# 本地事务

## 事务的基本性质：ACID

A(Atomicity：原子性)：原子性是指事务包含的所有操作要么全部成功，要么全部失败回滚

C(Consistency：一致性)：事务必须使数据库从一个一致性状态变换到另一个一致性状态，也就是说一个事务执行之前和执行之后都必须处于一致性状态。举例来说，假设用户A和用户B两者的钱加起来一共是1000，那么不管A和B之间如何转账、转几次账，事务结束后两个用户的钱相加起来应该还得是1000，这就是事务的一致性

I(Isolation：隔离性)：当多个用户并发访问数据库时，比如同时操作同一张表时，数据库为每一个用户开启的事务，不能被其他事务的操作所干扰，多个并发事务之间要相互隔离

D(Durability：持久性)：指一个事务一旦被提交了，那么对数据库中的数据的改变就是永久性的，即便是在数据库系统遇到故障的情况下也不会丢失提交事务的操作

## 事务的隔离级别

1. Read Uncommited(读未提交)

该隔离级别的事务会读到其它未提交事务的数据，此现象也称为脏读(产生脏读)

脏读：A事务读取到了B事务未提交的数据

1. Read Commited(读已提交)

该隔离级别的事务会读到其它已提交事务的数据，可以解决脏读，多次读取可能造成结果不一致，此现象称为不可重复读()。Oracle和sql server的默认隔离级别

不可重复读：对于修改来说：A事务第一次读取到B事务修改后的结果，但是C事务也修改了同一条数据，A事务再次读取就读取到了C事务的数据，跟刚刚读取到B事务的数据结果不一致。对于新增来说：A事务第一次读取到5条数据，但是C事务同时也新增了一条数据，A事务再次读取就读取到了6条数据，跟刚刚读取到的5条数据结果不一致。两次读取到的结果不一致称为不可重复读

1. Repeatable read(可重复读)

可重复读是指在一个事务内，多次读同一条数据或者查询数量。在这个事务还没有结束时，另外一个事务操作(增删改查)同一数据并且提交了事务，这样就在一个事务内两次或多次读到的数据是一样的，因此称为是可重复读。该隔离级别是mysql默认的隔离级别，但是会产生幻读，mysql的innodb引擎可以通过next-key locks机制来避免幻读

幻读：存在两个事务A和B，当A事务通过指定条件(主键id=1)查询数据库时，数据不存在，但是在事务A查询结束后，事务B向表里面插入了一条Id=1的数据，此时A事务通过id=1的条件查询数据库是查不到这条数据的，但是A事务也想插入一条id=1的数据，此时插入失败，会报主键冲突。对于事务A的业务来时是执行失败的，这里事务A就产生了幻读，因为事务A读取得数据状态并不能支持他下一步的业务

1. Serializable(序列化)

提供严格的事务隔离。它要求事务序列化执行，事务只能一个接着一个地执行，但不能并发执行，事务都是串行顺序执行的，mysql数据库的innoDB引擎会给读操作隐式加一把共享锁，从而避免了脏读、不可重复读和幻读

