NX 奈学教育

分布式事务框架Seata深入剖析与应用实践
分布式事务原理篇

主讲人：陈东

2020.7.15

目录

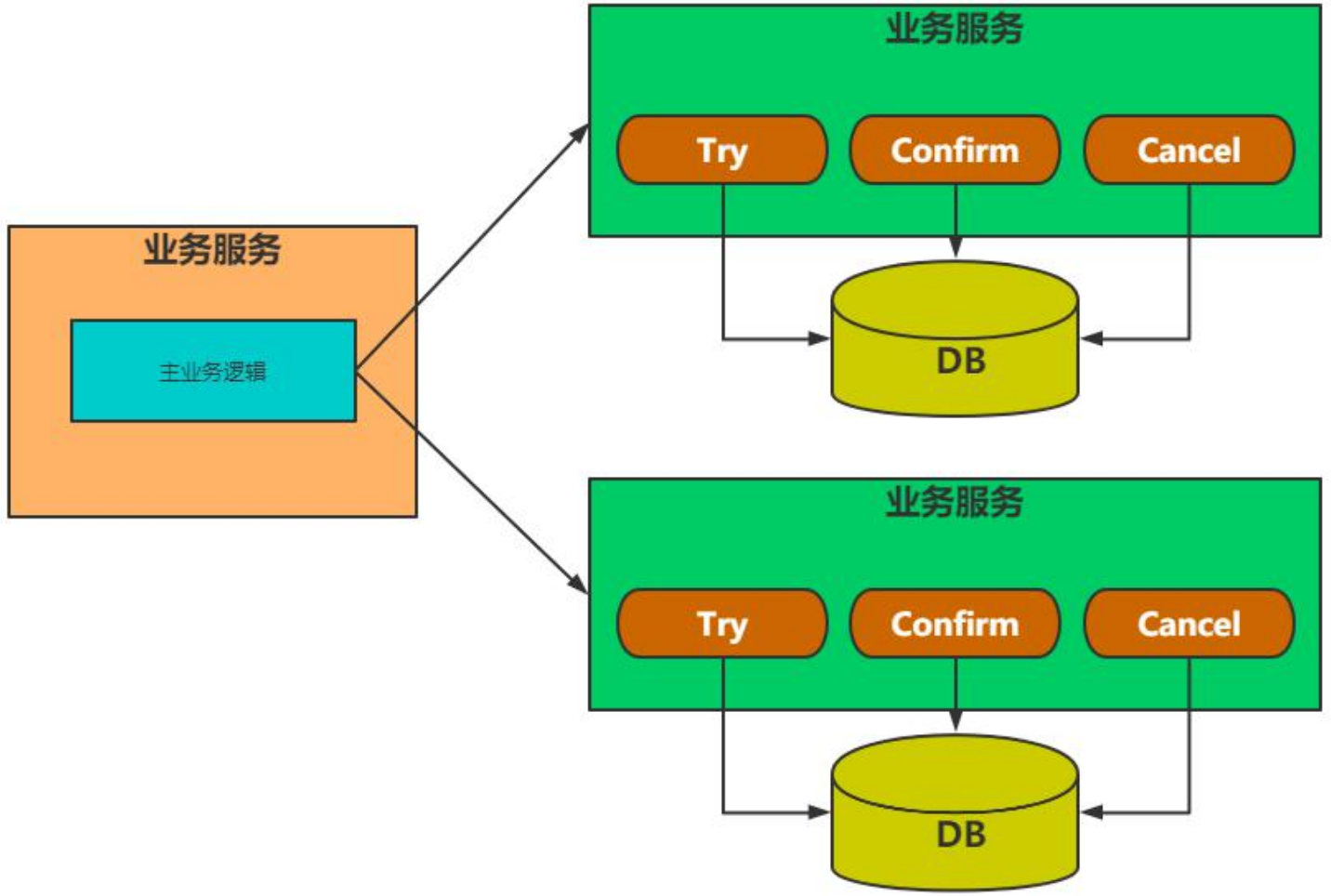
NX 奈学教育

NX 奈学教育

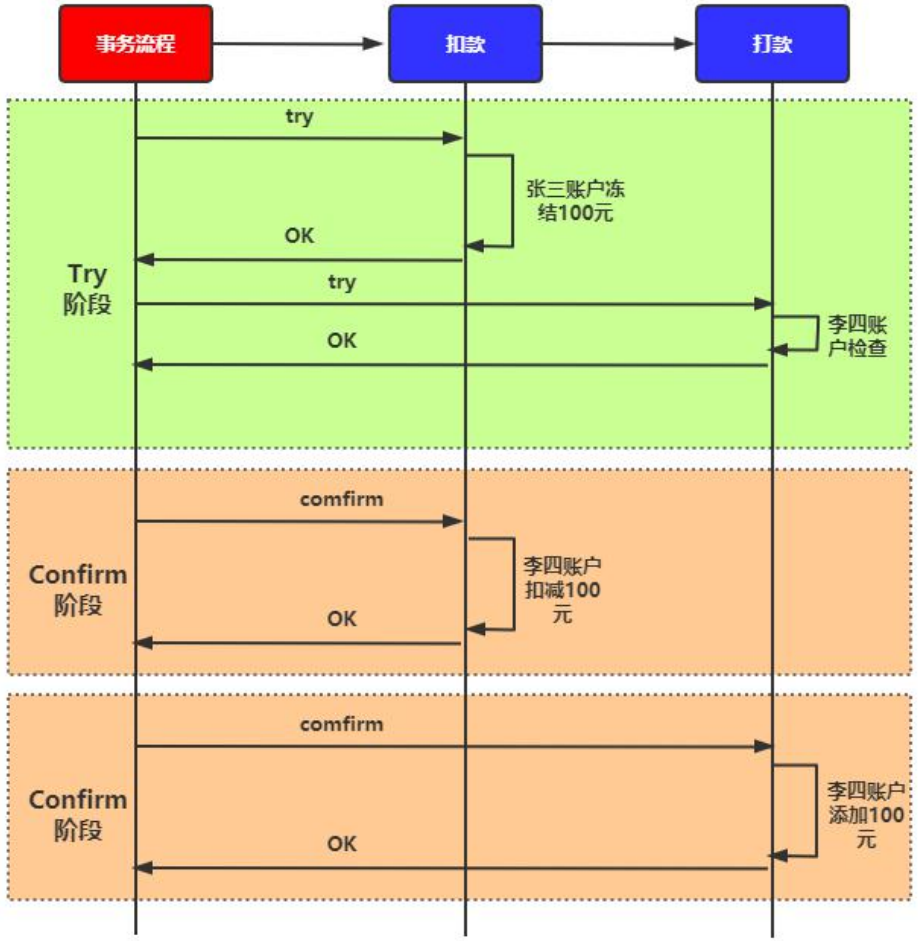
01. Seata TCC模式设计与实现原理剖析

TCC模式

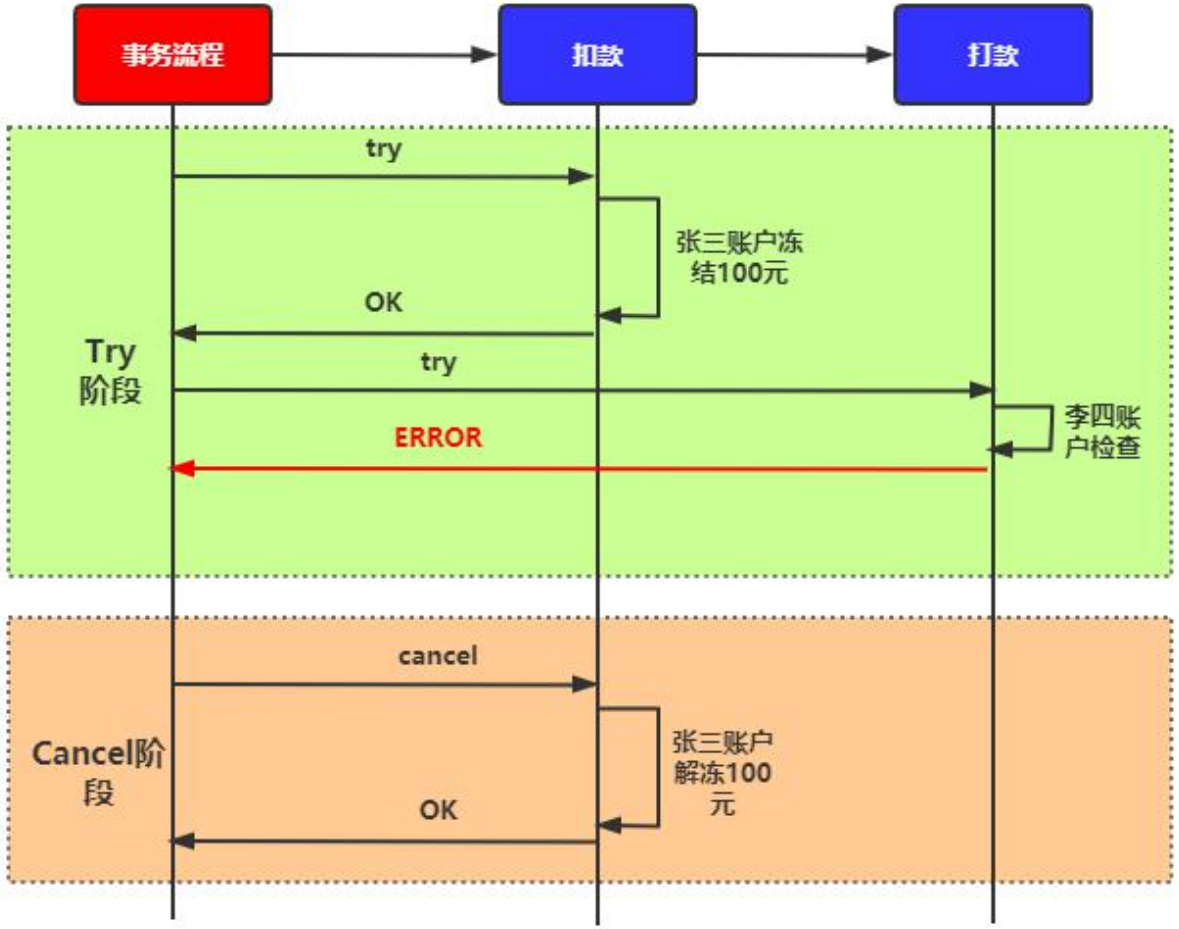
- 传统TCC模型
  - 尝试，预留资源；
  - 确认，使用Try阶段资源；
  - 取消，释放Try阶段预留的资源；



