[19] 中华人民共和国国家知识产权局

[51] Int. Cl.
G06F 7/00 (2006.01)
G06F 9/00 (2006.01)
G06F 17/30 (2006.01)

[12] 发明专利说明书

专利号 ZL 02817722.3

[45] 授权公告日 2009 年 1 月 7 日

[11] 授权公告号 CN 100449475C

[22] 申请日 2002.7.16 [21] 申请号 02817722.3

[30] 优先权

[32] 2001. 7.17 [33] US [31] 60/306,102

[32] 2002. 7.15 [33] US [31] 10/196,687

[86] 国际申请 PCT/US2002/022633 2002.7.16

[87] 国际公布 WO2003/009127 英 2003.1.30

[85] 进入国家阶段日期 2004.3.10

[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

[56] 参考文献

CN1225471A 1999.8.11

JP2001 - 256274A 2001.9.21

CN1264874A 2000.8.30

US5857100A 1999.1.5

US5586312A 1996.12.17

US5958004A 1999.9.28

WO00/55774A2 2000.9.21

MIS 事务处理技术的研究. 张功萱，齐广玉. 小型微型计算机系统，第 19 卷第 5 期. 1998

审查员 王 伟

[74] 专利代理机构 北京市柳沈律师事务所

代理人 郭定辉 黄小临

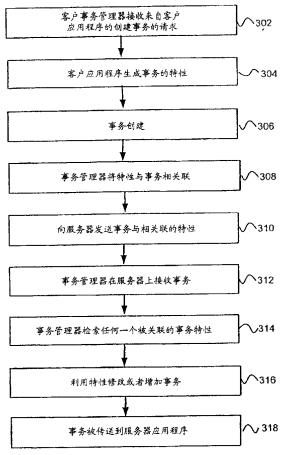
权利要求书 3 页 说明书 21 页 附图 7 页

[54] 发明名称

用于具有事务特性特征的事务处理的系统和方法

[57] 摘要

本发明提供了一种供事务处理系统使用的机制，该系统允许一个事务包括多个被关联的事务特性(308)。事务特性又允许命名对象与事务相关联。只要事务活动，事务特性就是可用的。当事务流到分布系统中的各服务器(310)或者在单个系统的不同软件模块之间流动时，可以按一种附件或者带外通信机制形式使用事务特性。



1、一种在事务从第一应用程序处理向第二应用程序处理传送期间将事务特性与事务相关联的系统，包括:

第一事务接口，包括允许所述第一应用程序处理指定将要与特定事务相关联的事务特性的功能件;

第一事务管理器，将所述事务特性与所述特定事务相关联，其中所述事务特性允许指定对象与事务相关联;

通信部件，把所述特定事务与所述相关联事务特性一起从所述第一事务管理器传送到第二事务管理器;

第二事务管理器，检索与所述特定事务相关联的所述事务特性；和

第二事务接口，包括允许所述第二应用程序处理检索与所述特定事务相关联的所述事务特性并以附加事务特性修改或添加该事务的功能件。

2、根据权利要求 1 所述的系统，其中所述第一事务接口包括允许所述第一应用程序处理指定将要与特定事务相关联的事务特性的多个功能件，多个事务特性可以关联所述特定事务。

3、根据权利要求 1 所述的系统，其中所述第一事务接口包括供第一应用程序处理使用的 setProperty 功能件，将附加信息与事务上下文一起传播。

4、根据权利要求 1 所述的系统，其中所述事务特性是关键字值对，所述关键字值对的关键字元素是字串，以及所述关键字值对的所述值元素是串行化实体。

5、根据权利要求 1 所述的系统，其中每个所述事务接口都扩展 Java 事务接口。

6、根据权利要求 1 所述的系统，其中所述第一事务接口中的所述功能件是把事务名称与事务相关联的 setName 功能件。

7、根据权利要求 6 所述的系统，其中所述 setName 功能件用事务类型命名事务。

8、根据权利要求 1 所述的系统，其中所述第二事务接口包括 addProperties 功能件，用于将一组特性添加到所述特定事务。

9、根据权利要求 8 所述的系统，其中所述第二事务接口还包括 setProperty 功能件，用于设置与所述特定事务预先关联的特性的值。

10、根据权利要求 1 所述的系统，其中当所述第一应用程序处理启动事务时，所述第一事务管理器创建唯一的事务标识符。

11、根据权利要求 1 所述的系统，其中一旦检索到所述事务特性，第二事务管理器就把事务答复传送给第一事务管理器。

12、一种在事务从第一应用程序处理向第二应用程序处理传送期间将事务特性与事务相关联的方法，包括以下步骤:

在第一应用程序处理上，通过对第一事务接口的调用，指定将要与特定事务相关联的事务特性;

在第一事务管理器上，将所述事务特性与所述特定事务相关联，其中所述事务特性允许将指定对象与事务相关联;

把所述特定事务与所述相关联事务特性传送到第二应用程序处理;

在第二事务管理器上，检索与所述特定事务相关联的所述事务特性；和

在第二应用程序处理上，通过对第二事务接口的调用检索与所述特定事务相关联的所述事务特性并以附加事务特性修改或添加该事务。

13、根据权利要求 12 所述的方法，还包括在检索到与特定事务相关联的事务特性后，答复所述第一事务管理器的步骤。

14、根据权利要求 12 所述的方法，其中所述第一事务接口包括允许所述第一应用程序处理指定将要与特定事务相关联的事务特性的多个功能件，多个事务特性可以关联所述特定事务。

15、根据权利要求 12 所述的方法，所述第一事务接口包括供第一应用程序处理使用的 setProperty 功能件，将附加信息与事务上下文一起传播。

16、根据权利要求 12 所述的方法，其中所述事务特性是关键字值对，所述关键字值对的关键字元素是字串，以及所述关键字值对的所述值元素是串行化实体。

17、根据权利要求 12 所述的方法，其中每个所述事务接口都扩展 Java 事务接口。

18、根据权利要求 12 所述的方法，其中所述第一事务接口包括把事务名称与事务相关联的 setName 功能件。

19、根据权利要求 18 所述的方法，其中所述 setName 功能件用事务类型命名事务。

20、根据权利要求 12 所述的方法，其中所述第二事务接口包括

addProperties 功能件，用于将一组特性添加到所述特定事务。

21、根据权利要求 20 所述的方法，其中所述第二事务接口还包括 setProperty 功能件，用于设置与所述特定事务预先关联的特性的值。

22、根据权利要求 12 所述的方法，其中当所述第一应用程序处理启动事务时，所述第一事务管理器创建唯一的事务标识符。

用于具有事务特性特征的事务处理的系统和方法

技术领域

总的来说，本发明涉及应用程序和事务服务器，特别是涉及一种包括带有一个事务或者一个消息的诸多事务特性的系统。

优先权要求

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_____的美国实用新型申请"具有事务属性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60/306102的美国临时申请"具有事务属性特征的事务处理的系统"，上述申请在此引用，以资参考。

交叉参考

本申请涉及共同待审的实用新型申请"具有委托提交特征的事务处理的系统（SYSTEM FOR TRANSACTION PROCESSING WITH DELEGATED COMMIT FEATURE）"（申请号____，发明人：Edward P. Felt, Priscilla Fung, Alex Somogyi, Sriram Srinivasan；2002年7月15日提交），和"具有同步调回处理特征的事务处理的系统（SYSTEM FOR TRANSACTION PROCESSING WITH SYNCHRONIZED CALLBACK PROCESSING FEATURE）"（申请号_____，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

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客户方案，而且还不把所得到的平台独立的J2EE基于组件的技术方案捆绑到任何一个卖方

