

1. 分布式事务问题

1.1. 分布式之前

1.2. 分布式之后

1.3. 一句话

2. Seata简介

2.1. 是什么

2.2. 能干嘛

2.2.1. 分布式事务处理过程的-ID+三组件模型

2.2.1.1. Transaction ID XID

2.2.1.2. 3组件概念

2.2.2. 处理过程

2.3. 去哪下

2.4. 怎么玩

3. Seata-Server安装

3.1. 官网地址

3.2. 下载版本

3.3. seata-server-0.9.0.zip解压到指定目录并修改conf目录下的file.conf配置文件

3.4. mysql5.7数据库新建库seata

3.5. 在seata库里建表

3.6. 修改seata-server-0.9.0\seata\conf目录下的registry.conf配置文件

3.7. 先启动Nacos端口号8848

3.8. 再启动seata-server

4. 订单/库存/账户业务数据库准备

4.1. 以下演示都需要先启动Nacos后启动Seata，保证两个都OK

4.2. 分布式事务业务说明

4.2.1. 业务说明

4.2.2. 业务流程

4.3. 创建业务数据库

- 4.4. 按照上述3库分别建对应业务表
- 4.5. 按照上述3库分别建对应的回滚日志表
- 4.6. 最终效果
- 5. 订单/库存/账户业务微服务准备
 - 5.1. 业务需求
 - 5.2. 新建订单Order-Module
 - 5.2.1. 新建seata-order-service2001
 - 5.2.2. POM
 - 5.2.3. YML
 - 5.2.4. file.conf
 - 5.2.5. registry.conf
 - 5.2.6. domain
 - 5.2.6.1. CommonResult
 - 5.2.6.2. Order
 - 5.2.7. Dao接口及实现
 - 5.2.7.1. OrderDao
 - 5.2.7.2. OrderMapper.xml
 - 5.2.8. Service接口及实现
 - 5.2.8.1. OrderService
 - 5.2.8.2. OrderServiceImpl
 - 5.2.8.3. StorageService
 - 5.2.8.4. AccountService
 - 5.2.9. Controller
 - 5.2.10. config配置
 - 5.2.10.1. MyBatisConfig
 - 5.2.10.2. DataSourceProxyConfig
 - 5.2.11. 主启动
 - 5.3. 新建库存Storage-Module
 - 5.3.1. 新建seata-order-service2002
 - 5.3.2. POM

5.3.3. YML

5.3.4. file.conf

5.3.5. registry.conf

5.3.6. domain

5.3.6.1. CommonResult

5.3.6.2. Storage

5.3.7. Dao接口及实现

5.3.7.1. StorageDao

5.3.7.2. StorageMapper.xml

5.3.8. Service接口及实现

5.3.8.1. StorageService

5.3.8.2. StorageServiceImpl

5.3.9. Controller

5.3.10. Config配置

5.3.10.1. MyBatisConfig

5.3.10.2. DataSourceProxyConfig

5.3.11. 主启动

5.4. 新建账户Account-Module

5.4.1. 新建seata-order-service2003

5.4.2. POM

5.4.3. YML

5.4.4. file.conf

5.4.5. registry.conf

5.4.6. domain

5.4.6.1. CommonResult

5.4.6.2. Account

5.4.7. Dao接口及实现

5.4.7.1. AccountDao

5.4.7.2. AccountMapper.xml

5.4.8. Service接口及实现

5.4.8.1. AccountService

5.4.8.2. AccountServiceImpl

5.4.9. Controller

5.4.10. Config配置

5.4.10.1. MyBatisConfig

5.4.10.2. DataSourceProxyConfig

5.4.11. 主启动

6. 测试

6.1. 下订单->减库存->扣余额->改（订单）状态

6.2. 数据库初始情况

6.3. 正常下单

6.4. 超时异常，没加@GlobalTransactional

6.5. 超时异常，添加@GlobalTransactional

7. Seata之原理简介

7.1. Seata

7.2. 再看TC/TM/RM三大组件

7.2.1. 分布式事务的执行流程

7.3. AT模式如何做到对业务的无侵入

7.3.1. 是什么

7.3.2. 一阶段加载

7.3.3. 二阶段提交

7.3.4. 二阶段回滚

7.4. 补充

1. 分布式事务问题

1.1. 分布式之前

单机单库没这个问题

从1: 1 -> 1:N -> N: N

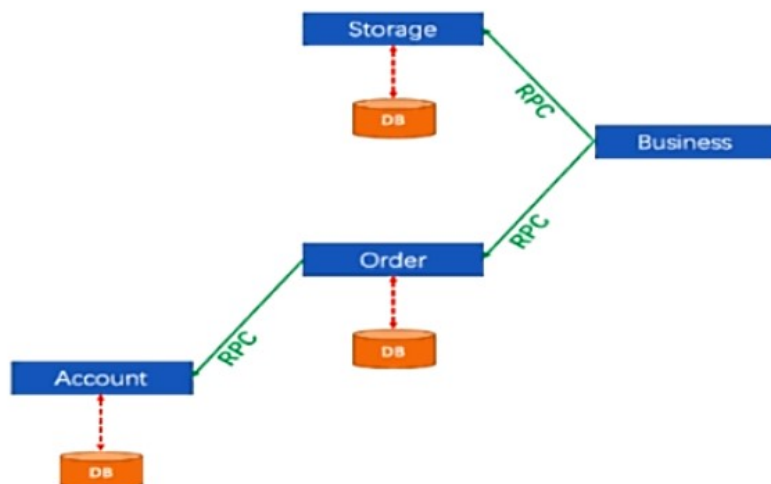
1.2. 分布式之后

单体应用被拆分成微服务应用，原来的三个模块被拆分成三个独立的应用，分别使用三个独立的数据源，业务操作需要调用三个服务来完成。此时每个服务内部的数据一致性由本地事务来保证，但是全局的数据一致性问题没法保证

用户购买商品的业务逻辑。整个业务逻辑由3个微服务提供支持：

- 仓储服务：对给定的商品扣除仓储数量。
- 订单服务：根据采购需求创建订单。
- 帐户服务：从用户帐户中扣除余额。

架构图



1.3. 一句话

一次业务操作需要跨多个数据源或需要跨多个系统进行远程调用，就会产生分布式事务问题

2. Seata简介

2.1. 是什么

Seata是一款开源的分布式事务解决方案，致力于在微服务架构下提供高性能和简单易用的分布式事务服务

官网地址: <http://seata.io/zh-cn/>

2.2. 能干嘛

一个典型的分布式事务过程

2.2.1. 分布式事务处理过程的-ID+三组件模型

2.2.1.1. Transaction ID XID

全局唯一的事务ID

2.2.1.2. 3组件概念

Transaction Coordinator(TC)

事务协调器，维护全局事务的运行状态，负责协调并驱动全局事务的提交或回滚；

Transaction Manager(TM)

