

<<重构>>

图书基本信息

书名：<<重构>>

13位ISBN编号：9787115221704

10位ISBN编号：7115221707

出版时间：2010

出版时间：人民邮电出版社

作者：Martin Fowler

页数：428

译者：熊节

版权说明：本站所提供下载的PDF图书仅提供预览和简介，请支持正版图书。

更多资源请访问：<http://www.tushu007.com>

## &lt;&lt;重构&gt;&gt;

## 前言

第一次听到“重构”这个词，是在2001年10月。

在当时，它的思想足以令我感到震撼。

软件自有其美感所在。

软件工程希望建立完美的需求与设计，按照既有的规范编写标准划一的代码，这是结构的美；快速迭代和RAD颠覆“全知全能”的神话，用近乎刀劈斧砍（crack）的方式解决问题，在混沌的循环往复中实现需求，这是解构的美；而Kent Beck与Martin Fowler两人站在一起，以XP那敏捷而又严谨的方法论演绎了重构的美——我不知道是谁最初把refactoring一词翻译为“重构”，或许无心插柳，却成了点睛之笔。

我一直是设计模式的爱好者。

曾经在我的思想中，软件开发应该有一个“理想国”——当然，在这个理想国维持着完美秩序的，不是哲学家，而是模式。

设计模式给我们的，不仅仅是一些具体问题的解决方案，更有追求完美“理型”的渴望。

但是，Joshua Kerievsky在那篇著名的《模式与XP》（收录于《极限编程研究》一书）中明白地指出：在设计前期使用模式常常导致过度工程（over-engineering）。

这是一个残酷的现实，单凭对完美的追求无法写出实用的代码，而“实用”是软件压倒一切的要素。

从一篇《停止过度工程》开始，Kerievsky撰写了“Refactoring to Patterns”系列文章。

这位犹太人用他民族性的睿智头脑，敏锐地发现了软件的后结构主义道路。

而让设计模式在飞速变化的网络时代重新闪现光辉的，又是重构的力量。

在一篇流传甚广的帖子里，有人把《重构》与《设计模式》并列为“Java行业的圣经”。

在我看来这种并列其实并不准确。

实际上，尽管我如此喜爱这本《重构》，但自从完成翻译之后，就再也没有读过它。

不，不是因为我已经对它烂熟于心，而是因为重构已经变成了我的另一种生活方式，变成了我每天的“面包与黄油”，变成了我们整个团队的空气与水，以至于无需再到书中寻找任何“神谕”。

而《设计模式》，我倒是放在手边时常翻阅，因为总是记得不那么真切。

## <<重构>>

### 内容概要

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。

## <<重构>>

### 作者简介

作者：(美国)福勒(Martin Fowler) 译者：熊节Martin Fowler，世界软件开发大师，在面向对象分析设计、UML、模式、XP和重构等领域都有卓越贡献，现为著名软件开发咨询公司ThoughtWorks的首席科学家。

他的多部著作《分析模式》、《UML精粹》和《企业应用架构模式》等都已经成为脍炙人口的经典。熊节，ThoughtWorks中国公司的高级咨询师、架构师和项目经理，在大型企业应用及互联网应用的架构和管理方面拥有丰富经验。

