分布式 JAVA 2 数据库系统开发指南

Distributed JAVA 2 Platform Database Development

[美] Stewart Birnam 著

孟纯城 译

清华大学出版

分布式 JAVA 2 数据库系统开发指南

[美] Stewart Birnam 著

孟纯城 译

清 华 大 学 出 版 社

（京）新登字 158 号

内 容 简 介

本书内容包括 JDBC、RMI、Swing、Servlet 和命令行程序的设计，以及如何综合运用这些技术构建一个系统，以便采用 Oracle 数据库和 Linux 操作系统。本书的前几章主要介绍了一些多层系统的背景知识。随后的一些章节讲述具体的编程技术，以及如何复用代码在 Swing 应用程序或 Servlet 中访问数据库。

本书的读者对象包括分布式数据库系统设计与开发人员、IS/IT 管理人员、想学习和研究新的技术的软件开发人员、正在学习编程的人员等。

北京市版权局著作权合同登记号：图字 01-2002-3162 号

书　　名：分布式 JAVA 2 数据库系统开发指南
作　　者：[美] Stewart Birnam
译　　者：孟纯城
责任编辑：王　松
出 版 者：清华大学出版社（北京清华大学学研大厦，邮编 100084）
http://www.tup.tsinghua.edu.cn
印 刷 者：世界知识印刷厂
发 行 者：新华书店总店北京发行所
开　　本：787×960 1/16　**印张**：16　**字数**：344 千字
版　　次：2002 年 8 月第 1 版　2002 年 8 月第 1 次印刷
书　　号：ISBN 7-302-05755-9/TP・3402
印　　数：0001～4000
定　　价：36.00 元

目　录

第 1 章　分布式数据库应用程序设计

本章主要内容

- 技术概览
- 数据库应用程序模型
- 两层模型
- N 层模型（3 层或更多）
- 两种模型的对比
- 移植到多层结构

本章介绍了多层系统及其体系结构的背景知识，目的是为程序员和项目管理人员提供一个理论基础，以便理解分布式系统。如果您已经掌握了这些内容，可以跳过本章，直接学习第 2 章。

1.1　技 术 概 览

1.1.1　理解系统

构建一个分布式数据库应用系统比开发简单的应用程序所涉及的东西要多得多。这意味着需要了解系统间如何通过网络进行交互；意味着需要弄清数据库理想运行的条件；意味着了解所有系统组件的优点和缺点，在某种程度上，尝试将问题分布到这些系统，以便发挥最佳性能。

因此，除非从头开始构建系统，否则就要进行大量的分析，找出现有系统值得保留的部分和系统的瓶颈所在。另外，成本和现有系统的要求可能会对如何构建系统产生影响。有时候，需要优先考虑系统的扩展能力；而有时候却需要考虑怎样才能在获得最大灵活性的同时，发挥出最佳性能。

像这类复杂的问题，最适宜用 Java 技术来解决。无论是集成老系统，还是从头构建新系统，采用 Java 技术可在一个相对短暂的开发周期中，构建出稳定而复杂的系统。然而，在正式编写程序前，需要先了解一些分布式应用系统的硬件问题。本章将介绍一些影响应用程序开发的硬件问题，回顾传统的数据库计算模型，让大家了解分布式系统的优越性，并熟练地掌握一些技术术语。

1.1.2 硬件和网络配置

构建系统前，需要了解清楚当前环境中的服务器和工作站，以及它们之间是如何连网的（100BaseT/10BaseT/千兆以太网？LAN/WAN？防火墙？代理？）。同样重要的是，在每台计算机上运行的操作系统是什么？服务器和工作站分别运行不同的操作系统是很常见的。不过，基于应用软件的需求，目前越来越多的人为客户机选配“另类”操作系统（非Win32）。特别是随着另类操作系统的发展，现在的工作站并不仅仅局限于 Win32，而是包括了 Solaris、Linux、FreeBSD、Win32 及 MacOS 等多种操作系统。服务器可能会运行 Solaris、True64、AIX、IRIX 等操作系统，甚至可以运行 Linux 和 FreeBSD 操作系统。我相信，随着应用程序需求的不断提高，将迫使计算机环境从同质走向异质，这种百花齐放的情况还会持续下去。但是，不管是否出现这种情况，我们都应提前准备好用 Java 技术来使用它们。编写本书时，我发现支持 Java 1.1 的操作系统至少包括：FreeBSD、Linux（Intel 及 Alpha）、HPUX、AIX、Irix、Tru64 以及 MacOS，当然还有 Solaris 和 Win32。

