

重构

改善既有代码的设计

(第2版)

REFACTORING

[美] 马丁·福勒 (Martin Fowler) 著
熊节 林从羽 译

Improving the Design of Existing Code
SECOND EDITION





Pearson

重构

改善既有代码的设计

(第2版)

REFACTORING

[美] 马丁·福勒 (Martin Fowler) 著

熊节 林从羽 译

Improving the Design of Existing Code

SECOND EDITION

人民邮电出版社

北京

图书在版编目 (C I P) 数据

重构：改善既有代码的设计：第2版 / (美) 马丁·福勒 (Martin Fowler) 著；熊节，林从羽译。—北京：人民邮电出版社，2019.5
书名原文：Refactoring: Improving the Design of Existing Code, Second Edition
ISBN 978-7-115-50865-2

I. ①重… II. ①马… ②熊… ③林… III. ①机器代码程序—程序设计 IV. ①TP311.11

中国版本图书馆CIP数据核字(2019)第035067号

内 容 提 要

本书是经典著作《重构》出版 20 年后的新版。书中清晰揭示了重构的过程，解释了重构的原理和最佳实践方式，并给出了何时以及何地应该开始挖掘代码以求改善。书中给出了 60 多个可行的重构，每个重构都介绍了一种经过验证的代码变换手法的动机和技术。本书提出的重构准则将帮助开发人员一次一小步地修改代码，从而减少了开发过程中的风险。

本书适合软件开发人员、项目管理人员等阅读，也可作为高等院校计算机及相关专业师生的参考读物。

-
- ◆ 著 [美]马丁·福勒 (Martin Fowler)
译 熊 节 林从羽
责任编辑 杨海玲
责任印制 焦志炜
 - ◆ 人民邮电出版社出版发行 北京市丰台区成寿寺路 11 号
邮编 100164 电子邮件 315@ptpress.com.cn
网址 <http://www.ptpress.com.cn>
大厂聚鑫印刷有限责任公司印刷
 - ◆ 开本：800×1000 1/16
印张：28.5
字数：546 千字 2019 年 5 月第 1 版
印数：1-30 000 册 2019 年 5 月河北第 1 次印刷
- 著作权合同登记号 图字：01-2019-0847 号
-

定价：99.00 元

读者服务热线：(010)81055410 印装质量热线：(010)81055316

反盗版热线：(010)81055315

广告经营许可证：京东工商广登字 20170147 号

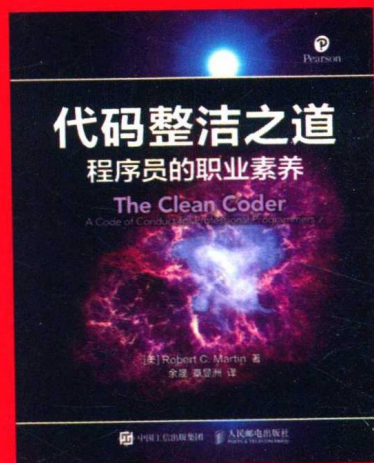
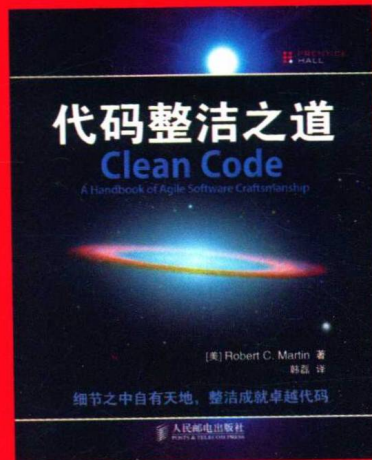
作者简介



马丁·福勒 (Martin Fowler)

世界级软件开发大师，ThoughtWorks 首席科学家。

他是一位作家、演说者、咨询师和泛软件开发领域的意见领袖。他致力于改善企业级的软件设计，对优秀的设计以及支撑优秀设计的工程实践孜孜以求。他在重构、面向对象分析设计、模式、XP 和 UML 等领域都有卓越贡献，著有《重构：改善既有代码的设计》《分析模式》《领域特定语言》等经典著作。



版 权 声 明

Authorized Translation from the English language edition, entitled REFACTORING: IMPROVING THE DESIGN OF EXISTING CODE, 2nd Edition by FOWLER, MARTIN, published by Pearson Education, Inc, Copyright © 2019 Pearson Education, Inc.

