3天挑战架构师级MySQL海量数据设计与实践

NX 奈学教育

内容介绍

Day1 MySQL架构体系设计深入剖析篇

MySQL架构体系拆解以及设计深度剖析；
MySQL存储引擎原理拆解以及设计深度剖析；
MySQL锁实现原理拆解以及设计深度剖析；
MySQL事务实现原理拆解以及设计深度剖析；

Day 2 企业千亿级海量数据并发分库分表设计方法论提炼篇

千亿级海量数据高并发场景分库分表设计方法论；
千亿级海量数据高并发场景主键设计选择；
千亿级海量数据高并发场景分片键设计选择；
千亿级海量数据高并发场景分库实践落地方案；
千亿级海量数据高并发场景分表实践落地方案；

Day 3 企业千亿级海量数据真实案例设计与实战篇

万亿级微信消息垂直拆分真实案例实战；
企业级数据库Sharding Sphere分库分表应用设计实战；
企业级分布式事务阿里巴巴Seata应用设计实战；
企业级MySQL数据库高可用应用设计与实战；

玄姐：为什么说π型人才是社会上最稀缺的人才

陈东

前58集团架构师， 前转转公司架构平台部负责人、高级架构师、技术委员会核心成员， 主导了转转基础架构部门从0到1的建设工作， 负责转转RPC框架和服务治理生态的落地、消息队列的研发和多元化存储体系的建设，以及众多核心基础组件的设计研发和产品化工作，擅长后端架构、中间件、服务治理、存储等技术方向，对即时通讯系统有深刻的研究。

今晚20:00准时开课，请稍后…

奈学教育 出品

目录

NX 奈学教育

NX 奈学教育

01.企业级数据库Sharding Sphere分库分表应用设计实战

数据库扩展带来的问题

- 请求路由
- 分布式事务

数据库扩展带来的问题

- **请求路由**
  - 分表规则
  - 写入路由
  - 查询路由
- 分布式事务

分页查询怎么办?

