PEARSON
HZ BOOKS
华章科技

世界级计算机专家Michael C. Feathers的经典之作，软件开发大师Robert C. Martin作序倾情推荐，修改遗留代码的权威指南

深入剖析修改遗留代码的各种方法和策略，从理解遗留代码、为其编码测试、重构及增加特性等方面给出大量实用建议，是所有程序开发人员必读之作

名家经典系列

Working Effectively with Legacy Code

[美] Michael C. Feathers 著
侯伯薇 译

修改代码的艺术

机械工业出版社
China Machine Press

修改代码的艺术

（美） Michael C. Feathers 著
侯伯薇 译

Working Effectively
with
Legacy Code

机械工业出版社
China Machine Press

图书在版编目（CIP）数据

修改代码的艺术 /（美）费瑟（Feathers，M. C.）著；侯伯薇译 . —北京：机械工业出版社，2014.6
（名家经典系列）
书名原文：Working Effectively with Legacy Code

ISBN 978-7-111-46625-3

I. 修… II. ① 费… ② 侯… III. 软件开发 IV. TP311. 52

中国版本图书馆 CIP 数据核字（2014）第 092254 号

本书版权登记号：图字：01-2012-4657

修改代码的艺术

[美] Michael C. Feathers 著

出版发行：机械工业出版社（北京市西城区百万庄大街 22 号 邮政编码：100037）
责任编辑：关 敏 责任校对：董纪丽
印 刷：北京市荣盛彩色印刷有限公司 版 次：2014 年 6 月第 1 版第 1 次印刷
开 本：186mm × 240mm 1/16 印 张：20.5
书 号：ISBN 978-7-111-46625-3 定 价：79.00 元

凡购本书，如有缺页、倒页、脱页，由本社发行部调换
客服热线：（010）88378991 88361066 投稿热线：（010）88379604
购书热线：（010）68326294 88379649 68995259 读者信箱：hzjsj@hzbook.com

译 者 序

翻译完这一本经典的计算机软件著作，不由得想起了自己程序员生涯中多个重要的时刻。

第一次见到传说中的计算机还是在高中时期，听说学校组建了计算机房，就跑到门口去膜拜了一下，当时最好的一台机器是386，令我惊叹不已。可惜的是，当时自己已经高三，而计算机的课程是为高一、高二的同学们准备的，结果就这样和计算机失之交臂。

上大学之后，我终于接触到了计算机，并且开始学习DOS系统，还学会了中英文打字。最神奇的就是，我把从《电脑报》上抄下来的程序键入计算机，用Basic编译执行之后，竟然成功地让计算机上的喇叭发出了音乐声。

大学期间我不仅学习了各种编程技术，还自学了不少其他领域的应用技术，如图像处理、视频编辑、三维动画制作、网页制作等。结果在毕业的时候，阴差阳错，我并没有直接加入程序员的队伍中。然而，似乎是冥冥中有个声音在督促我，所以我一直没有放弃编程技术的学习。

命中注定我会成为一名程序员。两年之后，我加入了一家软件公司，终于继续自己喜欢的程序员生涯，开始用代码来实现自己的价值，编写出各种各样的管理系统。之后多年，直到现在，我很庆幸自己仍然可以每天用编写代码的方式来帮助他人。

我的目标就是成为一名快乐的程序员，所以，在生涯之中会有一些独特的时刻，比方说：

2007年，我第一次和朋友合作，将一本计算机图书从英文翻译成中文，并从此一发而不可收拾。

2008年，我加入了现在这家公司，和业务人员的合作更加紧密，开始不断思考自己编写程序的目的和价值。

2009年，我加入了InfoQ中文站，不仅翻译新闻和文章，还写了不少原创的内容，直到现在还乐此不疲。

2010年，我开始在大连组织程序员社区活动，希望通过自己的努力，建立起一个程序员交流的平台，坚持到现在，已经组织了20多场各种主题的聚会。

2012年，我和CSDN一起做高校巡讲，把自己的一些感悟分享给同学们，虽然各地奔波，但乐在其中。

……

无数个时刻，就像是链条的一个个环节，串在一起，织就了我自己的程序员生涯。

能够翻译这样优秀的一本图书，也是非常重要的一个时刻。当我敲入第一个字母，就和这本书联系在了一起；当完成最后一段的翻译，也并不是和这本书的关系完结，而是联系得更加紧密。

