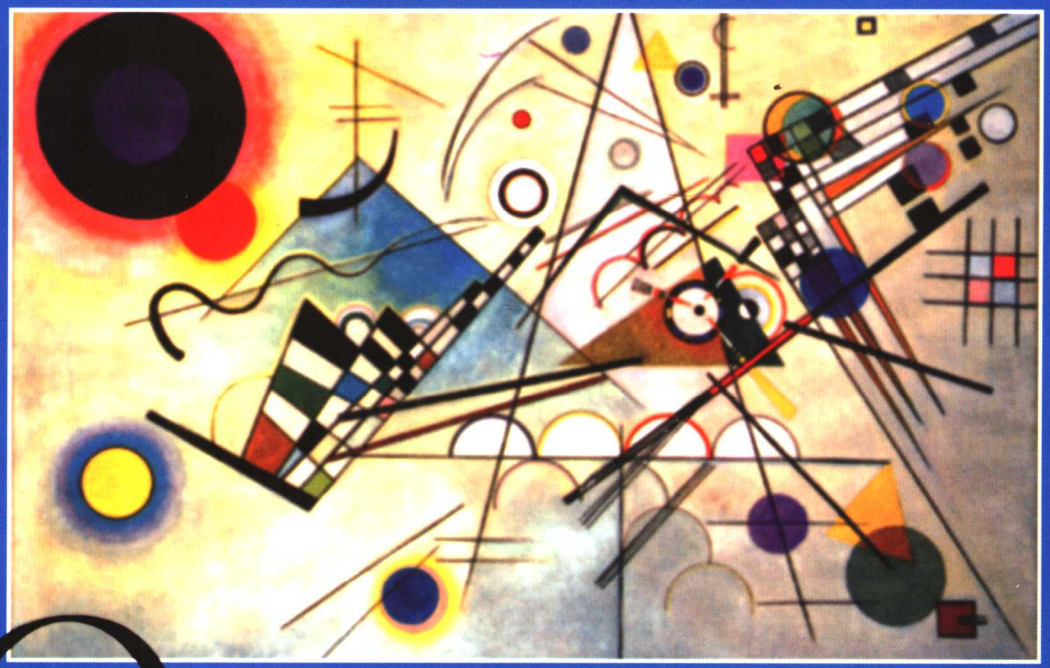
PEARSON
Addison Wesley

Domain-Driven Design

Tackling Complexity in the Heart of Software

领域驱动设计

——软件核心复杂性应对之道



(美) Eric Evans 著

陈大峰 张泽鑫 等译

UMLChina
特别推荐

PEARSON Education

清华大学出版社

领域驱动设计

——软件核心复杂性应对之道

(美) Eric Evans 著

陈大峰 张泽鑫 等译

清华大学出版社

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/(美)伊文斯(Evans，E.) 著；陈大峰，张泽鑫 等译.
—北京：清华大学出版社，2006.3
书名原文：Domain-Driven Design——Tackling Complexity in the Heart of Software
ISBN 7-302-11576-1

Ⅰ. 领… Ⅱ. ①伊… ②陈… ③张… Ⅲ. 软件设计 Ⅳ. TP311.5

中国版本图书馆 CIP 数据核字(2005)第 091124 号

出 版 者：清华大学出版社 **地 址**：北京清华大学学研大厦
http://www.tup.com.cn **邮 编**：100084
社 总 机：010-62770175 **客户服务**：010-62776969
组稿编辑：曹 康
文稿编辑：徐燕华
封面设计：康 博
版式设计：康 博
印 刷 者：清华大学印刷厂
装 订 者：三河市金元印装有限公司
发 行 者：新华书店总店北京发行所
开 本：185×230 **印张**：25.5 **字数**：496 千字
版 次：2006 年 3 月第 1 版 2006 年 3 月第1 次印刷
书 号：ISBN 7-302-11576-1/TP·7568
印 数：1～4000
定 价：48.00 元