TCC处理流程



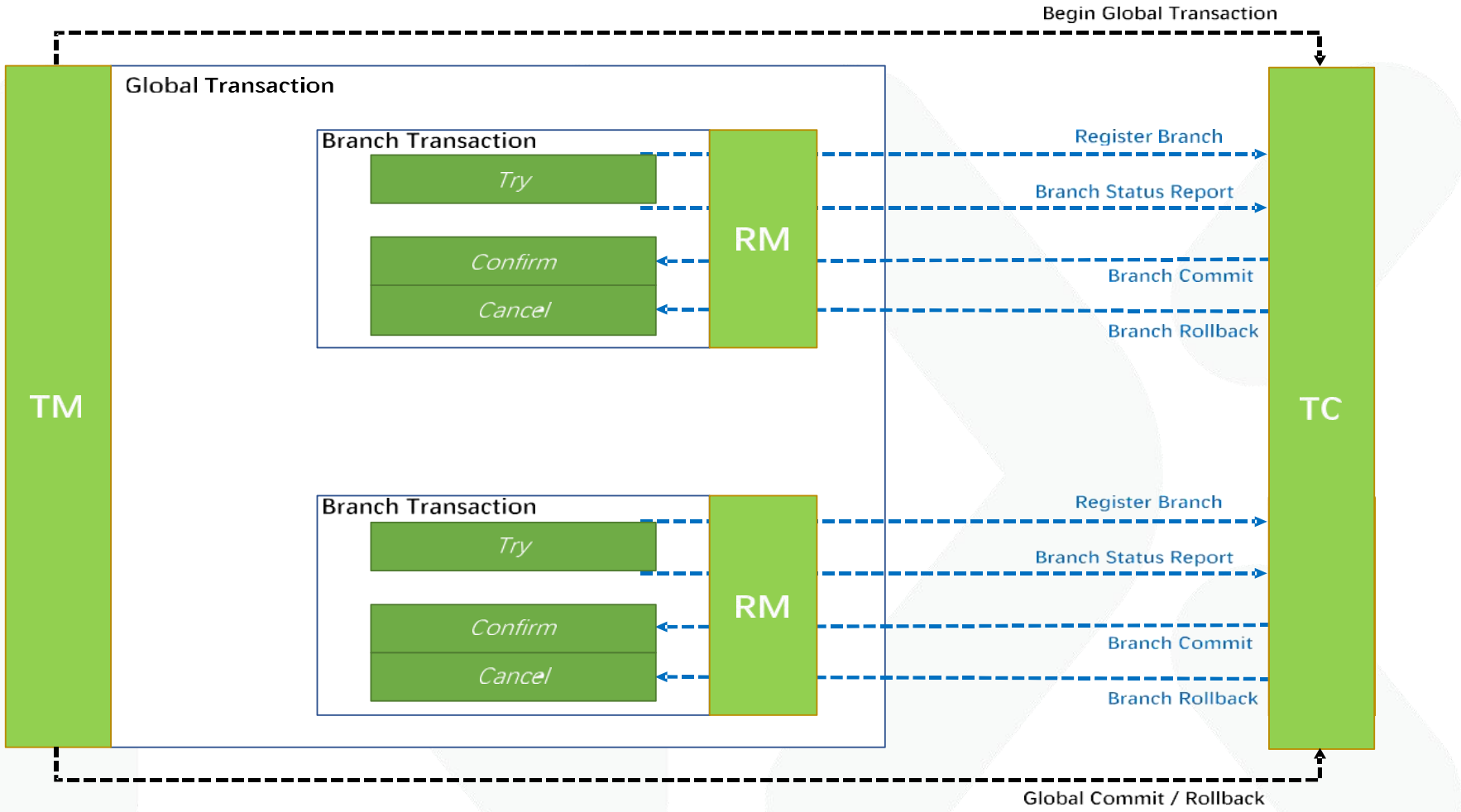
TCC成功流程



TCC失败流程

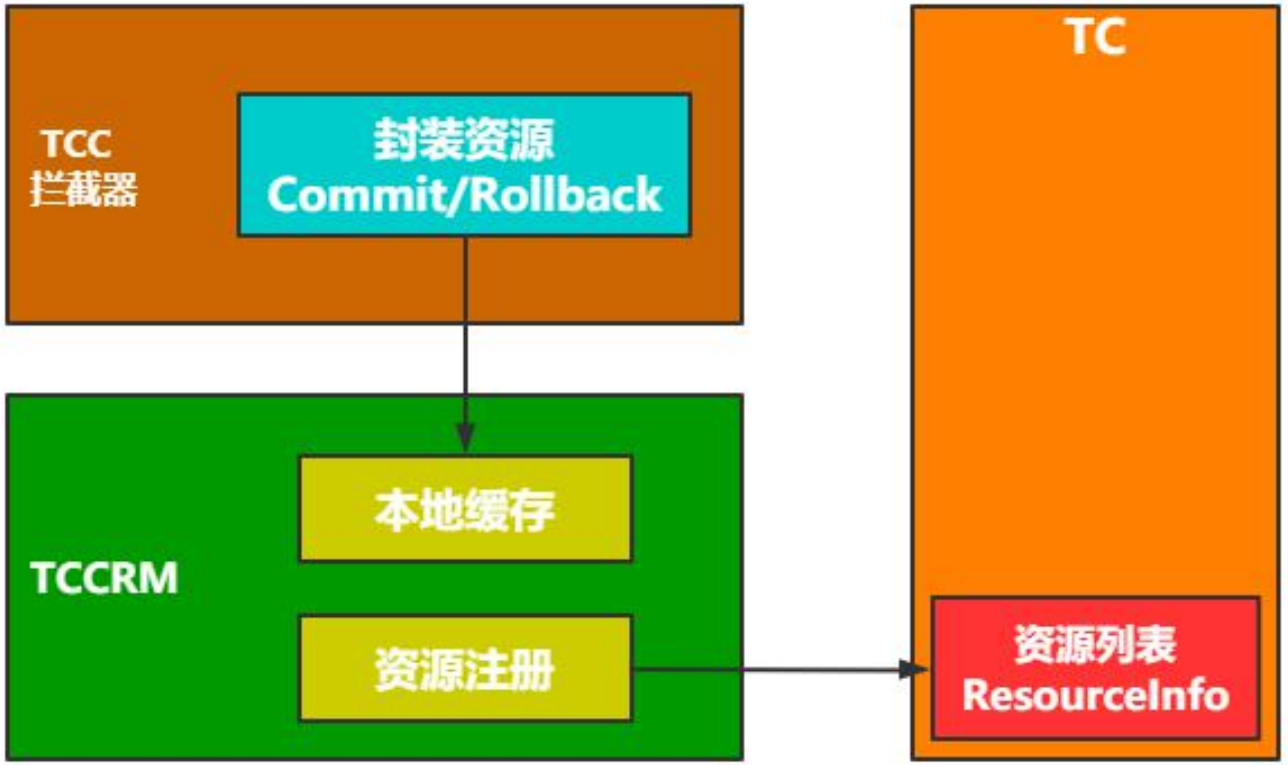
TCC模式

- Seata TCC
  - TM发起全局事务；
  - TC调度提交/回滚；



TCC模式

- Seata TCC实现原理
  - SPI扩展TCCResourceManger；
  - TCCResource，封装Confirm和Cancel方法；
  - 资源本地缓存
  - TC资源列表；

