HZ BOOKS

微软公司核心技术书库

Microsoft Press

Microsoft .NET Remoting 权威指南

(美) Scott McLean
James Naftel 著
Kim Williams

张昆琪 等译

Microsoft .net

机械工业出版社
China Machine Press

微软公司核心技术书库

Microsoft .NET Remoting 权威指南

Scott McLean
(美) James Naftel 著
Kim Williams
张昆琪 等译

机械工业出版社
China Machine Press

本书涵盖了.NET Remoting的各种主要特性，主要讲解了.NET Remoting的体系结构、用.NET Remoting建立分布式应用程序层次、简单对象访问协议和消息流、消息与代理对象、消息接收器与语境、通道与通道接收器以及序列化格式程序等内容。书中还特别讲述了如何使用.NET Remoting的可插入式体系结构来扩展和定制分布式应用程序。

本书概念清晰、条理性强，提供了大量的范例，是一本针对基于Internet的分布式应用开发不可多得的好书。

Scott McLean, James Naftel, and Kim Williams: Microsoft .NET Remoting (ISBN:0-7356-1778-3)

版权登记号：图字：01-2003-2019

图书在版编目（CIP）数据

Microsoft .NET Remoting 权威指南/（美）麦卡林（McLean, S.）等著；张昆琪等译. -北京：机械工业出版社，2003.5

（微软公司核心技术书库）

书名原文：Microsoft .NET Remoting

ISBN 7-111-11909-6

Ⅰ.M… Ⅱ.①麦… ②张… Ⅲ.计算机网络-程序设计 Ⅳ.TP393

中国版本图书馆CIP数据核字（2003）第090424号

机械工业出版社（北京市西城区百万庄大街22号 邮政编码 100037）

责任编辑：杨 梅 武恩玉

北京瑞德印刷有限公司印刷 · 新华书店北京发行所发行

2003年5月第1版第1次印刷

787mm × 1092mm 1/16 · 15.5印张

印数：0 001- 4 000册

定价：30.00元

译 者 序

计算机新技术的层出不穷，推动了分布式计算开发技术的飞速发展。诸如DCOM、Java RMI及CORBA分布式应用技术经过多年的发展，还一直与日益增长的企业需求同步前进。在当今的环境中，分布式应用技术应当具有高效、可扩展、能够支持事务、可与其他不同技术协同工作、可高度配置、在Internet上运行等等特点。

Microsoft .NET具有安全性、可升级性、可靠性、灵活性和互用性等特点，同时具有功能强大的类库和友好的界面，以及对开放标准的强大支持，使其在分布式计算开发方面具有其他工具无法比拟的优势。Microsoft .NET Framework为创建和扩展通过远程对象进行交互的分布式应用程序提供了灵活的模型：.NET Remoting。本书深入讨论了 .NET Remoting的体系结构，并提供了具体的C#编码范例，说明如何扩展和定制 .NET Remoting。作者以其丰富的实践经验剖析了 .NET Remoting的功能，并且提供范例以清晰地说明如何定制 .NET Remoting的关键部分。

本书的内容主要有理解分布式应用程序的开发、理解 .NET Remoting的体系结构，以及如何扩展 .NET Remoting、使用 .NET Remoting建立分布式应用程序、简单对象访问协议（SOAP）和 .NET Remoting消息流、消息及创建自定义代理、创建自定义消息接收器和自定义上下文、创建自定义传输通道和自定义通道接收器、创建自定义序列化格式程序和自定义序列化格式程序接收器。

本书是为那些具有一些 .NET Framwork程序开发经验，并且想要学会如何使用 .NET Framwork远程建立分布式应用程序的读者编写的。本书引导读者分阶段循序渐进完成 .NET Remoting应用程序的开发和实现：分析业务需求、定义技术体系、设计解决方案、编码/实现、测试/调试、实施部署、维护/故障检修，使得读者既可以理解 .NET Remoting的概念，又可以充分掌握 .NET Remoting技术。

