(19)中华人民共和国国家知识产权局

(12)发明专利

(10)授权公告号 CN 1989489 B
(45)授权公告日 2010.12.08

(21)申请号 200580024786.X
(22)申请日 2005.05.20
(30)优先权数据
10/851,813 2004.05.20 US
(85)PCT申请进入国家阶段日
2007.01.22
(86)PCT申请的申请数据
PCT/EP2005/005502 2005.05.20
(87)PCT申请的公布数据
WO2005/114405 EN 2005.12.01
(73)专利权人 SAP股份公司
地址 德国瓦尔多夫
(72)发明人 奥利弗·施米特 诺伯特·库克
埃德加·洛特 马丁·斯特拉斯伯格
阿尔诺·希尔根伯格
拉尔夫·施梅尔特 简·多瑟特
(74)专利代理机构 北京市柳沈律师事务所 11105
代理人 邵亚丽 李晓舒
(51)Int.Cl.
G06F 9/46(2006.01)
G06F 9/445(2006.01)
(56)对比文件
US 6253256 B1,2001.06.26, 对比文件3说明书第2栏第61行-第3栏第12行，第5栏第62行-第7栏第43行、附图2,4.
审查员 张露薇

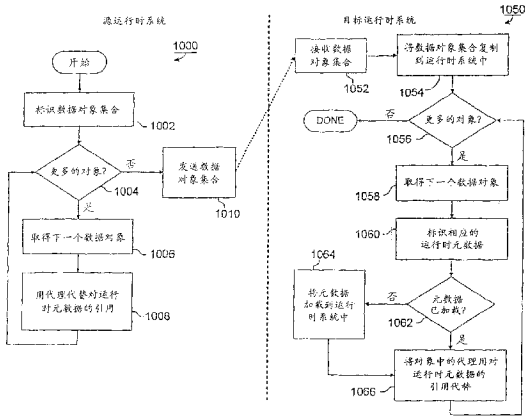
权利要求书 2 页 说明书 30 页 附图 16 页

(54)发明名称

数据处理方法和装置

(57)摘要

用于在运行时系统中共享对象的方法和装置，包括计算机系统和程序产品。对象的集合被标识，每一个对象均具有对关于第一运行时系统中的对象的运行时元数据的引用。对每一个对象中的运行时元数据的引用被用代理代替，并且所述对象的集合被发送到第二运行时系统，在第二运行时系统中，每一个对象中的代理被用对关于第二运行时系统中的对象的运行时元数据的引用代替。在某些实施方案中，如果运行时元数据并非已经可用，则可以将其安装在第二运行时系统中。所述对象的集合可以包括可在置于不同的物理机器上的运行时系统（包括虚拟机）之间共享的对象的共享闭包。在多个机器上的运行时系统之间共享对象可以在用户会话之间以可扩展的方式提供隔离。



CN 1989489 B

1. 一种数据处理方法，包含：

在第一运行时系统中标识要被与第二运行时系统共享或者要在第二运行时系统中使用的数据对象的集合，所述数据对象的集合中的每一个数据对象均包含对关于第一运行时系统中的数据对象的运行时元数据的引用；

对于所述数据对象的集合中的每一个数据对象，用代理代替所述数据对象中的对运行时元数据的引用；和

将所述数据对象的集合传送到第二运行时系统。

2. 如权利要求 1 所述的方法，其中，对于所述数据对象的集合中的每一个数据对象，对关于所述数据对象的运行时元数据的引用包含对所述数据对象是其实例的类的运行时表示的引用。

3. 如权利要求 1 或 2 所述的方法，其中，所述第一运行时系统包含第一虚拟机，并且所述第二运行时系统包含第二虚拟机。

4. 如权利要求 3 所述的方法，其中，所述第一虚拟机和所述第二虚拟机是 Java 虚拟机或公共语言运行时虚拟机。

5. 如权利要求 1 所述的方法，其中，所述第一运行时系统被置于第一物理机器上，并且所述第二运行时系统被置于不同于第一物理机器的第二物理机器上。

6. 如权利要求 1 所述的方法，其中，所述数据对象的集合由第一数据对象和一个或更多个被引用的数据对象的传递闭包组成，标识所述数据对象的集合包含标识一个或更多个被引用的数据对象的传递闭包，每一个被引用的数据对象均被所述第一数据对象引用。

7. 一种数据处理方法，包含：

从第一运行时系统接收数据对象的集合，该数据对象的集合中的每一个数据对象均包含代理；

将所述数据对象的集合复制到第二运行时系统中；和

对于所述数据对象的集合中的每一个数据对象：

标识关于所述数据对象的运行时元数据；和

用对关于所述数据对象的运行时元数据的引用代替所述数据对象中的代理。

8. 如权利要求 7 所述的方法，还包括：

对于所述数据对象的集合中的每一个数据对象：

确定在所述第二运行时系统中是否可获得关于所述数据对象的运行时元数据；和

如果在所述第二运行时系统中不可获得关于所述数据对象的运行时元数据，则将关于所述数据对象的运行时元数据加载在所述第二运行时系统中。

9. 如权利要求 7 或 8 所述的方法，其中，对于所述数据对象的集合中的每一个数据对象，对关于所述数据对象的运行时元数据的引用包含对所述数据对象是其实例的类的运行时表示的引用。

10. 如权利要求 9 所述的方法，还包括：

对于所述数据对象的集合中的每一个数据对象：

确定在所述第二运行时系统中是否可获得所述数据对象是其实例的类的运行时表示；和

如果在所述第二运行时系统中不可获得所述数据对象是其实例的类的运行时表示，则

将所述数据对象是其实例的类的运行时表示加载在所述第二运行时系统中。

11. 如权利要求 7 所述的方法，其中，所述第一运行时系统包含第一虚拟机，并且所述第二运行时系统包含第二虚拟机。

12. 如权利要求 11 所述的方法，其中，所述第一虚拟机和所述第二虚拟机是 Java 虚拟机或公共语言运行时虚拟机。

13. 如权利要求 7 所述的方法，其中，所述第一运行时系统被置于第一物理机器上，并且所述第二运行时系统被置于不同于第一物理机器的第二物理机器上。

14. 如权利要求 7 所述的方法，其中，所述数据对象的集合由第一数据对象和一个或更多个被引用的数据对象的传递闭包组成，每一个被引用的数据对象均被所述第一数据对象引用。

15. 如权利要求 7 所述的方法，其中，标识所述运行时元数据和代替所述数据对象的集合中每一个数据对象中的所述代理的操作在接收到所述数据对象的集合之后立刻发生。

16. 如权利要求 7 所述的方法，其中，对于所述数据对象的集合中的每一个数据对象，标识所述运行时元数据和代替所述数据对象中的代理的操作在所述数据对象在所述第二运行时系统中被首次访问时发生。

17. 一种数据处理装置，包含 ：

用于在第一运行时系统中标识要被与第二运行时系统共享或者要在第二运行时系统中使用的数据对象的集合的装置，该数据对象的集合中的每一个数据对象均包含对关于第一运行时系统中的数据对象的运行时元数据的引用 ；

用于用代理代替所述数据对象的集合中的每一个数据对象中的对运行时元数据的引用的装置 ；和

用于将所述数据对象的集合传送到第二运行时系统的装置。

18. 一种数据处理装置，包含 ：

用于从第一运行时系统接收数据对象的集合的装置，该数据对象的集合中的每一个数据对象均包含代理 ；

用于将所述数据对象的集合复制到第二运行时系统中的装置 ；和

用于对于所述数据对象的集合中的每一个数据对象 ：标识关于所述数据对象的运行时元数据、和用对关于所述数据对象的运行时元数据的引用代替所述数据对象中的代理的装置。

数据处理方法和装置

技术领域

[0001] 本申请涉及数据处理。

背景技术

[0002] 某些服务器，例如企业服务器或者其他的大型服务器，一般可以被刻画为请求处理引擎，因为它们处理大量通常较小的用户请求，这些用户请求属于用户会话。处理请求通常涉及在服务器上执行的运行时系统（例如 Java 虚拟机）中运行用户代码（例如 Java 服务器小程序 (servlet) 或企业 Java Bean）。在这种服务器中的可扩展性 (scalability) 传统上是通过使用线程来实现的——例如使用多线程虚拟机 (virtual machine，VM) 来处理相应于许多个用户会话的请求。但是系统鲁棒性要求用户会话之间有很强的隔离，当大量用户会话运行于单个 VM 中时，这可能很难实现。

[0003] 操作系统能够为进程提供近乎完美的隔离。在某些操作系统中，崩溃的进程 (crashed process) 将不会影响其他的进程，并且将不泄漏或遗留被分配的资源。在概念上，有可能通过为每一个用户会话分配一个操作系统 (OS) 进程并在所分配的进程内为该用户会话运行 VM 来隔离用户会话，并从而提高服务器的鲁棒性。但是，在某些情形下（例如在存在大量用户会话的情形下），由于在相应大量的进程之间切换将导致的 OS 调度开销以及由于这种方法将消耗的资源所致，这种方法可能并不实用。OS 进程一般未被设计成像用户会话那样细致地对实体建模。

发明内容

[0004] 本发明提供了方法和装置，包括计算机程序产品，它们实施了用于共享数据并在用户会话之间提供隔离的技术。

[0005] 在一个总体方面，这些技术的特征是一种计算机程序产品，所述计算机程序产品可运行，以便使得数据处理装置：接收第一数据对象的标识，该第一数据对象是第一运行时类的实例；确定所述第一运行时类是否可共享；以 及，确定所述第一数据对象是否引用一个或更多个被引用的数据对象。如果所述第一数据对象确实引用了一个或更多个被引用的数据对象，则所述计算机程序产品还可运行，以便使得所述数据处理装置遍历所述一个或更多个被引用的数据对象，并且针对每一个被遍历的数据对象，确定所述被遍历的数据对象是其实例的运行时类是否可共享。

[0006] 有益的实施方案可能包括下列特征中的一个或更多个。遍历所述一个或更多个被引用的数据对象可以包括递归地遍历所述一个或更多个被引用的数据对象的传递闭包 (transitive closure) 中的每一个数据对象。

[0007] 如果所述第一运行时类可共享，并且每一个被遍历的数据对象的运行时类均可共享，则所述第一数据对象和每一个被遍历的数据对象均能够被分组到一个对象组中，并且所述对象组可以被复制到共享存储器区域中。

[0008] 如果第一运行时类不可共享，或者如果至少一个被遍历的数据对象的运行时类不

可共享，则可以产生否定状态指示。

[0009] 确定所述第一运行时类是否可共享可以包括确定所述第一运行时类先前是否已被声明为可共享。

[0010] 所述第一运行时类可以从一个或更多个基类被派生，并且可以包括一个或更多个域。确定所述第一运行时类是否可共享可以包括：确定所述第一运行时类是否实施了串行化接口；确定在所述第一运行时类的对象实例的串行化或者解串行化期间是否执行了定制代码(custom code)；确定是否所有基类都是可串行化的；确定是否所有域都被串行化；以及，确定第一运行时类的对象实例是否影响垃圾回收。

[0011] 所述第一运行时类可以是Java类，并且串行化接口可以是java.io.Serializable。

[0012] 确定是否执行了定制代码可以包括：确定第一运行时类是否包括预先确定的方法集合中的一种方法。预先确定的方法集合可以包括：readObject( )、wirteObject( )、readExternal( )、writeExternal( )、readResolve( ) 和 writeReplace( ) 方法。

[0013] 确定是否所有基类都可串行化可以包括：确定基类中的每一个类是否均实施了串行化接口，并且，如果基类中的类未实施串行化接口，则确定该类是否包括平凡缺省构造器(trivial default constructor)。

[0014] 确定是否所有域都被串行化可以包括：确定是否任一个域均为瞬态域。确定所有域是否都被串行化还可以包括确定是否任一个域均是serialPersisitentFields域。

[0015] 确定第一运行时类的对象实例是否影响垃圾回收可以包括：确定第一运行时类是否包含平凡终结器(trivial finalizer)。在第一运行时类是Java类的情况下，确定第一运行时类的对象实例是否影响垃圾回收还可以包括确定第一运行时类是否被从java.lang.ref.Reference类派生。

[0016] 确定第一运行时类是否可共享还可以包括：确定第一运行时类的运行时表示是否可共享，和确定第一运行时类的类加载器是否可共享。

[0017] 确定第一运行时类的运行时表示是否可共享可以包括：确定所述运行时表示是否被存储在第一预先确定的位置。确定第一运行时类的类加载器是否可共享可以包括确定所述类加载器是否被存储在第二预先确定的位置。

[0018] 在另一个方面，这些技术的特征是一种计算机程序产品，所述计算机程序产品可运行，以便使得数据处理装置：接收第一运行时系统中的第一数据对象的标识，所述第一数据对象引用零个或更多个被引用的数据对象；标识由第一数据对象和被引用的数据对象的传递闭包组成的数据对象共享闭包；以及，确定数据对象的所述共享闭包是否可在第二运行时系统中使用。

[0019] 有益的实施方案可以包括下列特征中的一个或更多个。第一和第二运行时系统可以是虚拟机，包括Java虚拟机或者公共语言运行时虚拟机。数据对象的共享闭包可以包括用户上下文信息。

[0020] 确定共享闭包是否可在第二运行时系统中使用可以包括：确定数据对象的共享闭包中的每一个数据对象是否均是无需执行定制代码就可串行化的。

[0021] 确定共享闭包是否可在第二运行时系统中使用可以包括确定所述共享闭包中的每一个数据对象的运行时类是否是可共享的。

[0022] 每一个数据对象均是其实例的运行时类可以被从一个或更多个基类派生，并且可以具有一个或更多个域，并且确定运行时类是否可共享可以包括：确定所述运行时类是否实施了串行化接口；确定在所述运行时类的对象实例的串行化或者解串行化期间是否执行了定制代码；确定是否所有基类都是可串行化的；确定是否所有域都被串行化；以及，确定所述运行时类的对象实例是否影响垃圾回收。

[0023] 如果数据对象的共享闭包不可在第二运行时系统中使用，则可以产生否定状态指示。

[0024] 数据对象的共享闭包可以被复制到共享存储器区域。所述计算机程序产品还可运行，以便使得数据处理装置确定在共享存储器区域中是否存在数据对象的共享闭包的先前版本，并将版本号和数据对象的共享闭包相关联。

[0025] 在又一个方面，这些技术的特征是一种计算机程序产品，所述计算机程序产品可运行，以便使得数据处理装置：接收标识符、标识与所述标识符相关联的共享闭包、以及将所述共享闭包与运行时系统相关联。所述共享闭包被置于共享存储器区域中，并由第一数据对象和被所述第一数据对象引用的数据对象的传递闭包组成。

[0026] 有益的实施方案可以包括下列特征中的一个或更多个。所述运行时系统可以是Java虚拟机或者公共语言运行时虚拟机。标识共享闭包可以包括标识所述共享闭包的当前版本。

[0027] 通过将所述共享闭包映射或者复制到与所述运行时系统相关联的地址空间，可以将所述共享闭包与所述运行时系统相关联。在所述共享闭包被映射到和所述运行时系统相关联的地址空间中的情况下，将所述共享闭包与所述运行时系统相关联还可以包括防止对所述共享闭包的写访问，或者，在检测到对所述共享闭包中的数据对象的第一次写访问时，将所述共享闭包复制到与所述运行时系统相关联的地址空间中。

[0028] 共享闭包可以被标记为已删除。将共享闭包标记为已删除可以包括防止将所述共享闭包与额外的运行时系统相关联。

[0029] 在又一个方面，这些技术的特征是一种计算机程序产品，所述计算机程序产品可运行，以便使得数据处理装置：标识数据对象的集合，所述数据对象的集合中的每一个数据对象均具有对关于第一运行时系统中的数据对象的运行时元数据的引用。所述计算机程序产品还可运行，以便使得所述数据处理装置：在每一个数据对象中用代理来代替对运行时数据的引用；以及，将所述数据对象集合发送到第二运行时系统。

[0030] 有益的实施方案可以包括下列特征中的一个或更多个。对于所述数据对象集合中的每一个数据对象，对关于数据对象的运行时元数据的引用可以包括对所述数据对象是其实例的类的运行时表示的引用。

[0031] 所述第一运行时系统可以包括第一虚拟机，并且所述第二运行时系统可以包括第二虚拟机。第一和第二虚拟机可以是 Java 虚拟机或者公共语言运行时虚拟机。第一运行时系统可以被置于第一物理机器上，并且第二运行时系 统可以被置于不同的第二物理机器上。

[0032] 标识数据对象的集合可以包括标识一个或更多个被引用数据对象的传递闭包，其中每一个被引用的数据对象均是被第一数据对象引用的对象。

[0033] 在又一个方面，这些技术的特征是一种计算机程序产品，所述计算机程序产品可

运行，以便使得数据处理装置：从第一运行时系统接收数据对象的集合，其中所述数据对象的集合中的每一个数据对象均包含代理；将所述数据对象的集合复制到第二运行时系统中；以及，对于所述数据对象的集合中的每一个数据对象，标识关于所述数据对象的运行时元数据并利用对关于所述数据对象的运行时元数据的引用来代替所述数据对象中的代理。

[0034] 有益的实施方案可以包括下列特征中的一个或更多个。所述计算机程序产品还可以运行，以便：确定在第二运行时系统中是否可获得关于每一个数据对象的运行时元数据；以及，如果在第二运行时系统中不可获得运行时元数据，则将关于每一个数据对象的运行时元数据安装在第二运行时系统中。

[0035] 对于每一个数据对象，对关于数据对象的运行时元数据的引用可以包括对数据对象是其实例的类的运行时表示的引用。所述计算机程序产品还可运行，以便：确定在第二运行时系统中是否可获得每一个对象实例的类的运行时表示；以及，如果在第二运行时系统中不可获得所述运行时表示，则将每一个对象实例的类的运行时表示安装在第二运行时系统中。

[0036] 第一运行时系统可以包括第一虚拟机，并且所述第二运行时系统可以包括第二虚拟机。第一和第二虚拟机可以是 Java 虚拟机或者公共语言运行时虚拟机。第一运行时系统可以被置于第一物理机器上，并且第二运行时系统可以被置于不同的第二物理机器上。

[0037] 所述数据对象的集合由第一数据对象和一个或更多个被引用的数据对象的传递闭包组成，其中，每一个被引用的数据对象均是被第一数据对象引用的对象。

[0038] 标识运行时元数据以及代替数据对象的集合中的每一个数据对象中的代理的操作大致可以在接收到数据对象集合之后立刻发生。或者，当在第二运行时系统中访问每一个数据对象时，可以发生那些操作。

[0039] 在又一个方面，这些技术的特征是一种计算机程序产品，所述计算机程序产品可运行，以便使得数据处理装置：为用户会话初始化运行时系统；在共享存储器区域中创建数据对象的共享闭包；接收相应于用户会话的请求； 从操作系统进程的集合中选择第一进程；将所述运行时系统绑定到第一进程；以及，将数据对象的共享闭包与所述运行时系统相关联。所述数据对象的共享闭包由第一数据对象和一个或更多个被引用的数据对象的传递闭包组成，每一个被引用的数据对象均被第一数据对象引用。

[0040] 有益的实施方案可以包括下列特征中的一个或更多个。运行时系统可以包括虚拟机。所述虚拟机可以是 Java 虚拟机或者公共语言运行时虚拟机。

[0041] 数据对象的共享闭包可以包括相应于用户会话的用户上下文信息。

[0042] 将共享闭包与运行时系统相关联可以包括将所述共享闭包绑定到第一进程。将共享闭包绑定到第一进程可以包括将所述共享闭包映射或者复制到第一进程的地址空间。如果所述共享闭包被映射到第一进程的地址空间中，则绑定所述共享闭包还可以包括防止对所述共享闭包的写访问，或者，在检测到对共享闭包中的数据对象之一的第一次写访问时，将所述共享闭包复制到第一进程的地址空间中。

[0043] 所述计算机程序产品还可以运行，以便使得所述数据处理装置：为第二用户会话初始化第二运行时系统；接收相应于第二用户会话的第二请求；从操作系统进程的集合中选择第二进程；将所述第二运行时系统绑定到第二进程；以及，将数据对象的共享闭包与所述第二运行时系统相关联。

[0044] 在又一个方面中，这些技术的特征是一种计算机程序产品，所述计算机程序产品可运行，以便使得数据处理装置：在共享存储器区域中存储相应于用户会话的用户上下文；接收相应于用户会话的请求；从操作系统进程的集合中选择具有地址空间的进程；从运行时系统的集合中选择运行时系统；将所述运行时系统绑定到所述进程；以及，将所述用户上下文与所述运行时系统相关联。

[0045] 有益的实施方案可以包括下列特征中的一个或更多个。运行时系统的集合中的每一个运行时系统均可以包括虚拟机。虚拟机可以是 Java 虚拟机或者公共语言运行时虚拟机。

[0046] 用户上下文可以包括由第一数据对象和被第一数据对象引用的数据对象的传递闭包组成的共享闭包。

[0047] 运行时系统可以被存储在共享存储器区域中，并且将所述运行时系统绑定到所述进程可以包括将相应于所述运行时系统的共享存储器区域部分映射到所述进程的地址空间中。