1.1.3 系统瓶颈

一旦从掌握的情况中归纳出要点，就能清楚当前操作环境中有几处瓶颈很难轻易解决，或者需要马上修改预算。了解这些情况，有助于判断是否需要尝试用软件解决这些问题。

1.1.4 为明天做好准备

我们还需要为将来做计划，设计的系统需要有足够的灵活性以应付未来的变化。现在使用的 Linux 服务器是否会在明年被 Solaris 服务器取代？是否会有一定比例的 Win32 操作系统要被 Linux 操作系统所取代？如果系统的任何一部分依赖于具体的平台，日后必然会带来问题。例如，如果采用了一种第三方应用程序服务器或者其他中间件，就会受限于供应商所支持的操作系统。即便您决定使用 Java 2，也只能局限于当前支持它的操作平台和其他软件。

例如，为了使用服务器端的应用程序，Oracle 8i 数据库有一个内置的 Java 虚拟机（JVM）。这种 Java 虚拟机为数据库作了优化，而且可以提高在线事务处理应用程序的性能。Java 虚拟机基于 Java 1.1.6 版。如果计划采用这种技术，在应用程序框架中，应该考虑维护两个不同 Java 版本所涉及的潜在问题。

刚开始编写本书时，只有 Java 1.1 能在各平台通用。Java 2 的许多 alpha 和 beta 版本可以用于 Win32 和 Solaris 外的其他平台，但 Java 2 还需要进一步完善。目前，如果想开发可靠的、易于配置的软件，必须采用 Java 1.1。幸运的是，服务器端并没有过多地限制，而且 Java 2 的许多特性（如 Swing 和 JNDI）都有独立的软件包，以便 Java 1.1 用户使用。

1.1.5　将系统连接到一起

把系统连接到一起需要两样东西：网络和 Java 技术。首先，确保可以访问所有计划连接的计算机。例如，可能需要通过代理与其他系统通信。也许可以自由访问一些系统，但由于这些系统过于繁忙，需要以对通信速度影响尽可能低的方式与它们通信。早期发现这些问题有助于设计出更好的系统。

1.2　数据库应用程序模型

让我们快速回顾一下目前流行的各种数据库应用程序模型，以便清楚地了解即将面临的一些系统问题。许多数据库开发人员共有的缺点，就是将硬件和系统的规划留给系统管理员。这样一来，在协调软件问题和硬件问题时，就会留下一个无人管理的巨大空间。开发人员有责任为管理员构建一个合理的数据库，因此需要从根本上了解系统是如何设计的。