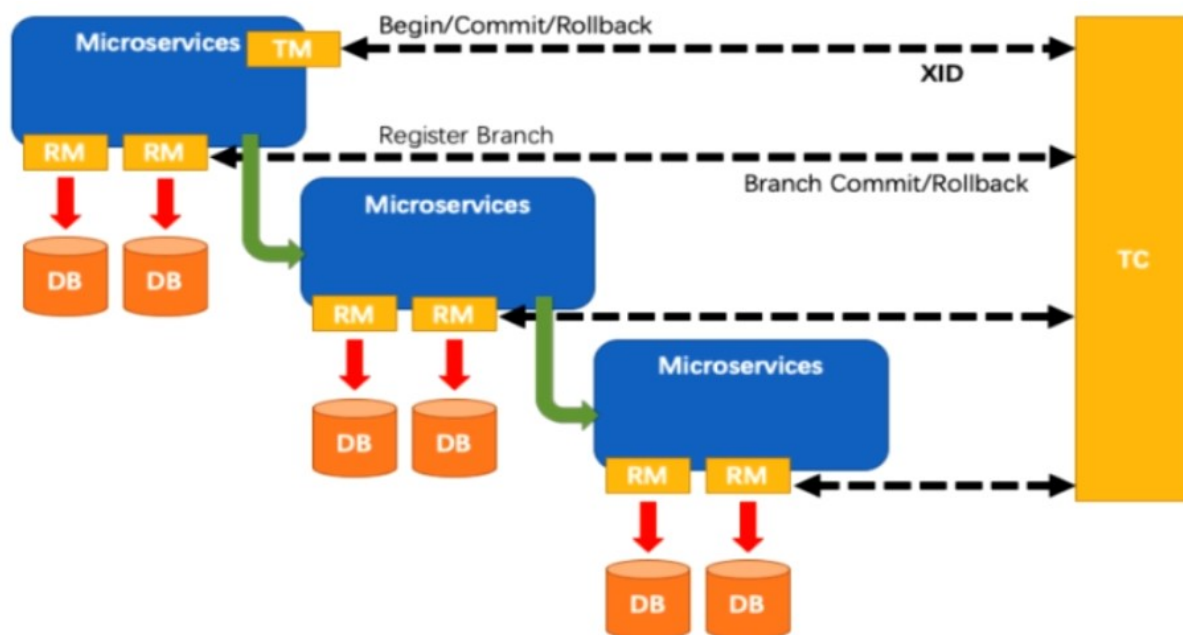
控制全局事务的边界，负责开启一个全局事务，并最终发起全局提交或全局回滚的决议；

Resource Manager(RM)

控制分支事务，负责分支注册，状态汇报，并接收事务协调器的指令，驱动分支（本地）事务的提交和回滚；

2.2.2. 处理过程

- 1.TM向TC申请开启一个全局事务，全局事务创建成功并生成一个全局唯一的XID
- 2.XID在微服务调用链路的上下文传播；
- 3.RM向TC注册分支事务，将其纳入XID对应全局事务的管辖；
- 4.TM向TC发起针对XID的全局提交或回滚决议；
- 5.TC调度XID下管辖的全部分支事务完成提交或回滚请求



2.3. 去哪下

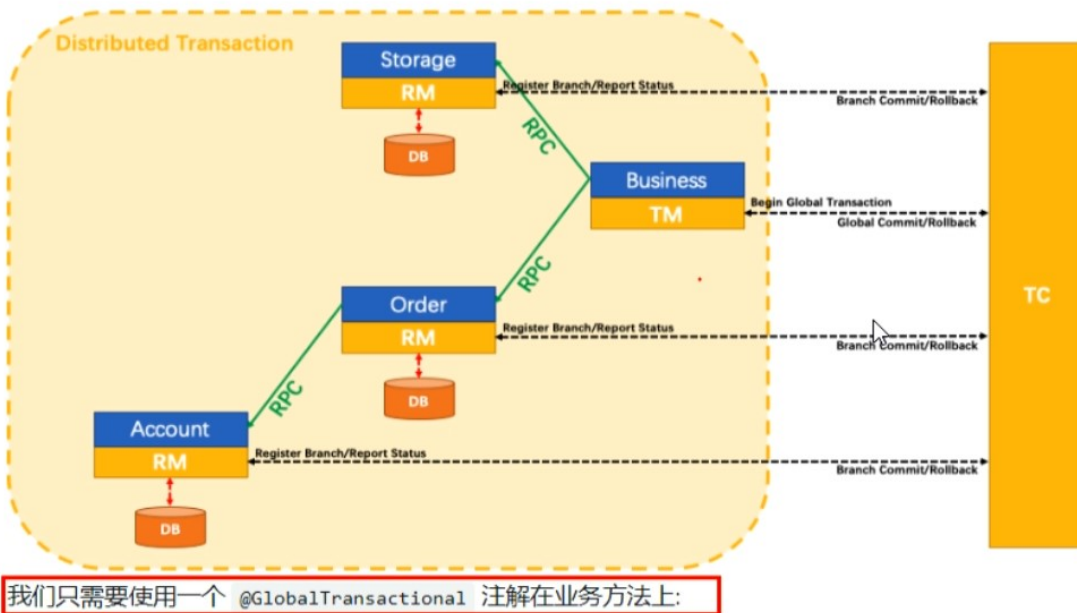
发布说明: <https://github.com/seata/seata/releases>

2.4. 怎么玩

Spring 本地 @Transactional

全局@GlobalTransactional: SEATA的分布式交易解决方案

SEATA 的分布式交易解决方案



3. Seata-Server安装

3.1. 官网地址

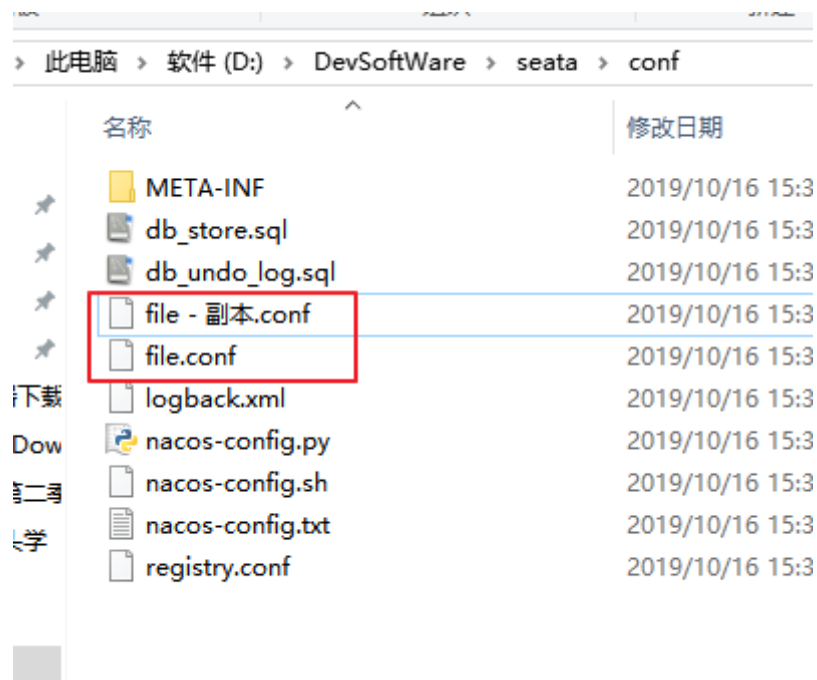
<http://seata.io/zh-cn/>

3.2. 下载版本

这里使用的版本为0.9.0

3.3. seata-server-0.9.0.zip解压到指定目录并修改conf目录下的file.conf配置文件

1) 先备份原始file.conf文件



2) 主要修改：自定义事务组名称+事务日志存储模式为db+数据库连接信息

3) file.conf

- service模块

```
1. vgroup_mapping.my_test_tx_group = "fsp_tx_group"
```

```
28 }
29 service {
30     #vgroup->rgroup
31     vgroup_mapping.my_test_tx_group = "fsp_tx_group"
32     #only support single node
33     default.grouplist = "127.0.0.1:8091"
34     #degrade current not support
35     enableDegrade = false
36     #disable
37     disable = false
38     #unit ms,s,m,h,d represents milliseconds, seconds, minutes, hours, days, default permanent
39     max.commit.retry.timeout = "-1"
40     max.rollback.retry.timeout = "-1"
41 }
42
```

