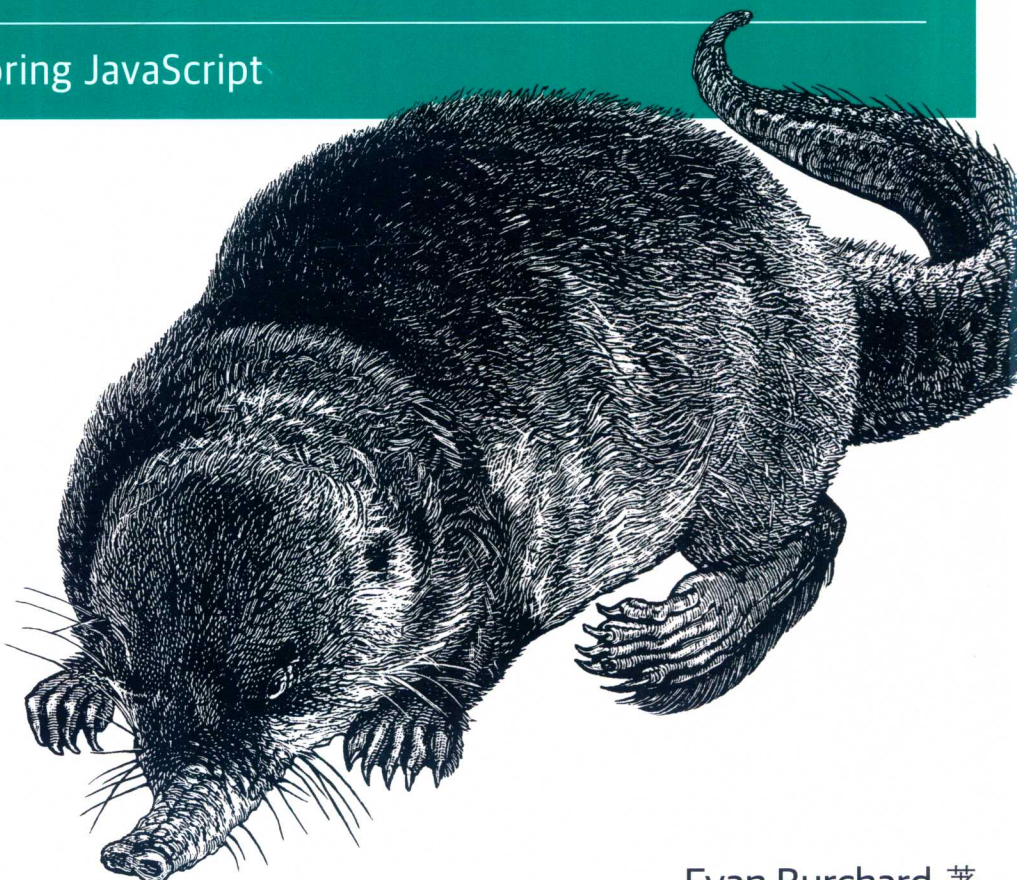
O'REILLY®

重构 JavaScript

Refactoring JavaScript



非外借

Evan Burchard 著
韩天奇 译

中国电力出版社

重构JavaScript

Evan Burchard 著

韩天奇 译

topol • Tokyo O'REILLY®

Media, Inc. 授权中国电力出版社出版

中国电力出版社

图书在版编目（CIP）数据

重构JavaScript / (美) 埃文·伯查德（Evan Burchard）著；韩天奇译. — 北京：中国电力出版社，2018.10

书名原文：Refactoring JavaScript

ISBN 978-7-5198-2354-2

I. ①重… II. ①埃… ②韩… III. ①JAVA语言－程序设计 IV. ①TP312.8

中国版本图书馆CIP数据核字(2018)第194680号

北京市版权局著作权合同登记 图字：01-2018-3792号

出版发行：中国电力出版社
地　　址：北京市东城区北京站西街19号（邮政编码100005）
网　　址：http://www.cepp.sgcc.com.cn
责任编辑：刘　炽（liuchi1030@163.com）
责任校对：王小鹏
装帧设计：Karen Montgomery，张　健
责任印制：杨晓东

