极客大学

极客大学 Java 进阶训练营
第 16 课
超越分库分表-分布式事务

KimmKing

Apache Dubbo/ShardingSphere PMC

个人介绍

极客大学

Apache Dubbo/ShardingSphere PMC

前某集团高级技术总监/阿里架构师/某银行北京研发中心负责人

阿里云 MVP、腾讯 TVP、TGO 会员

10 多年研发管理和架构经验

熟悉海量并发低延迟交易系统的设计实现

极客大学

目录

极客大学

1. 分布式事务

为什么需要分布式事务

极客大学

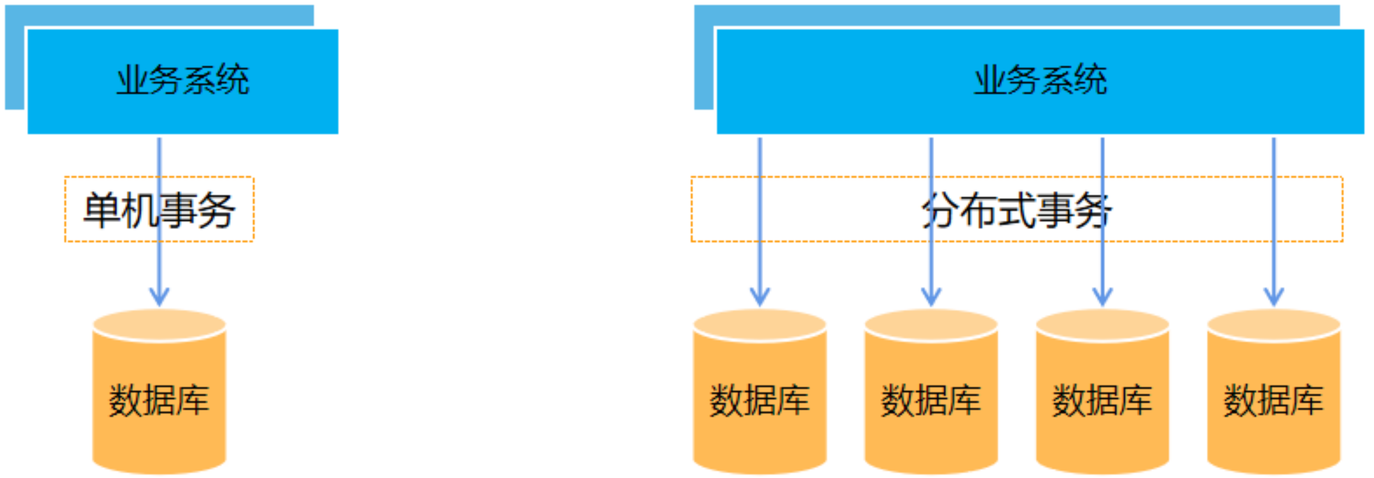
读写压力	多机集群	主从复制
高可用性	故障转移	主从切换
容量问题	数据库拆分	分库分表

一致性问题	分布式事务	XA/柔性事务

为什么需要分布式事务

业务系统的复杂度提升，数据量的增加，比如导致出现分布式事务。

随着互联网、金融等行业的快速发展，业务越来越复杂，一个完整的业务往往需要调用多个子业务或服务，随着业务的不断增多，涉及的服务及数据也越来越多，越来越复杂。传统的系统难以支撑，出现了应用和数据库等的分布式系统。分布式系统又带来了数据一致性的问题，从而产生了分布式事务。



什么叫分布式事务

分布式条件下，多个节点操作的整体事务一致性。

特别是在微服务场景下，业务 A 和业务 B 关联，事务 A 成功，事务 B 失败，由于跨系统，就会导致不被感知。此时从整体来看，数据是不一致的。

此时，业务系统只能拿到不完全的 A 服务数据，缺失 B 服务的数据。

如何实现分布式下的一致性

典型情况下是两个思路：

1、理想状态：直接想单机数据库事务一样，多个数据库自动通过某种协调机制，实现了跨数据库节点的一致性。

使用场景：要求严格的一致性，比如金融交易类业务。

2、一般情况：可以容忍一段时间的数据不一致，最终通过超时终止，调度补偿，等等方式，实现数据的最终状态一致性。

使用场景：准实时或非实时的处理，比如 T+1 的各类操作，或者电商类操作。

如何实现分布式下的一致性

1、强一致：XA

2、弱一致：

1) 不用事务，业务侧补偿冲正

2) 所谓的柔性事务，使用一套事务框架保证最终一致的事务

2. XA 分布式事务

XA 分布式事务协议

基于第一个强一致的思路，就有了基于数据库本身支持的协议，XA 分布式事务。

XA 整体设计思路可以概括为，如何在现有事务模型上微调扩展，实现分布式事务。

X/Open，即现在的 open group，是一个独立的组织，主要负责制定各种行业技术标准。X/Open 组织主要由各大知名公司或者厂商进行支持，这些组织不光遵循 X/Open 组织定义的行业技术标准，也参与到标准的制定。

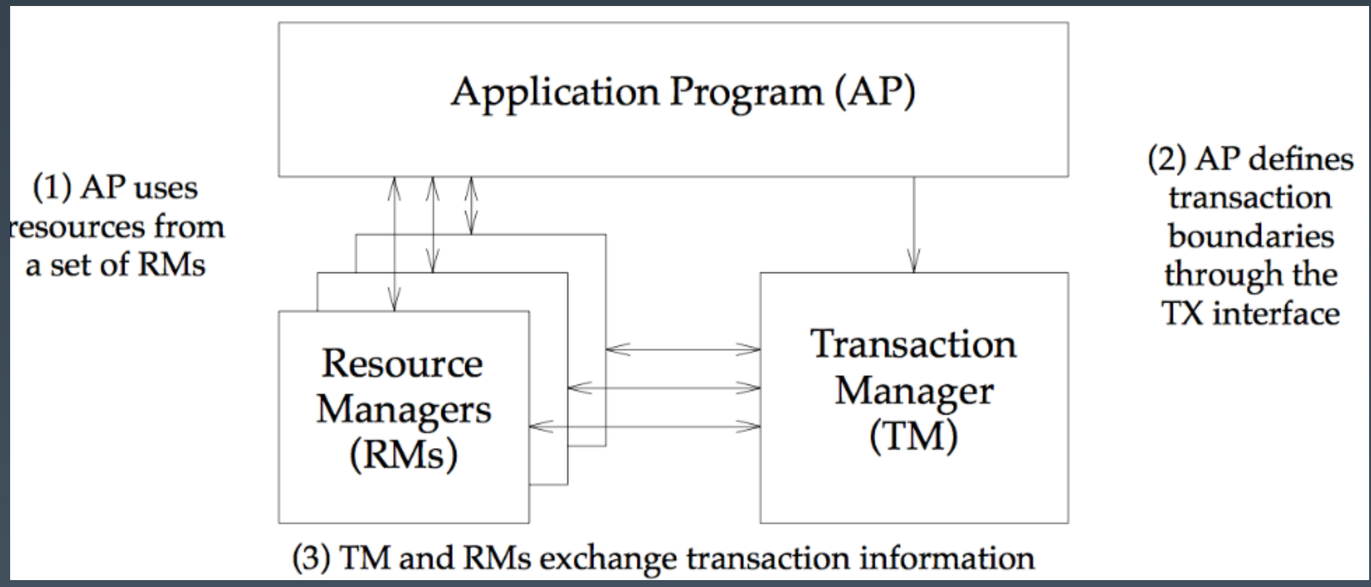
XA 分布式事务协议

应用程序(Application Program ，简称 AP)：用于定义事务边界(即定义事务的开始和结束)，并且在事务边界内对资源进行操作。

资源管理器(Resource Manager，简称 RM)：如数据库、文件系统等，并提供访问资源的方式

事务管理器(Transaction Manager ，简称 TM)：负责分配事务唯一标识，监控事务的执行进度，并负责事务的提交、回滚等。

X/Open DTP 模型与 XA 规范



XA 分布式事务协议

Name	Description
ax_reg	Register an RM with a TM.
ax_unreg	Unregister an RM with a TM.
xa_close	Terminate the AP's use of an RM.
xa_commit	Tell the RM to commit a transaction branch.
xa_complete	Test an asynchronous *xa_* operation for completion.
xa_end	Dissociate the thread from a transaction branch.
xa_forget	Permit the RM to discard its knowledge of a heuristically-completed transaction branch.
xa_open	Initialise an RM for use by an AP.
xa_prepare	Ask the RM to prepare to commit a transaction branch.
xa_recover	Get a list of XIDs the RM has prepared or heuristically completed.
xa_rollback	Tell the RM to roll back a transaction branch.
xa_start	Start or resume a transaction branch - associate an XID with future work that the thread requests of the RM.