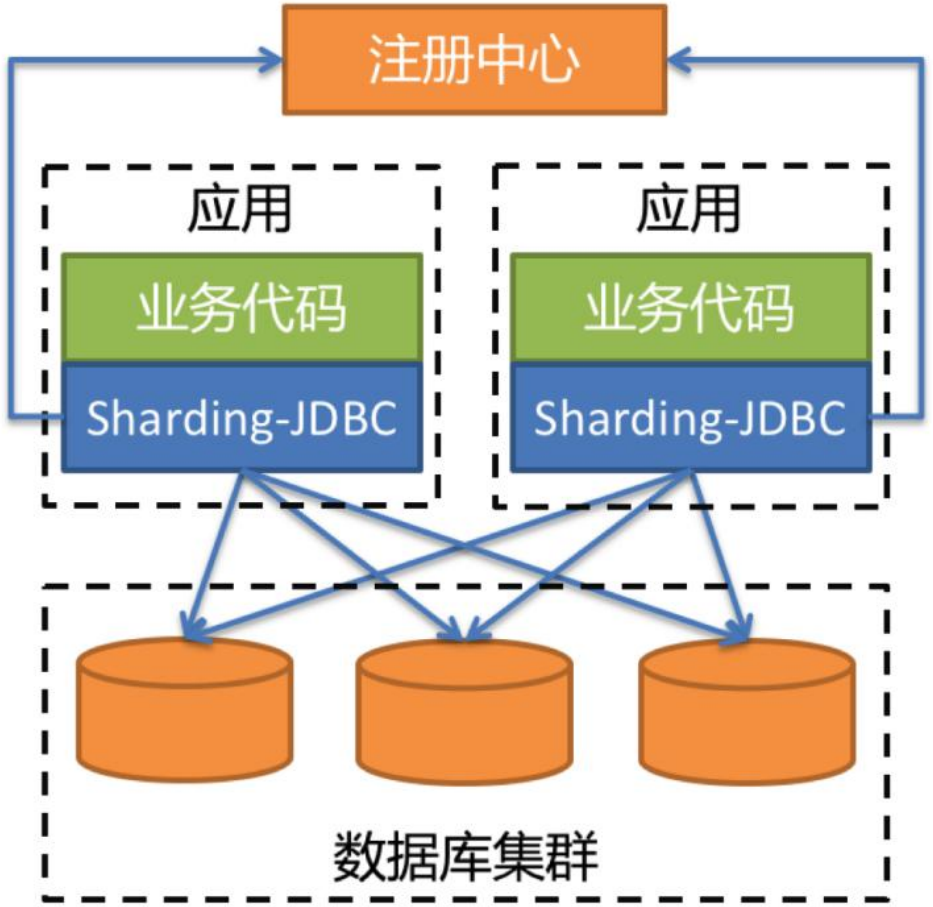
数据库扩展带来的问题

- 请求路由
- **分布式事务**

业务规避！！！

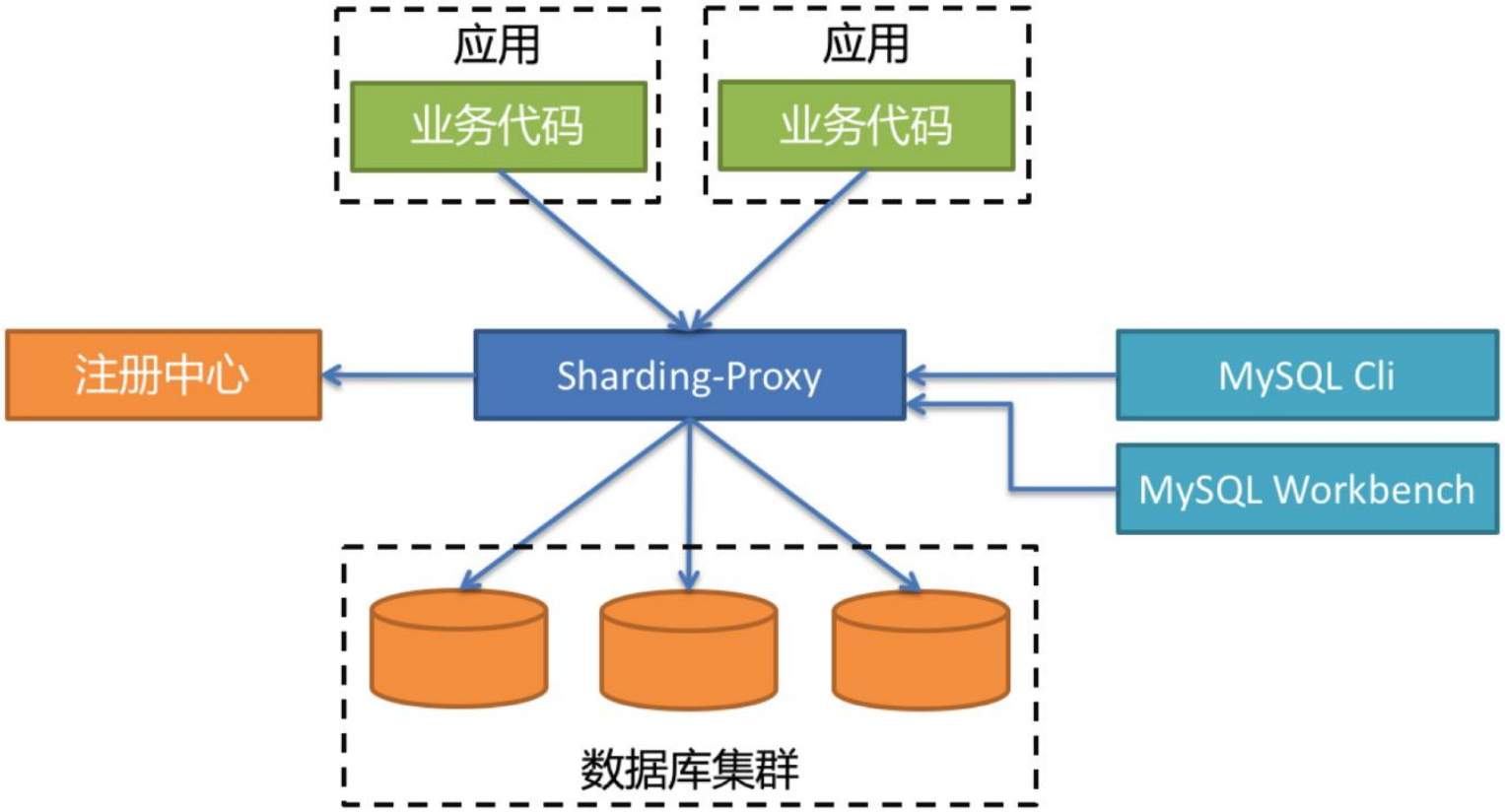
分库分表解决方案—请求路由

- Sharding JDBC
  - 客户端集成
- Sharding-Proxy
  - 代理模式



分库分表解决方案—请求路由

- Sharding JDBC
  - 客户端集成
- Sharding-Proxy
  - 代理模式



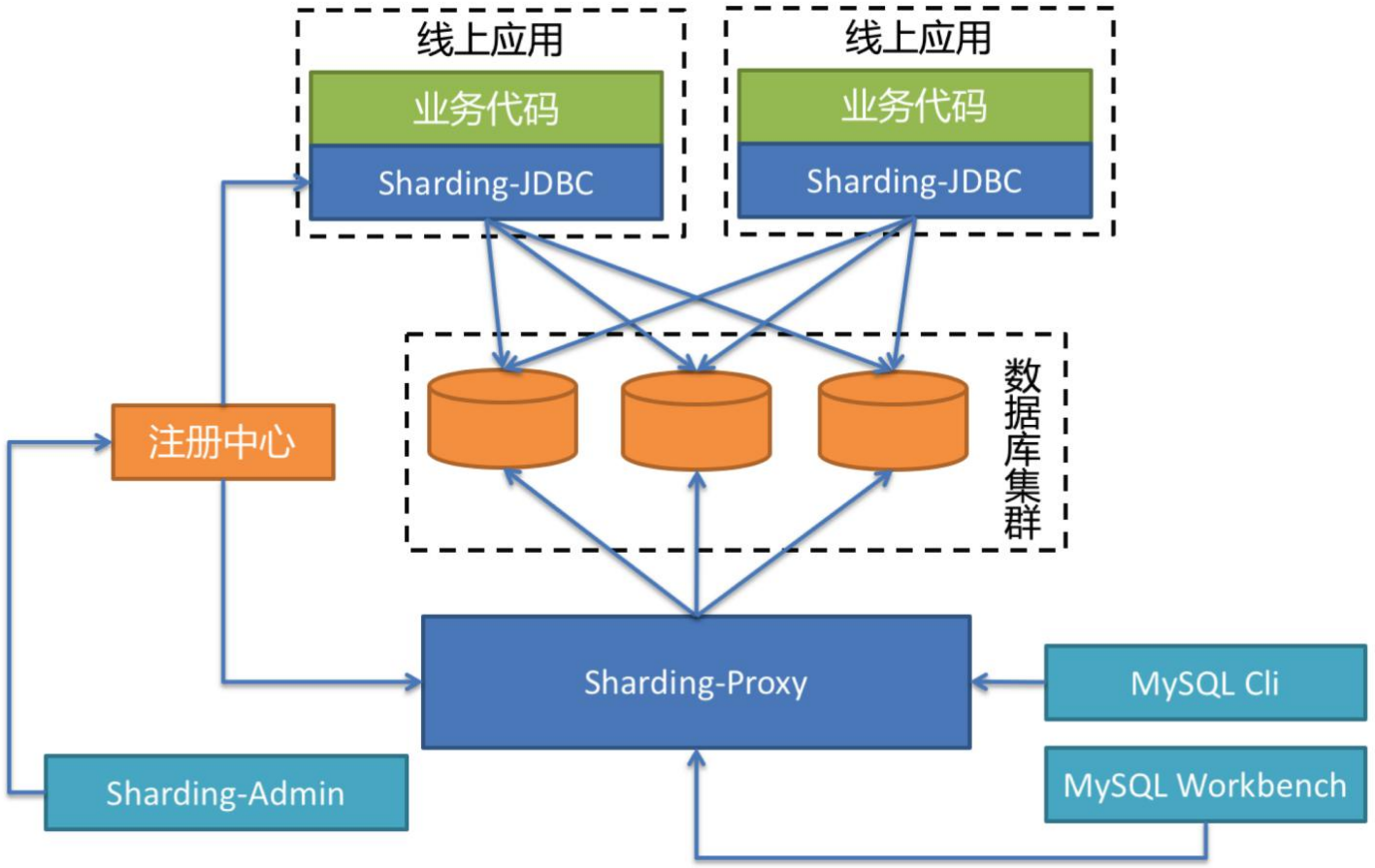
分库分表解决方案—请求路由

- Sharding JDBC
- Sharding-Proxy

对比项	Sharding-JDBC	Sharding-Proxy
数据库	任意	单一
异构语言	仅Java	任意
连接数	高	低
性能	损耗低	损耗略高
去中心化	是	否
静态入口	无	有

分库分表解决方案—请求路由

- 混合模式



NX 奈学教育

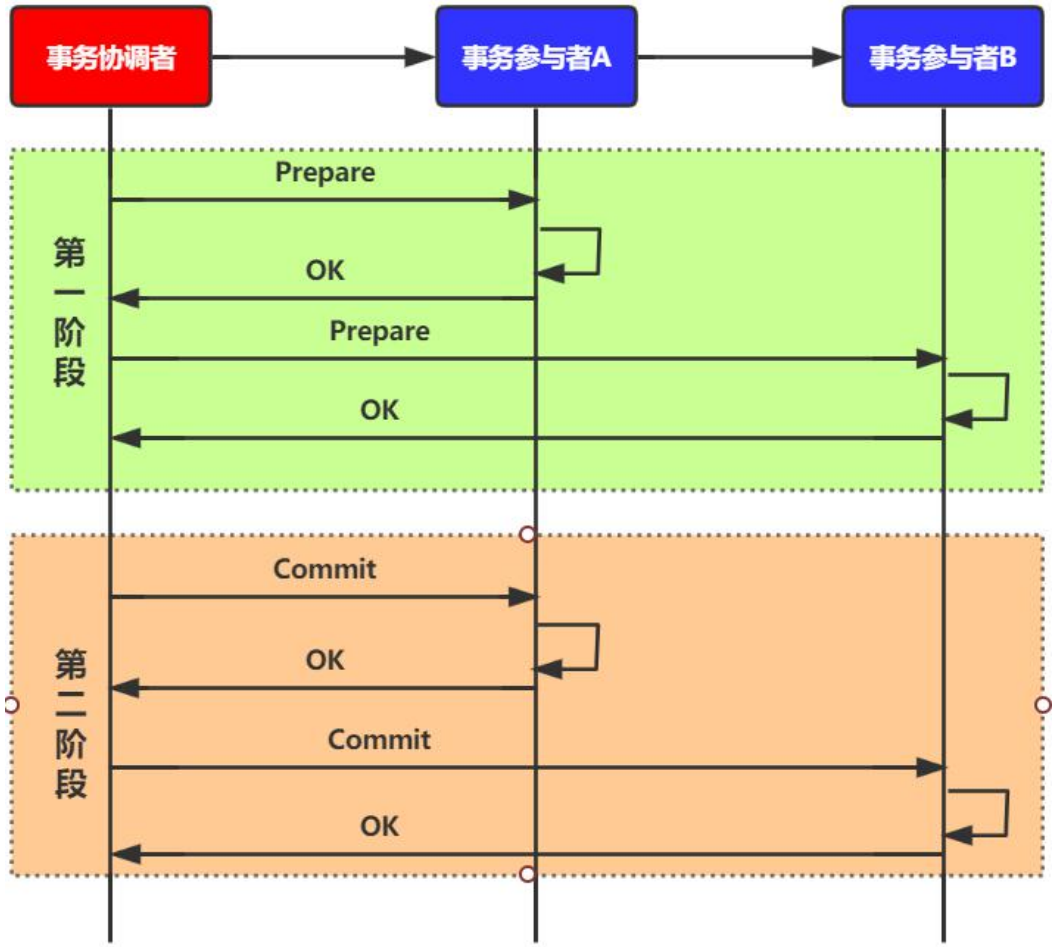
02.企业级分布式事务阿里巴巴Seata应用设计实战

分布式事务

- 强一致
- 柔性事务
- 事务消息

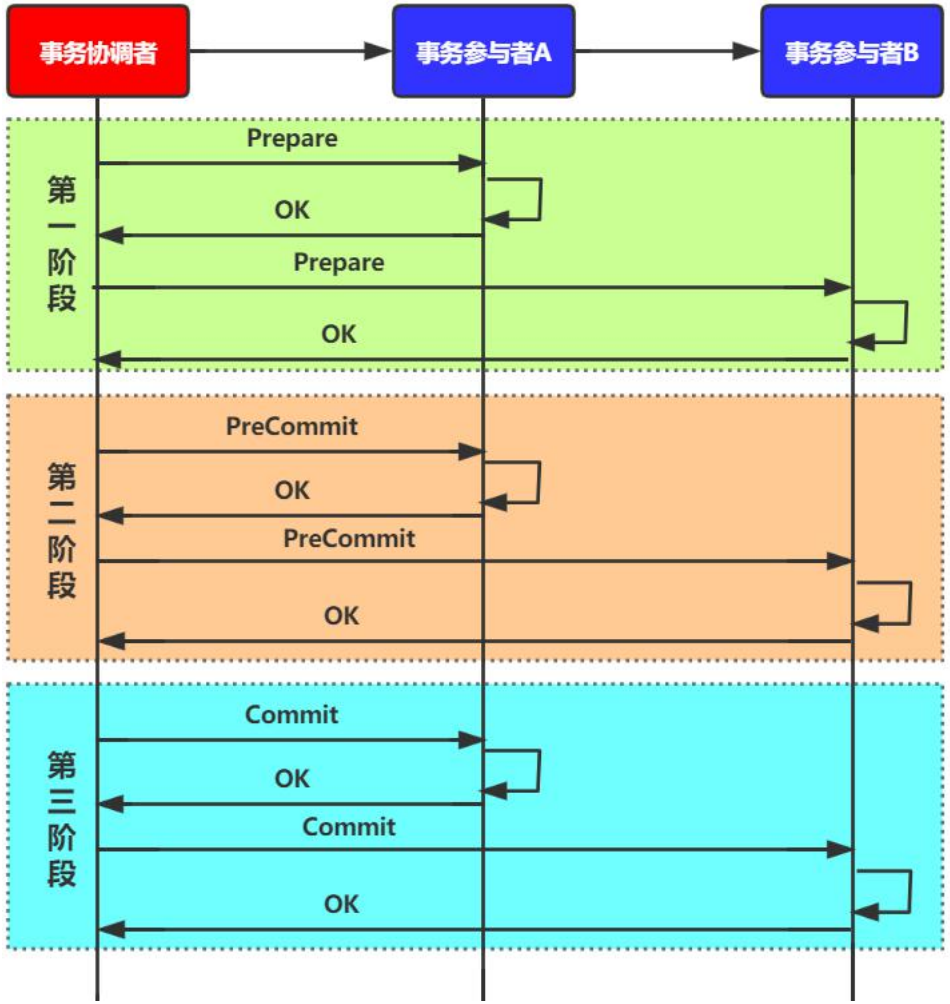
强一致协议

- 一致性协议
  - 两阶段提交 2PC
  - 三阶段提交 3PC
- 落地方案
  - XA规范



强一致协议

- 一致性协议
  - 两阶段提交 2PC
  - 三阶段提交 3PC
- 落地方案
  - XA规范



强一致协议

- 一致性协议
  - 两阶段提交 2PC
  - 三阶段提交 3PC
- 落地方案
  - XA规范
    - 资源管理器-事务参与者
    - 事务管理器-事务协调者

