



HZ BOOKS

PEARSON

经典重读

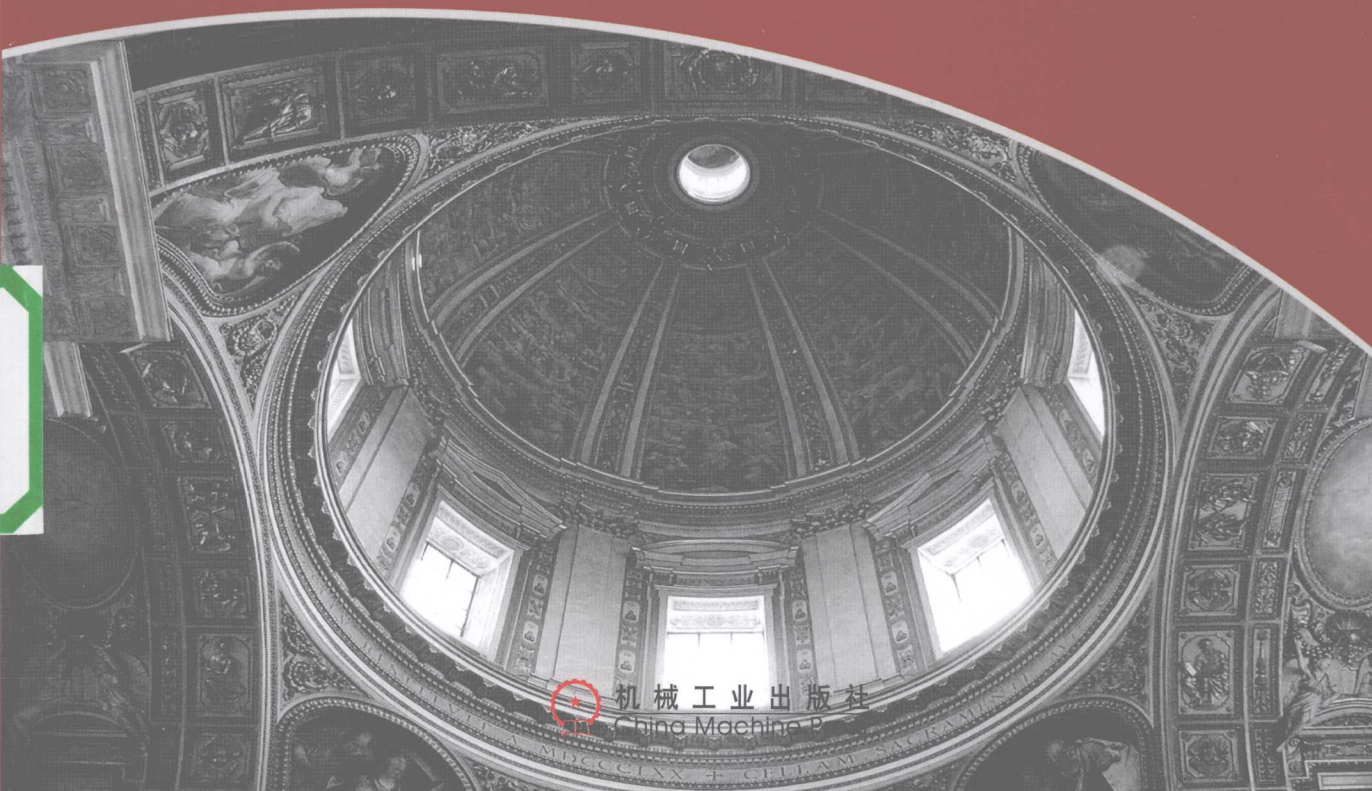
*Patterns of Enterprise Application Architecture*

# 企业应用架构模式

(英) Martin Fowler 著

王怀民 周斌 译

UMLChina 审校



机械工业出版社

China Machine Press

经典重读

*Patterns of Enterprise Application Architecture*

# 企业应用架构模式

(英) Martin Fowler 著

王怀民 周斌 译

UMLChina 审校



机械工业出版社  
China Machine Press

本书作者是当今面向对象软件开发的权威，他在一组专家级合作者的帮助下，将40多种经常出现的解决方案转化成模式，最终写成这本能够应用于任何一种企业应用平台的、关于解决方案的、不可或缺的手册。本书获得了2003年度美国软件开发杂志图书类的生产效率奖和读者选择奖。本书分为两大部分。第一部分是关于如何开发企业应用的简单介绍。第二部分是本书的主体，是关于模式的详细参考手册，每个模式都给出使用方法和实现信息，并配以详细的Java代码或C#代码示例。此外，整本书中还用了大量UML图来进一步阐明有关概念。

本书是为致力于设计和构建企业应用的软件架构师、设计人员和编程人员而写的，同时也可作为高等院校计算机专业及软件学院相关课程的参考教材。

Simplified Chinese edition copyright © 2010 by Pearson Education Asia Limited and China Machine Press.

Original English language title: *Patterns of Enterprise Application Architecture* (ISBN 978-0-321-12742-6) by Martin Fowler, Copyright © 2003.

All rights reserved.

Published by arrangement with the original publisher, Pearson Education, Inc., publishing as Addison-Wesley.

