分布式事务是一个业界难题，要提出一个满足所有业务场景要求的分布式事务方案是非常困难的。对于分布式事务中间件，应该重点关注以下三个方面：

* **编程模型**

1）对于服务提供者，需要不入侵业务；

2）对于服务使用者，使用分布式事务和使用本地事务一样；

* **可用性**

1）分布式中间件本身的高可用

2）极端情况下，如果分布式中间件出现了问题，不能马上恢复，导致部分事务不能正常结束，如何做降级处理？

* **性能**

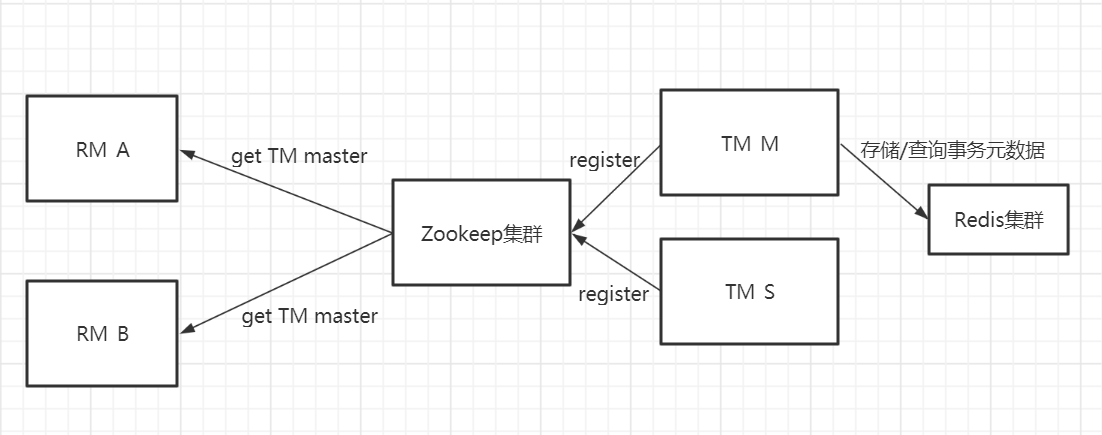
加入分布式事务的支持后，由于涉及跨系统的调用，性能下降是必然的，业务的RT也会升高，要保证整个系统的吞吐量，就要求更多的并发。因此需要减少影响分布式事务并发的因素。

我们设计的分布式事务解决方案，也正是朝这三个方向做了一些尝试，同时与目前业界最流行的开源分布式事务解决方案Seata做了对比分析。

1. **业务接入零配置**

业务接入我们的分布式事务框架，无需配置，跟使用Spring本地事务类似。借鉴Springboot约定大于配置的设计思想，我们根据业务场景集成了一些默认配置，业务导入jar包后，即插即用，当然业务也可以根据自己的场景进行定制。

