一个 Web 服务事务处理模型: 结构、算法和事务补偿

唐飞龙, 李明禄, 曹 健
(上海交通大学计算机科学与工程系, 上海 200030)

摘 要: 事务处理是 Web 服务能否用于电子商务的关键技术之一. Web 服务事务处理必须提供协调短事务和长事务的能力. 本文提出了一个能够同时处理原子事务和聚合事务的模型, 研究了其协调算法、状态转换及恢复机制. 聚合事务允许候选者独立提交, 并使用补偿事务来撤销已提交的子事务所带来的影响, 可以较好地满足 Web 服务环境下对长事务的要求, 为 Web 服务投入商用提供了有力支持.

关键词: Web 服务; 模型; 原子事务; 聚合事务; 补偿事务
中图分类号: TP393 文献标识码: A 文章编号: 0372-2112 (2003) 12A-2074-05

A Transaction Model for Web Services: Architecture, Algorithms and Transaction Compensation

TANG Fei-long, LI Ming-lu, CAO Jian
(*Department of Computer Science and Engineering, Shanghai Jiaotong University, Shanghai* 200030, *China*)

Abstract: Transaction is one of the key technologies whether Web Services can be applied to e-business applications. Transaction technologies for Web Services have to provide the abilities to coordinate short-lived and long-lived transactions. This paper proposes a Web Services transaction model, which can simultaneously handle Atomic transaction and Cohesion transaction, and researches corresponding coordination algorithms, state conversion and recovery mechanisms. Cohesion transaction allows candidates to commit independently, and undoes committed sub-transactions through the compensation transactions so that it can satisfy the requirements for long-lived transaction in Web Services environment, and provide powerful support for commercial web services.

Key words: Web services; model; atomic transaction; cohesion transaction; compensation transaction

1 引言

Web 服务[1]正在成为基于 Internet 的新一代计算平台, 它是一种以标准的工业技术来描述(WSDL)、发布与发现(UDDI)以及通讯(SOAP)的自包含、自描述、松散耦合的模块化软构件. 由于 Web 服务平台的不可靠性[2], 提供事务处理能力是其能否投入商业应用的关键技术之一.

事务是容错系统故障, 提高系统可靠性的有效手段. 事务可以对应用编程人员隐藏复杂的协调过程、异常处理和恢复细节, 减轻应用编程人员的负担, 使他们不必关心复杂的底层活动. 传统的分布式事务具有 ACID 性质: (1) 原子性(Atomicity)-事务的所有参加者要么都提交, 要么都撤销; (2) 一致性(Consistency)-事务不会破坏系统的恒定, 事务执行的结果必须使系统从一个一致性状态变化到另一个一致性状态; (3) 独立性(Isolation)- 并发执行的各个事务间不会相互干扰; (4) 持久性(Durability)-事务一旦提交, 它对系统的改变就是永久的, 其他任何操作或故障都不会对其产生影响. 为了实现严格的 ACID 性质, 事务必须是短生命期的; 系统必须是紧耦合的; 协调者必须对参加者拥有完全的控制权[3]. 因此, 严格的 ACID 事务不适用于 Web 服务环境, 因为 Web 服务系统具有以下特性: (1) 长事务: 由于商务处理, 网络延迟和用户的交互, 一个 Web 服务事务处理过程往往会持续较长的时间, 使得锁定资源的策略不再适用. (2) 自治性: Web 服务提供者拥有对服务的控制权, 其他的应用可能无法锁定它需要的资源. (3) 松耦合: Web 服务之间是松散耦合的. (4) 更多潜在的故障: 不仅进程和机器可能崩溃, 广域网络也比传统分布式系统的内部网更加不可靠. (5) 一次事务过程可能跨越多个组织.

关于分布式事务处理, 已经出现了不少标准和模型, 广泛应用的有 X/Open 的分布式事务处理参考模型(Distributed transaction processing, DTP)、OMG 基于 CORBA 的对象事务服务(Object transaction service, OTS)等. DTP 定义了三种角色(应用程序、事务管理器和资源管理器)和二种接口(TX 和 XA 接

收稿日期: 2003-09-30; 修回日期: 2003-12-29
基金项目: 国家 973 计划项目(No. 2002CB312002); 国家 863 项目(No. 2001AA415310); 上海市科委科技攻关重大项目(No. 03dz15027); 重点项目(No. 025115033)

口). 事务管理器使用标准的 2PC 协议协调各个资源管理器来完成全局事务. 但是, 它们都只能有效地应用在企业内部; 无法满足 Web 服务环境对长事务的要求.

