Advanced J2EE Platform Development

Applying Integration Tier Patterns

J2EE 平台高级开发

——应用集成层模式

- 使移植项目标准化、简单化
- 在传统系统中应用基于模式的J2EE应用程序体系结构
- 构建集成层，以使J2EE和传统组件相互独立
- 加强传统系统的移植能力，提高可复用性，避免不必要的重复设计和编程工作

Sun microsystems

(瑞典) Torbjörn Dahlén
Thorbiörn Fritzon 著　陈菊明　孟浩文　译

Java™ 2 Platform, Enterprise Edition Series

Pearson Education

清华大学出版社

J2EE 平台高级开发

——应用集成层模式

(瑞典) Torbjörn Dahlén
Thorbiörn Fritzon 著
陈菊明 孟浩文 译

清华大学出版社
北 京

EISBN: 0-13-044912-1

北京市版权局著作权合同登记号 图字：01-2003-8792

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。

图书在版编目(CIP)数据

J2EE 平台高级开发——应用集成层模式/(瑞典)戴林(Dahlén,T.)，(瑞典)弗雷泽(Fritzon,T.)著；陈菊明，孟浩文译.—北京：清华大学出版社，2004

书名原文：Advanced J2EE Platform Development Applying Integration Tier Patterns

ISBN 7-302-08640-0

Ⅰ.J… Ⅱ.①戴…②弗…③陈…④孟… Ⅲ.JAVA 语言—程序设计 Ⅳ.TP312

中国版本图书馆 CIP 数据核字(2004)第 044453 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：杜一民

封面设计：康 博

版式设计：康 博

印 刷 者：北京鑫丰华彩印有限公司

装 订 者：三河市金元装订厂

发 行 者：新华书店总店北京发行所

开 本：185×230 **印张**：11.25 **字数**：194 千字

版 次：2004 年 6 月第 1 版 2004 年 6 月第1 次印刷

书 号：ISBN 7-302-08640-0/TP·6195

印 数：1～4000

定 价：28.00 元

前　言

本书综述了我们用 Java2™ 平台企业版(J2EE™)开发的一系列应用项目，这些项目是我们在 1998 年到 2001 年间为 Sun 公司的瑞典客户开发的。

在这些项目的开发过程中，我们意识到，传统的系统集成是一种很特殊的操作形式，同时许多面向对象的应用程序开发项目都没有使用域模型。

以上因素是我们写这本书的初衷。目的在于与读者分享来自于实际项目中的实践经验和技术。希望读者能够从本书中受益并且进一步完善本书思想，也希望未来的基于 J2EE 平台的应用程序开发和传统的系统集成不再采用非正式方法，而是更加牢固地以上述思想和最好的实践经验为基础。

这本书没有提供任何固定的解决方案或者是一劳永逸的方法。在基于 J2EE 平台的应用程序中集成传统系统是一项艰巨的任务，需要对传统的应用软件和面向对象设计有全面的理解。我们已经找到了一种可行的方法，可以在现实环境中使用 J2EE 集成层模式(Integration Tiers Pattern)，这是值得读者关注的一种主要设计原则。本书描述的一些设计模式已经应用在实际产品中，还有一些设计模式只是理论，没有经过全面的测试。但是我们依然希望，读者在阅读本书的过程中能够获得可以应用到今后的应用软件集成中的实用工具。

客户经常提出这样的需求：减少对主机的依赖性，把基于 J2EE 平台开发的应用软件与不同的传统系统快速集成在一起，于是我们创造了一个术语："应用软件在企业信息系统中的可移植性(Application Portability Across Enterprise Information Systems)"。我们认为基于 J2EE 平台开发的软件应该是"可移植的"，即能够简单地集成到新的系统中，需要改动的 JAVA™ 代码尽可能少。我们相信如果读者应用本书所阐述的思想，就可以在开发中实现"可移植性"。

本书描述的所有设计和解决方案都来自于实际的客户需求。本书展现了在银行、电信等各行各业中，成千上万的 J2EE 开发者每天需要面对的一种现实情况。我们衷心地祝福他们，希望他们为连接现代技术和传统世界做出最好的努力。