的产品和应用程序接口上。

J2EE 规范定义了以下各种组件：应用程序客户组件、企业 JavaBeans、servlet（小服务程序）和 Java 服务器页（JSP）（还称之为 Web 组件）、小应用程序（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（超文本标记语言）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。

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。

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，它们共同构成执行 bean 的运行时间环境。

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。

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）数据源对象的位置，而环境项目又用来查找尾端层中的资源以及消息连接。

在 Web 和企业 bean 组件上的部署时间，可以在 Web 和企业 bean 组件上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。

某些应用程序服务器，比如产自 BEA Systems 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。安全区域允许管理员把来自现有的授权或者确认系统的信息输入到 ACL 中。

Java Servlet

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 被汇编成 Java 字节码，并且 servlet 事件在运行时间被保存在存储器中，每个客户请求在单独线程中运行。servelt 使它容易地以动态方式产生到达 HTTP 答复流的数据。每个客户请求按一个新连接来执行，所以流控制不会自然干预请求。为此，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。

Java 服务器页面

JSP 页面是一个开发 servlet 的基于文本的核心表达途径（presentation-centric way）。JSP 页面提供了 servlet 的所有优点，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 页可以和 JavaBeans 一起用来定义 Web 模板，通过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

数据显现，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。

数据库访问业务（JDBC）

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（scrollable result sets）和批更新的功能。

Java 消息传递业务（JMS）

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：

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。

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。

Java 接口定义语言（IDL）

CORBA 对象使用 IDL 指定一个契约，即它们如何与其它对象进行交互。有了 Java IDL，就可以在 Java 领域 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。

企业 JavaBran（EJB）

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（container）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 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。

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。

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。

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 瞬态字段值加会话 bean 字段可达到的所有对象。有状态 bean 不代表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。

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。

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 是全规模分布式计算协议（full-scale distributed computing protocol），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。

事务管理

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:

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。

一致性——一个成功的事务把数据库从以前的有效状态转换到新的有效状态。

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。

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）。

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，则返回这些步骤。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，则返回记入借方操作。

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:

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。

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。

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。

Weblogic 服务器是应用服务器的一个实例，该应用服务器支持用于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数据库中，或者从所有数据库充分返回事务更新，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:

事务始发者——启动事务。事务始发者可以是一个用户应用程序、一个企业 JavaBeans、或者 JMS 客户。

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，事务管理员帮助资源管理员决定提交或者返回待决事务。

可恢复资源——提供数据的持久性存储。该资源通常是一个数据库。

资源管理器——提供对信息集合和处理的访问。

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）。

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。

商务事务的支持

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）。

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：

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。

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，或者返回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返回所有编辑（在此情况下，各数据库编辑不一定进行 EJB 或 RMI 调用。一个客户如一个小应用程序使用 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。）

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。

上述传统的事务管理方法的问题是，它们未考虑将描述或功能特性附加到特定事务上，或者与其相关联。例如，应用程序或者事务服务器不能标识任何给定事务的上下文信息。这可以关联由旅行代理商使用的系统，但是不能标识一个例如是一个频繁飞行乘客的特定顾客（和由此得到的事务），并且以一个不同方式处理这些特定事务时，或许具有一个较高优先级。正在调用的对象不具备事务上下文的知识，并且不能基于这样一个上下文作出明智的决定。电子商务的日益增加的复杂性和电子商务顾客的需求，决定了必须把附加灵活性加入到事务消息传递或者应用系统中，以满足这些需要。

发明内容

本发明提供了一种供事务处理系统使用的机制，该事务处理系统允许一个事务包括多个相关联的事务特性，而这些事务特性又允许指定对象与特定事务相关联。无论事务何时激活都可以得到事务特性。当事务流向分布式系统的各服务器时，事务特性可以用作“附件”或者带外通信机制。

JTA 通常提供商务事务的支持。本发明的一个实施例提供的 JTA 的实施提供了以下商务事务的支持：

当客户应用程序启动事务时创建一个唯一的事务标识符，并且支持描述事务表达的商务处理的可选事务名称；

当代表事务访问资源管理器时，通知资源管理器。资源管理器随后锁定访问的记录，直至事务结束；以及

当事务完成时，组织两阶段提交，从而确保事务的所有参与者可靠地提交它们的更新（即，或者执行所述更新，或者不执行更新）。它使该提交与使用 Open Group's XA 协议进行更新的任何一个数据库相一致，当必须停止事

务时执行返回过程，当出现故障时执行恢复过程。它确定在系统性故障的时候，哪些事务在机器内激活，然后确定是否应当返回或者提交事务。

本发明提供的 JTA 还管理事务超时-如果商务事务耗费了太长的时间或者由于故障导致仅仅部分完成，则系统采取行动，自动地发出事务和自由资源的一个超时，比如数据锁定。

一般来说，本发明的一个实施例包括一种在事务从第一应用处理到第二应用处理的通信期间，将事务特性与事务相关联的系统，该系统包括：第一事务接口，包括允许所述第一应用程序处理指定将要与特定事务相关联的事务特性的功能件；第一事务管理器，将所述事务特性与所述特定事务相关联，其中所述事务特性允许指定对象与事务相关联；通信机制，把所述特定事务与所述相关联事务特性一起从所述第一事务管理器传送到第二事务管理器；第二事务管理器，检索与所述特定事务相关联的所述事务特性；和第二事务接口，包括允许所述第二应用程序处理检索与所述特定事务相关联的所述事务特性并以附加事务特性修改或增加该事务的功能件。

本发明的第二实施例包括一种在事务从第一应用处理到第二应用处理的通信期间，将事务特性与事务相关联的方法，该方法包括以下步骤：在第一应用程序处理上，通过对第一事务接口的调用，指定将要与特定事务相关联的事务特性；在第一事务管理器上，将所述事务特性与所述特定事务相关联，其中所述事务特性允许指定对象与事务相关联；把所述特定事务与所述相关联事务特性传送到第二应用程序处理；在第二事务管理器上，检索与所述特定事务相关联的所述事务特性；和在第二应用程序处理上，通过对第二事务接口的调用检索与所述特定事务相关联的所述事务特性，并以附加事务特性修改或增加该事务。

本发明的另一实施例包括一种在事务从第一应用程序处理向多个其他应用程序处理传送期间将事务特性与事务相关联的系统，包括：服务器，具有第一应用程序处理，以及允许所述第一应用程序处理指定将要与事务相关联的事务特性的第一事务接口；第一事务管理器，用于将所述事务特性与所述事务相关联，并将所述事务与所述相关联的事务特性从所述第一事务管理器一起传送到多个其他应用程序处理；以及多个其他应用程序处理，其中，该多个其他应用程序处理中的每一个都包括事务管理器和与之相关联的事务接口，并且该多个其他应用程序处理中的每一个在将该事务传送到另一个应用

程序处理或者答复该第一应用程序处理之前，都将该事务与所述相关联的事务特性一起进行接收，并独立地修改或增加该事务特性。

本发明的另一实施例包括一种在事务从第一应用程序处理向多个其他应用程序处理传送期间将事务特性与事务相关联的方法，包括：在第一应用程序处理中产生事务；指定将要与该事务相关联的事务特性；将所述事务特性与该事务相关联；以及将具有该关联的事务特性的事务传送到多个其他应用程序处理，其中，该多个其他应用程序处理中的每一个在将具有任何经修改或增加的事务特性的事务传送到另一个应用程序处理或者答复该第一应用程序处理之前，都将该事务与所述相关联的事务特性一起进行接收，并独立地修改或增加该事务特性。

