PEARSON Addison Wesley

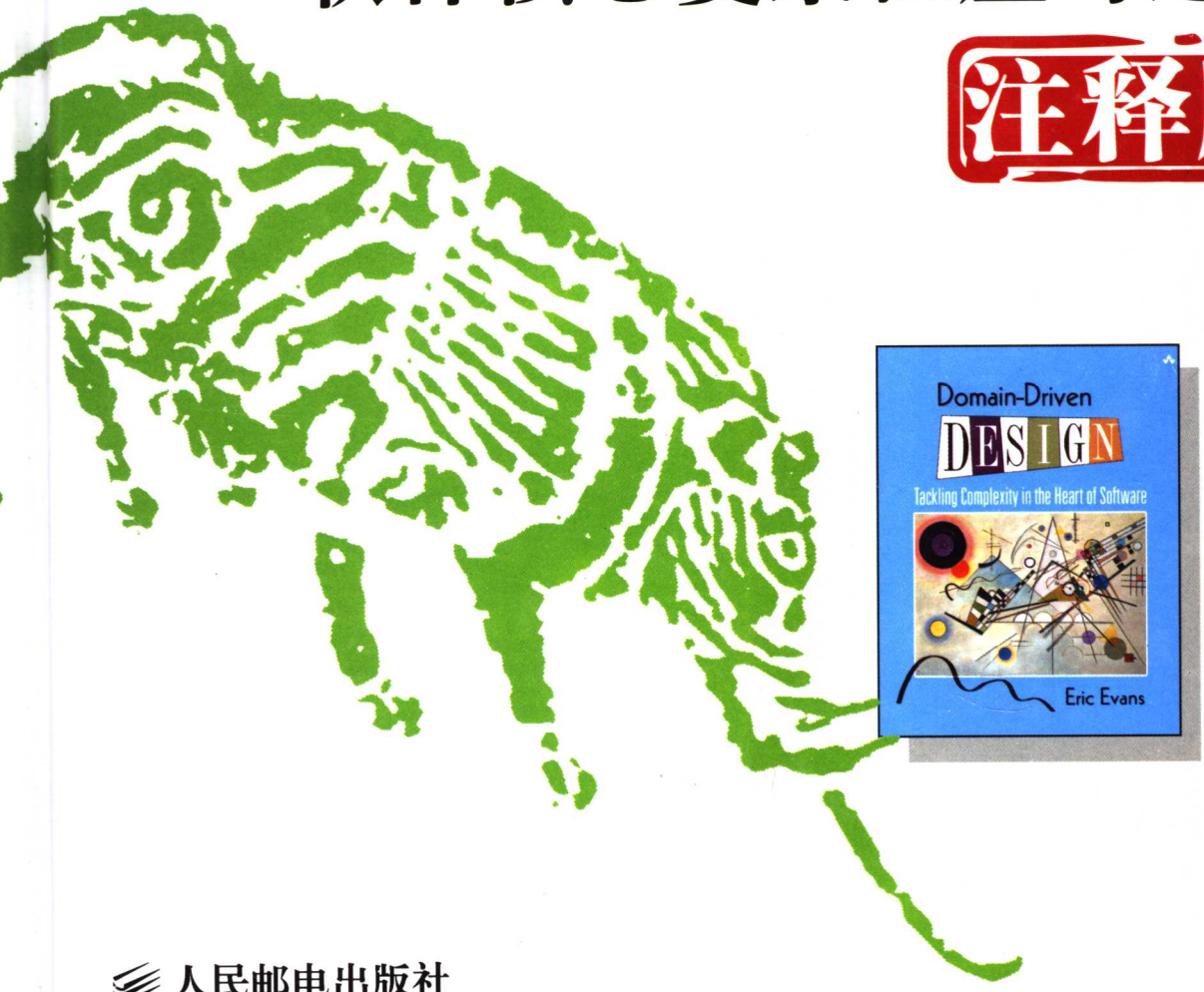
典藏原版书苑

Domain-Driven Design
Tackling Complexity in the Heart of Software

[美] Eric Evans 著
孙向晖 注释

领域驱动设计
——软件核心复杂性应对之道

注释版



人民邮电出版社
POSTS & TELECOM PRESS

领域驱动设计——软件核心复杂性应对之道

（注释版）

[美] Eric Evans 著

孙向晖 注释

人民邮电出版社

北京

图书在版编目（CIP）数据

领域驱动设计：软件核心复杂性应对之道：注释版：英文/（美）埃文斯（Evans, E.）著；孙向晖注释. —北京：人民邮电出版社，2007.11
（典藏原版书苑）
ISBN 978-7-115-15618-1