致谢

我们感谢为本书的出版作出贡献的朋友和同事：Sun 公司的优秀工程师 John Crupi 以及他的同事 Danny Malks 和 Deepak Alur，他们编写的《Core J2EE Patterns》一书为我们指明了写作方向。感谢 Sun 公司的 JAVA 设计师—— Patricia Petterson，他对本书提供了大量有价值的反馈和建议。非常感谢 Sun 出版社的 Rachel Borden 对这本书的信任以及 Sun JAVA 中心 Stu Stern 和 Staffan Calais 的支持。

我们同时感谢 Sun 公司的其他人员，他们也不同程度地参与了本书的创作：Oliver Corbun，Milena Volkova，Cori Kaylor，Renato Ribeiro，Mike Alread，Sun 实验室的 ACE 团队和其他许多人。

感谢 Prentice Hall 的工作人员：Greg Doench，Brandt Kenna 和 Eileen Clark。

特别感谢曾帮助过我们的客户：Anneli Axell，Leif Forsberg，Jerzy Kawa 和 Björn Hellström，他们提出了很多需要解决的问题。同时非常感谢我们的朋友——Gunnar Bråding，他提出了许多伟大的思想。

最后，感谢我们的家人和朋友，没有他们的支持就没有这本书的出版。

内容简介

J2EE 提供了一种集成方法，企业利用这种方法可以把运行在低并发量的客户/服务器环境中的传统软件封装起来，以集成到高并发量的多层次的 Internet 环境中。但是，如何合理地封装传统系统，并把它们运用到 Internet 环境中，这个问题变得愈发重要。这本书解决了这个问题，并且讨论了一些能够使得封装处理更加详尽和有效的方法和技术。利用集成层可以让用户只注意传统系统部分的属性和需求，同时保护软件的 J2EE 部分。

在多数场合中，术语“可移植性(Portability)”指的是软件在不同的硬件和操作系统中迁移的能力。但是在这本书中，“可移植性”是指在传统系统的基础上开发的

应用程序无需经过大量重新设计和编码就可以迁移到其他数据源的能力。对于“可移植性”的这两种定义来说，标准化是关键，封装处理的标准化可以使“可移植性”在更多方面体现出优势。

封装传统系统的任务艰巨，需要全面地理解两个领域：面向对象的 J2EE 领域和传统系统领域(例如 COBOL/主机环境)。如果封装系统是基于一个定义良好的处理过程，就可以在常规软件开发项目之外进行封装工作。如果某一个特定软件开发项目没有时间限制，封装处理就能够按照固有的合理步骤进行，为随后的软件开发项目提供一个牢固的基础，在这个基础上的项目开发可以节省大量的开发时间和投入。

这本书主要针对业务处理和系统分析员、架构师和设计者，他们都在基于 J2EE 的平台上进行软件开发，具有集成传统系统的需求。

读者如果具有以下的知识：面向对象、 Rational 统一过程、J2EE、主机系统、事务处理监听，那么阅读本书将会有更大的收获。

动机和背景

通过不断地兼并和收购，企业会不断地成长和变化。某一个子公司开发和使用的软件最终会被另一个子公司使用，这时问题就出现了。假设两个子公司在独立的传统系统上运行各自的 IT 系统，兼并和收购后经常出现的情况是：由于其中一个子公司使用的传统系统通常提供不同的服务和信息结构，因此这个子公司为了能够使用另一个子公司的 J2EE 软件而必须做大量的重新设计和编码工作。没有对这些传统系统的合理封装，集成和运行在不同系统上的 J2EE 软件将需要重新设计和编码。

本书提供了一种封装传统系统的方法，这种方法运用域模型策略，简单但是功能强大。使用这种域模型策略不仅可以帮助 J2EE 软件移植到任何已经存在的企业信息系统，而且可以在这些系统中随意地改动 J2EE 软件。这种策略包括以下一些概念：

