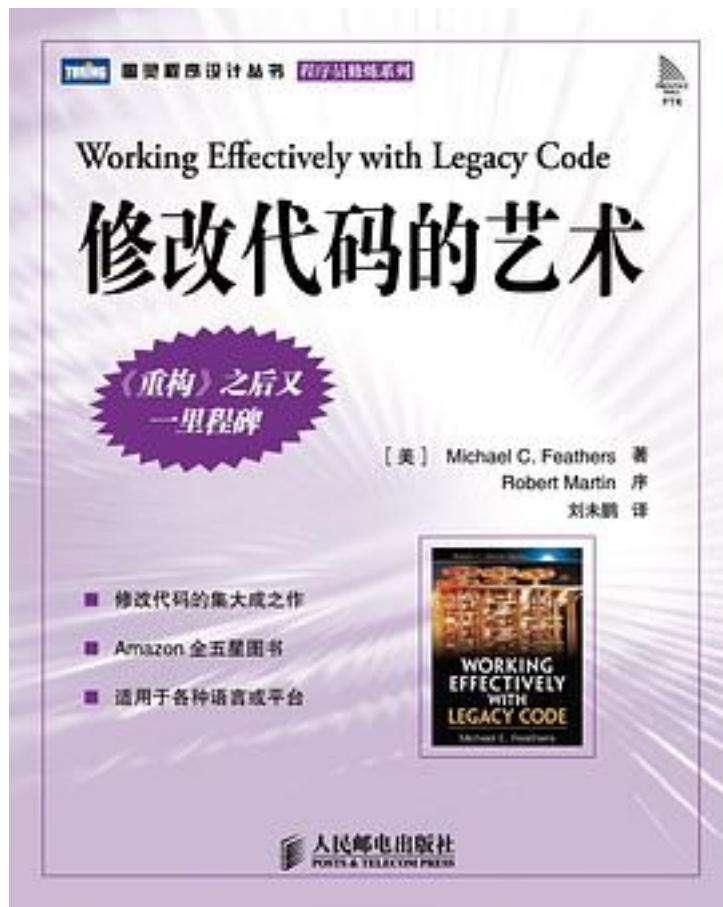


修改代码的艺术



[修改代码的艺术 下载链接1](#)

著者: (美) Michael C. Feathers

出版者:机械工业出版社

出版时间:2014-6-15

装帧:平装

isbn:9787111466253

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。

修改代码时，你觉得容易吗？当你修改代码时，能否几乎即时地获得反馈？你理解那些代码吗？如果对于这些问题的答案是否定的，那么你面对的就是遗留代码，它们正在浪费你开发工作的时间和金钱。

在本书中，作者为更有效地处理大规模、缺少测试的遗留代码提供了自始至终的策略。本书内容来自Michael创建的非常知名的Object Mentor公司的研习会，Michael使用那些技术来指导并帮助了成千上万位开发者、技术经理和测试人员，让他们的遗留系统处于可控状态。

本书主要内容：

理解修改软件的机制：添加特性、修正缺陷、改进设计、优化性能

把遗留代码放到测试用具之中

编写测试，防止引入新的问题

包含Java、C++、C和C#的示例，其中介绍的大多数技术适用于其他任何语言或平台

精确地确定要在哪些地方修改代码

处理非面向对象的遗留代码

处理看起来没有任何结构的应用程序

作者介绍：

Michael C. Feathers 世界级软件开发大师，就职于Object

Mentor公司（这是一家世界领先的提供软件领域的指导、技能开发、知识传播和领导力服务的公司）。他是ACM和IEEE成员，也是CppUnit（从JUnit移植到C++上的单元测试框架）和FitCpp（FIT集成测试框架在C++上的实现）的最初作者，曾3次担任OOPSLA会议的CodeFest主席。目前他在世界范围内提供测试驱动开发、重构、面向对象设计、Java、C#、C++以及极限编程方面的培训和指导。

译者简介

侯伯薇

中荷人寿保险有限公司高级系统分析师，InfoQ中文站翻译团队主编，拥有十多年开发经验，目前致力于技术与业务的融合，让开发出来的程序能够真正提高业务人员的工作效率。热衷于通过翻译和演讲的方式与广大程序员分享和交流，曾翻译过多本技术书籍和几百篇技术短文，并在Scrumgathering、QClub、敏捷之旅等活动上做过技术演讲

