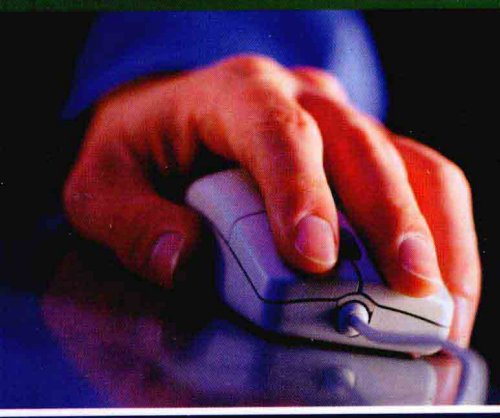
高等学校计算机系列规划教材

COMPUTER



Java EE Web 编程技术教程

刘甫迎　饶　斌　郑显举　杨雅志　编著

電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY http://www.phei.com.cn

高等学校计算机系列规划教材

Java EE Web 编程技术教程

刘甫迎　饶　斌　郑显举　杨雅志　编著

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

内 容 简 介

本书详细介绍了 Java EE 基础；Java EE 的可视化集成开发平台（Eclipse 及运行环境）、Java Applet 及 JDBC、Web 层编程技术、Java EE 轻型框架技术、EJB 技术、Java EE 持久性数据管理、Web 服务与 SOA 技术、Java 消息服务等异步技术和 Java EE 综合使用实例。本书既突出轻型框架 Hibernate、Struts 2、Spring，又有企业 JavaBean（EJB 3.0）分布式重型框架；既有 Web 服务，也有面向服务结构（SOA）新技术，其内容主要集中在企业级 Java 项目所需的重要的 API 和工具上，使本书成为一本较完整的 Java EE Web 编程技术教程。本书共 10 章及 3 个附录，有实例、习题、教学大纲和实验指导书等。

本书可作为本科院校、高职院校计算机及相关专业课程教材，也适合 Web 编程开发人员使用、参考。

图书在版编目（CIP）数据

Java EE Web 编程技术教程 / 刘甫迎，饶斌，郑显举等编著. —北京：电子工业出版社，2010.7
（高等学校计算机系列规划教材）
ISBN 978-7-121-06504-0

Ⅰ. ①J… Ⅱ. ①刘… ②饶… ③郑… Ⅲ. ①JAVA 语言－程序设计－高等学校－教材 Ⅳ. ①TP312

中国版本图书馆 CIP 数据核字（2010）第 118611 号

策划编辑：吕 迈
责任编辑：毕军志
印 刷：北京市李史山胶印厂
装 订：
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：787×1092 1/16 印张：24 字数：614.4 千字
印 次：2010 年 7 月第 1 次印刷
印 数：4 000 册 定价：39.00 元

凡所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。

前　言

在当今网络时代，无论因特网（Internet）、内联网（Intranet）、外联网（Extranet）都离不开 Web 技术的应用，其使用规模和水平已成为衡量某个国家信息化程度的一个重要标志。Web 编程技术已成为计算机领域中最重要的技术之一，它是软件学科中一个不可或缺的分支，是高等学校计算机专业和信息管理专业一门专业基础课，越来越多的人希望学习 Web 编程技术。

随着网络技术尤其是 Web 应用技术的发展，企业级应用对系统各方面的性能要求越来越高，特别是速度、安全、可靠性以及分布式应用等方面，在一定程度上决定着系统能否成功。在这些要求的共同作用下，SUN 的 Java EE（Java 平台企业版）规范利用 Java 编程语言和企业 API 的强大功能，包括 EJB 技术，提供了一种业界领先的 Web 编程技术平台。较之微软的.NET 平台，Java EE 更加适宜大型企业级项目的开发（对中小型企业项目更是游刃有余）。

企业级 Web 应用技术呼唤一本较完整、实用的《Java EE Web 编程技术教程》出台，本教科书拟适应此需要。

本书的主要特点：

（1）体现最新技术。Java EE（Java 平台企业版）是 J2EE 1.5 以后的称谓，是替代日益成熟的 J2EE 的革命性规范。全书以 Java EE 为基础，体现了内容的先进性（详见第 1 章）。

