网络编程实战丛书

- 快速入门：本书在结构上由浅入深，步步进阶，引导读者快速掌握 EJB 的概念和编程
- 实用性强：本书对 EJB 的每一个概念都使用具体的实例进行讲解，针对性强，具有很高的实用和参考价值

Enterprise JavaBeans 编程实战

屈晓晖　编著

科学出版社

网络编程实战丛书

Enterprise JavaBeans

编程实战

屈晓晖　编著

科学出版社

2001

内 容 简 介

本书从服务器端的组件结构出发，由浅入深地引导读者全面掌握和深入领会 J2EE 的核心技术 Enterprise JavaBeans，并使用其在电子商务领域内的具体应用指导读者掌握 EJB 技术的开发，本书将提供大量实例供读者在实际开发中参考。通过对本书的阅读，相信读者会逐步掌握 J2EE 核心技术 Enterprise JavaBeans。

本书是关于 Enterprise JavaBeans 技术的入门书籍，具有较强的实用性和指导性，对于当前从事电子商务系统应用程序体系结构设计人员、服务器及客户端的程序开发人员、应用程序扩展功能人员、技术支持和管理人员来说是重要的开发指导书，同时也是高等院校相关专业师生教学和自学参考书。

图书在版编目（CIP）数据

Enterprise JavaBeans 编程实战/屈晓晖编著. —北京：科学出版社，2001

（网络编程实战丛书）

ISBN 7-03-009716-5

Ⅰ.E… Ⅱ.屈… Ⅲ.JAVA 语言—程序设计 Ⅳ.TP312

中国版本图书馆 CIP 数据核字（2001）第 055585 号

科学出版社 出版

北京东黄城根北街16号

邮政编码:100717

http://www.sciencep.com

双青印刷厂 印刷

科学出版社发行 各地新华书店经销

*

2001 年 8 月第 一 版 开本：787×1092 1/16

2001 年 8 月第一次印刷 印张：15 1/2

印数：1—5 000 字数：353 000

定价：22.00 元

(如有印装质量问题,我社负责调换<路通>)

前　言

随着国内电子商务应用的发展，Java 以其高安全性和与平台无关的特点，日益成为互联网开发的主导语言，而 Enterprise JavaBeans（EJB）技术则是 Java2 Enterprise Edition（J2EE）的核心。EJB 技术自 1998 年 3 月问世以来很受好评，它在平台供应商和企业的开发小组中都保持着空前的发展势头，因为 EJB 技术简化了中间件组件的开发，这些中间件组件都是事务性、可伸缩和可移植的。EJB Server 通过为中间件的服务提供自动支持，降低了开发中间件的复杂程度，同时，对企业来说避免了资源的重复构建，节约了生产成本。

这本书将使你深入理解 Enterprise JavaBeans 技术，并使用大量的例子引导你一步一步、循序渐进地掌握 Enterprise JavaBeans 的开发步骤。

本书在写作过程中主要参考了 Ed Roman 的“Mastering Enterprise JavaBeans™ and the Java2™Platform，Enterprise Edition”一书。Ed Roman 在高端中间件技术方面是一个世界级的专家，他曾积极参与了 Sun Microsystems 的企业级 Java 解决方案，并且建立和部署了许多企业级应用程序，包括一些应用程序服务器产品。为讲解精确，本书使用其中部分组件代码在 Bea Weblogic 5.1 环境下调试通过。

本书是关于 Enterprise JavaBeans 的使用指南，它介绍了有关 EJB 的概念和开发方法，并提供了大量的 EJB 范例供读者练习，同时本书也介绍了用于开发企业应用的 Java2 Enterprise Edition 平台，EJB 则是该平台关键的组成部分。通过对本书的阅读，读者会深入理解 EJB 和 J2EE 及其在电子商务系统中的部署。

EJB 和 J2EE 包含了许多高级的概念，包括分布式计算、数据库、安全及组件驱动软件等。本书不对 J2EE 进行全面的讨论，旨在讲解使用 EJB 开发可重用组件的一些概念和方法。