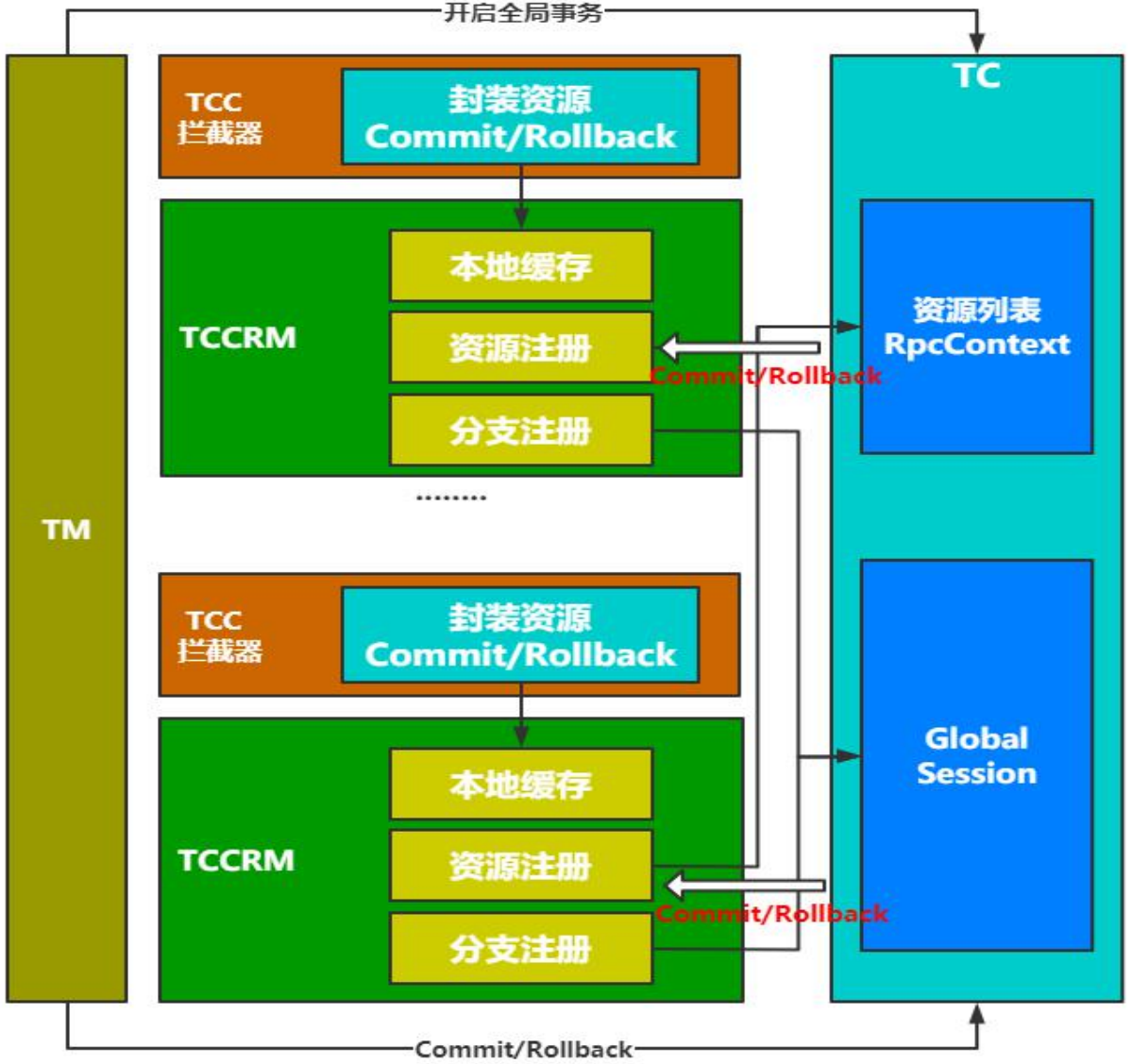


Seata TCC模式设计与实现原理剖析

奈学教育

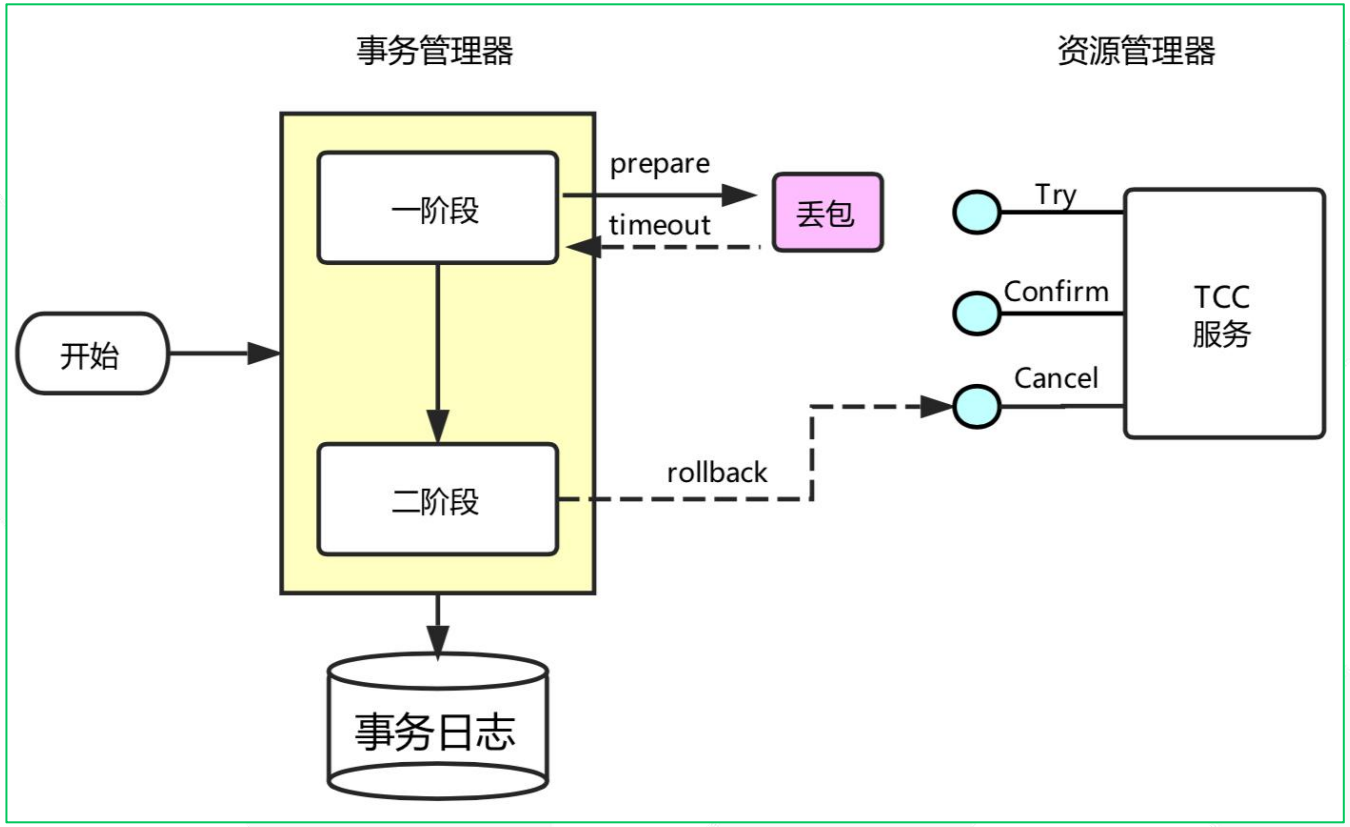
TCC模式

- Seata TCC实现原理
  - 全局事务开启
  - 分支事务中注册
  - 事务提交/回滚



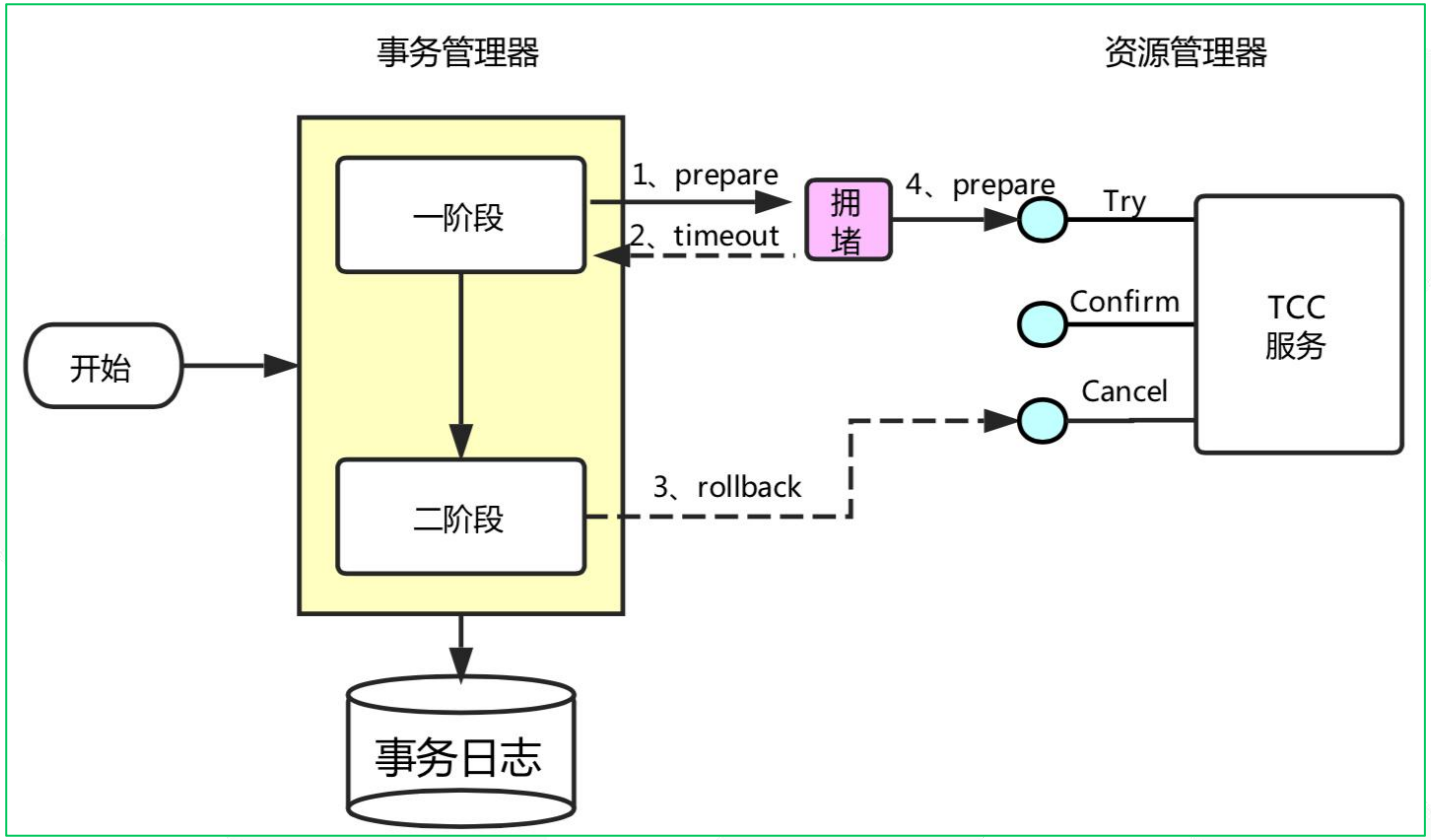
TCC模式-允许空回滚

- 空回滚
  - Try未执行，Cancel执行了；
- 出现原因
  - Try超时(丢包)；
  - 分布式事务回滚，触发Cancel；
  - 未收到Try，收到Cancel；
- 解决
  - 允许空回滚；



TCC模式-防悬挂控制

- 悬挂
  - Cancel 比 Try 先执行；
- 出现原因
  - Try 超时(阻塞)；
  - 分布式事务回滚，触发Cancel；
  - 拥堵的Try 在Cancel之后到达；
- 解决
  - 拒绝空回滚后的Try操作；



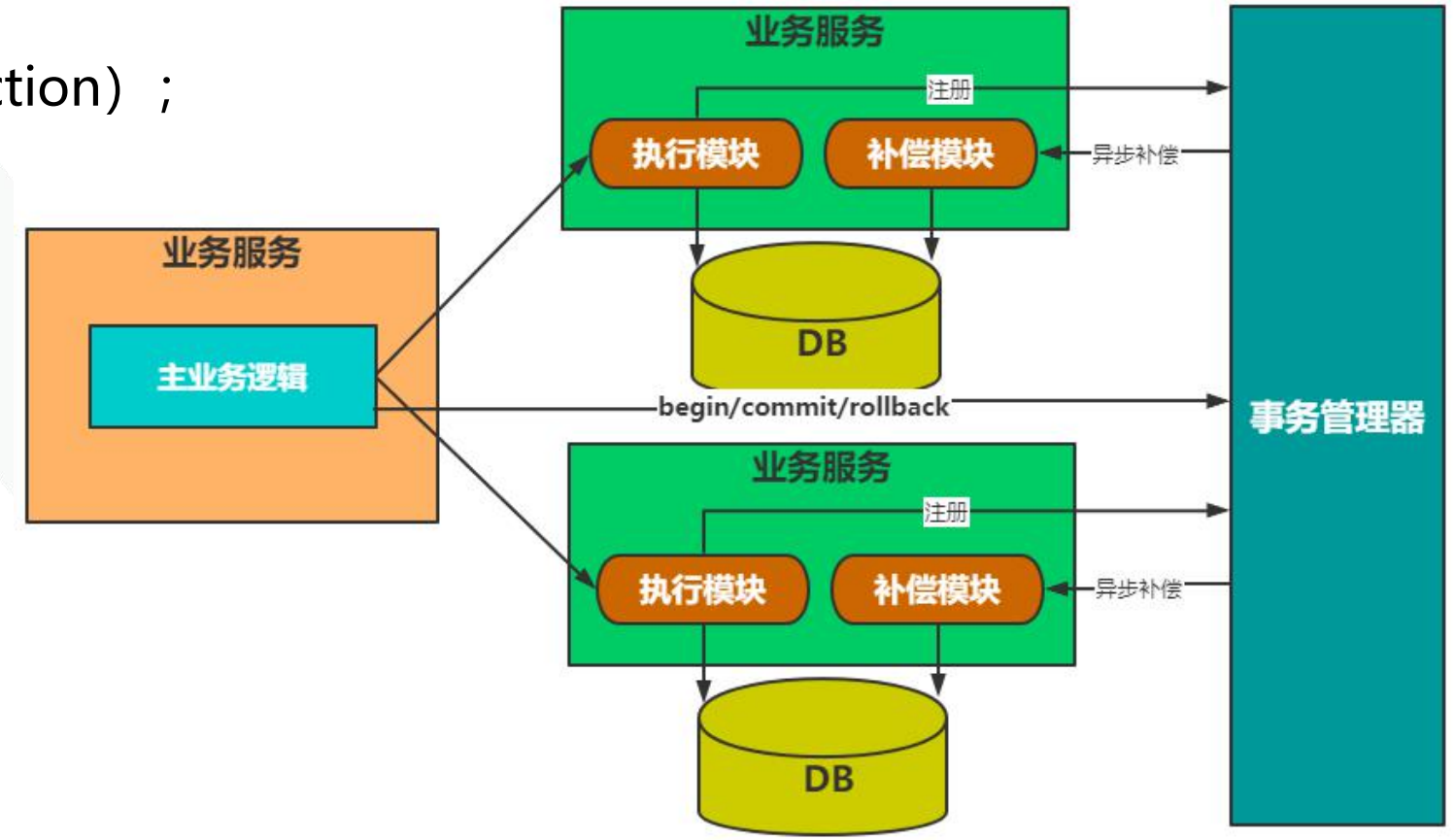
TCC模式-幂等控制

- 现象
  - 超时重试、补偿都会导致 TCC 服务的 Try、Confirm 或者 Cancel 操作被重复执行；
- 解决
  - Confirm、Cancel开发时需要考虑幂等控制；

02. Seata Saga模式设计与实现原理剖析

Saga模式

- 源于1987 Paper Sagas论文；
- 处理长活事务（long lived transaction）；
- 将分布式事务差分成若干本地事务；
- 每个事务有执行模块和补偿模块；