（2）突出主流技术。例如，叙述了 Java EE 的可视化集成开发平台主流技术 Eclipse、其运行环境的 Web 服务器主流技术 Tomcat 和应用服务器主流技术 JBoss，框架环境采用 MyEclipse，并且将之贯穿全书（见第 2 章、第 5 章）。

（3）注重基础性内容。例如，叙述了 Java EE 规范中的 Java Applet（小程序）及 JDBC，为后面章节学习提供了基础（见第 3 章）。

（4）注意全面性。既突出 Java EE 轻型框架 Struts、Hibernate、Spring 的重点（见第 5 章），又有企业 JavaBean（EJB 3.0）分布式可复用组件重型框架的内容（见第 6 章）；既有 Java EE 的 Web 服务，也有面向服务结构（SOA）新技术（见第 8 章）。还叙述了 JMS、Ajax 异步技术（见第 9 章）。

（5）将 JSP、Servlet、JSTL、JSF 作为 Web 层编程叙述（见第 4 章），并把 Java EE 持久性数据管理内容单独作为一章（见第 7 章）。

（6）本书注重理论与实践相结合，突出实践动手能力和实用性。有实例（见第 10 章）、实验指导书（见附录 B），便于读者参考、使用，力图使学生学习本书后便基本可以编制基于 Java EE 的 Web 应用系统。

（7）本书附有教学大纲（见附录 A）、习题，图文并茂，便于学习与教学。

（8）本书作者长期从事 Web 编程技术教材的编写、教学和科研工作，有丰富的教学和开发经验，并将其融入本书中。

本书由刘甫迎、饶斌、郑显举、杨雅志编著。刘甫迎编著第 1 章、第 3 章、第 4 章、第 9 章；饶斌编著第 2 章、第 6 章、第 8 章、第 10 章；郑显举编著第 7 章和附录 A、B、C；杨雅志编著第 5 章；全书由刘甫迎统稿。在编著过程中谢林芮、曾克蓉、李朝蓉等做了许多辅助工作，在此一并表示感谢！

由于水平有限，错误难免，请斧正。

刘甫迎

2010 年 4 月

目　录

第 1 章　Java EE 基础

本章主要介绍了 Web 应用的定义、Web 应用的体系结构、Java EE 的体系结构、几个典型体系结构例子、Java EE 应用程序的构成等，以便对 Java EE 的概念有一个基本的了解。Java EE 是功能强的 N 层应用系统规范，本章还叙述了 Java EE 的由来与发展（从 J2EE 到 Java EE）、Java EE 的新功能、集成开发环境 IDE（JBuilder、Eclipse、NetBeans 的比较）、编译和部署一个 JSP 页等，为后面章节的学习提供了 Java EE 概念的基础。

1.1　Web 应用基本概念

仅仅用了几年时间，在世界范围内，无论是信息的提供方式还是使用方式都因 Internet 而发生了改变，其硬件和软件技术使得每个人能够成为信息的使用者，而且几乎所有人都能够作为信息的提供者。Internet——特别是 World Wide Web（WWW），在非常短的时间内就已经被公众认为是重要的信息共享平台，许多组织都在尽力创建有用的 Web 应用，从而为使用者提供更大的价值。

这些 Web 应用允许使用者在线购买图书和光盘。它们使得企业可以使用 Internet 来实施安全的事务处理。工人利用 Web 应用寻找工作；老板利用 Web 应用寻找雇员；使用由经纪人提供的在线应用可以购进和抛售股票；旅行者则可以利用 Web 应用来预订机票和酒店。这样的例子还有很多。很明显，目前无论是在公共的 Internet 上，还是在数不胜数的公司内部网（Intranet）中，都存在着大量有用的 Web 应用。

本书重点介绍基于 Java 平台企业版（Java EE）规范的企业 Web 应用所需的技术。

1.1.1　Web 应用定义

本书中，Web 应用有一个非常通用的定义——这是一种通过 Internet 技术加以连接的客户/服务器软件，可以传输其处理的数据。通过“Internet 技术”指的是在使用者和提供者之间，组成相应网络基础架构的硬件和软件的集合。Web 应用可以通过专门的客户端软件来访问，也可能利用一个或多个有关的 Web 页面访问，这些页面基于某种特定的用途可进行逻辑分组。这里所说的用途可以是任意一件事情，例如，可以是买书、处理股票订单，也可能仅仅是作为让使用者阅读的内容。