本书所涉及的技术

Java2 Enterprise Edition 平台是一套复杂的企业 APIs，它能使你建立强健可伸缩的多用户安全的企业级应用。J2EE 是非常巨大的，它包含了大量的概念。本书只讨论使用 EJB 编程时所涉及的有关 J2EE 的以下几个主要部分：

- Enterprise JavaBeans 1.0 版
- 如何在 EJB 中使用 Java Database Connectivity（JDBC）
- Java Transaction API（JTA）
- Servlets 和 EJB

在本书中，我们没有讨论 Java Server Page（JSP）技术。JSP 使用基于 Web 的表现层来增强你的企业级部署，我们只讨论与 JSP 紧密联系的 Java Servlets 技术（JSP 在运行时要编译成 Servlets）。

本书的组织结构

本书分为以下三个部分：

第一部分：EJB 概述。这一部分将讨论分布式构架、多层部署及其他一些结构，包括 J2EE 平台。在这部分里，我们将阐述 EJB 在 J2EE 中所扮演的角色，并为读者提供 EJB 快速开发指南。本部分对 EJB 新手来说很重要。

第二部分：EJB 开发指南。我们将讨论如何编写两种类型的 EJB：会话 Bean 和实体 Bean。我们将利用丰富的实例讲解编写这两种 EJB 所需要的基本知识，并且详细阐述两种会话 Bean（有状态的和无状态的会话 Bean）和两种实体 Bean（Bean 管理持久性实体 Bean 和容器管理持久性实体 Bean）。

第三部分：使用 EJB 构建电子商务系统。阐述如何在实际的电子商务系统中使用 EJB 技术。我们将使用一个电子商务应用演示如何使用 EJB 建立一个电子商务 Web 站点。我们将从需求分析开始，然后设计一套企业 Bean 和 Java Servlets 来实现这些需求，最后实现每一个组件并阐述如何在实际部署中来组织这些组件。当你读完这一部分，就应该能牢固掌握如何使用 EJB 解决现实生活中的问题了。

本书中，我们尽量使每一个例子都相互关联并能运行，其程度也由易到难，使得新手和老手都能找到自己编写代码的模板。随着章节的深入，我们会详细介绍每个 API 函数的用法。在第三部分中也有许多可以重用的企业 Bean，你可以参考它构建自己的企业 Bean。

由于时间及能力所限，书中错漏在所难免，恳请广大读者批评指正。

最后预祝广大读者学习顺利、愉快！

作 者

目　录

第一部分

第二部分

第三部分

第一部分

Enterprise JavaBeans 概述

这一部分将介绍服务器端的开发平台——Java2 Enterprise Edition（J2EE）及其重要的组件构架 Enterprise JavaBeans（EJB）。J2EE 是由许多概念和编程标准组成的，其所有部分都是用 Java 语言编写的。使用 J2EE 可快速地构建分布式的、可伸缩的、可靠安全的服务器端应用程序。

第 1 章介绍服务器端的组件构架。读者将会了解诸如可伸缩性、可移植性、资源管理和安全之类的服务器端开发需求，以及对快速应用程序开发的组件构架如 EJB 和 COM+的需求：所有这些我们都使用 Sun Microsystem 的 J2EE 这个真正意义上的服务器开发平台来讲解。

第 2 章概述 Enterprise JavaBeans 的组件模型。我们将会明白不同的开发商是如何使用 EJB 协同工作来解决商业问题的。我们将学习 EJB 部署中每一部分的作用、各种不同的功能软件模块以及它们是如何相联系的。

第 3 章详细介绍 EJB 中的事务处理。我们将会了解事务在 EJB 开发中的地位和作用，以及如何在 EJB 中实现事务。

第 1 章　服务器端组件构架

Enterprise JavaBeans（EJB）是一个服务器端组件结构，它简化了在 Java 中构建分布式对象应用程序的过程。使用 EJB 可以写出可伸缩的、可靠安全的应用程序，而不用亲自编写复杂的分布式对象构架。EJB 是快速开发服务器端应用程序组件技术。通过使用一个系统现有的分布式基础构架，就可以利用 Java 语言简单快速地创建服务器端的组件。EJB 是为支持跨平台可重用的中间件服务而设计的。EJB 是一种复杂的技术，要掌握它需要彻底详细的论述。这一章我们将讨论 EJB 周边的一些主要概念。作为本书讨论的重点，Java2 Enterprise Edition（J2EE）平台是一个企业技术的集成，其中 EJB 是一个完整的部分。通过理解和掌握 J2EE，你就能利用 Java 创建可重用的面向对象的企业级服务程序。

