第35卷 第24期 计 算 机 工 程 2009年12月
Vol.35 No.24 Computer Engineering December 2009
·软件技术与数据库· 文章编号：1000—3428(2009)24—0099—04 文献标识码：A 中图分类号：N945

基于 Web Service 的工作流补偿机制

张晓雯，黄永忠，李占峻
(信息工程大学信息工程学院，郑州 450002)

摘 要：针对 Web Service 事务处理机制中的补偿问题，提出一种半自动的事务补偿机制，使用数据库系统中的触发器技术，解决了在 Web Service 分布、异构平台上对事务活动进行统一的补偿问题，将机制应用在 Web Service 中，可以规范补偿逻辑，从而保证系统的一致性，Web Service 设计者可以确定补偿的规则，并在事务运行时动态生成补偿策略。实验结果表明，该方法有利于集中利用补偿策略，具有良好的可扩展性。

关键词：Web 服务；工作流；触发器；事务；补偿

Workflow Compensation Mechanism Based on Web Service

ZHANG Xiao-wen, HUANG Yong-zhong, LI Zhan-jun
(Institute of Information Engineering, Information Engineering University, Zhengzhou 450002)

【Abstract】 Aiming at the compensation problem in Web Service transaction processing mechanism, this paper presents a semi-automatic compensation mechanism for the transaction. The mechanism uses the database system triggers technology to solve unified compensation in the distributed and heterogeneous platforms. This mechanism is applied in Web Service, which can standardize compensation logic in order to ensure system consistency. Web Service designers can determine the compensation rules, and dynamically generate compensation at the transaction run-time. Experimental results show this method is beneficial to focus on the use of compensation strategies, with a certain degree of expansibility.

【Key words】 Web Service; workflow; trigger; transaction; compensation

1 概述

在保持数据一致性的关键领域，事务处理起着重要作用，如交易系统和医院的信息系统。一个事务映射到数据库中，就是从一个一致性状态通过自动执行一系列操作到另一个一致性状态，同时隐藏发生的影响，并恢复失败的操作。因此，事务被划分成适当的块，以便于系统恢复。

多种类型软件的可重构性由事务保证，如工作流系统、移动系统和最近兴起的 Web 服务系统。新的事务应用不断出现，传统工作流的局限性就暴露出来并为大家所知。为处理这些领域的事务，许多折中或扩展的事务模型已经出现[1]。

传统的两阶段提交协议(2PC)已经用于协同工作(如 X/Open DTP, CORBA OTS)。由于松耦合和自治系统环境需要网络服务，因此基于两阶段的提交协议不能很好地发挥作用。

触发器机制是促进补偿机制形成的有效统一的方法，通过建立事务补偿触发器机制，允许网络服务设计者指定补偿规则，这个规则在运行时动态生成补偿事务。这种机制支持简单的、一致性的补偿，也支持通过用户的半自动补偿。需要注意的是，本文中提到的补偿机制，不仅局限在网络服务上，其他依靠补偿来恢复事务的环境，也可以使用该方法。

2 Web Service 和事务补偿

Web Service 是种新的面向函数和方法的应用集成技术，它基于 XML 文档进行服务描述、服务请求和反馈结果，基于 HTTP 协议进行信息传递，易于被访问和返回结果。同时，由于 Web Service 基于 W3C 的开放协议，独立于平台和操作系统，不同的平台和操作系统上实现的 Web Service 在很大程度上可以做到互操作，这就使得异构平台上的应用易于集成。此外，相对于基于 RPC 和 API 的静态集成(即使它们在客户机和服务器通信时使用 XML)，Web Service 提供一种功态的集成方案，所有的服务都可以通过 UDDI 标准动态地被发现、绑定和使用，容易适应系统的变动，提高系统的灵活性和伸缩性，从其本质上来说是一个松散耦合、可复用的分布式计算模型，同时它将应用程序功能概念化成任务，从而形成面向任务的开发和工作流。基于 Web Service 的工作流系统实现对包装成 Web Service 的企业业务活动进行调用和控制，使得业务流中的不同业务活动的交互更加简单和方便[2]。

Web Service 之间的相互作用，通常是通过会话事务进行处理，在每个 Web Service 中，事务被分为很多个子事务，也称为事务组件。单个事务的行为，通常由本地的潜在数据库处理。此外，会话事务的行为必须通过管理协调子事务集得到保证。