本书理论和实例紧密结合，使读者在练习实例的过程中，既学到了理论知识又锻炼了动手能力。作者在多年经验基础上的心血累积保证了本书的权威性、翔实性和实用性。希望读者能够在技术和管理两个方面去把握本书，并把本书里的知识同具体的环境结合起来，从而建立真正有效的 .NET Remoting分布式计算环境。

全书由张昆琪、王尚武、方平、邓盛骋、陈小冲、郭龙永、王治、李鹃君、常欣、李桦、时丁、迟可可、陆思奇、秦鼎印、卫霖等进行翻译，前导工作室全体工作人员共同完成了本书的翻译、录排、校对等工作。本书最后由宋涛统稿。由于时间仓促，且译者的经验和水平有限，译文的不妥之处总是存在，殷切地期望读者能不吝赐教、提出宝贵意见，便于我们提高翻译水平，为大家奉献更新、更好、更专业的书籍！

宋 涛

2002年12月

关于作者

Scott McLean

Scott McLean是在一台Atari 400计算机上开始他的编程生涯的，掌握了Atari BASIC后，他自学了6502汇编程序。几年之后，他应征进入美国海军，在那里他作为快反潜艇上的一名海军“核潜艇”服役了六年。从海军光荣退伍之后，Scott回到学校并获得University of Georgia计算机科学专业的学士学位。

现在作为XcelleNet, Inc.的一名软件工程师，他主要研究企业服务器应用体系及分布式系统开发。他已经使用多线程、套接字、I/O完备端口、COM、ATL及 .NET开发了各种应用；还曾为 .NET Magazine Online写过一篇关于.NET Remoting的文章；同时他还是*Visual C++ .NET*一书的一名合著者，那是WROX出版社为.NET开发人员提供的一本初级读物。Scott也是www.thinkdotnet.com——一个为开发者提供的在线资源的一位共同创始人和投稿人。

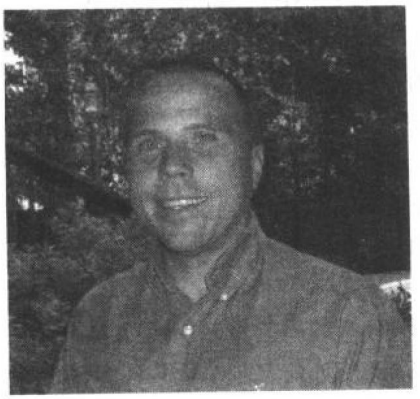
James Naftel

James Naftel是在Allied Collection and Credit Bureau公司开始他的计算生涯的，位于佐治亚州亚特兰大市郊外。开始的时候，他甚至不知道什么是DOS提示符。业主Rex Gallogly向他讲授计算机的相关内容之后，他逐渐有兴趣学习越来越多的计算机知识。当时，他在Georgia State University主修商业。对计算机的热爱这份新发现促使他把专业改为计算机科学。就在转换专业的同时他说服了当时的女朋友April嫁给了他。James和他的新婚妻子搬到了佐治亚州的Athens，加入到University of Georgia（UGA）。在UGA的同时，James也为一家名为PICS的咨询公司工作，在那里他主要负责用Microsoft Visual FoxPro开发库存应用程序。回到亚特兰大地区后，James从Georgia State University毕业并获得了计算机科学的学士学位。

毕业后，James受雇于XcelleNet公司，现在他在那里已经是重要的软件工程师了。他曾经在很多不同类型的企业数据库应用开发及分布式系统的应用领域工作过，而且现在领导一个小组开发数据库同步技术。他与妻子、两个女儿以及两只狗一同居住在佐治亚州的亚特兰大地区。

作为www.thinkdotnet.com的共同创始人和投稿人，James已经在那里发表了许多文章，而且还为*Windows Developer Journal*撰写Microsoft Visual Studio插件的有关文章。他真正的爱好就是使用那些编程语言，尤其是C++和C#。

