第9章 事务处理中间件

李会格 博士/讲师 京江学院

E-mail: 1034434100@qq.com

前言

- 分布式事务需处理并发进程，涉及到操作系统、文件系统、编程语言、数据通信、数据库系统、系统管理及应用软件等领域，是一个相对复杂的任务。

- 事务处理中间件（Transaction Processing Middleware，简称TPM）是在分布、异构环境下保证事务完整性和数据完整性的一种环境平台，它提供了一种专门针对联机事务处理系统而设计的事务控制机制。

- 本章将从分布式事务基础、EJB事务体系、JTA事务处理等方面入手介绍事务处理中间件的概念和用法。

大纲

- 事务处理基础
  - ✓ 事务的概念
  - ✓ JDBC的事务
- 分布式事务处理
  - ✓ 分布式事务
  - ✓ 事务处理中间件
  - ✓ 两阶段提交2PC
  - ✓ 2PC的应用
- EJB事务体系结构
  - ✓ 容器管理的事务CMT
  - ✓ Bean管理的事务BMT
- JTA事务处理
- 小结

事务的概念

- 事务（Transaction）是恢复和并发控制的基本单位。
- 事务是保证数据库从一个一致性的状态永久地变成另外一个一致性状态的根本，是计算机应用中不可或缺的组件模型。
- 基于事务的程序设计保证了用户操作的原子性（Atomicity）、一致性（Consistency）、隔离性（Isolation）和持久性（Durability）：

事务的概念

- 原子性（Atomicity）
  - ✓ 一个事务是一个不可分割的工作单位，事务中包括的诸操作要么都做，要么都不做
- 一致性（Consistency）
  - ✓ 事务必须是使数据库从一个一致性状态变到另一个一致性状态。一致性与原子性是密切相关的
- 隔离性（Isolation）
  - ✓ 一个事务的执行不能被其他事务干扰。即一个事务内部的操作及使用的数据对并发的其他事务是隔离的，并发执行的各个事务之间不能互相干扰
- 持久性（Durability）
  - ✓ 持久性也称永久性（Permanence），指一个事务一旦提交，它对数据库中数据的改变就应该是永久性的。接下来的其他操作或故障不应该对其有任何影响。

事务的概念

- 银行卡转账例子：用户A需要将账户中的500元人民币转移到用户 B 的账户中。该操作可分为两步：
  - ✓ 将 A 账户中的金额减少500元；
  - ✓ 将 B 账户中的金额增加500。
- 这两个操作必须保证 ACID 的事务属性：即要么全部成功，要么全部失败；
  - ✓ 假如没有事务保障，用户的账号金额可能发生问题：假如第一步操作成功而第二步失败，那么用户 A 账户中的金额将就减少500元，而用户 B 的账号却没有任何增加；
  - ✓ 如果第一步出错而第二步成功，那么用户 A 的账户金额不变而用户B的账号将凭空增加500元。上述任何一种错误都将导致数据不一致问题。因此，事务缺失对于一个稳定的生产系统是不可接受的。

事务的概念

- 具体到数据库或应用中，事务是访问并可能更新数据库中各种数据项的一个程序执行单元（unit）。
  - ✓ 事务通常由高级数据库操纵语言或编程语言（如SQL，C++或Java）书写的用户程序的执行所引起，并用形如begin transaction和end transaction语句（或函数调用）来界定。
  - ✓ 事务由事务开始（begin transaction）和事务结束（end transaction）之间执行的全体操作组成。
  - ✓ 只有当事务中的所有操作都正常完成了，整个事务才能被提交到数据库。如果有一项操作没有完成，就必须撤销整个事务。

大纲

- 事务处理基础
  - ✓ 事务的概念
  - ✓ JDBC的事务
- 分布式事务处理
  - ✓ 分布式事务
  - ✓ 事务处理中间件
  - ✓ 两阶段提交2PC
  - ✓ 2PC的应用
- EJB事务体系结构
  - ✓ 容器管理的事务CMT
  - ✓ Bean管理的事务BMT
- JTA事务处理
- 小结

JDBC的事务

- JDBC(Java Data Base Connectivity)是Java与数据库的接口规范，其包含了大部份基本数据操作功能，也包括了事务处理的功能。JDBC的事务是基于连接(connection)进行管理的。

- 在JDBC中是通过Connection对象进行事务的管理，默认是自动提交事务。

JDBC的事务

- Connection 接口（java.sql.Connection）提供了两种事务模式：自动提交和手工提交。事务操作缺省是自动提交。一条对数据库的更新表达式代表一项事务操作，操作成功后，系统将自动调用commit() 来提交，否则将调用rollback() 来回滚。

```
try {
    conn = DriverManager.getConnection(
          "jdbc:oracle:thin:@host:1521:SID","username","userpwd");
    conn.setAutoCommit(false); //禁止自动提交，设置回滚点
    stmt = conn.createStatement();
    stmt.executeUpdate("alter table ..."); //数据库更新操作 1
    stmt.executeUpdate("insert into table ..."); //数据库更新操作 2
    conn.commit(); //事务提交
} catch(Exception e){
    e.printStackTrace();
    try {
        conn.rollback(); //操作不成功则回滚
    } catch(Exception e){
        e.printStackTrace();
    }
}
```

大纲

- 事务处理基础
  - ✓ 事务的概念
  - ✓ JDBC的事务
- 分布式事务处理
  - ✓ 分布式事务
  - ✓ 事务处理中间件
  - ✓ 两阶段提交2PC
  - ✓ 2PC的应用
- EJB事务体系结构
  - ✓ 容器管理的事务CMT
  - ✓ Bean管理的事务BMT
- JTA事务处理
- 小结