目录: 目录

译者序

序

前言

第一部分 修改机制

第1章 修改软件 2

1.1 修改软件的四大原因 2

1.1.1 增加特性和修正缺陷 2

1.1.2 改善设计 4

1.1.3 优化 4

1.2 组合在一起	4
第2章 利用反馈	7
2.1 什么是单元测试	9
2.2 高层次测试	11
2.3 测试覆盖	11
2.4 遗留代码修改方法	14
2.4.1 确定变更点	14
2.4.2 找到测试点	14
2.4.3 打破依赖关系	14
2.4.4 编写测试	15
2.4.5 做出修改并重构	15
2.5 本书其他部分	15
第3章 感知和分离	16
3.1 伪协作程序	17
3.1.1 伪对象	17
3.1.2 伪对象的两面	20
3.1.3 伪对象总结	20
3.1.4 模拟对象	21
第4章 接缝模型	22
4.1 大片的文本	22
4.2 接缝	23
4.3 接缝类型	25
4.3.1 预处理接缝	26
4.3.2 链接接缝	28
4.3.3 对象接缝	31
第5章 工具	36
5.1 自动化重构工具	36
5.2 模拟对象	38
5.3 单元测试用具	38
5.3.1 JUnit	39
5.3.2 CppUnitLite	40
5.3.3 NUnit	41
5.3.4 其他xUnit框架	42
5.4 一般测试用具	42
5.4.1 集成测试框架 (Framework for Integrated Test, FIT)	42
5.4.2 Fitnesse	43
第二部分 修改软件	
第6章 时间很紧张，但还需要修改	46
6.1 新生方法 (Sprout Method)	48
6.2 新生类 (Sprout Class)	50
6.3 包装方法	54
6.4 包装类	57
6.5 小结	61
第7章 永远都无法完成的修改	62
7.1 理解	62
7.2 延迟时间	63
7.3 打破依赖关系	63
7.4 构建依赖关系	64
7.5 小结	67
第8章 如何添加新特性	68
8.1 测试驱动开发	68
8.1.1 编写失败的测试案例	69
8.1.2 对其进行编译	69
8.1.3 使其通过	69

8.1.4 去除重复的内容	70
8.1.5 编写失败的测试案例	70
8.1.6 对其进行编译	70
8.1.7 使其通过	71
8.1.8 去除重复的内容	71
8.1.9 编写失败的测试案例	71
8.1.10 对其进行编译	71
8.1.11 使其通过	72
8.1.12 去除重复的内容	73
8.2 根据差异编程	74
8.3 小结	81
第9章 无法把类放到测试用具中	82
9.1 恶人的参数	82
9.2 具有隐藏依赖的情况	88
9.3 构造Blob的情况	90
9.4 恶人的全局依赖	92
9.5 可怕的Include依赖	99
9.6 洋葱皮参数	102
9.7 别名参数	104
第10章 无法在测试用具中运行方法	107
10.1 隐藏方法的情况	107
10.2 “有帮助的”语言特性	110
10.3 检测不到的副作用	112
第11章 我需要修改代码，应该测试哪些方法	119
11.1 推断影响	119
11.2 正向推理	124
11.3 影响传播	128
11.4 推理影响的工具	129
11.5 从影响分析中学习	131
11.6 简化影响草图	132
第12章 我需要在一个地方做多处变更，需要为所有涉及的类打破依赖关系吗	134
12.1 拦截点	135
12.1.1 简单的情况	135
12.1.2 更高层次的拦截点	137
12.2 使用夹点来判断设计	140
12.3 夹点陷阱	141
第13章 我需要修改代码，但不知道要编写哪些测试	142
13.1 鉴定测试	142
13.2 鉴定类	145
13.3 定向测试 (Targeted Testing)	146
13.4 编写鉴定测试的启示	150
第14章 对库的依赖让我快要崩溃了	151
第15章 应用全是API调用	153
第16章 对代码理解不够，所以无法修改	160
16.1 做笔记，画草图	160
16.2 列表标记	161
16.2.1 分离职责	162
16.2.2 理解方法结构	162
16.2.3 提取方法	162
16.2.4 理解变更的影响	162
16.3 临时重构	162
16.4 删除没有用的代码	163
第17章 应用没有结构	164
17.1 讲述系统的故事	165