为此，我们需要回顾一下常见的数据库应用程序模型。介绍了基础的硬件和网络问题，希望开发人员能够清楚地构建系统，并将其作为系统设计的参考。

1.2.1　两层模型

许多数据库应用系统使用两层模型。实际上，目前许多商业系统都采用这种模型，因为这种模型比较容易实现和设计。该模型的基本结构如图 1.1 所示。

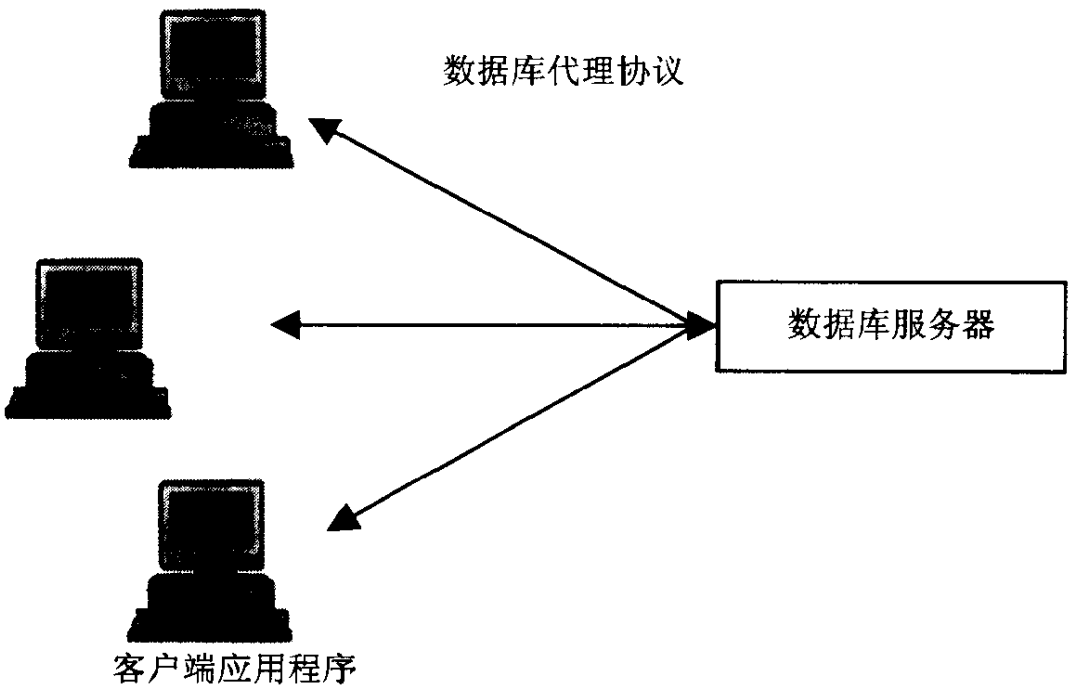

图 1.1　两层数据库模型

1. 硬件配置

大多数采用此模型的系统都有中型或大型的数据库服务器，通常是具有多处理器的计

算机，并运行数据库供应商支持的 Unix 版本。一般来说，此类系统往往需要投入大量的时间和金钱（多 CPU 计算机一般相当昂贵）。公司决定：必须升级计算机，并且每周 7 天、每天 24 小时运行。这是尝试提高性能的第一步，因而系统的运行会非常出色——至少我们希望如此。很可能是由于增加了系统的成本，它才会如此出色；因此，我们的任务是在不多花钱的前提下，让系统更加有效地工作。

提示 如果还有余钱，最好的改善方式就是购买内存。这样能显著地提高数据库应用程序的性能（假定您是一位熟练的数据库管理员，能利用所增加的内存修改数据库）。

2. 网络拓扑

在两层数据库模型中，当运行应用程序时，客户机程序会与数据库维持一种直接的固定连接。无论用户是否使用应用程序，此连接都会处于激活状态。连接一般由供应商协议组成（如 Oracle 的 SQLNet），并依赖于顶级传输协议（如 TCP/IP 或 DECNet）。这要在客户机和服务器间，建立一个复杂的连接。无论是否提供了传输协议，供应商协议会不停地进行错误校验和连接监视。对于大多数数据库开发人员来说，这是一个合理的、符合期望的情况，无需任何担心。然而，如您所见，仅仅是维护系统连接，就需要相当多的系统消耗。如果一个标准用户同时运行几个网络应用程序（Netscape、电子邮件等等），这对系统资源的消耗，将会影响工作站的运行。在某些操作系统上，甚至会引起工作站的崩溃。

3. 管理

为了配置采用两层数据库应用程序模型，每一台客户机均需安装数据库供应商提供的客户端软件，以及客户操作平台的编译库。安装客户端软件可能不太容易。例如，Oracle 的客户端软件需要配置几个用户环境变量，还需要编辑一些系统配置文件，以便指向主机的数据库。

若工作站运行了多个数据库应用程序，则可能会因安装了多个版本的客户端软件而出问题，有时是因为疏忽，有时是由于需要。

4. 支持——一个案例

数据库用户许可证一般根据可能同时使用数据库的用户数量来定义。因此，假设一个数据库定义了 5 个用户许可证。早上第一件事，财务部的用户启动了数据库应用系统并开始浏览。一个用户输入了一些数据，然后去吃午饭、开会、或者坐下来看 1 小时的邮件。突然，市场部产生了新业务，需要修改数据库，他们需要更新客户信息，或往数据库输入新账户。当他们尝试连接数据库时，由于数据库服务器统计到已有 5 个用户在线，所以数据库服务器拒绝了他们的连接请求。

只要用户拥有数据库用户许可证，数据库服务器就允许他们连接数据库。从应用程序角度看，系统并没有考虑用户是否在执行操作。实际上，对于服务器来说，这种连接也是活动的，因为供应商协议层经常发出一些请求，以确保数据库处于激活和运行状态。由于没有考虑用户不进行操作的情况，所以市场部用户在其他用户退出应用程序前，无法进入应用程序。

因此，用户给 IT 部打电话，IT 部给数据库管理员打电话，她通过检查数据库以确定谁正在使用。她打电话告诉 IT 部的技术支持人员，财务部的 5 个用户正在使用数据库。IT 部的技术支持人员打电话给每一个用户，确定谁在真正使用数据库。最终，他们让其中一位没有使用数据库的用户退出了应用程序。随后，IT 部的技术支持人员打电话给市场部用户，市场部用户尝试登录。说尝试，是因为在这段开放的时间中，其他人也有可能尝试进入，而且也许正在设法这样做，所以连接开放的时间很短。结果是市场部人员仍然不能登录，并且开始大骂应用程序是垃圾，编程人员不称职等等。其实，市场部人员只需 2 分钟做一个简单修改。