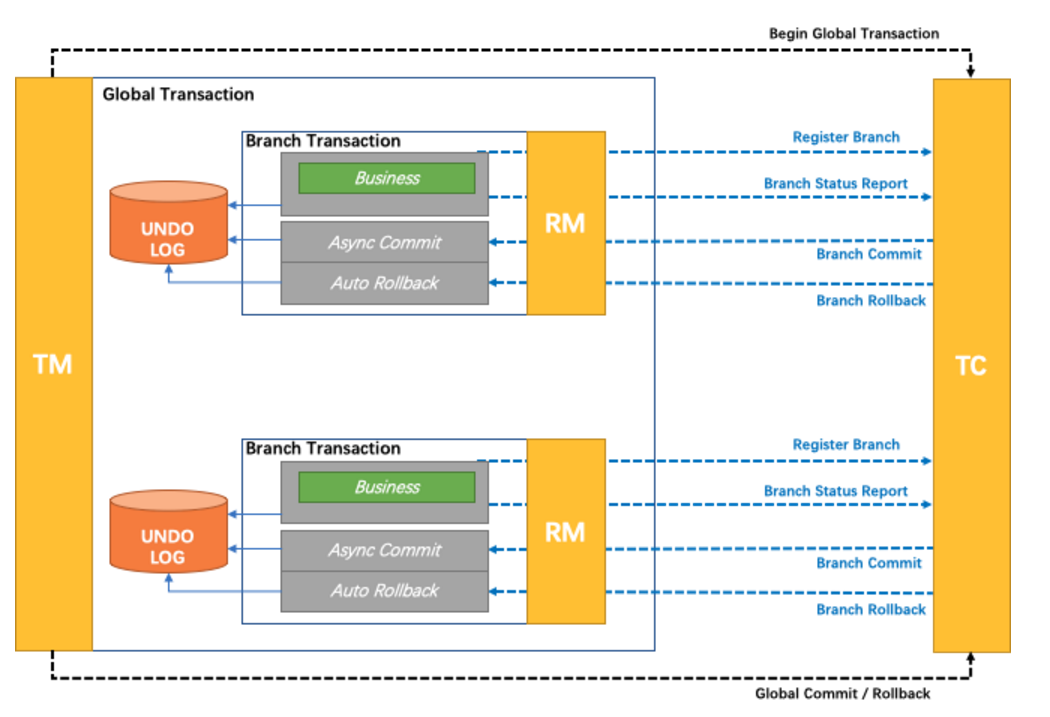
1. **分布式事务处理框架高可用**



1. 事务元数据（全局事务标识XID及其状态、分支事务标识brandId及其状态、全局事务与分支事务的映射关系等）存储和查询
2. TM集群由Zookeeper负责故障选举；
3. TM定时任务数据存储在Redis上？TM\_M故障后，无需同步定时数据，TM\_S升主后直接从Redis上读取定时任务（上次定时任务启动时间）