思考：向前恢复/向后恢复

向后恢复：记录事务执行链路

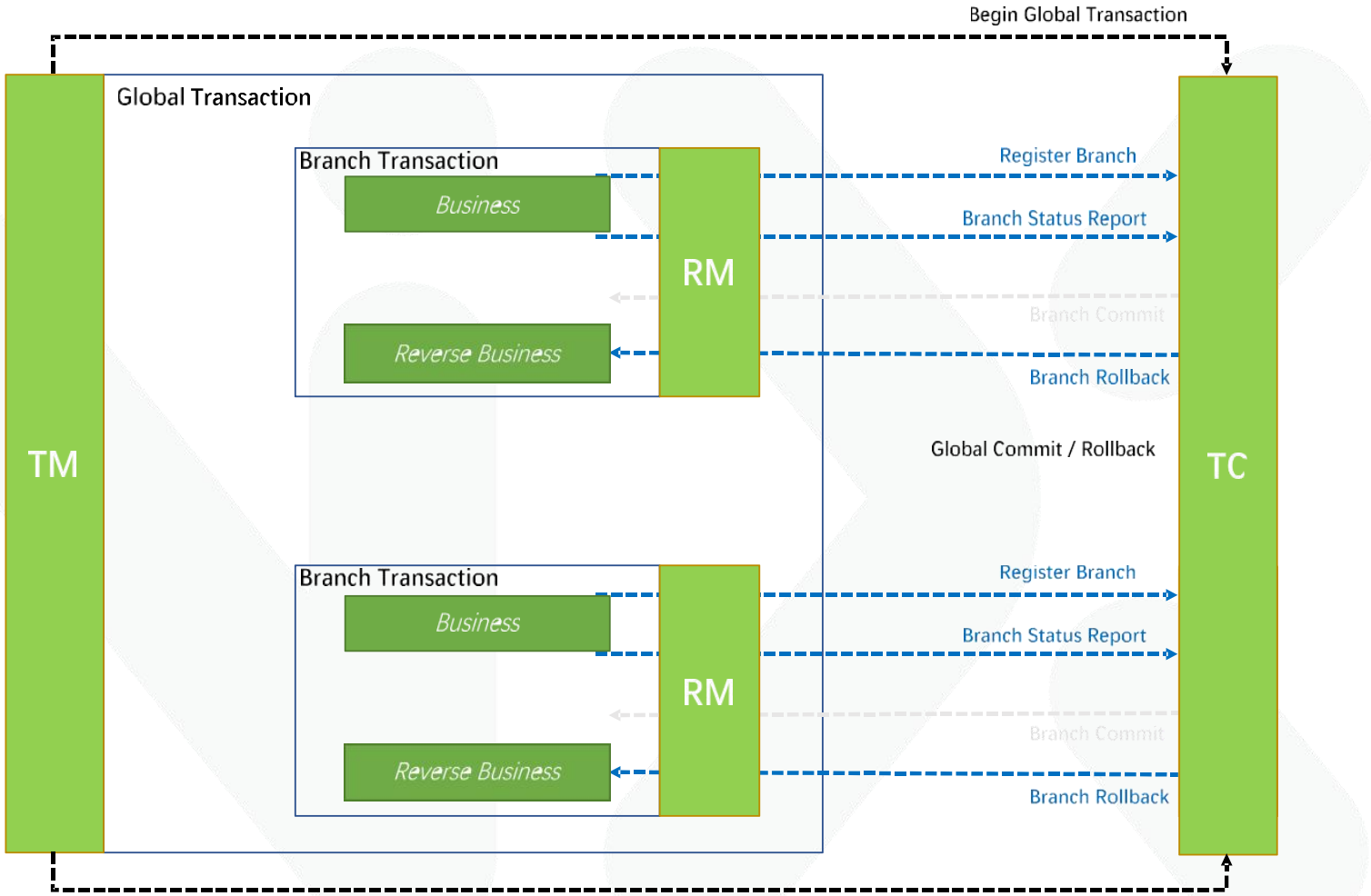
向前恢复：了解全局事务

Saga模式

- 优点：
  - 子事务的服务之间松耦合，少依赖，增删流程节点，不会影响原有服务实现。
  - 子事务专注业务逻辑即可，不需关注数据一致性问题，出错时，由协调器驱动子事务节点，完成补偿。
- 缺点
  - 不保证隔离
  - 业务方保证幂等

