JAVA 技术丛书

J2EE应用与实践技巧

——Java设计模式、自动化与性能

J2EE Best Practices

Java Design Patterns, Automation, and Performance

[美] Darren Broemmer 著

于洁 李稳 韩国栋 等译

WILEY

電子工業出版社
Publishing House of Electronics Industry
http://www.phei.com.cn

Java 技术丛书

J2EE 应用与实践技巧

——Java 设计模式、自动化与性能

J2EE Best Practices

Java Design Patterns, Automation, and Performance

[美] Darren Broemmer 著

于 洁 李 稳 韩国栋 等译

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

内容简介

J2EE技术正在成为开发基于Internet和事务处理的业务应用程序的一种广泛使用的平台技术。在J2EE项目实施过程中，需要应用健壮的应用程序设计模式。本书从开发者的角度提出了一种称为参考架构的应用程序结构，主要包括业务对象架构、基于服务的架构和用户交互架构这3个组成部分。这种设计模式可以适应动态变化的业务和用户需求，书中详细介绍了如何利用组件技术实现这种应用程序结构，同时也介绍了如何高效地开发J2EE组件并将其集成到应用程序中，展现了一系列J2EE应用开发的实践技巧，其中包括如何评价和选择适当的软件组件以及服务。全书从原理和技术角度分析了影响系统总体性能的原因以及解决方案。利用贯穿全书的银行应用程序，读者一定会对所有概念和实践技巧有更深刻的理解。另外，本书还讨论了有关应用程序如何集成诸如Jakarta Struts等第三方技术、安全性、性能工程以及重用的问题。

本书可以作为J2EE开发人员的参考用书，也可以为J2EE项目的管理人员、学习J2EE有关内容的读者提供很有价值的帮助。

Darren Broemmer: **J2EE Best Practices: Java Design Patterns, Automation, and Performance.**
ISBN 0-471-22885-0

版权贸易合同登记号　图字：01-2002-5358

图书在版编目（CIP）数据

J2EE应用与实践技巧——Java设计模式、自动化与性能/（美）布罗默（Broemmer, D.）著；于洁等译.
-北京：电子工业出版社，2004.6
（Java技术丛书）
书名原文：J2EE Best Practices: Java Design Patterns, Automation, and Performance
ISBN 7-120-00087-X

Ⅰ.J...　Ⅱ.①布...　②于...　Ⅲ.JAVA语言-程序设计　Ⅳ.TP312

中国版本图书馆CIP数据核字（2004）第050521号

责任编辑：马　岚
印　　刷：北京兴华印刷厂
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
经　　销：各地新华书店
开　　本：787 × 980　1/16　　印张：27.25　字数：549千字
印　　次：2004年6月第1次印刷
定　　价：45.00元

凡购买电子工业出版社的图书，如有缺损问题，请向购买书店调换；若书店售缺，请与本社发行部联系。联系电话：（010）68279077。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。

译者序

随着Internet应用的不断发展和下一代分布式计算模型Web服务的出现，J2EE技术正在成为一种开发基于Internet和事务型应用程序的通用平台技术，开发人员可以在此平台上建立灵活且可重用的组件和应用程序，因此它在过去几年里一直引领着企业应用的潮流，使得企业应用系统的开发变得更加迅速且便捷。J2EE本身是一个标准，它为不同厂商创建平台产品提供了标准，使不同J2EE平台产品之间的交互成为可能。

使用J2EE进行应用程序开发应当仔细考虑平台提供的服务以及应用程序组件如何能够最好地利用这些服务。本书为高效地构造J2EE组件并将其集成到应用程序中提供了许多实践技巧，其中包括如何评价和选择正确的软件组件集合和服务。本书给出了J2EE软件开发方法和使用参考架构来构建应用程序的实践技巧，首先由业务应用程序的概念开始，分别定义了作为参考架构重要组成部分的业务对象架构、基于服务的架构和用户交互架构的概念及相关内容，从原理和技术上分析了各部分的相关性和实现方法，随后给出了相应的应用程序架构的实现和实例应用。并且用一个银行应用程序的例子贯穿全书，形象地说明了J2EE的实际应用。在讨论了架构以及许多实践技巧之后，在书中的最后部分又更加深入地讨论了应用程序的安全性、性能和重用问题。