## 事务的传播行为

1. PROPAGATION\_REQUIRED

表示当前方法必须运行在事务中，如果事务存在，则在该事务中运行，如果不存在，则启动一个新的事务。默认的事务传播行为，也是最常用的

1. PROPAGATIO\_SUPPORTS

表示当前方法不需要事务上下文，如果事务存在，则在该事务中运行，如果不存在，不在事务中运行

1. PROPAGATION\_MANDATORY

表示当前方法必须在事务中运行，如果事务存在，则在该事务中运行，如果事务不存在，则会抛异常

1. PROPAGATION\_REQUIRED\_NEW

表示当前方法必须运行在自己的事务中，一个新的事务将被启动，如果存在当前事务，在该方法执行期间，当前事务会被挂起

1. PROPAGATION\_NOT\_SUPPORTED

表示当前方法不应该运行在事务中，如果存在当前事务，在该方法执行期间，当前事务会被挂起

1. PROPAGATION\_NEVER

表示当前方法不应该运行在事务上下文中，如果当前正有一个事务在运行，则会抛出异常

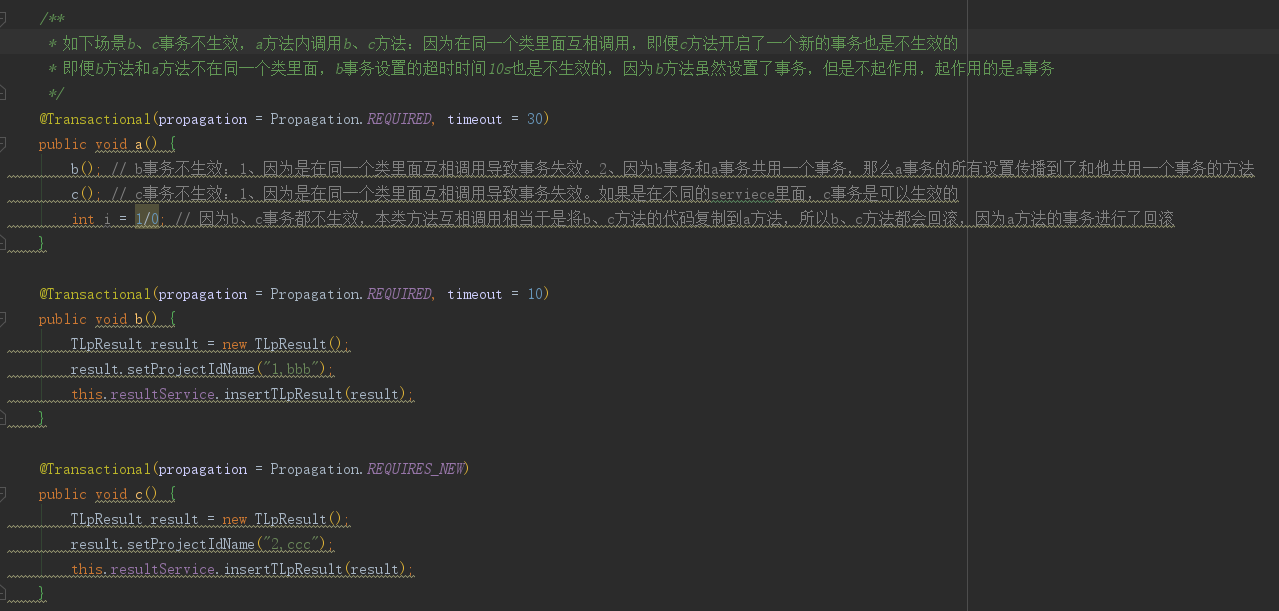
1. PROPAGATION\_NESTED

如果当前已经存在一个事务，则该方法会在嵌套事务中运行，嵌套事务可以独立于当前事务进行单独的提交或回滚，如果当前事务不存在，则与PROPAGATION\_REQUIRED的传播行为一样

## springBoot事务关键点

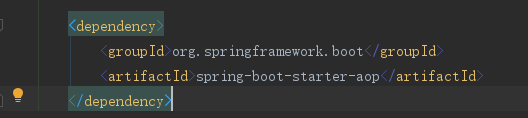
### 1.4.1 问题描述

在同一个类里面，编写3个方法，a、b、c，内部调用的时候，a方法内部调用b、c方法，会导致事务设置失效，原因是因为没有用到代理对象，相当于是把b、c方法的代码复制到a方法