Ⅰ. 领… Ⅱ. ①埃…②孙… Ⅲ. 软件设计—英文 Ⅳ. TP311.5

中国版本图书馆 CIP 数据核字（2007）第 109540 号

版 权 声 明

Original edition, entitled Domain-Driven Design: Tackling Complexity in the Heart of Software, 0321125215 by Eric Evans, published by Pearson Education, Inc, publishing as Addison-Wesley, Copyright © 2004 by Eric Evans. All rights reserved. No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording or by any information storage retrieval system, without permission from Pearson Education, Inc.

China edition published by PEARSON EDUCATION ASIA LTD., and POSTS & TELECOMMUNICATIONS PRESS Copyright © 2007.

This edition is manufactured in the People's Republic of China, and is authorized for sale only in People's Republic of China excluding Hong Kong, Macau and Taiwan.

仅限于中华人民共和国境内（不包括中国香港、澳门特别行政区和中国台湾地区）销售。

本书封面贴有 Pearson Education (培生教育出版集团) 激光防伪标签。无标签者不得销售。

典藏原版书苑

领域驱动设计——软件核心复杂性应对之道（注释版）

◆ 著 ［美］Eric Evans
注 释 孙向晖
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
新华书店总店北京发行所经销

◆ 开本：800×1000 1/16
印张：35.25
字数：786 千字 2007 年 11 月第 1 版
印数：1－3 000 册 2007 年 11 月北京第 1 次印刷
著作权合同登记号 图字：01-2006-4173 号

ISBN 978-7-115-15618-1/TP

定价：89.00 元

读者服务热线：(010)67132705 印装质量热线：(010)67129223

内容提要

本书向读者介绍了领域驱动设计的系统化方法，展现了针对设计的可扩展的最佳实践集合，展示了通过经验验证过的技术以及处理软件开发项目所面对的复杂领域的基本原则。本书围绕设计和开发的实践，结合若干真实项目中的案例，向读者阐述如何在真实的软件开发中应用领域驱动设计。

在注释内容中，对原文中的案例背景、相关知识和作者要体现的要点做了重点的阐述，并将注释者在软件工程中宝贵的实践经验与读者共同分享，以便帮助读者对“领域驱动”这一主题进行发散性的、系统化的思考。

出版说明

正如软件工程名家 Roger S. Pressman 所言，“软件工程从少数拥护者所实践的朦胧的思想，演化成一个正式的工程学科。今天，它已经被承认为一个值得认真地研究、细心地学习和热烈地争论的主题。”

在计算机专业书籍中，软件工程领域的许多经典可能既是最易阅读的，又是最需要花费时间去领悟的。它既不富含烦琐的形式化推导，亦无特定的运行环境，更不充斥确定的源代码，读者可以一目十行地顺畅浏览软件工程书籍中大段的文字描述，不会因为时而出现的代码片段或公式阻碍阅读速度，然而读毕掩卷，有时一片茫然，犹如入宝山而空手归，有时满怀感想，却一时理不出头绪。此刻，如果能在书边看到行家的注解或点评，无论是旁敲侧击，还是当头棒喝，常能有拨云见日、豁然开朗之效。中国古典文化有注疏传统，将这类述而不作的经典诠释方式结合到新兴的技术学科中，未尝不是一种有益的探索。