- store模块

```
1. mode = "db"
2. url = "jdbc:mysql://127.0.0.1:3306/seata"
3. user = "root"
4. password = "你自己的密码"
```

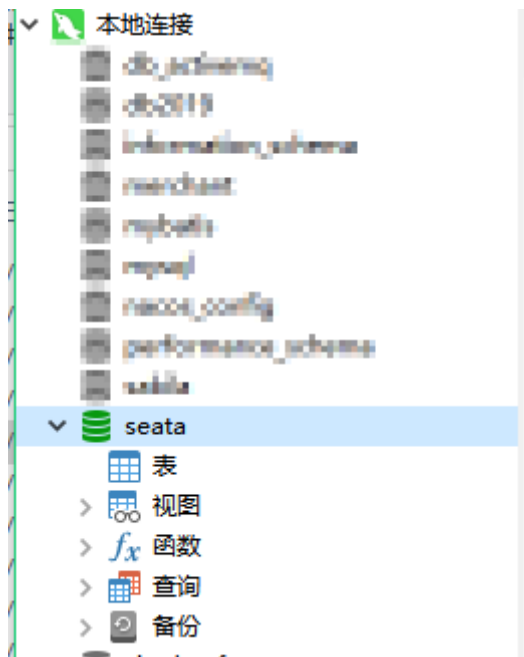


```

54 ## transaction log store
55 store {
56     ## store mode: file、db
57     mode = "db"
58
59     ## file store
60     file {
61         dir = "sessionStore"
62
63         # branch session size , if exceeded first try compress lockkey, still exceeded throws exceptions
64         max-branch-session-size = 16384
65         # globe session size , if exceeded throws exceptions
66         max-global-session-size = 512
67         # file buffer size , if exceeded allocate new buffer
68         file-write-buffer-cache-size = 16384
69         # when recover batch read size
70         session.reload.read_size = 100
71         # async, sync
72         flush-disk-mode = async
73     }
74
75     ## database store
76     db {
77         ## the implement of javax.sql.DataSource, such as DruidDataSource(druid)/BasicDataSource(dbcp) etc.
78         datasource = "dbcp"
79         ## mysql/oracle/h2/oceanbase etc.
80         db-type = "mysql"
81         driver-class-name = "com.mysql.jdbc.Driver"
82         url = "jdbc:mysql://127.0.0.1:3306/seata"
83         user = "root"
84         password = "root"
85     }
86 }

```

3.4. mysql5.7数据库新建库seata



3.5. 在seata库里建表

建表db_store.sql在\seata-server-0.9.0\seata\conf目录里面 => db_store.sql

> 此电脑 > 软件 (D:) > DevSoftWare > seata > conf

	名称	修改日期	类型
	META-INF	2019/10/16 15:38	文件夹
	db_store.sql	2019/10/16 15:38	SQL 文件
	db_undo_log.sql	2019/10/16 15:38	SQL 文件
	file - 副本.conf	2019/10/16 15:38	CON
	file.conf	2020/7/12 9:34	CON
请下载	logback.xml	2019/10/16 15:38	XML
kDow	nacos-config.py	2019/10/16 15:38	Python 文件
第二遍	nacos-config.sh	2019/10/16 15:38	SH 文件
头学	nacos-config.txt	2019/10/16 15:38	文本文件
	registry.conf	2019/10/16 15:38	CON

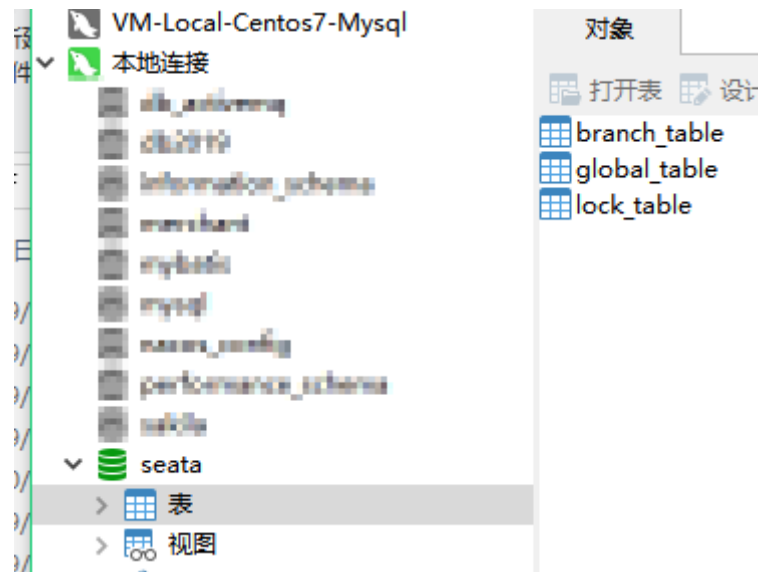
```
1.  -- the table to store GlobalSession data
2.  drop table if exists `global_table`;
3.  create table `global_table` (
4.      `xid` varchar(128) not null,
5.      `transaction_id` bigint,
6.      `status` tinyint not null,
7.      `application_id` varchar(32),
8.      `transaction_service_group` varchar(32),
9.      `transaction_name` varchar(128),
10.     `timeout` int,
11.     `begin_time` bigint,
12.     `application_data` varchar(2000),
13.     `gmt_create` datetime,
14.     `gmt_modified` datetime,
15.     primary key (`xid`),
16.     key `idx_gmt_modified_status` (`gmt_modified`, `status`),
17.     key `idx_transaction_id` (`transaction_id`)
18. );
19.
20. -- the table to store BranchSession data
21. drop table if exists `branch_table`;
22. create table `branch_table` (
23.     `branch_id` bigint not null,
24.     `xid` varchar(128) not null,
25.     `transaction_id` bigint ,
26.     `resource_group_id` varchar(32),
27.     `resource_id` varchar(256) ,
28.     `lock_key` varchar(128) ,
29.     `branch_type` varchar(8) ,
30.     `status` tinyint,
31.     `client_id` varchar(64),
32.     `application_data` varchar(2000),
33.     `gmt_create` datetime,
34.     `gmt_modified` datetime,
35.     primary key (`branch_id`),
36.     key `idx_xid` (`xid`)
37. );
38.
39. -- the table to store lock data
40. drop table if exists `lock_table`;
41. create table `lock_table` (
42.     `row_key` varchar(128) not null,
43.     `xid` varchar(96),
44.     `transaction_id` long ,
45.     `branch_id` long,
46.     `resource_id` varchar(256) ,
47.     `table_name` varchar(32) ,
48.     `pk` varchar(36) ,
49.     `gmt_create` datetime ,
```

```

50.     `gmt_modified` datetime,
51.     primary key(`row_key`)
52. );