[0048] 将用户上下文与所述运行时系统相关联可以包括将用户上下文绑定到所述进程。将所述用户上下文绑定到所述进程可以包括将用户上下文映射或者复制到所述进程的地址空间中。如果用户上下文被映射到所述进程地址空间中，则绑定所述用户上下文还可以包括在检测到对所述用户上下文的第一次写访问时将所述用户上下文复制到所述进程的地址空间中。

[0049] 所述计算机程序产品还可以运行，以便使得数据处理装置：将所述用户上下文从所述进程解除绑定。将所述用户上下文从所述进程解除绑定可以包括将所述用户上下文复制到共享存储器区域。将所述用户上下文从所述进程解除绑定还可以包括：确定在共享存储器区域中是否存在先前版本的所述用户上下文；以及，如果先前版本的所述用户上下文不存在，则在共享存储器区域中创建新版本的所述用户上下文。

[0050] 在检测到所述进程被阻塞时，运行时系统和用户上下文都可以被从所述进程解除绑定。检测所述进程被阻塞可以包括检测所述进程正在等待输入 / 输出 (I/O) 事件完成。在检测到 I/O 事件已经完成时，可以从操作系统进程的集合中选择可用的进程，并且可以将运行时系统和用户上下文绑定到所述可用的进程。

[0051] 所述计算机程序产品还可以运行，以便使得所述数据处理装置：将所述运行时系统从所述进程解除绑定。

[0052] 操作系统进程的集合可以被跨越两个或更多个物理机器分布，并且所述进程可以在来自所述两个或更多个物理机器的第一机器上执行。用户上下文可以包括具有代理的第一数据对象。将用户上下文与运行时系统相关联可以包括用对关于第一数据对象的运行时元数据的引用代替所述代理，所述运行时元数据被存储在第一机器上。

[0053] 在又一个方面，这些技术的特征是一种计算机服务器，所述计算机服务器具有进程集合、运行时系统的集合、用于存储多个用户上下文的共享存储器区域，以及分派器 (dispatcher) 部件。所述分派器部件可运行，以便：接收相应于所述多个用户上下文中的一个用户上下文的请求；从所述进程的集合中选择可用的进程；从所述运行时系统的集合中选择可用的运行时系统；以及，将所述用户上下文和可用的运行时系统的标识传送到所述可用的进程，用于处理所述请求。

[0054] 有益的实施方案可以包括下列特征中的一个或更多个。所述运行时系统 的集合中的每一个运行时系统均可以包括虚拟机。所述虚拟机可以是 Java 虚拟机或者公共语言运行时虚拟机。可以将所述进程集合中的进程数量设置成小于或者等于所述运行时系统中的运行时系统数量。

[0055] 在又一个方面中，这些技术的特征是一种计算机程序产品，所述计算机程序产品可运行，以便使得数据处理装置：在进程的集合中的每一个进程中初始化运行时系统；在共享存储器区域中存储相应于用户会话的用户上下文；接收相应于所述用户会话的请求；从所述进程的集合中选择进程；以及，将所述用户上下文和所述被选择的进程中的所述运行时系统相关联以便处理所述请求。

[0056] 有益的实施方案可以包括下列特征中的一个或更多个。将所述用户上下文和所述被选择的进程中的运行时系统相关联可以包括将所述用户上下文绑定到所述被选择的进程。每一个进程中的运行时系统可以包括虚拟机。所述虚拟机可以是 Java 虚拟机或者公共语言运行时虚拟机。

[0057] 可以实施这里描述的技术来实现一个或更多个下列优点。一般可以使用这些技术来分解一个运行时系统中的复杂数据结构，并以另一个运行时系统的本机格式 (native format) 将其重装。更具体地，可以使用这些技术来在运行时系统（例如 VM) 之间共享对象。可共享类和可共享对象实例可以被自动标识。未被自动标识为可共享的类可以被手工分析，以便确定它们是否仍是可共享的。可共享对象可以被分组成称为共享闭包的集合，所述共享闭包可以和多个运行时系统相关联。可以共享多个类型的对象（例如包括构成用户上下文的对象），并且可以创建和共享对象的多个版本。可以通过多个机制访问被共享的对象，包括将所述被共享的对象从共享存储器区域映射或复制到和一个或更多个运行时系统相关联的地址空间中。可以限制对共享对象的访问，从而防止运行时系统以使得共享对象中的数据无效或无法在其他运行时系统中使用的方式将其覆盖。还可以将共享对象标记为已删除或垃圾回收 (garbage collected)。

[0058] 可以在不同物理机器上（例如在集群架构 (cluster architectures) 中）的运行时系统之间共享对象。对源运行时系统中的运行时元数据的引用可以被用对目标运行时系统中的运行时元数据的引用标识和代替。运行时元数据可以被加载到目标系统中，并且可以主动或被动地（例如在被要求时）插入对这些运行时元数据的引用。共享对象可以降低资源消耗（例如可以使用更少的 存储器，因为对象可以由多个运行时系统共享）以及时间消耗（例如可以使用更少的时间，因为不必在多个运行时系统中创建或初始化被共享的对象）。

[0059] 还可以使用这里描述的技术在用户会话之间以可扩展的方式提供隔离，从而使得服务器能够稳健地处理相应于大量用户会话的请求。通过给每一个用户会话分配一个 VM 并且将 VM 绑定到 OS 工作进程以便处理请求，可以将用户会话隔离。工作进程的数量可以被调整，以便提高吞吐量（例如可以将工作进程的数量设置成和被分配给服务器的处理器的数量相匹配）。VM 可以共享包括用户上下文的对象，以便降低资源和时间消耗。

[0060] 或者，不是给每一个用户会话分配一个 VM，而是也可以在用户会话之间共享 VM。因此，可以用固定数量的 VM 实施服务器。也可以用可变但是有限（例如最大）数量的 VM 实施服务器。在用户会话之间共享 VM 可以降低每一个用户会话的开销，特别是在用户上下文

较小的环境中更是如此。尽管共享 VM，但是通过只将一个 VM 和一个用户上下文连接到每一个工作进程，并且使用被连接的 VM 来处理相应于被连接的用户上下文的用户会话的请求，仍能使用户会话保持隔离。吞吐量仍旧可以被优化，因为只要工作进程成为可用，用户上下文和 VM 就都能够被绑定到进程并在所述进程中执行。

[0061] 为了以如上所述的隔离方式共享 VM，可以将用户上下文和 VM 解除关联。用户上下文可以被多个 VM 访问或共享，例如，通过将用户上下文存储在可被服务器中的所有 VM 访问的共享存储器区域中。通过将 VM 绑定到工作进程，将相关用户上下文和所述 VM 相关联，并利用相关联的用户上下文在所述工作进程中执行 VM，可以处理请求。通过一次只将一个 VM 和一个用户上下文与进程相关联，用户会话可以被彼此隔离，所以如果 VM 或者工作进程崩溃，只有相关联的用户会话才可能受到影响。

[0062] 可以使用多个机制将用户上下文与 VM 相关联，并且可以提供多个类型的对用户上下文中的数据的访问。通过使用低成本机制来将用户上下文与 VM 相关联（例如将用户上下文和 VM 映射到进程中），以及通过将阻塞 VM 从其工作进程分离并使用那些工作进程来作用于其他请求，吞吐量可以被优化。用户上下文可以在集群架构中被共享，其中，VM 跨越多个物理机器分布。可以使用分派器进程或部件以便以分布总体工作负荷或最小化跨越机器共享用户上下文的需求的方式在 VM 中分布请求。即使 VM 被永久性地绑定到工作进程，也可以在 VM 之间共享用户上下文（因此可以在用户会话之间 共享 VM）。

[0063] 为每一个用户会话分配单独的 VM 或者以如上所述的隔离方式共享 VM 可以在资源消耗方面产生可计量性。尽管针对特定 VM 或操作系统进程监视资源消耗可能是简单易见的，但是确定在 VM 或进程中运行的程序的哪个部分对特定资源（例如存储器、处理时间或用于垃圾回收的时间）的消耗负责却可能很困难。但是，如果一次只有一个用户上下文在 VM 中执行，则可以以每用户会话为基础计量资源，这经常是令人期望的。

[0064] 在又一个方面中，提供一种计算机实施的方法，包含：在第一运行时系统中标识要被与第二运行时系统共享或者要在第二运行时系统中使用的数据对象的集合，所述数据对象的集合中的每一个数据对象均包含对关于第一运行时系统中的数据对象的运行时元数据的引用；对于所述数据对象的集合中的每一个数据对象，用代理代替所述数据对象中的对运行时元数据的引用；和将所述数据对象的集合传送到第二运行时系统。

[0065] 在又一个方面中，提供一种计算机实施的方法，包含：从第一运行时系统接收数据对象的集合，该数据对象的集合中的每一个数据对象均包含代理；将所述数据对象的集合复制到第二运行时系统中；和对于所述数据对象的集合中的每一个数据对象：标识关于所述数据对象的运行时元数据；和用对关于所述数据对象的运行时元数据的引用代替所述数据对象中的代理。

[0066] 在又一个方面中，提供一种数据处理装置，包含：用于在第一运行时系统中标识要被与第二运行时系统共享或者要在第二运行时系统中使用的数据对象的集合的装置，该数据对象的集合中的每一个数据对象均包含对关于第一运行时系统中的数据对象的运行时元数据的引用；用于用代理代替所述数据对象的集合中的每一个数据对象中的对运行时元数据的引用的装置；和用于将所述数据对象的集合发送到第二运行时系统的装置。

[0067] 在又一个方面中，提供一种数据处理装置，包含：用于从第一运行时系统接收数据对象的集合的装置，该数据对象的集合中的每一个数据对象均包含代理；用于将所述数据

对象的集合复制到第二运行时系统中的装置；和用于标识关于所述数据对象的集合中的每一个数据对象的运行时元数据和用对关于所述数据对象的运行时元数据的引用代替每一个数据对象中的代理的装置。

[0068] 本发明的一个实施方案提供了全部上述优点。

[0069] 可以使用计算机程序、方法、系统或装置，或者计算机程序、方法或系统的任意组合来实施这些一般和具体方面。在下面的描述和附图中给出了本发明的一个或更多个实施例的细节。从这些描述、附图和权利要求，本发明的其他特征、目的和优点将很清晰。

附图说明

[0070] 图 1 是客户端 / 服务器系统的框图。

[0071] 图 2 是图 1 的客户端 / 服务器系统中的服务器的框图。

[0072] 图 3 是图 2 中服务器的另一个框图。

[0073] 图 4 是示出图 2 服务器中对请求的处理的过程流程图。

[0074] 图 5 是示出服务器中共享对象的使用的框图。

[0075] 图 6 是示出用于对象实例是否可共享的过程的流程图。

[0076] 图 7 是示出用于确定类是否可共享的过程的流程图。

[0077] 图 8 和图 9 是示出用于创建和使用共享对象的过程的流程图。

[0078] 图 10 是示出用于跨越多个机器共享对象的流程图。

[0079] 图 11 是示出用于在服务器中使用共享对象的流程图。

[0080] 图 12 是图 1 的客户端 / 服务器系统中的另一个服务器的框图。

[0081] 图 13 是图 12 中的服务器的另一个框图。

[0082] 图 14 是示出在图 12 的服务器中对请求的处理的框图。

[0083] 图 15 是示出在图 12 的服务器中对请求的处理的过程流程图。

[0084] 图 16 是生成在图 1 的客户端 / 服务器系统中的另一个服务器中对请求的处理的过程流程图。

[0085] 在各个附图中相同的附图标记和名称指示相同的元件。

具体实施方式

[0086] 隔离用户会话

[0087] 图 1 示出了客户端 / 服务器系统 100，其中，网络 150 将服务器 200 链接到客户端系统 102、104、106。服务器 200 是适于实施根据本发明的装置、程序或方法的可编程数据处理系统。服务器 200 为处理用户请求的一个或更多个运行时系统提供核心操作环境。服务器 200 包括处理器 202 和存储器 250。可以使用存储器 250 存储操作系统、用于在网络 150 上进行通信的传输控制协议 / 互联网协议 (TCP/IP) 堆栈，以及由处理器 202 执行的机器可执行指令。在一些实施方案中，服务器 200 可以包括多个处理器，每一个处理器均可用来执行机器可执行指令。存储器 250 可以包括共享存储器区域 255(在后面的图中示出)，共享存储器区域 255 可被在服务器 200 中执行的多个操作系统进程访问。可在客户端 / 服务器系统 100 中使用的适当的服务器的例子是 Java2 平台企业版 (J2EE) 兼容服务器，例如由德国的 SAP AG(SAP) 开发的网络应用服务器 (Web Application server) 或由纽约州 Armonk

的 IBM 公司开发的网际应用服务器 (WebSphere Application server)。

[0088] 客户端系统 102、104、106 可以执行多个应用或应用接口。应用或应用接口的每一个实例均可以构成用户会话。每一个用户会话均可以产生一个或更多个要被服务器 200 处理的请求。请求可以包括要在服务器 200 上的运行时系统 (例如 VM330) 上执行的指令。

[0089] 运行时系统是一种执行用户请求中的指令或者代码，并为所述代码提供运行时服务的代码执行环境。核心运行时服务可以包括例如进程、线程和存储器管理 (例如展示 (lay out) 服务器存储器 250 中的对象、共享对象、管理对对象的引用，以及将对象垃圾回收) 的功能。增强的运行时服务可以包括例如错误处理和建立安全性和连接性的功能。

[0090] 运行时系统的一个例子是虚拟机。虚拟机 (VM) 是一种抽象的机器，它可以包括指令集、寄存器集合、堆栈、堆，以及方法区域，像是真实的机器或处理器。VM 实际上担任了程序代码和实际处理器或硬件平台之间的接口，所述程序代码要在所述实际处理器或硬件平台上执行。所述程序代码包括来自操纵 VM 的资源的 VM 指令集的指令。VM 在该 VM 正在其上运行的处理器或硬件平台上执行指令，并操纵该处理器或硬件平台的资源，以便实 现所述程序代码的指令。以这种方式，相同的程序代码可以在多个处理器或者硬件平台上执行而不必针对每一个处理器或硬件平台被重写或重新编译。相反，VM 被针对每一个处理器或硬件平台实施，并且相同的程序代码可以在每一个 VM 中执行。可以在由处理器或硬件平台识别的代码中实施 VM。或者，可以在直接内置于处理器中的代码中实施 VM。

[0091] 例如，可以将 Java 源程序代码编译为叫做字节代码 (bytecode) 的程序代码。字节代码可以在运行于任意处理器或平台上的 Java VM 上执行。JavaVM 可以一次一个指令地解释所述字节代码，或者，可以使用即时 (just-in-time，JIT) 编译器针对真实的处理器或平台进一步编译所述字节代码。

[0092] 除了 Java VM 以外，VM 的其他例子包括高级商业应用编程语言 (Advanced Business Application Programming，ABAP)VM 和公共语言运行时 (Common Language Runtime，CLR)VM。ABAP 是用于为 SAP R/3 系统开发应用的编程语言，SAP R/3 系统是一种由 SAP 开发的被广泛安装的商业应用系统。公共语言运行时是一种被管理的代码执行环境，由华盛顿州 Redmond 的微软公司开发。为了简洁的目的，本说明书中的讨论集中在虚拟机上，但是要理解，这里描述的技术也可以用于其他类型的运行时系统。

[0093] 为了将用户会话彼此隔离从而提高系统的鲁棒性，服务器 200 可以被实施成为每一个用户均提供其自己的 VM。更具体地，可以为每一个用户会话提供其自己的可连接进程虚拟机 (process-attachable virtual machine，PAVM)，可连接进程虚拟机是可以被连接到 OS 进程和从 OS 进程分离的 VM。

[0094] 图 2 示出了使用可连接进程 VM(process-attachable VM) 的服务器 200 的实施方案。可连接进程 VM 被实例化并被用于每一个用户会话——例如，VM301 被实例化并用于用户会话 1，并且 VM303 被实例化并用于用户会话 2。总的来说，如果有 U 个用户会话，则服务器 200 将具有 U 个相应的 VM。这在图 2 中概念性地由 VM309 示出，VM309 相应于用户会话 U。

[0095] 图 2 中示出的服务器 200 的实施方案还包括 :用来存储可连接进程 VM 的共享存储器区域 255、工作进程的进程池 400，以及分派器进程 410。工作进程的进程池 400 包含许多被分配给服务器 200 的操作系统进程——例如，工作进程 401、403 和 409。总的来说，可以

给服务器 200 分配 P 个工作进程，如在图 2 中概念性地由工作进程 409 所示那样。为了降低同步开销和上下文 切换 (context switch) 的数量，被分配给服务器 200 的操作系统进程数量 P 应该大致等于在服务器 200 运行于其上的计算机上可用的处理器数量。但是，即使服务器 200 只使用一个操作系统进程来处理用户请求，也能够获得改善的隔离和鲁棒性的益处。

[0096] 在操作中，当图 2 中的服务器 200 接收到请求时，特殊的进程（分派器进程 410）将所述请求分派给工作进程池 400 中的可用工作进程之一。假定所述请求相应于用户会话 1（用户会话 1 转而相应于 VM301）并且分派器进程 410 将所述请求分派给工作进程 401，为处理所述请求，服务器 200 将 VM301 连接或者绑定到工作进程 401。在所述请求已被处理以后，服务器可以将 VM301 从工作进程 401 分离。然后，工作进程 401 可被用来处理另一个请求。例如，如果服务器 200 接收到相应于用户会话 2 的新请求，并且分派器进程 410 将该请求分派给工作进程 401，则相应的 VM303 能够被连接到工作进程 401 以便处理该新的请求。继续这个例子，如果在工作进程 401 仍在被用来处理相应于用户会话 2 的请求时服务器 200 接收到另一个相应于用户会话 1 的请求，则分派器进程 410 可以将所述来自用户会话 1 的新请求分派给工作进程 403，并且用于用户会话 1 的 VM301 可以被连接到工作进程 403，以便处理来自用户会话 1 的新请求。

[0097] 不要求服务器 200 具有分派器进程 410。例如，在另外的实施方案中，用户请求可以被顺次地分配到被分配给服务器 200 的进程。每一个进程均能够维持请求队列，并连接相应于所述队列前端的请求的用户会话的可连接进程 VM，以便处理该请求。

[0098] 如上所述，可连接进程 VM 是能够被连接到 OS 进程和从 OS 进程分离的 VM。为了能够将 VM 从进程分离（以及将 VM 连接到另一个进程），需要去除在所述 VM 和该 VM 在其中运行的进程之间的亲和性。当 VM 被从进程分离时，所述 VM 的状态需要是持久的。当所述 VM 被连接到另一个进程时，所述 VM 的状态需要是非持久的。因此，需要以使得所述状态能够持久和非持久的方式产生和维持 VM 的状态。而且，当将 VM 从进程分离时使所述 VM 的状态持久以及当将所述 VM 连接到不同的进程时使所述 VM 的状态非持久最好应该是低成本的操作。这可以通过将 VM 的状态存储在可由分配给所述服务器的 OS 进程访问的共享存储器中来实现。

[0099] 为了使被置于共享存储器中的 VM 访问相应于用户会话的用户上下文， 所述用户上下文——包括用户堆 (heap) 和用户堆栈 (stack)——必须也被置于共享存储器中。（或者，用户上下文可以被复制到 VM 被绑定到的进程的地址空间——下面更详细地讨论这种实施方案）。因此，在服务器 200 的一个实施方案中，用户上下文也被存储在共享存储器中。将用户堆存储在共享存储器中是简单易见的——可以简单地从共享存储器段分配用于堆的存储器。存储用户堆栈可能更困难，因为在某些实例中，例如 Java VM，用户堆栈和 VM 堆栈被混和在一起。在这种情形下，一个解决方案是将 VM 的完整堆栈，包括用户堆栈，存储在共享存储器中。在一个实施方案中，这通过将 VM 实施为操作系统协同例程 (co-routine) 来实现。

[0100] 典型的子例程显示了分层关系。例如，子例程 A 在调用子例程 B 时挂起，子例程 B 在结束时将控制返回子例程 A，子例程 A 从挂起点继续执行。相反，协同例程具有平行的而非分层的关系。因此，例如协同例程 A 在调用协同例程 B 时挂起，但是协同例程 B 在将控制

返回协同例程 A 时也挂起。对于协同例程 B，这种返回控制好像是调用协同例程 A。当协同例程 A 随后调用协同例程 B 并挂起时，协同例程 B 表现得就好像其先前对协同例程 A 的调用已经返回一样，并且它从该调用点继续执行。因此，控制在两个协同例程之间反射，每一个均在其先前被暂停的地方继续。

[0101] 协同例程可以被比做线程，因为在某些实施方案中协同例程的每一个均具有其自己的堆栈并共享堆。但是，线程和协同例程之间的一个差别是操作系统承担线程之间的调度，而程序员必须承担协同进程之间的调度。如下面所解释的那样，协同例程可以被用来模拟线程。例如，Linux glibc 库内的函数集合，包括 setcontext()、getcontext()、makecontext() 和 swapcontext() 函数，可被用来在进程内开始新的协同例程，在协同例程之间切换，并且重要的是为协同例程的堆栈提供存储器。该最后一个特征可被用来从共享存储器分配 VM 的堆栈（以及将堆栈存储在共享存储器中）。

