主讲老师：Fox

> 有道笔记链接：https://note.youdao.com/s/XOut4XHc

课前须知：

Seata源码分析会讲两节课：

1. 从全局事务角度分析Seata设计（侧重点在全局事务的设计）
2. 从两阶段提交，自动补偿机制，隔离性的角度分析Seata设计（侧重点在分支事务的设计）

1. Seata整体架构

1.1 Seata的三大角色

在 Seata 的架构中，一共有三个角色：

- **TC (Transaction Coordinator) - 事务协调者**

维护全局和分支事务的状态，驱动全局事务提交或回滚。

- **TM (Transaction Manager) - 事务管理器**

定义全局事务的范围：开始全局事务、提交或回滚全局事务。

- **RM (Resource Manager) - 资源管理器**

管理分支事务处理的资源，与TC交谈以注册分支事务和报告分支事务的状态，并驱动分支事务提交或回滚。

其中，TC 为单独部署的 Server 服务端，TM 和 RM 为嵌入到应用中的 Client 客户端。

1.2 Seata的生命周期

在 Seata 中，一个分布式事务的生命周期如下：

1. TM 请求 TC 开启一个全局事务。TC 会生成一个 XID 作为该全局事务的编号。XID会在微服务的调用链路中传播，保证将多个微服务的子事务关联在一起。
2. RM 请求 TC 将本地事务注册为全局事务的分支事务，通过全局事务的 XID 进行关联。
3. TM 请求 TC 告诉 XID 对应的全局事务是进行提交还是回滚。
4. TC 驱动 RM 们将 XID 对应的自己的本地事务进行提交还是回滚。

1.3 AT模式设计思路

Seata AT模式的核心是对业务无侵入，是一种改进后的两阶段提交，其设计思路如下:

- 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。

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。核心在于对业务sql进行解析，转换成undolog，并同时入库，这是怎么做的呢?

二阶段

- 分布式事务操作成功，则TC通知RM异步删除undolog
- 分布式事务操作失败，TM向TC发送回滚请求，RM 收到协调器TC发来的回滚请求，通过 XID 和 Branch ID 找到相应的回滚日志记录，通过回滚记录生成反向的更新 SQL 并执行，以完成分支的回滚。

2. Seata核心接口和实现类

TransactionManager

DefaultTransactionManager

TransactionManagerHolder为创建单例TransactionManager的工厂，可以使用EnhancedServiceLoader的spi机制加载用户自定义的类，默认为DefaultTransactionManager。

GlobalTransaction

GlobalTransaction接口提供给用户开启事务，提交，回滚，获取状态等方法。

DefaultGlobalTransaction

DefaultGlobalTransaction是GlobalTransaction接口的默认实现，它持有TransactionManager对象，默认开启事务超时时间为60秒，默认名称为default，因为调用者的业务方法可能多重嵌套创建多

个GlobalTransaction对象开启事务方法，因此GlobalTransaction有GlobalTransactionRole角色属性，只有Launcher角色的才有开启、提交、回滚事务的权利。

GlobalTransactionContext

GlobalTransactionContext为操作GlobalTransaction的工具类，提供创建新的GlobalTransaction，获取当前线程有的GlobalTransaction等方法。

GlobalTransactionScanner

GlobalTransactionScanner继承AbstractAutoProxyCreator类，即实现了SmartInstantiationAwareBeanPostProcessor接口，会在spring容器启动初始化bean的时候，对bean进行代理操作。wrapIfNecessary为继承父类代理bean的核心方法，如果用户配置了service.disableGlobalTransaction为false属性则注解不生效直接返回，否则对GlobalTransactional或GlobalLock的方法进行拦截代理。

GlobalTransactionalInterceptor

GlobalTransactionalInterceptor实现aop的MethodInterceptor接口，对有@GlobalTransactional或GlobalLock注解的方法进行代理。

TransactionalTemplate

TransactionalTemplate模板类提供了一个开启事务，执行业务，成功提交和失败回滚的模板方法execute(TransactionalExecutor business)。

DefaultCoordinator

DefaultCoordinator即为TC，全局事务默认的事务协调器。它继承AbstractTCInboundHandler接口，为TC接收RM和TM的request请求数据，是进行相应处理的处理器。实现TransactionMessageHandler接口，去处理收到的RPC信息。实现ResourceManagerInbound接口，发送至RM的branchCommit，branchRollback请求。