译者序

在项目中担任过分析和设计工作的人，对于下面一些问题，一定会与译者一样深有同感：

- 概念混淆，术语混乱——在讨论时，经常发现不同的人把同一个词理解为不同的概念，导致沟通无法顺利进行；
- 设计似乎很难理解——开发人员无法很快抓住设计的重点，甚至会出现不同程度和方向的曲解；
- 代码也很难理解——阅读代码比编写代码更痛苦，即使它严格地遵循了缩进规则和命名规范；
- 当需求发生变化时，发现要对设计作大量修改——框架、模式似乎并未带来所需的灵活性；
- 当系统的复杂性达到相当程度时，整个项目似乎会无可避免地滑入“焦油坑”，或者为维护工作而疲于奔命。

如果您想思考为什么会出现这些问题，以及如何解决它们的话，请您深入地阅读本书。在阅读本书的过程中，您一定能够体会到一种豁然开朗的惊喜、一种跃跃欲试的冲动。

一旦您试着把书中的理念付诸实践，您就会发现，这些问题其实并不是不可捉摸的——您会觉得自己洞若观火，然后把它们手到擒来——这种经历曾经让译者欣喜若狂。例如，维护并严格遵循“通用语言”来处理第 1 个问题；对模型进行“精炼”、采用“大比例结构”等来解决第 2 个问题；“释意接口”、“声明性设计”等解决第 3 个问题，等等。

“领域驱动设计”认为，对于大多数软件项目而言，其根本着眼点应该集中于领域和领域逻辑；而复杂的领域设计应该是基于模型的。许多系统的真正复杂之处不在于技

术，而在于领域本身，在于业务用户及其执行的业务活动。如果在设计时没有获得对领域的深刻理解，没有通过模型将复杂的领域逻辑以模型概念和模型元素的形式清晰地表达出来，那么无论我们使用多么先进、多么流行的平台和设施，都难以保证项目的真正成功。

领域驱动设计能有效地加快软件项目的开发速度，并大大提高我们所能解决的问题的复杂度。它指导我们从混乱和复杂的领域中找出秩序和规律，抽象出一套通用语言，并通过恰当地运用一系列模式和策略来发挥通用语言的巨大作用。这是一个相当需要技巧和经验的过程。能够真正深入地理解、掌握和运用这些技巧和经验就已非常不易，而将这些技巧和经验总结和整理出来就显得尤为珍贵。

阅读本书将是学习这些技巧和经验的非常直接、快速、事半功倍的途径——这是我们在翻译过程中的深切感受。本书不仅为我们描述了领域驱动设计所需的技巧和经验，而且把它们组织整理成为了一种系统、清晰的知识体系，使我们能够循序渐进地学习和掌握领域驱动设计的精要所在。同时，丰富的、生动具体的实例使我们能够更轻松地领会领域驱动设计的各种原则和策略。读者一定能从本书中获取有用的知识和思想、掌握和运用领域驱动设计，并领导软件项目走向成功。

本书的翻译工作由陈大峰、张泽鑫等人共同完成，并由 UMLChina 的孙向晖、潘加宇负责技术审校。在此，译者对二位所做的工作表示衷心的感谢。

本书的翻译、编辑和审校工作前后历时两年多，可谓精心打造，希望其出版质量能够让广大读者满意。但由于我们水平有限，可能仍会有偏差甚至错误之处存在，恳请读者批评指正。信息反馈邮箱：fwhbook@tup.tsinghua.edu.cn。

译　者

2004 年 12 月

很多原因都会造成软件开发的复杂性，然而其核心则是由于问题领域本身的复杂性。如果要在复杂的企业中实现自动化，那么您的软件便不可能避开这种复杂性——它所能够做的仅仅是对复杂的问题进行控制。

控制复杂问题的关键是建立一个好的领域模型，它越过问题域的表象介绍其底层的结构，给软件开发人员提供所需要的方法。一个好的领域模型有非常重要的价值，但建立它却不是一件容易的事情。很少有人能够出色地完成，并且建立的方法也很难传授。

Eric Evans 是为数不多能够建立出色领域模型的人员之一，这是我在与他共事的时候发现的。我们的合作虽然短暂却非常有意思，从那时起我们开始保持着联系，我也仔细地研读了这本书。

