

主讲老师：Fox

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 文档：16 分布式事务Seata源码分析.note

2 链接：[http://note.youdao.com/noteshare?](http://note.youdao.com/noteshare?id=27c1776bec0985856b758475deda59a2&sub=36072485CD0343DB9F82FB0484546FEF)

[id=27c1776bec0985856b758475deda59a2&sub=36072485CD0343DB9F82FB0484546FEF](http://note.youdao.com/noteshare?id=27c1776bec0985856b758475deda59a2&sub=36072485CD0343DB9F82FB0484546FEF)

1. Seata核心接口和实现类

TransactionManager

DefaultTransactionManager

GlobalTransaction

DefaultGlobalTransaction

GlobalTransactionScanner

GlobalTransactionalInterceptor

TransactionalTemplate

DefaultCoordinator

Core

GlobalSession

BranchSession

LockManager

Locker

ResourceManager

DataSourceManager

UndoLogManager

Resource

DataSourceProxy

ConnectionProxy

ExecuteTemplate

Executor

SQLRecognizer

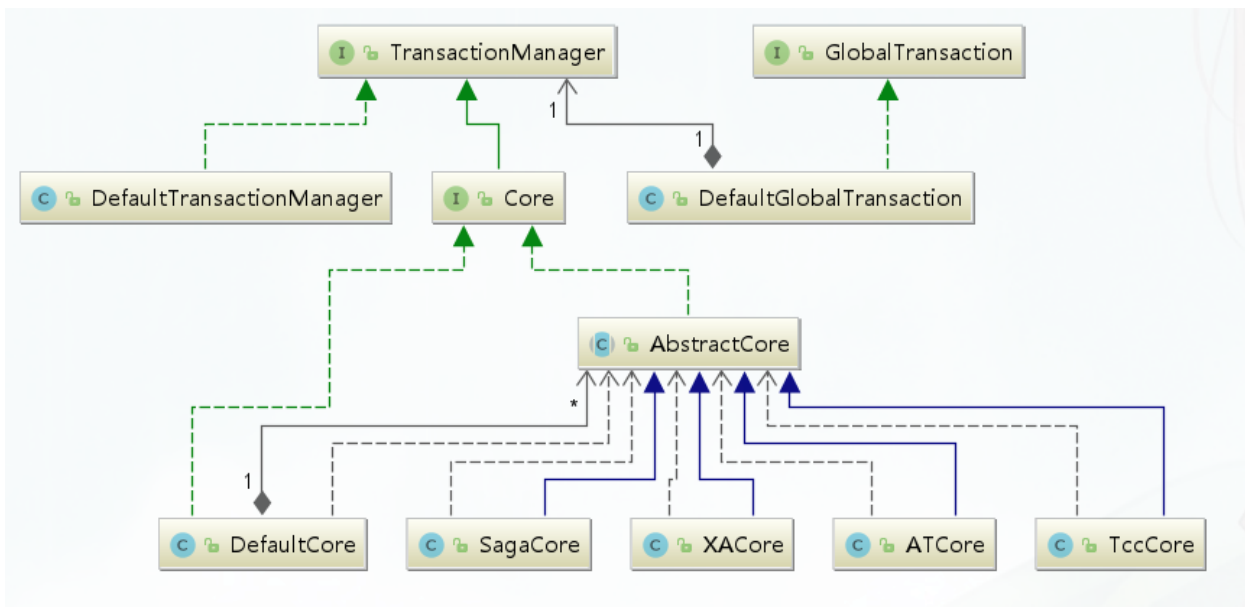
UndoExecutorFactory

2. 源码分析

1. Seata核心接口和实现类

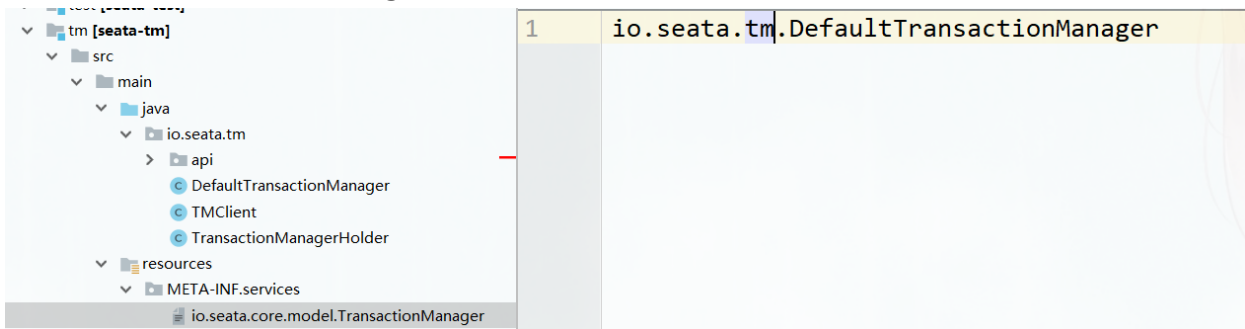
TransactionManager

I	TransactionManager	
m	begin(String, String, String, int)	String
m	commit(String)	GlobalStatus
m	rollback(String)	GlobalStatus
m	getStatus(String)	GlobalStatus