作为敏捷方法学顾问和重构专家，他拥有在各种技术平台、编程语言、软件形态的项目中实施重构的丰富经验，并曾主持极具挑战性的超大规模电信软件系列重构工作。

## &lt;&lt;重构&gt;&gt;

## 书籍目录

Chapter 1 : Refactoring , a First Example 重构, 第一个例子 The Starting Point 起点 The First Step in Refactoring 重构第一步 Decomposing and Redistributing the Statement Method 分解并重组statement方法 Replacing the Conditional Logic on Price Code with Polymorphism 用多态代替价格条件逻辑代码 Final Thoughts 结语 Chapter 2 : Principles in Refactoring 重构原则 Defining Refactoring 何谓重构 Why Should You Refactor? 为何重构 When Should You Refactor? 何时重构 What Do I Tell My Manager? 怎样说服经理 Problems with Refactoring 重构的问题 Refactoring and Design 重构与设计 Refactoring and Performance 重构与性能 Where Did Refactoring Come From? 重构的起源 Chapter 3 : Bad Smells in Code(by Kent Beck and Martin Fowler) 代码坏味 Duplicated Code 重复代码 Long Method 过长方法 Large Class 过长类 Long Parameter List 过长参数列表 Divergent Change 发散式变化 Shotgun Surgery 霰弹式修改 Feature Envy 特性依恋 Data Clumps 数据泥团 Primitive Obsession 基本类型偏执 Switch Statements switch语句 Parallel Inheritance Hierarchies 平行继承体系 Lazy Class 冗余类 Speculative Generality 理论上的一般性 Temporary Field 临时字段 Message Chains 消息链 Middle Man 中间人 Inappropriate Intimacy 过度亲密 Alternative Classes with Different Interfaces 接口不同的等效类 Incomplete Library Class 不完整的库类 Data Class 数据类 Refused Bequest 拒绝继承 Comments 注释过多 Chapter 4 : Building Tests 构建测试 The Value of Self-testing Code 自测试代码的重要性 The JUnit Testing Framework Junit测试框架 Adding More Tests 添加更多测试 Chapter 5 : Toward a Catalog of Refactorings 重构目录 Format of the Refactorings 重构描述的格式 Finding References 寻找引用 How Mature Are These Refactorings? 这些重构的成熟度如何 Chapter 6 : Composing Methods 组合方法 Extract Method 提取方法 Inline Method 内联方法 Inline Temp 内联临时变量 \*Replace Temp with Query 用查询方法代替临时变量 Introduce Explaining Variable 引入解释性变量 Split Temporary Variable 分离临时变量 \*Remove Assignments to Parameters 去除参数赋值 Replace Method with Method Object 用方法对象代替方法 Substitute Algorithm 替换算法 Chapter 7 : Moving Features Between Objects 在对象之间移动特性 \*Move Method 移动方法 Move Field 移动字段 Extract Class 提取类 Inline Class 内联类 Hide Delegate 隐藏委托类 Remove Middle Man 去除中间人 Introduce Foreign Method 引入外加方法 \*Introduce Local Extension 引入本地扩展类 Chapter 8 : Organizing Data 组织数据 Self Encapsulate Field 自封装字段 Replace Data Value with Object 用对象代替数据值 Change Value to Reference 将值对象改为引用对象 Change Reference to Value 将引用对象改为值对象 Replace Array with Object 用对象代替数组 Duplicate Observed Data 重复被观察数据 \*Change Unidirectional Association to Bidirectional 将单向关联改为双向 Change Bidirectional Association to Unidirectional 将双向关联改为单向 \*Replace Magic Number with Symbolic Constant 用字面常量代替魔数 Encapsulate Field 封装字段 Encapsulate Collection 封装集合 Replace Record with Data Class 用数据类代替记录 \*Replace Type Code with Class 用类代替类型码 Replace Type Code with Subclasses 用子类代替类型码 Replace Type Code with State/Strategy 用State/Strategy 代替类型码 Replace Subclass with Fields 用字段代替子类 Chapter 9 : Simplifying Conditional Expressions 简化条件语句 Decompose Conditional 分解条件语句 Consolidate Conditional Expression 合并条件语句 Consolidate Duplicate Conditional Fragments 合并重复的条件片段 Remove Control Flag 去除控制标志 Replace Nested Conditional with Guard Clauses 用守卫语句代替嵌套条件语句 Replace Conditional with Polymorphism 用多态代替条件语句 Introduce Null Object 引入Null对象 Introduce Assertion 引入断言 Chapter 10 : Making Method Calls Simpler 简化方法调用 Rename Method 重命名方法 Add Parameter 添加参数 Remove Parameter 去除参数 Separate query from Modifier 将查询方法与修改方法分离 Parameterize Method 参数化方法 Replace Parameter with Explicit Methods 用显式方法代替参数 Preserve Whole Object 保持对象完整 Replace Parameter with Method 用方法代替参数 Introduce Parameter Object 引入参数对象 Remove Setting Method 去除设置方法 Hide Method 隐藏方法 Replace Constructor with Factory Method 用工厂方法代替构造器 Encapsulate