[0102] 在服务器 200 的一个实施方案中，当用户会话开始时，产生并初始化相应于用户会话的可连接进程 VM。共享存储器的被称为“会话存储器”的专用块 (private block) 被分配给 VM。从共享存储器的这个专用块直接分配 VM 的堆和堆栈。使用共享存储器来存储 VM 的状态使得将 VM 绑定或者连接到 OS 进程的过程实质上是空操作，因为操作系统进程可以简单地将 VM 的会话存储器映射到其地址空间中。类似地，将 VM 从 OS 进程分离的过程只要求 将 VM 的会话存储器简单地从 OS 进程的地址空间解除映射。实际上没有移动或者复制数据。

[0103] 为了能够将 VM 从进程分离（以及将 VM 连接到另一个进程），VM 使用的输入 / 输出 (I/O) 资源，例如文件句柄和套接字 (socket)，也需要是持久的。需要以使得 I/O 资源能够是持久和非持久的方式来产生和保持 VM 使用的 I/O 资源。这可以通过使用额外的间接层次来访问 I/O 资源（或这些资源的代理）实现。例如，VM 看作文件或套接字描述符的东西实际上只不过是该文件或套接字的句柄。句柄自身就可以是可持久的，例如，文件句柄可以被存储在 VM 的会话存储器中。或者，可以通过使用资源管理器使得句柄可持久——例如，在套接字的情况下，可以使用描述符传递来通知套接字管理器关于针对特定套接字的 I/O 请求；然后，当 I/O 请求完成时，套接字管理器可以通知分派器。

[0104] 某些 VM 可以是多线程的。线程实质上是占位符 (placeholder) 信息，该信息使得程序能够在程序的单次使用中处理多个并发用户或者服务请求。从程序的观点，线程是为一个单独用户或者特定服务请求服务所需的信息。如果多个用户正在并发地使用程序，或者如果程序接收到并发请求，则针对每一个这样的用户或者请求均创建和维持一个线程。当程序另外代表不同的用户或者请求重新进入时，线程允许程序知道哪个用户或请求正被服务。

[0105] 对于共享存储器来说，本机 OS 线程不能很容易地持久。因此，为了能够将多线程 VM 从进程分离（以及将 VM 连接到另一个进程），可以由用户层机制模拟本机线程。采用用户层机制模拟本机线程有时候被叫做提供“绿色线程”功能。绿色线程是由用户而非操作系统调度的用户层线程。

[0106] 可被用来模拟本机线程的一个机制是操作系统协同例程。如上面所说明的那样，除了程序员（而非操作系统）承担在协同线程之间调度以外，协同例程类似于线程。因此，用来实施 VM 线程的本机 OS 线程可以被映射到协同例程。和线程管理与调度相关的所有数

据结构，包括线程的调用堆栈、互斥体和用于 Java 监视器的条件变量，可以被保存在 VM 的会话存储器中。在 Java VM 的情况下，它既可以包括 Java 堆栈也可以包括 VM 实施所用的 C 堆栈，例如动态方法调用的 Java 本机接口 (Java Native Interface，JNI) 实施的 C 堆栈。

[0107] 操作系统一般先发制人地调度线程，以便使公平性最大化（即给予每一 个线程在某个时间点运行的机会）。相反，由程序员处理的协同例程调度通常不是先发制人的。但是，那不一定是个缺点，因为在服务器的上下文中，请求吞吐量经常比公平性更重要。请求吞吐量是可扩展性的基本目标，通过使用批处理策略来调度协同例程可以使其最大化。在批处理策略中，在每一个 VM 内，当进入等待状态时（例如当 I/O 上或例如 Java 监视器的监视器上的阻塞时），每一个协同例程向调度器（有时仍称为线程调度器）合作地让步。作为上面描述的 I/O 重定向机制的一部分，可以包括阻塞 I/O 调用和线程调度器之间的协作。在会话存储器中可以实施互斥体和条件变量而无需使用操作系统锁定基元作为受调度器控制的变量。

[0108] 协同例程调用可以持续一个 PAVM，直到其所有协同例程已经进入等待状态，指示用户请求或者完成，或者在等待 I/O。在任一情况下，PAVM 均可被从工作进程分离。当下一个请求来自用户会话时，或者当 I/O 请求完成时，分派器进程 410 能够将 PAVM 重新连接到可用工作进程。

[0109] 如上所述，用来实施 VM 线程的本机线程可以被映射到协同例程。相反，只在 VM 中内部使用的例如用于垃圾回收的本机线程可以被用同步函数调用代替。使用同步函数调用可被认为是一种模拟内部本机线程的方法。或者，如果要由内部本机线程执行的函数要被异步地执行，则可以在服务器内被指定的较低优先级进程上调度和执行该函数。而另一种选择是将某些函数一起省略。例如，有可能在用户会话终结时简单地释放所有的会话存储器而非在用户会话在活动时执行垃圾回收。

[0110] 图 3 示出了示范性 Java2 平台企业版 (J2EE) 应用服务器 200，它包括共享存储器 255 和工作进程 401、403。J2EE 是针对对于大企业来说较为典型的大型机规模计算而设计的 Java 平台。通过产生标准化的、可重用模块化部件并使得各层能够自动处理编程的很多方面，J2EE 简化了在分层、瘦客户端环境中的应用开发。J2EE 包括 Java2 平台标准版 (J2SE) 的很多部件，例如 Java2 开发包 (Java2 Development Kit，JDK)、对公共对象请求代理架构 (Common Object Request Broker Architecture，CORBA)、Java 数据库连接 2.0(Java Database Connectivity 2.0，JDBC) 和安全性模型的支持。J2EE 还包括对企业 Java Bean(Enterprise Java Bean，EJB)、Java 服务器小程序应用编程接口 (API) 和 Java 服务器页面 (Java Server Pages，JSP) 的支持。

[0111] 每一个工作进程 401、403 均可以包括容器 204 和例如远程方法调用 (Remote Method Invocation，RMI) 接口 206 和数据库管理系统 (databasemanagement system，DB) 接口 208 的服务接口。RMI 是面向对象的编程技术，它使得不同计算机上的对象能够通过分布式网络交互作用。RMI 是通常叫做远程过程调用 (remote procedure call，RPC) 的协议的 Java 版本，但是具有附加的将一个或更多个对象与请求一起传递的能力。

[0112] 容器 204 可以包括为服务器架构提供比由本机 OS 提供的框架更适合的框架的 OS 接口。该 OS 接口可以从本机 OS 接管某些函数的责任，例如调度、存储器管理、进程架构、集群、负载平衡和联网。通过接管这些函数，容器 204 能够以将例如请求吞吐量的某些目标进

行优化的方式控制关键资源的使用，所述关键资源例如处理器和存储器。容器 204 也可以起到向在服务器上执行的应用隐藏该服务器的本机操作系统细节的作用。

[0113] 此外，容器 204 起到接口的作用，通过所述接口，可连接进程 VM 可以被连接到进程、在所述进程中执行，以及从所述进程分离。因此，在图 3 中所示的服务器实施方案中，容器 204 提供了一些用于使用可连接进程 VM 的功能，例如使 VM 的状态持久和非持久（例如，通过将分配给 VM 的共享存储器块映射到进程的地址空间中）和 VM 线程或协同例程的调度。

[0114] 在图 3 中的例子中，第一用户会话（用户会话 1）被绑定到工作进程 401。用户会话 1 包括相应的用户上下文 501 和 VM301。共享存储器 255 的块 257 被分派给用户会话 1。用户上下文 501 和 VM301 的状态被存储在共享存储器块 257 中。在这个例子中，相应于用户会话 1 的请求已被接收并分派到工作进程 401。为了处理这个请求，相应于用户会话 1 的 VM301 被绑定到工作进程 401。因此，用户上下文 501 和 VM301 的状态都已经被从共享存储器块 257 映射到工作进程 401 的地址空间中。然后，VM301 被工作进程 401 执行，以便处理所述请求。当请求已被处理时，用户会话 1 可以被从工作进程 401 解除绑定，意味着相应的 VM 和用户上下文可以从工作进程 401 分离（例如通过将用户上下文 501 和 VM301 的状态从工作进程 401 的地址空间解除映射）。

[0115] 图 3 还示出了被绑定到工作进程 403 的用户会话 2。用户会话 2 包括相应的用户上下文 503 和 VM303。共享存储器 255 的块 259 被分配给用户会话 2。用户上下文 503 和 VM303 的状态被存储在共享存储器块 259 中，并被映射到工作进程 403 的地址空间中。当来自用户会话 2 的请求已经被处理时， 用户会话 2 可以从工作进程 403 解除绑定，意味着相应的 VM 和用户上下文可以从工作进程 403 分离（例如通过将用户上下文 503 和 VM303 的状态从工作进程 403 的地址空间解除映射）。

[0116] 如果新请求自用户会话 1 抵达，假定或者因为 VM303 处于等待状态，或者因为 VM303 已完成处理来自用户会话 2 的请求，VM303 已经从工作进程 403 分离，则该新请求可以被分派到例如工作进程 403。然后，相应于用户会话 1 的 VM301 和用户上下文 501 可以被绑定到工作进程 403。在一个实施方案中，这不要求移动或复制任何数据——而是将共享存储器块 257（或者其适当部分）简单地映射到工作进程 403 的地址空间中。然后，工作进程 403 可以执行 VM301 以便处理来自用户会话 1 的新请求。

[0117] 因此，用户会话可以被绑定到不同进程并能在不同进程之间移动。以这种方式，只要工作进程成为可用，请求就可以被处理。而且，将用户会话映射到不同进程一般是非常便宜的操作。结果，请求吞吐量得到优化。使用可连接进程 VM 还导致了可扩展的服务器，因为可以通过给服务器分配更多进程进一步优化请求吞吐量。还有可能给底层的计算机添加更多的处理器，以便更好地处理增加数量的进程。

[0118] 使用可连接进程 VM 还使得服务器稳健。这是因 OS 在进程（例如服务器进程池 400 中的工作进程）之间提供的隔离所致。而且，还有可能通过一次只将一个用户会话的存储器和 VM 映射到工作进程中保护与用户会话相关联的存储器和状态。

[0119] 在服务器 200 的另一个实施方案中，两个或更多个 PAVM 被连接到一个进程并在所述进程中执行。例如，ABAP VM 和 Java VM 可以在一个进程中执行。在一个进程中运行 ABAP VM 和 Java VM 实现了在同一应用中既使用 ABAP 也使用 Java 部件。因此，可以开发应用以

有利地利用来自 ABAP 和 Java 环境这两者的有用部件。下面描述的技术可被用来在一个进程中执行多个 VM。例如，进程可被用来执行 Java VM 和 ABAP VM 以及 CLR VM。

[0120] 通过使用协同例程可以实现在一个进程内执行多个 VM。在 ABAP VM 和 Java VM 要在一个进程中运行的例子中，可以使用两个协同例程——一个用于 ABAP VM，一个用于 Java VM。除了包括 Java VM 或者 ABAP VM 以外，每一个协同例程还可以包括容器（或容器的一部分），以便处理例如调度和两个 VM 之间以及与服务器外部的应用的通信这些函数。

[0121] 继续 ABAP/Java 例子，如果 ABAP VM 是应用将使用的主要 VM，则执行 ABAP VM 的协同例程可以包括如上所述的容器。在 ABAP VM 内执行的请求第一次调用 Java 对象时，容器可以为 Java VM 堆栈（包括 Java 用户堆栈）分配存储器，并开始新的协同例程来执行 Java VM。可以被传递到新的协同例程的参数包括被分配的存储器的位置和大小，以及所述新的协同例程的启动函数（在这种情况下是 Java VM 自身）。然后，Java VM 可以开始主类 (mainclass)，例如实施可被用来运行 J2EE 部件的 J2EE 容器，J2EE 部件例如 Java 服务器小程序、企业 Java Bean 或 Java 服务器页面。在其执行的同时，Java VM 能够调用容器功能，以便例如从消息区域读取请求并将响应写到所述消息区域。对于调度，Java VM 在其已经写过响应或者请求之后能够将控制返回 ABAP VM。然后，在新请求或对其自己的请求的响应抵达后，该 Java VM 可以被重新调度。

[0122] 图 4 是示出范例过程 450 的流程图，过程 450 用于在图 2 中所示的服务器实施方案中处理请求。过程 450 通过使用可连接进程 VM 以可扩展的方式在用户会话之间提供隔离。当用户会话开始时，为该用户会话产生并初始化可连接进程 VM(452)。这个操作可以包括为该 VM 分配存储器块。所分配的存储器可以是共享存储器（以便使得该 VM 能够被服务器中的多个进程访问），但是它也可以被指定为专用的，意味着该存储器块将只属于新创建的 VM。

[0123] 在存储器块已经被分配给 VM 后，该 VM 可以被存储在该存储器块中。VM 的计算状态（包括 VM 堆栈和堆）以及相应的用户会话的用户上下文（包括用户堆栈和堆）都可以被存储在该存储器块中。用户上下文可以包括例如文件的 I/O 资源的句柄，以及例如套接字的 I/O 资源的代理（例如资源管理器）的句柄。

[0124] 初始化 VM 可能是昂贵的操作，因为它可能涉及加载、验证和解析几个类（例如 Java 系统类），以及执行系统类中的许多静态初始化器 (staticinitializer)。通过使用预先初始化的“主”VM，可以降低这些初始化开销。不是从零开始初始化新的 VM，而是主 VM 的存储器块可以被简单地复制到新 VM 的存储器块中。将主 VM 的存储器块的模板映像 (image) 复制到新 VM 的存储器块中使得新 VM 能够在已经被初始化的状态中开始运行。如果某些初始化操作只能在新 VM 实际启动时被执行，则可以使用部分初始化 VM 的模板映像，以便在开始后，新 VM 只需要执行要求在实际启动时间执行的那些操作。

[0125] 通过另一种优化可以进一步降低初始化（以及执行）开销：将类型信息（例如被加载的类的运行时表示）存储在共享存储器可以被所有 VM 访问的部分中。这个技术可以降低每一个 VM 所致的类加载、验证和解析的开销，并且如果用来共享可能被每一个用户上下文使用的系统类的字节代码，则可能特别有用。可以使用类似的技术共享用户类的字节代码。在使用即时 (JIT) 编译器的实施方案中也可以共享编译过的代码。

[0126] 在 VM 已经被初始化后，VM 可被用来处理来自相应的用户会话的用户请求。当接

收到来自相应的用户会话的用户请求时 (454)，选择来自被分配给服务器的进程池的可用进程来处理请求 (456)。然后，发送了该请求的用户会话的 PAVM 被绑定到被选择的进程 (458)。

[0127] 如果用来存储 VM 的存储器块是可被服务器的所有进程访问的共享存储器，则绑定实质上可能是空操作——例如，存储器块或其一部分，可以被简单地映射到被选择的进程的地址空间中。被映射的部分可以包括 VM 的计算状态。或者，在不同的方式中 VM 可以是非持久的。例如，VM 的计算状态可以被从文件复制。但是，这种操作的性能可能不好，特别是和将共享存储器映射到进程地址空间中的高效操作相比，所述高效操作通常不要求复制或者移动数据。

[0128] 在 VM 已经被绑定到被选择的进程后，VM 可以被进程执行，以便处理用户请求 (460)。这可能涉及执行两个或更多个协同例程来在 VM 内模拟线程。这还可能涉及执行同步函数调用来模拟内部线程的执行（例如执行垃圾回收的线程）。如果这样的内部线程或者另一个函数被异步地执行，则被分配给服务器的进程中一个可以被指定为低优先级进程，并被用来执行用于 VM 的函数。例如，这个技术可被用来在 VM 中执行垃圾回收。可以调度更重要的函数在正常或较高优先级进程中运行。

[0129] 在 VM 已经处理过用户请求之后，VM 从被选择的进程分离 (462)。然后，被选择的进程可以被返回可用进程池（或者被标记为可用），以便它可以被用来处理服务器接收到的新请求。和绑定一样，将 VM 从进程分离可能是简单、低成本的操作 :VM 的共享存储器块可以被简单地从进程地址空间解除映射。或者，使 VM 持久可能涉及更复杂或更昂贵的操作，例如将 VM 的 计算状态保存到文件。将 VM 连接或者绑定到进程并将该 VM 从该进程分离或者解除绑定可能是彼此的镜像——即为使 VM 非持久而执行的操作可以被逆转，以便使 VM 持久。然后，服务器可以等待来自用户会话的另一个请求 (454)。

[0130] 当用户会话终结时，其相应的 VM 也可以被终结，并且其全部资源，包括其被分配的存储器，可以被释放。如果 VM 的存储器在用户会话末尾被释放，则有可能在 VM 的寿命期间省略垃圾回收。在用户会话末尾终结 VM 的另一选择是重新使用该 VM，即将该 VM 与不同的用户会话相关联并使用该 VM 来处理来相应于该用户会话的请求。对于要求保持非常小的用户上下文（甚至没有用户上下文）的应用以及具有非常短的用户会话的应用来说，这个技术可能特别有用（不要求保持用户上下文的应用有时候被称为无状态应用）。对于所有这些应用，使用顺次可重用 VM 池可以帮助将和创建、初始化、维持和终结 VM 相关联的开销最小化。

[0131] 关于可连接进程 VM 的实施和使用的额外细节可以在 N.Kuck、H.Kuck、E.Lott、C.Rohland 和 O.Schmidt 的 SAP VM Container :Using ProcessAttachable Virtual Machines to Provide Isolation and Scalability for LargeServers(2002年8月1日)（在第二届 USENIX Java 虚拟机研究和技术研讨会 (Java VM'02) 上作为正在开展的工作报告被提交和展示了未出版的摘要）中找到。

[0132] 创建和使用共享对象

[0133] 为每一个用户会话分配一个 VM 意味着给每一个用户会话均提供了其自己的 VM 堆（例如 Java 堆）。因此，在使用单个多线程 VM 的服务器 200 的实施方案中将只存在一次的 VM 堆中的数据对象被针对每一个用户会话复制一次。这可能既导致资源消耗（例如用于存

储被复制的数据对象的存储器）也导致时间消耗（例如用于建立和初始化被复制的数据对象的时间）。

[0134] 作为一个简单的例子，考虑代表在应用启动时刻被解析的可扩展标记语言（XML）配置文件的大型 Java 数据结构。在为相应于应用的每一个用户会话复制这样的数据结构时，服务器 200 浪费了除第一个 VM 和用户会话以外全部的 CPU 时间（用于解析 XML 文件和构建数据结构）和存储器（用于存储数据结构）这两者。

[0135] 为了缓和这个问题，服务器 200 的一个实施方案使得数据对象能够在 VM 之间共享。在服务器 200 的这个实施方案中，共享存储器区域或堆被用来存储可被多个 VM 访问的数据对象。

[0136] 共享存储器堆中的数据对象一般应该不具有任何对任何专用堆（例如各 VM 的专用堆）的引用或者指针。这是因为如果共享存储器堆中的对象具有带有对一个特定 VM 中的专用对象的引用的成员变量，则该引用对于所有其他使用该共享对象的 VM 来说将是无效的。更正式地说，可以按如下考虑这个限制：对于每一个共享对象，被初始对象引用的对象的传递闭包始终应该只包含共享对象。

[0137] 因此，在服务器 200 的一个实施方案中，对象不被单独地放入共享存储器堆——而是将对象按叫做“共享闭包”的组放入共享存储器堆。共享闭包是初始对象加上被该初始对象引用的所有对象的传递闭包。

[0138] 在图 5 中在概念上示出了通过共享闭包共享对象，其中，通过将第一对象 601 和第一对象 601 引用的所有对象的传递闭包分组在一起，已经在第一 VM301 中标识了共享闭包 600。在已经标识了共享闭包 600 后，VM301 能够在共享存储器区域 255 中创建共享闭包，例如通过将共享闭包 600 复制到共享存储器区域 255 中。在已经在共享存储器区域 255 中创建了共享闭包 601 之后，它可以被服务器中的 VM 访问（例如 VM301、303 和 305）。VM 能够通过例如将共享闭包 600 从共享存储器区域 255 映射或者复制到该 VM 正在其中执行的进程的地址空间中，访问来自共享存储器区域 255 的共享闭包 600。下面更详细地讨论共享闭包的创建和使用。

[0139] 为了可在共享闭包中使用，对象必须“可共享”。总的来说，如果一个运行时系统（例如 Java VM）中的复杂数据结构（例如堆或其一部分）可以被分解并且随后被用第二运行时系统的本机格式重装而不打破该数据结构的内部一致性或功能，则该数据结构可被与第二运行时系统共享。

[0140] 在服务器 200 的一个实施方案中，通过将共享闭包复制到共享存储器和从共享存储器复制共享闭包来共享对象。对象要在这个实施方案中可共享，所述对象必须能够承受到另一个 VM 的地址空间中的透明深复制（transparentdeep-copy）而不打破对象的内部一致性或功能。下面更详细地讨论了针对这种实施方案的可共享性要求。

[0141] 尽管可共享性的大多数方面一般是对象类的属性，但是对象实例的可共享性不仅依赖于其类的属性，还依赖于对象实例的成员变量的类型。在成员 变量可能具有直到运行时才能被确定的运行时类型的情况下，必须在运行时确定共享闭包内对象实例的可共享性。