m	globalReport(String, GlobalStatus)	GlobalStatus



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。

GlobalTransaction		
(m)	begin()	void
(m)	begin(int)	void
(m)	begin(int, String)	void
(m)	commit()	void
(m)	rollback()	void
(m)	suspend()	SuspendedResourcesHolder
(m)	resume(SuspendedResourcesHolder)	void
(m)	getStatus()	GlobalStatus
(m)	getXid()	String
(m)	globalReport(GlobalStatus)	void
(m)	getLocalStatus()	GlobalStatus

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@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(TransactionExecutor business)。

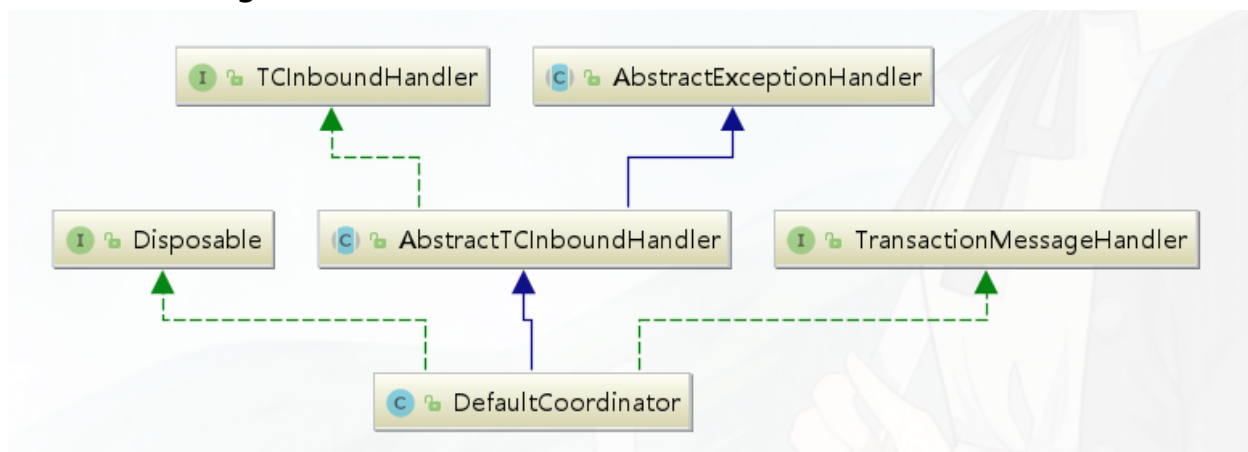
```
try {
    // 2. If the tx role is 'GlobalTransactionRole.Launcher', send transaction to TC,
    //     else do nothing. Of course, the hooks will still be triggered.
    beginTransaction(txInfo, tx);

    Object rs;
    try {
        // Do Your Business
        rs = business.execute();
    } catch (Throwable ex) {
        // 3. The needed business exception to rollback.
        completeTransactionAfterThrowing(txInfo, tx, ex);
        throw ex;
    }

    // 4. everything is fine, commit.
    commitTransaction(tx);
}
```