XA 接口

xa_start：负责开启或者恢复一个事务分支

xa_end：负责取消当前线程与事务分支的关联

xa_prepare：询问 RM 是否准备好提交事务分支

xa_commit：通知 RM 提交事务分支

xa_rollback：通知 RM 回滚事务分支

xa_recover：需要恢复的 XA 事务

思考：为什么 XA 事务又叫两阶段事务？

XA 分布式事务协议

MySQL 从5.0.3开始支持 InnoDB 引擎的 XA 分布式事务，MySQL Connector/J 从5.0.0版本开始支持 XA。

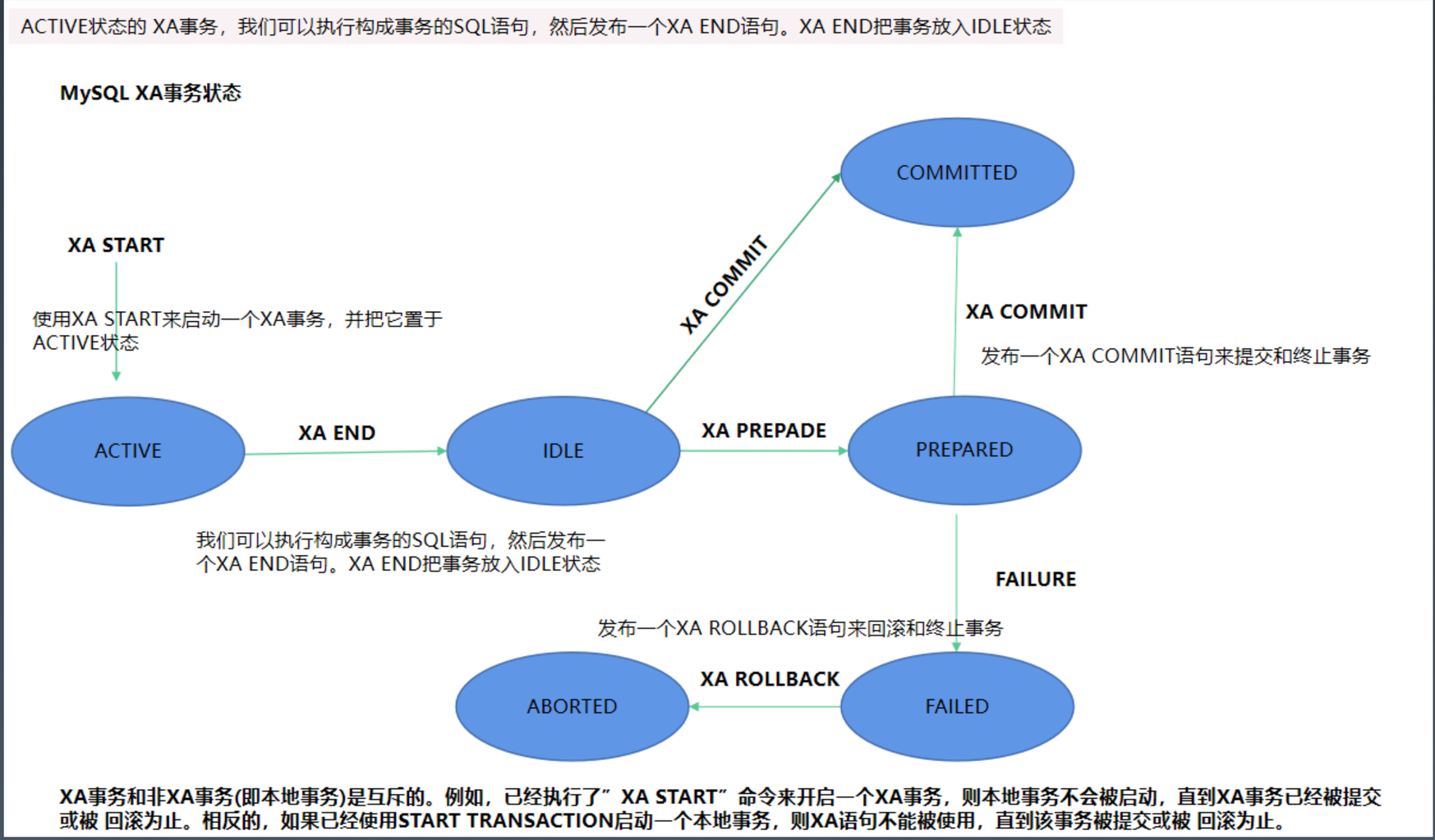
```
mysql> mysql> show engines;
+--------------------+---------+----------------------------------------------------------------+--------------+------+------------+
| Engine             | Support | Comment                                                        | Transactions | XA   | Savepoints |
+--------------------+---------+----------------------------------------------------------------+--------------+------+------------+
| InnoDB             | DEFAULT | Supports transactions, row-level locking, and foreign keys     | YES          | YES  | YES        |
| MRG_MYISAM         | YES     | Collection of identical MyISAM tables                          | NO           | NO   | NO         |
| MEMORY             | YES     | Hash based, stored in memory, useful for temporary tables      | NO           | NO   | NO         |
| BLACKHOLE          | YES     | /dev/null storage engine (anything you write to it disappears) | NO           | NO   | NO         |
| MyISAM             | YES     | MyISAM storage engine                                          | NO           | NO   | NO         |
| CSV                | YES     | CSV storage engine                                             | NO           | NO   | NO         |
| ARCHIVE            | YES     | Archive storage engine                                         | NO           | NO   | NO         |
| PERFORMANCE_SCHEMA | YES     | Performance Schema                                             | NO           | NO   | NO         |
| FEDERATED          | NO      | Federated MySQL storage engine                                 | NULL         | NULL | NULL       |
+--------------------+---------+----------------------------------------------------------------+--------------+------+------------+
```

在 DTP 模型中，MySQL 属于资源管理器(RM)。分布式事务中存在多个 RM，由事务管理器 TM 来统一进行协调。

```
XA {START|BEGIN} xid [JOIN|RESUME]   //开启XA事务，如果使用的是XA START而不是XA BEGIN，那么不支持[JOIN|RESUME]，xid是一个唯一值，表示事务分支标
XA END xid [SUSPEND [FOR MIGRATE]]   //结束一个XA事务，不支持[SUSPEND [FOR MIGRATE]]
XA PREPARE xid 准备提交
XA COMMIT xid [ONE PHASE] //提交，如果使用了ONE PHASE，则表示使用一阶段提交。两阶段提交协议中，如果只有一个RM参与，那么可以优化为一阶段提交
XA ROLLBACK xid  //回滚
XA RECOVER [CONVERT XID]  //列出所有处于PREPARE阶段的XA事务
```

