



(12) 发明专利

(10) 授权公告号 CN 102567015 B

(45) 授权公告日 2016.02.03

(21) 申请号 201110359976.4

40 行-17 栏第 34 行, 图 1-18.

(22) 申请日 2011.11.14

US 5634114 A, 1997.05.27, 说明书第 3 栏第 1 段-第 10 栏第 1 段, 图 1-6.

(30) 优先权数据

12/982,970 2010.12.31 US

US 2007/0169103 A1, 2007.07.19, 全文.

(73) 专利权人 国际商业机器公司

地址 美国纽约

审查员 柏娟花

(72) 发明人 D·G·沃德 S·J·韦勃

(74) 专利代理机构 北京市中咨律师事务所

11247

代理人 张亚非 于静

(51) Int. Cl.

G06F 9/44(2006.01)

G06F 9/445(2006.01)

(56) 对比文件

US 6154878 A, 2000.11.28, 说明书第 2 栏第 40 行-17 栏第 34 行, 图 1-18.

US 6154878 A, 2000.11.28, 说明书第 2 栏第

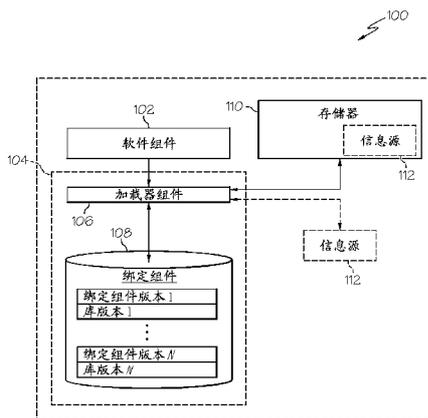
权利要求书3页 说明书13页 附图6页

(54) 发明名称

用于执行动态软件版本选择的方法和系统

(57) 摘要

动态软件版本选择是通过将绑定库与软件组件关联来执行的。所述绑定库包括加载器组件和绑定组件,其中所述加载器组件充当软件组件和绑定组件之间的中介,所述绑定组件包括软件组件端逻辑。动态软件版本选择进一步包括加载器组件中判定将库从绑定组件加载到处理设备的存储器的逻辑,以及由加载器组件选择所需的被请求库版本,其中所需库版本是从与绑定库的绑定组件关联的至少两个库版本中选择的。仍进一步地,动态软件版本选择包括在不需要重新启动关联的软件组件的情况下,将所需库版本加载到计算机处理系统的存储器。



1. 一种执行动态软件版本选择的方法,包括:
将绑定库与软件组件进行关联,所述绑定库包括加载器组件和绑定组件,其中:
所述加载器组件充当软件组件和绑定组件之间的中介,以及
所述绑定组件包括软件组件端逻辑;
通过网络使用绑定库与中间件应用的服务器交互;
使用程序版本表存储用于相应程序的特定库版本;
由加载器组件识别触发器,以便将库从绑定组件加载到处理设备的存储器;
由加载器组件选择所需的被请求库版本,其中所需库版本是从与绑定库的绑定组件关联的至少两个库版本中选择的;以及
将所需库版本加载到计算机处理系统的存储器,其中,所述加载包括:
卸载已加载到所述存储器的当前库版本;
加载新的库版本,
所述方法进一步包括:
使用库版本表识别库版本;以及
存储跟踪当前库版本的库版本变量;
其中:
由加载器组件选择所需的被请求库版本包括:
由加载器组件读取库版本变量的值;以及
根据所述库版本变量的值选择所需库版本。
2. 如权利要求 1 中所述的方法,进一步包括:
由加载器组件从软件组件接收软件组件程序接口调用;
将接收的调用转发到已加载到存储器的所需库版本的相应软件组件编程接口。
3. 如权利要求 1 中所述的方法,进一步包括:
在不更改绑定库的加载器组件的情况下,使用更新的库版本更新绑定库的绑定组件;
通过加载器组件加载更新的库版本,其中软件组件在加载期间不关闭,所述加载包括:
断开连接绑定组件的先前库版本;以及
连接更新的库版本;以及
由加载器组件将来自软件组件的调用转发到已加载的更新的库版本。
4. 如权利要求 1 中所述的方法,其中所述方法还包括由加载器组件接收将库从绑定组件加载到处理设备的存储器的请求,所述由加载器组件接收将库从绑定组件加载到处理设备的存储器的请求包括:
从用户软件组件接收断开连接调用;以及
从用户软件组件接收连接调用。
5. 如权利要求 1 中所述的方法,其中所述方法还包括由加载器组件接收将库从绑定组件加载到处理设备的存储器的请求,所述由加载器组件接收将库从绑定组件加载到处理设备的存储器的请求包括:
从软件组件接收任何调用。
6. 一种用于执行动态软件版本选择的系统,包括:

被配置为将绑定库与软件组件进行关联的装置,所述绑定库包括加载器组件和绑定组件,其中:

所述加载器组件充当软件组件和绑定组件之间的中介,以及
所述绑定组件包括软件组件端逻辑;

被配置为通过网络使用绑定库与中间件应用的服务器交互的装置;

被配置为使用程序版本表存储用于相应程序的特定库版本的装置;

被配置为由加载器组件识别触发器,以将库从绑定组件加载到处理设备的存储器的装置;

被配置为由加载器组件选择所需的被请求库版本的装置,其中所需库版本是从与绑定库的绑定组件关联的至少两个库版本中选择的;以及

被配置为将所需库版本加载到计算机处理系统的存储器的装置,其中,所述加载包括:

卸载已加载到所述存储器的当前库版本;

加载新的库版本,

所述系统进一步包括:

被配置为使用库版本表识别库版本的装置;以及

被配置为存储跟踪当前库版本的库版本变量的装置;

其中:

被配置为由加载器组件选择所需的被请求库版本的装置包括:

被配置为由加载器组件读取库版本变量的值的装置;以及

被配置为根据所述库版本变量的值选择所需库版本的装置。

7. 如权利要求 6 中所述的系统,进一步包括:

被配置为由加载器组件从软件组件接收软件组件程序接口调用的装置;

被配置为将接收的调用转发到已加载到存储器的所需库版本的相应软件组件编程接口的装置。

8. 如权利要求 6 中所述的系统,进一步包括:

被配置为在不更改绑定库的加载器组件的情况下,使用更新的库版本更新绑定库的绑定组件的装置;

被配置为通过加载器组件加载更新的库版本的装置,其中软件组件在加载期间不关闭,所述加载包括:

被配置为断开连接绑定组件的先前库版本的装置;以及

被配置为连接更新的库版本的装置;以及

被配置为由加载器组件将来自软件组件的调用转发到已加载的更新的库版本的装置。

9. 如权利要求 6 中所述的系统,其中所述系统还包括被配置为由加载器组件接收将库从绑定组件加载到处理设备的存储器的请求的装置,所述被配置为由加载器组件接收将库从绑定组件加载到处理设备的存储器的请求的装置包括:

被配置为从用户软件组件接收断开连接调用的装置;以及

被配置为从用户软件组件接收连接调用的装置。

10. 如权利要求 6 中所述的系统,其中所述系统还包括被配置为由加载器组件接收将

库从绑定组件加载到处理设备的存储器的请求的装置,所述被配置为由加载器组件接收将库从绑定组件加载到处理设备的存储器的请求的装置包括:

被配置为从软件组件接收任何调用的装置。

11. 如权利要求 6 中所述的系统,进一步包括被配置为根据新加载的绑定组件的需要,将内部状态数据结构从其当前形式迁移到新形式的装置。

12. 如权利要求 11 中所述的系统,其中执行迁移所需的逻辑由绑定组件的每个版本提供。

用于执行动态软件版本选择的方法和系统

技术领域

[0001] 本发明的各方面一般地涉及软件版本选择,更具体地说,涉及在版本更改期间,以对支持的过程或服务干扰最小或无干扰的方式动态地选择软件版本。

背景技术

[0002] 中间件可使一台或多台机器上运行的多个进程进行交互。因此,中间件在支持诸如事务处理系统与消息传递和排队系统之类的复杂分布式应用方面非常有用。作为一个实例,形式为联机事务处理(OLTP)系统的中间件经常用于支持关键业务过程和服务。例如,联机事务处理系统可用于便利和管理在订单录入系统中捕获新数据,更新现有数据,处理数据录入和检索等的面向事务的应用。作为另一示例,形式为消息排队系统的中间件可用于通过将应用特定的消息写入队列以及从队列检索这些消息来便利程序间通信,而不需要通信程序之间具有专门的逻辑连接。

[0003] 包括现代 OLTP 系统、消息和排队系统等的中间件一般支持跨网络的事务并且可以跨由不同实体控制的一个或多个域。因此,诸如 OLTP 应用与消息和排队应用之类的中间件应用通常使用允许事务在网络内的不同计算机平台上运行的客户机/服务器处理和代理操作(brokering)来实现。因此,对此类中间件系统的软件升级通常需要关闭应用,从而使关联的应用脱机以允许进行软件安装和必要时的配置。