在翻译的过程中，有苦有乐，多亏了家人和朋友的支持，才能够最终完成这项任务，能够让自己几个月的辛苦画上一个还算圆满的句号。千言万语化成一句，感谢！

希望读者朋友在阅读这本书的时候，也能够发现属于自己的那个时刻。阅读完之后，对软件开发会有新的认识，能够在面对大量遗留代码的时候不再手足无措，而是可以耐心地采用各种各样的手段，不断改进，驯服那些原本看起来像怪兽一样的代码，让它们可以更好地发挥作用。

当我们在多年后的某个时刻回顾过往的各个时刻时，相信一定会发现，其实每个时刻都是那么重要。

所以，请大家开始享受阅读这本书的这个时刻吧。

序

“……就从那一刻开始……”

Michael Feathers 在介绍这本书的时候，使用了上面这句话来描述他对软件的热情是怎样产生的。

“……就从那一刻开始……”

你是否有过那种感觉？你是否发现生命中有那么一个时刻，然后说：“……就从那一刻开始……？”是否有一件事情改变了你的生命进程，最终让你拿起这本书并开始阅读这篇序？

当在我身上发生这样的事情时，我才上六年级。当时我对科学、宇宙以及所有技术的事物都感兴趣。妈妈在商店找到了一台塑料计算机，并给我买了回来。它叫做 Digi-Comp I。40 年后，那台小小的塑料计算机还放在我的书架上，因为那是一种纪念。就是它点燃了我对软件无法抑制的热情的火花。它让我第一次了解到，编写程序来为人们解决问题是多么有趣的一件事。它只由 3 个塑料的 S-R 触发器以及 6 个塑料与门构成，但那已经足够了——已经可以够我使用了。对于我来说……就从那一刻开始……

但不久之后，我就意识到软件系统总是会变成一团糟，这让我的热情冷却了下来。随着时间推移，原本在程序员头脑中水晶般透彻的设计会开始腐坏，就像一块臭肉。我们去年构建的非常好的小系统，在下一年中就会成为由复杂的函数和变量组成的可怕的、乱七八糟的代码。

为什么会发生这样的事情呢？系统为什么会变坏呢？为什么无法保持整洁呢？

有时我们把这归咎于我们的客户。有时我们会抱怨他们总是变更需求。我们安慰自己说，如果客户能够坚持他们当初描述的所需功能，那么设计就会很好。修改需求完全是客户的错误。

这就是问题所在：需求变更。无法承受需求变更的设计不是好设计。每个有竞争力的软件开发者的目标就是，创建能够承受变更的设计。

这个问题看起来非常难以解决。事实上，它是如此难以解决，以至于所有曾经创建出来的系统都不得不慢慢、逐渐地腐坏。这种腐坏的情况如此普遍，所以我们为这些腐坏的代码起了个专门的名字，叫做：遗留代码（legacy code）。

遗留代码，这个词直击程序员的内心。它会让人联想到“在黑暗、乱糟糟的灌木丛中艰难跋涉，脚下是吸血的蚂蟥，身旁飞的是蜇人的昆虫，空气中满是黑暗、黏糊糊、沉重、腐烂的垃圾一样的气

味”。尽管我们首次编程时尝到的滋味非常美妙，但修改遗留代码的痛苦会浇灭你的热情之火。

我们很多人都尝试过寻找方法，避免让代码成为遗留代码。我们已经写过关于能够帮助程序员保持系统整洁的原则、模式和实践的书籍。但 Michael Feathers 对于我们没有注意到的方面拥有非常深刻的见解。单纯的防止并不完美。即便是在最严格的开发团队中，他们知道最好的原则，使用最佳的模式，并遵循最佳实践，但还是会一次次创造出垃圾代码。代码的腐坏仍然会发生。只是试图防止腐坏并不够，你需要能够让这个过程反转。