WS-Coordination 和 WS-Transaction[4, 5] 是由 IBM、Microsoft、和 BEA 公司提出的 Web 服务事务处理模型, WS-Coordination 描述了可以容纳多种协调协议的 Web 服务事务处理框架, 定义了协调者的组成元素: 激活服务(Activation service, 主要用于创建事务上下文)、注册服务(Registration service, 参加者可以使用它注册协调协议以参加到事务中)和一组协调协议. WS-transaction 定义了原子事务和商务活动两类不同的事务, 但对后者没有给出具体的协调方案, 至今也没有实现产品. OASIS 的 Business transaction protocol (BTP)[6, 7]的主要工作是定义了一组在协调者与参加者之间传递的消息. 尽管 HP 已经发布了其 BTP 产品 Web services transactions (HP-WST)[8], 但其复杂的消息结构和工作流管理限制了其广泛应用. 此外, BTP 缺乏灵活的恢复机制, 没有提供补偿能力. 本文提出的基于代理技术的事务处理模型, 能够同时协调 Web 服务环境下的短事务和长事务, 具有自动可靠的故障恢复机制. 其实现简单, 编程方便. 通过使用补偿技术, 既不需要在整个事务过程中锁定资源, 也不会由于取消某个子事务而撤销整个事务, 提高了应用的并发度和系统效率.

2 Web 服务事务处理模型

Web 服务事务处理模块是构建在 Web 服务之上的一个中间层, 为应用提供事务能力(见图 1). 其核心部件代理的动作取决于请求的类型: 如果代理被请求以创建一个事务, 它发送 Coordination context(CC) 消息给远程 Web 服务的代理, 并在本地创建一个协调者 Coordinator; 如果代理收到一个 CC 消息, 它创建一个参加者 Participant(为原子事务)或候选者 Candidate(为聚合事务). 在模型中, 协调者和参加者(候选者)是为每一个事务动态产生的. 其优点一是可以减少对系统资源的消耗; 二是能够使用多线程技术.

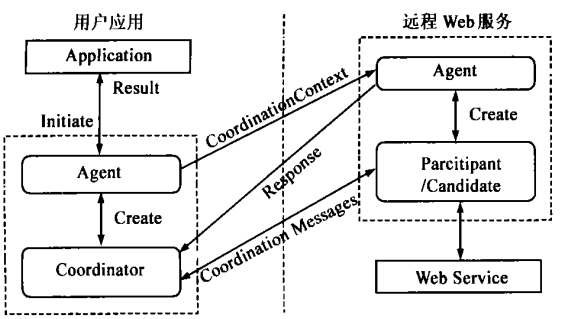

图 1　用于 Web 服务的事务处理模型

代理由下列部件组成: (1) 动态产生的协调者或参加者(候选者). 它们在事务生命期内交互协调信息以管理事务并且一直存活到本事务结束. (2) 补偿事务产生器. 用于为聚合事务的子事务自动产生相应的补偿操作并在子事务提交后将这些操作组合成一个补偿事务. (3) 时间服务. 检测超时. 根据设置的时间产生驱动信号, 使未提交的失败事务回滚或为聚合事务的已提交子事务启动相应的补偿事务. (4) 日志: 记录所有事务操作及其状态. (5) 接口: 包括为应用程序调用的应用接口和设置补偿事务策略的规则预定义接口.

3 Web 服务事务协调机制与算法

在 Web 服务环境下, 事务必须提供协调短时操作和长时间的商业活动的能力.

3.1 事务类型

根据应用需要, Web 服务环境下的事务分成两类.

定义 **1**　原子事务(Atomic transaction, AT) 是指所有参加者要么都提交要么都撤销的事务, 用于协调短生命期的操作.

