[19] 中华人民共和国国家知识产权局

[51] Int. Cl[7]
G06F 15/16

[12] 发明专利说明书

[21] ZL 专利号 99814675.7

[45] 授权公告日 2003 年 9 月 24 日

[11] 授权公告号 CN 1122229C

[22] 申请日 1999.10.21 [21] 申请号 99814675.7
[30] 优先权
[32] 1998.11. 5 [33] US [31] 60/107,167
[32] 1999. 9.23 [33] US [31] 09/405,318
[86] 国际申请 PCT/US99/24561 1999.10.21
[87] 国际公布 WO00/28431 英 2000.5.18
[85] 进入国家阶段日期 2001.6.18
[71] 专利权人 BEA 系统公司
地址 美国加利福尼亚州
[72] 发明人 迪安·B·雅各布斯
安诺·R·兰根
审查员 田 竞

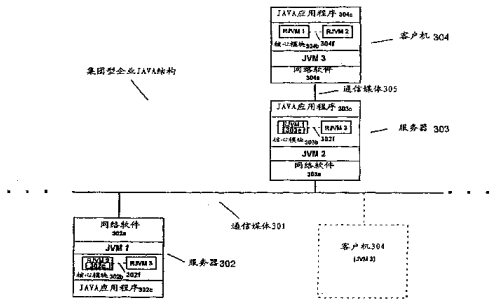
[74] 专利代理机构 北京市柳沈律师事务所
代理人 黄小临

权利要求书 3 页 说明书 23 页 附图 15 页

[54] 发明名称 包含消息传送核心的分布式处理系统

[57] 摘要

本发明提供了一种集团型企业 Java 分布式处理系统，该分布式处理系统包括与通信媒体连接的第一和第二计算机。 第一计算机包括 Java 虚拟机(JVM)和含有远程 Java 虚拟机(RJVM)的、用于传送消息的核心软件层。 第二计算机包括 JVM 和含有 RJVM 的核心软件层。 消息从 RJVM 传送到一个计算机中的 JVM，再传送到第二个计算机的 JVM 和 RJVM。 消息可以通过中间服务器转发或在网络重新配置之后重新择径。 每个计算机包括含有复制处理器的智能存根模块，包括负载平衡软件部分和故障处理软件部分。 每个计算机包括在节点上存储智能存根模块库的复制服务命名树。 该计算机可以以无状态、无状态工厂、或有状态编程模型编程。 集团型企业 Java 分布式处理系统允许提高规模可伸缩性和容错性。



ISSN1008-4274

1. 一种分布式处理系统，包括：

一个通信媒体；

一个与该通信媒体连接的第一处理设备，具有模拟第一处理设备（“JVM1”）的第一软件程序，所述第一处理设备还包括一第一核心软件层，所述第一核心软件层具有一第一数据结构（“RJVM2”），所述第一处理设备还包括一存根模块，所述存根模块包括复制处理器，所述复制处理器含有负载平衡软件部分和用于选择多个被复制的对象之一的故障处理软件部分；

与通信该媒体连接的第二处理设备，具有模拟第二处理设备（“JVM2”）的第二软件程序，所述第二处理设备还包括一第二核心软件层，所述第二核心软件层具有一第二数据结构（“RJVM1”），其中来自第一处理设备的消息通过从第一处理设备中的第一核心软件层和第一软件程序到第二处理设备中的第二软件程序和第二核心软件层传送到第二处理设备；和，

其中第一处理设备中的第一软件程序是 Java™ 虚拟机（“JVM”），第一处理设备中的第一数据结构是远程 Java™ 虚拟机（“RJVM”）和其中，

第二处理设备中的第二软件程序是 Java™ 虚拟机（“JVM”），第二处理设备中的第二数据结构是远程 Java™ 虚拟机（“RJVM”），和其中第二处理设备中的 RJVM 对应于第一处理设备中的 JVM。

2. 如权利要求 1 所述的分布式处理系统，其中第一处理设备中的第一核心软件层包括接头管理器软件部分。

3. 如权利要求 1 所述的分布式处理系统，其中第一处理设备中的第一核心软件层包括线程管理器软件部分。

4. 如权利要求 1 所述的分布式处理系统，其中第一处理设备中的 RJVM 包括消息择径软件部分。

5. 如权利要求 1 所述的分布式处理系统，其中第一处理设备中的 RJVM 包括消息压缩软件部分。

6. 如权利要求 1 所述的分布式处理系统，其中第一处理设备中的 RJVM 包括对等消失检测软件部分。

7. 如权利要求 1 所述的分布式处理系统，其中第一处理设备利用从由 TCP、SSL、HTTP 隧道技术和 IIOP 隧道技术构成的协议组中选择的协议与

第二处理设备通信。

8. 如权利要求 1 所述的分布式处理系统，其中第一处理设备包括用于存储 Java™ 应用程序的存储器。

9. 如权利要求 1 所述的分布式处理系统，其中第一处理设备是客户机和第二处理设备是响应客户机请求向客户机提供服务的服务器。

10. 如权利要求 1 所述的分布式处理系统，其中第一处理设备是第二处理设备的对等物。

11. 如权利要求 10 所述的分布式处理系统，还包括:

与第二处理设备连接的第二通信媒体；和

与第二通信媒体连接的第三处理设备，具有 1）模拟第三处理设备（“JVM3”）的第三软件程序，和 2）具有第三数据结构（“RJVM1”）和第四数据结构（“RJVM2”）的第三核心软件层。

12. 如权利要求 11 所述的分布式处理系统，其中第二处理设备将消息从第一处理设备转发到第三处理设备。

13. 如权利要求 12 所述的分布式处理系统，其中第一处理设备中的第一核心软件层包括用于缩减消息的缩减表，和第三处理设备包括用于读取消息的复制缩减表。

14. 如权利要求 12 所述的分布式处理系统，其中第三处理设备与第一通信媒体连接，和其中消息通过从第一处理设备中的第一核心软件层到第三处理设备中的第三核心软件层传送到第三处理设备。

15. 如权利要求 1 所述的分布式处理系统，其中第一处理设备包括 Enterprise Java™ Bean 对象。

16. 如权利要求 1 所述的分布式处理系统，其中第一处理设备包括让 RA 存根模块库存储在树节点上的命名树，和第二处理设备包括命名树的复制件。

17. 如权利要求 1 所述的分布式处理系统，其中第一处理设备包括用于 1）将 RA 存根模块分配到第一和第二处理设备和 2）将命名树分配到第一和第二处理设备的多点发送程序。

18. 如权利要求 1 所述的分布式处理系统，其中第一处理设备包括以无状态程序模型编码的应用程序。

19. 如权利要求 18 所述的分布式处理系统，其中应用程序包括无状态会话豆模块。

20. 如权利要求 1 所述的分布式处理系统，其中应用程序包括以无状态工厂程序模型编码的应用程序。

21. 如权利要求 20 所述的分布式处理系统，其中应用程序包括有状态会话豆模块。

22. 如权利要求 1 所述的分布式处理系统，其中第一处理设备包括以有状态程序模型编码的应用程序。

23. 如权利要求 22 所述的分布式处理系统，其中应用程序包括实体会话豆模块。

包含消息传送核心的分布式处理系统

发明领域

本发明涉及分布式处理系统，具体地涉及分布式处理系统中的计算机软件。

相互参照相关申请

本申请要求 1998 年 11 月 5 日提出的美国临时申请 No.60/107，167 的权益。

下列同时待审的美国专利申请已转让给本申请的受让人，将它们的公开文本插在这里以供参考：

（A）申请号：尚不知（代理人备案号（Attorney Docket）No.BEAS1029），由 Dean B.Jacobs 和 Eric M.Halpern 提出申请（尚不知），名称为：分布式处理系统中的智能存根模块或企业 JAVA™ 豆模块（“A SMART STUB OR ENTERPRISE JAVA™ BEAN IN A DISTRIBUTED PROCESSING SYSTEM”）；

（B）申请号：尚不知（代理人备案号：BEAS1030），由 Dean B.Jacobs 和 Eric M.Halpern 提出申请（申请日尚不知），名称为：分布式处理系统中的复制命名服务（“A DUPLICATED NAMING SERVICE IN A DISTRIBUTED PROCESSING SYSTEM”）；和

（C）申请号：尚不知（代理人备案号：BEAS1031），由 Dean B.Jacobs 和 Anno R.Langen 提出申请（申请日尚不知），原名称为：在安全分布式处理系统中的集团型企业 JAVA™（“CLUSTERED ENTERPRISE JAVA™ IN A SECURE DISTRIBUTED PROCESSING SYSTEM”）。

发明背景

存在若干种类型的分布式处理系统。一般地说，分布式处理系统包括多个处理设备，例如，两个与通信媒体连接的计算机。通信媒体可以包括有线媒体、无线媒体、或它们的组合，例如，以太（Ethernet）局域网或蜂窝网。在分布式处理系统中，至少有一个处理设备可以将通信媒体上的信息传送到

另一个处理设备上。

图 1 所示的客户机／服务器结构 110 是一种类型的分布式处理系统。客户机／服务器结构 110 包括至少两个处理设备，表示为客户机 105 和应用程序服务器 103。附加的客户机也可以与通信媒体 104 连接，例如客户机 108。

通常，服务器 103 是商用逻辑模块（business logic）的宿主和／或协调事务处理以向另外的处理设备，例如客户机 105 和／或客户机 108 提供服务。应用程序服务器 103 通常用提供服务的软件编程。软件可以利用各种各样的编程模型，例如图 1a-b 所示的 Enterprise Java™ Bean（“EJB”）100b 编程。服务可以包括，例如，检索和传送来自数据库的数据，提供图像和／或解方程。例如，服务器 103 可以响应来自客户机 105 的请求，通过通信媒体 102 从永久性存储器件 101 中的数据库 101a 检索数据。然后，应用程序服务器 103 可以通过通信媒体 104 传送请求数据到客户机 105。

客户机是利用来自服务器的服务和可以请求服务的处理设备。用户 106 经常与客户机 105 交互操作，并可以使客户机 105 通过通信媒体 104 从应用程序服务器 103 请求服务。客户机经常处理与最终用户的直接交互操作，例如接受请求和显示结果。

各种各样不同类型的软件可以用于编程应用程序服务器 103 和／或客户机 105。一种编程语言是 Java™ 编程语言。Java™ 应用程序目标代码被装载到 Java™ 虚拟机（“JVM”）中。JVM 是装载到模拟特定机器或处理设备的处理设备中的程序。关于 Java™ 编程语言的更多信息可以从 http://www.javasoft.com 站点上获得，将这个站点列在这里以供参考。

图 1b 表示若干种 Java™ 企业应用编程接口（“API”）100，这些接口允许 Java™ 应用代码仍然独立于基本事务处理系统、数据库、和网络基础设施。Java™ 企业 API 100 包括：例如，远程方法调用（“RMI”）100a、EJB 100b、和 Java™ 命名和目录接口（JNDI）100c。

RMI 100a 是经常用在如下所述的对等结构（peer-peer architecture）中的分布式编程模型。具体地说，一组类别和接口使一个 Java™ 对象能够调用在不同 JVM 上运行的另一个 Java™ 对象的公开方法。

EJB 100b 的示例通常用在如上所述的客户机／服务器结构中。EJB 100b 的示例是软件成分或可以与其它成分组合的可重用预建的一段封装的应用程序代码。通常，EJB 100b 的示例包括商用逻辑单元。存储在服务器 103 中的

EJB 100b 示例通常管理永久性、事务处理、并发性、线程处理和安全性。

JNDI 100c 向 Java™ 软件应用程序提供目录和命令功能。

客户机/服务器结构 110 存在着许多缺点。首先，因为服务器 103 不得不处理许多连接，所以结构 110 的规模不能太大。换言之，可以附加到服务器 103 上的客户机的数量受到限制。另外，加倍地添加处理设备（客户机）未必向你提供加倍的性能。其次，难以维护客户机 105 和 108 上的应用程序代码。第三，结构 110 易遭受一些系统故障或单点故障。如果服务器 103 发生故障并无法备份，那么客户机 105 将不能获得服务。

图 1c 表示多层结构 160。客户机 151 和 152 管理与最终用户的直接交互操作，接受请求和显示结果。应用程序服务器 153 寄存应用程序代码、协调通信、同步和事务处理。数据库服务器 154 和永久性存储器 101 提供数据的持续事务型管理。

多层结构 160 具有与客户机/服务器结构 110 相似的上述缺点。

图 2a 表示对等结构 214。处理设备 216、217 和 218 与通信媒体 213 连接。处理设备 216、217 和 218 包括操作系统 211a，211b，211c 以及通过媒体 213 通信用的网络软件 210a、210b、和 210c。通常，在对等结构中的每个处理设备具有相似的处理能力和应用程序。对等程序模型的例子包括 Common Object Request Broker Architecture（公用对象请求中介架构）（"CORBA"，）和 Distributed Object Component Model（分布式对象组件模型）（"DCOM"）结构。

在一个平台特定的分布式处理系统中，各个处理设备可以运行相同的操作系统。这样就允许诸如共享盘、多尾盘（multi-tailed disk）和高速互连之类的专有硬件用于处理设备之间的通信。该特定的分布式处理系统平台的例子包括 IBM® Corporation's S/390 Parallel Sysplex®、Compaq's Tandem Division Himalaya 服务器、Compaq's Digital Equipment Corporation™（DEC™）Division OpenVMS™ Cluster 软件、和 Microsoft Corporation Windows NT® Cluster Services（Wolfpack）。

图 2b 示出了事务处理（TP）结构 220。具体地说，TP 结构 220 示出了 BEA® Systems, Inc.TUXEDO® 结构。TP 监视器 224 分别通过通信媒体 280、281 和 282 与处理设备 ATM 221、PC 222 和 TP 监视器 223 连接。ATM 221 可以是自动出纳机，PC 222 可以是个人计算机，和 TP 监视器 223 可以是另一个事务处理监视器。TP 监视器 224 通过通信媒体 283、284 和 285 与后台

服务器 225、226 和 227 连接。服务器 225 通过通信媒体 286 与永久性存储装置 287、存储数据库 289 连接。TP 监视器 224 包括工作流控制器 224a，工作流控制器 224a 用于将来自诸如 ATM 221、PC 222、或 TP 监视器 223 的处理设备的服务请求择径(routing)到诸如服务器 225、226 和 227 的各种服务器。工作流控制器 224a 允许做到（1）服务器之间的工作负载平衡，（2）受限制的规模可伸缩性或允许附加的服务器和／或客户机，（3）冗余后台服务器的容错性（或服务请求可以由工作流控制器发送到没有故障的服务器），和（4）会话集中以限制同时连接到后台服务器的数量。其它事务处理结构的例子包括 IBM® Corporation's CICS®、Compaq's Tandem Division Pathway/Ford/TS、Compaq's DEC™ ACMS、和 Transarc Corporation's Encina。