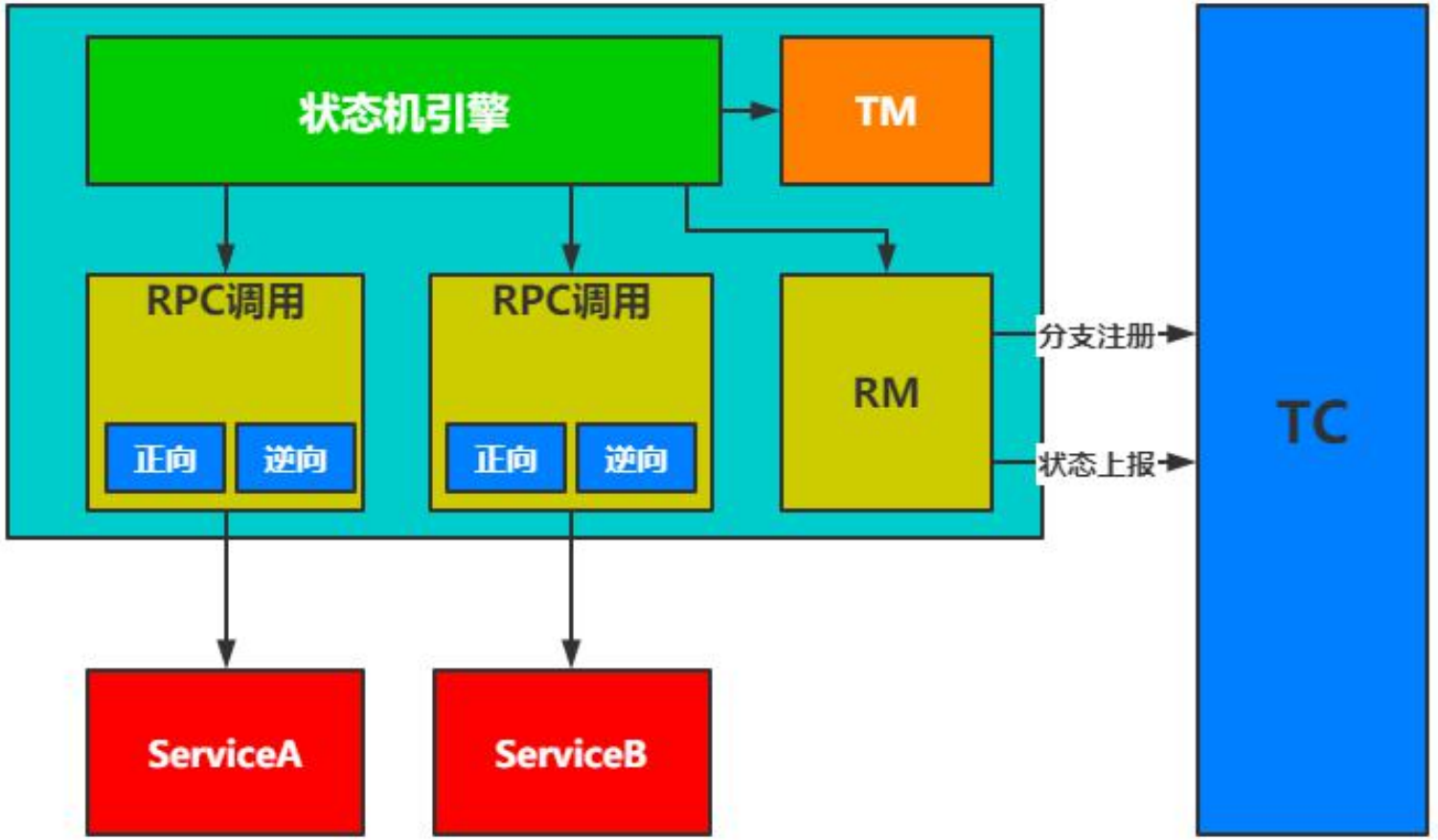
Seate Saga模式

- 每个事务参与者提交“本地事务”；
- 一阶段、补偿由业务开发实现；
- 基于状态机引擎控制事务流转；



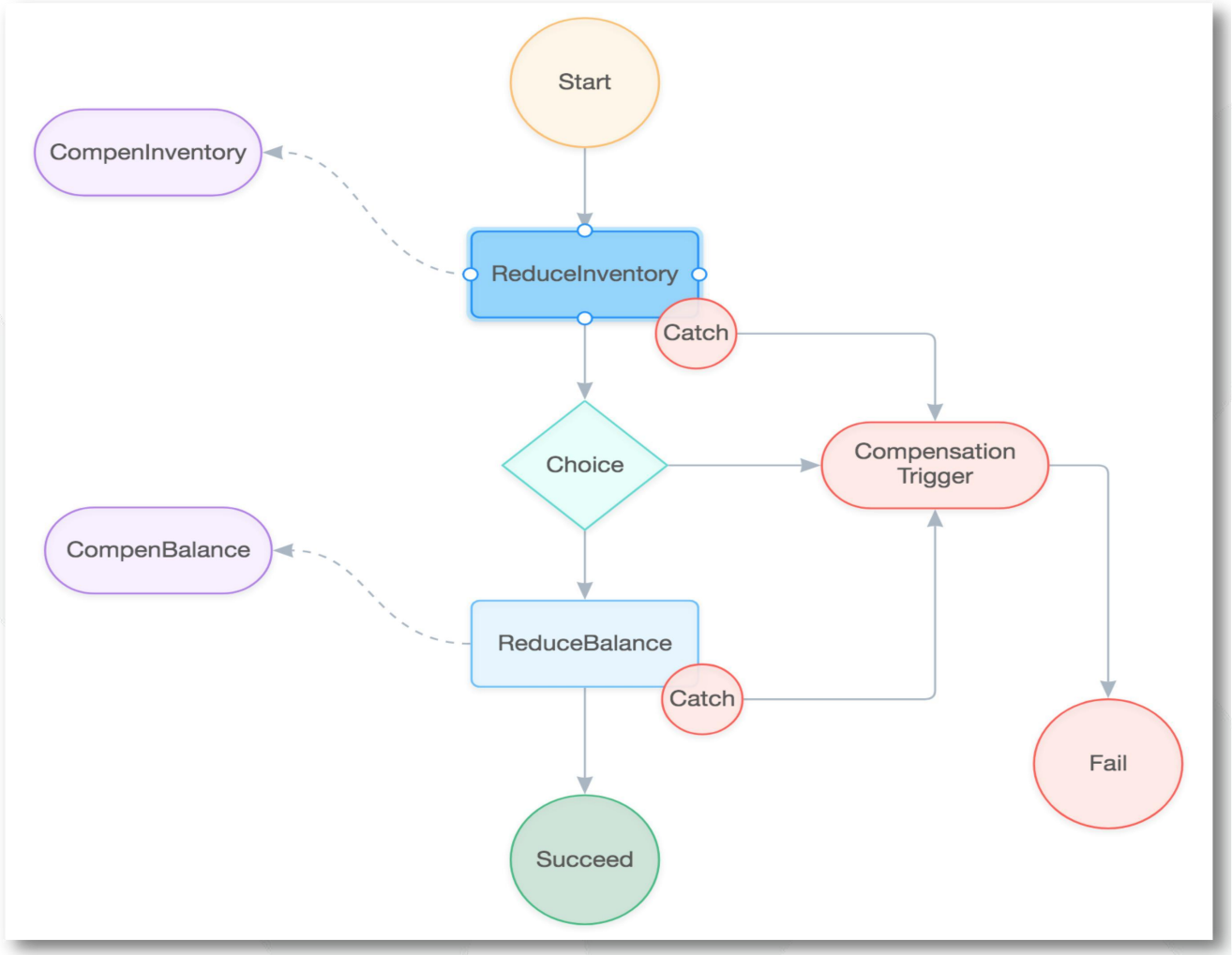
Seate Saga模式

- “事务”不再是数据事务；
- 确保RPC调用的原子性



基本原理

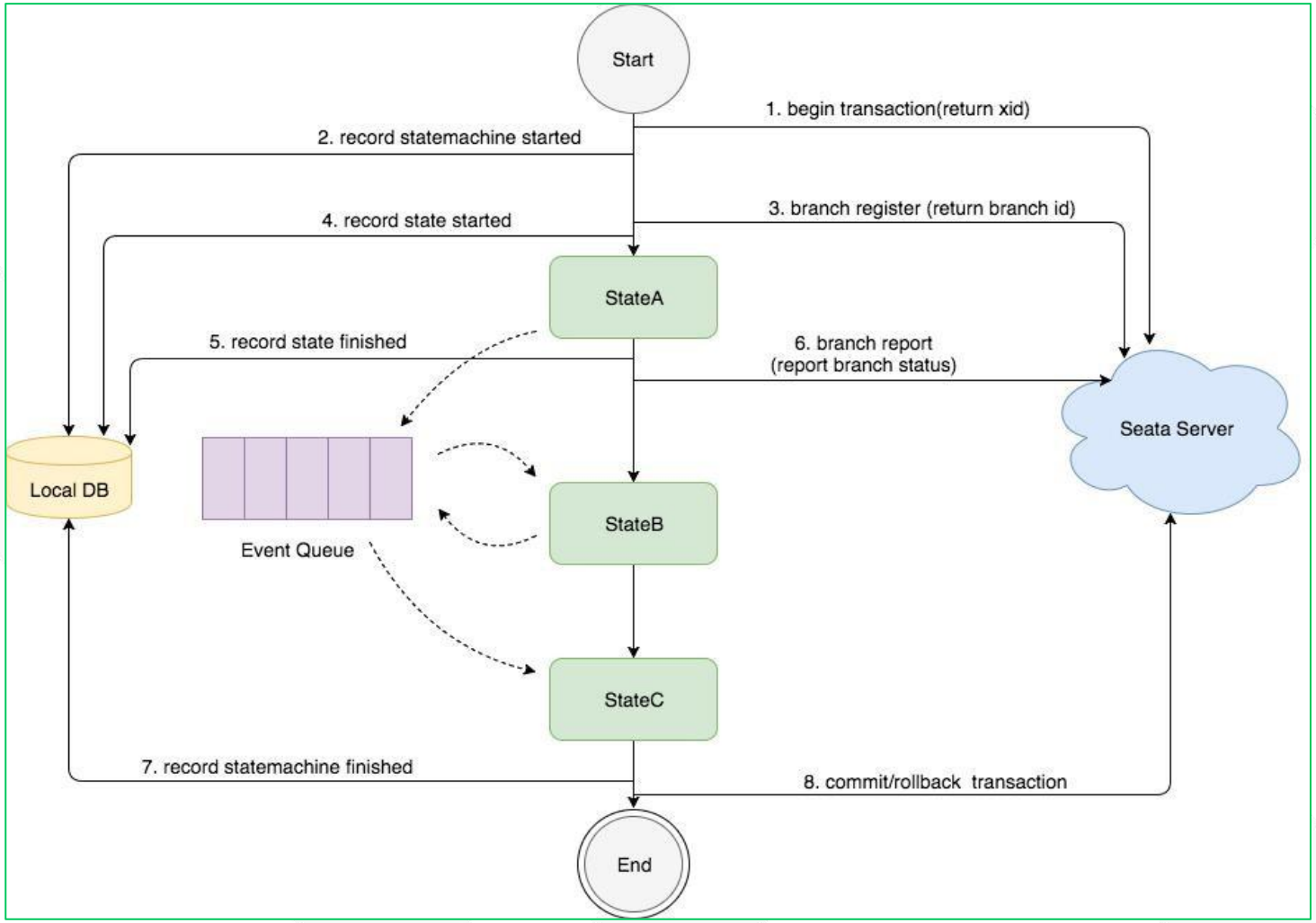
- 基于状态图生成状态语言定义文件；
- 每个状态节点配置补偿节点；
- 状态机引擎驱动执行；



状态机引擎原理

- 基于事件驱动架构，每个节点都可以异步执行，极大提高吞吐量；
- 事务日志存到与业务系统相同的数据库存提高性能；

思考：状态机集成在业务模块中，宕机如何处理？

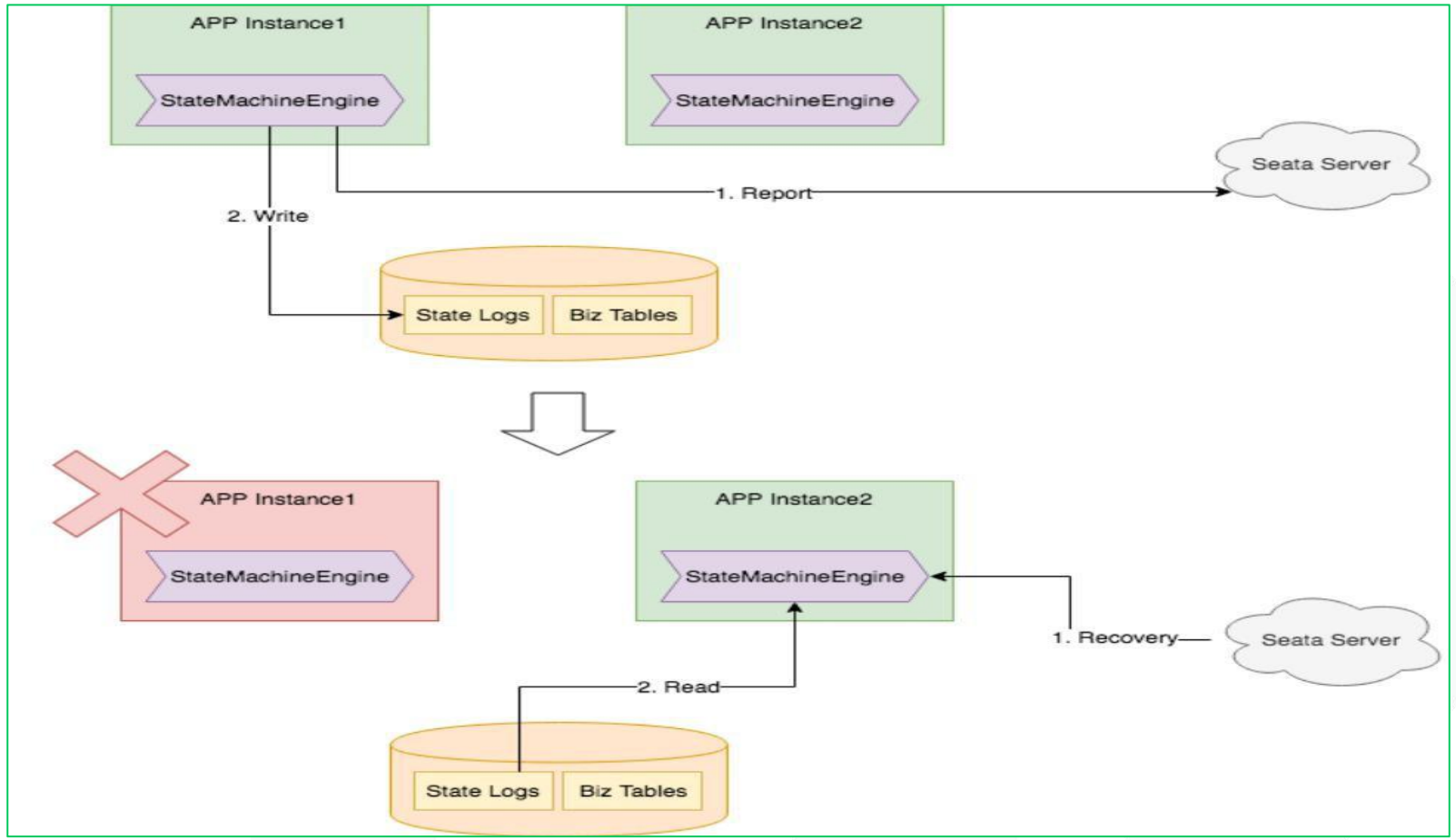


状态机引擎高可用

- 应用正常运行时：
  - 状态机引擎是无状态的；
  - 引擎会上报状态到TC；
  - 状态机日志写入到本地数据库；

- 应用实例宕机时：
  - TC会发送事务恢复请求到 还存活的应用实例；
  - 状态机引擎从数据库装载日志恢复状 态机上下文继续执行；

状态机引擎高可用


APP Instance1
StateMachineEngine
APP Instance2
StateMachineEngine
Seata Server
1. Report
2. Write
State Logs
Biz Tables
APP Instance1
StateMachineEngine
APP Instance2
StateMachineEngine
1. Recovery
Seata Server
2. Read
State Logs
Biz Tables

Saga实现对比

模式	优点	缺点
状态机引擎	➢ 可以用可视化工具来定义业务流程，标准化，可读性高 ➢ 提高业务分析人员与程序开发人员的沟通效率 ➢ 业务状态管理：流程本质是状态机，可反映业务状态的流转 ➢ 提高异常处理灵活性：可以实现事后恢复“ 向前重试” 或“ 向后补偿”	➢ 业务流程实际是由 JAVA 程序与 DSL 配置组成，程序与配置分离，开发起来比较繁琐 ➢ 如果是改造现有业务，对业务侵入性高 ➢ 状态机引擎实现成本高
AOP Proxy	➢ 程序与注解是在一起的，开发简单，学习成本低 ➢ 方便接入现有业务 ➢ 基于动态代理拦截器，框架实现成本低	➢ 框架无法提供业务状态管理 ➢ 难以实现事后恢复“ 向前重试”

Saga状态机编排对比

模式	优点	缺点
集中式	➢ 子事务的服务之间松耦合，少依赖，增删流程节点，不会影响原有服务实现。 ➢ 子事务专注业务逻辑即可，不需关注数据一致性问题，出错时，由协调器驱动子事务节点，完成补偿。	➢ 强依赖TC，要求TC具有高可用、高可靠等特性
协同式	➢ 具有服务自治的优势	➢ 子事务之间紧耦合，已有流程中增删服务成本巨大，且有循环依赖 ➢ 子事务需要处理数据一致性，增加额外的业务复杂性 ➢ 随着服务数量增加，服务之间的关系难以理解，如右图死星所示

NX 奈学教育

03. Seata XA模式设计与实现原理剖析

XA模式

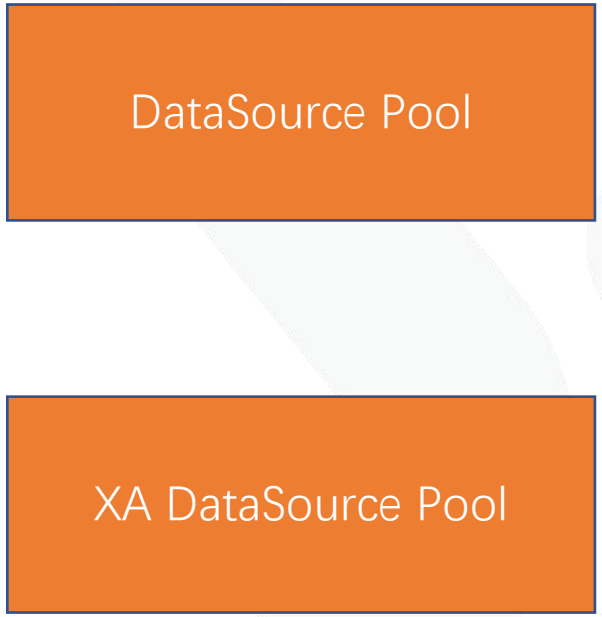
- 介绍
  - 基于 事务资源 本身提供对XA规范和协议的支持
- 核心价值
  - 从 场景 上，满足 全局一致性 的需求。
  - 从 应用上，保持与 AT 模式一致的无侵入。
  - 从 机制 上，适应分布式微服务架构的特点。