由于软件工程著作之精妙之处常在概念和方法，故而多半都是自然语言的文字论述，而并不以程序语言为载体，这使得对软件工程著作的翻译相比其他领域更富挑战性。有如文学作品的翻译，往往难以顾全“信、达、雅”，软件工程经典的译者，也很少能有自信声明自己的译文在保持技术准确的同时流畅地传递了原文的语言个性——很多思维的微妙之处，都在原著文字的起承转合和字里行间。为原文提供中文注释，在技术上的启发意义之外也能扫清一些语言和文化差异造成的障碍，让更多力所能及的读者都能方便地直接欣赏原文，窥经典之全豹。

另一方面，正如 Roger S. Pressman 说：“软件工程将发生变化——对此我们可以肯定。” Frederick P. Brooks 在《人月神话》中所倡导的“唯一不变的是变化本身”，人们只有在变化中才能体味永恒。在当前的实践和技术氛围中咀嚼软件工程的理论，这样才是这一学科真正的活力源泉。注释者结合软件工程近年发展趋势，重访当年的经典，令读者能在软件技术发展的脉络中领会经典的精粹。借用 UMLChina 团队的口号“软件以用为本”，经典终究是为实践服务的，而我们出注释版图书最终的目的也是希望通过国内相关领域专业人士的注释，令英文原版符合国内读者的阅读需求，为原文增添技术上的辅助、语言上的答疑，以及抛砖引玉，激发读者的思辨。

此为试读，需要完整PDF请访问：www.ertongbook.com

Praise for *Domain-Driven Design*

"This book belongs on the shelf of every thoughtful software developer."

—Kent Beck

"Eric Evans has written a fantastic book on how you can make the design of your software match your mental model of the problem domain you are addressing.

"His book is very compatible with XP. It is not about drawing pictures of a domain; it is about how you think of it, the language you use to talk about it, and how you organize your software to reflect your improving understanding of it. Eric thinks that learning about your problem domain is as likely to happen at the end of your project as at the beginning, and so refactoring is a big part of his technique.

"The book is a fun read. Eric has lots of interesting stories, and he has a way with words. I see this book as essential reading for software developers—it is a future classic."

—Ralph Johnson, author of *Design Patterns*

"If you don't think you are getting value from your investment in object-oriented programming, this book will tell you what you've forgotten to do."

—Ward Cunningham

"What Eric has managed to capture is a part of the design process that experienced object designers have always used, but that we have been singularly unsuccessful as a group in conveying to the rest of the industry. We've given away bits and pieces of this knowledge . . . but we've never organized and systematized the principles of building domain logic. This book is important."

—Kyle Brown, author of *Enterprise Java Programming with IBM WebSphere*

"Eric Evans convincingly argues for the importance of domain modeling as the central focus of development and provides a solid framework and set of techniques for accomplishing it. This is timeless wisdom, and will hold up long after the methodologies *du jour* have gone out of fashion."

—Dave Collins, author of *Designing Object-Oriented User Interfaces*

"Eric weaves real-world experience modeling—and building—business applications into a practical, useful book. Written from the perspective of a trusted practitioner, Eric's descriptions of ubiquitous language, the benefits of sharing models with users, object life-cycle management, logical and physical application structuring, and the process and results of deep refactoring are major contributions to our field."

—Luke Hohmann, author of *Beyond Software Architecture*

在实践中展望

我所在的团队，是一个创新型的团队，给我足够的空间和权力，让我放手在若干项目中实践领域建模。如果您不是一个“悲观了的完美主义者”，就应该完全有理由相信，我们可以将“领域驱动设计”应用到更大规模的系统中去。即使在动态语言兴起和 SOA 流行的现在和将来，“领域驱动设计”也将会是引领我们的王者正道。这是我的实践结论。

几乎每一种语言、开发框架和工具面世的时候，都希望有这样的效果：把开发人员解放出来，让他（她）更专注于对业务逻辑的处理。可惜，这往往被人忽视，很多人热衷于（甚至有些狂热地）追逐着新的语言和工具，却忽视了真正需要关注的内容。应该按照什么样的方式专注于对业务逻辑的处理？有好的成功经验吗？本书本着对软件开发最本质、最朴素的还原态度，给出了很好的参考框架。对于这样一本书，如果能读通读透，那将会有事半功倍的收效。而能够为这样一本书做注释，我感到非常荣幸。

