体系结构

第 1 章将对应用服务器体系结构作一个整体描述，说明它在业务环境中所起的作用，该章同时将对应用服务器中的多层客户／服务器计算、分布式应用和中间件等关键技术进行初步介绍。

第 1 章　什么是应用服务器

最近几年以来，许多软件供应商纷纷发布了他们自己所谓的应用服务器软件包。Inprise 公司、Oracle 公司、BEA 公司以及其他许多公司均相继进入应用服务器领域，针对企业级计算业务不断扩展他们的产品生产线。那么，究竟什么是应用服务器呢？

本章将分析一下应用服务器技术为何会在下一代企业级计算中发挥重要作用的原因，主要包括如下几个主题：

- 二层计算模式与多层计算模式的区别
- 为什么要选择多层客户／服务器计算模式
- 应用服务器的作用
- 应用服务器的成本及缺点
- 从传统的客户／服务器向 N 层计算转移

1.1　二层计算模式与多层计算模式的区别

与传统的二层客户／服务器计算相比，多层计算具有很多优势。在激烈竞争的压力下，数据库产品由于不断提高性能和增加新的功能特性，已经发展得非常成熟。像 Microsoft Access，Borland Delphi 和 C++ Builder 这样的客户端开发工具已经变得非常容易使用，以至于很多程序代码都可以自动生成，甚至网络环境也比以前更加容易安装和维护。

然而正如大多数客户／服务器的开发人员很快就会发现的那样，这简直是太容易了。于是，新的应用在服务器上成倍增加，随着计算机设备的大幅度降价，更多的客户端机器不断连接上服务器。这样在很短的时间内，服务器就超负荷运行了。即使把所有的内存槽插满，增加更多的 CPU，花费大量的资金升级网络，用户仍然抱怨系统响应时间太慢。

为了解决这一问题，业界现正在推行三层或 N 层(多层)客户／服务器计算模式。在该计算模式下，服务器本身就是一个计算机网络，可不断进行扩充以满足不断增长的客户端需求。该计算模式不是由客户端软件直接与数据库服务器进行通信，而是由一个叫做应用服务器的中间层软件向客户端提供服务（见图 1-1）。这样就最大限度地减少了与数据库服务器的连接数量，并可将相关的处理过程分担在多台服务器计算机上完成。同时，由于大多数处理过程转移给了应用服务器，它也大大缩小了客户端软件的体积。现在，客户端软件可以变得像在网页上填写表格那样简单。

由于大多数处理过程已经转移到了中间层，所以，建立应用服务器可能将变成一个困难而复杂的过程，需要一整套全新的工具和技巧。在应用服务器方面，每一位软件供

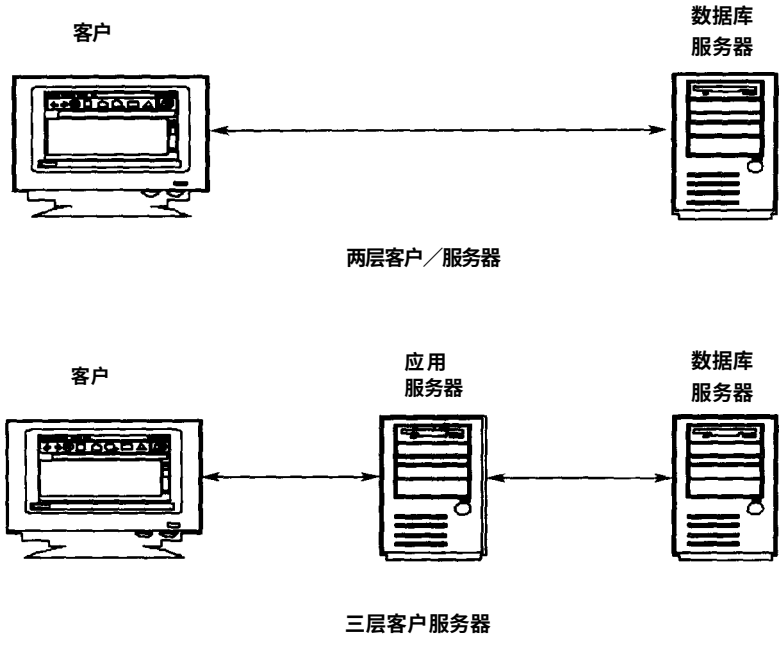

图 1-1　两层与三层客户/服务器的比较

应商都在努力推销自己的中间件工具，但是使用这些工具的技术仍停留于发展的早期阶段。在使用这些工具方面，能获取到大量的信息资源，如相关软件的体系结构、提供的服务、协议以及如何使这些软件工具运行起来等等。但即使到现在，也很难找到如何构建这些软件的有用的实际知识。本书将讨论一些相关的原理和实例，以帮助软件开发人员自己开发中间件工具解决相关的业务问题。

.2　为什么选择多层客户／服务器计算模式

作者的大部分咨询工作都是与保健管理这一业务有关。在这一行业中，随着业务规则经常发生变化，有大量信息必须进行仔细审查。这就要求软件必须能有效和灵活地处理大量的数据，特别因为该行业要求尽力使不断增加的开支与保健质量需求相平衡，同时对政府的各项规定予以考虑。作者的工作是从基于大型机系统开始的，但是几年以前，作者开始为新开始业务和医药特定相关的小型工作机构建二层的客户／服务器系统。