4）RMs只与主TM通信，减少TCP连接数量

1. **分布式事务处理流程亮点**
2. **全局一致性（与Seata AT模式相比）**

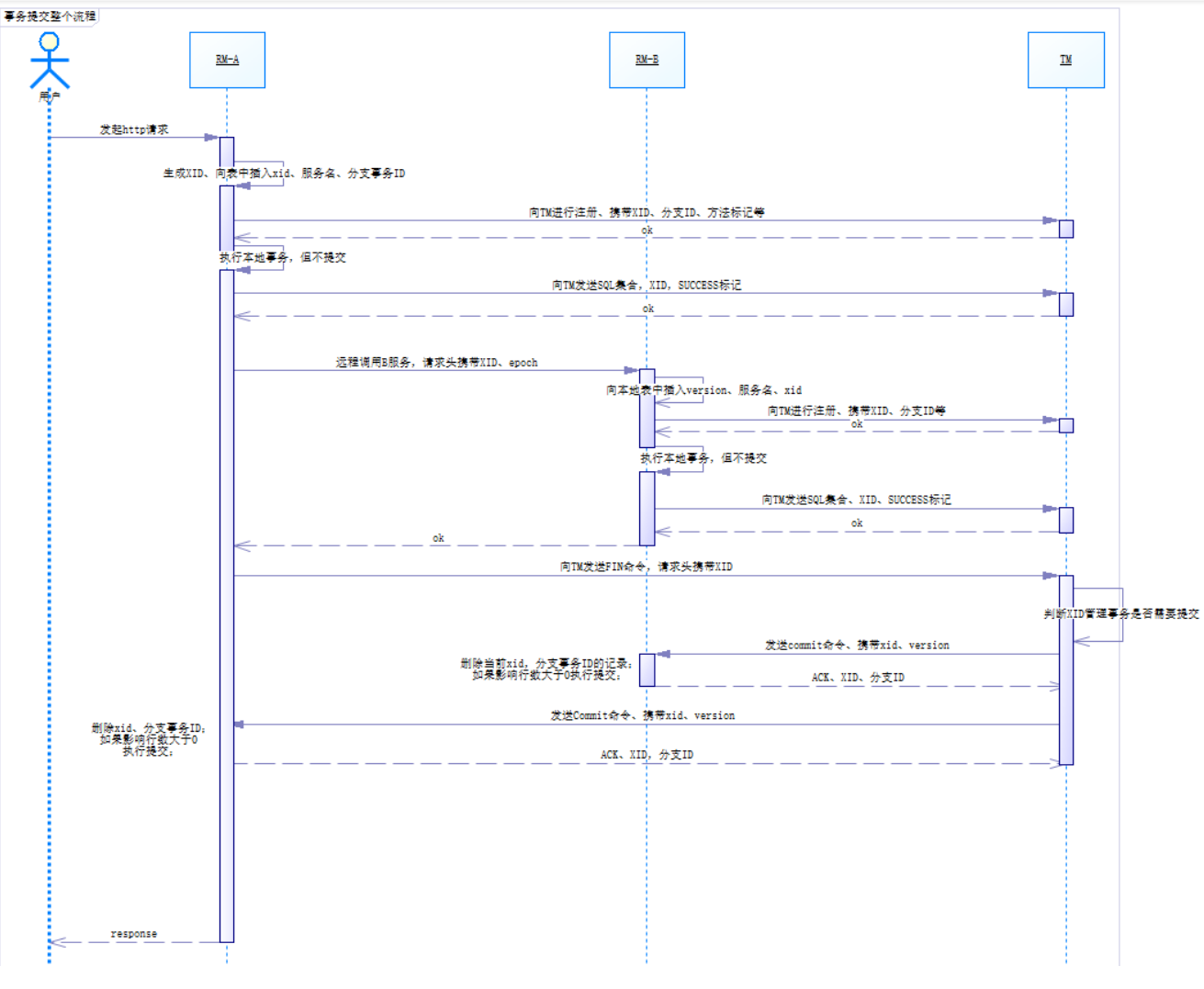


AT执行阶段：业务数据和回滚日志记录在同一个本地事务中提交；

AT完成阶段：

分支提交：异步删除回滚日志记录

分支回滚：依据回滚日志进行方向补偿更新



执行阶段：执行本地事务，但不提交

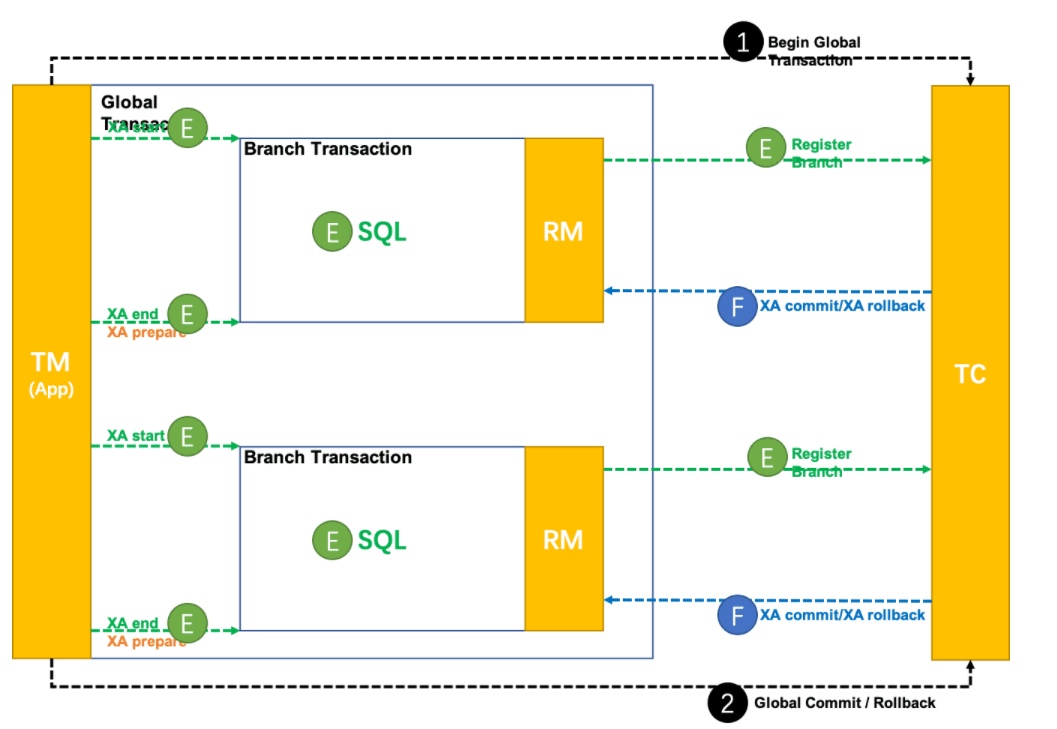
完成阶段：分支事务提交或者回滚

对比分析：

对于AT这种补偿型事务模式，在执行阶段就提交分支事务，比如将表m的字段a:100->50，此时其它业务来读取该字段a为50，后面完成阶段又回滚，表m的字段a:50->100，那之前其它业务读取的a=50就属于脏数据。