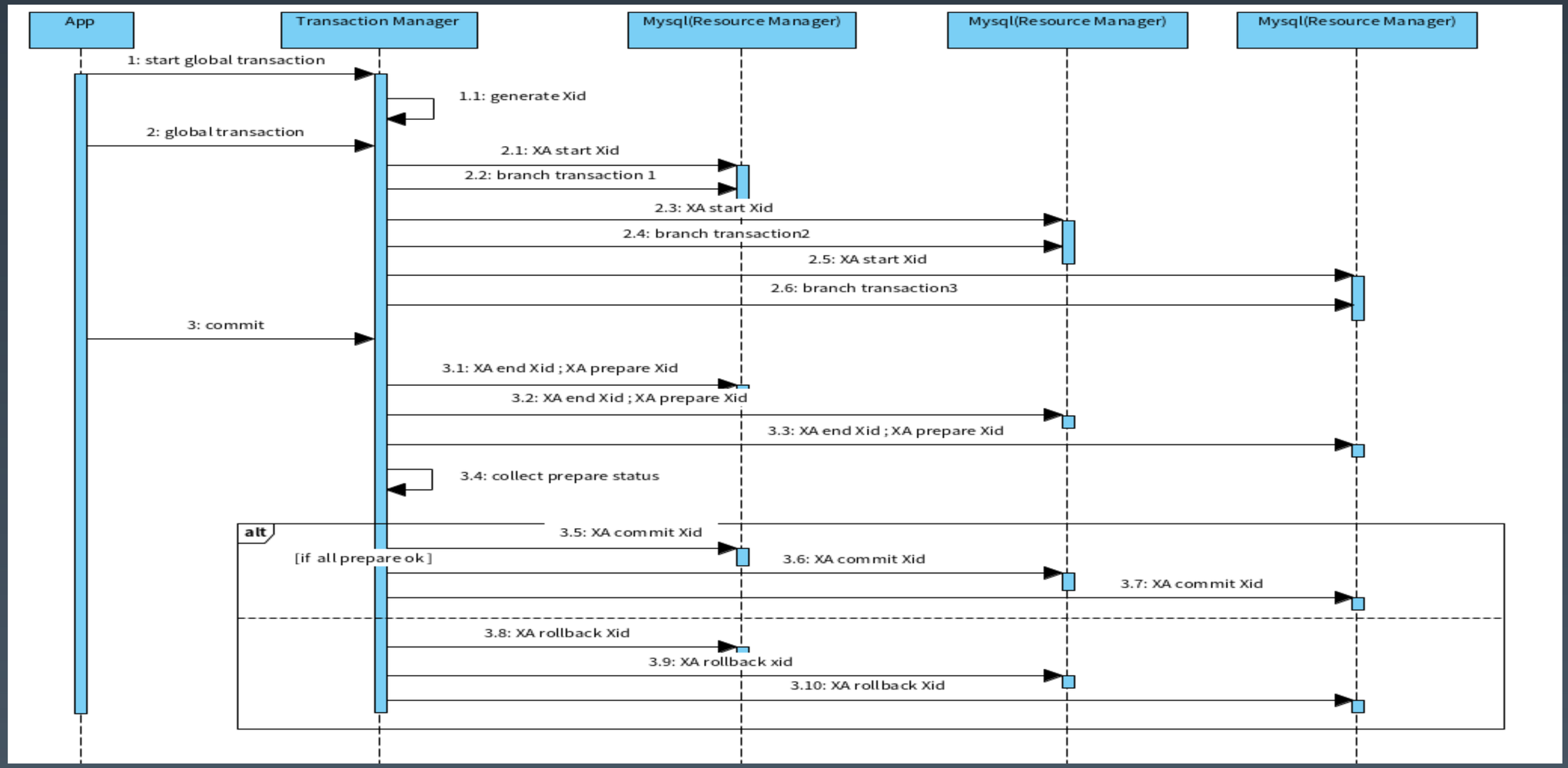
XA 分布式事务协议

- MySQL XA 事务状态



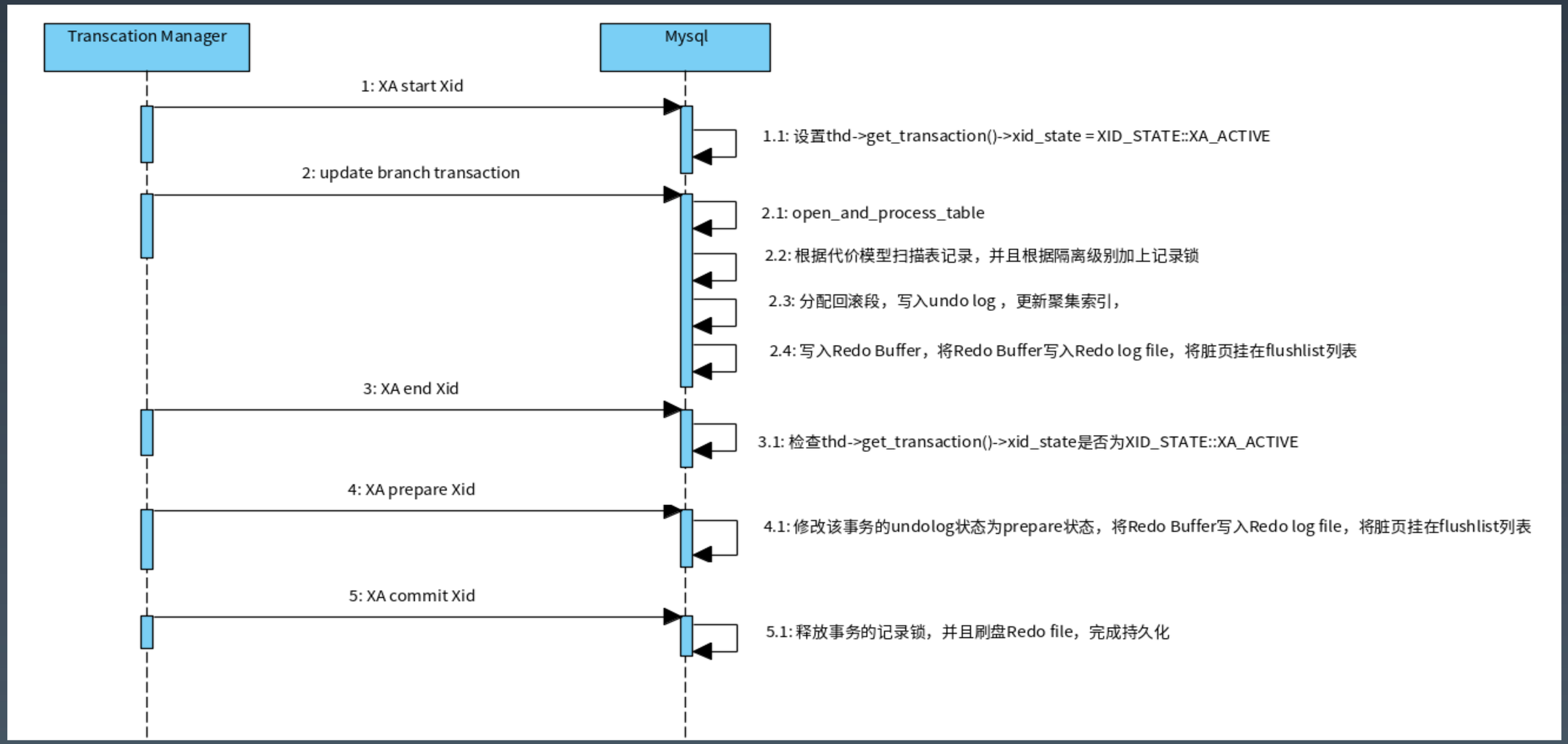
XA 分布式事务协议

- 完整的 XA 事务处理过程



XA 分布式事务协议

- 单个 MySQL 的内部操作



XA 分布式事务协议

- 思考一个问题：XA 过程中，事务失败怎么办？

1、业务 SQL 执行过程，某个 RM 崩溃怎么处理？

2、全部 prepare 后，某个 RM 崩溃怎么处理？

3、commit 时，某个 RM 崩溃怎么办？

XA 分布式事务协议

- 5.7对 MySQL XA 的优化/bug 修复

MySQL各版本对XA的支持与优化

MySQL < 5.7 版本会出现的问题（后续我们可以来重现）

- 已经prepare的事务，在客户端退出或者服务宕机的时候，2PC的事务会被回滚。
- 在服务器故障重启提交后，相应的Binlog被丢失

MySQL 5.6版本在客户端退出的时候，自动把已经prepare的事务回滚了，那么MySQL为什么要这样做？这主要取决于MySQL的内部实现，MySQL 5.7以前的版本，对于prepare的事务，MySQL是不会记录binlog的（官方说是减少fsync，起到了优化的作用）。只有当分布式事务提交的时候才会把前面的操作写入binlog信息，所以对于binlog来说，分布式事务与普通的事务没有区别，而prepare以前的操作信息都保存在连接的IO_CACHE中，如果这个时候客户端退出了，以前的binlog信息都会被丢失，再次重连后允许提交的话，会造成Binlog丢失，从而造成主从数据的不一致，所以官方在客户端退出的时候直接把已经prepare的事务都回滚了！

MySQL > 5.7 版本的优化（https://dev.mysql.com/worklog/task/?id=6860）

MySQL对于分布式事务，在prepare的时候就完成了写Binlog的操作，通过新增一种叫XA_prepare_log_event的event类型来实现，这是与以前版本的主要区别(以前版本prepare时不写Binlog)。！

XA 分布式事务协议

- 主流支持 XA 的框架，比较推荐 Atomikos 和 narayana

主流的开源XA分布式事务解决方案

ATOMIKOS	narayana	SEATA
TM：去中心化设计，性能较高	TM：去中心化设计，性能较高	TM：中心化设计，性能较差，BUG多
日志存储件：只支持文件	日志存储件：支持文件，数据库	日志存储件：支持文件，数据库
扩展性：较好	扩展性：一般	扩展性：一般
事务恢复：只支持单机事务恢复	事务恢复：支持集群模式恢复	事务恢复：问题很多，未能正确恢复
标准的XA实现	标准的XA实现	非标准的XA实现

XA 分布式事务协议

- 注意：XA 默认不会改变隔离级别

XA协议存在的问题

1.同步阻塞问题 - 一般情况下，不需要调高隔离级别

全局事务内部包含了多个独立的事务分支，这一组事务分支要不都成功，要不都失败。各个事务分支的ACID特性共同构成了全局事务的ACID特性。
也就是将单个事务分支的支持的ACID特性提升一个层次(up a level)到分布式事务的范畴。即使在非分布事务中(即本地事务)，如果对操作读很敏感，我们也需要将事务隔离级别设置为SERIALIZABLE。而对于分布式事务来说，更是如此，可重复读隔离级别不足以保证分布式事务一致性。
也就是说，如果我们使用mysql来支持XA分布式事务的话，那么最好将事务隔离级别设置为SERIALIZABLE。 地球人都知道，
SERIALIZABLE(串行化)是四个事务隔离级别中最高的一个级别，也是执行效率最低的一个级别

2.单点故障 - 成熟的XA框架需要考虑TM的高可用性

由于协调者的重要性，一旦协调者TM发生故障。参与者RM会一直阻塞下去。尤其在第二阶段，协调者发生故障，那么所有的参与者还都处于锁定事务资源的状态中，而无法继续完成事务操作。（如果是协调者挂掉，可以重新选举一个协调者，但是无法解决因为协调者宕机导致的参与者处于阻塞状态的问题）

3.数据不一致 - 极端情况下，一定有事务失败问题，需要监控和人工处理

在二阶段提交的阶段二中，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，这回导致只有一部分参与者接受到了commit请求。而在这部分参与者接到commit请求之后就会执行commit操作。但是其他部分未接到commit请求的机器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性的现象。

3. BASE 柔性事务

BASE 柔性事务

本地事务 -> XA(2PC) -> BASE

如果将实现了 ACID 的事务要素的事务称为刚性事务的话，那么基于 BASE 事务要素的事务则称为柔性事务。 BASE 是基本可用、柔性状态和最终一致性这三个要素的缩写。

- 基本可用（Basically Available）保证分布式事务参与方不一定同时在线。
- 柔性状态（Soft state）则允许系统状态更新有一定的延时，这个延时对客户来说不一定能够察觉。
- 而最终一致性（Eventually consistent）通常是通过消息传递的方式保证系统的最终一致性。

在 ACID 事务中对隔离性的要求很高，在事务执行过程中，必须将所有的资源锁定。 柔性事务的理念则是通过业务逻辑将互斥锁操作从资源层面上移至业务层面。通过放宽对强一致性要求，来换取系统吞吐量的提升。

BASE 柔性事务

本地事务 -> XA(2PC) -> BASE

	本地事务	*两（三）阶段事务*	*柔性事务*
业务改造	无	无	实现相关接口
一致性	不支持	支持	最终一致
隔离性	不支持	支持	业务方保证
并发性能	无影响	严重衰退	略微衰退
适合场景	业务方处理不一致	短事务 & 低并发	长事务 & 高并发