TP 结构 220 也存在着许多缺点。首先，单个处理设备或 TP 监视器 224 的故障可以使网络变成不能工作。其次，规模可伸缩性或与 TP 监视器 224 连接的处理设备（服务器和客户机两者）的数量可能受 TP 监视器 224 的硬件或软件的限制。第三，将客户机请求择径到服务器的灵活性受到限制。例如，如果通信媒体 280 不能工作，但通信媒体 290 可以使用，那么，ATM 221 通常不可以通过通信媒体 290 直接从服务器 225 请求服务，而必须访问 TP 监视器 224。第四，客户机通常不知道后台服务器或其它处理设备的状态。第五，没有工业标准软件或 API 用于负载平衡。和第六，即使客户机含有能够进行高效服务的相关信息，客户机通常也不可以选择特定的服务器。

因此，最好是能提供一种分布式处理系统，尤其是能提供一种具有现有技术分布式处理系统的优点而没有其固有缺点的分布式处理系统软件。该软件应该考虑到通常用在客户机／服务器、多层或对等分布式处理系统中的工业标准 API。该软件应该支持各种各样的计算机编程模型。并且，该软件还应该能允许（1）提高容错性、（2）有效率的规模可伸缩性、（3）有效的负载平衡、和（4）会话集中控制。改进了的计算机软件应该考虑到重新择径或网络重新配置。此外，该计算机软件还应该允许处理设备状态的确定。

发明概述

本发明的目的是提供一种改进分布式处理系统，尤其提供一种供分布式处理系统用的计算机软件。该计算机软件改进了分布式处理系统的容错性，以及能有高效率的规模可伸缩性。该计算机软件允许有效的负载平衡和会话

集中。该计算机软件支持重新择径或计算机网络的重新配置。该计算机软件支持各种各样的计算机编程模型和允许用在客户机／服务器和对等分布式处理结构两者中的工业标准 API 的使用。该计算机软件允许服务器或其它处理设备状态的确定。该计算机软件还支持在包括安全性模型在内的各种情况下的消息转发。

根据本发明的一个方面，分布式处理系统包括与第一处理设备和第二处理设备连接的通信媒体。第一处理设备包括模拟处理设备（“JVM1”）的第一软件程序，含有具有数据结构（“RJVM1”）的第一核心软件层。 第二处理设备包括模拟处理设备（“JVM2”）的第一软件程序，含有具有数据结构（“RJVM2”）的第一核心软件层。来自第一处理设备的消息通过第一处理设备中的第一核心软件层和第一软件程序传送到第二处理设备中的第一核心软件层和第一软件程序。

根据本发明的另一个方面，在第一处理设备中的第一软件程序是 Java™ 虚拟机（“JVM”），和在第一处理设备中的数据结构是远程 Java™ 虚拟机（“RJVM”）。类似地，在第二处理设备中的第一软件程序是 JVM，和在第二处理设备中的数据结构是 RJVM。第二处理设备中的 RJVM 对应于第一处理设备中的 JVM。

根据本发明的另一个方面，第一处理设备中的 RJVM 包括接头（socket）管理器软件部分、线程（thread）管理器软件部分、消息择径软件部分、消息压缩软件部分、和／或对等消失（peer-gone）检测软件部分。

根据本发明的另一个方面，第一处理设备利用从由下列协议组成的协议组中选择的协议与第二处理设备进行通信：Transmission Control Protocol（传输控制协议）（“TCP”）、Secure Sockets Layer（安全接头协议）（“SSL”）、Hypertext Transport Protocol（超文本传输协议）（“HTTP”）隧道技术、和 Internet InterORB Protocol（互联网 ORB 间协议）（“IIOP”）隧道技术。

根据本发明的另一个方面，第一处理设备包括存储 Java™ 应用程序的存储器。

根据本发明的另一个方面，第一处理设备是第二处理设备的对等物。此外，第一处理设备是服务器，和第二处理设备是客户机。

根据本发明的另一个方面，第二通信媒体与第二处理设备连接。第三处理设备与第二通信媒体连接。第三处理设备包括模拟处理设备（“JVM3”）的

第一软件程序，含有具有第一数据结构（“RJVM1”）和第二数据结构（“RJVM2”）的核心软件层。

根据本发明的另一个方面，第一处理设备包括含有复制管理器软件部分的存根模块（“stub”）。复制管理器软件部分包括负载平衡软件部分和故障处理(failover)软件部分。

根据本发明的另一个方面，第一处理设备包括 Enterprise Java™ Bean 对象。

根据本发明的另一个方面，第一处理设备包括存储在树节点上的、含有存根模块库的命名树，和第二处理设备包括该命名树的复制件。

根据本发明的另一个方面，第一处理设备包括以无状态程序模型（stateless program model）编码的应用程序，和该应用程序包括无状态会话豆模块（stateless session bean）。

根据本发明的另一个方面，第一处理设备包括以无状态工厂程序模型（stateless factory program model）编码的应用程序，和该应用程序包括有状态会话豆模块（stateful session bean）。

根据本发明的另一个方面，第一处理设备包括以有状态程序模型（stateful program model）编码的应用程序，和该应用程序包括实体会话豆模块（entity session bean）。

根据本发明的另一个方面，提供包括信息存储媒体的制品。该制品包含将消息从第一处理设备中的 RJVM 传送到第二处理设备中的 RJVM 的第一组数字信息。

根据本发明的另一个方面，该制品包含第一组数字信息，包括含有从多个服务提供者中选择一个服务提供者的负载平衡软件程序的存根模块。

根据本发明的另一个方面，存根模块含有从多个服务提供者中删除一个发生了故障的服务提供者的故障处理软件部分。

根据本发明的另一个方面，负载平衡软件部分基于对一个特定服务提供者的亲缘关系(affinity)选择一个服务提供者。

根据本发明的另一个方面，负载平衡软件部分以循环（round robin）方式选择服务提供者。

根据本发明的另一个方面，负载平衡软件部分以随机方式选择服务提供者。

根据本发明的另一个方面，负载平衡软件部分根据每个服务提供者的负载从多个服务提供者中选择一个服务提供者。

根据本发明的另一个方面，负载平衡软件部分根据所请求的数据类型从多个服务提供者中选择一个服务提供者。

根据本发明的另一个方面，负载平衡软件部分根据物理上最接近的服务提供者从多个服务提供者中选择一个服务提供者。

根据本发明的另一个方面，负载平衡软件部分根据每个服务提供者响应的时间周期从多个服务提供者中选择一个服务提供者。

根据本发明的另一个方面，该制品包含第一组数字信息，包括从多个服务提供者中选择一个服务提供者的 Enterprise Java™ Bean 对象。

根据本发明的另一个方面，存根模块存储在分布式处理系统中的处理设备中。存根模块包括由下列步骤组成的方法：获取服务提供者列表，和从该服务提供者列表中选择一个服务提供者。

根据本发明的另一个方面，该方法还包括从服务提供者列表中删除发生故障的服务提供者。

根据本发明的另一个方面，一种设备包括与第一处理设备和第二处理设备连接的通信媒体。第一处理设备存储命名树，该命名树包括用于访问服务提供者的远程方法调用（“RMI”）存根模块。第二处理设备含有复制的命名树和服务提供者。

根据本发明的另一个方面，命名树含有包括当前服务提供者的服务库的节点。

根据本发明的另一个方面，服务库包括存根模块(stub)。

根据本发明的另一个方面，分布式处理系统包括与第二计算机连接的第一计算机。第一计算机含有命名树，包括访问服务提供者的远程调用存根模块。第二计算机含有复制的命名树和服务提供者。

根据本发明的另一个方面，提供了包括与第二处理设备连接的第一处理设备的分布式处理系统。第一处理设备具有 JVM 和包括第一 RJVM 的第一核心软件层。第二处理设备具有第一JVM和包括第二RJVM的第一核心软件层。当在第一 JVM 和第二 JVM 之间没有可用的接头时，可以将消息从第一处理设备传送到第二处理设备。

根据本发明的另一个方面，第一处理设备在防火墙之后，在小应用程序

(applet)安全模型下运行，或者是一个客户机，和第二处理设备也是一个客户机。

通过参照随后的附图、详细说明和权利要求书，可以看出本发明的其它特征和优点。

附图简述

图 1a 表示现有技术的客户机／服务器结构;

图 1b 表示现有技术的 Java™ 企业 API;

图 1c 表示多层结构;

图 2a 表示现有技术的对等结构;

图 2b 表示现有技术的事务处理结构;

图 3a 表示本发明实施例的简化软件方块图;

图 3b 表示图 3a 所示的核心模块的简化软件方块图;

图 3c 表示集团型(clustered)企业 Java™ 结构;

图 4 表示集团型企业 Java™ 命名服务结构;

图 5a 表示智能存根模块（smart stub）结构;

图 5b 表示 EJB 对象结构;

图 6a 是说明负载平衡方法的控制流程图;

图 6b-g 是说明负载平衡方法的控制流程图;

图 7 是说明故障处理方法的的控制流程图;

图 8 表示在图 3－5 所示的集团型企业 Java™ 结构中客户机／服务器的硬件和软件部分。

通过参照附图和如下的详细说明，可以更好地理解本发明。在附图中，相同的标号表示相同的部分。

详细说明

Ⅰ.集团型企业 Java™ 分布式处理系统

A. 集团型企业 Java™ 软件结构

图 3a 示出了根据本发明实施例的、在集团型企业 Java™ 系统的处理设备中的软件层的简化方块图 380。下面描述集团型企业 Java™ 分布式处理系统的详细说明。软件的第一层包括通信媒体软件驱动器 351，用于传送和接收在诸如以太网局域网之类的通信媒体上的信息。包括传输控制协议（“TCP”）

软件部分 353 和因特网（“IP”）软件部分 352 的操作系统 310 是以特定格式检索提取和发送信息包或信息块的上层软件层。“上层”软件层一般定义为一个利用或访问一个或多个“下层”软件层的软件部分或几个软件部分。接着，实现 JVM 354。然后，将含有远程 Java™ 虚拟机 356 的核心模块层 355 放置在 JVM 354 的上面。下面要详述的核心模块 355 用于在集团型企业 Java™ 分布式处理系统中的处理设备之间传送消息。远程方法调用 357 和企业 Java™ 豆模块(bean)358 是核心模块 355 的上层软件层。EJB 358 是用于各种 Java™ 应用程序的容器。

图 3b 表示图 3a 所示的核心模块 355 的细节图。核心模块 355 包括接头管理器部分 363、线程管理器部分 364、和 RJVM 356。RJVM 356 是包括消息择径软件部分 360、含有简表 161c 的消息压缩软件部分 361、和对等消失检测软件部分 362 的数据结构。RJVM 356 和线程管理器部分 364 与接头管理器部分 363 交互操作以在处理设备之间传输信息。

B. 分布式处理系统

图 3c 表示集团型企业 Java™ 分布式处理系统 300 的简化方块图。处理设备与通信媒体 301 连接。通信媒体 301 可以是有线和／或无线的通信媒体，或它们的组合。在一个实施例中，通信媒体 301 是局域网（LAN）。在另一个实施例中，通信媒体 301 是广域网（WAN），例如，因特网或万维网（WWW）。再在另一个实施例中，通信媒体 301 是 LAN 和 WAN 两者。

各种不同类型的处理设备都可以与通信媒体 301 连接。在一个实施例中，处理设备可以是如下所述的、如图 8 所示的通用计算机 800。本领域的普通技术人员应该理解，图 8 和下列说明仅描述了一种特定类型的处理设备，而按照本发明的实施例可以使用带有不同软件和硬件配置的许多其它类型的处理设备。在另一个实施例中，处理设备可以是打印机、手持式计算机、膝上型计算机、扫描仪、蜂窝式电话、寻呼机、或它们的等效物。

图 3c 表示本发明中服务器 302 和 303 与通信媒体 301 连接的实施例。服务器 303 还与通信媒体 305 连接，通信媒体 305 可以具有与上面针对通信媒体 301 所述相似的实施例。客户机 304 也与通信媒体 305 连接。在另一个实施例中，客户机 304 可以如图 3 中虚线和方块所示的那样与通信媒体 301 连接。应该理解为，在另一个实施例中，服务器 302 是（1）客户机和服务器两者、或（2）客户机。类似地，图 3 表示了其中表示三个处理设备的实施例，

而本发明的其它实施例包括了如省略号所示的许多其它处理设备或通信媒体。

服务器 302 分别利用网络软件 302a 和网络软件 303a 将通信媒体 301 上的信息传输到服务器 303。在一个实施例中，网络软件 302a、303a、和 304a 包括通信媒体软件驱动器 351、传输控制协议软件 353 和因特网协议软件 352（“TCP／IP”）。客户机 304 还包括网络软件 304a,用于通过通信媒体 305 将信息传输到服务器 303。服务器 303 中的网络软件 303a 还用于通过通信媒体 305 将信息传输到客户机 304。

根据本发明的实施例，集团型企业 Java™ 结构 300 中的每个处理设备都包括支持多层和对等功能两者的消息传递核心模块 355。核心模块是用于向处理设备上的其它软件程序提供基本服务的软件程序。

具体地说，服务器 302、服务器 303、和客户机 304 分别含有核心模块 302b、303b、和 304b。具体地，为了使两个 JVM 交互操作，无论它们是客户机还是服务器，每个 JVM 都构造代表另一个的 RJVM。消息从一侧上的上层发出，通过相应的 RJVM，跨过通信媒体，再通过对等的 RJVM，最后被传送到另一侧上的上层。在各种实施例中，消息可以利用各种不同的协议传输，包括（但不限于）：Transmission Control Protocol/Internet Protocol（“TCP/IP”）、Secure Sockets Layer（“SSL”）、Hypertext Transport Protocol（“HTTP”）隧道技术、和 Internet InterORB Protocol（“IIOP”）隧道技术、和它们的组合。RJVM 和接头管理器创建和维护以这些协议为基本接头和在上层中的所有对象之间共享它们。接头(socket)是代表分布式处理系统中的处理设备之间的端点(terminal)的逻辑位置。核心模块维护执行线程库，和线程管理器软件部分 364 多路复用接头读取和请求执行之间的线程。线程是执行程序代码段或功能的序列。

例如，服务器 302 包括 JVM1 和 Java™ 应用程序 302c。服务器 302 还包括代表服务器 303 的 JVM2 的 RJVM2。如果消息要从服务器 302 发送到服务器 303，则消息通过服务器 302 中的 RJVM2 发送到服务器 303 中的 RJVM1。