```

查看表是否导入成功

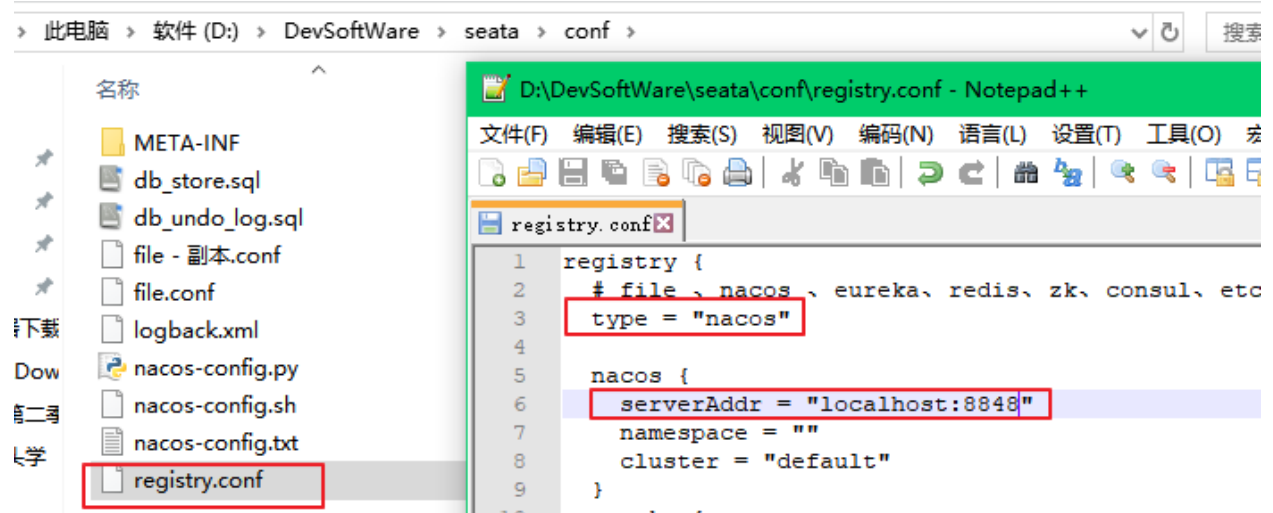


3.6. 修改seata-server-0.9.0\seata\conf目录下的registry.conf配置文件

```

1. registry {
2.     # file 、 nacos 、 eureka、 redis、 zk、 consul、 etcd3、 sofa
3.     type = "nacos"
4.
5.     nacos {
6.         serverAddr = "localhost:8848"
7.         namespace = ""
8.         cluster = "default"
9.     }


```



目的是：指明注册中心为nacos，及修改nacos连接信息

3.7. 先启动Nacos端口号8848

启动nacos\bin下startup.cmd



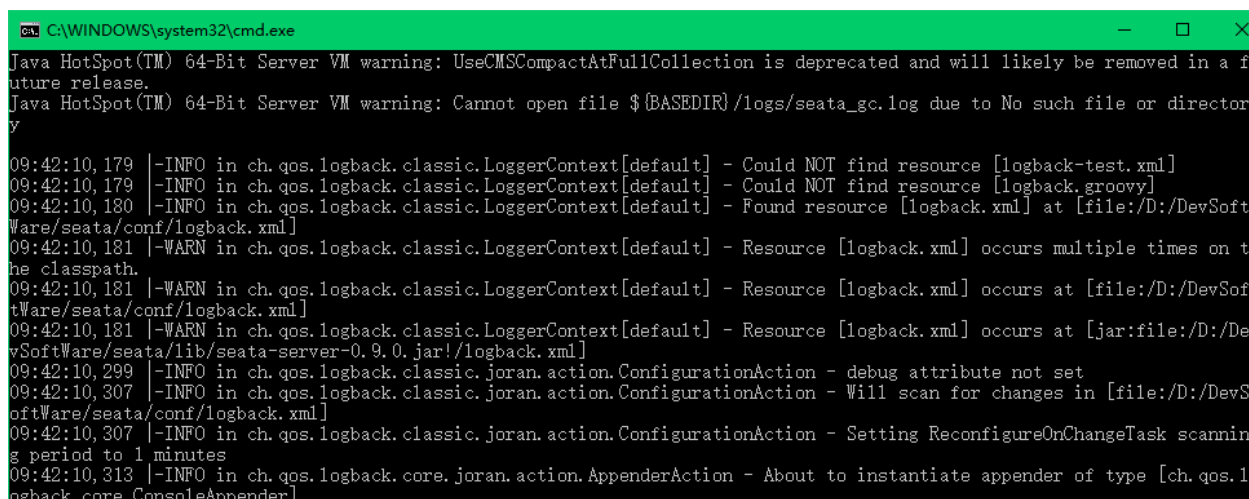
```
C:\WINDOWS\system32\cmd.exe

Nacos 1.1.4
Running in stand alone mode, All function modules
Port: 8848
Pid: 13108
Console: http://192.168.5.1:8848/nacos/index.html
https://nacos.io

2020-07-12 09:40:59,805 INFO Bean 'org.springframework.security.config.annotation.configuration.ObjectPostProcessorConfiguration' of type [org.springframework.security.config.annotation.configuration.ObjectPostProcessorConfiguration$$EnhancerBySpringCGLIB$$c7a5e308] is not eligible for getting processed by all BeanPostProcessors (for example: not eligible for auto-proxying)
2020-07-12 09:40:59,978 INFO Bean 'objectPostProcessor' of type [org.springframework.security.config.annotation.configuration.AutowireBeanFactoryObjectPostProcessor] is not eligible for getting processed by all BeanPostProcessors (for example: not eligible for auto-proxying)
```

3.8. 再启动seata-server

启动seata-server-0.9.0\seata\bin下seata-server.bat



```
C:\WINDOWS\system32\cmd.exe

Java HotSpot(TM) 64-Bit Server VM warning: UseCMSCompactAtFullCollection is deprecated and will likely be removed in a future release.
Java HotSpot(TM) 64-Bit Server VM warning: Cannot open file $(BASEDIR)/logs/seata_gc.log due to No such file or directory

09:42:10,179 |-INFO in ch.qos.logback.classic.LoggerContext[default] - Could NOT find resource [logback-test.xml]
09:42:10,179 |-INFO in ch.qos.logback.classic.LoggerContext[default] - Could NOT find resource [logback.groovy]
09:42:10,180 |-INFO in ch.qos.logback.classic.LoggerContext[default] - Found resource [logback.xml] at [file:/D:/DevSoftware/seata/conf/logback.xml]
09:42:10,181 |-WARN in ch.qos.logback.classic.LoggerContext[default] - Resource [logback.xml] occurs multiple times on the classpath.
09:42:10,181 |-WARN in ch.qos.logback.classic.LoggerContext[default] - Resource [logback.xml] occurs at [file:/D:/DevSoftware/seata/conf/logback.xml]
09:42:10,181 |-WARN in ch.qos.logback.classic.LoggerContext[default] - Resource [logback.xml] occurs at [jar:file:/D:/DevSoftware/seata/lib/seata-server-0.9.0.jar!/logback.xml]
09:42:10,299 |-INFO in ch.qos.logback.classic.joran.action.ConfigurationAction - debug attribute not set
09:42:10,307 |-INFO in ch.qos.logback.classic.joran.action.ConfigurationAction - Will scan for changes in [file:/D:/DevSoftware/seata/conf/logback.xml]
09:42:10,307 |-INFO in ch.qos.logback.classic.joran.action.ConfigurationAction - Setting ReconfigureOnChangeTask scanning period to 1 minutes
09:42:10,313 |-INFO in ch.qos.logback.core.joran.action.AppenderAction - About to instantiate appender of type [ch.qos.logback.core.ConsoleAppender]
```

4. 订单/库存/账户业务数据库准备

4.1. 以下演示都需要先启动Nacos后启动Seata，保证两个都OK

Seata没启动报错no available server to connect

4.2. 分布式事务业务说明

4.2.1. 业务说明

这里我们会创建三个服务，一个订单服务，一个库存服务，一个账户服务。

当用户下单时，会在订单服务中创建一个订单，然后通过远程调用库存服务来扣减下单商品的库存，再通过远程调用账户服务来扣减用户账户里面的余额，最后在订单服务中修改订单状态为已完成