原子事务具有 ACID 性质, 所有的参加者同步提交, 事务的中间状态是不可见的, 提交之前进行的操作都只是试探性的(即不持久并且其他活动看不到). 如果参与者都投票表示它们能够成功执行, 协调者就要求所有的参加者提交. 否则, 协调者要求所有的参加者异常终止. 提交使试探性操作变为持久操作并可以被其他事务看到. 异常终止使试探性操作看起来好像从没发生过一样. 原子事务在完成之前一直占用着数据资源(基于封锁)和物理资源(如内存、线程和连接).

定义 **2**　聚合事务(Cohesion transaction, CT) 是指允许一部分候选者提交而其余的候选者中止的事务, 用于协调长生命期的商业活动. 聚合事务的子事务 $T_1, T_2, \cdots, T_n$ 可以是原子事务或子聚合事务, 后者可以进一步划分, 直至最终结果为原子事务, 从而形成嵌套、层次性的结构. 其任一状态都是外部可见的. 聚合事务的子事务之间没有锁定机制, 其提交前的故障用回滚操作消除, 提交后的恢复采用补偿事务来实现. 聚合事务放松了独立性和原子性, 以适合 Web 服务中长事务和松耦合性质.

3.2 协调机制

在 Web 服务环境中, 事务处理集中在如何协调子事务而不是处理每个子事务的内部.

(1) 协调原子事务

事务创建. 用户方代理接受应用程序的请求, 向远程 Web 服务(参加者)的代理发送 CC 消息. 该消息包含了创建一个事务所需的必要信息, 包括事务类型 AT, 事务标识符, 协调者地址以及相应的超时参数等. 参加者根据 CC 中的信息以 Response 消息作出响应, 表明它是否愿意参加该事务活动.

事务准备. 收到协调者的 Prepare 消息后, 每一个参加者 P_i (i= 1, 2, …, N) 为其子任务的执行分配必要的资源. 如果成功, 它向协调者报告 Prepared 消息. 否则, 报告 Notprepared 消息.

事务提交. 收到 N 个 Prepared 消息后, 协调者送出 Commit 消息给所有参加者. 否则, 协调者发出 Abort 消息给每一个参加者, 命令它们放弃已分配的资源. 如果收到 Commit 消息, 参加者: ①记录提交信息在日志中; ②执行相应的子任务, 并将最终执行结果报告协调者.

提交失败的恢复. 协调者在间隔 T_2 内收集提交消息. 如果任一个参加者报告了 Failed 消息或者返回的消息数量少于 N, 协调者向用户报告事务失败并发送 Rollback 消息给所有参

加者,使它们都恢复到提交前的状态.如果收到 N 个 Committed 消息,事务正确完成.其状态转换如图 2(a)所示.

事务嵌套.在事务执行过程中,如果某参加者 P_i 本身含有子事务,它将递归地应用上述机制,形成嵌套的事务树.此时该 P_i 不仅是一个参加者,同时也作为其子事务 P_{ij} 的协调者.

(2) 协调聚合事务

代理接受应用的请求创建一个事务.类似于原子事务,不同的是事务类型为 CT.

候选者独立提交各自子事务.协调者发送 Enroll 消息给所有候选者.候选者分配资源后,直接提交事务.如果提交不成功,则自动回滚已执行的操作,并返回 Aborted 消息,此后它们从事务中被移除;如果提交成功,则该子事务的补偿事务也被同时生成,并返回包含执行结果的 Committed 消息.

用户确认已经成功提交的候选者.通过协调者,用户在时间T 内分别通过 Confirm 或 Cancel 消息来确认或取消它们.对失败的候选者,用户不必应答它们并继续发送 CC 消息给新的候选者,直到成功或尝试 M 次.

成功提交的候选者确认用户的选择.在时间 T 内,如果收到 Confirm 消息,候选者响应一个 Confirmed 消息,使子事务的结果不可再变.否则 Cancel 消息或者超时 T 后的时间触发(没有收到 Confirm/ Cancel 消息)都会启动补偿事务,将该 Web 服务恢复到提交之前的状态,当补偿事务完成后,候选者进入 Cancelled 状态,子事务提交所造成的影响被消除,如图 2(b)所示.图中实线矩形表示协调者和候选者的状态,虚线矩形仅表示候选者的状态.