随着客户/服务器软件开始增长，作者发现了几个问题。首先，当一定数量的工作站在同时使用时，系统响应时间就变得很慢。升级数据库服务器后有所缓解，但只有当增加更多的工作站后，效果才能够显现出来。其次，开发工具对于交互式和屏幕操作相关的软件开发很有效，但缺少执行与大型机批量处理类似的能力，仍然不能迅速处理大量的数据。像 Crystal Report 这样的工具在使用中有些帮助，但它们不能提供应用软件所必

须的计算的灵活性。

最终促使作者研究多层计算的是有关业务规则处理的问题。当一个请求提交到数据库服务器时，一些关键信息（如：病人、医生、诊断结果、服务日期和医疗过程代码等）必须经过一整套业务规则的审查，例如：

- 该成员是否有资格接受服务？
- 该服务是否经过授权？
- 对于诊断结果，该服务是否适当？
- 该医生是否有权提供这些服务？
- 该服务项目是否适合于病人的年龄和性别？

在请求被结算之前，常常要经过二三百项规则的效验。一旦这些规则都通过效验后，各项费用和收益明细必须能够相匹配，以决定医生应该得到多少，病人应付哪一部分。这些代码最好尽量用 Visual Basic 或者 Delphi 进行编写。使用这些工具也许能够完成这些功能（我们已经这样做过），但是若没有某种形式的后台处理过程，就不可能在有限的合理响应时间内完成这些操作。我们的解决方案是：在批处理模式中的空闲时间中处理这些业务规则，但是这使得操作者没法获知他们的工作何时完成。

在多层计算模式下，可以将业务规则处理转移到一台单独的服务器上，当请求信息输入后便可测试这些业务规则。许多数据表可以加载到内存线型业务对象中，以加快处理过程。更为复杂的规则检查可以放在较低优先级的后台运行。判定支持和报告的批处理可移到独立的机器上，在这个独立的机器上，数据能被复制成一个数据库应用。

1.3 应用服务器的作用

除了前面所描述的功能之外，应用服务器还可以解决传统的二层客户／服务器计算中的许多其他不足之处，同时提供许多新的优点。应用服务器通过提供可升级软件，帮助系统管理员在多台机器范围之间获得更好的系统性能。它为系统设计人员提供清晰定义的逻辑边界，使设计人员创建与实际业务更为接近的业务对象。从某种意义上来看，软件开发也更加容易，因为程序代码被分解得更小，更小的模块和服务意味着也更加容易重用。应用集成也变得更加简单，使用中间件服务进行数据格式转换，简化了在不同供应商的机器之间的通信过程。

1.3.1 可升级性

多层客户／服务器计算最明显的优势就是可升级性，因为计算工作量是在几台机器之间进行分担。无论花费多少钱购买最先进的超级服务器，任何单台计算机的处理能力都是有限的。将同样数量的资金选择几台中等的服务器，可能会获得更强的计算能力，同时甚至还可能节省花销。

真正省钱的地方在于节省了升级费用。超级服务器可能有一定限度的升级能力，但是当它无法再升级的时候，将不得不被更加昂贵的超级器服务器所取代。公司不仅要承担更换新服务器的开销，还要复制原有机器中的所有内容。而使用分布式服务器时，唯一要增加的花费就是一台新增加的中等服务器的费用。

1.3.2 分布式处理

另一个优点就是数据库和应用服务器可以尽可能地按照靠近需要完成工作的地方而分布。假如在旧金山进行订单输入，在亚特兰大完成生产和存货清单输入，那么使数据库更接近业务工作地点，就比将数据都存储在芝加哥的总部要好得多。这样，网络传输将降低到最小程度，因为在旧金山本地就可以完成订单输入，并且在旧金山与亚特兰大之间只需通过最小数量的网络路由便可检验存货水平。若芝加哥需要管理报告，数据可以在芝加哥和亚特兰大进行汇总，然后将统计报表发送到芝加哥。

分布式处理还可用于将远程数据进行本地化存储。这就更加减少了网络传输，甚至在无法进行远程连接的时候也可进行业务数据处理。在旧金山的存货清单对象包括现有存货数量和近期订单保留的数量。它可以定期向亚特兰大发送消息来保留这些订单项，并得到实际存货的新数量以进行更新。在网络不通时，订单输入可继续进行，因为本地计算机保存有目前可用存货数量的近似值。一旦网络恢复，更新消息便可以纠正这些差异。

.3.3 可重用的业务对象

应用服务器是一个反映业务处理过程的服务和对象的仓库。由于这些过程是用业务语言描述，而不是用计算机语言进行描述，这对开发人员来说，更容易将业务需求转换成有效的软件设计。随着软件开发人员与业务人员之间的交流更加明确，软件设计也将会更加贴近实际业务的需求。于是，软件将可能更早交付，同时将只需更低的生产成本。

一旦应用服务器实现完成后，已经开发出的对象和服务可为另一项目所重用。大多数功能都对开发团队开放，而不是仅限制在一个项目中。 Visual Basic 提供一套被反复使用的 GUI 控件，大多数中间件实现需要有一个通用的标准接口和组件模型，这样将使重用更加容易和更有效地降低成本。

1.3.4 业务规则处理

大多数二层客户／服务器强调以数据为中心的软件设计，由客户端软件提供一个用户接口来维护存储在数据库服务器中的数据信息。几乎所有的操作都在数据库以外的客户端进行；这种处理方式需要传统的网络传输。一些处理过程可以移到数据库服务器，但这些需要专用存储过程语言，这些存储过程语言都局限于每个单独的数据库服务器供应商。

