(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 1554056 B
(45)授权公告日 2010.06.16

(21)申请号 02817719.3

(22)申请日 2002.07.16

(30)优先权数据
60/308,693 2001.07.30 US
10/196,322 2002.07.15 US

(85)PCT申请进入国家阶段日
2004.03.10

(86)PCT申请的申请数据
PCT/US2002/022481 2002.07.16

(87)PCT申请的公布数据
WO2003/009093 EN 2003.01.30

(73)专利权人 BEA 系统公司
地址 美国加利福尼亚州

(72)发明人 爱德华·P·费尔特 普里西拉·冯
亚历山大·J·索莫吉
斯里拉姆·斯里尼范桑

(74)专利代理机构 北京市柳沈律师事务所 11105
代理人 郭定辉 黄小临

(51)Int.Cl.
G06F 15/16(2006.01)

(56)对比文件
US 576489 A,1998.06.09, 第 4 栏第 30-33 行,第 7 栏第 42 至第 51 行,第 8 栏第 10 至 32 行.

审查员 李琼

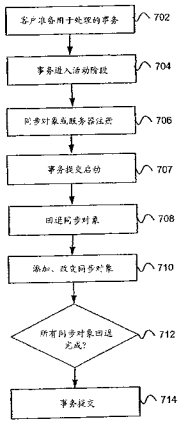
权利要求书 2 页 说明书 16 页 附图 14 页

(54)发明名称

具有同步回退处理特征的事务处理的系统和方法

(57)摘要

本发明提供一种在事务或者消息完成之前允许事务回退处理被同步或者实施的机制。在事务活动阶段期间(704),可以在任何一个服务器上注册参与事务的附加同步对象。调用提交功能(commit function)之后,在事务预准备阶段处理期间(708)调用每个同步对象。一个同步对象可以注册(708、710、712)另一个同步对象,调用未预先包含在事务中的服务器,或者把一个新 XA 资源并入事务中。本发明的效果是提供一种请求传送的循环方案,它允许在提交(714)之前把附加资源加入到(710)环中。



CN 1554056 B

1. 一种将事务回退与事务完成同步的事务处理系统，包括：

事务日志器，由事务管理器在该事务日志器中记录事务；

同步对象和资源的初始集合；

服务器链，在其上存在所述同步对象和资源的初始集合；以及事务协调器，在第一服务器上执行，用于注册所述同步对象和资源的初始集合和服务器链，其中，事务子协调器在该服务器链上执行；

接收请求来提交所述事务；

处理整个事务预准备阶段的事务，包括

采用对所述链上的所述事务子协调器之一的开始预准备调用来启动所述预准备处理，

允许所述事务子协调器动态地添加附加同步对象和资源到已注册的资源的初始集合中，包括当所述附加同步对象和资源处于附加服务器中时，将该附加服务器添加到所述链中，修改所述事务以反映所述链中附加服务器和在其上的附加同步对象和资源，

按照需要以循环流形式重复所述预准备步骤，直到所述链中的所有服务器已经完成所述预准备阶段为止，以及

将预准备确认返回到所述事务协调器。

2. 根据权利要求 1 所述的系统，其中，所述事务协调器是从所述事务子协调器中选择的。

3. 根据权利要求 2 所述的系统，其中，每个所述事务子协调器能够从客户接收请求来处理事务，并且对于特定事务请求，被选择为所述事务协调器的事务子协调器是从客户接收请求的事务子协调器。

4. 根据权利要求 1 所述的系统，其中，在将所述预准备确认从所述链中最后被调用的服务器发送到第一服务器之前，能够在所述预准备阶段期间的任何时间将附加服务器到事务中。

5. 一种将事务回退与事务完成同步的事务处理方法，包括以下步骤：

在事务日志器中记录事务；

确定同步对象和资源的初始集合以及在其上存在所述同步对象和资源的初始集合的服务器链；

从参与该事务的事务管理器中选择的事务协调器；

注册所述同步对象和资源的初始集合和服务器链；

接收请求来提交所述事务；

在事务协调器上处理整个事务预准备阶段的事务，包括

采用对所述链上的所述事务子协调器之一的开始预准备调用来在事务协调器上启动所述预准备处理，

允许所述事务子协调器动态地添加附加同步对象和资源到已注册的资源的初始集合中，包括当所述附加同步对象和资源处于附加服务器中时，将该附加服务器添加到所述链中，

修改所述事务以反映所述链中附加服务器和在其上的附加同步对象和资源，

按照需要以循环流形式重复所述预准备步骤，直到所述链中的所有服务器已经完成所述预准备阶段为止，以及

将预准备确认返回到所述事务协调器。

6. 根据权利要求 5 所述的方法，其中，所述事务协调器是从所述事务子协调器中选择的。

7. 根据权利要求 6 所述的方法，其中，每个所述事务子协调器能够从容户接收请求来处理事务，并且对于特定事务请求，被选择为所述事务协调器的事务子协调器是从客户接收请求的事务子协调器。

8. 根据权利要求 5 所述的方法，其中，在将所述预准备确认从所述链中最后被调用的服务器发送到第一服务器之前，能够在所述预准备阶段期间的任何时间将附加服务器到事务中。

具有同步回退处理特征的事务处理的系统和方法

技术领域

[0001] 本发明一般涉及应用程序和事务服务器，特别是涉及一种允许回退 (callback) 处理与事务或者消息的完成相同步的系统。

[0002] 优先权要求

[0003] 本申请要求以下美国申请的优先权：于 2002 年 7 月 15 日提交的申请号为 10/196,322 的美国实用新型申请“具有同步回退处理特征的事务处理的系统和方法”，以及于 2001 年 7 月 17 日提交的申请号为 60/306105 的美国临时申请“具有同步回退处理特征的事务处理的系统”，以及于 2001 年 7 月 30 日提交的申请号为 60/308693 的“具有同步回退处理特征的事务处理的系统”，上述申请在此引用，以资参考。

[0004] 交叉参考

[0005] 本申请涉及共同待审的实用新型申请“具有事务特性特征的事务处理的系统 (SYSTEM AND METHOD FOR TRANSACTION PROCESSING WITHPROPERTY FEATURE)”（申请号 10/196,687，发明人：Edward P.Felt，PriscillaFung，Alex Somogyi，Sriram Srinivasan；2002 年 7 月 15 日提交），和“具有委托提交特征的事务处理的系统 (SYSTEM AND METHOD FOR TRANSACTIONPROCESSING WITH DELEGATED COMMIT FEATURE”（申请号 10/196,297，发明人：Edward P.Felt，Priscilla Fung，Alex Somogyi，Sriram Srinivasan，2002 年 7 月 15 日提交）；在此对这两件申请整体引用，以资参考。

背景技术

[0006] Java 2 平台企业版 (J2EE) 规范定义了开发多层企业应用程序的一个当前标准。J2EE 为企业应用程序的设计、开发、装配和部署提供了一种基于组件的方法，既降低了成本又能够更快速设计和实施。J2EE 向开发者提供了多层分布式应用程序模型、再利用组件的能力、统一标准的安全模型以及灵活的事务控制。它们不仅能够比以往更快地向市场提交创新的顾客方案，而且还不把所得到的平台独立的 J2EE 基于组件的方案捆绑到任何一个卖方的产品和应用程序接口上。

[0007] J2EE 规范定义了以下各种组件：应用程序客户组件；企业 JavaBeans(EJB)；servlet（小服务程序）和 Java 服务器页 (JSP)（还称之为 Web 组件）；小程序 (applet)。多层分布式应用程序模型是指根据功能将应用程序划分成多个组件，并且不同的应用程序组件可以构成相同或者不同服务器上的 J2EE 应用程序。实际安装应用程序组件依赖于应用程序组件属于多层 J2EE 环境中的哪个层。这些层在图 1 中示出。如图所示，应用程序服务器层 4 用来开发 EJB 容器和 / 或表现容器如 servlet、JSP 和 html 页 14。这些容器又用作客户层 2 与尾端层 6 之间的接口，其中在客户层 2 中部署客户 8 和客户应用程序，而尾端层 6 用来担当企业或传统应用程序如企业资源规划 (ERP) 系统的宿主。

[0008] 客户层 - 这些可以是浏览器、基于 Java 的程序，或者运行于客户层内并位于公司防火墙内侧和外侧的其它 Web 使能编程环境。

[0009] 应用程序服务器层 - 该层通常担当支持客户请求的表现逻辑和商务逻辑组合的

宿主。表现逻辑经由显示 HTML 页的 JSP 页和 servlet 来支持，而商务逻辑经由远程方法调用 (RMI) 对象和 EJB 12 来支持。EJB 依赖事务的容器环境、寿命周期和状态管理、资源组合、安全性等，它们共同构成运行 bean 的运行时间环境。

[0010] 尾端层 - 这通常是现存应用程序和数据存储的组合。它还被称之为企业信息系统 (EIS) 层，因为它可以包含如企业资源规划 (ERP)、主机事务处理、数据库系统和其它老式信息系统那样的系统。

[0011] 由于 J2EE 应用程序的各组件分别运行，并且常常是不同装置，因此需要有一种查找和引用其它码和资源的客户及应用程序服务器层码的方式。客户和应用程序码例如可以使用 Java 命名和目录接口 (JNDI)16 查找用户定义的对象如企业 bean 和环境项目如 Java 数据库连接器 (JDBC) 数据源对象位置，而环境项目又用来查找尾端层中的资源以及消息连接。

[0012] 在 Web 和企业 bean 组件上的部署时间，可以配置应用程序行为如安全性和事务管理。该部署时间特征将应用程序逻辑与可能随组装变化的配置设置相分离。J2EE 安全性模型使开发商配置一个 Web 或者企业 bean 组件，以便系统资源只能由授权用户访问。例如，Web 组件可以被配置成提示输入用户姓名和口令。企业 bean 组件可以被配置成仅仅使特定组中的人可以调用某种方法。作为选择，servlet 组件可以被配置成具有某些可以访问每个人的方法，以及具有一些在一个组织中只有某些被特许的人才可访问的方法。可以为所有方法都能用于每个人或者所有方法仅能用于一些选择的人的另一环境配置相同的 servlet 组件。

[0013] 某些应用程序服务器，比如产自 BEA 系统公司 (San Jose，california) 的 WebLogic 服务器，使用了一个访问控制目录 (ACL) 机构，该机构考虑了运行在服务器上的组件使用的细颗粒控制 (fine-grained control)。通过使用 ACL，开发商可以以 Java 方法等级定义哪个用户或者哪些用户组可以执行什么或者不可以执行什么。该 ACL 机构覆盖了除 EJB 外的运行在应用程序服务器上的任何事，该 EJB 具有 EJB 规范所定义的它们自己的访问控制机构。安全区域允许管理器把来自现有的授权或确认系统的信息输入到 ACL 中。

[0014] Java Servlet

[0015] servlet 是一个扩展 Web 服务器的功能的程序。servlet 接收来自客户的一个请求，动态地生成答复 (也许查询数据库以履行该请求)，然后向客户发送包含 HTML 或 XML 文件的答复。Servlet 类似于 CGI 但通常更适合于写，因为 servlet 使用 Java 类别和流。它们运行更快，因为 Servlet 被编译成 Java 字节码，并且 servlet 事件在运行时间被保存在存储器中，每个客户请求运行在单独线程中。servelt 使它容易地以动态方式产生到达 HTTP 答复流的数据。客户请求在新连接结束后来执行，所以流控制不会自然干预请求。为此，该对话管理保持请求之间的特定客户的状态。在某些应用程序服务器中，诸多 sevlet 利用 HTTP 对话对象保存它们的方法请求之间的状态。为了克服故障 (failover) 目的，可以在群集环境 (clustered environment) 中复制该对象。

[0016] Java 服务器页面

[0017] JSP 页面是一个开发 servlet 的基于文本的表达中心途径 (way)。JSP 页面提供了 servlet 的所有利益，当与 JavaBeans 类组合时，它提供了一种容易的方式来保持内容并显示逻辑分离。JSP 页面和 servlet 比公共网关接口 (CGI) 更理想，因为它们是独立平台，并

且使用少量开销。JSP 页面可以和 JavaBeans 一起用来定义 Web 模板，该模板建立具有相似外貌和感觉的由多个页面构成的 Web 站。JavaBeans 类执行数据再现，所以模板没有 Java 码。这意味着它们可以由 HTML 编辑器来保持。使用 JSP 页面的简单的基于 Web 的应用程序可以用来把内容捆绑到使用替代 JavaBeans 类的自定义标记或 scriptlet 的应用程序逻辑上。自定义标记被装入到标记库中，而该标记库被输送到一个 JSP 页面中。Scriptlet 是直接装入 JSP 页面的小 Java 码段。

[0018] 数据库访问业务 (JDBC)

[0019] JDBC 充当一个抵达关系数据库的桥，并且效法 ODBC（开放式数据库连接）规范。通过使用驱动器断开数据库与程序码的连接。JDBC 的某些实施提供了对高级数据型的支持，并且还支持滚动结果集和批更新的功能。

[0020] Java 消息传递业务 (JMS)

[0021] JMS 是用来支持 Java 程序间消息交换的 J2EE 机制。Java 就是这样支持异步通信的，其中发送方和接收方不需要彼此知道，因而可以独立操作。JMS 支持以下两种消息传递模型：

[0022] 点对点 - 基于消息队列。在该模型中，消息制作商把消息发送到队列上。消息顾客可以把该消息附加到队列上以便收听消息。当消息到达队列时，用户可以将消息从队列中取出，并对此答复。可以只向一个队列发送消息，并且将只由一个顾客使用该消息。顾客可以任意过滤消息，以选出它们想要的正确消息类型。

[0023] 公布和预定 - 允许消息制作商发送消息到一个专题，该专题的所有注册用户检索这些消息。在此情况下，许多顾客可收到相同的消息。

[0024] Java 接口定义语言 (IDL)