## &lt;&lt;重构&gt;&gt;

Downcast 封装向下转型    Replace Error Code with Exception 用异常代替错误码    Replace Exception with Test 用测试代替异常    Chapter 11 : Dealing with Generalization 处理泛化关系    Pull Up Field 上移字段    Pull UP Method 上移方法    Pull Up Constructor Body 上移构造器主体    Push Down Method 下移方法    Push Down Field 下移字段    Extract Subclass 提取子类    Extract Superclass 提取超类    Extract Interface 提取接口    Collapse Hierarchy 合并继承层次    Form Template Method 形成Template Method    Replace Inheritance with Delegation 用委托代替继承    Replace Delegation with Inheritance 用继承代替委托    Chapter 12 : Big Refactorings(by Kent Beck and Martin Fowler) 大型重构    Tease Apart Inheritance 分解继承层次    Convert Procedural Design to Objects 将过程式设计转换为面向对象    Separate Domain from Presentation 将领域逻辑与表现分离    Extract Hierarchy 提取继承层次    Chapter 13 : Refactoring , Reuse , and Reality(by William Opdyke) 重构 , 复用与现实    A Reality Check 现实的检验    Why Are Developers Reluctant to Refactor Their Programs ? 开发人员为何不愿重构程序    A Reality Check(Revisited) 再谈现实的检验    Resources and References for Refactoring 重构的资源 and 参考文献    Implications Regarding Software Reuse and Technology Transfer 对软件复用与技术传播的意义    A Final Note 结语    References 参考文献    Chapter 14 : Refactoring Tools(by Don Roberts and John Brant) 重构工具    Refactoring with a Tool 使用工具重构    Technical Criteria for a Refactoring Tool 重构工具的技术标准    Practical Criteria for a Refactoring Tool 重构工具的实用标准    Wrap Up 结语    Chapter 15 : Putting It All Together(by Kent Beck) 集大成    References 参考文献    List of Soundbites 要点列表    Updates 更新内容    Index 索引

## &lt;&lt;重构&gt;&gt;

## 章节摘录

插图：现在，重构的处境也是如此。

我们知道重构的好处，我们知道重构可以给我们的工作带来立竿见影的改变。

但是我们还没有获得足够的经验，我们还看不到它的局限性。

这一节比我希望的要短。

暂且如此吧。

随着更多人学会重构技巧，我们也将对它有更多了解。

对你而言这意味着：虽然我坚决认为你应该尝试一下重构，获得它所提供的利益，但与此同时，你也应该时时监控其过程，注意寻找重构可能引入的问题。

请让我们知道你所遭遇的问题。

随着对重构的了解日益增多，我们将找出更多解决办法，并清楚知道哪些问题是真正难以解决的。

数据库重构经常出问题的一个领域就是数据库。

绝大多数商用程序都与它们背后的数据库结构紧密耦合在一起，这也是数据库结构如此难以修改的原因之一。

另一个原因是数据迁移（migration）。

就算你非常小心地将系统分层，将数据库结构和对象模型间的依赖降至最低，但数据库结构的改变还是让你不得不迁移所有数据，这可能是件漫长而烦琐的工作。

在非对象数据库中，解决这个问题的办法之一就是：在对象模型和数据库模型之间插入一个分隔层，这就可以隔离两个模型各自的变化。

升级某一模型时无需同时升级另一模型，只需升级上述的分隔层即可。

这样的分隔层会增加系统复杂度，但可以给你带来很大的灵活度。

如果你同时拥有多个数据库，或如果数据库模型较为复杂使你难以控制，那么即使不进行重构，这分隔层也是很重要的。

你无需一开始就插入分隔层，可以在发现对象模型变得不稳定时再产生它，这样你就可以为你的改变找到最好的平衡点。

对开发者而言，对象数据库既有帮助也有妨碍。

某些面向对象数据库提供不同版本的对象之间的自动迁移功能，这减少了数据迁移时的工作量，但还是会损失一定时间。

如果各数据库之间的数据迁移并非自动进行，你就必须自行完成迁移工作，这个工作量可是很大的。

这种情况下你必须更加留神类中的数据结构变化。

你仍然可以放心将类的行为转移过去，但转移字段时必须格外小心。

数据尚未被转移前你就得先运用访问函数造成“数据已经转移”的假象。

一旦你确定知道数据应该放在何处，就可以一次性地将数据迁移过去。

## <<重构>>

### 编辑推荐

重构，一言以蔽之，就是在不改变外部行为的前提下，有条不紊地改善代码。

多年前，正是《重构:改善既有代码的设计》原版的出版，使重构终于从编程高手们的小圈子走出，成为众多普通程序员日常开发工作中不可或缺的一部分。

《重构:改善既有代码的设计》也因此成为与《设计模式》齐名的经典著作，被译为中、德、俄、日等众多语言，在世界范围内畅销不衰。

《重构:改善既有代码的设计》凝聚了软件开发社区专家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。

今天，无论是重构本身，业界对重构的理解，还是开发工具对重构的支持力度，都与《重构:改善既有代码的设计》最初出版时不可同日而语，但书中所蕴涵的意味和精华，依然值得反复咀嚼，而且往往能够常读常新。



#### 版权说明

本站所提供下载的PDF图书仅提供预览和简介，请支持正版图书。

更多资源请访问:<http://www.tushu007.com>