应用服务器的开发强调业务对象的构建，而不是数据如何存储。每个组件同时包括服务和数据，这样不仅可允许更宽范围的数据完整性检查，同时有能力从对象包含的数据中派生出附加信息。服务也可以封装模拟实际业务的业务规则和处理过程。当发货单输入时，发货单对象可以智能地检查顾客对象，以保证其信用度足以被批准。当服务请求存储数据时，这些规则和处理过程是自动执行的。

1.3.5 跨平台集成

由于大多数机构已经有大量的应用软件在使用，中间件供应商在跨平台集成方面已

经作了相当大的努力。因此，开发人员不必关心底层的数据格式转换、字节顺序表示方法，或其他供应商特定的数据信息。中间件还可以使用高级接口定义语言（IDL），与多种编程语言相连接。当在一种语言中声明函数时， IDL 编译器会生成在许多编程语言中的转换代码。这种接口定义语言允许一种语言编写的程序调用另一种语言编写的函数或者访问对象，即使它们处于不同的计算机上也同样如此。

1.4 应用服务器的成本及缺点

实现一个应用服务器尽管有很多优点，但这项技术并不适用于每种项目。多层开发需要大量基本的前期投入，这些投入也许并不会有立竿见影的效果。应用服务器是一个复杂的软件，需要一整套新的技术和工具。大多数中间件软件包都是基于面向对象的设计和编程概念的，这些要求有很高的抽象水平和更高的学习能力。许多中间件软件同时也依赖于组件结构体系，必须严格遵守新的编程标准。组件和模块也必须尽量普遍以便于日后重用。应用服务器技术解决了许多问题，但同时也带来了许多它自己的困难之处。

1.4.1 长期投入

采用应用服务器体系结构是一个长期的、企业级的投入。对于必须以像“互联网时代”的高速度的项目，或者为仅进行功能有限的单个项目而言，这并不是一个合适的选择。这是一种企业级的结构体系，需要新的硬件配置、中间件、编程模型、管理工具，以及最重要的是一种全新的看待软件开发的方法。开发第一个项目并非易事。要花大量时间进行试验纠错、评估工具、学习中间件的特性、创建基本结构而不是应用。单纯从一个单独应用的角度来看，它决不会物有所值。这种技术只有作为创建一个企业新的体系结构的第一步时才会有意义。

1.4.2 中间件的获取

一个或多个中间件包很可能已经驻留在你的硬盘上了。 Microsoft 公司已将其组件对象模型体系结构(COM 和 DCOM)与最新版本的 Windows 捆绑在一起发售。Microsoft 的事务服务器 (MTS)正准备和 SQL Server 7 一起在 Windows NT 平台上发布。Java 开发包提供一个称为 RMI （远程方法调用）的简单的对象请求代理(ORB)，该软件包括在 Java Software Developers Kit(SDK)1.1 版或更高的版本中。如此，为什么还要选择其他的中间件解决方案呢？

大多数这样的中间件包都与一个特定平台捆绑在一起，但是一个综合性的中间件解决方案必须能跨越多种计算机平台、编程语言和数据库。如何选择中间件依赖于现有的硬件和程序语言，以及未来的扩展需求。 COM、MTS 和 RMI 都是供应商、编程语言或平台特定的软件。如果一个机构已经是统一为 Microsoft 或 Java 平台，这不会有什么问题。然而，这其中的任一种选择都可能限制机构未来扩充和增长的需求。

最初的购买价格也只是中间件成本的开始。任何选择也都必须考虑到人员培训、硬件和网络的投入、编程和管理的成本。培训和启动的开支可能大大超过购买甚至是最昂贵的中间件软件包。

1.4.3 新的思维方式

多层客户／服务器同样要求在考虑软件方面具有新的思维方式。尽管编程已经是一个相当抽象的能力，而面向对象的软件设计和编程则要求更高的抽象水平。与单纯的面向过程的执行方式不同，面向对象的方式需要同时在几台计算机上运行的多个进程取代单个连续的执行，面向对象的方式需要将在几台机器上同时运行多个进程之间的交互情况同时显现出来。

在整个项目执行过程中，能从外部寻求到各种可用的咨询服务以引导项目并得到培训机会，但这却需要付出非常高的费用。现在也有大量可用的工具能帮助完成这种转变，但每一种这样的工具却又需要增加额外的购买和培训费用。明智的投资能够大大增加成功的机会，若投资方向不对就可能得不偿失了。

1.4.4 编码牛仔的结束

在名为“ Coding Cowboy and Independent Systems”《编码牛仔与独立系统》一书中，Warren Keuffel 和 Bryce.Carey (Keuffel 1998）举了一个美国西部的例子。在边远地区的牛仔生活独立，不为外部世界所困扰。当铁路穿过该地方的时候，这种独立精神就突然发生变化了。若牛仔们无法将他们穿过铁路的时间与火车经过的标准时间表相统一，牛群就可能被“ 捕捉器 ”捕获。直到最近，程序员还可能制定自己的规则；但自从互联网和电子商务出现后，软件开发人员现必须遵循公共的标准，否则，他们的“ 牛群 ”就会被铁路“ 捕捉器 ”所捕获。