印　刷：北京天宇星印刷厂
版　次：2018年10月第一版
印　次：2018年10月北京第一次印刷
开　本：750毫米×980毫米　16开本
印　张：25.5
字　数：490千字
印　数：0001—3000册
定　价：88.00元

O'Reilly Media, Inc.介绍

O'Reilly Media通过图书、杂志、在线服务、调查研究和会议等方式传播创新知识。自1978年开始，O'Reilly一直都是前沿发展的见证者和推动者。超级极客们正在开创着未来，而我们关注真正重要的技术趋势——通过放大那些“细微的信号”来刺激社会对新科技的应用。作为技术社区中活跃的参与者，O'Reilly的发展充满了对创新的倡导、创造和发扬光大。

O'Reilly为软件开发人员带来革命性的“动物书”；创建第一个商业网站（GNN）；组织了影响深远的开放源代码峰会，以至于开源软件运动以此命名；创立了《Make》杂志，从而成为DIY革命的主要先锋；公司一如既往地通过多种形式缔结信息与人的纽带。O'Reilly的会议和峰会集聚了众多超级极客和高瞻远瞩的商业领袖，共同描绘出开创新产业的革命性思想。作为技术人士获取信息的选择，O'Reilly现在还将先锋专家的知识传递给普通的计算机用户。无论是通过书籍出版，在线服务或者面授课程，每一项O'Reilly的产品都反映了公司不可动摇的理念——信息是激发创新的力量。

业界评论

“O'Reilly Radar博客有口皆碑。”

——Wired

“O'Reilly凭借一系列（真希望当初我也想到了）非凡想法建立了数百万美元的业务。”

——Business 2.0

“O'Reilly Conference是聚集关键思想领袖的绝对典范。”

——CRN

“一本O'Reilly的书就代表一个有用、有前途、需要学习的主题。”

——Irish Times

“Tim是位特立独行的商人，他不光放眼于最长远、最广阔的视野并且切实地按照Yogi Berra的建议去做了：‘如果你在路上遇到岔路口，走小路（岔路）。’回顾过去Tim似乎每一次都选择了小路，而且有几次都是一闪即逝的机会，尽管大路也不错。”

——Linux Journal

献给 Jade，再次并且永远。

目录

序

我仍然记得在 1999 年 Martin Fowler 的《重构：改善既有代码的设计》出版时读了这本书。这本书给了我一个启示：我从未见过代码被认为是可塑的。程序员倾向于从头开始重写代码库，但这本书认为可以通过小的、有原则的和比较安全的步骤来演进和清理现有的代码。在这样做的时候，测试提供了一张额外的安全网，使你能够自信地向前推进。我将始终遵循这本书中的一条建议，每当你写代码时，始终保持两种行为完全分开：实现新功能和重构现有的代码。如果这样做，你会避免同时做太多事情，并且会产生更少的错误。

《重构 JavaScript》采用重构的思想，将其应用于 JavaScript 的世界。JavaScript 的动态特性意味着与更多的静态语言（如 Java）相比，你需要使用不同的重构技术。在 Java 中，你有静态类型。而且继承和多态被使用得很频繁。对于 JavaScript，你通常依赖于静态检查工具（如 ESLint 和 Flow），并可根据需要灵活调整对象。函数式编程技术也更受欢迎。此外，测试发挥了更重要的作用，但它们往往也更轻量级。对于所有这些问题以及更多内容（例如，异步代码），本书均已经涵盖了！