本书提供了关于如何快速并高质量地建立应用程序的概念和实例，对于每个使用J2EE建立业务应用程序的Java技术人员都将会有所帮助，可供计算机专业的大学生、研究生，以及从事软件开发和电子商务开发的软件工程师和网络工程师学习与参考。

参加本书翻译工作的有于洁、韩国栋、李稳、王勇、黄瑶、常鸿、曹勇刚和李诺等。由于译者水平有限，译文中难免有不妥之处，恳请读者指正。

前言

Java 2 企业版（J2EE）技术正在成为开发基于 Internet 和事务处理的业务应用程序的一种广泛使用的平台技术。它提供了一个健壮的开发平台，在此平台上可以建立灵活的、可重用的组件和应用程序。J2EE 技术是一个强有力的标准，由于它提供了诸如 HTTP 请求处理（Java servlet API）、事务管理（Enterprise JavaBeans）和消息（Java Message Service）等许多基础服务，因此非常适合基于 Internet 的应用程序。但是，J2EE 也是一个复杂并不断变化的标准，技术人员要考虑很多设计决策和性能问题，必须要考虑到每个组件服务在应用程序处理上增加的一定程度的开销。另外，也必须为每个组件和应用程序设计并开发一些公共的业务逻辑功能，例如错误处理等。

使用 J2EE 进行的应用程序开发应当仔细考虑平台提供的服务以及应用程序组件如何才能最好地利用这些服务。本书为高效地构造 J2EE 组件并将其集成到应用程序中提供了许多实践技巧，其中包括如何评价和选择正确的软件组件集合和服务。这与其他行业的工作没有什么不同，木匠和钢铁工人也都使用架构计划来制造东西，只是他们为完成工作而使用的工具完全不同。建立在 J2EE 上的可伸缩的、模块化的架构将由适当的、结合了自定义公共业务逻辑功能基础的 J2EE 服务集合组成。

本书及相关技术的概述

本书给出了一系列 J2EE 软件开发的实践技巧，然后使用这些技巧构造了一个应用程序架构方法（即参考架构）。参考架构提供了使用 J2EE 技术快速构造事务型业务应用程序的基础。参考架构的设计和实现基于一系列用于优化和自动化 J2EE 开发的指导原则。

参考架构的指导原则

构造参考架构的目标是创建一个开发环境，在此开发环境中能够快速建立应用程序并且能够让应用程序具有良好的性能、可靠的质量和可重用性。这些指导原则有：

- 将已证实的设计模式应用到 J2EE 中
- 自动完成公共功能
- 使用元数据驱动的、可配置的基础组件
- 考虑性能和可伸缩性

这些原则对于促进架构发展和建立开发的基础而言是必要的。我们将在全书中详细讨论这些概念，也会在 J2EE 架构的每一部分中应用它们。可以使用这些概念和它们的公共基础逻辑形式的实现来优化和自动化多数普通软件的开发以及特定的 J2EE 开发。为了能够提出一个解决方案，在快速开发、加速性能、提升质量和增加可重用性这些需求之间求得平衡，在设计架构和应用程序组件时进行可靠的分析是非常重要的。