1、写入加锁
2、记录事务执行状态

分布式事务

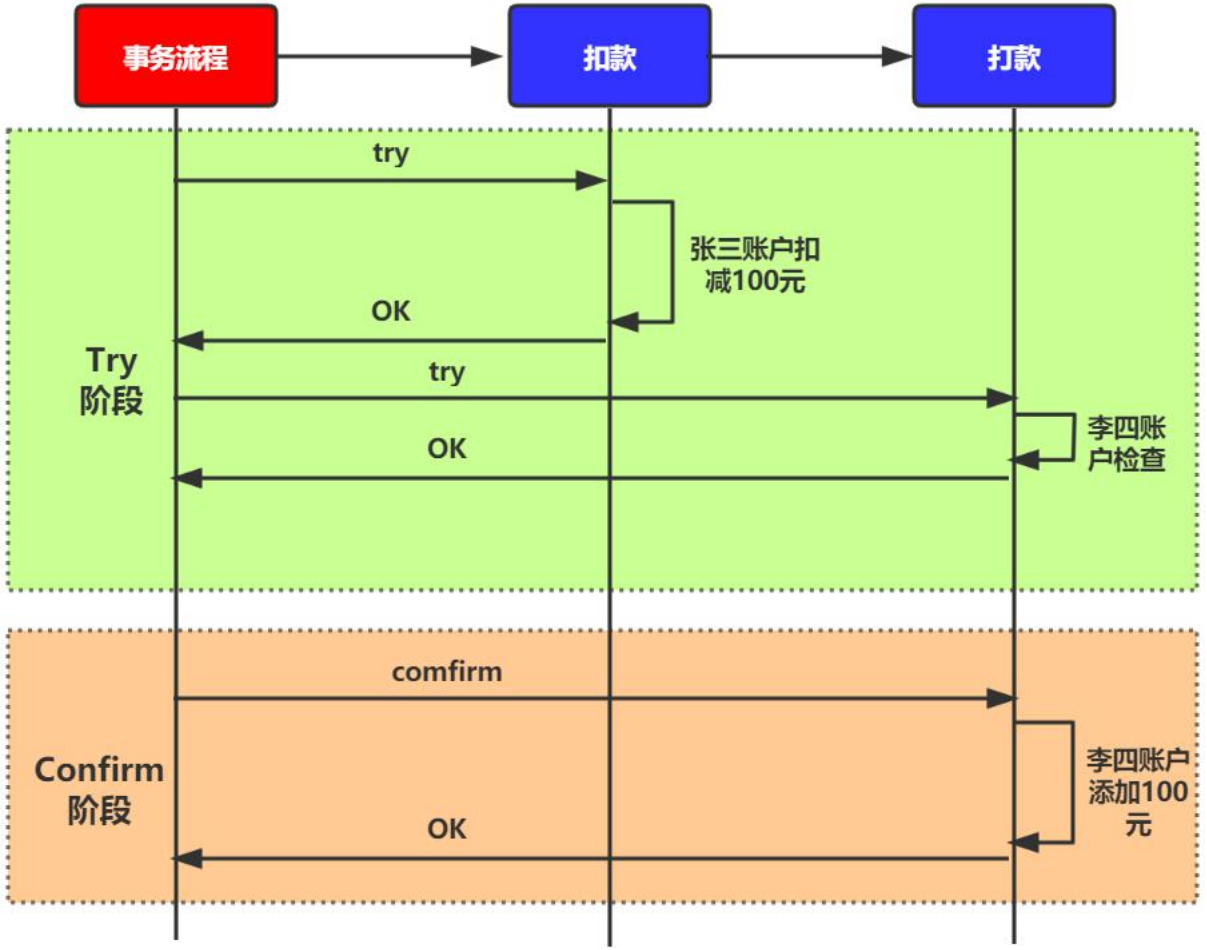
- 强一致
- 柔性事务
- 事务消息

最终一致性方案

- TCC（Try-Confirm-Cancel）
- SAGA模型

最终一致性方案

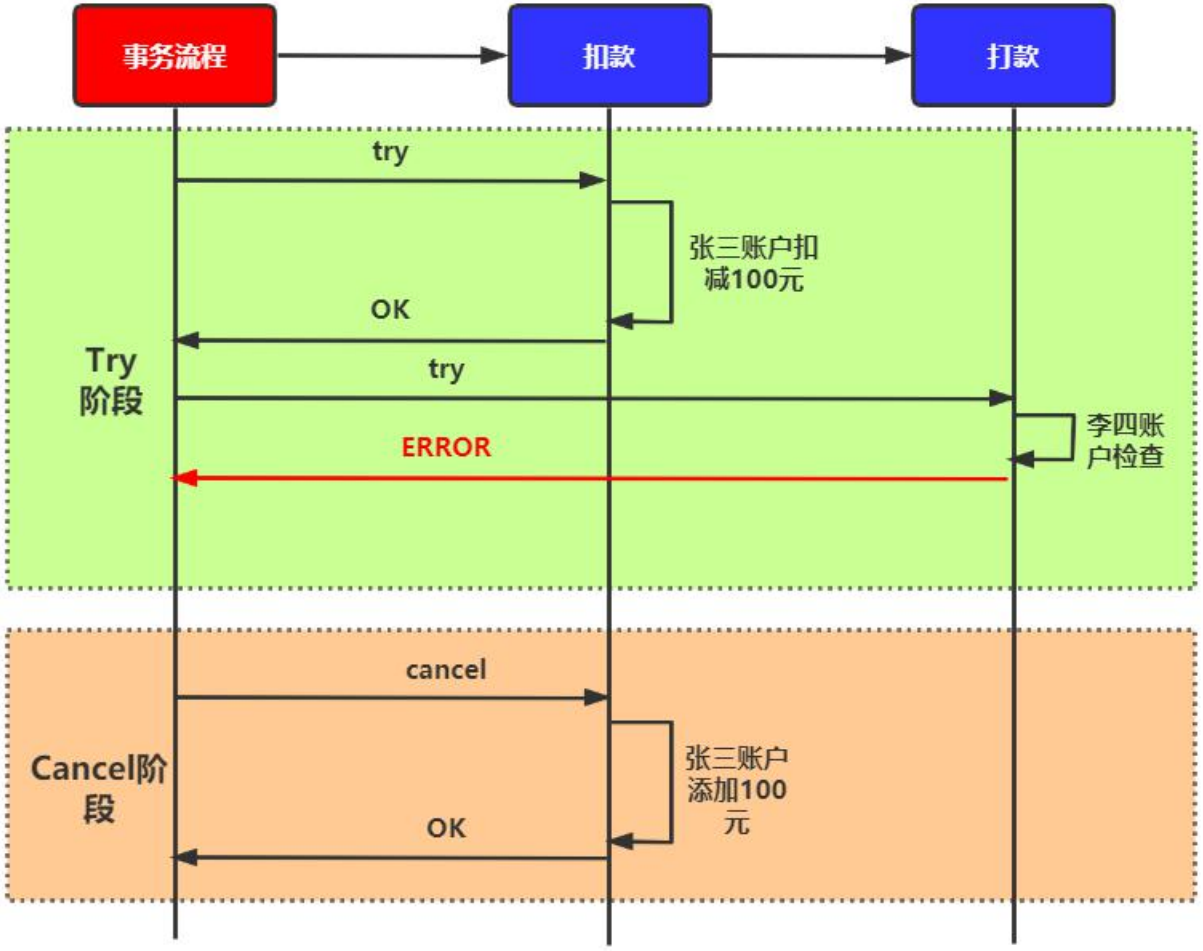
- TCC（Try-Confirm-Cancel）
  - 尝试执行业务，预留资源；
  - 确认执行业务，使用Try阶段资源；
  - 取消执行业务，释放Try阶段预留的资源；
- SAGA模型



TCC成功处理流程

最终一致性方案

- TCC（Try-Confirm-Cancel）
  - 尝试执行业务，预留资源；
  - 确认执行业务，使用Try阶段资源；
  - 取消执行业务，释放Try阶段预留的资源；
- SAGA模型



TCC失败处理流程

最终一致性方案

- TCC（Try-Confirm-Cancel）
- SAGA模型
  - 一个分布式事务拆分为多个本地事务；
  - 本地事务都有相应的执行模块和补偿模块；
  - 事务管理器负责在事务失败时调度执行补偿逻辑；

分布式事务

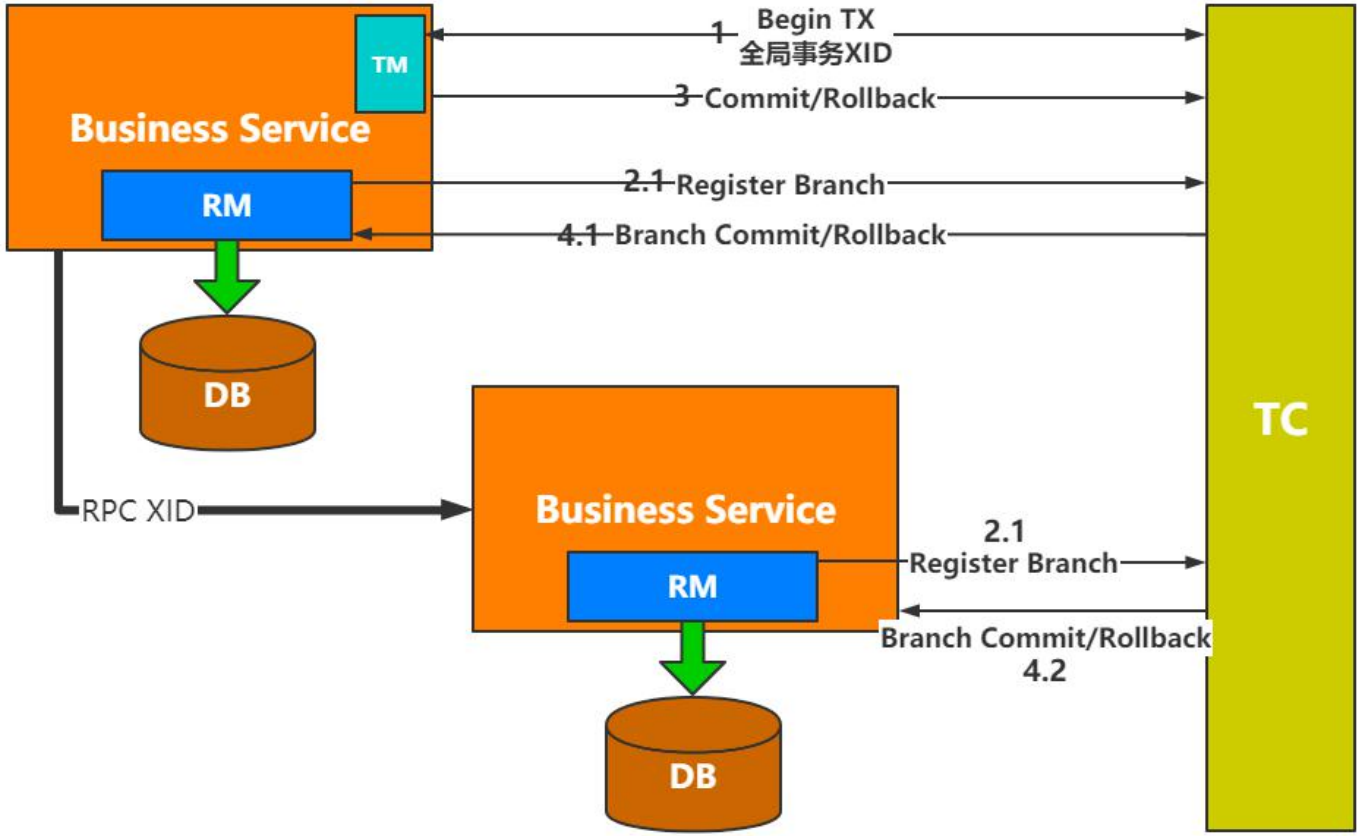
- 强一致
- 柔性事务
- 事务消息
  - 简化了分布式事务模型
  - 对业务友好