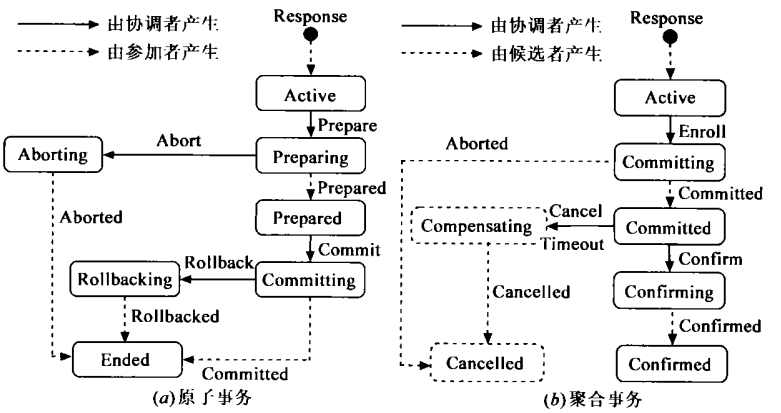

图 2 状态转换图

3.3 协调算法

(1) 原子事务协调算法

参数 t 表示等待时间. $n1$ 和 $n2$ 表示协调者收到的消息数量. N 指的是参加者的数量.

协调者算法 1:

```
ActionOfParent{
  step1: initiate a AT
  agent creates Coordinator;
  agent sends CC to all agents of Pi;
  wait for Response from agent of Pi;
  if timeout exit;
  step2: prepare for the transaction commit
  send Prepare to all Participants;
  while (t ≤ T1) and (n1< N)
    wait for and record incoming messages;
  step3: commit the transaction
  if (n1= N) and (all n1 messages are Prepared) {
    record commit in log;
    send Commit to all Participants;
    while (t ≤ T2) and (n2< N)
      wait for and record incoming messages;
    if (n2< N) or (not N messages are Committed)
    {  send Rollback to all Participants;
       exit after receiving all Rollbacked;  }
  } else {  send Abort to all Participants;
            exit after receiving all Aborted;}
}
```

参加者算法 2:

```
ActionOfChild{
  step1: join in the transaction
  agent creates Participant after receiving CC;
  agent sends Response to Coordinator;
  step2: allocate resources
  wait for Prepare from Coordinator;
  if timeout exit;
  success: = allocate resources;
  if( success)
    send Prepared to Coordinator;
  else exit;
  step3: commit sub-transaction
  while (t ≤ T3) & (message isn't Commit or Abort)
    wait for incoming message;
  if (message is Commit) {
    record commit in log;
    commit sub-transaction;
  //for nested sub-transaction, call ActionOfParent;
    send Committed to Coordinator;
  } else {  cancel allocation;
            exit;  }
}
```

(2) 聚合事务协调算法

在聚合事务中,一个很重要的参数就是协调者与候选者之间协商的有效时间 T.在时间 T 内,协调者能够根据用户需要确认或取消已成功提交的候选者.所以,一个聚合事务可能有多个正确的结果.

协调者算法 3:

```
ActionOfSuperior{
  step1: initiate a CT
  agent creates Coordinator;
  while (transaction doesn't complete) {
    agent sends CC to agents of candidates;
    wait for Response from agent;
  step2: enroll candidate
    send Enroll to Candidate;
  step3: confirm/ cancel candidate
    while( t ≤ T) {
      wait and record incoming messages;
      if (message is Committed)
        if ( user selects some candiates) {
          send Confirm to Candidates;
          wait for Confirmed messages;
        } else {
          send Cancel to Candidates;
          wait for Cancelled messages;
        }
  }
}}
```

候选者算法 4:

```
ActionOfInferior{
  step1: join in the transaction
  agent creates Candidate;
  agent sends Response to Coordinator;
  step2: commit sub-transaction
  wait for Enroll from Coordinator;
  if timeout exit;
  allocate resources;
  commit and generate compensation transaction;
  //for nested subtransaction, call ActionOfSuperior
  step3: confirm/ compensate sub-transaction
  if( commit successfully) {
    send Committed to Coordinator;
    while( t ≤ T) {
      wait for incoming message;
      if (message is Cancel) {
        execute compensation transaction;
      } else if (message is Confirm) {
            send Confirmed;
            cancel allocation; }
  }}}
```

4　补偿事务

定义 3　补偿事务是指用来撤销已提交的子事务所产生影响的事务.