[0025] CORBA 对象使用 IDL 指定一个契约，即它们如何与其它对象进行相互作用。有了 Java IDL，就可以在 Java 领域和 CORBA 领域之间定义契约。自 SUN 的 JDK1.2 开始，都包括一个 ORB，它允许 Java 应用程序经由互联网 InterORB(IIOP) 协议调用远程 CORBA 对象。

[0026] 企业 JavaBran(EJB)

[0027] EJB 组件为封装商务逻辑而设计，所以开发商不必关心典型作业如数据访问、事务支持、安全、高速缓存和并行操作的代码编程。在 EJB 规范中，这些作业是 EJB 容器的职责。企业 bran 由接口和类别组成。客户经由企业 bean 的本地和远程接口访问企业 bean 方法。本地接口提供创建、消除和定位企业 bean 的方法，远程接口提供商务方法。在部署时，容器创建源自这些接口的类别，然后利用该类别向客户提供访问，使客户寻找创建、消除、定位和调用的企业 bean 的商务方法。企业 bean 类提供商务方法、创建方法以及探测器 (finder) 方法的实施；如果 bean 管理它自己的持续性，则提供其寿命周期方法的实施。

[0028] 有两种类型的企业 bean：实体 bean 和会话 bean。会话 bean 表达与客户的瞬态对话，并且可以执行数据库读和写。会话 bean 可以调用 JDBC 调用本身，或者可以使用实体 bean 进行调用，在此情况下，会话 bean 是实体 bean 的一个客户。会话 bean 的字段包含对话状态并且是瞬态的。如果服务器或者客户崩溃，则会话 bean 完结。

[0029] 实体 bean 表达数据库中的数据以及作用于该数据的方法。在用于一个雇员信息表的关系数据库上下文中，可能存在用于该表每一行的一个 bean。实体 bean 是事务性的和长寿的。只要数据保持在数据库中，实体 bean 就存在。该模型可以容易地用于关系数据

库，并且不限于对象数据库。

[0030] 会话 bean 可以是有状态的或者无状态的。有状态会话 bean 包含代表客户的对话状态。对话状态是会话 bean 瞬态字段值加之可以从会话 bean 字段得到的所有对象。有状态 bean 不表达持久数据存储中的数据，但是它们可以访问和更新代表客户的数据。无状态会话 bean 没有任何用于特定客户的状态信息。它们通常提供不保持任何特定状态的服务器侧行为。无状态会话 bean 需要少量的系统资源。提供属类业务或者表达已存数据的共享意图的商务对象是一个无状态会话 bean 的良好候选者。

[0031] 使用容器管理持久性访问关系数据库的企业 bean 不需要开发商使用任何用于数据库访问的 JDBC2.0API，因为这由容器处置。然而，如果 bean 管理持久性被用来访问一个不同于关系数据库的企业信息系统，或者如果存在访问一个不同于关系数据库的企业信息系统之需要，则必须提供适当的代码。

[0032] 在企业 bean 使用 bean 管理持久性访问数据库的情况下，具有 JDBC2.0API 码的 bean 的寿命周期方法必须被实施为处理数据装载和存储数据，以及保持运行时间与持久数据库存储之间的一致性。尽管 Web 层使用 HTTP 或者 HTTPS 在层之间传递数据，但是 EJB 层使用了 RMI-IIOP。RMI-IIOP 是满标度分布式计算协议，该协议向任何客户或 Web 层提供访问企业 bean 的程序，以直接访问 EJB 层中的业务。这些业务包括引用企业 bean 的 JNDI、发送和接收异步消息的 Java 消息传递业务 (JMS) 和用于关系数据库访问的 JDBC。

[0033] 事务管理

[0034] 任何应用程序服务器如 WebLogic 服务器系统的最基本特征之一是事务管理。事务是保证数据库事务被精确完成并使其呈现高性能事务的所有“ACID”特性的手段，包括：

[0035] 原子性 - 事务对数据库作出的所有变化都是永久性的；否则，返回所有变化。

[0036] 一致性 - 一个成功的事务把数据库从以前的活动状态转换到新的活动状态。

[0037] 隔离 - 在事务完成其工作之前，对数据库作出的变化是其它操作看不到的。

[0038] 持久性 - 事务对数据库作出的变化在系统或媒介以后出现故障时，仍然有效（保存）。

[0039] J2EE 事务模型使应用程序开发商在部署时间，指定包括单一事务的多个方法间的关系，所以把一个事务中的所有方法视为一个单一单元。这是理想的，因为事务是必须全部完成的一系列步骤，或者，如果这些步骤没有全部完成，则倒退这些步骤。例如，开发商可以有企业 bean 中的一系列方法，即通过将资金记入借方的第一帐户，然后存入第二帐户，把资金从一个银行帐户转移到另一个帐户。在该实例中，它们希望把实体转移操作视为一个单位，所以，如果在记入借方之后和存入之前存在一个故障，则回退记入借方操作。

[0040] 在安装期间，事务特性被指定到应用程序组件上，允许开发商经过应用程序组件把诸多方法集合成事务。这样，可以在 J2EE 应用程序内改变应用程序组件，并且重新分配事务特性而不改变代码。Java 事务业务 (JTS) 和 Java 事务 APJ(JTP) 在 J2EE 中构成事务支持的基础，特别是构成对 EJB 和 JDBC2.0 的事务支持的基础。JTS 规范是用于事务管理的低级应用程序接口 (API)，其中该事务管理把 Java 映射到对象管理组 (PMG) 对象事务业务。JTA 规范由 Sun Microsystems 与事务处理和数据库系统业界的主要工业伙伴共同开发，该规范指定了事务管理器、资源管理器、应用程序服务器与事务应用程序之间的标准 Java 接口。具体而言，JTA 是一个包括以下两个部分的高级 API：

[0041] 事务接口－它能够使分布式组件执行的工作受全局事务制约，并且是标记或者标识构成事务的操作组的一个通道。

[0042] XA 资源接口－一个基于 X/Open 接口或者 XA 接口的接口，该接口能够启动分布式事务的处理。这些接口都包含对一个以上资源（如一个数据库或者队列之内或者之间）上的事务的协调。

[0043] 开发商通常不需要关心用 JTA 编程显性事务，因为该工作是通过由容器处理的 JDBC 和 EJB APL 执行的，并且通过应用程序部署说明符来配置。开发商可以把精力集中到事务的设计上，而不是集中到事务的实现上。

[0044] Weblogic 服务器支持企业应用程序的分布式事务以及两阶段提交协议。分布式事务是以协调方式更新多资源管理器（比如多个数据库）的事务。相反，一个本地事务更新单一资源管理器。两阶段提交协议是在两个或多个资源管理器上协调单一事务的方法。它是这样保证数据完整性的，通过确保事务更新被提交到所有参与的数据库中，或者从所有数据库完全回退事务更新，使其还原到事务开始之前的状态。换言之，更新所有参与的数据库，或者不更新它们。分布式事务包括以下参与者：

[0045] 事务始发器－启动事务。事务始发器可以是一个用户应用程序、一个企业 JavaBean 或者 JMS 客户。

[0046] 事务管理器－管理代表应用程序的事务。事务管理器通过与正在参与这些事务的资源管理器进行通信，来协调来自应用程序的开始和完成事务的命令。当资源管理器在事务处理期间发生故障时，事务管理器协助资源管理器决定提交还是回退待决事务。

[0047] 可恢复的资源－提供数据的持久性存储。该资源通常是一个数据库。

[0048] 资源管理器－提供对信息和处理的集合的访问。

[0049] 事务察觉 JDBC 驱动器 (transaction-aware JDBC driver) 是公共资源管理器。资源管理器提供事务能力和动作的持久性：它们是分布式事务内访问和控制的实体。资源管理器与特定资源之间的通信被称为一个事务分支 (branch)。

[0050] 两阶段提交协议的第一阶段被称之为准备阶段。所需的更新必须被记录到事务日志文件中，资源必须经过一个资源管理器指示它准备进行改变。资源可以投票提交更新也可以返回到先前状态。第二阶段中出现的事情取决于资源如何投票。如果所有资源都投票提交，则更新参与事务的所有资源。如果一个或多个资源投票为返回，则将参与事务的所有资源返回到它们的先前状态。

[0051] 商务事务的支持

[0052] 这些事务是以下说明的实例情况中的适宜事务（虽然这些情况仅仅是例证而不是穷举）。

[0053] 作为第一实例，客户应用程序需要调用几个对象，这些对象可以包括对一个或多个数据库的写操作。如果任何一个调用不成功，则必须返回写入（写入存储器，更典型的是写入数据库）的任何状态。例如，考虑一个旅游代理应用程序。客户程序需要安排到远地点的旅程；例如，从法国的斯特拉斯堡到澳大利亚的阿利斯斯普林斯。这样一个旅程不可避免需要多个单个飞行预约。客户应用程序通过按顺序预约旅程的每个单个段来工作；例如，斯特拉斯堡到巴黎，巴黎到纽约，纽约到洛杉矶。然而，如果任何一个单独飞行预约不能做到，则客户应用程序需要一个通道取消接近那一点的所有飞行预约。客户应用程序需要与

服务器应用程序管理的对象交谈，并且需要对特定对象实例作出多个调用。交谈也许具有以下的一个或多个特征：

[0054] 在每个连续调用期间或者之后，将数据高速缓存到存储器中或者写入数据库中；在调用结束时，将数据写入数据库中；客户应用程序需要对象在一个存储器内保持每个调用之间的上下文；也就是，每个连续调用使用交谈期间保持在存储器中的数据；在交谈结束时，客户应用程序需要有能力消除交谈期间或者交谈结束时可能出现的所有数据库写操作。

[0055] 作为一个替代实例，考虑一个基于互联网的联机购物车应用程序。客户应用程序的用户浏览联机目录并作出多个购物选择。当用户选择它们想要购买的物品时，用户进行检验并输入其信用卡信息以进行购买。如果信用卡检验失败，购物应用程序需要一个机制来消除购物车中的所有待购买选择，或者回退交谈期间作出的任何购买事务。在对对象的单个客户调用的范围内，对象对数据库中的数据执行多个编辑。如果这些编辑之一失败，则对象需要一个机制回退所有编辑（在此情况下，各数据库编辑不一定进行 EJB 或 RMI 调用。一个客户如一个小程序使用 JNDI 可以获得对事务和事务管理器对象的引用，并开始一个事务。）

[0056] 作为另一个实例，考虑一个银行业应用程序。客户调用对出纳员对象的转帐操作。转帐操作需要出纳员对象对银行数据库作出以下调用：调用关于一个帐户的借记方法；调用关于另一个帐户的借记方法。如果银行数据库的借记调用失败，则银行业应用程序需要一个机制来返回先前的借记调用。

[0057] 上述传统的事务管理方法的问题是，在处理事务返回的方式中效率低下。每个提交服务器必须直接接触每个参与的服务器，然后再通知客户。此外，传统方法缺少灵活性，例如，在调用提交功能之后，没有允许新服务器或资源参与事务的机制。这限制了传统方法的可伸缩性和适用性，并阻碍了今日商务所需的日益增加的复杂应用程序的开发。

发明内容

[0058] 本发明提供了一种供应用程序服务器或者事务处理系统使用的机制，允许在事务完成之前同步（或生效）回退处理。在事务的寿命周期，存在几种状态转换，包括活动、切换、预准备、准备、记录与提交状态之间的转换。根据本发明的一个实施例，在活动阶段期间可以在任何服务器上注册同步对象。在提交 () 功能被调用之后，在准备处理阶段调用每个同步对象。一个同步对象可以注册另一个对象，或者可允许一个同步对象调用一个以前未包含于事务中的服务器。同步对象还可以包含事务中的新 XA 资源。在本文中，术语同步和同步对象是指需要事务管理器在调用提交之后，调用一个完成前同步 (Synchronization.beforeCompletion) 方法的 JTA 规范所指定的编程接口。应用程序码在事务的活动阶段注册这些回退。

[0059] 循环控制流在服务器的一个环中流动，并在完成所有同步回退时结束于提交服务器。如果以后添加新同步对象或者服务器，则可以把它们插入环中。所有同步回退在准备任何资源之前完成，以处理同步回退需要访问资源的这些事件。与提交服务器应当直接接触每个参与的服务器的传统方法相比，该循环控制流处理减少了消息的数量。动态增加服务器和资源、沿循环控制流传送该服务器目录的能力，极大地增加了事务处理系统、应用程

序服务器和建于该服务器顶部的应用程序的灵活性。

[0060] 在实施例中，本发明包括一种将事务回退与一个事务完成相同步的事务处理系统，该系统包括：事务日志器，由事务管理器在该事务日志器中记录事务；同步对象和资源的初始集合；服务器链，在其上存在所述同步对象和资源的初始集合；以及事务协调器，在第一服务器上执行，用于注册所述同步对象和资源的初始集合和服务器链，其中，事务子协调器在该服务器链上执行；接收请求来提交所述事务；处理整个事务预准备阶段的事务，包括采用对所述链上的所述事务子协调器之一的开始预准备调用来启动所述预准备处理，允许所述事务子协调器动态地添加附加同步对象和资源到已注册的资源的初始集合中，包括当所述附加同步对象和资源处于附加服务器中时，将该附加服务器添加到所述链中，修改所述事务以反映所述链中附加服务器和在其上的附加同步对象和资源，按照需要以循环流形式重复所述预准备步骤，直到所述链中的所有服务器已经完成所述预准备阶段为止，以及将预准备确认返回到所述事务协调器。

