语义锁模式及死锁解除机制在 Web 服务中的应用①

项　慨　(湖北经济学院 计算机学院 湖北 武汉 430205)

摘　要: 提出了一种 Web 服务环境中的语义锁模式，该锁模式通过事务申请的资源数量，动态控制加锁粒度，并且根据语义锁中的语义信息，提出了能够最大限度地减少企业经济损失的死锁解除机制，该模式增加了 Web 服务中事务间的并发度，降低了死锁发生的概率，从而有效提高了企业的经济效益。

关键词: 语义锁；死锁解除；锁服务；Web 服务

Semantic Lock Model and Application of Deadlock Lifting Mechanism to Web Services

XIANG Kai (School of Computing, Hubei University of Economics, Wuhan 430205, China)

Abstract: The paper proposes a semantic lock mode in the Web services environment. The lock mode implements the dynamic control of locking granularity, based on the amount of resources, to apply by current transactions and also explores the deadlock lifting mechanism to minimize loss of enterprise spending on the basis of the semantic information in the semantic lock. The mode increases transaction concurrency amony transactions in Web services and reduces the probability of the occurrence of the deadlock, which effectively improves the economic efficiency of enterprises.

Keywords: semantic lock; deadlock to lift; locks services; web services

1　前言

Web 服务是一种基于标准的应用集成方式，它可以将运行在 Internet 上的分布式应用集成在一起。Web 服务具有动态特性和互操作性，它把一切都看作服务，这种服务可以通过消息传递的方式被动态地发现和组织。目前，可以通过工作流的方式将多个 Web 服务集成为一个业务流程，这种方式主要的技术有基于 XML 的业务流程语言 BPEL4WS[1]和 WSFL[2]等。并且，在对 Web 服务进行集成时，还需要通过一定的机制保证其一致性和可靠性，因此 IBM 和 Microsoft 先后发布了 Web 服务协作(WS_Coordination)规范以及 Web 服务事务（WS_Transaction）规范，它们负责协调整个业务流程中多个 Web 服务的协调与合作。但是，在 Web 服务环境中多个事务并发执行时，会出现与传统的事务环境，如操作系统，数据库等相同的死锁问题，但是，WS-Coordination 和 WS-Transaction 规范中并没有提供此问题的解决机制，这样会造成基于 Web 服务的电子商务企业的经济损失。

基于以上原因，本文提出了一种适合于 Web 服务环境的语义锁模式，该锁模式能够极大地增加 Web 服务环境中事务间的并发度，提高企业的经济效益。并且，我还根据语义锁中的语义信息，提出了能够最大限度地减少企业经济损失的死锁解除方法。

2　基于数量的语义锁模式

由于 Web 服务是电子商务最重要的组成部分，因此在对 Web 服务中的元素加锁时，需要将 Web 服务提供者，即企业的经济效益摆在首位，增加 Web 服务环境中事务之间的并发性，以及在死锁发生时，能够以一种最大限度降低企业经济损失的方式解除死锁。

Web 中的加锁对象通常是按等级存储的，若通过小粒度加锁某个对象，它的子对象也会被加锁，导致并发度减少；若加锁粒度太大，又会增大加锁操作的系统开销。虽然传统的意向锁中运用了多粒度加锁的

① 基金项目:湖北省教育厅科技处研究项目(B20081905)
收稿时间:2009-12-23;收到修改稿时间:2010-01-19

方法，但是由于缺乏语义信息，它仍旧无法根据事务请求状况，准确分配锁资源。本文根据 Web 服务中的操作与增量操作的相似性，提出了一种新的锁模式：语义锁。

定义 1. 语义锁由读量锁 S，增量锁 INC(U)，减量锁 DEC(V)三部分组成，其中读量锁 S 与以往的读操作 R 相同，INC(U)表示将资源数增加 U，DEC(V)表示将资源数减少 V，语义锁主要负责对 Web 服务中事务要调用的各种类型的资源进行加锁、解锁[3]。

在传统的排它锁中，当一个事务对资源加上排它锁后，其它的事务就不能再对该资源进行修改，这样，事务之间的并发度会大大降低，语义锁将事务请求时资源申请的数量引用到锁模式中，使得加锁的粒度不局限于资源的种类上，而是资源的个体上,从而与传统的排它锁相比，在一定增加了事务之间的并发度。语义锁与传统的增量锁相比较可以看出，在增量锁中，资源增减的数目对于是否能够得到锁无关紧要，但是，在语义锁中，对资源增减的数目关系到是否能够加锁，而且发生死锁后，语义锁中的语义信息能够帮助我们最优化地解除死锁。