XA模式优劣势

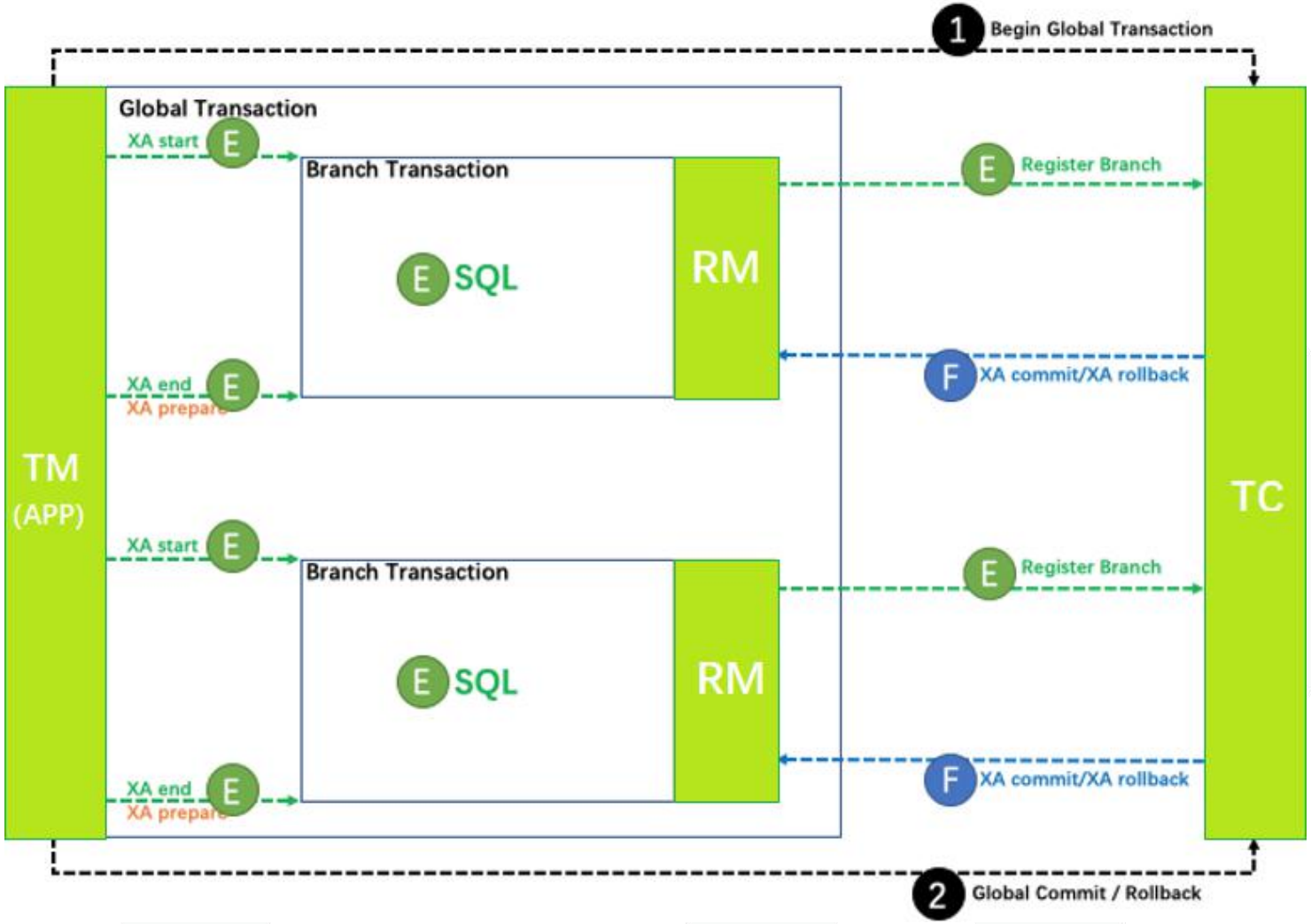
- 优势
  - **业务无侵入**：无需业务逻辑参与。
  - **数据库的支持广泛**：XA协议被主流关系型数据库广泛支持，不需要额外的适配即可使用。
  - **多语言支持容易**：对RM要求较少，为不同语言开发SDK更容易。
  - **传统基于XA应用的迁移**：原XA事务平滑迁移到Seata平台。
- 劣势
  - **数据锁定**：数据在整个事务处理过程结束前，都被锁定，读写都按隔离级别的定义约束起来；
  - **协议阻塞**：XA prepare后，分支事务进入阻塞阶段，收到 XA commit 或 XA rollback 前必须阻塞等待。
  - **性能差**：性能损耗主要来自两方面：一方面，事务协调过程，增加单个事务的RT；另一方面，并发事务数据的锁冲突，降低吞吐。

XA模式核心设计

- 组成
  - 执行阶段（Execute）：
    - XA start/XA end/XA prepare + 注册分支
  - 完成阶段（Finish）：
    - XA commit/XA rollback
- XAConnection获取
  - 根据开发者的普通 DataSource 来创建
  - 要求开发者配置 XADataSource

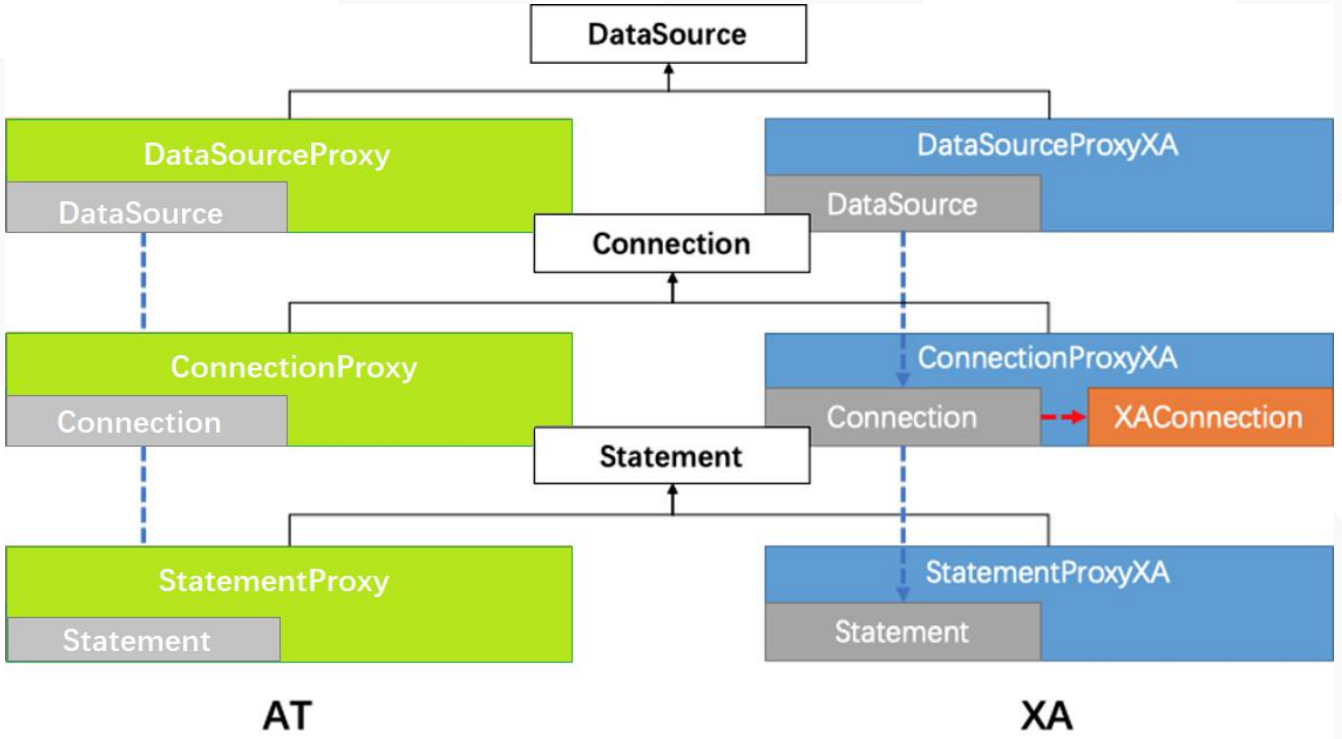


XA模式核心设计


1 Begin Global Transaction
Global Transaction
XA start
E
Branch Transaction
E SQL
RM
E Register Branch
F XA commit/XA rollback
XA end
XA prepare
TM
(APP)
TC
XA start
E
Branch Transaction
E SQL
RM
E Register Branch
F XA commit/XA rollback
XA end
XA prepare
2 Global Commit / Rollback

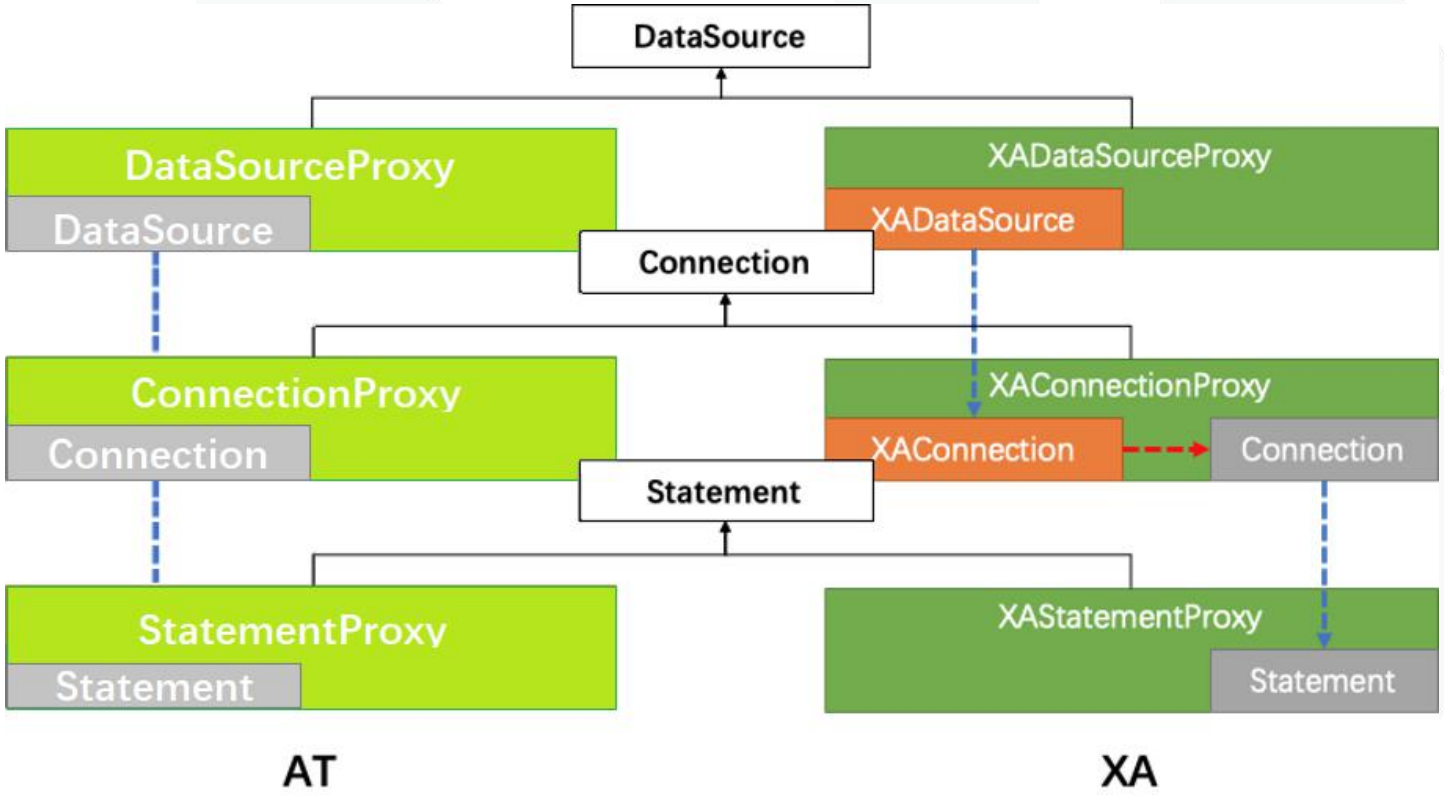
数据源代理

- 优势
  - 对开发者比较友好，开发者完全不必关心 XA 层面的任何问题，保持本地编程模型即可。
- 局限
  - 无法保证兼容的正确性
- 本质原因
  - 为数据库驱动程序补充功能