组件和中间件体系结构需要遵循更多的规则和标准。对象和组件必须适应严格的标准，并准确实现已定义的接口方法。现在，大多数这些功能都可由编程工具实现，但设计和结构考虑必须符合这些标准，否则所完成的应用将可能根本无法运行。开发人员同时还必须密切配合，以确保接口和通信方式相一致，以及各种对象和模块之间能够相互协同工作。

1.4.5 软件的重用

软件的重用正如它可能会有许多问题一样，它同样会带来许多好处。组件不仅要满足现有目标，同时还要预先考虑未来的需求。实现和测试可重用软件需要花费更长的时间，而且开发费用也会显著增加，但是一般来说，它的收益要大于这些额外成本。额外的成本和可能的困难将被软件的灵活性和可升级性所抵消。同时，由于应用服务是面向整个开发团队开放，并且组件可以得到重用，所以将来的成本将会减少。

1.5 从传统的客户／服务器向 N层计算转移

从传统的客户／服务器向多层体系结构转移需要一定的时间和进行规划。管理层必须能支持这种转换，并愿意接受从前期到后期增加的各种费用。体系结构、中间件和开发工具必须经过评估和选择，然后，为了支持开发，还必须购买和安装服务器和网络设备。同时，必须建立综合的培训策略，以使开发人员能够很快地掌握这些新工具。当然，所

有这些工作必须建立在支持现有计算环境的前提下进行（Mowbray, 1997）。

通常最好的方法就是选择一个小型、高度可视化的试验项目。由于大多数时间都用于决定体系结构问题和学习新的软件工具上，小型项目将最大程度地缩短开发时间。与此同时，该项目也必须能够产生切实、可见的好处，以验证技术和对余下的转换工作所需的资源和时间花费是可行的。

一个工作流跟踪应用通常能提供一个很好的初始项目——例如顾客询问、电话跟踪、工作日程安排、软件问题跟踪或者其他类似的应用。这些应用仅有一定限度的数据输入，需要业务逻辑将对象从一个状态移到另一个状态（如将问题转交合适的人，从“开始”到“完成”改变对象状态等等），不需要大量复杂的数据存储。同时，应用逻辑包括几个创新的方法，以使设计者不仅仅考虑数据存储和检索的问题。

像这样的试验性项目也会最大程度地减少最初的投入。开发用的机器可以进行重新配置，以适应新的体系结构。软件工具和中间件在 30 天~90天内的评估版本可以从供应商的互联网址上免费下载。开发人员通常都会抓住机会利用这些新技术。只要记住开发的重点在于必须在短时间产生切实成效就行，否则开发人员就只有继续进行以前的工作。

一旦测试项目实施完成，并且将该技术卖给了投资方，就是确定体系结构和中间件的时候了。把整个客户／服务器体系结构策略做成文档，并开始开发接口和开发标准。培训开发团队，确定需要的服务器和网络设备。同时，还要记得购买从网上下载的中间件和工具的许可证。一旦这些工具到期后，所有的开发工作就会骤然停止。

最后，制作一个开发计划，以周期性地检查体系结构和开发标准。确定相关的测试标准以提高进度。同时还要不断接受新的开发工具和方法，评估看来有前途的产品和方法。不断提供训练以确保初始的体系结构和开发文档不会过时（McConnell，1997）。一切到位后，就可以正式开始多层客户／服务器软件的开发。

1.6 小结

应用服务器和多层计算是一项新兴技术，它可以为各种行业带来许多好处。这种技术不应只是简单地使用，而是要有机构对此进行强力的投入。这种新技术可以解决与传统大型机和二层客户／服务器相关的许多问题。随着这一技术的不断成熟，它必将在软件开发过程中发挥至关重要的作用。

- 应用服务器的建立，是将许多计算机通过连成网络，以提供可扩展的、可升级的应用平台。
- 二层客户／服务器是一项成熟的技术，它可以解决基本的业务问题，但是却很难处理复杂的业务逻辑和很高的用户访问量。
- 应用服务器技术通过增加一个中间应用层将业务处理过程隔离，从而改善了二层模型。
- 分布式平台通过在许多计算机之间的代理工作，可提供更好的计算能力，同时也更容易扩充以适应不断增长的需求。
- 由于将开发工作分解成更小的任务，并提供代码重用的框架，软件开发得到大大增强。

- 应用服务器确实需要额外的工具和投入，这在取得效益之前是所必须承受的。
- 运用实验项目开始着手应用服务器的开发，以确保相关平台能够满足实际的业务环境。

参考文献

Keuffer, Warren, and Bryce Cargy. "Coding Cowboys and Interdependent Systems." *Software Development Magazine*, April 1998:31,32.

McConnell,Stephen.*Software Project Survival Guide*.Redmond, Washington:Microsoft Press,1997.

Mowbray,Thomas J.,and William A.Ruh. *Inside CORBA-Distributed Object Standards and Applications*. Reading ,Massachusetts: Addison Wesley Longman,1997.