图 1 给出了架构的输入和输出。该图基本上说明了指导原则以及在应用程序的开发中使用架构的优点。

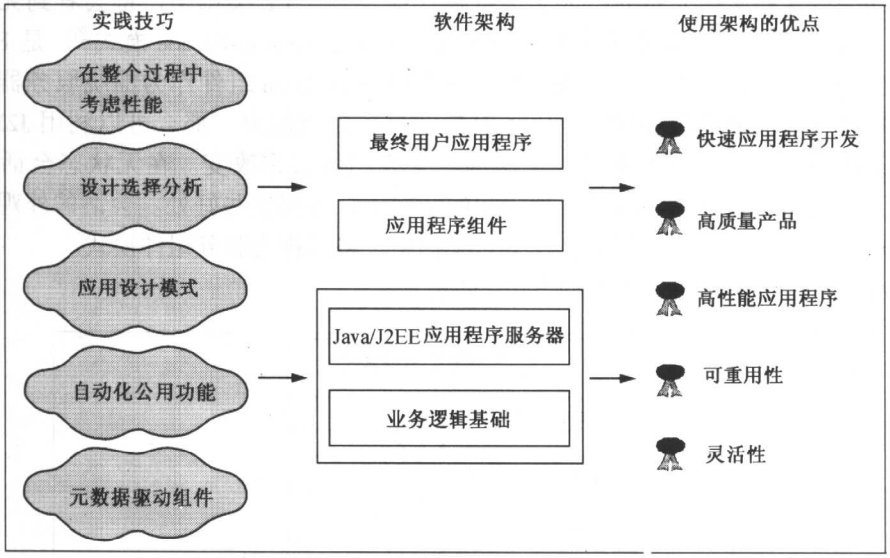

图 1　架构原则及其优点

这些原则是利用 J2EE 开发应用程序的动机和基础。我们将对 J2EE 企业架构中的每个方面进行研究，研究它们的行为和特性。通过了解它们的特性并使用开发原则和实践技巧，可以创建一种能够有效使用此技术的方法，达到应用程序开发的目的。

图 1 中右侧的目标，例如灵活性和可重用性，应当在每个软件开发计划一开始就考虑和表示出来。我们已经在两种不同的层次级别上实现了这些目标：前面描述的软件架构层和应用程序组件设计。参考架构将指导应用程序设计的大部分工作，因此在企业的软件开发开始前就理解和区分这些层次级别是很重要的。这两种层次级别中的每一种都将为终端用户和开发组织提供不同程度的好处。

应用已证实的设计模式

设计模式是一个定义好的对象交互，使用它来解决软件开发中重复的问题。现

在已经有许多文档化的设计模式[①]，提供了各种已证实的方法，能够用面向对象技术来解决通用问题。同样，我们可以把其中的一些模式应用到 J2EE 架构中。例如，在基于服务的架构中的服务的概念。参考架构中的服务组件层类似于 Facade 和 Mediator 模式（Gamma et al. 1995）。服务器组件为客户端提供了一个简单的接口，将表示组件从后端业务逻辑组件中分离出来。这有利于提高可重用性和简化客户端的视图。如果为服务组件增加一个标准接口，就能够从一个前端组件实现 Command 模式（Gamma et al. 1995）。这样做能够在前端建立一个通用的、可配置的控制器组件，调用标准的后端服务。

如果将这些文档完备的已证实设计模式应用到 J2EE 架构中，将会看到无状态会话 Bean 是基于服务架构的一个完美实现。这就是 Session-Facade 模式[②]，是 Facade 模式应用于 Session EJB 上的实现。如果只是把会话 Bean 组件作为围绕服务器对象的包装，以增加分发服务和管理服务对象的事务处理能力，那么可以利用 J2EE 的基于组件的服务，而根本不需要对面向对象视图做很多改变。在无状态会话 Bean 情况下，也可以在不增加很多处理时间的情况下，得到这些好处。会话的外观是服务实现对象的 EJB 组件的包装，在参考架构中，将其称为服务组件模式。

图 2 描述了服务组件模式的 UML 表述。

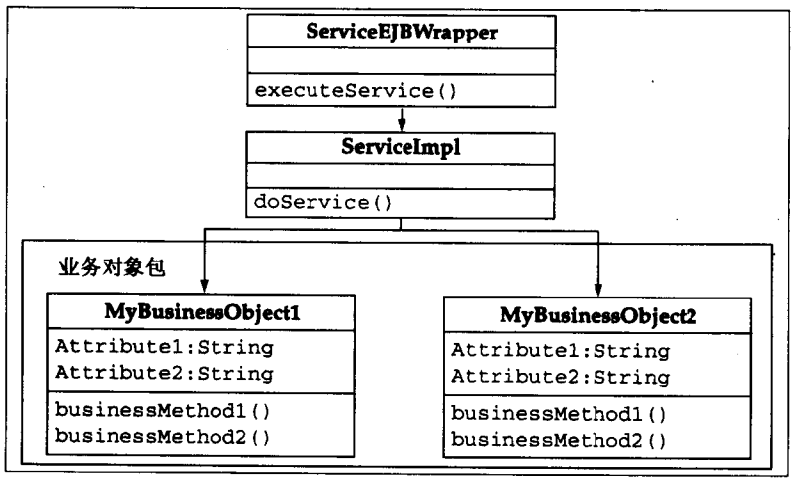

图 2　服务组件模式的 UML 图

业务对象和表示层组件也有大量可用的已被证实的设计模式的例子。模板方法（Template Method）模式提供了优秀的机制（Gamma et al. 1995），为业务对象和服务对象提供可扩充的基础组件。在业务对象中，为公共的操作（例如 save 操作）提供

① E. Gamma, R. Helm, R. Johnson, J. Vlissides, 1995. *Design Patterns*. Boston, MA: Addison-Wesley.

