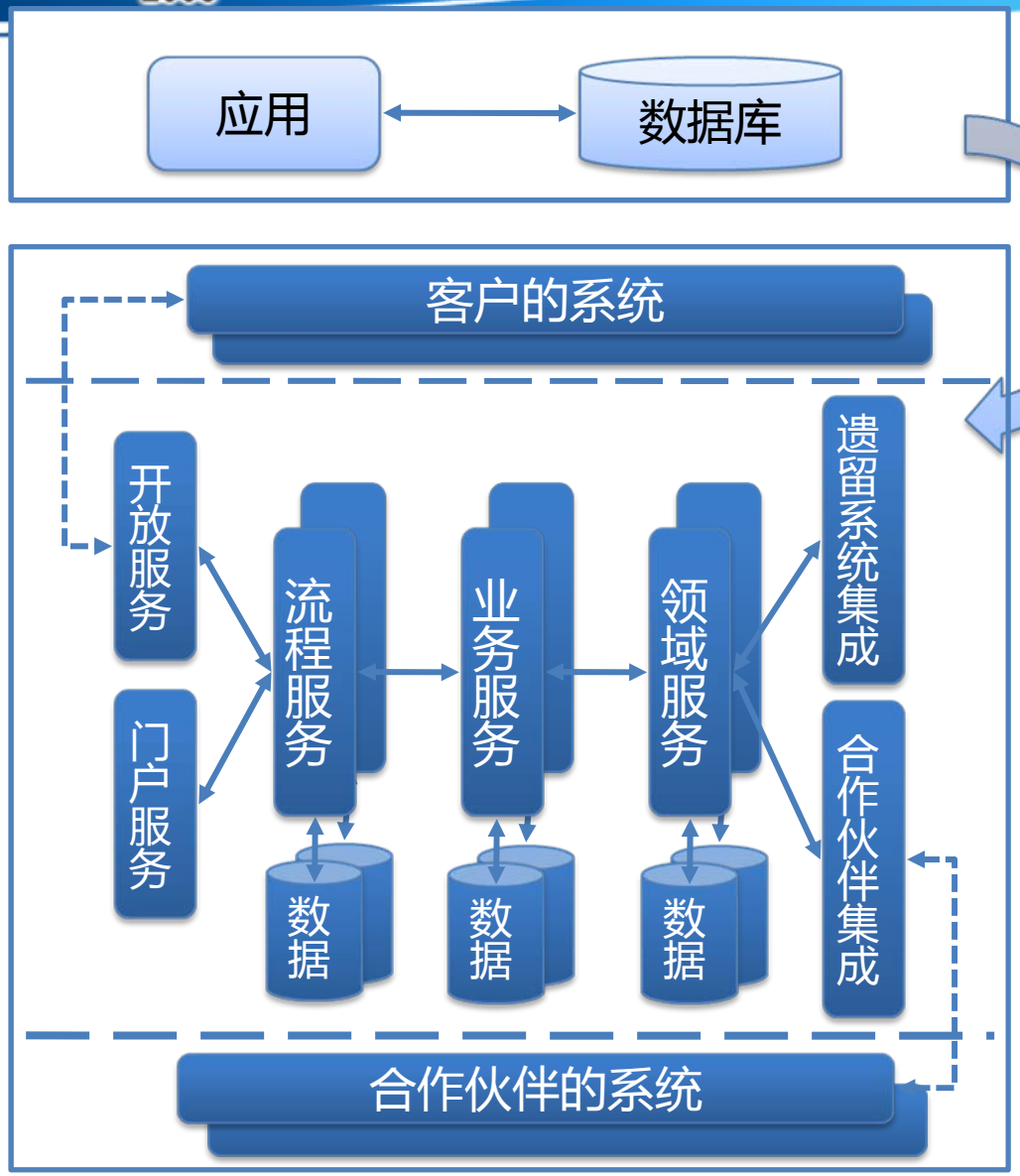
大规模SOA系统中的分布事务处理

程立
支付宝产品技术与用户体验部
2008年12月

提要



- 从单应用系统的事务
- 到大规模SOA系统中的事务
- 内容提要
  - **山穷水尽**(背景与历史)
  - **柳暗花明**(原则与模式)
  - **又一山寨**(框架与设施)

山穷水尽

Googling

- “transaction processing”

  约有1,940,000项符合的查询结果
- “distributed transaction”

  约有260,000项符合的查询结果
- “distributed transaction” + practice

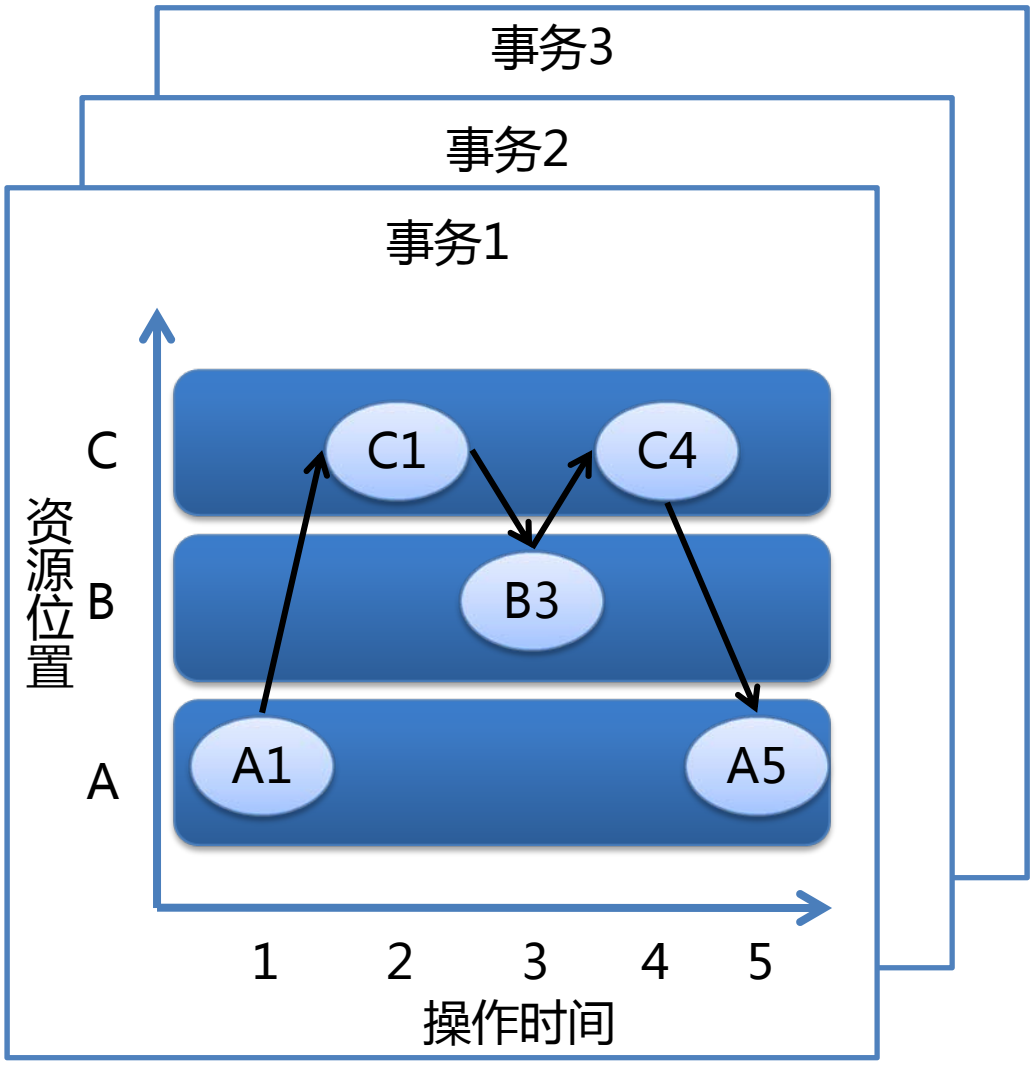
  约有24,700项符合的查询结果
- “distributed transaction” + “success story”

  约有265项符合的查询结果 ☹
- “distributed transaction” + sucks

  约有1,370项符合的结果
- “distributed transaction” + hope

  约有17,500项符合的结果 ☺

事务



事务:

由一组操作构成的**可靠**、**独立**的工作单元

ACID:

- Atomicity(原子性)
- Consistency(一致性)
- Isolation(隔离性)
- Durability(持久性)

难点:

- 高度并发
- 资源分布
- 大时间跨度

本地事务

本地事务

事务由资源管理器（如DBMS）**本地**管理

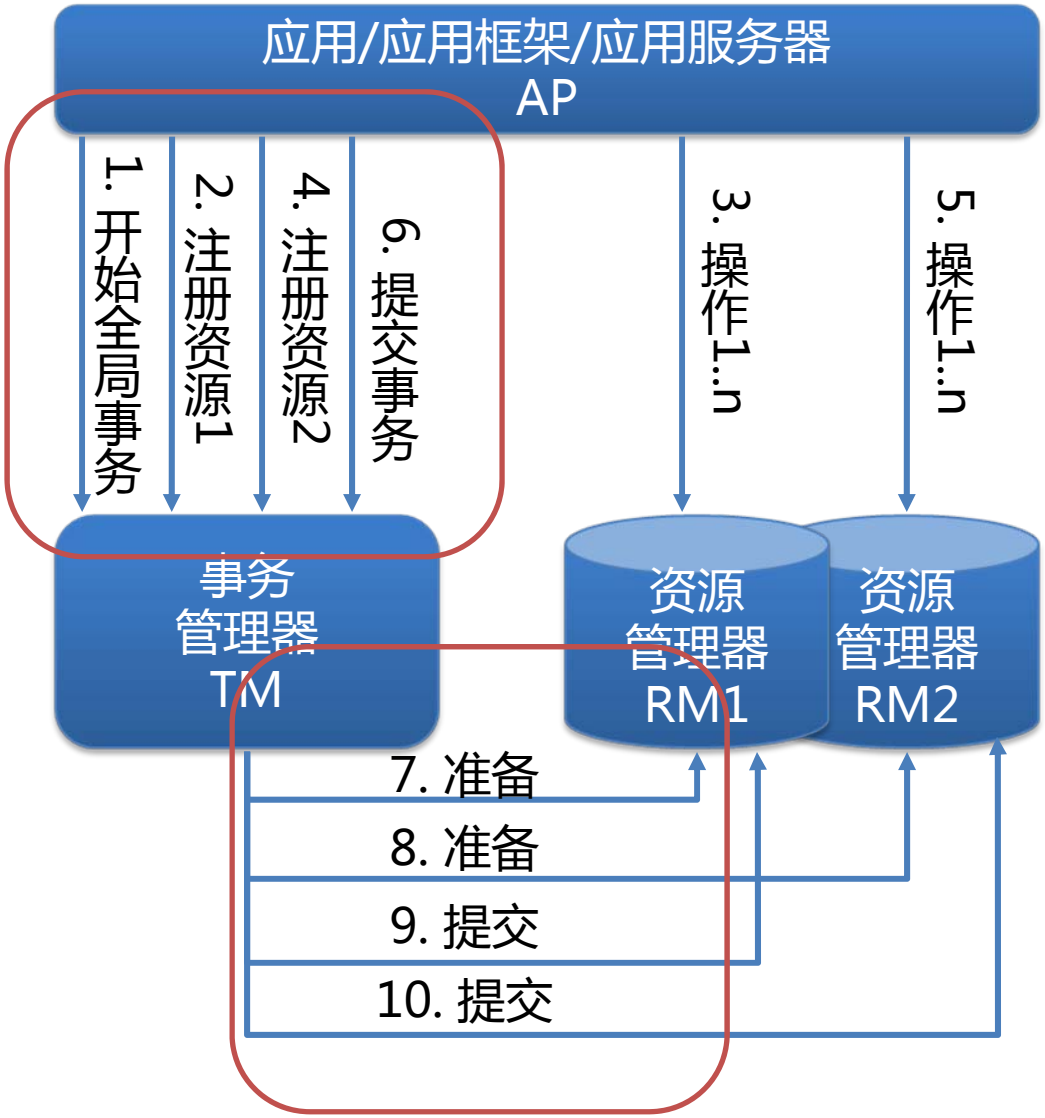
优点

- 支持严格的ACID属性
- 可靠
- 高效
- 状态可以只在资源管理器中维护
- 应用编程模型简单(在框架或平台的支持)