第 2 章　应用服务器剖析

根据软件供应商提供的文字材料，转向多层客户／服务器计算看起来似乎很简单，只需要购买几个软件产品、创建一些 Web 页面并编写一点点应用程序代码即可。最近，在微软公司的产品介绍会上，一位公司代表用不到 10min 的时间，就创建出一个简单的三层应用程序（微软公司，1998）。这位代表另外再用几行 VBScript 代码就创建出一个 Web 页面，显示可见大约 20 行的 Visual Basic 代码已经安装在事务服务器上，然后再用鼠标点击几下，一个验证顾客号的应用程序便大功告成，一切看起来是多么容易。但事务服务器代码写得很差，如输入顾客号 123456789，事务服务器仅仅返回“ credit OK”（信用认可）信息。

微软公司不是唯一一家使用这种方法推销中间件产品的。尽管各供应商使得开发过程看起来多么容易，但这些产品确实是非常复杂的软件。光学习编程约定和协议就可能需要几个星期，而要生产出能解决实际业务问题的真正代码可能尚需要几个月的时间。对开发人员而言，如微软公司的事务服务器（Microsoft Transaction Server ）之类的工具，可能使编程工作变得稍微容易一些，如提供通信协议和开发框架等等，但即使是最简单的服务，也可能远不止 20 行代码。

本章将从体系结构的角度考察应用服务器，同时将分析几类主要的中间件软件。本章包括如下主题：

- 应用服务器体系结构概述
- 中间件：应用服务器的黏合剂
- 中间件分类
- 将中间件应用到应用服务器体系结构中
- 可选的应用服务器体系结构
- 应用服务器整合

2.1　应用服务器体系结构概述

应用服务器包括客户／服务器软件解决方案的中间层。用户接口程序向应用服务器请求服务，然后，这些服务向数据库或其他应用服务器存储或检索数据。在这两者之间便是业务对象集合，以负责执行服务和业务规则。如图 2-1 所示。

服务接口层相当于应用服务器的“前门”。每个用户接口程序得到授权，可使用一套服务或远程过程，这些服务或远程过程隐藏了所有业务对象的细节和应用服务器上的持

久数据。服务可能是一个获取顾客数据的请求，或是将银行存款发送到指定顾客的账户。然后，这些服务封装在服务接口对象中，向用户接口程序提供一个简单对象，并由该对象处理所有与应用服务器之间的交互过程。

业务对象层是许多软件对象的集合，这些软件对象封装了一定的业务过程和规则。一个设计良好的业务对象应该是用业务术语而不是程序语言进行描述，并应该具有一定的普遍性，以便于不同的应用程序所重用。服务接口层是基于应用程序、进行特定服务的应用程序。与此相反，业务对象层从功能上更接近于模拟真实的业务过程。

持久层作为一个对象代理，在持久存储设备上创建和存储业务对象，或与其他遗留系统进行接口通信。当需要业务对象时，通过调用持久层从关系数据库或其他持久存储设备中检索数据，并用检索到的数据创建新的业务对象。一旦不再需要某一业务对象时，持久层负责在将该对象从内存中删除之前保存数据。持久层的设计特别依赖于现有的数据结构和业务对象层的要求。

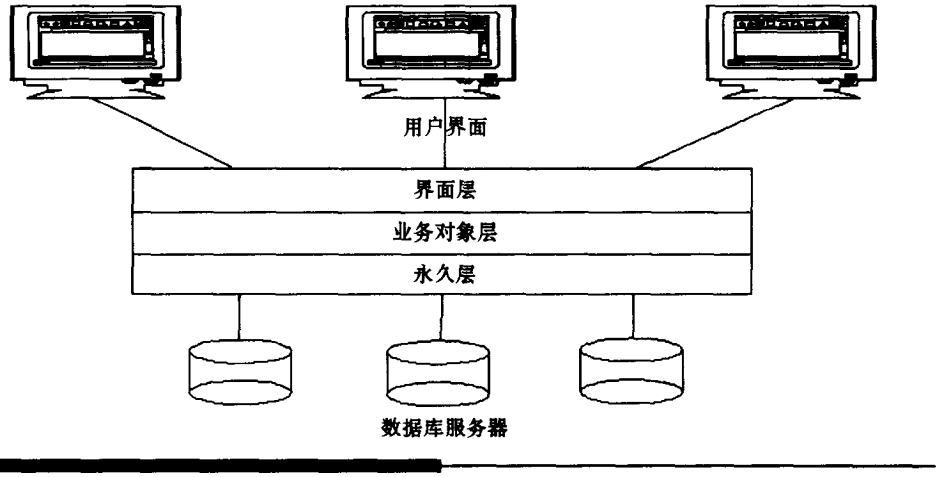

图 2-1　应用服务器层

一个或多个中间件产品将所有这些层绑定在一起，它允许一台机器上的程序调用另一台机器上执行程序的函数，或互相进行数据传输。各中间件之间有着许多不同的编程模型、通信体系、内置服务和管理工具等等。选择最好的中间件体系结构将大大增加实现应用服务器的成功几率。

2.2　中间件：应用服务器的黏合剂

在探讨应用服务器层的每个细节之前，应该先理解中间件的概念和它在应用服务器体系结构中所起的作用。简要地说，中间件就是一类提供多台计算机上程序与程序之间进行通信的软件。应用服务器使用中间件进行客户软件与服务接口的通信、持久层与数据库的通信，及更经常的是对象与对象之间的通信，以提供跨越多台服务器计算机的可升级特性。

中间件提供多台计算机间的通信、编程语言以及数据表示方法。对于编程人员而言，

调用同一程序内的函数与调用远程计算机上的函数几乎没有什么分别。中间件可以使在Web 浏览器上运行的 Java 程序访问在 IBM 大型机上用 Cobol 语言编写的程序函数,就像使用另外的 Java 对象提供的方法。访问远程对象通常需要一些额外的设置，但是一旦该设置完成，这些远程对象的位置对于编程人员而言就变得相对透明。

