TURING 图灵程序设计丛书 程序员修炼系列

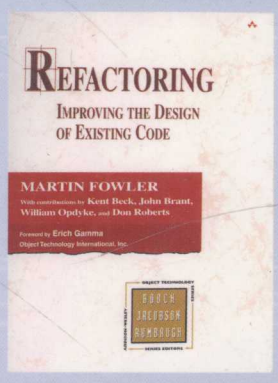
Addison Wesley

Refactoring Improving the Design of Existing Code

重构

改善既有代码的设计 英文注释版

[美] Martin Fowler 著



- 软件开发的不朽经典
- 生动阐述重构原理和具体做法
- 新添大量重构方法，使你与时俱进
- 丰富的词汇和背景注释，助你轻松读经典

人民邮电出版社
POSTS & TELECOM PRESS

TURING 图灵程序设计丛书 程序员修炼系列

Refactoring
Improving the Design of Existing Code

重构
改善既有代码的设计

英文注释版

[美] Martin Fowler 著

人民邮电出版社
北 京

图书在版编目（CIP）数据

重构：改善既有代码的设计：英文注释版 /（美）福勒（Fowler, M.）著. —北京：人民邮电出版社，2008.2
（图灵程序设计丛书）
ISBN 978-7-115-16804-7

I. 重… II. 福… III. 软件开发—英文 IV. TP311.62

中国版本图书馆 CIP 数据核字（2007）第 140937 号

内 容 提 要

本书清晰地揭示了重构的过程，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70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你一次一小步地修改你的代码，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。

图灵程序设计丛书

重构：改善既有代码的设计（英文注释版）

◆ 著 [美] Martin Fowler
责任编辑 傅志红

◆ 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 电子函件 315@ptpress.com.cn
网址 http://www.ptpress.com.cn
三河市海波印务有限公司印刷
新华书店总店北京发行所经销

◆ 开本：800×1000 1/16
印张：30.75
字数：590 千字 2008 年 2 月第 1 版
印数：1 – 3 000 册 2008 年 2 月河北第 1 次印刷

著作权合同登记号 图字：01-2006-3662 号

ISBN 978-7-115-16804-7/TP

定价：69.00 元

读者服务热线：(010)88593802 印装质量热线：(010)67129223
反盗版热线：(010)67171154

版 权 声 明

出 版 说 明

经过半年多的努力，“图灵程序设计丛书”的新成员——几本英文注释版图书终于与大家见面了。

本次出版的《重构》、《企业应用架构模式》、《敏捷软件开发》（Java 版和 C#版）和《程序员修炼之道》五部著作都是软件行业公认的为数不多的真正的经典之作，如果排除特定于具体语言、工具和技术的图书，它们可以毫无疑问地入选所有软件开发人员必读技术书目的前十名（至于其他候选者，脑中闪过《设计模式》、《人月神话》、《计算机程序设计艺术》、《编程珠玑》、《代码大全》……）。

这其中，**《重构》**曾经与《设计模式》、《反模式》（中文版即将由人民邮电出版社出版）、《解析极限编程——拥抱变化》并称为软件工程四大名著。Martin Fowler 虽然不是重构技术的创造者，但是由于 Kent Beck、John Brant、William Opdyke 和 Don Roberts 等先驱的襄助，本书也成为实至名归的开创性著作。由于近年来工具支持的加强，重构已经越来越成为开发人员日常不可或缺的技术。**《企业应用架构模式》**曾经荣获 2003 年 Software Development 杂志读者选择大奖和 Jolt 生产效率奖，是 Martin Fowler 的另一部名著，某种意义上也是真正奠定他在技术界中地位的一部重要著作。如果说《设计模式》是程序设计层次的圣经，《面向模式的软件架构》是架构设计层次的圣经的话，此书则当之无愧地可以称为企业级应用的圣经。企业级开发已成为主流，其中的困难与挑战是显而易见的，而这一领域的图书一直不多，所以此书更加弥足珍贵。Fowler 的这两部著作都是对业界多年积累的经验的总结，实战性非常强，不仅使你知其所以然，更让你知道如何实现、各种实现方式的适用场合、实现中需要注意哪些问题，等等。书总体上采用教程+参考的形式，教程部分只有 100 页左右，实际阅读时，先精读此部分，参考部分可以通过边干边学方式学习，精华尽在此中。