局限

- 不具备分布事务处理能力
- 隔离的最小单位由资源管理器决定，如数据库中的一条记录

全局事务(DTP模型)



全局事务

事务由全局事务管理器**全局**管理

事务管理器

管理全局事务状态与参与的资源，协同资源的一致提交/回滚

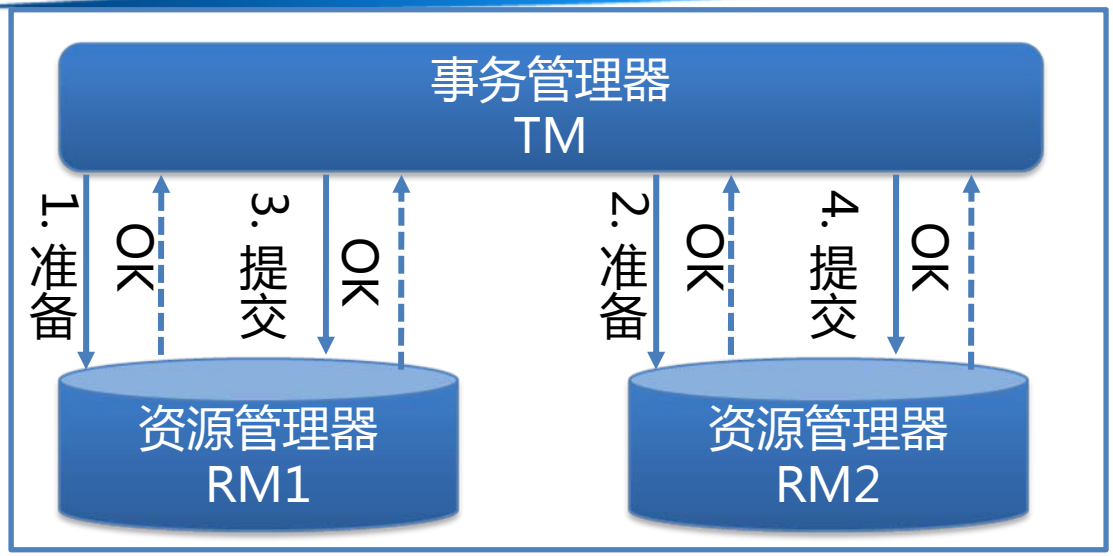
TX协议

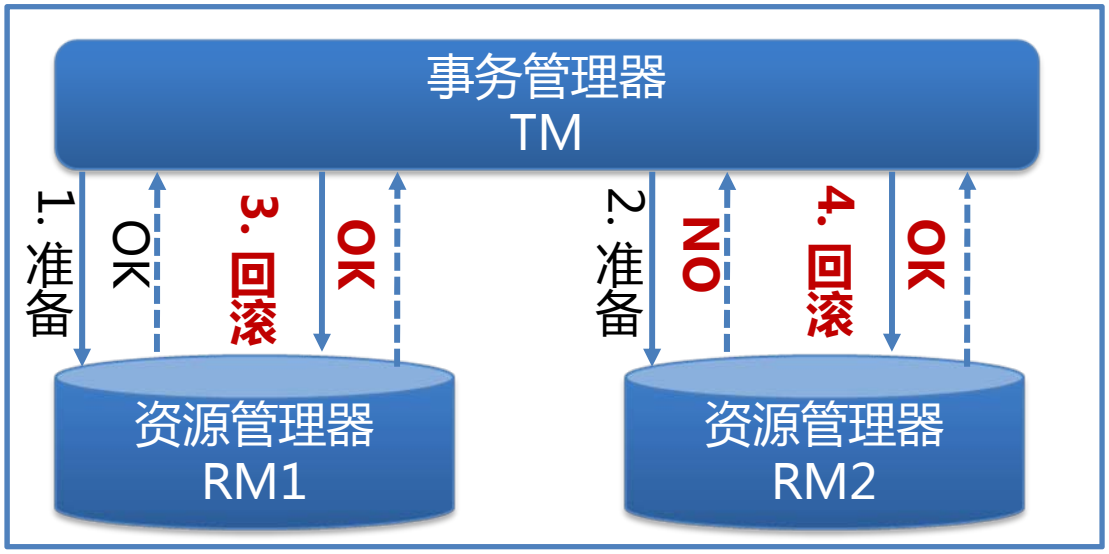
应用或应用服务器与事务管理器的接口

XA协议

全局事务管理器与资源管理器的接口

两阶段提交(Two Phase Commit)





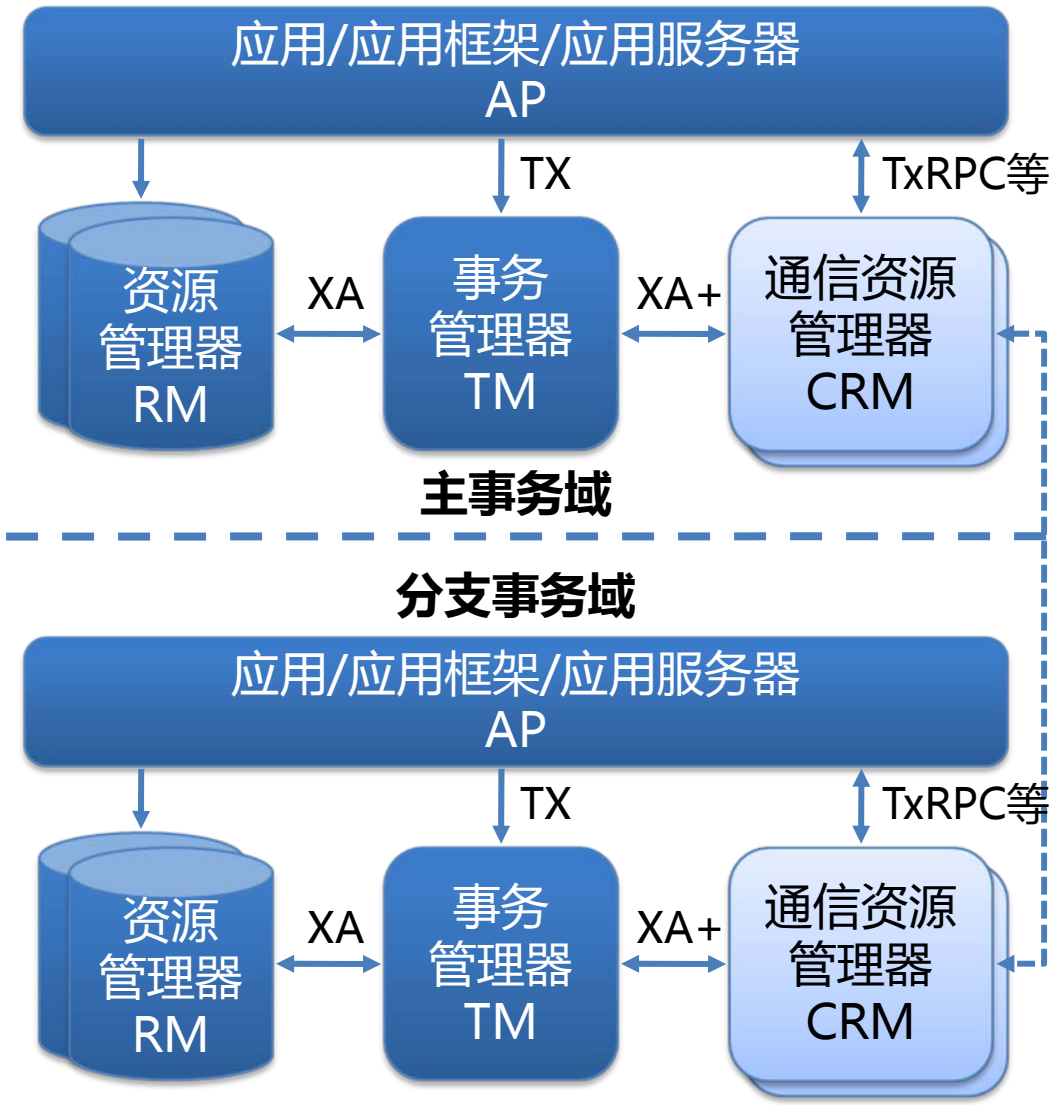
准备操作与ACID

- A: 准备后，仍可提交与回滚
- C: 准备时，一致性检查必须OK
- I: 准备后，事务结果仍然只在事务内可见
- D: 准备后，事务结果已经持久

局限

- 协议成本 (准备操作是一定必须的吗)
- 准备阶段的持久成本
- 全局事务状态的持久成本
- 潜在故障点多带来的脆弱性
- 准备后，提交前的故障引发一系列隔离与恢复难题

跨域的全局事务(DTP模型)



问题

- 事务上下文如何跨域传递?
- 多事务管理器如何协同?
- 异构事务域间的标准是什么?

