
DDD Community
Broadview
www.broadview.com.cn
PEARSON
实现
IMPLEMENTING
DOMAIN-DRIVEN DESIGN
领域驱动设计
Vaughn Vernon 著
滕云 译
张逸 审
· 涵盖DDD各个方面
· 大量示例代码
· 案例研究贯穿全书
· 理论和实践紧密结合
· 程序员进阶佳作
DDD之父Eric Evans作序
電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
http://www.phei.com.cn

Broadview
www.broadview.com.cn
博文视点·IT出版旗舰品牌
技术凝聚实力·专业创新出版

“对于那些希望提升自己技能的软件开发者来说，《实现领域驱动设计》将是一本绝佳的好书。”

——Randy Stafford，自由架构师，Oracle Coherence产品部

“对于那些希望实际应用DDD的人来说，这是一本必读之作。”

——Udi Dahan，NServiceBus创始人

《实现领域驱动设计》采用一种自顶向下的方式向我们讲述了DDD的战略设计模式和战术编程工具，并使这两者之间自然地衔接起来。Vaughn Vernon向我们展示了如何将DDD实现应用于现代的软件架构，并且强调业务领域的重要性和价值，同时又不失向技术层面的折中考虑。

本书建立在Evic Evans的《领域驱动设计》之上，通过我们所熟知的示例领域向我们讲解实际的DDD实现技术。每种设计原则都有真实的Java例子作为支撑，并且这些例子对于C#程序员来说也适用。所有的Java例子都出自于同一个案例研究：一个大型的基于Scrum的SaaS多租户系统。

作者带领我们超越了“DDD-Lite”的局限，DDD-Lite即是将DDD单纯地作为一套技术工具集来使用。通过讲解限界上下文、上下文映射图和通用语言，作者全面地向我们展示了DDD的“战略设计模式”。通过书中所讲到的技术和例子，我们可以加快软件开发速度，提升软件质量，使我们的软件更具灵活性和可伸缩性，同时更加紧密地与软件的业务目标保持一致。

本书内容包括：

- 以正确的方式带领你进入DDD世界，从而快速地从中获取价值。
- 将DDD用于不同的架构中，包括六边形架构、SOA、REST、CQRS、事件驱动架构和基于数据网格的架构。
- 适当地设计和实现实体——并且何时应该使用值对象而不是实体。
- 掌握DDD的领域事件技术。
- 通过ORM、NoSQL等实现资源库。

作者介绍： Vaughn Vernon是一个经验丰富的软件工匠，在软件设计、开发和架构方面拥有超过25年的从业经验。他提倡通过创新来简化软件的设计和实现。从20世纪80年代开始，他便开始使用面向对象语言进行编程；在90年代早期，他便在领域建模中应用了领域驱动设计，那时他使用的是Smalltalk语言。他在全球范围之内提供软件咨询和演讲，此外，他还在许多国家教授《实现领域驱动设计》的课程。

译者介绍： 滕云，ThoughtWorks软件工程师。当初抱着“非飞行器设计专业不读”的想法考入西北工业大学，却不料学起了机械和汽车。在尝尽了“从天上掉到地下”的滋味之后，又转行软件开发。目前主要从事银行、保险等领域的企业级软件开发，感兴趣的技术领域包括Java EE、Linux、领域驱动设计和构建自动化等。个人博客：http://www.davenkin.me（无知者云）。

上架建议：计算机科学>软件工程

PEARSON
www.pearson.com

ISBN 978-7-121-22448-5
9 787121 224485 >
定价：99.00元

责任编辑：张春雨
封面设计：李　玲

实现领域驱动设计

IMPLEMENTING DOMAIN-DRIVEN DESIGN

Vaughn Vernon 著
滕云 译
张逸 审

電子工業出版社
Publishing House of Electronics Industry
北京•BEIJING

内 容 简 介

领域驱动设计（DDD）是教我们如何做好软件的，同时也是教我们如何更好地使用面向对象技术的。它为我们提供了设计软件的全新视角，同时也给开发者留下了一大难题：如何将领域驱动设计付诸实践？Vaughn Vernon 的这本《实现领域驱动设计》为我们给出了全面的解答。

本书分别从战略和战术层面详尽地讨论了如何实现 DDD，其中包含了大量的最佳实践、设计准则和对一些问题的折中性讨论。全书共分为 14 章，在 DDD 战略部分，本书向我们讲解了领域、限界上下文、上下文映射图和架构等内容，战术部分包括实体、值对象、领域服务、领域事件、聚合和资源库等内容。一个虚构的案例研究贯穿全书，这对于实例讲解 DDD 实现来说非常有用。

本书在 DDD 的思想和实现之间建立起了一座桥梁，架构师和程序员均可阅读，同时也可以作为一本 DDD 参考书。

图书在版编目（CIP）数据

实现领域驱动设计/（美）弗农（Vernon,V.）著；滕云译. —北京：电子工业出版社，2014.3

书名原文：Implementing domain-driven design

ISBN 978-7-121-22448-5

Ⅰ.①实… Ⅱ.①弗… ②滕… Ⅲ.①软件设计 Ⅳ.①TP311.5

中国版本图书馆 CIP 数据核字(2014)第 021688 号

策划编辑：张春雨

责任编辑：张春雨

印　　刷：北京丰源印刷厂

装　　订：河北省三河市路通装订厂

出版发行：电子工业出版社

北京市海淀区万寿路 173 信箱　　邮编：100036

开　　本：787×980　1/16　　印张：36.5　　字数：817.6 千字

印　　次：2014 年 3 月第 1 次印刷

定　　价：99.00 元

凡购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：（010）88254888。

质量投诉请发邮件至 zlts@phei.com.cn，盗版侵权举报请发邮件至 dbqq@phei.com.cn。

服务热线：（010）88258888。

译者序

2013年秋天的某个周末，我在公司翻译本书。对面一个刚入职的大学毕业生向我问起一个关于软件建模的问题，我给她讲到了实体、领域服务和限界上下文等概念。语毕，我立即有两点感触：第一，我已经被DDD深深影响了；第二，即便是对于那些小的软件项目，DDD依然有用武之地。

通常的看法是，DDD更适合于大型的软件系统，这也使很多软件开发者对DDD敬而远之。其实不然，DDD首先作为一种思想而存在着，它无关系统大小，而是教我们如何做好软件。像实体、值对象这样的概念，我相信大家都在自己的项目中或多或少地用到，这些概念并不只存在于大型系统中，而是对任何系统皆然，因为它们体现了软件模型的本质所在。

曾经有一段时间，我希望在软件开发技能上有另一个层次的提高，于是我选择了DDD。在看完Eric Evans那本DDD开山之作之后，我了解了不少，也迷茫了许多。和其他人一样，我更希望看到一些实际的DDD例子。在网上搜索一番，我发现了本书；读完示例章节，我便爱不释手了。