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。

封底无防伪标均为盗版

版权所有，侵权必究

本书法律顾问 北京市展达律师事务所

本书版权登记号：图字：01-2010-2049

图书在版编目（CIP）数据

企业应用架构模式 / (英) 福勒 (Fowler, M.) 著；王怀民，周斌译. —北京：机械工业出版社，2010.4 (经典重读)

书名原文：Patterns of Enterprise Application Architecture

ISBN 978-7-111-30393-0

I. 企… II. ①福… ②王… ③周… III. 软件工具—程序设计—应用—企业管理 IV. F270.7

中国版本图书馆CIP数据核字 (2010) 第064108号

机械工业出版社 (北京市西城区百万庄大街22号 邮政编码 100037)

责任编辑：迟振春

北京京北印刷有限公司印刷

2010年4月第1版第1次印刷

186mm × 240mm · 23.75印张

标准书号：ISBN 978-7-111-30393-0

定价：59.00元

凡购本书，如有缺页、倒页、脱页，由本社发行部调换

客服热线：(010) 88378991；88361066

购书热线：(010) 68326294；88379649；68995259

投稿热线：(010) 88379604

读者信箱：hzsj@hzbook.com

# 译者序

“每一个模式描述了一个在我们周围不断重复发生的问题，以及该问题的解决方案的核心。这样，你就能一次又一次地使用该方案而不必做重复劳动。”

——Christopher Alexander

本书是面向对象大师Martin Fowler继《Analysis Patterns》、《UML Distilled》、《Planning Extreme Programming》、《Refactoring》之后的又一力作。

“温故而知新”。Fowler在本书中再次向我们证明了《礼记》中这句古训的震撼力——他在回头审视自己及同仁多年来从事企业应用开发的经验和教训后，归纳总结了40多种企业应用架构的设计模式。这些模式从不同层次、不同侧面向我们展示了什么是好的企业应用架构？如何设计好的企业应用？

正如作者自己所言，企业应用在某些方面比其他软件（如电信通信软件）复杂得多：纷繁复杂的企业数据、“不合逻辑”的业务规则、变化莫测的用户需求，等等。环顾四周——CORBA、J2EE、.NET——企业应用开发技术可谓“前仆后继、层出不穷”，开发平台的种类之多就更不必说。

招式套路可以千变万化，扎实深厚的“内功”却是始终如一！虽然企业应用涉及的软件技术不断翻新，但是基本的架构及设计思想却没有太多变化。将以前行之有效的设计思路和方法加以适当调整，并应用到当前的问题上，是最高效的做法。在一组专家级合作者的帮助下，Martin将40多种经常出现的解决方案转化成模式，最终融会成这本“内功心法”。在仔细研读、用心揣摩本书之后，希望它能够帮助你应对任何一种企业应用平台，驾驭任何一种企业应用技术——无论是现在的技术还是未来的技术。

熟悉Fowler的读者都知道，这位大师的写作风格可谓是“深入浅出，娓娓道来”。本书也是一样。前8章是关于企业应用的背景知识，如分层架构、Web表现、业务逻辑、数据库映射、并发、会话、分布策略，等等。在此基础上，随后的各章分别对与这些背景知识相关的设计模式进行了详细的介绍。与其他设计模式的书一样，本书从模式的使用场景、解决方案、UML表示等方面予以介绍，详略有致。就连示例的编程语言的选取——Java和C#——也是与他的写作风格一脉相承的。

夜已深，窗外依旧是绵绵不断的早春小雨。让我们酌一杯清茶，一起来品味大师的话，一起来品味“源于实践、指导实践”的苦涩与甘甜——

“模式的关键点是它们源于实践。必须观察人们的工作过程，发现其中好的设计，并找出‘这些解决方案的核心’。这不是一个简单的过程，但是一旦发现了某个模式，它将是非常有价值的。对于我来说，价值之一是能够撰写这样一本参考书。你不必通读本书的全部内容，也不必通读任何一本有关模式的书。只需要了解到这些模式都是干什么的、它们解决什么问题、它们是如何解决问题的，就足够了。这样，一旦你碰到类似问题，就可以从书中找出相应的模式。

那时，你再深入了解相应的模式也为时不晚。”

	知	见	闻	不
	之	之	之	闻
首	不	不	不	不
子	若	若	若	若
	行	知	见	闻
	之	之	之	之

本书翻译初稿的过程中得到了丁博、王树凤、朱锐、林繁、托明福的大力帮助。王怀民、周斌分别统审了全书。非常感谢UMLChina潘加宇、蒋芳在校对过程中的宝贵意见和建议。

译者

2004年3月27日于长沙

# 前言

1999年春天，我飞抵芝加哥为ThoughtWorks公司正在开发的一个项目担任顾问。ThoughtWorks是一个规模虽小但正在快速成长的应用开发公司。这个项目属于那种极富挑战性的企业级应用，它是一个后端租赁系统。简单说，它处理的是租户签字认可后所有与租赁有关的事务，包括发送账单、处理某些人对所租房屋资产的改造、追踪那些未按时缴纳账单的租户、指出如果某人提前归还资产应当如何处理等。在你意识到租赁合同极度复杂并且总在不断变化之前，这听起来好像并不太难实现。它的业务“逻辑”几乎不能套用任何已有的逻辑模式，因为那些“逻辑”归根到底是商人们为争夺生意而制定的，一些古怪的小改动都可能对赢得某笔交易起关键作用。因此，每次生意上的一点点胜利就意味着系统复杂性的又一次增加。

我对此类问题情有独钟：如何捕获这些复杂性，并设计一个面向对象的系统来处理它们。事实上，我一直坚信面向对象的最大优点在于它能够使复杂逻辑易于处理。为复杂业务逻辑开发一个好的领域模型很困难，但在此问题中却是恰得其所。

当然，单纯的领域模型也并非灵丹妙药。我们的领域模型必须持久化到数据库中，但是与许多项目一样，我们当时使用的是关系数据库。我们还必须将该模型与用户界面关联起来，还要能支持远程应用程序对本软件的使用，同时还必须将我们的软件与第三方的软件包整合。这些工作都基于一种被称为J2EE的新技术，当时全世界没有人在这一方面有实战经验。