分布式事务

- 分布式事务是指事务的参与者、支持事务的服务器、资源服务器以及事务管理器分别位于不同的分布式系统的不同节点之上。

- 分布式事务需处理并发进程，涉及到操作系统、文件系统、编程语言、数据通信、数据库系统、系统管理及应用软件等领域，是一个相对复杂的任务。

- 分布式事务处理（Transaction Processing，简称TP）系统旨在协助在分布式环境中跨异类的事务识别资源的事务。

分布式事务

- 例子
- 某一电商网站的数据初始是由单机支撑，现在对该网站进行拆解，分离出了订单中心、用户中心、库存中心等，对于订单中心、用户中心以及库存中心分别有专门的数据库存储订单信息、用户信息和库存信息。
  - ✓ 此时如果要同时对订单和库存进行操作，那么就会涉及到订单数据库和库存数据库，那么为了保证数据的一致性，就需要用到分布式事务处理。
  - ✓ 换个角度看，一个操作会由许多不同的小操作组成，而这些小的操作分布在不同的服务器上且属于不同的应用；
  - ✓ 而分布式事务就是要保证这些小操作要么全部成功，要么全部失败。
  - ✓ 本质上来说，分布式事务就是为了保证不同数据库的数据一致性。

大纲

- 事务处理基础
  - ✓ 事务的概念
  - ✓ JDBC的事务
- 分布式事务处理
  - ✓ 分布式事务
  - ✓ 事务处理中间件
  - ✓ 两阶段提交2PC
  - ✓ 2PC的应用
- EJB事务体系结构
  - ✓ 容器管理的事务CMT
  - ✓ Bean管理的事务BMT
- JTA事务处理
- 小结

事务处理中间件

- 事务处理中间件（Transaction Processing Middleware，简称TPM）是在分布、异构环境下保证事务完整性和数据完整性的一种环境平台，它提供了一种专门针对联机事务处理系统而设计的事务控制机制。
- TPM在联机事务处理过程中负责处理分布式事务的完整性、并发控制、负载均衡以及出错恢复等。
- TPM能够把自身的事务管理功能和数据库已有的事务管理能力有机结合在一起，实现对分布式事务处理的全局管理。

事务处理中间件

- 一般地，联机事务处理系统常需要处理大量的分布式事务。这个过程涉及到多个数据库，并且这些数据库可能是异构的。
- 在分布式 TPM系统的支持下，应用程序可以将不同的活动合并为一个事务性单元，这些活动包括从“消息队列”检索消息、将消息存储在Microsoft SQL Server数据库中等等。
- 由于分布式事务跨多个数据库资源，涉及到多个结点的数据的更新，故强制 ACID 属性维护所有资源上的数据一致性是很重要的。任何一个节点的失效或者结点间通信的失效都有可能导致分布式事务的失败。

大纲

- 事务处理基础
  - ✓ 事务的概念
  - ✓ JDBC的事务
- 分布式事务处理
  - ✓ 分布式事务
  - ✓ 事务处理中间件
  - ✓ 两阶段提交2PC
  - ✓ 2PC的应用
- EJB事务体系结构
  - ✓ 容器管理的事务CMT
  - ✓ Bean管理的事务BMT
- JTA事务处理
- 小结

两阶段提交2PC

- 在分布式系统中，事务往往包含有多个参与者的活动，为了保证事务的完整性，分布式事务通常采用两阶段提交协议（Two-Phase Commitment Protocol，简称2PC）来提交。两阶段提交是实现分布式事务的关键。

两阶段提交2PC

- 基本过程
- 2PC中存在两种类型的节点：协调节点和数据节点。
- 协调节点（或称协调者）负责充当分布式事务协调器的角色。事务协调器负责整个事务，并使之与网络中的其他事务管理器（或称参与者）协同工作，管理多个数据节点在事务操作中数据的一致性问题。协调者可以作为事务的发起者，也可作为事务的一个参与者。

两阶段提交2PC

- 基本过程
- 在分布式事务中，一个事务通常涉及到多个参与者。参与者也可以看作是数据在多个节点的备份。2PC通常分为两个阶段进行：提交请求阶段（Commit Request Phase）和提交阶段（Commit Phase）：
  - ✓ 提交请求阶段（Commit Request Phase）也称为投票阶段（Voting Phase）。协调者发送请求给参与者，通知参与者提交或取消事务。参与者进入投票过程，每个参与者回复给协调者自己的投票结果：同意（事务在本地执行成功）或取消（事务本地执行失败）。
  - ✓ 提交阶段（Commit Phase）。协调者对上一阶段参与者的投票结果进行表决。当所有投票为“同意”时提交事务，否则中止事务，并通知参与者。参与者接到通知后执行相应操作。