通信资源管理器

管理事务域间或事务域内的通信，允许全局事务信息跨域传递

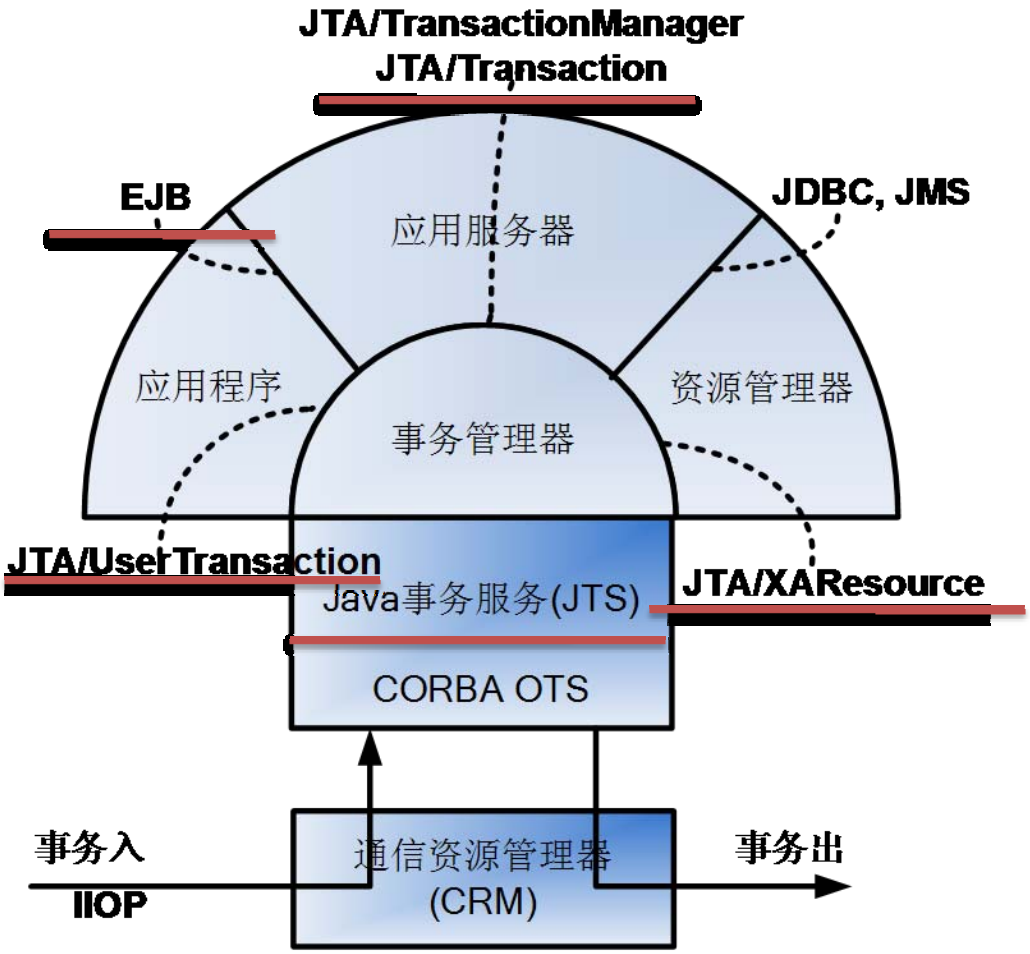
XA+协议

是XA的超集，增加指令使事务管理器间可以相互协同

局限

- 更高协议成本
- 脆弱，故障点多
- 故障影响大，恢复困难
- 复杂，更多架构与平台约束

Java企业平台中的分布事务实现



JTA

面向应用、应用服务器与资源管理器的高层事务接口

JTS

JTA事务管理器的实现标准，向上支持JTA，向下通过CORBA OTS实现跨事务域的互操作性

EJB

基于组件的应用编程模型，通过声明式事务管理进一步简化事务应用的编程

优点

- 简单一致的编程模型
- 跨域分布处理的ACID保证

局限

- DTP模型本身的局限
- 缺少充分公开的大规模、高可用、密集事务应用的成功案例

其它资源

WS-Transaction标准

OASIS组织通过的Web Service事务标准，包含WS-Cordination、WS-AtomicTransaction、WS-BusinessActivity

JbossTransactions系统

开源的JTA、JTS、WS-Transaction标准的实现

Paxos算法

分布式系统中就某个提议达成一致决议的算法族

柳暗花明

原则

- 真正重要的是满足业务需求，而不是追求抽象、绝对的系统特性
- 帽子戏法：两只手最多能拿几顶帽子？
- 酸碱平衡(ACID-BASE Balance)

模式

- 服务模式
  1. 可查询操作
  2. 幂等操作
  3. TCC操作
  4. 可补偿操作
- 复合模式
  1. 定期校对
  2. 可靠消息
  3. TCC
  4. 补偿

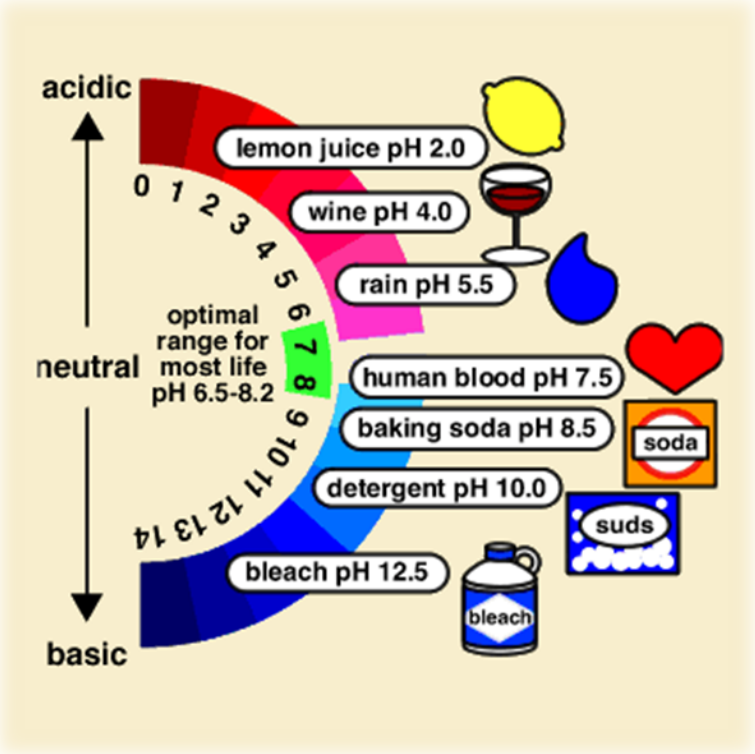
帽子戏法

CAP定理

对于共享数据系统，只能同时拥有以下三项中的两个：

- Consistency(一致性): 所有用户看到一致的数据
- Availability(可用性): 总能找到一个可用的数据复本
- Tolerance to Network Partition(分区容忍性): 即使在系统被分区的情况下，仍然满足上述两点

酸碱平衡



BASE

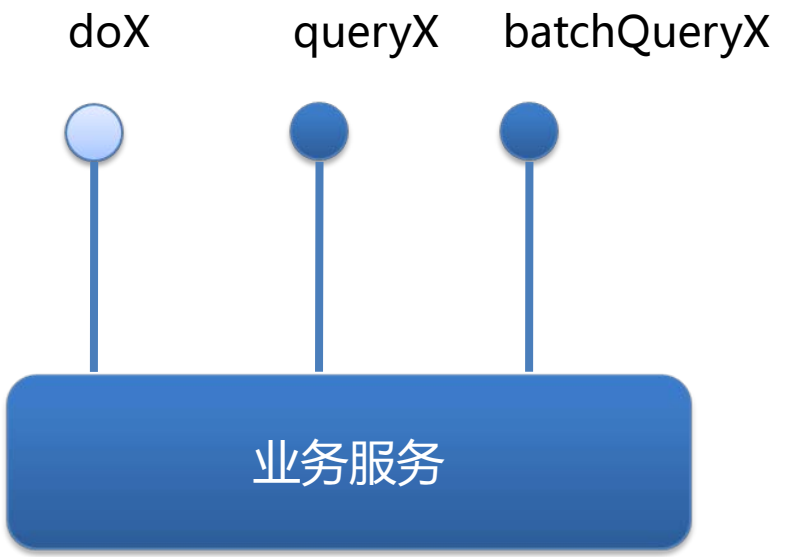
- BA(**B**asic **A**vailability)
  基本可用性
- S(**S**oft state)
  柔性状态
- E(**E**ventuall consistency)
  最终一致性

eBay的BASE最佳实践

Randy Shoup，eBay的杰出架构师

- At eBay, we allow absolutely no client-side or distributed transactions of any.
- Of course, we do employ **various techniques** to help the system reach eventual consistency: careful ordering of database operations, asynchronous recovery events, and reconciliation or settlement batches.
- We choose the technique according to the consistency demands of the **particular use case**.

服务模式1: 可查询操作



服务操作的可标识性

- 服务操作具有全局唯一标识
  - 可以使用业务单据号
  - 或者使用系统分配的操作流水号
  - 或者使用操作资源的唯一标识+操作类型的组合
- 操作有唯一的、确定的时间(约定以谁的时间为准)

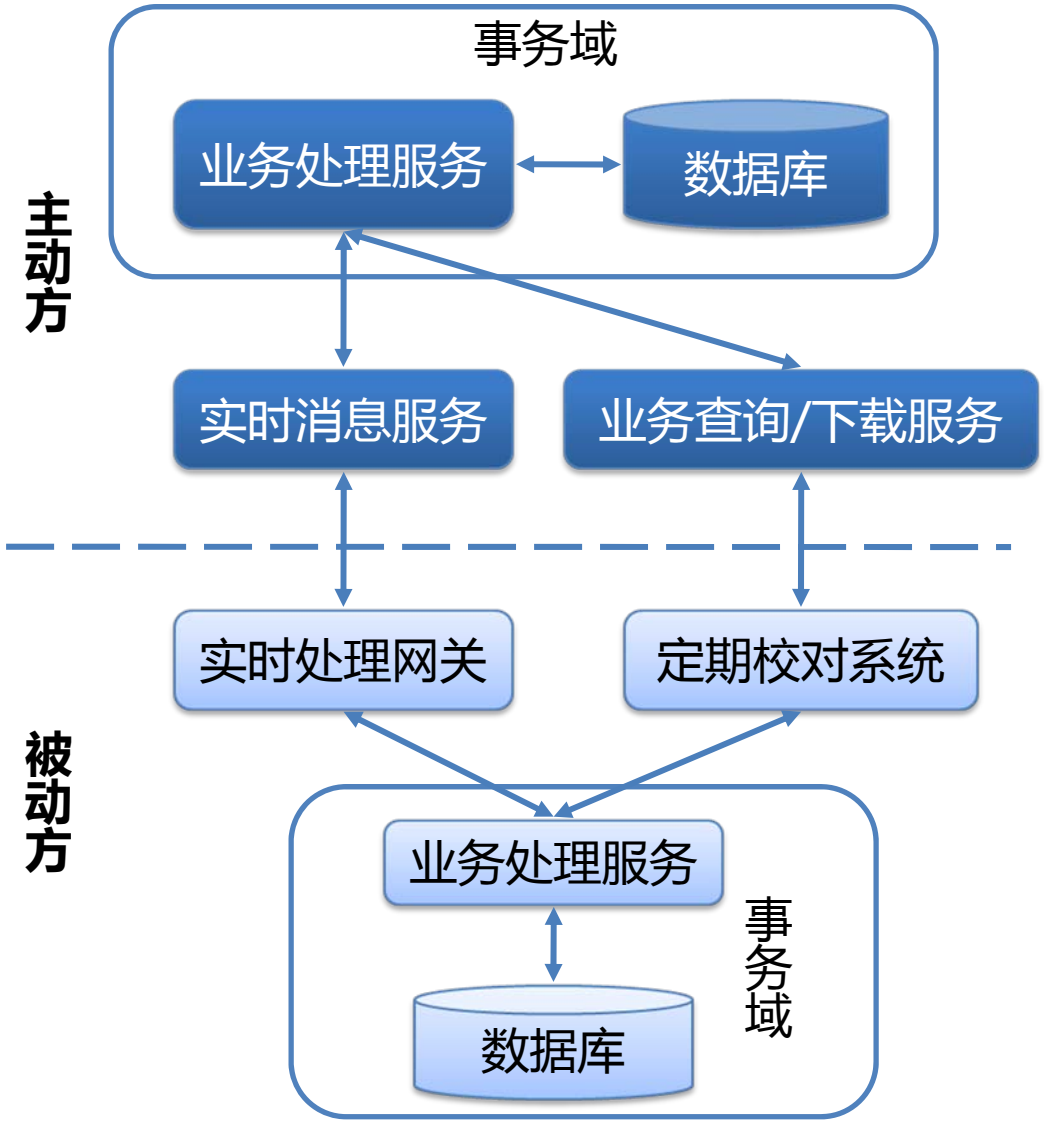
单笔查询

- 使用全局唯一的服务操作标识，查询操作执行结果
- 小心“处理中”状态

批量查询

使用时间区段与(或)一组服务操作的标识，查询一批操作执行结果

复合模式1: 定期校对



实现

- 业务活动的主动方，在完成业务处理之后，向业务活动的被动方发送消息。允许消息丢失。
- 业务活动的被动方根据定时策略，向业务活动主动方查询，恢复丢失的业务消息。