在聚合事务中, 每一个候选者可以独立地提交其相应任务, 它们都有一个相关联的补偿事务. 如果用户取消某个已提交的子事务, 其相应的补偿事务将被调用, 以撤销该子事务的提交所造成的变化, 使系统返回到一致状态.

补偿事务使取消已提交的事务成为可能, 它可以在不放弃其他子事务的前提下保证数据的完整性, 提高了系统的效率. 如图 3 中, 每一个子事务 T_i 提交时都会产生一个相应的补偿事务 C_i; 子协调者 T_1 在 T_{12} 被撤销后(通过补偿事务 C_{12})重新选择新的候选者 T_{13}.

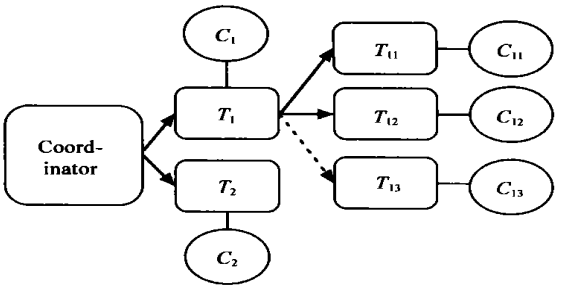

图 3　一个聚合事务 T 及其补偿事务

4.1　补偿操作的产生

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: (1) 数据修改操作(如对数据库的 insert, update 以及 delete 操作); (2) 事务协调消息(如 begin, commit 和 abort 等); (3) 用户自定义的事件; (4) 系统事件(如系统或机器重启).

根据补偿规则, 补偿事务产生器在事件驱动下产生补偿操作. 例如, 当某子事务删除一个数据元素时, 删除事件产生相应的补偿操作:

```
on Delete
do compensationOperation (' Insert', data)
```

根据这个规则, 子事务中的每一个 delete 操作将导致一次 insert 被删除的数据元素的补偿操作. 函数 compensationOperation 可以被任何嵌入在商务逻辑中用于补偿的功能模块所替代. 例如给某客户发一份电子邮件. 补偿操作是在子事务的执行过程中动态产生的, 假如子事务失败了, 则在该子事务执行过程中所产生的所有补偿操作也被放弃.

4.2　补偿事务的产生与执行

补偿事务的产生策略: (1) 基于应用逻辑, Web 服务提供者通过规则预定义接口设计补偿操作的产生规则; (2) 补偿事务产生器根据指定的规则, 在子事务执行过程中动态产生补偿操作; (3) 子事务提交时, Commit 事件驱动补偿事务产生器将所有补偿操作封装成一个补偿事务并将其存储在数据库中.

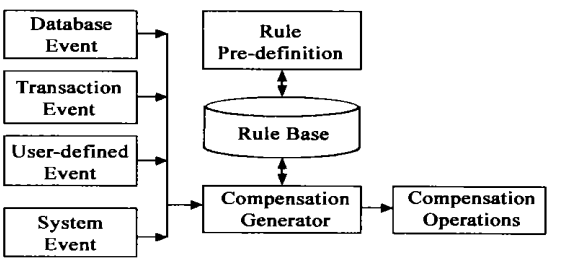

图 4　补偿事务产生过程

补偿事务的产生过程如图 4 所示, 其规则预定义接口主要用来设计补偿规则, 方法有:

setCompensationRule ( ) : 为一个 Web 服务设置补偿操作的规则.

getCompensationRule ( ) : 获取一个 Web 服务的补偿规则.

聚合事务通过 Cancel 消息或时间服务的超时触发来启动一个补偿事务.

5 实现

(1) 开发环境

使用 Java 作为主要的开发语言, SQL Server 2000 作为数据库服务器, 我们开发了一个原型系统, 验证了模型和算法的可行性. 所有的 Java 类使用 J2SDK 1.4.0._01 编译, 系统的每一台机器上都安装了代理. 协调者和参加者(候选者)都是新创建的 Java 对象, 它们交互协调信息以完成对事务活动的管理, 两者都存活到本事务的结束.