[0061] 在另一实施例中，本发明包括一种将事务回退与事务完成同步的事务处理方法，包括以下步骤：在事务日志器中记录事务；确定同步对象和资源的初始集合以及在其上存在所述同步对象和资源的初始集合的服务器链；从参与该事务的事务管理器中选择的事务协调器；注册所述同步对象和资源的初始集合和服务器链；接收请求来提交所述事务；在事务协调器上处理整个事务预准备阶段的事务，包括采用对所述链上的所述事务子协调器之一的开始预准备调用来在事务协调器上启动所述预准备处理，允许所述事务子协调器动态地添加附加同步对象和资源到已注册的资源的初始集合中，包括当所述附加同步对象和资源处于附加服务器中时，将该附加服务器添加到所述链中，修改所述事务以反映所述链中附加服务器和在其上的附加同步对象和资源，按照需要以循环流形式重复所述预准备步骤，直到所述链中的所有服务器已经完成所述预准备阶段为止，以及将预准备确认返回到所述事务协调器。

附图说明

[0062] 图 1 显示了 J2EE 体系结构的图示；

[0063] 图 2 显示了本发明实施例的同步回退特征的图示；

[0064] 图 3 显示了供本发明使用的典型客户服务器系统的图示；

[0065] 图 4 显示了本发明实施例的各种事务状态的图示；

[0066] 图 5 显示了本发明实施例的事务同步回退系统的第一图示；

[0067] 图 6 显示了本发明实施例的事务同步回退处理的流程图；

[0068] 图 7 显示了供本发明实施例的一个开始事务寿命周期的图示；

[0069] 图 8 显示了本发明实施例的事务传播寿命周期的图示；

[0070] 图 9 显示了本发明实施例的提交寿命周期的图示；

[0071] 图 10 显示了本发明实施例的预准备阶段寿命周期的图示；

[0072] 图 11 显示了本发明实施例的准备阶段寿命周期的图示；

[0073] 图 12 显示了本发明实施例的提交阶段寿命周期的图示；

[0074] 图 13 显示了本发明实施例的返回寿命周期的图示；

[0075] 图 14 显示了本发明实施例的恢复寿命周期的图示；

[0076] 图 15 图示说明了本发明实施例的事务如何在 EJB 应用程序中工作；以及

[0077] 图 16 图示说明了本发明实施例的事务如何在 RMI 应用程序中工作。

具体实施方式

[0078] 一般来说，本发明提供了供事务或应用程序服务器使用的一种机制，它允许在事务完成之前同步回退处理。在该时期，可以在事务活动阶段把同步对象注册在任何一个服务器上。当调用提交功能时，又在准备阶段调用每个同步对象。这样做的优点是，可以在提交周期之前将同步对象添加到链上。一个同步对象可以注册另一个对象或者可允许一个同步对象调用以前未包含在事务中的服务器。

[0079] 循环控制在服务器之间的一个环中流动，并在所有同步回退完成时结束于一个提交服务器上。如果以后增加新同步对象或者服务器，则将它们插入环中。在准备任何一个资源之前完成所有同步回退，以处理同步回退需要访问资源的事件。与提交服务器应当直接接触每个参与服务器的传统方法相比，循环控制流减少了消息的数量。动态增加服务器和资源、沿循环控制流传送该服务器目录的能力，极大地增加了事务处理系统、应用程序服务器和建于该服务器顶部的应用程序的灵活性。

[0080] 图 2 图示说明了本发明的一个实施例，显示了在提交和回退阶段之前，附加服务器和同步对象如何被包含在同步回退环中。如图 2 所示，包括客户应用程序 104 的客户 102 启动或者开始包含一个服务器 106 或者这些服务器上的资源的事务。通常，客户应用程序 104 将指定一个特定事务上下文 112，该上下文告知诸多服务器其中哪些服务器和哪些服务器资源对象或资源应当被包含在特定事务中。在这里，术语“同步对象”是指允许服务器参与事务的服务器对象。在包括基于 Java 系统的面向对象的系统中有规则地使用同步对象，以保持瞬态状态数据。在事务完成之间，同步对象将数据往返于向相同事务注册的一个或多个资源对象。例如，如图 2 所示，事务上下文 112 可以指定服务器 1 108 和服务器 A 110 将参与的事务，该事务包含一个或多个位于服务器 A 上的对象或资源 116。需要注意的是，在一个实施例中，服务器 106 的任何一个可以选作开始处理链的初始服务器 1。当预准备阶段期间，在提交时期之前将诸多服务器加入到环中。在提交事务之前的任何点上，客户应用程序可以改变事务上下文 122，以增加另一个服务器 B128，或许包括位于该服务器的任何一个同步对象和资源 132。这样，本发明就提供了一种将服务器 B 纳入准备环中的机制。在活动和准备阶段期间还可以增加多个服务器。事务上下文通常不易被客户应用程序直接改变。而是，当应用程序调用其它 J2EE 组件时，J2EE 容器将向事务注册附加同步对象或者征召附加 XA 资源对象；例如 (1) 当应用程序调用 EJB 时，EJB 容器将向代表 EJB 的事务管理器注册同步对象，或者 (2) 当应用程序经由 JDBC 访问数据库时，Weblogic 连接池将向事务管理器征召与数据库事件关联的 XA 资源。

[0081] 本领域熟练技术人员将会明白，尽管第一服务器 1 108 在图 2 中被标识为事务的协调器，但是任何一个服务器都可以承担这一给定事务的任务，并且可以使用各种不同方式选择被选作事务协调器的服务器，所述各种不同方式包括只选择客户所接触的第一服务器。事务协调器可以参与或者不参与提供供事务使用的实际资源。由此得到的本发明特征之一是可以修改客户应用程序以在任何时间点改变事务上下文，并立即实现同步对象或者资源的改变。例如，如果客户应用程序是 EJB，则可以在准备或提交阶段期间修改事务上下

文，将不同资源包含到不同服务器上，并且把这些资源动态地输入到处理链中。这允许在修改客户应用程序时有极大的灵活性。

[0082] 图 3 显示了供本发明使用的典型客户服务器系统。共同称之为客户应用程序的远程装置或应用程序 202 与应用程序服务器 204 通信，特别是与担任应用程序服务器宿主的应用程序通信。在客户端，客户应用程序使用事务管理器 206 发送和接收到达服务器（或者到达许多服务器）的事务 208。同样，在服务器端，事务管理器 210 管理事务。在一个实施例中，服务器还可以包含：一个事务协调器 214，用于协调事务提交处理；子协调器 216，用于参与事务以及执行各个阶段的提交处理，其中对于向本地事务管理器注册的同步和 XA 资源对象进行提交处理；和事务日志器 218，用于记录事务。当创建事务和处理事务时，将其记录到事务日志 220 中，用于日后审计、引用或者回退目的。本发明提供了允许特性与每个事务关联的事务管理器。客户或者服务器应用程序可以在传送事务 222 以及由此反应时分析这些特性。

[0083] 图 4 显示了本发明实施例的各种事务的图示。图 4 显示了事务寿命周期的各种状态或者阶段，包括活动 302、预准备 306、准备 308、记录 310 和提交 312 状态。在任何事务寿命周期期间，存在几个状态转换，包括活动、切换、预准备、准备、记录和提交状态之间的转换。在活动状态，运行于客户上的应用程序或者应用程序码被设计成在事务的活动执行期间接触几个服务器。客户负责记忆哪些服务器被接触以及对待事务的某些其它细节。根据本发明的一个实施例，所接触的第一服务器被设计为“提交服务器”或者事务协调器。当客户调用“提交”时，将提交事务的实际职责授权和移交给提交服务器。提交服务器除了负责处理提交的实事外，它还是一个常规服务器，在许多事件中，任何一个服务器可以承担提交服务器的任务，哪个服务器实际结束其职责（由此成为提交服务器）取决于实际实施。

[0084] 应用程序服务器通常支持各种应用程序类型和事务类型。J2EE 顺从应用程序服务器应当支持 EJB 和 RMI 应用程序。根据应用程序类型，可以使用各种事务处理方法。Weblogic 服务器产品是一个这样的应用程序服务器产品，并且为了图示说明的目的，在本文中经常描述其实施。本领域熟练技术人员将会明白，本发明提供的系统和方法同样可以与其它事务处理系统一起使用，以及与其它类型的应用服务器一起使用。

[0085] 图 5 示出了使用同步回退处理事务的本发明实施例的事务系统。当客户 602 试图通信或者处理事务时，可以接触多个服务器 604、606、608，以确定哪一个服务器将实际处理事务。根据实施，选择一个服务器充当提交服务器 / 事务协调器，而其它服务器仅被视为参与服务器。为了说明目的，图 5 中只示出了单个提交服务器和两个参与服务器，尽管人们将会明白在真实情况中可以使用许多服务器（并且非常可能用来提供可用性）。同样，尽管显示了一个指定的提交服务器和参与服务器，但是在另一个实施例中，多个服务器中的每个服务器可以充当提交服务器，或者仅充当参与服务器。本发明提供的系统对于所使用的服务器数量和类型是灵活的。一旦作出选择，提交服务器就负责处置提交处理。事务处理仅仅通过客户 602 与提交服务器 604 之间的通信来处置，而不需要客户与参与服务器 606 之间的其它通信。提交服务器本身与参与服务器 606（或服务器 606、608）一起处置事务处置，并且经由预准备、准备、记录和提交步骤移动事务。当提交处理完成后，提交服务器将提交返回到客户。在准备阶段期间的任何时候以及在回退之前，附加服务器或者参与者可以被插入到事务处理中。

[0086] 同步回退的典型实例

[0087] 通过考虑包括多个子协调器（服务器）及其同步对象的一个事务的典型实例，可以说明上述处理。一个典型事务通过五个不同阶段进行：

[0088] 1. 活动阶段。

[0089] 2. 调用事务提交。

[0090] 3. 预准备阶段（该阶段包括同步回退的调用，即仅仅是完成前的同步，但不调用任何资源）。

[0091] 4. 准备阶段（该阶段仅包括调用资源，但不同步回退）。

[0092] 5. 提交阶段（该阶段也仅仅包含调用资源，但不同步回退）。

[0093] 在步骤 3 的预准备阶段期间，新同步对象可以向事务注册。还可以在预准备阶段增加新资源，使其参与事务。在步骤 4 和步骤 5 期间调用这些资源。通常增加资源，但不从事务中删除资源。新的子协调器（即服务器）可以在预准备阶段参与事务。

[0094] 例如，如果考虑存在一个客户和三个服务器 S1、S2、S3 的情况，则存在三个同步对象，即服务器 S1 上的 Sync1，服务器 2 上的 Sync2，服务器 3 上的 Sync3；以及三个资源对象，即 S1 上的 r1，S2 上的 r2 和 S3 上的 r3。如果客户启动或开始事务 T1，并且事务 T1 传播到服务器 S1，那么根据本发明一个实施例，服务器 S1 将成为该事务的提交服务器 / 事务协调器。其它实施例和实施可以使用从多个可用服务器选择事务协调器的替代机制。在事务 T1 的活动阶段，我们有以下状态信息：

[0095] ----------

[0096] 当前参与事务 T1 的子协调器（服务器）：S2

[0097] 向事务 T1 注册的同步对象：Sync1（来自服务器 S1），Sync2（来自服务器 S2）

[0098] 参与事务 T1 的资源：r1（来自服务器 S1），r2（来自服务器 S2）。

[0099] ----------

[0100] 当客户以后调用一个 T1.commit() 或等同功能，以提交事务时，以下事件序列发生：

[0101] 1. 提交移交给协调器，服务器 S1。

[0102] 2. 服务器 S1 执行预准备阶段。服务器 S1 调用本地注册的同步回退：Sync1。服务器 S1 将开始准备请求发送给链中的第一子协调器：S2。

[0103] 3. 服务器 S2 执行预准备阶段。服务器 S2 调用本地注册的同步回退的 Synchronization.beforeCompletion：Sync2。Sync2.beforeCompletion 功能调用新服务器 S3 上的 EJB。EJB 的商务方法调用新资源 r3。同时，服务器 S3 上的 EJB 容器还注册新的统统对象 Sync3（在服务器 S3 上）。

[0104] 作为结果，事务的状态信息目前改变如下：

[0105] ----------

[0106] 当前正在参与事务 T1 的子协调器（服务器）：S2，S3

[0107] 现在向 T1 注册的同步对象：Sync1（来自服务器 S1），Sync2（来自 S2），Sync3（来自服务器 S3）

[0108] 现在参与事务 T1 的资源：r1（来自服务器 S1），r2（来自服务器 S2），r3（来自服务器 S3）。

[0109] ----------

[0110] 服务器 S2 现在已经完成了它自己的同步回退。服务器 S2 沿着链把预准备请求传送到下一个子协调器，即服务器 S3。需要注意的是，当服务器 S2 首先进入它的准备阶段时，它是链中的最后的子协调器，但现在不是，服务器 S3 才是现在的链中的最后子协调器。

[0111] 4. 服务器 S3 现在执行预准备阶段。服务器 S3 调用本地注册的同步回退 :Sync3。服务器 S3 确定它是链中真正的最后子协调器，由于目前存在链，因此把一个预准备答复回送给事务协调器，即服务器 S1。

[0112] 5. 服务器（在该事件中还是事务协调器）S1 得知预准备阶段被执行。然后可以前进到准备以及随后的提交阶段。

[0113] 图 6 图示说明了显示本发明实施例的同步回退处理的步骤的流程图。在步骤 702，客户或者客户应用程序准备一个用于处理的事务。在活动阶段期间（步骤 704），注册将要参与事务处理的各种同步对象、资源或服务器（步骤 706）。当调用事务提交时（步骤 707），又调用每个同步对象或资源（步骤 708）。在任何时候，可以修改同步对象的目录，把诸如参与服务器的资源添加到回退环中（步骤 710）。当当前目录中的所有同步对象已经返回它们的回退时（步骤 712），提交事务（步骤 714）。

[0114] 在 beforeCompletion 期间对调用 bean 的 bean 的支持

[0115] EJB 和 JTA 规范都没有澄清如何处理这样一种情况 :即 bean 的 beforeCompletion 或 ejbStore 方法（准备前调用）调用另一个其 beforeCompletion 和 / 或 ejbStore 方法已经被调用的 bean。本发明支持多个服务器上的这样一种情况。