这正是本书的重点所在。它是关于反转腐坏过程的一本书。它会告诉我们如何面对复杂、一无所知、费解的系统，然后慢慢地、逐渐地、一段一段、一步一步，把它变成简单、结构良好、设计良好的系统，就像是扭转热力学系统中熵增的趋势一样。

我需要警告你的是，先不要太激动；反转腐坏的过程并不是那么容易，也不会很快。Michael 在本书中提出的技术、模式和工具都很有效，但那需要你付出努力、时间、耐心和细心。这本书并不是魔弹。它不会告诉你如何在一夜之间就消除所有系统中的腐坏代码，它只描述你在职业生涯剩下的时间都需要采取的一系列规则、概念和态度，那会帮助你把逐渐腐坏的系统变成逐渐改善的系统。

Robert C. Martin

前　言

你还记得自己写过的第一个程序吗？我还记得。那是我在早期的 PC 上编写的一个很小的图形程序。我开始编程的时间比大多数朋友都要晚。当然，当我还是个孩子的时候就见过计算机。我还记得，当第一次在一个办公室里看到微型计算机的时候，我就被深深触动了，但很多年过去了，我甚至都没有机会坐在一台计算机前面。稍后，当我十几岁的时候，我的一些朋友买了一些最早期的 TRS-80 计算机。我非常感兴趣，但也非常不安。我知道一旦开始玩计算机，就会被它深深吸引。它看起来太酷了。我不知道为什么我对自己如此了解，但我还是把它延后了。之后，在大学中，我的一位室友有一台计算机，我买了一套 C 语言编译器，从而可以自学编程。就是从那一刻开始。我一夜又一夜地试着编写程序，使用编译器自带的 emacs 编辑器写出大量源代码。那让我非常上瘾，而且很有挑战性，我爱死它了！

我希望你也有过类似的经历——让代码能够在计算机上工作的那种最原始的快乐。几乎所有我问过的程序员都有。那种快乐是让我们从事这份工作的原因之一，但一天又一天，快乐在哪里呢？

多年前的一天晚上，在完成工作之后，我给朋友 Erik Meade 打了一个电话。我知道 Erik 刚刚开始为一个新团队提供咨询服务，于是我问他：“他们做得如何？”他说：“伙计，他们在编写遗留代码。”这让我心头一震，在我的生命中有过几次那样的情况，这就是一次。我从内心理解他的说法。Erik 说的话非常准确地描述了我访问很多团队时所获得的第一感觉。他们非常努力，但在一天结束的时候，迫于时间的压力，繁重的历史包袱，或者缺少能够比较的更好代码，很多人都在编写遗留代码。

什么是遗留代码呢？这个词我使用了很长时间，但并没有准确的定义。让我们看一下严格的定义：遗留代码是我们从别人那里获得的代码，可能是我们公司从另一家公司获得的代码；也可能是最初某个团队迁移到另一个项目中的代码。遗留代码是别人的代码。但在程序员中，这个词有更丰富的含义。“遗留代码”随着时间推移拥有越来越多的含义。

当听到遗留代码这个词时，你会想到什么呢？如果你的情况和我类似，那么就会想到复杂、莫名其妙的结构，你需要修改但是无法真正理解的代码。你在思考中度过一个个不眠之夜，只为了增加特性，而那些特性本来应该很容易添加，然后你要考虑如何演示，团队中的每个人都非常讨厌那

些代码，它们看起来根本没人管，你死都不愿意去碰那些代码。你们其中有些人只要想想要让它变好就难受。看起来你们的努力没有任何价值。代码会以多种方式变差，很多方式都和代码是否来自另一个团队没有关系。