[0142] 因此，在对象实例具有运行时类型的服务器实施方案中，可以在可共享类和可共享对象实例之间加以区别。如果类满足可共享性标准，则该类可共享，下面提供了可共享性

标准的例子。如果对象实例的运行时类型是可共享类，并且如果它引用的所有对象都是可共享对象实例，则该对象实例是可共享对象实例。换句话说，如果下面的条件都被满足，则对象实例是可共享对象实例：(i) 对象的运行时类是可共享类，和 (ii) 对象实例的所有非空引用类型成员变量是可共享对象实例。

[0143] 第一个条件（对象的运行时类是可共享类）旨在保证运行时类的实例在语义上能够应付共享。下面提供了用于确定运行时类是否可共享的示范标准。虽然确定运行时类是否可共享只需要每个类进行一次，但是这种特性不可继承，因为派生类可能添加和共享不兼容的功能。

[0144] 第二个条件（对象实例的所有非空引用类型成员变量自身是可共享对象实例）旨在保证共享闭包中的所有对象可共享。通过递归检查对象实例中的引用可以确定是否满足这个条件。由于“可共享类”特性的非继承性所致，简单地检查对象的所有成员变量的已声明类型是不够的，因为尽管已声明类型可能是可共享的，但是运行时类型可能不是可共享的。

[0145] 图 6 示出了范例过程 650 的流程图，过程 650 可被用来确定对象实例是否是可共享对象实例。在图 6 中所示的过程中，首先接收到对象实例的标识 (652)。对象实例的运行时类被标识，并确定该运行时类是否是可共享类 (654)。如果运行时类不是可共享类（判断 656 的“否”分支），则过程以对象实例不可共享的指示结束 (658)。例如，通过提出“不可共享”异常可以做出这样的否定指示。

[0146] 如果对象实例的运行时类是可共享类（判断 656 的“是”分支），则该对象实例引用的对象被标识 (660)。然后，过程 650 遍历被引用的对象以便确定被引用的对象是否是可共享对象实例。如果有更多的被引用对象（判断 662 的“是”分支），则选择剩余的被引用对象之一 (664)。然后，确定被引用的对象是否是可共享对象实例 (666)。如果被引用的对象不是可共享对象实例（判断 668 的“否”分支），则过程 650 以初始对象实例不可共享的指示结束 (658)。这是因为被初始对象实例引用的对象之一不是可共享对象实例，并 且如先前所陈述的那样，对于要可共享的对象，初始对象引用的所有对象必须都是可共享对象实例。

[0147] 如果被引用的对象是可共享对象实例（判断 668 的“是”分支），则过程 650 查看是否有更多的被引用对象要被分析。如果有更多的被引用对象（判断 662 的“是”分支），则过程 650 选择剩余的被引用对象之一并和前面一样继续下去。如果不存在更多的被引用的对象（判断 662 的“否”分支）并且过程还没有终结，则那意味着所有被引用的对象已经被分析并被确定为可共享对象实例。因此，过程以初始对象实例是可共享的指示结束 (670)。

[0148] 确定被引用的对象是否是可共享对象实例可以被递归地进行——即针对被引用的对象可以再次调用过程 650，如图 6 中虚线所示。换个说法，被引用的对象可以被递归地遍历，所以可以针对被引用对象的传递闭包中的每一个对象做出确定。

[0149] 如果初始对象和被引用的对象的传递闭包中的所有对象都是可共享对象实例，则这些对象可以被分组到共享闭包中并被与另一个运行时系统共享（例如通过将共享闭包复制到共享存储器区域）。

[0150] 过程 650 可以被认为最终确定共享闭包中的每一个对象的运行时类是否是可共享类。如先前所说明的那样，在通过将共享闭包复制到共享存储器和从共享存储器复制共

享闭包来共享对象的实施方案中，如果对象能够承受到另一个 VM 的地址空间中的透明深复制 (transparent deep-copy) 而不打破对象的内部一致性或功能，则对象一般被认为是可共享的。在这样的实施方案中，如果 VM 串行化或解串行化类的对象实例时不执行任何定制代码，则一般可以认为类是可共享的。这个规则的基本原理是：如果 VM 不需要执行任何定制串行化或解串行化代码，则用来将共享闭包复制到共享堆（或从共享堆复制到 VM 的地址空间）的深复制操作在语义上等同于共享闭包中的对象的串行化和解串行化。因此，如果共享闭包已经被复制到共享堆，则将该共享闭包映射或者复制到其自己的地址空间的 VM 应该能够访问共享闭包中的对象而无需任何额外的解串行化对象所需的动作。

[0151] 图 7 示出了范例过程 750 的流程图，过程 750 可被用来确定特定类是否是可共享类。例如，在过程 650 要求确定特定运行时类的可共享性的情况下 (654) 可以使用过程 750。在图 7 中所示的过程中，首先接收到类（例如对象实例的运行时类）的标识 (752)。然后应用很多标准来确定该类是否可共 享。具体来说，类要可共享，则该类必须满足所有下列条件。

[0152] 首先，类必须可串行化 (754)。在 Java 类的情况下，这可以通过检查该类是否实施了标记接口 (marker interface)java.io.Serializable 确定。实施 java.io.Serializable 表明类的对象实例通常可以被以有意义的方式复制到另一个地址空间中。因此，如果类的确实施了 java.io.Serializable 接口，则第一个条件得到满足。

[0153] 第二，类绝不可包括任何定制串行化或解串行化代码 (756)。在 Java 类的情况下，这可以通过检查类是否实施了下列方法中的任何一个来确定：

[0154] private void readObject(ObjectInputStream)；

[0155] private void writeObject(ObjectOutputStream)；

[0156] public void readExternal(Objectlnput)；

[0157] public void writeExternal(ObjectOutput)；

[0158] {any access}Object readResolve()；

[0159] {any access}Object writeReplace()；

[0160] 上面的方法构成了在串行化或者解串行化期间执行的定制代码。这些定制代码不能被自动地证明等同于在创建共享闭包期间执行的深复制操作。因此，在使用深复制操作来创建共享闭包的情况下，实施上面函数中的任何一个排除了在过程 750 中将类自动地视为可共享类的可能。

[0161] 第三，所讨论的类的所有基类必须是可串行化的 (758)。在 Java 类的情况下，这可以通过检查是否所有的基类实施了 java.io.Serializable 或具有平凡缺省构造器来确定——如果是，则第三个条件得到满足。如果任何基类没有实施 java.io.Serializable，则其缺省构造器被在解串行化期间执行。如果缺省构造器是平凡的——即，如果构造器为空或者调用了基类的平凡缺省构造器，这可通过缺省构造器的递归查验确定——则缺省构造器的调用对解串行化没有任何影响。非平凡缺省构造器排除了在过程 750 中将类自动地视为可共享类的可能，因为缺省构造器可能包括不等同于深复制操作的定制代码。

[0162] 第四，所讨论的类的所有成员域必须被串行化 (760)。在 Java 类的情况下，这可以通过检查类是否具有任何瞬态域或 serialPersistentFields 域来确定。瞬态域是在解串行化期间被设置为其缺省值的域。因此，将具有瞬态域的类的对象实例解串行化可能不等

同于对象实例的深复制。因此，在类中存在瞬 态域排除了在过程 750 中将类自动地视为可共享类的可能。具有 serialPersistentFields 域的类也被排除，因为这些类只不过是指示具有瞬态域的类的另一种方式。

[0163] 第五，类绝不可具有任何垃圾回收副作用 (762)。被共享的对象可能具有和使用它们的 VM 的生命周期不同的生命周期，因此可能影响在 VM 内执行的垃圾回收算法。垃圾回收副作用排除了在过程 750 中将类自动地视为可共享类的可能，因为副作用可能妨碍垃圾回收算法的正确操作。在 Java 类的情况下，过程 750 能够通过检查类具有平凡终结器以及类不是从 java.lang.ref.Reference 类派生的来确定这个条件得到满足。平凡终结器是为空或者调用基类的平凡终结器的终结器。

[0164] 如果所有上面五个条件都被满足，则过程 750 以所讨论的类是可共享类的指示结束 (766)。另一方面，如果任一条件没被满足，则过程 750 以所讨论的类不是可共享类的指示结束 (764)。

[0165] 在服务器 200 的一个实施方案中，如果通过被自动施加的过程（例如过程 750）发现类是可共享的，或者如果该类先前已经被声明是可共享的，则该类被视为是可共享的。即，即使自动施加的对类的分析未能指示该类是可共享的，该类也可能是可共享的。

[0166] 如果类已经被检查过（例如通过手工查阅其源代码）并发现适于共享，则该类可以被声明为可共享。例如，在通过将共享闭包复制到共享存储器和从所述共享存储器复制所述共享闭包来共享对象的实施方案中，如果语义检查证明对上面规定的可共享性标准的所有违犯都是无害的，则类可以适于共享。如果尽管有那些违犯，但是可以说明用来将共享闭包复制到共享堆中（或者从共享堆复制到 VM 的地址空间中）的深复制操作在语义上等同于共享闭包中的对象的串行化和解串行化，则对可共享性标准的违犯一般是无害的。

[0167] 不满足上面规定的可共享性标准但是仍适于共享的类的一个简单例子是类 java.lang.String（因为该类在 Java2 平台标准版 1.3 中定义）。java.lang.String 类违犯了上面规定的第四个条件，因为它包括 serialPersistentFields 域。手工检查该类中的代码指出，为了在串行化期间实施类的对象实例的特殊处理包括了该域，所述特殊处理是串行化协议的要求。尽管如此，能够很容易地表明，对于该类来说，深复制的效果等同于串行化。因此，java.lang.String 类可被声明为可共享。

[0168] 不满足上面规定的可共享性标准但是仍适于共享的类的更为复杂的例子是类 java.util.Hashtable（因为该类在 Java2 平台标准版 1.3 中定义）。java.util.Hashtable 类违犯了上面规定的第二和第四个条件，因为它包含定制串行化方法和瞬态域。查阅该类中的代码指出，因为在串行化期间不保留哈希代码 (hashcode)，这迫使哈希表在解串行化期间重建其内容，所以要求定制串行化方法和瞬态域。但是由于深复制操作保留哈希代码，所以可以说明深复制操作等同于串行化和解串行化。结果，类 java.util.Hashtable 也可以被声明为可共享。

[0169] 通过在图 8 和图 9 中示出的范例过程 850 和 950，可以实现共享闭包的创建和使用，这在图 5 中被概念性地示出。

[0170] 过程 850 绘出了可用来创建共享闭包的示范过程。在过程 850 中，接收到第一运行时系统（例如 VM）中的初始对象的标识 (852)。然后，共享闭包——即初始对象加上该初始对象引用的所有对象的传递闭包——被标识 (854)，并确定该共享闭包是否可在另一运

行时系统（例如另一个 VM）中使用或者被其共享 (856)。例如，可以通过确定共享闭包中的对象是否可共享（或者更准确地，通过确定共享闭包中的每一个对象实例是否是可共享对象实例）来做出这个确定。在一个实施方案中，标识共享闭包并确定共享闭包中的对象是否是可共享对象实例的操作 (854、856) 由图 6 中所示的过程 650 实施。

[0171] 如果共享闭包不可以在另一个运行时系统中使用（判断 858 的“否”分支），则过程 850 提出异常或者产生某种类型的否定指示。例如，如果共享闭包中的对象不都是可共享对象实例，则过程可以提出指示初始对象及其共享闭包不可共享的异常。

[0172] 如果共享闭包可以在其他的运行时系统中使用（判断 858 的“是”分支），则过程 850 调用使得所述共享闭包对于其他的运行时系统可用的机制。例如，如果通过使用共享存储器共享对象，则共享闭包可以被复制到共享存储器区域 (862)。在其他的实施方案中，通过使用消息或其他的通信方法，共享闭包可以被传送到一个或更多个运行时系统（例如其他的 VM）。

[0173] 创建共享闭包的过程也可能涉及将规定的名称或者其他标识符和共享闭包相关联 (864)。随后其他的运行时系统可以使用这样的标识符来标识要被访问的共享闭包。

[0174] 在某些实施方案中，创建共享闭包的过程还涉及使用版本管理 (versioning)。在过程 850 中，通过使用和存储在共享存储器中的共享闭包相关联的版本号实现版本管理。当利用给定名称创建共享闭包时，确定在共享存储器中是否已经存在具有那个名称的共享闭包。如果的确存在这样的共享闭包（判断 866 的“是”分支），则增加和所述共享闭包相关联的当前版本号 (868)，并且新的当前版本号被与新创建的共享闭包相关联 (872)。如果在共享存储器中不存在具有给定名称的共享闭包（判断 866 的“否”分支），则新的共享闭包的当前版本号被设置为指示第一个版本的数字（例如 0 或者 1）(870)，并且被与新创建的共享闭包相关联 (872)。

[0175] 可以使用版本管理来更新共享闭包——例如，可以在先前给予共享闭包的相同名称下创建该共享闭包的新的、更新过的版本。在一个实施方案中，当创建新版本的被命名共享闭包时，所有将该被命名的共享闭包与 VM 相关联的后续操作都使用新版本的共享闭包。已经在访问共享闭包的 VM（例如将先前版本的共享闭包映射到其地址空间中的 VM）不受新版本影响——它们简单地将所有对象引用保持在旧版本。在这个实施方案中，多个版本的共享闭包可以在共享存储器中共存，直到过时版本不再被任何 VM 引用因而可以被垃圾回收为止。

[0176] 图 9 示出了用于访问和使用共享闭包的范例过程的流程图。在过程 950 中，名称或其他标识符被接收 (952)，并被用来标识相关联的共享闭包 (954)。所述名称或者其他的标识符相应于在创建共享闭包时（例如在过程 850 中的操作 864 中）被与所述共享闭包相关联的标识符。在实施了版本管理的情况下，如果存在多于一个版本的指定共享闭包，则标识指定共享闭包的最近被创建的版本。

[0177] 然后将被标识的共享闭包与运行时系统（例如 VM）相关联 (956)。在一个实施方案中，可以用下面两种方式之一将共享闭包与运行时系统相关联——或者通过将共享闭包从共享存储器区域映射到运行时系统的地址空间中，或者通过将共享闭包从共享存储器区域复制到运行时系统的地址空间中。在共享闭包已经被与运行时系统相关联之后，可以使用正常操作（例如正常 Java 操作）访问共享闭包内的对象 (962)。

[0178] 在某些实施方案中，对共享闭包中对象的访问可能依赖于共享闭包如何被与运行时系统相关联。例如，在一个实施方案中，如果共享闭包被映射到 VM 的地址空间中（判断 958 的“被映射”分支），则对共享闭包中对象的访问被限于只读访问 (960)。由于这个限制，任何写共享闭包中的对象实例的成员变量的尝试将导致错误。为了防止 VM 通过例如利用对 VM 的专用堆的引用来覆盖共享对象实例中的引用成员变量“打破”共享对象实例，或者防止 VM 打破共享对象的内部一致性或功能，这个限制可能很有用。

[0179] 在另一方面，如果共享闭包被复制到 VM 的地址空间中（判断 958 的“被复制”分支），则准许 VM 对被复制的对象的完整读一写访问。在这样的实施方案中，因此可以通过将共享闭包复制到 VM 的地址空间中，修改共享闭包中对象的内容，然后在共享存储器中创建新版本的共享闭包（例如使用图 8 中所示的过程 850）来更新所述共享闭包中的对象。

[0180] 可以使用其他方法来将共享闭包与运行时系统相关联以及提供从运行时系统对共享闭包中对象的访问。例如，可以使用按需求复制方法。在一个这样的实施方案中，共享闭包被映射到 VM 的地址空间中而不将对共享闭包的访问限于只读访问。相反，对共享闭包的访问被监视，并且在检测到第一次对共享闭包企图的写访问时，将共享闭包复制到 VM 的地址空间中，从而将共享闭包从被映射的共享闭包变换为被复制的共享闭包。然后，允许企图的写访问完成，并且对被复制的共享闭包的后续读和写访问可以按正常那样继续下去。如果在发生从被映射共享闭包到被复制共享闭包的变换时共享闭包的堆地址改变了，则对堆的现有引用必须被重定向到共享闭包的新创建的复本。或者，可以使用底层 OS 特征，以允许 OS 提供按需要复制功能而无堆地址改变的方式映射共享闭包。

[0181] 除了用于创建、映射和复制共享闭包的函数以外，应用程序接口 (API) 可以包括额外的用于管理共享对象的函数。例如，API 也可以包括“删除”函数。“删除”函数的一个实施方案取名称或者其他的标识符作为输入参数，并将共享存储器中相关联的共享闭包标记为被删除。将共享闭包标记为被删除不影响已经在访问共享闭包的 VM，但是排除了随后试图访问共享闭包（例如通过将共享闭包映射或者复制到其地址空间中）的 VM 这么做。

[0182] 在服务器 200 的其中共享闭包可以被映射到 VM 的地址空间中的实施方案中，通过追踪将共享闭包映射到地址空间中的 VM 的号码，可以执行被删除或过时版本的共享闭包的垃圾回收。每次 VM 将共享闭包映射到其地址空间中时可以递增计数。在其自己的垃圾回收过程中，当 VM 确定它不再包括 对先前被映射的共享闭包的任何引用时，计数可以被递减。当和特定共享闭包相关联的计数达到零时，可以从共享存储器删除该共享闭包。

[0183] 在 Java 中，对象实例通常包括对对象实例的类的运行时表示的引用，并且类运行时表示依次包含对类的类加载器的引用。因此，与 Java 对象实例相关联的类加载器和运行时表示被包括在对象实例的共享闭包中，这意味着运行时表示和类加载器自身必须是可共享的，以便对象实例是可共享的。因此，在包括类运行时表示和类加载器的服务器实施方案中，可以使用两个额外的标准来确定特定的类是否可共享。所述类应该具有可共享运行时表示和可共享类加载器。

[0184] 可以使用各种技术来处理类运行时表示和类加载器（即使得类运行时表示和类加载器“可共享”）。一种技术涉及实际地共享类运行时表示和类加载器。即，当对象实例被复制到共享存储器时，相应于该对象实例的类的运行时表示和类加载器也被复制到共享存储器，以使它们可以被所有的 VM 访问（例如通过映射操作）。这个技术的各种优化是可

能的。例如，在将类的运行时表示复制到共享存储器中之前，可以检查共享存储器以便确定在共享存储器中是否已经存在该类的运行时表示——如果是，则在被复制到共享存储器中的对象实例中对运行时表示的引用可以被简单地设置为参考已经存在于共享存储器中的运行时表示。

[0185] 第二种用于处理运行时表示和类加载器的技术不是共享它们，而是保证它们在每一个 VM 中被置于固定的位置。换句话说，每一个类的运行时表示和类加载器在每一个 VM 中必须被置于相同的固定地址。然后，对每一个对象实例中运行时表示的引用可以被设置到相应于该对象实例的类的运行时表示的位置。利用这种方法，无论对象实例被从共享存储器映射还是复制到地址空间中，对运行时表示的引用都是有效的。

[0186] 但是，固定每一个类的运行时表示的位置可能不实用。因此，用于处理运行时表示和类加载器的第三种方法是在对象实例被复制到 VM 中时调整对每一个对象实例的运行时表示的引用。和前面的技术一样，在这种技术中不共享运行时表示和类加载器——即每一个 VM 为每一个类存储其自己的运行时表示和类加载器。但是，和前面的技术不同，这种技术不要求每一个运行时表示和类加载器的位置在每一个 VM 中固定。相反，运行时表示和类加载器的位置在每一个 VM 中可以不同。当对象实例被复制到特定 VM 中时，合 适的类运行时表示的位置被确定，并且对象实例中相应的引用被设置到那个位置。

[0187] 用于处理运行时表示的第三种技术——调整每一个 VM 中对运行时表示的引用——排除了将对象实例同时映射到多个 VM 的可能。这是因为如前所示，被共享的对象实例不能具有对任何专用堆的引用。由于第三种技术调整引用以便参考每一个 VM 中的专用运行时表示，所以该技术只能在对象被复制到 VM 的地址空间中时使用，或者在对象一次只被一个 VM 访问的其他情况中（例如在对象实例可以被“排它地”映射，所以当对象实例被映射到一个 VM 中时，没有其他的 VM 能够映射该对象实例的实施方案中）使用。

[0188] 上面讨论的第三种技术在集群架构中可能有用，在集群架构中，VM 可以在多个物理机器上执行。运行时表示一般不被跨越多个物理机器共享，所以当对象实例被跨越多个物理机器共享时（例如当在一个物理机器上的 VM 中正被使用的对象实例被传送到第二个物理机器以便在该机器上的 VM 中使用时），必须调整对运行时表示的引用。

[0189] 图 10 示出了用于跨越多个机器使用对象的技术。该技术是上面讨论的用于处理运行时表示的第三种技术的一般化版本，它包括在源运行时系统中可以执行的某些操作和在目标运行时系统中可以执行的某些操作的组合。图 10 中的过程 1000 示出了在源运行时系统中可以执行的操作的例子，并且过程 1050 示出了在目标运行时系统中可以执行的操作的例子。总的来说，该技术涉及产生和使用动态存根 (stub on the fly)——对运行时元数据的引用被源运行时系统中的代理 (proxy) 代替，并且代理依次被用对目标运行时系统中的运行时元数据的引用代替。