发明内容

[0004] 根据本发明的方面,动态软件版本选择通过将绑定库与软件组件进行关联来执行。所述绑定库包括加载器组件和绑定组件,其中所述加载器组件充当软件组件和绑定组件之间的中介(intermediary),且所述绑定组件包括软件组件端逻辑。动态软件版本选择进一步包括由加载器组件识别触发器,以将库从绑定组件加载到处理设备的存储器,以及由加载器组件选择所需库版本,其中所需库版本是从与绑定库的绑定组件关联的至少两个库版本中选择的。仍进一步地,动态软件版本选择包括在不重新启动关联的软件组件的情况下,将所需库版本加载到计算机处理系统的存储器。例如,加载器组件所作的加载所需库版本的判定可以由控制过程所填充的库版本表或其他源控制。

附图说明

[0005] 图 1 是根据本发明的方面用于实现动态软件版本选择的系统的示意图。

[0006] 图 2 是示出根据本发明的方面用于实现动态软件版本选择的示例性方法的流程图。

[0007] 图 3 是根据本发明的方面用于在消息传递系统环境的示例性上下文中实现动态软件版本选择的系统的示意图。

[0008] 图 4 是根据本发明的方面使用共享库版本表的用于实现动态软件版本选择的系统的示意图。

[0009] 图 5 是根据本发明的方面使用共享库版本表和共享程序版本表的用于实现动态软件版本选择的系统的示意图。

[0010] 图 6 是具有用于实现如本文详细描述的本发明的方面的功能的计算机可读存储介质的计算机系统的方块图。

具体实施方式

[0011] 本发明的各方面提供了能够在不关闭软件系统或不以其他方式中断服务的情况下通过新软件修改软件系统的技术。软件修改包括,例如,实现软件升级,恢复到早期软件版本,重新加载软件版本,修复或修补软件版本等。因此,提供了能够在不发生中断的情况下通过新软件升级的软件系统,所述新软件例如包含对已知问题的修复和 / 或新功能,并且其部件可使用诸如服务等级之类的不同软件版本等。软件系统还可以回退到早期版本。此外,还可以通过新软件修改客户端系统和服务器端系统两者。

[0012] 进一步地,此处描述的本发明的各方面提供了在持续运行的进程内的应用编程接口 (API) 边界上管理版本化实现,从而不需要针对关联的软件代码进行任何进程重新启动。在不重新启动进程的情况下实施更新的能力甚至可以在进程为不受用于执行软件修改的更新机制控制的第三方应用 (例如,客户应用) 的情况下实现。因此,很容易实现本机代码库的运行时更新和修改。而且,更新机制可以实现动态内部状态处理以允许在运行进程之下的函数库版本转换,例如,在使用动态链接库的系统中的这种函数库版本转换。

[0013] 更进一步地,本发明的方面提供了更新机制,此机制便利了做出和控制迁移判定 (例如,如何以及何时升级、降级或以其他方式替换可能存储在库中的软件代码) 的能力。因此,此更新机制可以表现出动态行为,从而提供了软件版本修改过程和判定的灵活性。而且,此更新机制可以例如逐程序地或基于其他选择标准实现选择性软件版本控制。

[0014] 根据本发明的更进一步的方面,此更新机制可以包括与软件组件 (调用组件) 关联的绑定库。通过这种方式,调用组件不时地调用绑定库以实现应用编程接口 (API) 调用。绑定库实现为加载器组件和绑定组件。加载器组件充当中介以从软件组件接收调用并将等效的 API 调用发送到绑定组件。绑定组件包括软件组件端逻辑,所述逻辑包括至少两个库版本。每个库版本都包括用于对来自调用组件的应用编程接口调用做出响应的逻辑。

[0015] 加载器组件识别或以其他方式检测到触发器,并且作为对此的响应,实现判定逻辑。所述判定逻辑可以检查是否存在新的库版本,检测应该加载的所需库版本,引导加载器组件从绑定组件选择所需的库版本并将所述库版本加载到处理设备的存储器等。如果要替换加载到存储器的库版本,则在不需要重新启动关联的软件组件的情况下卸载旧的库版本并加载新的库版本。

[0016] 例如,在第一说明性示例中,仅仅选择的 API 调用 (例如,连接 / 断开连接 (或类似操作) API) 具有导致库版本重新加载的判定逻辑。因此,可以响应于从关联的软件组件接收到一个或多个特定命令 (例如,连接和断开连接 API 调用),触发加载器组件以检查加载到存储器的库版本,以及在必要时更新版本。如将在此更详细地描述那样,在该说明示例中,不需要状态迁移来实现库版本重新加载。

[0017] 作为另一说明性示例,所有对加载器组件的 API 调用都具有判定逻辑。因此,响应于从关联的软件组件接收任何 API 调用,触发加载器组件以检查加载到存储器的库版本,

以及在必要时更新版本。如将在此更详细地描述那样,在该说明性示例中,应该支持状态迁移。仍进一步地,无论是仅选择的 API 调用具有触发库版本检查的判定逻辑,还是所有 API 调用都具有触发库版本检查的判定逻辑,加载器组件都可支持程序级控制或程序特定的控制,以允许响应于接收的 API 调用,控制哪些程序将迁移库版本,哪些不迁移库版本。

[0018] 动态软件版本选择

[0019] 现在参考附图,具体为参考图 1,其示出了适合于根据本发明的方面实现动态软件版本选择的系统 100 的示意图。系统 100 可以在单个物理处理设备上实施。替代地,系统 100 可以跨诸如通过网络以通信的方式连接在一起的多个物理处理设备分布。在此方面,每个处理设备都可以包括用于实现此处更详细地描述的本发明的方面的硬件和 / 或软件处理组件。

[0020] 如图所示,系统 100 包括至少一个软件组件 102 和关联的绑定库 104。软件组件 102 可以包括例如调用另一组件(例如,关联的绑定库 104)的服务的任何软件。软件组件 102 可以例如实现为软件应用、程序、对象、服务、代理(agent)、操作系统等。在此方面,软件组件 102 在此还可以被称为调用组件 102。

[0021] 绑定库 104 实现为两个组件,包括加载器组件 106 和绑定组件 108。加载器组件 106 与软件组件 102 和绑定组件 108 进行交互以将绑定组件的关联库的正确版本加载到存储器 110。加载器组件 106 还可以可选地与一个或多个可选的信息源 112 进行交互,所述信息源可被提供以帮助进行版本管理。例如,如将在此更详细地描述那样,信息源 112 可以实现为指示绑定组件 108 所维护的关联库的当前版本的表,该表可以存储在存储器 110 中,也可以作为单独文件。

[0022] 绑定组件 108 包括一个或多个库版本,其包含软件组件端逻辑,包括至少两个库版本。每个库版本都包括对来自调用组件 102 的应用编程接口(API)调用做出响应的逻辑。如在此使用的,术语“库”旨在是一般性的和非限制性的。例如,每个“库”版本都可以包括任何相关的软件封装单元,例如,jar 文件、程序、模块、动态链接库等。而且,每个库版本都不限于本机代码库,因此可以用于诸如 Java 的实现。

[0023] 加载器组件 106 充当软件组件 102 和绑定库 104 的绑定组件 108 之间的中介。在此方面,加载器组件 106 可以是与软件组件 102 静态链接的唯一组件。根据本发明的方面,加载器组件 106 可以实现为库本身。在该示例性实现中,加载器组件 106 主要作为转发库发挥功能。也就是说,在多数时间,加载器组件 106 从软件组件 102 接受 API 调用并作为对此的响应,在与绑定库 108 关联的当前加载的库版本中调用等效的 API。

[0024] 根据本发明的进一步的方面,判定逻辑可以被插入从加载器组件 106 到适当的绑定组件 108 的库的转发功能的执行路径中。哪些转发功能应该具有判定逻辑以及哪些转发功能不应具有判定逻辑的选择取决于特定实现,下面将更详细地描述所述特定实现的示例。

[0025] 绑定组件 108 包括一个或多个库版本。例如,如图所示,绑定组件 108 包括“N”个库版本,其中 N 可以表示任意的库版本数量。每个库版本可以包括,例如,提供软件组件端逻辑的动态链接库,软件组件 102 使用此软件组件端逻辑执行特定的操作或功能,例如执行与中间件过程关联的 API 调用。例如,绑定组件 108 可以包括实现来自软件组件 102 的 API 调用或以其他方式对此调用做出响应的逻辑。

[0026] 如上面更详细地说明的, 绑定组件 108 可以通过各种形式实现。如图所示, 文件系统存储多个绑定组件版本, 每个绑定组件版本具有关联库。在另一示例性实现中, 例如当所有必要的程序和代码使用库封装时, 绑定组件 108 可以包括存储每个库版本的文件系统库, 图 3 示出其中一个库版本示例。还可以替代地实现其他绑定组件配置。