C.消息转发

集团型企业 Java™ 网络 300 能够通过中间服务器转发消息。如果客户机通过前台网关从后台服务器请求服务，则这种功能是重要的。例如，来自服

务器 302（客户机 302）的消息并具体地 JVM1 可以通过服务器 303（前台网关）或 JVM2 转发到客户机 304（后台服务器 304）或 JVM3。这种功能在控制会话集中或在服务器与各种客户机之间建立多少个连接时是重要的。

并且，消息转发可以用在在两个 JVM 之间不能创建接头的情况中。例如，消息的发送者正在小应用程序安全模式下运行，而这种模式不允许创建到原始服务器的接头。小应用程序安全模型的详细说明可以从 http://www.javasoft.com 站点上获得，将这个站点列在这里以供参考。另一个例子包括当消息的接收者处于防火墙之后时。此外，如下所述，消息转发也可以应用在发送者是客户机和接收者是客户机时，因此不接受任何输入接头。

例如，如果消息从服务器 302 发送到客户机 304，则该消息将不得不通过服务器 303 择径。具体地说，将使 RJVM3（代表客户机 304）之间的、如 302f 所示的消息切换是切换到服务器 302 中的 RJVM2（代表服务器 303）。消息将利用接头 302e 在服务器 302 中的 RJVM2 和服务器 303 中的 RJVM1 之间传输。然后，消息按虚线 303f 所示的那样，从服务器 303 中的 RJVM1 切换到服务器 303 中的 RJVM3。接着，消息将在服务器 303 中的 RJVM3 和客户机 304 中的 RJVM2 的接头之间传送。最后，消息将按虚线 304f 所示的那样，从客户机 304 中的 RJVM2 传送到客户机 304 中的 RJVM1。

D. 重新择径

客户机／服务器中的 RJVM 能够在任何时候将通信路径或通信媒体切换到其它的 RJVM。例如，如果客户机 304 创建到服务器 302 的直接接头，则服务器 302 就能够开始使用该接头来代替通过服务器 303 的消息转发。这个实施例用代表客户机 304 的虚线和方块示出。在一个实施例中，通过 RJVM 传输消息的使用确保了在出现网络重新配置之后可靠有序的消息传送。例如，如果客户机 304 被重新配置成通信媒体 301 来代替图 3 所示的通信媒体 305。在另一个实施例中，消息可以不按顺序传送。

RJVM 进行几种通过路由择径(routing)实现的端到端操作。首先，RJVM 负责各个客户机／服务器意外死机时的检测。在一个实施例中，如图 3b 所示对等消失选择软件部分 362 负责这种功能。在一个实施例中，当在预定时间间隔内没有发送其它消息时，RJVM 将心跳（heartbeat）消息发送到其它客户机／服务器。如果客户机／服务器在预定计数时间内没有接收到心跳消息，

则检测到应该发送心跳的客户机／服务器发生故障了。在一个实施例中，通过连接超时或发生故障的客户机／服务器是否在预定时间量内不发送消息来检测发生故障的客户机／服务器。在另一个实施例中，发生故障的接头指示发生故障的客户机／服务器。

其次，在消息串行化期间，各个 RJVM，特别是消息压缩软件 361，缩减共同传输的数据值以降低消息大小。为了实现这个目的，每个 JVM／RJVM 对保持匹配缩减表。例如，JVM1 包括缩减表，和 RJVM1 包括匹配缩减表。在消息在中间服务器之间转发期间，在路由中的中间服务器上消息的主体不进行去串行化。

E. 多层／对等功能

集团型企业 Java™ 结构 300 考虑到多层和对等编程。

集团型企业 Java™ 结构 300 支持与多层分布式处理结构一致的、用于客户机／服务器编程的显式语法。举例来说，如下的客户机方代码片段将信息消息写入服务器日志文件中：

```
T3Client clnt = new T3Client("t3://acme:7001");
LogServices log = clnt.getT3Services().log();
Log.info("Hello from a client");
```

第一行利用 t3 协议建立与顶级（acme） 服务器会话。如果 RJVM 还不存在，则每个 JVM 构造关于另一方的 RJVM，并建立底层(underlying)TCP 接头。这个会话的客户机方表示 - T3Client 对象 - 和服务器方表示通过这些 RJVM 进行通信。服务器方支持各种各样的服务，包括数据库访问、远程文件访问、工作空间、事件和日志记录。第二行获得 LogServices 对象，和第三行写消息。

集团型企业 Java™ 计算机结构 300 还支持与对等分布式处理结构一致的服务器中性的语法。举例来说，如下的代码片段从服务器上的遵从 JNDI 的命名服务中获得关于 RMI 对象的存根模块，并调用其方法之一。

```
Hashtable env = new Hashtable();
env.put(Context.PROVIDER_URL, "t3://acme:7001");
env.put(Context.INITIAL_CONTEXT_FACTORY,
"weblogic.jndi.WeblogicInitialContextFactory");
```

```
Context ctx = new InitialContext(env);
Example e = (Example)ctx.lookup("acme.eng.example");
result = e.example(37);
```

在一个实施例中，JNDI 命名上下文被封装成 RMI 对象以实现远程访问。因此，上面代码显示了一种类型的 RMI 引导(bootstrap)。前四行获得顶级服务器上关于初始上下文的 RMI stub。如果 RJVM 还不存在，则每一方都构造关于另一方的 RJVM，并建立用于 t3 协议的底层 TCP 接头。调用方对象－RMI stub 和被叫方对象－RMI impl－通过 RJVM 进行通信。第五行查找在名称 acme.eng.example 上的另一个 RMI 对象，即 Example 和第六行调用 Example 方法之一。在一个实施例中，Example impl 不在与命名服务相同的处理设备上。在另一个实施例中，Example impl 是在客户机上。Example 对象的调用导致了适当 RJVM 的创建，如果它们还不存在的话。

II.知晓复制的或智能的存根模块／EJB 对象

在图 3c 中，处理设备通过复制 RMI 和／或 EJB 对象能够向结构 300 中的其它处理设备提供服务。因此，结构 300 是易于伸缩的和容错的。通过将复制的 RMI 和／或 EJB 对象加入现有的处理设备或新附加的处理设备中，可以容易地将附加服务加入结构 300 中。并且，由于 RMI 和／或 EJB 对象可以通过结构 300 复制，因此，单个处理设备、多个处理设备、和／或通信媒体可能发生故障，但仍然不会使结构 300 不能工作或显著变坏。

图 5a 表示结构 500 中的知晓复制（“RA”）的或智能的存根模块 580。结构 500 包括与通信媒体 501 连接的客户机 504。服务器 502 和 503 分别与通信媒体 501 连接。永久性存储装置 509 分别通过通信媒体 560 和 561 与服务器 502 和 503 连接。在各种各样的实施例中，通信媒体 501、560 和 561 可以是如上所述的有线和／或无线通信媒体。类似地，在一个实施例中，客户机 504、服务器 502、和服务器 503 可以是如上所述的客户机和服务器两者。本领域的普通技术人员应该理解，在其它可替代的实施例中，多个其它服务器和客户机可以如省略号所示的那样包括在结构 500 中。此外，如上所述，在其它可替代的实施例中，客户机 504、服务器 502、和服务器 503 的硬件和软件配置如下面所描述的和图 8 所示的那样。

RA RMI 存根模块 580 是一种能够根据负载平衡方法 507 和／或故障处

理方法 508 找出几乎所有服务提供者并在它们之间切换的智能存根模块。在一个实施例中，RA 存根模块 580 包括选择适当负载平衡方法 507 和／或故障处理方法 507 的复制处理器 506。在其它可替代的实施例中，可以实施单负载平衡方法和／或单故障处理方法。在其它可替代的实施例中，复制处理器 506 可以包括多负载平衡方法和／或多故障处理方法以及它们的组合。在一个实施例中，复制处理器 506 实现如下接口：

```
public interface ReplicaHandler{
    Object loadBalance(Object currentProvider)throws
RefreshAbortedException;
    Object failOver(Object failedProvider,
    RemoteException e)throws
RemoteException;
}
```

紧接调用一种方法之前，RA 存根模块 580 调用负载平衡方法 507，它接管当前服务器，并返回替换者。例如，客户机 504 可能正在利用服务器 502 检索数据库 509a 或个人存储装置 509 的数据。由于服务器 502 负担服务请求过重，因此，负载平衡方法 507 可以切换到服务器 503。处理器 506 可能选择完全在调用方上的服务器替换者，也许利用关于服务器 502 负载的信息，或者，处理器 506 可以请求服务器 506 检索特定类型的数据。例如，处理器 506 可以选择特定服务器解方程，因为服务器已经提高了计算能力。在一个实施例中，复制处理器 506 实际上不需要在每次调用时都切换提供者，因为复制处理器 506 正试图使创建的连接数最少。

图 6a 是说明图 5a-b 所示的负载平衡软件 507 的控制流程图。应该理解，图 6a 是说明由负载平衡方法 507 中的软件完成的功能或步骤的逻辑序列的控制流程图。在其它可替代的实施例中，可以完成附加的功能或步骤。并且，在其它可替代的实施例中，硬件可以执行特定的功能或所有的功能。

负载平衡软件 507 从圆圈 600 所示的位置开始。然后，在逻辑块 601 中确定调用线程是否已经建立起对特定服务器的“亲缘关系”。客户机与协调它当前事务处理的服务器有亲缘关系，和服务器与它自己有亲缘关系。如果亲缘关系已建立，则将控制进到逻辑块 602，否则，则控制进到逻辑块 604。在逻辑块 602 中确定有亲缘关系的服务器是否提供所请求的服务。如果是，则

将控制进到逻辑块 603。否则，则控制进到逻辑块 604。将有亲缘关系的服务器上的服务提供者返回给逻辑块 603 中的客户机。在逻辑块 604 中，联系命名服务并获取更新了的当前服务提供者列表。在逻辑块 605 中调用 getNextProvider（取下一服务者）方法以获取服务提供者。getNextProvider 方法的各种实施例显示在图 6b-g 中，下面将作详细描述。在逻辑块 606 中获得服务。然后，如果在逻辑块 606 中没有提供服务，则调用故障处理方法 508，最后，负载平衡方法 507 从逻辑块 608 所示的位置退出。故障处理方法 508 的实施例表示在图 7 中，下面将作详细描述。

图 6b-g 示出了图 6a 的逻辑块 605 中所使用的 getNextProvider 方法的各种实施例。如图 6b 所示，getNextProvider 方法以循环方式选择服务提供者。如圆圈 621 所示进入 getNextProvider 方法 620。在逻辑块 622 中获取当前服务提供者列表。在逻辑块 623 中递增指针。在逻辑块 624 中根据指针选择下一个服务提供者，并在逻辑块 625 中返回新的服务提供者，最后，如圆圈 626 所示退出 getNextProvider 方法。

图 6c 示出了通过随机选择服务提供者来获取服务提供者的 getNextProvider 方法的另一种可替代实施例。如圆圈 631 所示进入 getNextProvider 方法 630。如逻辑块 632 所示的那样获取当前服务提供者列表。如逻辑块 633 所示的那样随机选择下一个服务提供者，并在逻辑块 634 中返回新的服务提供者，最后，如圆圈 635 所示退出 getNextProvider 方法。

getNextProvider 方法还有一个实施例显示在图 6d 中，它根据各服务提供者的负载获取服务提供者。如圆圈 641 所示进入 getNextProvider 方法 640。在逻辑块 642 中获取当前服务提供者列表。在逻辑块 643 中获取每个服务提供者的负载。然后，在逻辑块 644 中选择负载最轻的服务提供者。最后，在逻辑块 645 中返回新的服务提供者，并且如圆圈 646 所示退出 getNextProvider 方法。

getNextProvider 方法的再一个可替代的实施例显示在图 6e 中，它根据从服务提供者获得的数据类型获取服务提供者。如圆圈 651 所示进入 getNextProvider 方法 650。在逻辑块 652 中获取当前服务提供者列表。在逻辑块 653 中确定从服务提供者请求的数据的类型。然后，在逻辑块 654 中根据数据类型选择负载服务提供者。最后，在逻辑块 655 中返回新的服务提供者，并且如圆圈 656 所示退出 getNextProvider 方法。

getNextProvider 方法的再一个实施例表示在图 6f 中，它根据服务提供者物理位置选择服务提供者。如圆圈 661 所示进入 getNextProvider 方法 660。在逻辑块 662 中获取当前服务提供者列表。在逻辑块 663 中确定到每个服务提供者的物理距离，和在逻辑块 664 中选择到正在请求的客户机的物理距离最近的服务提供者。最后，在逻辑块 665 中返回新的服务提供者，并且如圆圈 666 所示退出 getNextProvider 方法 660。

getNextProvider 方法的还在一个的实施例显示在图 6g 中，它根据服务提供者响应以前的请求所花费的时间量选择服务提供者。如圆圈 671 所示进入 getNextProvider 方法 670 的控制。在逻辑块 672 中获取当前服务提供者列表。在逻辑块 673 中确定每个服务提供者响应特定消息的时间间隔。然后，在逻辑块 674 中选择以最短时间间隔响应的服务提供者。最后，在逻辑块 675 中返回新的服务提供者，并且如圆圈 676 所示退出来自 getNextProvider 方法的控制。

如果服务方法的调用以保证重试这样的方式出故障，则 RA 存根模块 580 调用故障处理方法 508，它接管发生故障的服务器和提示是什么故障的异常，并返回一重试用的新服务器。如果新服务器不适用，则 RA 存根模块 580 报告异常。

图 7 是说明图 5a-b 所示的故障处理软件 508 的控制流程图。如圆圈 700 所示进入故障处理方法 508。在逻辑块 701 从服务的当前提供者列表中删除发生故障的提供者。然后，调用 getNextProvider 方法以便获取服务提供者。最后，在逻辑块 703 中返回新的服务提供者，和如圆圈 704 所示退出故障处理方法 508。

虽然图 6－7 表示复制处理器 506 的实施例，但其它可替代的实施例还包括以循环方式实现的如下功能或它们的组合。

首先，要维护服务的服务器或服务提供者的列表。每当列表需要使用和列表最近没有被更新时，处理器 506 就联系如下所述的命名服务并获取提供者的最新列表。

其次，如果处理器 506 打算从列表中选择提供者和存在到主服务器的现存的 RJVM 级连接，该主服务器在最后的心跳周期未接收到消息，则处理器 506 跳过该提供者。在一个实施例中，由于对等的消失是在几个这样的心跳间隔之后确定的，因此，服务器可以在以后恢复。这样，就可以获取基于服