[0116] 图 7 至图 14 图示说明了根据可与 Weblogic 服务器一起使用的特定实施例，与同步回退处理的各个阶段相关联的寿命周期 ;本领域的熟练技术人员将会明白，在本发明精神和范围之内，其它寿命周期和实施例也可以用来提供同步回退。图 7 显示了本发明实施例的开始事务寿命周期的图示。如图 7 所示，处理开始于在事务管理器 804 上接收的事务请求。事务管理器 804 调用协调器定位器 805 去选择一个事务协调器 806。事务协调器 806 接收一个事务标识符，并将该标识符与新事务 830 相关联。该请求被分配给线程的一个池 (pool)，并由此返回到事务管理器 804。后续请求被分配给与该池不同的线程。

[0117] 图 8 显示了本发明一个实施例的事务传播寿命周期的图示。如图 8 所示，客户应用程序 802 对本地（客户）Java 虚拟机 (RJVM)807 进行远程方法调度，用于对服务器通信。客户事务管理器 804 拦截该请求并经由服务器 RJVM808 将该请求传送到服务器事务管理器 810。服务器事务管理器 810 负责调用服务器应用程序 812 上的方法。当从服务器应用程序 812 接收一个答复时，服务器事务管理器 810 又负责经由客户和服务器 RJVM 把答复传送到客户事务管理器 804。客户事务管理器 804 将答复返回到客户应用程序 802。

[0118] 图 9 显示了本发明实施例的提交寿命周期的图示。如图 9 所示，事务管理器 804 接收一个提交。该提交被传送到事务协调器 806 进行处理。当该提交完成时，将其返回到事务管理器 804。

[0119] 图 10 显示了本发明实施例的预准备阶段寿命周期的图示，显示了同步对象是如何被添加到 beforeCompletion 环中。如图 10 所示，事务协调器 806 采用对第一子协调器 814 的 startPreoare 调用启动预准备处理。子协调器负责事务通过 beforeCompletion 阶段。在 beforeCompletion 阶段完成之前的任何时候，同步对象 816 可以被调加到环中。

当添加了所有同步对象时，第二子协调器 818 同时对所有同步对象 820 起作用。然后把 PrePreare 确认返回到事务协调器 806。本领域熟练技术人员将会明白，在图 10 所示的寿命周期中以及下文所述的其它寿命周期中，第一和第二子协调器可以是同一实体。同样，第一和第二组同步对象可以是相同的，因为它们也许是 XA 资源对象的第一和第二组。本领域熟练技术人员还将会明白，尽管在这里给出的实例中仅显示了两个子协调器或者两组资源对象，但是本发明允许多个子协调器和多组同步对象和资源被同时包含在事务提交处理中。该方法主张的子协调器和同步对象使协调器通信交换的数量最小。

[0120] 图 11 显示了本发明实施例的使用 XAR 资源对象的准备阶段寿命周期的图示。如图 11 所示，事务协调器 806 采用对第一协调器 814 的 startPrepare 调用。子协调器（或多个子协调器）负责准备应付第一组 XAR 资源对象 817 的事务。与此同时，事务协调器可以发送对第二子协调器 818 的 startPrepare 调用，以准备应付第二组 XAR 资源对象 821 的事务。当准备了每个 XAR 资源对象时，将一个确认返回到事务协调器 896。

[0121] 图 12 显示了提交阶段寿命周期的图示。如图 12 所示，事务协调器 806 首先刷新事务日志。把 startCommit 调用发送到第一子协调器 814。第一子协调器发出一个对第一组 XAR 资源对象 817 的提交。与此同时，该事务协调器可以把 startCommit 发送给子协调器 818。第二子协调器发出一个对 XAR 资源对象 821 的提交。第二子协调器发出一个对第二组 XAR 资源对象 821 的提交。当这两个提交完成时，向协调器 806 发送一个确认。

[0122] 图 13 显示了本发明实施例的返回寿命周期的图示。如图 13 所示，当系统希望执行返回时，事务协调器 806 把 statrRollback 发送到第一协调器 814。协调器负责执行离开第一组 XAR 资源对象 817 的返回。与此同时，系统经由第二子协调器向第二组 XAR 资源对象 821 发出一个 startRollback 命令，由此可以在第二组 XAR 资源对象 821 上执行返回。当两个返回都完成时，把一个确认返回到事务协调器 806。

[0123] 图 14 显示了本发明实施例的恢复寿命周期的图示。如图 14 所示，恢复处理通过来自事务协调器 806 的请求启动，以访问事务恢复日志。当事务协调器 806 向子协调器 814 发出一个 startRecovery 请求以执行恢复时，子协调器使用事务恢复日志中的信息执行 XAR 资源对象组的恢复。子协调器 814 使用事务标识符正确地标识每个恢复，并且把该事务标识符返回到事务协调器 806。事务协调器 806 然后向子协调器 814 发送一个返回请求。

[0124] EJB 应用程序中的事务

[0125] 图 15 显示了如何处理 EJB 客户应用程序与服务器应用程序之间的事务。如图 15 所示，EJB 客户应用程序 402 使用 EJB 容器 406 把事务 404 传送到服务器应用程序 410。加入本发明的应用程序服务器支持 EJB 应用程序中的两类事务：容器管理的事务和 bean 管理的事务。

[0126] 在容器管理的事务中，服务器 EJB 容器负责管理事务定界。EJB 部署描述符中的事务属性被用来确定服务器 EJB 容器如何用每个方法调用来处理事务。在 bean 管理的事务中，EJB 本身管理事务界定。EJB 对 UserTransaction 对象进行显性方法调度以开始提交，如果需要，还返回事务。事务事件的序列在容器管理与 bean 管理的事务之间是不同的。对于带有容器管理事务的 EJB 应用程序，一个基本事务按以下方式工作：

[0127] • 在 EJB 部署描述符中，bean 供应者或者应用程序装配器指定容器管理定界的事务类型。

[0128] • 在 EJB 部署描述符中，bean 供应者或者应用程序装配器为 EJB 指定默认事务属性，它通常是以下设置之一 :“NotSupported”,“Required”。“Supports”,“RequiresNew”,“Mandatory”或者“Never”。在 EJB 部署描述符中，bean 供应者或者应用程序装配器为一个或多个方法随意地指定事务属性。

[0129] •当客户应用程序 402 在 EJB 中调用方法时，EJB 容器为该方法检验部署描述符中的事务属性设置。如果没有为该方法指定设置，则 EJB 使用该 EJB 的默认事务属性设置。

[0130] •EJB 容器 406 根据可应用的事务属性设置采取适当行动。例如，如果事物属性设置是“Required”，则 EJB 容器调用现存事务上下文内的方法，或者，如果客户未用事务上下文调用，则 EJB 容器在执行该方法之前开始一个新事务。

[0131] • 在另一个实例中，如果事务属性设置是“Mandatory”，则 EJB 容器调用现存事务上下文中的方法。如果客户未用事务上下文调用，则 EJB 容器丢弃异常错误。

[0132] •在商务方法 408 的调用期间，如果确定需要返回，则商务方法调用 EJB 返回方法，通知 EJB 容器该事务在方法将在调用结束时返回。

[0133] • 在方法执行结束和巴结果发送给客户之前，EJB 容器完成事务，这是通过在服务器应用程序 410 上提交事务（在此情况下，可以把事务写入事务日志 412）或者返回事务实现的。

[0134] 对于具有 bean 管理事务定界的 EJB 应用程序，基本事务以下述方式工作 :

[0135] •在 EJB 部署描述符中，bean 供应者或者应用程序装配器指定 bean 管理定界的事务类型。

[0136] • 客户应用程序 402 使用 JNDI 获取对服务器域的 UserTransaction 对象的引用。

[0137] •客户应用程序 402 开始一个使用 UserTransaction.begin 方法的事务，并且经过 EJB 容器发送一个请求给 EJB。EJB 上的所有操作在事务范围内运行。

[0138] •如果对这些操作任何一个的调用出现异常（明示或者作为通信失败的结果），则可以捕获该异常并且可以返回该事务 (UserTransaction.rollback)。

[0139] • 如果无异常出现，则客户向使用 UserTransaction.commit 方法的服务器应用程序 410 提交当前事务。该方法结束事务并开始操作的处理。只有当事务中的所有参与者都同意提交时才提交该事务。

[0140] • UserTranaction.commit 方法使 EJB 容器调用事务管理器完成该事务。

[0141] • 事务管理器负责与资源管理器协调，以更新任何一个数据库。

[0142] 事务样本 EJB 码

[0143] 以下部分提供了依据 EJB 应用程序的类别的样品码片段的走查。码片段展示了为 bean 管理事务定界利用 UserTransaction 对象的情况。该 bean 的部署描述符指定了事务定界的事务类型。

[0144] 事务软件包首先必须被输入和 / 或更新。列表 1 显示了输入事务必须的软件包的样本码。

[0145] 列表 1

```
[0146] import javax.naming.*;
[0147] import javax.transaction.UserTransaction;
[0148] import javax.transaction.SystemException
[0149] import javax.transaction.HeuristicMixedException
```

[0150] import javax.transaction.HeuristicRollbackException
[0151] import javax.transaction.NotSupportedException
[0152] import javax.transaction.RollbackException
[0153] import javax.transaction.IllegalStateException
[0154] import javax.transaction.SecurityException
[0155] import java.sql.*;
[0156] import java.util.*;
[0157] import weblogic.jndi.*;

[0158] 在输入这些类之后，将 UserTransaction 对象的事件初始化到零。下一步骤包括使用 JNDI 返回对象参考。列表 2 显示了使用 JNDI 查找对象参考的一个实例。

[0159] 列表 2

[0160] context ctx=null
[0161] Hashtable env=new Hashtable()
[0162] env.put(context.INITIAL_CONTEXT_FACTORY, “weblogic.jndi.WLInitialCont extFactory”);
[0163] env.put(Context.PROVIDER_URL, “t3://localhost:7001”);
[0164] env.put(Context.SECURITY_PRINCIPAL, “Fred”);
[0165] env.put(Context.SECURITY_CREDENTIALS, “secret”);
[0166] ctx=nex InitialContext(env);
[0167] UserTransaction tx=(UserTransaction)
[0168] ctx.lookup(“javax.transaction.UserTransaction”);

[0169] 随后必须生成或者开始事务。列表 3 显示了通过得到 UserTransaction 对象并调用事务开始方法来开始一个事务。该方法调用后以及完成事务之前出现的数据库操作存在于这些事务的范围之内。

[0170] 列表 3

[0171] UserTransaction tx=(UserTransaction)
[0172] ctx.lookup(“javax.transaction.UserTransaction”);
[0173] tx.begin();

[0174] EJB 实施中的最后步骤是完成事务。列表 4 显示了根据在任何一个数据库操作期间是否丢弃一个异常来完成事务的代码，该数据库操作是试图在事务范围之内进行的：

[0175] 列表 4

[0176] tx.commit()
[0177] //or:
[0178] tx.rollback();

[0179] 如果在任何数据库操作期间抛出异常，则应用程序调用 j ava.transaction.UserTransaction.rollback() 方法来回退事务。

[0180] 如果没有抛出异常，则应用程序调用 javax.Transaction.UserTransaction.commit() 方法，以试图在所有数据库操作成功完成之后提交事务。调用该方法结束事务并开始操作的处理，使 Weblogic 服务器 EJB 容器调用事务管理器王城事务。只有当事务的参与者都同意提交时才提交事务。

[0181] Weblogic 服务器 RMI 应用程序中的事务

[0182] 图 16 说明了事务如何在 RMI 应用程序中工作。对于 RMI 客户和服务器应用程序，基本事务按以下方式工作：

[0183] • 应用程序 502 使用 JNDI 将对象参考返回到服务器域的 UserTransaction 对象 506。

[0184] • 获得对象参考导致产生应用程序与该对象之间的一个对话状态。在完成（即提交或者返回）事务 504 之前，该对话状态一直持续。RMI 对象一旦被例示，就存储器中保持活动状态，直至它们被释放，通常是在服务器停机期间进行释放。

[0185] • 客户应用程序开始一个使用 UserTransactin.begin 方法的事务，并把请求发送给服务器应用程序。

[0186] • 服务器应用程序上的所有操作在事务范围之内执行。

[0187] •如果对这些操作的任何一个的调用出现异常（明示或者作为通信失败的结果），则可以捕获该异常并可以返回该事务 (UserTransaction.rollback)。

[0188] • 如果没有异常出现，则客户应用程序提交使用 UserTransaction.commit 方法的当前事务。该方法结束事务并开始操作的处理。只有当事务的所有参与者都同意提交时，才提交事务。

[0189] • UserTransaction.commit 方法使服务器调用事务管理器完成该事务。

[0190] • 事务管理器负责与资源管理器协调更新任何数据库 512。

[0191] 事务样本 RMI 码

[0192] 该部分提供了来自 RMI 应用程序中的类的样本码片段的走查。码片段演示了使用 RMI 事务的 UserTransaction 对象的情况。

[0193] 此外，作为初始步骤，需要时必须输入和 / 或更新程序包。列表 5 显示了输入必要的程序包。

[0194] 列表 5

```
[0195] import javax.naming.*;
[0196] import javax.rmi.*;
[0197] import javax.transaction.UserTransaction;
[0198] import javax.transaction.SystemException;
[0199] import javax.transaction.HeuristicMixedException;
[0200] import javax.transaction.HeuristicRollbackException;
[0201] import javax.transaction.NotSupportedException;
[0202] import javax.transaction.RollbackException;
[0203] import javax.transaction.IllegalStateException;
[0204] import javax.transaction.SecurityException;
[0205] import java.sql.*;
[0206] import java.util.*;
[0207] import weblogic.jndi.*;
```

[0208] 输入这些类后，把 UserTransaction 对象的事件初始化为零。

[0209] JNDI 用来吧对象参考返回到 UserTransaction 对象。列表 6 显示了搜索 JNDI 树，把对象参考返回到用于适当 Weblogic Server 域的 UserTransaction 对象的情况。