虽然J2EE是一项崭新的技术，但我们仍能从以往的经验中得到帮助。本人曾经长期使用C++、Smalltalk和CORBA来开发类似系统。ThoughtWorkers公司中许多人则有Forte方面的经验。可以说当时我们对关键架构已经有了思路，所要解决的仅是如何将之转化为J2EE实现。现在回过头来看三年前的设计，虽然它并不尽善尽美，但无疑经受住了时间的考验。

上述情况正是我撰写这本书的出发点。多年来，我曾看到过许多企业级应用项目。这些项目通常都包含相似的设计思路，这些设计思路已经被证明可以有效地处理企业应用中不可避免的复杂性。本书就是将这些设计思路升华为模式的一个起点。

本书分为两个部分，第一部分是一些叙述性的章节，它们主要讨论企业级应用程序设计中的一些重要议题。这些章节介绍了企业级应用程序架构的各种问题，并给出了大体的解决方案。而解决方案的细节则在本书的第二部分以模式的方式组织成文。这些模式仅仅是一些参考，我并不希望读者一页页地去细读。我的想法是：从头到尾将第一部分的叙述性章节读完，然后再根据兴趣和需求翻阅第二部分的有关章节。因此，本书是将简短的叙述和详尽的参考合二为一。

本书讨论的是企业级应用程序的设计。企业级应用程序涉及大量复杂数据的显示、操纵和存储以及对这些数据进行处理的业务流程的自动化。典型的例子有预订系统、金融系统、物流补给系统以及其他种种驱动现代商业运作的系统。企业级应用和嵌入式系统、控制系统、电信系统或者桌面应用程序不同，它们有自己特有的挑战和解决方案。因此，如果你在上述那些非企业领域工作，本书对你并没有益处（除非想体会一下企业级应用程序是怎么一回事）。如果需

要一本关于软件架构的通用性的书籍，我推荐[POSA]<sup>⊖</sup>。

在构建企业级应用时有许多架构方面的问题。我想本书恐怕很难一一详细列举。在软件开发方面，本人是迭代开发方法的忠实信徒。迭代开发的核心在于只要软件对用户有用，就应当交付，即使这个软件当时并没有完成。虽然著书与编写软件之间存在诸多差异，但我认为在这一点上异曲同工。也就是说，虽然本书尚不全面，但已初具雏形，可以为读者提供有关企业级应用程序架构方面的有益建议（至少我认为如此）。本书讨论的主要议题是：

- 企业级应用程序的分层
- 构建领域（业务）逻辑
- 构建基于Web的用户界面
- 将内存模块（尤其是对象）关联到关系数据库
- 在无状态环境下处理会话状态
- 分布原则

可能本书未涉及的议题更多。我很想撰写关于组织确认、合并消息和异步通信、安全、错误处理、集群、应用集成、架构重构、构建胖客户的用户界面等方面的书籍或文章。但是，由于时空限制以及思路尚未成熟，本书将不涉及上述内容。我只能希望在不久的将来能看到一些与这些工作相关的模式。也许我会撰写本书的第2卷并加入这些内容，或者是由其他人来补遗。

当然，基于消息的通信是相当重要的问题。在进行多应用程序集成时，人们正越来越多地用到异步的、基于消息的通信方法。即便是在同一应用程序内部，基于消息的通信也值得费上一些笔墨。

本书并非针对某一特定的软件平台。从20世纪80年代末到90年代初，我开始在基于Smalltalk、C++和CORBA的项目中使用这些模式。而20世纪90年代后期我在Java方面做了大量的工作，我发现这些模式可以很好地应用于较早的Java/CORBA系统和其后基于J2EE的工作中。近来，我开始在微软的.NET平台方面做一些探索，我发现这些模式同样有效。ThoughtWorkers公司的同事也介绍了他们的经验，尤其是在Forte方面。我不敢说这些模式能够通用于所有已经和即将被用于企业级应用的开发平台，但至少到目前为止，它们已经表现出足够的可重用性。

对于大多数模式，本书提供了相应的代码示例。这些例子所用的程序设计语言是我认为大多数读者都能够阅读和理解的语言。Java就是一个很好的选择。只要熟悉C或C++就可以读懂Java代码，Java远没有C++那么复杂。基本上，大多数C++程序员都能够理解Java，但反过来却并非如此。我是面向对象的信徒，也就自然更偏爱面向对象的语言。因此，大多数代码示例使用的是Java语言。写这本书时，微软的.NET环境正逐渐成熟，它的C#语言与Java有许多相同之处。所以某些代码示例也使用了C#语言，虽然这样做会多少有一些风险，因为.NET尚未得到大量应用，使用的术语可能尚未形成惯例。这两种语言都是基于C的语言，只要能读懂其中一种，即使对另一种语言或平台并不熟悉，要读懂它也并非难事。我的目的是使用一种能够让最多的软件开发者读懂的语言，即使这种语言并非他们所擅长或偏爱的。（谨向那些喜欢Smalltalk、Delphi、Visual Basic、Perl、Python、Ruby、COBOL或其他语言的读者致歉。我知道你们认为

---

<sup>⊖</sup> 本书的中文版（中译名为“面向模式的软件体系结构”）已由机械工业出版社出版。——编辑注



有比Java或C#更好的语言，我也认为如此！)

示例是用来阐述和解释模式思想的。它们并非可以直接使用的解决方案，任何情况下都需要做一些工作才能将它们用于你的应用程序之中。模式只是一个有益的起点，而非最终的解决之道。