熟悉最近大红大紫的 Ruby on Rails 的读者，一定知道 Andrew Hunt 和 David Thomas 以及他们创办的 The Pragmatic Programmers 公司，他们撰写和出版的 *Programming Ruby* 和 *Agile Web Development with Rails* 两本书是 RoR 关键性的推动力量。而这个如今大名鼎鼎的公司的名字就出自**《程序员修炼之道》**一书（英文版原名即 *The Pragmatic Programmer*）。此书从各个方面来看都令人称奇：它当然是一本技术图书，但是字里行间弥漫着的浓浓的人文气息、哲理性和幽默感，又时常让我们产生疑惑；它的篇幅不大（只有 300 来页），然而几乎涉及了软件开发和程序员生涯的一切——责任心、学习方法、思考方法、沟通、设计原则、版本控制、代码生成、

按契约设计、调试、测试、估算、并发、重构、需求、文档、各种工具……甚至包括如何发邮件、如何在 Windows 上使用 Unix 命令；内容层次跨度也极大，既可以高到团队组建，也可以深至代码级的具体细节，甚至还有不少编程习题！阅读此书的时候，常常会好奇，作者是何方神圣，能够将如此之多之丰富的内涵如此完美地熔于一炉。

《**敏捷软件开发**》则是另一位业界大师 Robert Martin（人称 Bob 大叔）的代表作品，Java 版（也含有不少 C++代码）荣获 2003 年 Jolt 大奖，去年年中又出版了 C#版，由 Robert 与他的儿子 Micah 合作，主要改动除了将代码换成 C#之外，还增加了 UML 建模、MVC 模式等方面的内容，脉络更加清晰。此书以“敏捷”命名，其实有很大的误导性，它的副标题“原则、模式和实践”才算名副其实，估计这也是原版 C#版书名改为 *Agile Principles, Patterns, and Practices in C#*的原因。原因很简单，此书只用了 100 页左右的篇幅概述敏捷开发方法，主体部分主要是对面向对象诸原则和 GoF 模式的阐释，其实是一部非常优秀的面向对象设计、实现和设计模式图书（C#版还是很好的 UML 教程），语言通俗有趣，而且实战性很强。书中的许多绝妙插图，使我们在会意一笑中，更深地理解一些原本艰涩的技术概念。如果你阅读《设计模式》（机械工业出版社）一书感觉有困难，或者在阅读《设计模式解析》（人民邮电出版社）之后需要在实际开发环境中进一步领悟模式，那么本书将是绝佳选择。

经典之所以成为经典，既在于它们不仅是业界大师的作品，凝聚了软件开发社区集体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；更因为它们很难一遍就领会其所有意蕴和精华，需要反复阅读，而且往往能够常读常新——某种意义上，你的阅读所得与你的经历是直接相关的，在不同阶段会有不同感受和收获，这也是前人说“少不读水浒”和“少不读杜甫”的原因。与此同时，经典的翻译往往难以尽善尽美，事实上这几部著作的中译本都多多少少存在各种问题，有的问题还非常严重。美国大诗人 Robert Frost 曾经说过，“Poetry is what gets lost in translation（诗歌就是在翻译中失去的东西）”，我们在长期的技术图书翻译和编审工作中也充分体会到，经典原著的许多关键的意境、暗示往往是难以用另外一种文字来表达的。有好的译本，当然能够起到事半功倍的作用，但是即便如此，直接或者对照着阅读原著仍然是有所裨益甚至必不可少的。而且，掌握英语，具备良好的英文技术文献阅读能力，也是今天平坦世界（读过《世界是平的》吗？）中一名专业软件开发人员必备的素养，而阅读名著原版，绝对是一种一举两得的英语进阶方式。

我们之所以在这几部著作大多出版多年（最早的原版初版于 1999 年），已经有了翻译版，甚至此前曾经还有过影印版行世的情况下，仍然下决心再次出版英文版，主要原因正是它们的经典性和长效性。事实上，这些书原版到现在确实依旧长销不衰，我们手上拿到的原版样书，无不已经是最近的印刷，印次达到十几次甚至二十几次；在 Amazon 等主要的网络书店上，它们也仍然位居销售榜前列，让许多热门的新书后辈也难以望其项背；在巴诺和鲍德斯这样的大型连锁书店，你会看到这些著作依旧被摆放在最显著的位置，心中涌起一种感动。反观国内

