PEARSON
Addison Wesley

典藏原版书苑

Implementation Patterns

[美] Kent Beck 著

实现模式（英文版）

“Kent是用代码来沟通的大师，他的代码条分缕析，明晓晰，如益友携手，如良师解惑，令人豁然开朗。”

—— Erich Gamma，IBM杰出工程师

人民邮电出版社
POSTS & TELECOM PRESS

典藏原版书苑

实现模式

（英文版）

[美] Kent Beck 著

人民邮电出版社
北京

图书在版编目（CIP）数据

实现模式：英文 /（美）贝克（Beck,K.）著．—北京：人民邮电出版社，2008.11
（典藏原版书苑）
ISBN 978-7-115-18709-3

Ⅰ．实… Ⅱ．贝… Ⅲ．程序设计—英文 Ⅳ．TP311.1

中国版本图书馆 CIP 数据核字（2008）第 129535 号

版 权 声 明

典藏原版书苑
实现模式（英文版）

◆ 著　　[美] Kent Beck
责任编辑　刘映欣
◆ 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
北京铭成印刷有限公司印刷
◆ 开本：700×1000　1/16
印张：10.75
字数：219 千字　　2008 年 11 月第 1 版
印数：1－3 500 册　　2008 年 11 月北京第 1 次印刷

著作权合同登记号　图字：01-2008-3845 号

ISBN 978-7-115-18709-3/TP

定价：29.00 元

读者服务热线：(010)67132705　印装质量热线：(010)67129223
反盗版热线：(010)67171154

内容提要

在本书中，作者将自己多年形成的编程习惯以及阅读既有代码的体验凝练成了编程中的价值观、原则和 77 种实现模式。

沟通、简单和灵活的价值观应当被所有开发人员所铭记。局部影响、最小化重复、将逻辑与数据捆绑等原则同样是通用性的指导思想，比价值观更贴近编程场景，在价值观和模式之间搭建了桥梁。在 77 个实现模式中，每一个模式都覆盖了编写简洁、清晰、易扩展、易维护的代码这一原则的某个方面。它们为日常的编程提供了丰富翔实的参考依据，并告诉大家这些代码如何为降低沟通成本和提高有效产出提供保障。

本书适用于各个阶段的开发者群体，刚刚涉足软件开发领域的新人能够透过大师的眼睛来看待编程，了解编程的价值观与原则；而具有丰富经验的资深工程师则可以通过这些模式进行反思，探究成功实践背后的意义。把价值观、原则和开发实践结合之后，日常开发工作便会以崭新迷人的形式呈现在我们面前。

Praise for *Implementation Patterns*

"Kent is a master at creating code that communicates well, is easy to understand, and is a pleasure to read. Every chapter of this book contains excellent explanations and insights into the smaller but important decisions we continuously have to make when creating quality code and classes."

—Erich Gamma, IBM Distinguished Engineer

"Many teams have a master developer who makes a rapid stream of good decisions all day long. Their code is easy to understand, quick to modify, and feels safe and comfortable to work with. If you ask how they thought to write something the way they did, they always have a good reason. This book will help you become the master developer on your team. The breadth and depth of topics will engage veteran programmers, who will pick up new tricks and improve on old habits, while the clarity makes it accessible to even novice developers."

—Russ Rufer, Silicon Valley Patterns Group

"Many people don't realize how readable code can be and how valuable that readability is. Kent has taught me so much, I'm glad this book gives everyone the chance to learn from him."

—Martin Fowler, chief scientist, ThoughtWorks

"Code should be worth reading, not just by the compiler, but by humans. Kent Beck distilled his experience into a cohesive collection of implementation patterns. These nuggets of advice will make your code truly worth reading."

—Gregor Hohpe, author of Enterprise Integration Patterns

"In this book Kent Beck shows how writing clear and readable code follows from the application of simple principles. *Implementation Patterns* will help developers write intention revealing code that is both easy to understand and flexible towards future extensions. A must read for developers who are serious about their code."

—Sven Gorts

"*Implementation Patterns* bridges the gap between design and coding. Beck introduces a new way of thinking about programming by basing his discussion on values and principles."

—Diomidis Spinellis, author of Code Reading *and* Code Quality

To Cindee: Thank you for your encouragement, insistence, food, soothing, irritation, editing, and tea. The dedication of a book is as a raisin to an elephant compared to what you give me. Bless you.