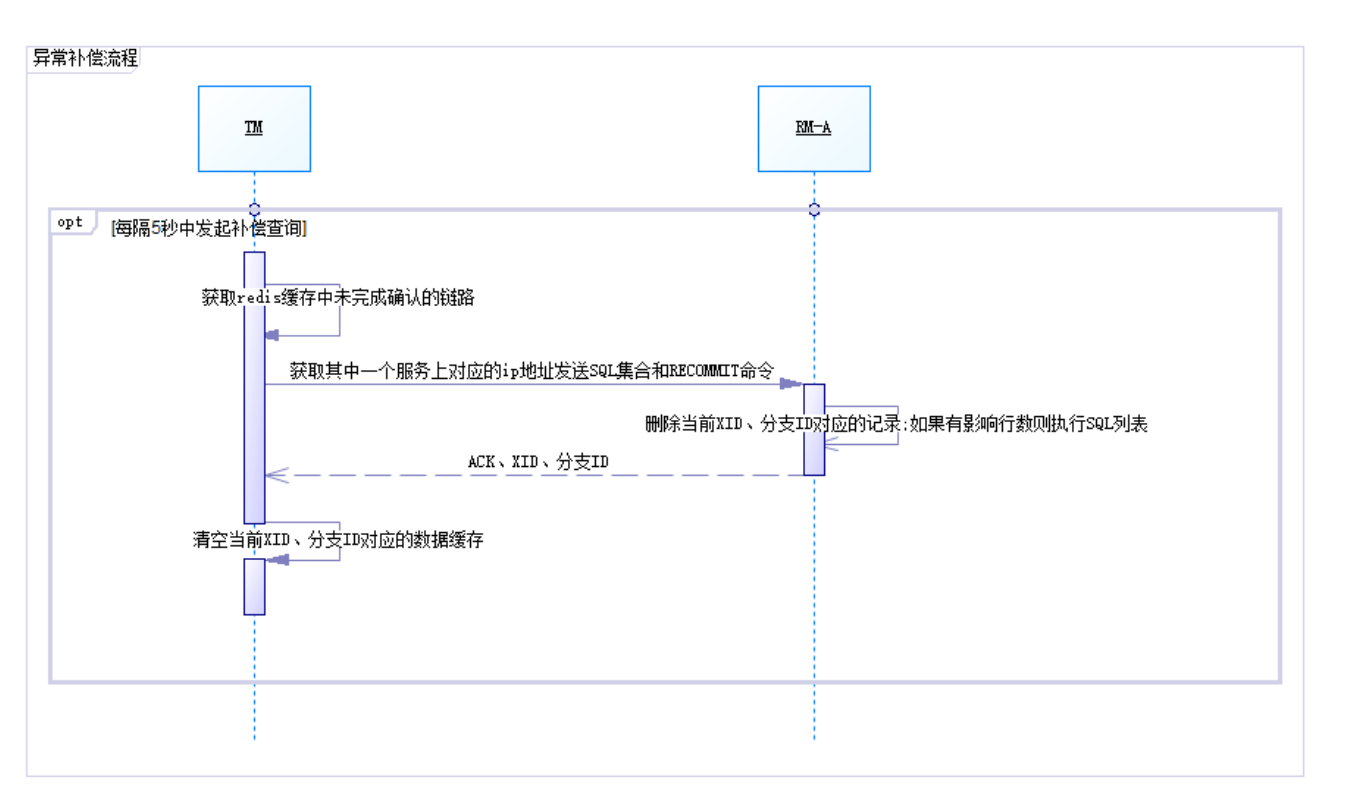
而我们的事务处理流程，由于在执行阶段不会提交本地事务，其它业务根据数据库的默认隔离级别（一般读已提交之上）就读取不到中间状态数据，而后面的完成阶段回滚，对其它业务也无感知，从而保证了全局一致性。

1. **并发性能好（与Seata XA模式对比）**



XA执行阶段：XA start/XA end/XA prepare + SQL +注册分支

XA完成阶段：XA commit/XA rollback



对比分析：

对于Seata XA模式，在执行阶段的XA prepare后，分支事务A就进入阻塞阶段，收到XA commit或XA rollback前都必须阻塞等待。如果此时分支事务A所在的RM\_A因为网络原因与TM失联了，那分支事务A锁定的数据会一直被锁定，甚至可能出现死锁。

对于这种情况，我们的异常补偿流程很好的解决了这个问题，解决思路：RM有一个兜底的定时器，RM发现与TM无法通信时，会先回滚执行阶段执行的事务；同时TM有一个异常补偿定时器，会根据全局事务的状态，决定对分支事务RM进行提交或者回滚；

**4、丰富的监控指标**

1）查询所有全局事务执行信息；

2）查询全局事务详情；