市场，它们的英文版几乎都已经绝版，更有中文版也绝版的奇怪现象。我们希望这批经典的出版，能够打破这一怪圈。

本次出版的英文注释版，除了按原版版权方的要求翻译了前言、序等文字，并原汁原味地保留了原书的正文部分之外，还在多位业界专家的大力支持下，利用页边和页脚的空白增加了一些注释，算是一种新的尝试，力求为读者能够提供更多的价值。

增加的注释主要是以下几种情况：

1. 直接注释单词。其目的就是方便读者，能够少查字典。事实上，我们在阅读（以及翻译）英语技术文献时，所遇到的直接困难并不是技术上的，而更多来自英语本身。因此我们所注释的词语或者语句，许多并不是术语，而是通用的单词或者习语。在注释时，我们并非简单地按字典的常见义项给出，而是充分参考原著的上下文，给出特定语境下的对应词。

2. 章节标题给出翻译，目录也改为双语对应。

3. 提示主题。其作用类似于一些图书页边的 recap（扼要重述或重点提示），有助于读者“在脑中读出目录来”（这是 Ruby 专家 Obi Fernandez 对阅读技术图书的建议）。

4. 知识更新和扩展。软件研发技术的发展日新月异，时光毕竟会在这些经典身上留下一些痕迹，我们力争根据注释时的最新技术进展为读者提供新的信息。这个工作并不容易，像《重构》这样作者一直维护着网站（即 refactoring.com）的实在不多，《企业应用架构模式》和《程序员修炼之道》还算有勘误，而《敏捷软件开发》的网页上只有书的一些摘录、评论和到 Amazon 的链接，连勘误都没有。好在，经典毕竟是经典，这方面需要做的工作总量倒是不大。

按我们最初的设计，如果能在注释版中增加一些导读性、读书心得式的文字，应该会更加完美，但是非常遗憾，由于工作量大、难以找到合适的人选、尺度很难把握等等原因，这次虽然做出了尝试，但并不令人满意。我们计划在图灵网站（www.turingbook.com）上提供类似的信息，构建一个“一起读经典”的社区阅读环境。分享是软件开发的原动力，我们期盼有更多的读者和业界专家能够以各种方式加入进来。

这种注释版是一种尝试，由于注释者水平和个人喜好各异，各本书的尺度也会有所差异，由于能力和时间问题，肯定会有各种各样的疏漏和讹误，欢迎大家批评指正。我们的报错信箱是：errata@turingbook.com。同时也可以在图灵网站各本书的配套网页上提交勘误。

图灵编辑部

2007年9月

序

重构（refactoring）的概念出自Smalltalk界，但很快就进入了其他语言阵营。重构在框架开发中不可或缺，所以这个术语是框架开发人员在讨论自己的技艺时油然而生的。当他们改善自己的类层次结构时，当他们热烈讨论又可以删去多少行代码时，重构的概念就产生了。设计者知道，框架不可能第一次就完美无缺，它必须随着设计者的经验增长而改进；设计者也知道，阅读和修改代码的次数远远多于编写代码。要使代码易读、易修改，秘诀就在于重构——对框架而言如此，对一般软件也如此。

那么，重构有什么问题吗？简而言之，重构是有风险的。它要修改的是可以工作的程序，这可能引入一些微妙的错误。如果重构不当，可能使数天乃至数星期的工作前功尽弃。如果重构时有章不循、草率为之，风险将更大。开始深入自己的代码时，很快就会发现一些需要修改的地方，于是你更加深入。探究越深，找到的重构机会就越多……于是修改也越多。最后你挖了一个大坑，自己却爬不出去了。为了避免自掘坟墓，重构必须按部就班地进行。我和合著者写《设计模式》一书时曾提到：设计模式为重构提供了目标。然而“确定目标”只是问题之一，转换程序达到目标，是另一个难题。