All rights reserved. No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording or by any information storage retrieval system, without permission from Pearson Education, Inc.

CHINESE SIMPLIFIED language edition published by POSTS & TELECOM PRESS, Copyright © 2019.

本书中文简体字版由 Pearson Education Inc 授权人民邮电出版社独家出版。未经出版者书面许可，不得以任何方式复制或抄袭本书内容。

本书封面贴有 Pearson Education（培生教育出版集团）激光防伪标签，无标签者不得销售。

版权所有，侵权必究。

对本书的赞誉

过去 20 年,《重构》一直是我案头常备的图书。每次重读,仍有感悟。对我而言,《重构》的意义不只在指导代码重构,更在于让人从一开始就知道什么是好的代码,并且尽量写出没有“坏味道”的代码。Martin Fowler 这次对本书进行的重构,体现了近年来编程领域的一些思潮变化。看来,既有设计,永远有改进空间。

——韩磊,《代码整洁之道》译者

重构早就成了软件开发从业者本能的一部分,每个 IDE 都内置了重构功能,每个程序员都定期重构自己的代码。技能上通常不再是问题,但是相对于当年第 1 版的读者,现在的程序员对于重构这个思想从何而来以及各种细节反而更陌生,这时候就更值得重新读一下这本书了。

——霍炬, PRESS.one CTO

有人说 Martin Fowler 改变了人类开发软件的模式,这一点也不过分,从《分析模式》《UML 精粹》《领域特定语言》,到这本《重构》新版可以看得出来,他的每一本书都是软件开发人员必备的案头读物。此前他参与的“敏捷宣言”,更是引领了整个行业对敏捷开发的认识,一直到现在。Martin Fowler 是我们 QCon 全球软件开发大会进入中国时的第一届讲师,也是在那次会议上,他让国内的技术社区领略了国际领先的开发模式,从此“敏捷”二字开始风行国内 IT 领域。

今年是 QCon 进入中国的第十个年头,我特别开心看到 Martin Fowler 又重写《重构》这本影响深远的书,他几乎完全替换了书中所引用的模式案例,并且基于现在用户的习惯,采用了 JavaScript 语言来做说明语言。数十年来他始终保持对技术的关注,对创新的热情,乐此不疲,这是 Martin 最令人敬佩的地方,也是非常值得我们每一个技术人学习的地方。

——霍泰稳,极客邦科技、InfoQ 中国创始人兼 CEO

当今软件开发的越来越快,带来的技术债也越来越多,我从 CSDN 自身的

网站系统开发中充分认识到重构的重要性——如果我们的程序员能理解和掌握重构的原则和方法，我们的系统就不会有这么多沉重的债务。真正本质的东西是不变的，《重构》在出版 20 年后推出了第 2 版，再次证明：越本质的越长久，也越重要。衷心期待更多的新一代开发者能从这本书吸收营养，开发出好味道的系统。

——蒋涛，CSDN 创始人、董事长

最早看到本书第 1 版的英文原版并决定引进国内，算起来已经是 20 年前的事了。虽然时间是最强大的重构工具，连书里的示例语言都从 Java 变成 JavaScript 了，但书中的理念和实践的价值并没有随时间流逝。这充分证明，即使在日新月异的 IT 技术世界里，不变的东西其实还是有的，这种书才是真正的经典，是技术人员应该优先研读并一读再读的。

——刘江，美团技术学院院长

“对于软件工程师来说，重构，并不是额外的工作，它就是编码本身。”直到我读过《重构》，并经过练习，才真正理解到这一点。真希望自己在 20 多年前写第一个软件时，就能读到这本书，从而能节省出大量调试或重复研究代码的时间。20 年过去了，《重构》这本书也根据当前软件设计及相关工具的发展进行了一部分修订，更加贴近当前的软件开发。希望更多的软件工程师能够应用这一技术节省出更多的时间。