本书所讨论的是 Web 应用，而不只是“Web 网站”。实际上，这二者之间的区别对于理解本书的关键主题相当重要。大多数非技术人员往往不会区别 Web 网站和 Web 应用。除了术语的提法不同之外，对他们而言，其作用都是一样的，都可以使之实现在线购书、在线预订机票、在线购票等操作。不过，对于一名技术人员，两者之间还是存在差别的。如果有人谈到类似于一个 Web 网站的性能问题时，技术人员可能会开始想到其后台的具体细节，会考虑所运行的软件是 Apache 还是 IIS，还有它使用的脚本是 Java Servlet、PHP 或 CGI、perl。技术人员和非技术人员在思路上所存在的不同可能会造成一定的误解。技术人员往往习惯性地把“Web 网站”与服务器端联系起来。而与此同时，他们都知道，Web 应用并不

仅仅包括服务器端，它还需要有网络和客户端。因此，一个 Web 网站（服务器）与 Web 应用（客户机、网络和服务器）并不是一回事。

虽然本书所强调的是服务器端解决方案，在此还会讨论客户端和联网的有关内容，这是因为对于终端用户如何理解 Web 应用，它们有着很重要的影响。也就是说，会讨论 Web 网站内端对端的交互，它意味着由客户到服务器，再返回到客户，这是一个重点。毕竟，大多数使用 Web 的人所关心的就是它的端到端的操作。

可以说使用者是通过客户端软件（即使用 Web 来检索和处理数据的 Web 浏览器或应用）来使用 Web 应用的，这些客户端软件运行在客户端硬件（即 PC、PDA 等）之上，而应用数据的提供以及生产者对具体处理的控制则均通过服务器端软件（即 Web 服务器、服务器端组件软件、数据库等）来实现，这些服务器端软件运行于服务器端硬件（即高端多处理器系统、集群等），将客户端与服务器进行连接（从调制解调器或客户端设备的网络端口到服务器端的网络设备）就形成了联网基础架构。

在此，需要强调一点，即作为服务器软件的一种，要把 Web 服务器特别区别出来，因为它在调度客户端与服务器之间的通信（HTTP）时总是起着一个核心的作用。在本书中，当提到“服务器端”时，一般都包括 Web 服务器。

1.1.2 Web 应用体系结构

1. 抽象 Web 应用体系结构

先不考虑具体情况，应用的体系结构必须能够体现业务逻辑、数据、接口和前面描述过的网络需求。实际上，在描述一个原型的应用体系结构时，最好是从一个非常通用的设计开始。然后再循序渐进地加入一些内容，如 Web 服务应该设置在哪里等。重要的是不要因特定的情况而迷失了方向。时间会改变、技术会改变，但是客户的需求基本上都保持不变。

2. 从客户到服务器

图 1-1 显示了一个非常抽象的应用组成，可以看到用户直接与接口交互，这样，接口需求也要在此满足。一个接口就是应用核心业务逻辑的代理，而业务逻辑则由一组操作组成，这些操作分别对应为手工完成的业务处理。执行此逻辑时，一般需要与数据交互，并对数据进行管理，就是要存储和查询数据。

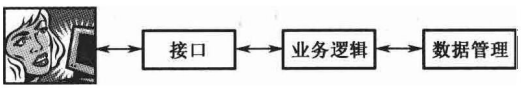


图 1-1 抽象应用体系结构

现在可以增加一些具体细节。一般来说，Web 应用包括一个通过网络与应用交互的用户。业务逻辑的执行可以在本地或远程完成。换一种说法，这表示客户可以很“胖”（本地逻辑），也可以很“瘦”（远程逻辑），一个胖客户和一个瘦客户的差别关键在于业务逻辑是在客户端或是在服务器端。无论是哪一种情况，至少接口不能远离用户，而且数据需要在某些服务器上进行集中。注意：接口总在客户端，而数据管理则总是在服务器端。

最后，还可以采用一种混合的设计，即业务逻辑既在客户端又在服务器端。实际上，

这种设计更为通用。数据验证即为一例，例如，要确保电话号码的形式如“123-456-7890”，尽管一般把它看做一个业务逻辑的问题，但是这种验证往往都通过类似于 JavaScript 等技术在客户端执行。