BASE 柔性事务

BASE 柔性事务常见模式

1、TCC

通过手动补偿处理

2、AT

通过自动补偿处理

4. TCC/AT 以及相关框架

什么是 TCC

BASE 柔性事务 TCC

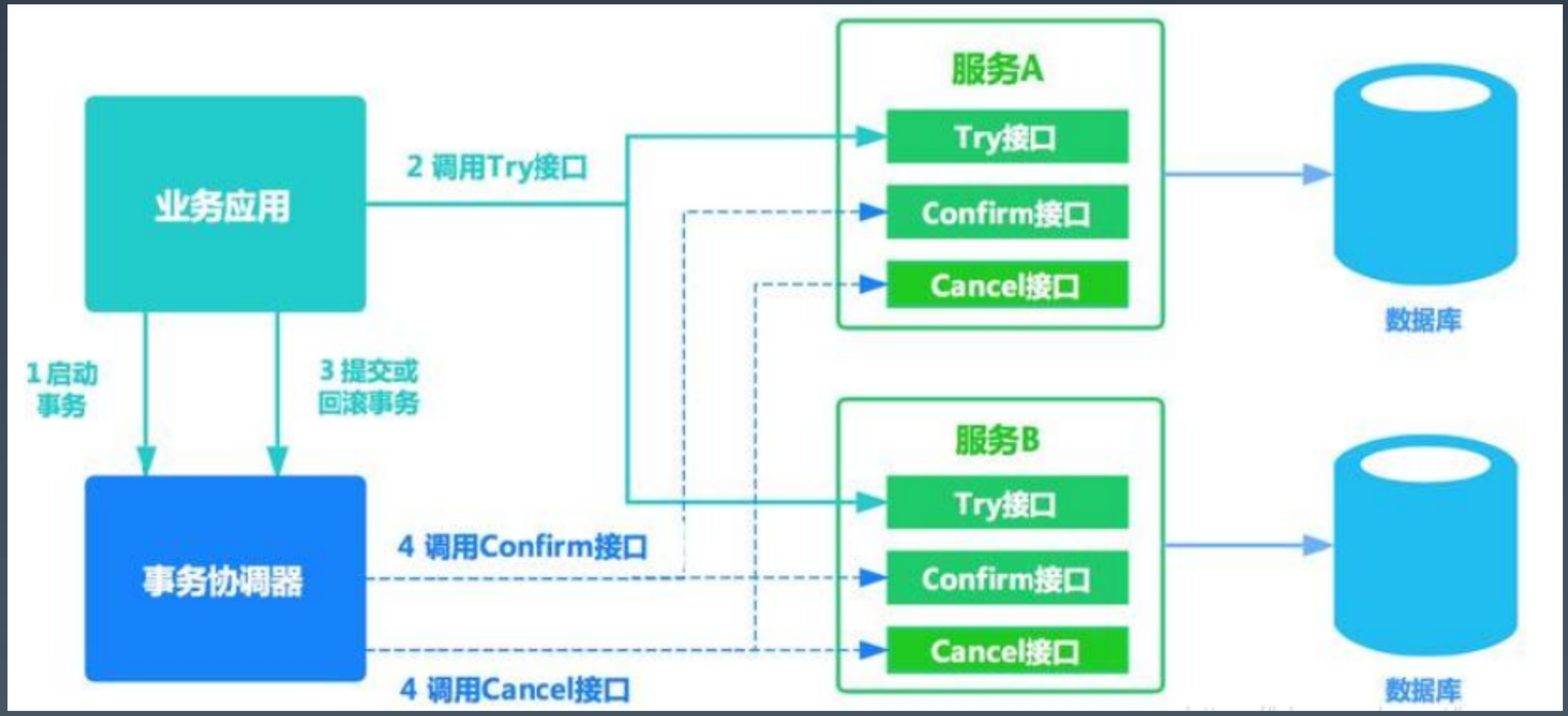
TCC 模式即将每个服务业务操作分为两个阶段，第一个阶段检查并预留相关资源，第二阶段根据所有服务业务的 Try 状态来操作，如果都成功，则进行 Confirm 操作，如果任意一个 Try 发生错误，则全部 Cancel。

TCC 使用要求就是业务接口都必须实现三段逻辑：

1. 准备操作 Try：完成所有业务检查，预留必须的业务资源。
2. 确认操作 Confirm：真正执行的业务逻辑，不做任何业务检查，只使用 Try 阶段预留的业务资源。因此，只要 Try 操作成功，Confirm 必须能成功。另外，Confirm 操作需满足幂等性，保证一笔分布式事务能且只能成功一次。
3. 取消操作 Cancel：释放 Try 阶段预留的业务资源。同样的，Cancel 操作也需要满足幂等性。