——乔梁，腾讯高级管理顾问、《持续交付 2.0》作者

重构是一项被低估了的技术能力。说起来，重构就是“不改变外在行为，而提高代码质量”这么简简单单的一句话，但其带来的影响却非常深远：它使我们在解决问题时可以放心地“先做对，再做好”——这种思路本身就可以极大地简化问题；它使我们消除无谓的意气之争——“所谓好，就是更少的坏味道”。我由衷地认为，切实地读懂了《重构》的程序员，在能力上都会获得一个数量级的提升。

——徐昊，ThoughtWorks 中国区技术总监

当我还是编程菜鸟，想写出漂亮的代码而不得门道的时候，《重构》这本书就告诉了我，其实高手的代码也不是一次书就的，只要按这本书里的方法去做，谁都能把代码写得那么好；当我还是职场新人，没来得及写出太多垃圾代码的时候，这本书就教会了我，应该去追求编写人能够读懂的而不是仅机器能够读懂的代码。多年以后的某时某刻，当你编码自信而敏捷，因代码清晰而受人尊重时，你会庆幸读过这本书，

你也会有些遗憾，应该再早一点去读这本书。无论过去了多少年，这本书，一直值得推荐。

——阎华，京东 7FRESH 架构师

在大获成功的《重构》第 1 版里，Martin Fowler 传达的核心理念是：代码会随时间流逝而烂掉。写得再好的程序代码，若是发布了就一直保持原样，照样会风化、破碎乃至分崩离析。这是客观规律，避免这种命运的唯一出路是持续重构。要想成为高素质的软件工程师，必须认识这一点。

20 年之后，Martin Fowler 用现身说法证明，经典的《重构》也会变得不合时宜，也需要重构。如今，不但讲解语言从 Java 改成了 JavaScript，原来的重构示例也做了很多调整，新增了 15 个示例，更重要的是，新版示例不再那么“面向对象”，应当会收获更广泛的读者群。

软件不死，重构不歇。

——余晟，《代码整洁之道：程序员的职业素养》译者

随着软件项目日积月累，系统维护成本变得越来越高昂是互联网团队共同面临的问题。用户在使用互联网系统的过程中，遇到的各类运行错误或者不可访问故障，以及开发团队面临的历史系统不可维护问题，很多时候是代码初次开发过程中各种细小的不规范引起的。持续优化已有代码是维护系统生命力最好的方法。《重构》是我推荐团队必读的技术图书之一。

——杨卫华（Tim Yang），微博研发副总经理

软件行业已经高速发展数十年，就好似一个崭新的城市，从一个个村屋矮房到高楼林立。而你的代码库就好比手下下的一个房间、一幢平房、一条街道、一片社区乃至是一座摩天大楼。作为一本经典的软件开发书，《重构》告诉我们的不仅仅是如何推倒重建、清理、装修，而是像一个规划师一样从目的、成本、手段、价值等综合维度来思考重构的意义。在开发业务的同时，《重构》常伴我左右，警醒我如何写出更有价值的软件。

——阴明，掘金社区创始人

重构，是一个优秀程序员的基本功，因为没人能保证其代码不随时间腐化，而重构会让代码重新焕发活力。整个软件行业对重构的认知始于 Martin Fowler 的《重构》，这本书让人们知道了“代码的坏味道”，见识到了“小步前行”的威力。时隔 20 年，Martin Fowler 重新执笔改写《重构》，20 年间的思维变迁就体现在这本书里，在第 1

版中，我们看到的是当时方兴未艾的面向对象，而第2版则透露出函数式编程的影响。如果说有什么程序员进阶秘笈，那就是不要错过 Martin Fowler 的任何一部著作，更何况是已经由时间证明过的重要著作《重构》的新版！