Kim Williams



Kim Williams是获得了一个音乐学位并从事演奏爵士钢琴后才开始他的计算机职业生涯的。几年后，他返回学校攻读了一个计算机科学学位，把他的计算机编程爱好变成了一种事业。毕业后他得到了梦想中的工作，编写反病毒软件及分解病毒。从事反病毒工作的同时他还开发了分布式企业安全应用。

加入XcelleNet公司后，Kim作为首要的软件工程师，使用过各种技术，例如Java RMI、DCOM、ATL和ASP。目前，他正领导小组开发大规模ASP .NET Web Services解决方案。Kim也是www.thinkdotnet.com的共同创始人和投稿人。目前他与妻子Patty和儿子Sean居住在Georgia的Atlanta，并且总是设法找时间弹弹钢琴。

前　言

分布式计算几乎已经成为所有软件开发中不可缺少的组成部分。.NET Remoting出现之前，DCOM是开发Microsoft平台上的分布式应用程序的首选方法。但是，对一般的开发人员来说，理解和使用DCOM都不那么容易。那就进入 .NET Remoting吧，它是一个有助于使用Microsoft .NET进行分布式应用开发的面向对象体系。正如 .NET Framework取代COM成为建立组件的首选方式一样，.NET Remoting取代了DCOM成为使用 .NET Framework建立分布式应用程序的首选方式。此外，.NET Remoting还提供 .NET Web服务的基础支持内容。因此，从根本上理解 .NET Remoting对开发人员转向使用 .NET Framework开发更加基于Internet的分布式应用是至关重要的。

本书深入讨论了 .NET Remoting的体系结构，并提供了具体的C#编码范例，说明如何扩展和定制 .NET Remoting。我们将剖析 .NET Remoting的功能，用以说明如何定制 .NET Remoting的关键部分。这些也正是 .NET Remoting真正出类拔萃的地方。此外，.NET Remoting体系还提供了许多可扩展性“钩子”(hook)，以便读者使用各种协议和配置选项。

一旦开始使用 .NET Framwork，你就会惊喜地发现使用 .NET Remoting建立分布式应用是多么的容易，与使用DCOM进行开发形成了很大的反差！而且，在扩展 .NET Remoting基础设施的时候，很快就会意识到 .NET Remoting的真正威力。一般而言，可以发现 .NET Remoting具有一个富含逻辑性且非常连贯的对象模型，这使得 .NET Remoting基础设施的配置简单修改和高级扩展更加容易。此外，.NET Remoting支持开放和基于Internet的标准，例如，Web服务和简单对象访问协议（SOAP）。当然，它不是完美的，通常任何新技术都有瑕疵。然而，我们几乎总能为自己遇到的问题找到合理的工作区（书中会指出这些工作区）。我们已经看到新技术的共享，而且我们相信，.NET Remoting对它的前身（DCOM）是个强有力的替代者，同时也是现今开放式Internet互联环境中支持分布式应用开发的强大工具。

预期读者

本书是为那些具有一些 .NET Framwork程序开发经验，并且想要学会如何使用 .NET Framework建立分布式应用程序的人编写的。书中详细介绍了 .NET Remoting，这个主题不需要任何预备知识。所有的范例均用C#实现，因而最好具有C#的工作技能。不过，我们并没有用到C#的众多高级特性。尽管应当熟练掌握 .NET Framwork 和C#，但对于具有C++、Microsoft Visual Basic .NET或Java背景的人来说，本书仍然非常易于理解。如果曾经使用这些语言中的任意一种编写过远程应用程序，就应当具备了掌握本书的足够知识。

