Broadview
www.broadview.com.cn
PEARSON

传世经典书丛
Eternal Classics

# 重构：改善既有代码的设计
（评注版）

Martin Fowler
Kent Beck
[美] John Brant 著
William Opdyke
Don Roberts

评注 张逸


Refactoring:
Improving the Design of Existing Code

電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
http://www.phei.com.cn

# 重构：改善既有代码的设计（评注版）

Refactoring: Improving the Design of Existing Code


Martin Fowler
Kent Beck
[美] John Brant 著
William Opdyke
Don Roberts

张 逸 评注



電子工業出版社
Publishing House of Electronics Industry
北京•BEIJING

## 内 容 简 介

重构，一言以蔽之，就是在不改变外部行为的前提下，有条不紊地改善代码。多年前，正是本书原版的出版，使重构终于从编程高手们的小圈子走出，成为众多普通程序员日常开发工作中不可或缺的一部分。本书也因此成为与《设计模式》齐名的经典著作，被译为中、德、俄、日等众多语言，在世界范围内畅销不衰。

本书凝聚了软件开发社区专家多年摸索而获得的宝贵经验，拥有不因时光流逝而磨灭的价值。今天，无论是重构本身，业界对重构的理解，还是开发工具对重构的支持力度，都与本书最初出版时不可同日而语，但书中所蕴涵的意味和精华，依然值得反复咀嚼，而且往往能够常读常新。

本评注版力邀国内资深专家执笔，在英文原著基础上增加中文点评与注释，旨在以先行者的学研心得与实践感悟，对读者阅读与学习加以点拨、指明捷径。



**图书在版编目（CIP）数据**

重构：改善既有代码的设计=Refactoring:Improving the Design of Existing Code：评注版 /（美）福勒（Fowler,M.）等著；张逸评注. —北京：电子工业出版社，2011.6

（传世经典书丛）

ISBN 978-7-121-13450-0

Ⅰ.①重… Ⅱ.①福… ②张… 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（2011）第 081732 号

策划编辑：张春雨

责任编辑：徐律平

文字编辑：张丹阳 王 静

印 刷：

装 订：北京中新伟业印刷有限公司

出版发行：电子工业出版社

北京市海淀区万寿路 173 信箱 邮编 100036

开 本：787×980 1/16 印张：28.25 字数：675 千字

印 次：2011 年 6 月第 1 次印刷

定 价：69.00 元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：（010）88254888。

质量投诉请发邮件至 zlts@phei.com.cn，盗版侵权举报请发邮件至 dbqq@phei.com.cn。

服务热线：（010）88258888。

# 悦读上品 得乎益友

孔子云："取乎其上，得乎其中；取乎其中，得乎其下；取乎其下，则无所得矣"。

对于读书求知而言，这句古训教我们去读好书，最好是好书中的上品——经典书。其中，科技人员要读的技术书，因为直接关乎客观是非与生产效率，阅读选材本更应慎重。然而，随着技术图书品种的日益丰富，发现经典书越来越难，尤其对于涉世尚浅的新读者，更为不易，而他们又往往是最需要阅读、提升的重要群体。

所谓经典书，或说上品，是指选材精良、内容精练、讲述生动、外延丰盈、表现手法体贴入微的读品，它们会成为读者的知识和经验库中的重要组成部分，并且拥有从不断重读中汲取养分的空间。因此，选择阅读上品的问题便成了有效阅读的首要问题。当然，这不只是效率问题，上品促成的既是对某一种技术、思想的真正理解和掌握，同时又是一种感悟或享受，是一种愉悦。

与技术本身类似，经典 IT 技术书多来自国外。深厚的积累、良好的写作氛围，使一批大师为全球技术学习者留下了璀璨的智慧瑰宝。就在那个年代即将远去之时，无须回眸，也能感受到这一部部厚重而深邃的经典著作，在造福无数读者后从未蒙尘的熠熠光辉。而这些凝结众多当今国内技术中坚美妙记忆与绝佳体验的技术图书，虽然尚在国外图书市场上大放异彩，却已逐渐淡出国人的视线。最为遗憾的是，迟迟未有可以填补空缺的新书问世。而无可替代，不正是经典书被奉为圭臬的原因？