3．N 层应用体系结构

应用体系结构一般都包括三种基本的组件：客户、网络和服务器。

1）客户

常见的客户有两种类型：一种是人类客户，另一种是自动的、基于软件的客户。人类客户通常有一台带有操作系统的主机并能访问网络。一般使用一个 Web 浏览器来访问 Web 应用，不过也可能使用定制的接口。浏览器采用 HTTP 协议通信。人类客户的一个独特之处在于它与服务器的会话不要求持续的服务，由于人类客户往往会有许多“思考时间”，这使得服务器可以将资源分配给其他请求服务的客户。

较之人类客户所用的主机，自动客户可能运行在一个功能更为强大的主机上。自动客户可能使用 HTTP 协议，也可能使用低级或专用协议实现通信。这一类客户还有可能使用类似于消息等技术进行通信，这样就不会涉及 Web 服务器，但是却会涉及其余的服务器端软件（如应用服务器和数据库）。自动客户不需要“思考时间”，所以可能会不断地向服务器提出请求。

2）网络

在客户和服务器之间的网络通常被称为 Internet，它是由分布在世界各地的许多主机和子网组成的。

主机相互通信时，其数据包将流经多个硬件和软件系统，并路由到最终目标。根据需要，其消息大多数使用 TCP/IP 协议完成通信。IP 代表网际协议，而 TCP 代表传输控制协议。TCP/IP 是一个面向连接的协议，它可以提供服务质量保证。即使底层网络并不可靠，但它也可以确保数据字节可靠地发送到通信双方。

因为 Internet 代表着一个不可靠的网络，而此协议几乎成为了它的默认“语言”。不过，仍要指出，还存在着其他的传输协议，最典型的是不可靠数据报协议（Unreliable Datagram Protocol，UDP）。此协议不是面向连接的，而且不能同样保证 QoS。不过，正是由于它不具备这些特性，反而使它比 TCP 效率更高。尽管 UDP 对于 Internet 应用来说一般是不能接受的，但是对于某些要求高性能的 Intranet 应用而言具有重大意义。

（1）客户端网络元素。客户通常并不直接与 Internet 相连。大多数人都通过一个服务提供商（称为 Internet 服务提供商）或 ISP（Internet Service Provider）来访问 Internet。在对 Internet 上的原始地址发出请求之前，客户往往会首先访问其浏览器缓存，查看是否已经有所需文档。浏览器缓存仅仅是客户主机的文件系统。请求一个页面时，如果它并不在本地的文件系统中，浏览器就会负责获取。提供此页面后，如果它是可以缓存的，则会在本地文件系统中保留一个副本以备日后访问。这样可以提高客户性能，因为如果没有此缓存，新的访问可能还需要消耗网络的往返时间。不过页面不会永远被缓存，因为它有到期时限，另外相应协议可以检查某个特定 Web 对象的更新情况。利用浏览缓存或其他缓存将远程 Web 页面数据进行本地存储（以避免为访问同一内容而要与原始服务器进行通信的开销），这一概念一般称为 Web 缓存。

代理缓存可以是软件也可以是硬件，它对频繁请求的 Web 页面进行缓存，这样 ISP 就不必重复地为其客户获取同样的页面。客户 A 首次获取 Web 页面 X 时，代理缓存会从原始

服务器请求页面，并将该页面的一个副本存储在其缓存中（假设此页面是可以缓存的），与此同时还会为客户 A 提供一个副本。当客户 B 请求页面 X 时，代理缓存只需从其缓存中获得页面，而不必再次访问网络了。这样就可以得到更好的客户性能。不仅如此，多个客户还可以分享此结果。图 1-2 显示了客户 Web 浏览器和中间的客户端缓存之间的关系。

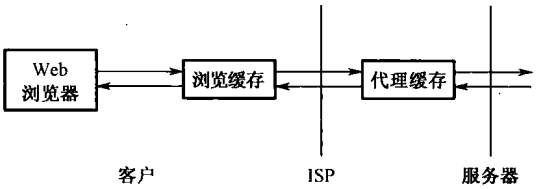


图 1-2　客户端网络基础架构

（2）服务器端网络元素。与客户端一样，服务器端也有一个 ISP；不过，可以对这里 ISP 的选择有所控制。了解带宽限制以及提供商可以访问的主干网络，在设计应用时这一点很重要。