2.1 资源的语义锁表

为了更好的协调语义锁的申请、添加和解除处理，我们将数量语义的概念引入到资源锁表中，在存储进程标识，进程状态的同时，加入了进程锁住的资源个数、该资源当前最强锁模式（组模式）以及资源个体价值等信息，并将资源上锁的类型扩展到语义锁范围。

我在资源的语义锁表中为每个资源节点分配一个事务等待列表。当有事务对某个资源提出加锁申请时，就在对应的等待列表中添加一个新条目。图 1 描述了一个使用资源语义锁表的样例[4]。

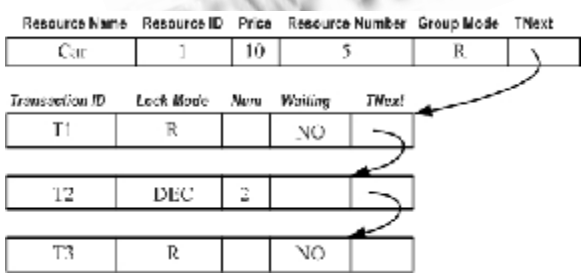

图 1 资源语义锁表

2.2 语义锁服务

由于 WS-Coordination 和 WS-Transaction 框架中定义的协调器只包含了激活服务(Activation Service)和注册服务(Registration Service)，这些服务仅能对 Web 服务的业务流程中单事务内部的多个 Web 服务进行协调，而无法对多事务并发执行时产生的问题进行控制和处理，因此，本文在其原有框架中加入一个语义锁服务(Lock Service, 如图 2)，它以 Web 服务规范为依据，在与其它 Web 服务之间传递消息时遵循简单对象访问协议(SOAP)，该服务负责对 Web 服务进行加锁，解锁以及死锁的检测[5]。

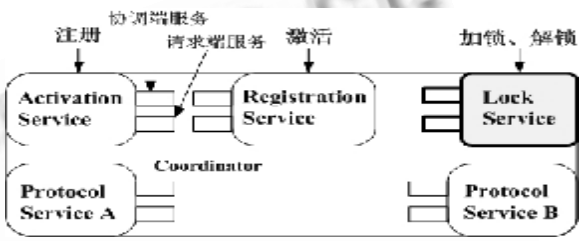

图 2 加入 Lock Service 后的 WS-Coordination 框架

2.3 语义锁的分配机制

2.3.1 语义锁的加锁处理

当事务对系统资源提出申请时，系统会根据该事务申请的资源数量，以及当前资源分配情况，合理安排资源语义锁表[6]：

① 当资源锁表中的组模式类型以及事务申请的锁模式均为读量锁时，将该事务条目表头中的等待状态置为 NO；

② 当资源锁表中的组模式类型为减量锁或增量锁并且事务申请的锁模式为减量锁时，比较此事务申请的资源个数与资源锁表中该资源的数目：

a.若申请的资源个数不大于该资源的数目，将该事务的等待状态置为 NO，并将资源锁表中该资源的数目减去事务所申请的资源个数；

b.否则，将该事务的等待状态置为 YES。

③ 当资源锁表中的组模式类型为减量锁或增量锁，且事务申请的锁模式为增量锁时，将该事务的等待状态置为 NO。为防止进程意外中止导致的非一致性问题，必须提交该事务后，方可将资源锁表中该资源的数目加上事务增加的资源个数。

2.3.2 语义锁的解锁处理

① 当等待解除的事务锁模式类型为读量锁时，扫描等待列表，辨别该表中是否有其他事务在读取该资源：

a．若有，则组模式不变，直接删除待解除锁的事务条目；

b．否则，判断等待列表中的第一个等待状态为 YES 的事务能否得到该资源上的锁：

若能，则修改资源的组模式，并将该事务的等待状态改为 NO；

否则，继续扫描等待列表；

②当等待解除的事务锁模式为减量锁而组模式类型为减量锁或增量锁时，扫描等待列表，辨别等待列表中是否有其他非等待事务的锁类型为增量锁或减量锁：

a．若有，则组模式不变，直接删除待解除锁的事务条目；

b．否则，判断等待列表中的第一个等待状态为 YES 的事务能否得到该资源上的锁：

若能，则修改资源的组模式，并将该事务的等待状态改为 NO；

否则，继续扫描等待列表；

③当等待解除的事务锁模式为增量锁而组模式的类型为减量锁或增量锁时，扫描等待列表，辨别等待列表中是否有其他非等待事务的锁类型为增量锁或减量锁：

a．若有，则组模式不变，并删除待解除锁的事务条目；更新资源锁表中的该资源数目；最后再次扫描等待列表，辨别等待列表中是否有其他等待事务的锁类型为减量锁，如果有，则判断该事务请求资源数是否不大于更新后的资源锁表中该资源的数目，