该操作跨越三个数据库，有两次远程调用，很明显会有分布式事务问题。

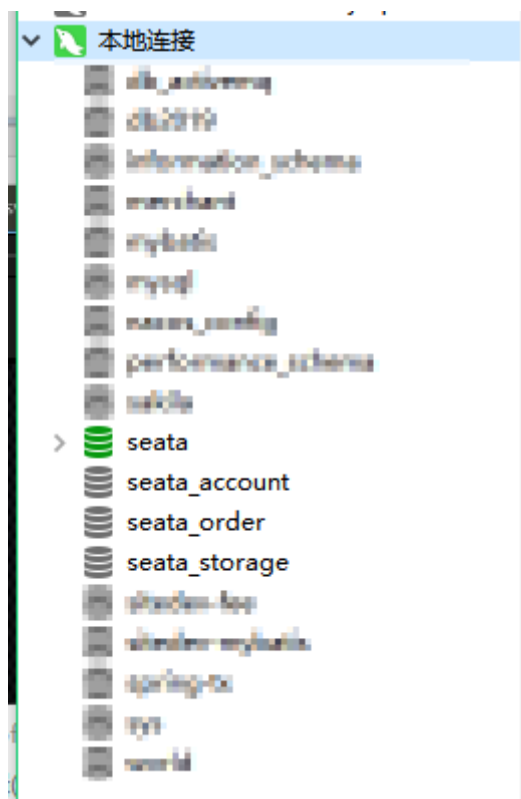
4.2.2. 业务流程

下订单-->扣库存-->减账户（余额）

4.3. 创建业务数据库

- 1) seata_order: 存储订单的数据库
- 2) seata_storage: 存储库存的数据库
- 3) seata_account: 存储账户信息的数据库
- 4) 建表SQL:

```
1. CREATE DATABASE seata_order;
2.
3. CREATE DATABASE seata_storage;
4.
5. CREATE DATABASE seata_account;
```



4.4. 按照上述3库分别建对应业务表

- 1) seata_order库下建t_order表

```

1.  USE seata_order;
2.
3.  CREATE TABLE t_order(
4.      `id` BIGINT(11) NOT NULL AUTO_INCREMENT PRIMARY KEY,
5.      `user_id` BIGINT(11) DEFAULT NULL COMMENT '用户id',
6.      `product_id` BIGINT(11) DEFAULT NULL COMMENT '产品id',
7.      `count` INT(11) DEFAULT NULL COMMENT '数量',
8.      `money` DECIMAL(11,0) DEFAULT NULL COMMENT '金额',
9.      `status` INT(1) DEFAULT NULL COMMENT '订单状态: 0: 创建中; 1: 已完结'
10. ) ENGINE=INNODB AUTO_INCREMENT=7 DEFAULT CHARSET=utf8;
11.
12. SELECT * FROM t_order;

```

2) seata_storage库下建t_storage表

```

1.  USE seata_storage;
2.
3.  CREATE TABLE t_storage(
4.      `id` BIGINT(11) NOT NULL AUTO_INCREMENT PRIMARY KEY,
5.      `product_id` BIGINT(11) DEFAULT NULL COMMENT '产品id',
6.      `total` INT(11) DEFAULT NULL COMMENT '总库存',
7.      `used` INT(11) DEFAULT NULL COMMENT '已用库存',
8.      `residue` INT(11) DEFAULT NULL COMMENT '剩余库存'
9. ) ENGINE=INNODB AUTO_INCREMENT=2 DEFAULT CHARSET=utf8;
10.
11. INSERT INTO seata_storage.t_storage(`id`,`product_id`,`total`,`used`,`residue`)
12. VALUES('1','1','100','0','100');
13.
14. SELECT * FROM t_storage;

```

3) seata_account库下建t_account表

```

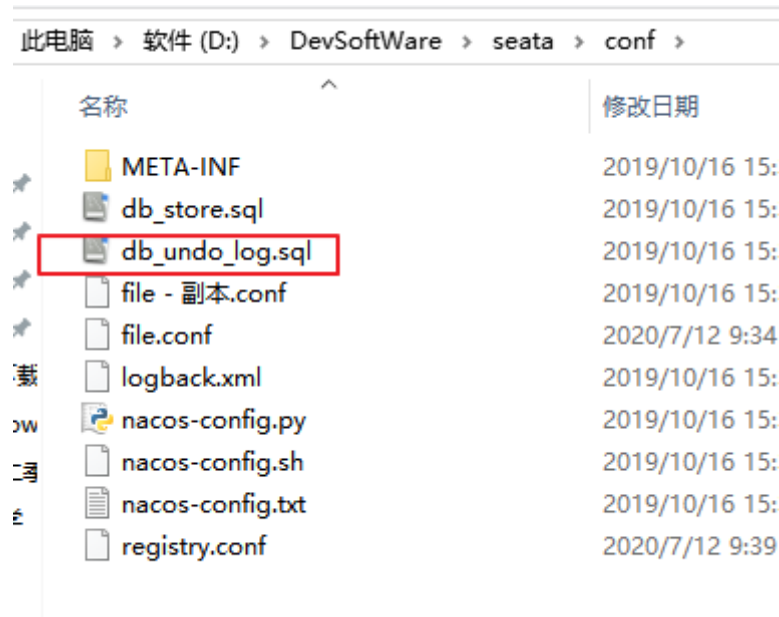
1.  USE seata_account;
2.
3.  CREATE TABLE t_account(
4.      `id` BIGINT(11) NOT NULL AUTO_INCREMENT PRIMARY KEY COMMENT 'id',
5.      `user_id` BIGINT(11) DEFAULT NULL COMMENT '用户id',
6.      `total` DECIMAL(10,0) DEFAULT NULL COMMENT '总额度',
7.      `used` DECIMAL(10,0) DEFAULT NULL COMMENT '已用余额',
8.      `residue` DECIMAL(10,0) DEFAULT '0' COMMENT '剩余可用额度'
9. ) ENGINE=INNODB AUTO_INCREMENT=2 DEFAULT CHARSET=utf8;
10.
11. INSERT INTO seata_account.t_account(`id`,`user_id`,`total`,`used`,`residue`)
12. VALUES('1','1','1000','0','1000')
13.
14. SELECT * FROM t_account;

```

4.5. 按照上述3库分别建对应的回滚日志表

1) 订单-库存-账户3个库下都需要建各自的回滚日志表

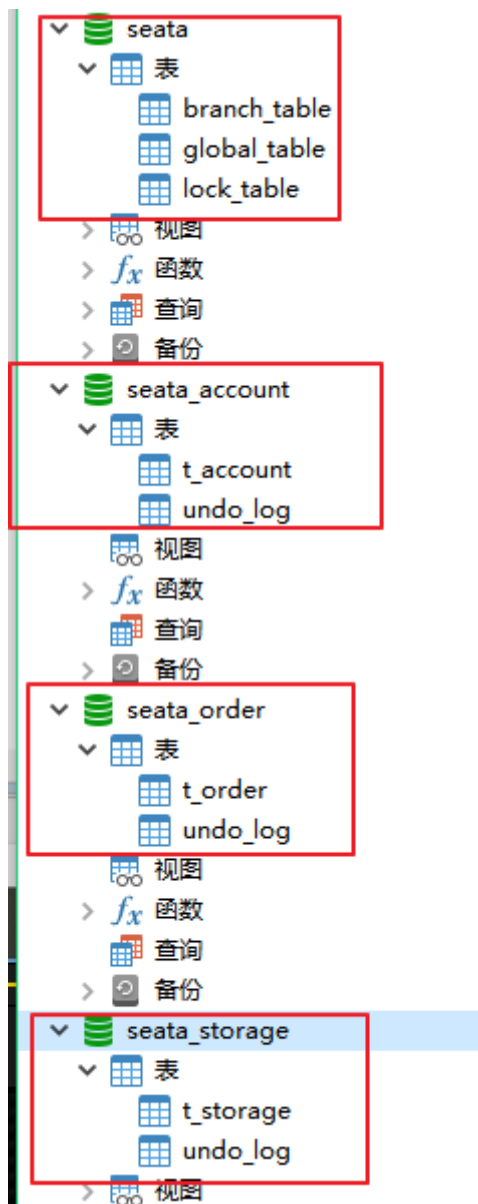
2) \seata-server-0.9.0\seata\conf目录下的db_undo_log.sql