② D. Alur, J. Crupi, D. Malks. 2001. *Core J2EE Patterns*. Mountain View, CA: Sun Java Center.

了模板，用以把对象数据永久保存在数据库中。基类或者模板，为子类和特定的业务对象提供了钩子（hook），以实现验证规则并预先保存或以后保存逻辑。EJB 使用了很多已经在 Java 语言中应用的设计模式。它们中的一些是业已存在的 Java 接口模式的变形，例如实体 Bean。实体 Bean 必须实现 javax.ejb.EntityBean 公共接口，提供插入、更新、删除逻辑的连接。已经讨论的每个架构层都建立在业已存在的模式之上，并且在 J2EE 之上也参照了其他一些软件架构内提供的灵活性和可重用性的模式。

自动化公用功能

自动化公用功能的方法有很多优点：

- 时间不会被浪费在单调的、容易产生错误的任务上。
- 是一个通过良好测试的高质量软件。能够完成功能而所需的代码较少，并且能够处理每个请求。大多数问题的处理基础实际上就是具有输入的黑盒处理。
- 自动化功能及其公用接口，使得开发和维护整个应用程序的软件一致性更加容易。

即使有很多易于使用的 API，例如 Java servlet API，在应用程序中还是要写许多功能函数。例如，处理用户表单提交是业务应用程序的一个公用功能。对于每个请求，都需要从 HttpServletRequest 对象中读取出提交表单中的数据，打包成一定的数据结构，发送到所请求的服务中或者后端的功能模块中。一种替代方法是，可以开发一种自定义的 servlet 或者 JSP 来处理所有页面的请求。这种方式并不是非常有效，因为在典型的业务应用程序中，表单数目非常多。我们可以发现，每个表单的请求都有一定的相似性，甚至在其中有一些公用的代码块。可以采取的替代方法就是抽取所有表单的基本处理流程，之后把它封装到一个 servlet 中，为所有含有表单的 Web 页面所公用。使用配置服务可以定义每个表单、它的输入数据以及处理请求的服务。几乎任何重复的功能或操作都是自动化处理的候选。本书研究事务处理的 Web 应用程序的本质，以定义能够自动化的一系列公用功能。事实证明，因为 Web 应用程序和 J2EE 应用程序架构的本质特征，每个应用程序都应当实现这样一些公用功能。一系列可配置的、实现这些公用功能的基础组件将会增加建立在参考架构上的应用程序功能的质量和数量。本书讨论了一系列公用功能并将它们应用到 Java 平台上，对基础层的附加需求不再讨论。为使应用程序具有可伸缩的模块化结构，要求我们完成一些有价值的基础工作。

通常把一组可配置的、能自动完成应用程序基本功能的基础组件称为一个框架。基于上面的设计准则，很多这样的基础组件都可以使用面向对象的设计模式来实现。这些框架组件和模式组成了用于快速开发高质量 J2EE 应用程序的参考架构。正如许多开发人员都了解的，开发特定应用逻辑所需的服务数量与开发平台已经提供的服务数量之间存在着差距。软件层，即本书中所指的业务逻辑层，会尝试弥补

这种差距。Java 和 J2EE 平台可以解决和填平这个鸿沟。但是，虽然有很多人致力于为这个平台添加服务，这个鸿沟仍然存在。由于企业开发的复杂性，不同的企业和组织有大量的不同需求以及很多设计考虑事项，需要我们用大量的时间才能给出符合所有这些需求的标准。事实上，随着基础平台和标准的发展，技术和问题领域同时也在增长，因此，这两者之间的差距就像由曲线所表示的微积分方程一样慢慢地接近零，但实际上却永远达不到。

技术框架内的自动化能力为各应用程序之间提供了一种高度的可重用性。可重用性是面向对象软件开发的“标志”。然而在很多实际环境下，重用是很难做到的。开发人员使用战略应用程序架构和指导原则，就能够从软件重用中获得益处。EJB 标准正朝着使应用程序之间具有标准的、可重用业务组件的方向努力。它扮演了在 J2EE 之上能够使组件重用的应用程序架构的角色。使用应用程序架构是很重要的，这种架构让组件不用增加非常多的开销就可以集成到系统的其他部分中去。