[0190] 更具体地说，在过程 1000 中，在源运行时系统中首先标识要被与目标运行时系统共享或者在目标运行时系统中使用的对象的集合 (1002)。例如，这个操作可能涉及标识源运行时系统中对象的共享闭包和验证该共享闭包中的所有对象都是可共享对象实例，如图 6 中的过程 650。但是，这个操作也可能非常简单——例如，可以选择单个对象在目标运行时系统中使用（但是如果选择了单个对象，则在被选择的对象中对源运行时系统中的一个或更多个被引用对象的任何引用在目标运行时系统中将不是有效的，除非做出特殊的安

排，例如通过将被引用的对象在两个运行时系统中置于相同地址，或者如上面描述的那样，调整被选择对象中的引用。）

[0191] 然后遍历被标识的对象集合中的对象。如果在对象集合中有更多的对象（判断1004的“是”分支），则取出下一个对象（1006），并且将该对象中对运行时元数据的任何引用（例如对对象类的运行时表示的引用）用代理代替（1008）。在所有对象已经被遍历之后（判断1004的“否”分支），对象集合被发送到目标运行时系统（1010）。当然，过程1000中的操作可以被以不同的顺序执行——例如，代理可以被放在对象中，并且对象可以在整个对象集合已被标识之前被发送到目标运行时系统。例如，这些操作可以作为标识共享闭包的递归过程的一部分被动态地执行。

[0192] 过程1050示出了在目标运行时系统中（例如在不同机器上执行的不同于源VM的目标VM中）可以使用的相应过程。数据对象集合被接收（1052），并被复制到目标运行时系统中（1054）。然后遍历接收到的对象集合中的对象。如果在接收到的对象集合中有更多的对象（判断1056的“是”分支），则取出下一个对象（1058）。然后，标识相应于该对象的相关运行时信息（1060）。这些信息可以包括例如对象的运行时类。然后，基于运行时信息，对象中的代理可以被用对目标运行时系统中的运行时元数据的引用代替（1066）。例如，一旦已经标识了对象的运行时类，则该对象中运行时表示的代理可以被用在目标运行时系统中对该类的运行时表示的引用代替。

[0193] 在某些情况中，在将对象中的一个或更多个代理用对目标运行时系统中的运行时元数据的引用代替之前，首先进行检查以便确定这些元数据是否已经被加载到目标运行时系统中。如果元数据还未被加载到目标运行时系统中（判断1062的“否”分支），则可以按需要加载元数据（1064）。例如，在将代理用对运行时表示的引用代替之前，可以调用类加载器将类的所述运行时表示加载到VM中。

[0194] 和过程1000一样，可以以不同的顺序执行过程1050中的操作——例如，当接收到对象时（即在已经接收到整个对象的集合之前），可以将代理用对运行时元数据的引用代替。进一步的变化也是可能的。例如，在接收到对象时或者在“按需要”的基础上（例如当对象在目标运行时系统中被首次访问时），代理可以被用对运行时元数据的引用代替。

[0195] 上面讨论的技术，包括共享闭包的创建和使用，以及对运行时元数据引用的处理，是将一个运行时系统中的复杂数据结构分解并以另一个运行时系统的本机格式重装所述数据结构的一般概念的特定实施方案。例如，这些技 术可用来在服务器环境中共享对象，在服务器环境中，复杂数据结构是堆（或其一部分，例如堆内的对象集合），并且运行时系统是VM。下面更详细地提供了使用上面的技术的例子。

[0196] 图11是用于在服务器中使用共享对象的范例过程1150的流程图。过程1150示出了共享对象可以怎样被结合可连接进程VM使用。在过程1150中，为用户会话创建VM(1152)，并且对象的共享闭包被创建并存储在共享存储器中（1154）。当服务器接收到相应于用户会话的用户请求时（1156），它从分配给服务器的进程池中选择可用进程（1158），并将为该用户会话创建的VM绑定到被选择的进程（1160）。

[0197] 如果VM不需要访问共享闭包中的对象（判断1162的“否”分支），则VM可以简单地处理请求（1164），在那之后VM可以从进程解除绑定（1166）。然后，进程可以被返回到可用进程池，并且服务器在其接收到新请求时（1156）可以再次将VM绑定到所述进程。

[0198] 在另一方面，如果 VM 的确需要访问共享闭包中的对象（判断 1162 的“是”分支），则共享闭包被与该 VM 相关联。可以用各种方法将共享闭包与 VM 相关联——例如，可以通过将共享闭包映射或者复制到被选择的进程的地址空间中将共享闭包绑定到被选择的进程。

[0199] 共享闭包与 VM 相关联的确切方式依赖于 VM 所需要的访问类型。如上所示，在一个服务器实施方案中，被映射的共享闭包总是被限于只读访问，以便防止映射 VM 通过将成员变量设置为在其他 VM 中无效的值而打破共享闭包中的共享对象实例。在这样的实施方案中，如果在过程 1150 中和用户会话相关联的 VM 只需要对对象的读访问（即如果 VM 不需要修改共享闭包中的任何对象，如判断 1168 的“否”分支所指示的那样），则共享对象可以被映射到被选择的进程的地址空间中 (1170)。然后，VM 可以处理请求 (1164)，按需要从共享对象读取信息。

[0200] 如果 VM 需要对共享对象的读 - 写访问（即如果 VM 需要修改一个或更多个对象，如判断 1168 的“是”分支所指示的那样），则共享闭包被复制到被选择的进程的地址空间中 (1172)。然后 VM 具有对共享对象完整的读 - 写访问，并且它能够按需要处理请求并修改对象 (1174)。当 VM 完成修改共享对象时，它能够将共享闭包复制回共享存储器 (1176)。如先前所指示的那样，在一个服务器实施方案中，通过重新创建共享闭包（例如通过调用“创建” 函数，将先前使用过的同一名称分配给共享闭包，并且如果在共享存储器中仍存在具有那个名称的旧版本共享闭包，则创建新版本的共享闭包），可以将共享闭包复制到共享存储器。和前面一样，在请求已被处理以后，VM 可以被从进程解除绑定 (1166)，进程可以被返回可用进程池，并且服务器能够再次等待接收新请求 (1156)。

[0201] 共享闭包中的对象可以包括任何类型的对象，包括存储用户上下文信息的对象，如下面的例子中所讨论的那样。而且，尽管过程 1150 只示出了一个访问共享闭包的 VM，但是共享闭包也可以被其他的 VM 访问（例如，其他的 VM 可以映射或复制该共享闭包，从而访问该共享闭包中的对象）。

[0202] 共享虚拟机

[0203] 图 2 和图 3 中所示的服务器实施方案很适于用户会话相应于较大的用户上下文的环境。但是，这些实施方案在用户上下文相当小（例如均小于 10 千字节）的环境中可能不是最佳的。即使具有在 VM 之间共享对象的能力，在这种环境中为每一个用户会话分配单独的 VM 可能也不实用，因为它导致每用户会话巨大的开销。

[0204] 图 12 示出了服务器 200 的不同的实施方案，所述实施方案在预期用户上下文相当小的环境中使用可能更实用。不是为每一个用户会话创建一个 VM，图 12 中所示的服务器 200 的实施方案使用共享 VM（例如 VM301、303、305、307 和 309）的池 300。服务器 200 为每一个用户会话创建用户上下文（例如用户上下文 501、503、505、507、509、511、513、515 和 519）。用户上下文被存储在共享存储器区域 255 中，VM 池 300 中的 VM 也是一样。和先前所讨论的实施方案一样，服务器 200 使用工作进程（例如工作进程 401、403、404 和 409）的池 400 和分派器进程 410。

[0205] 在工作中，图 12 中所示的服务器 200 的实施方案如下工作。当服务器 200 接收到用户请求时，分派器进程 410 从工作进程池 400 选择可用的工作进程（例如工作进程 404），并将用户请求分派给被选择的工作进程 404。然后，工作进程 404 从 VM 池 300 选择可用的 VM（例如 VM305）。然后，通过将 VM305 从共享存储器 255 映射到被选择的工作进程 404 的

地址空间中，将被选择的 VM305 绑定到被选择的工作进程 404。

[0206] 接着，工作进程 404 标识相应于用户请求的用户会话和相关联的用户上下文（例如用户上下文 501）。然后，例如通过将用户上下文 501 从共享存储 器区域 255 复制到工作进程 404 的地址空间中，将被标识的用户上下文 501 绑定到工作进程 404。现在，通过联合用户上下文 501 在工作进程 404 中执行 VM305，可以处理用户请求。

[0207] 为了标识相关的用户上下文和将该上下文与被选择的 VM 相关联，其他的实施方案是可能的。例如，分派器进程 410 能够标识相关的用户上下文并将标识该上下文的信息传递到被选择的 VM，然后，该用户上下文（例如构成该上下文的对象的共享闭包）可以被映射或者复制到被选择的工作进程的地址空间中，所述被选择的工作进程代表被选择的 VM。在概念上，这可以被看作这样的次序，其中：被标识的用户上下文被插入被选择的 VM，然后，被选择的 VM 被插入被选择的工作进程，尽管从技术角度，可以通过简单地将代表被选择的 VM 和被标识的用户上下文的共享存储器部分映射或者复制到被选择的工作进程的地址空间中来完成这个概念步骤。

[0208] 当请求已经被处理之后，VM305 和用户上下文 501 被从工作进程 404 解除绑定（例如通过将 VM305 从工作进程 404 解除映射，以及将用户上下文 501 复制回共享存储器区域 255）。然后，VM305 和工作进程 404 被再次标记为可用，并且它们可被用来处理服务器 200 接收到的额外请求。当然，VM305 和工作进程 404 在处理新请求时可能不配对在一起——例如，如果新请求被分派给工作进程 404 并且 VM305 忙于处理另一个请求，则工作进程 404 必须从 VM 池 300 选择另一个 VM（或者等待 VM 池 300 中的 VM 变得可用）。

[0209] 以这种方式，在图 12 中所示的服务器实施方案中的 VM 被共享（即服务器不是每用户会话分配一个 VM），却彼此隔离，因为每一个 VM 一次只专用于一个用户会话。而且，由于每一个 VM 均在隔离的进程中运行，如果 VM 崩溃了，则这样的崩溃可能只影响相应于正在崩溃的 VM 中被处理的请求的用户会话（依赖于操作系统将进程彼此隔离得有多好）。因此，这种使用共享但专用 VM 池的方法导致了稳健的服务器，但是具有比先前讨论的实施方案更少的开销。

[0210] 在为每一个用户会话分配一个 VM 的服务器实施方案中，可用使用各种优化来实现相同的益处——减小的开销。例如，在 VM 相应的用户会话末尾 VM 不是被终结，而是可以重用所述 VM。即，VM 可被与不同的用户会话相关联并用来处理相应于新的用户会话的请求。尽管有帮助，但是在存在大量 同时发生的用户会话时，这种方法仍要求巨大的开销，因为服务器需要为每一个用户会话实例化至少一个 VM。

[0211] 相反，图 12 中所示的服务器实施方案将 VM 的数量限制为有多少 VM 被分配给 VM 池 300。如先前所讨论的那样，这个实施方案很适于具有大量同时发生的用户会话和较小的用户上下文的环境。这样的环境一般可用由方程 P ＜＝ V ＜＜ U 刻画，其中，P 是分配给工作进程池 400 的工作进程的数量，V 是分配给 VM 池的 VM 的数量，并且 U 是正被服务器处理的用户会话的数量。这个方程的左边部分指示，分配给 VM 池的 VM 的数量一般应该大于或等于工作进程池中的工作进程的数量（因为否则即使所有的 VM 都在被使用，也将存在某些空闲的未使用进程）。方程的右边部分指示，VM 的数量可用远远小于用户会话的数量。再一次地，这是可能的，因为每一个 VM 可以被不同的用户会话共享，尽管不是在同一时间。

[0212] 通过在 VM 之间共享对象实现了图 12 中所示的服务器 200 的实施方案。在图 3 中

所示的服务器实施方案中，每一个用户上下文（例如用户上下文 301）被与一个 VM（例如 VM501）相关联了相应的用户会话的寿命那么长时间，并且，当从相应的用户会话接收到用户请求时，该用户上下文和其相关联的 VM 都被映射到工作进程中。因此，在图 3 中的实施方案中，每一个 VM 被与一个用户会话相关联，并且 VM 只能访问相关联的用户上下文——VM 不可能访问相应于另一个会话的用户上下文。例如，VM301 只能访问用户上下文 501，并且 VM303 只能访问用户上下文 503。因此，每一个 VM 只能处理相应于其相关联的用户上下文的请求。

[0213] 但是，上面讨论的共享技术使得 VM 有可能共享对象，包括构成用户上下文的对象。例如，用户上下文可以被作为共享闭包存储在共享存储器堆中。然后，共享闭包——因而用户上下文——可以被多个 VM 访问。

[0214] 因此可以如下描述图 12 中所示的服务器实施方案：首先，将 VM 和用户上下文与彼此解除关联；其次，用户上下文被存储在可被 VM 池中的所有 VM 访问的共享存储器堆中；最后，通过将相应的用户上下文和可用的 VM 映射到可用工作进程中来处理用户请求。

[0215] 在图 13 中概念性地示出了上面的方案。如图 13 中那样，图 13 中的 VM 和用户上下文被存储在共享存储器区域 255 中。但是，和图 3 中不同，图 13 中的 VM 不再被与特定用户上下文相关联——任何可用的 VM 均能和任何用 户上下文一起工作。例如，在图 3 中，VM301 只能与其相关联的用户上下文 501 一起工作。相反，在图 13 中，VM301 能够与共享存储器中的任何一个用户上下文一起工作，例如用户上下文 519。

[0216] 在某些实施方案中，一旦 VM 已经和一个用户上下文配对，则该 VM 必须和该用户上下文保持配对，直到相应的用户请求已经被处理为止。具体来说，在处理当前用户请求时，即使 VM 当前被阻塞（例如因为它正等待 I/O 事件），该 VM 也不能被标记为可用并被用来处理另一个用户请求。

[0217] 例如，在 Java 实施方案中，在请求正被处理时，不容易将 VM（它包括 VM 堆和 VM 堆栈）从用户上下文（它包括用户堆和用户堆栈）分开，因为 VM 堆栈被与用户堆栈混合在一起。因此，VM 必须和该用户上下文保持配对，直到相应的请求被处理并且用户堆栈变空为止。

[0218] 图 14 根据上面的方案示出了 VM 和用户上下文的配对。在服务器接收到相应于用户上下文 501 的请求之前，用户上下文 501 被存储在共享存储器中，并且 VM301 可用于处理相应于任何用户上下文的请求。当接收到相应于用户上下文 501 的请求时 (1402)，用户上下文 501 和 VM301 都被绑定到工作进程 401。因此，当 VM301 利用用户上下文 501 在工作进程 401 中运行时，用户上下文 501 和 VM301 被与彼此配对。

[0219] 在处理请求时的某一点，VM 中所有的线程或者协同例程可以阻塞（例如，如果它们都在等待 I/O 事件完成）。当它发生时 (1404)，VM301 和用户上下文 501 被从工作进程 401 解除绑定，从而释放工作进程 401 来作用于另一个请求。但是，VM301 未被标记为可用——它和用户上下文 501 保持配对，并保持空闲，直到至少一个线程能够再次运行为止（例如当被指示的 I/O 事件完成时）。当它发生时 (1406)，VM301 和用户上下文 501 能够被再次绑定到工作进程，以便 VM 能够继续利用该用户上下文执行。VM301 和用户上下文 501 可以不被绑定到它们先前被绑定到的同一工作进程 401——例如，如果另一个 VM 当前被绑定到工作进程 401，则 VM301 和用户上下文 501 被绑定到另一个可用的工作进程。

[0220] 最后，当请求已被处理并且响应被发送时 (1408)，VM301 和用户上下文 501 被从工作进程解除绑定。用户上下文 501 被再次存储在共享存储器中，并且 VM301 和工作进程都被标记为可用，以便作用于另一个请求。

[0221] 图 15 是示出用于在图 12 中所示的服务器实施方案中处理请求的范例过 程 1550 的流程图。在过程 1550 中，操作系统进程池和 VM 池被首先初始化 (1552)。为每一个用户会话创建用户上下文，并且所述用户上下文被存储在服务器中的共享存储器区域中 (1554)。例如，每一个用户上下文可以被作为对象的共享闭包存储。

[0222] 当服务器从用户会话接收到请求时 (1556)，它标识相应的用户上下文 (1558)，并从工作进程池选择可用进程 (1560) 以及从 VM 池选择可用 VM(1562)。然后，被选择的 VM 被绑定到被选择的进程 (1564)(例如，通过将 VM 从共享存储器区域映射到被选择进程的地址空间中)。在 VM 可以被用来处理请求之前，用户上下文必须被与 VM 相关联 (1566)。例如，这可以通过将用户上下文绑定到被选择的进程完成。如上所述，在一个实施方案中，如果共享闭包被复制到了进程中，则对该共享闭包中对象的访问不受限制，但是，如果该共享闭包被映射到进程中，则被限于只读访问。在这样的实施方案中，用户上下文被复制而不是映射到进程中，因为 VM 在处理请求的过程中可能需要修改用户上下文。但是，其他的实施方案是可能的——例如，用户上下文可以被映射到进程中，并且可以设置锁定变量，以便防止其他的进程或者 VM 在映射进程中的 VM 使用并可能修改那些对象时，访问该用户上下文中的对象。为了给予被选择的 VM 和进程对用户上下文的排他访问，额外的变化是可能的。

[0223] 一旦被选择的 VM 已经被绑定到被选择的进程并且用户上下文已经被与 VM 相关联，则通过在进程中利用用户上下文执行 VM 可以处理请求 (1568)。如上所述，在一个实施方案中，如果由于某个原因 VM 阻塞并且不能继续处理 (例如，如果 VM 正等待 I/O 事件发生)，则 VM 可以被从工作进程解除绑定。然后，该工作进程可以被标记为可用，并被用来作用于另一个请求。该 VM 也可以被标记为可用并被用来处理另一个请求，除非直到已经完成用户请求的处理才可能将 VM 和用户上下文解除关联。在后一情况下，VM 必须和用户上下文保持配对。当 VM 不再被阻塞时，VM 和用户上下文可以再次被绑定到可用工作进程，并且 VM 能够继续处理用户请求。

[0224] 当用户请求已被处理时，将 VM 绑定到工作进程和将用户上下文和 VM 相关联的操作可以被逆转——例如，用户上下文可以被和 VM 解除关联 (1570)，并且 VM 可以被从工作进程解除绑定 (1572)。用户上下文被和 VM 解除关联的方式依赖于用户上下文被与 VM 相关联的方式。如上所述，在一 个实施方案中，用户上下文被复制到被绑定到该 VM 的进程中，在这种情况下，通过将用户上下文复制回共享存储器区域，可以将用户上下文和 VM 解除关联。还是和先前所描述的一样，在某些实施方案中，如果现有版本的用户上下文仍存在于共享存储器区域中，则将用户上下文复制到共享存储器区域中可能导致创建新版本的用户上下文。和将 VM 绑定到工作进程一样，将 VM 从工作进程解除绑定也可以是简单的低成本操作。可以简单地将共享存储器的 VM 块从工作进程的地址空间解除映射。

[0225] 在用户上下文已经被和 VM 解除关联并且 VM 已经被从工作进程解除绑定以后，VM 和工作进程都可以被标记为可用，并被服务器用来处理其他的请求。

[0226] 过程 1550 中的操作可以被按不同的顺序 (例如，可用进程的选择可以发生在可用 VM 的选择之前、之后或者甚至同一时间) 并由服务器的不同部件执行。例如，分派器进

程（例如图 12 中的分派器进程 410）可以选择可用工作进程，并简单地将用户请求传送到被选择的进程，由进程决定选择可用 VM，将 VM 绑定到进程，标识相应于用户请求的用户上下文，并将用户上下文和被选择的 VM 相关联。或者，分派器进程可以自己执行那些操作中的一些——例如，分派器进程能够选择可用的 VM，标识相关的用户上下文，并将被选择的 VM 和被标识的用户上下文的标识发送到工作进程。其他的选择是可能的，其中，分派器进程、工作进程或者服务器中额外的部件执行这些操作。

[0227] 上面描述的技术，包括共享对象和用户上下文，也可以在其他的环境中使用。例如，可以在 VM 未被连接到进程并从进程分离的服务器实施方案中使用这些技术。图 16 是示出用于在这种服务器中处理请求的范例过程 1650 的流程图。在过程 1650 中，工作进程池被分配给服务器，并且在每一个工作进程中初始化 VM(1652)。在 VM 的寿命期内，每一个 VM 被绑定到其工作进程，意味着该 VM 不能被从该工作进程分离并被连接到另一个工作进程。

[0228] 和在过程 1550 中一样，为每一个用户会话创建用户上下文，并将其存储在服务器中的共享存储器区域中 (1654)。当服务器从用户会话接收到请求时 (1656)，它标识相应的用户上下文 (1658)，并从工作进程池选择可用的进程 (1660)。然后，用户上下文被和被选择的进程中的 VM 相关联 (1666)（例如，通过将用户上下文绑定到被选择的进程）。然后，通过利用用户上下文在 被选择的进程中执行 VM 可以处理请求 (1668)。当用户请求已经被处理时，用户上下文可以被和 VM 解除关联 (1670)（例如，通过将用户上下文从被选择的进程复制回共享存储器区域）。然后，工作进程（及其相关联的 VM）可以被标记为可用，并被服务器用来处理其他的请求。