在中国经典图书《石头记》的前 80 回，有若干的脂砚斋评本，据说，根据评本中的注释，读者会读出一个“宝黛悲剧”的别样故事。我没有“脂砚斋”那样的才华，也不敢说对 Eric Evans 的思维掌握颇深，但我是一个领域建模的实践者。沿着 Martin Fowler 的路线，从《分析模式》到《企业应用架构模式》一路走来，我都尽最大能力在参与的项目中进行选择性的实践。

《如何阅读一本书》中提到，最高层次的阅读是主题阅读。在获得为本书做注释的机会之前，我就已经对这本书研读了很多遍。在阅读本书的过程中，我将类似主题的有异同的观点，在书侧做了引用和连接，这些都是本书注释内容的雏形。这些内容或许有助于读者对“领域驱动”这一主题进行发散性、系统化的思考。

对于我来说，读书和思考是一个渐进的过程，正所谓“书读百遍”，再回首时，读书时的困顿和思路清醒后顿悟的喜悦，我都不吝笔墨记录下来。如今这些点滴感悟都已成为这本书的注释内容，这样，读者在阅读时如若遇到了同样的问题，可以借鉴一二。

深入理解书籍中的案例作为一个屡试不爽的阅读方式，能够让阅读者快速把握住作者思想中的精髓。类似于《分析模式》等领域相关的书籍，本书原著中列举的实例有着深厚的业务背景，可能会影响读者的阅读效果。因此在注释过程中，对案例背景、相关知识和作者要

体现的要点，我都做了重点的阐述。

“尽信书则不如无书”。国内的应用，有着更丰富的领域逻辑，有着更广阔的行业特性，这是国外环境所不具备的。所以，避免全盘的照搬，进行选择性的实践，才是正道。我在软件工程实践的过程中，有着成功的经验，也有比成功更珍贵的失败的教训，这是构成注释内容的又一重要部分。

现在摆在读者面前的，就是我反复研读本书的过程中记录下来的点滴，若能在读者按图索骥时带来些许帮助，则可宽我心。

本书中有多种不同风格的注释内容，其中一些以小标题加以提示。本书的注释约定包括以下几类。

中文目录及每一章的中心思想

在原书目录之后给出了全书的中文目录，并提炼了每一章的中心思想，给出该章的整体线索，以便在读者阅读时有清晰的阅读思路。

核心理念

提炼出技术核心点和阅读关键点，对其进行详细说明和相关扩展，从而为读者在实际工作中遇到的问题提供更多的解决策略和更大的发挥空间。

术语解读

注释时从专业人员的角度对书中专业术语的出处、意义以及用法进行阐述和解释。

背景知识

书中内容往往涵盖多个业务领域，注释时对不太普及的领域知识以及书中提及的国内读者不熟悉的艺术隐喻、历史典故以及国外的谚语进行解释说明。

触类旁通

原书中的内容对读者的启迪，以及对当今技术发展的现实意义。

图中要点

书中使用了大量的类图和示意图，作者在制作图时力图体现上下文中提到的重点内容，注释时为了省却读者反复查询和思考的烦恼，点出作者在图中埋下的伏笔。

需求特性

由于国内与国外存在业务知识和文化方面的差异，书中经常会用一整段的话来铺垫某个要实现的功能点，而且会在不同的章节段落中反复提及。注释时将这些功能特性进行了归纳汇总，以方便读者理解和查找。

感谢我的父母和妻子，是你们让我长年安心在他乡工作，你们是我的最爱。感谢人民邮电出版社，给我机会和足够的时间来完成这次的注释。感谢我的朋友们，他们在工作和生活上给了我很多帮助和支持。我怀着忐忑的心情，写下这些文字，期待与读者有更多的机会交流。

孙向晖（网名“豆豆他爹”）

2004 年开始做面向对象的技术咨询服务
浪潮软件质保中心副经理
IBM RUP 认证专家及 SOA 认证架构师

FOREWORD

There are many things that make software development complex. But the heart of this complexity is the essential intricacy of the problem domain itself. If you're trying to add automation to complicated human enterprise, then your software cannot dodge this complexity—all it can do is control it.