约束

- 被动方的处理结果不影响主动方的处理结果

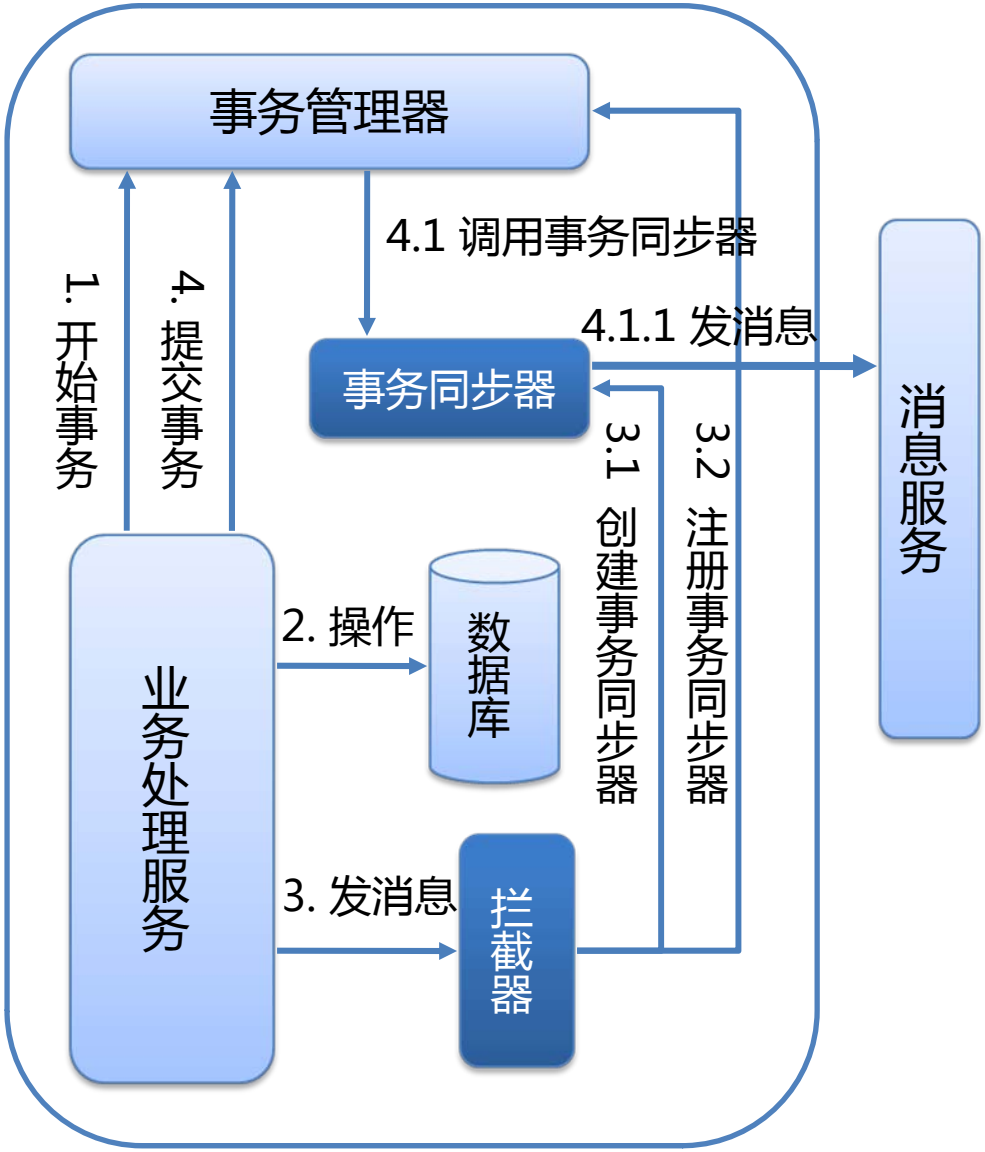
成本

- 业务查询与校对系统的建设成本

适用范围

- 对业务最终一致性的时间敏感度低
- 跨企业的业务活动

保证消息在事务提交后才发送



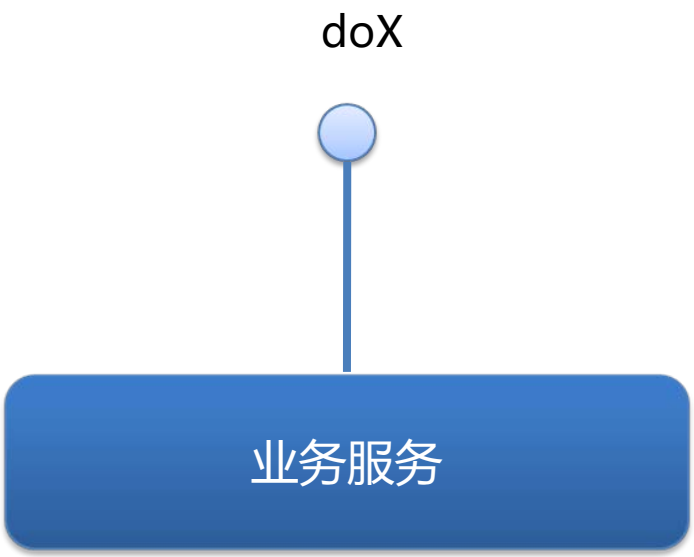
要求

消息发送必须严格在事务提交后方可进行

一种实现方案

- 使用拦截器拦截发送消息请求
- 拦截器检测到当前存在活动事务，就创建一个事务同步器
- 并向事务管理器注册事务同步器
- 业务处理事务完成后，事务管理器会调用事务同步器
- 事务同步器判断当前事务状态为已提交，才真正发送消息

服务模式2: 幂等操作



幂等性(Idempotenty)

f(f(x)) = f(x)

幂等操作

重复调用多次产生的业务结果与调用一次产生的业务结果相同

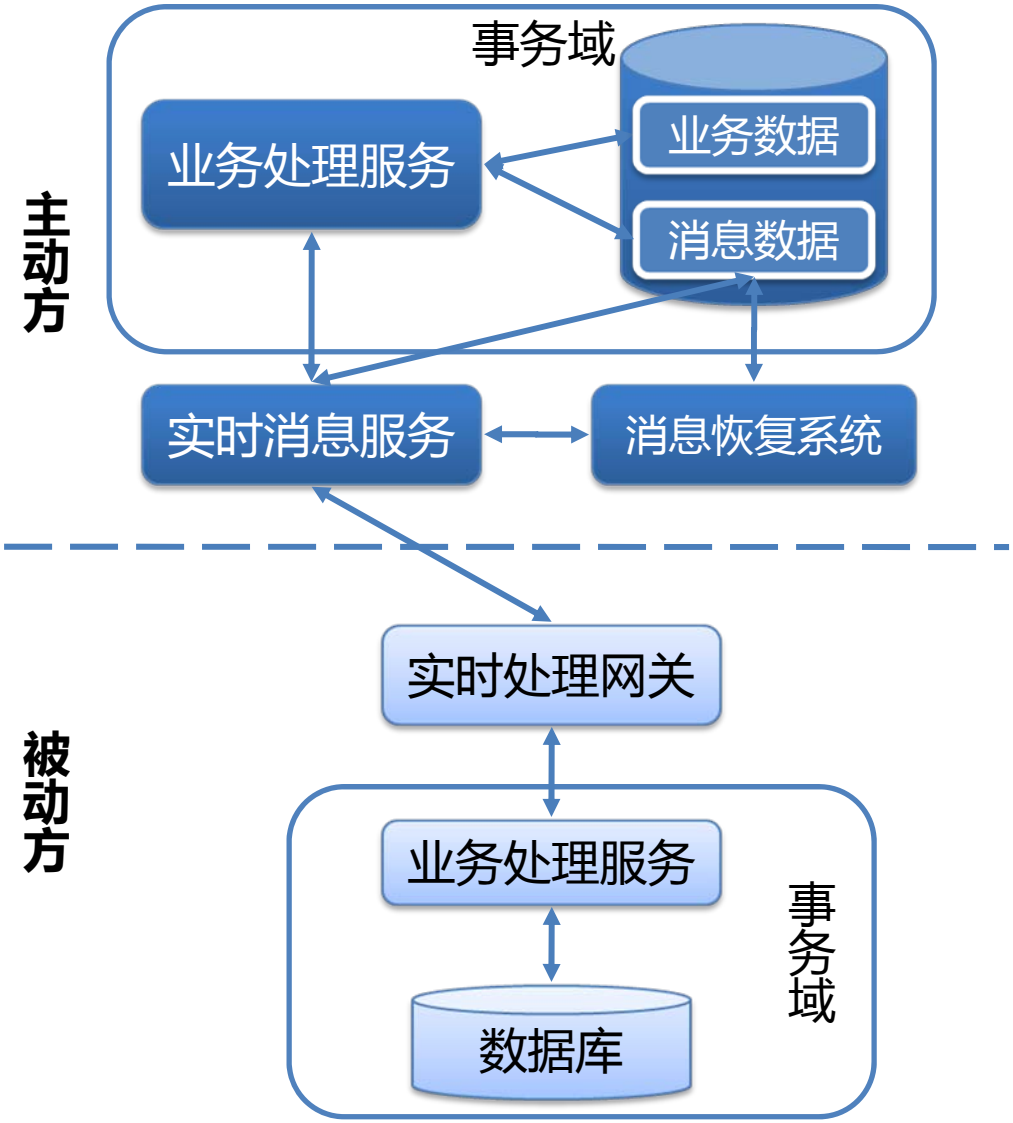
实现方式一

通过业务操作本身实现幂等性

实现方式二

- 系统缓存所有请求与处理结果
- 检测到重复请求之后，自动返回之前的处理结果

复合模式2: 可靠消息



实现

- 业务活动的主动方，在完成业务处理的同一个本地事务中，记录消息数据
- 业务处理事务提交后、通过实时消息服务通知业务被动方，实时通知成功后删除消息数据
- 消息恢复系统定期找到未成功发送的消息，交给实时消息服务补发送