DefaultCoordinator

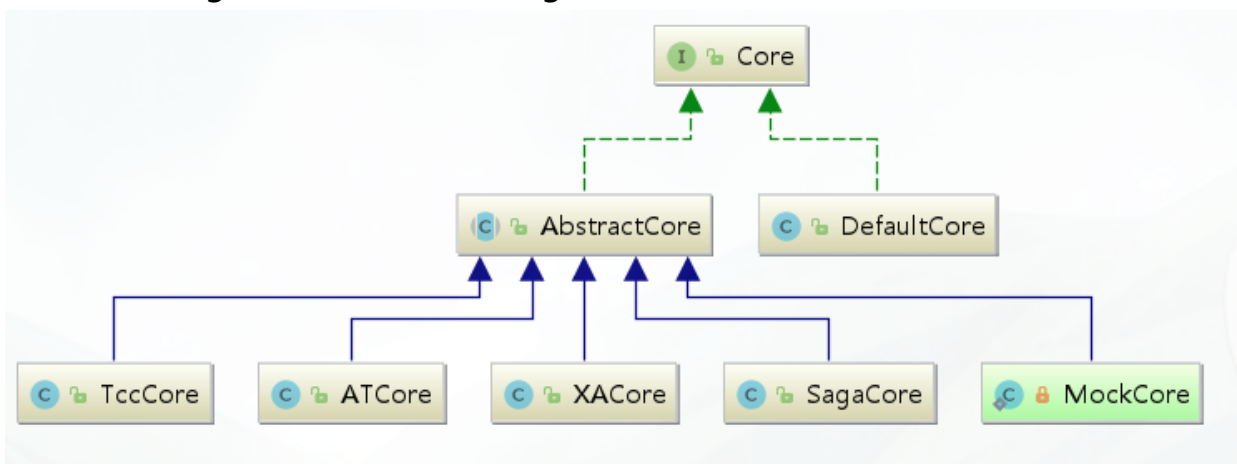
DefaultCoordinator即为TC，全局事务默认的事务协调器。它继承AbstractTCInboundHandler接口，为TC接收RM和TM的request请求数据，并进行相应处理的处理器。实现TransactionMessageHandler接口，去处理收到的RPC信息。实现ResourceManagerInbound接口，发送至RM的branchCommit，branchRollback请求。