本书是值得我们期待的！

本书的目的是描述并建立领域建模艺术的技术。使得我们在领域建模的时候能够参考其提供的框架，并且向读者讲授这个较难学习的技能。我在这本书中得到了很多新的思想和观念，我相信任何掌握传统概念模型的人从这本书中一定能够得到很多新的思想。

Eric 还巩固了我们已经掌握的很多知识。首先，在进行领域建模的时候，不能够将概念与实现分离。一个高效的领域建模人员不应该只会使用记事本和计算器，还要能够编写 Java 程序。一部分原因是离开对于实现问题的考虑，便无法建立一个有用的概念模型。然而概念与实现不可分割的主要原因是：领域模型最重要的价值在于提供一种通用的语言，将领域专家与技术人员联系在一起。

从本书中您还能够学到：领域模型并不是先建模然后接着实现的。和很多人一样，我开始拒绝接受“设计，然后建立”的定向思维。Eric 的经验告诉我们，真正有效的领域模型是随着时间慢慢发展得来的，即使最有经验的建模人员也会发现总是在系统的初

始版本实现后才会找到他们的最佳想法。

我认为，并且希望本书会是一本非常有影响力的书。在向人们讲述如何使用这个有价值的工具的同时增加了这个难于掌握的领域的结构性与内聚力。领域模型会对管理软件开发起到重大的影响——不管软件开发使用何种语言或环境实现。

最后还有一点非常重要的想法。从本书中，我看到 Eric 最值得我尊敬的一个方面是他敢于讨论还未取得成功的事情。大多数作者都想给读者留下一个无所不能的印象，而 Eric 却明白地告诉读者，他和我们中的大多数人一样，也曾有过成功和失败。重要的一点是，他能够从两者中汲取养分——对我们来说，更重要的是他会传播他所得到的经验。

Martin Fowler

前言

从领先的软件设计人员开始将领域建模及设计视为关键性课题到现在至少已经有20年的时间了，然而令人惊讶的是，几乎没有相关的文献来告诉大家应该做什么和如何做。尽管领域建模和设计并没有被明确地形式化，然而在对象领域中出现了一种潜在的哲学体系，它就是我所说的领域驱动设计(domain-driven design)。

我花费10年时间开发了几个商业和技术领域的复杂系统。在工作过程中，我尝试了几种已经出现在面向对象开发前沿领域的设计与开发程序。这些项目中有一些非常成功，也有少数几个最终失败。成功项目的共同特征是在迭代的设计中不断地完善领域模型并将它作为项目的骨干结构的一部分。

本书提供了进行设计决策的框架和讨论领域设计时使用的技术，集中了被广泛接受的优秀实践，这些案例都是在我自己的领域与工作中的经验积累。需要面对复杂领域的软件开发团队可以使用这个框架来系统地进行领域驱动设计。

比较3个项目

在我的记忆中，有3个项目能够作为说明动态领域建模设计如何影响开发结果的生动示例。尽管这3个项目都交付了实用的软件，然而只有一个达到了优秀的目标，并且生成了能够根据组织不断发展的要求进行持续完善的复杂软件。

我注意到一个非常迅速地提交了简单实用的基于Web的贸易系统的项目。开发人员们任凭自己的感觉进行开发，但这种态度并没有对他们的工作造成阻碍，因为一个简单软件的编写并不需要注意设计问题。初始成功的结果是，对于未来继续开发的要求极高。

这时我被要求进入第 2 版本的开发工作。仔细地研究了这个项目后，我发现他们缺少一个领域模型，甚至连项目通用的语言都没有，整个设计处于无结构状态。项目的领导者不同意我的论断，于是我拒绝了这个工作。一年后，这个开发团队陷入困境，无法交付第 2 个版本。尽管他们对技术的使用方式并无什么错误，从业务逻辑来看，我们还是应该克服这种情况。他们的第 1 个版本过早地固化导致了高额的维护代价。