使用为不同的接口和系统缓存的消息层，是一种接入不同组件的方法。在复杂的架构中，这是一种合理的解决方案，但对于简单的应用程序来说，它付出的代价却太高了。使用设计模式和自动化基础组件这两条原则，可以实现我们提倡的能够在领域内重用组件的两个主要策略。一个例子是基于服务的架构层，该层为基于过程的组件提供标准接口。通过建立用户表示层使用的标准接口，检索账户数据之类的服务就能被请求客户数据的不同页面重用。类似于账户存款和账户取款这样的服务在转账服务中作为一个构建块重用。架构中有一个服务层允许服务本身能够被不同客户端的设备重用。最后，可以通过参考架构中的配置基础层自动调用服务组件的标准接口。

使用元数据驱动的组件

通常我们把元数据定义为描述其他数据的数据。本书也用了“元数据”这个术语来指许多数据元素，这些元素定义了许多软件组件的属性和行为。例如，元数据可以是给定业务对象的属性列表和它们各自的数据类型，或者是 Web 页面的表单名字和相关的配置信息。定义这些组件的大量元数据出自用 UML 描述的设计模型。

使用元数据来驱动组件的规则也建立在先前的自动化软件开发规则之上。作为一个“框架”服务的输入，元数据可以自动完成并驱动 J2EE 组件的行为。在架构的各个层次都可以使用它。在业务对象中，可以用元数据来定义业务实体和它们的属性。在工作流或者事务层次上，可以用元数据驱动复杂任务的过程流。在用户接口层次上，元数据能够定义指定的 Web 页面表单以及应该如何处理表单。所有的这些应用程序元素都可以使用元数据来概括和定义，J2EE 规范本身就是用不同形式的元数据来配置和使用组件的。EJB 2.0 所采用的实现容器管理持久性的抽象方法就是一个极佳的例子。EJB 部署描述符包括了把 Bean 的属性映射到数据库表的元数

据，并定义了 Bean 与其他组件的每种可能的关系。

并不是每个过程或功能都必须用元数据定义。应该看到，这种方式还有一些缺陷，它并不是对每种情况都适用。与直接的代码行相比，要完成同样的工作，元数据抽象要增加额外的开销。在处理诸如单个数据库 I/O 请求之类的问题时，这个额外的开销很小，一般可以忽略。但在考虑软件开发方法时，尤其是对于那些事务吞吐量至关重要的应用程序，要考虑这些费用。

这种方式的另外一个潜在缺陷是它让代码的阅读和调试变得更困难。包含元数据的独立文件或存储库（repository）决定了代码流。关于这一点有很多争议，其中最主要的争论是，这些使用元数据的可配置的基础组件，对应用程序的其他部分来说是一个黑盒子，是严格测试过的组件。一旦组件工作正常，很少有人去看“框架”代码，应用程序的工作情况可简单地通过查看客户端代码和输入到服务中的元数据来判断。由于人们对于这些基础组件的持续使用，这个争论很快就消失了。另一个关于结构的争论是，对于书写得很好的面向对象的代码，由于方法一般都很小，读时不得不从一个对象跳转到另一个对象，才能弄清楚到底是怎么运行的。它很难让人从代码开始的地方有条不紊地往下读。在软件开发由编写大量程序代码转向面向对象开发时，也有同样的争论出现。通常来说，顺序的程序代码比面向对象的代码更容易读懂，但与这个微不足道甚至还未必是不利的因素相比，面向对象开发带来的好处要大得多。现在，同样的争论也出现在元数据驱动方式上。

和 J2EE 架构的很多其他方面一样，赞成者和反对者在对给定的设计决策做出选择之前都必须权衡利弊。这是因为可能会有很多种架构设计，解决方案通常采用一种中庸的方式，在其中将元数据用于关键组件，以得到最大好处。若业务应用程序的元素是数据密集型的，并且被频繁使用，如表处理和业务对象持久性，就要使用元数据来快速开发高质量的产品。

现在的发展趋势是以 XML 格式存储各种数据，元数据也不例外。以 XML 存储元数据有很多好处：

- 无论是在文件中还是在数据表中存储，XML 都提供了标准的格式。
- 很多设计工具能根据模型产生 XML 数据，现在很多工具支持 XMI（XML Metadata Interchange），这是对象元数据的一种标准 XML 格式。
- XML 能被多种工具创建和修改，这些工具包括 XML 编辑器和定制编写（custom-written）的工具，在很多情况下甚至可以用简单的文件编辑器。

