**中间件分布式事务系统Bee架构设计概要**

居静浩

1. **前言**

为了应对快速变化的市场需求、持续增长的业务量，支付系统通常会基于SOA进行构建与改造，以应对系统规模和复杂性的挑战，更好地进行企业内与企业间的协作。基于SOA图景，整个支付系统会拆分成一系列独立开发、自包含、自主运行的业务服务，并将这些服务通过各种机制灵活地组装成最终用户所需要的产品与解决方案。在SOA的系统架构下，一次业务请求将会跨越多个服务。

在多个服务协同完成一次业务时，由于业务约束（如红包不符合使用条件、账户余额不足等）、系统故障（如网络或系统超时或中断、数据库约束不满足等），都可能造成服务处理过程在任何一步无法继续，使数据处于不一致的状态，产生严重的业务后果，比如资产损失等。

传统的基于数据库本地事务的解决方案只能保障单个服务的一次处理具备原子性、隔离性、一致性与持久性，但无法保障多个分布服务间处理的一致性。因此，我们必须建立一套分布式服务处理多系统之间的协调机制，保障分布式服务处理的原子性、隔离性、一致性与持久性。

1. **事务与分布式事务**

事务（Transaction）是并发控制的基本单位。所谓的事务，它是一个操作序列，这些操作要么都执行，要么都不执行，它是一个不可分割的工作单位。例如，银行转账工作：从一个账号扣款并使另一个账号增款，这两个操作要么都执行，要么都不执行。所以，应该把它们看成一个事务。事务是数据库维护数据一致性的单位，在每个事务结束时，都能保持数据一致性。

针对上面的描述可以看出，事务的提出主要是为了解决并发情况下保持数据一致性的问题。

事务具有以下4个基本特征。

Atomic（原子性）：事务中包含的操作被看做一个逻辑单元，这个逻辑单元中的操作要么全部成功，要么全部失败。

Consistency（一致性）：只有合法的数据可以被写入数据库，否则事务应该将其回滚到最初状态。

Isolation（隔离性）：事务允许多个用户对同一个数据进行并发访问，而不破坏数据的正确性和完整性。同时，并行事务的修改必须与其他并行事务的修改相互独立。

Durability（持久性）：事务结束后，事务处理的结果必须能够得到固化。

根据业务系统和所操作的数据源的分布，会有四种情况，如图1所示：

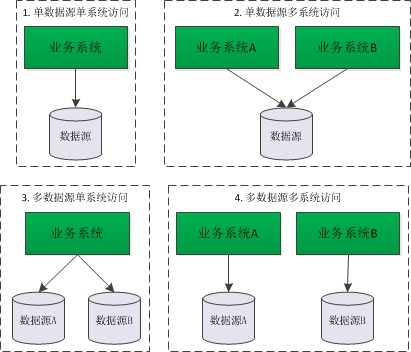


图1. 事务管理

第一种情况是单数据源单系统访问，只要使用对应数据源所提供的事务管理器（比如数据库事务管理器）就可以做到ACID。其他三种情况，都涉及到多个系统节点，每个节点虽然可以知晓自己的操作时成功或者失败，却无法知道其他节点的操作的成功或失败。当一个事务跨越多个节点时，为了保持事务的 ACID特性，需要引入一个作为协调者的组件，称为分布式事务管理器，由它来统一掌控所有节点(称作参与者)的操作结果并最终指示这些节点是否要把操作结果进行真正的提交(比如将更新 后的数据写入磁盘等等)。

在JavaEE规范中使用 2PC (2 Phase Commit, 两阶段提交) 来处理跨系统环境下的事务问题，但是 2PC 是反可伸缩模式，也就是说，在事务处理过程中，参与者需要一直持有资源直到整个分布式事务结束。这样，当业务规模达到千万级以上时，2PC 的局限性就越来越明显，系统可伸缩性会变得很差。基于此，我们采用BASE（Basically Availble基本可用, Soft-state软状态/柔性事务, Eventual Consistency最终一致性）的思想实现了一套类似 2PC 的分布式事务方案，这就是中间件分布式事务系统Bee。Bee在充分保障分布式环境下高可用性、高可靠性的同时兼顾数据一致性的要求，其最大的特点是保证数据最终一致 (Eventually consistent)