[0027] 加载器组件 106 不包含支持软件组件 102 的调用所需的任何逻辑。而是, 加载器组件 106 动态地将与绑定库 104 的绑定组件 108 关联的正确库版本加载到相应计算机处理系统的存储器 110 中。加载库的过程是在不需要重新启动关联的软件组件的情况下实现的。一旦选择了适当的库并将其加载到存储器 110, 加载器组件 106 便从软件组件 102 接收调用并将来自软件组件 102 的等效调用从软件组件 102 转发到从库加载到存储器 110 的相应逻辑。因此, 加载器组件 106 拦截和居间传递 (mediate) 来自软件组件 102 的调用。一旦已将库加载到存储器 110, 加载器组件 106 便会偶尔检查是否存在新的绑定库版本 (或回复到现有的但不同的绑定库版本的指令)。如果加载器组件 106 检测到需要更改加载到存储器 110 的库版本, 则实现此更改, 而无需重新启动底层的软件组件 102。

[0028] 在示例性实现中, 根据情况 (例如, 当加载器组件 106 发现当前加载的或以其他方式使用的绑定组件 108 的库版本的级别早于最新可用的绑定组件 108 的库版本), 加载器组件 106 可以动态地卸载已加载到存储器的当前库版本, 并动态地将绑定组件 108 中的最新库版本加载到存储器。加载器组件 106 还可以根据需要, 可选地调用迁移逻辑 (例如, 在新选定的库版本中提供的迁移逻辑) 来重新安排任何内部数据结构以符合新的库版本的要求。因此, 内部状态迁移由绑定库 104 自动和动态地进行管理。一旦将新的库版本加载到存储器并且迁移了内部状态数据结构 (需要时), 加载器组件 106 便可使用新库完成来自软件组件 102 的当前调用的处理 (如果已做出调用)。

[0029] 如将在此更详细地描述那样, 加载器组件 106 可以使用诸如一个或多个信息源 112 之类的资源来从绑定组件 108 选择适当的库版本进行加载。在此方面, 信息源 112 可以包括文件系统符号链接 (例如, 在类似 Unix 的文件系统上等)、控制文件、一个或多个表或数据库 (另存为文件或加载到存储器 110)、指针, 或用于识别来自绑定组件 108 的所需库的其他源。

[0030] 根据本发明的仍进一步的方面, 诸如管理器功能、管理员功能、监督员功能之类的常见控制功能可以监视或监督软件版本控制。例如, 管理器可以指令 (例如, 通过加载器组件 106) 在应用下次调用加载器组件 106 时所有相连的软件组件 102 使用所需库版本。管理器也可以不时地使用新的库版本更新绑定组件 108。因此, 绑定库 104 单独地或与一个或多个其他进程 (例如, 管理器或监控逻辑等) 结合地实现更新机制。

[0031] 此处描述的技术可以由允许通过绑定组件 108 内的库版本动态升级 / 降级绑定软件的规则或其他过程驱动。而且, 动态升级 / 降级绑定软件可以仅针对用户应用程序的子集发生, 可以针对特定应用发生, 或者可以在特定的时间和日期发生。

[0032] 现在参考图 2, 流程图示出了根据本发明的各方面执行动态软件版本选择的过程 200 的整个流程。过程 200 可以实现为一种方法, 或实现为包括硬件和软件组合的系统的一部分。例如, 软件可以存储在物理存储设备内, 以便当至少一个处理器处理所述软件时, 所述软件可以使相应的物理机器实现所示过程 200 指定的操作。因此, 例如, 过程 200 可以在参考图 1 示出和描述的系统 100 上实现。仍进一步地, 过程 200 可以实现为包括有形计算

机可读存储介质的计算机程序产品,所述有形计算机可读存储介质具有包含于其上的计算机可读程序代码。

[0033] 过程 200 包括将绑定库与软件组件进行关联 202。所述绑定库包括加载器组件和绑定组件,其中所述加载器组件充当软件组件和绑定组件之间的中介,所述绑定组件包括软件组件端逻辑。根据本发明的某些方面,加载器组件的工作是“识别”或“判定”需要重新加载绑定库。在此方面,加载器组件识别出必须加载不同的绑定库。

[0034] 过程 200 进一步包括由加载器组件触发对已加载库版本的验证 204。例如,加载器组件可以执行判定逻辑以,例如根据规则、条件或其他信息,判定是否应该修改当前已加载的库,例如,升级到新版本或回退到较早版本。可以通过接收与判定逻辑关联的特定 API 调用以触发对库版本的验证来实现所述触发。作为另一说明性示例,加载器组件接收的所有 API 调用都可以具有与其关联的判定逻辑。如果未加载库,则加载器组件可以使用相同的规则、条件或其他信息选择要加载到存储器的库版本。

[0035] 如果无需修改已加载到存储器的库版本,则过程 200 结束。否则,过程 200 进一步包括由加载器组件选择所需的被请求库版本 206。所需库版本可以从与绑定库的绑定组件关联的至少两个库版本中选择。所需的被请求库版本可以例如使用文件(例如,文件系统符号链接、引导选择的控制文件等)进行选择。作为替代示例,加载器组件可以查看表或其他信息源以确定要加载到存储器的所需版本的身份标识。例如,所述过程可以(例如,通过加载器组件)使用库版本表来识别库版本。因此,过程 200 可以存储加载器组件可访问和使用的跟踪当前库版本的库版本变量。因此,例如加载器组件可以读取库版本变量的值并根据所述库版本变量的值选择所需库版本。

[0036] 过程 200 仍进一步地包括在无需重新启动关联的软件组件的情况下,将所需库版本加载到计算机处理设备的存储器。一旦选定库版本并将其加载到存储器,所述过程可以进一步包括由加载器组件从软件组件接收软件应用程序接口调用,并将接收的调用转发到已加载到存储器的所需库版本的相应软件应用编程接口。

[0037] 中间件环境中的动态版本控制

[0038] 本发明的各方面通过以不宕机或不中断服务的方式,将新的动态链接库置入中间件系统来便利在诸如中间件环境之类的环境中的动态版本控制。此类实现进一步提供了中间件环境中的版本管理。

[0039] 现在参考图 3,其示出了用于实现动态软件版本选择的系统 300 的示意图。系统 300 实现本发明的方面,所述方面在许多方面与针对图 1 的系统 100 列出的各方面类似。但是,系统 300 是在消息传递系统环境的示例性上下文中示出的,以说明根据本发明的进一步方面的某些特征。

[0040] 系统 300 包括多个实现为用户应用的软件组件 302,每个软件组件 302 与包括加载器组件 306 和绑定组件 308 的绑定库 304 关联。绑定组件 308 包含处理消息传递和排队系统应用编程接口调用时所需的所有软件组件端逻辑。一般而言,包括加载器组件 306 和绑定组件 308 的绑定库 304 与参照图 1 描述的同名组件类似。

[0041] 示例性系统 300 可以集成消息和排队系统,例如由位于纽约阿莫克的国际商业机器公司推出的用于消息传递和排队的 MQSeries 中间件系统。使用此类安排,各种软件组件 302 可以通过将应用特定的数据(消息)写入队列以及从队列检索这些消息来相互通信,

而无需私有的专用逻辑连接来链接应用。在此方面,每个绑定库 304 可以包含例如一组可执行体 (executable),该组可执行体实现客户机 314 以便使用消息传递和排队系统进行通信。

[0042] 系统 300 进一步包括多个客户机处理设备 322,所述客户机处理设备在执行其关联的用户软件组件 302 的过程中跨网络 324 与服务器计算机 326 进行通信。典型的物理处理设备可以包括例如:服务器计算机、个人计算机、笔记本计算机、上网本计算机、平板计算机、事务性系统、目的驱动设备 (purpose-driven appliance)、诸如个人数字助理 (PDA) 之类的普及计算设备、掌上计算机、蜂窝式接入处理设备、智能手机或其他任何能够执行计算机代码的设备等。

[0043] 网络 324 提供各种处理设备 322 和服务器 326 之间的通信链路,并且可以受连网组件的支持,所述网络组件包括例如,路由器、集线器、防火墙、网络接口、有线或无线通信链路及相应的互连、蜂窝基站和相应的蜂窝转换技术(例如,在蜂窝和 tcp/ip 之间转换)等。而且,网络 324 可以包括使用一个或多个内部网、外联网、局域网 (LAN)、广域网 (WAN)、无线网络 (WIFI)、包括万网的因特网和 / 或其他安排以便以实时方式或者其他方式例如通过时间平移 (time shifting)、批处理等实现处理设备 322 和服务器计算机 326 间的通信的连接。

[0044] 服务器计算机 326 可用于,例如执行可体现为一组长时间运行的服务进程的队列管理器 328。队列管理器 328 定义针对每个用户软件组件 302 管理消息的发送和排队的运行时应用。在该说明性示例中,消息传递和排队系统在客户机 / 服务器配置中实现。替代地,队列管理器 328 的示例可以实现为每个处理设备 322 上的长时间运行的进程。