在业界，遗留代码这个词通常代表的是我们根本就不理解的难以修改的代码。但是经过和团队一起工作多年，帮助他们解决严重的代码问题，我得到了不同的定义。

对我来说，遗留代码是缺少测试的简单代码。我为这个定义而感到有些悲哀。测试和代码的好坏有什么关系？对我来说，答案很简单，那也是我在本书始终详细阐述的一个观点。

缺少测试的代码就是很差的代码。不管它编写得有多好；也不管它的面向对象以及封装做得有多好。有了测试，我们可以快速且有保证地修改代码的行为。相反，没有测试，我们就不知道代码是变得更好还是更糟糕。

你可能认为这有些过分。那么整洁的代码呢？如果代码非常整洁、结构良好，那就足够了吗？好吧，不要错怪我。我喜欢整洁的代码，甚至比大多数我认识的人还要喜欢，但是尽管整洁的代码非常棒，那还不够。当团队试图做重大的修改，而缺少测试的时候，就会冒很大的风险。就像是在做空中体操而没有保护网一样。那需要非常优秀的技能，以及对每步会发生什么都有非常清晰的了解。精确地知道如果你修改一系列变量会发生什么，就像是你翻跟斗翻出去的时候，有另一个人会抓住你的胳膊。如果你所在的团队代码非常清晰，你的形势要比其他程序员更好一些。在我的工作中，我发现很少有团队的代码能够达到那种整洁程度。它们看起来是统计上的不正常现象。并且，你知道吗？如果你没有起支持作用的测试，那么修改那样的代码会比拥有测试的团队慢。

是的，团队会变得更好，并开始编写更加清晰的代码，但是想要让旧代码变得清晰，需要花费很长时间。在很多情况下，那永远都不会有根本性的改变。因此，我把遗留代码定义为缺少测试的代码，这是没有问题的。它是很有效的定义，并且指出了解决方案。

到现在为止，我说了很多关于测试的事情，但本书并不是关于测试的一本书。本书是帮助大家有信心修改任何代码的书。在接下来的章节中，我会描述可以用来理解代码、为其编写测试、重构以及增加特性的技术。

当阅读这本书的时候，你会注意到的一点是，这并不是一本关于完美代码的书。我在书中使用的代码都是虚构的，因为我的工作需要和客户签订保密协议。但在很多例子中，我试图保持在那个领域所见到的代码的内在品质。我不想说例子总是很有代表性。肯定会有更好的代码，但坦白说，也总是会有一些代码，比我在本书使用的例子要差得多。除了需要为客户保密之外，我之所以不能把那样的代码放到本书中，也是为了不让你厌烦，而忽略隐藏在繁冗的细节中的重要问题。因此，很多例子都相对比较简洁。如果你看到一段代码，想：“不，他根本不知道，我的方法要比那大得

多，也差得多”请看一下我给出的建议，看它是否适用，即便例子看起来更简单一些。

本书中所讲述的技术已经在大型代码中经过检验，只是由于篇幅所限，例子才都比较小。特别是当你在代码片段中看到这样的省略号（…）的时候，你可以把它读作“在此插入 500 行丑陋的代码”：

```
m_pDispatcher->register(listener);
...
m_nMargins++;
```

本书不是关于完美代码的，甚至也不是关于完美设计的。好的设计应该是我们所有人的目标，但在遗留代码中，那是我们无法完全达到的。在某些章节中，我描述了向现有代码中增加新代码的方式，并展示了如何在添加的时候在头脑中想着好的设计原则。你可以先在遗留代码库中开辟出优秀代码区域，但是，如果你想要采取的变更步骤让代码更加丑陋，也不要感到惊奇。这项工作就像是外科手术。我们需要切开皮肤，穿过内脏，这时可能暂时不要考虑美观的问题。这位病人的主要器官和内脏能更好一些吗？是的。那么我们能不管最直接的问题，把他的伤口缝合，然后告诉他正确饮食，然后就训练他开始跑马拉松吗？我们可能可以，但我们真正需要做的就是正视病人的情况，治好有问题的部分，让他更加健康。他可能永远都无法成为奥运会运动员，但是我们不能让“最好”成为“更好”的敌人。代码库可以变得更健康，更容易在其中工作。当病人感觉好一些之后，就到了你可以帮助他进入更健康的生命循环中的时候了。那正是我们对遗留代码所应该做的事情。我们试图回到让我们感觉舒服的那个点；我们期望那样，并试图让修改代码更容易。当我们可以在团队中保持那种感觉的时候，设计就会变得更好。