简单的说，Bee 框架有如下特性：

* **最终一致：**事务处理过程中，会有短暂不一致的情况，但通过主从事务的协调和恢复系统，可以让事务的数据达到最终一致的目标。
* **协议简单：**Bee 定义了类似 2PC 的标准两阶段接口，业务系统只需要实现对应的接口就可以使用 Bee 的事务功能。
* **与 RPC 服务协议无关：**在 SOA 架构下，一个或多个 DB 操作往往被包装成一个一个的 Service，Service 与 Service 之间通过Dubbo协议通信。Bee 框架构建在 SOA 架构上，与底层协议无关。
* **与底层事务实现无关：** Bee 是一个抽象的基于 Service 层的概念，与底层事务实现无关，也就是说在 Bee 的范围内，无论是关系型数据库 MySQL，Oracle，还是 KV 存储 MemCache，或者列存数据库 HBase，只要将对其的操作包装成 Bee 的参与者，就可以接入到 Bee 事务范围内。

1. **Bee架构设计**

Bee的整体架构如图2所示：

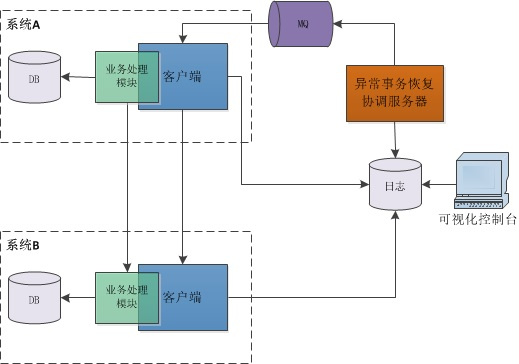


图2. Bee整体架构图

Bee由三个系统组成：

1. **Bee客户端**，面向业务系统，主要负责与业务系统的整合，管理分布式事务的状态流转，协调主事务与分支事务的同步，事务日志，异常事务恢复等。
2. **Bee异常事务恢复协调服务器**，独立服务，主要负责定时监视可能存在的超时分支事务，并通知客户端的主事务管理器进行异常事务恢复，定期清理孤儿分支事务，对已经完成的分布式事务进行归档整理等。
3. **Bee可视化管理控制台**，独立服务，主要负责分布式事务的状态流程可视化，事务执行分析统计，异常事务的跟踪等。