1.1　服务器端组件构架的必要性

为了理解 EJB 的优点，我们有必要首先了解所有开发人员在服务器端环境中部署组件时所面临的一些共同需求。

1.1.1　软件组件

软件组件的目的是创建能够插入容器中的可重用的组件。组件模型之所以能够在软件开发领域中快速流行，是因为它在开发中瞄准了若干个目标，即重用、集中于高端的开发、利用工具实现开发的自动化以及简化了部署过程。JavaBean，EJB 和 ActiveX/COM 都是组件模型的例子。

组件模型分为两种，即客户端组件和企业组件，客户端组件通常用于处理表现和用户界面之类的问题，比如 JavaBean，而企业组件关心的是为中间件提供面向事务的基础构造，如 EJB。使用这两种组件的任一种都能够实现商业逻辑的构造块，这种构造块的方法方便了作为组件的代码包的重用。由于容器提供了许多服务，因此组件开发人员能够编写高级的商业逻辑，而不用编写诸如网络通信之类的代码，允许开发人员装备组件的开发环境使开发自动化了。诸如 EJB 和 ActiveX/COM 之类的企业组件模型主要用来开发一些中间件服务。企业中间件的开发相当复杂，它不仅要考虑商业逻辑，而且要考虑并发性和伸缩性，同时还要考虑与不兼容平台的无缝集成。企业组件模型通过将所有这些因素内建在容器或服务器中，从而解决了中间件开发的复杂性。这样就可以使组件开发人员只集中编写商业逻辑，而无须考虑同步性、伸缩性、事务一致性、网络、分布式对象框架以及其他相关问题。换句话说，企业组件模型为开发人员提供了以下几个好处：

- 组件只包含商业逻辑
- 可嵌入性和可重用性

- 可伸缩性
- 资源管理
- 事务支持
- 并发管理

软件组件就是用来实现一组定义好的接口的代码。它是可以管理的，在逻辑上是分散的。组件不是一个完整的应用程序，它们不能单独运行，但是可被用来解决一些大的难题。

软件组件的功能是非常强大的。企业可以购买一个定义好的模块来解决某一个别的问题，将这些模块组装起来并结合其他组件就能解决一些大型企业问题。例如，假定一个软件组件是用来计算商品价格的，我们称之为“计价组件”，我们的目的是根据各种不同的因素计算出订购商品的总价格。比如这些价格因素是：

(1) 基础价：单个商品的零售价格。

(2) 数量折扣：购买某个数量的商品所给予的优惠。

(3) 捆绑折扣：同时购买某几类商品时所给予的优惠。

(4) 贵宾折扣：给予某个特殊客户的优惠。

(5) 地域折扣：根据客户所处的位置给予的优惠。

(6) 间接费用：如运费和税费。

这些价格规则在许多行业都同样适用，如电脑配件、日常用品、航空机票等都需要同种价格规则。显然，如果每个需要复杂计价规则的企业都各自编写自己的计价引擎，无疑会造成巨大的资源浪费。这样，开发商只需提供一个统一的可对不同客户反复重用的计价组件就能解决所有的问题。例如，邮政系统可以使用组件计算邮包的运费；汽车制造商使用计价组件区别不同型号汽车的价格；在线的商店可以使用组件区别星级不同的用户。当一个客户在网上订购了一些商品时，价格组件首先计算商品的价格，然后由另一个开发商的组件使用产生的价格给用户开账单，最后由第三方组件完成订购，并将商品送给终端用户。