[0045] 一般地讲,用户软件组件 302 使用 API 调用通过其关联的客户机 314 与队列管理器 328 进行通信,例如,以将消息发送到与其他客户应用关联的队列或从关联的队列检索已被其他软件组件 302 存放的消息。因此,队列管理器 328 与可能许多与不同的客户机处理设备 322 关联的不同的用户软件组件 302 进行交互,从而访问这些服务进程,每个进程都使用产品特定的绑定库 304。无论队列管理器是否在远端,例如位于作为客户机 / 服务器实现的一部分的服务器计算机 326 上,或者与绑定库 304 共同位于相应处理设备 322 的本地配置中,都可进行这种安排。

[0046] 希望能不时地更新在系统 300 内运行的各种组件的软件。假设消息传递和排队系统需要执行更新。根据本发明的各方面,可以在不中断服务的情况下动态地部署新的消息和排队系统软件版本。此类软件修改包括更改与每个用户软件组件 302 绑定的软件,例如由关联的绑定库 304 实例加载的关联的库版本。软件修改进一步包括更改消息传递和排队系统的内部进程,例如,更改队列管理器 328,更改、添加或删除与绑定组件 308 相关联地存储的库版本等。

[0047] 对于修改消息传递和排队系统内部软件的情况,此类内部进程可以被设计为在软件升级可用时或在软件升级可用之后立即可重新启动。与之相对,用户软件组件 302 具有进程生命周期要求,所述生命周期要求由用户业务要求决定并且由客户控制,而不受消息传递和排队系统控制。

[0048] 一般而言,与队列管理器 328 关联的服务进程可以实现为传统操作系统进程。MQSeries 队列管理器中的示例为用于通过网络与其他消息通道代理 (MCA) 进行消息读写

的 MCA 和本地队列管理器代理 (LQMA), 它们构成基本排队引擎。这些进程在如下意义上是被管理的, 即队列管理器 328 自己维护在需要时重用的可用 MCA 和 LQMA 进程池。空闲的 MCA 和 LQMA 进程实例在新工作到来时被重用。当在文件系统中安装新软件 (例如, 形式为新库或程序) 时, 队列管理器可以关闭这些程序的任何空闲实例并启动新进程。进程将在没有任何其他帮助的情况下使用新软件。类似地, 通过取消安装新软件文件的文件系统更改可以实现软件降级。

[0049] 用户软件组件 302 使用绑定库 304 (例如, 在 MQSeries 的情况中名为“libmqm”的库) 获得对消息传递和排队系统服务的访问。在一方面, 绑定库 304 向应用 304 提供固定的良好定义的接口。而且, 绑定库 304 与队列管理器 328 的内部进行接口连接。

[0050] 一般而言, 即使有新软件可用, 用户软件组件 302 也不能被强制重新启动。例如, 一般情况下, 服务器计算机 326 上的进程不能强制重新启动远程处理设备 322 上的软件组件 302。但是, 在说明性示例中, 当前的一组程序 / 库可执行体和新的一组程序 / 库可执行体两者在每个文件系统中是一起可用的, 如每个包括多个不同的库版本的绑定组件 308 所示。在此方面, 由与消息传递和排队系统关联的每个软件组件 302 所执行的软件 (例如, 客户机 314) 可以类似于参考图 1 和图 2 描述的方式进行更新。

[0051] 为了允许在不需要用户重新启动其软件组件 302 的情况下部署新的绑定库版本, 每个绑定库 304 通过类似于参考图 1 描述的加载器组件 106 和绑定组件 108 的方式被分为两个不同的组件, 包括加载器组件 306 和绑定库 308。加载器组件 306 可以实现为, 例如, 一个小型库。但是, 加载器组件 306 不包含支持消息传递和排队系统的应用编程接口所需的任何逻辑。而是, 加载器组件 306 将正确的绑定库版本加载到相应客户机处理设备 322 上的存储器。加载器组件 306 然后将等效的 API 调用从应用程序转发到已加载库 (例如, 客户机 314) 中的适当接口, 以实现将消息发送到消息队列, 从消息队列检索消息或以其他方式与消息传递和排队系统进行交互。

[0052] 一旦加载器组件 306 从绑定组件 308 选择并加载了库, 加载器组件 306 便会将来自软件组件 302 的所有 API 调用的等效版本转发到从绑定组件 308 加载的库。因此, 加载器组件 306 在用户软件组件 302 和从绑定组件 308 加载的库之间有效地拦截和居间传递与消息传递和排队系统关联的所有 API 调用。

[0053] 为了执行对客户处理设备 322 上的库的软件修改, 服务器 326 或其他任何源为绑定组件 308 提供相应库的新版本。绑定组件 308 存储库的方式使得每个库版本都能被相应的加载器组件 306 唯一地识别。进一步地, 可以为加载器组件 306 提供识别要加载的库版本的信息, 此处将更全面地进行描述的那样。但是, 用户软件组件 302 的运行实例可以继续使用当前已加载到关联的处理设备 322 的存储器的任何库版本, 直到发出相应的命令, 使加载器组件 306 使用新的库版本更新相应客户计算机 322 的存储器。

[0054] 当发生预定事件时, 加载器组件 306 可以卸载早期库版本 (如果已加载了早期版本的话)。相应地, 加载器组件 306 可以从绑定组件 308 加载新的库版本。

[0055] 根据说明性示例, 用户软件组件 302 可以使用指定的断开连接和重新连接 API 序列触发从绑定组件 308 加载新库。直到用户软件组件 302 执行该序列的应用编程接口调用为止, 用户软件组件 302 将继续使用旧的库版本软件。

[0056] 在示例性实现中, 软件库修改可以由用户软件组件 302 向绑定应用 304 发出断开

连接和重新连接API调用来触发。因此,加载器组件306可以在客户应用断开连接时动态地卸载已加载到存储器的当前库版本,并且随后在客户应用连接时动态地加载新的库版本。在MQSeries示例性实现中,从绑定组件308加载和卸载库是在加载器组件的MQCONN(MQ series 连接)和MQDISC(MQ series 断开连接)处理中实现的。该说明性示例中加载或卸载库的命令由相应用户软件组件302执行,因此允许用户应用控制更新时间。

[0057] 根据本发明的方面,在上述示例性实现中,仅连接/断开连接转发API需要具有判定逻辑。对于连接/断开连接API,可以在将调用转发到相应的库之前执行判定逻辑。因此,如果需要修改当前已加载的库版本,可以在执行已调用的API之前实施此更改。其他API可以触发从加载器组件306到关联的绑定库308的相应库版本的等效API调用,因此实现转发功能而在其中没有判定逻辑。在这种情况下,加载器组件306仅检查这些“特殊”的API(或类似指定的初始化/取消初始化API)中是否存在新的绑定库版本。其他API将不会触发加载器组件306执行版本检查。连接/断开连接API被认为“特殊”是因为它们建立和拆除绑定库304使用以处理其他API的内部状态。因此,无需支持加载器组件触发的状态迁移,因为连接/断开连接API自己会建立和拆除状态数据。因此,无需进行状态迁移。

[0058] 根据又一说明示例,所有/任何API(或API的子集)都具有检查库版本并在必要时触发库修改的判定逻辑。所述判定逻辑可以,例如,在将等效调用转发到已加载的库版本之前执行。由于所有API(或其子集)执行需要运行判定逻辑,因此,所述逻辑可以例如,通过使用存储指示当前所需的库版本的指示和/或可用库版本列表的基于存储器的共享库版本表来高效地工作。由于绑定库可能在API被调用时携带内部状态数据,因此,加载器组件306应该能够迁移绑定库状态,例如,通过使用新的绑定库中的迁移API来这样做。

[0059] 因此,根据本发明的方面,从绑定组件308选择要加载到存储器的库版本的操作脱离了操作系统的控制,而可以被置于加载器组件306的直接控制之下。加载器组件306可以在加载库之前执行任何所需的验证,并且可以实施较简单地依赖于底层操作系统库加载机制而言更为复杂的加载技术和策略。而且,加载器组件306能够执行回退到早期库版本软件级别的操作,例如,只要内部状态迁移是可能的并且受后来的绑定库中的迁移API支持。

[0060] 因此,本发明的方面通过消除针对系统软件(例如,该说明性示例中的中间件)的维护更新而计划的停机,帮助实现连续可用性。在此方面,此处描述的软件能力可以允许在不需要重新启动系统软件,甚至不需要应用在更改期间保持停顿(quiesce)的情况下实施更改。因此,即使当更改影响到应用接口时,应用也可以继续使用系统软件,从而两个(或更多个)不同的应用接口可以在应用更改的过程中由同一系统软件提供。

[0061] 有关动态地更新与中间件系统(例如,消息传递和排队系统)关联的软件的描述是通过展示而非限制的方式给出的,以便说明用于以不中断服务的方式执行动态软件修改的技术,该技术在本文中更全面地说明。

[0062] 库版本表

[0063] 现在参考图4,其示出用于实现动态软件版本选择的系统400的示意图。一般而言,系统400包括软件组件402和关联的绑定库404。绑定库404实现为两个不同的组件,包括加载器组件406和绑定组件408。在此方面,软件组件402、绑定库404、加载器组件406和绑定组件408基本上与参考图1-3中任何一个或多个图描述的同名组件类似。