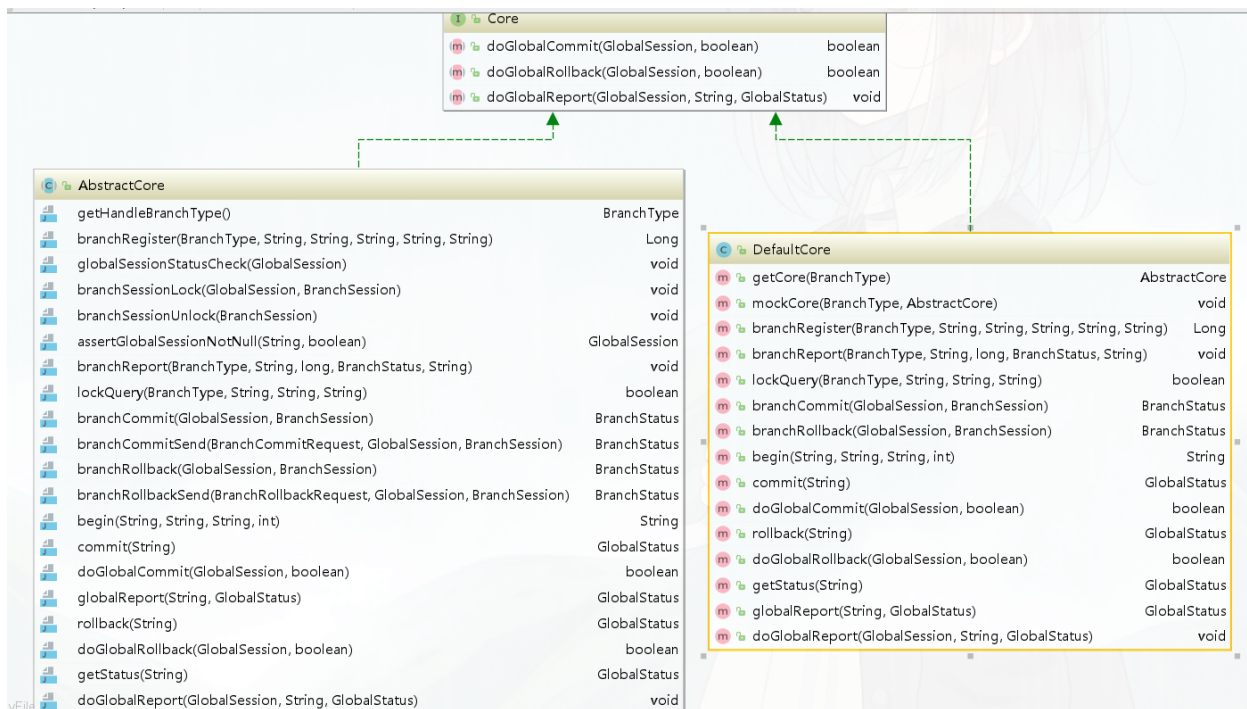


DefaultCoordinator		
m	doGlobalBegin(GlobalBeginRequest, GlobalBeginResponse, RpcContext)	void
m	doGlobalCommit(GlobalCommitRequest, GlobalCommitResponse, RpcContext)	void
m	doGlobalRollback(GlobalRollbackRequest, GlobalRollbackResponse, RpcContext)	void
m	doGlobalStatus(GlobalStatusRequest, GlobalStatusResponse, RpcContext)	void
m	doGlobalReport(GlobalReportRequest, GlobalReportResponse, RpcContext)	void
m	doBranchRegister(BranchRegisterRequest, BranchRegisterResponse, RpcContext)	void
m	doBranchReport(BranchReportRequest, BranchReportResponse, RpcContext)	void
m	doLockCheck(GlobalLockQueryRequest, GlobalLockQueryResponse, RpcContext)	void