序

这是一本关于如何写好代码的书。

如果你不认为写好代码是一件重要、困难而且有趣的事，请立即放下这本书。

什么是好的代码？可以工作的、性能良好的、不出 bug 的代码，就是好的代码吗？

所谓好的代码，除了其他所有要求以外，还应该清晰准确地传达写作者的想法。

Martin Fowler 在《重构：改善既有代码的设计》里说，“任何一个傻瓜都能写出机器能懂的代码。好的程序员应该写出人能懂的代码。”

如果你不同意这句话，请立即放下这本书。因为这是一本关于如何用代码与他人（而非机器）沟通的书。

任何读到这一行的程序员都应该读完这本书。

Steve McConnell 在《代码大全》里说，“不要过早优化，但也不要过早劣化”。这本书将告诉你如何在几乎不引入任何额外成本的前提下避免一些常见的低级错误——它们是常见的，因为几乎每个人都犯过并且还在犯着这些错误。

如果你确实没有时间，至少应该读完第 6 章“状态”。因为在各种常见的低级错误中最常见者就是关于“什么信息在什么地方”的决策错误。

在这样一本书的序言里说任何废话都将是佛头着粪。

所以，现在就祝你阅读愉快、编程愉快。

是为序。

——InfoQ 中文站

前　言

这是一本关于编程的书，更具体一点，是关于“如何编写别人能懂的代码”的书。编写出别人能读懂的代码没有任何神奇之处，这就与任何其他形式的写作一样：了解你的读者，在脑子里构想一个清晰的整体结构，让每个细节为故事的整体作出贡献。Java 提供了一些很好的交流机制，本书介绍的实现模式正是一些 Java 编程习惯，它们能让你编写出的代码更加易读。

也可以把实现模式看作一次思考，“关于这段代码，我想要告诉读者什么？”程序员大部分的时间都在自己的世界里绞尽脑汁，以至于用别人的视角来看待世界对他们来说是一次重大的转变。他们不仅要考虑“计算机会用这段代码做什么”，还要考虑“如何用这段代码与别人沟通我的想法”。这种视角上的转换有利于你的健康，也很可能有利于你的钱包，因为在软件开发中有大量的开销都被用在理解现有代码上了。

有一个叫做 Jeopardy 的美国游戏节目，由主持人给出问题的答案，参赛观众则来猜问题是什么。“猜一个词，表示扔出窗外。”“是 defenestration 吗？”“答对了。”

编程就好像 Jeopardy 游戏：答案用 Java 的基本语言构造给出，程序员则经常需要找出问题究竟是什么，即这些语言构造究竟是在解决什么问题。譬如说，如果答案是“把一个字段声明为 Set”，那么问题可能就是“我要怎样告诉其他程序员，这是一个不允许包含重复元素的集合？”本书介绍的实现模式列举了一组常见的编程问题，还有 Java 解决这些问题的方式。

和软件开发一样，范围管理对于写书同样重要。我现在就告诉你本书不是什么。它不是编程风格指南，因为其中包含了太多的解释，最终的决定权则完全交给你。它不是设计书籍，因为其中关注的主要是小范围的、程序员每天要做很多次的决策。它不是模式书籍，因为这些实现模式的格式各不相同、随需而变。它不是语言书籍，因为尽管其中谈到了一些 Java 语言的特性，但我在写作时假设你已经熟悉 Java 了。

实际上本书建立在一个相当不可靠的前提之上：好的代码是有意义的。我见过太多丑陋的代码给它们的主人赚着大把钞票，所以在我看来，软件要取得商业成功或者被广泛使

用，“好的代码质量”既不必要也不充分。即便如此，我仍然相信，尽管代码质量不能保证美好的未来，它仍然有其意义：有了质量良好的代码以后，业务需求能够被充满信心地开发和交付，软件用户能够及时调整方向以便应对机遇和竞争，开发团队能够在挑战和挫折面前保持高昂的斗志。总而言之，比起质量低劣、错误重重的代码，好的代码更有可能帮助用户取得业务上的成功。

即便不考虑长期的经济效应，我仍然会选择尽我所能地编写出好代码。就算活到古稀之年，你的一生也只有二十多亿秒而已，这宝贵的时间不该被浪费在不能被自己引以为傲的工作上。编写好的代码带给我满足感，不仅因为编程本身，还因为知道别人能够理解、欣赏、使用和扩展我的工作。

所以，说到底，这是一本关于责任的书。作为一个程序员，你拥有时间、拥有才华、拥有金钱、拥有机会。对于这些天赐的礼物，你要如何负责地使用？下面的篇幅里包含了我对于这个问题的答案：不仅为我自己、为我的 CPU 老弟编程，也为其他人编程。

致谢

我首先、最后并且始终要感谢 Cynthia Andres，我的搭档、编辑、支持者和首席讨债鬼。我的朋友 Paul Petralia 推动了这本书的写作，而且不断给我鼓励的电话。编辑 Chris Guzikowski 和我通过本书学会了如何在一起工作，他从 Pearson 的角度给了我一切需要的支持，让我能够完成写作。还要感谢 Pearson 的制作团队：Julie Nahil、John Fuller 和 Cynthia Kogut。Jennifer Kohnke 的插图不但包含了丰富的信息，而且非常人性化。本书的审阅者给我的书稿提供了清晰而又及时的反馈，为此我要感谢 Erich Gamma、Steve Metsker、Diomidis Spinellis、Tom deMarco、Michael Feathers、Doug Lea、Brad Abrams、Cliff Click、Pekka Abrahamson、Gregor Hohpe 和 Michele Marchesi。感谢 David Saff 指出“状态”与“行为”之间的平衡。最后，还要感谢我的孩子们，一想到他们乖乖呆在家里，我就有了尽快完成本书的动力。Lincoln、Lindsey、Forrest 和 Joëlle Andres，感谢你们。

Contents

Chapter 1

Introduction

Here we are together. You've picked up my book (it's yours now). You already write code. You have probably already developed a style of your own through your own experiences.