值得指出的是，图2中，异常事务恢复协调服务器和事务恢复器之间，都是通过MQ异步消息来协调的。MQ作为中间者，可以有效解耦N\*M的错综复杂关系。

* 1. **Bee客户端架构设计**

Bee客户端是整个Bee的最核心组成部分，架构如图3所示：

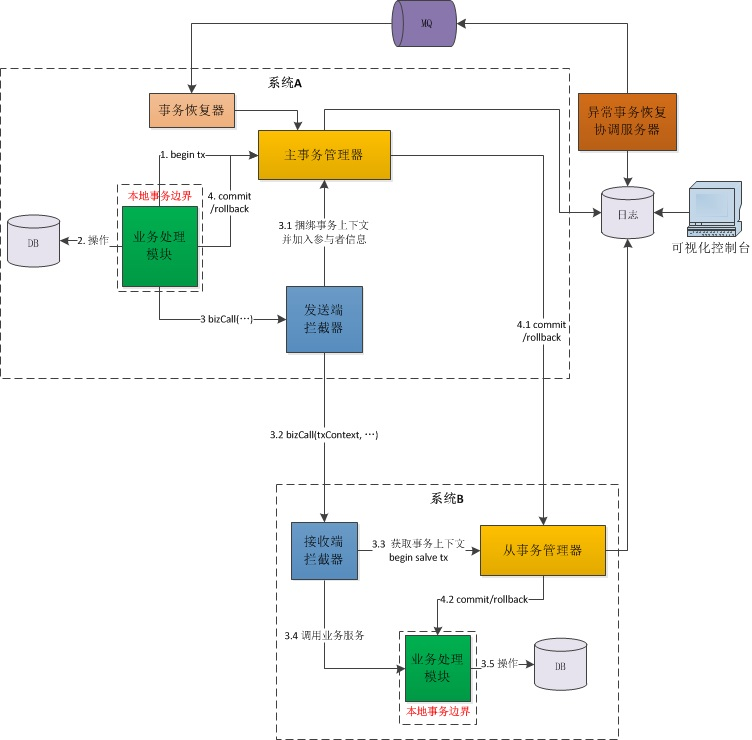


图3. Bee客户端架构图

Bee客户端的相关组件位于图3中虚线框标识的系统A和系统B内，其中标注了数字的线条表示一个正常分布式事务流程（即从begin到commit）在客户端各组件中的交互过程。下面是各组件的功能介绍：

* **事务管理器**，事务管理器管理着所有本地启动的业务活动，并负责协调本地业务活动和其所涉及到的各分支业务活动之间的同步性，每一个业务活动需要相应的BusinessActivityId作为唯一标示。事务管理器根据功能又细分为主事务管理器和分支事务管理器，事实上事务管理器并不直接操作业务对象的提交和回滚，而是直接和分支事务管理器进行交互。
  + 主事务管理器，是分布式事务的发起方和协调者，负责启动分布式事务，触发并创建相应的主事务日志，感知整个分布式事务状态来决定整个事务是 COMMIT 还是 ROLLBACK等。
  + 分支事务管理器，负责维护分支事务的参与者，包括同步参与者的状态（准备完成、提交完成、回滚完成），执行参与者的提交和回滚操作等。
* **服务拦截器**，拦截服务的调用和请求，根据功能又细分为发送端拦截器和接收端拦截器。
  + 发送端拦截器拦截服务调用，在消息中附加业务活动上下文，以实现业务活动上下文跨服务传递，并在当前业务活动中添加参与者原子动作。
  + 接收端拦截器拦截服务请求，从消息中析取业务活动上下文，启动分支业务活动，并注册到分支事务管理器中。
* **事务日志服务**，负责事务日志的持久化工作，具体可分为主事务日志和分支事务日志。主事务日志状态有三个：STARTED、COMMITTED、ROLLBACKED，分别对应主事务已启动、已提交、已回滚。分支事务状态有四个：STARTED、PREPARED、COMMITTED、ROLLBACKED，分别对应分支事务已启动、已准备就绪、已提交、已回滚。
* **事务恢复器**，负责异常事务的恢复，根据异常事务恢复协调服务器所发来的异常报告，事务恢复器根据情况协调各相关的分支事务管理器进行重新提交或者回滚等操作，在必要的情况下，还需要调用业务系统提供的主事务状态回查SPI实现来确定主事务的状态。

**注意事项**：

接入Bee客户端的业务系统需要遵守以下规则才能正确使用Bee分布式事务所提供的服务：

* 必须在本地事务块中开启分布式事务，即Bee的分布式事务必须包含在本地事务中。
* 必须将分支系统的调用包装成Bee客户端可以识别的参与者（参与者是分布式事务中的一个原子单位），需要两个步骤，对于调用方，需要将SOA服务在本地进行包装，并在调用方法体上设置PrepareCaller注解来标识这会启动一个分支事务。另一方面，服务提供方（即被调用方）需要在服务实现方法上设置Prepare注解来标识分支事务参与者身份，同时还必须实现Participant接口（它定义了commit和rollback 2个基本接口），并保证其业务数据的操作幂等性，同时必须保证 prepare 中的数据操作能够被提交 (COMMIT) 或者回滚 (ROLLBACK)。
  1. **异常事务恢复设计**