——郑晔，火币网首席架构师

如果看完本书，就兴冲冲地想要找一些代码来重构，那你就可能陷入某种“自嗨”之中了。

了解本书中列出的那些坏味道，不仅仅可以发现代码中的那些坏味道，更可以鞭策自己以及整个团队：在一开始的时候，就不写或者少些那种味道很坏的代码。还应该激励自己，深入地理解架构、理解业务、理解需求，减少因设计失误而导致徒劳无益地反复重构。

重构也是有成本的，所以应该思考如何降低重构的成本。我推荐每一个程序员都来学习“重构”这门手艺。因为学习《重构》，是为了减少“重构”！

——庄表伟，开源社理事、执行长，华为云 DevCloud 高级产品经理

重读《重构》，呼唤匠艺

（译者序）

2009年，在为《重构》第1版的中译本再版整理译稿时，我已经隐约察觉行业中对“重构”这个概念的矛盾张力。一方面，在这个“VUCA”（易变、不确定、复杂、模糊）横行的年代，有能力调整系统的内部结构，使其更具长期生命力，这是一个令人神往的期许。另一方面，重构的扎实功夫要学起来、做起来，颇不是件轻松的事，且不说详尽到近乎琐碎的重构手法，光是单元测试一事，怕是已有九成同行无法企及。结果，“重构”渐渐成了一块漂亮的招牌，大家都愿意挂上这个名号，可实际上干的却多是“刀劈斧砍”的勾当。

如今又是10年过去，只从国内的情况而论，“重构”概念的表里分离，大有愈演愈烈之势。随着当年的一线技术人员纷纷走上领导岗位，他们乐于将“重构”这块漂亮招牌用在更宽泛的环境下，例如系统架构乃至组织结构，都可以“重构”一下。然而基本功的欠缺，却也一路如影随形。当年在对象中的刀劈斧砍，如今被照搬到了架构、组织的调整。于是“重构”的痛苦回忆又一遍遍重演，甚而程度更深、影响更广、为害更烈。

此时转头看 Martin Fowler 时隔将近廿载后终于付梓的《重构》第2版，我不禁感叹于他对“微末功夫”的执着。在此书尚未成型之前，我和当时 ThoughtWorks 的同事曾有很多猜测，猜 Fowler 先生是否会在第2版中拔高层次，多谈谈设计乃至架构级别的重构手法，甚或跟随“敏捷组织”“精益企业”的风潮谈谈组织重构，也未为不可。孰料成书令我们跌破眼镜，Fowler 先生不仅没有拔高，反而把工夫做得更扎实了。

对比前后两版的重构列表，可以发现：第2版收录的重构手法在用途上更加内聚，在操作上更加连贯，更重视重构手法之间的组合运用。第1版中占了整章篇幅的“大型重构”，在第2版中全数删去。一些较为复杂的重构手法，例如复制“被监视数据”、塑造模板函数等，第2版也不再收录。而第2版中新增的重构手法，则多是提

炼变量、移动语句、拆分循环、拆分变量这样更加细致而微的操作。这些新增的手法看似简单，但直指大规模遗留代码中最常见的重构难点，正好补上了第1版中遗漏的细节。这一变化，正反映出Fowler先生对于重构一事一贯的态度：千里之行积于跬步，越是面对复杂多变的外部环境，越是要做好基本功、迈出扎实步。

识别坏味道、测试先行、行为保持的变更动作，是重构的基本功。在《重构》第2版里，重构手法的细节被再度打磨，重构过程比之第1版愈发流畅。细细品味重构手法中的前后步骤，琢磨作者是如何做到行为保持的，这是能启发读者举一反三的读书法。以保持对象完整重构手法为例，第1版中的做法是在原本函数上新添参数，而第2版的做法则是先新建一个空函数，在其中做完想要的调整之后，再整体替换原本函数。两相对比，无疑是新的做法更加可控、出错时测试失败的范围更小。

无独有偶，我在ThoughtWorks时的同事王健在开展大型的架构重构时，总结了重构的“十六字心法”，恰与保持对象完整重构手法在第2版中这个新的做法暗合。这十六字心法如是说：