两次RPC调用

RPC + 事务消息

Seata分布式事务流程

- TM向TC 申请开启一个全局事务，TC创建全局事务后返回全局唯一的XID，XID会在全局事务的上下文中传播；
- RM向TC注册分支事务，该分支事务归属于拥有相同XID的全局事务；
- TM向TC发起全局提交或回滚；
- TC调度XID下的分支事务完成提交或者回滚。

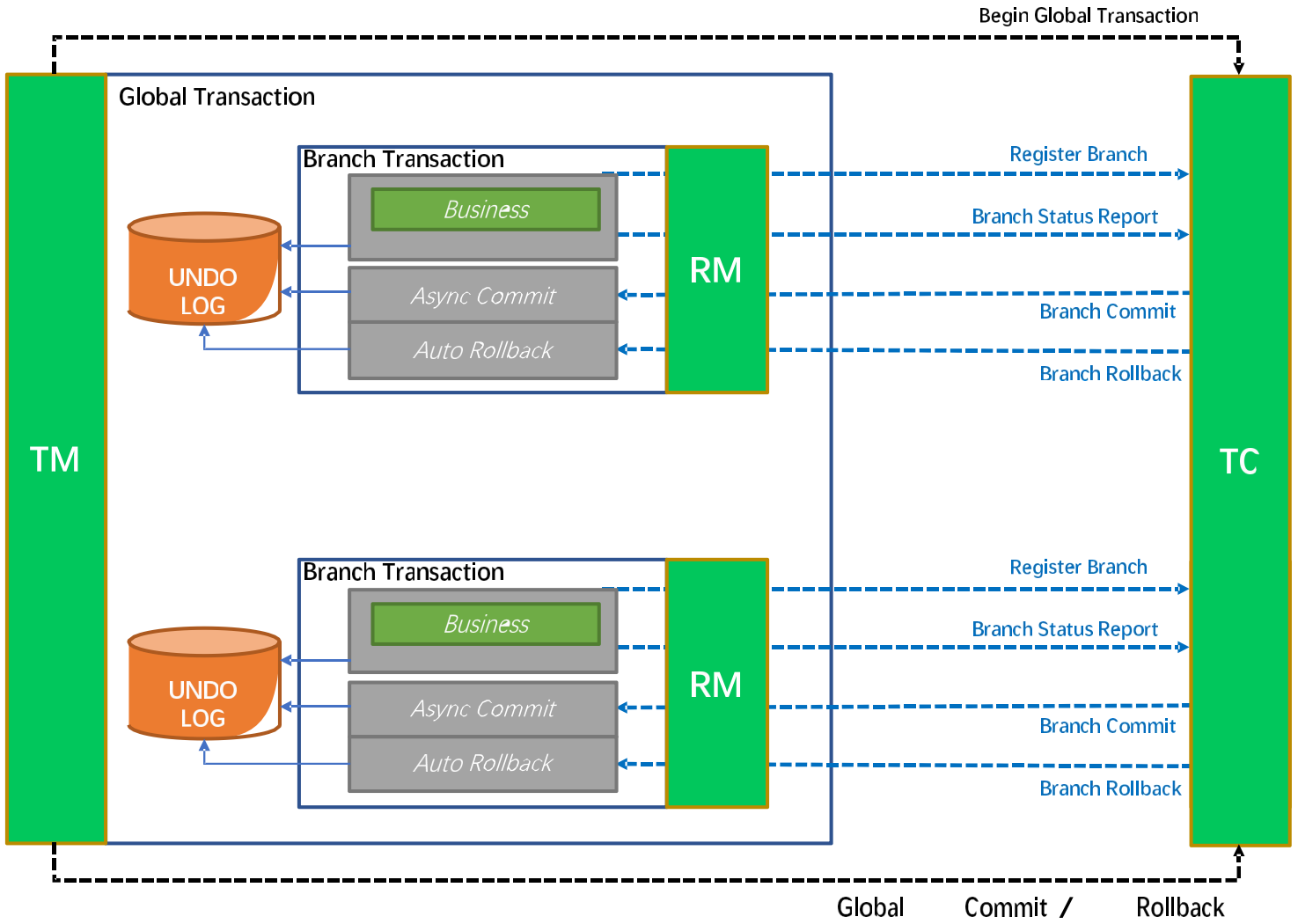


Seata-AT模式

- 介绍
  - 一种无侵入的分布式事务解决方案，2PC的广义实现。
  - 源自阿里云GTS AT模式的开源版。
- 核心价值
  - 低成本：编程模型不变，轻依赖不需要为分布式事务场景做特定设计。
  - 高性能：一阶段提交，不阻塞；连接释放，保证整个系统的吞吐。
  - 高可用：极端的异常情况下，可以暂时 跳过异常事务，保证系统可用。