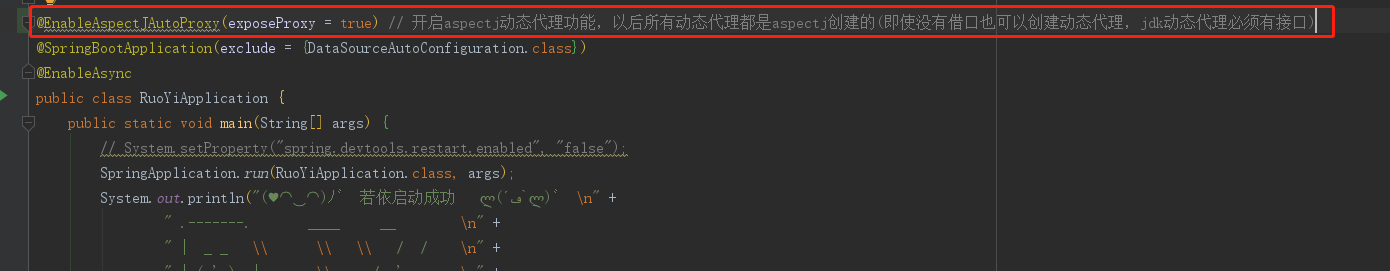
### 1.4.2 springboot解决本类方法互相调用事务失效问题

1、导入依赖：spring-boot-starter-aop里面引入了aspectjweaver

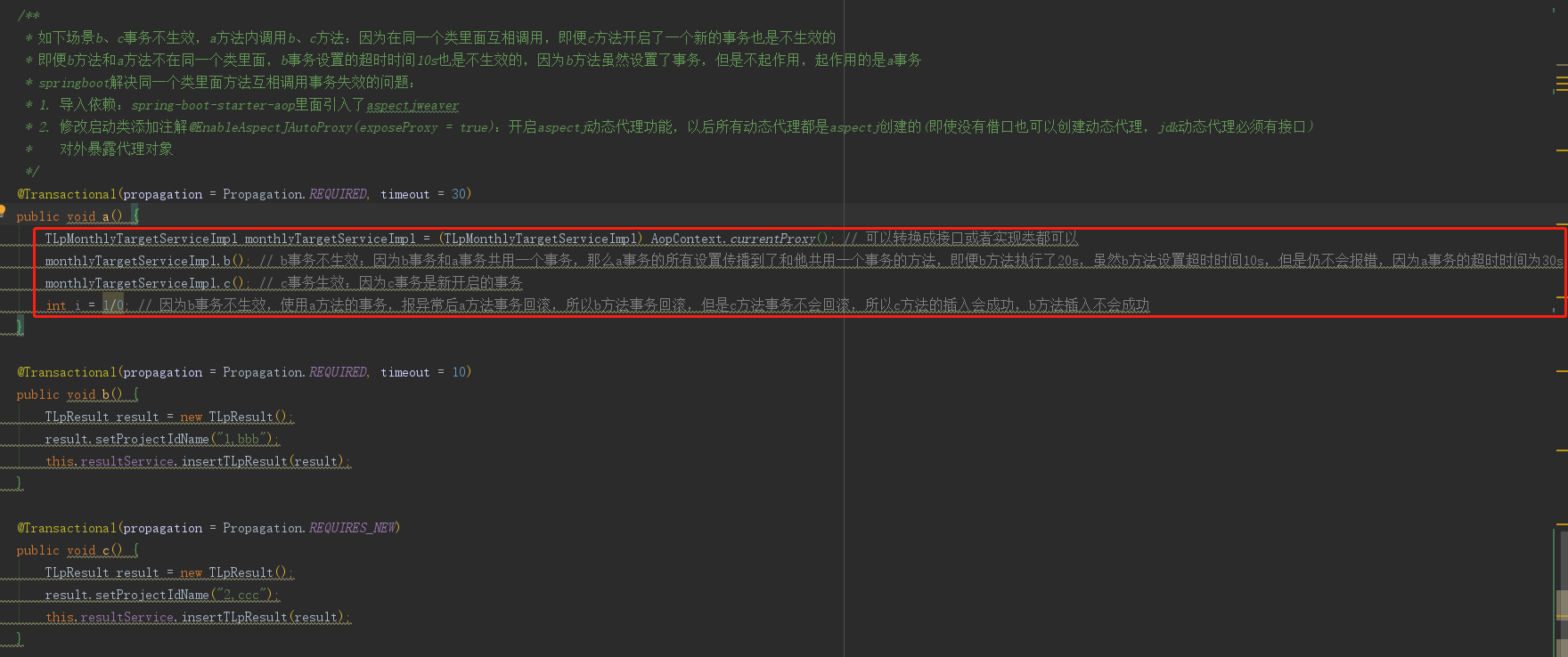


1. 修改启动类添加注解@EnableAspectJAutoProxy(exposeProxy = true)：开启aspectj动态代理功能，以后所有动态代理都是aspectj创建的(即使没有借口也可以创建动态代理，jdk动态代理必须有接口)