数据源代理

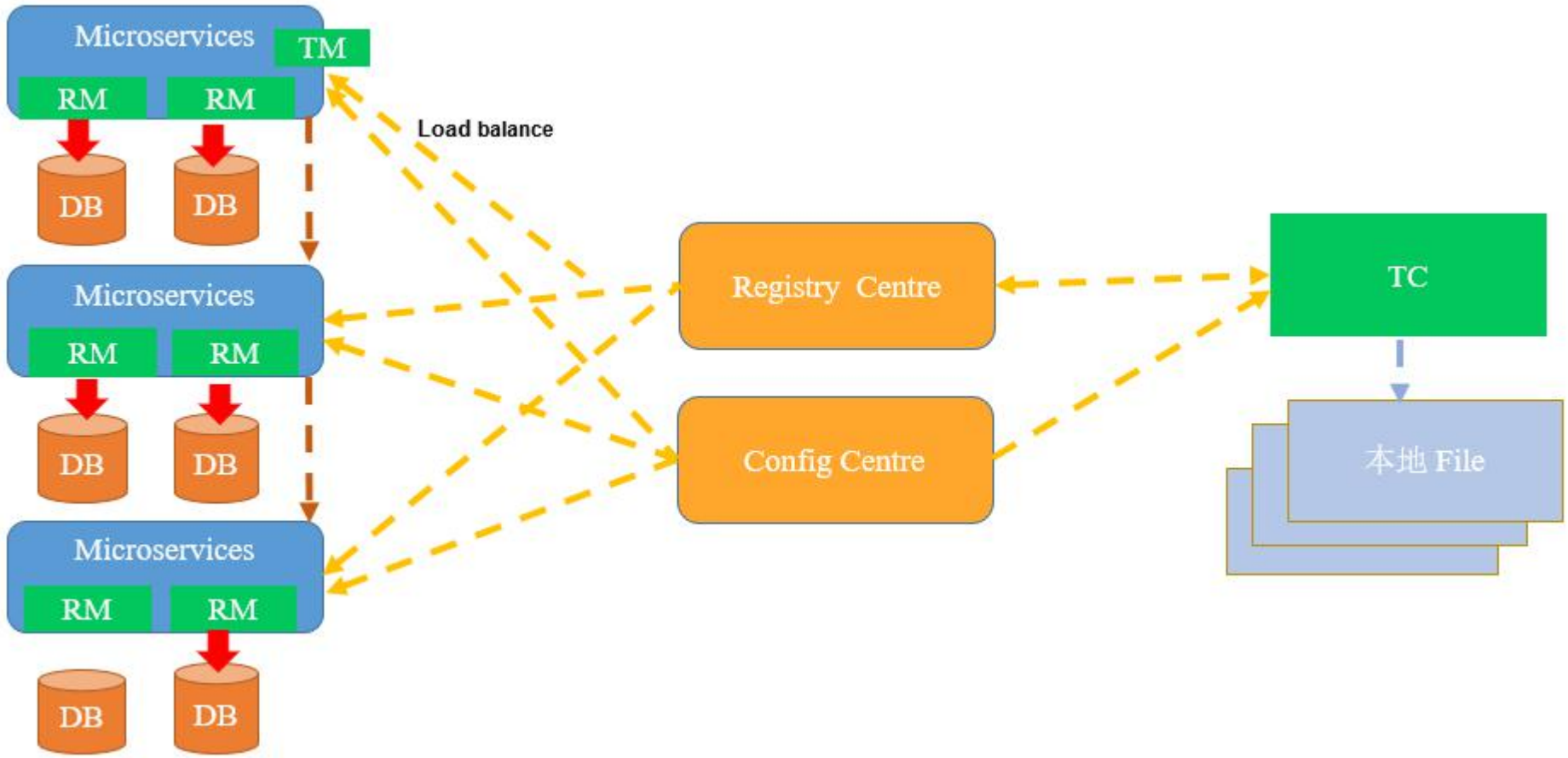
- 最终方案
  - 兼容两种方式，提高兼容性



04. Seata高可用方案

Seata高可用方案

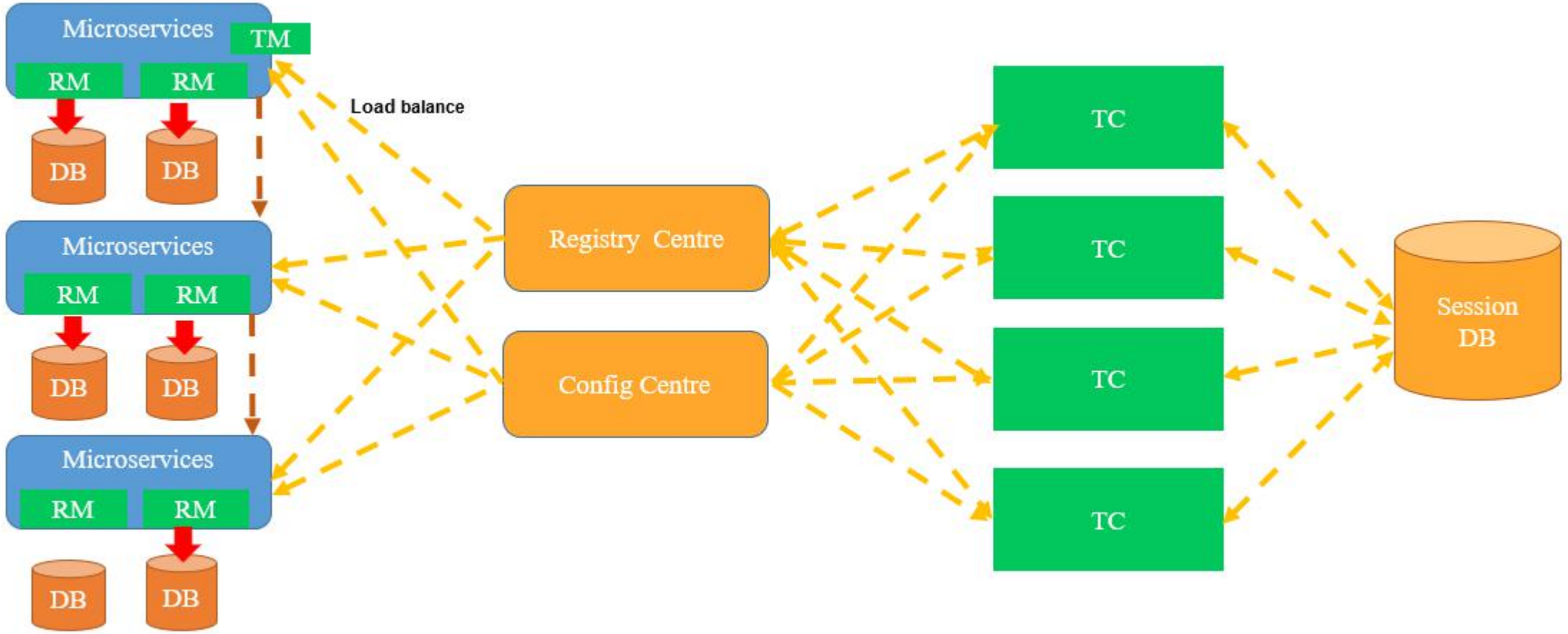
TC有状态部署



Seata高可用方案

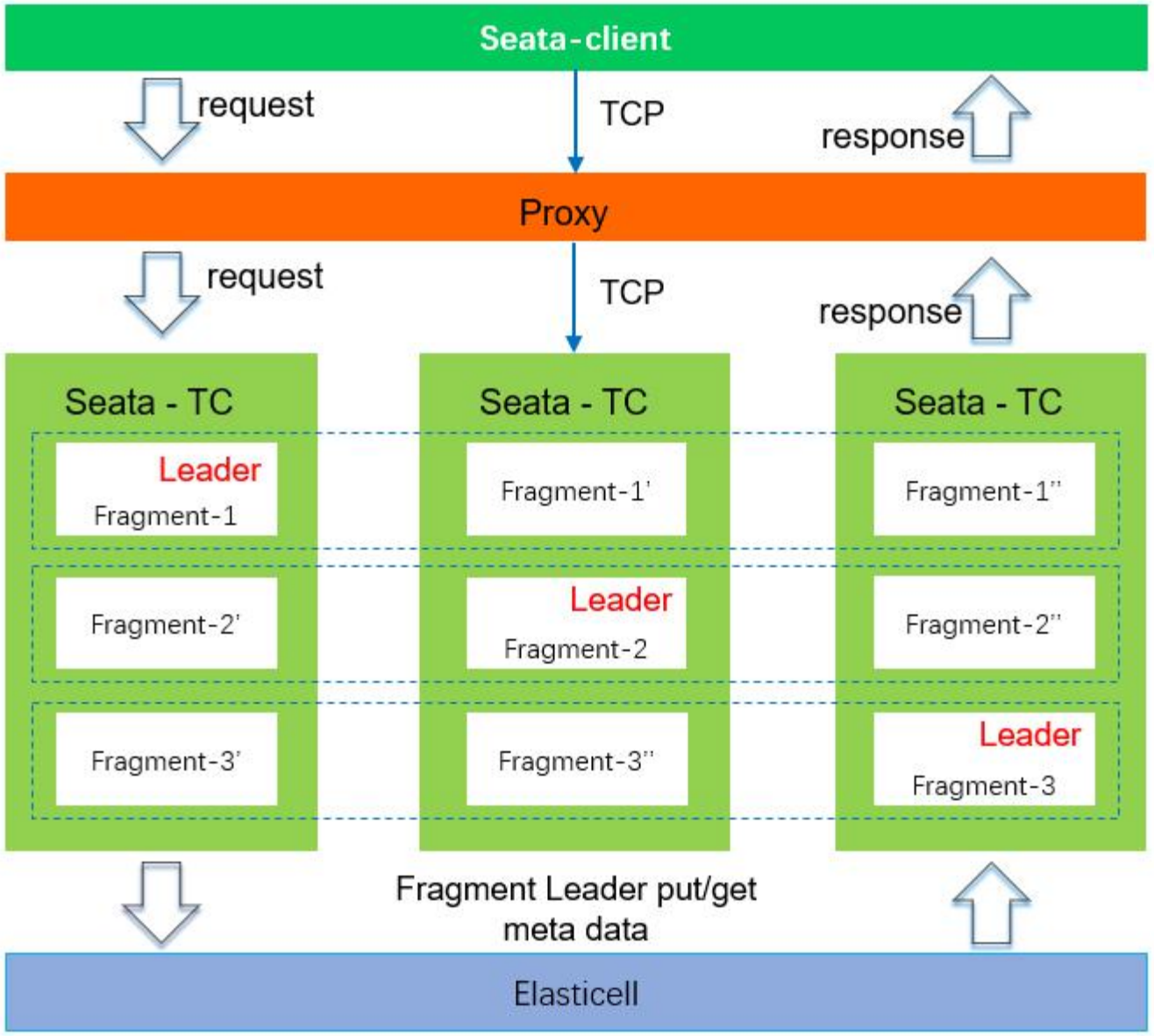
奈学教育

TC无状态部署



Seata高可用方案

一致性协议同步



Seata高可用方案

奈学教育

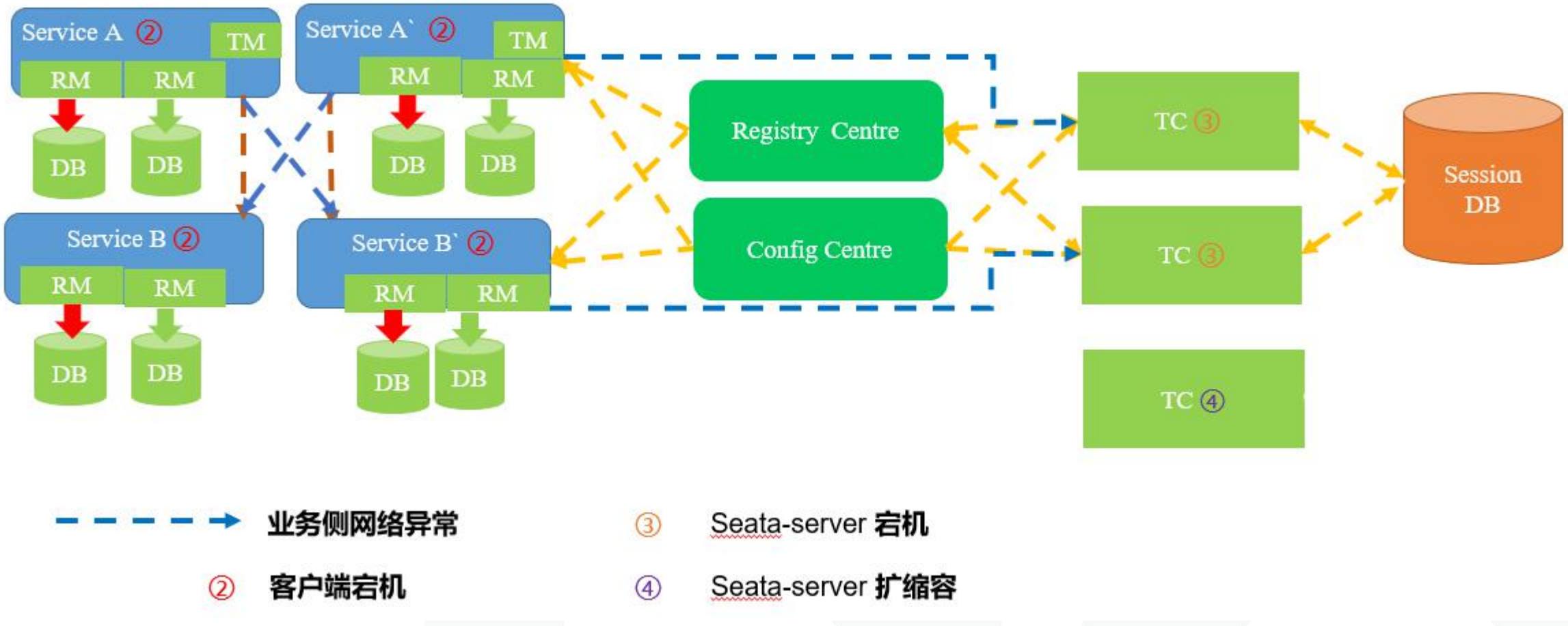
高可用设计

- 当出现业务侧网络异常恢复后，需要业务怎样处理?
- 当出现业务侧宕机（可恢复型和不可恢复型），如何处理?
- 当Seata-Server出现宕机（可恢复型和不可恢复型），如何处理?