附图说明

图 1 显示了 J2EE 体系结构的图示；

图 2 显示了本发明实施例的事务特性特征的图示；

图 3 显示了供本发明使用的典型系统的图示；

图 4 显示了本发明实施例的事务特性处理的流程图；

图 5 显示了本发明实施例的包括事务特性的事务表达的图示；

图 6 显示了本发明实施例的不同事务特性字段的图示；

图 7 显示了供本发明实施例使用或者与其结合的系统的一个实例；

图 8 显示了本发明实施例的设置事务特性对的样本码的列表；

图 9 显示了本发明实施例的检索事务特性对的样本码的列表。

具体实施方式

一般来说，本发明提供了供事务处理或者消息传递系统使用的系统和方法，以允许事务包括一个或多个关联的事务特性。事务特性可以用来允许指定对象与特定事务相关联。这样，当事务在分布式系统的不同服务器之间流动时，事务特性可以用作从服务器以及在服务器之间提供信息的一个带外通信机制。尽管这里所述的各种实施是以 Java 术语描述的，特别是描述了 EJB 实施，但是本领域熟练技术人员将会明白，这里所述的事务特性机制也可以供其它系统和其它实施使用。

图 2 显示了本发明一个实施例的事务特性的操作示意图。如图 2 所示，

包括客户应用程序 104 的客户 102 借助事务与多个服务器通信。如图 2 所示，所述的多个服务器包括服务器 A 106、服务器 B 108、服务器 C 110。在正常环境下，必须在服务器 A、B、C 之间流动的任何事务有关数据都必须使用带内通信来传递，即通过使用良好定义的接口从一个服务器到另一个服务器显性地传递信息。然而，本发明考虑了事务有关数据的带外通信，即可以借助实际信息流隐性地传递信息，并且不需要经由接口显性地发送。例如，如图 2 所示，客户应用程序 104 代表事务 112 请求使用服务器 A 106 的资源。作为事务上下文的部分，事务特性 114 可以与事务 112 相关联。任何特定事务可以包含一个以上的服务器。在图 2 所示的实例中，事务不仅包含服务器 A 106，还包含服务器 B 108 和服务器 C 110。当事务从服务器 A 流到服务器 B 然后流到服务器 C 上时，与该事务关联的特性可以被任何一个服务器修改和/或读出。例如，如图 2 所示，当包含特性 114 的事务 112 从服务器 A 传送服务器 B 时，可以附上一个附加事务特性 116。组合的事务特性 120 作为事务处理的部分被传送到服务器 C。同时可以把更新的事务特性 120 传送到或者送回 122 到服务器 A。当原始呼叫（从服务器 A 到服务器 B）返回到服务器 A 时，这些更新的事务特性还将被自动地送回到服务器 A。这样，系统提供了客户应用程序设置事务特性的能力，该事务特性经由分布系统中的不同服务器或者模块传播。其作用是提供特定事务所执行的软件组件的带外通信机制。下面详细说明描述该机制的附加实施。

因此，本发明提供了供任何事务处理系统使用的机制，该事务处理系统允许包括多个关联的事务特性。事务特性又允许指定对象与一个事务相关联。只要事务是有效的，事务特性就是可用的。当事务在分布系统中各个服务器之间流动或者进行通信，事务特性可以被用作附件或者带外通信机制。

在一个实施例中，在标准 JTA 规范不能覆盖实施细节的情况下，或者在需要附加能力的情况下，本发明提供了对标准 Java 事务 API 的扩展。本发明还提供了允许客户启动事务的能力。标准 JTA 事务管理器接口（Javax.transcation.TranscationMangager）被制作成经由 Java 命名目录接口（JNDI）用于客户和 bean 供应商。该接口允许客户和使用 bean 管理事务的 EJB 暂停和恢复事务。事务-事务的范围可以在两个群集内和在其之间操作。

图 3 显示了可利用本发明的典型系统的示意图。远程装置或应用程序 202 通常被称之为客户应用程序，它与应用程序服务器 204 通信，或者更典型地

与应用程序服务器上运行的应用程序进行通信。在客户端，客户应用程序使用事务管理器 206，并经由接口 207 向和从服务器（或者向或者从许多服务器）发送和接收事务 208、212。同样，在服务器端，事务管理器 210 和接口 211 用来管理事务与客户。在一个实施例中，任何一个服务器都可以充当用于启动事务的事务协调器 214、用于监视事务状态的子协调器 216、或者用于记录事务的事务日志器。显然，这些作业之每个还可以由同一（即一个）服务器处置，而不是由一组服务器处置。当事务发生时，可以将这些事务记录到事务数据库 220 中，以用于以后的审计、引用或者返回目的。本发明提供了允许诸多特性与每个事务相关联的事务管理器。当事务被传送 222 以及由此发生反应时，客户或者服务器应用程序可以分析这些特性。

为了实现这些功能，本发明的一个实施例提供了一组类别和接口，以作为对标准 JTA 的扩展（需要注意的是，尽管为了说明目的，在某些以下实例中使用了名称“Weblogic”以便将新类别于标准 J2EE/JTA 类别相区别，但是本发明不特别地局限于 Weblogic 系统，也不局限于下文介绍的示例类别或者接口）。

事务特性

图 4 显示了本发明实施例的事务特性处理的流程图。如图 4 所示，客户的事务管理器接收来自客户应用程序的创建或启动事务的请求（步骤 302）。客户应用程序（在某些情况下实际上可能是一个不同的参与客户应用程序）还可能在步骤 304 生成与该事务关联的特性。

在步骤 306，客户应用程序或者某些外部应用程序生成事务的特性。客户事务管理器接收来自客户应用程序的创建事务的请求，并把任何预期特性与事务相关联（步骤 308）。在步骤 310，将事务与相关联的特性一起发送给服务器。服务器的事务管理器接收上该事务（步骤 312），服务器指示事务管理器检索任何关联的事务特性（步骤 314）。任何发现与事务关联的特性被用来修改或者添加事务（步骤 316），并且将该事务传送给服务器应用程序（步骤 318）。

根据一个实施例，事务特性是关键字/数值对，事务特性包括一个和多个关键字以及诸多关联值。一旦设置了事务特性，就可以在它的整个寿命时间使其保持与事务关联。当事务传遍系统时，随着事务传遍整个系统，特性在

机器（计算机）之间传播。特性还可以保存到事务日志中，并在崩溃恢复处理期间恢复。根据本发明的实施例，如果不止一次设置事务特性，那么将保留最后数值。然而，在某些线程和/或多服务器情况下，该行为不是预期的甚至不可预测。

恢复和审计尾迹（audit-trail）过程通常需要比 JTA 事务接口供给更多的关于事务的信息；比如事务类型的信息，事务标识符的信息。返回的具体原因的信息，以及任何试差消息的信息。事务耗费的时间和剩余的时间量可以被用作某些操作的界线。

根据本发明的一个实施例，为了设置事务特性，事务业务的用户（或者客户应用程序）使用一个 getProperty/setProperty 接口传播附加信息与事务上下文。关键词可以是字串，数值可以是任何可串行化的实体。某些关键字，例如始于字"weblogic"的某些关键字可以被保留，或者具有特定含义；而其它关键字不能被业务理解，但是能够与事务一起传播，并且当事务流经系统时，可适用于事务中介物和事务的接受者。