[0210] 获得对象参考导致产生应用程序与该对象之间的对话状态。在完成事务（提交或返回）之前，该对话状态一直持续。一旦 RMI 对象被例示，就在存储器中保持其活动状态，直至它们被释放。

[0211] 列表 6

```
[0212] context ctx=null;
[0213] Hashtable env=new Hashtable();
[0214] env.put(context.INITIAL_CONTEXT_FACTORY, “weblogic.jndi.WLInitialContFactory”);
[0215] //parameters for the WebLogic Server.
```

[0216] //Substitute the correct hostname, port number
[0217] //user name, and password for your enviroment:
[0218] env.put(context.PROVIDER_URL, “t3://localhost:7001”);
[0219] env.put(Context.SECURITY_PRINCIPAL, “Fred”);
[0220] env.put(context.SECURITY_CREDENTIALS, “secret”);
[0221] ctx=new TnitialContext(env);
[0222] UserTransaction tx=(UserTransaction)
[0223] ctx.lookup(“javax.transaction.UserTranaction.”);

[0224] 然后生成或开始事务。列表7显示通过调用javax.transaction.UserTransaction.begin()方法开始事务。该方法调用后以及完成事务前发生的数据库操作存在于该事务范围之内。

[0225] 列表7

[0226] User Transaction tx=(UserTransaction)
[0227] ctx.lookup(“javax.transaction.UserTransaction”);
[0228] tx.begin();
[0229] completing a Transaction

[0230] 如同EJB实施一样,事务处理中的最后步骤是完成事务。列表8显示了根据是否丢弃异常完成事务,其中在该事务范围之内试图进行任何一个数据库操作期间进行所述丢弃:

[0231] 列表8

[0232] tx.commit()
[0233] //or:
[0234] tx.rollback();

[0235] 如果在任何一个数据库操作期间抛出异常,则应用程序调用javax.transaction.UserTransaction.rollback()方法。

[0236] 若未抛出异常,则应用程序调用javax.transaction.UserTransaction.commit()方法,以试图在成功完成所有数据库操作之后提交事务。调用该方法导致结束事务并开始操作的处理,使WebLogic服务器调用事务管理器完成事务。只有当事务的所有参与者都同意提交时,才提交事务。

[0237] 本发明的以上说明用于图示说明的目的,而不是打算用来穷举或者把本发明限制到公开的精确形式。显然,本领域熟练技术人员将会明白许多修改和变化。选择和描述诸多实施例是为了最好地解释本发明原理和实际应用,从而使本发明的熟练技术人员理解适合于特定预期应用的不同实施例以及具有各种修改的本发明。本发明的范围由以下权利要求和其等同物定义。

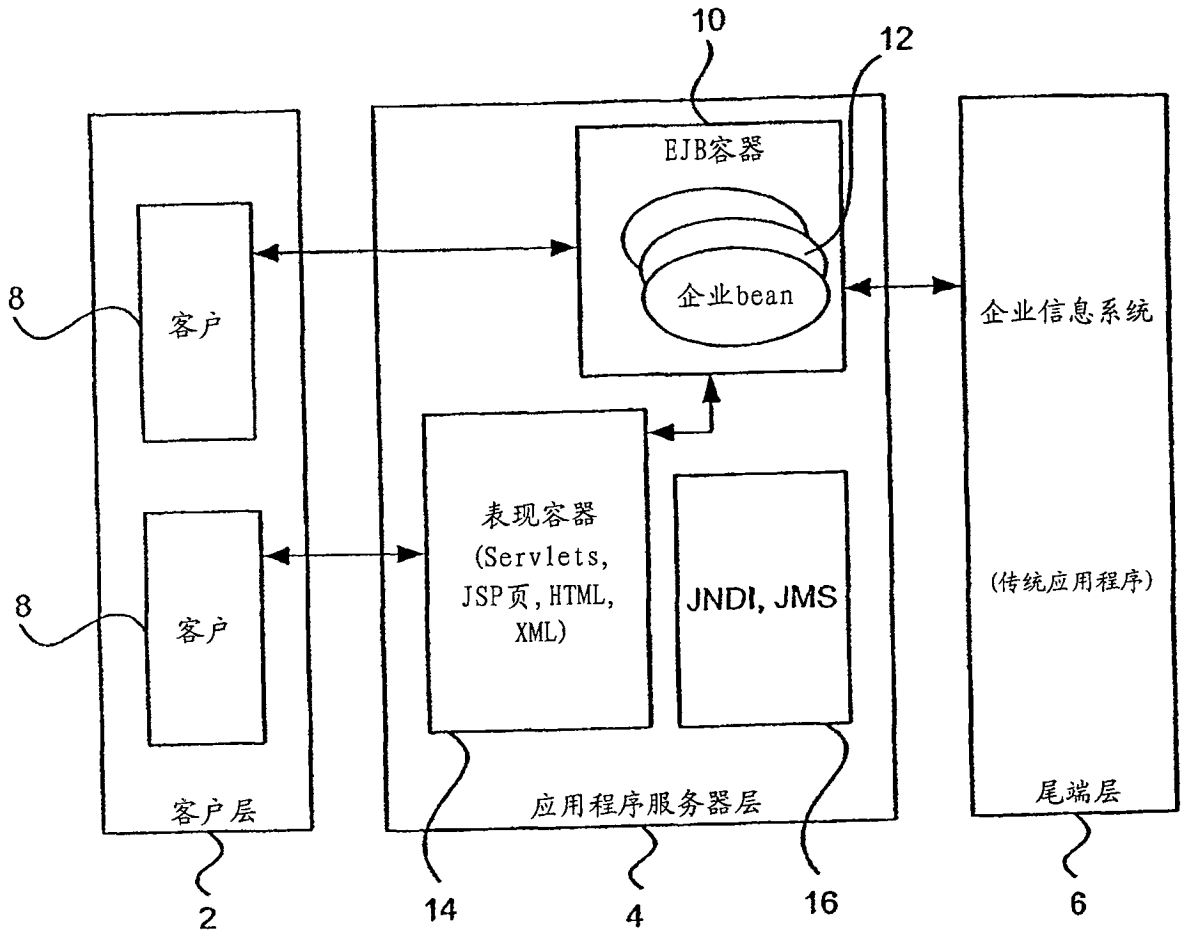

10
12
EJB容器
企业bean
企业信息系统
8
客户
8
客户
表现容器
(Servlets,
JSP页,HTML,
XML)
JNDI, JMS
(传统应用程序)
客户层
应用程序服务器层
尾端层
2
14
4
16
6

图 1

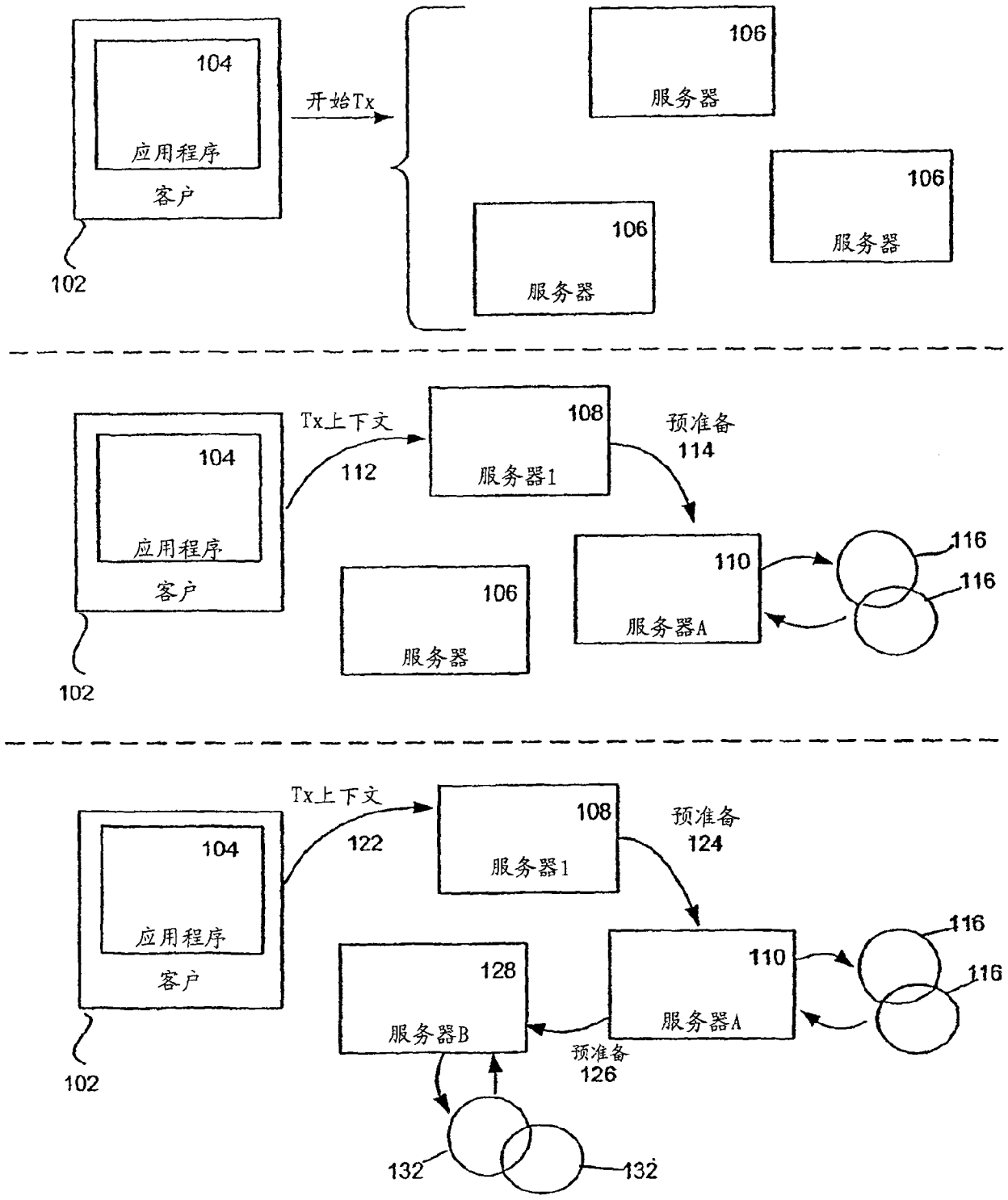

104
应用程序
客户
102
开始Tx
106
服务器
106
服务器
106
服务器
Tx上下文
112
108
服务器1
预准备
114
110
服务器A
116
116
106
服务器
Tx上下文
122
108
服务器1
预准备
124
128
服务器B
预准备
126
132
132

图 2

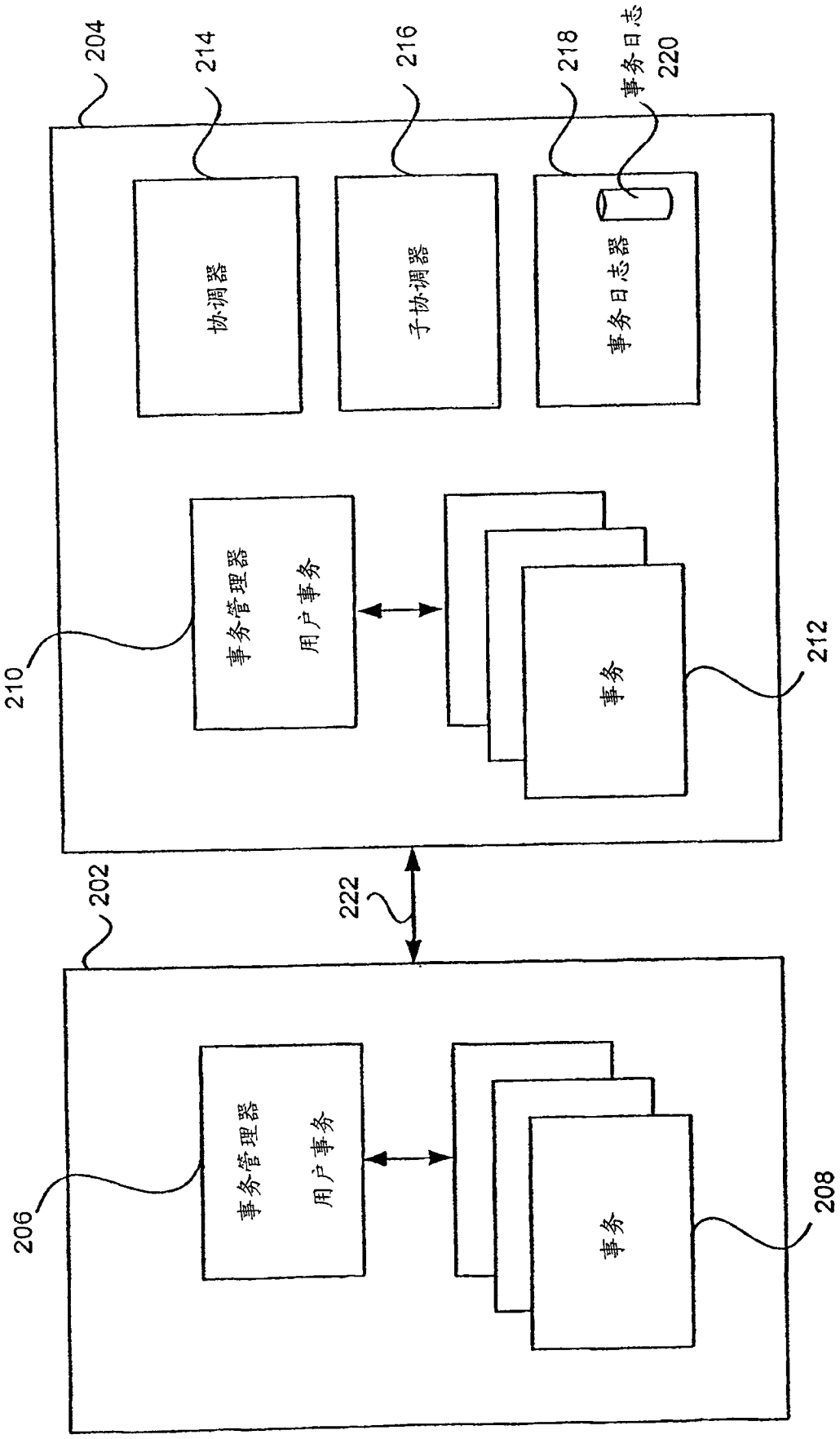

204
214
216
218
事务日志
220
协调器
子协调器
事务日志器
事务管理器
用户事务
210
事务
212
202
222
事务管理器
用户事务
206
事务
208