@EnableTransactionManagement(proxyTargetClass = true) // 这个不加也可以



3、使用AopContext.currentProxy()生成该方法所在接口或者实现类的代理对象，再进行方法调用



# 分布式事务

## 2.1 CAP定理与BASE理论

### 2.1.1 CAP定理(CAP原则)

CAP原则指的是：CAP最多只能同时实现两点，不可能三者兼得。一般来说，分区容错无法避免，因此P总是成立，剩下的C和A只能二选一

C(Consistency：一致性)：在分布式系统中的所有数据备份，在同一时刻是否是相同的值(等同于所有节点访问同一份最新的数据副本)

A(Availability：可用性)：在集群中一部分节点故障后，集群整体是否还能响应客户端的读写请求(对数据更新具备高可用性)

P(Partition tolerance：分区容错性)：大多数分布式系统都分布在多个子网络。每个子网络就叫做一个区(partition)。分区容错的意思是：区间通信可能失败

### 2.1.2 BASE理论

BASE理论是对CAP理论的延伸，思想是即使无法做到强一致性(CAP的一致性就是强一致性)，但要求可以达到最终一致性

Basically Available(基本可用)：基本可用是指分布式系统在出现故障的时候，允许损失部分可用性(响应时间、功能上的可用性)，

响应时间上的损失：正常情况下搜索引擎需要在0.5秒之内给用户返回相应的查询结果，但由于出现故障(机房断电或者断网)，查询结果的响应时间达到了1-2秒

功能上的损失：购物网站在购物高峰，为了保护系统的稳定性，部分消费者可能会被引导到一个降级页面

Soft State(软状态)：软状态是指允许系统存在中间状态，而该中间状态不会影响系统的整体可用性。分布式存储中一半一份数据会有多个版本，允许不同副本同步的延时就是软状态的体现。Mysql replication的异步复制也是软状态的一种

Eventual Consistency(最终一致性)：最终一致性是指系统中的所有数据副本经过一定的时间后，最终能够达到一致的状态。

## 2.2 分布式系统中实现一致性的算法

（1）raft算法

（2）paxos算法

（3）zab算法

## 2.3 强一致性、弱一致性、最终一致性

从客户角度讲，多线程并发访问时，更新过的数据在不同进程如何获取的不同策略决定了不同的一致性。

强一致性：对于关系型数据库，要求更新过的数据能被后续的访问都能看到

弱一致性：能够容忍后续的部分或者全部访问不到

最终一致性：经过一段时间后要求能访问到更新后的数据。是强一致性的一种特殊情况

## 2.4 分布式事务常见解决方案

刚性事务：遵循ACID原则，强一致性

柔性事务：遵循BASE理论，最终一致性。与刚性事务不同：柔性事务允许一定时间内，不同节点的数据不一致，但要求最终一致

### 2.4.1 两阶段提交(2 Phase Commit:2pc)

两阶段提交由协调者和参与者组成，共经过两个阶段和三个操作：seata

阶段一：准备阶段(prepare)

协调者通知参与者准备提交事务，参与者开始投票

参与者完成准备工作向协调者回应yes(但是不提交事务)

阶段二：提交(commit)/回滚(rollback)阶段

协调者根据参与者的投票结果发起最终的提交或者回滚指令

如果有参与者没有准备好则发起回滚指令，否则发起提交指令

下单减库存：应用程序连接两个数据库，订单数据库和库存数据库

（1）应用程序通过事务协调器向两个数据库发起prepar，两个数据库收到消息分别执行本地事务(记录日志)，但不提交，如果执行成功则回复yes，否则回复no

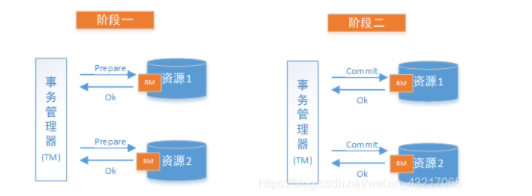
（2）事务协调器收到回复，只要由一方回复no则分别向参与者发起回滚事务，参与者开始回滚事务，全部为yes，此时向参与者发起提交事务。如果参与者有一方提交事务失败则由事务协调器发起回滚事务