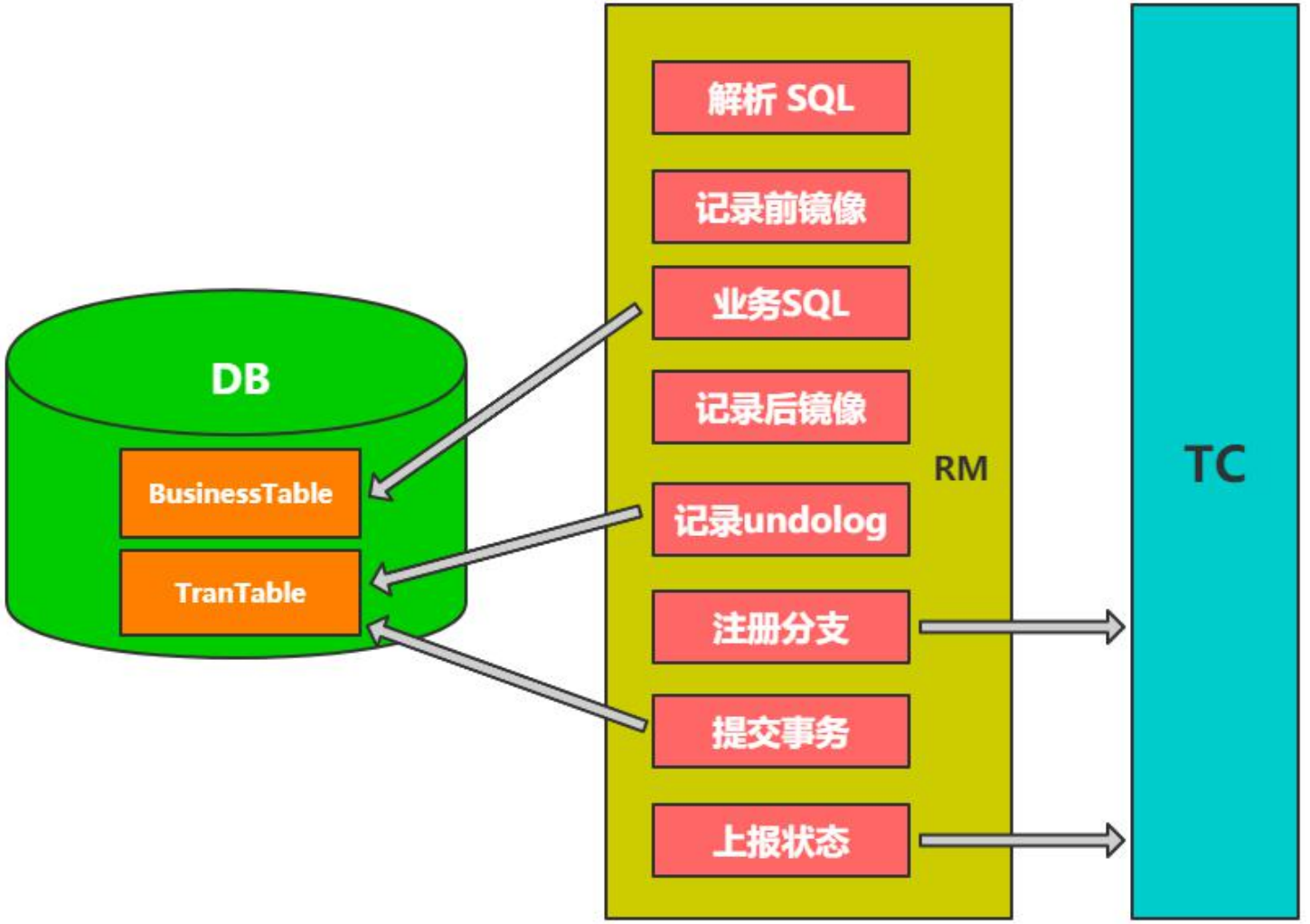
AT模式核心设计

- 一阶段：
  - 拦截“业务 SQL”；
  - 生成前镜像
  - 生成后镜像
- 二阶段
  - TC向所有RM发起提交/回滚；



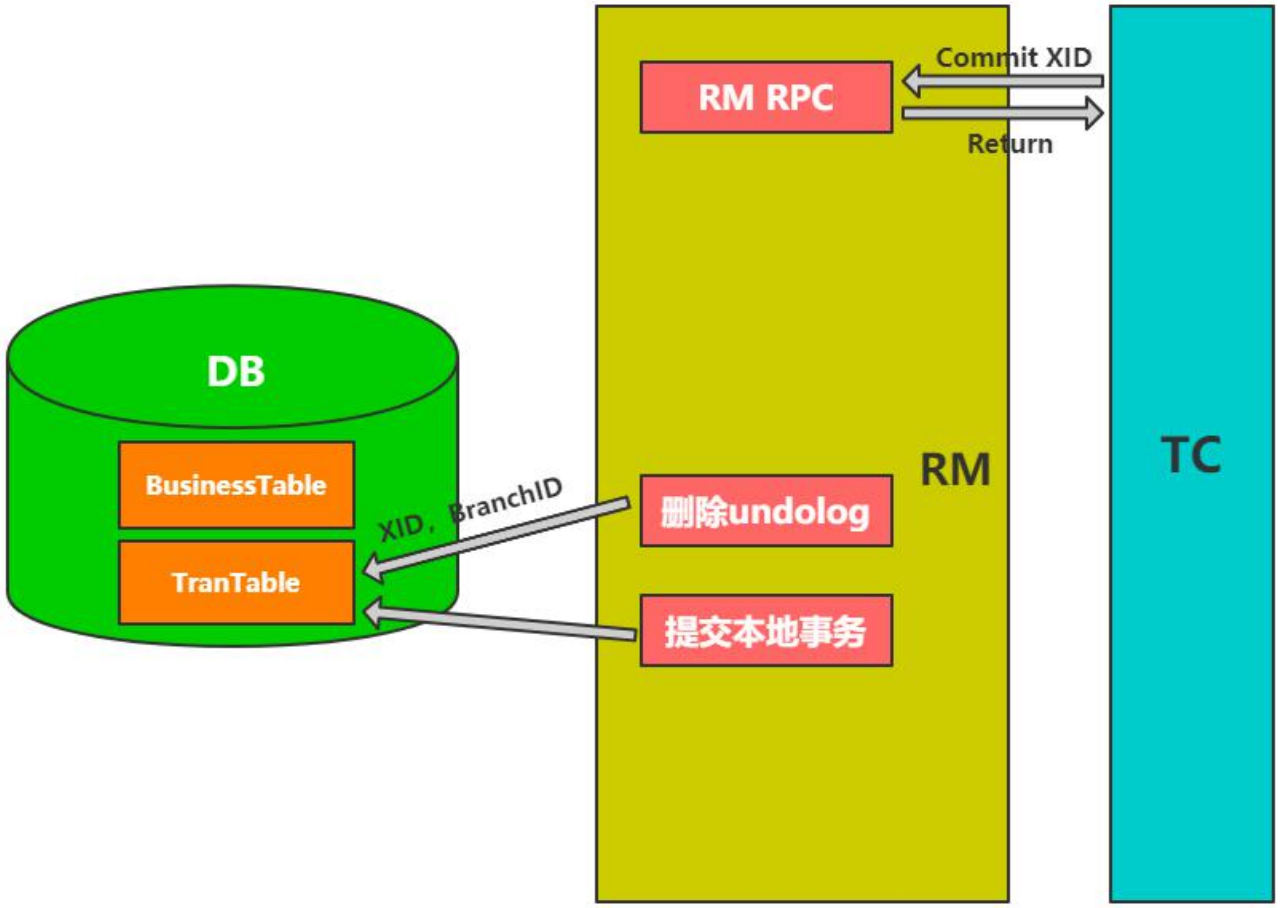
AT执行流程

- 拦截“业务 SQL”
  - 解析SQL语义
  - 提取表元数据
  - 保存原镜像
  - 执行业务SQL
  - 保存新镜像
- 利用本地事务保证ACID



AT完成阶段-提交

- 清理快照数据



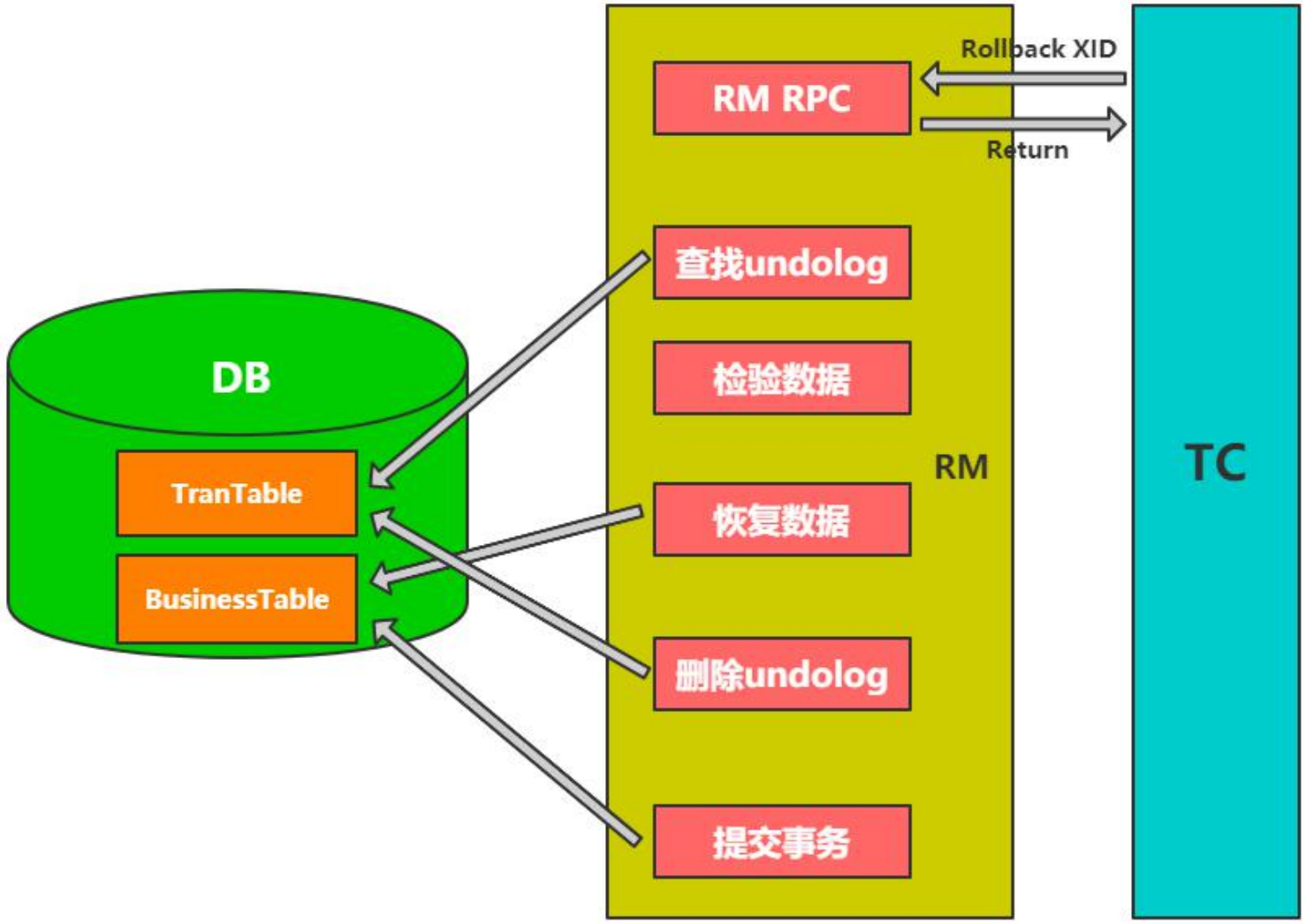
思考

- 分支事务已经提交，如何回滚；
- 防止其他事务对数据的修改；
  - 其他分布式事务的修改；
  - 非分布式事务的修改；

“行锁”

AT完成阶段-回滚

- 校验脏写
  - 新镜像 VS 当前数据库数据
- 还原数据
  - 根据前后镜像，生成逆向SQL
- 删除中间数据
  - 删除前后镜像



AT前后镜像

- 商品库存操作
  - 用户下单减库存

Pid	count
100	10

UPDATE product SET
count = count -1
where pid = **100**;

SELECT * FROM product where pid = **100**;

Pid	count
100	10

获取前镜像

UPDATE product SET count = count -1 where pid = **100**;

SELECT * FROM product where pid = **100**;

Pid	count
100	9

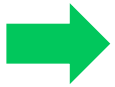
获取后镜像

AT前后镜像

- 商品库存操作
  - 用户下单减库存

Pid	count
100	10

Pid	count
100	9



```
{
        "branchId": XXXX,
        "undoItems": [{
                "afterImage": {
                        "rows": [{
                                "fields": [{
                                        "name": "pid",
                                        "type": XX,
                                        "value": 100
                                }, {
                                        "name": "count",
                                        "type": XX,
                                        "value": 10
                                }]
                        }],
                        "tableName": "product"
                },
                "beforeImage": {
                        "rows": [{
                                "fields": [{
                                        "name": "pid",
                                        "type": XX,
                                        "value": 100
                                }, {
                                        "name": "count",
                                        "type": XX,
                                        "value": 9
                                }]
                        }],
                        "tableName": "product"
                },
                "sqlType": "UPDATE"
        }],
        "xid": "xid:xxx"
}
```

undo log

行锁?

