，他们不断地往各自的浏览器中添加新的特性和各种版本的 JavaScript 实现。但他们彼此之间又缺乏共同遵守的标准，这给 JavaScript 的开发带来大量的负面影响，也给开发人员带来巨大的痛苦。因为在这种情况下，我们通常只能针对某一个具体的浏览器来编写脚本。如果我们把在这个浏览器上开发的脚本拿到其他浏览器上测试，就会发现它们完全不能工作。与此同时，由于浏览器厂商都在忙于继续增加新的浏览器特性，以至于根本没能及时更新相应的工具，这导致了开发工具的严重滞后。

尽管浏览器厂商引入的不兼容性使 Web 开发人员感到难以忍受，但这还只是问题的一个方面。而另一方面的问题则出在开发人员自己身上，他们在自己的 Web 页面中使用了太多的新特性，总是迫不及待地想引入浏览器提供的每一项新功能，以“加强”自己的页面。例如状态栏中的动画、闪烁的颜色、闪烁的文本、会摇晃的浏览器窗口、屏幕上的雪花效果、能跟踪对象的鼠标光标等，而这往往都是以牺牲实用性为代价的。这种滥用现象极大地损坏了 JavaScript 在业界的名声，以至于那些“真正的程序员”（这里特指那些具有更成熟的编程语言背景的开发人员，例如 Java 或 C/C++程序员）对 JavaScript 根本不屑一顾，或者仅仅把它当做一种用于前端设计的玩具。

出于上述原因，JavaScript 语言在一些 Web 项目中遭到了强烈抵制。某些项目甚至完全拒绝在客户端上进行任何的程序设计，转而只信任他们自己可以掌控的服务器端。确实，在当时的情况下，也没有什么理由值得我们花费双倍的时间来为这些不同的浏览器设计项

目，然后再花更多的时间去调试它们。

1.2　变革之风

这种情况一直持续到第一次浏览器大战结束。但在随后的几年中，Web 开发领域在一系列历史进程的推动下，终于发生了一些非常积极的变化。

- Microsoft 公司赢得了战争，但在之后的五年中（这或多或少算得上一个互联网时代了），他们停止了继续向 Internet Explorer 和 JScript 中添加新特性的动作，这给了其他浏览器充分的时间，使它们能够在功能上逐步完成对 IE 的追赶和超越。
- Web 标准在移动开发领域的重要性在开发人员和浏览器厂商那里得到一致的认可。这是很自然的，毕竟对于开发人员来说，谁也不想因为不同的浏览器而花费双倍（甚至更多）的开发时间，这促使各方都越来越倾向于遵守统一的开发标准。尽管目前，我们离建立一个完全统一的标准化环境还有很长的路要走，但目标已经很明确了，相信终会有实现的那一天的。
- 开发人员和技术本身也日趋成熟了，更多的人开始将注意力转移到东西本身的可用性上，并以此为基础，逐步加强在技术和功能方面的开发力度。

在这种健康环境的影响下，开发人员开始谋求一种更好的新型开发模式，以取代这些现有的开发方式。而随着类似 Gmail 和 Google Maps 这样的应用程序的相继出现，客户端的程序设计也开始逐渐变得丰富起来。显然，如今的 JavaScript 已经成为一种成熟的、在某些方面独一无二的、具有强大原型体系的面向对象语言。关于这点，最好的例子莫过于是对 XMLHttpRequest 对象的重新发现和推广，该对象起初不过是一个 IE-only 特性，但如今已经得到绝大多数浏览器的支持。通过 XMLHttpRequest 对象，JavaScript 可以用 HTTP 请求的形式从服务器上获得所需的新鲜内容，实现了页面的局部更新。这样一来，我们就不必每次都刷新整个页面。随着 XMLHttpRequest 对象的广泛应用，一种类桌面式的 Web 应用程序模式诞生了，我们称之为 AJAX 的应用程序。

1.3　分析现状

关于 JavaScript 语言，最有意思的是它必须要在一个宿主环境中运行。其中受欢迎的宿主环境当然就是浏览器了，但这并不是我们唯一的选择。JavaScript 完全可以运行在服务器端、桌面以及富媒体环境中。如今，我们可以使用 JavaScript 来实现以下功能：