优点：强一致性，部分关系型数据库支持

缺点：整个事务的执行需要由协调者在多个节点之间去协调，增加了事务的执行时间；

当协调者崩溃时，参与者不能做出最后的选择，因此参与者在协调者恢复之前保持阻塞等待，性能低下；

协调者存在单点故障，如果协调者出现故障，参与者一直处于锁定状态



### 2.4.2 3pc提交(三阶段提交)

3pc主要是为了解决2pc的阻塞问题

（1）引入了超时机制，同时在协调者和参与者中都引入超时机制

（2）在第一阶段和第二阶段中插入了一个准备阶段，保证了在最后提交之前各参与节点的状态是一致的

阶段一：CanCommit

（1）协调者向参与者发出CanCommit请求，询问是否可以提交事务，并等待所有参与者答复

（2）参与者收到CanCommit请求后，如果任务可以执行事务操作，就返回yes，否则返回no

阶段二：PreCommit

（1）协调者向所有参与者发出PreCommit请求，进入准备阶段

（2）参与者收到PreCommit请求后，执行事务操作但不提交事务，各参与者向协调者反馈ACK响应或NO响应，并等待最终指令

阶段三：DoCommit

（1）所有参与者均反馈ACK响应，即执行真正的事务提交，向所有参与者发出DoCommit请求

（2）参与者收到DoCommit请求后，会正式执行事务提交，并释放这个事务期间占用的资源

（3）参与者向协调者反馈ACK响应，完成事务提交

### 2.4.3柔性事务-TCC事务补偿型方案

TCC(Try Confirm Cancel)事务补偿是基于2PC实现的业务层事务控制方案

Try：检查及预留业务资源完成提交事务前的检查

Confirm：确定执行业务操作，对Try阶段预留的资源正式执行

Cancel：取消执行业务操作，对Try阶段预留的资源进行释放

下单减库存

应用程序连接两个数据库，订单数据库和库存数据库

（1）Try：下单业务有订单服务和库存服务协同完成，在try阶段订单服务和库存服务完成检查和预留资源

订单服务减产当前是否满足提交订单的需求(eg:存在未完成订单不允许提交新的订单)

库存服务检查当前是否有充足的库存，并锁定资源

（2）Confirm：订单服务和库存服务完成Try后开始正式执行资源操作：生成订单信息和扣减库存

（3）Cancel：如果订单服务和库存服务有一方出现失败则全部取消操作：删除订单和还原库存

优点：保证数据最终一致性，在业务层实现事务控制，灵活性好

缺点：每个参与者都需要实现try/confirm/cancel三个接口，并且接口都需要实现幂等性，因为在try、confirm、cancel失败后要重试

### 2.4.4柔性事务-最大努力通知型方案：mq

最大努力通知方案实现方式比较简单，本质上就是通过定期校对，适用于数据一致性时间要求不太高的场合

按规律进行通知，不保证数据一定能通知成功，但会提供可查询操作接口进行核对。这种方案主要用在与第三方系统通讯。eg：调用微信或者支付宝支付后的支付结果通知。这种方案也是结合mq进行实现。eg：通过mq发送http请求，设置最大通知次数，达到通知次数后就不再通知

案例：银行通知、商户通知等(各大交易业务平台间的用户通知：多次通知、查询校对、对账文件)，支付宝的支付成功异步回调

### 2.4.5 柔性事务-基于可靠消息的最终一致性方案：mq

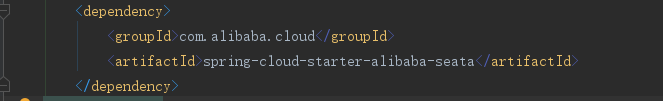
业务处理服务在业务事务提交之前，向实时消息服务请求发送消息，实时消息服务只记录消息数据，而不是真正的发送，业务处理服务在业务事务提交之后，向实时消息服务确认发送。只有在得到确认发送指令后，实时消息服务才会真正发送