可重用组件的确很受青睐，因为它加快了应用程序的开发。你可以快速地使用一个写好的组件组装你的应用程序，而不需要考虑组件内部的复杂的算法。这样，一旦完成了组件的接口设计，开发商就可以把组件卖给其他公司来专门从事组件的买卖交易了。

1.1.2 组件构架

为了方便组件的开发，应该有一个标准的方法来构建、管理和维护组件。这些方法包括：

(1) 用于开发组件的工具。在组件的构建过程中，开发人员只需专注于编写组件内部的核心逻辑，这样就加快了程序的开发过程。一些集成开发环境（IDE）如 Symantec 公司的 Visual Cafe、IBM 公司的 VisualAge for Java 或 Inprise 公司的 Jbuilder，都能帮助 Java 开发人员快速开发和调试组件。也有其他的一些开发商如 Inline Software 提供了专门用于 EJB 开发的工具。

(2) 用于管理组件的容器。组件容器为组件提供了一个运行环境，同时也提供了一套大多数组件都需要的通用服务。例如，必要时容器会自动对组件进行初始化，这样就

减轻了组件开发人员的负担。为了将组件和容器结合起来，你必须在容器和组件之间建立一个规则。利用这个规则容器就可以对组件进行规范化管理。

（3）用于部署和维护组件的工具。当从组件开发商那里购买了一些组件时，必须有一套用来部署和维护组件的工具。例如，应该有一个为特殊环境定制组件的方法。以计价组件为例，应该有一个帮助我们定制所要计价商品的工具。

上述每一个方法对于组件的开发都很重要。当然，作为一个组件开发员，可能你更关心的是组件的商业逻辑，而对容器和工具并不关心。一个好的组件结构可以为不同开发商开发组件提供必要的标准、组件容器和工具。这样，有了组件结构标准，开发人员就能随心所欲地在程序中实现商业逻辑。

1.1.3　组件构架的理想语言

一个组件要能成功地解决商业问题，组件开发人员和使用组件的客户都必须使用相同的语法来调用组件的方法。因此，组件开发商必须公布调用组件的规则，客户端代码必须遵守这些规则。

当开发商发布一个新版本时，开发商的用户也想升级，这就会产生许多问题：用户是否还能调用老版本组件的代码？用户是否需要重新编译它的代码？更严重的是组件的规则是否发生了变化？如果发生了变化，用户是否有必要修改客户端代码来适应新的组件规则？等等。

幸运的是，面向对象的设计方法通过将组件接口与实现相分离，从而使以上这些问题得以解决。

组件的接口定义了调用组件的规则。例如，接口定义了获取组件的方法和参数，接口对客户隐藏了实现代码，因此客户端只需处理最终的结果，即调用组件的方法。

组件的实现是对象提供的核心编程逻辑，它有一些非常具体的算法、逻辑和数据。这些数据对于组件来说是私有的，并且对于所有调用组件的客户端代码而言都是隐藏的。

为了有效地使接口与实现相分离，开发人员必须对组件的接口编写客户端代码，这就叫基于接口的编程。通过将接口与实现相分离，不用修改任何客户端代码就可以改变组件的商业逻辑。例如，你可以使用一个完全不同的方法（或算法）来实现某个接口使之达到相同的目的。这是可能的，因为 Java 在编译阶段（只有接口需要）不考虑具体的实现代码，因此与客户端代码相联系的只有一些接口函数。

Java 语言通过使用 interface 和 class 关键字在语法层上支持了接口与实现的分离。由于 Java 是一种解释性语言，将代码分成离散的类文件，确保了在发布一个新的组件版本时不需要对客户端代码重新编译。

除了支持接口与实现的分离，Java 还是一种面向对象的跨平台开发语言，并有着广泛的行业支持，这就使 Java 语言成为一种理想的构建组件的技术。

1.1.4　Java 中的组件构架

到这里你已经明白了什么是组件构架。我们再来看一下 Java 的组件构架。可能你对 JavaBean 并不陌生。JavaBean 也是 Java 的组件模型。在 JavaBean 规范中定义了事