关于网络服务器端部分还有另一个重要的方面，即把可能到来的连接分发给服务器资源的方式。本地的负载平衡器为实现多台主机之间工作负载的分摊，既提供了简单的方法，也提供了很复杂的技术。请求可以基于 Web 服务器的可用性被路由，或者也可以根据请求本身的实质进行分发。例如，图像请求以某种方式发送，而对静态页面的请求则采用其他方式发送。Cisco Local Director 就是一种典型的负载平衡硬件设备。

所平衡的负载通常在 Web 服务器场（Server farm）中进行分发。每个 farm 都由一组设计用来访问相同类型内容的 Web 服务器所组成。各 Web 服务器通常在单独的主机上，建立冗余可以提高 Web 站点的可靠性。

最后，可以在服务器端建立反向代理缓存，对于频繁访问的对象，通过提供对其快速访问来减少对服务器的请求。反向代理缓存的工作和客户端代理缓存的工作很类似，它会保存频繁请求对象。在服务器端这是很有用的，因为即使在多个客户的 ISP 并不相同的情况下，它也允许将经常访问的内容加以缓存。图 1-3 显示了这种部署策略，在此涉及一个负载平衡器、反向代理缓存和 Web 服务器场。

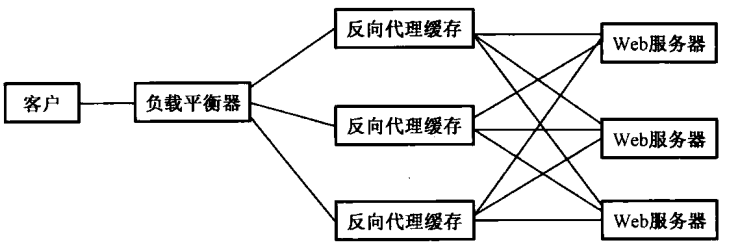


图 1-3　服务器端网络基础架构

（3）界于客户端与服务器端之间的网络元素。它们的存在是通过减少一些请求量特别大的资源的拥堵情况，提供一种通用而经济的服务。不过，如果了解中间缓存是如何工作的，以及 HTTP 协议的工作原理，那么至少可以向网络提供有关的数据的本质特性。有了足够的信息，中间网络元素就可以显著地减少网站上对于静态 Web 页面的负载。

一般来讲，客户端和服务器端之间的通信包括客户 ISP，它能够提供 T1 和 T3 速度，并通过主干网络服务提供商（Network Service Provider，NSP）转发请求。而 NSP 则以 OC-1

和 OC-3 速度实现通信。这些 NSP 提供了对网络接入点（Network Access Point，NAP）的访问，NAP 则是一些公共的交换设施，在此各个 ISP 可以通过一种称为 ISP 对等（ISP peering）的过程相互访问。NAP 分布在世界各个地方，将其合在一起，就代表把 Internet 主干“缝”在一起的所有点，即“针眼”。NAP 上的通信速度相当快，例如，可能达到 OC-12（622 Mbps 或更高），而且采用的是点对点的方式传输。

为了对客户与服务器的连接有更确切的体会，请参见代码清单：例 1-1，这是 traceroute 的输出，traceroute 是一个诊断型网络工具，可以跟踪由客户到服务器的数据包。下面代码中的数据包来自 Carnegie Mellon 大学，其目标为 Yahoo。

【例 1-1】 代码清单 traceroute 输出，描述了 CMU 到 Yahoo 的路由情况。

```
1  CAMPUS-VLAN4.CMU.NET(128.2.4.1)1.744ms 1.052ms 0.992ms
2  RTRBONE-FA4-0-0.GM.CMU.NET(128.2.0.2)37.317ms 54.990ms 75.095ms
3  Nss5.psc.net(198.32.224.254)2.747ms 1.874ms 1.557ms
4  12.124.235.73(12.124.235.73)11.408ms 22.782ms21.471ms
5  gbr1-p100.wswdc.ip.att.net(12.123.9.42)17.880ms 21.404ms23.662ms
6  gbr4-p00.wswdc.ip.att.net(12.122.1.222)13.569ms 10.793ms 11.525ms
7  ggr1-p370.wsw3dc.ip.att.net(12.123.9.53)11.814ms 10.948ms 10.540ms
8  ibr01-p5-0.stng01.exodus.net(216.32.173.185)12.872ms 20.572ms 20.885ms
9  dcr02-g9-0.stng01.exodus.net(216.33.96.145)29.428ms 10.619ms 10.550ms
10  csr21-ve240.stng01.exodus.net(216.33.98.2)10.998ms 32.657ms 19.938ms
11  216.35.210.122.(216.35.210.122)11.231ms 20.915ms 32.128ms
12  www7.dcx.yaboo.com(64.58.76.176)36.600ms 10.768ms 12.029ms
```