[0064] 根据本发明的进一步方面,一般而言,当客户应用 402 发出 API 调用时,可触发加载和使用新的库软件。如将在此更详细地描述那样,可以响应于加载器组件 406 接收特殊 API(例如,连接和断开连接 API),触发版本检查和库重新加载。作为又一说明性示例,还可以响应于加载器组件 406 从相应的调用软件组件接收所有 API,触发版本检查和库重新加载。

[0065] 在所示的示例性实现中,系统 400 进一步包括共享库版本表 430。所述共享库版本表存储与绑定组件 408 存储的各种库版本相关的信息。例如,如图所示,库表 430 包含与相应的绑定组件 408 关联的所有可用库的名称及版本号。此外,共享表 430 可以包含库版本变量 432,该变量存储有关哪个库版本(例如,哪个共享表项)表示应该加载到存储器的当前库版本的指示。

[0066] 所述共享表可以由诸如参考图 3 描述的队列管理器 328 之类的任何管理进程进行填充。而且,可以实现某种机制来将可用库版本发布到例如共享库表 430。

[0067] 当发生预定触发事件时,例如,收到与相应判定逻辑关联的“特殊”API 调用,收到所有 API 调用(与适当判定逻辑关联的所有 API 调用)时,或根据其他判定逻辑(取决于特定实现),加载器组件 406 检查共享表 430 以查看用户程序 402 是否正在使用正确的当前可用库版本。如果软件组件 402 未使用正确的库版本,则加载器组件 406 安排(orchestrate)卸载旧的库版本并加载正确的新版本。例如,加载器组件 406 可以将库版本变量 432 中存储的值与已加载到存储器的库版本进行比较。

[0068] 因此,例如,每当用户软件组件 402 做出与相应判定逻辑关联的 API 调用,加载器组件 406 都可以针对当前使用的库版本执行可用库检查。当加载器组件 406 发现当前已加载的库与最新可用库不在同一级别上时,加载器组件便可以动态地卸载当前库并动态地加载指定库。可选地,可以实现对新库中的特殊迁移 API 的调用来,例如,重新安排任何内部状态数据结构以符合新库的要求(如有必要),并且使用新库完成当前 API 调用的处理。

[0069] 在某些示例性实现中,管理器或监督进程(例如,参考图 3 描述的队列管理器)可以在应用下一次做出 API 调用时,引导所有相连的用户应用程序使用不同的(例如,更新或更旧的)库软件版本。替代地,其他规则、时间条件等可以决定何时执行软件版本修改。

[0070] 程序特定的库版本控制

[0071] 现在参考图 5,其示出用于实现动态软件版本选择的系统 500 的示意图,该系统扩展了动态软件版本选择以支持“程序特定的”控制,该“程序特定的”控制允许一定程度上控制哪些程序将迁移到新的/不同的版本以及哪些程序不应该迁移。一般而言,系统 500 包括软件组件 502 和关联的绑定库 504。绑定库 504 实现为两个不同的组件,包括加载器组件 506 和绑定组件 508。在此方面,软件组件 502、绑定库 504、加载器组件 506 和绑定组件 508 基本上与参考图 1-4 中任何一个或多个图描述的同名组件类似。

[0072] 一般而言,当执行应用编程接口调用时,所有软件组件 502 可倾向于迁移到指定的当前库版本,通常是最新版本。但是,根据本发明的进一步方面,提供了对于哪些程序应该迁移到当前版本以及是否存在某些不应该迁移的程序的控制。

[0073] 如图所示,图 5 可以包括共享库版本表 530,它与参考图 4 描述的共享库版本表 430 类似。但是,所示实现进一步包括共享程序版本表 540。共享程序版本表 540 存储多个软件组件以及每个应用的相应库版本。例如,如图所示,第一软件程序“UPDATE”被配置为

使用库版本 2。但是,软件程序“REPORT”被配置为使用库版本 3。

[0074] 因此,如果调用加载器组件 506 的软件组件 502 是软件程序 UPDATE,则加载器组件 506(当被与相应判定逻辑关联的适当 API 触发时)确保在处理调用之前先加载库版本 2。相应地,如果调用加载器组件 506 的软件组件 502 是软件程序 REPORT,则加载器组件 506(当被与相应判定逻辑关联的适当 API 触发时)确保在处理调用之前先加载库版本 3。通过对共享程序版本表 540 进行中央控制,管理组件可以判定何时个别程序应该开始迁移到新的或不同的库版本。

[0075] 程序库版本表 540 将程序名(或与程序名匹配的模式等)映射到库版本。程序库版本表 540 由适当的管理进程(例如,图 3 示例中的队列管理器 328 或其他任何适当的监督进程)进行填充。而且,程序库版本表 540 包含具有关联的特定库版本的程序的名称或名称模式。加载器组件 506 在做出版本升级或降级判定之前,先在程序库版本表中找到调用程序的名称。

[0076] 可以通过使指向最新可用或当前库版本的单行与所有程序名匹配(例如使用通配符或其他适合的语言),扩展加载器组件 506 的程序特定的判定逻辑,例如以便迁移所有程序。所述实现可要求所有程序与程序库版本表中的至少一行匹配,或者例如在程序名不与任何行匹配的情况下,实现可以简单地默认为预定版本或先前版本。

[0077] 封装

[0078] 已参考使用动态加载库的软件实现描述了本发明的各方面。在此方面,传统上使用程序的软件实现可以被封装到动态可加载库,而非以传统程序格式封装。一旦经过封装,便可使用此处参考图 1-5 更全面描述的各种软件修改技术。例如,可以通过将用于应用的联机事务处理(OLTP)系统的大部分或全部封装到动态可加载库而非程序中来实现动态软件版本管理。完成此操作后,便可在不关闭与 OLTP 系统交互的程序的程序的情况下加载和使用新的 OLTP 软件。完全不中断与 OLTP 系统无关的程序功能,同时使用新软件升级与 OLTP 系统相关的程序功能。

[0079] 通过将 OLTP 功能体现为由用户业务程序或 OLTP 系统本身使用的一组库,可以在客户端环境和许多服务器端环境中实现用于零宕机升级的动态软件库选择。在此,如在此将更详细地描述那样,通过允许程序在其自己选择的时间逻辑上与 OLTP 系统断开连接,动态地卸载现有 OLTP 软件(库),动态地加载新的 OLTP 软件(库),然后使用新软件重新连接到 OLTP 系统来实现支持零中断升级。

[0080] 尽管严格来讲,在执行卸载旧库和加载新库时会中断服务,但是所述中断很短暂并且仅限于执行这些步骤的特定客户机进程。而且,选择要使用的库/软件版本不再是操作系统操作,而是被置于 OLTP 软件套的直接控制之下。OLTP 软件可以执行任何所需的验证,并且可以实现较仅依赖于底层操作系统而言更为复杂的加载技术和策略。此外,还可以使用此方法回退到早期版本。

[0081] 示例性计算机系统

[0082] 现在参考图 6,其示出根据本发明的方面实现为计算机系统 602 的数据处理系统 600 的方块图。计算机系统 602 可与很多种通用或专用计算系统环境或配置一起操作。而且,计算机系统 602 可以在计算机系统执行的计算机系统可执行指令(例如,程序模块)的一般上下文中进行描述。通常,程序模块可以包括执行特定任务或实现特定抽象数据类型

的例程、程序、对象、组件、逻辑、数据结构等。

[0083] 计算机系统 602 的组件可以包括但不限于一个或多个处理器或处理单元 604，例如对称多处理器 (SMP) 系统或包括与系统总线相连的多个处理器的其他配置。替代地，可以采用单处理器。计算机系统 602 的组件进一步包括系统存储器 606 以及将包括系统存储器 606 的各种系统组件连接到处理器 604 的总线 608。存储器 606 可以包括形式为易失性存储器的计算机系统可读介质，例如随机存取存储器 (RAM) 610 和 / 或高速缓冲存储器 612。存储器 606 也可以包括其他可拆装 / 不可拆装的、易失性 / 非易失性的计算机系统存储介质，例如诸如一个或多个硬盘驱动器之类的存储系统 614。存储器 606 还可以包括至少一个上面包含计算机可读程序代码的计算机可读存储介质 616，所述计算机可读程序代码包括被配置为执行此处更全面描述的本发明的实施例的功能的计算机可读程序代码。

[0084] 计算机系统 602 还可以包括跨总线 608 与处理器 604 进行通信的输入 / 输出 (I/O) 接口 618 和 / 或网络适配器 620。计算机系统 602 还可以与外部设备 622 和 / 或显示器 624 接口连接。其他硬件和 / 或软件组件可以与计算机系统 602 结合使用。

[0085] 所述数据处理系统可以例如包括 IBM RS/6000 系统，这是由位于纽约阿莫克的国际商业机器公司推出的一款运行 Advanced Interactive Executive (AIX) 操作系统的产品。诸如 Java 之类的面向对象的编程系统可以与操作系统一起运行并提供从数据处理系统上执行的 Java 程序或应用到操作系统的调用。