如您所见，服务器和客户机（在上述事件中，是用户和技术支持人员）都浪费了资源去维护一条无用的连接。这里所说的无用的连接是指浏览应用程序的连接，而不是访问数据库的连接。而且，由于数据库已授权给同时使用的 5 个用户，故其他应用程序丧失了输入数据的权利。更糟的是，我们可能会遇到这种情况，公司员工需要在同一时间（假定是下午 4：30）输入少量的数据。我们可能不仅要处理数据库所允许的连接数，而且主机会重复尝试不必要的打开和关闭操作，并拒绝大量的并发连接申请。更糟的是，因无法改变您的状况，用户和服务部门不断出问题。这种数据库特性非常难于就此特殊问题对用户和技术支持人员进行培训。最后，让人觉得是由于您的失误才使大家处于这种状态中。

鉴于以上原因，我们思考用如下方法来解决：

我们可以购买更多的许可证来解决这个问题，但这种解决方法成本相当昂贵。特别是在系统本身没有得到改善下，况且很可能没有预算购买许可证的花费。

从软件开发角度看到的另一个问题是，此模型将大量的应用程序事务逻辑强行嵌入客户机程序中。因此，软件可能非常难于配置，同时也会给修改软件或者纠正 bug（泛指程序中的错误）带来困难。往返工作站安装最新版应用程序，只会延长程序开发的等待时间。同样，此模型也对 IT 技术支持人员（他们需要访问客户机并安装最新的软件版本）提出了更高的要求。