图 3

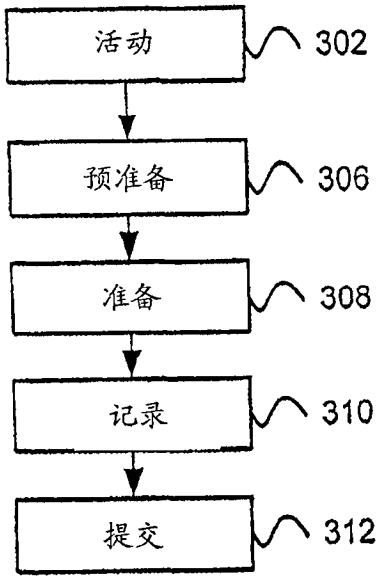

活动
302
预准备
306
准备
308
记录
310
提交
312

图 4

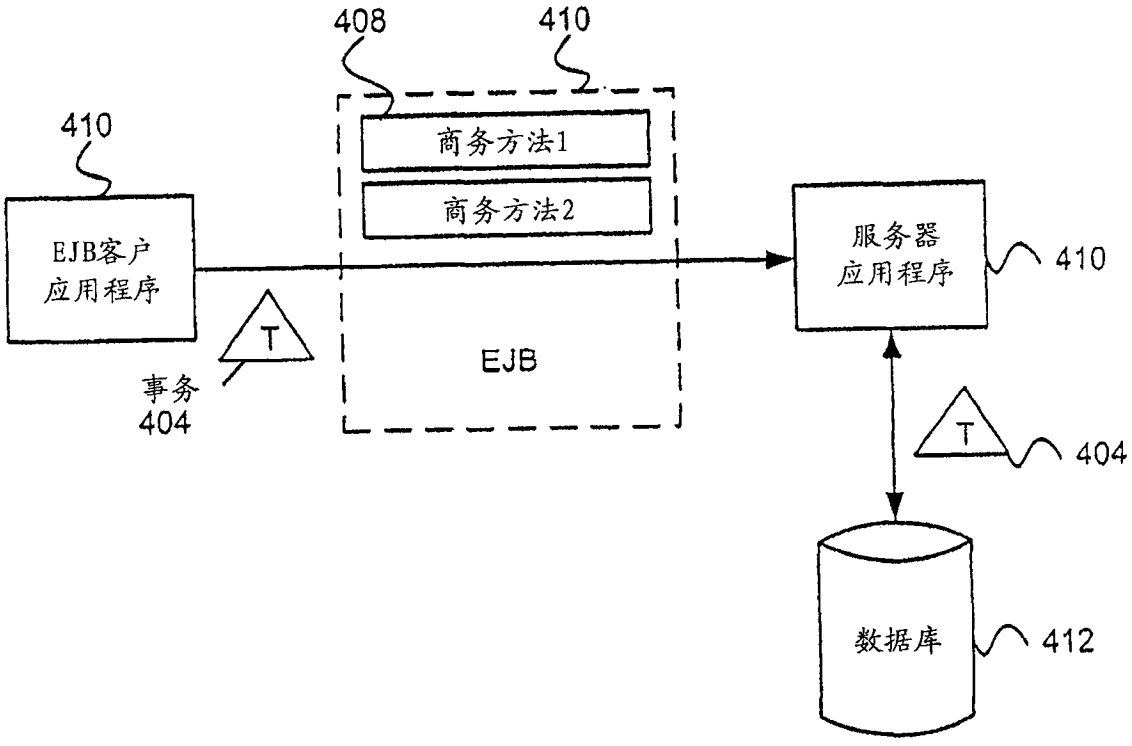

408
410
410
商务方法1
商务方法2
EJB客户
应用程序
T
事务
404
EJB
服务器
应用程序
410
T
404
数据库
412

图 15

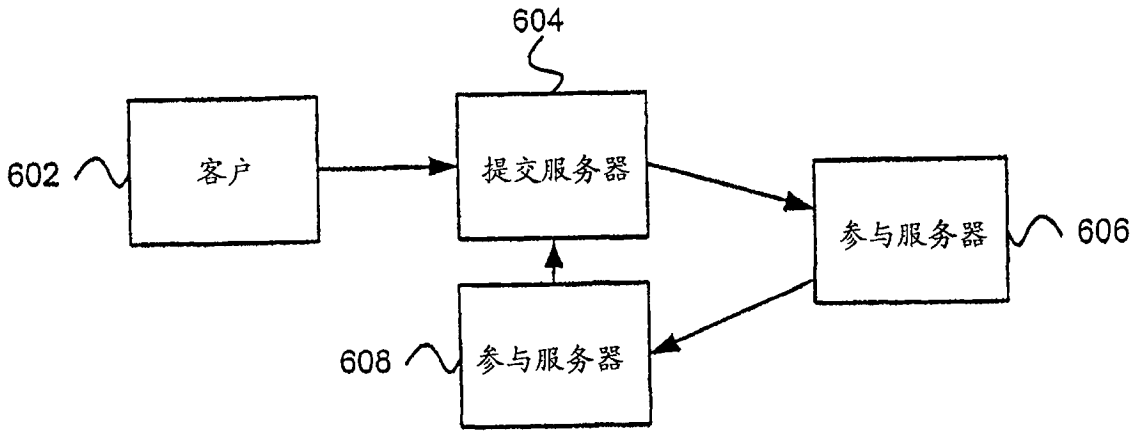

604
602
客户
提交服务器
参与服务器
606
608
参与服务器

图 5

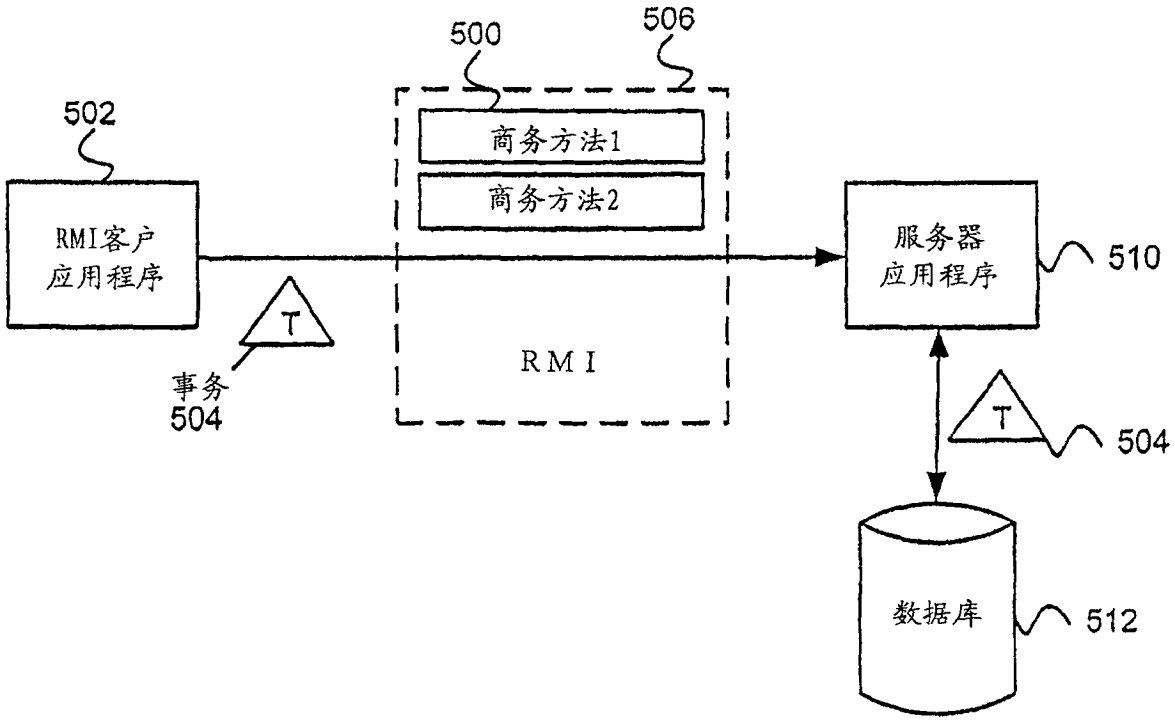

500
506
502
商务方法1
商务方法2
RMI客户
应用程序
服务器
应用程序
510
T
事务
504
RMI
T
504
数据库
512

图 16

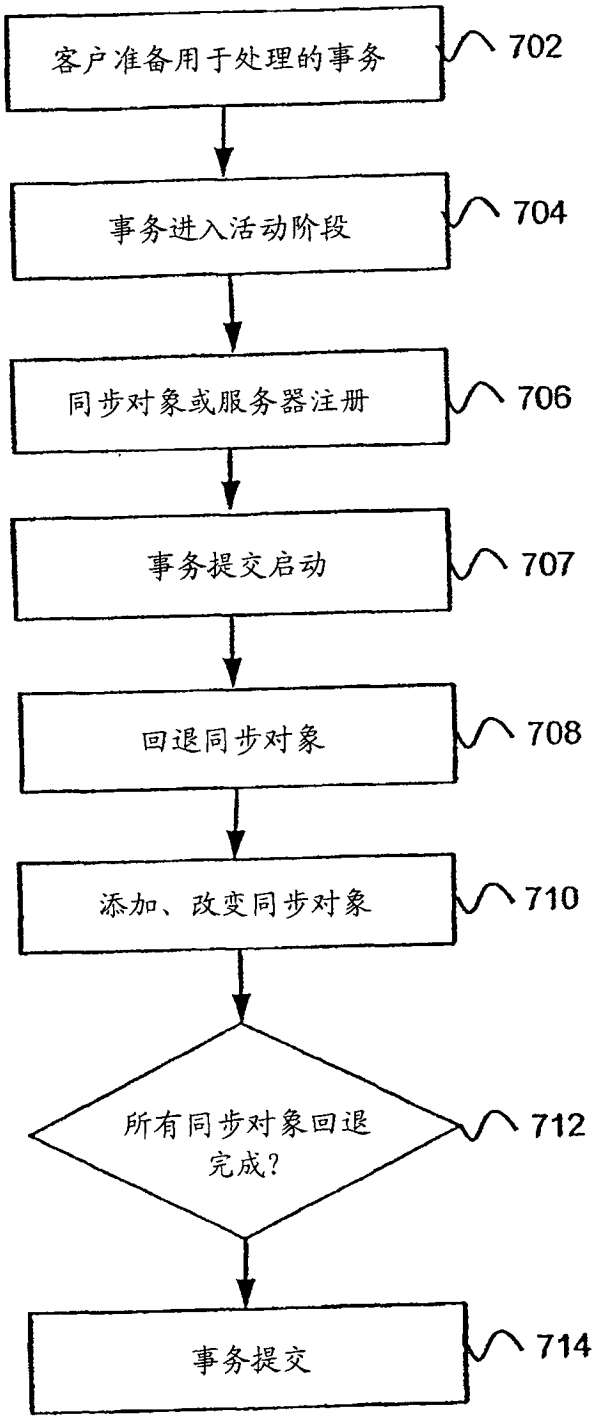

客户准备用于处理的事务
702
事务进入活动阶段
704
同步对象或服务器注册
706
事务提交启动
707
回退同步对象
708
添加、改变同步对象
710
所有同步对象回退完成?
712
事务提交
714