3) 建表SQL

```
1. drop table `undo_log`;  
2. CREATE TABLE `undo_log` (  
3.   `id` bigint(20) NOT NULL AUTO_INCREMENT,  
4.   `branch_id` bigint(20) NOT NULL,  
5.   `xid` varchar(100) NOT NULL,  
6.   `context` varchar(128) NOT NULL,  
7.   `rollback_info` longblob NOT NULL,  
8.   `log_status` int(11) NOT NULL,  
9.   `log_created` datetime NOT NULL,  
10.  `log_modified` datetime NOT NULL,  
11.  `ext` varchar(100) DEFAULT NULL,  
12.  PRIMARY KEY (`id`),  
13.  UNIQUE KEY `ux_undo_log` (`xid`, `branch_id`)  
14. ) ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT CHARSET=utf8;
```

4.6. 最终效果




5. 订单/库存/账户业务微服务准备

5.1. 业务需求

下订单->减库存->扣余额->改（订单）状态

5.2. 新建订单Order-Module

5.2.1. 新建seata-order-service2001

 New Module

Parent:

Name:

Location:

► Artifact Coordinates

5.2.2. POM

```
1. <?xml version="1.0" encoding="UTF-8"?>
2. <project xmlns="http://maven.apache.org/POM/4.0.0"
3.         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4.         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
5.     <parent>
6.         <artifactId>cloud2020</artifactId>
7.         <groupId>cn.sitedev.springcloud</groupId>
8.         <version>1.0-SNAPSHOT</version>
9.     </parent>
10.    <modelVersion>4.0.0</modelVersion>
11.
12.    <artifactId>seata-order-service2001</artifactId>
13.
14.    <dependencies>
15.        <!--nacos-->
16.        <dependency>
17.            <groupId>com.alibaba.cloud</groupId>
18.            <artifactId>spring-cloud-starter-alibaba-nacos-
discovery</artifactId>
19.        </dependency>
20.        <!--seata-->
21.        <dependency>
22.            <groupId>com.alibaba.cloud</groupId>
23.            <artifactId>spring-cloud-starter-alibaba-seata</artifactId>
24.            <exclusions>
25.                <exclusion>
26.                    <artifactId>seata-all</artifactId>
27.                    <groupId>io.seata</groupId>
28.                </exclusion>
29.            </exclusions>
30.        </dependency>
31.        <dependency>
32.            <groupId>io.seata</groupId>
33.            <artifactId>seata-all</artifactId>
34.            <version>0.9.0</version>
35.        </dependency>
36.        <!--feign-->
37.        <dependency>
38.            <groupId>org.springframework.cloud</groupId>
39.            <artifactId>spring-cloud-starter-openfeign</artifactId>
40.        </dependency>
41.        <!--web-actuator-->
42.        <dependency>
43.            <groupId>org.springframework.boot</groupId>
44.            <artifactId>spring-boot-starter-web</artifactId>
45.        </dependency>
46.        <dependency>
47.            <groupId>org.springframework.boot</groupId>
```

```
48.         <artifactId>spring-boot-starter-actuator</artifactId>
49.     </dependency>
50.     <!--mysql-druid-->
51.     <dependency>
52.         <groupId>mysql</groupId>
53.         <artifactId>mysql-connector-java</artifactId>
54.         <version>5.1.37</version>
55.     </dependency>
56.     <dependency>
57.         <groupId>com.alibaba</groupId>
58.         <artifactId>druid-spring-boot-starter</artifactId>
59.         <version>1.1.10</version>
60.     </dependency>
61.     <dependency>
62.         <groupId>org.mybatis.spring.boot</groupId>
63.         <artifactId>mybatis-spring-boot-starter</artifactId>
64.         <version>2.0.0</version>
65.     </dependency>
66.     <dependency>
67.         <groupId>org.springframework.boot</groupId>
68.         <artifactId>spring-boot-starter-test</artifactId>
69.         <scope>test</scope>
70.     </dependency>
71.     <dependency>
72.         <groupId>org.projectlombok</groupId>
73.         <artifactId>lombok</artifactId>
74.         <optional>true</optional>
75.     </dependency>
76. </dependencies>
77.
78. </project>
```

5.2.3. YML

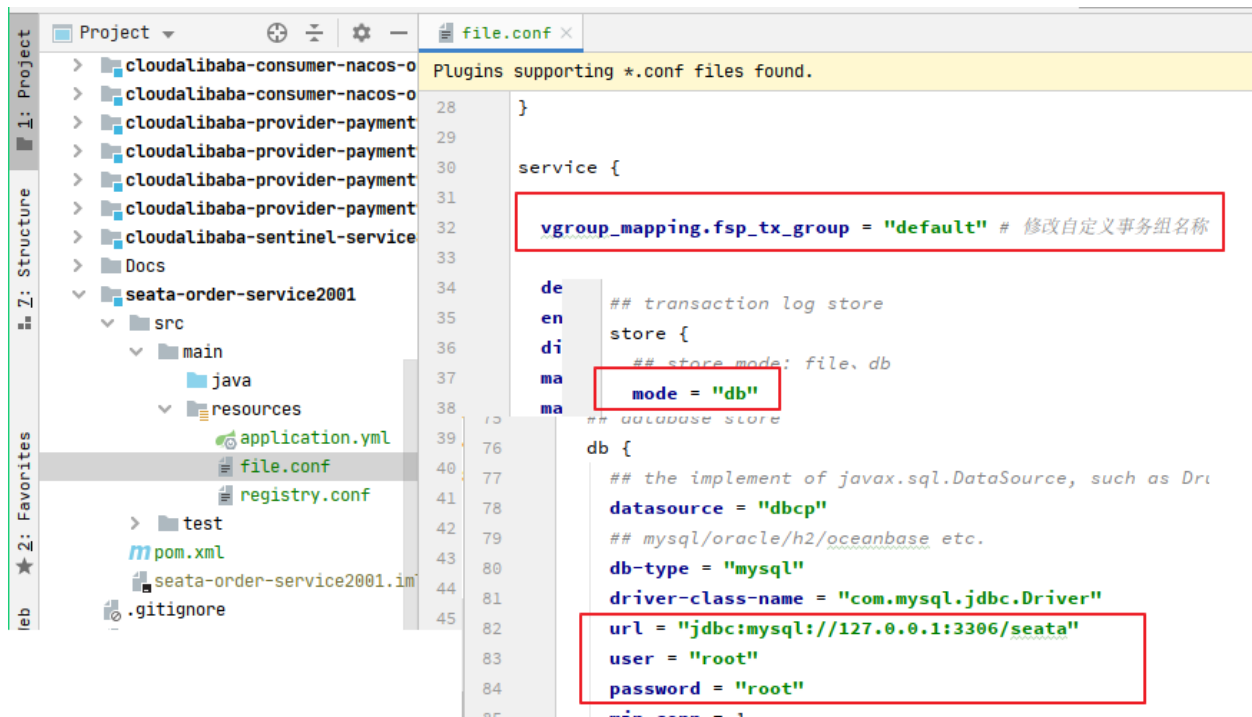
```
1.  server:
2.    port: 2001
3.
4.  spring:
5.    application:
6.      name: seata-order-service
7.    cloud:
8.      alibaba:
9.        seata:
10.         #自定义事务组名称需要与seata-server中的对应
11.         tx-service-group: fsp_tx_group
12.      nacos:
13.        discovery:
14.          server-addr: localhost:8848
15.    datasource:
16.      driver-class-name: com.mysql.jdbc.Driver
17.      url: jdbc:mysql://localhost:3306/seata_order
18.      username: root
19.      password: root
20.
21.  feign:
22.    hystrix:
23.      enabled: false
24.
25.  logging:
26.    level:
27.      io:
28.        seata: info
29.
30.  mybatis:
31.    mapperLocations: classpath:mapper/*.xml
```