不同于传统的分布式事务，会话事务不遵守传统事务的 ACID 性质。由于 Web Service 对松散耦合的性质和自治的要求，因此基于传统的两阶段提交协议就不再适用。如果一个或多个子事务终止，会话事务可能被取消也可能不被取消，这都取决于有关业务逻辑的服务。

对于子事务的恢复，研究人员已经提出使用补偿事务[3]，这在语义上恢复了子事务提交所做的工作。补偿事务保留了数据库的完整性，并且没有丢弃其他事务。补偿事务的概念

基金项目：国家部委基金资助项目
作者简介：张晓雯(1984－)，女，硕士研究生，主研方向：网格计算，分布式系统；黄永忠，教授、博士生导师；李占峻，硕士研究生
收稿日期：2009-10-10 **E-mail：**zhang_x_w@126.com

是由 Garcia-Molina 首先提出的，后来被应用在事务模型中。补偿事务的定义如下：

定义 补偿事务，即在语义上回滚事务 T 的部分影响，不执行级联回滚事务，维持系统一致性状态。

一般来说，设计补偿事务规则首先在被补偿的事务上执行撤销，然后还原其一致性或建立其他重要属性。举例来说，删除插入的数据，现存统计表中可能有错误并且需要更新，以恢复数据库中一致性状态。

虽然设计补偿事务是一个棘手问题，但是现在的工作流系统能够处理许多情况，都有足够的补偿事务，这些补偿事务被视为正常处理的一部分。本文提出使用触发器来指定和执行补偿事务的机制，这个机制应用在 Web Service 层上，可以规范并执行补偿逻辑，以保证系统的一致性。Web Service 设计者可以确定补偿的规则，并在事务运行时动态生成补偿。

3 基于触发器的补偿机制

基于触发器的补偿方法，是事务补偿中基于主动数据库的触发器概念，即事件-条件-活动(Event Condition Action, ECA)规则。为使这种方法适用于 Web 服务的范围，Web 服务必须在持触发器的数据库系统之上执行。支持传统事务(即 ACID 事务)也必须提供数据库系统，主要的数据库系统，无论使用触发器还是传统事务，基本上都遵从 SQL 1999 标准。

3.1 触发器

支持触发器是主动数据库的显著特点。主动数据库随时监视着系统的运行，当符合条件的情况出现，触发就及时被触发并引发操作。触发器的一般形式为

On Event
If Condition
Do Action

产生补偿操作采用事件驱动机制，只有那些影响应用系统的数据或状态的操作才会产生相应的补偿操作。驱动事件包括：数据修改操作(如对数据库的 insert, update 以及 delete 操作)；事务协调消息(如 begin, commit 和 abort 等)；用户自定义的事件；系统事件(如系统或机器重启)。

对触发器的支持扩展了数据库系统的功能。在传统的被动式数据库中，外部应用要根据不同的情况选择不同的数据库，并且行为只能在应用中执行。使用触发器后，数据库可以时时监控系统运行，更方便的是，行为可以在自身数据库中运行，简化了操作。

3.2 指定和执行补偿事务使用触发器

事务补偿过程通过 3 步来完成：

(1)生成补偿操作的规则，此规则基于应用程序逻辑，由 Web Service 开发者在事前指定；

(2)在子事务执行期间，根据此规则，数据库系统生成相应的补偿事务；

(3)当子事务提交时，对于子事务的补偿操作，融合进补偿事务中，所产生的补偿操作存储在数据库中供以后执行，根据恢复协议实现其功能。

在数据库中通过触发器定义补偿规则，所有改变数据库状态的子事务操作类型(更新、删除、插入等)都需要定义补偿规则。触发器规则由数据库模式设计器或者 Web 服务开发者使用 DBMS 的触发器语言定义。执行子事务时，每个操作都会产生一个相应的事件，如果执行操作修改了数据库状态，就会引起触发器的操作。在触发器执行时，对事件上下文进行分析，典型的是对数据的新旧状态进行分析，并确定操作类型和参数。如果确定该子事务不能得到补偿，此次操作可能会被拒绝，或者子事务不允许提交；否则，补偿操作会执行，其信息是从存储在数据库中的相关事务中抽取出来的。例如，如果一个事务中的某子事务表示向数据库中插入一个数据，那么该“插入”事件产生相应的补偿操作即对同一数据项的“删除”，此过程的规则描述代码简化如下：

```
on Insert
if done.equals(false)
then
do storeComOper ('Del'，newData)
```