- 为企业创建域模型的渐进式开发方法。
- 基于 J2EE 的体系结构，用于封装传统的系统。
- 基于组件的业务规则框架。
- 对象查询语言语法分析器和传统系统调用生成器。

- 用来生成实现封装体系结构的 JAVA 类的描述工具。
- 有助于实现封装体系结构的体系结构和设计模式集合。

基于域模型策略的传统系统封装使得封装过程独立于软件开发过程。每个传统系统都可以封装成一个域对象层，从而使 J2EE 应用程序与传统系统中的代码相对独立。所以，只要提供了一个域对象层，就可以对基于这个域对象层的 J2EE 应用程序进行部署。

全书概述

第 1 章解释了如何开发一个用于捕获企业日常业务实体的域模型。这个在书中称作通用域模型(Common Domain Model)的模型是设计对象层的基础，这个对象层是 J2EE 应用程序的传统系统集成层。

第 2 章介绍了如何从一个域模型生成一个设计模型，同时描述两者之间的追溯关系，以及设计应用程序的通常准则。本章重点介绍连接 J2EE 应用程序和传统系统的集成层。

第 3 章讲述集成层的实现，讨论如何把它集成到 J2EE 连接器体系结构(J2EE Connector Architecture，JCA)。

第 4 章通过 3 个传统系统的实例，具体讲述了实现一个可重用集成层的步骤。将银行的一个微观域模型作为例子贯穿整个章节。本章具体描述传统系统实例中的事务如何映射为基于域模型的传输对象。

第 5 章提供了一个示例，基于第 4 章讲述的通用域模型和集成层的方法，讲述如何设计和实现 J2EE 应用程序。

附录 A 包括一个模式目录，对本书中出现的模式进行全面描述。

附录 B 剖析了 3 个传统系统的实例。这些系统用 IBM 信息管理系统(Information Management System，IMS)作为数据库，IMS 是银行应用软件(比如 OS/390)的流行事务监控器。

目　录

CHAPTER

1

域模型的创建

本章主题

- 创建通用域模型
- 使用 UML 设计通用域模型的类图
- 维护这个通用域模型

域模型描述企业的核心业务中的信息实体，以及它们的属性和相互之间的关系。例如，客户、方案和产品等概念都是相当普通的信息实体，它们在信息系统中经常被使用，并把传统的系统和数据库作为自身的物理表示形式。那些在整个企业中共享的模型，作为一致的需求规范和软件复用的基础，特别地，这个模型：

- 提供了企业核心业务的数据视图，这个视图对于需求提供者和应用程序开发者来说都是一致的。
- 提供了一种从实现阶段追踪需求阶段的方法。

本书将这样的共享模型称作通用域模型(common domain model)。

通用域模型为业务处理分析人员和应用程序开发人员提供了统一的词汇表，这个词汇表可以用来陈述需求和实现设计。如果在应用程序开发的整个过程中——即需求描述、分析、设计和实现——这些词汇都保持一致，那么从实现阶段到最初需求阶段的追踪才能够实现。这种方法可以提高产品质量，同时使维护变得更加容易。这个模型建立后可以作为整个开发过程的一个基本组成部分，以用于企业当前和今后的应用程序开发项目。

本章将描述如何创建一个涵盖企业核心业务实体的通用域模型。此类模型最好采用循序渐进的开发方法，因为对于这样一种重大的革新方式，企业核心业务中的每一个实体、关系和业务规则都需要定义、讨论达成一致意见、最后成文，此过程可能历时数年。

1.1 创建通用域模型

业务对象模型把一个企业的活动、员工和实体描述成业务用例(Business Use Case)的实现(The Unified Software Development Process, I. Jacobsson, G. Booch, J. Rumbaugh, Addison-Wesley 1999)。这个模型是指导企业应用程序结构的需求基础。业务对象模型中的实体代表了可以由用例操作的一类“事情(Things)”。这些“事情”在传统系统的数据库中通常采用结构化描述。在应用程序的开发过程中保留实体的含义可以产生一种机制，利用这种机制可以从实现阶段跟踪到最初需求阶段。