## 本书的读者

本书面向的是正在构建企业级应用、希望增进对架构相关问题的理解和沟通的编程人员、设计人员和软件架构师。

我假定本书的大多数读者可以归为两类：一些人所面对的需求并非大规模的，因此准备自己从零开始构建软件；另一些人则有大规模的需求，将使用某些工具。对于前者，我认为本书中提及的模式将有助于工作的启动。尽管在许多领域，所需的知识远超出本书中模式所给予的内容，但本书提供了一个比我当初进入该领域时更高的起点。对于那些工具的使用者，我希望本书能揭开工具的一些内幕，帮助他们决策如何选择工具所支持的模式中的哪一种。例如，使用对象-关系映射工具同时意味着你必须决定如何映射某些特定情况。本书所提供的模式将有助于你做出这样的决策。

当然，还可能第三类读者，他们有大规模的需求，同时又希望亲自构建软件。在此，我首先要忠告他们的是：请先考虑使用已有的工具。我已经见过不止一个项目花费了大量时间来建造框架，而这些框架并非该项目所真正要解决的问题。如果你仍固执己见，那也只能听天由命。但要记住，本书中代码示例为提高可理解性都被有意简化过，在实际使用时，往往需要对它们大动干戈才能满足要求。

由于模式是可复现问题的通用解决方案，因此可能有的读者对这些模式已经有所接触。如果你从事企业级应用开发已经有一段时间，可能会很熟悉其中大部分模式。本书中并不包含任何新的东西，正相反：这是一本关于（我们这一行业的）已有知识的书。如果你是这一领域的新手，我希望本书将帮助你学习这些技术。如果你熟悉这些技术，我希望本书有助于你与其他人沟通。模式的重要作用就在于其创建了一个通用的词汇表，例如，你称某个类是远程外观，其他设计人员就都知道你指的是什么。

## 致谢

与其他书籍一样，本节将涉及多年来以不同方式与我一起工作的很多人。他们以许多方式对本书提供过帮助。本书中一些重要内容是由他人提供的，其中某些人我可能已经无法回忆起他们的名字。在这里，我所能做的是对那些仍铭记在心的人表示感谢。

首先要感谢的是我的合作者，David Rice，他是我在ThoughtWorks公司的同事，他为本书做出了巨大的贡献：撰写了本书的1/10。当我们一起努力以保证本书能如期交付时（他当时还要从事客户支持的工作），我们曾在若干个深夜通过即时消息进行协商，交谈中他坦承他总算明白了为何写一本书是如此困难却又如此吸引人。

Matt Foemmel是ThoughtWorks公司的另一个同事。尽管他文笔犀利冷峻，是本书的一个十分中肯的批评家，但他为代码示例做出了很大的贡献。Randy Stafford为本书贡献了服务层模式，



他一直是此模式的极力倡导者。我还要感谢Edward Heatt和Rob Mee所做的贡献，特别是Rob在复审本书时所发现的缺失。Rob是我的最佳审阅者：他不仅发现少了某些内容，而且他还帮我写了一节来弥补这个缺失！

同样，对本书一流的正式审阅者，我的言辞远不能表达我的感激之情：

John Brewer	Rob Mee
Kyle Brown	Gerard Meszarios
Jens Coldewey	Dirk Riehle
John Crupi	Randy Stafford
Leonard Fenster	David Siegel
Alan Knight	Kai Yu

我几乎要把ThoughtWorks公司的电话号码簿列在此处了，因为太多同事与我讨论过他们的设计和经历，在这一项目上帮助过我。许多模式在我脑中成型，是因为我有机会与众多天才设计师讨论，因此我只好对整个公司表示感谢。

Kyle Brown、Rachel Reinitz和Bobby Woolf在百忙之中抽出时间与我一道在北卡罗来纳对本书进行了长期而细致的审阅。他们在本书中注入了他们睿智的光芒。尤其是与Kyle的几次长时间的电话交谈令我获益匪浅。

2000年初我与Alan Knight和Kai Yu一起为Java One大会准备了一个演讲，这是本书最初的雏形。在对他们所提供的帮助致谢的同时，我还要感谢Josh Mackenzie、Rebecca Parsons和Dave Rice，他们其后协助我提炼了这些演讲及其思想。Jim Newkrik付出了很大努力协助我熟悉.NET平台。

我与这个领域的许多专家有过令人惬意的交谈或合作，从而在他们身上学到了不少东西。尤其想对Colleen Roe、David Muirhead和Randy Stafford表示感谢，他们将自己在Gemstone的Foodsmart示例系统上的工作成果与我共享。我在Bruce Eckel所主持的Crested Butte讨论会上也参与过一些重要的会谈，因此应当向近年来这一会议的与会者致谢。Joshua Kerievsky虽然没有时间对本书做一次全面的审阅，但他是模式方面的一个优秀顾问。

我还从UIUC阅读组那里获得了相当大的帮助，他们作为读者对本书提出了坦诚的批评。我要感谢：Ariel Gertzenstein、Bosko Zivaljevic、Brad Jones、Brian Foote、Brian Marick、Federico Balaguer、Joseph Yoder、John Brant、Mike Hewner、Ralph Johnson和 Weerasak Witthawaskul。

前UIUC成员Dragos Manolescu及其小组给了我一些反馈。感谢Muhammad Anan、Brian Doyle、Emad Ghosheh、Glenn Graessele、Daniel Hein、Prabhakaran Kumarakulasingam、Joe Quint、John Reinke、Kevin Reynolds、Sripriya Srinivasan和 Tirumala Vaddiraju。