业务场景分析

- 商品库存操作
  - 用户下单减库存
  - 商户增加库存

每个下都是单独立的分布式事务，不同的XID

商品增加库存，不是分布式事务

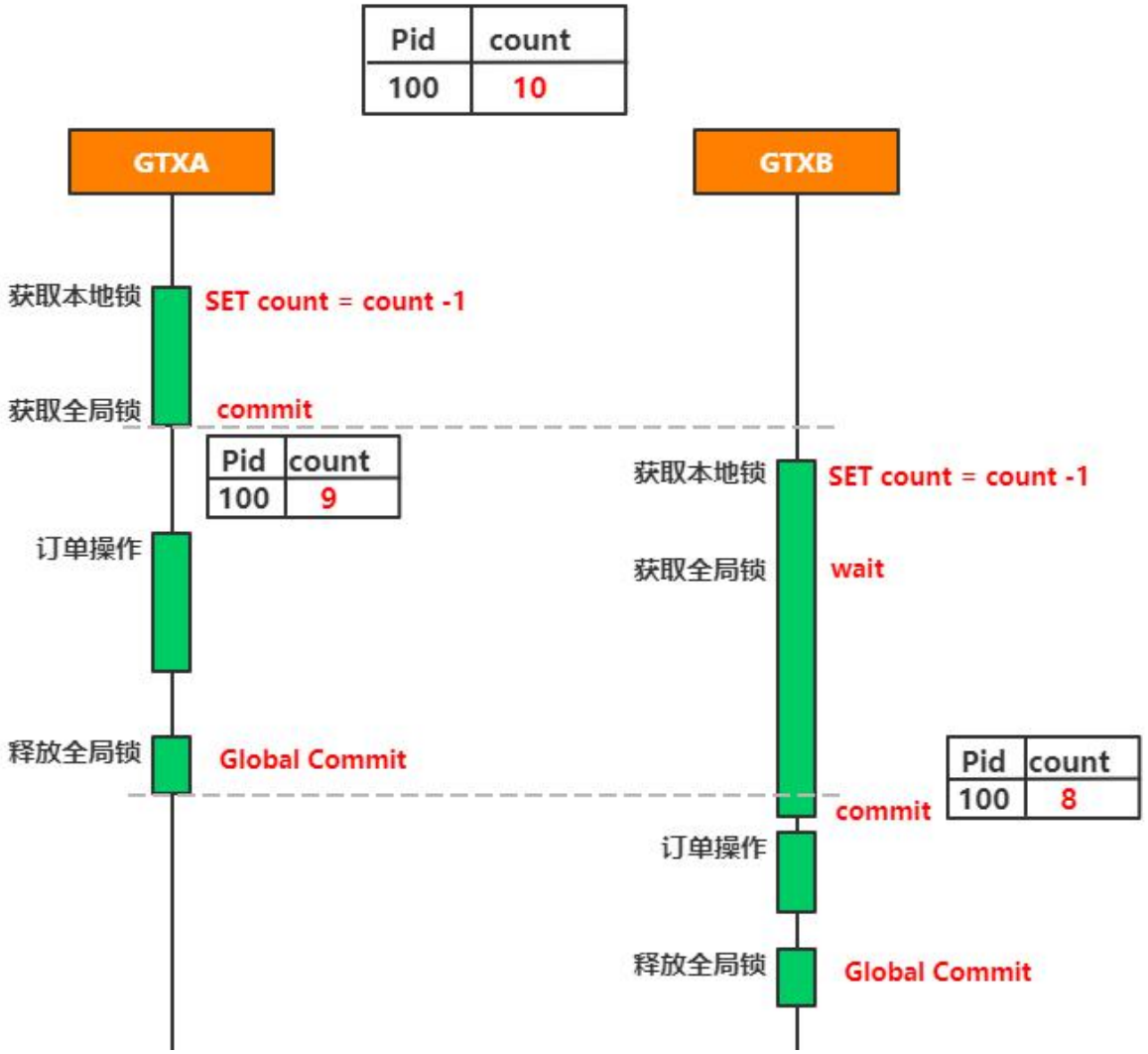
不检查全局锁，锁失效

隔离级别

- **Read Uncommitted**（读取未提交内容）：最低隔离级别，会读取到其他事务未提交的数据；
- **Read Committed**（读取提交内容）：事务过程中可以读取到其他事务已提交的数据
- **Repeatable Read**（可重复读）：每次读取相同结果集，不管其他事务是否提交；
- **Serializable**（可串行化）：事务排队，隔离级别最高，性能最差；

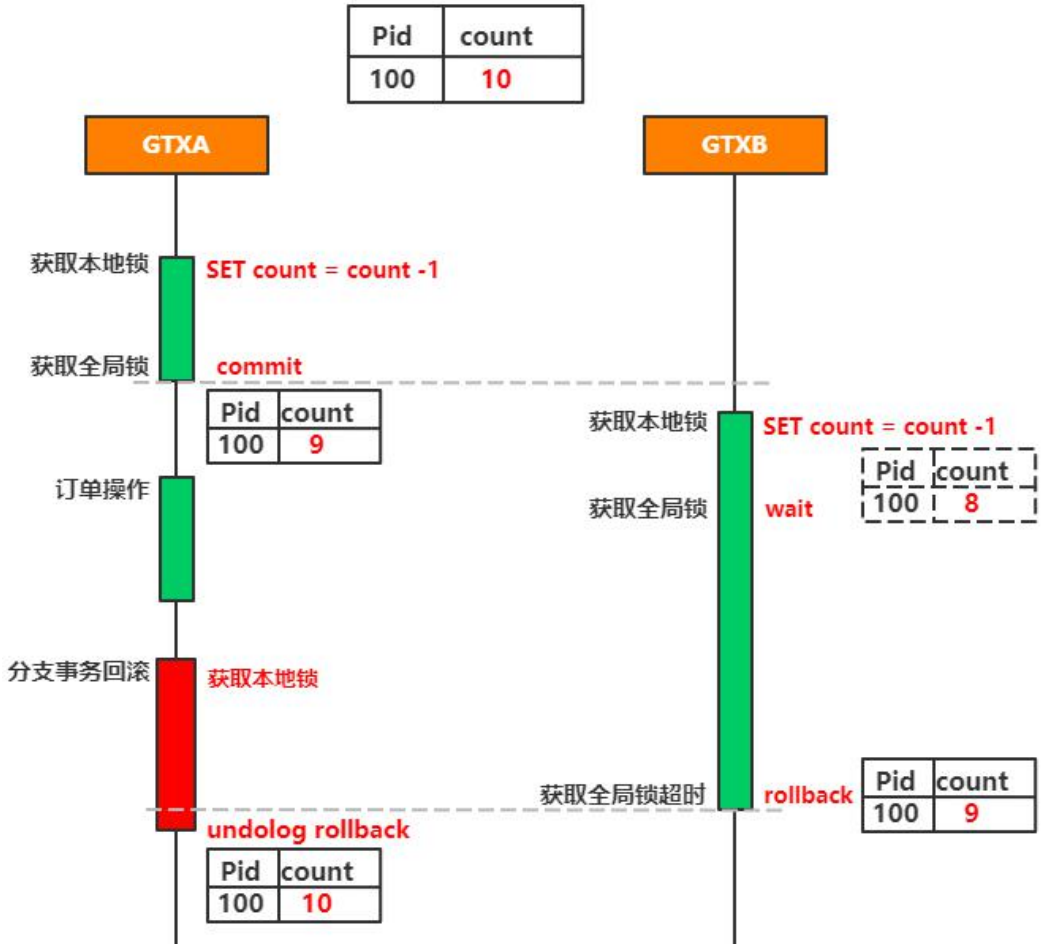
Seata隔离性

- 写隔离
  - **分支事务提交前拿到全局锁；**
  - 拿全局锁超时，回滚本地事务，释放本地锁；



Seata隔离性

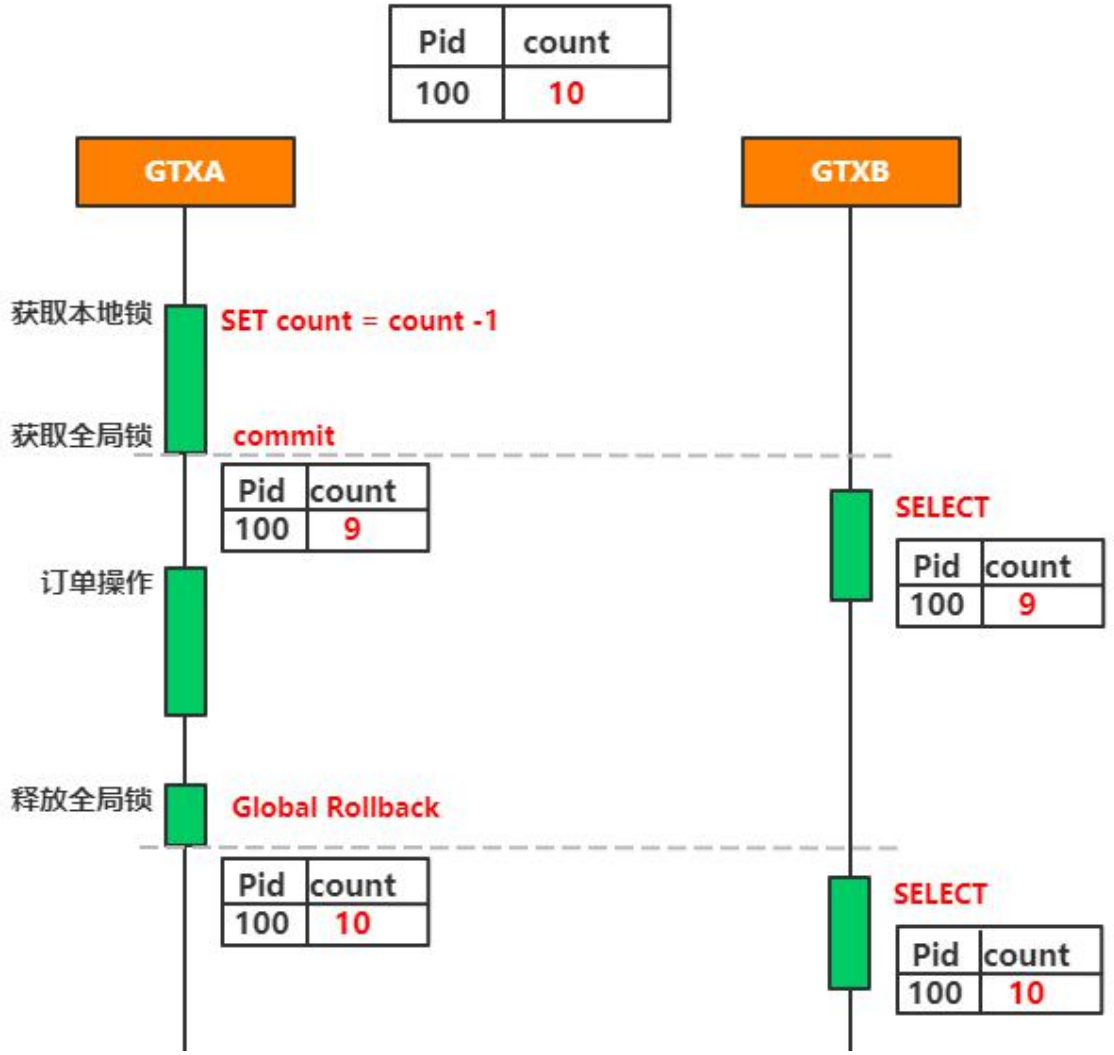
- 写隔离
  - 分支事务提交前拿到全局锁；
  - **拿全局锁超时，回滚本地事务，释放本地锁；**



Seata隔离性

- 读隔离（数据库读已提交）
  - 默认全局隔离级别是读未提交；

如何做到读已提交?