[0229] 从图 15 和图 16 的比较可以看出，过程 1550 和 1650 类似，只不过后一过程不包括将 VM 绑定到工作进程和将其从工作进程解除绑定的操作。尽管过程 1650 包括更少的操作，但是实施该过程的服务器——即其中 VM 不被连接到工作进程并从工作进程分离的服务器——不太可能和使用可连接进程 VM 的服务器一样有效率，因为在前一服务器中当 VM 阻塞时，VM 在其中执行的工作进程不能被用来处理其他的请求。相反，在具有可连接进程 VM 的服务器中，当 VM 阻塞时，该 VM 可以被从其工作进程分离，并且另一个 VM 可以被连接到它的位置并被用来作用于另一个请求。

[0230] 本说明书中描述的技术的各种替代和优化是可能的。例如，通过将所有 VM 公有的数据存储在共享存储器中，并将这些数据映射到所有的工作进程中，可以进一步改善资源消耗和性能。如果类型信息（例如被加载的 Java 类的运行时表示）被以这种方式共享，则针对每一个 VM 用于类加载、验证和解析的开销可以降低。

[0231] 又例如，可以使用另外的机制来共享数据和信息（既包括 VM 也包括用户上下文）——例如，可以通过使用文件或消息系统而非共享存储器来共享信息。但是，这样的机制可能不像共享存储器一样令人期望，因为它们可能需要更复杂或更昂贵的操作来使得要被共享的信息持久和非持久。

[0232] 又例如，为了获得期望的结果，这里描述的过程不要求示出的特定次序或者顺序。例如，在图 15 和图 16 中绘出的过程中，创建、标识和将用户上下文与 VM 相关联的操作可以被在整个过程内很多不同的地方执行。在这里所描述的过程的某些实施方案中，多任务和并行处理可能更佳。

[0233] 而且，可以使用各种部件或者实体来执行这里所描述的过程中的操作。例如，过程 1550 中的操作可以由服务器内不同的部件执行。例如，在一个实施方案中，分派器进程（例如图 12 中的分派器进程 410）选择可用的工作进程，并简单地将用户请求传送到被选择的进程，由进程决定选择可用的 VM，将 VM 绑定到进程，标识相应于用户请求的用户上下文，并将用户上下文和被选择的 VM 相关联。或者，分派器进程可以自己执行那些操作中的一些—— 例如，分派器进程能够选择可用的 VM，标识相关的用户上下文，并将被选择的 VM 和被标识的用户上下文的标识发送到工作进程。其他的选择是可能的，其中，分派器进程、工作进程或者服务器中额外的部件执行这些操作。例如，没有分派器进程就可以实施服务器，并且可以使用另一种机制在工作进程之间分布请求。

[0234] 本发明以及本说明书中描述的所有功能操作，可以被实施在数字电子电路中或计算机软件、固件或硬件中，或者被实施在其组合中，所述计算机软件、固件或硬件包括在本说明书中公开的结构装置及其结构等同物。本发明可以被实施为一个或更多个计算机程序产品，即一个或更多个有形地具体实施在信息载体中的计算机程序，例如，具体实施在机器可读存储设备中或者在供数据处理装置执行或控制其操作的传播信号中，该数据处理装置例如是可编程处理器、计算机或多个计算机。计算机程序（也被称为程序、软件、软件应用或代码）可以用任何形式的编程语言书写，包括汇编或解释语言，并且它可以被以任何形式部署，包括作为独立程序或作为模块、部件、子例程或其他适于在计算机环境中使用的单元。计算机程序不必对应于文件。程序可被存储在文件的一部分中，该文件可以具有其它程序或数据，程序可以被存储为专用于所关心的程序的单一文件，或者，程序可以被存储为多个协调的文件（例如存储一个或多个模块、子程序或代码部分的多个文件）。计算机程序可以被部署成在一个或多个计算机上执行，所述计算机在一个地点或跨过多个地点分布，并通过通信网络互连。

[0235] 这里所描述的过程和逻辑流，包括本发明的方法步骤，可以由一个或更多个可编程处理器执行，所述处理器执行一个或更多个计算机程序，以便通过操作输入数据并产生输出来执行本发明的功能。所述过程和逻辑流也可以由专用逻辑电路执行，并且本发明的装置可以被实施为专用逻辑电路，所述专用逻辑电路例如 FPGA（现场可编程门阵列）或 ASIC（专用集成电路）。

[0236] 例如，适于计算机程序的执行的处理器包括通用和专用微处理器，以及任何种类的数字计算机的任何一个或更多个处理器。通常，处理器将从只读存储器或随机访问存储器或这两者接收指令和数据。计算机的必要元件是用于执行指令的处理器和用于存储指令和数据的一个或更多个存储器设备。通常，计算机也将包括一个或更多个海量存储设备用于存储数据，或者计算机将被可操作地耦合成从所述海量存储设备接收数据或将数据转移到所述海量 存储设备，或者进行这两者，所述海量存储设备例如是磁盘、磁光盘或光盘。适于具体实施计算机程序指令和数据的信息载体包括所有形式的非易失存储器，例如包括半导体存储器，如 EPROM、EEPROM 和快闪存储器器件；例如内置硬盘或可移动盘的磁盘；磁光盘；和 CD ROM 和 DVD ROM 盘。处理器和存储器可以被专用逻辑电路补充或包含在专用逻辑电路中。

[0237] 本发明可以被实施在计算系统中，所述计算系统包括后端部件（例如数据服务器），或者所述计算系统包括中间件部件（例如应用服务器）；或者所述计算系统包括前端

部件（例如具有图形用户界面或网络浏览器的客户端计算机，用户可以通过所述图形用户界面或网络浏览器和本发明的实施方案进行交互作用），或者，所述计算系统可以包括这些后端、中间件或前端部件的任意组合。系统的部件可以通过任何数字数据通信的形式或介质互连，例如通信网络。通信网络的例子包括局域网（“LAN”）和广域网（“WAN”），例如国际互联网。

[0238] 计算系统可以包括客户端和服务器。客户端和服务器通常彼此距离遥远，并且一般通过通信网络交互作用。通过在各自计算机上运行并且彼此具有客户端－服务器关系的计算机程序产生了客户端和服务器的关系。

[0239] 已经按照具体实施例描述了本发明，但是其他的实施例可以被实施并处于所附权利要求书的范围内。例如，如上所述，本发明的操作可以被按不同的次序执行并仍获得期望的结果。其他的实施例处于所附权利要求书的范围内。

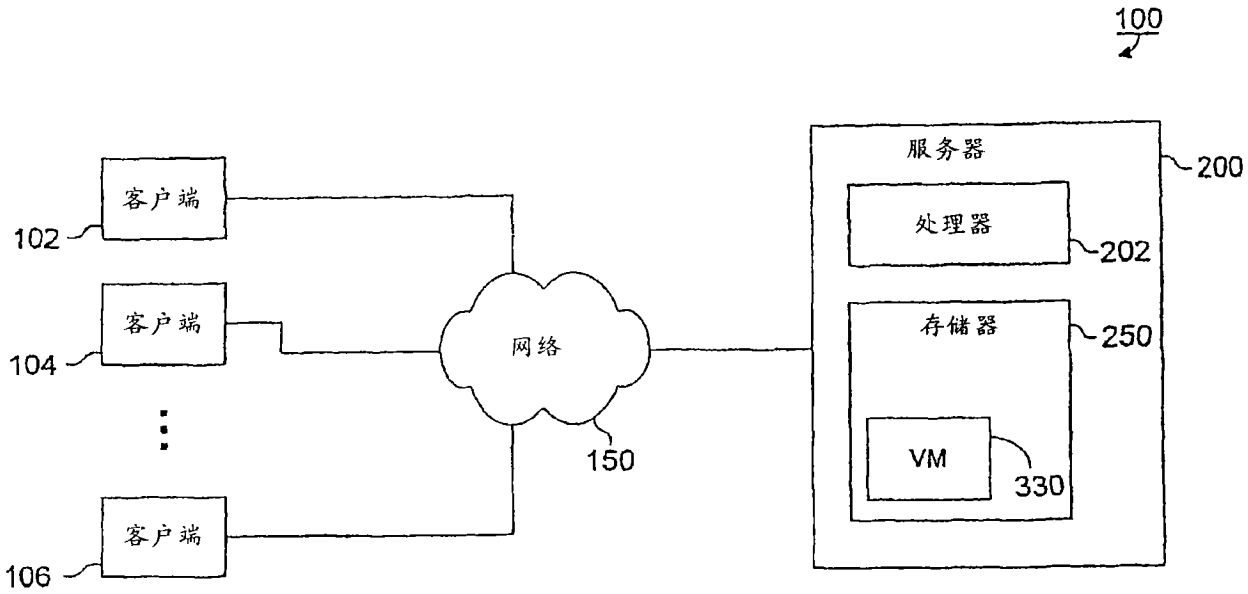


图 1

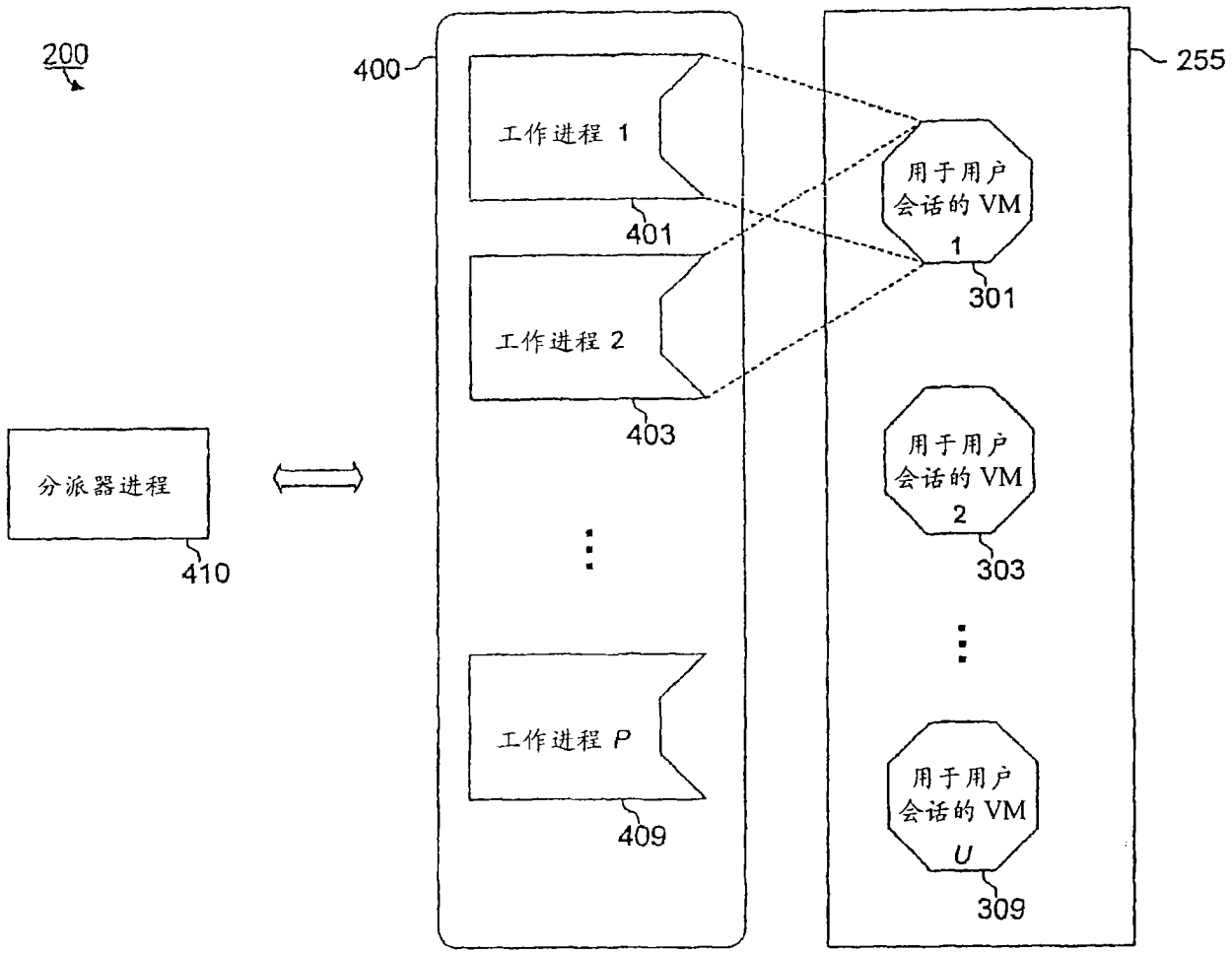

200
400
255
工作进程 1
401
工作进程 2
403
工作进程 P
409
用于用户会话的 VM 1
301
用于用户会话的 VM 2
303
用于用户会话的 VM U
309
分派器进程
410

图 2

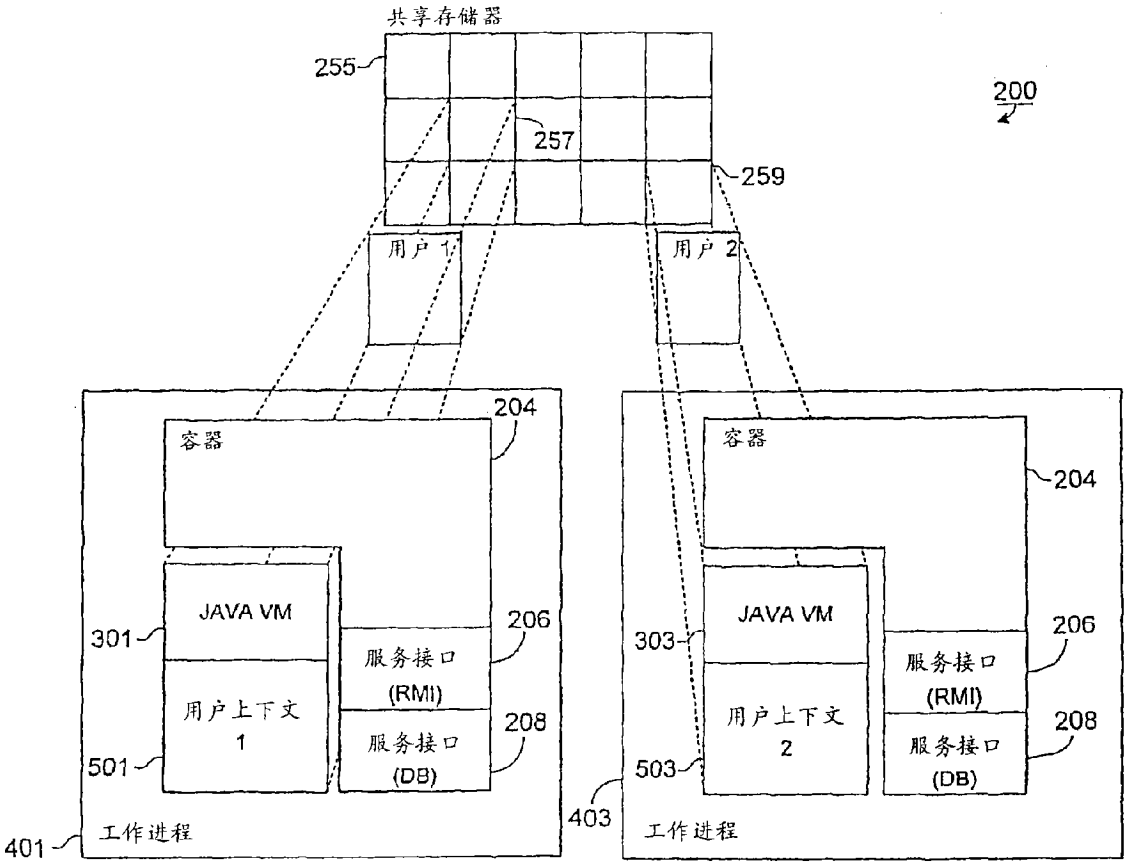

共享存储器
255
257
259
200
用户 1
用户 2
容器
204
JAVA VM
301
用户上下文
1
501
服务接口
(RMI)
206
服务接口
(DB)
208
工作进程
401
容器
204
JAVA VM
303
用户上下文
2
503
服务接口
(RMI)
206
服务接口
(DB)
208
403
工作进程

图 3

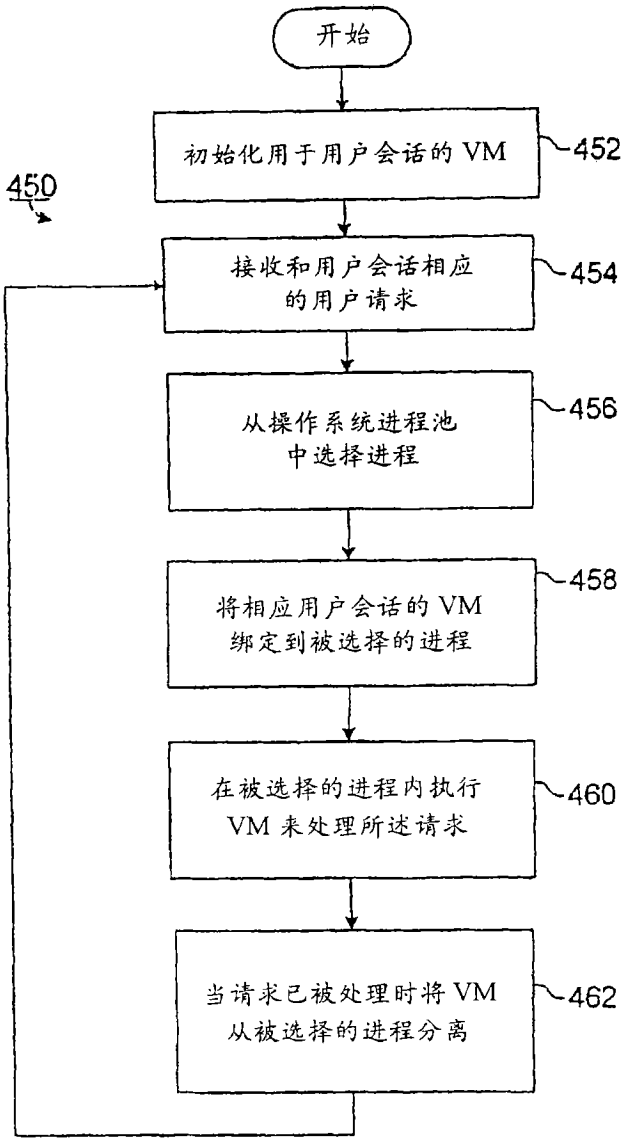

开始
初始化用于用户会话的 VM
452
450
接收和用户会话相应的用户请求
454
从操作系统进程池中选择进程
456
将相应用户会话的 VM 绑定到被选择的进程
458
在被选择的进程内执行 VM 来处理所述请求
460
当请求已被处理时将 VM 从被选择的进程分离
462