### 2.4.6 springcloud alibaba seata

## 2.5 springcloud alibaba seata使用

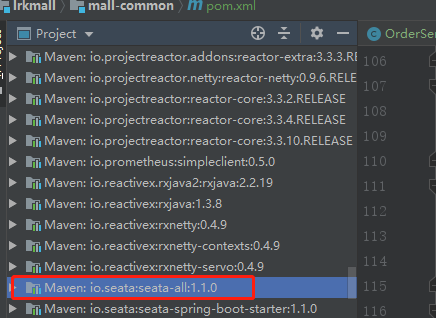
参考文档：<https://www.520java.com/f/article/33.html>

1. 引入seata依赖



（2）下载并解压，这里的seata版本需要根据引入依赖的seata-all的版本来下载

<https://github.com/seata/seata/releases>



注意：高版本的seata压缩包conf配置文件下面没有如下东西

db\_store.sql、db\_undo\_log.sql：sql脚本可以使用0.9.0

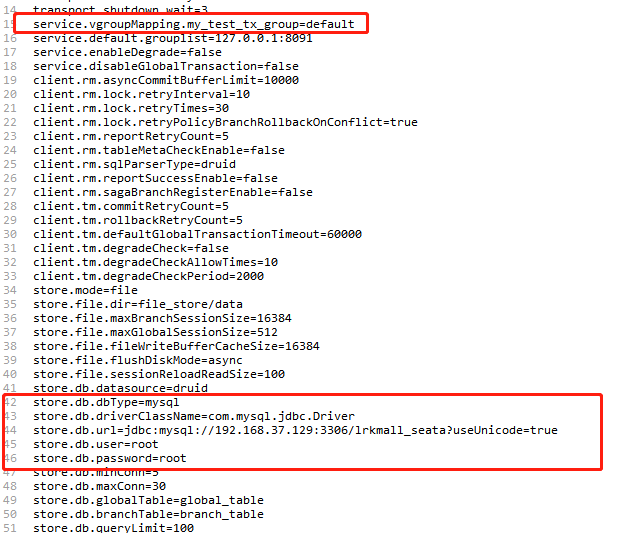
nacos-config.sh、nacos-config.txt：这俩必须从对应的版本进行下载。

下载地址：<https://github.com/seata/seata/tree/1.1.0>

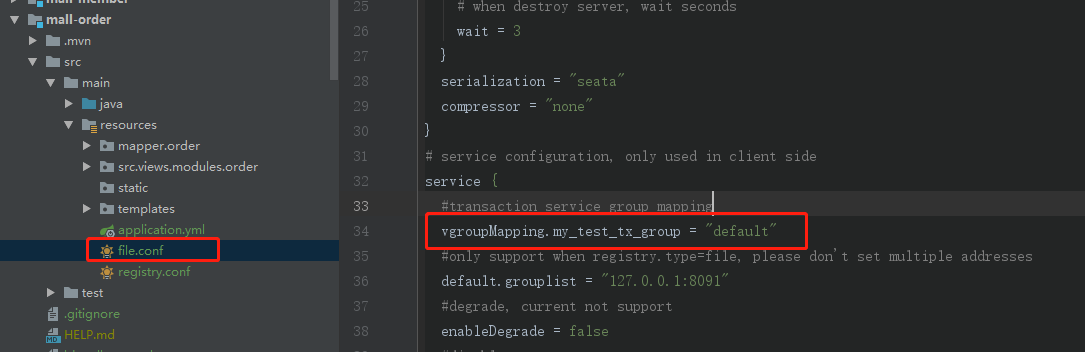
并且1.0.0版本以后的nacos-config.txt名字变为config.txt，下载好之后不要放到conf文件夹下面，需要放到conf上一级(即在seata文件夹下面)，config.txt文件需要修改两个地方： service.vgroupMapping组名(必须要与代码里面application.yml配置文件中一样以及代码中引入的file.conf中的vgroupMapping一样)