Kent Back为我提供了许多思路，尤其是他为特殊情况模式所起的名字。Jim Odell将我领入了顾问咨询、教学和写作的世界，我由衷地感谢他。

当我写这本书时，曾将草稿放在Web上。期间许多人通过电子邮件向我指出问题、提出疑问或者讨论其他替代方案。他们中有Michael Banks、Mark Bernstein、Graham Berrisford、Bjorn Beskow、Bryan Boreham、Sean Broadley、Peris Brodsky、Paul Campbell、Chester Chen、John Coakley、Bob Corrick、Pascal Costanza、Andy Czerwonka、Martin Diehl、Daniel Drasin、Juan

Gomez Duaso, Don Dwiggins, Peter Foreman, Russell Freeman, Peter Gassmann, Jason Gorman, Dan Green, Lars Gregori, Rick Hansen, Tobin Harris, Russel Healey, Christian Heller, Richard Henderson, Kyle Hermenean, Carsten Heyl, Akira Hirasawa, Eric Kaun, Kirk Knoernschild, Jesper Ladegaard, Chris Lopez, Paolo Marino, Jeremy Miller, Ivan Mitrovic, Thomas Neumann, Judy Obee, Paolo Parovel, Trevor Pinkney, Tomas Restrepo, Joel Rieder, Matthew Roberts, Stefan Roock, Ken Rosha, Andy Schneider, Alexandre Semenov, Stan Silvert, Geoff Soutter, Volker Termath, Christopher Thames, Volker Turau, Knut Wannheden, Marc Wallace, Stefan Wenig, Brad Wiemerslage, Mark Windholtz, Michael Yoon.