m	timeoutCheck()	void
m	handleRetryRollbacking()	void
m	handleRetryCommitting()	void
m	isRetryTimeout(long, long, long)	boolean
m	handleAsyncCommitting()	void
m	undoLogDelete()	void
m	init()	void
m	onRequest(AbstractMessage, RpcContext)	AbstractResultMessage
m	onResponse(AbstractResultMessage, RpcContext)	void
m	destroy()	void

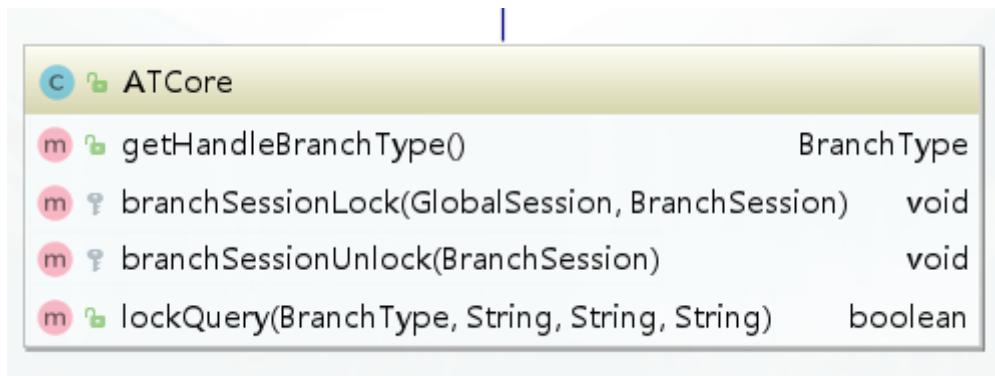
Core

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

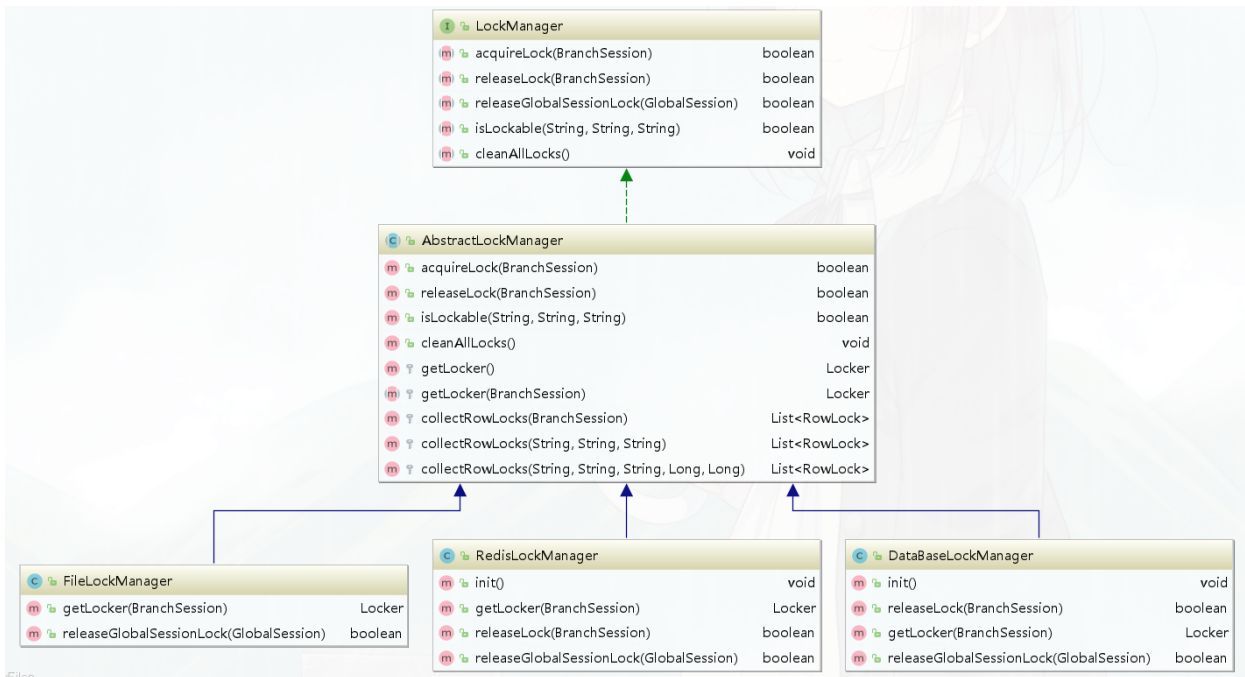
GlobalSession是seata协调器DefaultCoordinator管理维护的重要部件，当用户开启全局分布式事务，TM调用begin方法请求至TC，TC则创建GlobalSession实例对象，返回唯一的xid。它实现SessionLifecycle接口，提供begin，changeStatus，changeBranchStatus，addBranch，removeBranch等操作session和branchSession的方法。

