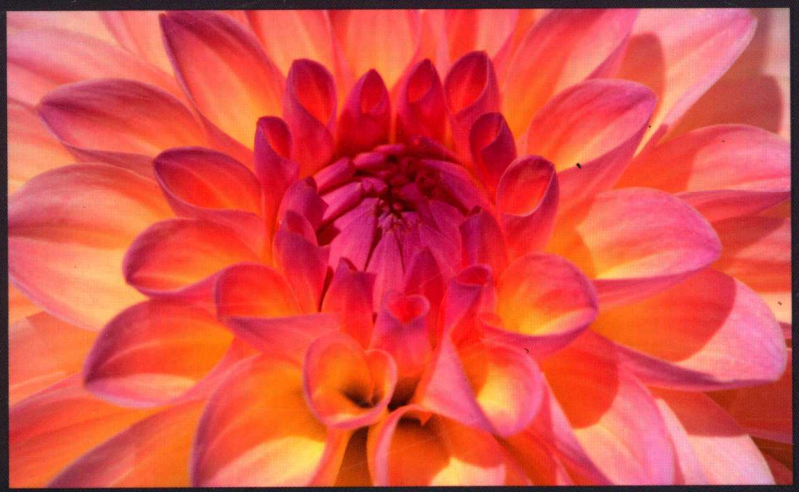
Pearson

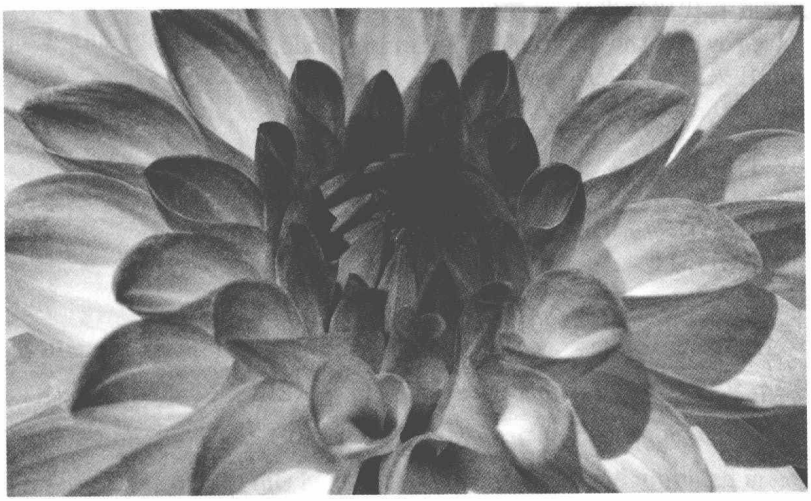


重构
改善既有代码的设计
（第2版·英文版）

[美] 马丁·福勒（Martin Fowler） 著

Refactoring
Improving the Design of Existing Code, Second Edition

中国工信出版集团
人民邮电出版社
POSTS & TELECOM PRESS



重构

改善既有代码的设计

（第2版·英文版）

[美] 马丁·福勒（Martin Fowler） 著

Refactoring

Improving the Design of Existing Code, Second Edition

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 : 第2版 =
Refactoring: Improving the Design of Existing Code,
Second Edition : 英文 / (美) 马丁·福勒
(Martin Fowler) 著. -- 北京 : 人民邮电出版社,
2019.5
ISBN 978-7-115-51008-2

Ⅰ. ①重… Ⅱ. ①马… Ⅲ. ①机器代码程序－程序设计－英文 Ⅳ. ①TP311.11

中国版本图书馆CIP数据核字(2019)第053190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）
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
三河市君旺印务有限公司印刷

◆ 开本：800×1000 1/16
印张：27
字数：513千字 2019年5月第1版
印数：1－3 000册 2019年5月河北第1次印刷

著作权合同登记号 图字：01-2019-0848号

定价：128.00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

版权声明

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 在万米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，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，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。

后来，这个项目重新启动，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。这个团队效能的提升，以及重构在其中扮演的角色，启发了我撰写本书的第 1 版，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。