两阶段提交2PC

- 基本过程

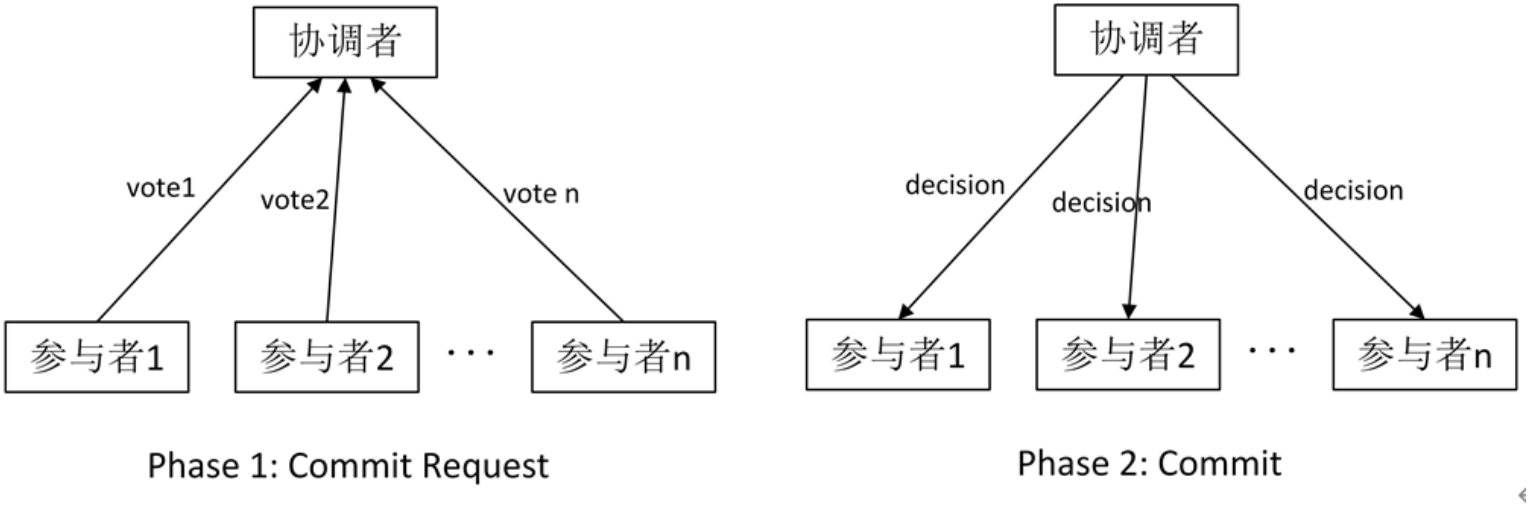


图 8-1 两阶段提交过程示意图

两阶段提交2PC

- 基本过程
- 在第1阶段，协调者首先在自身节点的日志中写入一条日志记录，然后所有参与者发送消息prepare T，询问这些参与者（包括自身），是否能够提交这个事务。
- 参与者在接受到该prepare T 消息后，会根据自身的情况进行事务的预处理。
  - ✓ 如果参与者能够提交该事务，则会将日志写入磁盘，并返回给协调者一个ready T信息，同时自身进入预提交状态。
  - ✓ 若不能提交该事务，则记录日志，并返回一个not commit T信息给协调者；同时撤销在自身对数据库做的更改。参与者能够推迟发送响应的时间，但最终一定要发送。

两阶段提交2PC

- 基本过程
- 在第2阶段，协调者会收集所有参与者的意见。
  - ✓ 如果收到参与者发来的not commit T信息，则表示该事务不能被提交。协调者会将Abort T 记录到日志中，并向所有参与者发送一个Abort T 信息，让所有参与者撤销自身上所有的预操作。
  - ✓ 如果协调者收到所有参与者发来prepare T信息，那么协调者会将Commit T日志写入磁盘，并向所有参与者发送一个Commit T信息，提交该事务。若协调者迟迟未收到某个参与者发来的信息，则认为该参与者发送了一个VOTE_ABORT信息，从而取消该事务的执行。参与者接收到协调者发来的Abort T信息以后，参与者会终止提交，并将Abort T 记录到日志中。
  - ✓ 如果参与者收到的是Commit T信息，则会将事务进行提交，并写入记录。

两阶段提交2PC

- 为了实现分布式事务，必须使用一种协议用来在分布式事务的各个参与者之间传递事务上下文信息。IIOP（Internet Inter-ORB Protocol，互联网内部对象请求代理协议）便是这种协议。
- 两阶段提交保证了分布式事务的原子性，这些子事务要么都做，要么都不做。而数据库的一致性是由数据库的完整性约束实现的，持久性则是通过commit日志来实现，而非由两阶段提交来保证。

两阶段提交2PC

- 异常处理
- 一般情况下，两阶段提交机制都能较好地运行。当事务进行过程中有参与者宕机，在其重启以后，可以通过询问其他参与者或者协调者，从而得知该事务是否提交。但前提是各个参与者在进行每一步操作时，都会事先写入日志。
- 极端的情况是，如果参与者收不到协调者的Commit 或Abort指令，参与者将处于“状态未知”的阶段，从而不知道要如何操作。
  - ✓ 比如，如果所有的参与者完成第1阶段的回复后（可能全部yes，可能全部no，可能部分yes部分no）。如果协调者在此时宕机，那么所有的参与者将无所适从。此时，就可能需要数据库管理员的介入，防止数据库进入一个不一致的状态。

两阶段提交2PC

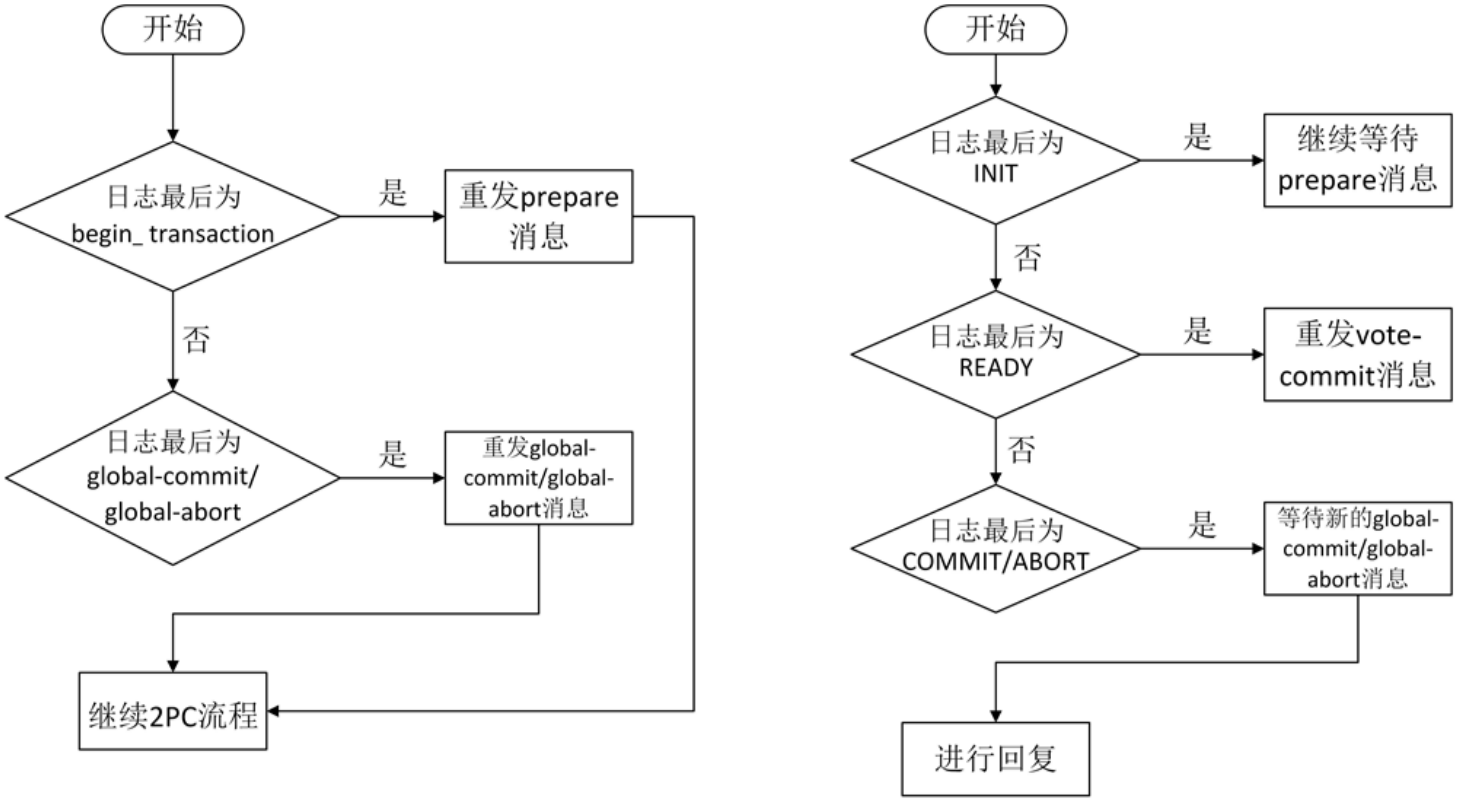
- 异常处理-协调者节点宕机恢复
- 协调节点几种可能的日志记录：begin_transaction，global-commit或global-abort，end_ transaction。
- 协调者宕机恢复后，先让事务恢复到其最新日志记录。
  - ✓ 若是begin_ transaction，表示协调者处于WAIT状态，此时可能已经发送过prepare消息，也可能没有发过。但可以确认，一定没有发送过global-commit或global-abort消息。此时只需要重发prepare消息，即使参与者已经收到并回复过prepare消息，此时只需重新发一条即可，不影响一致性。
  - ✓ 如果日志中最后是global-commit或global-abort日志。说明宕机前处于COMMIT或ABORT状态，此时协调者只需向参与者再发一次global-commit或global-abort消息，继续2PC流程。