- 创建具有强大而丰富的 Web 应用程序（这种应用程序往往运行于 Web 浏览器中，例如 Gmail）。
- 编写类似 ASP 这样的服务器端脚本，或者使用 Rhino（这是一种用 Java 实现的 JavaScript 引擎）这样的框架进行编程。
- 创建某些富媒体式的应用程序（如 Flash、Flex），这其中用到的 ActionScript 就是一种基于 ECMAScript 标准的脚本语言。
- 编写 Windows 桌面自动化管理脚本任务，我们可以使用 Windows 自带的脚本宿主环境。
- 为一些桌面应用程序编写扩展或插件，例如 Firefox、Dreamweaver、Fiddler。
- 创建一些桌面型 Web 应用程序，这些应用程序往往会使用离线型数据库来存储信息，例如 Google Gears。
- 创建 Yahoo! Widgets、Mac Dashboard 这样的小工具或某些桌面型 Adobe Air 应用程序。

当然，这里列出的也远远不是该语言应用的全部。JavaScript 应用的确发端于 Web 页面，但如今，几乎可以说它们已经无所不在了。

1.4 展望未来

对于未来的情况，我们在这里只能做一些猜测。但几乎可以肯定地说，JavaScript 语言必将还会有它的一席之地。毕竟，在过去相当长的一段时间里，JavaScript 在被严重低估、始终未得到充分利用（或者被错误地滥用了）的情况下，依然几乎每天都能有很多新的、有趣的 JavaScript 应用被开发出来。尽管它们都是一些简单的、内嵌于 HTML 标签中（例如 onclick 事件）的单行式代码。而如今的开发人员所面对的商业开发往往要复杂得多，这需要良好的设计和规划，以及合适的应用扩展和程序库。JavaScript 必将在其中得到真正的用武之地，开发人员无疑会更加重视它独有的面向对象特性，以获取越来越多的便利。

一旦 JavaScript 成为未来招聘中的“必需项”，您在这方面的知识储备就会成为能否成功应聘某些 Web 开发职位的决定性因素。例如，我们在面试时常会被问到这样的问题：“JavaScript 是一种面向对象语言吗？如果是，您在 JavaScript 中是如何实现继承的呢？”读了这本书之后，您就会对这样的面试有充分的准备，并有可能凭借一些连面试官自己都不知道的知识而打动他们。

1.5　面向对象的程序设计

在我们深入学习 JavaScript 之前，首先要了解一下“面向对象”的具体含义，以及这种程序设计风格的主要特征。下面我们列出了一系列在面向对象程序设计（OOP）中最常用到的概念：

- 对象、方法、属性
- 类
- 封装
- 聚合
- 重用与继承
- 多态

现在，让我们来进行逐一阐述。

1.5.1　对象

既然这种程序设计风格叫做面向对象，那么它的重点就在于对象。而所谓的对象，实质上是指“事物”（包括人和物）在程序设计语言中的表现形式。这里的“事物”可以是任何东西（如某个客观存在的对象，或者某些较为抽象的概念）。例如，对于猫这种常见对象来说，我们可以看到它们具有某些明确的特征（如颜色、名字、体型等），能执行某些动作（如喵喵叫、睡觉、躲起来、逃跑等）。在 OOP 语义中，这些对象特征就叫做属性，而那些动作就称之为方法。

此外，我们还有一个口语方面的类比[①]：

- 对象往往是用名词来表示的（如 book、person）
- 方法一般都是些动词（如 read、run）
- 属性值则往往是一些形容词

我们可以来试一下。例如，在“The black cat sleeps on my head”这个句子中，“the cat”（名词）就是一个对象，“black”（形容词）则是一个颜色属性值，而“sleep”（动词）则代

[①] 这里应该特指英文环境，在中文这种形而上的语言环境中，这种类比或许并不是太合适。——译者注

表一个动作，也就是 OOP 语义中的方法。甚至，为了进一步证明这种类比的合理性，我们也可以将句子中的“on my head”看做动作“sleep”的一个限定条件，因此，它也可以被当做传递给 `sleep` 方法的一个参数。

1.5.2 类

在现实生活中，相似的对象之间往往都有一些共同的组成特征。例如蜂鸟和老鹰都具有鸟类的特征，因此它们可以被统称为鸟类。在 OOP 中，类实际上就是对象的设计蓝图或者制作配方。“对象”这个词，我们有时候也叫做“实例”，所以我们可以说老鹰是鸟类的一个实例[①]。我们能基于相同的类创建出许多不同的对象。因为类更多的是一种模板，而对象就是在这些模板的基础上被创建出来的。