事务接口

事务接口用来控制事务特性特征的运行。以下接口示出了本发明的特定实施，因为它可应用于 Weblogic 服务器系统。本领域熟练技术人员将会明白，在保持本发明的精神和范围的同时，可以修改以下说明的接口和技术，以供其它系统使用。

weblogic.transcation.TransactionManager

该事务管理器接收扩展了 J2EE 标准 Javax.transaction.Transaction 管理器接口，并且由 JTA 事务管理器对象支持。它允许 XA 资源在建立时借助事务管理器注册和解注册它们自己。可以通过 JNDI 引用 TansactionManager，以防止必须输入任何应用 application-server-specific（程序-服务器-指定）类别：

```
javax.naming. Context ctx= get InitialContext ();
TransactionManager tm =(TransactionManager)
ctx. lookuo('javax. transaction. TransactionManager");
tm. begin(); // 开始事务。
Transaction tx =tm. get Transaction();
```

tm. commit();

事务管理器可以被所有应用程序线程高速缓存和使用。在此方式下，仅需要在处理寿命时间作一次 JNDI 查找。获得和使用用户事务遵循与上述相同的方式。用户事务可以从 JNDI 中获得，然后高速缓存和重用该用户事务。

java. naming. Context act = getInitialContext();

UserTranscation ut = (UserTranscation)

ctx. lookup("javax. transaction.UserTransaction") ;

ut. being() ; // begin a transaction.

ut. commit() ;

weblogic.transcation. Transaction

weblogic.transcation.Transaction 接口是扩展标准 javax.transaction.Transaction 的事务接口，并且允许用户或者应用程序得到和/或设置事务特性。在一个实施例中，事务接口包括以下方法:

setName(java.lang.String n)

setName()方法或功能将名称与事务相关联。事务名称被打算用作对事务的应用程序特定语义（"类型"）的参考，但是对于事务的每个实例不是唯一的。事务名称可以被认为表达商务事务的类别。指定名称给事务可以提供更好的差错报告。如果出现差错，就把内部生成的差错报告并入一个名称。例如，功能 tx.setName（"资金转帐"）在金融业务操作中用来给予事务名称"资金转帐"。

getName()

getName()功能检索先前与一个事务关联的名称，（即它恢复事务名称）。

addProperties(java.util.Map props)

addProperties()功能通过调用特性的每个元素，将一组特性添加到一个事务上。该功能接受作为参数的多个特性-（即添加一组关键字/数值对）。

setProperty(java.lang.String key, java.io.Serializable value)

setProperty()功能设置与事务关联的特性值。该功能接受作为参数的关键字-（一串关键字），和数值-（特性值，其中零的数值意味着删除特性）。

getProperty(java.lang.String key)

getProperty()功能获得特定特性的数值，并接受作为参数的关键字-（特性名称）。则该功能恢复特性数值，或者如果复位，则为零。

getProperties()

getProperty()功能获得当前与事务关联的所有特性。该功能返回包含所有关键字/数值对的对象。

setRollbackOnly(java.lang.String msg, java.lang.Throwable reason)

setRollbackOnly()功能将返回字串消息和异常事件包装成 NestedException 事例，并用该新的异常来调用 setRollbackOnly。

setRollbackOnly(java.lang.Throwable reason)

该功能()提供了发生任何返回的原因。在事务业务内触发的所有返回设置该原因（超时，XA 差错，完成之前未经检查的异常，无能力接触协调器，等）。该功能接受作为参数的原因数值-（返回的原因）。

getRollbackReason

getRollbackReason()功能检索与事务关联的返回原因（若有的话），并把返回原因返回到任何调用应用程序。

getHeuristicErrorMessage()

getHeuristicErrorMessage()返回与事务关联的（若有的话）启发式差错消息。

getXID()

getXID()获得事务标识符。在某些事例中，事务处理在调用一个资源时可以把某些附加信息添加到 Xid's 限定符上，所以在调用资源之前对

tx.getXID().getGlobalTransactionId()调用（在第一时间），并且在调用后，一个资源可以不同于实际返回的资源。

例如在金融业务情况下，事务标识符（xid）可以包括分支限定符，它指示启动事务的分支。当事务被送达另一个分支时，可以修改该限定符，但不修改事务本身的内容。在某些实施例中，由系统提供一些保证：

1. xid.euqals()不受影响。从 tx.getXID()获得的 xid 和从后续时间获得的另一个 xid 被认为相等（对于相同事务）。该保证不扩展到 xid 的组分即限定符。

2. 在其 XA 资源方法中，XA 资源总是被供应给定事务的相同 xid（具有相同的限定符）。

getStatusAsString

getStatusAsString() 功能返回一个字串的事务状态该字串包含如 javax.transaction.Status 中指定的主状态，此外还用一个次要状态装饰（比如“记录”，“预先准备”）。

getMillisSinceBegin()

getMillisSinceBegin()功能在几毫秒内返回该事务现有的时间量。

getTimeToLiveMillis()

getTimeToLiveMillis()功能几毫秒内返回事务剩余的时间量。该时间仅仅在事务有效时有意义，当它正在重试一个提交或返回时则没有意义。

weblogic.transaction.TxHelper

TxHelper 类允许用户或者应用程序获得当前事务管理器和事务。

```
import    weblogic. transaction. TxHelper;
transactionMaganer tm =TxHelper. gerTransactionManager();
transaction tx = TxHelper. gerTransaction();
```

事务对象可以转到事务接口的扩展，所述事务接口传送其它上下文数据并获得具体状态信息。

```
import weblogic.transaction.Transaction;
Transaction tx = (Transaction) TxHelper. getTransaction();
```

weblogic.tranaction.XARespurce

该接口提供 XA 资源的删除能力。TA 资源接口还表达两阶段提交协议参与中的资源契约，包括获取、删除、准备、提交和恢复。JTA 规范没有使某些细节标准化，如何启动事务资源，或者用户或应用程序应该如何获得一个实现 TransactionManager 或 UserTransaction 接口的对象。下面部分公开了这样的细节，并且还通过实例描述了可以由本发明提供的事务和事务管理器接口的多个扩展。

图 5 显示了本发明实施例包括的事务特性的事务 402 的表达视图。所示的事务格式仅为可以用于金融业务环境中的事务格式的实例，并且包括格式 ID 404，全局 ID 406，协调器 410，时标 412，计数器 414，分支限定符 416 和资源名称 418。该事务格式可以用来把特性或限定符添加到特定事务上，或许把该事务识别为属于顾客（即，例如“黄金”或“白金”级顾客）的特定类别，或许标识一个始发或者中间银行分支。本领域熟练技术人员将会明白所示的实例格式只是一个实施例的图示说明，在保持本发明精神和范围的同时，还可以使用其它格式。

图 6 显示了本发明一个实施例的各种事务特性字段的图示。该特性包括跟随数据的格式版本号 502、事务标识符 504、使用期限 506、System_Defined 特性字段 508、其它特性 510、资源管理器名称 512、带不注册同步返回的子协调器名称 514 和带注册同步返回的子协调器名称 516。这些字段的某些可能是必需的，也可以认为是可选字段。

对事务管理器和用户事务的引用

尽管 JTA 规范指定了这些接口，但是该规范未批准一个得到当前事务的方式。EJB 规范仅提供了从 bean 内部得到当前事务的方式。使用本发明就不必输入任何特定类别来获得对这些接口的引用。

对客户启动事务的支持

JTA 规范没有明确定义客户如小应用程序是否可以获得对事务管理器的引用和启动事务。使用本发明，就可以使其成为可能。J2EE 标准允许 J2EE 客户使用 JNDI 活动地获得对一个用户事务对象的引用。利用本发明，该设施