务器负载的负载平衡。

第三，当提供者发生故障时，处理器 506 从列表中删除该提供者。这样就避免了重复尝试使用不工作的服务提供者所引起的延迟。

第四，如果服务正在从作为服务的提供者的宿主的服务器调用的，那么就使用该提供者。这有利于链式调用服务的提供者的共置托管(co-location)。

第五，如果服务正在在事务处理的范围内被调用和起事务处理协调者作用的服务器作为服务的提供者的宿主，那么就使用该提供者。这有利于在事务处理范围内的提供者的共置托管。

能够在方法调用期间发生的故障可以分类成:（1）与应用程序有关的、或（2）与基础设施有关的。在发生与应用程序有关的故障的情况下，RA 存根模块 580 将不重试操作，因为不能期望改善这种情况。在发生与基础设施有关的故障的情况下，RA 存根模块 580 也许能够，或也许不能够可靠地重试操作。一些初始非幂等操作，例如递增数据库中字段的值，可能完成。在一个实施例中，只有当（1）用户已经声明服务方法是幂等的，或者（2）系统可以确定请求的处理从未开始。作为后者的一个例子，如果作为负载平衡方法的一部分，存根模块 580 切换到其宿主已经发生故障的服务提供者，RA 存根模块 580 将重试。作为另一个例子，RA 存根模块 580 得到对事务处理操作的否定确认，则 RA 存根模块 580 将重试。

RMI 编译器识别命令该编译器生成用于对象的 RA 存根模块的特殊标志。附加的标志可以用于规定服务方法是幂等的(idempotent)。在一个实施例中，RA 存根模块 580 将利用如上所述的和图 5a 所示的复制处理器。附加的标志可以用于规定不同的处理器。另外，在服务被布置的点上，即被结合在如下所述的集团型命名服务的点上，处理器可以被取代。

图 5b 表示本发明的另一个实施例，其中 EJB 对象 551 用于取代图 5a 所示的存根模块。

III.复制的遵从 JNDI 的命令服务

如图 4 所示，对结构 400 中的服务提供者的访问是通过遵从 JNDI 的命名服务获得的，在整个结构 400 中复制遵从 JNDI 的命令服务，使得不存在任何单个故障点。因此，如果提供遵从 JNDI 的命名服务的处理设备发生了故障，则可以使用具有复制的命名服务的另一个处理设备。为了提供服务的示例，服务器在复制命名树中的特定节点上通告服务的提供者。在一个实施例中，

每个服务器将用于提供者的 RA 存根模块附加到存储在命名树的服务器备份中的节点上的兼容服务库。如果新提供(offer)的类型与现有库中提供的类型不兼容，则使新提供暂停，并通过 ConflictHandler 接口作出往调用(callback)。在撤消任一种类型的提供之后，另一种提供将最终被装在每一处。当客户机查找服务时，客户机获取联系服务库的 RA 存根模块以刷新服务提供者的客户机列表。

图 4 表示结构 400 中复制的命名服务。在一个实施例中，服务器 302 和 303 分别提供示例的服务提供者 P1 和 P2，并且分别拥有命名服务树 402 和 403 的复制品。命名服务树 402 和 403 中的节点 acme.eng.example 分别拥有服务库 402a 和 403a，包含到 Example 服务提供者 P1 和 P2 的参考点。客户机 304 通过在 acme.eng.example 节点上进行命名服务查找获取 RA 存根模块 304e。存根模块 304e 联系服务库的示例以获取到可用服务提供者的参考点的当前列表。存根模块 304e 可以按负载平衡和故障处理的需要在服务库的示例之间切换。

关于命名服务的初始上下文的存根模块是知晓复制的或智能的存根模块，它最初作命名服务提供者之间的负载平衡，并在发生故障的情况下进行切换。命名服务树的每个示例包含当前命名服务提供者的完整列表。存根模块从当前正在使用的示例中获取刷新列表。为了引导这个处理，系统使用 Domain Naming Service（域名命名服务）（“DNS”）以找出示例的（可能不完整的）初始列表和从它们之一中获取完整列表。举例来说，可以按如下获取关于于命名服务的初始上下文的存根模块:

```
Hashtable env = new Hashtable();
env.put(Context.PROVIDER_URL, “t3://acmeCluster:7001”);
env.put(Context.INITIAL_CONTEXT_FACTORY,
“weblogic.jndi.WebLogiclnitialContextFactor”);
Context ctx = new InitialContext(env);
```

结构中的一些分组服务器已经被约束在名称 acmeCluster 下的 DNS 中。并且，应用程序仍然能够规定各个服务器的地址，但是，当应用程序首先试图获取存根模块时，应用程序将存在单个故障点。

可靠的多点传送协议是所期望的。在一个实施例中，通过 IP 多点传送或点到点协议分配提供者存根，并创建复制的命名树。在 IP 多点传送实施例中，

有三种类型的消息：Heartbeat、Announcement（通知）、StateDump（状态转储）。Heartbeat 用于在服务器之间传输信息，和通过它们的存在与否，识别发生故障的服务器。Announcement 包含一组服务的提供和撤消。来自每个服务器的 Announcement 按顺序编号。每个接收器处理 Announcement 以便识别丢失的 Announcement。每个服务器在它的 Heartbeat 中包括已经发送的最后一个 Announcement 的序号。关于丢失的 Announcement 的 Negative Acknowledgement("NAK"，否认)包括在随后输出的 Heartbeat 中。为了处理 NAK，每个服务器都保留服务器已经发送的最后几个 Announcement 的列表。如果关于 Announcement 已经被删除的 NAK 到达，服务器就发送 StateDump，它包含服务器的服务的完整列表和它下一个 Announcement 的序号。当新服务器加入到现有的结构中时，新服务器 NAK（否认）来自每个其它服务器的第一消息，这导致 StateDump 被发送。如果服务器在预定时间间隔之后没有从另一个服务器接收到心跳（Heartbeat），则服务器撤消由没有产生心跳（Heartbeat）的服务器提供的所有服务。

Ⅳ.编程模型

用在图 3－5 所示的结构中的应用程序随要处理应用程序状态的方式而定使用三种基本编程模型之一：（1）无状态的或直接的、（2）无状态工厂(stateless factory)的或间接的、或（3）有状态的或有目标的。在无状态模型中，通过命名服务查查找表返回的智能存根模块直接参考服务提供者。

```
Example e = (Example) ctx.lookup("acme.eng.example");
result1 = e.example(37);
result2 = e.example(38);
```

在该例中，由于为了负载平衡，智能存根模块能够在不同的服务提供者之间切换，因此，对实例的两种调用可以由不同的服务提供者来处理。因此，Example 服务对象不能够代表应用程序在内部存储信息。通常，只有当提供者无状态时，才使用无状态模型。举例来说，纯粹无状态提供者可以计算其变元的一些数学函数，并返回结果。无状态提供者可以为它们自己存储信息，例如为了财会的目的。更重要的是，无状态提供者可以访问底层的永久性存储器，和根据需要将应用程序的状态装入存储器中。例如，为了使实例返回作为变元进入其中的所有值的运行和。该实例可以从数据库中读取以前的和，

加入它的当前的变元中，写出新的值，然后将其返回。这种无状态服务模型提高了规模可伸缩性。

在无状态工厂编程模型中，通过查找返回的智能存根模块是创建其自身不是智能存根模块的所需服务提供者的工厂。

```
ExampleFactory gf = (ExampleFactory)
ctx.lookup("acme.eng.example");
Example e = gf.create();
result1 = e.example(37);
result2 = e.example(38);
```

在该例中，确保对实例的两种调用由同一服务提供者来处理。因此，服务提供者可以代表应用程序可靠地存储信息。无状态工厂模型应该应用在调用者需要与提供者进行"对话"时，例如，调用者和提供者可以进行往复协商。知晓复制的存根模块一般与无状态和无状态工厂模型中的相同。唯一不同之处在于存根模块是涉及服务提供者，还是涉及服务提供者工厂。

提供者工厂存根模块在努力创建提供者的过程中可以随意地进行故障处理，因为这种操作是幂等的。为了进一步提高间接服务的可用性，应用程序代码必须包含围绕服务创建和调用的显式重试循环。

```
while(true){
  try{
      Example e = gf.create();
      result1 = e.example(37);
      result2 = e.example(38);
      break;
  }catch (Exception e){
      if (!retryWarranred(e))
          throw e;
  }
}
```

这将会，例如，处理由工厂成功创建的提供者 e 的故障。在这种情况下，应用程序代码应该确定是否完成非幂等操作。为了进一步提高适用性，应用程序代码可以尝试取消这样的操作并重试。

在有状态编程模型中，服务提供者是由一些唯一系统级关键字标识的长寿命、有状态对象。可以利用这种模型访问的"实体"的例子包括远程文件系统和数据库表中的行。目标提供者可以由许多客户机访问许多次，这与每个提供者只由一个客户机使用一次的其它两种模型不同。用于目标提供者的存根模块可以通过关键字只不过是命名服务名称的直接查找表，或通过关键字包括对创建操作的变元的工厂两者之一获取。在每一种情况中，存根模块将不作负载平衡或故障处理。即使作的话，重试必须再次显式地获取存根模块。

在 EJB 中存在三种类型的豆模块，其中的每一种映射到三种编程模型之一。无状态会话豆模块(bean)是为特定调用者创建的，但在调用之间不保持内部状态。无状态会话豆模块映射到无状态模型。有状态会话豆模块是为特定调用者创建的，并在调用之间保持内部状态。有状态会话豆模块映射到无状态工厂模型。实体豆模块是由全系统关键字标识的奇异的、有状态对象。实体豆模块映射到有状态模型。所有三种类型的豆模块都由称为 EJB 家庭(home)的工厂创建的。在一个实施例中，它们创建的 EJB 家庭和豆模块两者都用 RMI 标记。在图 3－5 所示的结构中，用于 EJB 家庭的存根模块是智能存根模块。用于无状态会话豆模块的存根模块是智能存根模块，而用于有状态会话豆模块和实体豆模块的存根模块则不是。用于基于 EJB 的服务的复制处理器可以在它的布置(deplotment)描述符中规定。

为了创建间接的、基于 RMI 的服务，这是当对象要为调用者保持状态时所需要的，应用程序代码必须显式地构建工厂。有目标的、基于 RMI 的服务可以通过无需任何特殊标志地运行 RMI 编译器，然后将所得的服务结合到所复制的命名树中来创建。用于对象的存根模块将直接结合到命名树的每个示例中和将不创建任何服务库。这样就提供了关键字是命名服务名称的有目标服务。在一个实施例中，这用于创建远程文件系统。

Ⅴ.硬件和软件部分

图 8 表示图 3－5 所示的示例性服务器和／或客户机的硬件和软件部分。图 8 的系统包括通用计算机 800，它通过诸如连接 829 之类的一个或多个通信媒体与 LAN 840,并且也与在这里如因特网 880 所示的 WAN 相连接。通过 LAN 840，计算机 800 可以与诸如文件服务器 841 之类的其它本地计算机通信。在一个实施例中，文件服务器 841 是如图 3 所示的服务器 303。通过因特网 880，计算机 800 可以与诸如万维网服务器 881 之类的、本地的和远程

的其它计算机通信。在一个实施例中，万维网服务器 881 是如图 3 所示的服务器 303。应该理解，从计算机 800 到因特网 880 的连接可以通过各种各样的方式实现，例如，直接通过连接 829，通过局域网 840，或通过调制解调器（图中未示出）。

计算机 800 是个人或办公用计算机，它们可以是，例如，工作站、个人计算机、或其它单用户或多用户系统；一个示例性的实施例使用了 Sun SPARC-20 工作站（Sun Microsystems, Inc., Mountain View, CA）。为了便于说明，可以将计算机 800 方便地划分成硬件部分 801 和软件部分 802；但是，本领域的普通技术人员应该理解，这种划分是概念性的，并且多少有些随意性，在硬件和软件之间的界线也不是严格的。此外，应该理解，在主计算机与它的附属外设之间的界线也不是严格的，尤其是，被认为是一些计算机的外设的部分也可以被认为是其它计算机的整体部分。因此，例如，用户 I/O 820 可以包括键盘、鼠标、和显示监视器，它们的每一种既可以被认为是外围设备，也可以被认为是计算机本身的一部分，用户 I/O 820 还可以包括通常被认为是外设的本机打印机。作为另一个例子，永久性存储器 808 可以包括 CD－ROM（光盘只读存储器）单元，它既可以是外设，也可以内置在计算机中。

硬件部分 801 包括处理器（CPU）805、存储器 806、永久性存储器件 808、用户 I/O 820、和网络接口 825，它们都与总线 810 连接。这些部分都是本领域的普通技术人员所熟知的，因此，只需作简要说明。

处理器 805 可以是，例如，微处理器、或为了多重处理而构成的微处理器的集合。

存储器 806 可以包括只读存储器（ROM）、随机访问存储器（RAM）、虚拟存储器、或其它存储技术，它们既可以是单个形式，也可以是组合形式。永久性存储器件 808 可以包括，例如，硬磁盘、软磁盘、或其它永久性读－写数据存储技术，它们既可以是单个形式，也可以是组合形式。它还可以包括大型或归档存储器件，例如可以通过 CD－ROM 或其它大容量存储技术提供的那种。（请注意，文件服务器 841 提供了处理器 805 可以利用的附加存储能力。）

用户 I/O（输入/输出）硬件 820 通常包括诸如 CRT 或平板显示器之类的视频显示监视器、字母数字式键盘、和鼠标或其它点击设备，可选地，还可以包括打印机、光学扫描仪，或供用户输入输出用的其它设备。

网络 I/O 硬件 825 提供了计算机 800 和外界的接口。更明确地说，网络 I/O 硬件 825 让处理器 805 通过连接 829 与通过 LAN 840 和因特网 880 的其它处理器和设备通信。

软件部分 802 包括操作系统 850 和在操作系统 310 控制下的一组任务，例如，Java™ 应用程序 860、和重要的是，JVM 软件 354 和核心模块 355。操作系统 310 还让处理器 805 控制诸如永久性存储器件 808、用户 I/O 820、和网络接口 825 之类的各种设备。处理器 805 与存储器 806 和计算机系统 800 的其它部分相联系执行操作系统 310、应用 860、JVM 354、和核心 355 的软件。在一个实施例中，软件 802 包括如图 3c 的服务器 302 中所示的网络软件 302a、JVM1、RJVM2 和 RJVM3。在一个实施例中，Java™ 应用程序 860 是如图 3c 所示的 Java™ 应用程序 302c。