阅读愉快！

——Axel Rauschmayer

前言

欢迎阅读本书。在本书中，我们将探讨如何编写更好的 JavaScript，从经典的重构技术中获取灵感，同时探索各种风格的编程方式。

为什么写本书

不管你喜欢与否，JavaScript 都不会消失。无论使用什么框架、“编译成 JS”的语言或库，如果底层的 JavaScript 代码质量很糟糕的话，错误和性能将会是始终存在的问题。重写，包括移植到当前热门的框架，是非常昂贵且不可预测的。错误不会凭空消失，它们仍会在新的代码中重现。如果把事情变得更复杂，至少要将功能暂时丢弃掉。

本书为如何最好地避免不理智地写 JavaScript 提供了清晰的指导。糟糕的代码不必保持这种方式。让它变得更好并不需要让人望而生畏，也不会太昂贵。

本书的读者对象

本书是写给那些有写糟糕代码经验，并想要写更好代码的人。也是写给那些在前端或者后端写 JavaScript 的人。还是写给那些写 JavaScript 以及由于 JavaScript 对浏览器平台的垄断而被困住了的人。

你如果是一个纯粹的初学者，那么你可能首先要写几个月糟糕的代码。如果你不想写更好的代码，你可能没有读完这本书的耐心。如果这两种情况都不能描述你，那我们继续吧。

有趣的是，有许多努力使 JavaScript 变得更好，同时也有其他努力使它过时。写优秀

的和糟糕的 JavaScript 代码的方法在持续扩展。框架对于处理复杂性起很大作用，但是局限于框架的程序员会被限制住。如果你发现你（或者你的代码库）在一个框架之外（或者在更混乱的边缘）很难工作，本书应该会给你提供新的思路去处理你的工作。

如果你在测试、调试或代码置信度方面有困难，本书对你是有帮助的。

我们大多数人没有在完美的代码库上工作，特别是在 JavaScript 中，工程师可能主要使用 Ruby、Python、Java 等。本书所做的是帮助你识别代码库中哪些部分是糟糕的，同时提供了大量的改进选择。

如何使用本书

第 1~5 章描述了 JavaScript、重构、质量、置信度与测试之间的相互作用。在很多书中，通常将测试放在最后。在本书中，对于我们探索的代码类型，这是不合适的。测试对于置信度至关重要。置信度对于重构至关重要。重构对于质量至关重要，这也是我们的目标：

测试 → 置信度 → 重构 → 质量

JavaScript（及其生态系统）恰好提供了转换发生的空间，所以开篇这些章节必须包括对语言本身的探索。如果你对刚才描述的转换完全理解，你可以略读或跳过这些章节。虽然不推荐，但这是你的书，所以你可以随意使用它。如果你认为它最适合用作门挡，或用来取暖或做某种牺牲，那就去做吧。如果你确实发现了本书有非常规用途，给我发图片或视频：*http://evanburchard.com/contact* 或在 Twitter 和 GitHub 上 @evanburchard。

我也可以把电子版烧掉或用作门挡吗？

不幸的是，不行。但是，由于本书符合知识共享许可协议，你可以自由地分享 HTML 版本以及任何其他在 *http://refactoringjs.com* 上的文件的链接。

在第 5 章之后，内容变得更难，尤其是如果你跳过了第 1~5 章。有更多的代码需要写和跟随。在第 6 章和第 7 章，我们将讨论重构函数和对象，我们不会回避一些更复杂的 JavaScript 代码。总的来说，这些章节的目标是为改进代码提供选择，而不需要从根本上改变接口。通过应用这两章中的技术，你将能够将混乱的代码库变成有良好质量基准的代码。

第 8 章将我们的架构扩展到包含（或避免）层次结构的视图。

第 9~11 章专门针对特定的主题（分别是设计模式、异步编程和函数式编程），它们可以让你的代码超越基准代码，这必然涉及更积极的变化。在第 9 章的设计模式中，我们认识到从 JavaScript 的面向对象的方面扩展和借鉴的方法，并覆盖了重构和面向对象编程（OOP）之间的历史联系。在第 10 章中，我们处理了许多 JavaScript 代码库同时做很多事情的现实情况。在第 11 章中，我们将通过几个库来体验函数式编程，这些库提供了超出标准数组函数（`forEach`、`map`、`reduce` 等）所提供的接口。