The goal of this book is to help you communicate your intentions through your code. The book begins with an overview of programming and patterns (chapters 2-4). The remainder of the book (chapters 5-8) is a series of short essays, patterns, on how to use the features of Java to write readable code. It closes with a chapter on how to modify the advice here if you are writing frameworks instead of applications. Throughout, the book is focused on programming techniques that enhance communication.

There are several steps to communicating through code. First I had to become conscious while programming. I had been programming for years when I first started writing implementation patterns. I was astonished to discover that, even though programming decisions came smoothly and quickly to me, I couldn't explain why I was so sure a method should be called such-and-so or that a bit of logic belonged in this object over here. The first step towards communicating was slowing down long enough to become aware of what I was thinking, to stop pretending that I coded by instinct.

The second step was acknowledging the importance of other people. I found programming satisfying, but I am self-centered. Before I could write communicative code I needed to believe that other people were as important as I was. Programming is hardly ever a solitary communion between one man and one machine. Caring about other people is a conscious decision, and one that requires practice.

Which brings me to the third step. Once I had exposed my thinking to sunlight and fresh air and acknowledged that other people had as much right to exist as I did, I needed to demonstrate my new perspective in practice. I use the implementation patterns here to program consciously and for others as well as myself.

You can read this book strictly for technical content—useful tricks with explanations. However, I thought it fair to warn you that there is a whole lot more going on, at least for me.

You can find those technical bits by thumbing through the patterns chapters. One effective strategy for learning this material is to read it just before you need to use it. To read it "just-in-time", I suggest skipping right to chapter 5 and skimming through to the end, then keeping the book by you as you program. After you've used many of the patterns, you can come back to the introductory material for the philosophical background behind the ideas you've been using.

If you are interested in a thorough understanding of the material here, you can read straight through from the beginning. Unlike most of my books, however, the chapters here are quite long, so it will take concentration on your part to read end-to-end.

Most of the material in this book is organized as patterns. Most decisions in programming are similar to decisions that have come before. You might name a million variables in your programming career. You don't come up with a completely novel approach to naming each variable. The general constraints on naming are always the same: you need to convey the purpose, type, and lifetime of the variable to readers, you need to pick a name that's easy to read, you need to pick a name that's easy to write and format. Add to these general constraints the specifics of a particular variable and you come up with a workable name. Naming variables is an example of a pattern: the decision and its constraints repeat even though you might create a different name each time.

I think patterns often need different presentations. Sometimes an argumentative essay best explains a pattern, sometimes a diagram, sometimes a teaching story, sometimes an example. Rather than cram each pattern's description into a rigid format, I have described each in the way I thought best.

This book contains 77 explicitly named patterns, each covering some aspect of writing readable code. In addition, there are many smaller patterns or variants of patterns that I mention in passing. My goal with this book is to offer advice for how to approach most common, daily coding tasks so as to help future readers understand what the code is supposed to do.

This book fits somewhere between *Design Patterns* and a Java language manual. *Design Patterns* talks about decisions you might make a few times a day while developing, typically decisions that regulate the interaction between objects. You apply an implementation pattern every few seconds while programming. While language manuals are good at describing what you can do with Java, they don't talk much about why you would use a certain construct or what someone reading your code is likely to conclude from it.

Part of my philosophy in writing this book has been to stick to topics I know well. Concurrency issues, for example, are not addressed in these implementation patterns, not because concurrency isn't an important issue, but rather because it is not one on which I have a lot to say. My concurrency strategy has always been to isolate as much as possible concurrent parts of my applications. While I am generally successful in doing so, it's not something I can explain. I recommend a book such as *Java Concurrency in Practice* for a practical look at concurrency.

Another topic not addressed in this book is any notion of software process. The advice about communicating through code here is intended to work whether that code is written near the end of a long cycle or seconds after a failing test has been written. Software that costs less overall is good to have, whatever the sociological trappings within which it is written.

I also stop short of the edges of Java. I tend to be conservative in my technology choices because I have been burned too often pushing new features to their limits (it's a fine learning strategy but too risky for most development). So, you'll find here a pedestrian subset of Java. If you are motivated to use the latest features of Java, you can learn them from other sources.

Tour Guide

The book is divided into seven major sections as seen in Figure 1.1. Here they are:

- Introduction—these short chapters describe the importance and value of communicating through code and the philosophy behind patterns.
- Class—patterns describing how and why you might create classes and how classes encode logic.
- State—patterns for storing and retrieving state.
- Behavior—patterns for representing logic, especially alternative paths.
- Method—patterns for writing methods, reminding you what readers are likely to conclude from your choice of method decomposition and names.
- Collections—patterns for choosing and using collections.
- Evolving Frameworks—variations on the preceding patterns when building frameworks instead of applications.