[0086] 本领域的技术人员将理解，本发明的方面可以实现为系统、方法或计算机程序产品。因此，本发明的方面可以采取完全硬件实施例、完全软件实施例（包括固件、驻留软件、微代码等）或组合了软件和硬件方面的实施例的形式，所有这些软件和硬件方面在此一般地被称为“电路”、“模块”或“系统”。此外，本发明的方面可以采取体现在一个或多个计算机可读介质（在介质中具有计算机可读程序代码）中的计算机程序产品的形式。

[0087] 可以使用一个或多个计算机可读介质的任意组合。所述计算机可读介质可以是计算机可读信号介质或计算机可读存储介质。计算机可读存储介质例如可以是（但不限于）电、磁、光、电磁、红外线或半导体系统、装置或设备或上述各项任何适合的组合。计算机可读存储介质的更具体的示例（非穷举列表）可以包括以下项：具有一条或多条线的电连接、便携式计算机软盘、硬盘、随机存取存储器 (RAM)、只读存储器 (ROM)、可擦写可编程只读存储器 (EPROM 或闪存)、光纤、便携式光盘只读存储器 (CD-ROM)、光存储设备、磁存储设备或上述各项任何适合的组合。在本文档的上下文中，计算机可读存储介质可以是任何能够包含或存储由指令执行系统、装置或设备使用或与所述指令执行系统、装置或设备结合的程序有形介质。

[0088] 计算机可读信号介质可以包括其中包含计算机可读程序代码（例如，在基带中或作为载波的一部分）的传播数据信号。此类传播信号可以采取任何多样的形式，包括但不限于电磁、光或上述各项任何适合的组合。计算机可读信号介质可以是任何并非计算机可读存储介质以及传送、传播或传输由指令执行系统、装置或设备使用或与所述指令执行系统、装置或设备结合的程序的可读介质。

[0089] 可以使用任何适当的介质（包括但不限于无线、线缆、光缆、RF 等或上述任何适合的组合）来传输计算机可读介质中包含的程序代码。

[0090] 用于执行本发明的方面的操作的计算机程序代码可以使用包含一种或多种编程

语言的任意组合来编写,所述编程语言包括诸如 Java、Smalltalk、C++ 或类似语言之类的面向对象的编程语言或者诸如“C”编程语言或类似的编程语言的常规过程编程语言。所述程序代码可以完全地在用户计算机上执行,部分地在用户计算机上执行,作为独立的软件包执行,部分地在用户计算机上并部分地在远程计算机上执行,或者完全地在远程计算机或服务器上执行。在后者的情况中,所述远程计算机可以通过包括局域网(LAN)或广域网(WAN)的任何类型网络与用户的计算机相连,也可以与外部计算机进行连接(例如,使用因特网服务提供商通过因特网连接)。

[0091] 此处参考根据本发明的实施例的方法、装置(系统)和计算机程序产品的流程图和/或方块图对本发明的方面进行了描述。将理解,所述流程图和/或方块图的每个方块以及所述流程图和/或方块图中的方块的组合可以由计算机程序指令来实现。这些计算机程序指令可以被提供给通用计算机、专用计算机或其他可编程数据处理装置的处理器以产生机器,以便通过所述计算机或其他可编程数据处理装置的处理器执行的所述指令产生用于实现在一个或多个流程图和/或方块图方块中指定的功能/操作的装置。

[0092] 这些计算机程序指令也可以被存储在引导计算机、其他可编程数据处理装置或其他设备以特定方式执行功能的计算机可读介质中,以便存储在所述计算机可读介质中的所述指令产生一件包括实现在所述一个或多个流程图和/或方块图方块中指定的功能/操作的指令的制品。

[0093] 所述计算机程序指令还可被加载到计算机、其他可编程数据处理装置或其他设备,以导致在所述计算机、其他可编程装置或其他设备上执行一系列操作步骤以产生计算机实现的过程,从而在所述计算机或其他可编程装置上执行的指令提供用于实现在一个或多个流程图和/或方块图方块中指定的功能/操作的过程。

[0094] 附图中的流程图和方块图示出了根据本发明的各种实施例的系统、方法和计算机程序产品的可能实施方式的架构、功能和操作。在此方面,所述流程图或方块图中的每个方块都可以表示代码的模块、段或部分,所述代码包括用于实现指定的逻辑功能的一个或多个可执行指令。还应指出,在某些备选实施方式中,在方块中说明的功能可以不按图中说明的顺序发生。例如,示为相继的两个方块可以实际上被基本同时地执行,或者某些时候,取决于所涉及的功能,可以以相反的顺序执行所述方块。还应指出,所述方块图和/或流程图的每个方块以及所述方块图和/或流程图中的方块的组合可以由执行指定功能或操作的基于专用硬件的系统或专用硬件和计算机指令的组合来实现。

[0095] 在此使用的术语仅出于描述特定实施例的目的,并非对本发明进行限制。正如在此使用的那样,单数形式“一”、“一个”和“所述”旨在同时包括复数形式,除非上下文另外明确指出。将进一步理解的是,当在本说明书中使用术语“包括”和/或“包含”指定存在所述特性、整数、步骤、操作、元件和/或组件,但并不排除其中存在或增加一个或多个其他特性、整数、步骤、操作、元件、组件和/或由此构成的组。

[0096] 下面权利要求中的所有装置加功能元件或步骤加功能元件的对应结构、材料、操作和等同物旨在包括用于与如具体声明的其他所声明的元件结合执行所述功能的任何结构、材料或操作。出于说明和描述目的给出了对本发明的描述,但是所述描述并非旨在是穷举的或是将本发明限于所公开的形式。在不偏离本发明的范围和精神的情况下,许多修改和变化对于本领域的技术人员来说都将是显而易见的。本发明的各方面的选择和描述是为