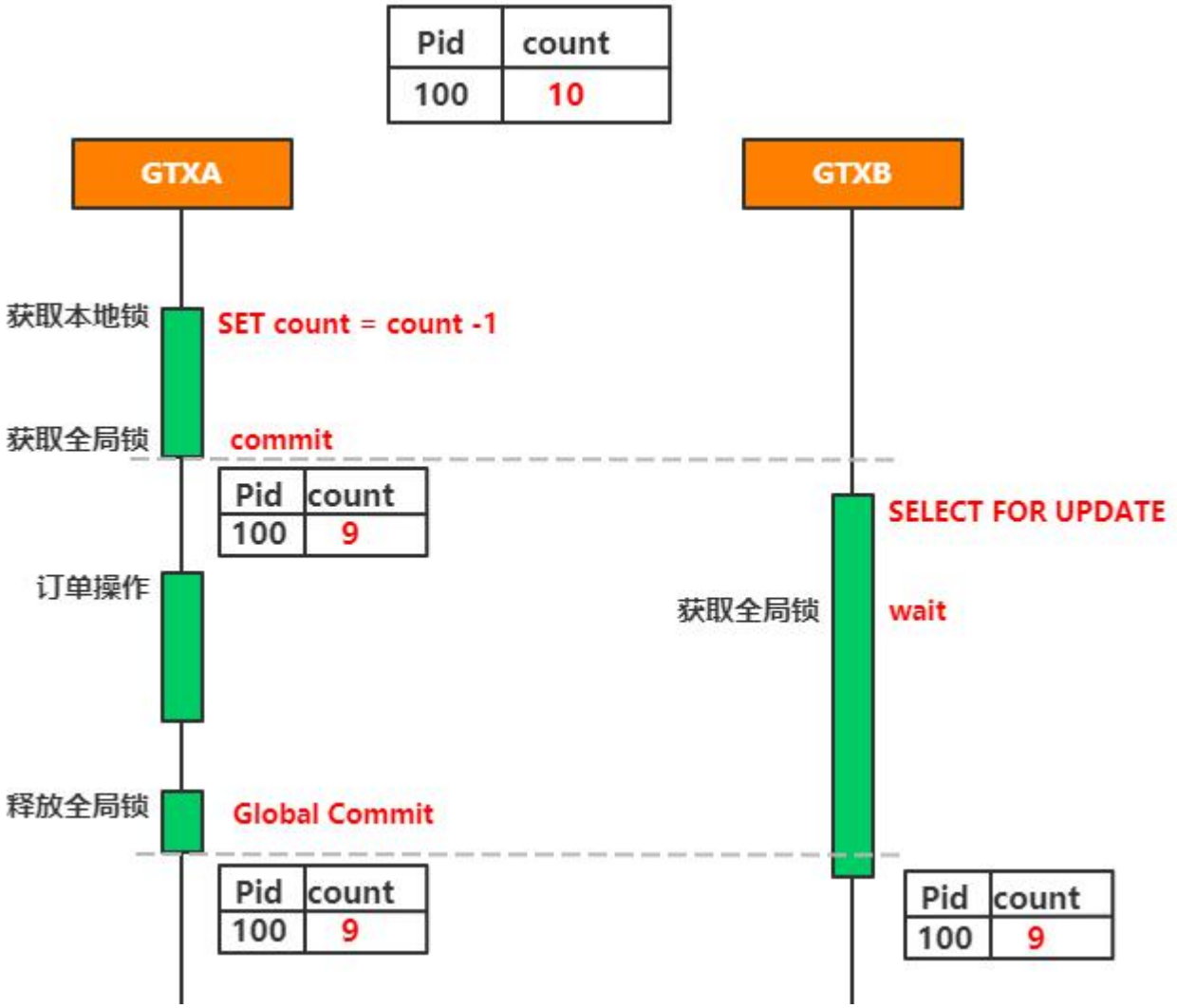


Seata隔离性

- 读隔离（数据库读已提交）
  - SELECT FOR UPDATE实现读已提交；

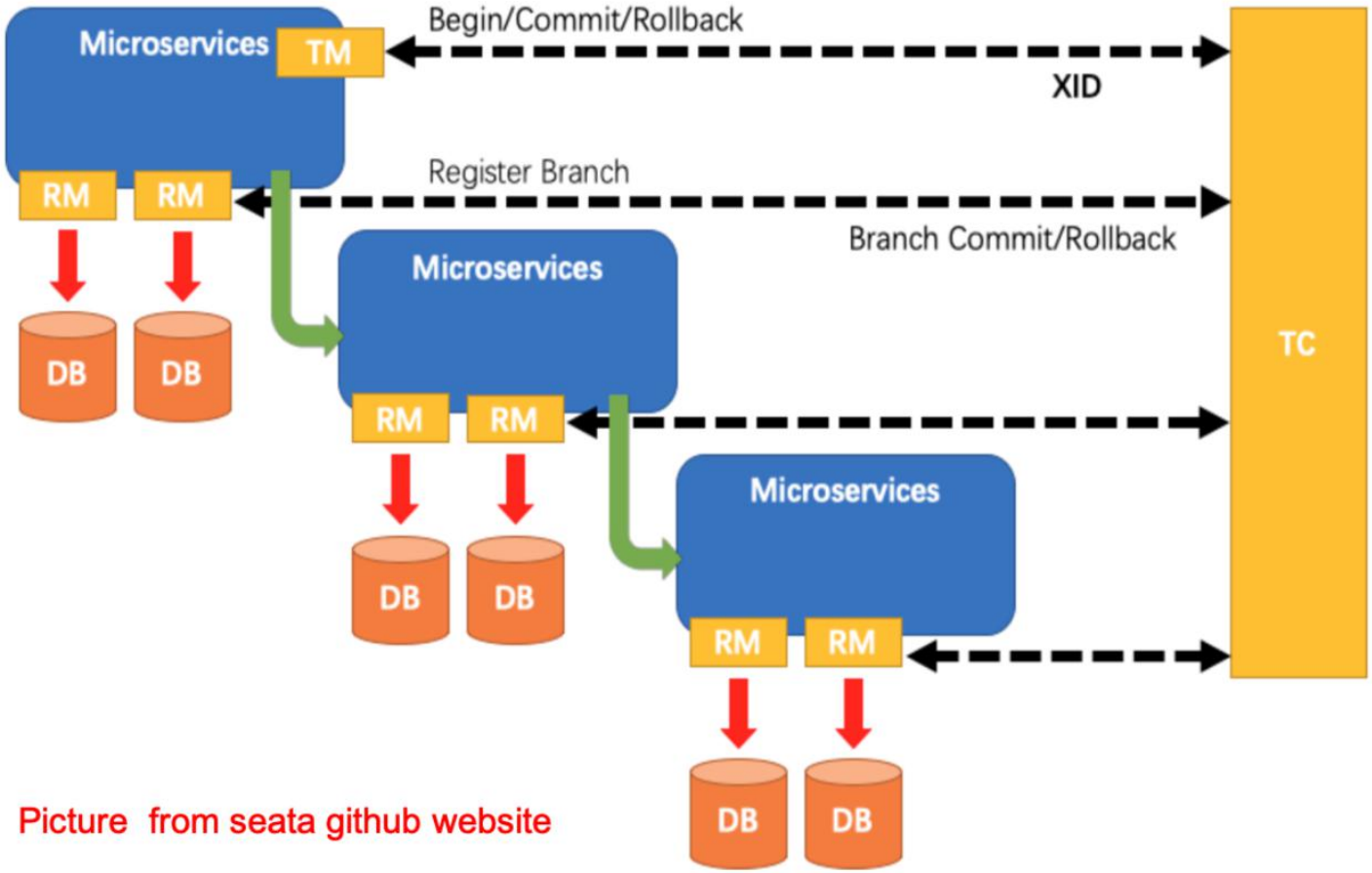
如何实现可重复读?

前镜像能否实现MVCC?