1.2.2　N 层模型（3 层或更多）

为了解决这些问题，我们将引入 N 层数据库模型。在 N 层模型中，中断服务器或供应

商服务器将采用简单协议处理客户机请求。随后，这些请求将通过复杂供应商协议传回数据库。如图 1.2 所示。

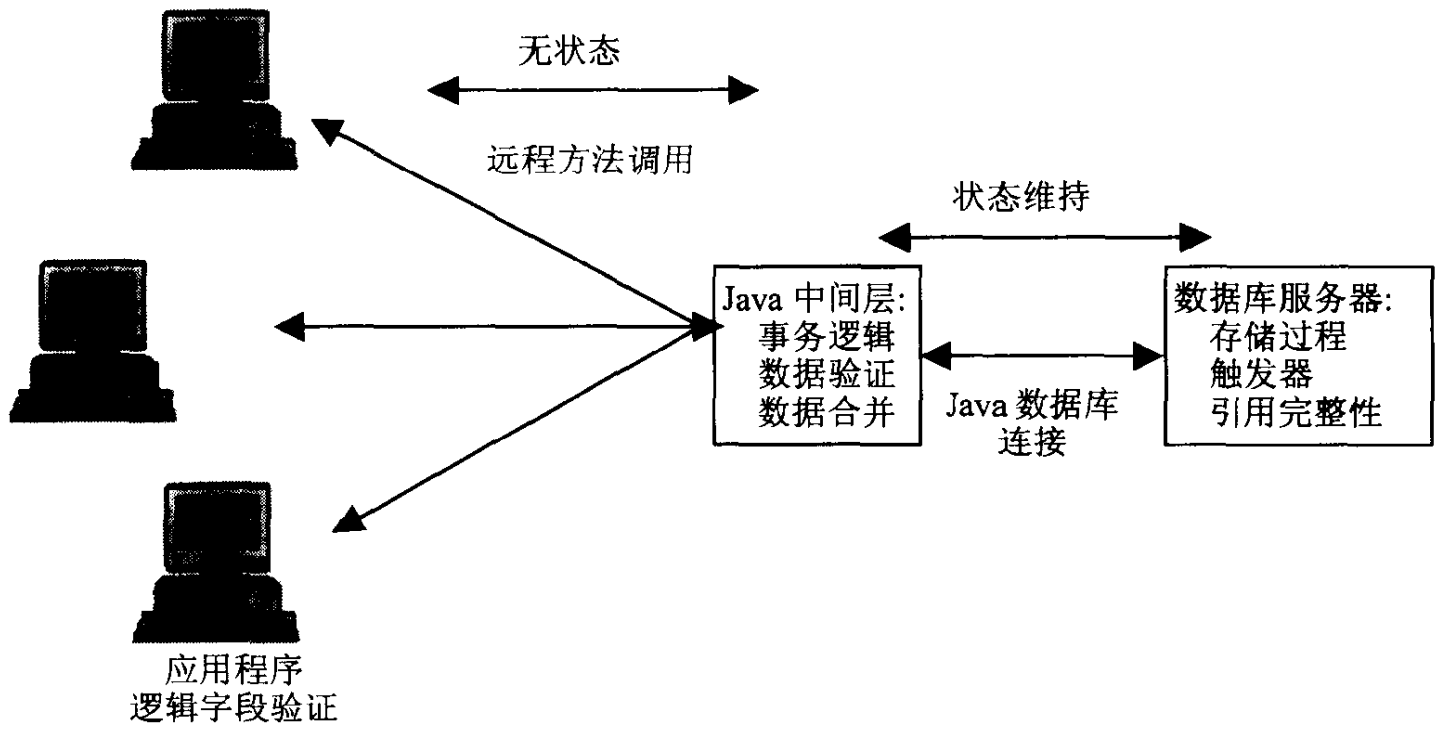

图 1.2　N 层模型

1. N 层模型的特征

N 层模型有如下特性。

- 客户机和服务器间的无状态连接。第一个特性是，客户机和服务器（现在是中间层）间的连接是无状态的（不保持开放）。因此，如果没有运行应用程序，无论客户机或服务器都不会耗费资源去维护一条无用的连接。

- 客户机和服务器间的通信无供应商专用协议。通过数据库连接中介，我们可以从专用的二进制平台及其复杂的安装中解放出来。客户端数据库模型的层数越少，客户机的操作系统就越稳定，也更易于调试。

  利用 Java 的远程方法调用技术（RMI），我们可以从实现套接字连接、协议，以及其他客户机和中间层的编程任务中解脱出来，更易于开发。

- 利用应用程序的类型和功能可以很容易地实现连接共享和加载平衡。我们可以有选择地构建中间层，强制多个客户机共享一个简单的、永久的数据库连接。通过让各客户机共享各种连接（利用应用程序功能，采用连接组实现），我们还可以实现加载平衡。例如，我们有一个数据记录对象、一个数据验证对象以及一个报告对象，每个对象都有各自的数据库连接。一台独立的客户机可以访问所有这些对象。根据应用程序的运行情况，有的客户机可能只访问了其中一个对象，而有的

客户机可能访问了所有的对象。在其他对象处理请求时，该请求将被列入专用对象的队列中。

- 中间层和数据库服务器间的永久的、开放式的连接。在初始化数据库连接时，会产生多次访问数据库的操作。从客户机删除这些任务后，初始化加载的时间就会缩短。中间层最好是高性能的计算机，它采用高速网络连接到数据库服务器。现在，数据库服务器需管理的资源很少。由于我们将连接功能移到了中间层，因而减少了数据库服务器的负担。

- Java 数据库连接保证了数据库的独立。现在很多数据库供应商为产品数据库提供 4 级 Java 数据库连接驱动程序（4 级驱动程序是指完全用 Java 实现的驱动程序）。这意味着我们不再依赖于二进制专用平台了。因为驱动程序是由 Java 实现的，故可以在任何操作平台上运行它（并且可连接到数据库）。另外，如果决定更换数据库供应商，或者需要从同一个中间层访问不同供应商提供的多种数据库。所要做的事情只是更改 JDBC 的 URL 连接，我们可能已经准备好了！

- Java 数据库连接保留了数据库连接模块的编程任务。此外，它由供应商支持，不同于其他内置于 Perl 或 Python 的模块。

- 可扩展性和适应性。采用本设计，系统可以增加与各种数据源之间的联系，无需考虑数据库供应商。中间层可以与各供应商的各种数据库会话。系统也可以访问大文件或其他网络服务（如 Web 或电子邮件）的信息。

2. 复杂的 N 层模型实例

（1）硬件配置。

图 1.3 展示了一个更为复杂的模型，为了将两层模型转换成 N 层模型，不需要对数据库的硬件和配置做任何改变。实际上，我们可以继续为其他两层应用程序同时提供服务。中间层此时成了另一台客户机。我们会对数据库服务器有疑问：添加新的中间层，许可证的总数是否会超过现有两层模型的客户机数量？

与数据库供应商取得联系，确保我们计划中的代理模式与并发用户许可证之间没有冲突。大多数数据库供应商会根据用户的需求，确定合理的许可证数量。

中间层服务器应该是中型服务器，其性能由计划服务的客户机数量决定。如果预算较少，可以选择采用一台快速工作站类型的计算机。预算和技术支持人员都会为我们指出这种解决方法。数据库服务器类似，一个稳定的、良好支持的 Unix 操作系统平台（像 Solaris）将会在长期运行中产生好的经济效益。