在我看来，DDD绝非是什么标新立异之物，我更倾向于将其看成是软件发展的自然结果。就像在20世纪六七十年代出现了软件危机之后，面向对象成为了人们的救赎；瀑布式开发过程遇到瓶颈时，敏捷被搬上了舞台；而DDD则是对传统的以数据为中心的建模方式的反思结果。另外，我们还有“OO Done Right”的说法，即DDD是以正确的方式来使用面向对象的。

我很看重事物发展过程的“自然”面。你把一组随机数交给一个中学生排序，估计她/他也是能琢磨出一种排序算法的，此时的排序算法很可能就是我们在大学里才学到的直接排序法。我想很多人也都在程序中创建过一些服务类，并且将事务边界放在这些类的方法上，那么此时，我们所创建的便是DDD所称为的应用服务（Application Service）。于是我们看到，很多知识都可以通过我们自己的自然逻辑思考而获得。但是，我们和那些大牛的区别在于，他们有能力将这些知识概念化、理论化、抽象化和系统化。对于本书的作者Vaughn Vernon来说，也是如此。

如果说Eric Evans的《领域驱动设计》为我们提出了一些高屋建瓴的思想，那么本书便把这些思想落到了实处。而在本书中，作者也多次引用了《领域驱动设计》的相关章节。除此之外，本书还大量地引用了Gamma等人的《设计模式》和Martin Fowler的《企业应用架构模式》。所以，本书不只是关于软件建模的，而

是正如本书的赞美者之一Randy Stafford所说，它还可以用于更加宽阔的软件架构领域。

在收到电子工业出版社的张春雨编辑寄来的英文原著时，我便开始盘算：一本600页的书，每天翻译一页，我需要将近两年；每天翻译两页，也得十个月……一次次的组合之后，我已经厌倦了。于是不管三七二十一，白天工作，晚上翻译，平时工作，假期翻译。结果，四个月完工。

退却的时候也是有的，并且总会有这样或那样的借口："今天不在状态"、"明天再翻译也不迟"，诸如此类，凡此种种。每当此时，我便想起在故乡的山坡上顶着烈日当午、弯腰锄地汗流滴土的父亲，于是再次打开已经合上的笔记本电脑……

感谢我的父母赋予我的精神动力，使我得以顺利地完成本书的翻译。

在翻译的过程中，我得到了我的同事，在DDD方面颇有建树的张逸[1]的大力帮助。忘不了的，是那些在下班路上和他一起探讨聚合、CQRS的日子。

感谢郑晔[2]和格茸扎西在我还没有开始翻译本书时便主动提出帮我审校。另外，还要感谢澳大利亚的Mark Ryall和加拿大的Rick Harcus向我提供的有关英语语言文化上的帮助。

2013年10月 成都

1 张逸，ThoughtWorks咨询师，著有《软件设计精要与模式》，译有《恰如其分的软件架构》等书。

2 郑晔，ThoughtWorks咨询师，2013年OracleDuke选择奖得主，译有《Scala程序设计》和《Clojure编程乐趣》等书。

本书赞誉

“在《实现领域驱动设计》中，Vaughn不仅为DDD领域做出了卓越的贡献，还为更宽阔的企业应用架构领域写上了厚重的一笔。例如，在架构和资源库等核心章节中，Vaughn向我们展示了如何将DDD与各种架构风格和持久化技术融合在一起——包括SOA、REST、NoSQL和数据网格等——其中很多都是在Eric Evans那本DDD开山之作出版之后才出现的。另外，书中还讲到了对实体、值对象、聚合、领域服务、事件、工厂和资源库的实现，其中包括大量的例子。一言以蔽之，我认为这本书非常全面。对于那些希望提升自己技能的软件开发者来说，《实现领域驱动设计》将是一本绝佳的好书。”

——Randy Stafford，自由架构师，Oracle Coherence产品部

“领域驱动设计是一套非常强大的思想工具，它深远地影响着软件开发团队的效率。问题在于，许多开发者在应用这套思想工具时会不时地迷失方向，他们需要更实际的指导建议。在本书中，Vaughn将理论与实践联系在了一起。除了为我们讲解那些易被误解的DDD概念之外，Vaughn还讲到了一些新的概念，比如命令/查询职责分离（CQRS）和事件源等。对于那些希望实际应用DDD的人来说，这是一本必读之作。”

——Udi Dahan，NServiceBus创始人

“多年以来，DDD的开发者们都希望获得一些更实际的帮助。Vaughn缝合了理论和实践之间的间隙，向大家提供了一套完整的DDD实现参考。他向我们展示了如何在当前软件项目中使用DDD，并且向我们提出了大量的实际建议。”

——Alberto Brandolini，DDD导师（由Eric Evans和Domain Language，Inc颁发证书）

“《实现领域驱动设计》清晰地向我们展示了DDD的核心话题。本书的写作风格非常友好，就像一个值得信赖的导师在给你讲课一样。读完本书，你将能够应用DDD的各个重要概念。我在阅读本书的时候，在很多章节中都做上了着重标记……我会经常地参考并推荐本书。”

——Paul Rayner，首席咨询师，DDD导师（由Eric Evans和Domain Language，Inc颁发证书），DDD Denver创始人。

“在我所教的DDD课程中，很重要的一点便是如何将所有的DDD理论付诸实践。有了本书，DDD社区便有了可供参考的资料。《实现领域驱动设计》包含了创建DDD系统的方方面面，从具体的实现细节到高层的设计思想。这是一本了不起的DDD参考书，同时也是Eric Evans那本DDD开山之作的极佳伴侣。”

——Patrik Fredriksson，DDD导师（由Eric Evans和Domain Language，Inc颁发证书）

“如果你关心软件工艺——你也应该这么做——那么领域驱动设计便是非常重要的一项技能，而《实现领域驱动设计》则向我们提供了一条迈向成功的捷径。本书详尽地讨论了DDD的战略模式和战术模式，使开发者能够立即将理论付诸实践。今后的业务软件系统将从本书中受益匪浅。”

——Dave Muirhead，首席咨询师， Blue River Systems 集团

“DDD既有理论，也有实践，这些都是每个开发者应该了解的，而本书则很好地弥补了理论与实践之间的差距。强烈推荐本书！”

——Rickard O berg，Java开发者，Neo Technology公司

“在《实现领域驱动设计》中，Vaughn采用了自顶向下的方法，首先讲到了DDD的战略模式，比如限界上下文和上下文映射图，然后讲到了战术模式，比如实体、值对象和领域服务等。案例研究贯穿全书，要从中有所学，你需要在该案例研究上下足功夫。如果你这么做了，你便能看到将DDD应用于复杂领域的意义所在。书中包含了大量的旁注、图标和示例代码。如果你希望使用当下最常见的架构风格来创建一个DDD系统，那么Vaughn的这本《实现领域驱动设计》便是我所推荐的。”