(2) 提供的应用接口:

应用接口定义了可供用户调用的方法, 它们用来启动, 停止, 提交事务等. 方法有:

beginTransaction(In txType, Out txHandle) : 启动一个新的事务.

prepareCommit(In txHandle, Out fail) : 分配资源, 为提交作准备.

rollback(In txHandle, Out fail) : 当事务尚未提交前, 回滚已执行的事务操作.

commit (In txHandle, Out fail) : 提交事务.

enroll(In txHandle, Out fail) : 要求聚合事务的子事务分配资源并提交事务.

startCompensationTransaction(In c_i) : 在聚合事务中, 启动子任务 t_i 对应的补偿事务 c_i.

getTransactionState(In txHandle) : 得到事务状态.

事务的启动与协调信息都是通过 SOAP 消息来传递的. 下面是一个创建原子事务的 CC 消息.

```
〈CoordinationContext
  myApp= "http://192.168.0.6/transaction" …〉
  〈myApp: Identifier〉http://192.168.0.6/
    transaction/1 〈/myApp: Identifier〉
  〈myApp: CoordinationType〉AT
  〈/myApp: CoordinationType〉
  〈myApp: CoordinatorAddress〉
    http://192.168.0.6/transaction
  〈/myApp: CoordinatorAddress〉
  〈myApp: Expires〉
    2003-08-11T10:00:00-12:00
  〈/myApp: Expires〉
〈/CoordinationContext〉
```

6 结论

事务处理是 Web 服务大规模投入商用的亟待解决但尚未解决的关键技术. 本文提出的Web服务事务处理模型同时具备协调短时操作和长时间的商业活动的能力, 通过提交前的回滚和提交后的补偿事务实现了良好的对用户透明的恢复机制. Web 服务的提供者可以灵活地设置补偿规则, 使得该模型可以较好地适应 Web 服务环境下各种各样的应用需求. 我们的后续工作包括: (1) 解决当聚合事务不可补偿时的恢复问题. (2) 将模型扩展成能满足商用要求的事务服务, 以减轻 Web 服务应用编程人员的负担.

参考文献:

[1] Michael N Huhns. Agents as Web services [J]. IEEE Internet Computing. 2002, 6(4): 93-95.

[2] T Mikalsen, S Tai, I Rouvellou. Transactional attitudes: Reliable composition of autonomous Web services [A]. International Conference on Dependable Systems and Networks [C]. USA: IEEE. 2002.

[3] K W Lee, H J Kim. Consistency preserving in transaction processing on the Web [A]. Web Information Systems Engineering [C]. USA: IEEE. June 2000. 190-195.

[4] F Cabrera, et al. Web Services Coordination (WS-Coordination) [S]. August, 2002. http://www.ibm.com/developerworks/library/ws-coor/.

[5] F Cabrera, et al. Web Services Transaction (WS-Transaction) [S]. August, 2002. http://www.ibm.com/developerworks/library/ws-transpec/.

[6] S Dalal, et al. Coordinating business transactions on the Web [J]. IEEE Internet Computing, 2003: 30-39.

[7] A Ceponkus, et al. Business Transaction Protocol V1.0 [S]. June, 2002. http://www.oasis-open.org/committees/download.php/1184/2002-06-03.BTP_cttee_spec_1.0.pdf.

[8] Hewlett-Packard. HP Web Service Transactions 1.0 Tech Preview [DB/OL]. Hewlett-Packard, Colorado USA. http://www.hpmiddleware.com/downloads/pdf/wst_specsheet.pdf, 2002.

作者简介:

唐飞龙 男, 1965 年生于安徽省, 博士生, 研究方向: 网格计算、Web Services、服务计算和计算机网络.

李明禄 男, 1965 年生于重庆, 博士、教授、博导, 上海交通大学计算机系副主任、网格计算中心主任、Web Services 研究中心主任, 上海市科委信息技术领域计算机软硬件主题专家, 中国计算机学会软件工程专业委员会委员, 研究方向: 网格计算与 Web Services、多媒体计算、空间信息处理.