了最佳地解释本发明的原理、实际应用,并且当适合于所构想的特定使用时,使得本领域的其他技术人员能够理解本发明的具有各种修改的各种实施例。

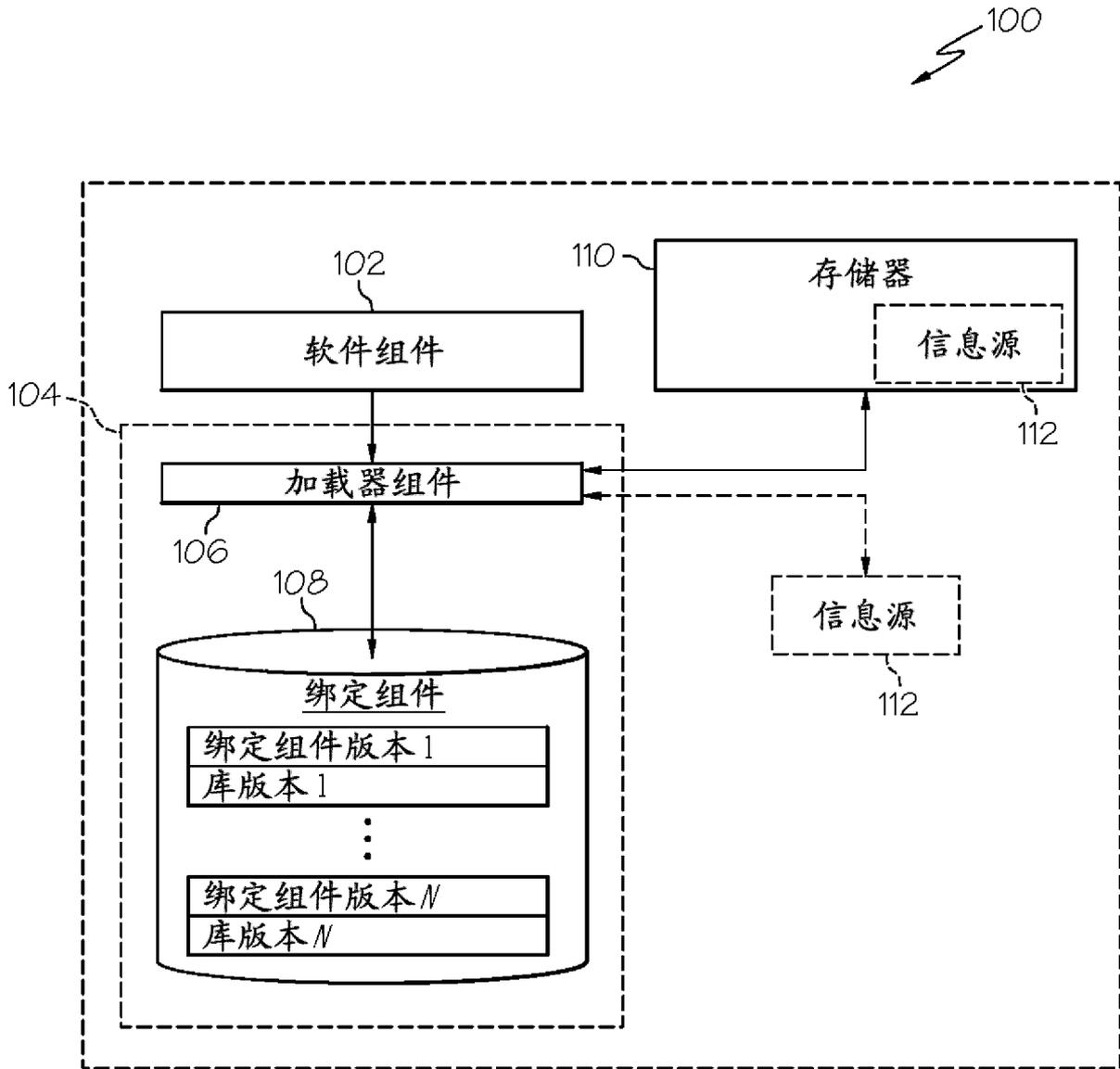


图 1

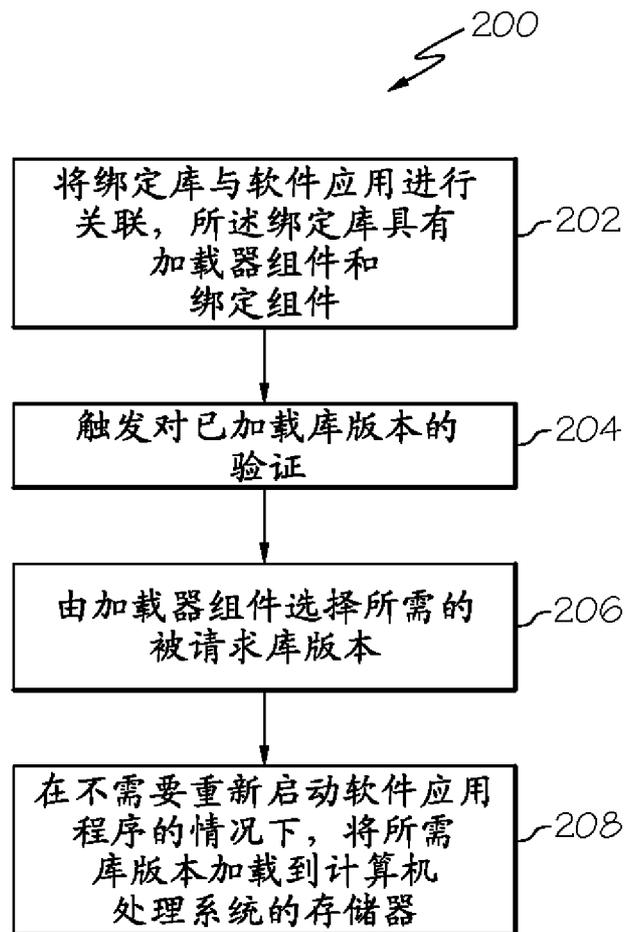


图 2