——Dan Haywood，《Domain-Driven Design with Naked Objects》作者

“本书采用了一种自顶向下的方式来讲解DDD，这种方式将DDD的战略模式和战术模式自然地衔接起来。在本书中，Vaughn强调了业务领域的价值，同时也给出了技术上的讨论。因此，DDD在软件开发中的角色也变得非常清晰。很多时候，我的团队，包括我本人，在应用DDD时都会遇到这样那样的麻烦。有了《实现领域驱动设计》的指导，我们得以克服种种挑战，进而将付出立即转化为业务价值。”

——Lev Gorodinski，首席架构师，DrillSpot.com

序

在本书中，Vaughn Vernon以一种特有的方式向我们展示了领域驱动设计（Domain-Driven Design，DDD）的各个方面，其中包括对新概念的解释、新的例子和原创的话题组织方式。我相信，这种新颖的方式可以帮助大家掌握DDD的各种微妙之处，特别是非常抽象的聚合和限界上下文。不同的人习惯用不同的方式来理解这些概念，而在缺少多种解释的情况下，想要了解这些微妙的抽象概念是非常困难的。

本书包含了在过去9年中出现在各种论文和讲稿中的对DDD的深层剖析，而这些是在之前的书籍中没有的。本书将领域事件与实体和值对象一道看作是模型的基础部件。另外，书中还讨论了“大泥球”（Big Ball of Mud）架构和如何将其放置在上下文映射图（Context Map）中。Vaughn还向我们阐述了六边形架构（Hexagonal Architecture），这种新兴的架构与分层架构相比，能够更好地描述我们要完成的事情。

我是在将近两年前第一次接触到本书内容的，那时Vaughn已经开始撰写本书有一段时间了。在第一次DDD峰会上，我们中的几个编写了关于DDD的若干话题，比如有关DDD的新知识，或者DDD社区所期待的一些针对性建议等。Vaughn负责写聚合部分，这一写便是一个有关聚合的文章系列，并且写得非常出色，最后，这个系列成为了本书中的一个章节。

在那次峰会上，与会人员们一致认为：一套更加具有规约性的DDD模式是大有裨益的。诚实地讲，对于软件开发中的任何问题，答案都是“得看情况”。然而，这对于那些希望学到实际应用技术的人来说却没多大用处。人们需要更加实际的指导。经验法则不见得一定要放之四海而皆准，但在通常情况下，他们可以工作得很好，也应该被首先尝试。出于自身的果决性，这些经验法则蕴含着解决问题的思想方法。Vaughn的这本《实现领域驱动设计》将各种明晰的建议很好地融合在一起，同时又给出了一些折中性的讨论，从而避免了将这些建议过于简单化。

一些额外的DDD模式，比如领域事件，已经成为了DDD的主流模式，人们也学会了如何应用这些模式，并尝试着在新架构和新技术中采用这些模式。在我的《领域驱动设计：软件核心复杂性应对之道》出版9年后，有太多关于DDD的新知识需要谈及，Vaughn的这本书则是最全面的阐述。

—Eric Evans
Domain Language, Inc.

前言

所有的计算都表明它不工作，唯一的做法是：使其工作。
——Pierre-Georges Latécoère
早期法国航空企业家

是的，我们将使其工作。然而，在软件开发过程中采用领域驱动设计却是困难的。即便是有能力的开发者，也很难找到实现领域驱动设计的正确方法。

起飞，着陆

在我小的时候，我的父亲学习过驾驶小型飞机。我们经常会全家出去飞行，有时会飞到另一个机场，在那里吃过午饭后再返回。当父亲时间有限而他依然想飞时，父亲便带上我一起在机场上空盘旋，起飞，着陆，再起飞，再着陆。

也会有些长途飞行，这时我们会带上一张由父亲先前绘制好的路线图。我们几个小孩便当起了领航员：将图上的标志对应着陆地上的地标，以确保我们没有跑偏航线。这是一件很有趣的事情，因为要识别远在地面上的物体是很有挑战性的。事实上，我敢肯定父亲根本不用我们领航便知道我们处于什么方位——他能看到仪表盘上的所有信息，并且他拥有仪表飞行执照。

空中的景观的确改变了我的视野。不时地，父亲和我会飞过我们乡下的房子。在几百英尺的高空中，我体会到了另一种“家”的概念，而这在之前是没有过的。当我们飞过自家的房子时，母亲和我的姐妹们便会跑到院子里向我们挥手。我知道那是她们，即便我看不清楚她们是谁。谈话肯定是不行的，连大声喊都不行，她们是听不见的。我还可以看到将我家和外面公路分开的护栏，平时我们会像走平衡木一样在护栏上面走来走去。从空中看，它们就像被细心编排过的小树枝一样。我们

家的院子很大，每每到了夏天，我都会开着割草机一排一排地修理院子里的草坪。而在空中时，我只能看到一片绿色，小草的叶子肯定是看不清楚的。

我喜欢在空中的时刻，直到现在我还不时回想起这些时刻，好像那个降落飞机的黄昏就发生在不久以前一样。虽然如此，在地面上的感觉依然是无法取代的，因为它给我一种脚踏实地的感觉。

着陆于领域驱动设计

一开始接触领域驱动设计（DDD）就像一个小孩之于飞行一样。天空中的景色是令人惊叹的，但有时我们却因为过于陌生而搞不明白它们到底是什么。要从甲地到乙地显得如此的遥远。然而，DDD的“成年人”们却总知道他们所处的方位，因为他们在很早之前便绘制好了路线图，并且能够完全按照仪表进行相应的操作。而还有很多人找不到“在地面上”的感觉，此时我们需要的是“稳定着陆”的能力，然后找到一张地图给我们指引方向。

Eric Evans的《领域驱动设计：软件核心复杂性应对之道》是一本经得住时间考验的经典之作。我坚定地相信，在接下来的几十年里，本书依然会是开发者的实用指导。和其他模式一样，该书为我们建立起了一种高屋建瓴式的宽阔视野。然而，对于如何实现DDD，我们可能将面对更多的挑战。通常来说，我们更渴望看到一些具体的例子。

我的目标之一便是帮助你来一个“软着陆”，保全飞机，然后沿着一条周知的线路带你回家。这将帮助你如何更好地去实现DDD，并且通过你所熟悉的工具和技术给出示例演示。当然，任何一个人都不可能一直呆在家里，所以我还会带领你到新的地带去冒险，这些地带你可能从来没有去过。冒险之路是险峻的，但是在正确的战术应对下，征服这些困难是可能的。在这条冒险之路上，你将学到另外的架构和模式来集成多个领域模型。你将接触到先前没有被研究过的集成方法，并且学到如何开发自治性服务。

我将向你提供一张对短途旅行和长途旅行均适用的地图，它可以帮助你更好地享受沿途风景，同时又不至于迷失途中。

对照地形，绘制飞行图

在软件开发的过程中，我们经常做的一件事便是将一种东西映射到另一种东西。我们将对象映射到数据库，映射到用户界面，或者映射到不同的应用层展现（包括作为消费方的其他系统或应用程序）。在所有这些映射中，我们很自然地希望在Evans提出的高层模式和具体实现之间存在一种映射。