从某种意义上来说，最后的三章脱离了我们最初的重构目标：在不改变接口的情况下改变实现细节。另一方面，这些接口都是有用的，有时是不可避免的。我们很容易发现自己为了性能而写异步代码。或者我们可能发现自己被困在一个代码库中，它在 OOP 或函数式编程（FP）投入了大量好或不好的尝试。因此，无论是通过选择还是通过我们继承的代码，都是我们应该关注并改进的 JavaScript 的部分。如果你对代码库采用了完全不同的范式，那么本书大部分内容所说的不是你所做的“重构”。

如果我们想要严格控制它，这些章节仍然在它们的范式中重构（OOP 到更好的 OOP，异步到更好的异步，以及 FP 到更好的 FP），如果我们想从最广泛的角度考虑我们的程序的执行（例如，运行 **`node myprogram.js`** 作为输入并且“being satisfied with how it ran”作为输出），那么即使在不同的范式间切换，我们也可以重构。我鼓励你先尝试较小的增量更改，这会更容易测试并且让你更有信心。

引用 William Opdyke 关于重构的原始论文：

> 语义等价的这种定义允许在整个程序中进行更改，只要输入到输出值的映射保持不变。想象一下，在受重构影响的程序周围绘制一个圆。从圆外观察的行为不会改变。对于一些重构，圆包围了大部分或全部程序。例如，如果一个变量在整个程序中被引用，那么更改其名称的重构将影响大部分程序。对于其他重构，这个圆覆盖的面积要小得多；例如，当包含在一个函数中的某个函数调用被内联扩展时，只有一部分函数体会受到影响。在这两种情况下，关键思想是从圆外部观察，调用的操作结果（包括副作用）和圆外的引用都不会改变[注 1]。

尽管我们可以自由地绘制圆的大小，我们常常看到“重构”这个词被随意使用，就好像它只意味着“改变代码”一样。正如我们在第 1 章所讨论的，它不是这样的。这在小规模的项目中很常见，比如我们浏览最多的页面。考虑第 8~10 章，首先作为架构的选择，其次在这些选择中创建更好的代码（安全且增量地）的可能性。例如，如果

注 1：William Opdyke，“Refactoring Object-Oriented Frameworks”（博士论文，伊利诺伊大学厄巴纳－香槟分校，1992），40。

有人说“使用异步代码来重构”，这可能太宽泛了，以至于难以以安全且增量的方式执行。但如果你想到第 9 章给了你这样做的力量，我不能阻止你。现在这是你的书。你可以绘制你想要的圆。

如果你对任何工具或概念感到困惑，你可能会发现附录是有帮助的。如果你正在寻找示例代码和其他信息，请访问本书的网站。你也可以在那里找到一个 HTML 版本的书，如果你喜欢这样阅读的话。

总之，本书用来了解：

- 重构。
- 测试。
- JavaScript。
- 重构与测试 JavaScript。
- 几个 JavaScript 范式。
- 使用这些 JavaScript 范式进行重构与测试。

或者（在成人监督下），书的纸质版可以用来生火并用于：

- 取暖。
- 抗议。
- 科技图书牺牲仪式。

http://refactoringjs.com/ 上的电子版文件可以按照知识共享许可限制的方式传播。

如果你有任何问题、疑问、抱怨或赞美，请随时通过我的网站联系我。

本书的一些术语

App，应用，程序

本书中的一些词是不精准的。App、应用、程序和网站大多数情况下是可以互换的。如果有任何困惑的地方，本书描述了提高 JavaScript 质量的基本原则，所以这些词不应该完全按照字面意义解读。也许你的代码是一个库或框架？无论如何，本书中的技巧应该都是适用的。