还可用于事务和事务管理器对象。

跨群集和非群集业务的能力

本发明允许事务在一个远程方法调用成为可能的时候前进。在特定事务范围之内接触的所有服务器是可协调的，这种协调被限制在强加了安全设置或者选择权的任何约束的范围内。

用户定义的数据段

由于事务上下文隐性跟随控制的线程，以及由于事务界线通常代表一个微小商务作业，因此本发明提供了由它们的类型命名的事务（例如"资金转帐"）的方式，以及将用户定义数据与上下文一起传送的方式。该信息随后可以用于差错报告和统计采集原因。

差错报告

为了改善差错报告，根据本发明，JTA 业务可以提供借助一种方式获得事务返回的潜在原因或者在调用 setRollbackOnly()或 rollbackzhiqian 提供原因的应用程序。标准 Java 规范 JTA 业务不供给这一需求。本发明还允许 JTA 业务获得事务标识符和详细状态报告；并确保所有消息包含具商务级别意义的充足细节。大多数传统事务系统将会列出一组事务标识符和它们的状态，这对于操作员几乎没有意义。

对 beforeCompletion 期间 bean 调用的支持

标准的 EJB 和标准 JTA 规范都没有定义如何处置 bran's beforeCompletion 或 ejbStore 方法调用另一个其 bedoreCompletion 和/或 ejStore 方法已经被调用的另一个 bean 的哪些情况。本发明支持这样一种通过诸多服务器的方案。作为按 beforeCompletion()返回方法运行的应用程序码的结果而联系的任何新服务器或资源都被跟踪。因而系统确保这些服务器和资源在事务提交处理中充分参与，包括必要时进行附加的 beforeCompletion()调回。

EJB 交互

当使用 EJB 生成一个被管理事务时，EBJ 容器在启动事务的时候，自动

地设置事务名称。如果开发商没有在调度描述符中供应一个名称，则 EJB 业务容器可以在开始一个事务的时候自动地设置一个事务超时。如果开发商未在调度描述符中供应一个超时值，则 EJB 容器将超时设置为零。在此情况下，使用基于配置参数的默认超时。EJB 调度描述符可以指定诸多特性，这些特性是为 EJB 容器开始的任何事务设置的。可以设置的一个有用特性是事务跟踪特性。跟踪特性允许在一个服务器上启动跟踪，然后经由具有该事务的网络流到第二服务器。跟踪专用于与单个事务关联的控制的线程，在繁忙的产生系统中这是一个优点。跟踪输出被打印在每个服务器差错日志文件中，跟踪输出具有采用应用程序码、资源或者配置设置诊断问题的用途。

JMS 交互

JMS 接受配置参数，以便控制事务名称、超时和事务特性。

实施例

图 7 示出了供本发明使用或者并入本发明的一个实例。如图所示，客户或者客户应用程序 602 生成一个事务 607。另一个客户应用程序 604，或服务器处理或者功能 606 也许希望修改或者增加事务，或许增加特性 608，该特性 608 将事务标识为属于一个非常重要的顾客或者一个频繁飞行的顾客等等。事务管理器处理 610 在与客户通信时操作，它或者作为在客户在运行的处理或者作为在中间服务器上运行的处理。该事务管理器处理 610 被用来将特性与事务相关联，例如通过使用 setProperty 命令或者设置特性接口实现该关联。然后将事务 607 与关联的特性 612 一起传送到一个收件人服务器（或者多个服务器）614，其中另一个事务管理器 616 通过使用 getProperty 接口检索关联的事务特性。根据关联的特性，接收收件人服务器或者处理以多种方式对事务起作用，以适应它们的需要。

图 8 显示了本发明实施例的用于设置一个事务特性对的样本码 702 的列表。此外，该实例还涉及金融业务环境，在该环境下，事务在客户应用程序与服务器应用程序之间来回传送，或许实行赊帐、借记、公平销售等。在图 8 所示的实例中，客户应用程序使用 setProperty 功能将有价值顾客特性（“等级”=“VIP”）与特定事务相关联。当事务在整个系统上从客户流动到服务器，或者或许从一个服务器流动到另一个服务器时，该“VIP”特性作为一个事务特

性与事务一起移动。

图 9 显示了本发明实施例的用于检索一个事务特性对的样本码 802 的列表。在该实例中，服务器应用程序例如使用一个对 getProperty 的调用，来检索与事务预先关联的“VIP”事务特性。在这里，服务器应用程序检索一个标识和增加事务的发起者或者所有者的特性的“等级”值。在图 9 所示的实例中，等于“VIT”的“等级”的值指示事务应当受到特别对待。服务器应用程序可以适当地响应这一事务特性，这或许通过以较高优先级处理事务请求，或许通过将事务传送到专门处理“VIP”事务的一个特定服务器进行响应。

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。

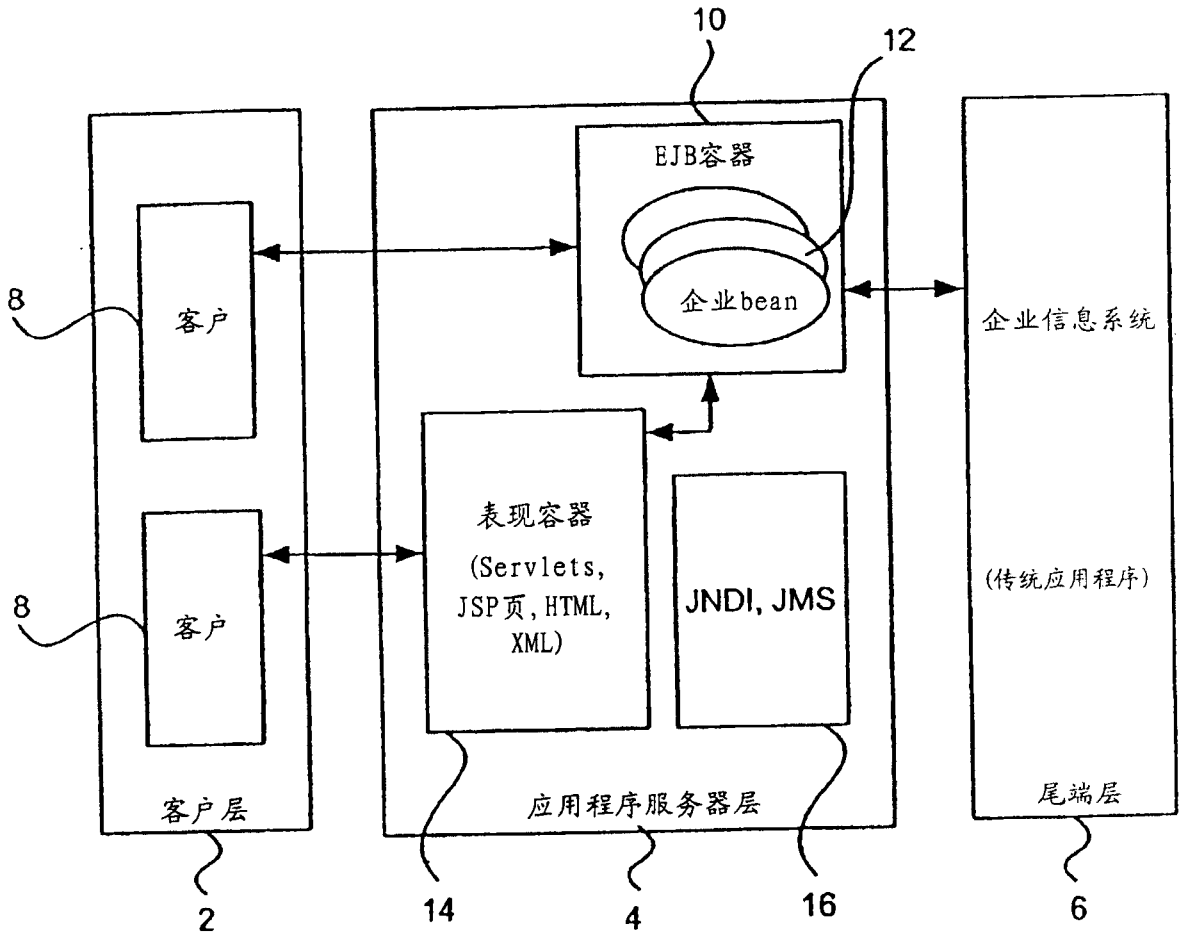

10
12
EJB容器
企业bean
企业信息系统
8
客户
8
客户
表现容器
(Servlets,
JSP页,HTML,
XML)
JNDI, JMS
(传统应用程序)
客户层
应用程序服务器层
尾端层
2
14
4
16
6