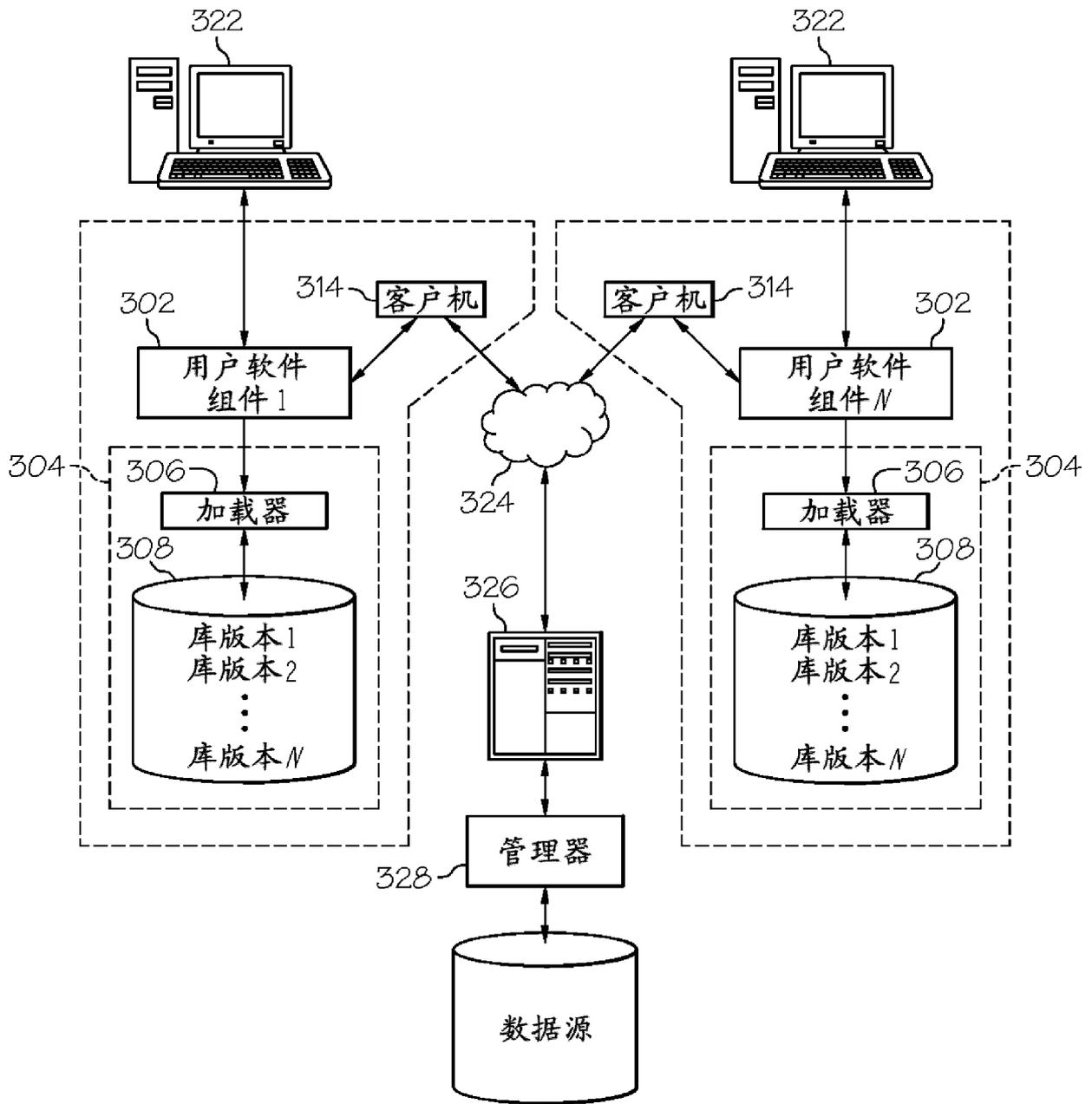


图 3

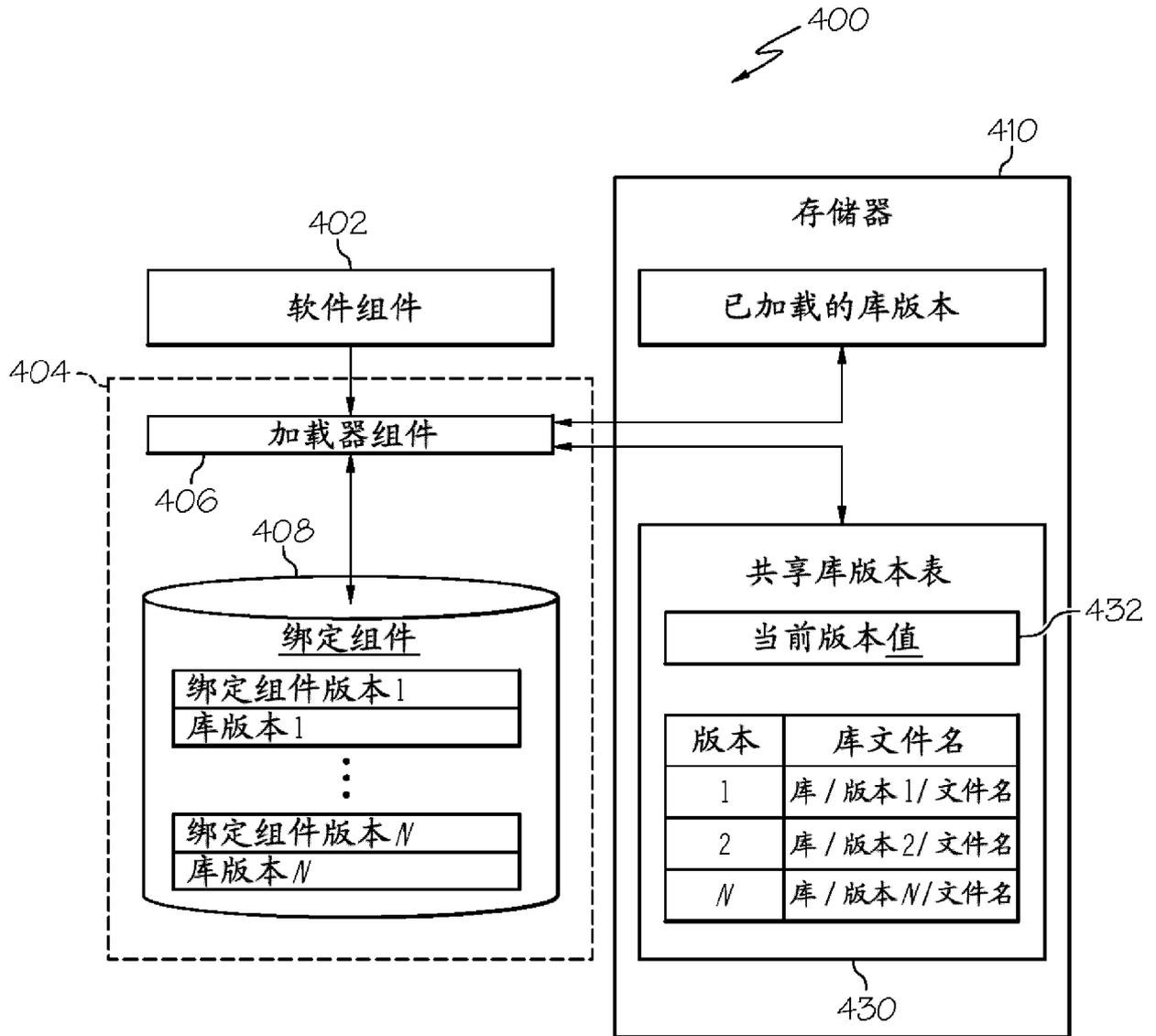


图 4

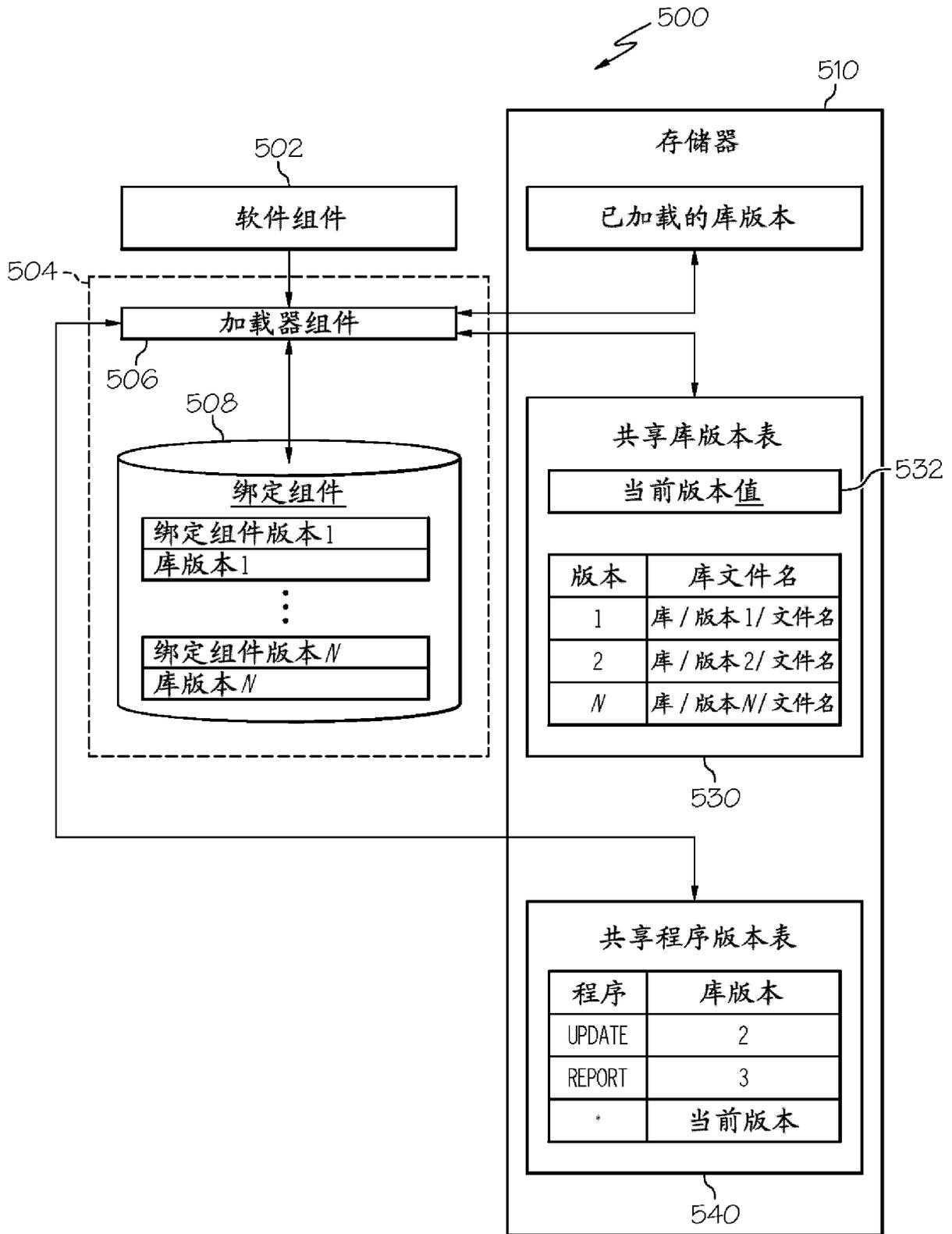


图 5

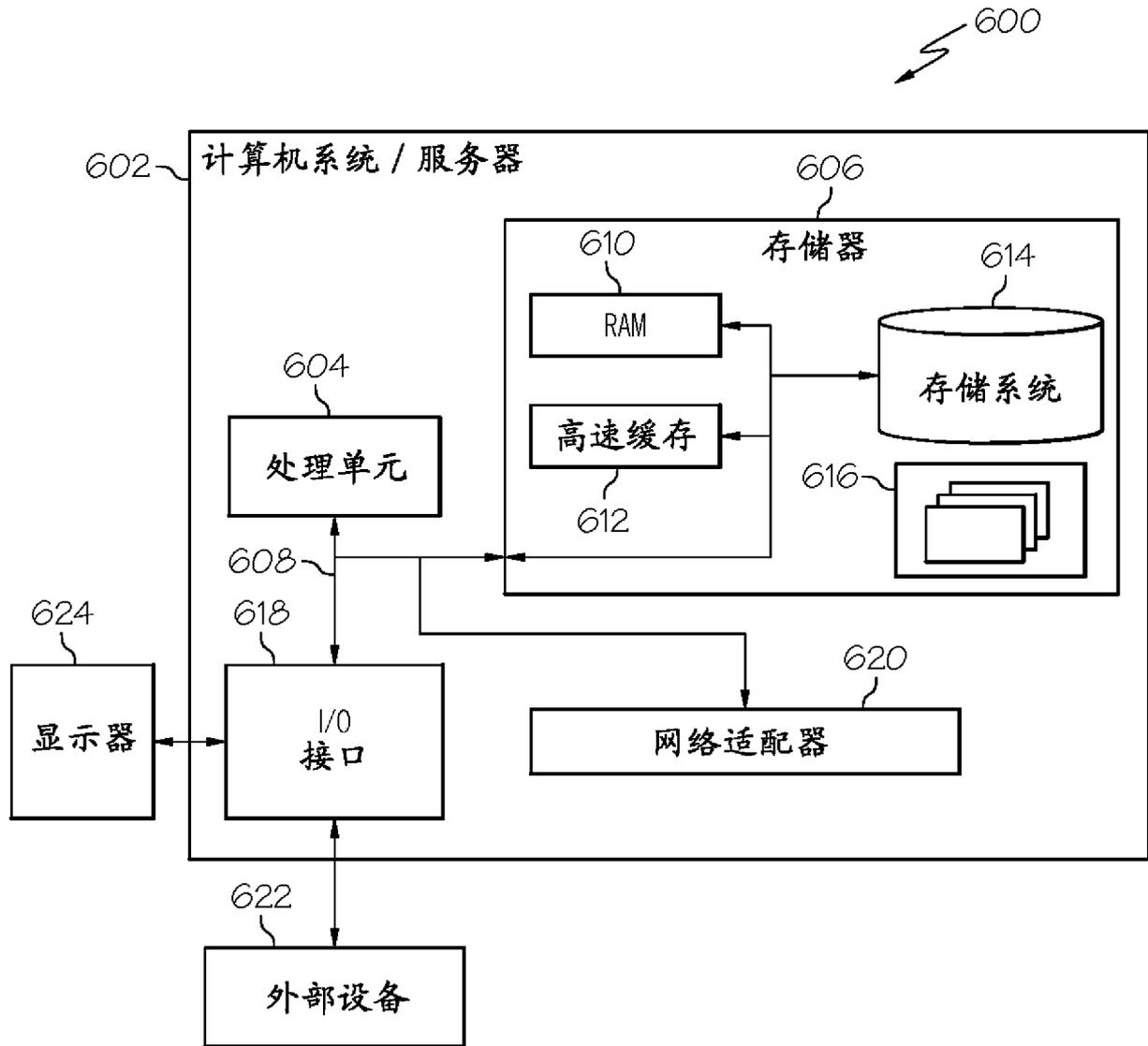


图 6