即便你已经接触过DDD，你依然有很多可以获益的地方。有时，DDD首先被看作是一套技术工具集，有人将此称为DDD-Lite。我们可能已经对实体、服务等DDD概念非常熟悉了，并且大胆地尝试着设计聚合，还通过资源库来管理持久化。这些模式是大家相对熟知的，使用起来很容易，我们甚至还使用了值对象。以上这些都属于战术设计模式范畴，也即更加偏向技术层面。这些模式可以很好地帮我们解决软件问题。而同时，对于战术性模式，我们依然有许多需要学习的。我将战术模式映射到实现层面。

你曾了解过战术建模之外的东西吗？你曾了解过被称为DDD“另一半”的战略设计模式吗？如果你还没有使用过限界上下文和上下文映射图，那么你很有可能也没有使用过通用语言。

如果说Evans在软件开发社区有一项发明，那便是通用语言。通用语言是一种团队协作模式，用于捕捉特定业务领域中的概念和术语。一个特定领域的软件模型通过不同的名词、形容词和动词来表达，这些词汇是开发团队正式使用的，而团队中应该包含一个或多个领域专家。然而，将通用语言仅限定于一些词汇则是错误的。就像自然语言反映人们的思想一样，DDD的通用语言反映了领域专家对于软件系统的思维模型。通用语言和那些战略和战术性的建模模式同等重要，在有些情况下甚至更具有持久性。

简单地讲，DDD-Lite将导致劣质的领域对象，因为通用语言、限界上下文和上下文映射图的作用太大了，你从其中获得的并不只是一套团队共用的语言。在限界上下文中用通用语言来表述一个领域模型可以增加业务价值，并且使我们确信所开发软件的正确性。即使从技术的角度，它也可以帮助我们创建更好的领域模型，这样的模型行为丰满，业务纯净，并且可以减少犯错误的可能性。因此，我将战略设计模式映射到了可理解的实际例子中。

本书对于DDD的映射可以帮助你同时体会到战略设计和战术设计的好处。通过一些具体的例子，你将感受到这些DDD映射的业务价值和技术展现力。

如果我们对于DDD的所有实践都只是停留在“地面上”，那将是令人失望的。过度地拘泥于细节将使我们丧失在空中俯瞰的机会。所以，不要将自己局限在地面的细节上，要勇敢地飞翔在空中，居高临下。搭上战略设计的航班，去了解限界上下文和上下文映射图，你将获得更广阔的视野。当你从DDD的航班中获益时，我的目的也就达到了。

各章概要

以下是各章的主要内容以及你将如何从中获益。

第1章：DDD入门

本章向你介绍DDD的好处，并且教你如何尽可能多地去实现DDD。你将学到当你在应对复杂的软件系统时，DDD可以为你的项目和团队带来什么。同时，你将了解到通常的DDD替代方案以及这些方案为什么会导致问题。作为对DDD的基础讲解，本章将教你如何在项目中开始采用DDD，还有如何向你的领域专家和技术团队推销DDD。在DDD的武装下，你将学会如何迎接挑战，勇往直前。

本章将介绍关于一个公司及其团队的案例研究，虽然该公司是虚构的，但是他们所面临的DDD挑战却是真实存在的。该公司旨在开发一个新的多租户SaaS (Software as a Service，软件即服务) 软件产品。不出所料，在使用DDD时，他们犯了一些常见的错误。不过还好，他们发现了这些错误，并解决了一些问题，因此项目还算没有偏离正轨。该团队需要开发一套基于Scrum的项目管理软件。该案例还会在本书的后续章节中连续讲到。每一种战略和战术模式都将教给这个团队。在这个过程中，团队有误入歧途的时候，但最终他们将向着成功的DDD实践昂首阔步。

第2章：领域、子域和限界上下文

领域、子域和核心域分别是什么？限界上下文是什么，我们为什么要使用它，并且如何使用？这些问题将在这个SaaS项目团队犯错误的时候给予解答。在他们的第一个DDD项目中，他们并不了解子域、限界上下文和通用语言这些概念。事实上，他们根本不知道什么是战略设计，只是采用了战术设计来解决一些技术问题。这样他们在开始设计领域模型的时候便遇到了不少问题。幸运的是，他们及时地意识到了这些问题，项目还有挽回的余地。

本章还讲到了如何使用限界上下文对模型进行分离，这是非常重要的；同时还讲到了一些模型分离不当的反例，并且给出了有效的实现建议。在采用了这些建议之后，该团队的成员们重新创建了两个不同的限界上下文。这种合理的模型分离带来的好处是引出了第三个限界上下文——核心域，这将是本书使用的主要例子。

对于那些苦于单单从技术层面应用DDD的人来说，本章应该能引起你的共鸣。如果你还是DDD战略设计的外行，那么本章将为你指明方向。

第3章：上下文映射图

上下文映射图帮助我们理解业务领域、模型间的边界，以及这些模型之间的集成方式。

上下文映射图绝对不只是绘制系统架构图这么简单，它处理的是不同限界上下文之间的关系，以及如何在不同的模型之间映射对象。对于在复杂的业务系统中使用好限界上下文，这是至关重要的。在第2章中，团队成员们在首次尝试限界上下文时碰到了问题。本章中，他们将学着如何利用上下文映射图来解决这些问题。这样的结果是产生了两个体面的限界上下文，这两个上下文将被另外一个负责核心域的团队所使用。

第4章：架构

我们都知道分层架构，但它是开发DDD软件的唯一方式吗，也或许还存在另外的方式？在本章中，我们将讲到：六边形架构（端口和适配器）、面向服务架构、REST、CQRS、事件驱动（管道和过滤器，长时处理过程，事件源）和数据网格，其中好几种架构都将被该团队成员所采用。

第5章：实体

在DDD的战术模式中，我们将首先讲到实体。团队成员们一开始过于强调实体的作用而忽视了值对象。受到数据库和持久化框架的影响，实体被该团队滥用了，此时他们开始讨论如何避免大范围地使用实体。

在本章中，你将看到很多优秀的实体设计例子。同时，本章还将讲到如何使用实体来表达通用语言，以及如何对实体进行测试、实现和持久化。

第6章：值对象

早些时候，团队成员们错过了采用值对象的好机会。他们过于注重为实体创建一些单一的属性，这种方式是欠妥的，更好的方式是将这些单一的属性聚合成一个不变的整体。本章将从不同的角度讲解如何设计值对象，以及在什么时候采用值对象会优于实体。同时，本章还包含了一些其他话题，比如值对象在集成中的角色和对标准类型的建模等。然后，本章讲到了如何设计以领域为中心的测试，如何实现值对象。此外，本章还讲到了在聚合中存储值对象时，如何避免持久化机制所带来的不利影响。