为了不让国内读者，尤其是即将步入技术生涯的新一代读者，就此错失这些滋养过先行者们的好书，以出版 IT 精品图书，满足技术人群需求为己任的我们，愿意承担这一使命。本次机遇惠顾了我们，让我们有机会携手权威的 Pearson 公司，精心推出"传世经典书丛"。

在我们眼中，"传世经典"的价值首先在于——既适合喜爱科技图书的读者，也符合专家们挑剔的标准。幸运的是，我们的确找到了这些堪称上品的佳作。丛书带给我们的幸运颇多，细数一下吧。

## 得以引荐大师著作

有恐思虑不周，我们大量参考了国外权威机构和网站的评选结果，并得到了 Pearson 的专业支持，又进

一步对符合标准之图书的国内外口碑与销售情况进行细致分析，也听取了国内技术专家的宝贵建议，才有幸选出对国内读者最富有技术养分的大师上品。

### 向深邃的技术内涵致敬

中外技术环境存在差异，很多享誉国外的好书未必适用于国内读者；且技术与应用瞬息万变，很容易让人心生迷惘或疲于奔命。本丛书的图书遴选，注重打好思考方法与技术理念的根基，旨在帮助读者修炼内功，提升境界，将技术真正融入个人知识体系，从而可以一通百通，从容面对随时涌现的技术变化。

### 翻译与评注的双项选择

引进优秀外版著作，将其翻译为中文供国内读者阅读，较为有效与常见。但另有一些外语水平较高、喜好阅读原版的读者，苦于对技术理解不足，不能充分体会原文表述的精妙，需要有人指导与点拨。而一批本土技术精英经过长期经典熏陶及实践锤炼，已足以胜任这一工作。有鉴于此，本丛书在翻译版的同时推出融合英文原著与中文点评、注释的评注版，供不同志趣的读者自由选择。

### 承蒙国内一流译(注)者的扶持

优秀的英文原著最终转化为真正的上品，尚需跨越翻译鸿沟，外版图书的翻译质量一直屡遭国内读者诟病。评注版的增值与含金量，同样依赖于评注者的高卓才具。好在，本丛书得到了久经考验的权威译(注)者的认可和支持，首肯我们选用其佳作，或亲自参与评注工作。正是他们的参与保证了经典的品质，既再次为我们的选材把关，更提供了一流的中文表述。

### 期望带给读者良好的阅读体验

一本好书带给人的愉悦不止于知识收获，良好的阅读感受同样不可缺少，且对学业不无助益。为让读者收获与上品相称的体验，我们在图书装帧设计与选材用料上同样不敢轻率，惟愿送到读者手中的除了珠玑章句，还有舒适与熨帖的视觉感受。

所有参与丛书出版的人员，尽管能力有限，却无不心怀严谨之心与完美愿望。如果读者朋友能从潜心阅读这些上品中偶有获益，不啻为对我们工作的最佳褒奖。若有阅读感悟，敬请拨冗告知，以鼓励我们继续在这一道路上贡献绵薄之力。如有不周之处，也请不吝指教。

电子工业出版社博文视点

# 评注者序

近十年来，若要讨论如何改进代码的质量，很难绕过 Martin Fowler 的这本经典著作。这本书已经影响了几代程序员，或许会持续不断地影响未来的一批程序员。遗憾的是，在现实中我们仍然看到了重构的步履维艰。一方面是因为程序员的重构技巧还有待磨砺，另一方面则是因为项目的压力，使得我们往往将重构视为鸡肋。

重构是可有可无的吗？*Practices of an Agile Developer* 的作者 Andy Hunt 认为“要投入时间和精力保持代码的整洁、敞亮。在项目中，代码应该是亮堂的，不应该有黑暗死角。”相信维护过遗留代码的程序员，在面对纠缠不清、复杂混乱的代码时，对于此言必有“心有戚戚焉”之感。重构是清理代码垃圾的有效手段，它有助于营造明晰的程序结构、一致的代码风格、有效的职责重用，能够恰如其分地在简单与复杂之间寻觅到代码之美，合理地权衡代码质量与开发效率，从而提升至优雅编码的境界。