件和属性等特征，Enterprise JavaBeans 也定义了一个 Java 组件模型。但是，EJB 组件模型和 JavaBean 组件模型是不同的。JavaBean 组件模型的侧重点是允许开发者在开发工具中可视化地操纵组件。JavaBean 规范详细地解释了组件之间事件登记、传递、识别和属性的使用。它是一种小型的应用程序组件，你可以将多个 JavaBean 组装成一个规模较大的组件，甚至是一个完整的应用程序。然而 JavaBean 是一种可开发的组件，而不是可部署的组件。你不能部署一个 JavaBean，因为它不是一个完整的应用程序。正因为 JavaBean 不能部署，所以它不需要赖以生存的运行环境，也不需要对它进行初始化和提供其他服务的容器，因为应用程序本身就是由 JavaBean 组成的。而 EJB 的侧重点是详细定义了一个可以方便部署 Java 组件构架的服务框架模型。因此其中并没提事件，因为 EJB 通常不发送和接收事件；同样也没提及属性，属性定制并不是在开发时进行的，而是在运行时（实际上是在部署时）通过一个部署描述符来定制的。EJB 组件是一种可以部署的组件，它必须被部署在一个提供运行服务的容器中。

JavaBean 和 Enterprise JavaBeans 是两个截然不同的概念。它们都是组件模型规范，但是前者说明了开发工具中应用程序组装的问题，而后者则侧重于部署组件的服务框架细节。不要错误地以为 JavaBean 是用于客户端的开发，EJB 是用于服务器端的开发。JavaBean 也可以作为非图形化服务器端 Java 应用开发的组件模型。它与 EJB 的区别是，当你使用 JavaBean 创建服务器应用时，你还得设计整个的服务框架。而 EJB 框架是现成的，你只需遵守它的 APIs 对于复杂的服务器端应用程序，显然使用 EJB 比重新开发更简单。

Enterprise JavaBeans 与其他两种类型的 Java 组件非常类似：Applets 和 Servlets。Applets 可以部署在一个网页中。网页浏览器的 Applet Viewer 为 Applet 提供了运行的容器。Servlets 可以部署在一个 Web Server 中，而 Web Server 的 Servlets 引擎为 Servlets 提供了一个运行的容器。EJB 部署在 Application Server 中，是 Application Server 为 EJB 提供运行时的容器。Applet，Servlets 和 EJB 之间的真正区别是每一种组件类型都是整个主域的一部分。Applets 是一个可以动态下载的 Java 小程序，并能在一个不信任的环境中执行。比如，一个 Applets 可以从 Web Server 中下载到 Web 浏览器，并能向终端用户显示用户接口。Servlets 是可以用来扩展 Web Server 功能的网络组件。Servlets 是面向 Request/Response 的，即它从一个客户主机（如 Web 浏览器）接受请求，并将响应返回给客户主机。这就使 Servlets 很适合执行 Web 任务。Applet 和 Servlets 都很适合处理客户端的操作，如完成一些与描述相关的逻辑和轻量级的商业逻辑操作。而客户端可以是一个 Web 浏览器，这种情况下 Applets 使用 JFC（Java 基础类库）提交用户界面；客户端也可以是一个 Web Server，这种情况下，是用 Servlets 以 HTML 的方式提交用户界面。这两种情况下，组件都是直接对终端用户进行处理。相反，EJB 的目的不是面向客户端，而是一个服务器端组件，它们用来完成服务器端的操作，比如执行一个复杂的商业事务算法。Application Server 为 EJB 提供服务器端环境和管理 EJB 的运行时容器。最后应该注意的是 Applet，Servlets 和 EJB 三者并非是可以相互替代的。你可以使用 JavaBean 作为开发部件来构建一个可部署的 EJB，也可以使用 Applets 或 Servlets 为 EJB 提供一个用户接口。Applet，Servlets 和 EJB 三者的关系如图 1.1 所示。