尽管编程变得透明，但是，中间件对分布式处理提供的支持可能会变得复杂和困难。在分布式环境中，不同程序的数据表示方法、字节顺序、浮点标准以及参数传递方法等可能存在一些差异。大多数中间件解决方案提供了解决大部分问题的服务。这些服务包括配置管理、转换不同的数据格式、以及对在不同编程语言间进行参数传递时提供一致的协议。提供的目录和命名服务可以定位连接到网络上的计算机的功能函数。通常还有生命周期管理和负载平衡功能，以确保远程程序在需要的时候驻留在内存中，并且各种执行程序可以在计算机之间进行有效合理的分布。

出于对透明性的需求，许多中间件体系结构都是协同努力开发的结果，并最终变成了业界标准。计算机制造商已经形成了诸如 Open Software Foundation(OSF)和 Object Management Group(OMG) 这样的协作组织，创建各种标准和规格说明，以保证在硬件平台和软件实现之间具有互操作能力。像开放软件组织（OSF）的分布式计算机环境(DCE, Distributed Computing Environment ）和对象管理组织（OMG）的公共对象请求代理体系(CORBA,Common Object Request Broker Architecture)这样的标准已经成为许多中间件实现的基础。在业界广泛支持的基础上，这些体系结构已经能够解决应用服务器的多种需求，并提供了与遗留系统进行集成的工具。

微软公司试图超越行业标准，发行了分布式组件对象模型（Distributed Component Object Model),简称 DCOM。它是微软公司最初的 COM(在 Windows 2000 中将被 COM+ 取代）的扩展。DCOM 是一个基于组件的分布式对象标准，这一标准主要限于微软公司的 Windows 平台，同时，其他一些平台上也有有限的第三方支持。相关的技术还包括 ActiveX, DNA 和 OLE 等。该体系结构的一个主要优势就是，这些支撑软件已经集成进 Windows 操作系统内部，因此中间件软件已经驻留在大多数桌面机上面。微软还提供各种体系结构选择，包括分布式对象、事务服务器和消息队列等。其缺点就是平台有所限制以及组件模型太复杂等。微软体系结构尽管不像 DCE 和 CORBA 那样有广泛的业界基础，但它对于统一使用符合微软产品的机构来说可能还是一个很好的选择。

2.3 中间件分类

中间件市场仍然在不断发展，但是几大标准的体系结构已经占据了主导地位。每个标准都致力于解决特定的体系结构问题，并适应多种编程模型。尽管不同类型的中间件在实际上有所重叠，并且某些供应商还提供各类中间件的组合方案，但是，大多数中间件体系结构均可以归结为以下一种或几种类型：

- 远程数据库协议
- 远程过程调用
- 分布式对象
- 事务处理监视器

- 消息代理
- 商业应用服务器

在随后描述每类产品时，提到的特定的供应商和产品仅作为举例使用。之所以引用这些名字，是因为他们来自于确定的公司，或产品名字容易识别。

2.3.1 中间件服务

正如操作系统可以提供许多功能，如支持文件系统、串行通信、打印机假脱机管理、日期和窗口管理一样，中间件产品也可提供一系列服务，以满足中间件编程人员的需要。这些服务虽然不直接用于进程间通信，但所提供的功能，可使中间件编程和管理变得更加容易。下面列举部分最常用的服务：

- 命名　命名是最通用的服务，可以在一个文本字符串（名字）和远程对象或过程之前建立关联。远程标识符通常是一些很难辨认的位字符串（如67474-932-8209943-21002 等），操作非常困难，并且几乎不可能输入。通过对每个进程提供一个网络范围的句柄，命名服务可使编程人员访问远程过程或对象更加容易。
- 目录　与命名紧密相连（许多标准将二者合而为一）是目录服务。目录服务可提供一套集中的列表，列举当前网络上激活的所有远程过程或对象。编程人员通过调用目录服务，可得到任何远程过程的位置。
- 生存周期　该服务提供在内存中创建、激活、停止以及删除过程的工具。该服务同时还可以将进程从一台计算机复制或移动到另一台计算机上。
- 持久性　除了处理远程对象的生命周期之外，持久性能使对象在不再需要时存储在磁盘上，而当再次加载回内存时，仍然还可以保持其属性。
- 并发性　有许多程序经常使用同一个远程过程或对象，必须提供能够控制并发访问的服务。关键的代码段为了不破坏局部变量或失去状态信息，可能不允许同时运行。并发服务让程序员在每个远程进程中指定怎样管理并发访问。
- 安全性　远程过程经常与程序或数据库属于同种类型的访问限制。在中间件产品中将安全性进行合并，允许按照统一的标准方式进行有限制的访问。
- 时间　为了确保对象能够正确地进行交互，在所有的机器中可能很有必要维护一张统一的时间引用表。当远程机器位于其他时区时，这可能会比较困难。时间服务提供了一个统一的时间引用表，以及与本地时区的转换。

2.3.2 远程数据库协议

远程数据库协议很可能是所有中间件实现中最为常见的方式。这种方式在客户／服务器数据库刚开始时就已经出现，对大多数客户／服务器开发人员而言也是非常熟悉的。该协议允许网络上一台客户机与服务器数据库进行通信，而不必关注底层的编程细节。这些的中间件软件包括微软公司的 ODBC（开放数据库互联）、ADO（数据访问／存取对象）、数据库网关以及数据库供应商提供的调用接口等等。