使用元数据还有一个比较有趣的效果，它把应用程序的设计部分从编码中分离出来，这样做很有帮助，原因如下：

- 由设计来驱动大量应用程序功能，意味着应用程序的变更很少需要改变代码，这提高了维护周期和部署的速度。

- XML 支持模型驱动的开发方法，设计模型成为系统文档中精确的一部分，并用来生成应用程序组件或基本组件的元数据输入。
- 应用程序代码的很多输入可以从设计模型中产生。业务实体的对象模型包含了实体的属性和它们之间的关系。元数据可以从设计工具导入到 XML 中。XMI 规范提供了导入的格式，并且设计工具也开始支持这样做。如果可配置的业务对象的基类能管理业务对象的属性与关系，那么几乎可以完全通过设计过程，利用元数据输入使业务对象的此部分自动化。当然，特定的业务方式和其他应用程序组件也会修改这些业务对象的属性并创建新的关系实例，但是业务逻辑可以通过元数据驱动的过程自动完成。

性能和可伸缩性

最后一点（也是最重要的一点）是性能设计。到项目的最后阶段才考虑这个问题，肯定会带来严重的后果，尤其是在如今这种快节奏的互联网时代。如果一个系统性能太差，人们很快会对它敬而远之。业务应用程序的使用者习惯于私人网络上的客户机 - 服务器的性能，而消费者和互联网用户也不会耐心等待加载 Web 站点的页面。因此，尽管计算机运行速度越来越快，而且可以选择更多的硬件（如果有升级方案），在开发过程开始时，还必须注意性能问题。这是设计过程的一部分，因为它很可能涉及到与系统其他方面（通常是应用程序提供给使用者的灵活性）的权衡问题。

Java 语言本身能在很多方面达到 C/C ++ 的性能，甚至是在要求苛刻的应用程序中，它也被广泛认为是一种高性能的语言。这主要归功于 just-in-time（JIT）编译器的改进，它现在能迅速转换 Java 字节码并执行代码优化。在服务器端，由于很多类要多次执行，它的这个优点尤其明显。从理论上讲，代码转换成本地指令的开销并不大，可以忽略，所以多数代码与编译过的 C ++ 代码性能是相似的。与 C ++ 相比，Java 仍然存在的一个缺陷是垃圾收集过程，它增加了一些开销。然而，与编程时的好处相比，包括在内存分配和管理方面，开销微不足道，因此它并不是一个真正的问题。事实上，处理器的速度一直在提高，两种语言本身的差别正变得无关紧要。然而，J2EE 提供的组件服务在语言之上增加了一层，组件服务对应用程序整体性能的影响不容忽视。既然 J2EE 提供了很多有价值的服务，如对象持久性、命名及目录服务等，就一定要斟酌它们带来的好处和开销。

在很多案例中，都采用 Enterprise Java 服务作为解决方案，它有很多优点，但也并不是一个整体的标准。在软件结构中通盘采用企业组件只是建立 J2EE 架构的一般倾向。关键的例子是使用实体 Bean，相对而言，实体 Bean 是相当重量级的组件，因此，当每个业务对象映射到数据库中的一行时，不应该在一个应用程序中建立各种业务对象模型。这样做会迅速降低应用程序的可伸缩性和可用性。对很多系统来

说，必须具有可伸缩的架构。本书将讨论在每层设计时应遵循的方针，在确定软件组件的基础及构建独立的组件时必须遵循这些原则。

本书的组织结构

本书由业务应用程序的概念开始讨论，随后给出了相应的应用程序结构的实现和实例应用。首先给出了参考架构方法和如何将此方法应用于 J2EE 技术。参考架构的三个基本层（业务对象、服务/处理和用户交互）都建立在底层之上，先从设计概念开始，然后进入相关的 J2EE 实践技巧讨论，最后用 J2EE 实现结束。每一层都采用通用开发基础方法进行讨论，除此之外还以一个贯穿全书的银行应用程序的例子给出了实践过程。在讨论了架构、实践技巧和实现之后，再回过头来更深入地讨论诸如应用程序安全、性能和重用的问题。