约束

- 被动方的处理结果不影响主动方的处理结果
- 被动方的消息处理操作是幂等操作

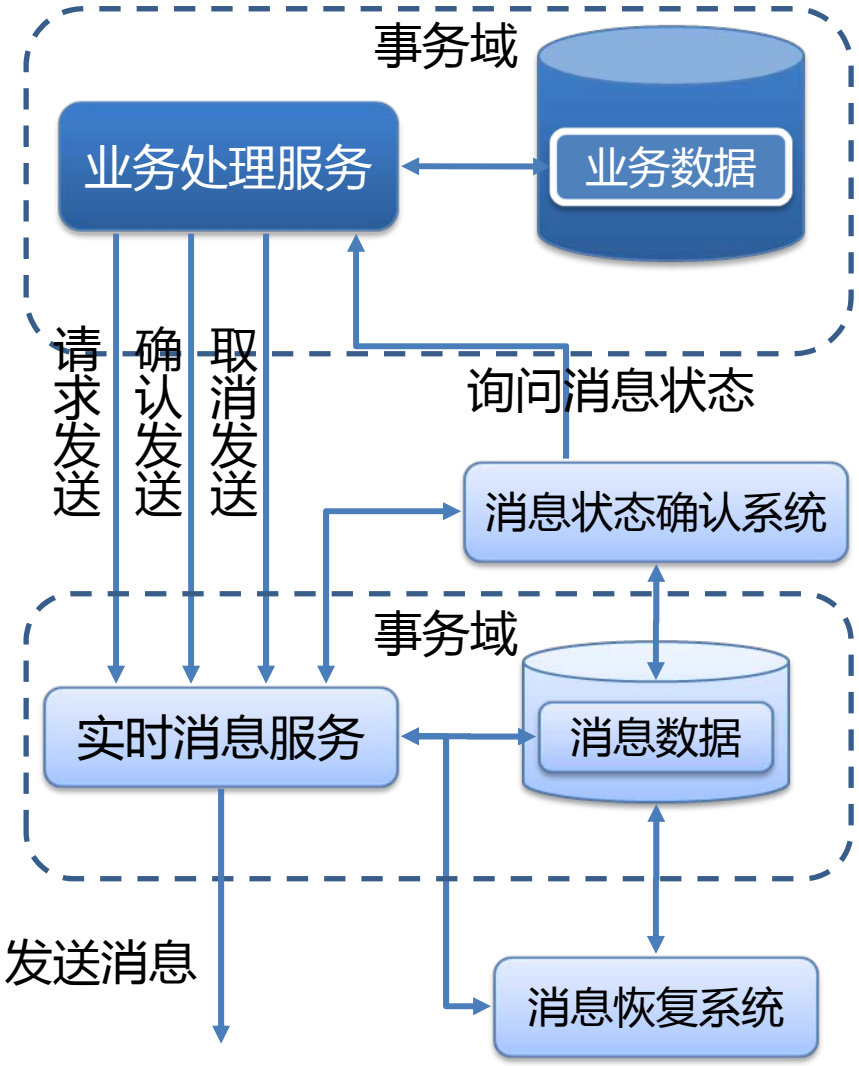
成本

- 可靠消息系统建设成本
- 消息数据CRUD成本

适用范围

- 对最终一致性时间敏感度较高
- 降低业务被动方实现成本

可靠消息的另一种实现



实现

- 业务处理服务在业务事务提交前，向实时消息服务请求发送消息，实时消息服务只记录消息数据，而不真正发送
- 业务处理服务在业务事务提交后，向实时消息服务确认发送。只有在得到确认发送指令后，实时消息服务才真正发送消息
- 业务处理服务在业务事务回滚后，向实时消息服务取消发送
- 消息状态确认系统定期找到未确认发送或回滚发送的消息，向业务处理服务询问消息状态，业务处理服务根据消息ID或消息内容确定该消息是否有效

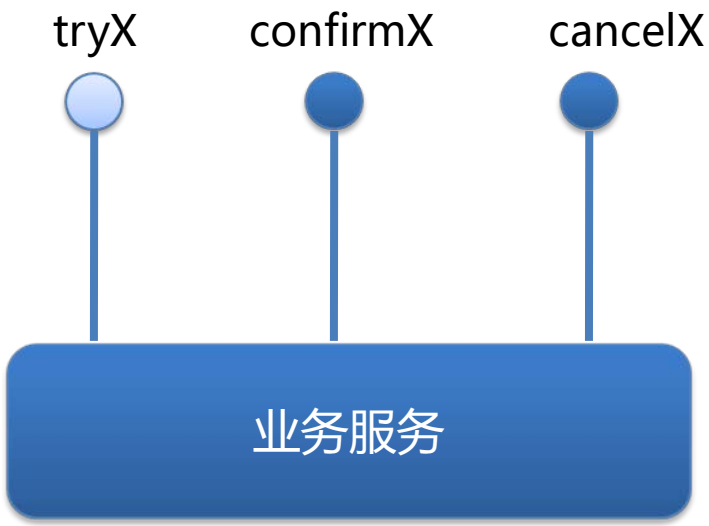
成本

- 一次消息发送需要两次请求
- 业务处理服务需实现消息状态回查接口

优点

- 消息数据独立存储、独立伸缩
- 降低业务系统与消息系统间的耦合

服务模式3: TCC操作



Try: 尝试执行业务

- 完成所有业务检查(一致性)
- 预留必须业务资源(准隔离性)

Confirm:确认执行业务

- 真正执行业务
- 不作任何业务检查
- 只使用Try阶段预留的业务资源
- Confirm操作满足幂等性

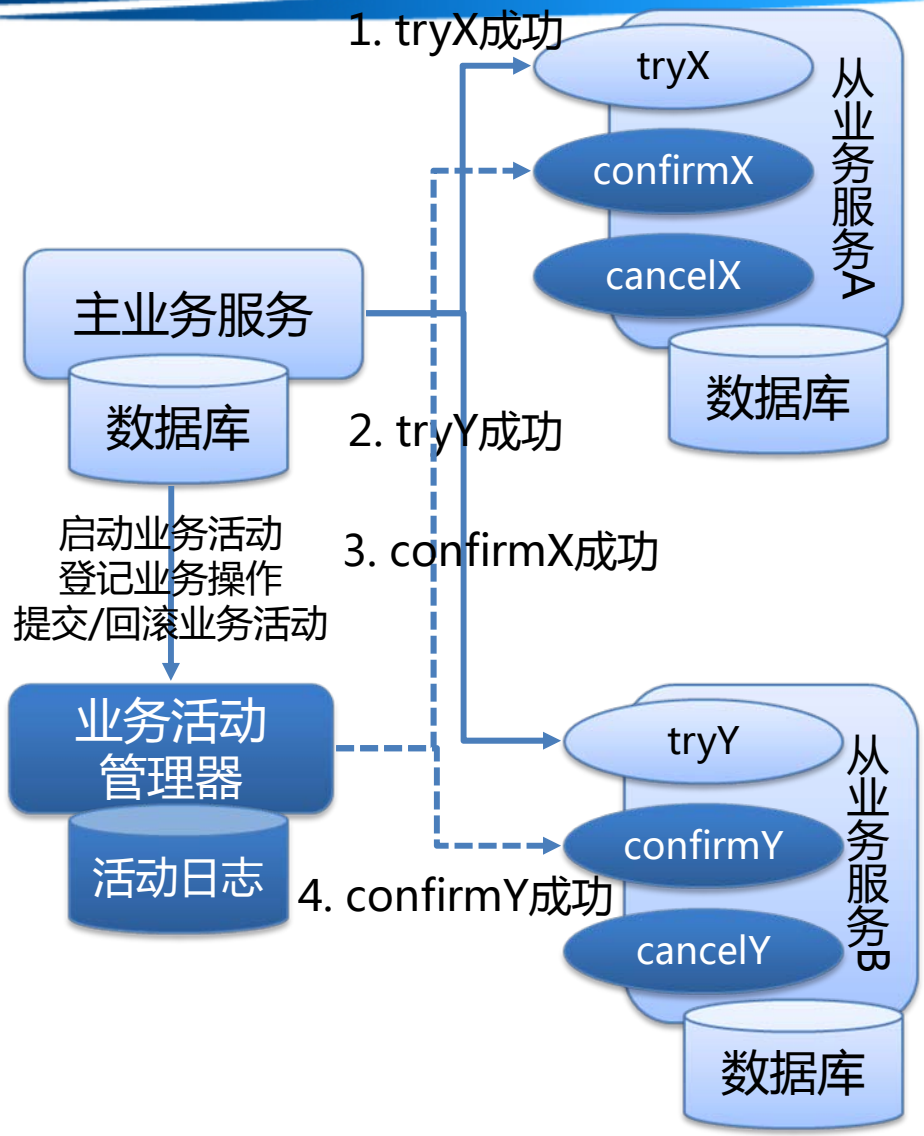
Cancel: 取消执行业务

- 释放Try阶段预留的业务资源
- Cancel操作满足幂等性

与2PC协议比较

- 位于业务服务层而非资源层
- 没有单独的准备(Prepare)阶段，Try操作兼备资源操作与准备能力
- Try操作可以灵活选择业务资源的锁定粒度
- 较高开发成本

复合模式3: TCC模式



实现

- 一个完整的业务活动由一个主业务服务与若干从业务服务组成
- 主业务服务负责发起并完成整个业务活动
- 从业务服务提供TCC型业务操作
- 业务活动管理器控制业务活动的一致性，它登记业务活动中的操作，并在业务活动提交时确认所有的TCC型操作的confirm操作，在业务活动取消时调用所有TCC型操作的cancel操作