远程数据库协议的主要目的是，通过一套简单的对象或函数调用，隐藏网络调用和通信协议的细节。它提供命名和位置服务，使得比较容易定位远程数据库，同时提供配

置服务，以便于在多种编程语言和机器格式之间进行数据转换。大部分远程数据库协议也可以通过 SQL、Java 或其他专有语言执行面向数据库的分布式处理的远程过程调用。

2.3.3 远程过程调用

OSF 的分布式计算环境（DCE，Distributed Computing Environment），是面向分布式计算开发标准的最早的全行业性的组织之一。如 IBM、HP、Digital Equipment 以及其他许多公司都符合和认可这些标准，这样就使得运行在他们机器上的程序可以调用驻留在其他远程机器的软件功能。DCE 规格说明包括一系列标准，如远程过程调用、安全性、目录服务、时间服务、线程以及分布式文件服务等等（参见开放组织文档）。

在 DCE 或其他远程过程调用体系结构中，两台计算机间的一个远程过程调用要求调用程序知道一些标识符，可以用来定位远程计算机上的过程。然后，这些名字或标识符可以保存在目录服务或名字服务中。当远程过程创建时，其就在指定计算机上的某类目录或仓库中进行注册；然后，一旦调用程序需要访问该过程时，就向目录或名字服务发送一个文本字符串以检索该信息。接着，调用程序便可接收到一个识别远程计算机网络位置的指针和远程过程的地址。

一旦该地址已经得到，调用程序便调用位于本地计算机上的函数，将远程过程信息转换成通用的格式，然后将该信息通过网络发送到远程计算机上。远程计算机将这些数据转换成其自身的格式并执行该过程，然后将执行结果又传回本地计算机（调用程序驻留的机器）上，这些结果先转换成本地计算机格式，然后传回调用程序。如图 2-2 所示。

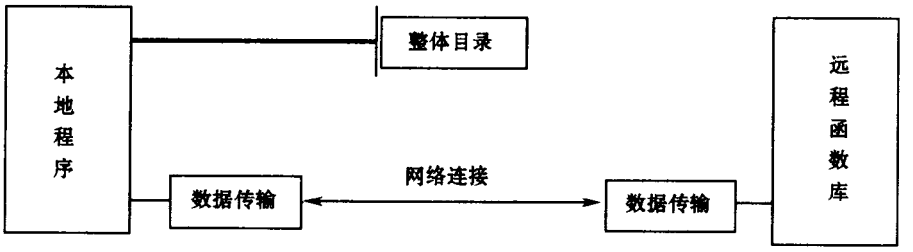

图 2-2 远程过程调用

大多数远程过程体系结构要求过程与状态无关。也就是说，远程过程不能用于在过程调用之间记忆数据。在每个远程过程调用之前，内存变量必须重新进行初始化，以确保这些过程能正确运行。这对于编程人员来说既有利又不利。编程人员可以不必关注多个用户间的数据交互，但与此同时，任何状态变量必须由调用程序进行保持，并由编程人员进行管理。

尽管人们的大部分兴趣集中在分布式对象，远程过程调用仍然占有一席之地。DCE 自从成为工业标准以来，已经超过 10 年的时间。DCE 非常稳定，并且相关的应用范例也很容易找到。DCE 可用于大多数大型机和小型机平台上，因此，将遗留软件集成进应用服务器时，DCE 应该是一个相当不错的选择。

2.3.4 分布式对象

随着编程人员转向面向对象技术，分布式编程行动组织也由原来的分布式函数库转向分布式对象。正如远程过程调用一样，分布式对象也已经开发了相当长的时间，并形成了几个很成熟的标准，以提供计算机平台之间的互操作能力。

与 OSF 类似，对象管理组织（OMG，Object Management Group）面向分布式对象创建了一套名为 CORBA （公共对象请求代理体系结构）的标准和规格说明。CORBA 标准现已得到业界最广泛的支持，在分布式对象标准中是最具扩展性的。该标准提供从 PC 机到大型机的交叉平台的支持，并且 CORBA 兼容的软件几乎可以用任何语言进行编写。

Sun 微系统公司在 Java 平台上对 CORBA 提供了扩展支持，并在最近发布了新的类库，简化了 Java SDK 1.2 版的 CORBA 访问。Java 平台同时也支持更加简单的分布式对象模型，名为 RMI（远程方法调用，Remote Method Invocation）。RMI 仅能在 Java 运行环境中使用，但它却对局域网或互联网上的分布式对象的访问提供了一种简单的方法。

微软公司在采用 CORBA 技术上进展比较缓慢，因为他们已经在自己的分布式对象模型——分布式组件对象模型（DCOM）——中进行了大量投入。DCOM 是基于组件对象模型（COM）,COM 是一种为了满足创建混合文档需求的编程模型，并对 Windows 操作系统提供一致的编程模式。DCOM 主要限于 Windows 操作系统，但同时在其他平台上也有一些有限的第三方支持。

每一种标准都有其优点和缺点，但有关这些的详细讨论已经超出了本书的范围。在本章末列举了能提供详细分析的参考资料索引，这些参考资料中的一种或多种在选择一种分布式对象体系结构时可能会相当有用。

尽管各分布式对象标准在实现任务的方法上可能不一样，但这些标准之间仍然有许多相似之处。如远程过程调用，每个标准均提供命名服务以定位对象，并提供配置特性以在多种编程语言或多台机器之间转换数据表示方法。每个标准同时也提供生命周期和负载平衡管理，但这些任务由于需要在对象内维护属性或状态，因此是很复杂的。同时，由于这些标准是与对象而不是过程调用一起工作，额外的服务需要外部化或将对象的表示方法转换成能在网络上进行传输的字节流。