处理这种高度复杂的问题要求对领域逻辑的设计采用更加认真的方法。在我事业的早期，非常幸运地能够完成一项格外重视领域设计的项目。该项目的复杂性不小于前面提到的第一个任务，也同样是一开始向贸易商提交了一个简单的应用软件，适度地完成初始工作。但是这次情况有所不同，初始交付的版本不断地加速开发。每一次迭代都会对前一个版本功能的整合与细化提出令人兴奋的新意见。开发团队能够灵活地进行扩展以反馈贸易商的要求。这种向上的良好发展轨迹直接归功于一个明确的领域模型，迭代地改进并快速地编码。随着团队对领域更进一步的洞察，模型也随之进一步深化。开发人员之间甚至开发人员与领域专家之间的交流得以改善，项目的设计也不像以前那样带来艰巨的维护任务，而变得易于修改和扩展了。

遗憾的是，项目并不会仅仅因为认真进行建模而进入一个良性循环。我过去接触过一个项目，开始时基于领域模型是要建立一个全球企业系统，但是经过几年屡屡受挫，不得不降低目标而落入俗套。这个团队有良好的工具与对业务的深入理解，并且对于建模也格外重视。然而拙劣的开发角色划分使得模型与实现相互分离，因此设计并没有反映出分析的深度。在任何情况下，详细的业务对象设计并不是保证它们在复杂精细项目中完美结合的充分条件。再三的迭代并不能够提高编码质量，因为开发人员之间技术水平不均衡，而他们又不了解面向实际的运行软件而建立基于模型的对象的技术与其本身的风格。时间一点点过去，开发工作陷入复杂的泥潭，团队也丧失了对系统的整体把握。经过几年的工作，该项目并没有生产出有用的软件，该团队却不得不放弃其初始时建模的目标。

复杂度的难题

很多原因会导致一个项目偏离正确轨道：官僚主义、不明确的目标、资源的匮乏以及其他诸多因素。但是设计能够在很大程度上决定软件的复杂度。当复杂度变得难以控制时，开发人员便不再能够很好地理解软件，从而也不能够方便和安全地对它进行改变与扩展。另一方面，一个优秀的设计会为开发这些复杂特性带来机会。

一些设计因素是技术上的，在网络、数据库和其他软件技术方面的设计已经有了很多研究，这些问题的解决方法也有许多书籍专门论述。开发人员们不断提升自己的技能并紧跟着每一次的技术进步。

然而许多应用程序最大的复杂性不在技术方面，而是在领域本身，用户的活动或业务方面。如果在设计中没有处理好领域复杂性，那么基础设施技术再好也不管用。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：

- 对于多数软件项目，主要的焦点应该在领域及领域逻辑方面。
- 复杂的领域设计应该基于一个模型来进行。

领域驱动的设计既是一种思考的方式也是一系列的要优先考虑的事情，目的在于加速处理复杂领域的软件项目。为了达到这个目标，本书提供了大量的设计实践、技巧和原则。

设计与开发过程

设计方面的书籍与过程方面的书籍很少相互提及，其中每个主题本身就是一个很复杂的问题。本书是一本关于设计方面的著作，但是我认为设计与过程是不可分割的。设计理念必须得以成功实现，否则它们将仅仅止步于学术讨论。

在人们学习设计技术时，常常会为各种可能性感到兴奋不已。接着他们会遇到实际项目的杂乱现状，他们无法将新的设计思路应用在必须使用的技术上。或者是他们不知道何时应该抛开设计方面的局限，何时又应该严格地遵循设计，寻找一个正确的解决方案。开发人员们相互讨论抽象的应用程序设计原则，但更为实际的是讨论现实的项目如何完成。因此，尽管这是一本设计书籍，在需要时我仍然会涉及到过程领域的知识。这样做更有助于在合适的上下文中讨论设计原则。