我在本书中所描述的技术，都是多年以来和客户一起协作试图对无规则的代码加以控制的过程中发现和学到的。我偶尔还会强调遗留代码。当我最初开始在 Object Mentor 公司工作的时候，我的工作包括帮助团队解决严重的问题，改善他们的技能和交互，帮助他们达到能够持续交付高质量代码的程度。我通常会使用极限编程实践来帮助团队掌控他们的工作，紧密协作，然后交付。我常常觉得极限编程不仅仅是开发软件的一种方式，更应该是创造出好的团队形式的方式，只是恰好以每两周一次的频率交付了优秀的软件。

但是，从最开始就有一个问题。很多最初的 XP 项目都是从头开始的项目。我见到的客户都拥有非常庞大的代码库，正处于麻烦之中。他们需要某种方式来掌控他们的工作并开始交付。随着时间推移，我发现自己总是和客户一次又一次做着同样的事情。这种感觉在我和一个开发金融项目的团队协作时达到了顶峰。在我参与之前，他们就已经意识到，单元测试是非常好的东西，但是他们所执行的测试是完整场景的测试，会与数据库做多次交互，并执行大量代码。因此非常难以编写测试，而且团队不会经常运行它，因为运行一次要花费太长时间。当我和他们坐在一起打破依赖关系，并为小段代码编写测试的时候，我有一种似曾相识的可怕感觉。看起来我和所有见过的团队都

是在做这种工作，而那是一种没有人想要考虑的东西。那是一种乏味的工作，当你想要以可控的方式来处理代码而且知道如何做的时候，才会去做。然后，我决定总结一下我们会如何解决这些问题，并把它们写下来，从而帮助团队让他们的代码更容易处理，那会很有价值。

关于例子还要说明一下：我使用了多种不同编程语言编写的示例。那些例子是由 Java、C++ 和 C 语言编写的。我选择 Java 是因为它是一种非常常见的语言，包含了 C++ 的例子是因为它在遗留环境中代表了某些特殊的挑战，而选择 C 语言是因为它凸显了很多在过程化遗留代码中所存在的问题。通过这三种语言，我覆盖了在遗留代码中出现的大多数问题。然而，如果你所使用的语言并没有出现在示例代码中，那么也要阅读那些代码。我讲述的很多技术都可以用在其他语言中，如 Delphi、Visual Basic、COBOL 以及 FORTRAN。

我希望你会发现本书中的技术很有帮助，让你可以找回编程的乐趣。编程可以是一种回报丰厚并且让人享受的工作。如果你在日常的工作中没有感觉到那一点，我希望我在本书中提供给你的技术能够让你找到那种感觉，并在团队中传播。

如何使用本书

我试过用多种不同的形式来写这本书，直到最后确定为现在的这个样子。当处理遗留代码的时候，很多不同的技术和实践都非常有用，很难独立开来说明。如果你能够找到接缝、创建伪对象、使用一些技术打破依赖关系，那么最简单的修改会变得更加容易。为了让本书更易于理解，随时可用，我决定把很大一部分内容（第二部分）以 FAQ（经常问到的问题）的形式来组织。因为特定的技术通常需要使用其他技术，FAQ 章节相互之间的关联性很强。在几乎每章中，你都会看到带有章节号的引用，你可以查看其他章节和部分，其中会描述特定的技术和重构。如果这让你在试图寻找问题答案的时候，不得不大范围翻阅本书，那么我非常抱歉，但我认为你最好那么做，而不是一页接一页地阅读，试图理解所有技术是怎么做的。

在第二部分，我试图回答在处理遗留代码的过程中所遇到的常见问题。每章的标题都是一个特定的问题。这让章节的名称很长，但希望那可以让你快速找到帮助你解决特定问题的部分。

