TURING 图灵程序设计丛书

Addison Wesley

Refactoring Improving the Design of Existing Code

重构 改善既有代码的设计

（英文版）

Erich Gamma 作序推荐

[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
- 普通程序员进阶到编程高手必须修炼的秘笈

人民邮电出版社
POSTS & TELECOM PRESS

TURING 图灵程序设计丛书

Refactoring Improving the Design of Existing Code

重构 改善既有代码的设计

（英文版）

[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 / (美) 福勒(Fowler,M.) 著. -- 北京 : 人民邮电出版社, 2010.11
(图灵程序设计丛书)
书名原文: Refactoring: Improving the Design of Existing Code
ISBN 978-7-115-23914-3

Ⅰ. ①重… Ⅱ. ①福… 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(2010)第189045号

内 容 提 要

本书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
北京鑫正大印刷有限公司印刷

◆ 开本：800×1000 1/16
印张：28.25
字数：524千字 2010年11月第1版
印数：1－2 500册 2010年11月北京第1次印刷

著作权合同登记号 图字：01-2006-3662号

ISBN 978-7-115-23914-3

定价：69.00元

读者服务热线：(010)51095186 印装质量热线：(010)67129223
反盗版热线：(010)67171154

版 权 声 明

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。挖得愈深，找到的重构机会就越多，于是你的修改也愈多……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

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二个星期。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几率。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，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* (255)——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所客串写就的。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愈来愈广的语言，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

Contents