本书并不局限于某一特定的方法学，然而它是面向新的“敏捷开发过程”系列的。它假设项目实践中有几个约定俗成的惯例。下面两个惯例是本书所采用方法的先决条件。

- **开发工作是迭代的。**迭代开发过程已经被提倡并实践了几十年，它是敏捷开发方法的基础。关于敏捷开发和极限编程(或 XP)的文献有很多讨论，如 *Surviving Object-Oriented Projects*(Cockburn 1998)和 *Extreme Programming Explained*(Beck 1999)。
- **开发人员与领域专家之间关系密切。**领域驱动设计需要大量深入的领域知识以及对于其中关键概念的关注。这是一项了解领域与了解如何建立软件的人们之间的合作。因为开发工作是迭代进行的，因此这种合作要始终贯穿于项目的生命周期。

Kent Beck、Ward Cunningham 和其他人认为(参见 *Extreme Programming Explained* [Beck 2000])最后的编程实现是敏捷开发过程中最重要的，也是我最经常需要处理的工作。在本书中，为了更加具体地阐述，我使用 XP 作为设计与过程交互讨论的基础。举例所用的原则可以方便地适用于其他的敏捷开发过程。

最近几年，人们开始质疑精细开发方法学，认为它们产生了无用的、静态的文档以及强制性的前置计划和设计。相反，敏捷开发过程，例如 XP，则强调对变化和不确定性的应对。

极限编程承认设计决策的重要性，但是它极力反对前置设计。它花费大量精力进行交流和提高项目的迅速转向能力。有了这样的反应能力，开发人员可以在项目的每个阶段使用“可运行的最简单事物”，然后进行持续重构，进行许多小的设计改善，最后完成符合用户需求的设计。

这种极保守行为对于一些过度的设计狂热者是一种非常必要的解药。那些使得项目举步维艰的大量文档并没有什么价值。由于开发团队害怕所做的设计不完善，使这些项目深受“分析瘫痪”之苦。这种情况必须要有所改变。

遗憾的是，这些开发过程的观念常常被曲解。每个人对于“简单”都有不同的定义。持续的重构是一系列小型的再设计；那些缺乏一致的设计原则的开发人员将编写出难以理解或修改的代码——而这恰恰与敏捷背道而驰。尽管由于对各种无法预料的需求的担心常常导致过度工程(overengineering)，然而试图避免过度工程却可能导致另一种担心：不敢进行任何深入的设计思考。

实际上，XP 最适合具有敏锐设计感知的开发人员。XP 过程假设您能够通过重构完善一个设计，并且您会频繁而迅速地进行重构。但是前面的设计选择会使得重构本身变得更容易或更困难。XP 过程尝试增加团队交流，然而不同的模型和设计选择会使交流变得明确或混淆。

本书将设计与开发实践结合在一起，并且举例说明领域驱动设计与敏捷开发如何相互补充。在敏捷开发过程环境中精细的领域建模方法能够加速开发。与领域开发过程之间的相互关系使得这种方法比其他凭空考虑的“纯”设计方法更加实际。

本书结构

本书分为 4 个主要部分：

第 I 部分：让领域模型发挥作用。介绍了领域驱动开发的基本目标；这些目标推动后面几章的实践。由于软件开发有很多方法，第 1 章定义了一些术语并说明了对使用领域模型驱动交流和设计的总的看法。

第II部分：模型驱动设计的构建块。将面向对象领域建模实践的核心浓缩为一些基本的构建块。这部分着力于在模型与实际之间搭建桥梁并运行软件。这些标准模式的共享使得设计得以有序进行，团队成员更加容易了解彼此的工作。使用标准模式更有助于使用通用语言的术语，这样所有的团队成员便可以使用它们来讨论模型和设计决策。

但是本部分主要关注的是保持模型与实现之间相互协调以及提高效率的决策种类。这种协调需要注意到单个元素的细节。在这种小规模上的工作为开发人员采用第III和第IV部分的建模方法提供了稳定的平台。

第III部分：面向更深层理解的重构。超越了构建块范围而着力于将它们装配成可见结果的实际模型。本部分并没有直接介绍深奥的设计原则，而是着重讨论发现过程。有价值的模型通常不会立刻产生，它们需要对领域进行深入理解。基于幼稚的模型实现初始的设计，然后一次又一次地改变它。团队每次增加对领域的理解后，模型将被改变以反映进一步获得的知识，代码也随之重构来反映更深层次的模型，使其更接近可用的应用程序。因此，偶尔的困难可能正是突破到一个更加深入的模型，获得更完美设计的机会。