可以看到 CMU 是通过主干提供商 AT&T 和 Exodus Communications 与 Yahoo 建立连接的。

最近几年，对于客户和服务器之间这个领域的优化方法也得到了关注，即内容分发。内容分发（Content distribution）是从 Web 缓存中产生的，这是提供商用于复制其内容的一种方法，在复制时需要通过作为反向代理缓存的提供商，如 Akamai 等内容分发者对其内容进行策略性的复制，从而保证客户访问相当迅速，而且也不会涉及原来的服务器。内容分发的解决方案通常针对的是类似于图像等对带宽有压力的对象，这种对象即使不需要服务器端应用逻辑，也会很快消耗掉服务器端的带宽。

4. 服务器

服务器端应用体系结构不仅是最复杂的，而且也是最有趣的。通过对应用的客户和网络部分特性的了解并适当应用，可以从某种程度上对应用性能进行优化，而这些工作还会给服务器端带来更大的影响。可扩展性和性能方面存在的最大难题就在于此。图 1-4 显示了此体系结构的主要部分。从左到右，请求处理器通常是应用请求第一个到达的组件。

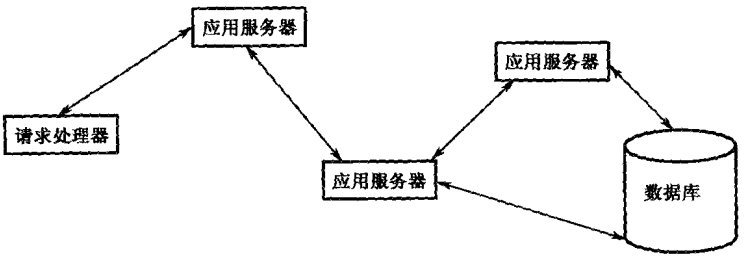


图 1-4　服务器端组织

请求处理器的一个例子即为 Web 服务器。Web 服务器有两个作用：对其能够处理的请求予以解决，如果不能解决则重新路由。对于静态的 Web 页面，Web 服务器可以通过访问文件系统来得到所需要的页面，从而在本地解决请求。作为一个请求路由器，Web 服务器可以确定请求的类型，并将它派给合适的处理程序，在本书中处理程序一般指的是一个 Java Servlet 引擎或 Servlet 容器。Servlet 容器会调用合适的 Servlet 实例，即某个 Java 类的一个方法。此 Servlet 将负责解开打包的请求参数，而且很有可能还将构造 HTML 响应。在此期间，它可能会完成许多工作，例如，与一个应用服务器或数据库建立联系以实现查询处理。

请求处理器的作用是识别请求的本质特性，并将其发送到一个实例上，该实例具有相应的功能可以执行所需的业务逻辑。这种机制通常被称为一个应用服务器。例如，应用服务器往往用于查找某个应用逻辑的实例。此逻辑的形式可以是一个 Java 类，并实现为一个企业 JavaBeans（Enterprise JavaBeans，EJB）或一个 CORBA 对象。EJB 和 CORBA 等技术均为中间件技术，可以通过应用服务器使应用逻辑更具健壮性、可扩展性和互操作性（应用服务器要对其进行管理）。

在一个设计合理的系统中，这些工作都是在后台完成的。应用逻辑的代码编写与应用服务器无关。它们明显的不同之处在于：一个提供访问，另一个提供业务功能。问题在于要为应用服务器提供自动管理业务逻辑部署的能力，在此要保证底层代码透明性的同时，还要为服务器机制提供最大的灵活性。本书后面将讲解 EJB 容器和 EJB 如何支持技术人员构建真正的应用服务器。