第 1 章介绍并讨论业务应用程序的公共特性。公共特性被抽象出来作为应用程序架构方法的基础。本章介绍了参考架构层并定义了每层内的组件，同时简要介绍了 J2EE 平台，将参考架构作为 J2EE 组件映射到它的实现上。本章提出了“模型 - 视图 - 控制器”架构模式（通常在 Web 开发中又称为 Model 2 方法），作为跨越 J2EE 技术和参考架构两者的模式。

第 2 章涵盖了参考架构中业务对象层的设计元素。本章以银行应用程序的对象模型为实例进行介绍，讨论了业务对象组件和在 J2EE 的实现操作中需要注意的问题。设计元素的讨论包括有状态与无状态的比较，实体 Bean 与常规的 Java 对象的比较，持久性机制以及事务并发。

第 3 章讨论了业务对象职责的前半部分的实现，包括属性管理、业务规则验证和处理错误情况。由于业务对象内功能的数量很多，我们将它们的实现分别在第 3 章和第 4 章讨论。这两章讲述了账户业务对象的显式实现，介绍了通用的属性管理方法，描述了可用于所有业务对象的元数据驱动基类的实现。另外，还介绍了业务对象的标准接口，从而可以普遍并一致地处理所有的对象；讨论了值对象和批量存取方法；介绍了错误列表机制和如何管理对象的一系列可配置的业务错误；讨论了常规错误和异常处理技术，以及它们在业务对象实现中的应用。

第 4 章讨论了业务对象职责的后半部分的实现，包括对象数据在数据库中的持久性、管理与其他对象的交互以及使用模板方法模式建立能够扩展和重用的业务逻辑模板。这里讨论的持久性操作包括 JDBC 的显式使用、元数据驱动的 JDBC 框架、第三方和开放源码持久性框架，以及实体 Bean 容器管理持久性。本章给出了每个操作例子以及它们的实现。通过构造业务对象工厂来抽象出业务对象的生命周期，给出了 JDBC，实体 Bean 和 Castor（一种通用的 Java 开放源码持久性框架）的实现。本章讨论了把对象集合服务作为将业务对象用于只读操作的一种更快的开发选择，

给出了在此种选择下使用 JDBC 的实践技巧；讨论了数据缓存和基于 JMS 的刷新机制，将其作为能够防止不必要的数据库 I/O 操作的可选方案；讨论了聚合的业务对象内容以及在标准业务对象接口上添加的相应方法；也讨论了参考架构的关键概念——模板方法模式以及可扩展性的自动化。第 4 章构造了保存模板、对象创建模板和聚合对象模板的实现方法，从而能够自动完成基本业务对象的功能。这里还讨论了所有元数据 DTD 以及它们的实现。在本章的结尾，读者将获得一系列设计概念以及可以快速建立健壮业务对象组件的代码。

第 5 章涵盖了设计元素和参考架构的服务组件层的基本原理，讨论了面向过程的对象的基本元素和需要考虑的实现选项。服务被分成数据更新和数据检索两类。这里还引入了将会话 Bean 作为常规 Java 实现类的组件包装的概念。本章主要讨论了服务组件的接口，选择标准接口的益处，以及对于不同数据结构（如 XML，值对象和参数列表）的处理方式。

第 6 章描述了参考架构方式中服务组件的实现，并给出了默认接口和标准接口的例子。为了创建标准服务接口，创建服务数据类时封装了值对象、参数列表和错误数据。本章构造了常规 Java 类的 EJB 包装的实现，介绍了用于标准错误处理、事务管理和调用实现类的服务组件基类，讨论了数据检索和更新服务的一般职责，构造了如 TransferFunds，ChangeAddress 和 GetAccountList 的银行应用程序服务的实现，也讨论了建立通用的可重用服务、在服务中调用其他服务以及使用控制器模式的策略。

第 7 章包括参考架构的用户交互层的设计元素和公共特征。基于 Web 的用户交互的关键特征被抽象为事件、动作、服务和 Web 页面。使用这些抽象和设计思路能够将控制器架构的核心内容分成 8 个步骤。这些步骤在一定程度上自动执行，也可以在控制器和动作类中被有效地划分。在这里也简单描述了 JSP/servlet 架构，讨论了其状态管理的设计思路。本章讨论了在 J2EE 中使用“模型 - 视图 - 控制器”架构的实践技巧，包括了管理会话大小、JSP 模板和使用可重用的自定义标记包装表示逻辑。本章的最后一节给出了 Jakarta Struts 项目的概述和 Model 2 架构的开放源码实现。这里讨论了 Struts 的控制器架构，利用 JSP 标记库可以很容易地在动态 Web 页面中集成请求处理功能。