第7章：领域服务

本章将讲到，在领域模型中，什么时候应该将一个概念建模成粒度适中，并且无状态的领域服务。你将学到何时应该使用领域服务而不是实体或值对象，以及如何使用领域服务来处理业务逻辑和技术上的集成。团队成员们向我们展示了何时应该使用领域服务，以及如何设计领域服务。

第8章：领域事件

Eric Evans并没有在他的书中正式介绍领域事件，领域事件是在他那本书出版之后才进入人们视野的。在本章中，你将学到为什么领域事件如此有用，以及使用领域事件的不同方法。领域事件甚至被用来辅助集成和自治性服务。在软件系统中，我们经常使用一些技术层面的事件机制，但本章将着重讲解领域事件与这些事件机制的区别。本章还将指导你如何设计并实现领域事件，包括一些可行的方案和对这些方案的权衡选择。然后，本章将讲到如何创建一个发布-订阅机制；如何利用事件来集成整个企业软件中的各个订阅方；如何创建和管理事件存储；如何处理消息机制所面临的常见挑战等。

第9章：模块

对于模型中的对象，我们应该如何将他们组织在大小适中的容器中呢？我们又如何保证不同容器中的对象之间只存在有限的耦合？另外，我们如何对这些容器进行命名以体现通用语言？除了包和命名空间之外，我们如何使用由语言和框架提供的现代模块化机制，比如OSGi和Jigsaw？在本章中，你将看到SaaS团队成员是如何在不同的项目中使用模块的。

第10章：聚合

在DDD的战术模式中，聚合可能是最不容易理解的了。然而，在遵循一定的经验法则的情况下，我们是能够更简单、更快地实现聚合的。在本章中你将学到：如何利用聚合在不同的小规模对象集群间创建一致性边界，从而降低模型的复杂性。由于在细枝末节上花了太多精力，SaaS团队成员们在设计聚合时总是磕磕绊绊。我们将仔细研究该团队所面临的挑战，并且分析错误的原因以及他们的应对策略。结果，团队成员们对他们的核心域有了更深层次的理解。我们将看到，在合理的事务处理和保证最终一致性 (Eventual Consistency) 的前提下，该团队更正了他们所犯的错误，并且在一个分布式环境中设计出了更具有伸缩性和更高效的模型。

第11章：工厂

工厂已经在[Gamma et al.]中被大量地谈及了，为什么还要讲呢？本章并不打算重蹈覆辙，而是将重点放在“工厂应该存在于何处”这个问题上。在本章中，我们将讲到在DDD中实现工厂的技巧。团队成员在他们的核心域中创建的工厂可以简化客户端接口，并且对模型的消费方起到保护作用，从而避免了在多租户环境中引入灾难性的bug。

第12章：资源库

资源库只是一个数据访问对象 (Data Access Object, DAO) 吗？如果不是，它们之间有什么区别呢？我们为什么应该将资源库看成是对集合的模拟而非数据库呢？在本章中，我们将讲到如何利用ORM来实现资源库，其中有两种ORM方案，一种采用基于网格的分布式缓存，另一种则采用NoSQL的键值对存储。团队成员们可以采用任何一种作为他们的持久化机制。

第13章：集成限界上下文

到现在为止，你已经了解了战略层次的上下文映射图和多种战术层次的模式。本章将讲到，在DDD中，我们如何通过上下文映射图来集成不同的模型。在团队对核心域和其他辅助性的限界上下文进行集成时，我们将给出相应的建议和指导。

第14章：应用程序

对于每一个核心域的通用语言，我们都设计了相应的模型，并且进行了足够的测试，模型工作正常。然而，客户应该如何使用我们的模型呢？他们应该使用DTO将数据在模型和用户界面之间传输吗？或者存在其他方案可以实现模型和展现组件间的数据传递？DDD中的应用服务和基础设施是如何工作的？对于这些问题，本章都将做出解答。

附录A：聚合与事件源：A+ES

事件源是一种持久化聚合的重要技术，同时也是事件驱动架构的基础。事件源通过一系列的事件来表示聚合的所有状态。通过有序的事件重放，我们可以重新构建聚合的状态。当然，使用事件源的前提是：它能够简化对数据的持久化，并且能够捕捉到那些具有复杂行为属性的概念。

Java和开发工具

本书中的绝大多数例子都是使用Java语言编写的。我本来可以用C#的，但是我有意识地使用了Java。

首先，我认为Java社区正在抛弃好的软件设计和开发实践。现在，对于多数Java项目而言，要在其中找到一个好的领域对象恐怕是困难的。在我看来，Scrum和敏捷被人们看成了优良设计的替代品，而其中的产品待定项（Product Backlog）被看成了设计本身。多数敏捷人士并不会过多地去思考这些待定项是否会影响到业务模型。我得说明，Scrum的本意绝对不是要取代设计。不管有多少项目经理想将你捆绑在持续交付这条路上，我得说Scrum并不仅仅是要取悦于那些甘特图（Gantt chart）的追随者们。然而，太多的时候，情况的确是这样的。

我认为这是个很大的问题，所以我想鼓励Java社区重新回到领域建模中来，同时我会通过本书向大家说明，设计是可以使我们获益的。

此外，在.NET社区中已经有很好的DDD资源了，比如Jimmy Nilsson的《领域驱动设计与模式实战》[Nilsson]。由于Jimmy的出色工作和其他人对Alt.NET的倡导，.NET社区中正掀起一阵优秀设计的开发浪潮，这是Java社区需要注意的。

其次，我意识到C#.NET人员在理解Java代码上并不存在什么困难。由于很多DDD社区的人都在使用C#.NET，而本书的早期校对人员也都是C#程序员，但是我从来就没有收到他们的抱怨。因此，我便不用顾虑这些了。

在我写这本书时，业内正将目光从关系型数据库转向基于文档和键值对的存储方案。这是有原因的，Martin Fowler将这些存储方案称为“面向聚合存储”。这种命名是恰当的，它很好地描述了在DDD中使用NoSQL的好处。

但是，就我从事咨询的经验来看，很多开发者还是认定了关系型数据库和对象-关系映射。因此我想，NoSQL追随者们应该能够理解我在书中包含对象-关系映射的章节。然而，我的确得承认，这可能会招致那些认为存在对象-关系阻抗失配（Object-Relational Impedance）的人的鄙视。这无所谓，对此我表示接受，因为绝大多数人在他们的日常工作中都还得面对这种对象-关系阻抗失配。

当然，在第12章“资源库”中，我同样提供了基于文档的、键值对的和数据网格的存储方案。在多处地方，我都讨论到了NoSQL对聚合设计的影响。NoSQL趋势很有可能持续下去，那些对象-关系型的开发者们应该注意了。在本书中你将看到，我能够同时理解两个阵营的观点，并且对于双方的观点我都同意。这些都是技术趋势所导致的摩擦，而这对于积极的变革是有必要的。