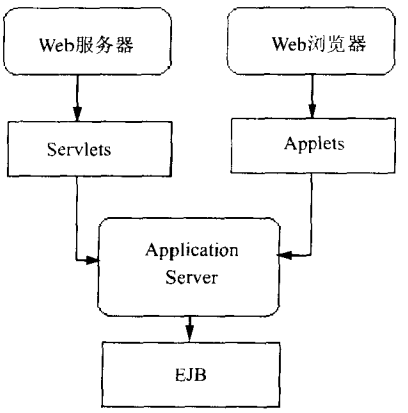

图 1.1　Applet、Servlets 和 EJB 三者的关系

1.2　服务器端的组件构架

一个完整的组件构架为使用组件的各方提供了极大的方便：开发人员可以用它编写可重用组件，开发商可以编写组件容器来为组件提供一个运行的环境和服务，还可以提供组件的开发、部署和维护工具。

这种组件构架允许开发商提供一套大多数组件都需要的通用服务来节省开发和部署时间。当其他产品需要时，组件开发人员只需购买这些服务而不需亲自建立这些服务。如果你非要自己建立这些服务，就要面临艰难的开发和维护工作。我们将会发现服务器端软件解决了许多需要高端服务的难题。

服务器端的部署是一个软件，它能同时安全、可靠、高效地支持并发用户的操作。服务器端部署的例子有：

- 银行应用：多个 ATM 机连到一个中心银行服务器。
- 连锁店应用：由多个连锁店向一个中心店发送求购信息。
- 支持中心：各个支持工程师都从中心服务器将客户数据调到自己的终端。
- 保险代理商：各保险销售机构都有一个连接到中心服务器的终端。
- Web 站点：成千上万的用户连到 Web 服务器，Web 服务器需要连到中心服务器请求数据和逻辑。

建立一个强健的服务器端部署并不容易，往往会遇到很多问题，如可伸缩性、可维护性、安全性、可靠性等。由于这么多的客户都依赖于你的中心服务器部署，一旦中心服务器遭到攻击将会出现灾难性后果。因此，服务器端的部署需要精心编写、精心测试，并需要运行环境特别稳定。

好的部署软件会将逻辑划分为各个层，每个层在整个部署中都负有不同的责任，而每个层都可以有一个或多个组件。注意这些层都是抽象的，它们不会与物理分布一一对应。一个分层的系统就是设计精巧的系统，因为每个层都负责一个独立的任务。有以下

几种典型的分割层：

（1）表现层。包含一些处理用户界面和用户操作的组件，单机版的表现层可以使用 Visual Basic 等前端开发工具来编写，基于 Web 部署的表现层可以使用 Java Servlets、Java Server Page 或 Java Applets 来编写。

（2）商业逻辑层。包含一些协同解决商业问题的组件。这些组件可以是高性能的引擎，其编写也要使用安全性强的语言如 Java 或 C++等。

（3）数据层。它被商业逻辑层用来维持状态的持久性。数据层的核心是一个或多个为存储状态提供场所的数据库。

将应用程序分成这些逻辑层的好处是能使每个层之间相互分离。比如，为了最大限度地降低对商业逻辑层和数据层的影响，应该在表现层放置一个视图来改变表现层；同样，为了减少对其他层的影响，可以在商业逻辑层放置不同的商业规则，或者在数据层放置不同的数据库。

在两层结构中，其中的两层与第三层分离，形成两个被物理分割的层。在三层结构中，这三个抽象的逻辑层被分割成三个物理分布的层。每种结构中，层层之间被物理边界划分开，这些物理边界包括主机、过程或操作组。将应用程序分成两层或多层到底有什么好处呢？

1.2.1 两层（Two-Tier）构架

传统方式下，大多数高端部署都是两层构架。两层部署将商业逻辑层和表现层或数据层结合在一起。有两个结合的可能：一种是表现层和商业逻辑层的结合，另一种是将一些商业逻辑层放在数据层。下面就讨论每一种情况。

1. 表现层和商业逻辑层的结合

表现层和商业逻辑层可以放在一个层中，而把数据获取层放在另外一个层中，这样就在商业逻辑层和数据层之间划分了一道界线。如果你将前一个层看作客户端，而将后一个层看作服务器，这种构架就是“胖”客户端和“瘦”服务器结构。