自本书第 1 版问世至今，读者的反馈甚佳，重构的理念已经被广泛接纳，成为编程的词汇表中不可或缺的部分。然而，对于一本与编程相关的书而言，18 年已经太漫长，因此我感到，是时候回头重新修订这本书了。我几乎重写了全书的每一页，但从其内涵而言，整本书又几乎没有改变。重构的精髓仍然一如既往，大部分关键的重构手法也大体不变。我希望这次修订能帮助更多的读者学会如何有效地进行重构。

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，可以最大限度地减小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。在软件开发的大部分历史时期，大部分人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。哪怕手上有一个糟糕的设计，甚至是一堆混乱的代码，我们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：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。

有了重构以后，工作的平衡点开始发生变化。我发现设计不是在一开始完成的，而是在整个开发过程中逐渐浮现出来。在系统构筑过程中，我学会了如何不断改进设计。这个“构筑-设计”的反复互动，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编写的重构指南。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，图书应该以概念介绍开头。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。因此，我决定用一个实例作为开路先锋。第 1 章展示了一个小程序，其中有些常见的设计缺陷，我把它重构得更容易理解和修改。其间你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，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面临的一些挑战。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 章介绍如何在代码中构筑测试。

从第 5 章往后的篇幅就是本书的核心部分——重构名录。尽管不能说是一份巨细靡遗的列表，却足以覆盖大多数开发者可能用到的关键重构手法。这份重构名录的源头是 20 世纪 90 年代后期我开始学习重构时的笔记，直到今天我仍然不时查阅这些笔记，作为对我不甚可靠的记忆力的补充。每当我想做点什么——例如拆分阶段（154）——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。

一本 Web 优先的书

万维网对我们的社会影响深远，尤其是改变了我们获取信息的方式。在撰写本书第 1 版时，关于软件开发的知识大多通过出版物传播。而时至今日，我的大部分信息都来自网上。这个趋势给像我这样的写作者带来了一个挑战：今日世界还有图书的一席之地吗？今天的图书应该是什么形态？

我相信像这样一本书仍然有其价值，但也需要作出改变。图书的价值在于把大量信息以内聚的方式整合起来。在撰写本书的过程中，我尝试用连贯一致的方式来组织和涵盖大量各有特色的重构手法。

但这个聚合的整体是一个抽象的文学作品，尽管传统上只能以纸质图书的形式呈现，未来却未必非得如此。出版行业仍然将纸质图书视为首要的呈现形式，虽然我们已经满怀热情地接纳了电子书，但是电子图书毕竟也只是在原来纸质图书结构的基础

上做了电子化的呈现。

我想通过这本书探索一条不同的路径。本书的权威版本是它的网站（或者叫“Web版”）。如果你购买了纸质版或者电子版，就会同时获得访问 Web 版的权限。（关于如何在 InformIT 网站上注册你的商品，请留意下文的提示。）纸质版图书是网站内容的精选，并整理成适合印刷的形式。纸质版并不尝试包含网站上的所有重构手法，尤其是考虑到未来我很有可能在 Web 版中增加更多重构手法。与此相似，电子书又是 Web 版的另一个呈现，其中包含的重构手法列表可能与纸质版不同，毕竟电子书在售出之后也可以相对容易地更新和添加内容。

在写下这些文字时，我无从知晓你正在阅读的是在线 Web 版、手机上的电子书、纸质版图书还是别的什么超乎我想象的形式。我尽力写一本有用的书，不论你用什么形式来汲取其中的知识。

如果你想查看本书 Web 版（只有英文版），并及时获得内容更新和勘误，请到 InformIT 网站注册这本书。你需要首先打开 informit.com/register 页面，登录你的 InformIT 账户（如果没有 InformIT 账户的话，需要先注册一个），然后（进入“Registered Products”标签）输入本书英文原版的 ISBN“9780134757599”，单击“Submit”按钮。然后网站会向你提出一个与本书内容有关的问题，所以请确保纸质书或电子书就放在手边。成功注册以后，进入“Account”页面，打开“Digital Purchases”标签，单击本书标题下面的“Launch”按钮，就能看到本书的 Web 版。

For access to the web edition (English only) and updates or corrections as they become available, register your copy on the InformIT web site. To start the registration process, go to informit.com/register and log in (or create an account if you don’t have one). Enter 9780134757599 in the box labeled ISBN and click Submit. You will be asked a challenge question, so be sure to have your copy of the book available. After you’ve successfully registered your copy, open the “Digital Purchases” tab on your Account page and click on the link under this title to “Launch” the web edition.