在修改软件部分的前面，是一系列介绍性章节（第一部分），后面是一系列重构列表（第三部分），那在处理遗留代码的工作中非常有用。请阅读介绍性章节，特别是第 4 章。这些章为之后的技术提供了具体情境和命名法。此外，如果你发现没有在上下文中描述的术语，那么就可以在术语表中查找。

第三部分中的重构比较特殊，因为它们是故意在没有测试的情况下完成的，目的是为了编写测试。我建议你阅读所有内容，从而可以在开始驯服遗留代码的时候看到更多的可能性。

致谢

首先，非常感谢我的妻子 Ann 以及孩子 Deborah 和 Ryan。他们的爱和支持让这本书以及所有的学习成为现实。我还要感谢“Bob 大叔”Martin，他是 Object Mentor 公司的主席和创始人。他严密而有效的开发和设计方法能够把重要内容从无关紧要的内容中分离出来。大概十年前，我快要被大量不切实际的建议所淹没，正是他带领我走出了那段困境。还要感谢 Bob 给了我机会看到更多代码，并在过去五年间和更多人一起工作，我从来没有想象过能够那样。

我还要感谢 Kent Beck、Martin Fowler、Ron Jeffries 和 Ward Cunningham，他们经常会给我提出建议，并且教给我很多关于团队工作、设计和编程方面的知识。特别要感谢所有审阅了初稿的人。正式审稿人有 Sven Gorts、Robert C. Martin、Erik Meade 和 Bill Wake；非正式审稿人有 Robert Koss 博士、James Grenning、Lowell Lindstrom、Micah Martin、Russ Rufer 和硅谷伙伴小组以及 James Newkirk。

还要感谢审阅了我早期放在网上的书稿的人们，他们的反馈在我重新组织了形式之后，对书的方向产生了很大影响。我先要说声抱歉，因为名单可能会有所遗漏，他们是：Darren Hobbs、Martin Lippert、Keith Nicholas、PhlipPlumlee、C. Keith Ray、Rober Blum、Bill Burris、William Caputo、Brian Marick、Steve Freeman、David Putman、Emily Bache、Dave Astels、Russel Hill、Christian Sepulveda 和 Brian Christopher Robinson。

还要感谢 Joshua Kerievsky，他做了非常关键的早期审阅工作；感谢 Jeff Langr，他在我写这本书的整个过程中给出了很多建议，并做了大量审阅工作。

审稿人让我的草稿变得更加合理，但如果还有错误的话，那都是我自己的问题。

感谢 Martin Fowler、Ralph Johnson、Bill Opdyke、Don Roberts 和 John Brant 在重构领域做出的卓越工作。那给了我很大启发。

特别要感谢 Jay Packlick、Jacques Morel、Sabre Holdings 的 Kelly Mower、Workshare Technology 的 Graham Wright，他们都提供了很多支持和反馈。

还要特别感谢 Paul Petralia、Michelle Vincenti、Lori Lyons、Krista Hansing，以及 Prentice-Hall 团队中的所有人。谢谢 Paul，你给了我这个第一次写书的人太多帮助和鼓励。

特别感谢 Gary 和 Joan Feathers、April Roberts、RaimundEge 博士、David Lopez de Quintana、Carlos Perez、Carlos M. Rodriguez、John C. Comfort 博士，多年来你们给了我大量帮助和鼓励。我还要感谢 Brian Button 帮忙提供了第 21 章中的示例，当我们一起开发重构课程的时候，他花费了一个小时来编写那段代码，那是我最喜欢的一段课程用代码。

另外，还要特别感谢 Janik Top，他的乐曲 De Futura 在写这本书的最后几个星期中一直陪伴着我。

最后，我还要感谢过去多年间一起工作的所有同事，他们的深刻见解和挑战让书中的内容更加充实。

Michael Feathers

mfeathers@objectmentor.com

www.objectmentor.com

www.michaelfeathers.com

目　录

第二部分 修改软件