为使业务实体受到重视，应该同时在词汇表和类图中表示它们。类图只关注企业的实体，不关注企业的员工和活动，通常把类图视为一个域模型。域模型提供了在用例中描述应用程序功能需求的名词。此外，包含域模型的类图还是分析和设计模型的基础，用这些模型可以描述用例的实现。接下来将描述把域模型转化为设计模型的过程。

通用域模型包含了核心实体以及它们的属性和关系，这些核心实体通常用于由不同部门和子公司共同开发的应用程序。为了让新的应用程序能够适应传统系统和数据库在底层的改变，域模型必须独立于这些系统的服务和数据记录。因为这个模型应该拥有尽可能广泛的适用性，在这个模型中定义的所有元素都应该被整个企业接受和认可。这个模型中描述的实体、属性和关系必须在所有的应用程序中有同样的解释。

因此，通用域模型通常没有包括应用程序需要的所有实体、属性和关系。因此特定应用程序的域模型，不仅要包括通用域模型中实体、属性和关系的子集，也要

包括一些应用程序所特有(或者部门所特有)的实体、属性和关系。第 4 章将描述专用域模型和通用域模型集成的过程。

注意，创建一个适用于整个企业的通用域模型可能需要几年的时间，而且面临着这样的风险——在应用程序开发项目开始应用这个模型之前，这个模型已经过时了。为了尽快建立一个通用域模型，重要的是让业务过程分析人员和应用程序开发人员尽快地认可和使用这个模型。域模型最好是逐渐完善的，并且是在许多应用程序开发项目的参与者的协助下创建的。

通用域模型的第一次改进应该以当前大量关键项目使用的对象和数据模型为基础，同时这些项目的分析者、架构师和开发者应该参与最初的改进开发。为了使这些人的价值最大化，重要的是使他们在开发的项目中因应用该模型而立即获益。

收集大量对象或数据模型后就可以进行比较。为了找到相似点和通用概念，每个概念的含义必须一致。在不同的对象和数据模型中，相同的概念可能有不同的名称，或者不同的概念有相同的名称。因此每个项目的负责人需要详细解释项目中的模型和用到的概念，这样可以精确地比较这些模型。

在进行上述比较的过程中，应该在几个项目的对象或数据模型中找出实体。这些实体都是通用域模型的候选项。选定一些通用实体后就可以创建通用域模型。这个模型应该包括以下 3 个组成部分：

- 类图：类图包括实体、属性和关系，这些可以用于所有的备选项目中。这个模型的所有类必须能被许多不同的应用开发项目中的对象或数据模型所追踪。
- 词汇表：包括所有实体、属性和关系的文本描述。
- 业务规则目录：这个目录包括企业内所有的基础业务规则，这些规则涉及通用域模型的实体、属性和关系。

创建一个企业范围的域模型通常很困难，需要很长时间。但是以下的章节将介绍一种快捷方法，可以在相对较短的时间内创建一个通用域模型。这种快捷方法是一个递增的过程，即开发这个域模型需要几个步骤，每一次的改进都会使模型逐渐完善。最初模型的开发需要由一组业务过程分析人员和应用程序开发人员完成，书中把这些分析人员和开发人员组成的开发小组称作“通用域模型工作组(Common Domain Model Workgroup)”。

1.1.1 通用域模型工作组

建立一个通用域模型需要许多业务领域的专家、架构师和设计者的共同努力。

为了协调组成员的知识结构，保证组员就业务领域达成一致意见，通用域模型开发小组应该考虑整个企业的全局利益，而不是某个业务领域的利益。

开发小组的首要任务是保证通用域模型的策略已经在企业内建立，所有的业务领域都能够理解这个模型的重要性，并且从中受益。其次，说服人们为模型的创建作出贡献非常关键。工作组必须负责以下事情：