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。

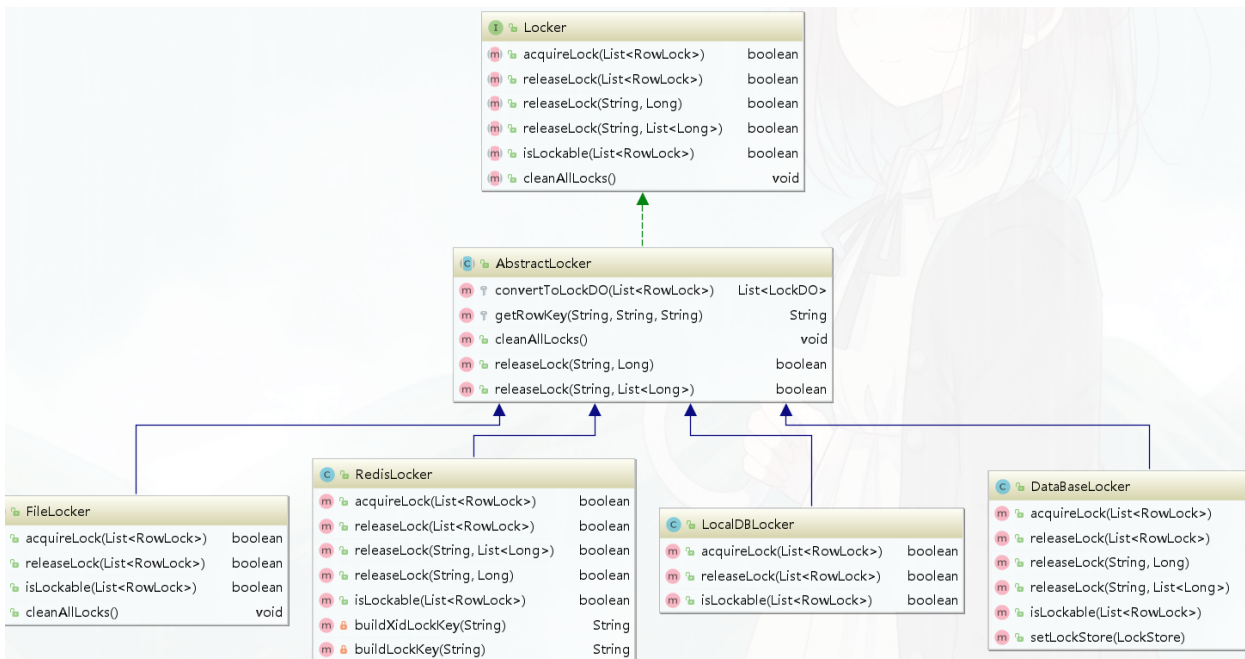
LockManager

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。



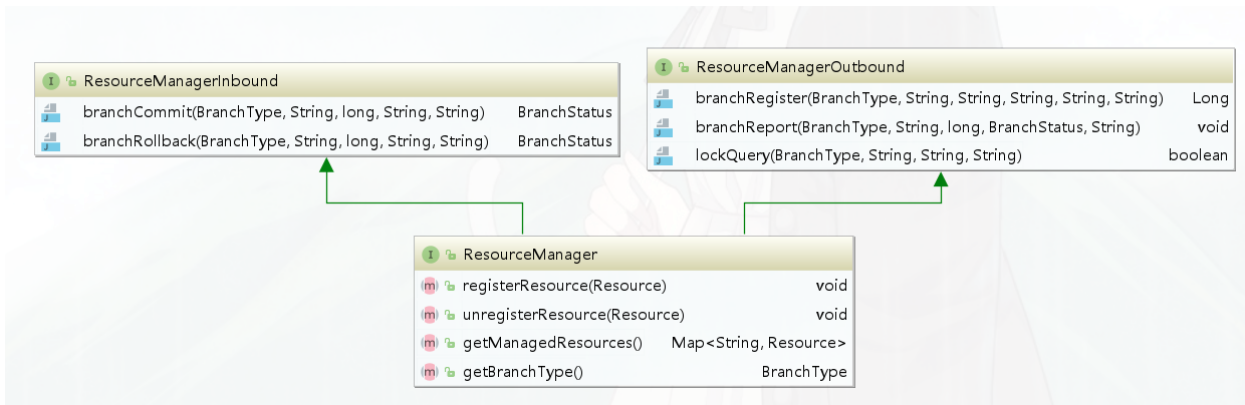
Locker

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。



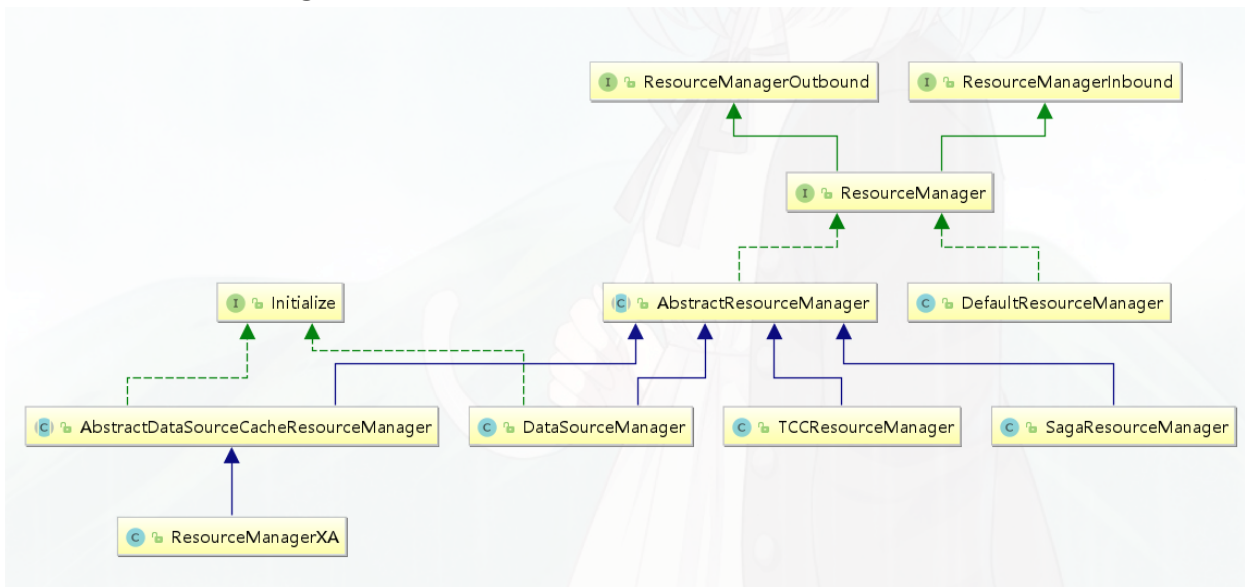
ResourceManager

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Resource的注册，提交以及回滚等

c	DataSourceManager	
m	setAsyncWorker(ResourceManagerInbound)	void
m	lockQuery(BranchType, String, String, String)	boolean
m	initAsyncWorker(ResourceManagerInbound)	void
m	init()	void
m	registerResource(Resource)	void
m	unregisterResource(Resource)	void
m	get(String)	DataSourceProxy
m	branchCommit(BranchType, String, long, String, String)	BranchStatus
m	branchRollback(BranchType, String, long, String, String)	BranchStatus
m	getManagedResources()	Map<String, Resource>
m	getBranchType()	BranchType