ShardingSphere与Seata集成

ShardingSphere如何集成Seata
将数据源封装为Seata的数据源



NX 奈学教育

03.企业级MySQL数据库高可用应用设计与实战

数据库冗余部署

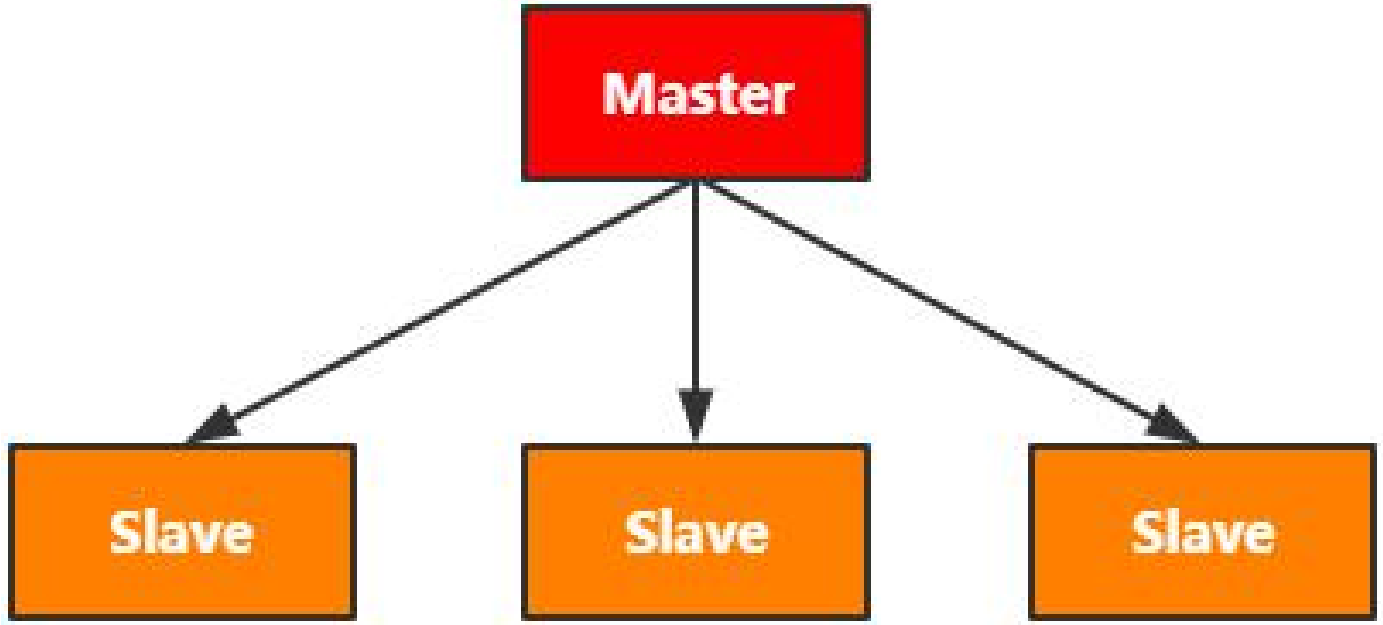
- 一主一从或多从
- 级联部署

尽量保证数据不丢失

承载更多的查询流量，提高系统访问性能

数据库冗余部署

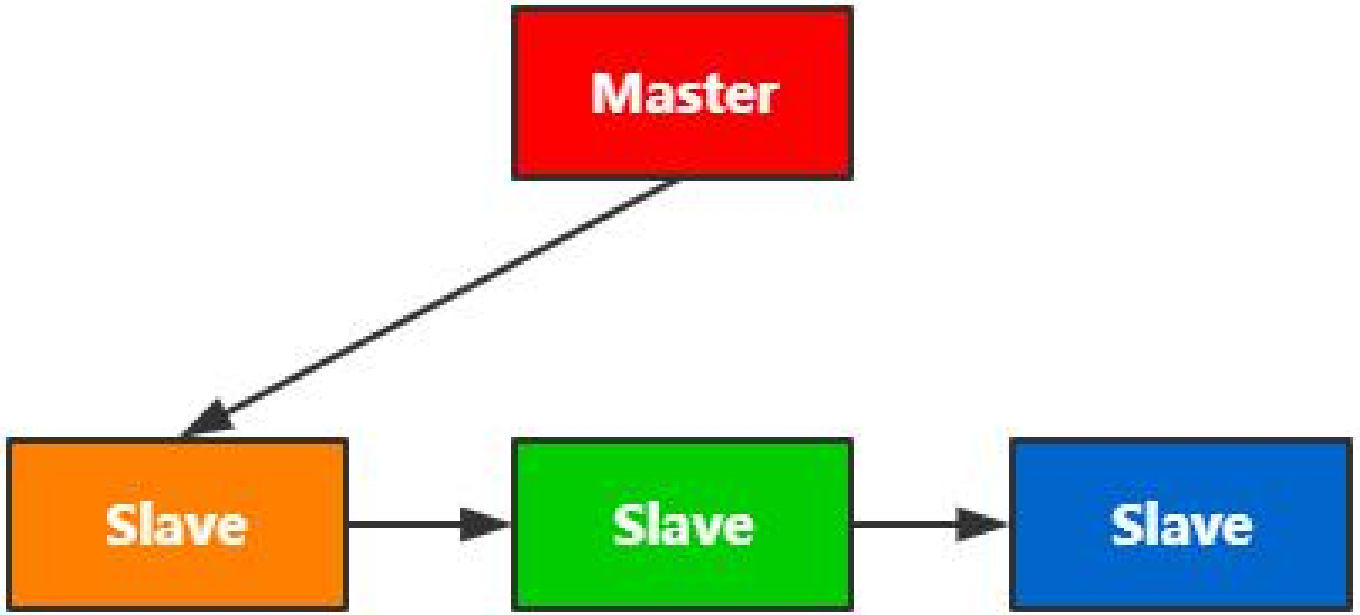
- **一主一从或多从**
- 级联部署



可靠性高，同时保证数据同步效率，但增加了主库的压力

数据库冗余部署

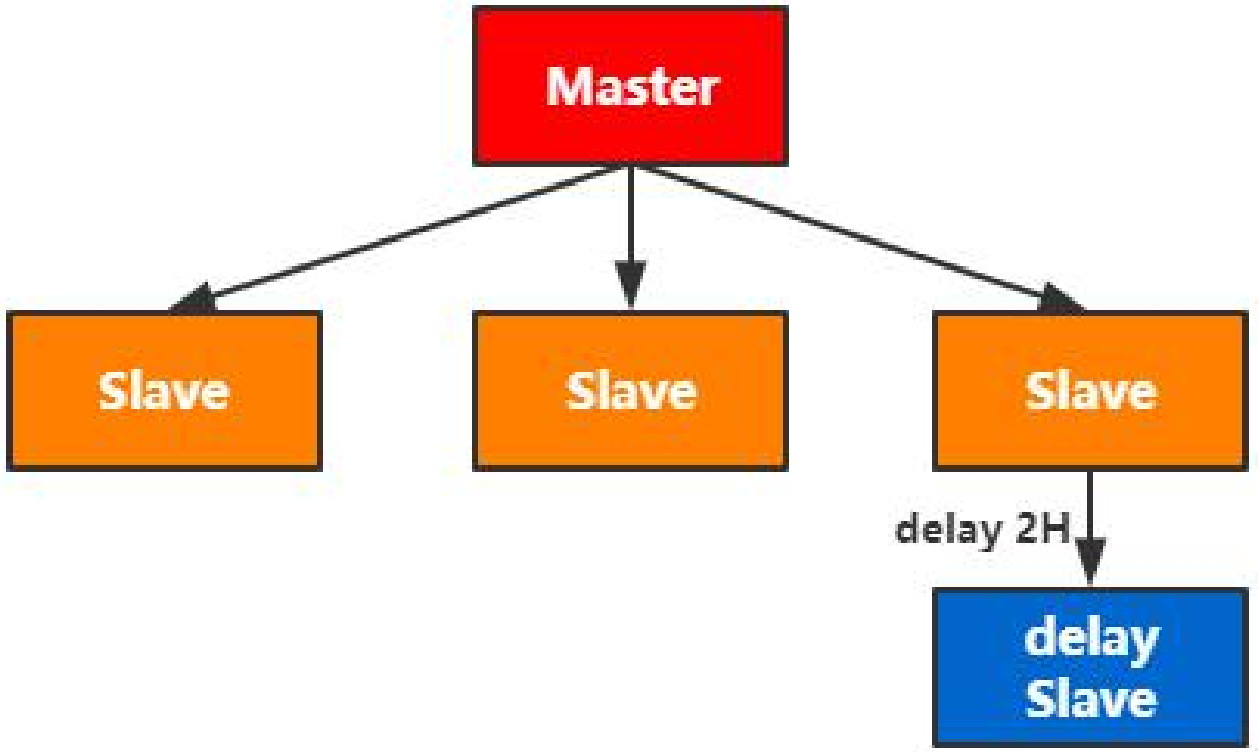
- 一主一从或多从
- **级联部署**



降低了主的压力，但串联同步方式可靠性相对较低

延时库部署案例

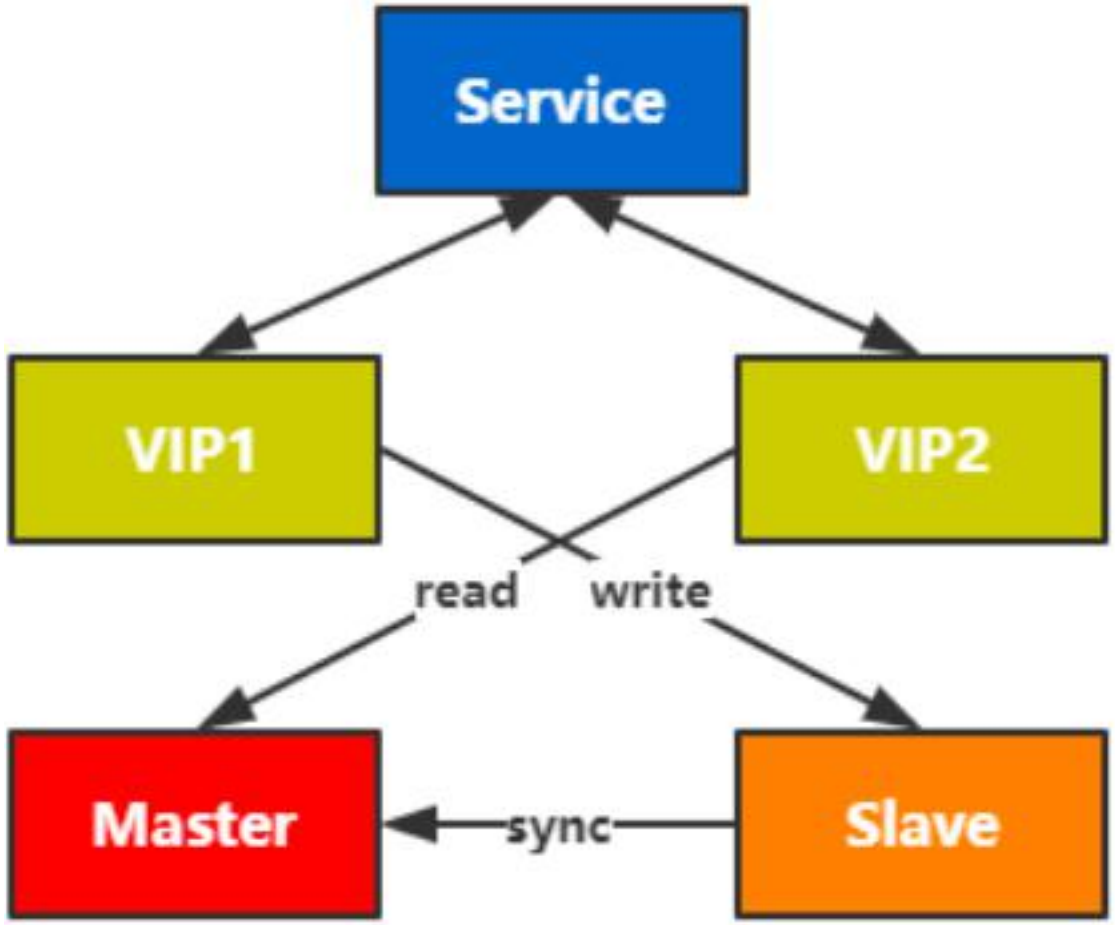
- 一主多从扛线上流量
- 级联部署延迟从做备份



可以利用延时从恢复被误删除的数据

数据库高可用方案

- 提供两个VIP
  - VIP1指向主库，VIP2指向从库
- 主库发生故障
  - VIP1和VIP2都指向从库
- 故障节点恢复
  - VIP2指向新的从库



NX 奈学教育

欢迎关注本人公众号
“架构之美”