但是我们要明白，JavaScript 与 C++或 Java 这种传统的面向对象语言不同，它实际上压根儿没有类。该语言的一切都是基于对象的，其所依靠的是一套原型系统（这里的原型（prototype）实际上也是一种对象，我们稍后再来详细讨论这个问题）。在传统的面向对象语言中，我们一般会这样描述自己的做法：“我基于 Person 类创建了一个叫做 Bob 的新对象。”，而在这种基于原型的面向对象语言中，我们则会这样描述：“我将现有的 Person 对象扩展成了一个叫做 Bob 的新对象。”

1.5.3 封装

封装则是另一个 OOP 相关的概念，它主要用于阐述对象中所包含（或封装）的内容，它们通常由两部分组成：

- 相关的数据（用于存储属性）。
- 基于这些数据所能做的事（所能调用的方法）。

但除此之外，封装这个术语中还包含了一层隐藏信息的概念，这完全是另一方面的问题。因此，我们在理解这个概念时，必须要留意它在具体的 OOP 语境中的含义。

以一个 MP3 播放器为例。如果假设这是一个对象，那么作为该对象的用户，我们无疑需要一些类似于像按钮、显示屏这样的工作接口。这些接口能帮助我们使用该对象（如播放歌曲之类）。至于它们内部是如何工作的，我们并不清楚，而且多数情况下也不会在乎这些。换句话说，这些接口的实现对我们来说是不可见的。同样的，在 OOP 中也是如此。当我们在代码中调用一个对象的方法时，无论该对象是来自我们自己的实现还是某个第三方

[①] 至少在中文环境中，老鹰更像是鸟类的一个子类。希望读者在理解对象与类的关系时，不要过分依赖这种类比。——译者注

库，我们都不需要知道该方法是如何工作的。在编译型语言中，我们甚至都无法查看这些对象的工作代码。而由于 JavaScript 是一种解释型语言，源代码是可以查看的。但至少在这个概念上它们是一致的，即我们只需要知道所操作对象的接口，而不必去关心它的具体实现。

关于信息隐藏，还有另一方面内容，即方法与属性的可见性。在某些语言中，我们能通过 public、private、protected 这些关键字来限定方法和属性的可见性。这种限定分类定义了对象用户所能访问的层次。例如，private 方法只有其所在对象内部的代码才有权访问，而 public 方法则是任何人都能访问的。在 JavaScript 中，尽管所有的方法和属性都是 public 的，但是我们将会看到，该语言还是提供了一些隐藏数据的方法，以保护程序的隐密性。

1.5.4　聚合

所谓聚合，有时候也叫做组合，实际上是指我们将几个现有对象合并成一个新对象的过程。总之，这个概念所强调的就是这种将多个对象合而为一的能力。通过聚合这种强有力的方法，我们可以将一个问题分解成多个更小的问题。这样一来，问题就会显得更易于管理（便于我们各个击破）。当一个问题域的复杂程度令我们难以接受时，我们就可以考虑将它分解成若干子问题区，并且必要的话，这些问题区还可以再继续分解成更小的分区。这样做有利于我们从几个不同的抽象层次来考虑这个问题。例如，个人电脑是一个非常复杂的对象，我们不可能知道它启动时所发生的全部事情。但如果我们将这个问题的抽象级别降低到一定的程度，只关注它几个组件对象的初始化工作，例如监视器对象、鼠标对象、键盘对象等，我们就很容易深入了解这些子对象情况，然后再将这些部分的结果合并起来，之前那个复杂问题就迎刃而解了。

我们还可以找到其他类似情况，例如 Book 是由一个或多个 author 对象、publisher 对象、若干 chapter 对象以及一组 table 对象等合并（聚合）而成的对象。

1.5.5　继承

通过继承这种方式，我们可以非常优雅地实现对现有代码的重用。例如，我们有一个叫做 Person 的一般性对象，其中包含一些姓名、出生日期之类的属性，以及一些功能性函数，如步行、谈话、睡觉、吃饭等。然后，当我们发现自己需要一个 Programmer 对象时，当然，这时候你可以再将 Person 对象中所有的方法与属性重新实现一遍，但除此之外还有一种更聪明的做法，即我们可以让 Programmer 继承自 Person，这样就省去了我们不少工作。因为 Programmer 对象只需要实现属于它自己的那部分特殊功能（例如“编写代码”），而其余部分只需重用 Person 的实现即可。

在传统的 OOP 环境中，继承通常指的是类与类之间的关系，但由于 JavaScript 中不存