Martin Fowler和本书另几位作者清楚阐述了重构过程，他们为面向对象软件开发做出了重大贡献。本书解释了重构的原理和最佳实践，并指出在什么场合应该开始深入研究代码，进行改进。本书的核心部分是一个完整的重构目录，其中每一个重构都介绍了一种经过验证的代码转换的动机和做法。某些重构，如Extract Method和Move Field乍看起来显而易见，但切不可掉以轻心，因为理解这种重构的做法正是有条不紊地进行重构的关键。本书中的重构将帮助你每次一小步地修改代码，从而减少了设计修改过程中的风险。很快，这些重构的名字就会成为你日常的开发词汇。

我第一次体验严格的、每次一小步的重构，是在30 000英尺高空与Kent Beck进行结对编程（pair-programming）[①]。我们应用本书中的重构，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，而且感到开发时自己的压力也小了很多。所以，我强烈推荐你尝试这些重构，你的生活和你的程序都将因此更美好。

——*Erich Gamma*[②]

---

① Gamma此处指的是1998年与Beck在前往OOPSLA会议的飞机上的一次合作。其结果就是单元测试框架JUnit。Kent Bemt是极限编程的创始人，也是设计模式、重构、测试驱动开发等技术的先驱。所谓结对编程，是一种极限编程实践，两位程序员共同在一台电脑上完成代码编写，一人编程时，另一人在旁观察思考。

② Erich Gamma是《设计模式》一书的第一作者，Eclipse平台的主架构师，目前在领导团队协作平台的Jazz的开发。

前　言

从前，有一位顾问受邀为一个软件开发项目做咨询。这个顾问看了看开发人员已编写的一些代码，他发现系统的中心是一个类层次结构。浏览这个结构时，他发现它非常凌乱，上层类对类的工作方式做了一些假设，这些假设都体现在继承代码中了。但是这些假设并不适合所有子类，因此子类中不得不大量地重定义（override）。只要在超类内做点修改，就可以减少许多重定义。在另一些地方，超类的某些意图并未被很好地理解，因此超类的有些行为在子类内重复了。还有一些地方，好几个子类做相同的事情，显然可以把这些代码移到类层次更上层去。

这位顾问于是建议项目经理先查看并清理代码，但是经理似乎不以为然，毕竟程序看上去还能运行，而且项目面临的进度压力很大。于是经理只是说，过些时候再抽时间做。

顾问也把发现的情况告诉了开发这个类层次的程序员。程序员都很敏锐，看出了问题的严重性。他们知道这并不全是自己的错，有时候的确是旁观者清，当局者迷。因此程序员用了一两天清理类层次，完成时删掉了其中一半代码，而功能并未减少。他们对此十分满意，而且发现现在在类层次中加入新类或使用系统中其他的类更快也更容易了。

项目经理可不那么高兴。进度很紧，还有很多工作要做，这些程序员白白耗费两天时间，干的工作与系统几个月后要交付的多数功能毫无关系。原先的代码不是运行得很正常吗，而新设计不过就是纯了一点，清晰了一点。项目的目的是交付可以有效运行的代码，而不是要让一个学究满意。顾问接下来又建议应该类似地清理系统的其他核心部分，这会使整个项目停顿一两个星期。所有这些工作似乎只是为了让代码看起来更漂亮，而不能给系统添加任何新功能。

对这个故事你有什么感想？你认为这个顾问建议进一步清理程序对吗？或者你认为应该遵循那句古老的工程谚语：“如果能运行，就不要去修”？

我必须承认自己有偏见，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了，无法调试，也无法达到可以接受的性能。

后来，另一个顾问Kent Beck受邀重启项目，几乎从头开始重写整个系统。他的许多做法都与众不同，其中最重要的就是坚持以重构持续不断地清理代码。这个项目的成功，以及重构在成功中扮演的角色，启发了我写作这本书，从而能够把Kent和其他人在使用重构改进软件质量中所获得的知识传播出去。

什么是重构

所谓重构是一种修改软件系统的过程，“它必须在不改变代码外在行为的情况下，改进程序的内部结构”。重构是一种训练有素、有条不紊的代码清理方式，需要尽量减少引入错误的机率，