此外，还有许多我不认识或已经遗忘了的人，在此要向他们表达同样真诚的谢意。最诚挚的谢意依旧要献给我的妻子Cindy，她与我同历风雨，我将永远铭刻在心。

# 模式列表

**Active Record (活动记录)**: 包装数据库表或视图中的行、封装数据库访问、并将业务逻辑加到该数据上的对象。

**Application Controller (应用控制器)**: 处理屏幕导航和应用流程的集中控制点。

**Association Table Mapping (关联表映射)**: 将关联存储为表形式, 并通过外键将与之相关的表连接起来。

**Class Table Inheritance (类表继承)**: 表示了类的继承层次, 每个类都对应一个表。

**Client Session State (客户会话状态)**: 将会话状态存储于客户端。

**Coarse-Grained Lock (粗粒度锁)**: 用一个锁将一组相关对象都锁起来。

**Concrete Table Inheritance (具体表继承)**: 表示了类的继承层次, 层次中的每一个具体类对应一个表。

**Data Mapper (数据映射器)**: 由映射器组成的一个层次, 它在对象和数据库之间移动数据, 并保证它们之间相互独立、映射器间相互独立。

**Data Transfer Object (数据传输对象)**: 在进程间进行数据传递的对象, 目的是减少进程间方法调用的次数。

**Database Session State (数据库会话状态)**: 将会话状态作为提交的数据存储于数据库中。

**Dependent Mapping (依赖映射)**: 让一个类为一个子类进行数据库映射。

**Domain Model (领域模型)**: 既包含行为, 又包含数据的领域的对象模型。

**Embedded Value (嵌入值)**: 将一个对象映射到另一个对象的表的多个域中。

**Foreign Key Mapping (外键映射)**: 将对象间的一个关联映射成表间一个外键引用。

**Front Controller (前端控制器)**: 处理Web站点中所有请求的控制器。

**Gateway (入口)**: 封装对外部系统或资源进行访问的对象。

**Identity Field (标识域)**: 将数据库ID域保存到对象中, 用来维护一个内存对象与一个数据库行之间的对应关系。

**Identity Map (标识映射)**: 通过将每个被加载对象放入一个映射中, 确保每个对象都被加载且仅加载一次。当引用对象时使用映射来查找对象。

**Implicit Lock (隐含锁)**: 允许框架或层超类型申请离线锁。

**Inheritance Mappers (继承映射器)**: 组织数据库映射器的一种结构, 能处理继承层次。

**Layer Supertype (层超类型)**: 某一层中充当所有类型的超类型的类型。

**Lazy Load (延迟加载)**: 对象中并不包含所有想要的数据库, 但是知道如何获取这些数据。

**Mapper (映射器)**: 在两个独立对象间建立起通信的对象。

**Metadata Mapping (元数据映射)**: 在元数据中保存对象-关系映射的细节。

**Model View Controller (模型-视图-控制器)**: 将用户界面交互分割成三个不同的角色。

**Money (货币)**: 表示货币值。

**Optimistic Offline Lock (乐观离线锁)**: 通过检测冲突并回滚事务来防止并发业务事务间的冲突。

**Page Controller (页控制器)**: 处理Web站点中特定页面请求或动作的对象。

**Pessimistic Offline Lock (悲观离线锁)**: 通过每次只允许一个业务事务访问数据来防止并发业务事务间的冲突。

**Plugin (插件)**: 在配置阶段而不是在编译阶段连接类。

**Query Object (查询对象)**: 表示数据库查询的对象。

**Record Set (记录集)**: 列表数据的内存表示。

**Registry (注册表)**: 其他对象可以通过其找到公共对象或服务的一个知名对象。

**Remote Façade (远程外观)**: 在细粒度对象上提供粗粒度的外观, 以改进跨网络的有效性。

**Repository (资源库)**: 通过用来访问领域对象的一个类似集合的接口, 在领域与数据映射层之间进行协调。

**Row Data Gateway (行数据入口)**: 在一个数据源中充当到一条记录的入口的对象。每条记录(数据库行)对应一个对象实例。

**Separated Interface (分离接口)**: 在一个与其实现分离的包中定义一个接口。

**Serialized LOB (序列化LOB)**: 通过序列化方法将一张对象图保存在一个大对象(LOB)中, 然后将大对象存储在数据库的域中。

**Server Session State (服务器会话状态)**: 将会话状态以序列化方式保存在一个服务器系统上。

**Service Layer (服务层)**: 通过一个服务层来定义应用的边界, 这个服务层建立了一组可用操作并协调应用程序对每个操作请求的响应。

**Service Stub (服务桩)**: 在测试过程中去除对有问题服务的依赖。

**Single Table Inheritance (单表继承)**: 通过一个表来表示类的一个继承层次, 表中各列对应不同类中的所有域。

**Special Case (特殊情况)**: 为特定情况提供特殊行为的一个子类。

**Table Data Gateway (表数据入口)**: 充当到数据库表的入口的对象。一个实例处理表中的所有行。

**Table Module (表模块)**: 通过一个实例处理数据库表或视图中所有行的业务逻辑。

**Template View (模板视图)**: 通过在HTML页面中嵌入标记来将有关信息融入HTML。

**Transaction Script (事务脚本)**: 通过过程来组织业务逻辑。其中, 每个过程处理来自表现层的一个请求。

**Transform View (转换视图)**: 逐条处理领域数据并将它们转换成HTML的一个视图。

**Two Step View (两步视图)**: 通过两步将领域数据转换成HTML: 首先形成某种逻辑页面, 然后将逻辑页面转换成HTML。

**Unit of Work (工作单元)**: 维护由业务事务所影响的对象列表, 并协调更改的写出以及并发问题的解决。

**Value Object (值对象)**: 类似于金钱或数据范围的、小而简单的对象, 其相等性不是基于ID的。

# 目 录

译者序		
前言		
模式列表		
引言	1	
0.1 架构	1	
0.2 企业应用	2	
0.3 企业应用的种类	3	
0.4 关于性能的考虑	4	
0.5 模式	6	
0.5.1 模式的结构	7	
0.5.2 模式的局限性	9	
<b>第一部分 表 述</b>		
第1章 分层	12	
1.1 企业应用中层次的演化	13	
1.2 三个基本层次	14	
1.3 为各层选择运行环境	15	
第2章 组织领域逻辑	19	
2.1 抉择	22	
2.2 服务层	23	
第3章 映射到关系数据库	25	
3.1 架构模式	25	
3.2 行为问题	28	
3.3 读取数据	29	
3.4 结构映射模式	30	
3.4.1 关系的映射	30	
3.4.2 继承	33	
3.5 建立映射	34	
3.6 使用元数据	35	
3.7 数据库连接	36	
3.8 其他问题	38	
3.9 进一步阅读	38	
第4章 Web表现层	39	
4.1 视图模式	41	
4.2 输入控制器模式	43	
4.3 进一步阅读	43	
第5章 并发	45	
5.1 并发问题	45	
5.2 执行语境	46	
5.3 隔离与不变性	47	
5.4 乐观并发控制和悲观并发控制	48	
5.4.1 避免不一致读	49	
5.4.2 死锁	49	
5.5 事务	50	
5.5.1 ACID	51	
5.5.2 事务资源	51	
5.5.3 减少事务隔离以提高灵活性	52	
5.5.4 业务事务和系统事务	53	
5.6 离线并发控制的模式	54	
5.7 应用服务器并发	55	
5.8 进一步阅读	56	
第6章 会话状态	57	
6.1 无状态的价值	57	
6.2 会话状态	58	
6.3 存储会话状态的方法	59	
第7章 分布策略	61	
7.1 分布对象的诱惑	61	
7.2 远程接口和本地接口	62	
7.3 必须使用分布的情况	63	
7.4 关于分布边界	64	
7.5 分布接口	64	
第8章 通盘考虑	67	
8.1 从领域层开始	67	
8.2 深入到数据源层	68	