什么是 TCC

TCC 不依赖 RM 对分布式事务的支持，而是通过对业务逻辑的分解来实现分布式事务，不同于AT的是就是需要自行定义各个阶段的逻辑，对业务有侵入。



什么是 TCC

TCC 需要注意的几个问题：

1、允许空回滚

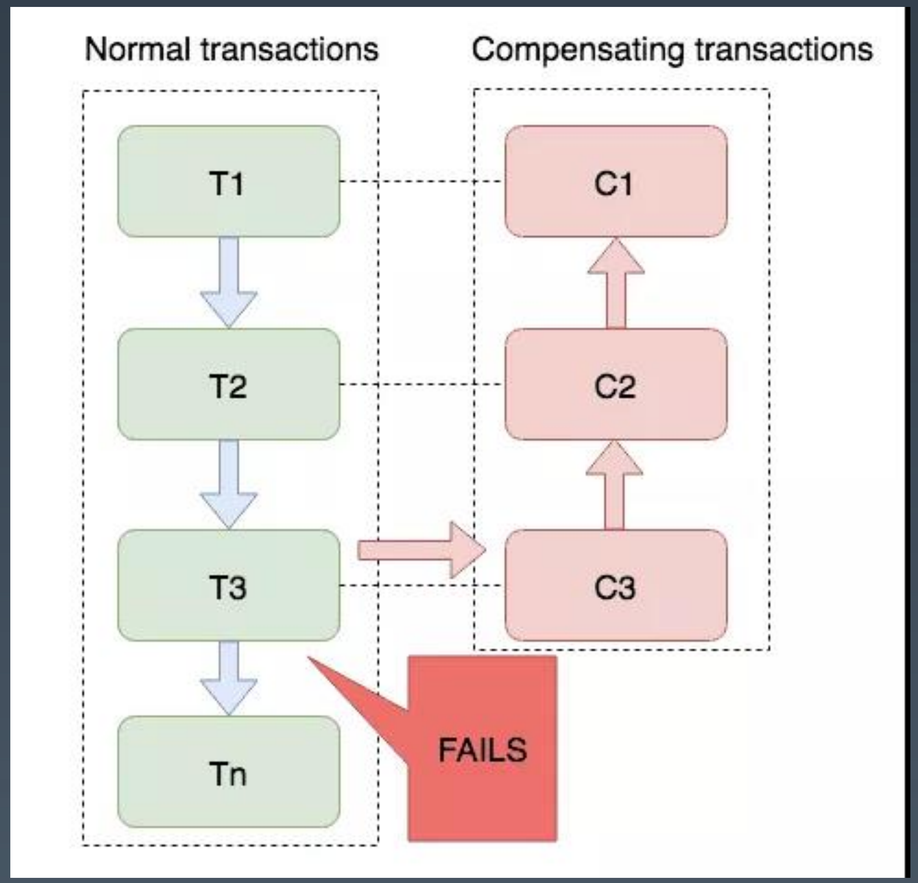
2、防悬挂控制

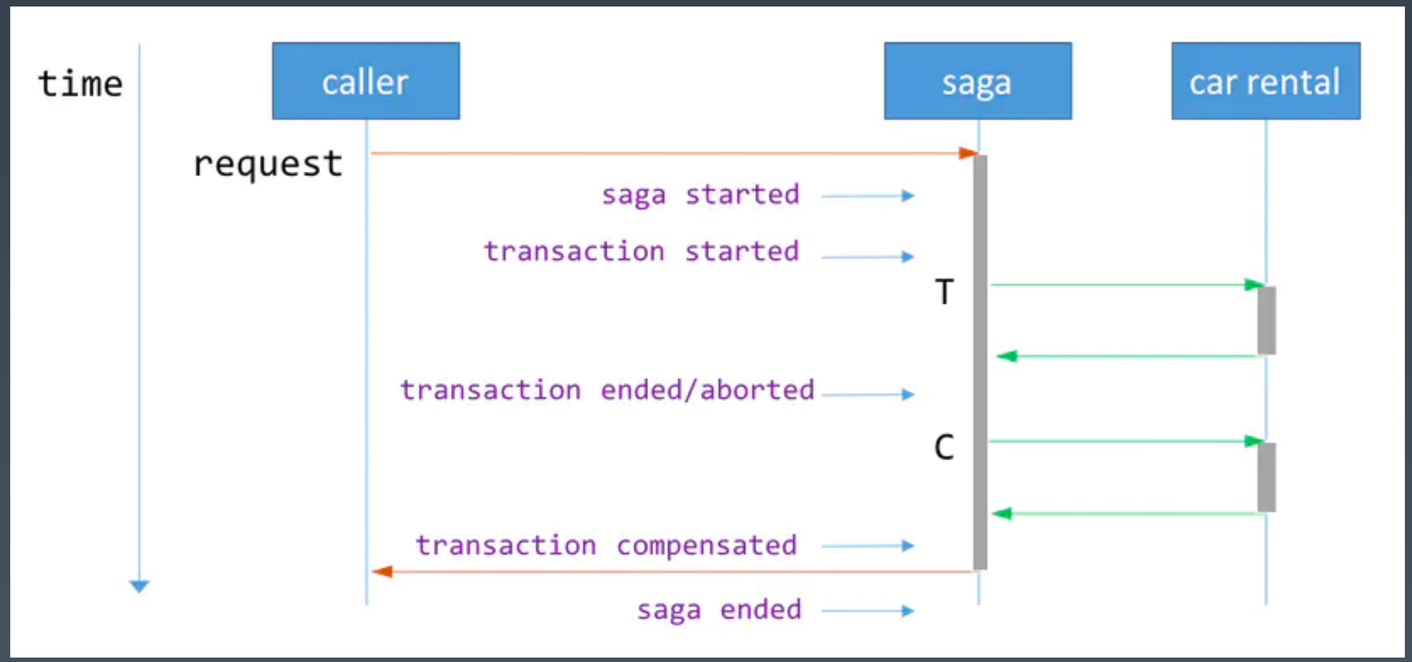
3、幂等设计

什么是 SAGA

Saga 模式没有 try 阶段，直接提交事务。

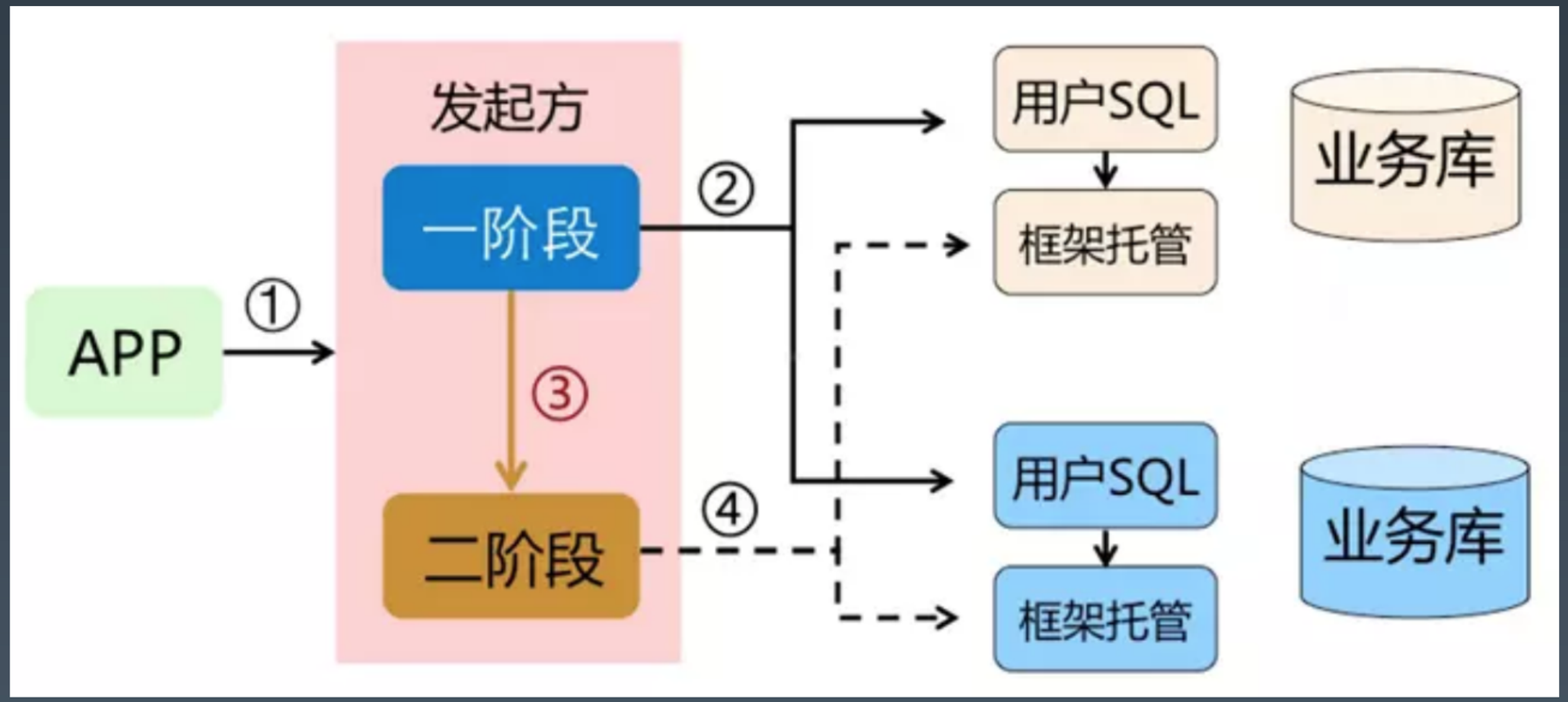
复杂情况下，对回滚操作的设计要求较高。





什么是 AT

AT 模式就是两阶段提交，自动生成反向 SQL



柔性事务下隔离级别

极客大学

事务特性

- 原子性（Atomicity）：正常情况下保证。
- 一致性（Consistency），在某个时间点，会出现 A 库和 B 库的数据违反一致性要求的情况，但是最终是一致的。
- 隔离性（Isolation），在某个时间点，A 事务能够读到B事务部分提交的结果。
- 持久性（Durability），和本地事务一样，只要 commit 则数据被持久。

隔离级别

- 一般情况下都是读已提交（全局锁）、读未提交（无全局锁）。

Seata

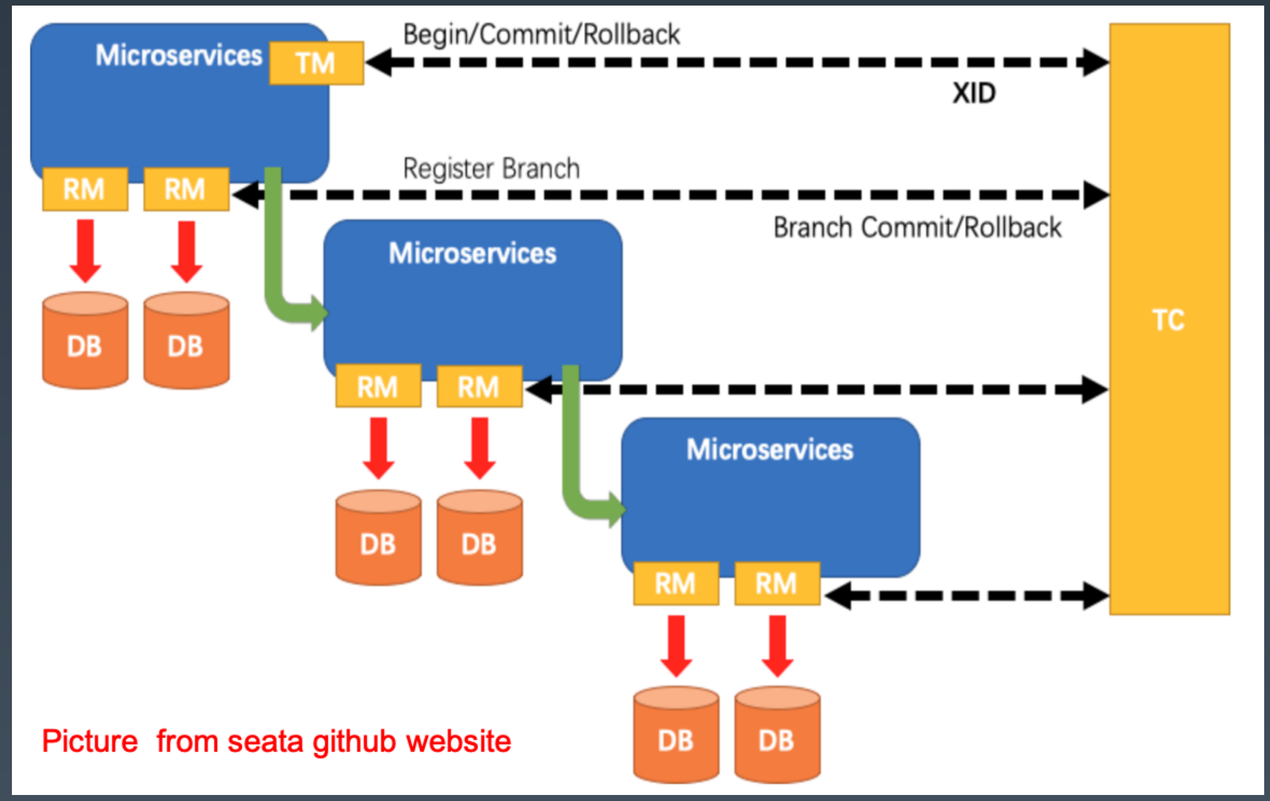
Seata-TCC/AT 柔性事务

Seata 是阿里集团和蚂蚁金服联合打造的分布式事务框架。 其 AT 事务的目标是在微服务架构下，提供增量的事务 ACID 语意，让开发者像使用本地事务一样，使用分布式事务，核心理念同 Apache ShardingSphere 一脉相承。

Seata AT 事务模型包含TM (事务管理器)，RM (资源管理器) 和 TC (事务协调器)。 TC 是一个独立部署的服务，TM 和 RM 以 jar 包的方式同业务应用一同部署，它们同 TC 建立长连接，在整个事务生命周期内，保持远程通信。 TM 是全局事务的发起方，负责全局事务的开启，提交和回滚。 RM 是全局事务的参与者，负责分支事务的执行结果上报，并且通过 TC 的协调进行分支事务的提交和回滚。