图 6

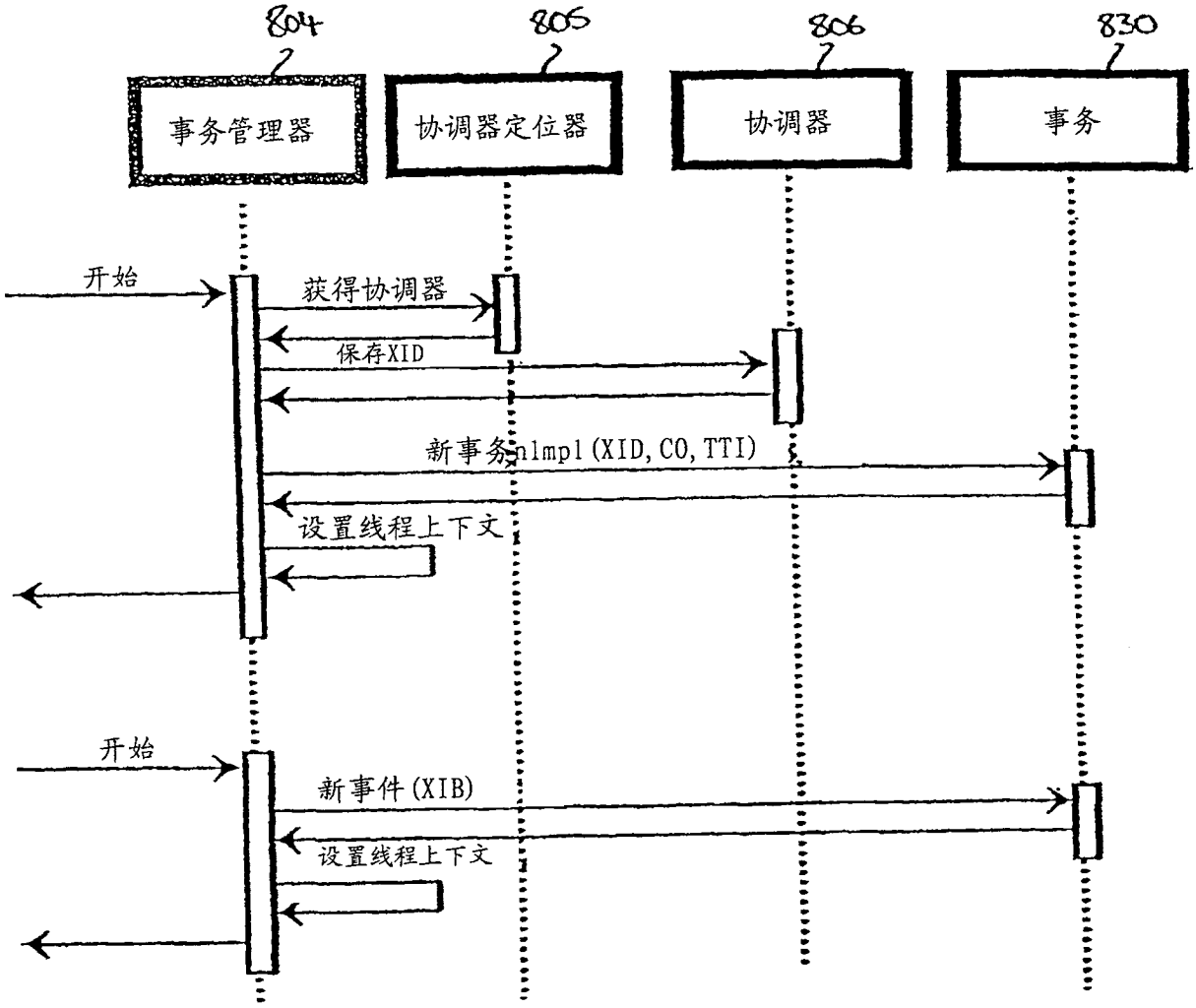

804
805
806
830
事务管理器
协调器定位器
协调器
事务
开始
获得协调器
保存XID
新事务Impl(XID,CO,TTI)
设置线程上下文
开始
新事件(XIB)
设置线程上下文

图 7

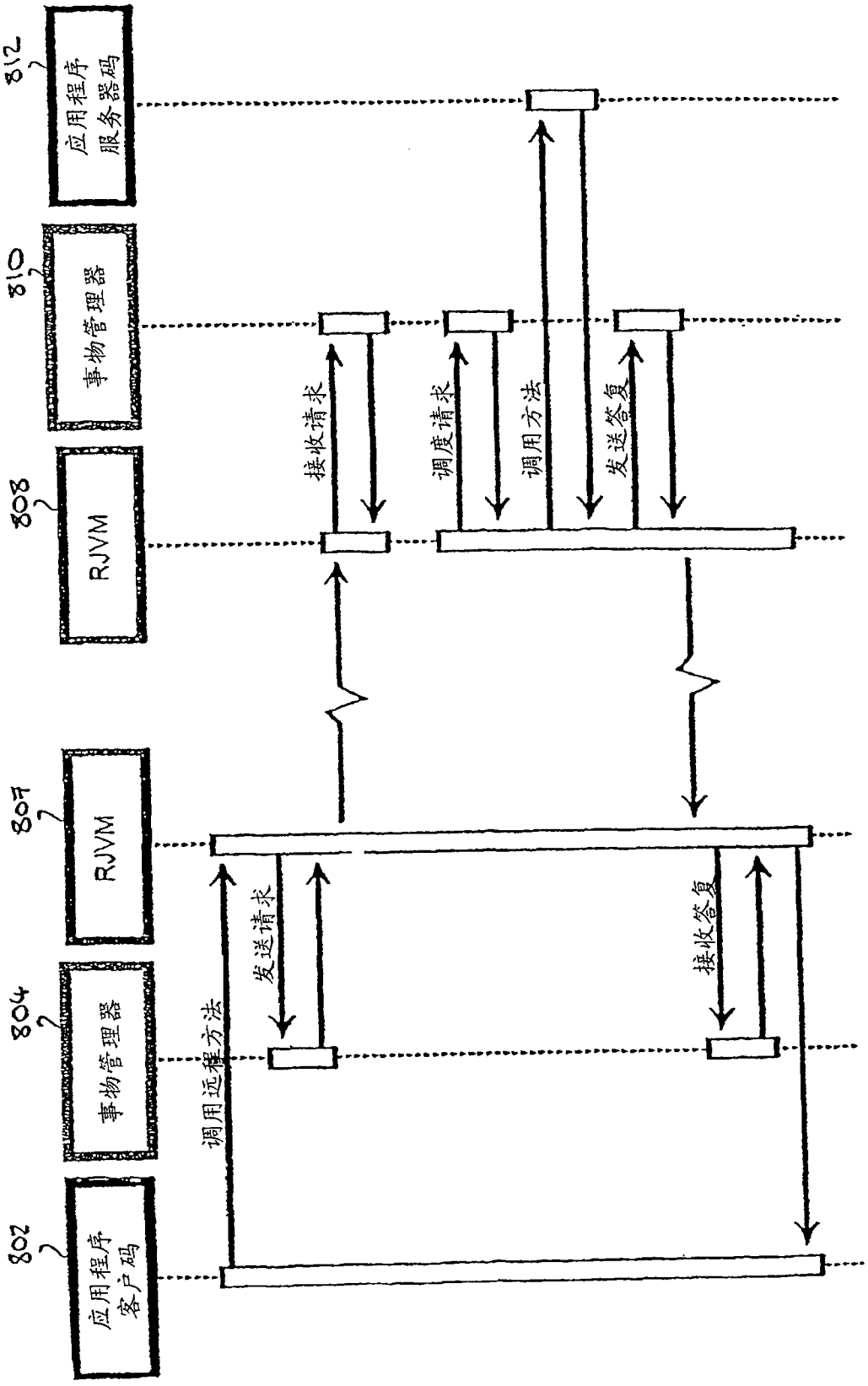

812
应用程序服务器码
810
事物管理器
808
RJVM
807
RJVM
804
事物管理器
802
应用程序客户码
接收请求
调度请求
调用方法
发送答复
发送请求
调用远程方法
接收答复