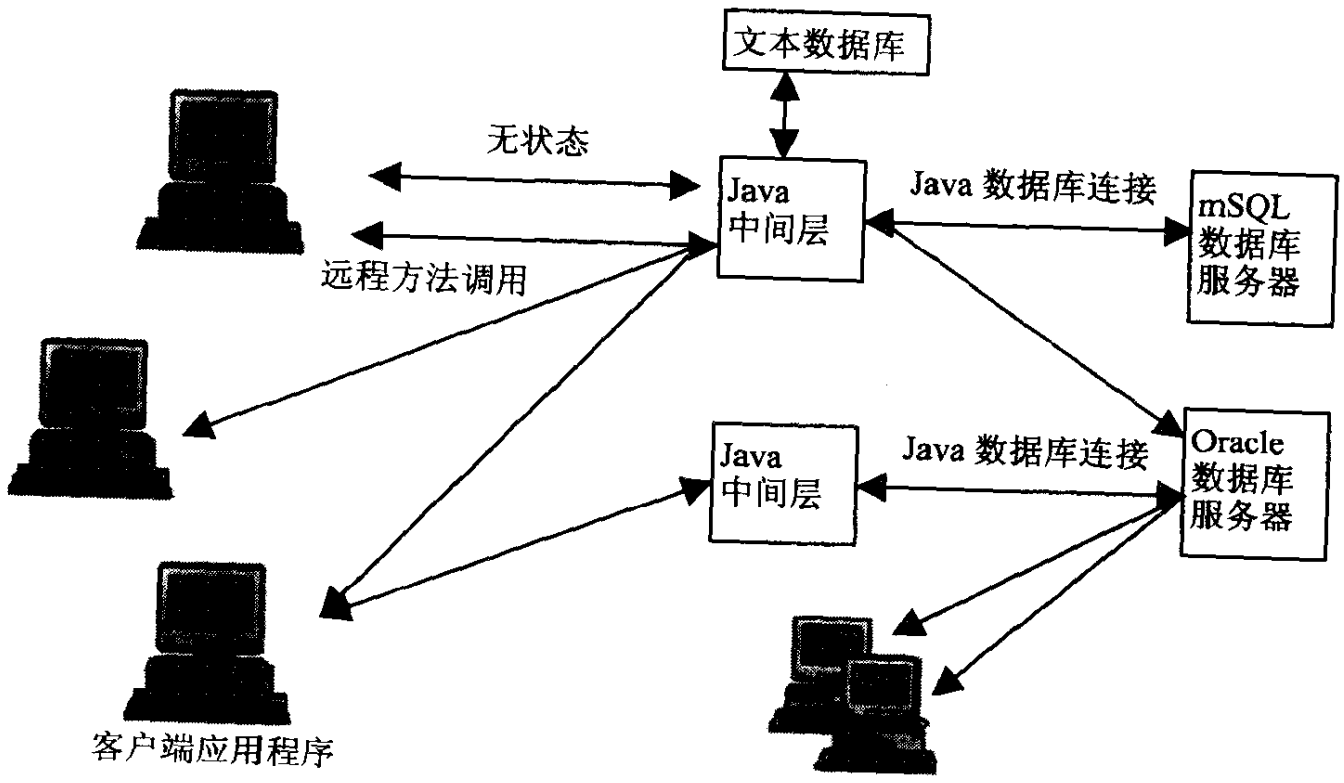

图 1.3 复杂实例

（2）网络拓扑。

中间层确保客户机与数据库之间的稳定连接，扮演着先前客户机的角色。中间层使用基本的供应商协议管理连接，此协议与原先客户机所用的协议相同。同时，它采用 Web 服务器所用的方式，接受来自客户机的网络连接请求。实际上，很多 Web 服务器使用这一方式，为某些类型的数据库提供公共访问，无需用户直接访问数据库。如在 *Amazon.com* 网站上浏览图书，就是通过中间层（他们的 Web 服务器）访问数据库的。Web 客户机不仅可以访问中间层，而且还可以作为图形用户接口，甚至可以作为命令行客户机。实际上，甚至可以将 Web 服务器访问中间层作为一种性能的优化，也可以在同一台计算机上同时运行 Web 服务器和中间层。这完全取决于预算和需求。

客户机和中间层的连接是无状态的。也就是说，客户机在需要时与中间层连接，用后将连接断开。这很像我们打电话。拨号、聊天、通话完毕后挂断。想像一下，如果我们采用两层模型与数据库会话的方式来打电话。您拿起电话，聊了一会儿，然后放下听筒去吃三明治。可能一个小时后，您又拿起了听筒说了一些事情，然后再次放下听筒。在所有的空闲时间中，您始终占着那条电话线。我们永远也不会这样打电话，但很多人却用这种方式开发数据库。