Seata

Seata 管理的分布式事务的典型生命周期：



TM 要求 TC 开始一个全新的全局事务。

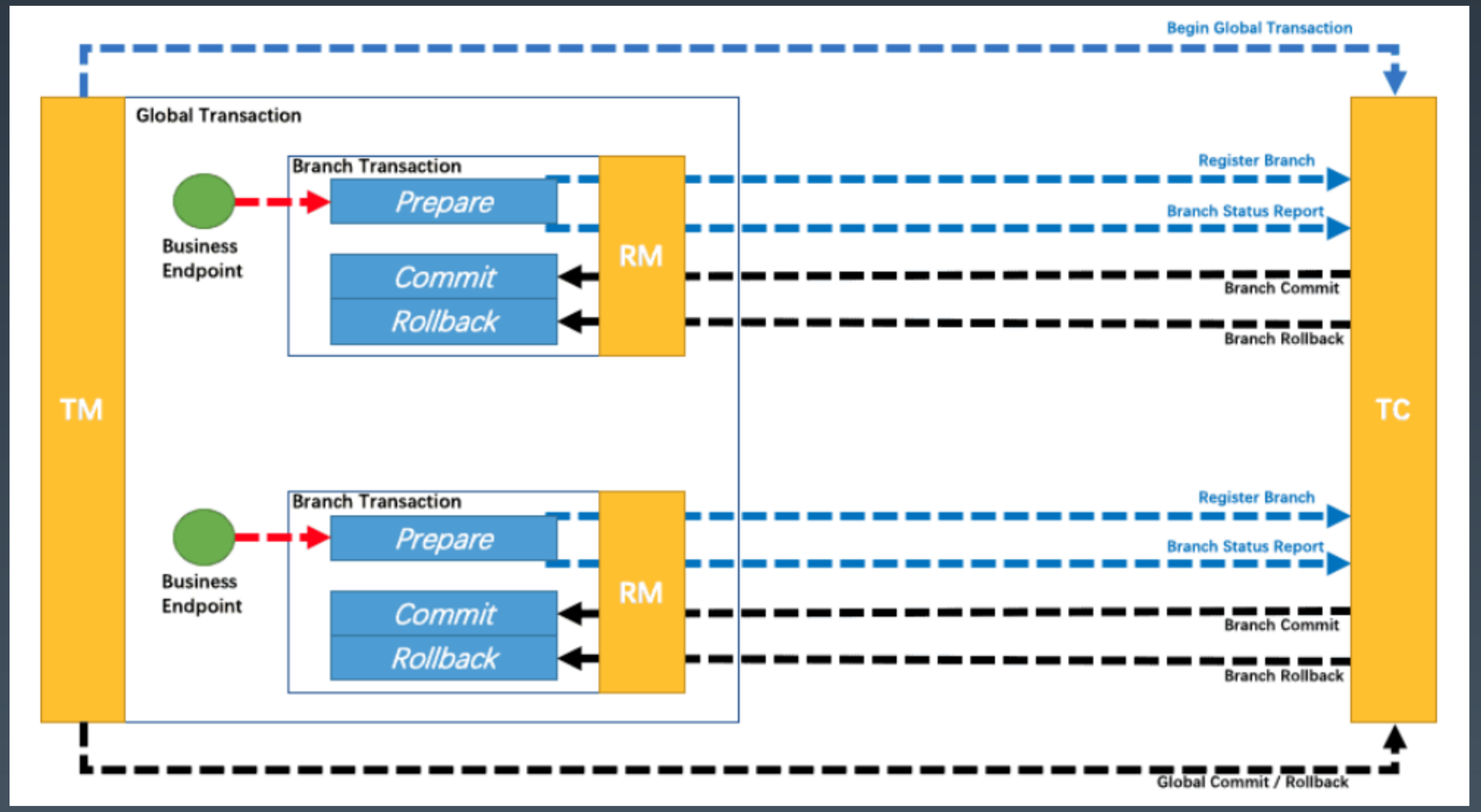
TC 生成一个代表该全局事务的 XID。

XID 贯穿于微服务的整个调用链。

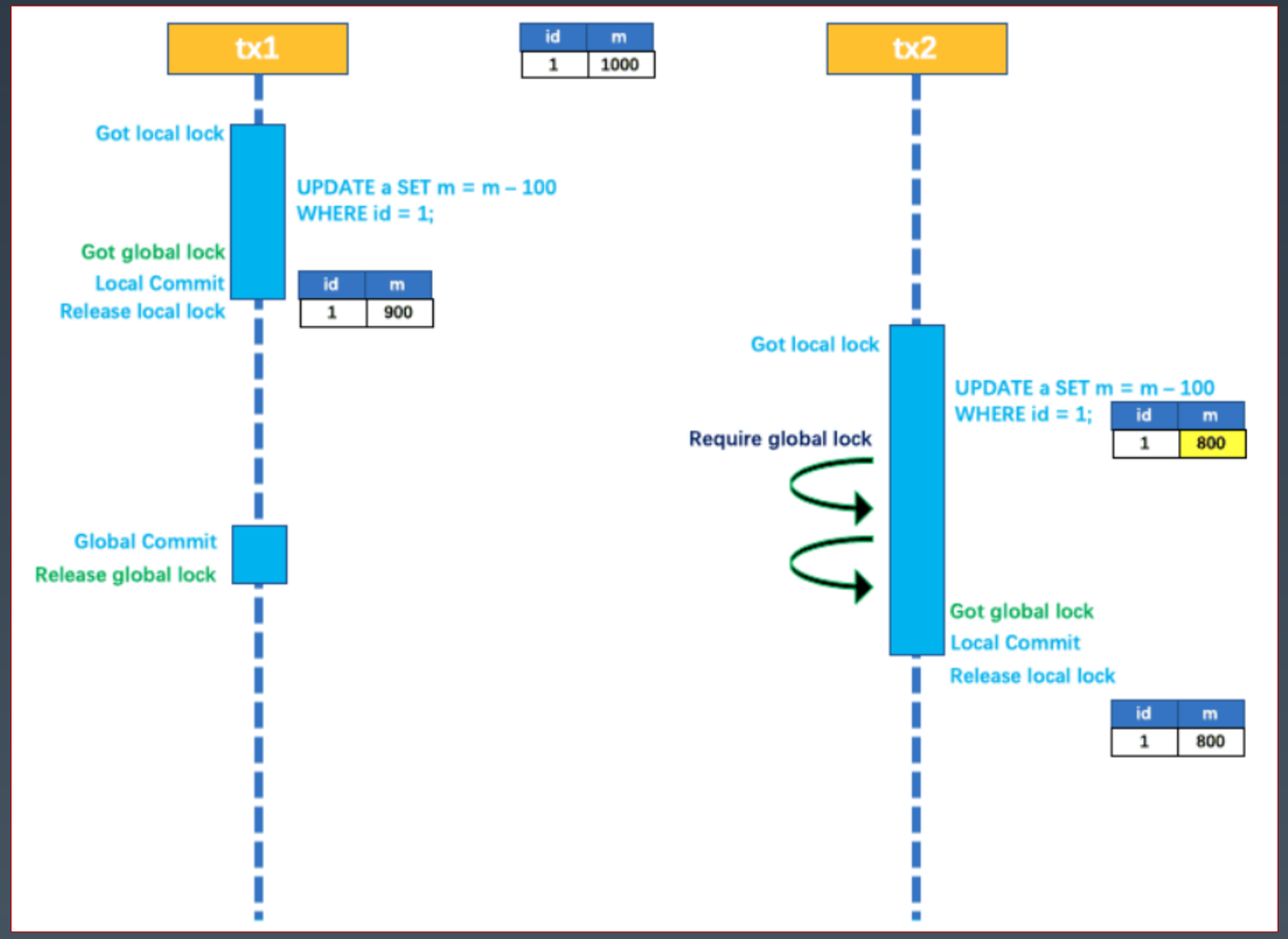
TM 要求 TC 提交或回滚 XID 对应全局事务。

TC 驱动 XID 对应的全局事务下的所有分支事务完成提交或回滚。

Seata - TCC


Begin Global Transaction
Global Transaction
TM
Branch Transaction
Prepare
Commit
Rollback
Business Endpoint
RM
Register Branch
Branch Status Report
Branch Commit
Branch Rollback
TC
Global Commit / Rollback

Seata - AT 原理



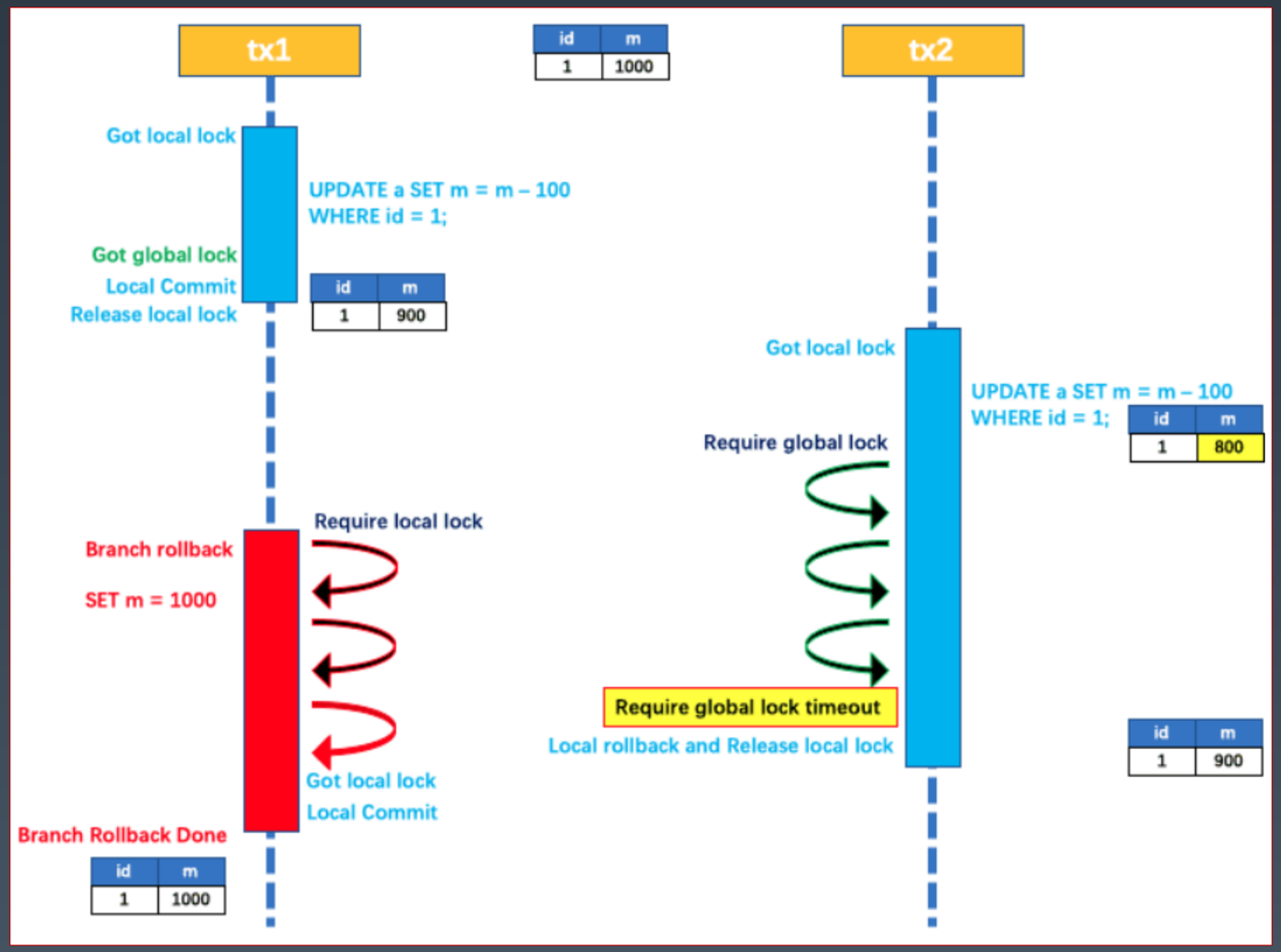
两阶段提交协议的演变：

一阶段：业务数据和回滚日志记录在同一个本地事务中提交，释放本地锁和连接资源。

二阶段：

提交异步化，非常快速地完成。
回滚通过一阶段的回滚日志进行反向补偿。

Seata - AT 原理



通过全局锁的方式，实现读写隔离。

1、本地锁控制本地操作；

2、全局锁控制全局提交。

hmily

Hmily 是一个高性能分布式事务框架，开源于2017年，目前有 2800 个 Star，基于 TCC 原理实现，使用 Java 语言开发（JDK1.8+），天然支持 Dubbo、SpringCloud、Motan 等微服务框架的分布式事务。