根据该规则，在子事务执行过程中，每执行一次插入操作，都会在执行时产生相应的删除操作作为其补偿操作。分析上述代码，当表示插入操作完成的标示为假时，也就是对应补偿操作的执行条件得到满足时，触发器会触发对应的删除操作 storeComOper()，其入口参数有 2 个：指明执行动作的常数参数'Del'以及在已经执行的插入操作中的插入值，这里用 newData 表示，由于插入操作的数值不同，因此该参数设为变量参数，在异常发生时，通过记录相关的参数就可以获得正确的数值。图 1 为在数据库中生成并储存补偿操作的过程。

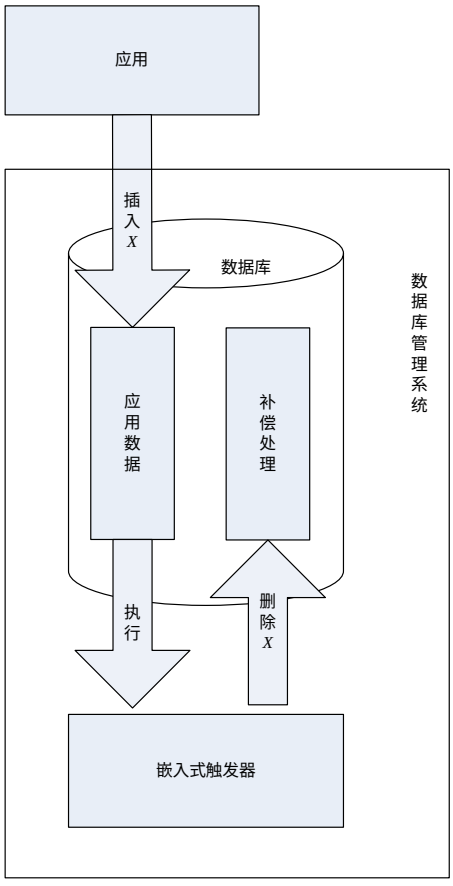

图 1 补偿操作的生成及储存

图 1 给出了存储补偿操作的具体过程，这里需要注意的是，补偿操作作为常规数据元素存储在数据库中的过程。可以看出，在特定数据库系统中使用嵌入式补偿事务已经取代了单一的存储补偿操作功能，由于其可以充分利用 DBMS 的语言规则创造各种合法的结构，因此这些功能可以任意复杂并且覆盖全面。在补偿事务中，相关事件用来为补偿操作提

供信息，条件语言决定是否进补偿行操作以及进行何种补偿操作。补偿操作的范围很广泛，包括对自身的撤消操作(如上例所示)以及调用外部函数进行补偿等。生成补偿操作的触发器通常采用以下形式：

on Event

do Generate Compensating Operations(Event Context)

If compensation cannot be performed Then reject the operation Or enforce commit-dependency between subtransaction and top-level transaction

Else Store Compensating Operations

触发器中条件子句(即 if 子句)是左触发，作为规则中行动的一部分，该规则应该无条件执行。

在某一事务运行过程中，触发器对子事务产生的补偿操作，是在触发事件发生后被告知并立即执行，这就是所谓的即时耦合模式[4]。对包含某一子事务的上下文再次运行，就是为了保证已提交的子事务能够得到补偿。这就需要终止子事务的执行，且所有在子事务执行过程中产生的补偿操作都要被终止。生成的补偿操作将一直保存在数据库中，直到子事务成功提交。

当子事务中的所有操作都通过触发器进行了上述处理，并且子事务也已提交，此时，存储的补偿操作就构成了补偿事务。在子事务提交时(最典型的是在应用程序执行一个提交操作时)，数据库管理系统就会生成一个事务补偿事件，该事件把已经生成的补偿操作组合成补偿事务。当下一个子事务执行时，这个先前生成的补偿操作就会包含在新的补偿事务中。