图 8

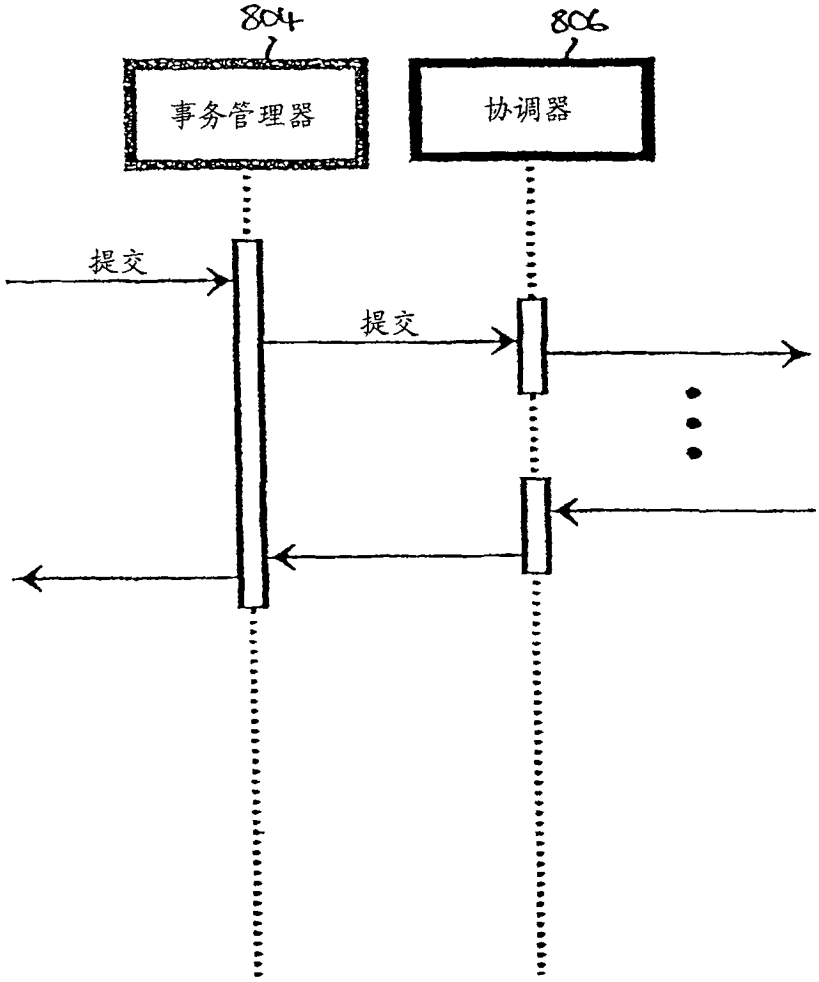


图 9

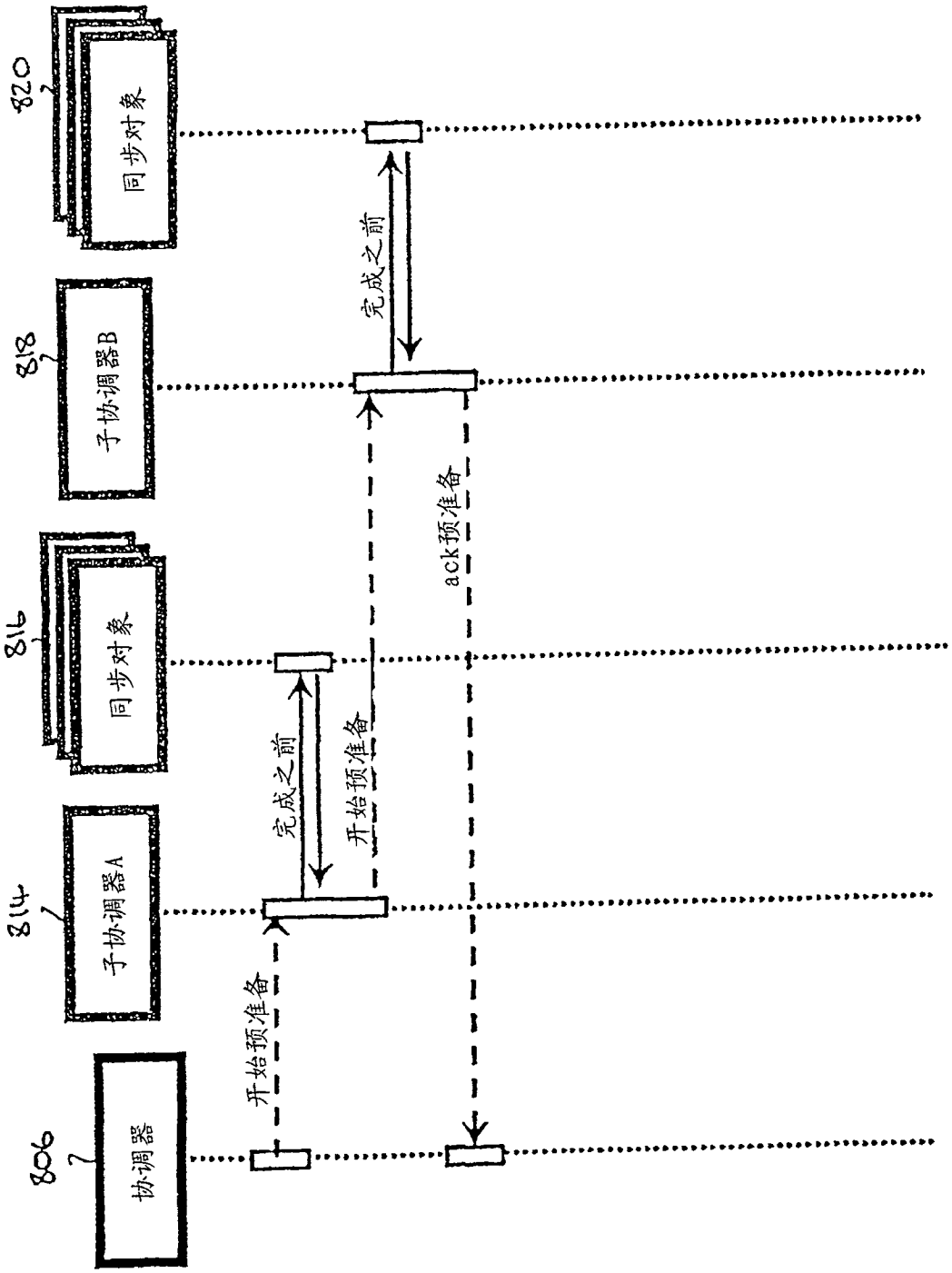

820
同步对象
818
子协调器B
816
同步对象
814
子协调器A
806
协调器
完成之前
ack预准备
完成之前
开始预准备
开始预准备

图 10

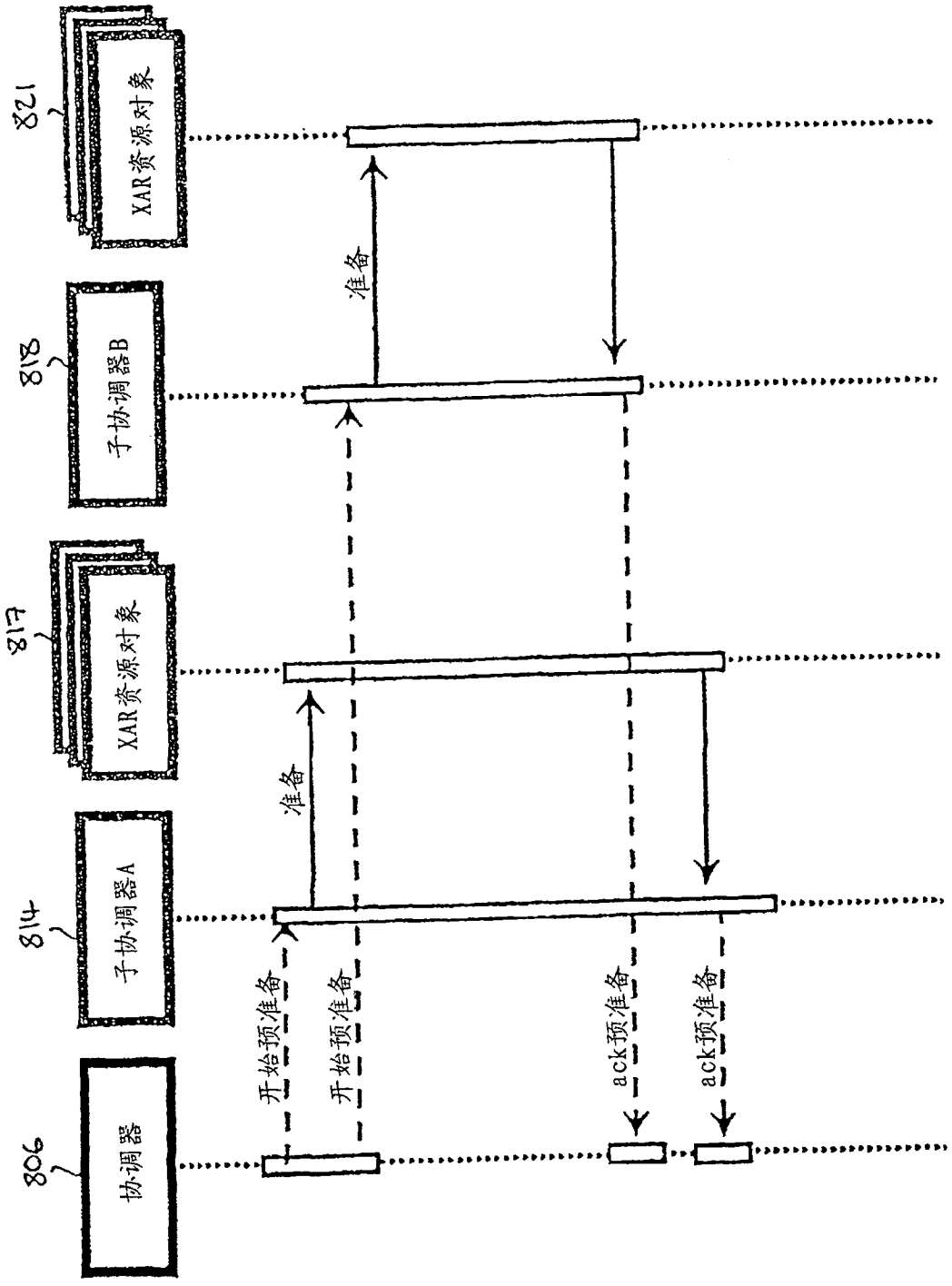

821
XAR资源对象
818
子协调器B
817
XAR资源对象
814
子协调器A
806
协调器
准备
准备
开始预准备
开始预准备
ack预准备
ack预准备

图 11

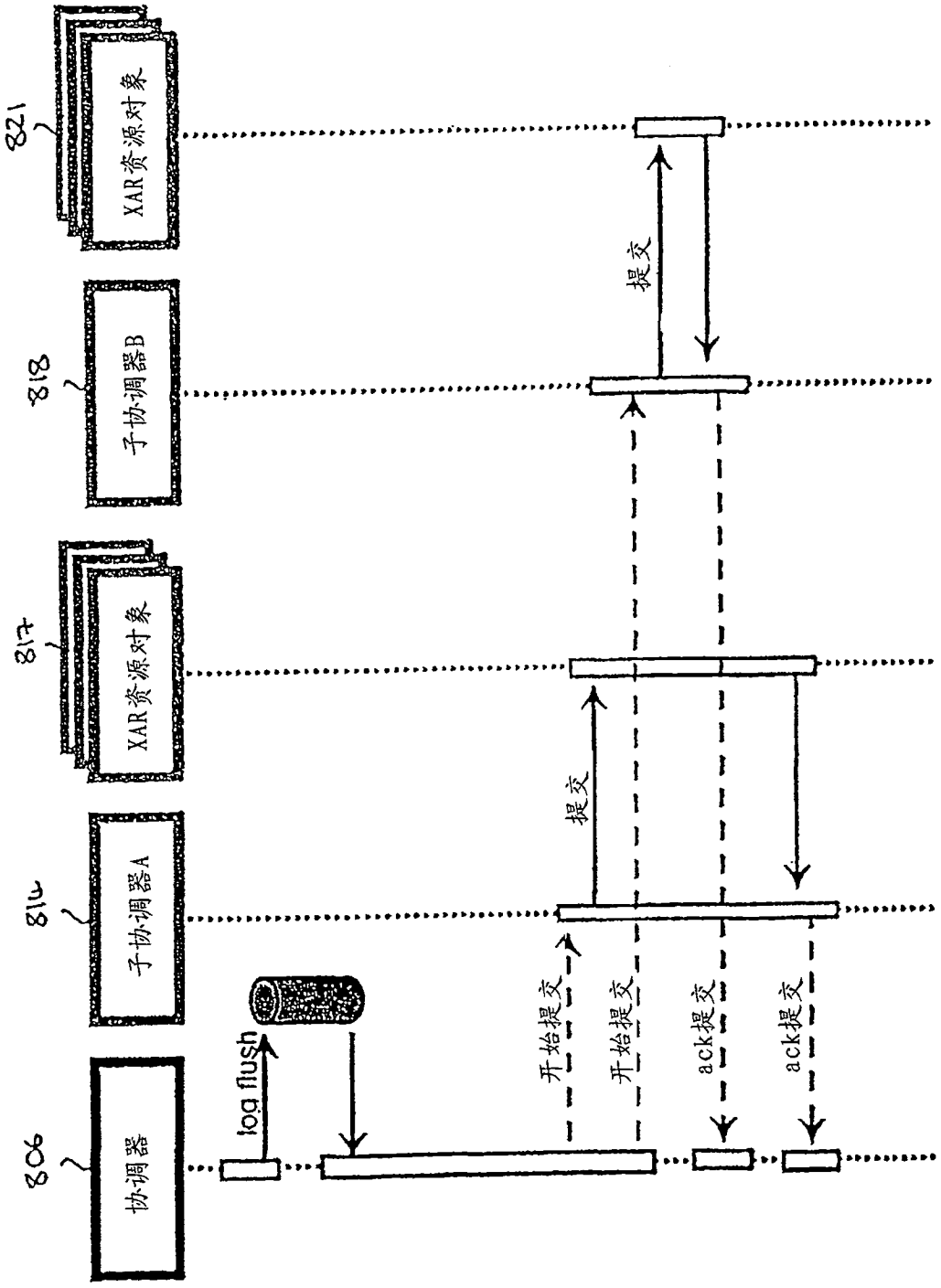

821
XAR资源对象
818
子协调器B
817
XAR资源对象
816
子协调器A
806
协调器
log flush
开始提交
开始提交
ack提交
ack提交
提交
提交

图 12

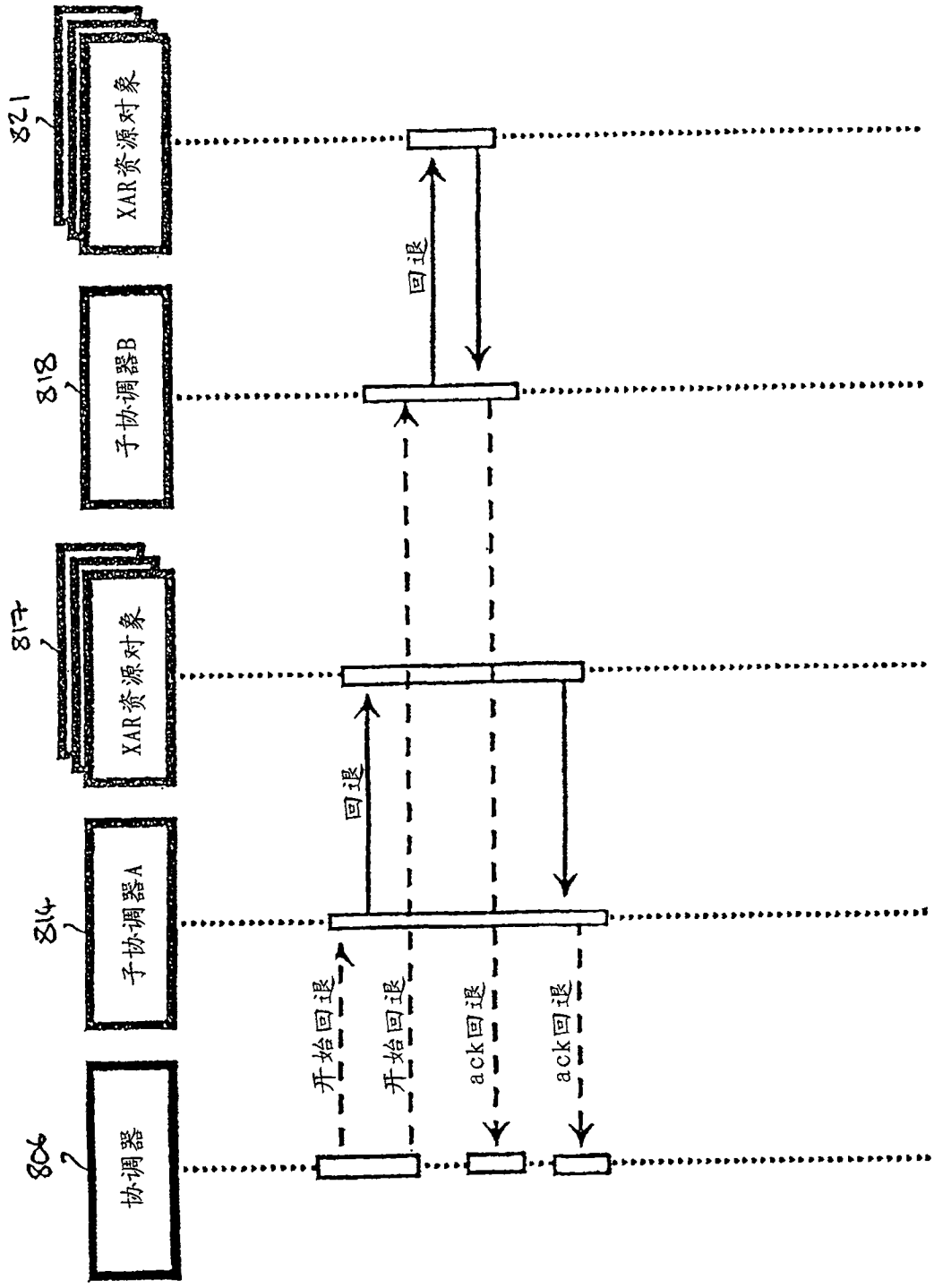

821
XAR资源对象
818
子协调器B
817
XAR资源对象
814
子协调器A
806
协调器
回退
回退
开始回退
开始回退
ack回退
ack回退

图 13

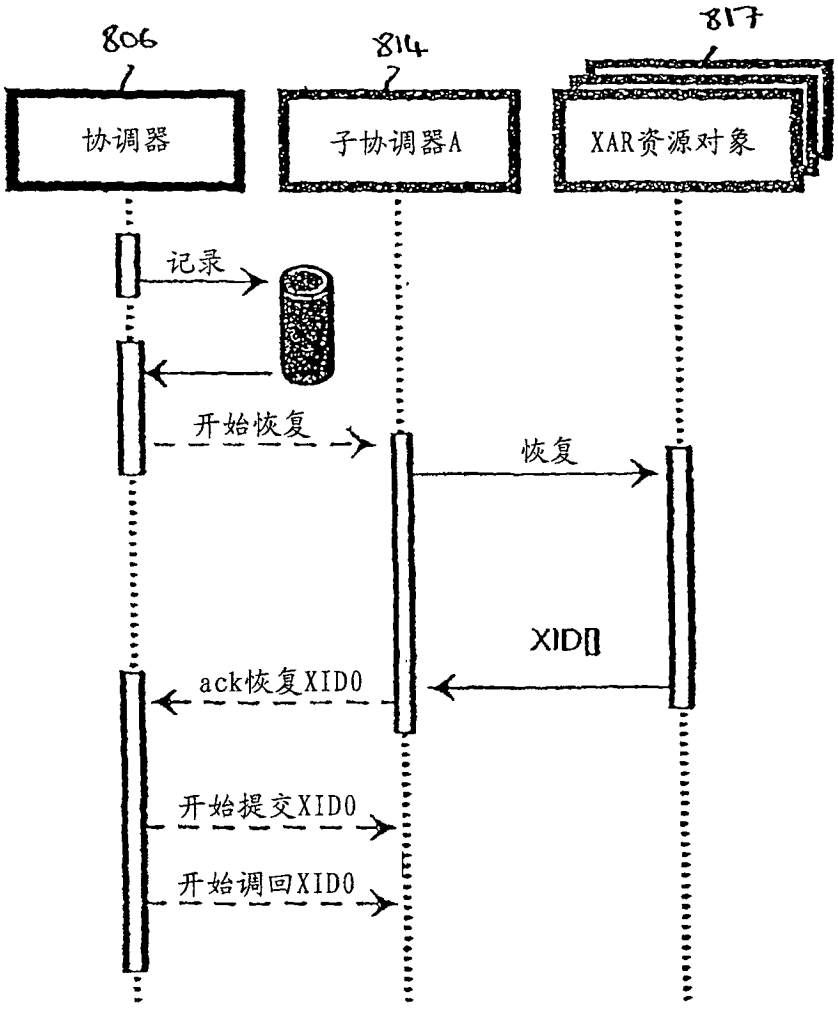

806
814
817
协调器
子协调器A
XAR资源对象
记录
开始恢复
恢复
XID[]
ack恢复XID0
开始提交XID0
开始调回XID0

图 14