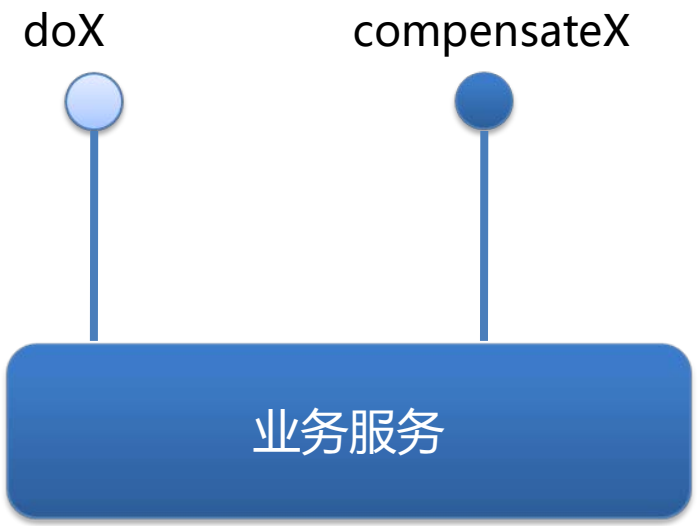
成本

- 实现TCC操作的成本
- 业务活动结束时confirm或cancel操作的执行成本
- 业务活动日志成本

适用范围

- 强隔离性、严格一致性要求的业务活动
- 适用于执行时间较短的业务

服务模式4: 可补偿操作



do: 真正执行业务

- 完成业务处理
- 业务执行结果外部可见

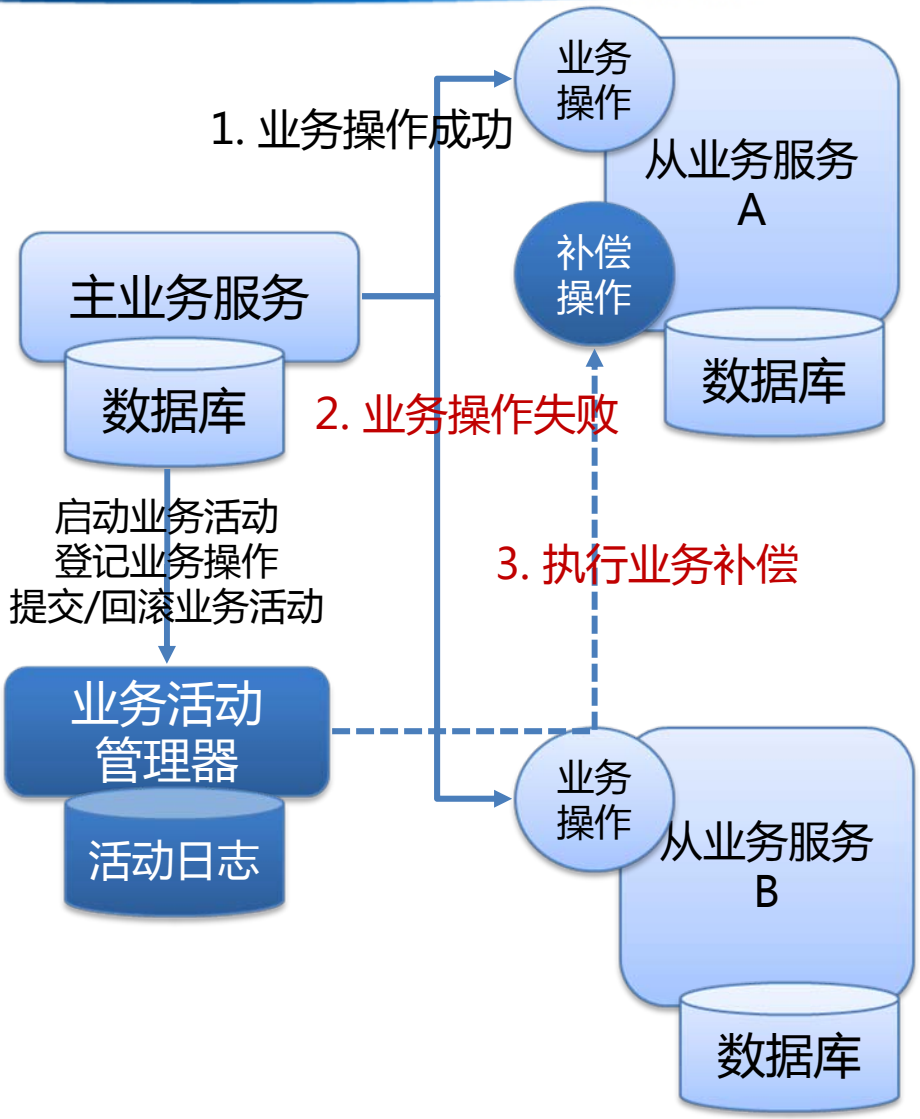
compensate:业务补偿

- 抵销(或部分抵销)正向业务操作的业务结果
- 补偿操作满足幂等性

约束

- 补偿在业务上可行
- 由于业务执行结果未隔离、或者补偿不完整带来的风险与成本可控

复合模式4: 补偿模式



实现

- 一个完整的业务活动由一个主业务服务与若干从业务服务组成，一般由主业务服务发起并结束整个业务活动
- 从业务服务提供的业务操作提供补偿操作，补偿操作可以抵销(或部分抵销)正向业务操作的业务结果
- 业务活动管理器控制业务活动的一致性，它登记业务活动中的操作，并在业务活动取消时调用补偿操作

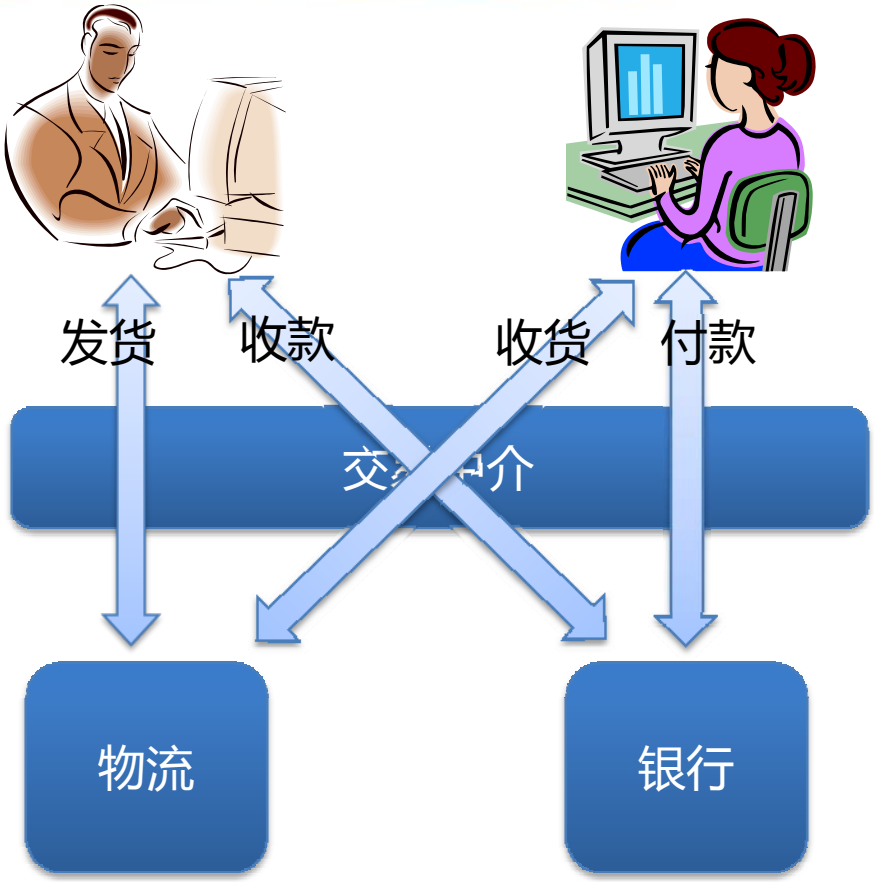
成本

- 从业务服务实现补偿操作的成本
- 由于无法完全补偿带来的业务成本
- 业务活动回滚时，需要额外调用补偿操作

适用范围

- 弱隔离性、弱一致性要求的业务活动
- 特别适用于执行时间较长的业务，如工作流

商业流程也是事务



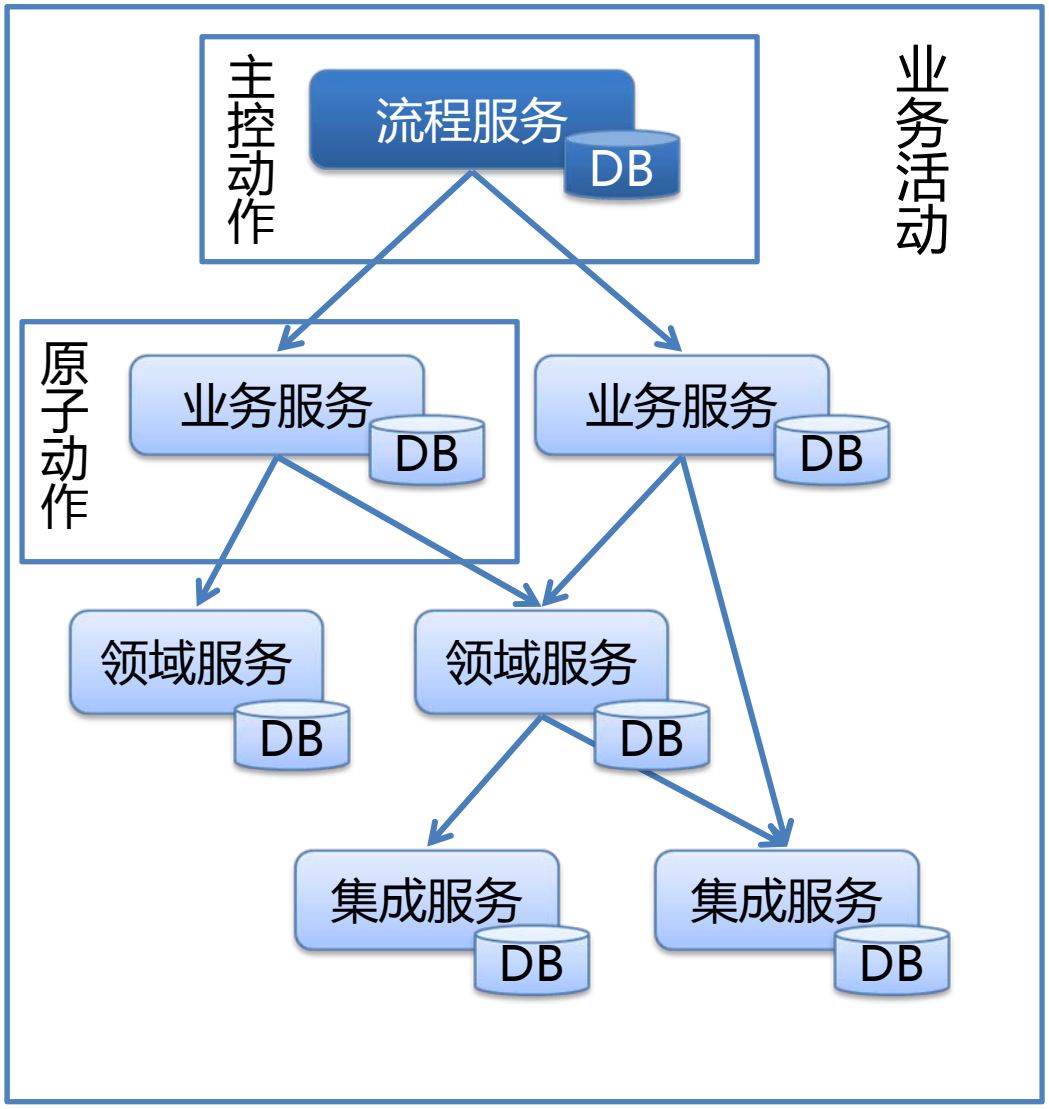
- 事务管理有时是商业流程与用户体验的一部分
- 并非所有问题都可以或应该由系统来解决