图 1

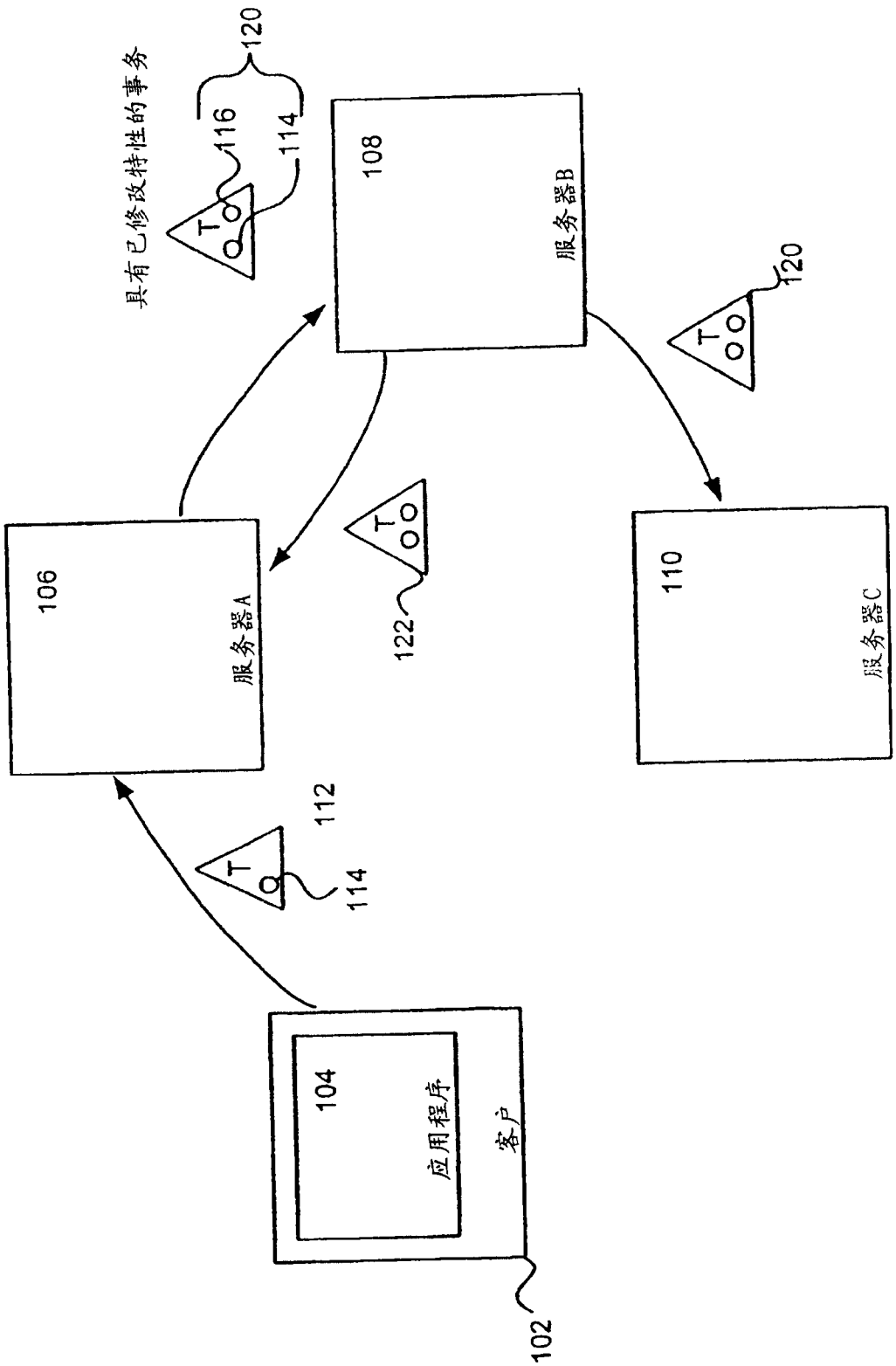

具有已修改特性的事务
120
116
114
108
服务器B
106
服务器A
122
110
服务器C
120
112
114
104
应用程序
客户
102