致谢

非常感谢Addison-Wesley出版社给我机会出版本书。正如我之前在上课和演讲时所说，我将Addison-Wesley看成是一个懂得DDD价值的出版商。在本书的编辑过程中，Christopher Guzikowski和Chris Zahn (Dr. Z)给了我很大的支持。那天，Christopher Guzikowski打电话给我，说他希望我成为他的签约作家。我是不会忘记那一天的，我也不会忘记Christopher Guzikowski对我的鼓励。当然，是Dr. Z将本书的文本变成了可出版的状态。感谢我的出版编辑Elizabeth Ryan协调本书的出版细节。同时，我还要感谢我的技术编辑，Barbara Wood。

回到从前，Eric Evans花了他职业生涯里的5年时间完成了DDD的定义工作。没有他的努力，没有从SmallTalk和模式社区中迸发出来的智慧，许多开发者都只能依旧苦苦摸索，最终交付劣质的软件。可悲的是，这样的问题太常见了。正如Eric所说，那些劣质的软件以及开发团队无创新式的枯燥性几乎使他离开软件领域。因此，我们欠Eric一个大大的感谢。

Eric邀请我参加了2011年的DDD峰会。会毕，大家一致认为，DDD的领导层应该提供一套指导以帮助更多的开发者在DDD上取得成功。那时，我已经写本书有很长一段时间了，并且我们充分地体会到了开发者们所缺少的东西。我自告奋勇，决定写一个文章系列来介绍有关聚合的“经验法则”。之后，我将这个名为“高效聚合设计 (Effective Aggregate Design)”的文章系列当成了本书第10章的基础。当该系列文章在dddcommunity.org网站上发布时，我才知道，人们对这样的指导真是如饥似渴。感谢那些DDD领导层中审阅了这个文章系列的同仁们，并感谢他们为本书提供的建议和反馈。Eric Evans和Paul Rayner对该文章系列做了多次细致的审阅。另外，我还从Udi Dahan、Greg Young、Jimmy Nilsson、Niclas Hedhman和Rickard O berg处获得了反馈。

特别感谢DDD社区的资深成员，Randy Stafford。几年前，我在丹佛举行DDD演讲，Randy也参加了。之后，他敦促我更多地参与到更大的DDD社区中去。一段时

间之后，Randy将我介绍给了Eric Evans，由此我得以在DDD社区中与大家一起讨论问题。我的一些想法并不那么容易达到，而Eric则说服我们将关注点放在一些具有近期价值的东西上。正是有了那次讨论，才有了后来2011年的DDD峰会。虽然Randy由于忙于Oracle Coherence相关工作而无法参与本书的撰写，我想以后我是可以和他合作来写点什么的。

非常感谢Rinat Abdullin、Stefan Tilkov和Wes Williams，他们都为本书撰写了一些专题内容。要了解有关DDD的一切几乎是不可能的，要在软件开发的各个领域都成为专家更不可能。这也是为什么我邀请他们撰写本书的第4章和附录A中的专题。感谢Stefan Tilkov在REST方面给我的帮助，感谢Wes Williams在GemFire上的经验，也感谢Rinat Abdullin与我们分享有关事件源和聚合实现方面的知识。

本书早期审阅者之一是Leo Gorodinsk。我第一次见到Leo是在丹佛。他根据自己的项目中采用DDD的经历向本书提出了很多宝贵的反馈。我也希望本书能够像他帮助我一样帮助他。我将Leo看成是DDD未来的一部分。

还有很多人都为本书的至少一章提出了反馈。其中，那些更具批评性的反馈提供者有Gojko Adzic、Alberto Brandolini、Udi Dahan、Dan Haywood、Dave Muirhead和Stefan Tilkov。特别是，Dan Haywood和Gojko Adzic提供了很多早期的反馈，其中主要是关于本书“最难读”的那些内容。我很高兴他们能够忍耐下去并且帮我做出更正。Alberto Brandolini在战略设计，特别是上下文映射图方面的洞见使得我将关注点集中在这些概念的核心上。DaveMuirhead在面向对象设计、领域建模、对象持久化和内存数据网格方面——包括GemFire和Coherence——都拥有非常丰富的经验。本书中对对象持久化历史和实现细节的讲解便是受他的影响而完成的。除了在REST方面的贡献，Stefan Tilkov还在SOA、管道和过滤器方面向我提供了额外的支持。最后，UdiDahan帮助我澄清了有关CQRS、长时处理过程（即Sagas）和NServiceBus方面的概念。其他为本书提供了有价值反馈的还有：Rinat Abdullin、Svein Arne Ackenhausen、Javier Ruiz Aranguren、William Doman、Chuck Durfee、Craig Hoff、Aeden Jameson、Jiwei Wu、Josh Maletz、Tom Marrs、Michael McCarthy、Rob Meidal、Jon Slenk、Aaron Stockton、Tom Stockton、Chris Sutton和Wes Williams。

Scorpio Steele为本书提供了非常棒的插图。Scorpio使IDDD团队的每一个人都成为了超级英雄。我的朋友Kerry Gilbert为本书做了非技术性的审阅。其他人的帮助使得本书在技术上是正确的，而Kerry则在行文语法方面给了我很大的帮助。

我的父母为我的写作提供了灵感，在我这一生中，他们一直在支持着我。我的父亲——本书“牛仔的逻辑”幽默片段中的AJ——并不只是一个牛仔。不要搞错了。成为一个不错的牛仔已经非常好了，而我的父亲则在很多方面都展现出了他的才艺。除了喜欢飞行之外，我的父亲还是一个优秀的土木工程师、土地测量员，一个有天赋的谈判高手。另外，他还依旧喜欢着数学，并且研究星系。在我10岁的时候，我父亲就教我如何求解直角三角形。谢谢您，父亲，在我很小的时候就教给我这些。还要感谢我的母亲，她总是在我面临挑战时给予我鼓励和支持。

虽然本书是献给我的妻子Nicole和我们的儿子Tristan的，我还是想在这里再特别提及一下。他们使得我坚持写下去并最终完成本书。没有他们的支持和鼓励，这些都是不可能的。太感谢你们了，我亲爱的Nicole和Tristan。

关于作者

Vaughn Vernon是一个经验丰富的软件工匠，在软件设计、开发和架构方面拥有超过25年的从业经验。他提倡通过创新来简化软件的设计和实现。从20世纪80年代开始，他便开始使用面向对象语言进行编程；在20世纪90年代早期，他便在领域建模中应用了领域驱动设计，那时他使用的是Smalltalk语言。他在很多业务领域都有从业经验，包括航空、环境、地理、保险、医学和电信等领域。同时，Vaughn在技术上也取得了很大的成功，包括开发可重用的框架和类库等。他在全球范围之内提供软件咨询和演讲，此外，他还在许多国家教授《实现领域驱动设计》的课程。你可以通过www.VaughnVernon.co访问到他的最新研究成果。他的Twitter：@VaughnVernon。

如何使用本书

Eric Evans在他那本《领域驱动设计》中向我们展示了一整套模式语言。模式语言是相互关联的众多软件模式的一个集合，任何一种模式都会引用并依赖于其他一种或多种模式。这意味着什么呢?