The key to controlling complexity is a good domain model, a model that goes beyond a surface vision of a domain by introducing an underlying structure, which gives the software developers the leverage they need. A good domain model can be incredibly valuable, but it's not something that's easy to make. Few people can do it well, and it's very hard to teach.

Eric Evans is one of those few who can create domain models well. I discovered this by working with him—one of those wonderful times when you find a client who's more skilled than you are. Our collaboration was short but enormous fun. Since then we've stayed in touch, and I've watched this book gestate slowly.

It's been well worth the wait.

This book has evolved into one that satisfies a huge ambition: To describe and build a vocabulary about the very art of domain modeling. To provide a frame of reference through which we can explain this activity as well as teach this hard-to-learn skill. It's a book that's given me many new ideas as it has taken shape, and I'd be astonished if even old hands at conceptual modeling don't get a raft of new ideas from reading this book.

Eric also cements many of the things that we've learned over the years. First, in domain modeling, you shouldn't separate the concepts from the implementation. An effective domain modeler can not only use a whiteboard with an accountant, but also write Java with a programmer. Partly this is true because you cannot build a

此为试读,需要完整PDF请访问: www.ertongbook.com

useful conceptual model without considering implementation issues. But the primary reason why concepts and implementation belong together is this: The greatest value of a domain model is that it provides a *ubiquitous language* that ties domain experts and technologists together.

Another lesson you'll learn from this book is that domain models aren't first modeled and then implemented. Like many people, I've come to reject the phased thinking of "design, then build." But the lesson of Eric's experience is that the really powerful domain models evolve over time, and even the most experienced modelers find that they gain their best ideas after the initial releases of a system.

I think, and hope, that this will be an enormously influential book. One that will add structure and cohesion to a very slippery field while it teaches a lot of people how to use a valuable tool. Domain models can have big consequences in controlling software development—in whatever language or environment they are implemented.

One final yet important thought. One of things I most respect about this book is that Eric is not afraid to talk about the times when he *hasn't* been successful. Most authors like to maintain an air of disinterested omnipotence. Eric makes it clear that like most of us, he's tasted both success and disappointment. The important thing is that he can learn from both—and more important for us is that he can pass on his lessons.

Martin Fowler
April 2003

前　言

意识超前的软件人员将领域建模和设计作为重要课题已经有 20 多年的历史了，但奇怪的是，关于需要做什么和怎么做却很少见诸于文献。尽管领域建模和设计没有被明确地规范，但在对象领域中却暗潮涌动，出现了一种新的哲学体系，我称之为“领域驱动设计”（domain-driven design）。

在过去的 10 年中，我在多个业务和技术领域内开发复杂的系统。在工作中，我尝试了很多处于面向对象开发技术前沿的设计和开发过程中的最佳实践。一些项目非常成功，当然，也有些以失败告终了。成功的项目都有一个共同的特征：通过迭代的设计，一个完善的领域模型逐渐成为了项目的核心骨架。

本书提供了一个设计决策的框架，以及一组讨论领域设计时会用到的技术词汇表，集中了被广泛认同的最佳实践，以及我自己的见解和经验。需要面对复杂领域的软件开发团队可以借助本框架进行系统的领域驱动设计。

比较 3 个项目

在我的记忆中有 3 个项目能够生动地说明领域设计实践是如何戏剧性地影响了开发结果的。虽然这 3 个项目都交付了实用的软件，但只有一个达到了不凡的目标，并且产生了能够根据组织发展要求不断完善的复杂软件。

最初入手时，我注意到一个提交了简单实用的基于 Web 的贸易系统项目。开发者凭借着自己的感觉进行开发，但这并没有给他们带来什么阻碍，因为简单软件的编写很少需要注意到设计问题。开始的成功所造成的后果是，对未来开发的期望值会非常高，这时需要我开始第 2 个版本的研发。仔细研究这个项目，我发现缺少一个领域模型，甚至项目中连通用语言都没有，整个设计处于无结构的状况。但项目负责人不同意我的观点，所以我拒绝了这份工作。一年后，这个团队发现他们陷入了困境，无法交付第 2 个版本。虽然他们使用技术的方式没有问题，但最终还是（被他们轻视了的）业务逻辑击败了他们。第一个版本过早地固化，耗费了很高的维护成本。