5.2.4. file.conf

```
1.  transport {
2.      # tcp udt unix-domain-socket
3.      type = "TCP"
4.      #NIO NATIVE
5.      server = "NIO"
6.      #enable heartbeat
7.      heartbeat = true
8.      #thread factory for netty
9.      thread-factory {
10.         boss-thread-prefix = "NettyBoss"
11.         worker-thread-prefix = "NettyServerNIOWorker"
12.         server-executor-thread-prefix = "NettyServerBizHandler"
13.         share-boss-worker = false
14.         client-selector-thread-prefix = "NettyClientSelector"
15.         client-selector-thread-size = 1
16.         client-worker-thread-prefix = "NettyClientWorkerThread"
17.         # netty boss thread size,will not be used for UDT
18.         boss-thread-size = 1
19.         #auto default pin or 8
20.         worker-thread-size = 8
21.     }
22.     shutdown {
23.         # when destroy server, wait seconds
24.         wait = 3
25.     }
26.     serialization = "seata"
27.     compressor = "none"
28. }
29.
30. service {
31.
32.     vgroup_mapping.fsp_tx_group = "default" # 修改自定义事务组名称
33.
34.     default.grouplist = "127.0.0.1:8091"
35.     enableDegradе = false
36.     disable = false
37.     max.commit.retry.timeout = "-1"
38.     max.rollback.retry.timeout = "-1"
39.     disableGlobalTransaction = false
40. }
41.
42.
43. client {
44.     async.commit.buffer.limit = 10000
45.     lock {
46.         retry.internal = 10
47.         retry.times = 30
48.     }
49.     report.retry.count = 5
```

```
50.     tm.commit.retry.count = 1
51.     tm.rollback.retry.count = 1
52. }
53.
54. ## transaction log store
55. store {
56.     ## store mode: file、db
57.     mode = "db"
58.
59.     ## file store
60.     file {
61.         dir = "sessionStore"
62.
63.         # branch session size , if exceeded first try compress lockkey, still
        exceeded throws exceptions
64.         max-branch-session-size = 16384
65.         # globe session size , if exceeded throws exceptions
66.         max-global-session-size = 512
67.         # file buffer size , if exceeded allocate new buffer
68.         file-write-buffer-cache-size = 16384
69.         # when recover batch read size
70.         session.reload.read_size = 100
71.         # async, sync
72.         flush-disk-mode = async
73.     }
74.
75.     ## database store
76.     db {
77.         ## the implement of javax.sql.DataSource, such as
        DruidDataSource(druid)/BasicDataSource(dbcp) etc.
78.         datasource = "dbcp"
79.         ## mysql/oracle/h2/oceanbase etc.
80.         db-type = "mysql"
81.         driver-class-name = "com.mysql.jdbc.Driver"
82.         url = "jdbc:mysql://127.0.0.1:3306/seata"
83.         user = "root"
84.         password = "root"
85.         min-conn = 1
86.         max-conn = 3
87.         global.table = "global_table"
88.         branch.table = "branch_table"
89.         lock-table = "lock_table"
90.         query-limit = 100
91.     }
92. }
93. lock {
94.     ## the lock store mode: local、remote
95.     mode = "remote"
96.
```

```
97.     local {
98.         ## store locks in user's database
99.     }
100.
101.     remote {
102.         ## store locks in the seata's server
103.     }
104. }
105. recovery {
106.     #schedule committing retry period in milliseconds
107.     committing-retry-period = 1000
108.     #schedule asyn committing retry period in milliseconds
109.     asyn-committing-retry-period = 1000
110.     #schedule rollbacking retry period in milliseconds
111.     rollbacking-retry-period = 1000
112.     #schedule timeout retry period in milliseconds
113.     timeout-retry-period = 1000
114. }
115.
116. transaction {
117.     undo.data.validation = true
118.     undo.log.serialization = "jackson"
119.     undo.log.save.days = 7
120.     #schedule delete expired undo_log in milliseconds
121.     undo.log.delete.period = 86400000
122.     undo.log.table = "undo_log"
123. }
124.
125. ## metrics settings
126. metrics {
127.     enabled = false
128.     registry-type = "compact"
129.     # multi exporters use comma divided
130.     exporter-list = "prometheus"
131.     exporter-prometheus-port = 9898
132. }
133.
134. support {
135.     ## spring
136.     spring {
137.         # auto proxy the DataSource bean
138.         datasource.autoproxy = false
139.     }
140. }
```