对于每个子事务的顶层事务来说，上述过程都是重复的。至于顶层事务是否需要恢复，以及已经存储的补偿事务是否需要执行，这都要依照恢复协议的规则。

3.3 恢复策略

触发器驱动的方法利用触发器定义若干事件，描述补偿事务。除了数据的改变和事务操作事件，触发器还定义了系统事件(如系统重启)，用于启动恢复。用户定义的事件可能被触发，并且得到一个当前执行的子事务的标识符。这个标识符可能之后被用于为这个制定的子事务初始化恢复活动。

在该方法中，当前的 Web 服务中内建的 DBMS 事务管理功能不会受到任何影响，子事务以及补偿子事务将作为传统的事务而被执行，并且可以被 DBMS 作为一般的事务执行，且 DBMS 事务管理能够自动地处理所有传统事务的恢复。因为子事务以及补偿事务都是传统 ACID 事务，所以都能够被 DBMS 通过传统的基于 Undo/Redo 的方法自动执行[5]。

当子事务需要终止并进行补偿操作时，执行如下算法：

```
if(T is SC)
{
Ta=repeal;          //T 的执行状态(Ta)置为撤销
Tc=ϕ;
for each Ti do
{
    Tc=Tc+Ti;   //将所有的 Ti 组成 Tc
}
for(i=1 to N,step=1)
{
    if(stat Tc)&(Tc=AD)&(submit Tc)
//开始执行事务、设置补偿事务的状态为 AD(已执行)并且提交
//补偿事务
    {
        goto Succeed;
    }
    }
    Failed:
      Set Top_transaction Failed;
    Notify Administrator compensate Failed;
    //通知系统管理员此补偿事务失败并进行人工干预
      Return;
    Succeed:
      Affirm repeal;     //确认撤销已成功完成
      Return;
    }
```

该算法能够保证补偿事务的成功执行并作出正确的响应。在此算法中，将补偿事务看作一个整体，只有当所有的补偿操作执行成功之后才进行提交。例如，在补偿事务提交之前，事务执行的系统发生崩溃等异常事件，那么由补偿操作所产生的结果和状态的变更将由系统在其恢复过程进行相关参数和状态的回滚处理；需要说明的是，对补偿事务可以重复执行，提高了成功的可能性，如果在重试之后依然失败，那么将终止补偿事务的执行，并设置其再次执行的条件，以便条件满足时执行相应的操作；在补偿事务终止之后，将情况通知系统管理员，管理员会根据具体的情况进行正确的处理，最终保证补偿操作的完成。

3.4 基于触发器补偿方法的优势

基于触发器的补偿提供了重要的补偿策略。在一个自治的 Web Service 环境中，掌握事务逻辑的权威人士(如 Web Service 设计师)，可以负责制定统一、一致的补偿政策。由于补偿是数据库(或服务)中不可分割的一部分，因此 Web Service 的外部用户不应拥有制定补偿规则的权限。使用触发器执行补偿策略，是全面的、实时的，这就意味着通过触发器实施的补偿策略可以在任何时候发挥作用(只要触发器无故障)。

在补偿规则的约束下，该方法推动了半自动补偿事务的发展。根据事务在运行期间的状态，触发器会产生不同类型的补偿事务，和其他一致性约束一样，补偿的条件和规则在触发器中定义，如参照完整性。

从服务使用者的角度来看，补偿是透明的，如在基于 SQL 的系统中，SQL 用于更新数据库，并没有在补偿执行时用来指定额外信息，这使得服务看起来是自动恢复的。

由于补偿事务规范是从应用程序中抽象出来的，因此这种复杂性导致了用于服务组合(也可能是补偿)的应用软件越来越少。

基于触发器的补偿提供了一个模块化的方式来表达补偿策略。此外，使用基于触发器的补偿，可以对现有数据库在任何时候进行添加，它只影响数据库中使用补偿的那一部分；在补偿中使用触发器，触发器可以一直工作，这就不像资源，时而存在时而消亡；另外，除了补偿外，触发器仍然可以在数据库中作其他用途。

3.5 应用实例

现以采购流程为例说明 Web 服务中事务补偿机制的处理。实例包括制造企业 *A*、物流公司 *B*、信用评估机构 *C*、原材料 *M*、供应商 *D* 和运输公司 *E*。物流公司 *B* 从制造企业 *A*