要处理高难的复杂问题，需要对领域逻辑的设计采取更加认真的方法。在我早期的职业

生涯中，我很幸运地完成了一个强调领域设计的项目。这个项目的领域复杂度不亚于先前的那个项目，初期也成功地向贸易商交付了一个简单的应用系统。但在这个案例中，初始的交付版本后，伴随着的是成功的加速开发。每一个迭代都对前一次发布的功能整合和细化提出令人兴奋的新创意。团队能够通过灵活的扩展能力满足贸易商的要求。这种蓬勃向上的局面归功于深层的领域模型，它被不断地细化，并在编码中得到展现。当团队对领域有了新的理解后，模型也会随之深化。无论是开发人员之间，还是开发人员和领域专家乃至设计人员之间的沟通质量都得以完善。因为易于修改和扩展，项目的维护重担也随之减轻。

不幸的是，项目并不因为认真建模就会步入良性循环。我经历过的一个项目，项目人员狂热地想基于领域模型建立一个全球化的企业系统，但经历了若干的失败后，它不得不降低了期望值，沦落到与平常无二。这个团队有很好的工具，对业务有非常深的理解，对建模也格外重视。但蹩脚的开发角色划分使得建模和实现相分裂了，使得设计不能很好地反映深层分析。在任何情况下，详细的业务对象的设计并不能充分保证他们可以在精细的应用中结合。不断的迭代并不能提升编码质量，由于开发人员之间技能的参差不齐，他们没有意识去针对实用的、可运行的软件来建立基于模型的对象的体系风格和技术。时间一点点过去，开发工作陷入复杂的泥潭，团队失去了对系统的整体掌控。经过若干年的努力，项目虽然也产生了适当的可用软件，但团队放弃了早先关注建模的想法。

复杂度的挑战

很多原因可以导致一个项目偏离轨道：官僚主义（bureaucracy）、目标不明确、缺乏资源以及其他没有提到的原因。但设计能够很大程度上决定软件的复杂度。当复杂度难以掌控时，开发人员就不能很好地理解软件，更不用说轻松、安全地修改或者扩展它了。从另一方面讲，一个优先的设计会创造机会来完成这些复杂特性。

一些设计因素是技术层面的。在网络、数据库和软件的其他技术维度方面的设计上，已经取得了很大的进展。有许多书籍讲解了如何解决这些方面的问题。许多的开发人员跟随着技术的变革，提升了自己这方面的能力。

但许多应用最大的复杂性不是技术方面的，而在于领域本身，在于用户业务活动的复杂性。如果在设计中没有处理好领域复杂性，那么再好的基础设施技术也没有用。一个成功的设计必须系统地处理软件的这个核心方面。

本书的前提有两个：

- 对多数软件项目而言，主要焦点应该放在领域和领域逻辑上；
- 复杂的领域设计应该基于一个模型。

领域驱动设计既是一种思考方式也是一系列优先要考虑的事情，目的在于加速要处理复杂领域的软件项目。为达到这个目标，本书提供了大量的设计实践、技巧和原理。

设计与开发过程

在设计书籍和过程书籍中，它们之间很少互相提及，因为它们的每个主题本身都是很复杂的。这是一本设计方面的书籍，不过我认为设计和过程不应分开来。设计理念应被成功实现，否则，它们将只停留在学术讨论上。

当人们学习设计技巧时，会为各种可能性感到兴奋，接着他们遭遇到现实项目的杂乱状况。他们无法适应将新的设计思路渗透到必要的技术中，或者他们不知道何时应为特殊的设计问题而放弃，何时又应遵守设计原则，寻找一个正确的解决方案。开发人员之间确实相互讨论抽象的应用设计原理，但更实际的做法是讨论现实的项目如何完成。因此，尽管这是一本设计书籍，在必要时，我仍会穿插过程领域的知识。这有助于在适当的章节中讨论设计原理。