若是，将该事务的等待状态改为 NO。

否则，该事务的等待状态不变；

b.否则，判断等待列表中的第一个等待状态为 YES 的事务能否得到该资源上的锁，

若能，则修改资源的组模式，并将该事务的等待状态改为 NO；

否则，继续扫描等待列表。

2.4 语义锁的兼容规则

为了保证事务执行的正确性和事务之间的可串行化，语义锁在运用过程中必须满足一定的规则。为了便于表达，我们定义：sli（X）表示 Ti 向资源 X 申请一个 S 锁，ili（X，U）表示 Ti 向资源 X 申请一个将 X 的值增加 U 的增加锁，dli（X，V）表示 Ti 向资源 X 申请一个将 X 的值减少 V 的减少锁，ui（X）表示 Ti 释放 X 上的所有锁[6]。

规则 1：事务的一致性问题：

当需要对 Web 服务内的资源 X 进行 S 操作时，必须申请该资源上的 S 锁；同理，在 dli（X，U）操作前，必须先申请 DEC（U）锁；在 ili（X，V）操作前，必须先申请 INC（V）锁。对于任何一个事务 Ti 内部：

① 在读操作 Ri（X）之前，必须有 sli（X）操作，且两个操作之间不存在 ui（X）；

② 在增加操作 INC（X，U）之前，必须有 ili（X，U）操作，且两个操作之间不存在 ui（X）；

③ 在减少操作 DEC（X，V）之前，必须有 dli（X，V）操作，且两个操作之间不存在 ui（X）；

④ 在所有的加锁操作之后，必须有一个针对加锁操作中元素的解锁操作。

规则 2：根据事务的两段锁(Two-phase locking)协议，在任一事务内部，所有的加锁操作必须在解锁操作之前。即事务 Ti 中，所有的加锁操作 sli(X)、ili(X，U)、dli(X，V)必须在 ui(X)之前。

规则 3：事务之间锁的冲突关系：

①由于读操作在增量操作之前还是之后，与传统的增量锁模式相同，在语义锁模式中，S 锁与 INC（U）锁（DEC（V）锁）之间是冲突的，共享锁相互之间是兼容的。即，如果在一个事务调度中出现 sli（X），且 sli（X）后没有 ui（X）操作，那么则不能再出现 dlj（X，U）和 ilj（X，V）操作。反之，若在调度中出现 dlj（X，U）和 ilj（X，V）操作，并且此后的操作中未出现 uj（X）操作，那么就不能再出现 sli（X）操作。

②无论 U 和 V 为何值，INC（U）锁模式与 INC（U）和 DEC（V）锁模式之间是不冲突的。即无论 ilj（X，U）操作和 dlj（X，V）操作的后面是否有 uj（X），ili（X，U）均可出现在 ilj（X，U）和 dlj（X，V）操作之后。

③如果资源上没有锁，并且事务将减少资源个数 V2 时，若 V2 小于或等于 Web 服务中的资源个数 N，则可以得到 DEC（V2）锁，即当 V2≦N 时，R1=Y，否则 R1=N。

④当事务 T1 已经持有资源上的增量锁 INC（U1）时，若事务 T2 要申请资源上的 DEC（V2）锁，那么为了保证 DEC（V2）操作的成功，当 V2≦N 时，R2=Y，否则 R2=N。即只有当 V2 不大于 N 时，才能在 ili（X，

U1）之后执行 dlj（X，V2）操作。

⑤当事务 T2 已经持有资源上的 DEC（V1）锁时，若事务 T2 要申请资源上的 DEC（V2）锁，那么为了保证 DEC（V2）操作的成功，当 V1+V2≦N 时，R3=Y，否则 R3=N。即只有当要申请的资源个数 V2 与正在申请的资源个数 V1 之和不大于资源个数 N 时，才能在 dli（X，V1）之后执行 dlj（X，V2）操作。语义锁的兼容矩阵如表 1。

表 1　语义锁兼容矩阵

锁请求	NL	S	INC（U1）	DEC（V1）
S	Y	Y	N	N
INC(U2)	Y	N	Y	Y
DEC(V2)	R1	N	R2	R3

3　语义锁的死锁解除机制

通过 Web 服务中的语义锁机制，多个事务可以同时申请 Web 服务中的资源，但是当资源数不能满足多个事务的资源请求时，可能会在事务之间形成等待环路，即死锁。由于 Web 服务处于一个全分布、松耦合的环境，因此可能产生局限于一个节点内部的集中式死锁或跨越多个节点的分布式死锁。本文结合锁中的语义信息和传统的锁处理方法，提出了更合理的死锁解除机制。