图 2

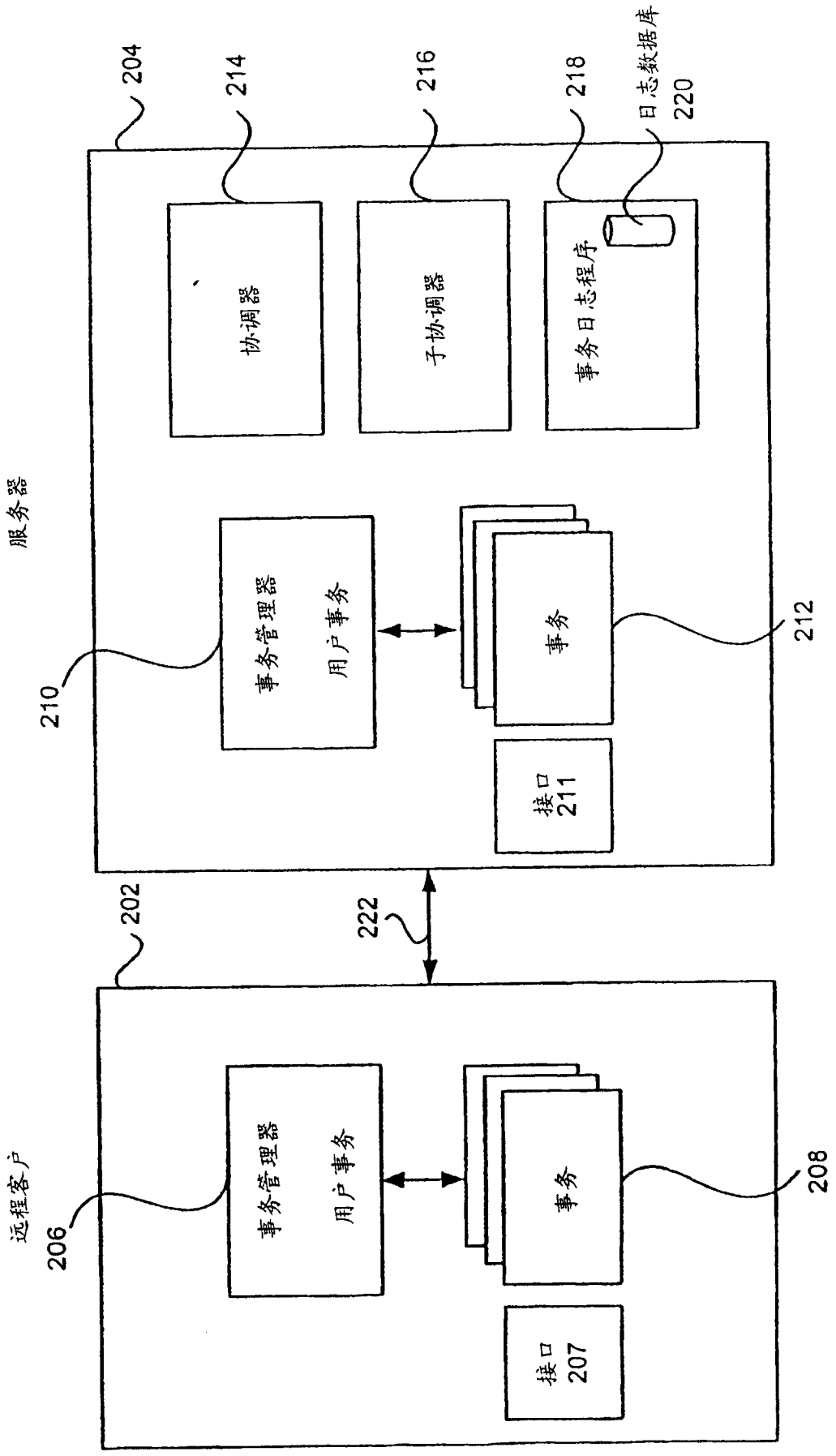

远程客户
206
事务管理器
用户事务
事务
208
接口
207
202
222
服务器
210
事务管理器
用户事务
事务
212
接口
211
204
协调器
214
子协调器
216
事务日志程序
218
日志数据库
220

图 3

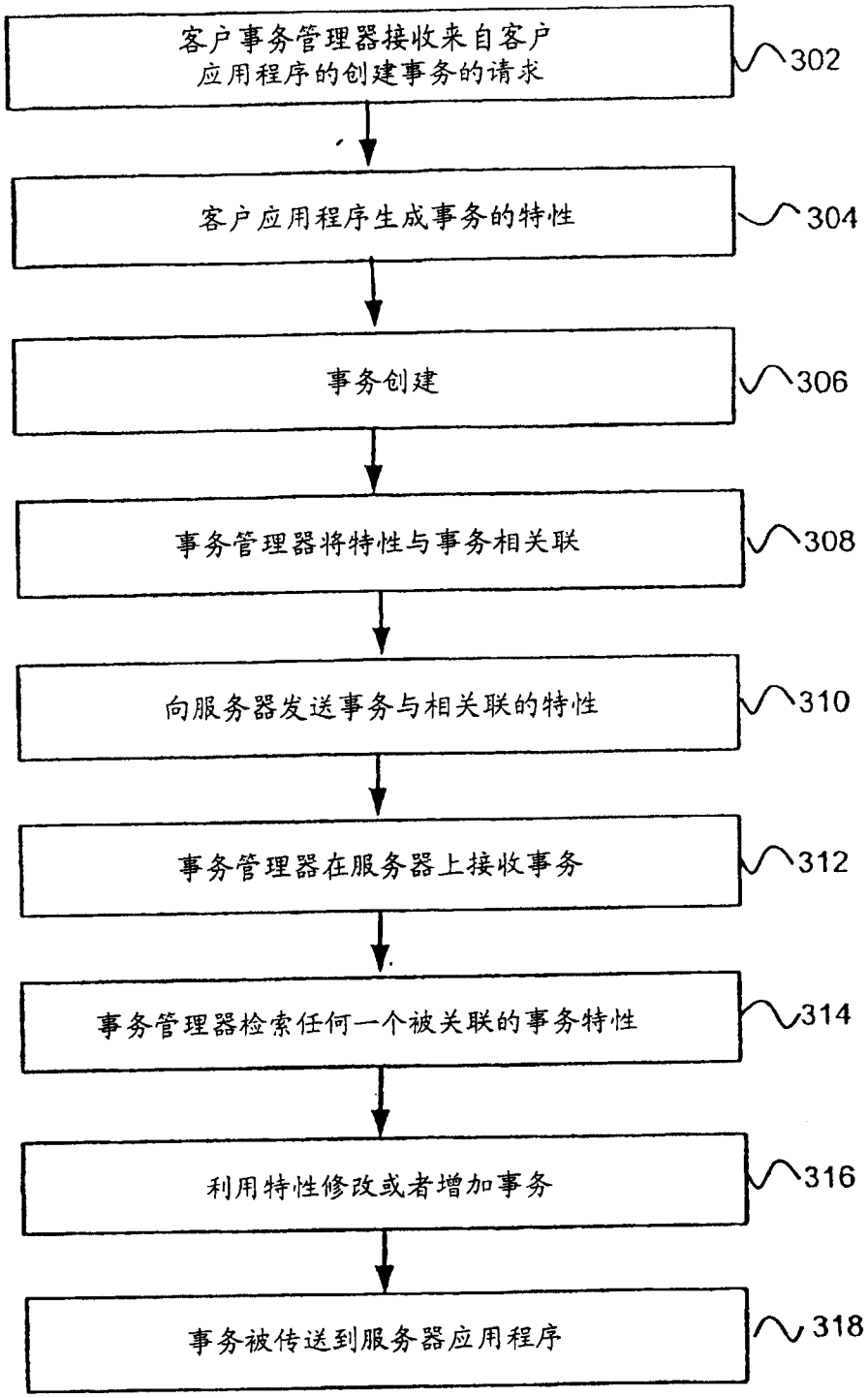

客户事务管理器接收来自客户应用程序的创建事务的请求
302
客户应用程序生成事务的特性
304
事务创建
306
事务管理器将特性与事务相关联
308
向服务器发送事务与相关联的特性
310
事务管理器在服务器上接收事务
312
事务管理器检索任何一个被关联的事务特性
314
利用特性修改或者增加事务
316
事务被传送到服务器应用程序
318