又一山寨

框架与设施建设目标

- 一致的架构风格
- 简单的编程模型
- 万无一失
- 没商量: 高可用与可伸缩
- 轻量，可扩展
- 尽力优化性能
- 利用现有基础设施

概念架构



业务活动

一个完整的业务处理过程，处理中包含多个服务操作，这些服务操作构成了一棵调用树

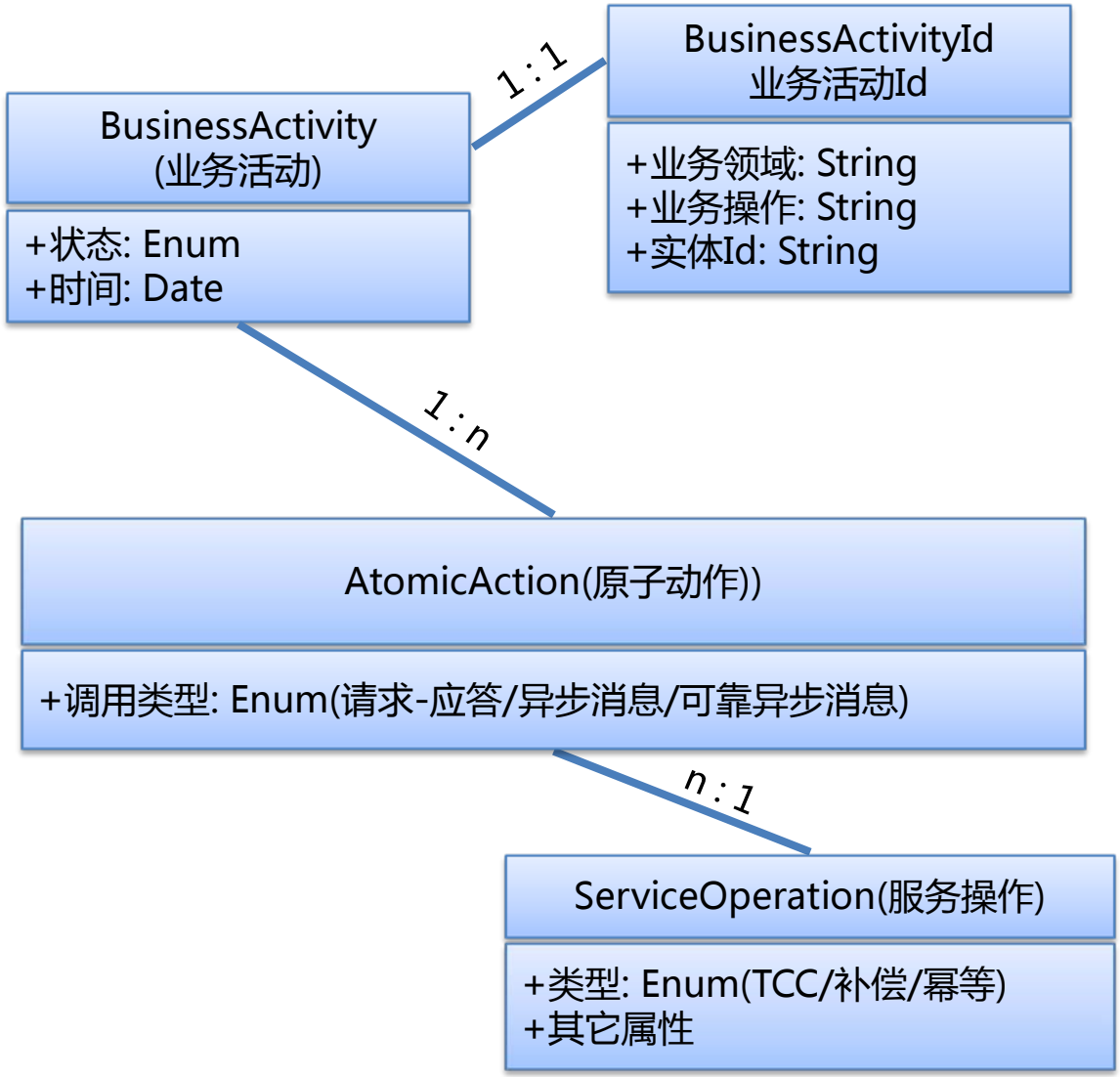
原子动作

- 对应于一次服务操作调用。每个原子动作作为业务活动的基本单元，通过本地事务满足ACID。
- 原子动作有不同的调用类型，如请求-应答、异步消息、异步可靠消息等。
- 原子动作对应的服务操作也有各种类型，如TCC型、可补偿型、幂等型等

主控动作

- 主控动作是整个业务活动的发起者，也是服务操作调用树的根
- 可以合理地让主控动作本身的提交与回滚决定整个业务活动的提交与回滚

业务活动模型



BusinessActivity

业务活动，其中包括的任意多个原子动作需要满足事务要求（ACID/BASE）

AtomicAction

原子动作，是一个业务活动中不可分的业务处理单元。一个原子动作的执行满足ACID。其背后，是通过对某个服务操作的一次调用实现的

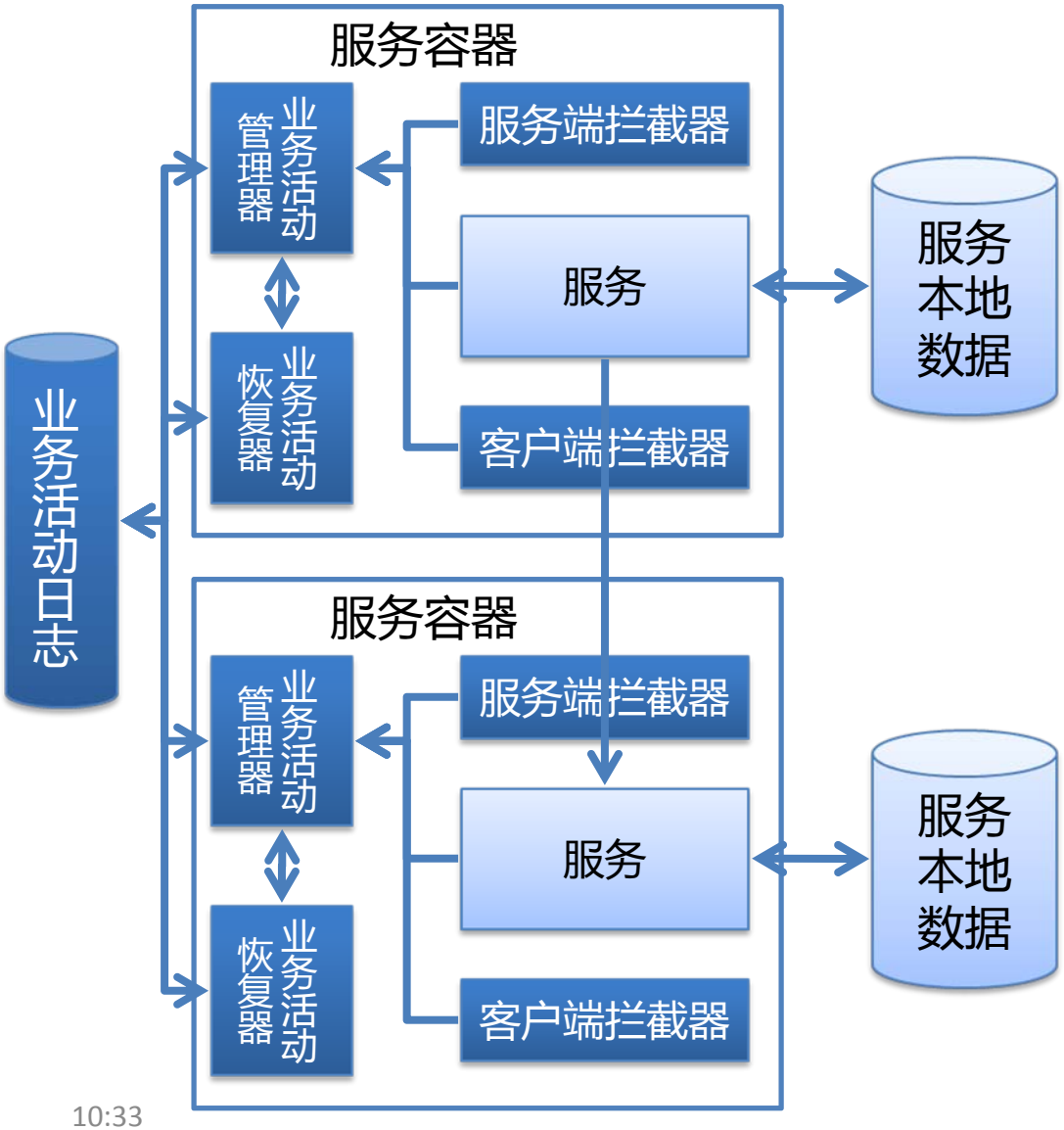
ServiceOperation

服务操作，是某个业务领域功能的原子实现。服务操作有类型标识，表明它是幂等型、TCC型还是补偿型操作

BusinessActivityId

业务活动Id，是业务活动的唯一标识。业务活动Id由三个部分组成，分别代表业务活动所属的业务领域、对应的业务操作与被操作的实体对象的Id

关键组件



业务活动管理器

管理业务活动上下文，操作业务活动日志，协同各个参与者完成业务活动的提交与回滚操作

客户端拦截器

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。在业务活动中添加原子动作

服务端拦截器

拦截服务请求，从消息中析取业务活动上下文，并启动本地业务活动

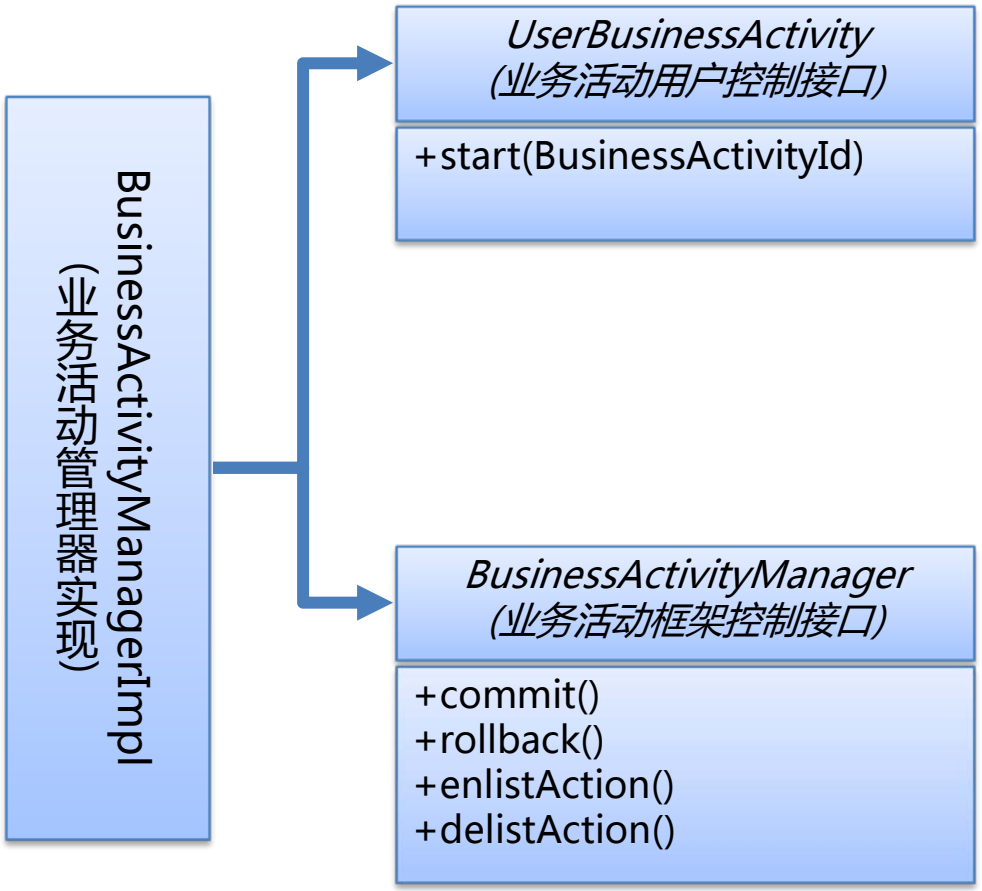
业务活动日志

记录业务活动的状态，记录参与业务活动的原子动作

业务活动恢复器

记录业务活动的状态，记录参与业务活动的原子动作

业务活动管理器(BAM)