两阶段提交2PC

- 异常处理-参与者节点宕机恢复
- 如果日志处于INIT状态，表示还未对本事务做出选择，继续等待prepare消息即可。
- 如果处于READY状态，说明已经收到了prepare消息，但无法得知是否已做出回复，所以重发vote-commit消息即可。注意这里是发送的vote-commit而不是vote-abort，因为只有本次事务可以提交，才会到READY状态；
- 如果日志最后是COMMIT或ABORT状态，则表示已经收到了global-commit或global-abort消息，但无法确定是否已经发送过了确认消息。此时由于协调者节点会不断重发消息，所以只需等待新的global-commit或global-abort消息，并进行回复即可。

两阶段提交2PC

- 异常处理



(1) 协调者节点宕机恢复　　(2) 参与者节点宕机恢复

图 8-2 协调者和参与者的异常处理流程图

两阶段提交2PC

- 超时问题
- 1、协调者在WAIT状态超时：一般有两种原因：a）协调者与某个参与者之间的网络断开；b）某个参与者宕机。这种超时，可以选择放弃整个事务。因为WAIT状态下，协调者一定未发送来global-abort或global-commit消息，因此只要向所有参与者发送global-abort停止事务就可以，不影响协议正确性。
- 2、协调者在COMMIT或ABORT状态超时：等待参与者对global-commit或global-abort的响应消息超时。这种情况下协调者只能不断重发global-commit或global-abort消息，直到所有参与者都响应。

两阶段提交2PC

- 超时问题
- 3、参与者INIT状态超时：此时还没收到prepare消息，直接abort即可。但可能导致原先可以提交的事务不能成功完成。
- 4、参与者READY状态超时：在READY状态，代表参与者收到prepare消息，并回复了vote-commit消息。此时参与者不能再改变自己的选择，只能不断重发vote-commit，直到收到global-abort或global-commit消息，继续下面流程。

两阶段提交2PC

- 2PC遇到的问题
- 1、同步阻塞问题：执行过程中，所有参与节点都是事务阻塞型的。当参与者占有公共资源时，其他第三方节点访问公共资源不得不处于阻塞状态。在大并发下有性能的问题。
- 2、单点故障：由于协调者的重要性，一旦协调者发生故障，参与者会一直阻塞下去。尤其在第二阶段，若协调者发生故障，那么所有的参与者将都处于锁定事务资源的状态中，无法继续完成事务操作。一个解决思路是重新选举一个协调者，但这无法解决因此而导致的使参与者处于阻塞状态的问题。

两阶段提交2PC

- 2PC遇到的问题
- 3、数据不一致：在2PC第二阶段中，当协调者向参与者发送commit请求之后，发生了局部网络异常或者在发送commit请求过程中协调者发生了故障，会导致只有一部分参与者接受到了commit请求。接到commit请求的部分参与者之后就会执行commit操作，但是其他未接到commit请求的节点则无法执行事务提交。于是整个分布式系统便出现了数据不一致的现象。
- 4、过于保守：当任意一个参与者节点宕机，那么协调者超时没收到响应，就会导致整个事务回滚失败。2PC没有设计相应的容错机制。

大纲

- 事务处理基础
  - ✓ 事务的概念
  - ✓ JDBC的事务
- 分布式事务处理
  - ✓ 分布式事务
  - ✓ 事务处理中间件
  - ✓ 两阶段提交2PC
  - ✓ 2PC的应用
- EJB事务体系结构
  - ✓ 容器管理的事务CMT
  - ✓ Bean管理的事务BMT
- JTA事务处理
- 小结