17.2 裸CRC	167
17.3 对话审查 (Conversation Scrutiny)	170
第18章 测试代码挡路了	171
18.1 类命名规范	171
18.2 测试位置	172
第19章 项目并非面向对象，如何才能够安全地修改	174
19.1 简单的案例	174
19.2 困难的案例	175
19.3 增加新行为	178
19.4 充分利用面向对象	180
19.5 完全面向对象	183
第20章 类太大了，我不想让它继续膨胀	186
20.1 查看职责	188
20.2 其他技术	199
20.3 继续前进	199
20.3.1 策略	199
20.3.2 战术	200
20.4 提取类之后	201
第21章 在各个地方修改的都是同样的代码	202
第22章 我需要修改一个巨兽方法，但无法为其编写测试	218
22.1 巨兽的种类	218
22.1.1 无序方法	218
22.1.2 缠结的方法	219
22.2 使用自动重构支持来对付巨兽	221
22.3 手动重构挑战	224
22.3.1 引入检测变量	224
22.3.2 提取你所知道的内容	227
22.3.3 收集依赖	228
22.3.4 打破方法对象	229
22.4 策略	229
22.4.1 记下方法的概要	230
22.4.2 找到序列	230
22.4.3 首先提取到当前类	231
22.4.4 提取小段代码	231
22.4.5 准备好重做提取	231
第23章 如何知道没有造成任何破坏	232
23.1 超感编辑 (Hyperaware Editing)	232
23.2 单一目标编辑	233
23.3 保留签名	234
23.4 依赖于编译器	236
23.5 结对编程	238
第24章 我要崩溃了，它不会再有任何改进	239
第三部分 打破依赖的技术	
第25章 打破依赖的技术	242
25.1 调整参数	242
25.2 分解方法对象	245
25.3 完善定义	251
25.4 封装全局引用	252
25.5 暴露静态方法	257
25.6 提取并重写调用	259
25.7 提取并重写工厂方法	261
25.8 提取并重写getter方法	262
25.9 提取实现器	265
25.10 提取接口	269

25.11 引入实例委托器 274
25.12 引入静态设置器 275
25.13 链接替换 280
25.14 参数化构造函数 280
25.15 参数化方法 284
25.16 原始化参数 (Primitivize Parameter) 285
25.17 上推特性 287
25.18 下推依赖 290
25.19 使用函数指针替换函数 293
25.20 使用getter方法替换全局引用 295
25.21 创建子类并重写方法 297
25.22 替代实例变量 299
25.23 模板重定义 302
25.24 文本重定义 305
附录 重构 307
术语表 311
· · · · · (收起)

[修改代码的艺术](#) [下载链接1](#)

标签

重构

软件工程

程序设计

计算机

编程

代码重构

软件开发

计算机科学

评论

1. 原书绝对是一本经典 2. 侯伯薇翻译的这版简直太烂了，成吨的烂翻译和文字错误 3. 想看看刘未鹏翻译的版本。

实践性很浓的一本书，看过之后更有助于写出易于测试的系统

经典之作，改bug的境界

这里再重复一遍重构的定义——在保持代码行为的基础上，提升代码的质量。重构专注于第二步，即如何提升代码的质量，而修改代码的艺术专注于第一步，即如何保持代码的行为。

提升代码质量并不困难，但保持代码行为就难多了，尤其是对没有测试的遗留代码（Legacy Code）而言——你需要首先引入测试，但遗留代码往往可测试性（Testability）很差，这时你就需要把代码变的可测试。修改代码的艺术包含大量的实用建议，用来把代码变的可测试（Testable），从而使重构变为可能，使提高代码质量变为可能。

稍微有些繁琐，可以需要使用的时候再细翻。拆分超大的类那篇，需要的时候，发现并没有讲细节，还是自己想了一些安全透明的方法来处理工作中的一些超大类的拆分（3300+行）

深入，透彻，具体

很OO，很Java（虽然也有适用于C这种过程式语言以及C++这种常用语言的建议）
不全是教条，值得参考。

[修改代码的艺术](#) [下载链接1](#)

书评

作为一个程序员，获取知识是让我不断前进的动力，而读书是我获取知识的一条重要途径。在这个“经典”、“必读”过剩的年代里，大多数的书都仅仅扮演着传播知识的角色，真正改变自己对某些问题看法的书其实少之有少。限于读书时的眼界和能力，在我列表中，让我拍案惊奇的书只有...