这意味着当你在阅读本书时，某个章节中出现的有些DDD模式并不会在该章节中讲到，甚至在该章节之前都没有被谈及到。不要担心，继续往下读，被引用的模式将在本书的其他章节中做详细讲解。

在本书中，我将使用下表中的行文惯例：

表G.1 本书的行文惯例

出现文本	含义
模式名字 (#)	1.该模式是第一次出现在本书中，或者 2.该模式已经在本章中出现了，并且非常重要。
限界上下文 (2)	表示所引用的限界上下文在第2章中有详细的讲解。
限界上下文	表明限界上下文已经在本章中出现过了，这里我并不会每次引用一个模式时都将其标为粗体并后加章号。
[REFERENCE]	表明对参考文献的引用
[Evans] 或 [Evans, Ref]	表明对于被引用的模式，你可以参考Evans的著作以获得更多信息。[Evans]表示他那本经典的《领域驱动设计》，[Evans, Ref]并不表示引用Evans那本书本身，而是另外的引用源，该引用源也引用了Evans书中模式，并且在此基础上有更新和扩展。
[Gamma et al.] 和 [Fowler, P of EAA]	[Gamma et al.]表示Gamma等人所著的那本经典的《设计模式》。 [Fowler, P of EAA]表示Martin Flowler的《企业应用架构模式》。 在本书中，我会经常引用到这两本书，虽然我还引用了其他的，但这两本是主要的。.

在阅读的过程中，如果遇到对某个模式的引用，比如限界上下文，此时你通常可以在另外某个章节找到对该模式的讲解。

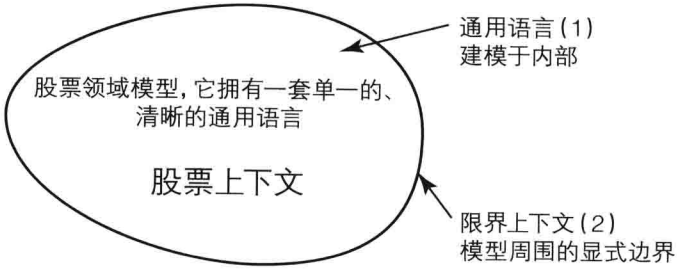
如果你已经读过[Evans]，并且对其中的模式有一定的了解，那么本书可以帮助你进一步澄清DDD的概念，然后引导你对既有的模型进行改进。此时你可能并不需要一个总览式的介绍。但是，如果你还是DDD的新手，那么下面的内容将为你讲到不同的DDD模式是如何协同工作的，并且如何更好地使用本书，接着往下读吧。

DDD总览

早些时候，我讲到了DDD的**通用语言**（Ubiquitous Language，1）。通用语言作用于某个**限界上下文**（Bounded Context，2），它对于领域建模是非常重要的，你应该好好地熟悉一下。请记住，不管你是在战术上还是战略上设计软件模型，你都应该保证：在一个特定的限界上下文中只使用一套通用语言，并且保证它的清晰性和简洁性。

战略建模

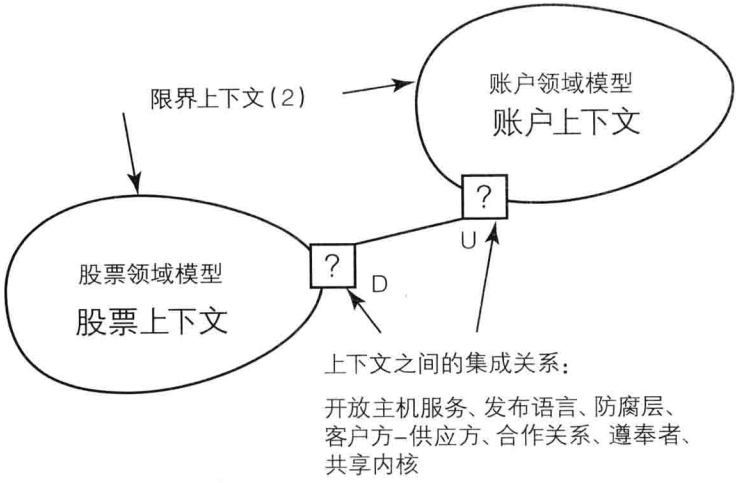
限界上下文是一种概念上的边界，领域模型便工作于其中。同时，限界上下文为通用语言提供了一套环境，项目成员便通过通用语言来表达软件模型，如图G.1所示。



图G.1 限界上下文和通用语言

在战略设计的过程中，你将发现**上下文映射图**（Context Map，3）是非常有用的，如图G.2所示。你的团队将使用上下文映射图来理解项目的范围。

以上我们简要地了解了DDD的战略设计，这是我们必须好好理解的概念。

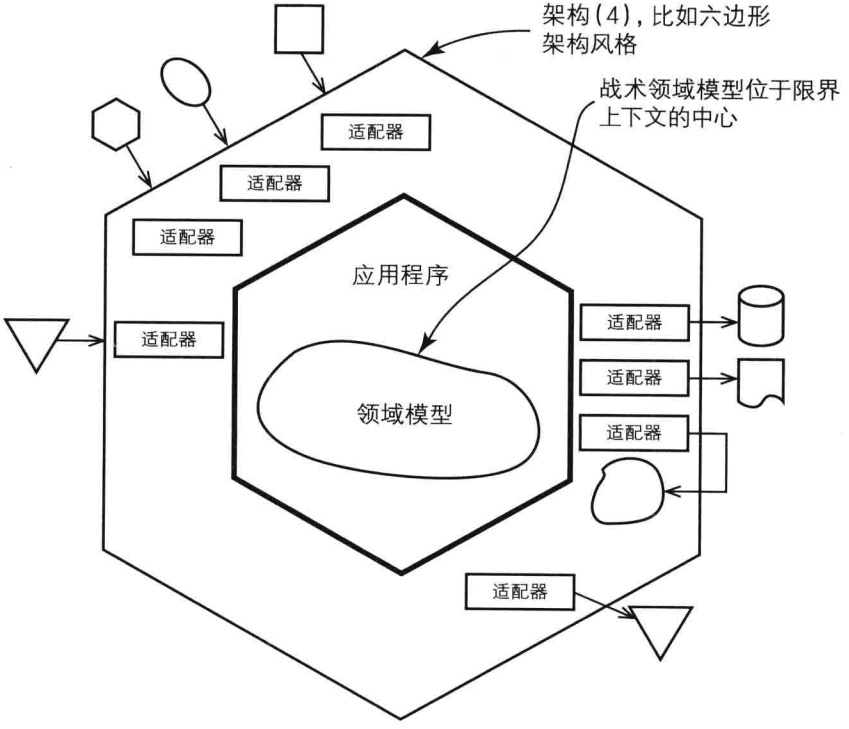


图G.2 上下文映射图展示限界上下文之间的关系

架构

有时，一个新的限界上下文或上下文映射图可能需要一种新的**架构**（Architecture，4）。你应该牢记：通过战略和战术设计而成的领域模型应该是架构中立的。当然，在模型周围和模型之间则是存在架构的。一种能够支撑限界上下文的架构是六边形（Hexagonal）架构，它可以辅助其他架构风格，比如**面向服务**（Service-Oriented）架构、REST和**事件驱动**（Event-Driven）等。六边形架构如图G.3所示，从表面看，这种架构有点复杂，但是事实上却恰恰相反。