如何呈现重构之美？关键在于发现丑陋而不堪忍受。对丑的憎恶实则是一种强悍的驱动力，它会驱使你运用重构，直到重构变成如呼吸一般自然而不可缺失。**重构无须专门的阶段，随时随地皆可进行**。经常的重构可以保证代码常拭常新，如利刃一般锋利。那种为了单一地追求开发速度，而置代码质量于不顾的做法，实则是杀鸡取卵，可以预期的结果就是系统的“破窗户”逐渐蔓延，最后落得不可收拾的下场。重构并非不可执行，关键还在于**我们对于重构的态度**和运用重构的习惯。必须在思想上认同重构的重要性；然后在技巧上不断提升重构技能，并辅以对重构工具的使用，就能最大程度地放大重构在软件开发中积极的一面。

若要提升重构技能，阅读本书就是必须，没有例外。那么，对于这样的经典之作而言，还有点评的必要吗？是否我们在做着画蛇添足的蠢事？Martin Fowler 已经做得足够好，任何点评都是一种饶舌？不尽然！身为点评者的我，如果仅满足于在本书的边边角角上，不痛不痒地发表不承担责任的语气词，那么对于这种点评，不要也罢。我在着手本书的点评工作时，对自己的定位是：我不是点评者，我是创作者。在点评本书的过程中，我是**以创作自己著作的态度来完成的**。

本书的大多数点评内容，并非片言只语，很多内容都是自己重构心得以及重构技巧的运用。我阅读参考了大量的书籍，例如《程序员修炼之道》、《重构与模式》、《领域驱动设计》、《反模式》、《软件架构的艺术》、《修改代码的艺术》、《代码整洁之道》等十余部相关书籍。毕竟站在软件世界的角度来看，Martin Fowler 写作本书的时代已经相当“古老”了。在这之后，产生了许多精彩的设计技巧、重构理念与方法。譬如在与重构相关的内容中，本书未曾论述的就包括：架构重构、界面重构、数据库重构、重构模式等内容。在点评过程中，我希望能以开放的方式描述自己阅读本书的感受，讲述自己重构的体验，并在能力范围之内给出自己的一些意见与看法。

我作为一名程序员，看到了太多漠视或者无视垃圾代码的同行们，这其中也包括曾经的我。但当我深陷遗留代码的痛苦时，对于这样的现状就变得痛心疾首了。近年来，软件业界对于软件工艺以及敏捷方法的推行，在一定程度上改善了人们的看法，但在实际推行中依旧困难重重。最大的起因就在于我们将程序员定位为“代码工人”，认为编码工作是低端程序员的责任。我们妄图创造如机械制造行业一般的“软件工厂”，却忽略了软件编程实则还有艺术的一面。我始终认为，代码仍然是架构的一部分，代码质量决定了架构质量，而重构则是延缓系统衰老的生命源泉。正所谓“千里之行始于足下”，就让重构之行，始于本书吧。

# 序

“重构”这个概念来自 Smalltalk 圈子，没多久就进入了其他语言阵营之中。由于重构是框架开发中不可缺少的一部分，所以当框架开发人员讨论自己的工作时，这个术语就诞生了。当他们精练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：重构具有风险。它必须修改运作中的程序，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星期的成果。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。我在《设计模式》书中和另外三位作者曾经提过：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衡量。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 Extract Method 和 Move Field 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，这就减少了过程中的风险。很快你就会把

这些重构手法和其名称加入自己的开发词典中，并且朗朗上口。

我第一次体验有讲究的、一次一小步的重构，是某次与 Kent Beck 在 30 000 英尺高空的飞行旅途中结对编程。我们运用本书收录的重构手法，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。我不但对最后结果更有信心，而且开发压力也小了很多。所以，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

《设计模式》第一作者，Eclipse 平台主架构师

熊节　译

# 前　言

从前，有位咨询顾问造访客户调研其开发项目。系统核心是个类继承体系，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，上层超类对于系统的运作做了一些假设，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。于是经理说，晚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，干的工作与要交付的多数功能毫无关系。原先的代码运行起来还算正常，他们的新设计看来有点过于追求完美。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完美东西。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。六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，也无法获得可被接受的性能。