8.2.1 事务脚本的数据源 .....	68	10.1.5 例: 使用ADO.NET数据集 (C#) .....	104
8.2.2 表模块的数据源 .....	69	10.2 行数据入口 (Row Data Gateway) .....	106
8.2.3 领域模型的数据源 .....	69	10.2.1 运行机制 .....	107
8.3 表现层 .....	69	10.2.2 使用时机 .....	108
8.4 一些关于具体技术的建议 .....	70	10.2.3 例: 人员记录 (Java) .....	108
8.4.1 Java和J2EE .....	70	10.2.4 例: 领域对象的数据保持器 (Java) .....	111
8.4.2 .NET .....	71	10.3 活动记录 (Active Record) .....	112
8.4.3 存储过程 .....	71	10.3.1 运行机制 .....	112
8.4.4 Web Services .....	72	10.3.2 使用时机 .....	113
8.5 其他分层方式 .....	72	10.3.3 例: 一个简单的Person类 (Java) .....	113
<b>第二部分 模 式</b>		10.4 数据映射器 (Data Mapper) .....	115
<b>第9章 领域逻辑模式</b> .....	76	10.4.1 运行机制 .....	116
9.1 事务脚本 (Transaction Script) .....	76	10.4.2 使用时机 .....	119
9.1.1 运行机制 .....	76	10.4.3 例: 一个简单的数据映射器 (Java) .....	119
9.1.2 使用时机 .....	77	10.4.4 例: 分离查找方法 (Java) .....	123
9.1.3 收入确认问题 .....	78	10.4.5 例: 创建一个空对象 (Java) .....	126
9.1.4 例: 收入确认 (Java) .....	78	<b>第11章 对象-关系行为模式</b> .....	129
9.2 领域模型 (Domain Model) .....	81	11.1 工作单元 (Unit of Work) .....	129
9.2.1 运行机制 .....	81	11.1.1 运行机制 .....	129
9.2.2 使用时机 .....	83	11.1.2 使用时机 .....	133
9.2.3 进一步阅读 .....	83	11.1.3 例: 使用对象注册的工作单元 (Java) .....	134
9.2.4 例: 收入确认 (Java) .....	84	11.2 标识映射 (Identity Map) .....	137
9.3 表模块 (Table Module) .....	87	11.2.1 运行机制 .....	137
9.3.1 运行机制 .....	88	11.2.2 使用时机 .....	139
9.3.2 使用时机 .....	90	11.2.3 例: 标识映射中的方法 (Java) .....	139
9.3.3 例: 基于表模块的收入确认 (C#) .....	90	11.3 延迟加载 (Lazy Load) .....	140
9.4 服务层 (Service Layer) .....	93	11.3.1 运作机制 .....	140
9.4.1 运行机制 .....	94	11.3.2 使用时机 .....	142
9.4.2 使用时机 .....	96	11.3.3 例: 延迟初始化 (Java) .....	142
9.4.3 进一步阅读 .....	96	11.3.4 例: 虚代理 (Java) .....	142
9.4.4 例: 收入确认 (Java) .....	96	11.3.5 例: 使用值保持器 (Java) .....	144
<b>第10章 数据源架构模式</b> .....	101	11.3.6 例: 使用重影 (C#) .....	144
10.1 表数据入口 (Table Data Gateway) .....	101	<b>第12章 对象-关系结构模式</b> .....	151
10.1.1 运行机制 .....	101	12.1 标识域 (Identity Field) .....	151
10.1.2 使用时机 .....	102	12.1.1 工作机制 .....	151
10.1.3 进一步阅读 .....	102	12.1.2 使用时机 .....	154
10.1.4 例: 人员入口 (C#) .....	103	12.1.3 进一步阅读 .....	154

12.1.4 例：整型键 (C#) .....	154	12.8.4 例：运动员和他们的家属 (C#) ...	203
12.1.5 例：使用键表 (Java) .....	155	12.9 具体表继承 (Concrete Table Inheritance) ...	208
12.1.6 例：使用组合键 (Java) .....	157	12.9.1 运行机制 .....	209
12.2 外键映射 (Foreign Key Mapping) .....	166	12.9.2 使用时机 .....	210
12.2.1 运行机制 .....	167	12.9.3 例：具体运动员 (C#) .....	210
12.2.2 使用时机 .....	169	12.10 继承映射器 (Inheritance Mappers) ...	214
12.2.3 例：单值引用 (Java) .....	169	12.10.1 运行机制 .....	215
12.2.4 例：多表查询 (Java) .....	172	12.10.2 使用时机 .....	216
12.2.5 例：引用集合 (C#) .....	173	<b>第13章 对象－关系元数据映射模式</b> .....	217
12.3 关联表映射 (Association Table Mapping) .....	175	13.1 元数据映射 (Metadata Mapping) .....	217
12.3.1 运行机制 .....	176	13.1.1 运行机制 .....	217
12.3.2 使用时机 .....	176	13.1.2 使用时机 .....	218
12.3.3 例：雇员和技能 (C#) .....	177	13.1.3 例：使用元数据和反射 (Java) .....	219
12.3.4 例：使用直接的SQL (Java) .....	179	13.2 查询对象 (Query Object) .....	224
12.3.5 例：用一次查询查多个雇员 (Java) ...	182	13.2.1 运行机制 .....	225
12.4 依赖映射 (Dependent Mapping) .....	186	13.2.2 使用时机 .....	225
12.4.1 运行机制 .....	186	13.2.3 进一步阅读 .....	226
12.4.2 使用时机 .....	187	13.2.4 例：简单的查询对象 (Java) .....	226
12.4.3 例：唱片和曲目 (Java) .....	188	13.3 资源库 (Repository) .....	228
12.5 嵌入值 (Embedded Value) .....	190	13.3.1 运行机制 .....	229
12.5.1 运行机制 .....	190	13.3.2 使用时机 .....	230
12.5.2 使用时机 .....	190	13.3.3 进一步阅读 .....	231
12.5.3 进一步阅读 .....	191	13.3.4 例：查找一个人所在的部门 (Java) .....	231
12.5.4 例：简单值对象 (Java) .....	191	13.3.5 例：资源库交换策略 (Java) .....	231
12.6 序列化LOB (Serialized LOB) .....	192	<b>第14章 Web表现模式</b> .....	233
12.6.1 运行机制 .....	193	14.1 模型－视图－控制器 (Model View Controller) .....	233
12.6.2 使用时机 .....	194	14.1.1 运行机制 .....	233
12.6.3 例：在XML中序列化一个部门层级 (Java) .....	194	14.1.2 使用时机 .....	234
12.7 单表继承 (Single Table Inheritance) ...	196	14.2 页面控制器 (Page Controller) .....	235
12.7.1 运行机制 .....	197	14.2.1 运行机制 .....	235
12.7.2 使用时机 .....	197	14.2.2 使用时机 .....	236
12.7.3 例：运动员的单表 (C#) .....	198	14.2.3 例：Servlet控制器和JSP视图的简单演示 (Java) .....	236
12.7.4 从数据库中加载对象 .....	199	14.2.4 例：使用JSP充当处理程序 (Java) ...	238
12.8 类表继承 (Class Table Inheritance) ...	202	14.2.5 例：代码隐藏的页面控制器 (C#) ...	241
12.8.1 运行机制 .....	202	14.3 前端控制器 (Front Controller) .....	243
12.8.2 使用时机 .....	203		
12.8.3 进一步阅读 .....	203		