本书不依赖于某种特定的方法学，但它面向新的“敏捷开发过程”家族。特别是，它假定项目中有一系列的实践。有两个实践作为应用本书方法的两个前置条件：

1. 开发是迭代进行的。迭代开发已经被提倡并实践了几十年，它是敏捷开发方法的基石。关于敏捷开发和极限编程，有很多好的文献讨论，如《对象软件项目求生法则》（Cockburn，1998）和《解析极限编程》（Beck，1999）。

2. 开发者和领域专家保持密切合作关系。领域驱动设计需要研究大量的知识，形成一个反应深层领域见解和关注关键概念的领域模型。这是一项在了解领域和了解如何建立软件的人们之间建立的合作。因为开发是迭代的，这些合作必须贯穿于项目的整个生命周期。

被 Kent、Beck、Ward Cunningham 和其他人提倡的极限编程（见《解析极限编程》），是敏捷过程中最卓越的、也是我最常使用的开发过程。为了让解释更具体，我将在全书中使用 XP 作为设计和过程的交互讨论的基础。当然，列举的原理也会轻松适用于其他的敏捷过程。

最近几年，人们对精细开发方法学产生了质疑，认为无用的静态的文档以及强制的预先的计划和设计加重了项目的负担。而敏捷过程（如 XP）则强调应付变化和不确定性的能力。

极限编程承认设计决策的重要性，但它强烈反对预先设计。相反，它强调将大量精力投向交流沟通和提高项目对变更的快速应对的能力上。有了这样的反应能力，开发人员可以在项目的任何阶段使用“可运行的最简单的事情”，然后持续重构，进行许多小的设计改进，最终获得能够满足客户真实需要的设计。

这种行动纲领成为若干过度设计狂热者的强力解药。那些项目被笨重而无价值的文档压制着。由于团队成员担心产生不完美的设计，他们忍受着“分析瘫痪”之苦。这种情况必须得到改善。

不幸的是，很多这样的过程理念被曲解。每个人对“简单”有着不同的定义。持续重构是一系列小的重设计，缺乏实用设计原则的开发人员将编出难以理解或变更的代码，这与敏捷思想背道而驰。并且，尽管对不能预料的需求的担心会导致过度工程，但试图避免过度工程却会导致另一种担心，即不敢深入思考任何设计。

实践上，XP 最适合最具敏锐设计理念的开发人员。XP 过程假设你能通过重构完善设计，

并能经常性地快速重构。但先前的设计选择会使得重构本身变得更简单或者更困难。XP 过程试图增进团队的沟通，但模型和设计选择会让沟通更明了或者更混乱。

本书将设计和开发实践结合起来，并举例说明领域驱动设计和敏捷开发是如何相互促进的。在敏捷开发过程中，精细的领域建模方法会加速开发。与领域开发过程之间的相互关系，使得这种方法比其他任何空泛的“纯”的设计方法更实用。

本书的结构

本书共分为 4 个主要部分：

第 1 部分让领域模型发挥作用　解释了领域驱动开发的基本目标，这些目标引发了后续章节的实践。由于软件开发有很多的方法，第 1 部分定义了术语，并对使用领域模型驱动交流沟通和设计给出总体介绍。

第 2 部分模型驱动设计的构造块　将面向对象领域建模的一些核心最佳实践浓缩为一整套基础构造块。这部分关注于贯通模型和实践之间的鸿沟，让软件运行。通过共享这些标准的模式，使得设计更加有序。团队成员可以更轻松地相互了解彼此的工作。使用标准的模式也有助于形成所有团队成员可以用来讨论模型和设计决策的公共语言词汇。

但是，这部分的要点是关注保持模型和实现相互协调的各种类型的决策，它们之间相辅相成。这种协调需要注意单个元素的细节。在这种细小的规模上仔细工作，为开发人员采用第 3 和第 4 部分的建模方法提供了坚实的基础。

第 3 部分面向更深层理解的重构　超越了构造块的粒度，致力于将它们组装成可用的实际模型。这部分没有直接跳到深奥的设计原理，而是强调发现的过程。有价值的模型往往不会立刻浮现，需要对领域的深层次的理解。这种理解甚至有可能源于实现和改进了基于对幼稚的模型的初始设计。每次团队加深了理解，模型会被转换以体现更深层次的知识，代码也会被重构以反映更深层的模型，让它接近更潜在的应用。因此，一时的困难可能会引发一个更深层模型的突破，并伴随着更加完美的设计。