后来，项目重新启动，几乎从头开始编写整个系统，Kent Beck 受邀做了顾问。他做了几件迥异以往的事，其中最重要的一件就是坚持以持续不断的重构行为来整理代码。这个项目的成功，以及重构在这个成功项目中扮演的角色，启发了我写这本书，如此一来我就能够把 Kent 和其他一些人已经学会的"以重构方式改进软件质量"的知识，传播给所有读者。

## 什么是重构

所谓重构（refactoring）是这样一个过程：在不改变代码外在行为的前提下，对代码做出修改，以改进程序的内部结构。重构是一种经千锤百炼形成的有条不紊的程序整理方法，可以最大限度地减少整理过程中引入错误的概率。本质上说，重构就是在代码写好之后改进它的设计。

"在代码写好之后改进它的设计"这种说法有点儿奇怪。按照目前对软件开发的理解，我们相信应该先设计而后编码：首先得有一个良好的设计，然后才能开始编码。但是，随着时间流逝，人们不断修改代码，于是根据原先设计所得的系统，整体结构逐渐衰弱。代码质量慢慢沉沦，编码工作从严谨的工程堕落为胡砍乱劈的随性行为。

"重构"正好与此相反。哪怕你手上有一个糟糕的设计，甚至是一堆混乱的代码，你也可以借由重构将它加工成设计良好的代码。重构的每个步骤都很简单，甚至显得有些过于简单：你只需要把某个字段从一个类移到另一个类，把某些代码从一个函数拉出来构成另一个函数，或是在继承体系中把某些代码

推上推下就行了。但是，聚沙成塔，这些小小的修改累积起来就可以根本改善设计质量。这和一般常见的“软件会慢慢腐烂”的观点恰恰相反。

通过重构，你可以找出改变的平衡点。你会发现所谓设计不再是一切动作的前提，而是在整个开发过程中逐渐浮现出来。在系统构筑过程中，你可以学习如何强化设计，其间带来的互动可以让一个程序在开发过程中持续保有良好的设计。

## 本书有什么

本书是一本为专业程序员而写的重构指南。我的目的是告诉你如何以一种可控制且高效率的方式进行重构。你将学会如何有条不紊地改进程序结构，而且不会引入错误，这就是正确的重构方式。

按照传统，图书应该以引言开头。尽管我也同意这个原则，但是我发现以概括性的讨论或定义来介绍重构，实在不是一件容易的事。所以我决定用一个实例做为开路先锋。第 1 章展示了一个小程序，其中有些常见的设计缺陷，我把它重构为更合格的面向对象程序。其间我们可以看到重构的过程，以及几个很有用的重构手法。如果你想知道重构到底是怎么回事儿，这一章不可不读。

第 2 章讨论重构的一般性原则、定义，以及进行重构的原因，我也大致介绍了重构所存在的一些问题。第 3 章由 Kent Beck 介绍如何嗅出代码中的“坏味道”，以及如何运用重构清除这些坏味道。测试在重构中扮演着非常重要的角色，第 4 章介绍如何运用一个简单而且开源的 Java 测试框架，在代码中构筑测试环境。

本书的核心部分——重构列表——从第 5 章延伸至第 12 章。它不能说是一份全面的列表，只是一个起步，其中包括迄今为止我在工作中整理下来的所有重构手法。每当我想做点什么——例如 *Replace Conditional with Polymorphism*（245）的时候，这份列表就会提醒我如何一步一步安全前进。我希望这是值得你日后一再回顾的部分。

本书介绍了其他人的许多研究成果，最后几章就是由他们之中的几位客串

所写的。Bill Opdyke 在第 13 章记述他将重构技术应用于商业开发过程中遇到的一些问题。Don Roberts 和 John Brant 在第 14 章展望重构技术的未来——自动化工具。我把最后一章（第 15 章）留给重构技术的顶尖大师 Kent Beck 来压轴。

### 在 Java 中运用重构

本书范例全部使用 Java 撰写。重构当然也可以在其他语言中实现，而且我也希望这本书能够给其他语言使用者带来帮助。但我觉得我最好在本书中只使用 Java，因为那是我最熟悉的语言。我会不时写下一些提示，告诉读者如何在其他语言中进行重构，不过我真心希望看到其他人在本书的基础上针对其他语言写出更多重构方面的书籍。