结构安排

本书分为如下8章。前两章是概念，其他章节集中于高级概念，并说明了如何最大限度地利

用 .NET Remoting所提供的可扩展性。

- 第1章　理解分布式应用开发　该章以讨论分布式体系结构和技术的历史作为开篇。讨论了远程过程调用（RPC）、DCOM、远程方法调用（RMI）及SOAP/XML技术。该章的目的是要说明这些过去技术的成功之处以及缺点。然后深入分析 .NET Remoting如何既满足过去又满足现在的分布式应用开发的需要。
- 第2章　理解 .NET Remoting体系结构　该章介绍了 .NET Remoting基础设施的主要结构部件。在后续章节中会深入研究这些内容。该章既可作为引言也可作为这些 .NET Remoting概念的参考资料。它介绍了构成 .NET Remoting体系结构的每个主要组成部分：激活（服务器端激活与客户端激活）、按引用列集、按值列集、租用、通道、消息以及格式程序等。
- 第3章　使用 .NET Remoting建立分布式应用程序　该章就如何使用 .NET Remoting提供的各种常用特性构建分布式应用程序做了详细的介绍。这里我们创建了一个假设的作业分配应用程序，用它来说明 .NET Remoting的基本概念，例如，客户端激活对象和服务器端激活对象。此外，这个应用程序还说明了如何使用 .NET Remoting实现Web服务。该章也展示了如何使用Microsoft Internet 信息服务（IIS）的强大安全特性给 .NET Remoting应用增加安全性，并说明了如何将一个远程对象发布为一个Web服务。
- 第4章　SOAP与消息流　该章是关于SOAP的入门读物，同时讨论了第3章中开发的客户端应用与服务器端应用之间交换的消息。此外还说明了由 .NET Remoting产生并使用的外围产品，让读者学习一点其他的内容。
- 第5章　消息与代理　该章一开始讨论了消息，这是扩展和定制 .NET Remoting基础设施的基础。该章也讨论了作为本地对象与远程对象之间桥梁的代理，客户端代码调用代理对象，接着由代理对象激活远程对象的方法。我们展示了开发自定义代理的三种方法，并解释了如何将它们插入 .NET Remoting的基础设施。我们使用自定义代理对象开发了两个范例应用程序：一个是在不能通过TCP进行连接（例如，由于防火墙的存在）的时候，动态地从使用TCP转换为使用HTTP；另一个提供负载平衡。
- 第6章　消息接收器与上下文　该章说明了如何使用 .NET Remoting环境对上下文中执行的对象实施规则和行为。该章还对什么是消息接收器链表以及它们为何是 .NET Remoting Framework中的一个重要可扩展点做了解释和说明，它们提供了强大的上下文侦听功能所依赖的基础。我们也对每一个不同的上下文相关消息接收器做了说明，并介绍了如何使用这些接收器。
- 第7章　通道与通道接收器　通道是 .NET Remoting的基础组成部分。该章首先说明了 .NET Remoting的HttpChannel的体系结构及其支持类，这样读者就能够更好地理解如何创建自定义通道。然后通过将文件系统用作 .NET Remoting的消息传输机制的自定义通道类型范例来讨论对 .NET Remoting的扩展。最后，创建了一个在用户定义的时间段内阻塞方法调用的自定义接收器。
- 第8章　序列化格式程序　最后一章仍然建立在前面章节所讨论的概念上，并详细描述了序列化格式程序。在介绍了一般的序列化概念后，我们向读者说明了如何通过创建自定义的

序列化格式程序和格式程序接收器来扩展 .NET Remoting。

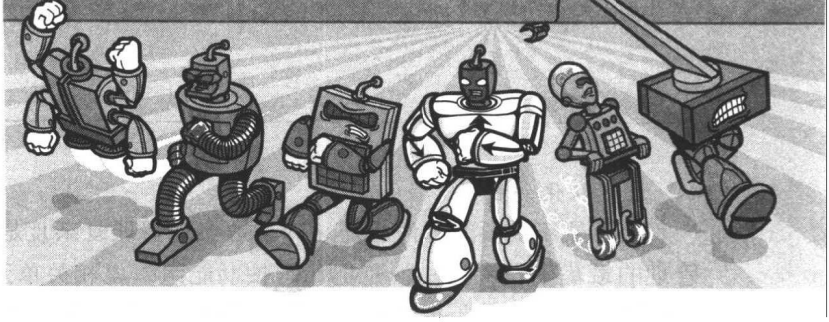
系统要求

要建立并执行书中的范例代码，需要Microsoft Visual Studio .NET。还需要IIS来运行Web服务和演示第3章讨论的安全技术。尽管使用拥有两台或更多计算机的网络才能最好地说明 .NET Remoting的许多特性，但本书中的所有范例代码都可以在单机上运行。