探索从不受任何限制，不过也不能随意进行。第 3 部分将深入研究能够指导决策方向的建模原理和有助于引导研究方向的技术。

第 4 部分战略设计　处理复杂系统、大型组织以及与外部系统和遗留系统交互时发生的各种情况。这个章节探索出统一应用到系统的 3 个原则：上下文环境、精练和大规模结构。战略设计决策由团队或在团队间制定。战略设计保证第 1 部分中的目标在更大范围上得以实现，适用于企业网范围的大型系统和应用。

本书中自始至终都采用真实项目中的真实案例进行讲解，绝非过于简单的玩具式的问题。

书中很多内容写成了一组“模式”的形式。读者可以不用关心这种形式就能理解这些内容，不过，如果哪位读者对模式的这种风格和格式感兴趣，可以阅读附录。

可以在网站 http://domaindrivendesign.org 找到补充资料，其中包括附加的示例代码和社区讨论。

谁应该阅读此书

本书主要针对面向对象软件开发人员。软件项目团队中的大多数成员可以从本书中的某些章节中受益。对于当前在一个项目中努力尝试做一些事情的人们，以及已经对类似项目拥有深厚经验的人们来说，阅读本书更有意义。

要从本书中受益，必须具备一些面向对象建模知识。案例中包括 UML 图和 java 代码，因此具备阅读这些语言的基本能力是非常必要的，但并不需要全部的细节。极限编程的知识能够开阔开发过程讨论的视角，但其内容应该为不具备这种背景知识的人们所理解。

对于中级软件开发人员，即那些已经了解了一些面向对象设计并可能已经阅读过一两本软件设计书籍的人而言，本书将介绍在一个软件项目中对象建模如何适应真实业务。本术将帮助中级开发人员学会如何应用精巧的建模和设计技能解决实际问题。

高级或者专业的软件开发人员会对书中用来处理领域的整体框架感兴趣。这个系统化的设计方法会帮助技术带头人带领他的团队沿着正确的方向前进。同样，书中统一的术语会帮助高级开发人员与同伴相互沟通。

本书采用的是叙述的形式，可以从头到尾阅读，也可以从任何一章开始读起。具有不同背景的读者可能会希望采用不同的阅读方式，但我建议所有的读者从第 1 部分的介绍以及第 1 章读起。之后，本书的核心部分是第 2 章、第 3 章、第 9 章和第 14 章。一些掌握了某些主题的读者可以借助阅读标题和粗体字来读取主要观点。更高级的读者会希望跳过第 1 部分和第 2 部分，第 3 部分和第 4 部分的内容或许更吸引他们。

除了上述主要读者群，分析人员和相关的项目技术经理也可通过阅读本书而受益。分析人员可以运用模型和设计之间的联系，在敏捷项目中获得更大的成效。分析人员也可以使用战略设计中的原则去更好地关注和组织他们的工作。

项目经理将重点关注如何让团队更有效率，他们更关注设计对业务专家和用户更有意义的软件。因为战略设计决策将团队组织结构和工作方式联系在一起，这些设计决策需要涉及项目领导能力，并对项目产生深远的影响。

领域驱动的团队

虽然理解领域驱动设计的个体开发人员能够获得有价值的设计技巧和观点，但是，只有当团队团结在一起使用领域驱动设计的方法并在项目中心上讨论领域模型时，才会产生更大的价值。这样的团队成员将共享一种能够促进他们沟通并将他们和软件联系在一起的语言。他们将会逐步根据模型产出明晰的实现，完成应用开发。他们将共享不同团队的设计工作是如何相互关联的大图，并系统地将注意力集中在对组织最有特色和最有价值的特性上。

在大多数的软件项目僵化死亡之际，领域驱动设计是一个困难的技术挑战，但它更能带来巨大的、开放的机会。

此为试读，需要完整PDF请访问：www.ertongbook.com