为了很好地与读者交流我的想法，我没有使用 Java 语言中特别复杂的部分。所以我避免使用内嵌类、反射机制、线程以及很多强大的 Java 特性。这是因为我希望尽可能清楚地展现重构的核心。

我应该提醒你，这些重构手法并不针对并发或分布式编程。那些主题会引出更多的考虑，本书并未涉及。

## 谁该阅读本书

本书的目标读者是专业程序员，也就是那些以编写软件为生的人。书中的示例和讨论，涉及大量需要详细阅读和理解的代码。这些例子都以 Java 写成。之所以选择 Java，因为它是一种应用范围越来越广的语言，而且任何具备 C 语言背景的人都可以轻易理解它。Java 是一种面向对象语言，而面向对象机制对于重构有很大帮助。

尽管关注对象是代码，但重构对于系统设计也有巨大影响。资深设计师和架构师也很有必要了解重构原理，并在自己的项目中运用重构技术。最好是由老资格、经验丰富的开发人员来引入重构技术，因为这样的人最能够透彻理解重构背后的原理，并根据情况加以调整，使之适用于特定工作领域。如果你使用的不是 Java，这一点尤其重要，因为你必须把我给出的范例以其他语言改写。

下面我要告诉你，如何能够在不通读全书的情况下充分用好它。

- 如果你想知道重构是什么，请阅读第 1 章，其中示例会让你清楚重构的过程。
- 如果你想知道为什么应该重构，请阅读第 1、2 章。它们告诉你重构是什么以及为什么应该重构。
- 如果你想知道该在什么地方重构，请阅读第 3 章。它会告诉你一些代码特征，这些特征指出“这里需要重构”。
- 如果你想着手进行重构，请完整阅读第 1～4 章，然后选择性地阅读重构列表。一开始只需概略浏览列表，看看其中有些什么，不必理解所有细节。一旦真正需要实施某个准则，再详细阅读它，从中获取帮助。列表部分是供查阅的参考性内容，你不必一次就把它全部读完。此外你还应该读一读列表之后其他作者的“客串章节”，特别是第 15 章。

## 站在前人的肩膀上

就在本书一开始的此时此刻，我必须说：这本书让我欠了一大笔人情债，欠那些在过去十年中做了大量研究工作并开创重构领域的人一大笔债。这本书原本应该由他们之中的某个人来写，但最后却是由我这个有时间有精力的人捡了便宜。

重构技术的两位最早倡导者是 Ward Cunningham 和 Kent Beck。他们很早就把重构作为开发过程的一个核心成分，并且在自己的开发过程中运用它。尤其需要说明的是，正因为和 Kent 的合作，才让我真正看到了重构的重要性，并直接激励了我写出这本书。

Ralph Johnson 在 UIUC（伊利诺伊大学厄巴纳-尚佩恩分校）领导了一个小组，这个小组因其在对象技术方面的实际贡献而声名远扬。Ralph 很早就是重构技术的拥护者，他的一些学生也一直在研究这个课题。Bill Opdyke 的博士论文是重构研究的第一份详细的书面成果。John Brant 和 Don Roberts 则早已不满足于写文章了，他们写了一个工具叫 Refactoring Browser（重构浏览器），对

Smalltalk 程序实施重构工程。

## 致谢

尽管有这些研究成果可以借鉴，我还是需要很多协助才能写出这本书。首先，并且也是最重要的，Kent Beck 给了我巨大的帮助。Kent 在底特律的某个酒吧和我谈起他正在为 *Smalltalk Report* 撰写一篇论文[Beck，hanoi]，从此播下本书的第一颗种子。那次谈话不但让我开始注意到重构技术，而且我还从中“偷”了许多想法放到本书第 1 章。Kent 也在其他很多方面帮助我，想出“代码味道”这个概念的是他，当我遇到各种困难时，鼓励我的人也是他，常常和我一起工作助我完成这本书的，还是他。我常常忍不住这么想：他完全可以自己把这本书写得更好。可惜有时间写书的人是我，所以我也只能希望自己不要做得太差。