- HA模式下需要Seata-Server 扩缩容时，如何处理?

Seata高可用方案

奈学教育

高可用设计



NX 奈学教育

05. 事务消息解决方案

事务消息解决方案

奈学教育

事务消息

- 简化了分布式事务模型
- 对业务友好

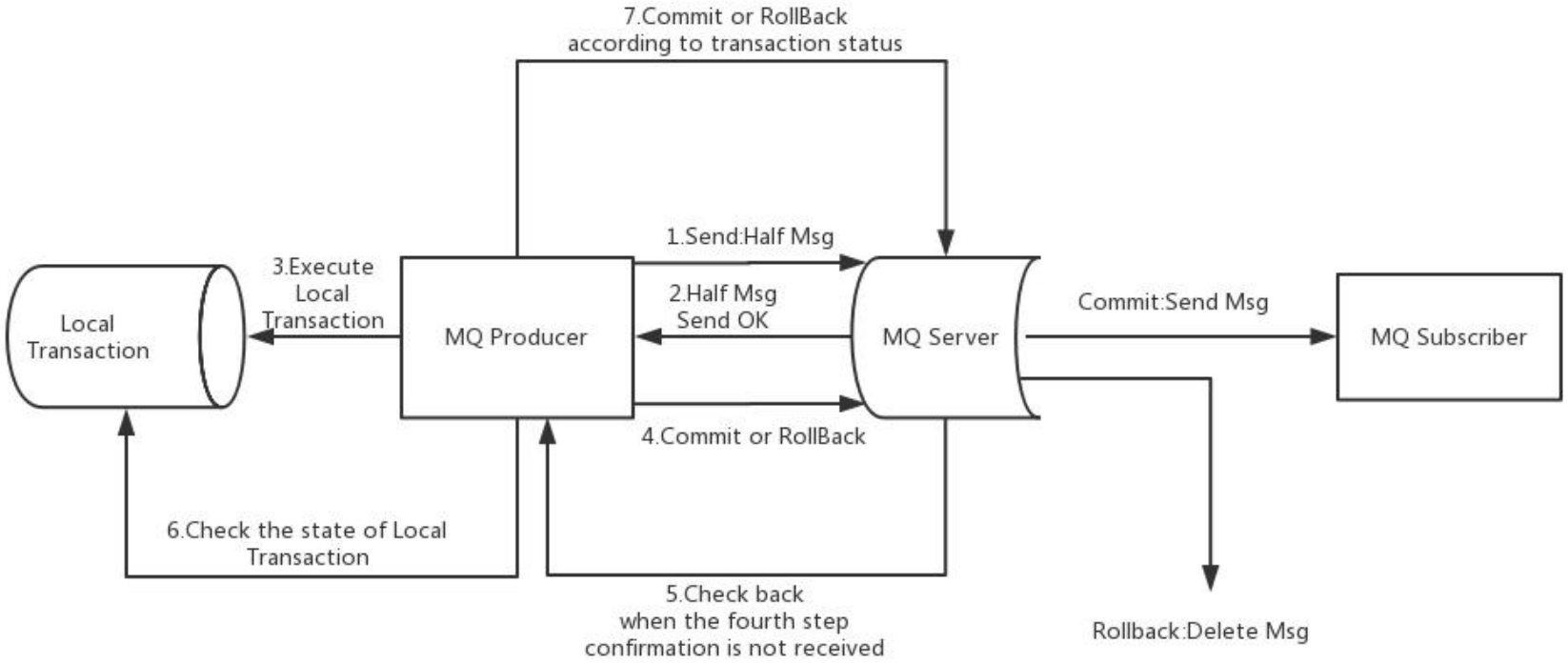
两次RPC调用

RPC + 事务消息

RocketMQ事务消息机制

两阶段提交

- 发送半消息
- 执行本地事务
- 发送Commit/Rollback
- 提供回查接口

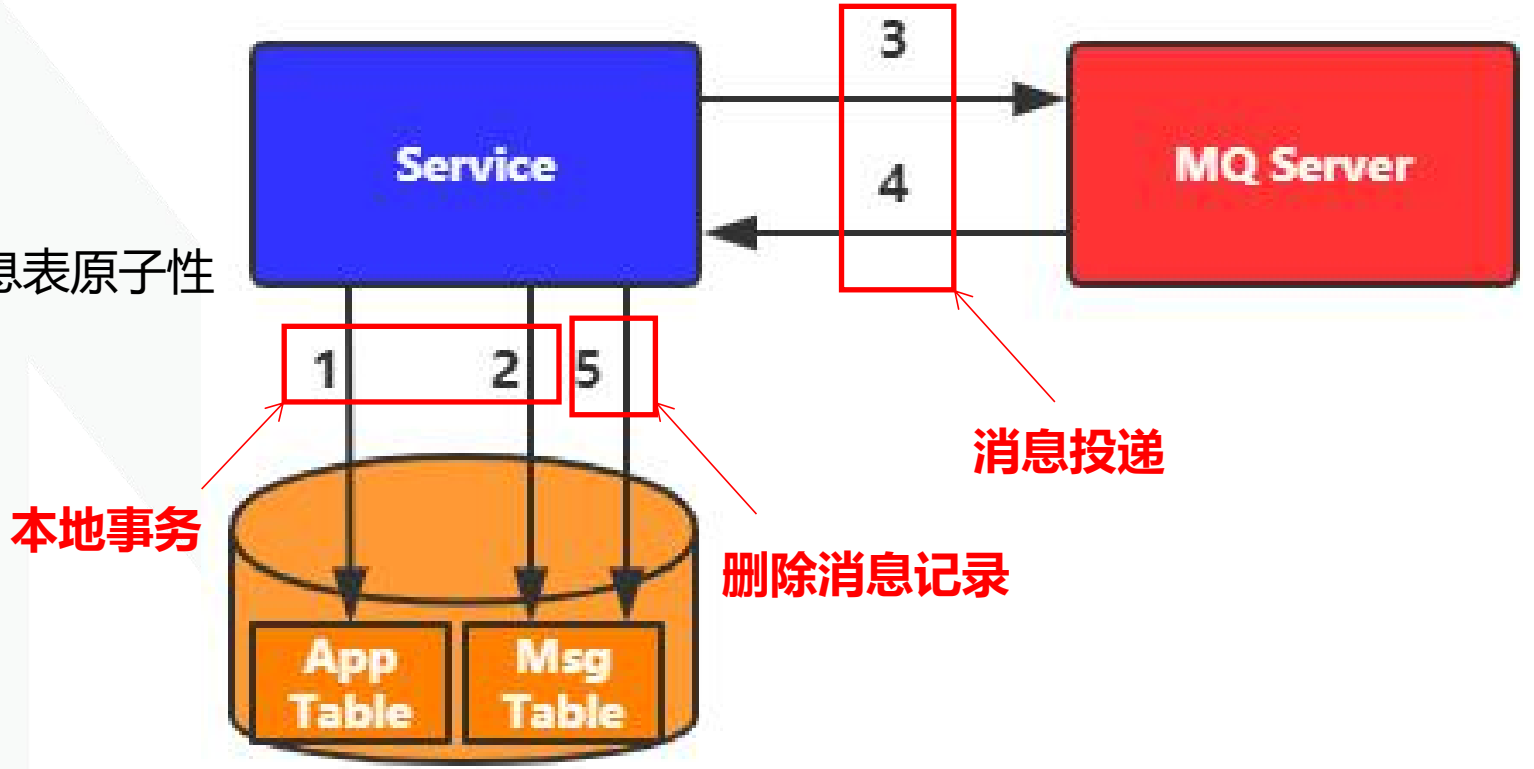


需要业务方提供回查接口，对业务侵入较大

事务消息实现

设计思路

- 通过客户端实现
- 事务消息表记录发消息事件
- 本地事务保证业务数据与写消息表原子性
- 事务管理器维护事务消息表
  - 扫库发送、清理



NX 奈学教育

欢迎关注本人公众号
“架构之美”