探索与生俱来就是可扩展的，然而它并不是随机的。第III部分主要研究能够在探索过程中指导决策的原则和帮助引导研究的技术。

第IV部分：战略性设计。处理复杂系统、较大型组织以及外部系统与老式系统交互中发生的各种情况。这部分提出了作为整体应用于系统的 3 个原则：上下文、精炼和大比例结构。战略性设计决策由团队制定，它们使得第 I 部分的目标大体实现，可能是适合企业范围网络的大型系统或应用程序。

本书自始至终都使用来自实际项目的真实例子进行论述，而不是采用过于简单的“玩具”问题。

补充的材料可以通过站点 http://domaindrivendesign.org 找到，其中包括附加的示例代码和相关讨论。

本书读者对象

本书主要针对面向对象的软件开发人员。一个软件项目团队的大多数成员都能够从本书的某些部分中得到收获。对于现在设计一个项目并尝试着做一些工作的人们，以及对于这些项目有很多经验的人们来说，本书都有很重要的意义。

从本书中会得到一些面向对象建模的知识，例如 UML 图和 Java 代码，因此阅读这些语言的基本能力是很重要的，但是并不需要掌握它们的细节问题。极限编程的知识能够增加开发过程讨论的角度，但是其内容应该能被不具备这些知识的人们所理解。

对于中级软件开发人员——那些已经了解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读者——本书将介绍在一个软件项目中对象建模如何适应现实生活。本书将帮助中级开发人员学会在实际问题中采用精妙的建模和设计技巧。

高级或专门的软件开发人员会对书中涉及领域的全面框架感兴趣。系统的设计方法会帮助项目领导者带领团队沿着正确的方向工作。本书始终使用的相关术语对于高级开发人员与同伴交流也很有帮助。

本书采用叙述形式，可以从头到尾也可以从任何一章开始阅读。具有不同背景的读者会希望采用不同的方法阅读本书，但是我建议所有的读者从第 I 部分的介绍以及第 1 章开始。除了这些，核心内容大部分集中在第 2 章、第 3 章、第 9 章和第 14 章。已经掌握了一些主题的读者可以通过阅读标题和粗体字来得到内容的要点。已有基础的读者可能希望跳过第 I 部分和第 II 部分，第III和第IV部分的内容可能会让他们更感兴趣。

除了这些主要的读者群外，相关的技术项目经理及分析人员也可通过阅读此书而受益。分析人员运用模型与设计的联系，获得在敏捷项目环境中更有效的成果。他们也可使用战略性设计的原则去更好地专注和组织他们的工作。

项目经理关注如何使一个团队更有效率以及如何设计对于业务专家和用户更有意义的软件。因为战略性设计决策与团队结构和工作方式有关，这些设计决策必定会涉及到项目领导者并对项目的发展产生重要的影响。

领域驱动开发团队

尽管一个理解领域驱动设计的单独开发人员能够获得有价值的设计技术和观点，然而当团队团结在一起使用领域驱动设计方法并在项目中心会谈上讨论领域模型时会得到最大的收获。通过这样做，团队成员能够进行比较多的交流，并保持这种交流与软件相关联。他们会依据模型逐步产生明晰的实现产品，得到应用程序开发的方法。他们将共享不同团队的设计工作如何相互关联的示意图，并系统地将注意力聚焦于对于团体最有价值和有特色的特征上。

当大多数的软件项目开始在僵化中死去时，领域驱动设计是一个困难的技术挑战，但它可以带来巨大的、开放的机会。

目 录

第Ⅰ部分 让领域模型发挥作用

第 1 章 消化知识 ····· 5
1.1 有效建模的因素 ····· 9
1.2 知识消化 ····· 10
1.3 持续学习 ····· 11
1.4 知识丰富的设计 ····· 12
1.5 深层模型 ····· 15