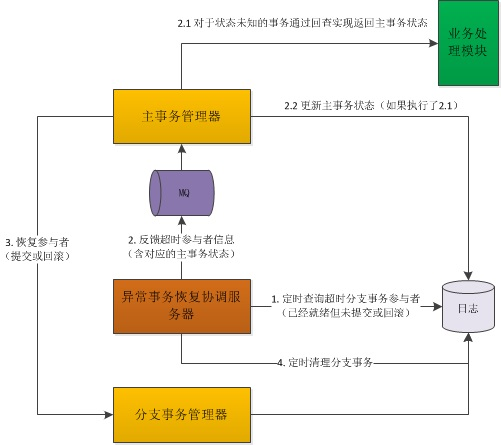


图4. 异常事务恢复

图4展示了Bee的事务恢复系统，事务恢复需要经过4个阶段：

* 1. 定时扫描超时分支事务，即扫描长时间处于PREPARED状态的分支事务，超时时间的定义应该在主事务启动的时候设定，可以有默认值，比如60秒。
  2. 归类反馈超时分支事务给对应的主事务，通过主事务ID来归类超时的分支事务，并将汇总后的信息发送给对应的主事务管理器。
  3. 主事务尝试恢复分支事务，主事务管理器根据主事务的状态来执行提交或者回滚，如果主事务的状态为STARTED（即没有COMMITTED或者ROLLBACKED），那么需要回查业务系统实现的回查SPI实现来确定主事务的真正状态。
  4. 定期清理分支事务，包括清理已经完成的主从事务和悬挂的无效分支事务，这里的清理不一定是指删除，而可能是移动到后台只读数据库中。
  5. **Bee可视化管理控制台设计**

可视化管理控制台是一个B/S系统，通过网页展示Bee的各项功能，功能点可能包括如下几点：

1. 主从事务之间的可视化关联关系
2. 主事务信息、执行状态等显示
3. 各分支事务详细信息、执行状态等显示
4. 统计事务执行的成功、失败、恢复等信息
5. 管理员操作，比如手动发起异常事务的提交或者回滚，清理事务信息等
6. 导入导出操作，如导入业务Id定义信息，导出事务执行统计结果等
7. **日志数据库设计**

日志数据库的表结构设计，主要是三张表，分别是主事务表、分支事务表和业务Id定义表。其中业务Id定义表是粗略设计，仅存放各业务系统相关的注册标识信息，主要用于异常恢复的时候使用。

下面是以MySQL为例的DDL，

CREATE TABLE MASTER\_TRANSACTION(

MTID BIGINT(20) NOT NULL AUTO\_INCREMENT COMMENT '主键',

BID BIGINT NOT NULL COMMENT '业务领域ID',

STATUS INT(1) NOT NULL COMMENT '主事务状态, STARTED, COMMITTED, ROLLBACKED',

CREATED\_TIME BIGINT(20) NOT NULL,

UPDATED\_TIME BIGINT(20),

PRIMARY KEY(MTID),

FOREIGN KEY (BID) REFERENCES BIZ\_DEFINITION(BID) ON DELETE CASCADE

)ENGINE=InnoDB;

CREATE TABLE SLAVE\_TRANSACTION(

STID BIGINT(20) NOT NULL AUTO\_INCREMENT COMMENT '主键',

MTID BIGINT(20) NOT NULL COMMENT '主事务表主键',

BID BIGINT NOT NULL COMMENT '业务领域ID',

STATUS INT(1) NOT NULL COMMENT '从事务状态, STARTED, PREPARED, COMMITTED, ROLLBACKED',

TIMEOUT INT(11) NOT NULL COMMENT '事务超时值，单位秒',

CREATED\_TIME BIGINT(20) NOT NULL,

UPDATED\_TIME BIGINT(20),

PRIMARY KEY(STID),

FOREIGN KEY (MTID) REFERENCES MASTER\_TRANSACTION(MTID) ON DELETE CASCADE,

FOREIGN KEY (BID) REFERENCES BIZ\_DEFINITION(BID) ON DELETE CASCADE

)ENGINE=InnoDB;

CREATE TABLE BIZ\_DEFINITION(

BID BIGINT(20) NOT NULL AUTO\_INCREMENT COMMENT '主键',

BIZ\_DOMAIN VARCHAR(32) NOT NULL COMMENT '业务领域',

BIZ\_OPERATION VARCHAR(32) NOT NULL COMMENT '业务操作',

BIZ\_ENTITYID VARCHAR(32) NOT NULL COMMENT '业务实体',

CREATED\_TIME BIGINT(20) NOT NULL,

UPDATED\_TIME BIGINT(20),

PRIMARY KEY(BID),

UNIQUE(BIZ\_DOMAIN, BIZ\_OPERATION, BIZ\_ENTITYID)

)ENGINE=InnoDB;