范例文件

本书的范例文件可以在Web上获得：http://www.microsoft.com/mspress/books/6172.asp。来到这个网站后要获取本书的参考内容，请单击网页右边More Information菜单里的Companion Content链接。这样就会装载参考内容页面，其中包含了下载范例文件的链接。

目　录



第1章 理解分布式应用开发

DCOM、Java RMI及CORBA等分布式应用技术已在多年的发展中一直与日益增长的企业需求同步前进。在当今的环境中，分布式应用技术应当具有高效的、可扩展的、能够支持事务、可与其他不同技术协同工作、可高度配置、在Internet上运行等特点。但是，并不是所有的应用程序在规模上都大到需要所有这些支持。在支持较小的系统时，分布式应用技术往往提供常见的默认行为，还要易于配置，这样才能让这些系统的分布尽可能的容易。单个远程技术看上去似乎不可能满足这些所有的需求。事实上，当今大多数分布式应用技术都是从较为适中的一类需求开始的，然后，经过多年发展取得了对其他需求的支持。

洗心革面常常是个好方法，这也是.NET Remoting的设计所采取的方法。.NET Remoting提供了具有可扩展性钩子的连贯对象模型以支持迄今为止开发人员使用DCOM建立的各种系统。.NET Remoting的设计者采用了那些最初不为DCOM设计者所知的最新技术。

尽管本章的意图并不在于帮助你决定使用哪种分布式应用技术，但是，如果你正在使用.NET进行所有的新开发，而且使用.NET Framework实现客户端和服务器，那么.NET Remoting可能是最佳选择。另一方面，如果现有的分布式应用是用非.NET Remoting技术实现的，.NET Framework则提供了与传统技术之间空前的互操作性：

- 如果现有系统是COM/DCOM，.NET Framework的“.NET-to-COM”互操作层功能完备且易于使用。该层允许随时间灵活地向.NET Remoting进行增量移植。
- 如果现有系统是基于诸如Java远程方法调用（RMI）或公用对象请求代理体系结构（CORBA）的非Microsoft分布式技术，这里仍有好方法。因为其他商家都采用了诸如XML和简单对象访问协议（SOAP）这样的开放标准，所以.NET Remoting对这些开放标准的支持可以提供跨商家、跨平台环境的通信。目前，基于Java的SOAP工具比比皆是。尽管CORBA的SOAP支持有些滞后，但是对象管理组织（OMG）目前正致力于形成官方CORBA/SOAP标准。

1.1 简短历史

从最广泛意义上来讲，分布式应用程序是一种将应用程序的处理分成几部分、联合两台或更多机器执行的程序。处理的划分意味着涉及到的数据也是分布式的。

1.1.1 分布式体系结构

.NET Remoting之前出现了大量的分布式应用解决方案。从这些早期系统技术中，.NET Remoting获得了许多有关分布式计算的经验，并形成了最新形式。

1. 模块化编程

除了最不重要的软件程序之外，适当地管理复杂度是所有开发的重要一步。对复杂度进行管理的最基本技术之一就是把代码按功能组织成相关单元。可以在多种级别上应用这种技术：把代码组织成过程、把过程组成类、类组成组件以及组件组成更大的相关子系统。分布式应用程序从这个概念中获益匪浅，而且在很多情况下都有助于实施这个概念，因为需要模块化将代码分布到各台机器。实际上，各种分布式体系结构的主要区别在于：赋予不同模块的职责以及提供模块间的不同交互。