此外，客户机也不再使用供应商专用协议与服务器（现在是中间层）通信。这意味着：

1）现在客户机连接比较简单，占用少量的资源可获得同样的信息。

2）客户机不再需要操作平台专用的驱动程序、共享对象或者 DLL/DSO。

分析以上两点。考虑您或您的 IT 部门需要花多长时间来保证供应商服务器、操作平台专用驱动程序和客户端软件正常工作。我们需要确保在正确的位置，采用正确的版本、正确的用户路径和正确的用户环境变量设置，正确加载供应商数据库客户端软件，有时比实际开发应用程序的工作量还大。另外，在安装和配置方面出现的问题通常会反应出应用程序的缺点（即便这些缺点不是您的构思或失误）。现在，采用 Java 技术可以将所有的问题都解决了。当前惟一需要维护的是中间层的供应商专用驱动程序。而中间层是一台可由您控制的计算机。

（3）管理。

从管理的角度来看，专为管理增加服务器似乎不太合理。然而，对中间层服务器管理人员的专业技术水平的要求低于商业数据库服务器管理人员。可以由普通的系统管理员进行管理。实际上，这台服务器的设置和支持非常类似于管理 Web 站点服务器。幸运的是，现在已经有许多人熟悉这种技能。在最后几章，我们将竭尽所能，努力让中间层管理类似于 Web 站点服务器管理，或者普通网络服务管理。因此，系统管理员或者 Web 站点管理员可以管理中间层，必要时，也可让数据库管理员来管理。

除了采用该服务器作为中间层外，我们还希望这台服务器可作为文件/应用程序服务器用，并采用 NFS、Samba、HTTP 和 FTP 协议分配软件（详情参见第 7 章）。这些著名的服务，需要一位称职的系统管理员来管理。

1.2.3　两种模型的对比

下面我要讲述两层数据库模型和 N 层数据库模型的区别。

特别要强调的是，从一种模型移植到另一种模型，需要考虑它们在设计上的区别。两层模型将所有的控制都放在了两个位置：数据库服务器和客户机。N 层模型可以让我们在 N 个位置进行控制。这是一个巨大的优势，但由于 N 层模型的初始化比较复杂，人们往往会忽略或误解这种优势。让我们再一次从软件角度比较这两种模型。

1. 应用程序逻辑

如图 1.4 所示，两层模型的应用程序逻辑只放在两处。其一是数据库服务器——采用存储过程、触发器和约束条件等方式；其二是客户机——采用域验证的方式，以及其他编码事务逻辑。

如图 1.5 所示，在 N 层模型中，可以采用更灵活的方式控制应用程序。我们仍然可以在数据库中使用触发器、程序和完整的约束条件。如果愿意，中间层可以采用与数据库无