有时我们过于强调架构而忽略了DDD建模的重要性。架构固然是好的，但是架构并非一成不变。此时我们须要正确地处理优先级，将重点放在领域模型上，因为领域模型将产生更多的业务价值，并且更具有持久性。



图G.3 六边形架构风格，领域模型位于软件的中心

战术建模

我们在限界上下文中进行DDD的战术建模。战术设计的一个重要模式是**聚合**（Aggregate，10），如图G.4所示。

聚合可以由单个**实体**（Entity，5）组成，也可以由一组实体和**值对象**（Value Object，6）组成，此时我们必须在聚合的整个生命周期中保证事务上的一致性。有效地对聚合进行建模是重要的，同时聚合又是DDD中最不容易理解的概念之一。你可能会问，既然聚合如此重要，那为什么要将其放在本书的后面呢？首先，本书中战术模式的出现顺序和[Evans]一样，此外，由于聚合以其他战术模式为基础，所以我们会先讲到实体、值对象等基本模式，再讲解聚合。

聚合实例通过**资源库**（Repository，12）进行持久化，另外，对聚合的查找和获取也通过资源库完成，如图G.4所示。

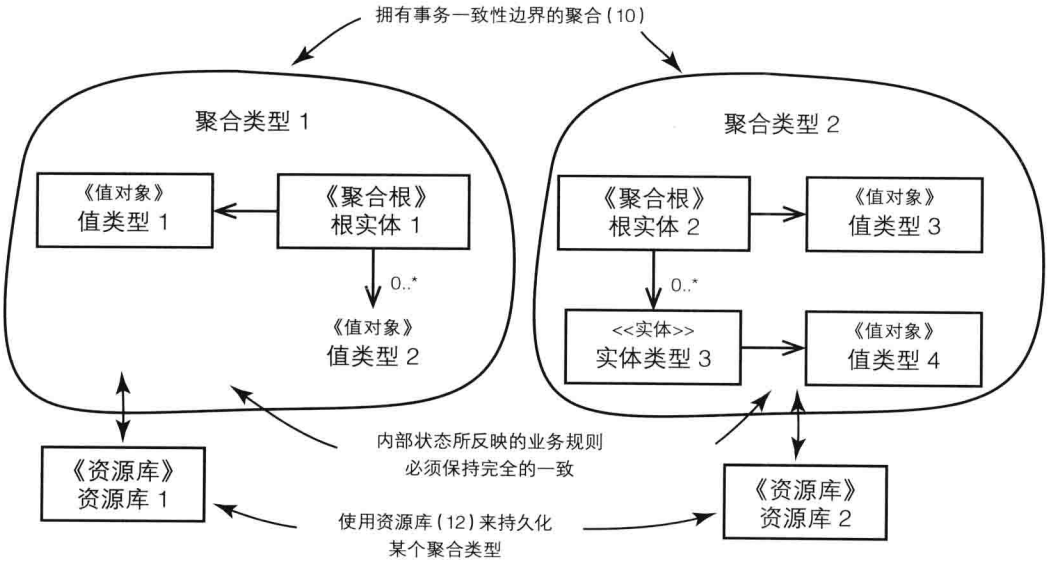
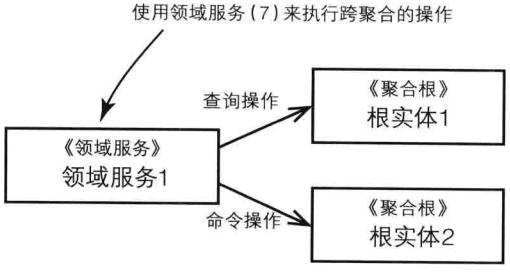


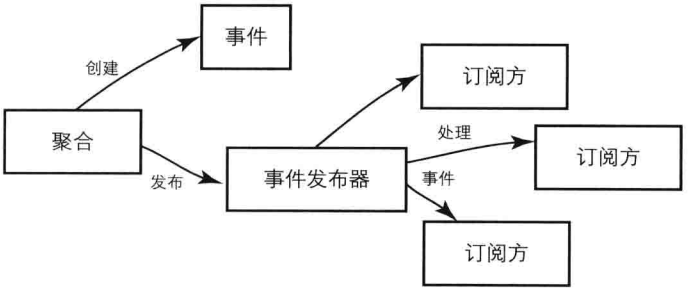
图 G.4 两个聚合类型，它们拥有各自的事务一致性边界

在领域模型中，有些业务操作并不能自然地放在实体或值对象上，此时我们可以使用无状态的**领域服务**（Domain Service，7），如图G.5所示。



图G.5 领域服务执行特定于领域的操作，其中可能涉及到多个领域对象

领域事件（Domain Event，8）表示领域模型中发生的重要事件。有多种方式可以对领域事件进行建模。在对聚合进行命令操作时，聚合本身将发布领域事件，如图G.6所示。



图G.6 领域事件可以由聚合发布

我们通常忽略了**模块（Module，9）**，但是正确地设计模块同样是重要的。简单来讲，我们可以将模块看成是Java中的包或C#中的命名空间。请记住，如果只是机械式地设计模块，而不是根据通用语言，那么我们将得不偿失。模块中包含的领域对象应该是内聚在一起的，如图G.7所示。

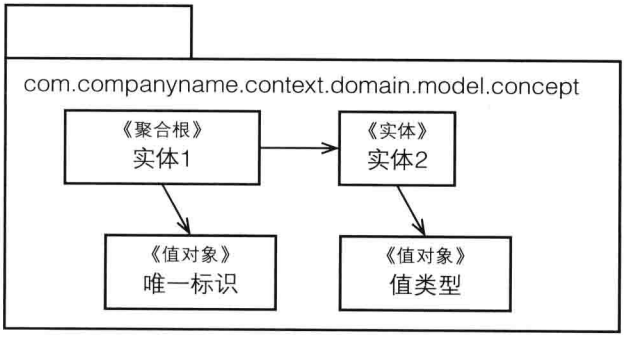


图 G.7 一个模块包含并组织内聚在一起的领域对象

好吧，现在来熟悉一下“牛仔的逻辑”幽默片段，以下便是一则：

牛仔的逻辑

AJ：“不要担心你的嘴巴包不下这块大肥肉，你的嘴巴可比你想象的大多了。”

LB：“你说的是‘头脑’吧，J。你的头脑比想象的大多了才对。”

目录