2. 客户端/服务器

客户端/服务器是最早也是最基础的分布式体系结构。广义上来说，客户端/服务器只是向服务器进程请求服务的客户端进程。一般客户端进程负责表示层（或用户接口），该层包括验证用户输入、向服务器发送调用，还有可能执行一些业务规则。然后服务器担当起引擎——通过执行业务逻辑并与数据库和文件系统这样的资源协同合作完成客户端请求。通常有许多客户与一个服务器进行通信。尽管本书是关于分布式应用程序开发的书籍，但我们也应当指出，客户端和服务器的职责一般不必划分到多台机器。应用程序的功能划分对运行在单一机器上的多进程来说是个好的设计方式。

3. N层 (N-Tier)

客户端/服务器应用程序也称为两层应用程序，因为客户端直接与服务器对话。两层结构通常实现起来相当容易，但只拥有有限的可扩展性。过去，开发者经常是由于如下原因发现需要n层设计的：一个应用程序运行在单个机器上。由于某些原因有人决定需要分布应用程序，这些理由可能包括想要向多个客户提供服务、开启资源访问或者利用一台功能强大的机器的高级处理能力。第一个尝试通常建立在两层设计上——原型运转良好，就认为一切都不错。随着更多的客户加入进来，事情开始逐渐慢下来。添加了太多的客户使系统屈服了。接着，试图升级服务器的硬件来解决这个问题，但这是个昂贵的选择，而且仅仅只能拖延实际问题的到来。

这个问题的一个可能的解决方案是改变体系结构使用三层或n层设计。图1-1显示的是三层体系结构如何致力于给系统添加中间层以执行各种任务的。有种选择就是将业务逻辑放在中间层上。这样，中间层检查客户端提供的数据的一致性，并根据业务需要处理这些数据。这项工作涉及到与数据层合作或者执行内存中的计算。如果一切顺利，中间层通常就会把它的结果提交给数据层进行存储或者返回给客户。这种设计的关键处在于处理责任的粒度分布。

即使是n层系统有多层位于同一机器，系统功能的逻辑划分还是非常有益的。开发人员或管理人员可以分别维护各层、将它们一同换出或者迁移到分离的机器上为以后升级提供需要。这就是为何三层（或实际上的n层）体系结构对软件维护和部署的可扩展性与灵活性来说都是最佳的。

4. 对等网络 (Peer-to-Peer)

前述的分布式体系结构其每层都有明确的作用。客户端/服务器层很容易就可以标为主/从或生产者/消费者。n层模型里的层往往分为诸如表示层、业务层或数据层这样的角色，然而并不总是需要这样。一些设计从更加合作的模型中获益不少，这种模型中的客户与服务器之间的界线没有那么清晰。由于这些分布式应用程序的主要功能是共享信息和处理，所以工作组方案就这样构建出来了。

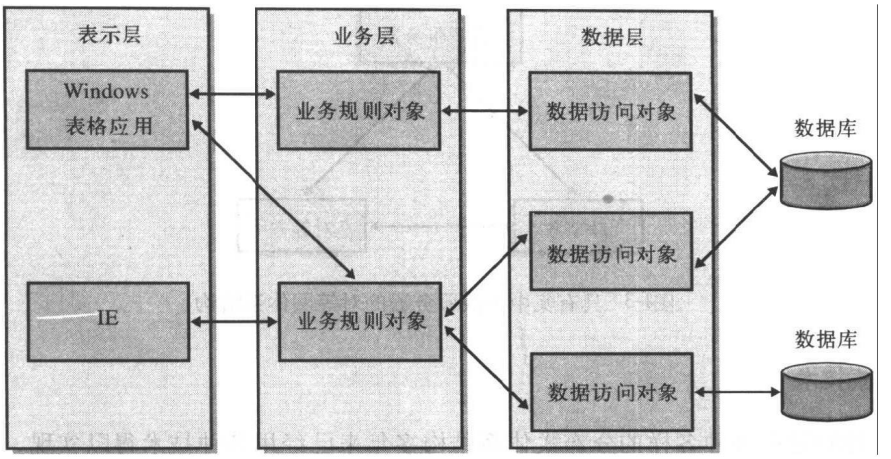


图1-1 三层体系结构

一个纯粹的端对端设计是由许多单个节点构成的，没有集中的服务器，如图1-2所示。如果没有一个众所周知（well-known）的主服务器，就必须有一个机制让个体之间可以相互找到对方。通常这是通过广播技术或一些预定义的配置参数实现的。