当软件系统的规模随着时间不断增长时，我们怎么构建和维护它？面对别人写好的大量的代码基，如何进行后续的可持续开发？TDD，单元测试，重构，设计模式这些看上去很美的技术，是如何应用的？

毫无疑问，这本书里不可能提供上诉问题的所有答案，但是它至...

这本书看的时间非常长，断断续续有3个星期了吧，不错的书，至少对我来说是这样，因为我现在就碰到了书中列出的种种问题：对已有的没有完善的单元测试的核心系统进行重构。为了保证少出乱子，不出乱子，我必须小心的对超大类，巨型方法采用各种重构手段进行修改，没有单元测试作保...

《修改代码的艺术》看完了

这本书很薄，但是看起来还是很吃力，里面介绍了很多重构的知识，而且有很多C++的内容，有的地方也是似懂非懂的，如果了解设计模式和重构，就会轻松很多，可能艺术这东西，本身就不容易懂吧。里面对单元测试的依赖性很强，其实还是一本不错的书，你完...

买这本书的原因一是这本书确实是一本关于修改老代码的经典，二来翻译者是中国地区InfoQ的主编。但是入手看了大概到100多页之后实在是忍不住要上来吐槽一下。

首先是翻译的通畅性，应该说是比较烂的水准只能说是将将达到合格的水准，这个可能是个人的偏见。但是...

14h:05 in 6

days。我的“重构三部曲”之三，（另外两本是《重构》，《从重构到模式》，这三本书让我对代码的理解有重生之感。大部分书都是教你怎么从0开始写好代码，但是现实是经常从接手已有的项目开始，所以这三本就很有价值。）这本书压箱底8，9年了，前些年有次囫囵吞枣看...

Java重构的必读书，非常实用，但有的时候我想，Java代码的重构如此复杂，是否说明面向对象的设计思路在很多场合并不适用呢？比如很多服务端的逻辑本身是典型的函数转换，如果使用FP范型开发会简单的多。
推而广之，如果一种技术在大多数程序员手里都越用越复杂，以至于需要专家...

很好的实战经验，快来取道。在最近的开发项目中经常想起本书讲解的一些技术，受益匪浅。虽然我并不是 working on legacy code
，但是项目代码从无到有到完善也是经历几个阶段的，在不断演化，不断修正。另一方面，一边写单元测试，也参考了本书。以前以为测试只是为了保...

一两个月前看到了这本书，那时候正对编写高质量的代码很感兴趣，于是借来读。这一个月断断续续的读完，实际上读书的时间仅有10天左右的业余时间。读的很浅，但也有小小的收获。
这本书讲解如何在不漂亮的旧代码下写漂亮的新代码，依照先有测试后有功能的思想，作者全书都围绕...

如果你想重构,重要的前提就是有强力的测试.哪怕你有自动化重构工具在手.
如果你想对既有代码进行测试,你就必须先重构,因为代码根本就没有办法在测试工具中实例化.
新写的代码大多是可以先进行测试,然后再挂接到原有代码中.而对付遗留的代码,我们则需要一点点地把代码抠出...

如果你想重构,重要的前提就是有强力的测试.哪怕你有自动化重构工具在手.
如果你想对既有代码进行测试,你就必须先重构,因为代码根本就没有办法在测试工具中实例化.
新写的代码大多是可以先进行测试,然后再挂接到原有代码中.而对付遗留的代码,我们则需要一点点地把代码抠出...

我发现很多网页里卓越的报价常常比当当的高，可是实际情况是点击链接后卓越比当当低！比如这本书实际报价：卓越是46.5，当当是46.6
不知道是不是最近卓越大范围调整了价格？！
顺便说一下，在csdn读书频道上也有类似情况。

[修改代码的艺术](#) [下载链接1](#)