第 2 章 交流及语言的使用 ····· 17
2.1 通用语言 ····· 17
2.2 利用对话改进模型 ····· 22
2.3 一个团队，一种语言 ····· 24
2.4 文档和图 ····· 25
2.4.1 书面的设计文档 ····· 27
2.4.2 执行的基础 ····· 29
2.5 说明性模型 ····· 29

第 3 章 将模型和实现绑定 ····· 32
3.1 模型驱动设计 ····· 33
3.2 建模范型和工具支持 ····· 36
3.3 突出主旨：为什么模型对用户很关键 ····· 41
3.4 实践型建模人员 ····· 43

第Ⅱ部分 模型驱动设计的构建块

第 4 章 分离领域 ····· 47
4.1 分层架构 ····· 47
4.1.1 层间的联系 ····· 51
4.1.2 架构框架 ····· 51
4.2 模型属于领域层 ····· 52
4.3 其他种类的隔离 ····· 55

第 5 章 软件中的模型描述 ····· 56
5.1 关联 ····· 57
5.2 实体(又称引用对象) ····· 62
5.2.1 实体建模 ····· 65
5.2.2 设计标识操作 ····· 66
5.3 值对象 ····· 68
5.3.1 设计值对象 ····· 71

5.3.2 设计包含值对象的关联……73
5.4 服务……74
5.4.1 服务和分隔的领域层……75
5.4.2 粒度……77
5.4.3 访问服务……77
5.5 模块(包)……77
5.5.1 敏捷的模块……79
5.5.2 基础结构驱动打包的缺陷…80
5.6 建模范式……82
5.6.1 对象范式的优势……82
5.6.2 对象世界中的非对象……84
5.6.3 在混合范式中使用模型驱动设计……85

第 6 章 领域对象的生命周期……87
6.1 聚合……88
6.2 工厂……96
6.2.1 工厂及其应用场所的选择…99
6.2.2 只需构造函数的情况……101
6.2.3 接口的设计……102
6.2.4 如何放置不变量的逻辑……103
6.2.5 实体工厂与值对象工厂……103
6.2.6 存储对象的重建……103
6.3 仓储……105
6.3.1 查询仓储……109
6.3.2 了解仓储实现的必要性……111
6.3.3 实现仓储……111
6.3.4 在框架内工作……113
6.3.5 与工厂的关系……113
6.4 为关系数据库设计对象……115

第 7 章 使用语言：扩展示例……117
7.1 货物运输系统概述……117
7.2 隔离领域：系统简介……119
7.3 区分实体和值对象……120
7.4 运输领域中的关联设计……121
7.5 聚合的边界……123
7.6 选择仓储……124
7.7 场景概述……125
7.7.1 应用特性示例：改变一件货物的目的地……126
7.7.2 应用特性示例：重复业务·126
7.8 对象的创建……126
7.8.1 Cargo 的工厂和构造函数…126
7.8.2 添加一个 Handling Event…127
7.9 停下来重构：Cargo 聚合的另一种设计……129
7.10 运输模型中的模块……131
7.11 引入新特性：配额检查……133
7.11.1 连接两个系统……134
7.11.2 改进模型：划分业务……135
7.11.3 性能调整……137
7.12 小结……137

第III部分 面向更深层理解的重构

第 8 章 突破……143
8.1 关于突破的故事……144
8.1.1 中看不中用的模型……144
8.1.2 突破……146
8.1.3 更深层的模型……148
8.1.4 冷静的决定……149
8.1.5 成效……150
8.2 时机……150

8.3 着眼于根本 ……151
8.4 尾声：一连串的新理解 ……151

第 9 章 隐含概念转变为显式概念 ……153
9.1 概念挖掘 ……153
9.1.1 倾听表达用语 ……154
9.1.2 检查不协调之处 ……157
9.1.3 研究矛盾之处 ……162
9.1.4 查阅书籍 ……162
9.1.5 尝试，再尝试 ……164
9.2 如何建模不太明显的概念 ……164
9.2.1 显式的约束 ……165
9.2.2 作为领域对象的流程 ……167
9.2.3 规格 ……168
9.2.4 规格的应用和实现 ……171