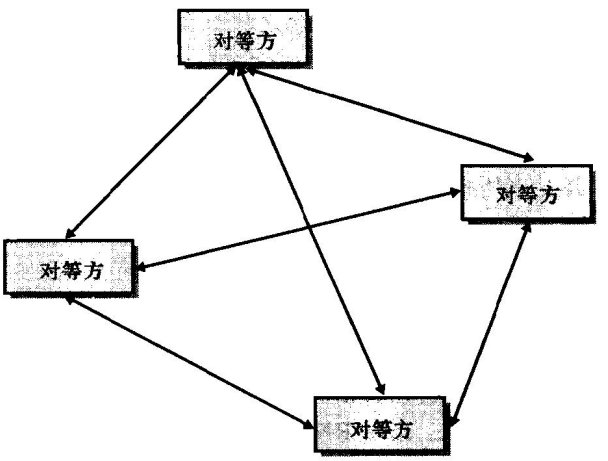


图1-2 对等网体系结构

Internet通常被认为是最具代表性的客户端/服务器体系结构，具有一个统一的Web服务器为许多瘦客户端提供服务。但是Internet也导致产生了一些准端对端应用程序，例如Napster和Gnutella。这些系统允许对等机器之间合作共享数据。这些对等个体使用一个集中的服务器进行对等发现和查找，如图1-3所示。尽管它们不是纯粹的对等体系结构，这些混合模型通常比完全分散的对等模型更好扩展，而且表现出同样的易协作的优点。

即使是多层设计也可以从放宽客户端－服务器角色区分中获益。客户端模块也作为服务器以及服务器作为客户端也是很常见的。在第3章“用 .NET Remoting建立分布式应用”中介绍客户端回调与事件时，我们会讨论客户端－服务器的这种角色的模糊性。

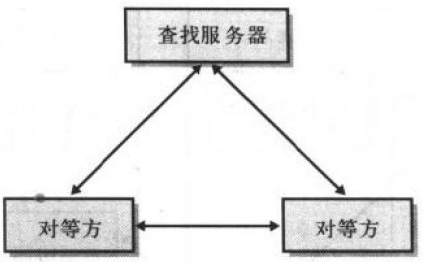


图1-3 具有集中查找服务器的对等网体系结构

1.1.2 分布式技术

我们讨论的这些各种各样的分布式体系结构多年来已经用各种技术得以实现。这些结构经实践证明是可取的，分布式应用开发已经在技术方面有了大大地提高。与10年前用以开发分布式应用程序的工具和抽象概念相比较，当今的开发者很容易就能办到！现在，与构建一个仅仅用于将数据从一台机器移到另一台机器的基础设施所花费的时间相比，我们可以用更多大量的时间来解决业务问题。来看看我们已经达到的程度。

1. 套接字

套接字是现代网络应用程序的基础抽象概念之一。套接字使通信看上去像基于流的I/O，从而为编程人员屏蔽了网络的低级细节。尽管套接字可以完全控制通信，但是建立复杂的、功能完备的分布式应用程序它们还需要做太多工作。使用基于流的I/O进行数据通信意味着开发人员必须构建消息传递系统，而且要建立并解释数据流。对大多数通用的分布式应用程序来说，这种工作太单调乏味了。开发者所需要的是一个高级抽象——让你觉得好像是执行本地函数或过程的调用。