2PC的应用

- XA协议
- XA协议由Tuxedo首先提出并交给X/Open组织，作为资源管理器（数据库）与事务管理器的接口标准。
- XA协议采用两阶段提交方式来管理分布式事务，定义了事务管理器与资源管理器之间通信的接口协议。
- XA定义了一系列的接口，包括xa_start: 启动XA事务；xa_end: 结束XA事务；xa_prepare: 准备阶段，XA事务预提交；xa_commit: 提交XA事务；xa_rollback: 回滚XA事务。

2PC的应用

- XA协议

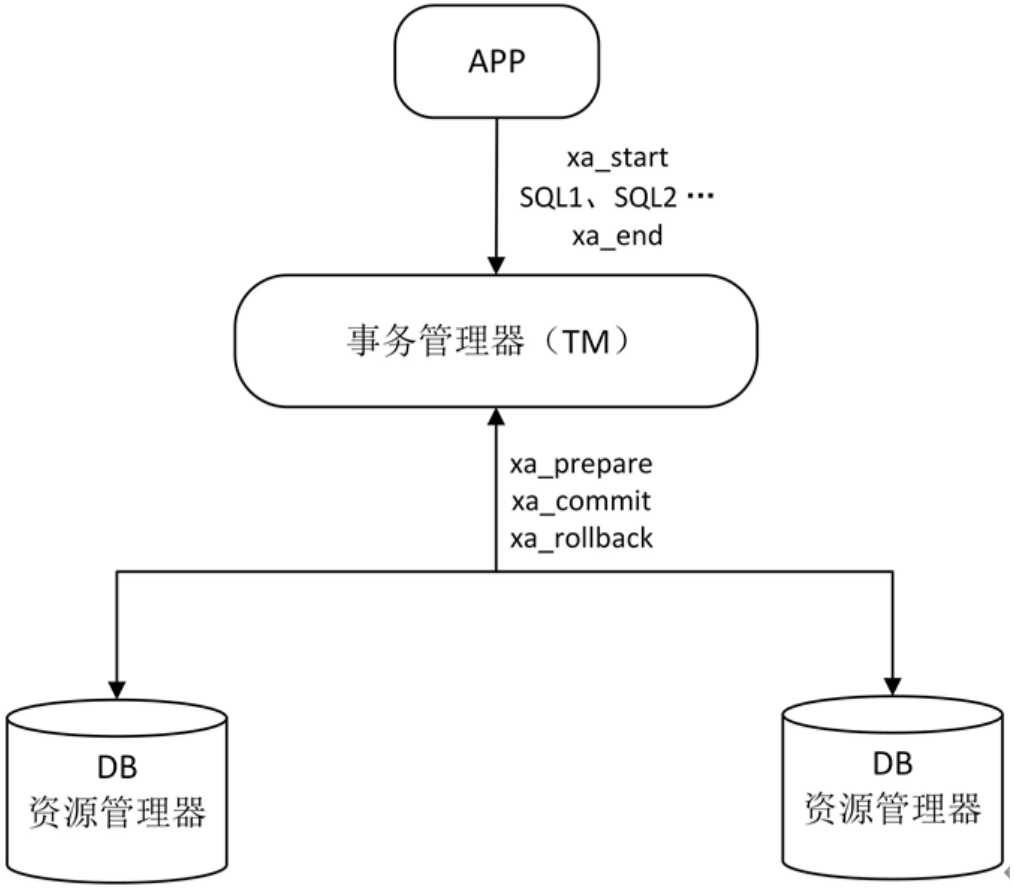


图 8-3 XA 工作机制

2PC的应用

- XA协议
- 一个数据库实现XA协议之后，便可作为一个资源管理器参与到分布式事务中。
- 在2PC的第一阶段，事务管理器协调所有数据库执行XA事务（xa_start、用户SQL、xa_end），并完成XA事务预提交（xa_prepare）。
- 在第二阶段，
  - ✓ 如果所有数据库上XA事务预提交均成功，那么事务管理器协调所有数据库提交XA事务（xa_commit）；
  - ✓ 如果任一数据库上XA事务预提交失败，那么事务管理器会协调所有数据组回滚XA事务（xa_rollback）

2PC的应用

- TCC（Try-Confirm-Cancel）模式
- TCC是Try、Confirm和Cancel 3个单词操作的缩写。
- Try操作对应2PC的第一阶段Prepare。检测、预留资源。
- Confirm对应2PC的第二阶段commit。业务系统执行提交。默认Confirm阶段是不会出错的；只要Try成功，Confirm则一定成功；
- Cancel对应2PC的第二阶段rollback。业务取消，预留资源释放。

2PC的应用

- TCC（Try-Confirm-Cancel）模式
- 在一个跨服务的业务操作中，首先业务发起方通过Try锁住服务中的业务资源进行资源预留，只有资源预留成功了，后续操作才能正常进行。
- Confirm操作是在Try之后进行，对Try阶段锁定的资源进行执行业务操作，类似于传统事务中的commit操作。
- Cancel操作是在操作异常或者失败时进行回滚的操作，类似于传统事务的rollback。在整个TCC方案中需要相关业务方分别提供TCC对应的功能，从而保证事务的强一致性，要么全部成功，要么全部回滚。

2PC的应用

- TCC（Try-Confirm-Cancel）模式
- TCC可以看作是应用层的2PC实现。用户通过编码实现TCC并发布成服务，该TCC服务可作为资源参与到分布式事务中。
- TCC资源管理器可以跨数据库、跨应用实现资源管理，将对不同的数据库访问、不同的业务操作通过编码方式转换一个原子操作，解决了复杂业务场景下的事务问题。
- TCC的每一个操作对于数据库来讲都是一个本地事务，操作结束则本地数据库事务结束，数据库的资源也就被释放了。这可以避免在数据库层面因为2PC对资源的占用而导致的性能低下问题。

大纲

- 事务处理基础
  - ✓ 事务的概念
  - ✓ JDBC的事务
- 分布式事务处理
  - ✓ 分布式事务
  - ✓ 事务处理中间件
  - ✓ 两阶段提交2PC
  - ✓ 2PC的应用
- EJB事务体系结构
  - ✓ 容器管理的事务CMT
  - ✓ Bean管理的事务BMT
- JTA事务处理
- 小结