本质上，重构就是在编写代码之后再改进设计。

“在编写代码之后再改进设计”？这种说法真奇怪，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。先有好的设计，然后才能开始编码。但是，随着时间流逝，代码在不断修改，于是系统的完整性、系统按原先设计所得的结构都逐渐减弱。代码慢慢腐化，编码也从工程沦为无章可循的乱来。

“重构”与这种做法正好相反。即使设计很糟糕，甚至一片混乱，你也可以通过重构将它改进为设计良好的代码。重构的每个步骤都很简单，甚至有些过分简单了：把某个字段（field）从一个类移到另一个类，把某些代码从一个方法（method）拉出来构成另一个方法，或是把一些代码在类层次中推上推下。但是，正所谓集腋成裘，这些小型修改的累积效应可以根本地改善设计。这和一般常见的“软件腐化”的观点恰恰相反。

通过重构，你可以找出改变的平衡点。你会发现设计不再是最先做出的，而是在开发过程中逐渐浮现出来。应该在构建系统的过程中学习如何改善设计。这种互动可以使一个程序的设计在开发过程中始终保持良好。

本书内容

本书是一本重构指南，针对职业程序员。我的目的是告诉你如何以一种可控且高效的方式进行重构。你将学会这样的重构方式：不在代码中引入错误，而是按部就班地改进结构。

按传统，书应该以一个简介开头。虽然我也同意这个原则，但是我发现概括地讨论或定义来介绍重构并不容易。所以我决定从一个实例开始。第1章给出一个含有一些常见设计缺陷的小程序，我把它重构为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的具体应用。如果你想知道重构到底是怎么回事，这一章非常重要。

第2章讲述重构的一般性原则、定义以及进行重构的原因，还概略讨论了重构存在的一些问题。第3事由Kent Beck介绍如何寻找代码中的“坏味”（bad smell），以及如何运用重构清除这些坏味。“测试”在重构中扮演非常重要的角色，第4章介绍如何使用一个简单的开源Java测试框架，在代码中构建测试。

从第5章至第12章是本书的核心部分——重构目录。这不是一份全面的目录，只是一个起点，其中包括迄今为止我在工作中整理下来的所有重构。每当我想做点什么（例如Replace Conditional with Polymorphism）的时候，这份目录就会提醒我如何安全地循序前进。我希望这部分你日后会一再查阅。

本书介绍了其他人的许多研究成果，最后几章就是特邀他们之中的几位撰写的。Bill Opdyke在第13章记述了他在商业开发中应用重构技术遇到的一些问题。Don Roberts和John Brant在第14章展望了重构技术的未来——自动化工具，最后的总结（第15章）留给了重构技术的大师Kent Beck。

所用语言

本书实例全部以Java编写。重构当然也可以在其他语言中使用，而且我也希望本书对其他语言的使用者也有用。但我觉得最好在本书中只使用Java，因为那是我最熟悉的语言。我会不时就如何在其他语言中进行重构添加一些提示，不过我真心希望其他人能以本书为基础针对其他语言写出更多重构书籍。

为了有助于表达思想，我不想使用Java语言中特别复杂的部分。所以我避免使用内部类、反射机制、线程以及许多其他更强大的Java特性。这是因为我希望尽可能清楚地说明重构的核心。

我应该提醒你，这些重构没有考虑并发（concurrent）或分布式（distributed）编程。那些主题会引出更多问题，不在本书范围之内。

谁该阅读本书

本书针对的是职业程序员，也就是那些以编写软件为职业的人。书中的示例和讨论，涉及大量需要详细阅读和理解的代码。这些例子都用Java编写。之所以选择Java，因为它是一种越来越流行的语言，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。

尽管关注的是代码，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，并在项目中运用重构技术。重构最好由德高望重、经验丰富的开发人员来引入，因为这样的人能够最好地理解重构背后的原理，并适用于特定工作环境。如果你使用Java以外的语言，这一点尤其必要，因为你必须把我给出的示例用其他语言改写。

如果想在不遍读全书的情况下获益最多，请按以下步骤进行。

- **如果想理解重构是什么**，请读第1章。其中示例会让你清楚重构过程。
- **如果想知道为什么重构**，请读前两章。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**，请读第3章。它会告诉你一些表示需要重构的代码特征。
- **如果你想实际进行重构**，请完整阅读前四章，然后浏览重构目录。先了解目录中大致有些什么，不必理解所有细节。一旦真正需要实施某个重构，详细阅读相应部分，让它帮助你实践。目录是一种参考章节，你不需要一次把它全部读完，此外你还应该读一读特邀专家撰写的章节，特别是第15章。