图 4

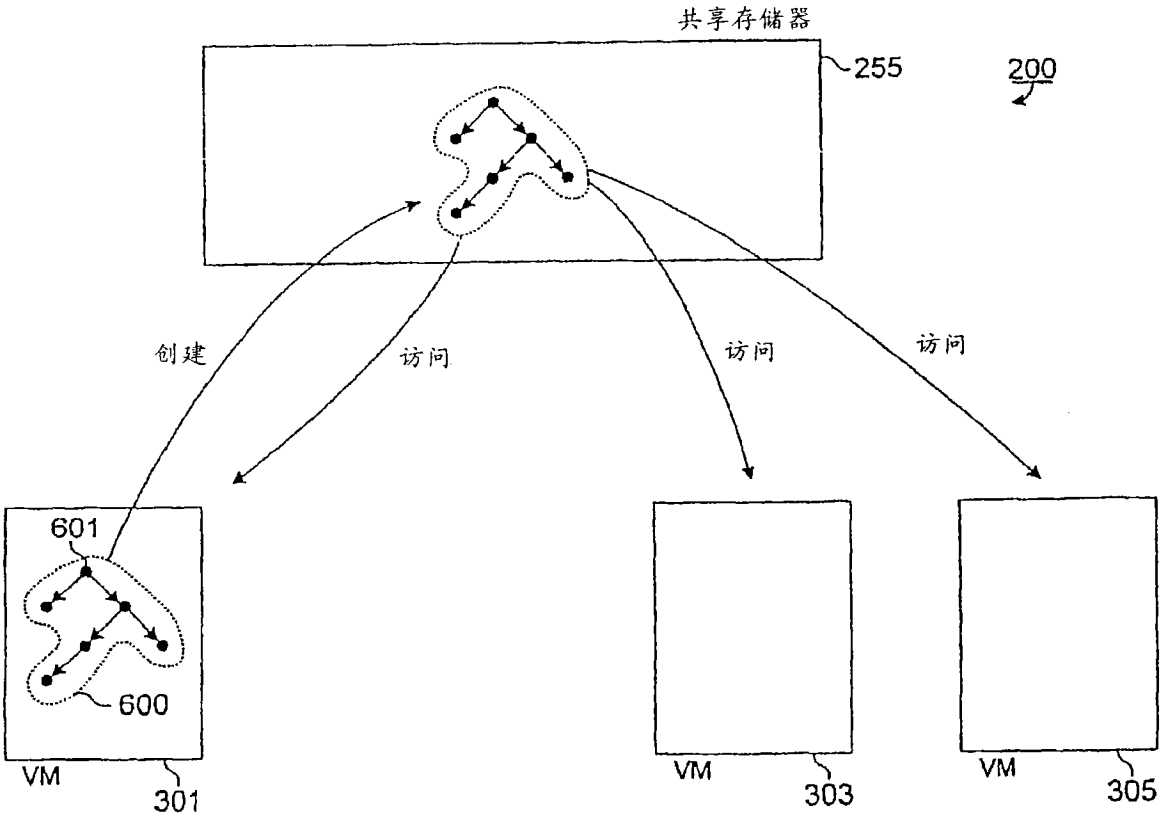


图 5

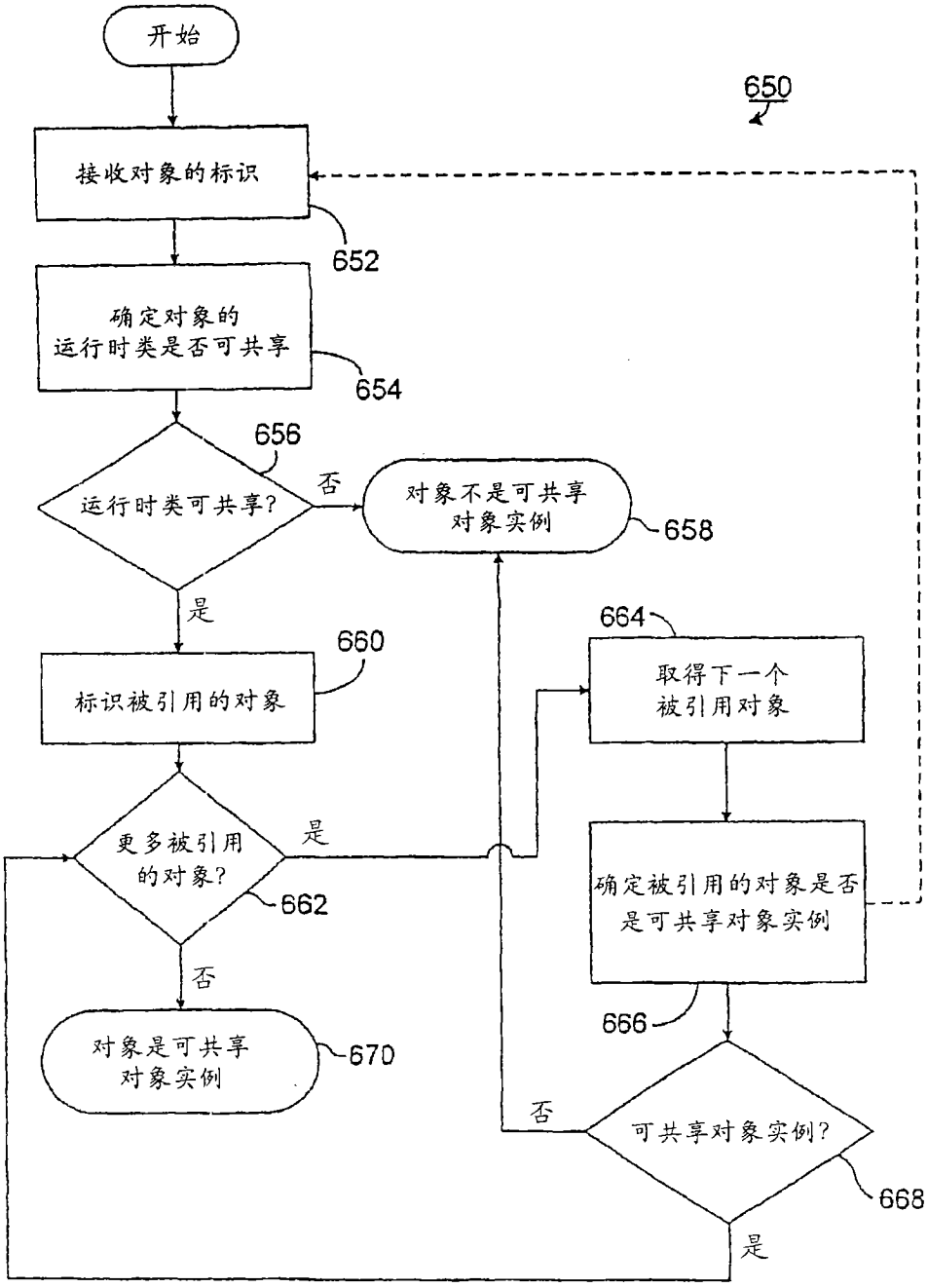

开始
650
接收对象的标识
652
确定对象的
运行时类是否可共享
654
656
运行时类可共享?
否
对象不是可共享
对象实例
658
是
660
标识被引用的对象
664
取得下一个
被引用对象
更多被引用
的对象?
是
662
确定被引用的对象是否
是可共享对象实例
否
666
对象是可共享
对象实例
670
否
可共享对象实例?
668
是

图 6

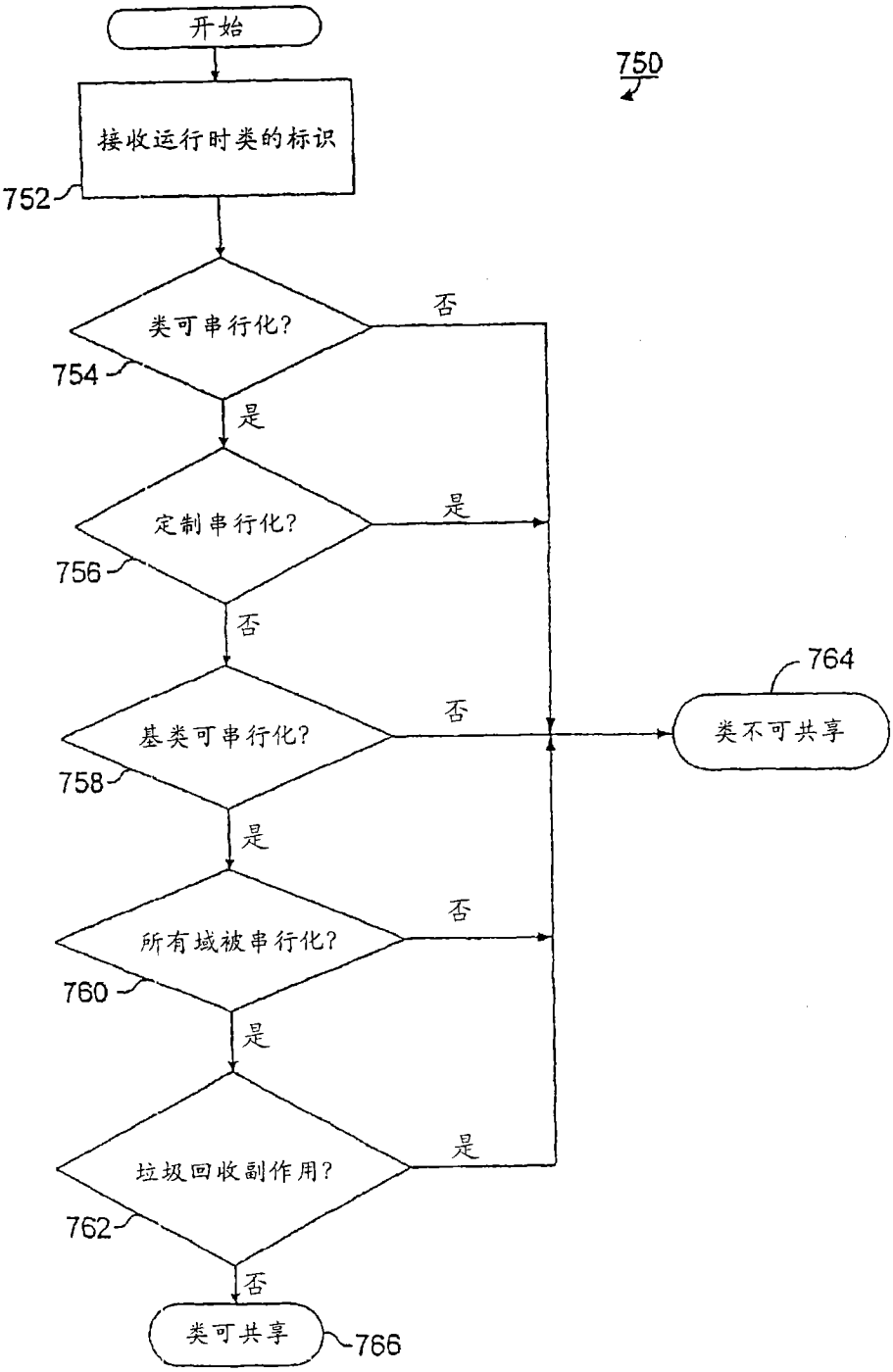

开始
750
接收运行时类的标识
752
类可串行化?
754
否
是
定制串行化?
756
是
否
基类可串行化?
758
否
764
类不可共享
是
所有域被串行化?
760
否
是
垃圾回收副作用?
762
是
否
类可共享
766

图 7

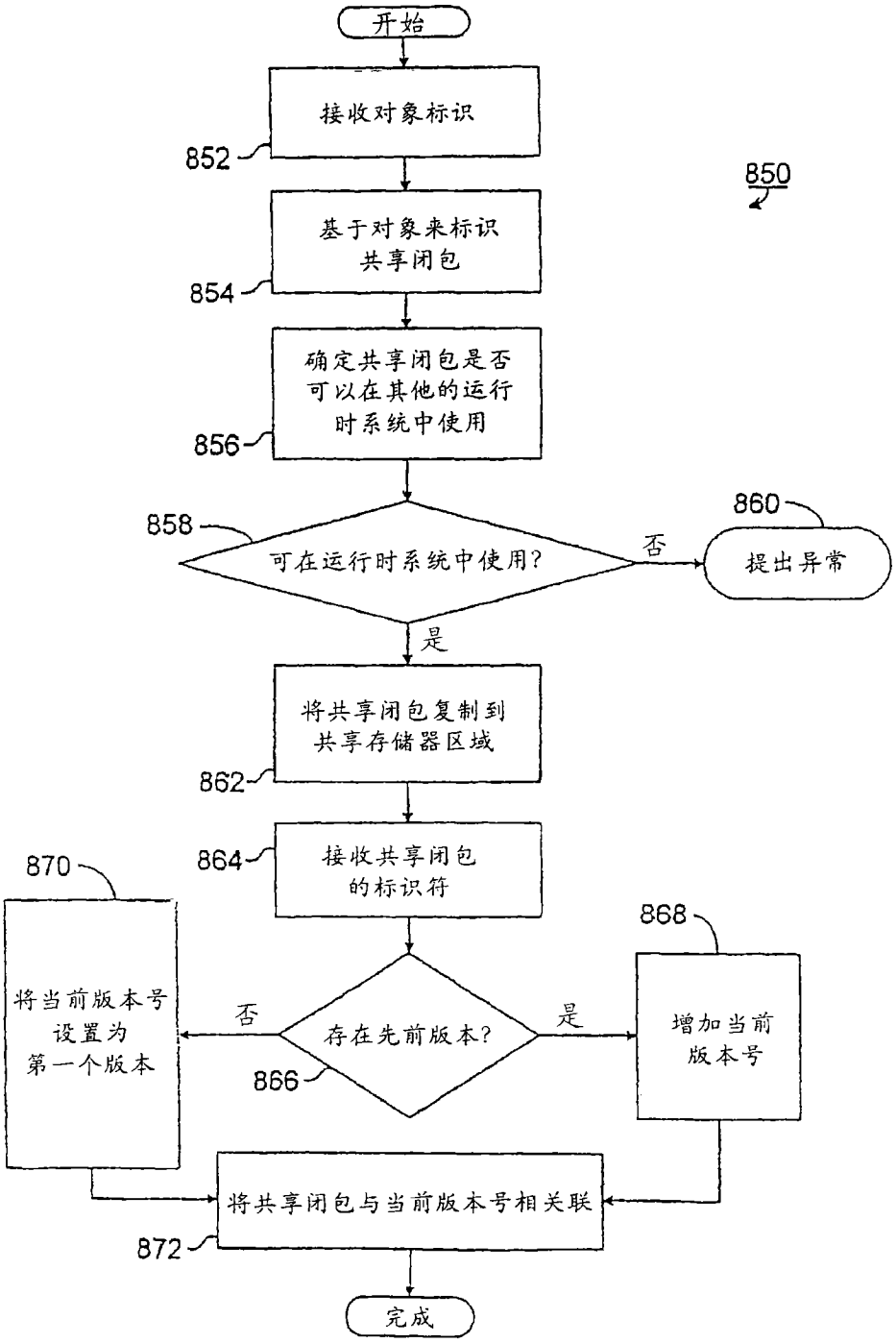

开始
接收对象标识
852
850
基于对象来标识
共享闭包
854
确定共享闭包是否
可以在其他的运行
时系统中使用
856
858
可在运行时系统中使用?
否
860
提出异常
是
将共享闭包复制到
共享存储器区域
862
接收共享闭包
的标识符
864
870
将当前版本号
设置为
第一个版本
否
存在先前版本?
866
是
868
增加当前
版本号
将共享闭包与当前版本号相关联
872
完成

图 8

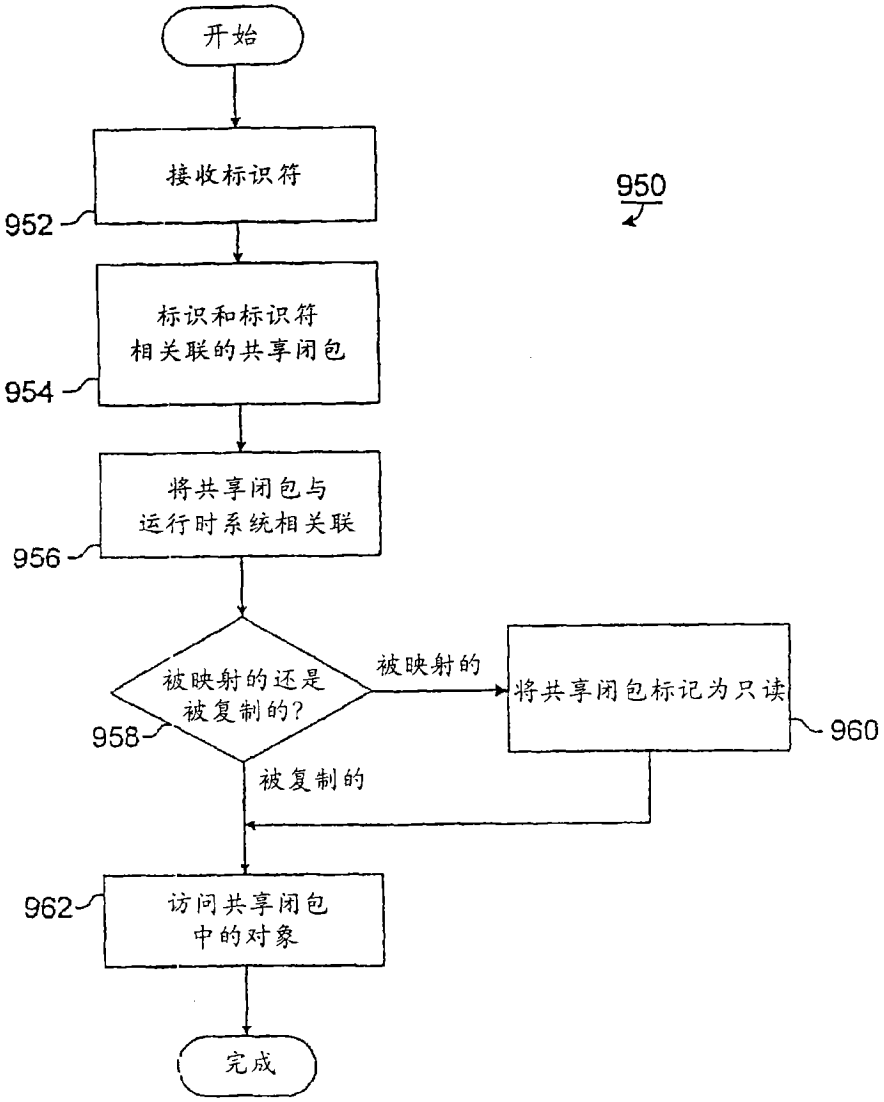

开始
接收标识符
952
950
标识和标识符
相关联的共享闭包
954
将共享闭包与
运行时系统相关联
956
被映射的还是
被复制的?
958
被映射的
将共享闭包标记为只读
960
被复制的
访问共享闭包
中的对象
962
完成

图 9

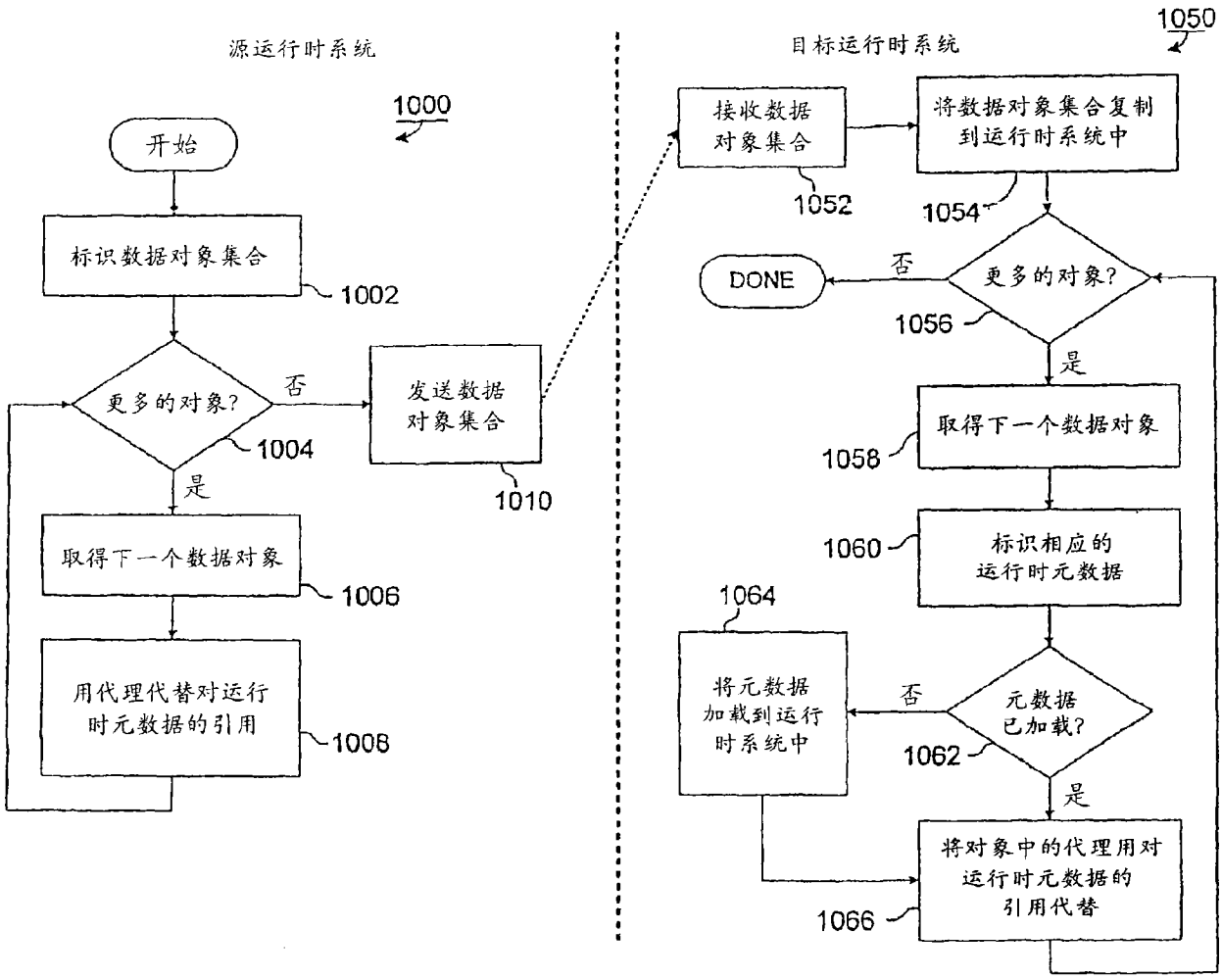

源运行时系统
目标运行时系统
1050
1000
开始
标识数据对象集合
1002
更多的对象?
否
1004
是
发送数据对象集合
1010
取得下一个数据对象
1006
用代理代替对运行时元数据的引用
1008
接收数据对象集合
1052
将数据对象集合复制到运行时系统中
1054
DONE
否
更多的对象?
1056
是
取得下一个数据对象
1058
标识相应的运行时元数据
1060
1064
将元数据加载到运行时系统中
否
元数据已加载?
1062
是
将对象中的代理用对运行时元数据的引用代替
1066