EJB事务体系结构

- EJB（Enterprise Java Bean）即企业级JavaBean，是一个可重用的、可移植的Java EE组件。
- EJB也是一种规范，目的在于为企业及应用开发人员实现后台业务提供一个标准方式，从而解决一些此前在作业过程中总是重复发生的问题。
- 其特点包括网络服务支持和核心开发工具（SDK）。

EJB事务体系结构

- EJB以一个标准方式自动处理了诸如数据持久化、事务集成、安全对策等不同应用的共有问题，使得软件开发人员可以专注于程序的特定需求。
- EJB有两种管理和使用事务的方式。第一种方式是通过容器管理的事务，称为CMT（Container-Managed Transaction）；另一种是通过Bean管理的事务，称为BMT（Bean-Managed Transaction）。

容器管理的事务CMT

- 在CMT中，容器自动提供事务的开始、提交和回滚操作，且在业务方法的开始和结束处标记事务的边界。
- 开发人员不需要手工编写代码，当程序遇到运行时异常，事务会自动回滚。
- 如果遇到非运行时异常而想要回滚事务的话可以使用SessionContext的setRollBackOnly() 方法来达到目的。

容器管理的事务CMT

```
@Stateless(name = "newhouseManager") //状态定义实列 Bean 提供远程 JNDI
@Remote(INewhouseManager.class) //定义远程接口
@Local(INewhouseManager.class) //定义本地接口
@TransactionManagement(TransactionManagementType.CONTAINER)//定义 CMT 还是 BMT
public class NewhouseManagerImpl implements INewhouseManager{

  @EJB(beanName = "newhouseDAO") //注入 DAO
  private IGenericDAO<Newhouse, Integer> newhouseDAO;

  @TransactionAttribute(TransactionAttributeType.REQUIRED)
  //这里来定义事务的传播特性，如果调用该组件的客户方已经开启了事务则加入原事务，否则开启一个新事务
  public Newhouse save(Newhouse entity) {
    LogUtil.log("saving Newhouse instance", Level.INFO, null);
    try {
      LogUtil.log("save successful", Level.INFO, null);
      entity.setBname("测试 1:" + new Date());
      newhouseDAO.create(entity);
      //插入第一条记录，此时事务还没有提交，数据库里面看不到该记录
      Newhouse entity2 = new Newhouse();
      entity2.setBname("测试 2");
      entity2.setPath(null);
      //这里设置 path 为 null 的话，会出现运行时异常，事务会回滚，entity1 和 entity2 将不会插入到 public 库的 newhouse 表中
      newhouseDAO.create(entity2);
    } catch (RuntimeException re) {
      LogUtil.log("save failed", Level.SEVERE, re);
      re.printStackTrace();
    }
    return null;
  }
}
```

大纲

- 事务处理基础
  - ✓ 事务的概念
  - ✓ JDBC的事务
- 分布式事务处理
  - ✓ 分布式事务
  - ✓ 事务处理中间件
  - ✓ 两阶段提交2PC
  - ✓ 2PC的应用
- EJB事务体系结构
  - ✓ 容器管理的事务CMT
  - ✓ Bean管理的事务BMT
- JTA事务处理
- 小结

Bean管理的事务BMT

- BMT主要是通过手动编程来实现事务的开启、提交和回滚。相对于CMT来说虽然增加了工作量，但是控制力度更细，且更加灵活。
- 在出现异常的时候可以回滚事务，也可以通过JMS返回或者远程调用返回值来控制事务的回滚或提交。
- 使用BMT需要用到UserTransaction这个类的实例来实现事务的begin、commit和rollback。可以通过EJB注解的方式获得这个类实例，也可以用EJBContext.getUserTransaction来获得。

Bean管理的事务BMT

```
@Stateless(name = "newhouseManager") //状态定义实列 Bean 提供远程 JNDI
@Remote(INewhouseManager.class) //定义远程接口
@Local(INewhouseManager.class) //定义本地接口
@TransactionManagement(TransactionManagementType.BEAN) //设置为 BMT 事务
public class NewhouseManagerImpl implements INewhouseManager{
   @Resource
   private UserTransaction ut; //注入 UserTransaction

   @EJB(beanName = "newhouseDAO") //注入 DAO 数据访问对象
   private IGenericDAO<Newhouse, Integer> newhouseDAO;

   @TransactionAttribute(TransactionAttributeType.REQUIRED)
   //设置事务的传播特性为 required
   public Newhouse save(Newhouse entity) {
      LogUtil.log("saving Newhouse instance", Level.INFO, null);
      try {
         ut.begin();//开启事务
         LogUtil.log("save successful", Level.INFO, null);
         entity.setBname("测试 1:" + new Date());
         newhouseDAO.create(entity);
         Newhouse entity2 = new Newhouse();
         entity2.setBname("测试 2");
         entity2.setPath(null);
         newhouseDAO.create(entity2);
         ut.commit(); //提交事务
      } catch(RuntimeException e) {
         ut.rollBack();//发生异常事务回滚
      }
   }
}

```

Bean管理的事务BMT

- 如果使用有状态的会话 Bean且需要跨越方法调用维护事务，那么BMT 则是唯一的选择。
- 当然BMT编程相对复杂，容易出错，且不能连接已有的事务。因此，当调用BMT方法时，会暂停已有事务，这限制了组件的重用。故一般在EJB中会优先考虑CMT事务管理。

大纲

- 事务处理基础
- 分布式事务处理
- EJB事务体系结构
- JTA事务处理
  - ✓ JTA的概念
  - ✓ JTA的实现架构
  - ✓ JTA编程的例子
- 小结

JTA的概念

- Java事务API（Java Transaction API，简称JTA）和Java事务服务（Java Transaction Service，简称JTS），为Java EE平台提供了分布式事务（Distributed Transaction）服务。
- JTA约定各角色进行事务上下文的交互，JTS则基于IIOP（一种软件交互协议）约定各个程序角色之间如何传递事务上下文。
- JTA是一种高层的、与实现和协议无关的API，应用程序和应用服务器都可以通过JTA实现事务管理。JTA允许应用程序执行分布式事务处理，在两个或多个网络计算机资源上访问并且更新数据。