Core

Core接口为seata处理全球事务协调器TC的核心处理器，它继承ResourceManagerOutbound接口，接受来自RM的rpc网络请求（branchRegister，branchReport，lockQuery）。同时继承

TransactionManager接口，接受来自TM的rpc网络请求（begin，commit,rollback,getStatus），另外提供提供3个接口方法。

ATCore

GlobalSession

GlobalSession是seata协调器DefaultCoordinator管理维护的重要部件，当用户开启全局分布式事务，TM调用begin方法请求至TC，TC则创建GlobalSession实例对象，返回唯一的xid。它实现SessionLifecycle接口，提供begin，changeStatus，changeBranchStatus，addBranch，removeBranch等操作session和branchSession的方法。

BranchSession

BranchSession为分支session，管理分支数据，受globalSession统一调度管理，它的lock和unlock方法由lockManger实现。

LockManager

DefaultLockManager是LockManager的默认实现，它获取branchSession的lockKey，转换成List<RowLock>，委派Locker进行处理。

Locker

Locker接口提供根据行数据获取锁，释放锁，是否锁住和清除所有锁的方法。

ResourceManager

ResourceManager是seata的重要组件之一，RM负责管理分支数据资源的事务。

AbstractResourceManager实现ResourceManager提供模板方法。DefaultResourceManager适配所有的ResourceManager，所有方法调用都委派给对应负责的ResourceManager处理。

DataSourceManager

此为AT模式核心管理器，DataSourceManager继承AbstractResourceManager，管理数据库Resouce的注册，提交以及回滚等

AsyncWorker DataSourceManager事务提交委派给AsyncWorker进行提交的，因为都成功了，无需回滚成功的数据，只需要删除生成的操作日志就行，采用异步方式，提高效率。

```
AsyncWorker#doBranchCommits
> UndoLogManagerFactory.getUndoLogManager(dataSourceProxy.getDbType())
    .batchDeleteUndoLog(xids, branchIds, conn)
```

UndoLogManager

Resource

Resource能被ResourceManager管理并且能够关联GlobalTransaction。

DataSourceProxy

DataSourceProxy实现Resource接口，BranchType为AT自动模式。它继承AbstractDataSourceProxy代理类，所有的DataSource相关的方法调用传入的targetDataSource代理类的方法，除了创建connection方法为创建ConnectionProxy代理类。对象初始化时获取连接的jdbcUrl作为resourceId,并注册至DefaultResourceManager进行管理。同时还提供获取原始连接不被代理的getPlainConnection方法。

ConnectionProxy

```
private void doCommit() throws SQLException {
    if (context.inGlobalTransaction()) {
        processGlobalTransactionCommit();
    } else if (context.isGlobalLockRequire()) {
        processLocalCommitWithGlobalLocks();
    } else {
        targetConnection.commit();
    }
}
```

```
private void processGlobalTransactionCommit() throws SQLException {
    try {
        register();
    } catch (TransactionException e) {
        recognizeLockKeyConflictException(e, context.buildLockKeys());
    }
    try {
        UndoLogManagerFactory.getUndoLogManager(this.getDbType()).flushUndoLogs(this);
        targetConnection.commit();
    } catch (Throwable ex) {
        LOGGER.error("process connectionProxy commit error: {}", ex.getMessage(), ex);
        report(false);
        throw new SQLException(ex);
    }
    if (IS_REPORT_SUCCESS_ENABLE) {
        report(true);
    }
    context.reset();
}
```

ExecuteTemplate

ExecuteTemplate为具体statement的execute，executeQuery和executeUpdate执行提供模板方法

Executor

SQLRecognizer

SQLRecognizer识别sql类型，获取表名，表别名以及原生sql

UndoExecutorFactory

UndoExecutorFactory根据sqlType生成对应的AbstractUndoExecutor。

UndoExecutor为生成执行undoSql的核心。如果全局事务回滚，它会根据beforeImage和afterImage以及sql类型生成对应的反向sql执行回滚数据，并添加脏数据校验机制，使回滚数据更加可靠。

3. Seata AT模式源码分析

Seata设计流程: https://www.processon.com/view/link/6311bfda1e0853187c0ecd8c

https://www.processon.com/view/link/6007f5c00791294a0e9b611a