图 10

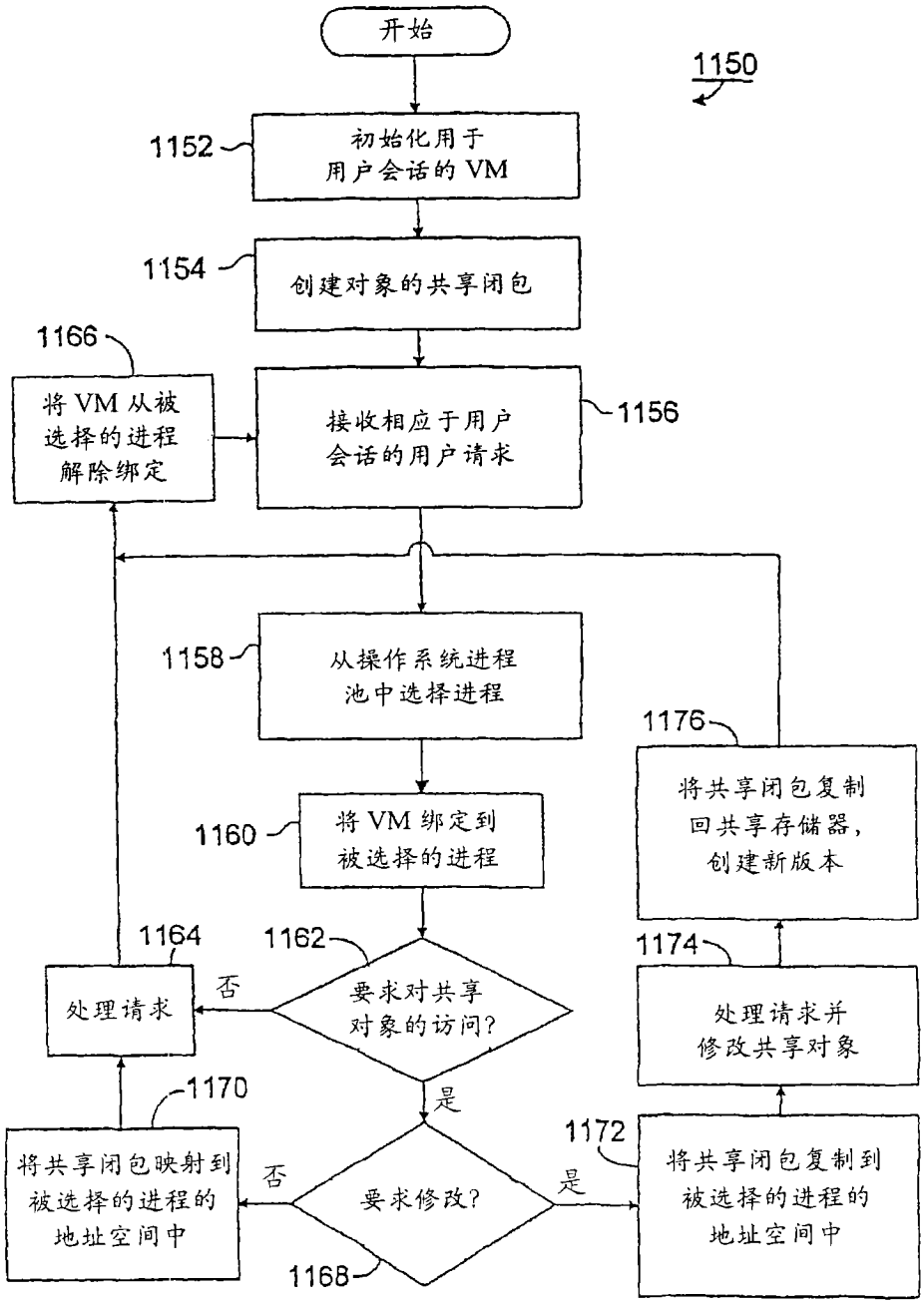

开始
1150
1152
初始化用于
用户会话的 VM
1154
创建对象的共享闭包
1166
将 VM 从被
选择的进程
解除绑定
接收相应于用户
会话的用户请求
1156
1158
从操作系统进程
池中选择进程
1176
将共享闭包复制
回共享存储器，
创建新版本
1160
将 VM 绑定到
被选择的进程
1164
1162
处理请求
否
要求对共享
对象的访问?
1174
处理请求并
修改共享对象
是
1170
1172
将共享闭包映射到
被选择的进程的
地址空间中
否
要求修改?
是
将共享闭包复制到
被选择的进程的
地址空间中
1168

图 11

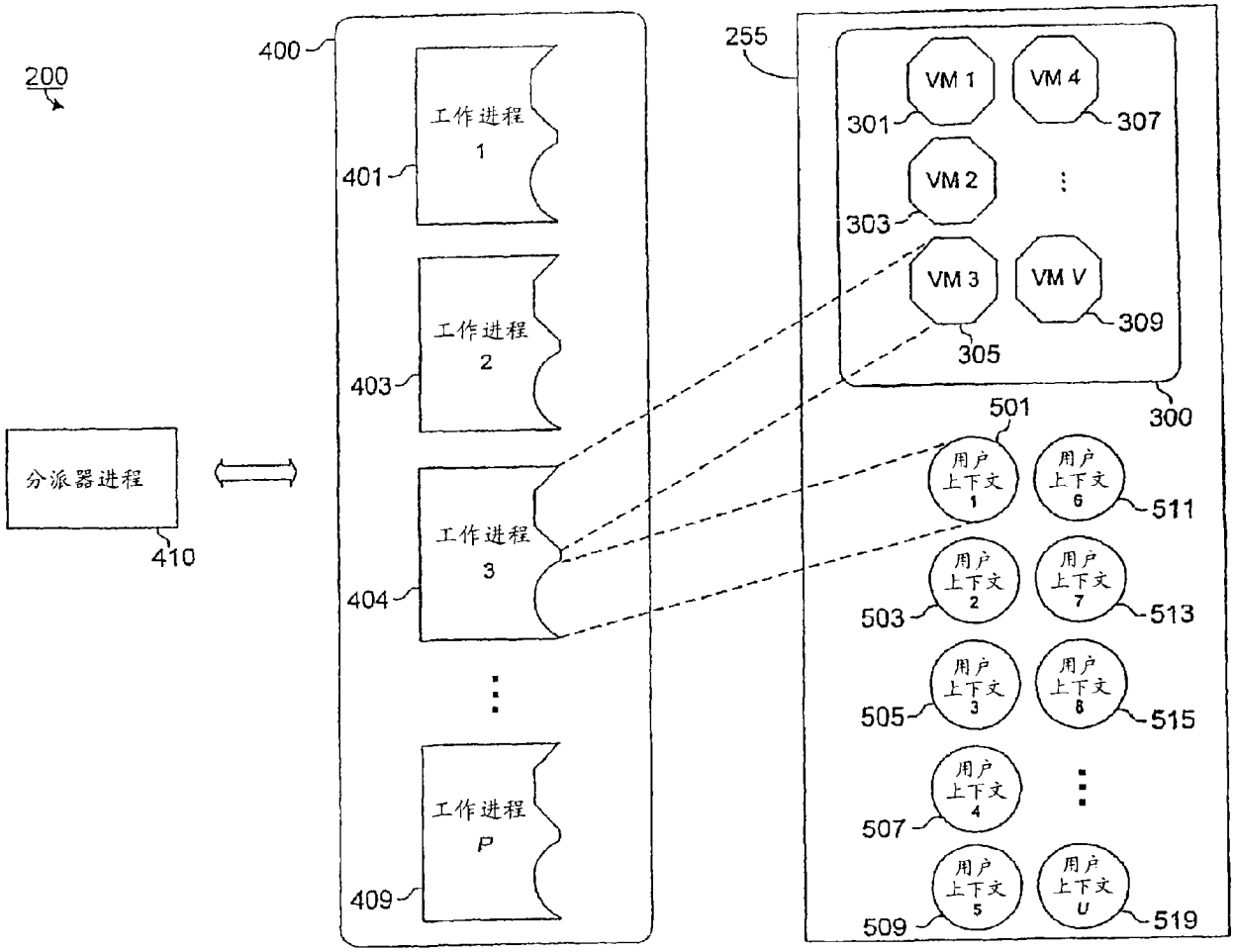

200
400
255
工作进程 1
401
工作进程 2
403
工作进程 3
404
工作进程 P
409
分派器进程
410
VM 1
VM 4
301
307
VM 2
303
VM 3
VM V
309
305
300
501
用户上下文 1
用户上下文 6
511
用户上下文 2
用户上下文 7
503
513
用户上下文 3
用户上下文 8
505
515
用户上下文 4
507
用户上下文 5
用户上下文 U
509
519

图 12

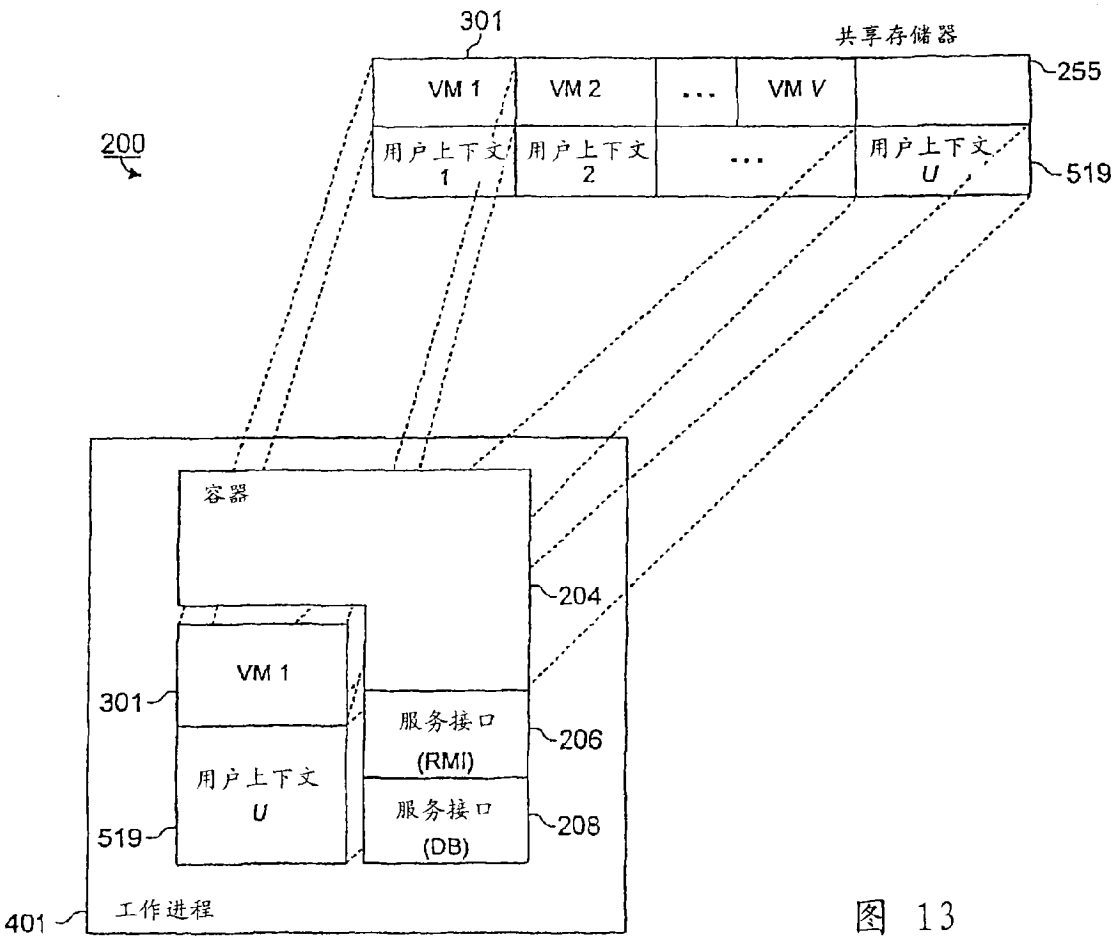

301
共享存储器
VM 1
VM 2
...
VM V
255
200
用户上下文 1
用户上下文 2
...
用户上下文 U
519
容器
204
VM 1
301
服务接口 (RMI)
206
用户上下文 U
519
服务接口 (DB)
208
401
工作进程

图 13

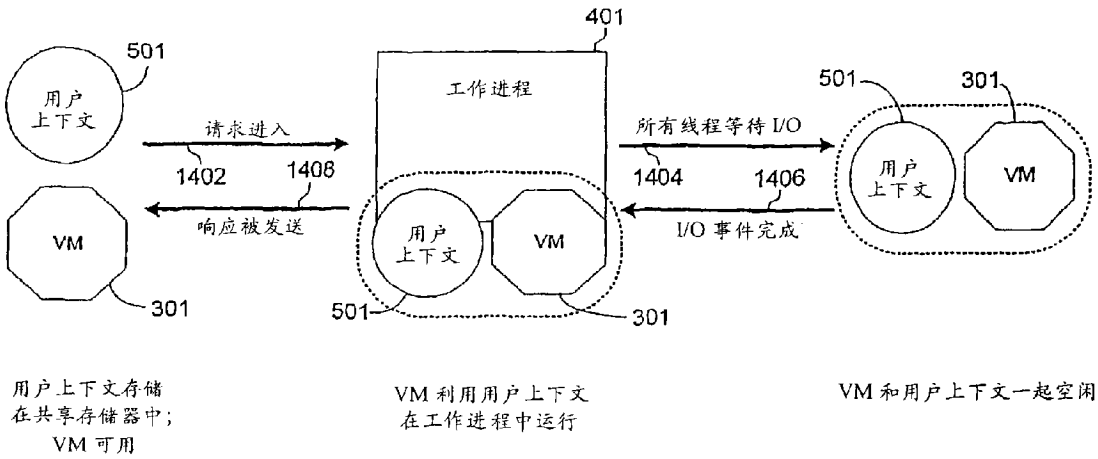

401
501
用户
上下文
工作进程
501
301
请求进入
1402
1408
响应被发送
所有线程等待 I/O
1404
1406
I/O 事件完成
用户
上下文
VM
VM
用户
上下文
VM
501
301
用户上下文存储
在共享存储器中；
VM 可用
VM 利用用户上下文
在工作进程中运行
VM 和用户上下文一起空闲

图 14

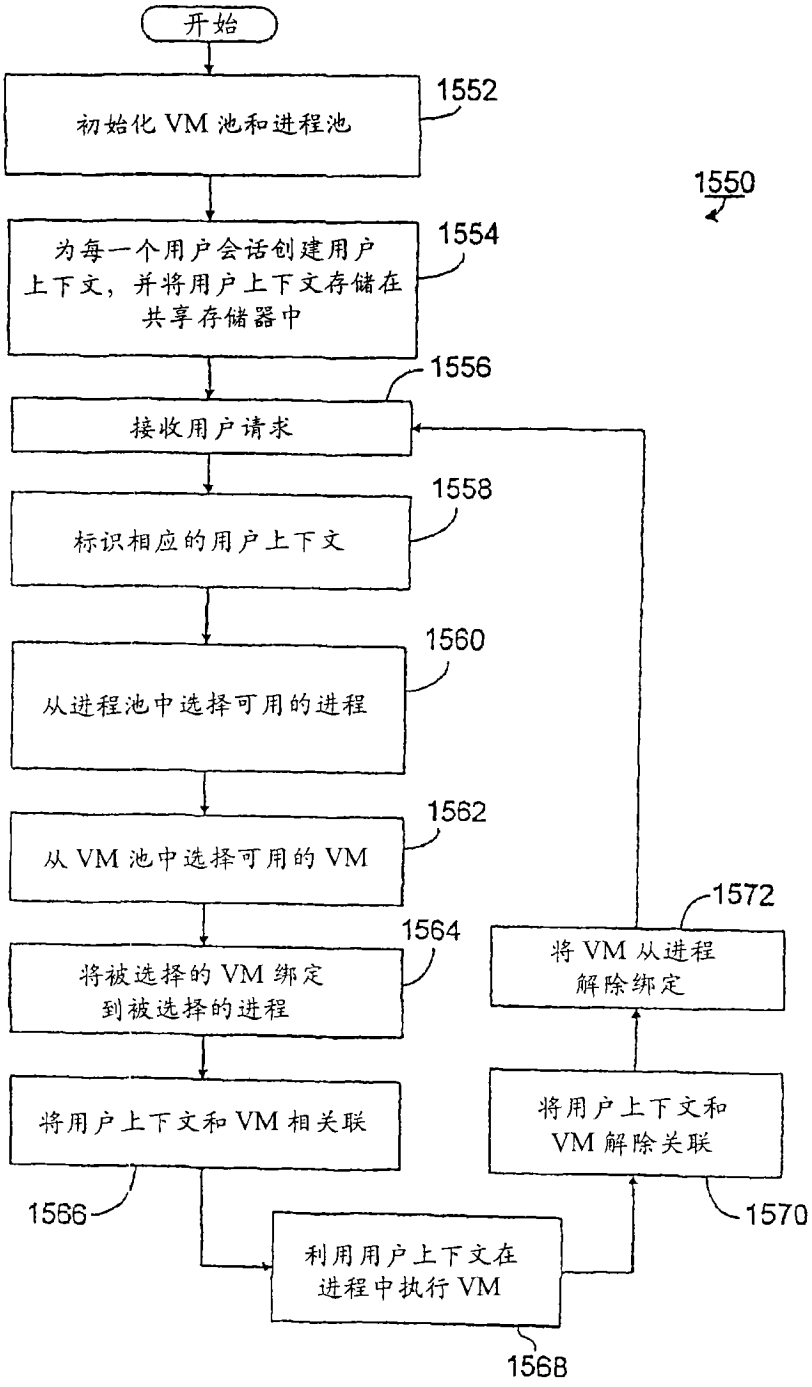

开始
初始化 VM 池和进程池
1552
1550
为每一个用户会话创建用户上下文，并将用户上下文存储在共享存储器中
1554
接收用户请求
1556
标识相应的用户上下文
1558
从进程池中选择可用的进程
1560
从 VM 池中选择可用的 VM
1562
将被选择的 VM 绑定到被选择的进程
1564
将用户上下文和 VM 相关联
1566
利用用户上下文在进程中执行 VM
1568
将用户上下文和 VM 解除关联
1570
将 VM 从进程解除绑定
1572

图 15

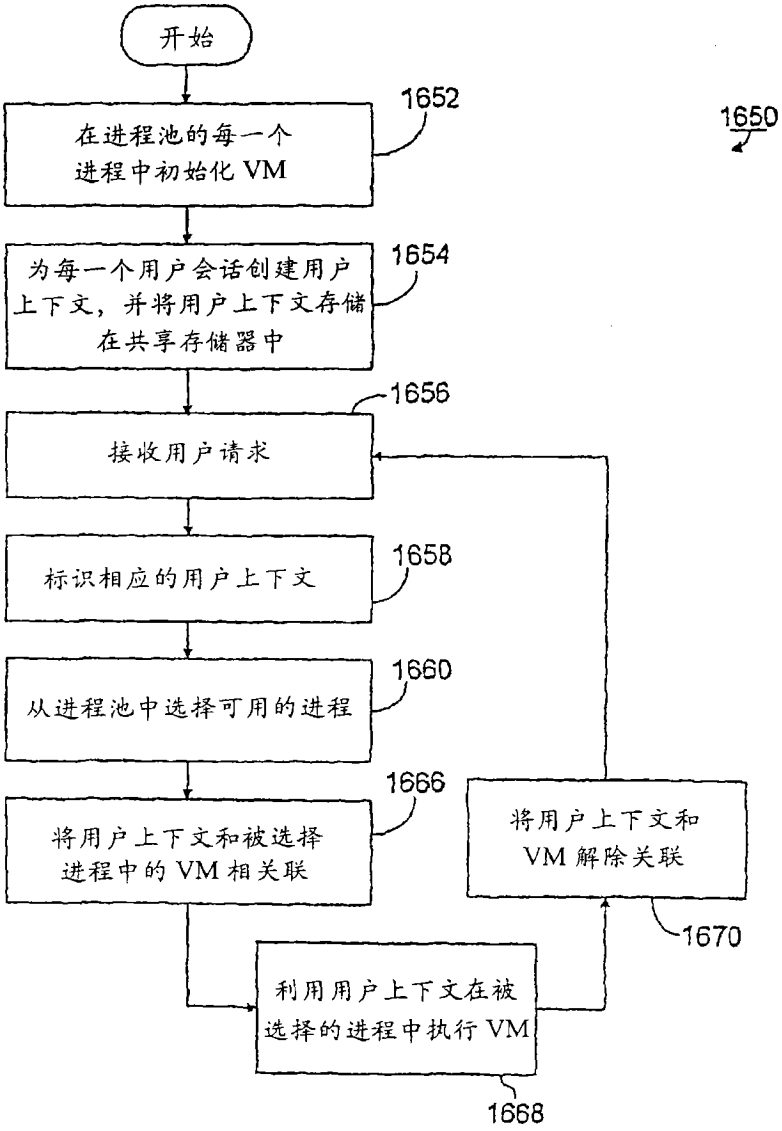

开始
在进程池的每一个进程中初始化 VM
1652
1650
为每一个用户会话创建用户上下文，并将用户上下文存储在共享存储器中
1654
1656
接收用户请求
标识相应的用户上下文
1658
从进程池中选择可用的进程
1660
将用户上下文和被选择进程中的 VM 相关联
1666
将用户上下文和 VM 解除关联
1670
利用用户上下文在被选择的进程中执行 VM
1668

图 16