本领域的普通技术人员应该理解，图 8 的系统的意图是用于举例说明的，而不是用于限制的，各种各样计算、通信、和信息设备都可以用来取代或附加到图 8 所表示的设备上。例如，通过因特网 880 的连接一般涉及通过中间路由器计算机（未示出）的分组交换，和计算机 800 在典型的万维网客户机会话期间可能访问任何数量的万维网服务器，包括，但决不仅限于，计算机 800 和万维网服务器 881。

上述对本发明优选实施例的说明是为了图示和描述的目的而提供的。它不是穷举的，或将本发明限制在所公开的精确形式上。显然，各种改进和变动对于本领域的普通技术人员来说是显而易见的。选择和描述实施例是为了最佳地解释本发明的原理和它的实际应用，从而使本领域的其它普通技术人员能够理解有关各种实施例的和包括适用于所设想的特定用途的各种改进在内的本发明。应采用下列权利要求和它们的等效物来限定本发明的范围。

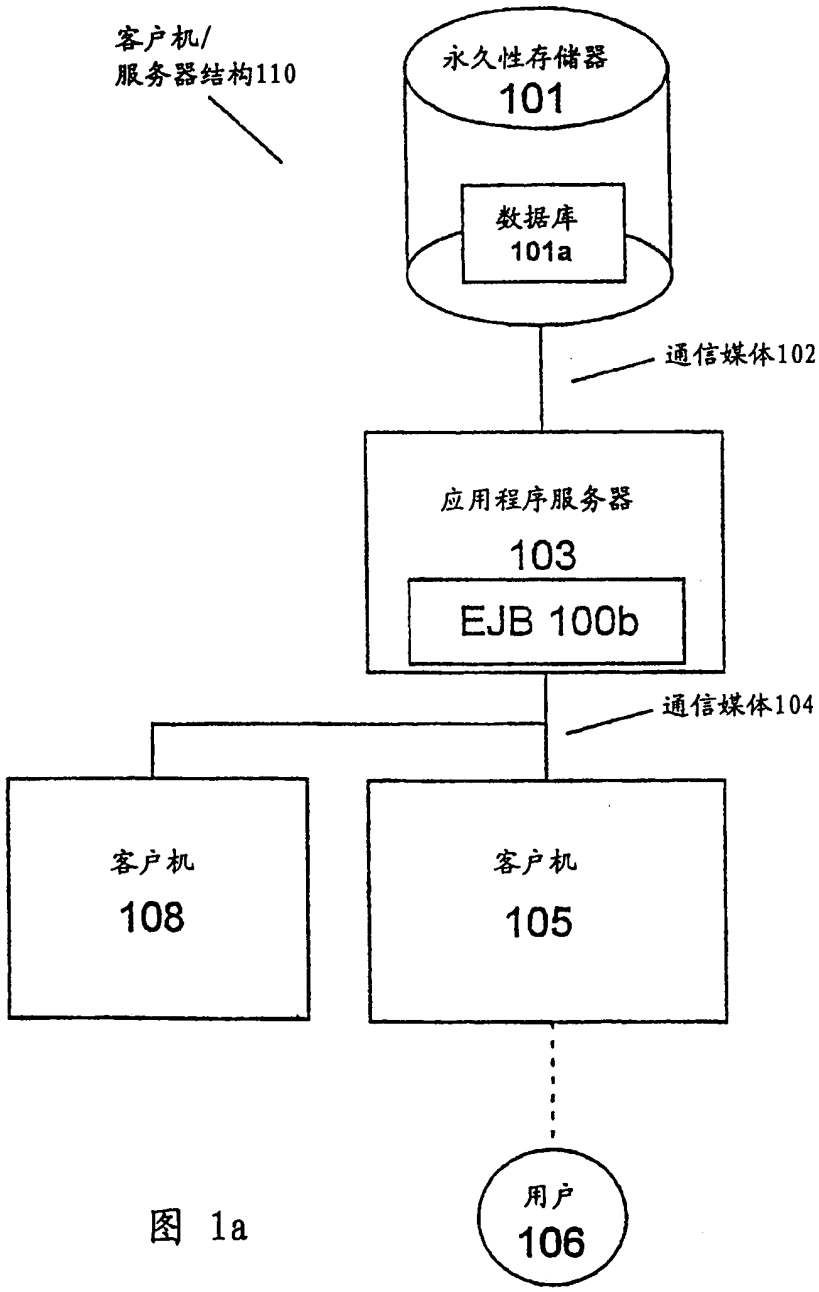

客户机/
服务器结构110
永久性存储器
101
数据库
101a
通信媒体102
应用程序服务器
103
EJB 100b
通信媒体104
客户机
108
客户机
105
用户
106

图 1a

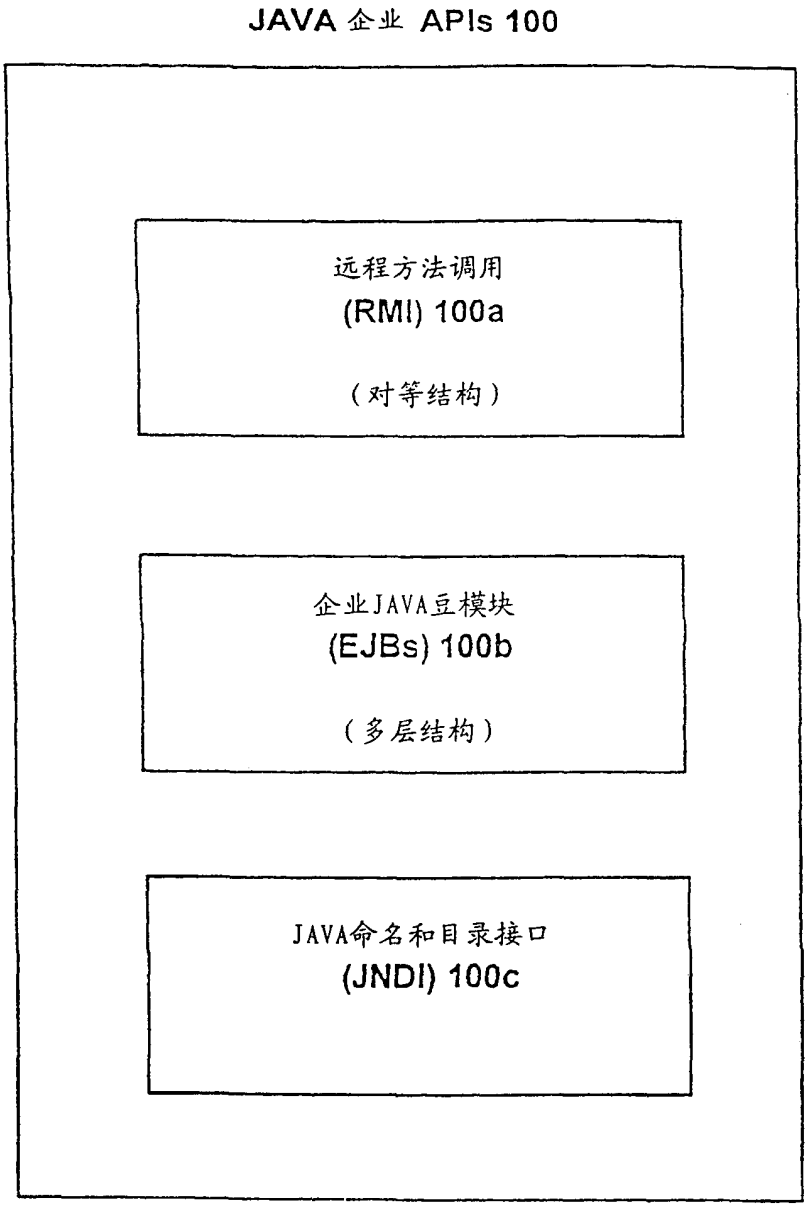

JAVA 企业 APIs 100
远程方法调用
(RMI) 100a
（对等结构）
企业JAVA豆模块
(EJBs) 100b
（多层结构）
JAVA命名和目录接口
(JNDI) 100c

图 1b

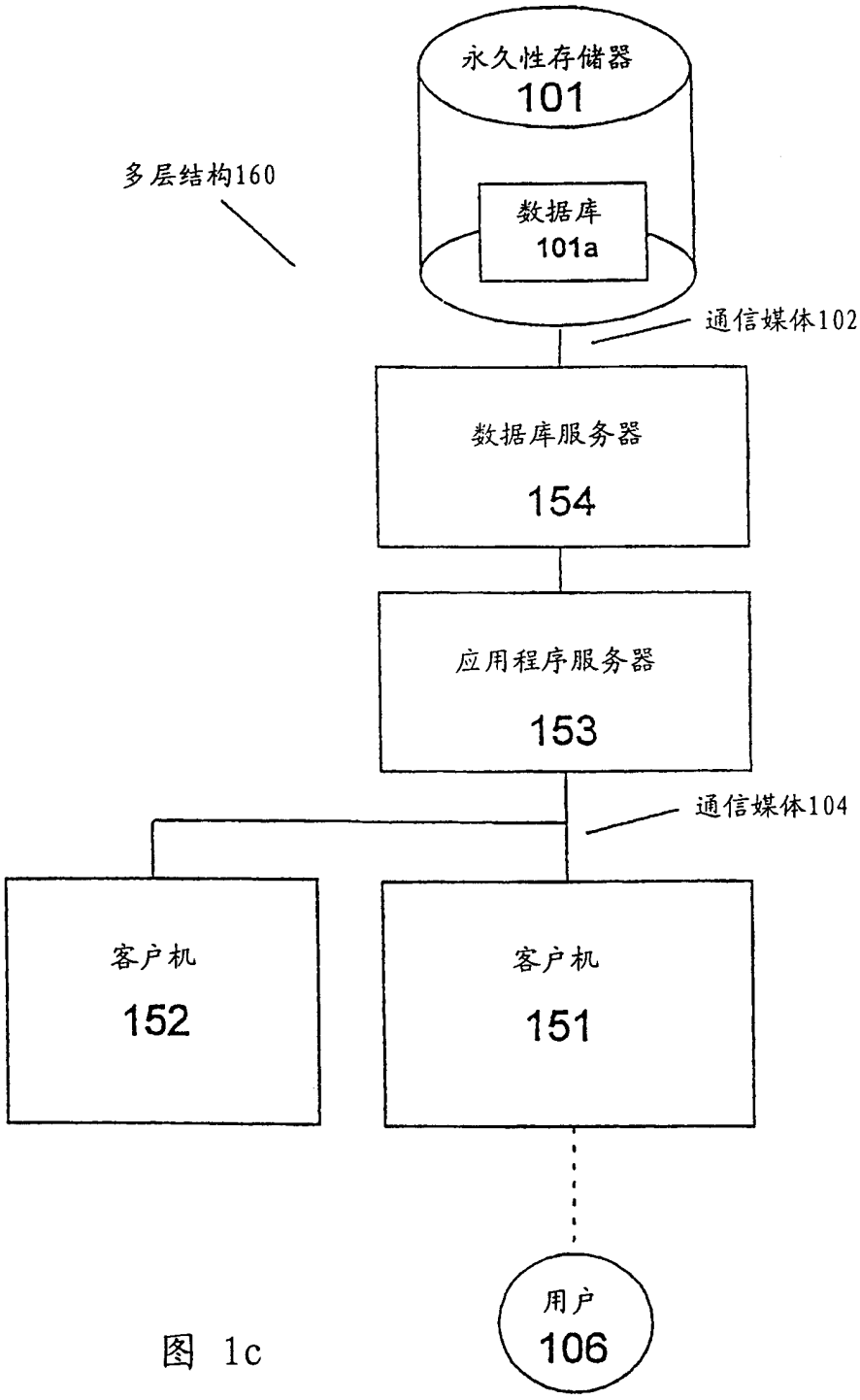

永久性存储器
101
多层结构160
数据库
101a
通信媒体102
数据库服务器
154
应用程序服务器
153
通信媒体104
客户机
152
客户机
151
用户
106

图 1c

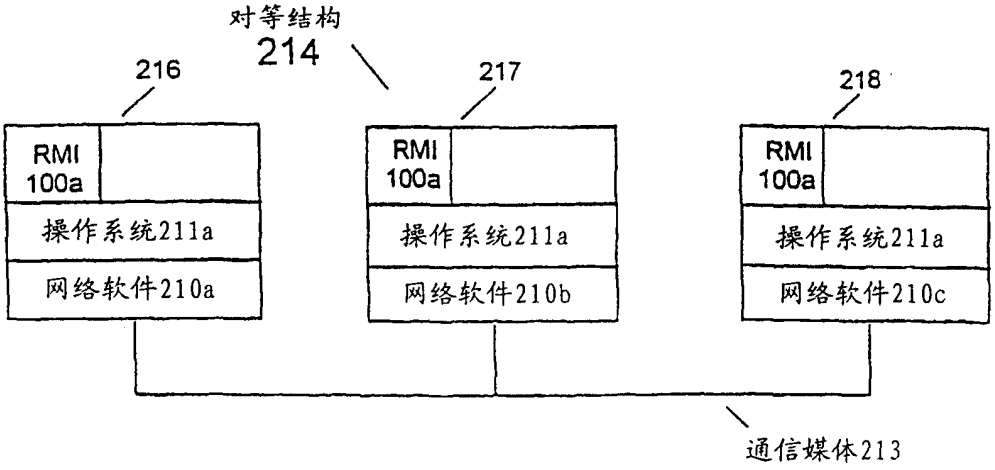

对等结构
214
216
217
218
RMI
100a
RMI
100a
RMI
100a
操作系统211a
网络软件210a
操作系统211a
网络软件210b
操作系统211a
网络软件210c
通信媒体213

图 2a

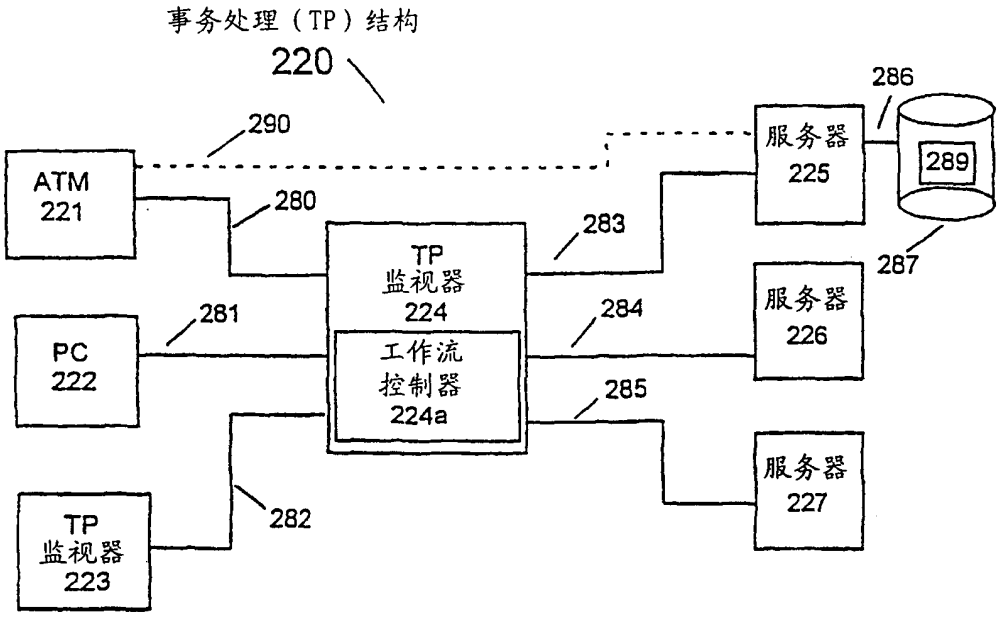

事务处理（TP）结构
220
290
286
ATM
221
280
TP
监视器
224
283
服务器
225
289
287
PC
222
281
工作流
控制器
224a
284
服务器
226
285
服务器
227
TP
监视器
223
282