- 向所有的业务领域宣传通用域模型的概念和模型建立的基本原理。
- 安排业务领域的代表加入工作组。
- 对所选的业务实体给出一致的定义。
- 标识参与的项目，并将每个项目对象和数据模型进行组合。
- 找到核心业务实体的通用命名器模型。
- 创建通用域模型的组成部分。
- 建立这些组成部分的维护过程。

因为创建通用域模型不仅需要业务领域的知识，还需要对象建模技术，所以开发小组的工作人员应该包括：业务过程分析人员、业务设计者、IT 架构师和应用程序设计者。这个开发小组(或者它的子集)最终应该形成一个维护小组，负责以下工作：更新通用域模型的组成，支持应用程序的项目开发，管理各个业务领域和开发项目对模型所做的努力。

当企业上下都认识到通用域模型的重要性，就应该成立一系列的工作组。这些工作组应涉足到企业中各个适合使用模型的业务领域。这些工作组的参与者应该做以下事情：

- 标识 3～10 个通用实体，这些实体可能出现在许多不同的应用程序中，涉及到不同的业务领域。
- 对所选的实体给出一致的定义。
- 选择参与的项目。

当选定实体，并且给出它们的一致定义以后，创建类图、词汇表和业务规则目录的最初版本。

应用程序开发项目应该有固定的对象或数据模型，以及业务词汇表，这些是第一次改进的基础。实现这次改进有一点很关键：每个项目的对象或数据模型都应该包括大多数由工作组定义的那些通用实体。

检查这些项目的模型，会发现在两个或多个模型中有许多实体是这些应用程序的通用命名器。应用程序的通用命名器可以标识适用于大多数应用程序实体。所以，

一些选定应用程序的通用命名器构成了最初的通用域模型。

1.1.2 类图的元素

为了加深理解和复用，构成最初的通用域模型的通用命名器应该用类图记录(最好采用 UML 表示法)。使用类图是为了明确地表示企业中的核心业务实体，因此类图必须容易理解，甚至应该能够让那些不熟悉对象模型的人也能阅读。

不仅 IT 架构师和应用程序设计者会使用到类图，不熟悉 UML 的人也要使用类图，因此类图不应包括任何复杂的类和关系结构，否则那些缺乏面向对象建模技术的人会对此产生歧义。类图应该使用直观并且一致的方式，表述实体、属性和关系。

易于理解是通用域模型的关键属性。如果一个通用域模型很难理解，则失去了使用的意义。使用下列指导原则，可以简单一致地设计类图：

(1) 实体

实体可以表示成一个类构造型：<<entity>>。因为实体通常是稳定的，生存周期长于任何应用程序，所以一个实体应该有一个惟一的标识符。没有惟一标识符的实体可以映射到一个类构造型<<utility>>(如下面介绍的一些实体)。

一般应该避免使用优选法，比如把许多实体常见的属性组合到一个单独的实体中。在域模型中，继承只能用来描述类型等级，类型无关的实体不能有继承关系。

(2) 类型

如果一个实体能够划分为不同的类型，则为每种类型定义成一个类构造型<<type>>，然后让这些类成为该实体的派生类。也就是说，域模型可以提供一个更好的办法，即把类型都表示成类，而不是给实体赋予类型这一属性。

(3) 关系

一个关系通常表示两个类的关联。但是因为关系具有属性，所以经常需要引入一个关联类。假设有一个实体 Party 和另一个 Party 有 Customer 的关系。这个 Customer 关系具有很多属性，比如客户标识符、期望利润和客户类型等。这时就应该用一个关联类来表示 Customer 关系，定义成<<entity>>，它具有客户标识符和期望利润两个属性，不同的客户类型应该是这个关联类的子类。

在很多情况中，相同的关系在不同的域模型中可能有所区别。比如，一个 Party 通常与其他 Party 有 Customer 关系，但是一个应用程序也许会对此加以限制——每次只能有一个 Party 和这个 Party 有关系。这时，通用域模型应该使用最常用的属性。在这个专用域模型中，这个关系应该通过关联类关联一个方法，这个方法可以是一