基于现有的应用数据，为了解决一个应用请求，数据必须在其所存储的位置被访问，这个位置一般是一个数据库。数据库要存储有关应用状态的信息，这包括用户、简表等所有作为应用数据组成部分的内容。但是，除了表示应用状态的数据以外，数据库还可以存储大量的业务逻辑。通过存储过程、触发器和数据库约束等形式即可达到此目的。数据库中的数据根据某种数据模型加以组织，这是一个表示数据应该如何存储的逻辑规范。在物理上，数据与数据表相关联。表数据（以及其他结构）则要写到磁盘上，这与文件系统中的数据完全类似。

注意对于应用服务器和数据库还可以有其他的客户，可能是基于消息的系统，也可能是原有的系统，它们使用如电子数据交换（EDI）等技术实现通信。这些客户往往是另外一些企业，而不是单独的用户，而且他们一般以批处理的模式与服务器端软件进行交互。

1.1.3 基于层的设计

在客户请示得到处理之前所经过的区域即称为层。每个层都与一个或多个逻辑相关联：表示、业务或数据访问，等等。2 层应用一般由表示层和数据层组成，例如，一个直接访问服务器端数据库的客户应用程序。3 层应用则更进一步：存在一个更瘦的客户端与某个 Servlet 或 CGI 程序联系，而后者再与数据库通信。表示层和数据层更明显地被分离，而服务器端也从数据访问逻辑中区别出了业务逻辑。一个 *n* 层应用则包括 3 级或更多的请求处理，其中包括数据库。对此可以举一个例子，一个客户与某个 Servlet 联系，而 Servlet 则与一组应用服务器通信，每个应用服务器均可以访问数据库。当前的许多 Web 应用系统都是 *n* 层的。

可能会感到疑惑，*n* 层的设计是不是有必要呢？其回报如何？直观地看，它使客户/服务器的连通性变得更为复杂。通信路径现在包括了客户→服务器 1→服务器 2→……→服务

器 n→数据库。看上去好像一次发送需要更多的中继，这意味着更大的延迟，同时出现故障或瘫痪的机会也越多。是这种设计不太适合吗？答案是：只要设置得当，它就会带来很好的效益。

1）提高模块化和组件的重用性

多个层意味着可以把工作进行划分，并且可以将部分问题交给独立的模块完成。但是如果不这样做又会怎样呢？如果所有的工作都在单独的服务器端程序中完成，情况又如何呢？

例如，假设需要建立一个服务器端模块，支持 Web 用户预订图书。在编写此模块时需要确保当 Web 用户对其下订单，当按下“确认（Confirm）”按钮时，应该完成以下事件：

（1）交互式客户请求的接收。

（2）信用卡检查。

（3）工作订单的创建。

（4）客户简表的更新。

（5）图书清单的更新。

（6）交互式客户响应的生成。

可以把所有这些操作都放在一个 Confirm-and-Pay-for-a-Book 模块中完成。但是假设两个月以后应用需求有所调整，需要处理 DVD 订单。想象一下会出现什么情况？需要创建一个新的模块，可以把它称为 Confirm-and-Pay-for-a-DVD 模块。以下是其所必需的操作：

（1）交互式客户请求的接收。

（2）信用卡检查。

（3）工作订单的创建。

（4）客户简表的更新。

（5）DVD 清单的更新。

（6）交互式客户响应的生成。

是不是看上去很熟悉？不能重用订书模块中的功能。此时，可以创建一个更为通用的模块，称之为 Confirm-and-pay-for-a-Product，并以此来解决问题。

现在假设要求支持某种机制从而可以处理大量新产品的预订。假如，有一个销售商每天晚上把订单发给您，您需要处理这些订单。在这种情况下，无法简单地复用前面的通用产品模块，因为请求的类型基本上是完全不同的。只得重新创建一个新模块，它必须包括以下操作：

（1）接收批请求。

（2）对于每个请求进行以下操作：① 信用卡检查。② 工作订单的创建。③ 客户简表的更新。④ 产品清单的更新。

（3）生成一个批总结响应。

对于未经精心设计的模块和组件，无法复用其中的独立操作。这一点完全与程序设计类似，如果不能很好地进行模块化，应用的复用性也很小。因此，往往不得不对功能进行复制而造成代码迅速膨胀起来，这也常常导致应用操作的不一致性。

模块和组件的可扩展性还可能存在死角，由于没有复用功能，需要将它复制到多个模块中，这样就会消耗掉服务器主机上的内存，而这是完全没有必要的。相反，如果将这些模块和组件分解为细粒度的组件，就不仅能够复用其功能，而且还可以有效地消除