JTA的概念

- JTA是一种高层的、与实现和协议无关的API，应用程序和应用服务器都可以通过JTA实现事务管理。JTA允许应用程序执行分布式事务处理，在两个或多个网络计算机资源上访问并且更新数据。
- 一般地，JTA事务都用于EJB中，用于分布式的多个数据源的事务控制。JTA也是用于管理事务的一套API。与JDBC相比，JTA主要用于管理分布式多个数据源的事务操作，而JDBC主要用于管理单个数据源的事务操作。

JTA的概念

- 在JTA中，一个分布式事务包括一个事务管理器（Transaction Manager）和一个或多个资源管理器（Resource Manager）。
- 资源管理器是任意类型的支持XA协议的持久化数据存储，事务管理器承担着所有事务参与单元的协调与控制。
- JTA 事务有效地屏蔽了底层事务资源，使应用可以以透明的方式参与到事务处理中。但与本地事务相比，XA协议的系统开销大。

大纲

- 事务处理基础
- 分布式事务处理
- EJB事务体系结构
- JTA事务处理
  - ✓ JTA的概念
  - ✓ JTA的实现架构
  - ✓ JTA编程的例子
- 小结

JTA的实现架构

- 根据所面向对象的不同，可以从两个方面理解JTA的事务管理器和资源管理器：面向开发人员的使用接口（事务管理器）和面向服务提供商的实现接口（资源管理器）。
- 开发接口的主要部分为8.4.3示例中引用的 UserTransaction 对象，开发人员通过此接口在信息系统中实现分布式事务
- 实现接口则用来规范提供商（如数据库连接提供商）所提供的事务服务，它约定了事务的资源管理功能，使得JTA可以在异构事务资源之间执行协同沟通。

JTA的实现架构

- 基于统一规范的不同实现，JTA可以协调和控制不同数据库或者JMS厂商的事务资源

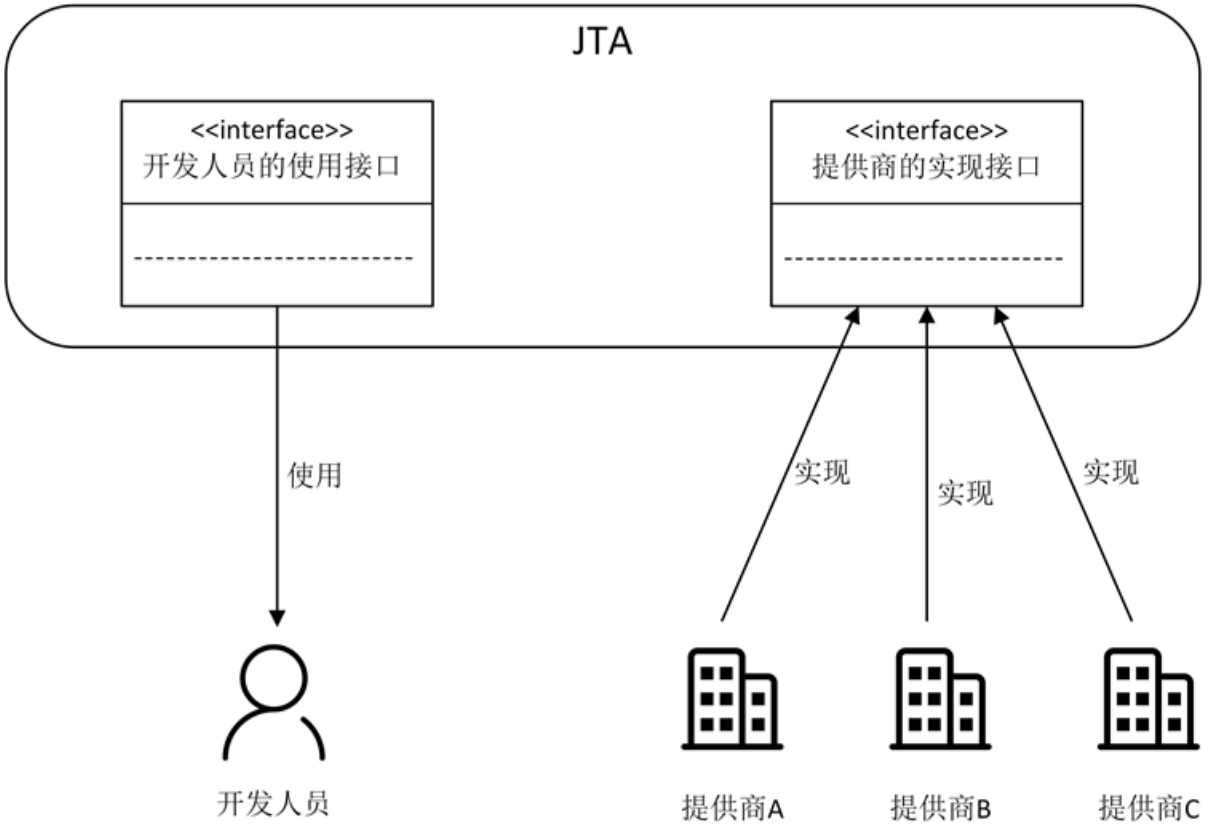


图 8-4 JTA 接口示意图

JTA的实现架构

- 开发人员使用开发人员接口，实现应用程序对全局事务的支持。各提供商（如数据库、JMS 等）依据提供商接口的规范提供事务资源管理功能。
- 事务管理器则把应用中的分布式事务映射到实际的事务资源，并在事务资源间进行协调与控制。

JTA的实现架构

- 面向开发人员的接口
- 面向开发人员的接口为 UserTransaction，它定义了如下的方法：
  - ✓ begin()：开始一个分布式事务。TransactionManager会创建一个Transaction事务对象，并把此对象通过ThreadLocale关联到当前线程上。
  - ✓ commit()：提交事务。TransactionManager会从当前线程下取出事务对象并把此对象所代表的事务提交。
  - ✓ rollback()：回滚事务。TransactionManager会从当前线程下取出事务对象并把此对象所代表的事务回滚。
  - ✓ getStatus()：返回关联到当前线程的分布式事务的状态。Status对象里边定义了所有的事务状态，感兴趣的读者可以参考API文档。
  - ✓ setRollbackOnly()：标识关联到当前线程的分布式事务将被回滚