JavaScript 代码范例

与软件开发中的大多数技术性领域一样，代码范例对于概念的阐释至关重要。不过，即使在不同的编程语言中，重构手法看上去也是大同小异的。虽然会有一些值得留心的语言特性，但重构手法的核心要素都是一样的。

我选择了用 JavaScript 来展现本书中的重构手法，因为我感到大多数读者都能看

懂这种语言。不过，即便你眼下正在使用的是别的编程语言，采用这些重构手法也应该不困难。我尽量不使用 JavaScript 任何复杂的特性，这样即便你对这门编程语言只有粗浅的了解，应该也能跟上重构的过程。另外，使用 JavaScript 展示重构手法，并不代表我推荐这门编程语言。

使用 JavaScript 展示代码范例，也不意味着本书介绍的技巧只适用于 JavaScript。本书的第 1 版采用了 Java，但很多从未写过任何 Java 代码的程序员也同样认为这些技巧很有用。我曾经尝试过用十多种不同的编程语言来呈现这些范例，以此展示重构手法的通用性，不过这对普通读者而言只会带来困惑。本书是为所有编程语言背景的程序员所作，除了阅读“范例”小节时需要一些基本的 JavaScript 知识，本书的其余部分都不特定于任何具体的编程语言。我希望读者能汲取本书的内容，并将其应用于自己日常使用的编程语言。具体而言，我希望读者能先理解本书中的 JavaScript 范例代码，然后再将其适配到自己习惯的编程语言。

因此，除了在特殊情况下，当我谈到“类”“模块”“函数”等词汇时，我都按照它们在程序设计领域的一般含义来使用这些词，而不是以其在 JavaScript 语言模型中的特殊含义来使用。

我只把 JavaScript 用作一种示例语言，因此我会尽量避免使用其他程序员可能不太熟悉的编程风格。这不是一本“用 JavaScript 进行重构”的书，而是一本关于重构的通用书籍，只是采用了 JavaScript 作为示例。有很多 JavaScript 特有的重构手法很有意思（如将回调重构成 promise 或 async/await），但这些不是本书要讨论的内容。

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。书中的范例和讨论，涉及大量需要详细阅读和理解的代码。这些例子都用 JavaScript 写成，不过这些重构手法应该适用于大部分编程语言。为了理解书中的内容，读者需要有一定的编程经验，但需要的知识并不多。

本书的首要目标读者群是想要学习重构的软件开发者，同时对于已经理解重构的人也有价值——本书可以作为一本教学辅助书。在本书中，我用了大量篇幅详细解释各个重构手法的过程和原理，因此有经验的开发人员可以用本书来指导同事。

尽管本书的关注对象是代码，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。最好是由有威望的、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 JavaScript 而是其

他编程语言，这一点尤其重要，因为你必须把我给出的范例用其他编程语言改写。

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，其中的示例会让你弄清楚重构的过程。
- **如果你想知道为什么应该重构**，请阅读前两章，它们会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**，请完整阅读前四章，然后选择性地阅读重构名录。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重构手法，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。

给形形色色的重构手法命名是编写本书的重要部分。合适的词汇能帮助我们彼此沟通。当一名开发者向另一名开发者提出建议，将一段代码提取成为一个函数，或者将计算逻辑拆分成几个阶段，双方都能理解*提炼函数*（106）和*拆分阶段*（154）是什么意思。这份词汇表也能帮助开发者选择自动化的重构手法。