第 10 章 柔性设计 ……184
10.1 释意接口 ……186
10.2 无副作用函数 ……190
10.3 断言 ……194
10.4 概念轮廓 ……197
10.5 孤立类 ……201
10.6 操作封闭 ……203
10.7 声明性设计 ……205
10.8 一个声明性风格的设计 ……207
10.9 攻击角度 ……215
10.9.1 切分子领域 ……215
10.9.2 尽可能利用现成的形式 ……216

第 11 章 应用分析模式 ……225

第 12 章 把设计模式和模型联系起来 ……237
12.1 策略 ……238
12.2 组合 ……241
12.3 为什么不用 Flyweight？ ……245

第 13 章 向更深层理解重构 ……247
13.1 发起重构 ……247
13.2 探索团队 ……248
13.3 前期工作 ……249
13.4 针对开发人员设计 ……249
13.5 时机选择 ……250
13.6 将危机视为机会 ……250

第Ⅳ部分 战略性设计

第 14 章 维护模型完整性 ……255
14.1 限界上下文 ……257
14.2 持续集成 ……261
14.3 上下文映射 ……263
14.3.1 在上下文边界上的测试 ……269
14.3.2 组织和文档化上下文映射 ……269
14.4 限界上下文之间的关系 ……270
14.5 共享内核 ……271
14.6 顾客/供应商开发团队 ……272
14.7 同流者 ……275
14.8 防腐层 ……277
14.8.1 设计防腐层的接口 ……279
14.8.2 实现防腐层 ……279
14.8.3 一个关于警戒的故事 ……282
14.9 隔离方式 ……282
14.10 开放主机服务 ……284
14.11 公布语言 ……284
14.12 盲人摸象 ……287
14.13 选择模型上下文的策略 ……290
14.13.1 团队或更高层的决策 ……290

14.13.2 把自己放在上下文中…291
14.13.3 转换边界……291
14.13.4 接受我们不能改变的东西：描绘外部系统…292
14.13.5 与外部系统的关系…292
14.13.6 在设计系统……293
14.13.7 满足不同模型的特别需要……293
14.13.8 部署……294
14.13.9 权衡……295
14.13.10 考虑项目已经进行的情况……295
14.14 转换……296
14.14.1 合并上下文：隔离方式→共享内核……296
14.14.2 合并上下文：共享内核→持续集成……297
14.14.3 逐步淘汰原有系统…298
14.14.4 开放主机服务→公布语言……299

第 15 章 精炼……301
15.1 核心领域……302
15.1.1 选择核心……305
15.1.2 谁来负责精炼工作……305
15.2 精炼的逐步升级……306
15.3 通用子域……307
15.3.1 通用不一定可重用……312
15.3.2 项目风险管理……313
15.4 领域愿景声明……313
15.5 突出核心……315
15.5.1 精炼文档……316
15.5.2 把核心标记出来……317
15.5.3 把精炼文档作为开发过程的工具……318
15.6 内聚机制……319
15.6.1 通用子域与内聚机制…320
15.6.2 属于核心领域的机制…321
15.7 精炼到声明性风格……321
15.8 隔离核心……322
15.8.1 创建隔离核心的代价…323
15.8.2 推进团队决策……323
15.9 抽象核心……328
15.10 深层模型精炼……329
15.11 选择重构的目标……329

第 16 章 大比例结构……330
16.1 渐进顺序……333
16.2 系统隐喻……335
16.3 职责层……337
16.4 知识级别……349
16.5 插件框架……356
16.6 结构的约束……360
16.7 重构到合适的结构……361
16.7.1 最小化……362
16.7.2 交流和自律……362
16.7.3 结构重组产生柔性设计…362
16.7.4 精炼为开发指路……363

第 17 章 综合应用战略性设计……364
17.1 大比例结构和限界上下文的结合……364
17.2 大比例结构和精炼的结合…367
17.3 首先进行评估……369
17.4 由谁制定策略……369
17.4.1 在开发过程中自发产生…369