图 2b

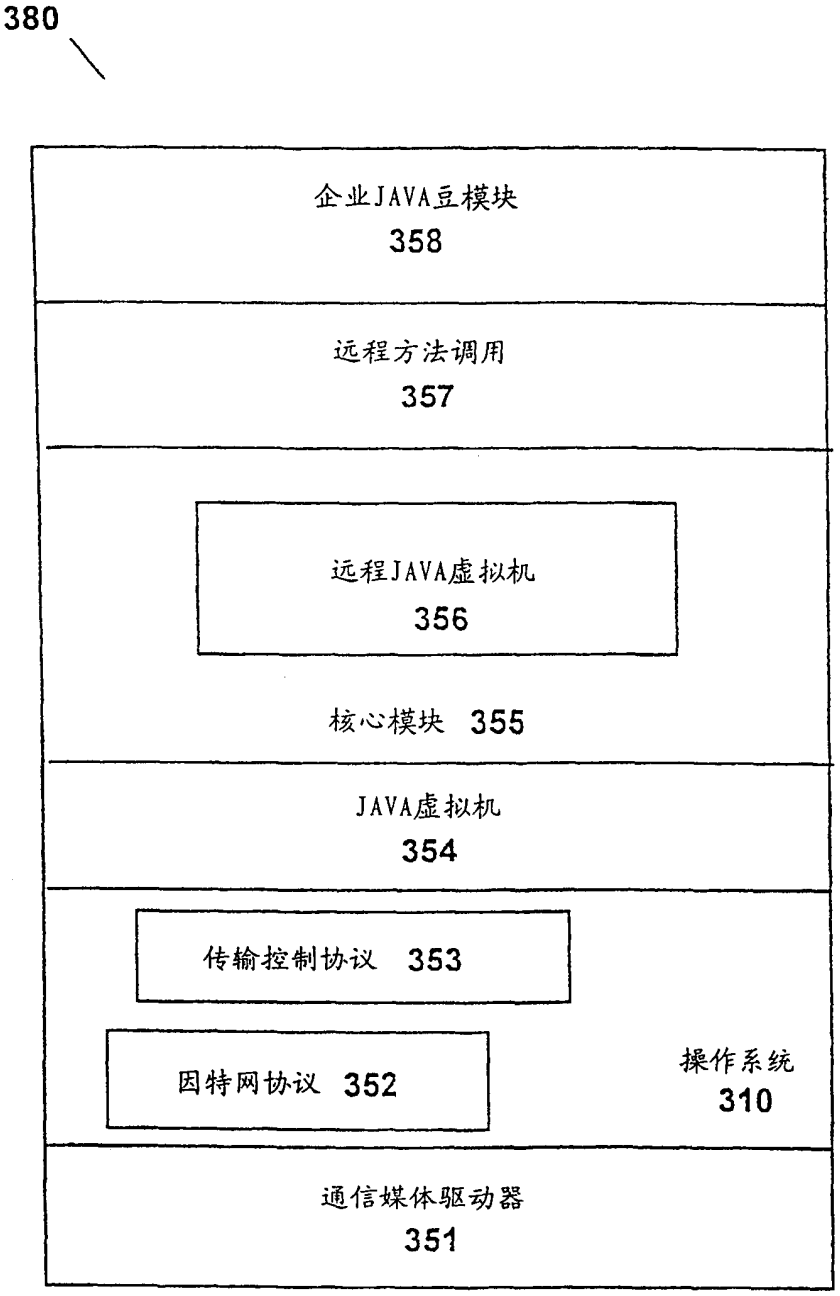

380
企业JAVA豆模块
358
远程方法调用
357
远程JAVA虚拟机
356
核心模块 355
JAVA虚拟机
354
传输控制协议 353
因特网协议 352
操作系统
310
通信媒体驱动器
351

图 3a

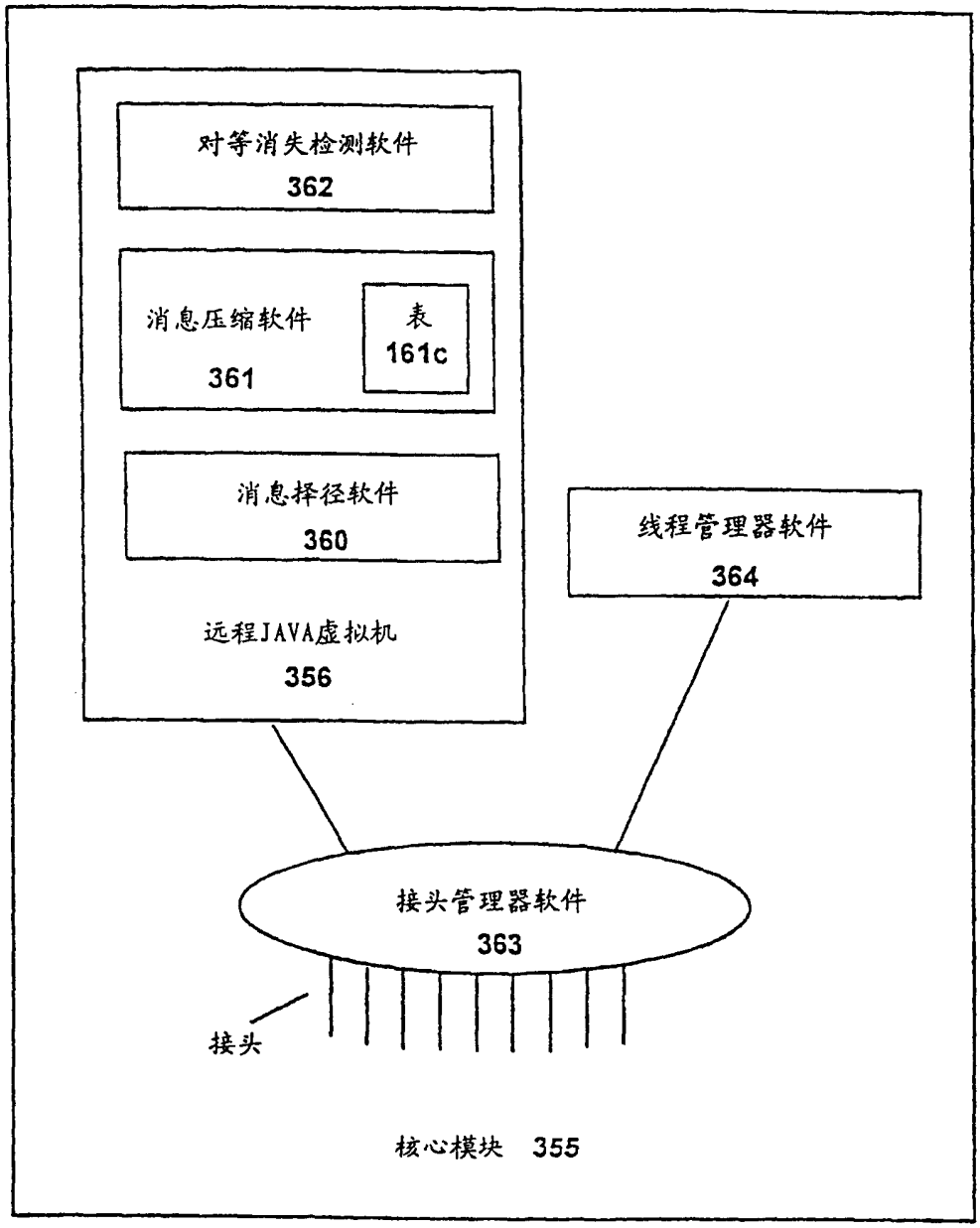

对等消失检测软件
362
消息压缩软件
361
表
161c
消息择径软件
360
远程JAVA虚拟机
356
线程管理器软件
364
接头管理器软件
363
接头
核心模块 355

图 3b

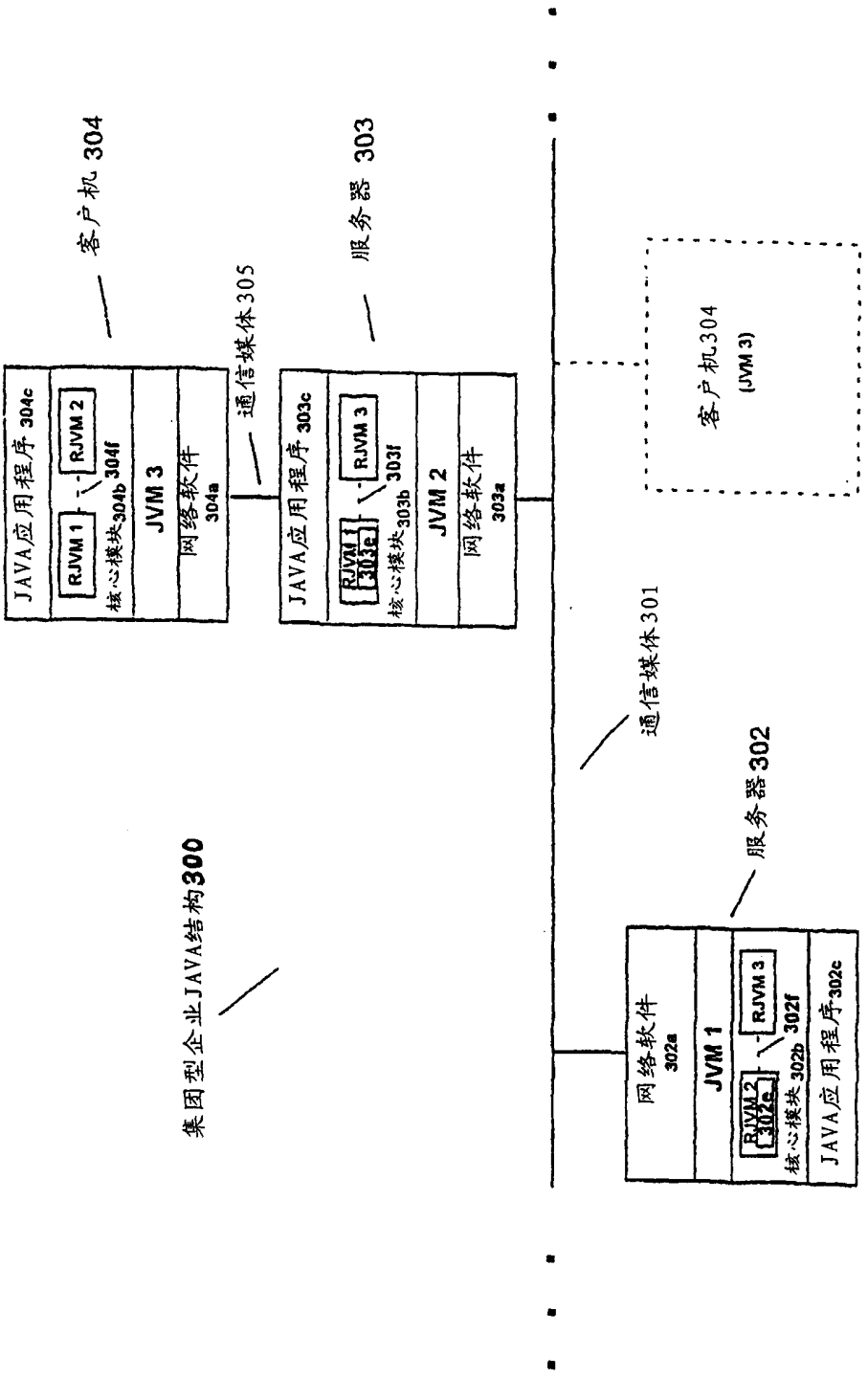

集团型企业JAVA结构300
客户机 304
服务器 303
通信媒体305
通信媒体301
服务器302
JAVA应用程序 304c
RJVM 1
RJVM 2
核心模块304b
304f
JVM 3
网络软件
304a
JAVA应用程序 303c
RJVM 1
303e
RJVM 3
核心模块303b
303f
JVM 2
网络软件
303a
客户机304
(JVM 3)
网络软件
302a
JVM 1
RJVM 2
302e
RJVM 3
核心模块302b
302f
JAVA应用程序302c