写这本书的时候，我希望能把一些专家经验直接与你分享，所以我非常感激那些花时间为本书添砖加瓦的人。Kent Beck、John Brant、William Opdyke 和 Don Roberts 编撰或合写了本书部分章节。此外 Rich Garzaniti 和 Ron Jeffries 帮我添加了一些有用的文中注解。

在任何一本此类技术书里，作者都会告诉你，技术审阅者提供了巨大的帮助。一如既往，Addison-Wesley 出版社的 Carter Shanklin 和他的团队组织了强大的审稿人阵容，他们是：

- Ken Auer，Rolemodel 软件公司
- Joshua Bloch，Sun 公司 Java 软件部
- John Brant，UIUC
- Scott Corley，High Voltage 软件公司
- Ward Cunningham，Cunningham & Cunningham 公司
- Stéphane Ducasse
- Erich Gamma，对象技术国际公司

- Ron Jeffries
- Ralph Johnson，伊利诺伊大学
- Joshua Kerievsky，Industrial Logic 公司
- Doug Lea，纽约州立大学 Oswego 分校
- Sander Tichelaar

他们大大提高了本书的可读性和准确性，并且至少去掉了一些任何手稿都可能会藏有的错误。在此我要特别感谢两个效果显著的建议，它们让我的书看上去耳目一新：Ward 和 Ron 建议我以重构前后效果并列对照的方式写第 1 章，Joshua Kerievsky 建议我在重构列表中画出代码草图。

除了正式审阅小组，还有很多非正式的审阅者。这些人或看过我的手稿，或关注我的网页并留下对我很有帮助的意见。他们是 Leif Bennett，Michael Feathers，Michael Finney，Neil Galarneau，Hisham Ghazouli，Tony Gould，John Isner，Brian Marick，Ralf Reissing，John Salt，Mark Swanson，Dave Thomas 和 Don Wells。我相信肯定还有一些被我遗忘的人，请容我在此向你们道歉，并致上我的谢意。

有一个特别有趣的审阅小组，就是“恶名昭彰”的 UIUC 读书小组。本书反映出他们的众多研究成果，我要特别感谢他们用录音记录的意见。这个小组成员包括 Fredrico“Fred”Balaguer，John Brant，Ian Chai，Brian Foote，Alejandra Garrido，Zhijiang“John”Han，Peter Hatch，Ralph Johnson，Songyu“Raymond”Lu，Dragos-Anton Manolescu，Hiroaki Nakamura，James Overturf，Don Roberts，Chieko Shirai，Les Tyrell 和 Joe Yoder。

任何好想法都需要在严酷的生产环境中接受检验。我看到重构对于克莱斯勒综合薪资系统（Chrysler Comprehensive Compensation，C3）发挥了巨大的作用。我要感谢那个团队的所有成员：Ann Anderson，Ed Anderi，Ralph Beattie，Kent Beck，David Bryant，Bob Coe，Marie DeArment，Margaret Fronczak，Rich Garzaniti，Dennis Gore，Brian Hacker，Chet Hendrickson，Ron Jeffries，Doug Joppie，David Kim，Paul Kowalsky，Debbie Mueller，Tom Murasky，Richard

Nutter，Adrian Pantea，Matt Saigeon，Don Thomas 和 Don Wells。和他们一起工作所获得的第一手数据，巩固了我对重构原理和作用的认识。他们使用重构技术所取得的进步极大程度地帮助我看到：重构技术应用于历时多年的大型项目中，可以起到何等的作用！

再提一句，我得到了 Addison-Wesley 出版社的 J. Carter Shanklin 及其团队的帮助，包括 Krysia Bebick、Susan Cestone、Chuck Dutton、Kristin Erickson、John Fuller、Christopher Guzikowski、Simone Payment 和 Genevieve Rajewski。与优秀出版商合作是一个令人愉快的经历，他们为我提供了大量的支持和帮助。

谈到支持，为一本书付出最多的，总是距离作者最近的人。那就是现在已成为我妻子的 Cindy。感谢她，当我埋首工作的时候，还是一样爱我。即使在我投入写书时，也总会不断想起她。

Martin Fowler
于马萨诸塞州 Melrose 市
fowler @acm.org
http://www.martinfowler.com
http://www.refactoring.com

熊节　译