hmily 功能

支持嵌套事务(Nested transaction support)等复杂场景

支持 RPC 事务恢复，超时异常恢复等，具有高稳定性

基于异步 Confirm 和 Cancel 设计，相比其他方式具有更高性能

基于 SPI 和 API 机制设计，定制性强，具有高扩展性

本地事务的多种存储支持 : redis/mongodb/zookeeper/file/mysql

事务日志的多种序列化支持 ：java/hessian/kryo/protostuff

基于高性能组件 disruptor 的异步日志性能良好

实现了 SpringBoot-Starter，开箱即用，集成方便

采用 Aspect AOP 切面思想与 Spring 无缝集成，天然支持集群

实现了基于 VUE 的 UI 界面，方便监控和管理

hmily

MainService：事务发起者（业务服务）

TxManage：事务协调者

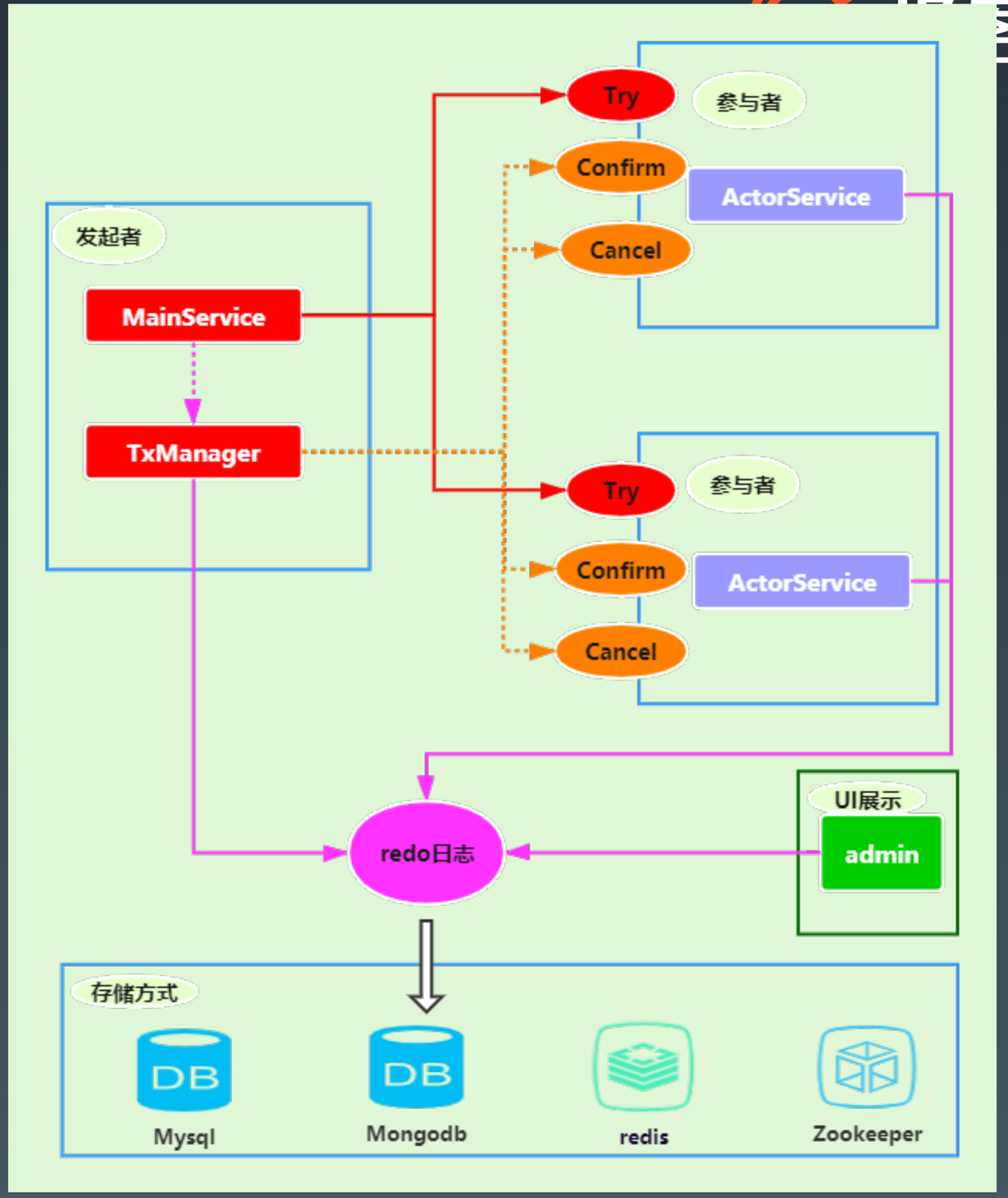
ActorService：事务参与者（多个业务服务）

Try：事务执行

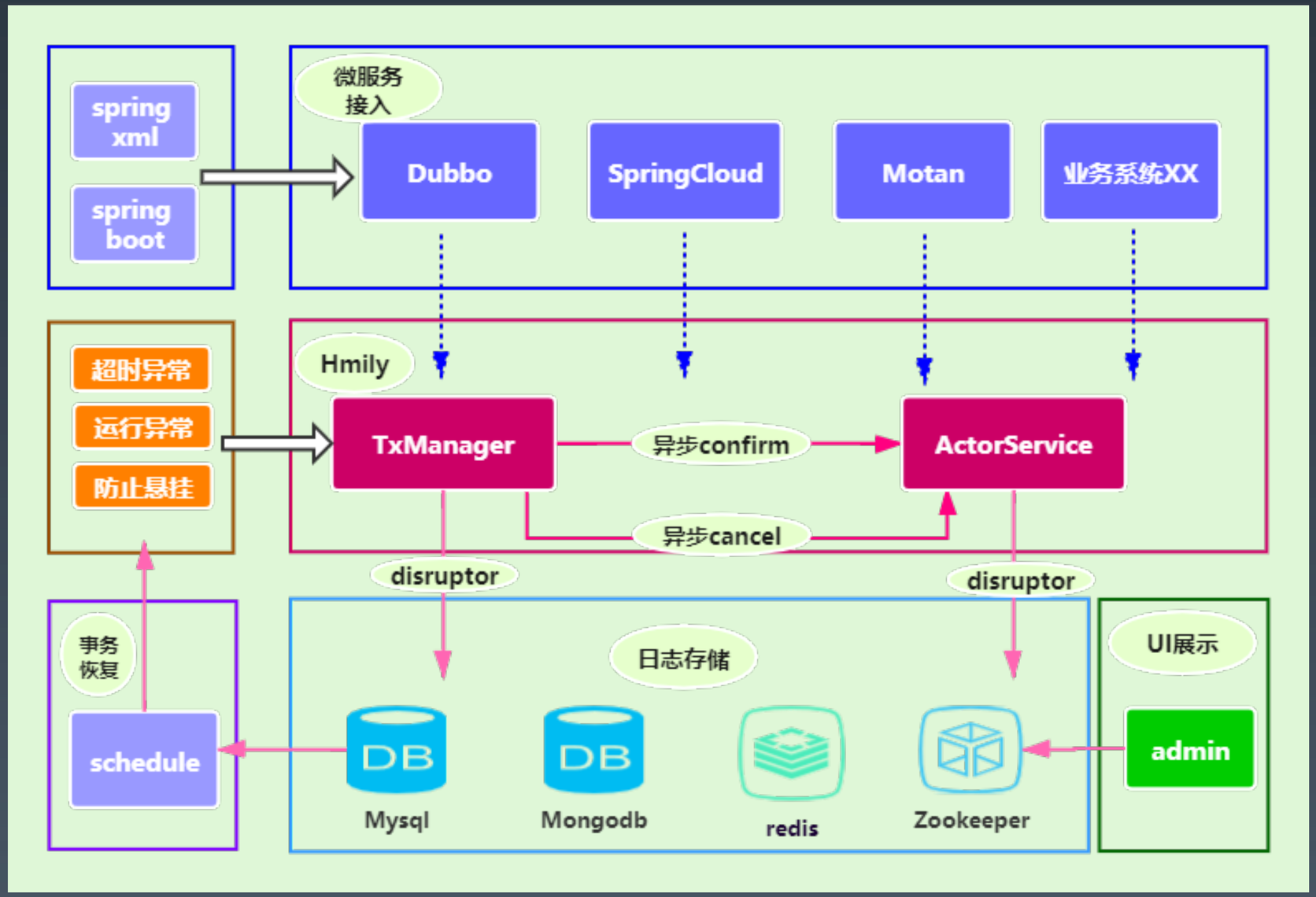
Confirm：事务确认

Cancel：事务回滚

Redo 日志 ：可以选择任意一种进行存储



hmily


spring xml
spring boot
微服务接入
Dubbo
SpringCloud
Motan
业务系统XX
超时异常
运行异常
防止悬挂
Hmily
TxManager
异步confirm
ActorService
异步cancel
disruptor
disruptor
事务恢复
schedule
日志存储
DB
Mysql
DB
Mongodb
redis
Zookeeper
UI展示
admin