处收到采购单，在提取采购单的信息后，流程根据 *A* 的账号到信用评估机构 *C* 评估其信用，评估通过后发送请求到合适的供应商进行定购。如果原材料 *M* 定购成功，物流公司将向运输公司预定运输车辆。物流公司等待 2 个定购请求的确认，收到确认后，物流公司 *B* 会通知制造企业 *A* 流程成功完成，并将预约确认号和原料运送细节发送给制造企业 *A*。一旦制造企业 *A* 被告知它请求的采购成功或失败后，它就可以做是否生产的准备了。该过程的控制流如图 2 所示。

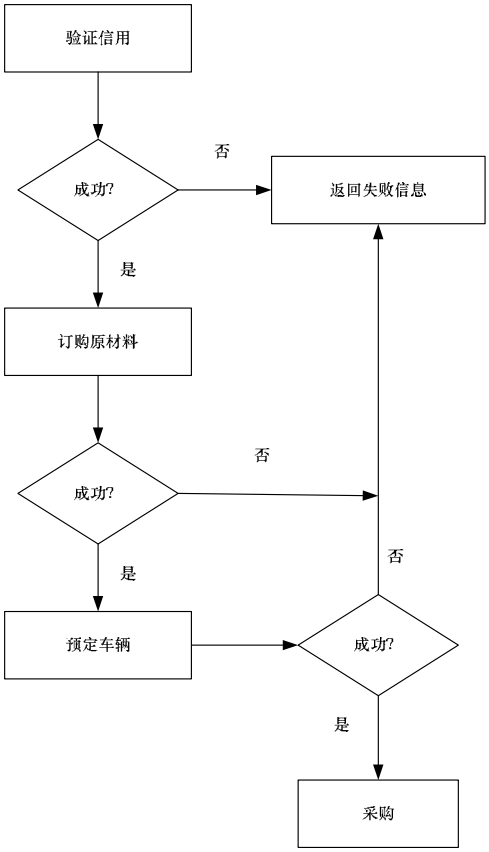

图 2　合成服务的控制流

和本例相关的代码片段如下：

```
< process>
< sequence>
< receive partner=“customerA”
portType=“purchaseOrderPT”
operation=“SendPurchadeOrder”
container= “PO”
</receive> //接收采购单
< invoke partner=“CreditBureauC”
portType=“CheckCreditPT”
operation= “CheckCredit”>
</invoke>//评估信用
< invoke partner=“supplierD”
portType=“materialPT”
operation=“RequestMaterial”
inputContainer=“materialRequest”
outputContainer=“materialInfo”>
< source linkName=“material-to-invoice”>
</invoke> //定购原材料
< invoke partner= “shippingPoviderE”
portType=“shippingPT”
operation= “RequestShipping”
inputContainer=“shippingRequest”
outputContainer=“shippingInfo”>
< source linkName=“ship-to-invoice”>
</invoke> //预定运输车辆
< reply partner= “CustomerA”
portType=“purchaseOrderPT”
operation=“SendPurchaseOrder”
container= “Invoice” />
</sequence> //返回采购信息
</process>
```

4　结束语

由于 Web Service 松耦合和自治的特性，因此需要相应的扩展事务模型与之相配合。为了在多方参与的事务中保持自治，Web Service 需要用到补偿事务，本文提出的基于触发器的 Web Service 补偿模式能够较好地处理事务撤销时的恢复问题，并可以半自动的方式提供服务，且触发器的逻辑在数据库(也就是对 Web 服务而言的底层)系统中实现事务的补偿，有利于集中利用补偿策略，具有较好的可扩展性。

参考文献

[1] 丁　柯, 金蓓弘, 冯玉琳. 事务工作流的建模和分析[J]. 计算机学报, 2003, 26(10): 1304-1311.

[2] 许　红, 王　茜. 基于 Web Service 的工作流管理系统的研究与实现[J]. 计算机应用与软件, 2006, 23(4): 44-47.

[3] 董云卫. 工作流管理系统的事务建模研究[D]. 西安: 西北大学, 2004.

[4] 朱锦泉, 苑森淼. 基于事务的工作流异常处理模型及方法[J]. 吉林大学学报: 工学版, 2003, 33(3): 46-50.

[5] 许　峰, 徐碧云, 黄　皓, 等. Web 服务事务中的补偿机制研究与实现[J]. 计算机科学, 2006, 33(7): 242-244.

编辑　陈　文