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 20 世纪 90 年代做了大量研究工作并开创重构领域的人一大笔债。学习他们的经验启发了我撰写本书第 1 版，尽管已经过去了很多年，我仍然必须感谢他们打下的基础。这本书原本应该由他们之中的某个人来写，但最后却让我这个有时间、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软件开发过程的一块基石，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 合作，我才真正看到了重构的重要性，并直接受到激励写了这本书。

Ralph Johnson 在 UIUC（伊利诺伊大学厄巴纳-香槟分校）领导了一个小组，这个小组因其在对象技术方面的实用贡献而声名远扬。Ralph 很早就是重构的拥护者，他的一些学生也在重构领域的发展前期做出重要研究。**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，他们创造了第一个自动化的重构工具，这个叫作 Refactoring Browser（重构浏览器）的工具可以用于重构 Smalltalk 程序。

自本书第 1 版问世以来，很多人推动了重构领域的发展。尤其是，开发工具中的自动化重构功能，让程序员的生活轻松了许多。如今我只要简单地敲几下键盘就可以给一个被大量使用的函数改名，对此我已经习以为常，但在这快捷的操作背后，离不开 IDE 开发团队的辛勤劳动。

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成本书。本书的第 1 版极大地得益于 Kent Beck 的经验与鼓励。起初向我介绍重构的是他，鼓励我开始书面记录重构手法的是他，帮助我把重构手法组织成型的也是他，提出“代码味道”这个概念的还是他。我常常感觉，他本可以把本书的第 1 版写得更好——如果当时他不是在忙着撰写极限编程的奠基之作《解析极限编程》的话。

我认识的所有技术图书作者都会提到，技术审稿人提供了巨大的帮助。我们的作品都会有巨大的缺陷，只有同行审稿人能发现这些缺陷。我自己并不常做技术审稿，部分原因是我认为自己并不擅长，所以我对优秀的技术审稿人总是满怀敬意。帮别人审稿所得的报酬微不足道，所以这完全是一项慷慨之举。

正式开始写这本书时，我建了一个邮件列表，其中都是能给我提供反馈的建议者。随着写作的进展，我不断把新的草稿发到这个小组里，请他们给我反馈。我要感谢这些人在邮件列表中提供的反馈：**Arlo Belshee**、**Avdi Grimm**、**Beth Anders-Beck**、**Bill Wake**、**Brian Guthrie**、**Brian Marick**、**Chad Wathington**、**Dave Farley**、**David Rice**、**Don Roberts**、**Fred George**、**Giles Alexander**、**Greg Doench**、**Hugo Corbucci**、**Ivan Moore**、**James Shore**、**Jay Fields**、**Jessica Kerr**、**Joshua Kerievsky**、**Kevlin Henney**、**Luciano Ramalho**、**Marcos Brizeno**、**Michael Feathers**、**Patrick Kua**、**Pete Hodgson**、**Rebecca Parsons** 和 **Trisha Gee**。

在这群人中，我要特别感谢 **Beth Anders-Beck**、**James Shore** 和 **Pete Hodgson** 在 JavaScript 方面给我的帮助。

有了一个比较完整的初稿之后，我将它发送出去，寻求更多的审阅意见，因为我希望有一些全新的眼光来纵览全书。**William Chargin** 和 **Michael Hunger** 提供了极其详尽的审阅意见。我还从 **Bob Martin** 和 **Scott Davis** 那里得到了很多有用的意见。**Bill Wake** 也对本书初稿做了完整的审阅，并在邮件列表中给出了他的意见。

我在 ThoughtWorks 的同事一直给我的写作提供想法和反馈。数不胜数的问题、评论和观点推动了本书的思考与写作。作为 ThoughtWorks 员工最好的一件事，就是这家公司允许我花大量时间来写作。我尤其要感谢 **Rebecca Parsons**（我们的 CTO）经常

与我交流，给了我很多想法。

在培生出版集团，**Greg Doench** 是负责本书的策划编辑，他解决了无数的问题，最终使本书得以出版；**Julie Nahil** 是责任编辑；**Dmitry Kirsanov** 负责文字编辑工作；**Alina Kirsanova** 负责排版和制作索引。我也很高兴与他们合作。

目录