第 8 章描述了如何使用 Struts 实现用户交互层，通过构造银行 Web 站点的实际例子来讨论并举例说明实现的各个方面。把构造更改地址的页面和查看账户的页面作为简单更新和数据检索功能的例子。把构造新客户向导作为多页面表单的例子。在这里还讨论了用户交互组件的实现策略，分别给出了事件对象和服务数据对象的实现操作，讨论并实现了参考架构中由前至后的错误处理。另外，还创建了一些自定义标记来举例说明可重用的表示逻辑的作用，它们与参考架构集成在一起，例如下拉标记能够自动从具体的对象缓存中获得数据。本章定义并讨论了在银行页面中

使用的 JSP 模板机制的实现，讨论和实现了为标准逻辑创建可扩展基础动作类。在本章结束时，读者将拥有一系列工具和设计概念，能够使用 J2EE 技术和业务逻辑基础迅速建立事务型 Web 站点。

第 9 章简要概述了 J2EE 中的应用程序安全性，并给出了它在银行应用程序中的使用。通过银行应用程序中的高级页面讨论了一些基于 Web 的应用程序的有趣设计特征。文中给出了一系列管理页面，介绍了表单上多个提交按钮以及同一表单上多个更新对象的实现策略。

第 10 章提出了一种解决性能问题的方法，这种方法能够在整个软件开发的生命周期中平衡开发重点问题。其中的一个重点是采用可扩展架构以及在确定解决方案前进行基准测试，目的是确定所提出方案的有效性。本章也讨论了度量和优化性能的策略，包括对象实例化、对象缓存以及如实体 Bean 等 J2EE 组件的使用。

第 11 章重点介绍了阻碍重用的常见障碍以及克服这些障碍的实践技巧。涉及的障碍包括社会方面的障碍以及技术上的限制等。J2EE 和参考架构都是重用架构的关键部分，这种重用架构基于可配置和可扩展性、标准接口的使用和分层模块架构。本章以参考架构的战略观点来考虑重用和适应性。

本书的读者对象

虽然架构设计者和各种不同技术程度的软件工程师也会在本书中找到设计思想、实现技术和有用的可重用代码，但是本书更适合于已经具有一些 J2EE 技术（例如 EJB 和 JSP/servlet）的人阅读。技术经理和其他信息技术专家也会在本书中找到许多对他们的工作有益的部分，例如有关安全、性能工程、重用以及战略架构等的章节。

由于本书提供了快速并高质量地建立应用程序的概念和实例，对于每个使用 J2EE 建立业务应用程序的技术人员都会有所帮助。许多市场上的 J2EE 书籍只提供了基本的 API 例子而没有深入讨论不同的 J2EE 架构的设计含义，也没有如何自动化 J2EE 组件开发的细节，而本书无论在理论上还是实践程度上都做到了这一点。

需要的工具

运行本书提供的示例应用程序和使用业务逻辑基础软件时，需要如下的工具：

- 任何兼容 J2EE 1.3 的应用程序服务器，例如 BEA WebLogic 6.1。
- Jakarta Struts 1.0 或更高版本，可在 http://jakarta.apache.org/struts 找到。
- （可选）Castor Data Binding Framework，ExoLab 项目的一部分，可在http://castor.exolab.org 找到。

网络资源

与我们合作的 Web 站点中包括了作为参考架构一部分而讨论到的业务逻辑基础的所有代码，包括了银行应用程序例子的代码，还包括了一些关于下列内容的链接，对应于相关的 Web 站点、开放源码项目和业界信息。

- J2EE 和 J2EE Blueprints
- Jakarta Struts 项目
- Jakarta Commons 项目
- Castor 项目
- 性能测试

小结

本章讨论了使用 J2EE 技术有效地建立 Internet 应用程序的一系列实践技巧的基础概念和原则。这些设计和开发指南促进了架构的创建过程，通过该架构可以快速开发具有良好性能、质量和可重用性的 Internet 应用程序。J2EE 提供了一个有力的标准，基于这个标准可以建立组件和应用程序。在正确的开发实践和软件资源下，基于 Web 的业务应用程序的开发更加接近于称为软件制造（即使用预制的组件和框架来建立应用程序）的过程。

目录