2. 远程过程调用

开放组织（正式名称为开放软件基金会）的分布式计算环境（DCE），为进行远程过程调用(RPC）定义了一个规范。采用RPC、适当的配置和数据类型的约束，开发者可以使用执行本地过程调用所需的许多相同语义实现远程通信。RPC引入了几个基本概念，这是所有现代分布式技术的基础，包括DCOM、CORBA、Java RMI以及现在的 .NET Remoting。下面是其中一些基础概念：

- 根（stub）——这段代码运行在客户端和服务器端，让远程过程调用看上去好像是本地的。举个例子，客户端代码调用根代码中看上去的确像服务器上执行的那些过程。然后根代码将调用转发给远程过程。
- 列集——这是从一个上下文向另一个上下文传递参数的过程。RPC中函数的参数被序列化为数据包在线缆中传输。
- 接口定义语言（IDL）——这种语言提供了描述调用句法的标准方法以及独立于任何具体编程语言的可远程调用过程的数据类型。某种Java RMI并不需要IDL，因为这种分布式应用技术仅支持一种语言：Java。

RPC使远程通信比套接字编程更加友好，代表了一个巨大的进步。但是随着时间的推进，业

界从过程性的编程转移为面向对象的开发。必然地，分布式对象技术离我们不会多远了。

1.2 分布式对象——一个受欢迎的抽象概念

现在大多数开发人员都将面向对象的设计与编程作为现代软件开发的原则。无论何时，当人类不得不处理创建大型软件系统这样的复杂问题时，使用有效的抽象概念非常重要。对象是现在使用的基础抽象概念。因此，如果开发者承认对象所带来的好处，将它们应用于分布式方案就会有意义了。

分布式对象技术允许运行在另一机器上的应用程序或对象访问某机器上运行的对象。正如RPC让远程过程好像本地过程一样，分布式对象技术让远程对象看上去像本地对象。DCOM、CORBA、Java RMI以及 .NET Remoting都是分布式对象技术的一个例子。尽管这些技术实现起来大不相同，而且建立在不同的业务哲学上，但它们在许多方面都非常的类似：

- 它们都建立在具有个性、具有状态或可以具有状态的对象基础上。开发者可以将具有实质上相同语义的远程对象作为本地对象。这样提供了单一的一元化编程模型，从而简化了分布式编程。可能的话，开发者可以分解出特定于分布式编程的公共语言部分，并放入配置层。
- 它们都与组件模型相关联。术语组件（component）可以用很多种方式定义，但是，在这里我们认为组件是可二进制部署的单独功能性单元。组件代表着面向对象的实践从白盒子重用发展到黑盒子重用。由于它们严格的公约，组件通常几乎没有相关性，而且可以经过重新编译和再定位成为功能单元。使用组件增加了部署的灵活性以及公用服务的提取效果。
- 它们都与企业服务相关。企业服务通常为事务、对象共享池、并发管理及对象定位等任务提供支持。这些服务对大容量系统提出了公共的要求而且很难实现。当分布式系统上的客户负载达到某一程度，这些服务就变得对可扩展性和数据完整性非常重要。由于这些服务很难实现，而且通常都非常需要，所以一般都将它们从开发者的编程领域中提取出来，并由分布式对象技术、一个应用服务器或操作系统提供。

1.3 分布式应用开发的优点

因为你已经选择了阅读一本关于 .NET Remoting的书，就可能已经考虑到了将应用程序分散开来的某些方案。为了完整起见，我们要提到选择分布应用程序的几个一般原因。

1.3.1 容错

分布式应用程序的好处之一——看起来似乎也是使用起来的一个困难——是容错观念。尽管这个概念很简单，但大部分的研究都集中在容错算法和体系结构上。容错是指一个系统内部出现故障时系统应当能够恢复。建立容错系统的一块基石就是冗余。例如，一辆汽车有两个前灯。如果一个前灯烧坏了，很可能就继续使用另一个前灯一段时间，让司机到达目的地。我们还可以期待司机在剩下的车灯烧坏之前更换那个故障前灯。

由于其本性，分布式应用开发利用机会将冗余概念应用于分布式对象从而建立容错软件系