在两层构架中，客户端应用程序通过 Database Bridge APIs 如 ODBC 或 JDBC 与数据库进行通信。这会导致客户端与正在使用的数据库脱节，因此每个开发商的宿主数据库必须符合 Database Bridge 的标准，而每个数据库开发商必须提供在 Database Bridge APIs 中进行调用的数据库驱动，如 ODBC 驱动或 JDBC 驱动。

两层构架拥有以下特点：

（1）部署代价大。数据库驱动必须安装并配置到每个客户端，有可能是成千上万台机器。

（2）数据库驱动转换代价大。将数据库驱动从一种转换为另一种目前需要重新安装每一台客户端主机的数据库驱动，其维护费用相当高。

（3）数据库模式转换代价大。“胖”客户端直接通过 JDBC、SQL/J 或 ODBC 存取数据库，这就意味着客户要直接对数据库模式进行处理。如果你要移植数据库模式来处理一个新的商业过程，就需要重新部署每一台客户端。

（4）数据库类型转换代价大。“胖”客户端是与数据库 API 绑定在一起的，如关系

数据库 API 或对象数据库 API，如果你决定在这些数据库之间进行转换，不仅必须重新部署每个客户端，还必须彻底修改客户端代码来适应新的数据库类型。

（5）商业逻辑移植代价大。改变商业逻辑层包括重新编译和部署客户端 Tie。

（6）数据库连接代价大。每一个数据库客户端都需要建立它自己的数据库连接，这些连接的建立在数量上是有限制的。即使客户端不再使用数据库，连接仍会保持而不能被其他用户使用。

（7）网络性能受损。每当你的商业逻辑执行一个数据库操作时，就需要重复穿越分割商业逻辑层和数据层的物理边界，这会明显延长数据操作的时间并使网络受阻。

2. 将某些商业逻辑层移到数据层

这种部署将一部分商业逻辑层和数据层放在一个独立的层中，如图 1.2 所示。如果你将表现层看作“客户端”将商业逻辑层和数据层看作“服务器”，这种构架就是“瘦”客户端和“胖”服务器构架。

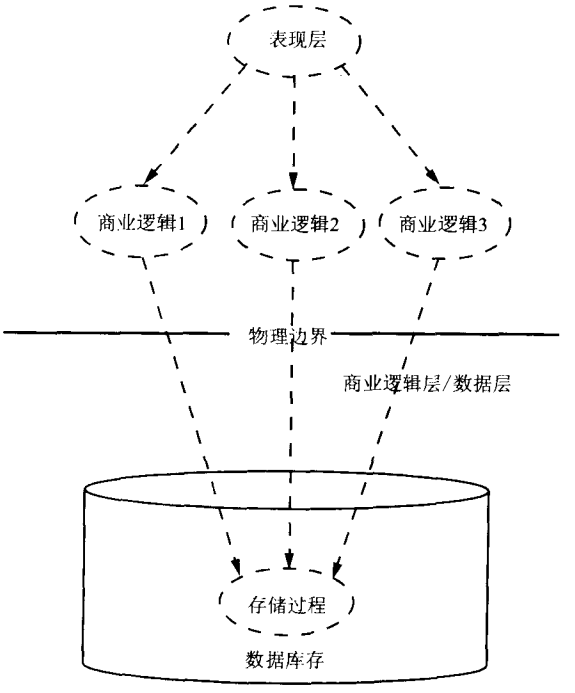

图 1.2　在两层结构中将数据获取逻辑放在第二层

（引自 Ed Roman，1999，稍作修改）

为了实现这种情况，就要将一部分商业逻辑（通常是与持久性有关的数据逻辑）放在数据库中。数据库允许在数据库的上下文中通过编写存储过程来执行逻辑。通过将部分逻辑放在存储过程中，就有效提高了系统的伸缩性能和执行性能。由于一些逻辑被放进了数据库，极大限度地降低了网络流量，使其他客户端能更快地完成网络操作。

然而，即使是这样，以前的问题仍会存在。实际上，这样会出现一些新的问题。比如，存储过程语言宿主性很强，每个客户端都与特定的数据库相连。利用 ODBC 或 JD-