14.3.1 运行机制 .....	244	<b>第16章 离线并发模式</b> .....	295
14.3.2 使用时机 .....	245	16.1 乐观离线锁 (Optimistic Offline Lock) ...	295
14.3.3 进一步阅读 .....	246	16.1.1 运行机制 .....	296
14.3.4 例: 简单的显示 (Java) .....	246	16.1.2 使用时机 .....	298
14.4 模板视图 (Template View) .....	248	16.1.3 例: 领域层与数据映射器 (Java) ...	298
14.4.1 运行机制 .....	249	16.2 悲观离线锁 (Pessimistic Offline Lock) ...	302
14.4.2 使用时机 .....	251	16.2.1 运行机制 .....	303
14.4.3 例: 分离的控制器, 使用JSP 充当视图 (Java) .....	252	16.2.2 使用时机 .....	305
14.4.4 例: ASP.NET服务器页面 (C#) ...	253	16.2.3 例: 简单锁管理对象 (Java) .....	305
14.5 转换视图 (Transform View) .....	256	16.3 粗粒度锁 (Coarse-Grained Lock) .....	310
14.5.1 运行机制 .....	256	16.3.1 运行机制 .....	310
14.5.2 使用时机 .....	257	16.3.2 使用时机 .....	312
14.5.3 例: 简单的转换 (Java) .....	257	16.3.3 例: 共享的乐观离线锁 (Java) .....	312
14.6 两步视图 (Two Step View) .....	259	16.3.4 例: 共享的悲观离线锁 (Java) .....	316
14.6.1 运行机制 .....	259	16.3.5 例: 根对象乐观离线锁 (Java) .....	317
14.6.2 使用时机 .....	260	16.4 隐含锁 (Implicit Lock) .....	318
14.6.3 例: 两阶XSLT (XSLT) .....	264	16.4.1 运行机制 .....	318
14.6.4 例: JSP和定制标记 (Java) .....	266	16.4.2 使用时机 .....	319
14.7 应用控制器 (Application Controller) ...	269	16.4.3 例: 隐含的悲观离线锁 (Java) .....	319
14.7.1 运行机制 .....	270	<b>第17章 会话状态模式</b> .....	321
14.7.2 使用时机 .....	271	17.1 客户会话状态 (Client Session State) ...	321
14.7.3 进一步阅读 .....	271	17.1.1 运行机制 .....	321
14.7.4 例: 状态模型应用控制器 (Java) ...	271	17.1.2 使用时机 .....	322
<b>第15章 分布模式</b> .....	275	17.2 服务器会话状态 (Server Session State) ...	322
15.1 远程外观 (Remote Facade) .....	275	17.2.1 运行机制 .....	322
15.1.1 运行机制 .....	276	17.2.2 使用时机 .....	324
15.1.2 使用时机 .....	278	17.3 数据库会话状态 (Database Session State) ...	324
15.1.3 例: 使用Java语言的会话 bean来作为远程外观 (Java) .....	278	17.3.1 运行机制 .....	324
15.1.4 例: Web Service (C#) .....	281	17.3.2 使用时机 .....	325
15.2 数据传输对象 (Data Transfer Object) ...	285	<b>第18章 基本模式</b> .....	327
15.2.1 运行机制 .....	285	18.1 入口 (Gateway) .....	327
15.2.2 使用时机 .....	288	18.1.1 运行机制 .....	327
15.2.3 进一步阅读 .....	289	18.1.2 使用时机 .....	328
15.2.4 例: 传输唱片信息 (Java) .....	289	18.1.3 例: 私有消息服务的入口 (Java) ...	329
15.2.5 例: 使用XML实现序列化 (Java) .....	293	18.2 映射器 (Mapper) .....	331
		18.2.1 运行机制 .....	332
		18.2.2 使用时机 .....	332
		18.3 层超类型 (Layer Supertype) .....	332

18.3.1 运行机制 .....	332	18.8 特殊情况 (Special Case) .....	346
18.3.2 使用时机 .....	333	18.8.1 运行机制 .....	347
18.3.3 例: 领域对象 (Java) .....	333	18.8.2 使用时机 .....	347
18.4 分离接口 (Separated Interface) .....	333	18.8.3 进一步阅读 .....	347
18.4.1 运行机制 .....	334	18.8.4 例: 一个简单的空对象 (C#) .....	347
18.4.2 使用时机 .....	335	18.9 插件 (Plugin) .....	348
18.5 注册表 (Registry) .....	335	18.9.1 运行机制 .....	349
18.5.1 运行机制 .....	336	18.9.2 使用时机 .....	350
18.5.2 使用时机 .....	337	18.9.3 例: ID生成器 (Java) .....	350
18.5.3 例: 单子注册表 (Java) .....	337	18.10 服务桩 (Service Stub) .....	352
18.5.4 例: 线程安全的注册表 (Java) .....	338	18.10.1 运行机制 .....	352
18.6 值对象 (Value Object) .....	339	18.10.2 使用时机 .....	353
18.6.1 运行机制 .....	339	18.10.3 例: 销售税服务 (Java) .....	353
18.6.2 使用时机 .....	340	18.11 记录集 (Record Set) .....	355
18.7 货币 (Money) .....	340	18.11.1 运行机制 .....	355
18.7.1 运行机制 .....	341	18.11.2 使用时机 .....	356
18.7.2 使用时机 .....	342	<b>参考文献</b> .....	<b>359</b>
18.7.3 例: 货币类 (Java) .....	343		