图 3c

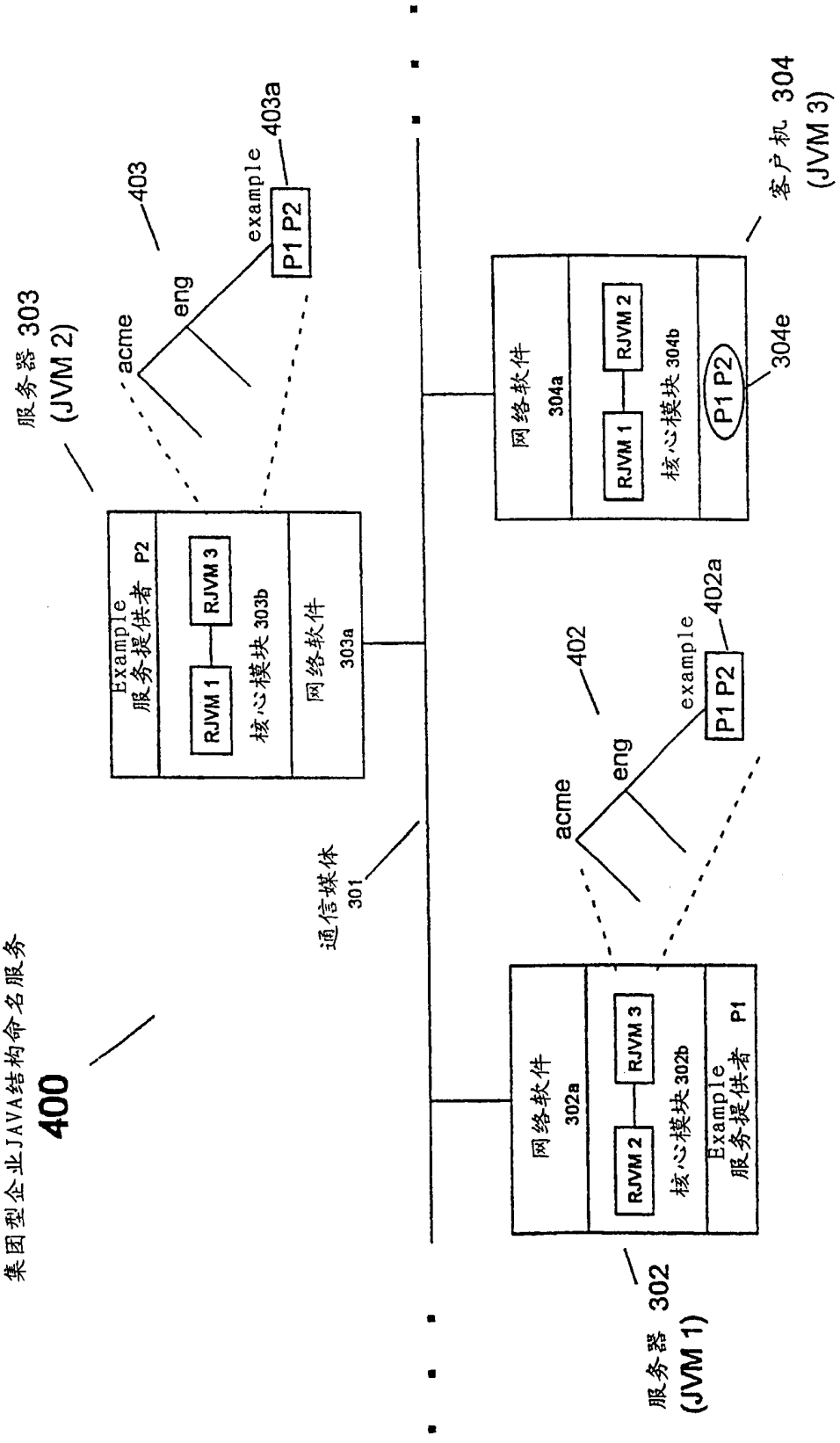

集团型企业JAVA结构命名服务
400
通信媒体
301
服务器 303
(JVM 2)
acme
eng
example
403
403a
P1 P2
Example
服务提供者 P2
RJVM 1
RJVM 3
核心模块 303b
网络软件
303a
客户机 304
(JVM 3)
网络软件
304a
RJVM 1
RJVM 2
核心模块 304b
P1 P2
304e
服务器 302
(JVM 1)
网络软件
302a
RJVM 2
RJVM 3
核心模块 302b
Example
服务提供者 P1
402
acme
eng
example
402a
P1 P2

图 4

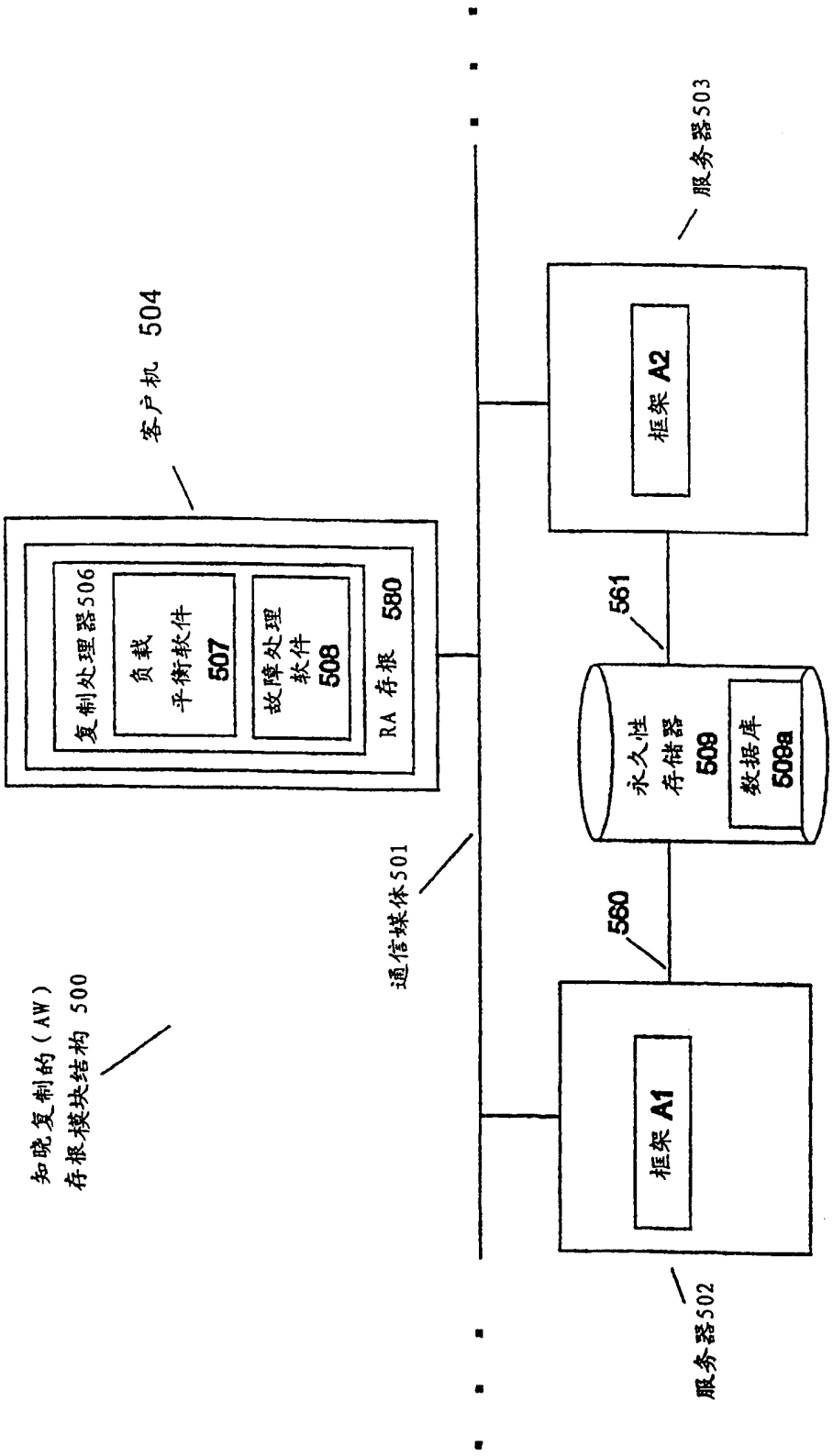

知晓复制的（AW）
存根模块结构 500
通信媒体501
客户机 504
复制处理器506
负载
平衡软件
507
故障处理
软件
508
RA 存根 580
服务器502
框架 A1
560
永久性
存储器
509
数据库
509a
561
框架 A2
服务器503

图 5a

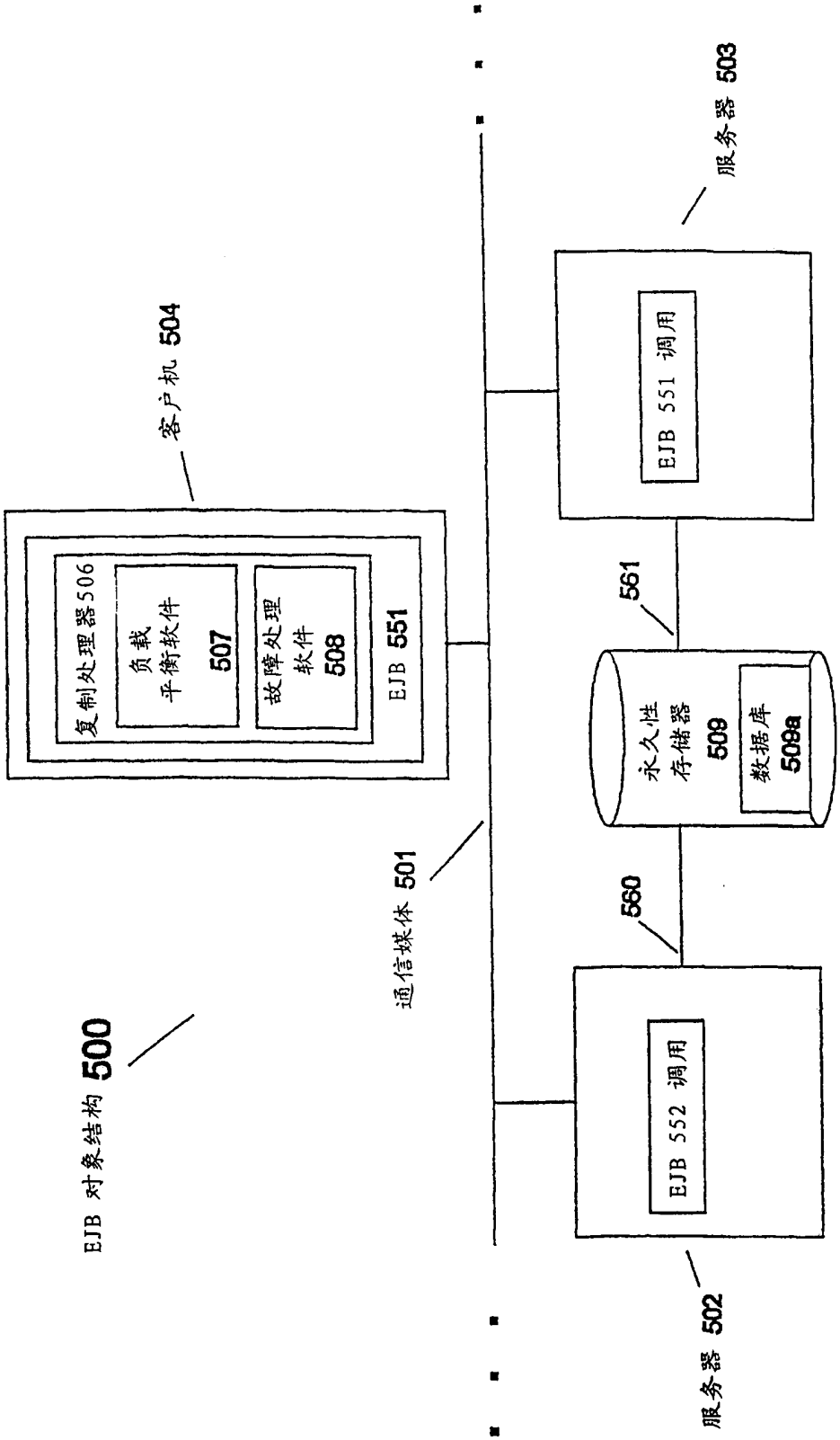

EJB 对象结构 500
通信媒体 501
服务器 502
服务器 503
客户机 504
复制处理器 506
负载平衡软件 507
故障处理软件 508
EJB 551
EJB 551 调用
561
永久性存储器 509
数据库 509a
560
EJB 552 调用

图 5b

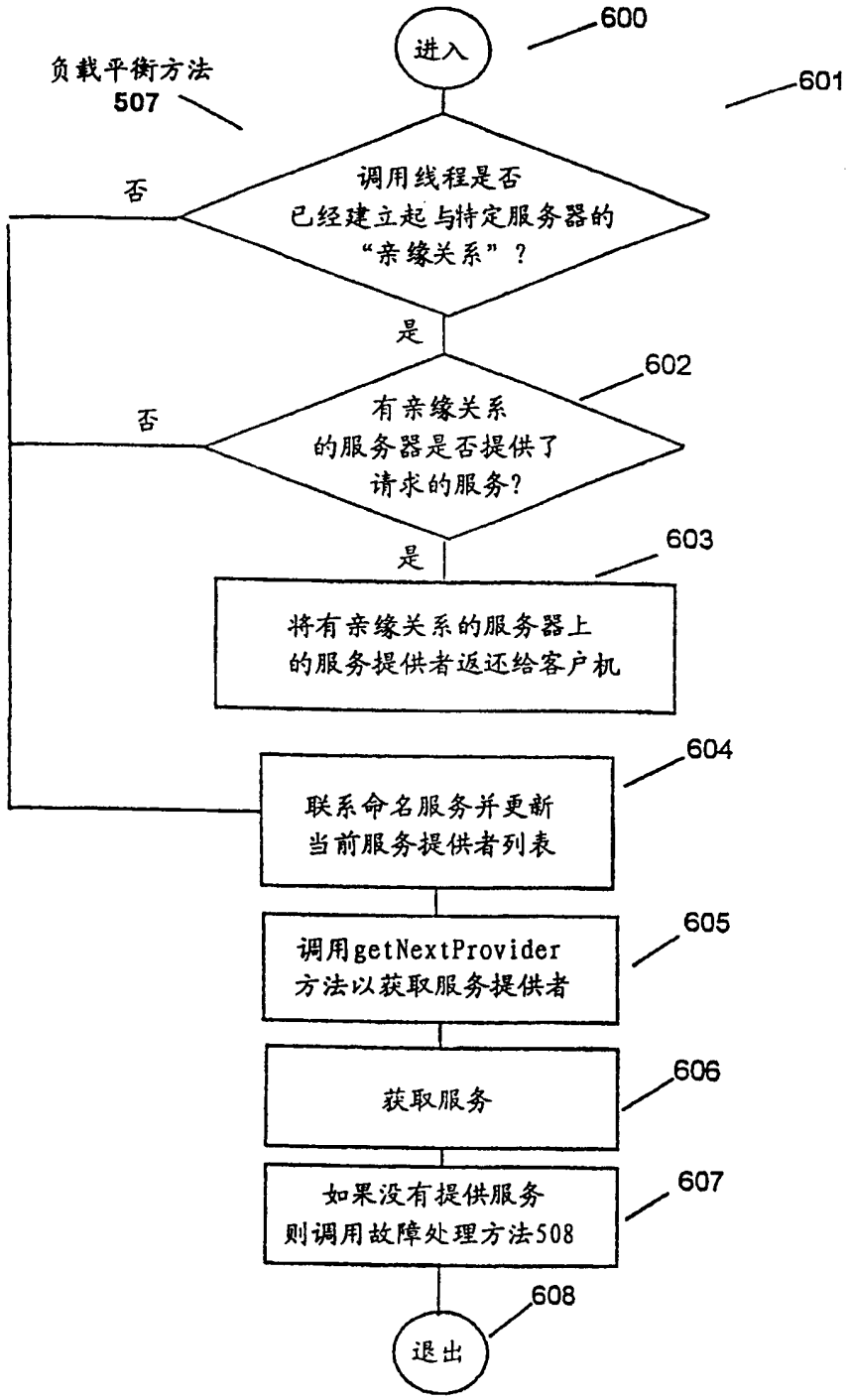

负载平衡方法
507
进入
600
601
调用线程是否
已经建立起与特定服务器的
“亲缘关系”？
否
是
602
有亲缘关系
的服务器是否提供了
请求的服务?
否
是
603
将有亲缘关系的服务器上
的服务提供者返还给客户机
604
联系命名服务并更新
当前服务提供者列表
605
调用getNextProvider
方法以获取服务提供者
606
获取服务
607
如果没有提供服务
则调用故障处理方法508
608
退出