关的方式处理事务逻辑。不仅客户机需要验证，而且需要验证往返中间层的对象，以及往返客户机的对象。最终，客户机当然能够执行特定应用程序的特定逻辑。

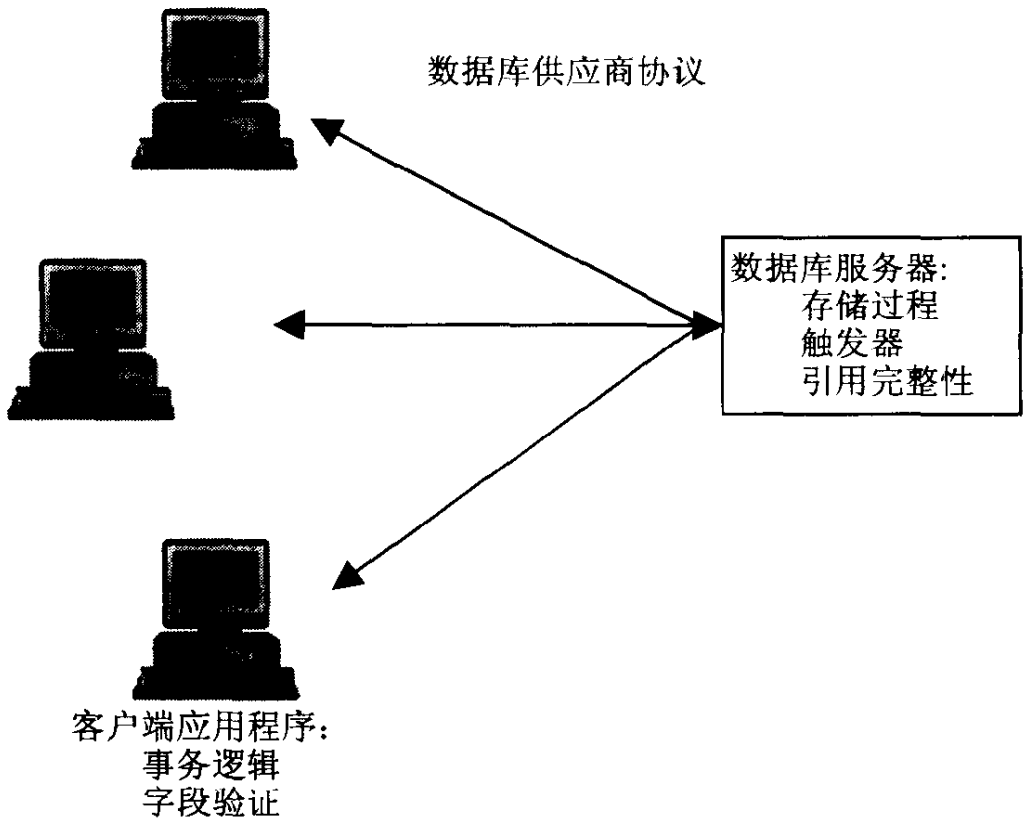

图 1.4　两层模型的应用程序逻辑

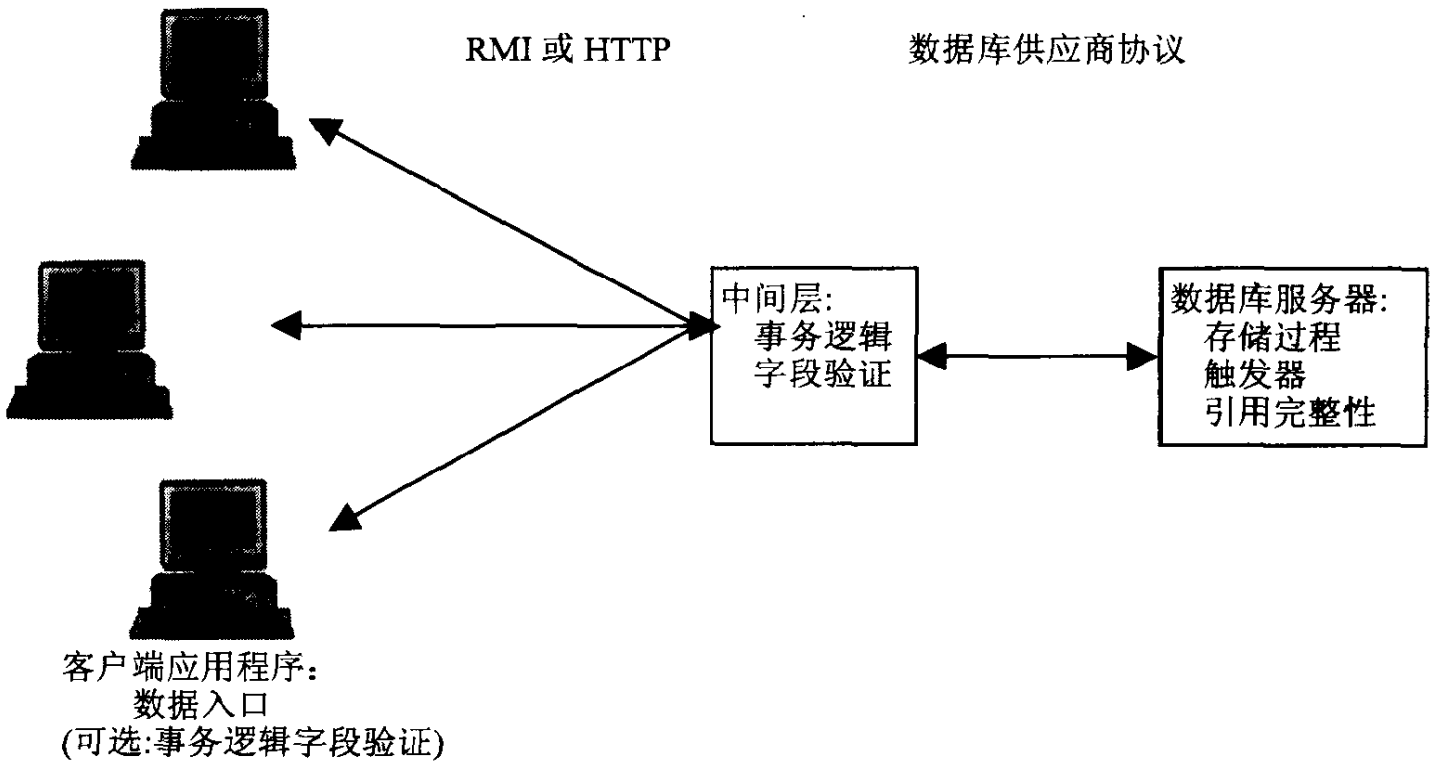

图 1.5　N 层模型的应用程序逻辑

该模型有以下 4 方面优点：