站在前人的肩膀上

现在，我必须说明，这本书让我欠下许多人情，必须感谢过去十年中付出心血开创了重构领域的人。如果由他们之中的某个人来写本书应该更理想，但最后写书的却是我这个既有时间

又有精力的人。

重构技术的两位最早先驱是Ward Cunningham和Kent Beck。他们在早年就把重构作为开发过程的核心之一，并且调整了自己的开发过程以应用重构。尤其需要说明的是，和Kent的合作使我真正看到了重构的重要性，并直接启发我写了这本书。

Ralph Johnson在伊利诺伊大学厄巴纳—尚佩恩分校领导了一个小组，这个小组因其对对象技术的实际贡献而闻名。Ralph是重构技术的长期拥护者，他的一些学生也曾研究这个课题。Bill Opdyke的博士论文是重构方面第一份详细的书面文献。John Brant和Don Roberts走得更远，他们写了一个工具——Refactoring Browser，用于重构Smalltalk程序。

致谢

尽管有这些研究成果可借鉴，我还是需要很多帮助才能写出这本书，首先，并且也是最重要的，Kent Beck给予了我巨大帮助。本书最早可以追溯到Kent在底特律的酒吧和我谈起他正在为*Smalltalk Report*杂志撰写一篇文章[Beck，hanoi]。谈话中许多想法都被我“偷”到本书第1章了，而且这次谈话还促使我开始做一些重构笔记。Kent在其他方面也提供了帮助。提出“代码坏味”这个概念的是他；当我遇到各种困难时，鼓励我的人也是他；一直助我完成这本书的，还是他。我常常忍不住想，如果他自己来写的话，完全可以写得更好。可惜有时间写书的人是我，所以我希望自己不辱使命。

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加材料的人。Kent Beck、John Brant、William Opdyke和Don Roberts撰写了本书部分章节。此外Rich Garzaniti和Ron Jeffries帮我添加了一些有用的旁注。

任何一位作者都会告诉你，技术审稿人对这样一本书提供了巨大的帮助。和往常一样，Addison-Wesley的Carter和他的团队组织了强大的审稿人阵容。他们是：

- Ken Auer，Rolemadel软件公司
- Joshua Bloch，Sun公司Java软件部
- John Brant，伊利诺伊大学
- Scott Corley，High Voltage软件公司
- Ward Cunningham，Cunningham & Cunningham公司
- Stéphane Ducasse
- Erich Gamma，Object Technology International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
- 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有的潜在错误。在此我要特别感谢两个效果显著的建议，这两个建议让我的书看上去耳目一新：Ward和Ron建议我以重构前后效果（包括代码和UML图）并列的方式写第1章，Joshua建议我在重构目录中画出代码梗概（code sketch）。

除了正式审稿人之外，还有很多非正式的审稿人。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人被我遗忘了，请允许我在此致歉，并致谢意。

有一个特别有趣的审稿小组，就是“臭名昭著”的伊利诺伊大学读书小组。由于本书反映了他们的众多工作，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。

任何好的想法都需要在严酷的产品环境中接受检验。我看到重构对于克莱斯勒综合薪酬（Chrysler Comprehensive Compensation，C3）系统产生了巨大的影响。我要感谢该开发团队的所有成员：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信息，加深了我对重构原理和优点的认识。观察他们使用重构技术所取得的进展，帮助我清楚地看到：重构技术应用于历时多年的大型项目中，可以起到怎样的作用。

再一次，我得到了Addison-Wesley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种令人愉快的经验，他们提供了大量的支持和帮助。

谈到支持，为一本书付出最多的，总是与作者最亲密的人。对我来说，那就是我的妻子Cindy。感谢她，当我埋首工作的时候，还是一样爱我。我虽在此书上投入了大量时间，但也总是时刻想念她。

Martin Fowler
Melrose, Massachusetts
fowler@acm.org
http://www.martinfowler.com
http://www.refactoring.com

Contents

① 重构前标*的，表示根据Refactoring.com网站做了更新。