图 6a

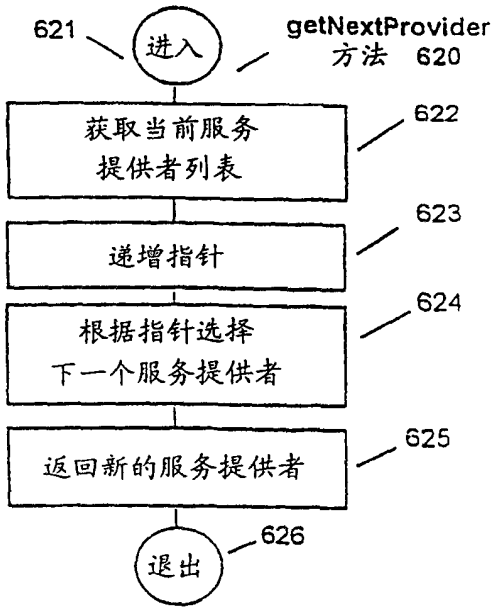


图 6b

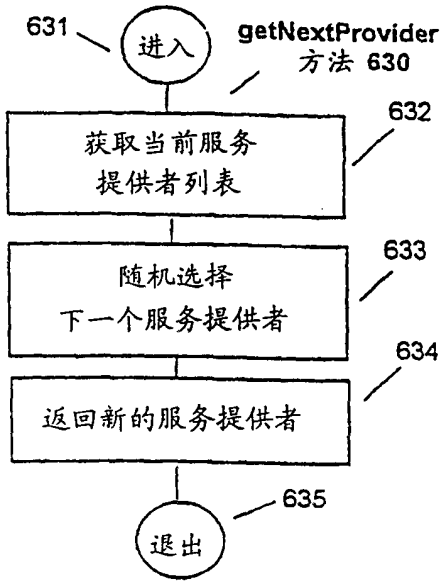


图 6c

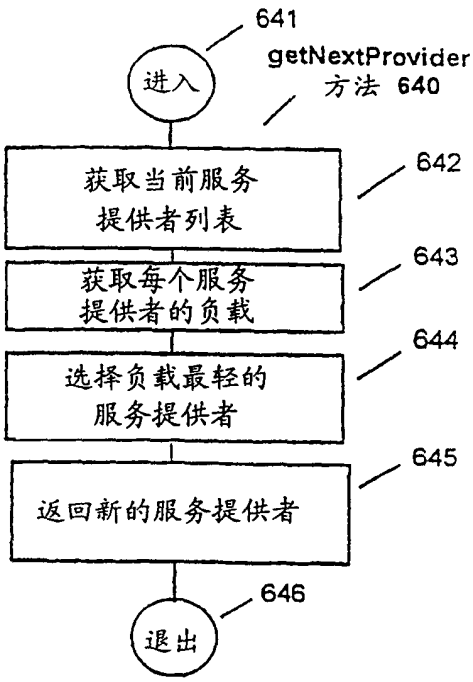


图 6d

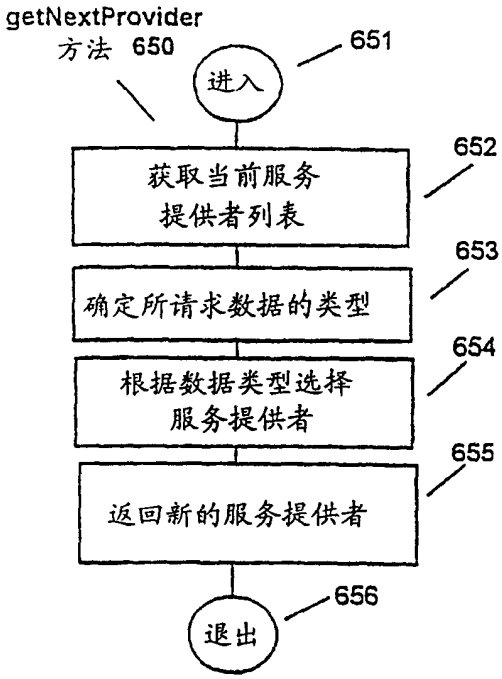


图 6e

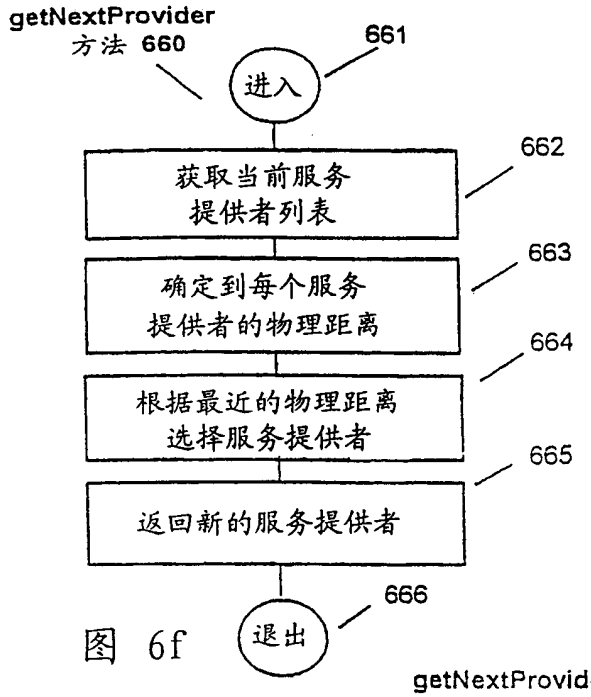

getNextProvider
方法 660
661
进入
662
获取当前服务
提供者列表
663
确定到每个服务
提供者的物理距离
664
根据最近的物理距离
选择服务提供者
665
返回新的服务提供者
666
退出

图 6f

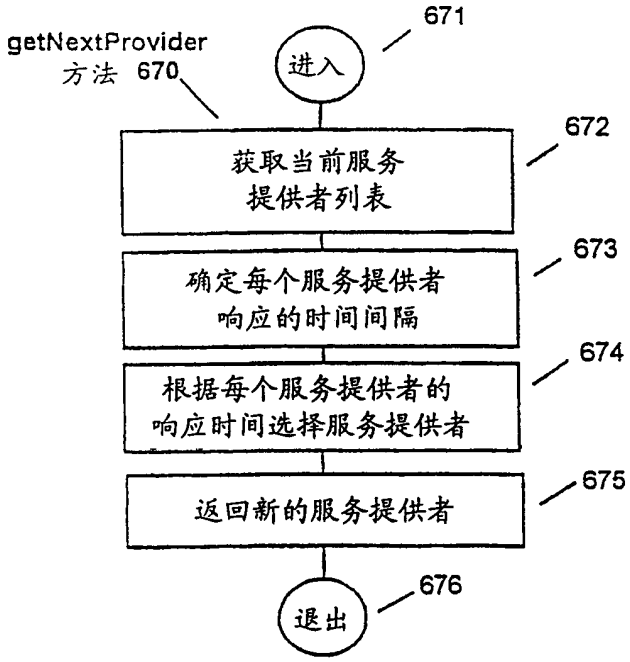

getNextProvider
方法 670
671
进入
672
获取当前服务
提供者列表
673
确定每个服务提供者
响应的时间间隔
674
根据每个服务提供者的
响应时间选择服务提供者
675
返回新的服务提供者
676
退出

图 6g

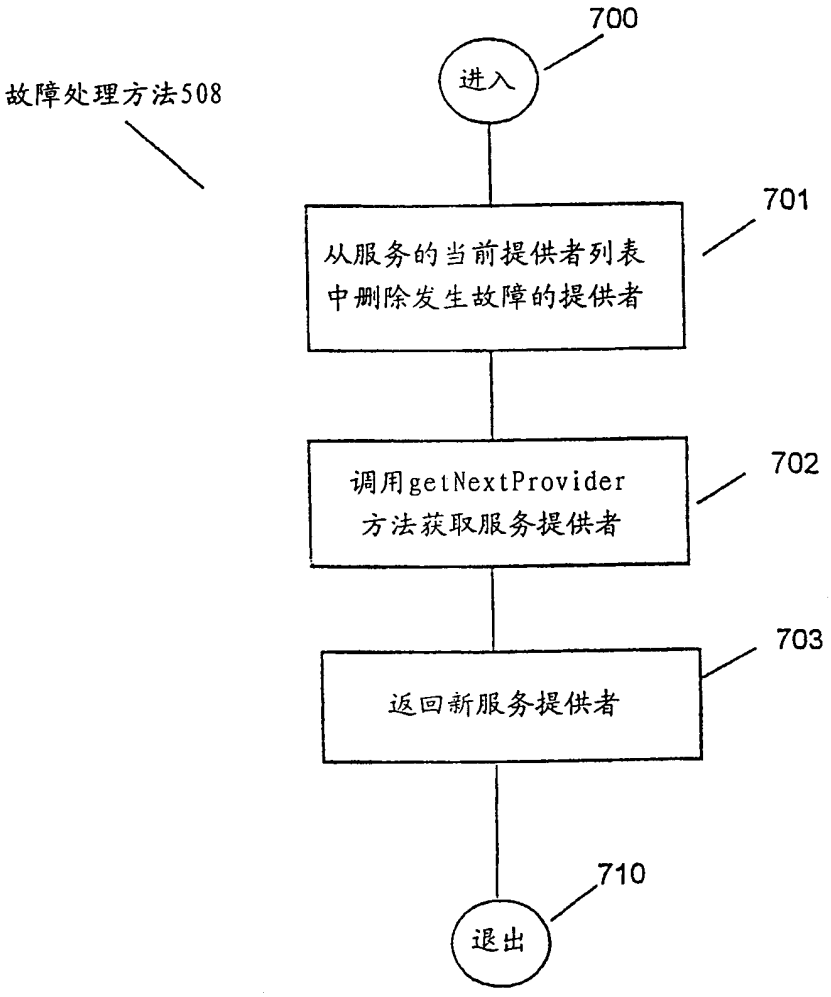

故障处理方法508
700
进入
701
从服务的当前提供者列表
中删除发生故障的提供者
702
调用getNextProvider
方法获取服务提供者
703
返回新服务提供者
710
退出

图 7

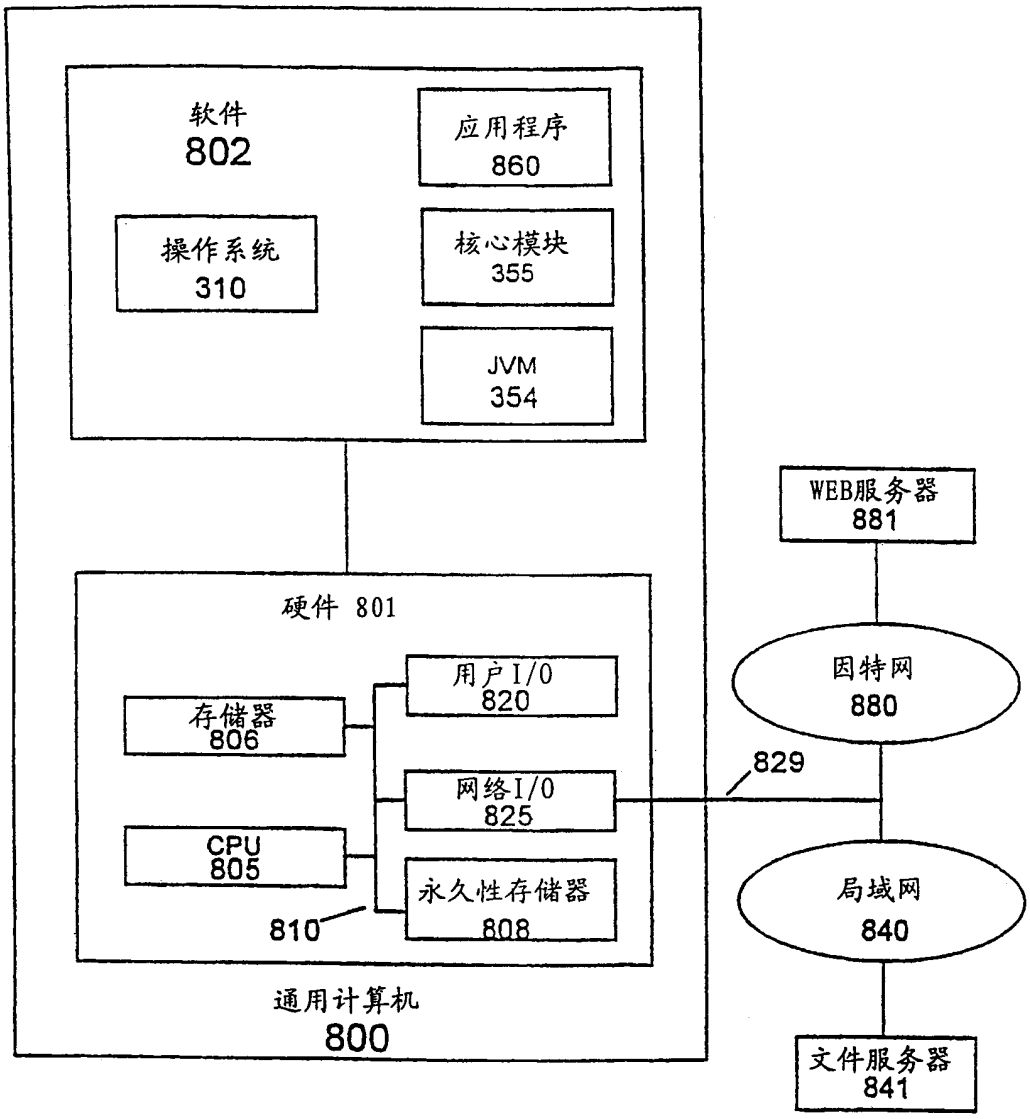

软件
802
应用程序
860
操作系统
310
核心模块
355
JVM
354
硬件 801
用户I/O
820
存储器
806
网络I/O
825
CPU
805
永久性存储器
808
810
通用计算机
800
WEB服务器
881
因特网
880
829
局域网
840
文件服务器
841

图 8