seata数据库的配置

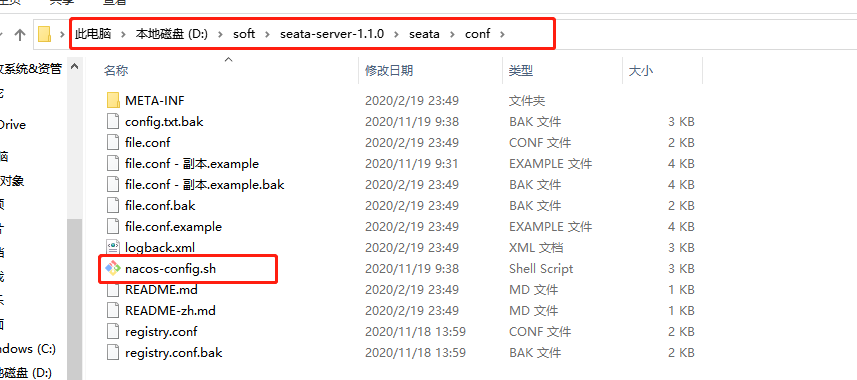








nacos-config.sh需要放到confi文件夹下，执行初始化脚本nacos-config.sh配置到nacos中



先启动nacos，再启动seata，seata会注册到nacos



（3）新建seata数据库：将conf文件夹下db\_store.sql脚本执行

（4）在每个业务数据库下新建undo\_log表：将conf文件夹下db\_undo\_log.sql脚本执行

（5）修改registry.conf配置文件

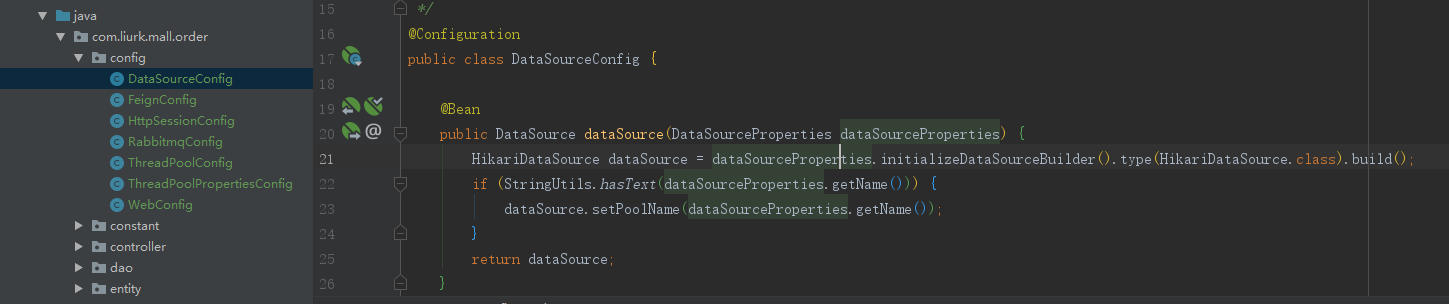
修改registry模块将type改为nacos并修改nacos配置文件serverAddr

（6）修改file.conf配置文件

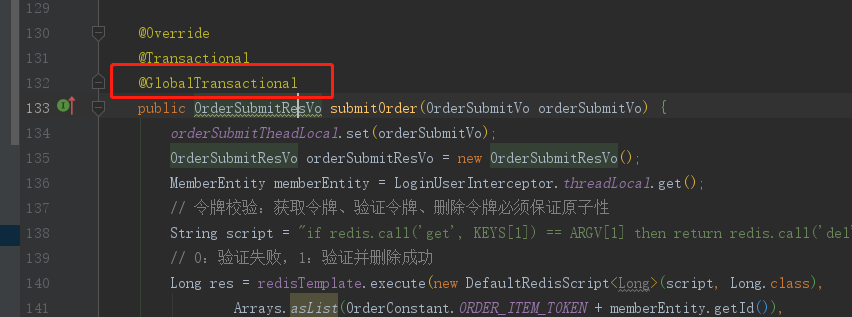
修改vgroupMapping以及store的mode为db

（7）将registry.conf、file.conf配置文件复制到对应的业务模块配置文件resources下

（8）编写配置类：让seata代理数据源



（9）在整个大事务的入口处添加注解：@GlobalTransaction，每个远程事务添加@Transaction



### seata缺点：

seata不适合高并发的分布式事务，因为seata会加锁，相当于串行化执行

## 2.6 消息队列+最终一致性