旧的不变，
新的创建，
一步切换，
旧的再见。

从这个视角品味一个个重构巨细靡遗的做法，读者大概能感受到重构与“刀劈斧砍”之间最根本的分歧。在很多重构（例如最常用的改变函数声明）的做法中，Fowler先生会引入“很快就会再次修改甚至删除”的临时元素。假如只看起止状态，这些变更过程中的临时元素似乎是浪费：为何不直接一步到位改变到完善的结果状态呢？然而这些临时元素所代表的，是对变更过程（而非只是结果）的设计。缺乏对过程的精心设计与必要投入，只抱着对结果的美好憧憬提刀上阵，遇到困难就靠“奋斗精神”和加班解决，这种“刀劈斧砍”不止发生在缺乏审慎的“重构”现场，又何尝不是我们这个行业的缩影？

是以，重构这门技艺，以及Fowler先生撰写《重构》的态度，代表的是软件开发的匠艺——对“正确的做事方式”的重视。在一个浮躁之风日盛的行业中，很多人会强调“只看结果”，轻视做事的过程与方式。然而对于软件开发的专业人士而言，如果忽视了过程与方式，也就等于放弃了我们自己的立身之本。Fowler先生近廿载对这本书、对重构手法的精心打磨，给了我们一个榜样：一个对匠艺上心的专业人士，日积月累对过程与方式的重视，是能有所成就的。

17年前，我以菜鸟之身读到《重构》，深受其中蕴涵的工匠精神感召，在Fowler先生与侯捷老师的帮助下，完成了本书第1版的翻译工作。如今再译本书第2版，来

自 ThoughtWorks 的青年才俊林从羽君主动请缨与我搭档合译，我亦将此视为匠艺传承的一桩美事。新一代程序员中，关注新工具、新框架、新商业模式者伙矣，关注面向对象、TDD、重构之类基本功者寥寥。林君年纪虽轻，却能平心静气磨砺技艺，对基本功学而时习，颇有老派工匠之风。当年藉由翻译《重构》一书，我从Fowler先生、侯捷老师身上学到他们的工匠精神，十余年来时时践行自勉。如今新一代软件工匠的代表人物林君接手此书，必会令工匠精神传承光大。

据说古时高僧有偈云：“时时勤拂拭，勿使惹尘埃。”代码当如是，专业人士的技艺亦当如是。与《重构》的诸位读者共勉。

熊节

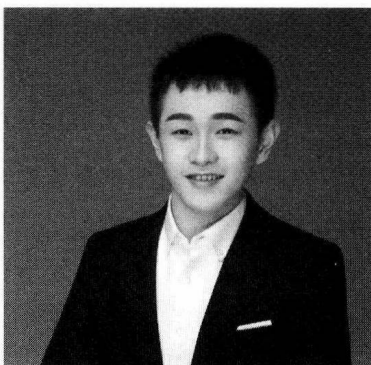
2019年1月26日于成都

译者简介

熊节 在 IT 行业已经打拼了 18 年，在金融、零售、政府、电信、制造业等行业的信息化建设方面有着丰富经验，是中国 IT 业敏捷浪潮的领军人物。熊节拥有利物浦大学 MBA 学位。



林从羽 ThoughtWorks 软件开发工程师，曾服务于国内外多家大型企业，致力于帮助团队更快更好地交付可工作的软件。拥抱敏捷精神，TDD 爱好者，纯键盘工作者。



第 1 版序

“重构”这个概念来自 Smalltalk 圈子，没多久就进入了其他编程语言阵营之中。因为重构是框架开发中不可缺少的一部分，所以当框架设计者讨论自己的工作时，这个术语就诞生了。当他们精炼自己的类继承体系时，当他们叫喊自己可以拿掉多少多少行代码时，重构的概念慢慢浮出水面。框架设计者知道，这东西不可能一开始就完全正确，它将随着设计者的经验成长而进化；他们也知道，代码被阅读和被修改的次数远远多于它被编写的次数。保持代码易读、易修改的关键，就是重构——对框架而言如此，对一般软件也如此。