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。

```

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

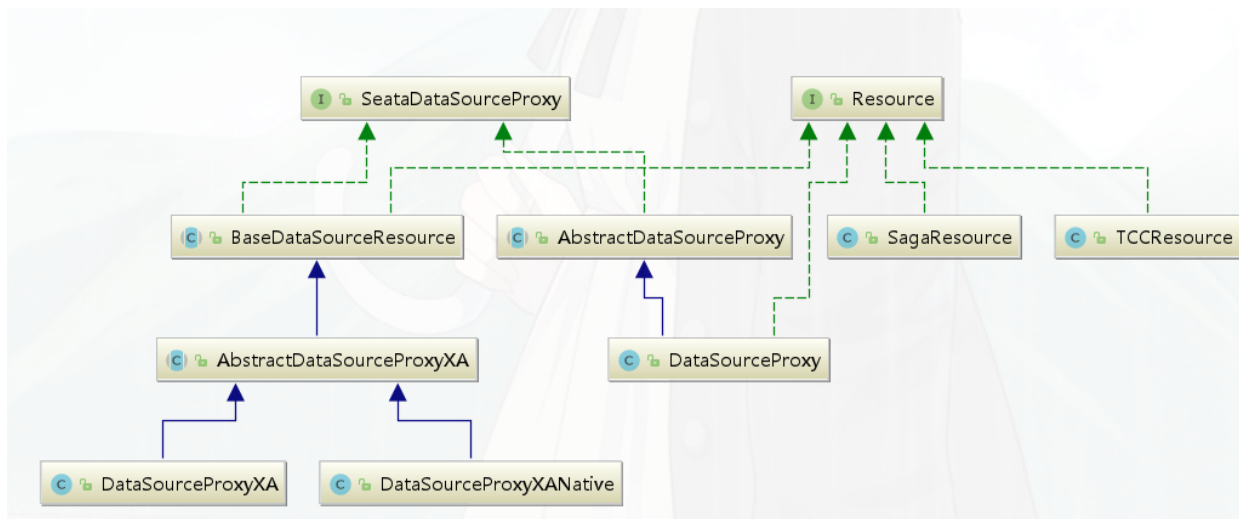
```

UndoLogManager

I	UndoLogManager	
m	flushUndoLogs(ConnectionProxy)	void
m	undo(DataSourceProxy, String, long)	void
m	deleteUndoLog(String, long, Connection)	void
m	batchDeleteUndoLog(Set<String>, Set<Long>, Connection)	void
m	deleteUndoLogByLogCreated(Date, int, Connection)	int