图 4

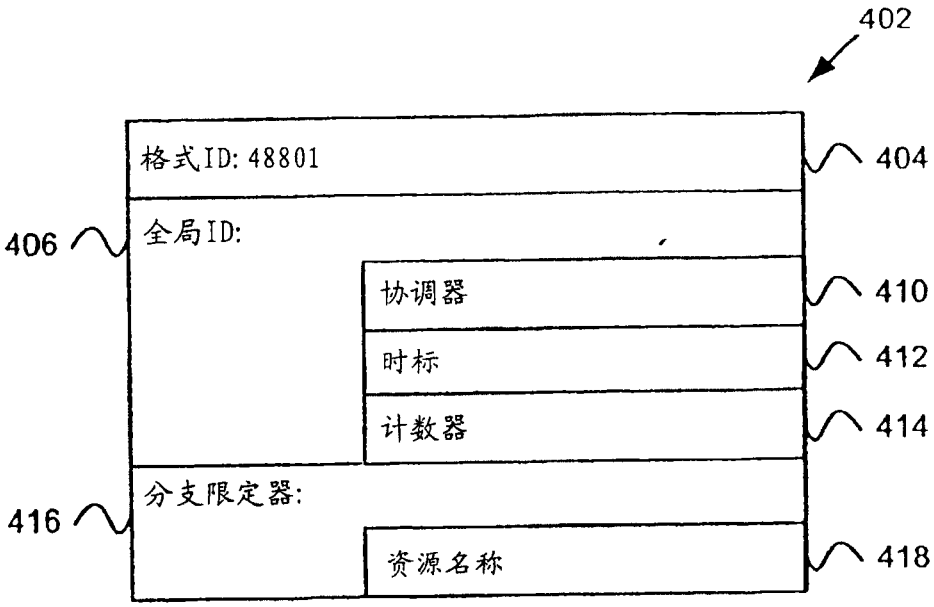

402
格式ID: 48801
404
406
全局ID:
协调器
410
时标
412
计数器
414
416
分支限定器:
资源名称
418

图 5

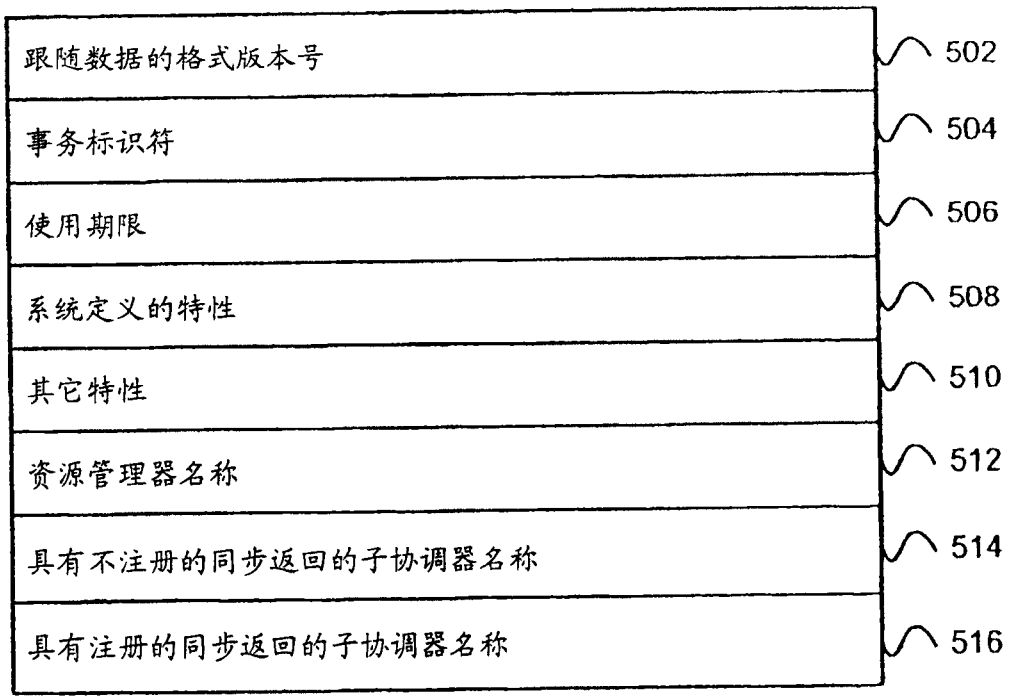

跟随数据的格式版本号
502
事务标识符
504
使用期限
506
系统定义的特性
508
其它特性
510
资源管理器名称
512
具有不注册的同步返回的子协调器名称
514
具有注册的同步返回的子协调器名称
516

图 6

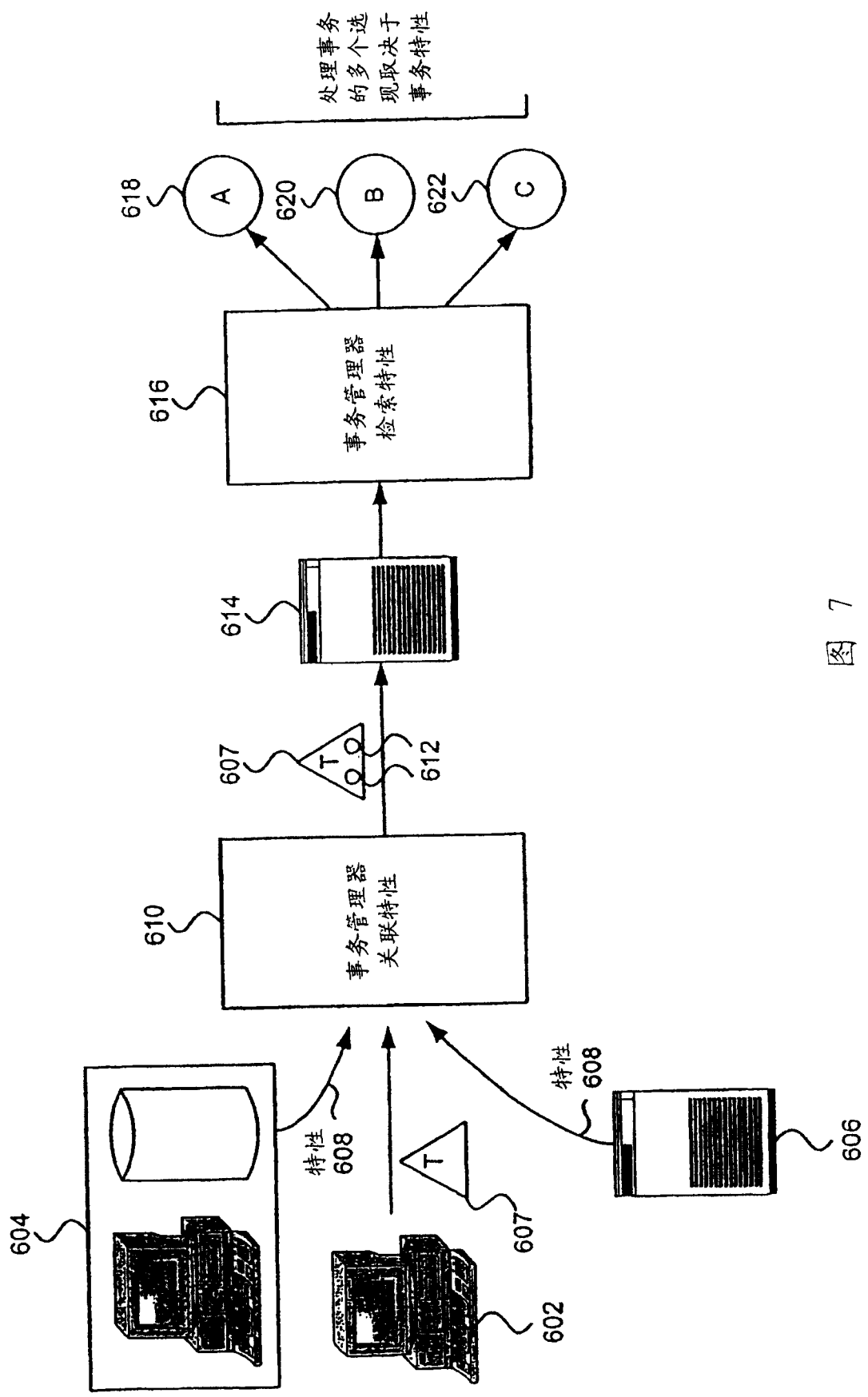

处理事务的多个选现取决于事务特性
618
A
620
B
622
C
616
事务管理器检索特性
614
607
T
Q
Q
612
610
事务管理器关联特性
特性608
特性608
T
607
604
602
606

图 7

```
import server.transaction.Transaction;
import server.transaction.TransactionManager;
import server.transaction.TxHelper;
...
TransactionManager tm = TxHelper.getTransactionManager();
Transaction tx;
tm.begin();
tx = (server.transaction.Transaction)tm.getTransaction();
tx.setProperty("rating", "VIP");
```

702

图 8

```
import server.transaction.Transaction;
import server.transaction.TransactionManager;
import server.transaction.TxHelper;
...
int balance;
...
TransactionManager tm = TxHelper.getTransactionManager();
Transaction tx;
tx = (server.transaction.Transaction)tm.getTransaction();
String rating = (String)tx.getProperty("rating");

if (balance < 0 && rating.equals("VIP") {
  // Overdrawn VIP customer gets special treatment
}
```

802

图 9