好极了，还有什么问题吗？问题很显然：重构有风险。它必须修改正在工作的程序，这可能引入一些不易察觉的错误。如果重构方式不恰当，可能毁掉你数天甚至数周的成果。如果重构时不做好准备，不遵守规则，风险就更大。你挖掘自己的代码，很快发现了一些值得修改的地方，于是你挖得更深。挖得越深，找到的重构机会就越多，于是你的修改也越多……最后你给自己挖了个大坑，却爬不出去了。为了避免自掘坟墓，重构必须系统化进行。我和三位合作者在写《设计模式》一书时曾经提过：设计模式为重构提供了目标。然而“确定目标”只是问题的一部分而已，改造程序以达到目标，是另一个难题。

Martin Fowler 和本书另几位作者清楚地揭示了重构过程，他们为面向对象软件开发所做的贡献难以估量。本书解释了重构的原理和最佳实践，并指出何时何地你应该开始挖掘你的代码以求改善。本书的核心是一系列完整的重构方法，其中每一项都介绍一种经过实践检验的代码变换手法的动机和技术。某些项目（如“提炼函数”和“搬移字段”）看起来可能很浅显，但不要掉以轻心，因为理解这类技术正是有条不紊地进行重构的关键。本书所提的这些重构手法将帮助你一次一小步地修改你的代码，这就降低了设计演进过程中的风险。很快你就会把这些重构手法及其名称加入自己的开发词典中，并且朗朗上口。

我第一次体验有条不紊的、一次一小步的重构，是某次与 Kent Beck 在三万英尺高空的飞行旅途中结对编程。我们运用本书中收录的重构手法，保证每次只走一步。最后，我对这种实践方式的效果感到十分惊讶。我不但对产生的代码更有信心，而且

开发压力也小了很多。因此，我极力推荐你试试这些重构手法，你和你的程序都将因此更美好。

——Erich Gamma
Object Technology International, Inc.
1999 年 1 月

前 言

从前，有位咨询顾问造访客户调研其开发项目。该系统的核心是一个类继承体系，顾问看了开发人员所写的一些代码。他发现整个体系相当凌乱，上层超类对系统的工作方式做了一些假设，下层子类实现这些假设。但是这些假设并不适合所有子类，导致覆写（override）工作非常繁重。只要在超类做点修改，就可以减少许多覆写工作。在另一些地方，超类的某些意图并未被良好理解，因此其中某些行为在子类内重复出现。还有一些地方，好几个子类做相同的事情，其实可以把它们搬到继承体系的上层去做。

这位顾问于是建议项目经理看看这些代码，把它们整理一下，但是项目经理并不热衷于此，毕竟程序看上去还可以运行，而且项目面临很大的进度压力。于是项目经理说，早些时候再抽时间做这些整理工作。

顾问也把他的想法告诉了在这个继承体系上工作的程序员，告诉他们可能发生的事情。程序员都很敏锐，马上就看出问题的严重性。他们知道这并不全是他们的错，有时候的确需要借助外力才能发现问题。程序员立刻用了一两天的时间整理好这个继承体系，并删掉了其中一半代码，功能毫发无损。他们对此十分满意，而且发现在继承体系中加入新的类或使用系统中的其他类都更快、更容易了。

项目经理并不高兴。进度排得很紧，有许多工作要做。系统必须在几个月之后发布，而这些程序员却白白耗费了两天时间，做的工作与未来几个月要交付的大量功能毫不相干。原先的代码运行起来还算正常。的确，新的设计更加“纯粹”、更加“整洁”。但项目要交付给客户的，是可以有效运行的代码，不是用以取悦学究的代码。顾问接下来又建议应该在系统的其他核心部分进行这样的整理工作，这会使整个项目停顿一至两个星期。所有这些工作只是为了让代码看起来更漂亮，并不能给系统添加任何新功能。

你对这个故事有什么感想？你认为这个顾问的建议（更进一步整理程序）是对的 吗？你会遵循那句古老的工程谚语吗：“如果它还可以运行，就不要动它。”

我必须承认自己有某些偏见，因为我就是那个顾问。6个月之后这个项目宣告失败，很大的原因是代码太复杂，无法调试，也无法将性能调优到可接受的水平。