JTA的实现架构

- 面向开发人员的接口

```
public interface UserTransaction {
   //创建与当前线程相关的事务
   void begin()throws NotSupportedException, SystemException;
   
   //事务提交。这个方法执行后，线程与事务没有任何关系
   void commit() throws RollbackException, HeuristicMixedException,
                        HeuristicRollbackException, SecurityException,
                        IllegalStateException, SystemException;
   
   //事务回滚。这个方法执行后，线程与事务没有任何关系
   void rollback() throws IllegalStateException, SecurityException, SystemException;
   
   //修改事务。事务的唯一可能的结果是回滚事务
   void setRollbackOnly() throws IllegalStateException, SystemException;
   
   //获取事务状态
   int getStatus() throws SystemException;
   
   /**
   * 设置事务超时，单位秒。若超时，将回滚到当前线程的起始方法
   *
   * 应用程序没有调用该方法，则有默认值
   * 若设置为 0，超时为默认值
   * 为负数，则抛出 SystemException
   */
   void setTransactionTimeout(int seconds) throws SystemException;
}

```

JTA的实现架构

- 面向提供商的实现接口
- 面向提供商的实现接口主要涉及到 Transaction 和 TransactionManager 两个对象。
- Transaction 代表了一个物理意义上的事务，在开发人员调用 UserTransaction.begin()方法时 TransactionManager 会创建一个 Transaction 事务对象（标志着事务的开始）并把此对象通过 ThreadLocale 关联到当前线程。UserTransaction 接口中的 commit()、rollback()，getStatus()等方法都将最终委托给 Transaction 类的对应方法执行。

JTA的实现架构

- 面向提供商的实现接口
- Hibernate等ORM工具都有自己的事务控制机制来保证事务，但同时它们还需要一种回调机制以便在事务完成时得到通知从而触发一些处理工作，如清除缓存等。这就涉及到Transaction的回调接口registerSynchronization。工具可以通过此接口将回调程序注入到事务中，当事务成功提交后，回调程序将被激活。

JTA的实现架构

- 面向提供商的实现接口
- TransactionManager本身并不承担实际的事务处理功能，它更多的是充当用户接口和实现接口之间的桥梁。
- 在开发人员调用UserTransaction.begin()方法时TransactionManager会创建一个Transaction事务对象（标志着事务的开始）并将此对象通过ThreadLocale关联到当前线程上；同样UserTransaction.commit()会调用TransactionManager.commit()方法从当前线程下取出事务对象Transaction并把该对象所代表的事务提交，即调用Transaction.commit()。

大纲

- 事务处理基础
- 分布式事务处理
- EJB事务体系结构
- JTA事务处理
  - ✓ JTA的概念
  - ✓ JTA的实现架构
  - ✓ JTA编程的例子
- 小结

JTA编程的例子

- JTA的实现框架有GeronimoTM/Jencks、SimpleJTA、Atomikos、JOTM以及JBossTS等。JTA和JTS提供了分布式事务服务，分布式事务包括事务管理器和XA协议的资源管理器。
- 资源管理器可看做是任意类型的持久化数据存储，事务管理器承担着事务协调与控制。

JTA编程的例子

- 使用 JTA 处理事务的示例如下，其中connA和connB是来自不同数据库的连接。

```
public void transferAccount() {
   UserTransaction userTx = null;
   Connection connA = null;
   Statement stmtA = null;
   Connection connB = null;
   Statement stmtB = null;
   try{
      //获得 Transaction 管理对象
      userTx = (UserTransaction)getContext().lookup("\
                  java:comp/UserTransaction");
      //从数据库 A 中取得数据库连接
      connA = getDataSourceA().getConnection();
      //从数据库 B 中取得数据库连接
      connB = getDataSourceB().getConnection();
      //启动事务
      userTx.begin();
      //将 A 账户中的金额减少 500
      stmtA = connA.createStatement();
      stmtA.execute("update t_account set amount = amount - 500 where account_id = 'A'");
      //将 B 账户中的金额增加 500
      stmtB = connB.createStatement();
      stmtB.execute("update t_account set amount = amount + 500 where account_id = 'B'");
      //提交事务
      userTx.commit();
      //事务提交：转账的两步操作同时成功（数据库 A 和 B 中的数据被同时更新）
```

JTA编程的例子

```
    } catch(SQLException sqle){
        try{
            //发生异常，回滚在本事务中的操纵
            userTx.rollback();
            //事务回滚：转账的两步操作完全撤销
            //数据库 A 和数据库 B 中的数据更新被同时撤销
            stmt.close();

            conn.close();
            sqle.printStackTrace();
        } catch(Exception ne){
            e.printStackTrace();
        }
    }
}

```

JTA编程的例子

- 使用JTA为EJB提供事务，结合注解的方式也非常直观，只需要提供@TransactionManagement和@TransactionAttribute，并提供相应属性配置即可，

```
@Stateless
@Remote({ ITemplateBean.class})
@TransactionManagement(TransactionManagementType.CONTAINER)
@TransactionAttribute(TransactionAttributeType.REQUIRED)
public class TemplateBeanImpl implements ITemplateBean {
   //code...
}
```

本章小结

- 事务处理中间件（Transaction Processing Middleware，简称TPM）是在分布、异构环境下保证事务完整性和数据完整性的一种环境平台，它提供了一种专门针对联机事务处理系统而设计的事务控制机制。

- 本章首先介绍了事务的基本概念，然后详细介绍了分布式事务处理，包括事务处理中间件的概念和两阶段提交协议；接着介绍了EJB事务体系结构，包括容器管理的事务处理和Bean管理的事务处理；最后介绍了JTA事务处理的机制，通过实际编程例子来帮助读者学习理解JTA的工作原理。