UserBusinessActivity接口

面向应用的接口，允许开发者通过本接口启动业务活动，指定业务活动Id。启动业务活动的业务服务成为本次业务活动的**主控业务服务**

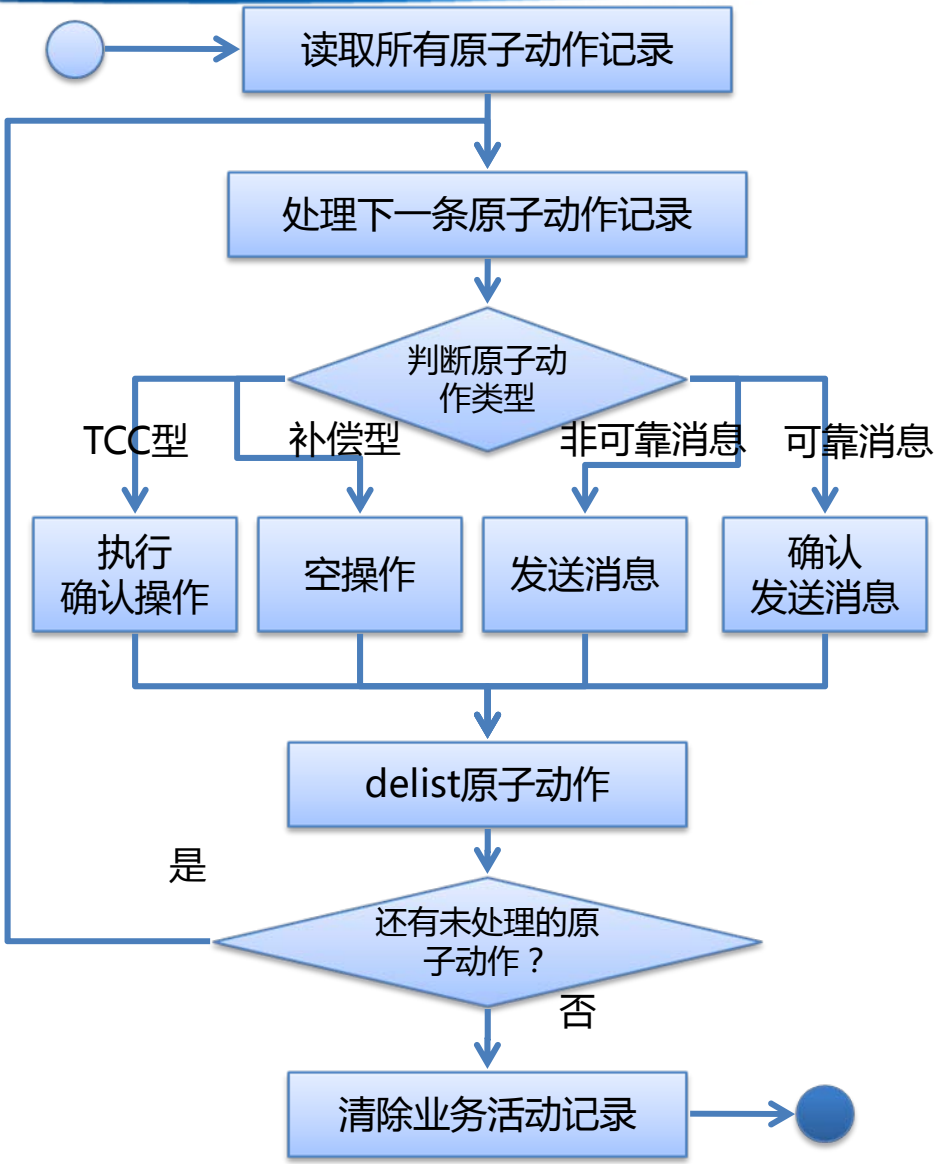
BusinessActivityManager接口

面向框架的接口，由框架实现者提交或回滚业务活动，以及将原子活动作为参与者添加到业务活动的上下文。业务活动的提交或回滚**由主控业务服务本地事务的提交或回滚决定**

BusinessActivityManagerImpl类

上述接口的实现

提交过程

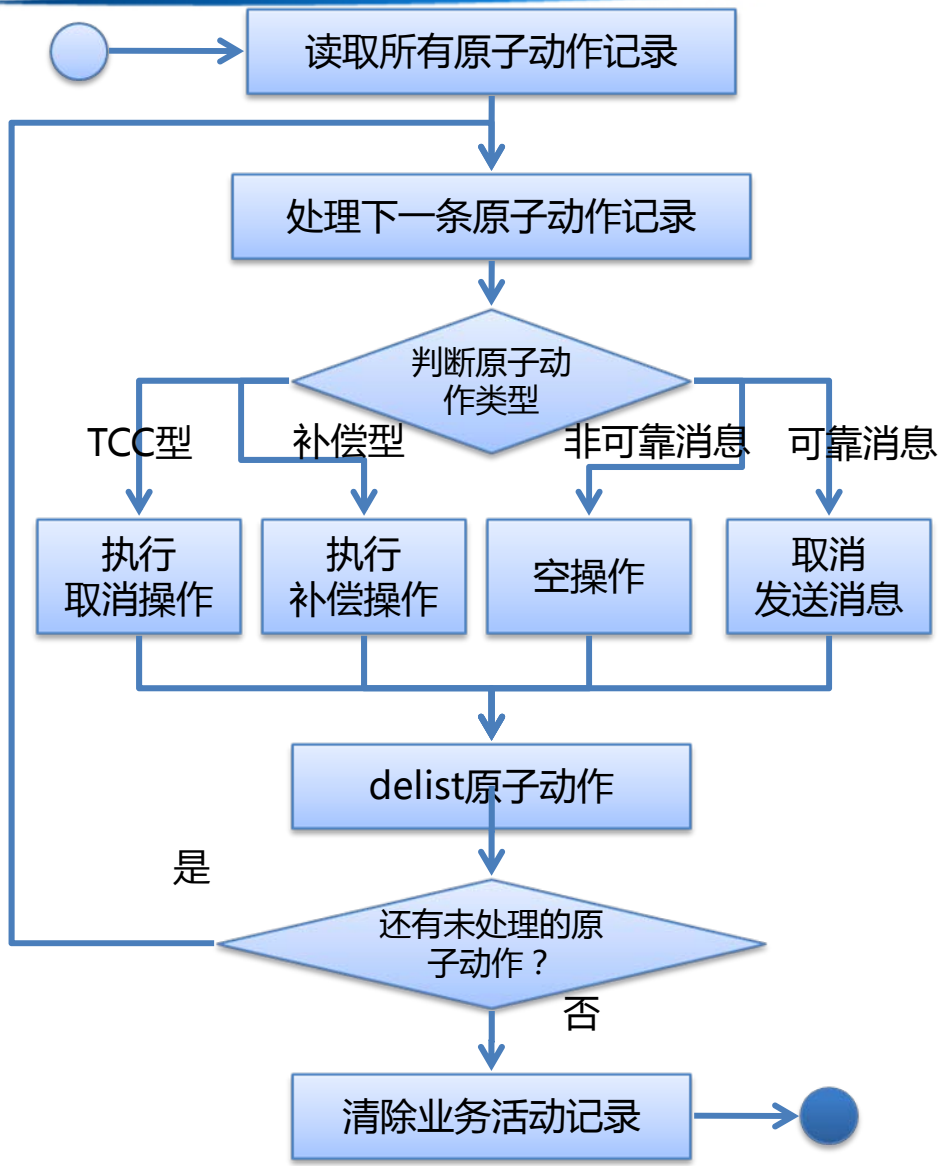


提交过程在主控动作的本地事务提交后，由业务活动管理器执行

实现策略

- 可以通过注册事务同步器监听主控动作的本地事务提交事件
- 提交过程可能时间较长，具体实现时，尽量让这个过程异步化、并行化
- 提交过程中可能发生故障造成处理中断，别担心，由恢复过程进行自动恢复

回滚过程

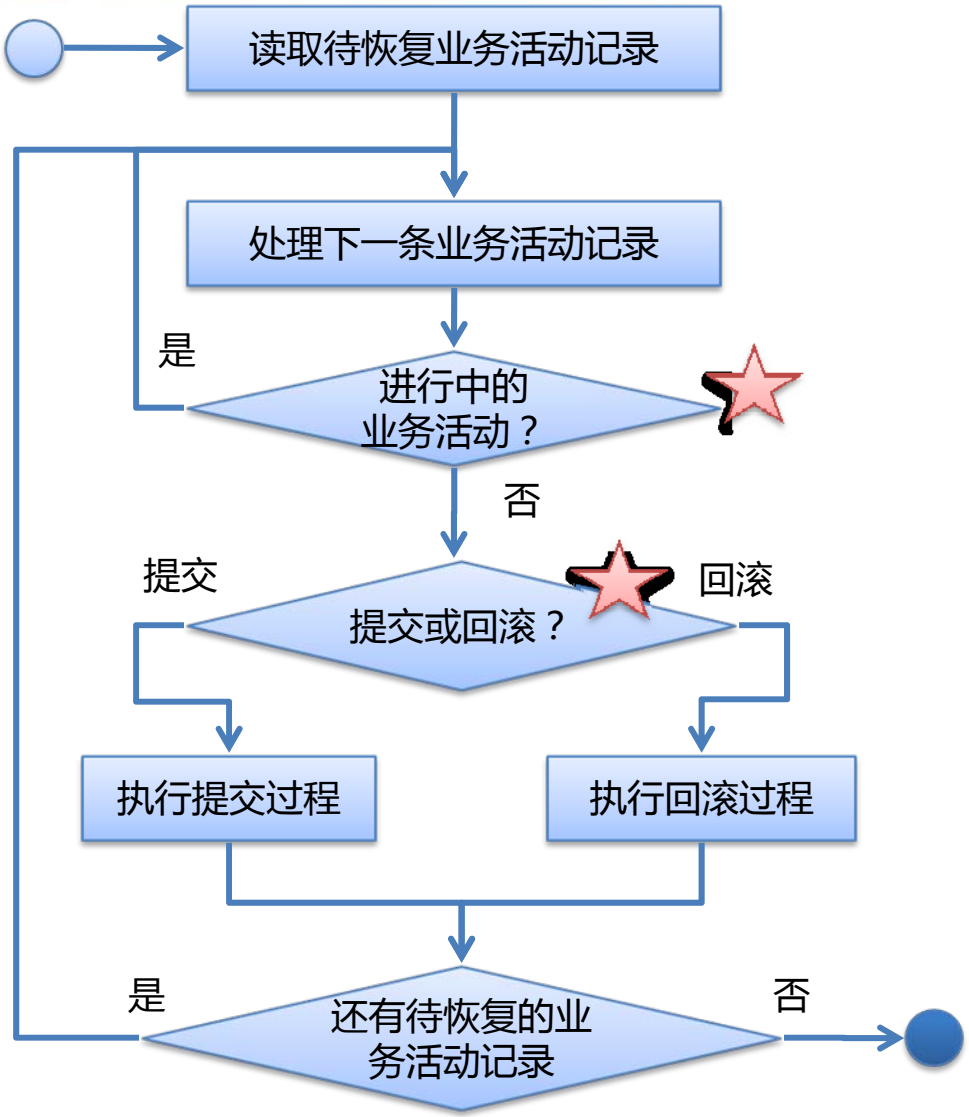


回滚过程在主控动作的本地事务回滚后，由业务活动管理器执行

实现策略

- 可以通过注册事务同步器监听主控动作的本地事务回滚事件
- 回滚过程可能时间较长，具体实现时，尽量让这个过程异步化、并行化
- 回滚过程中可能发生故障造成处理中断，别担心，由恢复过程进行自动恢复

恢复过程



恢复过程定期执行，恢复创建时间早于一定时间内的业务活动记录（如每分钟恢复创建时间在一分钟之前的业务活动记录）

实现策略

- 必须万无一失地判断业务活动是否在进行中
- 必须万无一失地判断业务活动应该提交或回滚
  - 例: 当业务活动日志与主控业务服务处于同一数据库时: 可以采用记录锁判断
  - 例: 当业务活动日志与主控业务服务处于不同数据库时: 向业务服务查询活动状态
- 监控

对基础设施的要求

服务框架

将业务活动管理切面与业务服务功能透明粘合起来，具备事务上下文传输能力

消息系统

支持各种消息质量等级，具备海量吞吐能力，具备灵活优先调度能力

数据存储

消息数据存储: 成本灵活、高性能、可伸缩的消息数据存储

业务活动日志管理: 高可靠、高性能、可伸缩

分布任务调度

可靠地调度系统中的各种任务，如业务活动恢复任务、消息恢复任务、定期校对任务等

服务注册中心

提供服务元数据集中注册、查询与管理能力，支持事务相关属性的描述

Thank you

本文档可以在http://www.dbanotes.net上下载