所有这三种分布式对象标准均使用某种形式的接口定义语言（IDL），以指明分布式对象系统中的对象定义。使用标准的 ASCII 文本文件或现存的程序代码,编译器(如 CORBA 的 IDL、DCOM 的 MIDL 或 RMI 的 RMIC 等）可以将该定义转换为程序模型，提供调用程序和远程对象之间的通信（如图 2-3 所示）。IDL 编译器同时能生成编程语言（如 C++或 RMI 的 Java 等)编译器使用的程序语言头文件(如 C++头文件),以解析本地程序中的对象名。

IDL 编译器亦可以产生存根和框架程序文件，以在两台计算机之间提供通信。存根文件包含一个对象定义，对本地程序来说，它看起来就像在远程计算机上访问对象一样。存根文件包含有与远程对象相同的方法名和属性，但是存根的方法仅仅包含将变量传送到远程对象的代码。在远程计算机上的框架程序的作用是作为这些网络消息的接收器，并用网络发送的参数调用远程对象的方法。

IDL 可以生成静态的对象引用，此时对象定义直接编译到程序中。除了访问静态对象，分布式对象体系结构同时提供对动态对象的访问。在编译时没发现的对象可以在程序运

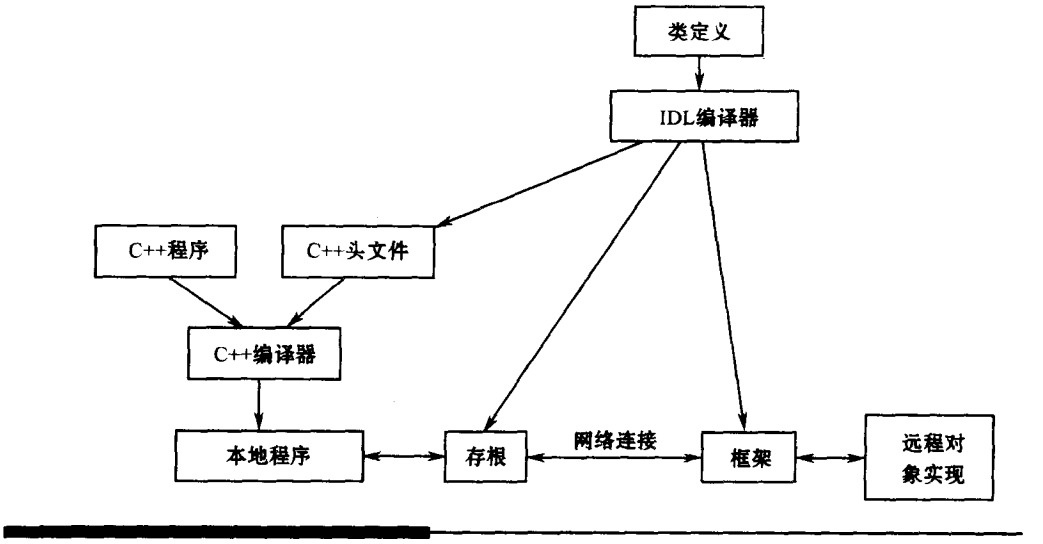

图 2-3　定义分布式对象

行时被找到并实例化。只要选择了对象，它就会被排列，以获取它的方法名和自变量；之后使用通用的协议可以对方法进行访问。动态对象可允许 Visual Basic 或 JavaBeans 等加入分布式体系结构的工具发现新的组件。

另一个从分布式对象发展起来的技术是组件模型。为了使动态对象易于使用，每一个对象都必须有标准化的接口，然后用该接口来发现动态对象的方法和属性。微软的 DCOM 使用 COM 技术——一种复杂的组件模型，它需要每个组件都实现某些标准化的接口和方法 。Sun 创建出需要在每个组件内都有严格命名约定的 JavaBean 标准，然后，JavaBean 框架使用名为“内省”（introspection）的语言特性以检索这些标准化的命名，并在组件框架内部进行解释。由于有严格的标准和额外的接口，组件体系结构使编程变得多少有些复杂。随着技术的不断进步，创建这些组件的工具会变得越来越完善，基于组件的开发会成为构建软件的标准方法。

2.3.5　事务处理监视器

事务处理监视器，像 IBM 的 CICS 和 BEA 的 Tuxedu 产品一样，通过使用事务处理层扩展了远程过程调用体系结构。该层采用了数据库的事务概念，并将之应用到分布式处理中。一系列操作可以绑定在一起作为一个事务；假如在过程中任何地方有一处操作错误，从事务开始已经完成的所有操作都将进行回退。这样，无论发生了什么样的错误，都能够保证数据的一致性和可靠性。

BEA 公司的销售材料举了一个使用自动取款机进行存款的例子。当客户提交了 $ 500 元的存款请求后，自动取款机就会向银行的计算机发送消息。计算机处理完存款后发回消息说明存款已经成功。如果状态消息在银行和 ATM 之间传送时发生了丢失，ATM 就不知道银行是否处理了信息。或许银行已收到了消息，将$500 元进行了登记，但是消息在返回 ATM 机时发生丢失。假如 ATM 试图再发送消息，银行就可能将$500 元的存款登