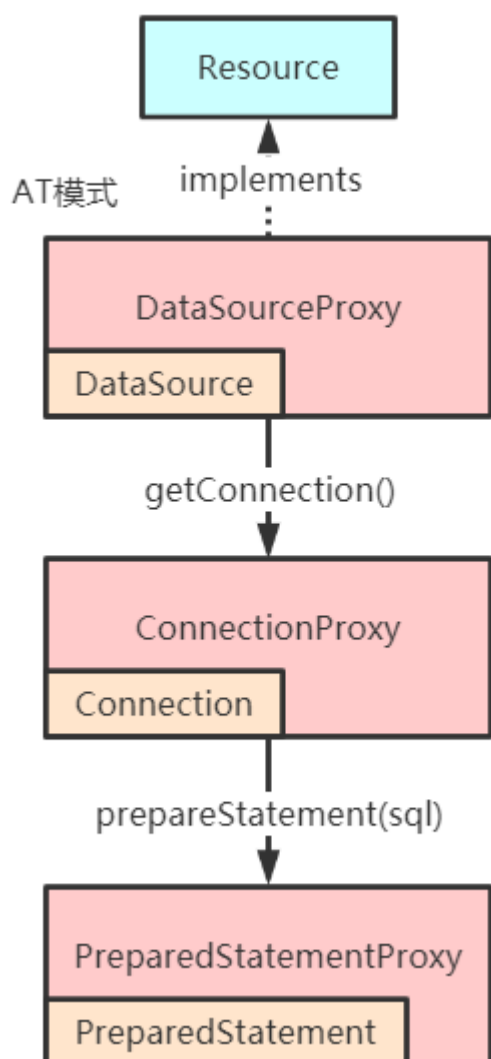
Resource

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。



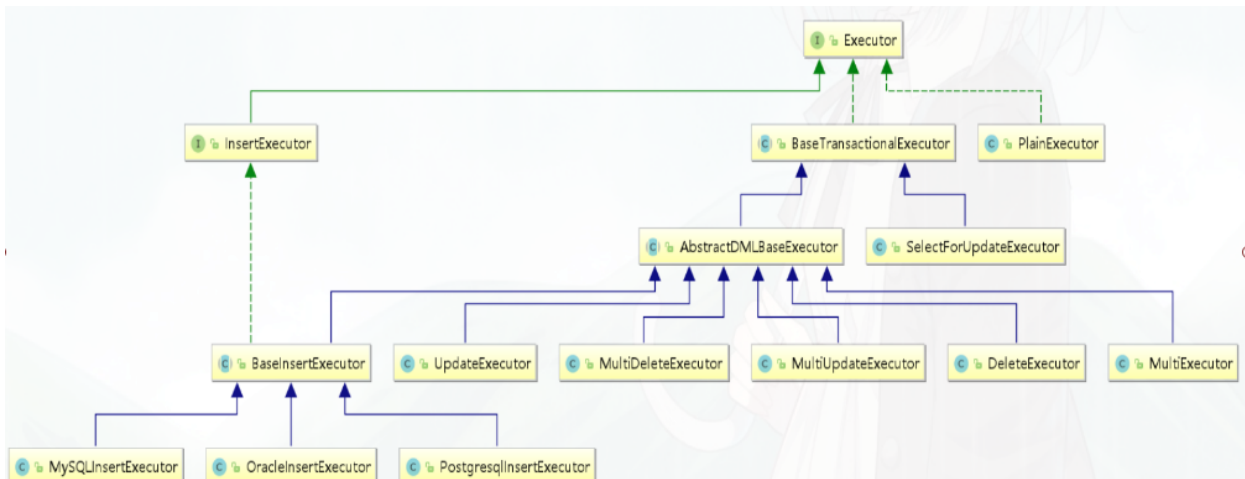
ConnectionProxy

```
1 private void doCommit() throws SQLException {
2     if (context.inGlobalTransaction()) {
3         processGlobalTransactionCommit();
4     } else if (context.isGlobalLockRequire()) {
5         processLocalCommitWithGlobalLocks();
6     } else {
7         targetConnection.commit();
8     }
9 }
10 private void processGlobalTransactionCommit() throws SQLException {
11     try {
12         register();
13     } catch (TransactionException e) {
14         recognizeLockKeyConflictException(e, context.buildLockKeys());
15     }
16     try {
17         UndoLogManagerFactory.getUndoLogManager(this.getDbType()).flushUndoLogs(this);
18         targetConnection.commit();
19     } catch (Throwable ex) {
20         LOGGER.error("process connectionProxy commit error: {}",
21             ex.getMessage(), ex);
22         report(false);
23         throw new SQLException(ex);
24     }
25     if (IS_REPORT_SUCCESS_ENABLE) {
26         report(true);
27     }
28     context.reset();
29 }
```

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

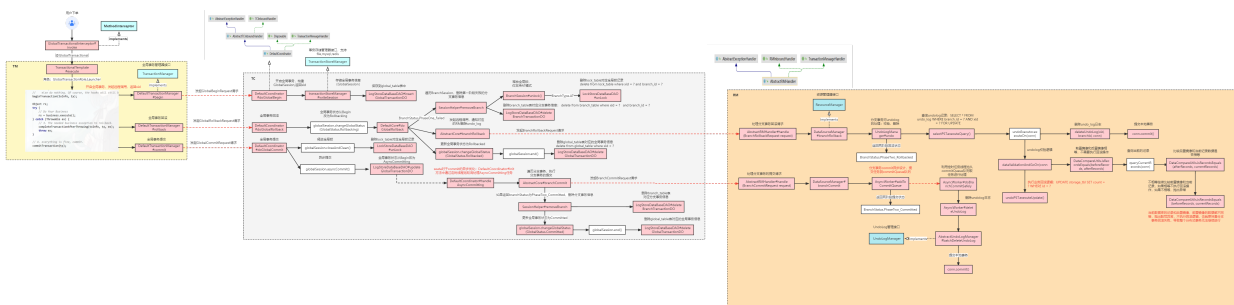
UndoExecutorFactory

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。

2. 源码分析

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



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

<https://www.processon.com/view/link/5f743063e0b34d0711f001d2>