通过词语和图表提高包容性

本书中的一些词可能不会对每个人都具有包容性。我试图平衡他和她的使用，我意识到这并不是每个人理想的方式。虽然我更倾向于使用单数形式，但这不是出版商的准则。

此外，我（太晚）意识到我对图表的依赖，特别是在第 5 章中，这可能对有视力障碍的读者造成严重的伤害。如果你觉得你因为这个原因错过了任何内容，请随时向我提出任何问题。

用户

在本书中还有一个词我真的很讨厌，就是用户。它并不精确，并且还在创建者（开发者 / 设计者）和消费者（用户）之间拉开了一定距离。更精确和友善的词通常特定于问题域，否则我们会被像“人”或“使用程序 / 网站的人”之类的术语困住。如果没有比人或用户（甚至包括客户）更具体的术语，它可能暗示着商业模式纯粹是将人们作为数据销售，但这是另一个讨论。

关键是，在本书中使用术语用户来传达一个熟悉的概念：使用程序 / 网站的人。此外，还没有广泛和准确的术语来代替用户体验（UX）或用户界面（UI）的相关术语。我为了节省精力，只是在这里谈论它，而不是在几个地方解释这个问题，也没有使用非标准或特定的术语来解释常见的抽象概念。

无论如何，我完全赞同“数据达芬奇”Edward Tufte 的话（及其含义）：

> 只有两个行业将其客户称为用户：非法药物和软件公司。

有一个叫做“道德设计”的运动，希望能在某种程度上帮助行业摆脱这一术语（以及源于它的不合理的做法）。

第三方库和社区

尽管我非常努力地展示了最好的工具来说明重构和测试 JavaScript 的基本原理，但有时你会发现某个工具并不适合你。好消息是，JavaScript 具有丰富的生态系统以供选择。我偏爱简单、灵活和职责单一的工具，但你可能会有不同的想法。在本书中，大型框架尤其没有被探究，因为它们往往有自己的其他工具的生态系统（通常它们本身相当活跃和多样）。我绝对会在你刚开始的时候推荐一个框架，但是它们在与基础语言的功能结合时才是最有用的，我相信本书在这方面会教你很多。

此外，每个工具、框架和库都伴随着社区和历史。正如我不相信任何一种工具能一直有效，我也不支持任何第三方代码或项目背后的社区。很多项目有行为准则，让你知道参与其中是否会让你愉快地利用你的时间。

API、接口、实现、“客户端代码”

这有点模糊，但我希望我能强调更多的不是对象的层次结构，而是一个设计良好的代码库接口的层次结构。当代码是一段简单的脚本时，我们希望它可以作为一个过程从上到下运行。随着代码库的成熟（通过设计），我们期望它在三个主要的层中发展（尽管这显然会扩展出更复杂的代码库）。

第一层在本书中被称为实现，是代码库深处的代码。对于重构，最重要的区别是实现层和下一层之间的区别。第二层可以称为接口或 API ，描述了使用代码库作为模块时应该期望与之交互的“公有”函数和对象。第三层一般被称为客户端代码或调用代码。它指的是与接口层交互的代码。这是使用模块的人写的代码，并且我们会写测试代码来测试接口层。

架构基础

在本书中，起初我们创建的程序非常没有条理，我们的重点（不管像 OOP 或 FP 这样的范式）在这三个层之间的划分。这使得代码是可测试的和可移植的。如果你主要依赖于提供自己组织的框架，那么你可能不熟悉这个过程。

输入（非局部与自由变量）

贯穿全书（特别是第 5 章），我们区分三种类型的输入：

- 显式输入（传递给函数的参数）。
- 隐式输入（指代包含上下文对象、函数或类的 this）。
- 非局部输入（函数或对象中使用的在其他地方定义的值）。

这里有两点要注意。首先，为了图解或其他原因，函数内创建的局部变量（或常量）不被认为是“输入”。其次，虽然本书使用非局部输入作为一个精确的术语，但更常见的名称是自由变量。然而，考虑到非局部输入可以是常量而不是变量，这样叫它是有点不精确。类似地，一些人使用术语绑定变量来指代我们所说的显式输入，在某种程度上，也指隐式输入。