5. ShardingSphere 对分布式事务的支持

ShardingSphere 对分布式事务的支持

由于应用的场景不同，需要开发者能够合理的在性能与功能之间权衡各种分布式事务。

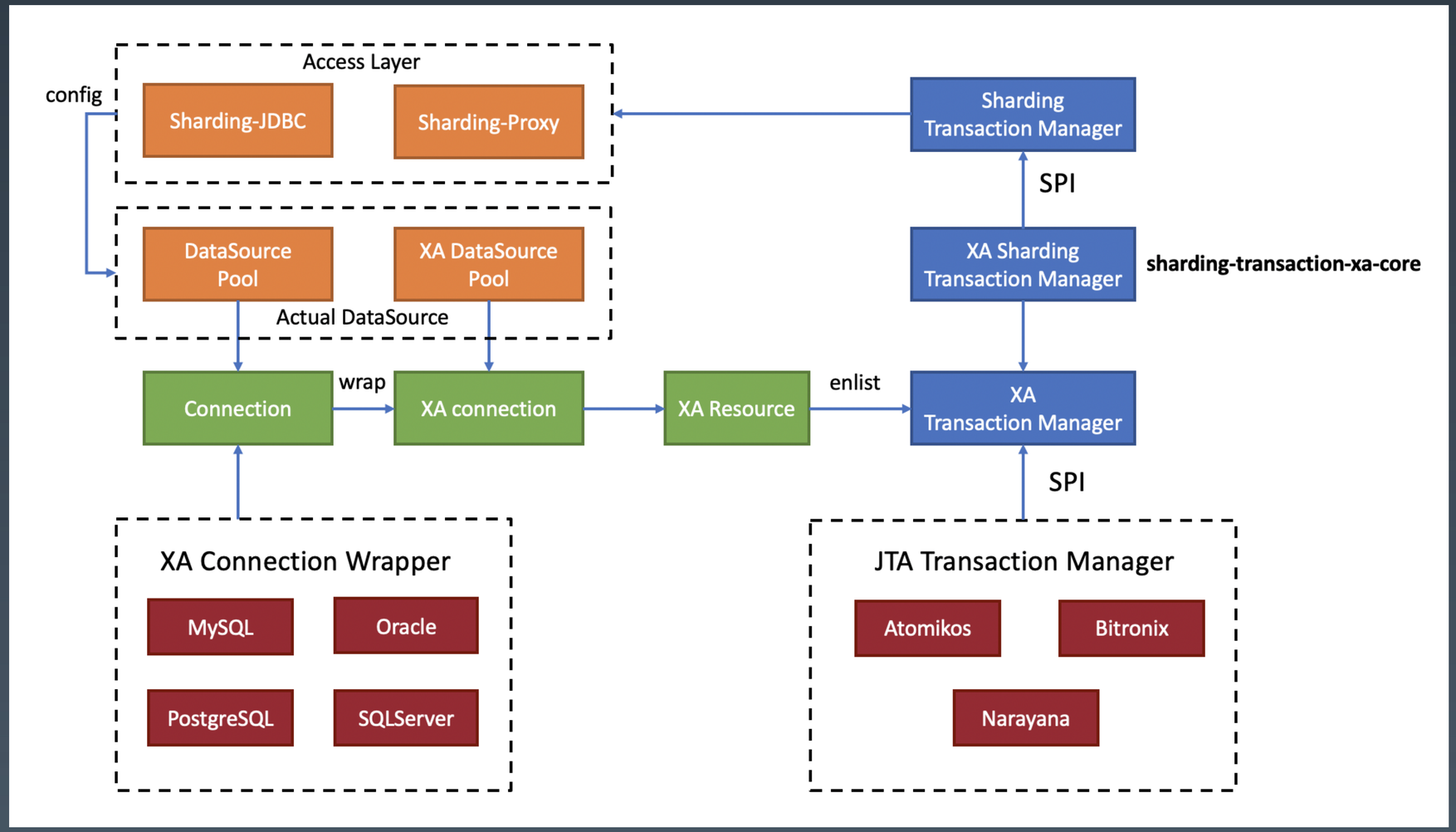
强一致的事务与柔性事务的 API 和功能并不完全相同，在它们之间并不能做到自由的透明切换。在开发决策阶段，就不得不在强一致的事务和柔性事务之间抉择，使得设计和开发成本被大幅增加。

基于 XA 的强一致事务使用相对简单，但是无法很好的应对互联网的高并发或复杂系统的长事务场景；柔性事务则需要开发者对应用进行改造，接入成本非常高，并且需要开发者自行实现资源锁定和反向补偿。

整合现有的成熟事务方案，为本地事务、两阶段事务和柔性事务提供统一的分布式事务接口，并弥补当前方案的不足，提供一站式的分布式事务解决方案是 Apache ShardingSphere 分布式事务模块的主要设计目标。

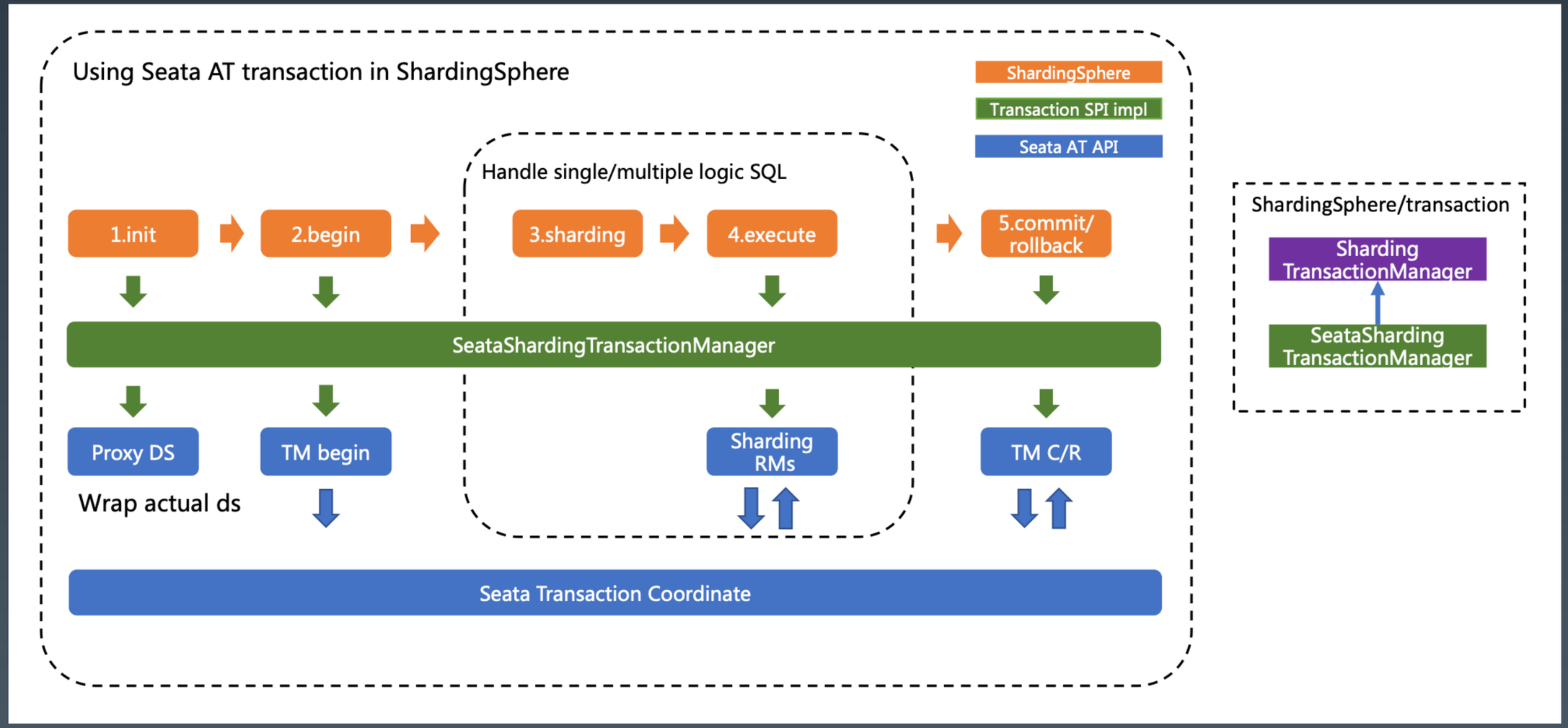
ShardingSphere 对分布式事务的支持

ShardingSphere 支持 XA 事务的常见几个开源实现



ShardingSphere 对分布式事务的支持

ShardingSphere 支持 Seata 的柔性事务。



ShardingSphere 对分布式事务的支持

ShardingSphere 的分布式事务模块。

极客大学

6.总结回顾与作业实践

第 16 课总结回顾

极客大学

分布式事务协议 XA

BASE 柔性事务

分布式事务 TCC/AT

ShardingSphere
对分布式事务的支持

第 16 课作业实践

1、（选做）列举常见的分布式事务，简单分析其使用场景和优缺点。

2、（必做）基于 hmily TCC 或 ShardingSphere 的 Atomikos XA 实现一个简单的分布式事务应用 demo（二选一），提交到 Github。

3、（选做）基于 ShardingSphere narayana XA 实现一个简单的分布式事务 demo。

4、（选做）基于 seata 框架实现 TCC 或 AT 模式的分布式事务 demo。

5、（选做☆）设计实现一个简单的 XA 分布式事务框架 demo，只需要能管理和调用2个 MySQL 的本地事务即可，不需要考虑全局事务的持久化和恢复、高可用等。

6、（选做☆）设计实现一个 TCC 分布式事务框架的简单 Demo，需要实现事务管理器，不需要实现全局事务的持久化和恢复、高可用等。

7、（选做☆）设计实现一个 AT 分布式事务框架的简单 Demo，仅需要支持根据主键 id 进行的单个删改操作的 SQL 或插入操作的事务。

THANKS!