在语义锁模式下，当一个事务对 Web 服务中的资源进行增减时，若资源总数允许，其他事务可同时对该资源进行增减，从而有效增加了 Web 服务中事务处理的并发度；并且，仅当 Web 服务中的资源数不能满足某些事务的需求时，才可能产生死锁，因此与传统的锁模式相比，死锁发生的频率大大减少。

由于 Web 服务是动态电子商务的基础，在 Web 服务环境中解除死锁时，我们需要将企业的经济效益放在首位，即在事务回滚时，根据企业经济利益智能选择回滚事务，而非传统方法那样随机选择或全部回滚。当某个节点的 Lock Service 检测到死锁后，就向死锁环中的各事务发送死锁消息，接收到消息的事务将其加锁信息返回给检测到死锁的节点的 Lock Service，然后由该 Lock Service 判断哪个或哪些事务是最佳的回滚事务，从而使企业能最大限度地减少经济损失。

Lock Service 中的返回信息主要包括：事务的 ID，事务的加锁状况，以及申请的资源个数等。由于死锁仅与事务申请的 DEC 锁相关，因此在 Lock Service 中仅需要考虑 DEC 锁之间的关系。本文中，我通过求最大值的方法来智能选择回滚事务[4]。

假设死锁中涉及到的事务有 T1，T2，…Tk，涉及到的资源有 R1，R2，…Rm，假设每种资源的个数分别为 N1，N2，…Nm，而对应于每种资源的权值分别为 W1，W2，…Wm。另外 T1，T2，…Tk 的资源请求分别为：

T1：DEC（R1，a11）；DEC（R2，a12）；…DEC（Rm，a1m）；

T2：DEC（R1，a21）；DEC（R2，a22）；…DEC（Rm，a2m）；

…

Tk：DEC（R1，ak1）；DEC（R2，ak2）；…DEC（Rm，akm）；

其中，aij 为事务 Ti 申请资源 Rj 的个数。

设 Ti 的业务量为 Bi，则可得：

$B1=a11W1+a12W2+\cdots+a1mWm$；

$B2=a21W1+a22W2+\cdots+a2mWm$；

…

$Bk=ak1W1++ak2W2+\cdots+akmWm$；

即选择回滚事务问题转换成了求 $B1x1+B2x2+\cdots+Bkxk$ 的最大值，它需要满足的条件如下：

$a11x1+a21x2+\cdots+ak1xk\leqq N1$；

$a12x1+a22x2+\cdots+ak2x\ k\leqq N2$；

…

$a1mx1++a2mx2+\cdots+akmxk\leqq Nm$；

其中 x1，x2，…xm 为 0 或 1。当 xi=0 时，表示将 Ti 回滚，当 xi=1 时，表示让 Ti 完成。

上式求解后可以获得业务量的最大值，即对应的各个 xi 值，从而智能的选择出了该回滚的事务项，并使企业能够获得最大的经济效益。

4　结束语

本文提出了一种适用于 Web 服务环境中的语义锁模式，该锁模式可以将事务的加锁粒度分配到资源的个体级别上，从而有效增加事务间的并发

（下转第 167 页）

(上接第 207 页)

度。同时，根据语义锁中的语义信息，本文还提出了能够最大限度地减少企业经济损失的死锁解除机制，智能选择回滚事务，可以在 Microsoft .NET 平台上模拟多 Web 服务集成的业务流程，并且设计出基于语义锁的事务处理过程以及死锁解除算法来验证该方法的优越性，下一步我将编程实现这种方法，并且进一步分析实验结果在 Web 服务中的作用。

参考文献

1 BPEL4WS.[2009-10-24].http://www-128.ibm.com/ developerworks/library/ws-bpel/.

2 WSFL.[2009-10-24].http://www-306.ibm.com/ software/solutions/webservices/pdf/WSFL.pdf.

3 Tong L, Carla SE, Alvin RL, Danie JS. Pulse: A Dynamic Deadlock Detection Mechanism Using Speculative Execution. Proc. of the 2005 USENIX Annual Technical Conference Anaheim, California, April 10–15, 2005.

4 赵明霞. Web 服务集成的协调框架中的死锁检测机制研究[硕士学位论文].武汉:武汉大学, 2004.

5 Antoine G, Mathieu B. A resource-control model based on deadlock avoidance.[2009-9-20]. http://cdc.ioc.ee/appsem04/webproc/indust/galland.pdf, 2004.

6 Jaehwan L, Vincent JM. A Novel Deadlock Avoidance Algorithm and Its Hardware Implementation. [2009-9-20].http://codesign.ece.gatech.edu/publications/jaehwan/paper/codes_2004.pdf, 2004.