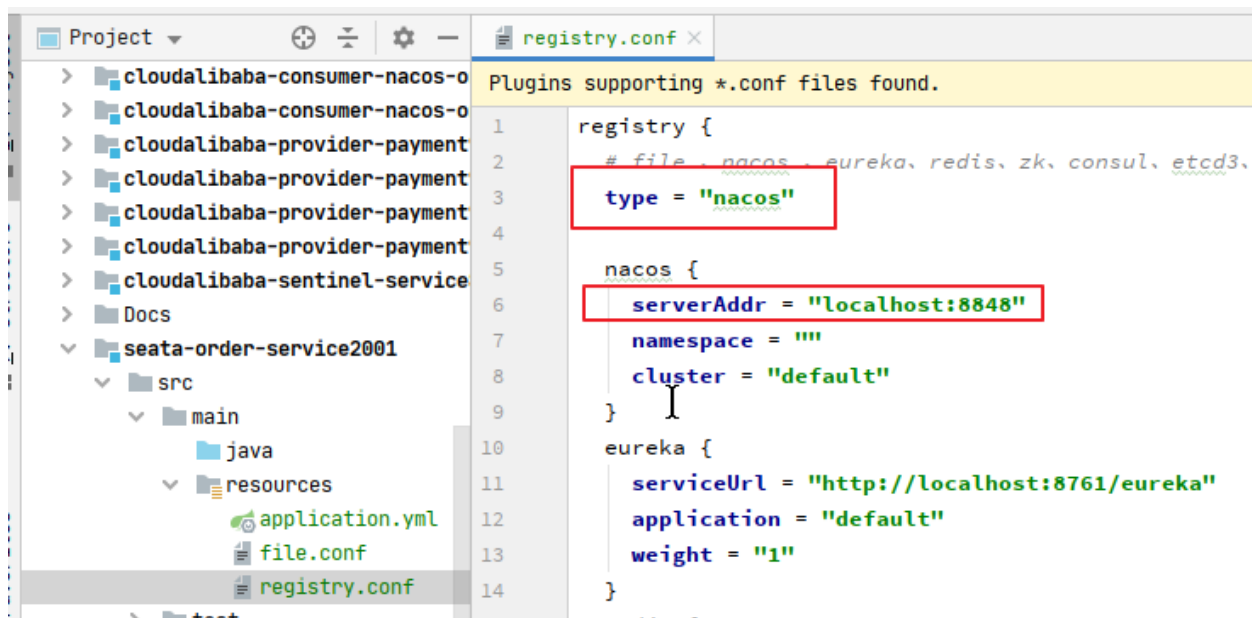
5.2.5. registry.conf

```
1. registry {
2.     # file 、nacos 、eureka、redis、zk、consul、etcd3、sofa
3.     type = "nacos"
4.
5.     nacos {
6.         serverAddr = "localhost:8848"
7.         namespace = ""
8.         cluster = "default"
9.     }
10.    eureka {
11.        serviceUrl = "http://localhost:8761/eureka"
12.        application = "default"
13.        weight = "1"
14.    }
15.    redis {
16.        serverAddr = "localhost:6379"
17.        db = "0"
18.    }
19.    zk {
20.        cluster = "default"
21.        serverAddr = "127.0.0.1:2181"
22.        session.timeout = 6000
23.        connect.timeout = 2000
24.    }
25.    consul {
26.        cluster = "default"
27.        serverAddr = "127.0.0.1:8500"
28.    }
29.    etcd3 {
30.        cluster = "default"
31.        serverAddr = "http://localhost:2379"
32.    }
33.    sofa {
34.        serverAddr = "127.0.0.1:9603"
35.        application = "default"
36.        region = "DEFAULT_ZONE"
37.        datacenter = "DefaultDataCenter"
38.        cluster = "default"
39.        group = "SEATA_GROUP"
40.        addressWaitTime = "3000"
41.    }
42.    file {
43.        name = "file.conf"
44.    }
45. }
46.
47. config {
48.     # file、nacos 、apollo、zk、consul、etcd3
49.     type = "file"
```

```

50.
51.   nacos {
52.       serverAddr = "localhost"
53.       namespace = ""
54.   }
55.   consul {
56.       serverAddr = "127.0.0.1:8500"
57.   }
58.   apollo {
59.       app.id = "seata-server"
60.       apollo.meta = "http://192.168.1.204:8801"
61.   }
62.   zk {
63.       serverAddr = "127.0.0.1:2181"
64.       session.timeout = 6000
65.       connect.timeout = 2000
66.   }
67.   etcd3 {
68.       serverAddr = "http://localhost:2379"
69.   }
70.   file {
71.       name = "file.conf"
72.   }
73. }

```



5.2.6. domain

5.2.6.1. CommonResult

```
1. package cn.sitedev.springcloud.alibaba.domain;
2.
3.
4. import lombok.AllArgsConstructor;
5. import lombok.Data;
6. import lombok.NoArgsConstructor;
7.
8. @Data
9. @AllArgsConstructor
10. @NoArgsConstructor
11. public class CommonResult<T> {
12.     private Integer code;
13.     private String message;
14.     private T data;
15.
16.     public CommonResult(Integer code, String message) {
17.         this(code, message, null);
18.     }
19. }
```

5.2.6.2. Order

```
1. package cn.sitedev.springcloud.alibaba.domain;
2.
3. import lombok.AllArgsConstructor;
4. import lombok.Data;
5. import lombok.NoArgsConstructor;
6.
7. import java.math.BigDecimal;
8.
9. @Data
10. @AllArgsConstructor
11. @NoArgsConstructor
12. public class Order {
13.     private Long id;
14.
15.     private Long userId;
16.
17.     private Long productId;
18.
19.     private Integer count;
20.
21.     private BigDecimal money;
22.
23.     private Integer status; //订单状态：0：创建中；1：已完结
24. }
```

5.2.7. Dao接口及实现

5.2.7.1. OrderDao

```
1. package cn.sitedev.springcloud.alibaba.dao;
2.
3. import cn.sitedev.springcloud.alibaba.domain.Order;
4. import org.apache.ibatis.annotations.Mapper;
5. import org.apache.ibatis.annotations.Param;
6.
7. @Mapper
8. public interface OrderDao {
9.     //新建订单
10.    void create(Order order);
11.
12.    //修改订单状态，从零改为1
13.    void update(@Param("userId") Long userId, @Param("status") Integer status);
14. }
```

5.2.7.2. OrderMapper.xml

resources文件夹下新建mapper文件夹后添加

```

1. <?xml version="1.0" encoding="UTF-8" ?>
2. <!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
   "http://mybatis.org/dtd/mybatis-3-mapper.dtd" >
3.
4. <mapper namespace="cn.sitedev.springcloud.alibaba.dao.OrderDao">
5.
6.     <resultMap id="BaseResultMap"
7. type="cn.sitedev.springcloud.alibaba.domain.Order">
8.         <id column="id" property="id" jdbcType="BIGINT"/>
9.         <result column="user_id" property="userId" jdbcType="BIGINT"/>
10.        <result column="product_id" property="productId" jdbcType="BIGINT"/>
11.        <result column="count" property="count" jdbcType="INTEGER"/>
12.        <result column="money" property="money" jdbcType="DECIMAL"/>
13.        <result column="status" property="status" jdbcType="INTEGER"/>
14.    </resultMap>
15.
16.    <insert id="create">
17.        insert into t_order (id,user_id,product_id,count,money,status)
18.        values (null,#{userId},#{productId},#{count},#{money},0);
19.    </insert>
20.
21.    <update id="update">
22.        update t_order set status = 1
23.        where user_id=#{userId} and status = #{status};
24.    </update>
25.
26. </mapper>

```

5.2.8. Service接口及实现

5.2.8.1. OrderService

```

1. package cn.sitedev.springcloud.alibaba.service;
2.
3. import cn.sitedev.springcloud.alibaba.domain.Order;
4.
5. public interface OrderService {
6.     void create(Order order);
7. }

```

5.2.8.2. OrderServiceImpl

```
1. package cn.sitedev.springcloud.alibaba.service.impl;
2.
3. import cn.sitedev.springcloud.alibaba.dao.OrderDao;
4. import cn.sitedev.springcloud.alibaba.domain.Order;
5. import cn.sitedev.springcloud.alibaba.service.AccountService;
6. import cn.sitedev.springcloud.alibaba.service.OrderService;
7. import cn.sitedev.springcloud.alibaba.service.StorageService;
8. import io.seata.spring.annotation.GlobalTransactional;
9. import lombok.extern.slf4j.Slf4j;
10. import org.springframework.stereotype.Service;
11.
12. import javax.annotation.Resource;
13.
14. @Service
15. @Slf4j
16. public class OrderServiceImpl implements OrderService {
17.     @Resource
18.     private OrderDao orderDao;
19.     @Resource
20.     private StorageService storageService;
21.     @Resource
22.     private AccountService accountService;
23.
24.     /**
25.      * 创建订单->调用库存服务扣减库存->调用账户服务扣减账户余额->修改订单状态
26.      */
27.
28.     @Override
29.     @GlobalTransactional(name = "fsp-create-order", rollbackFor =
Exception.class)
30.     public void create(Order order) {
31.         log.info("----->开始新建订单");
32.         //新建订单
33.         orderDao.create(order);
34.
35.         //扣减库存
36.         log.info("----->订单微服务开始调用库存，做扣减Count");
37.         storageService.decrease(order.getProductId(), order.getCount());
38.         log.info("----->订单微服务开始调用库存，做扣减end");
39.
40.         //扣减账户
41.         log.info("----->订单微服务开始调用账户，做扣减Money");
42.         accountService.decrease(order.getUserId(), order.getMoney());
43.         log.info("----->订单微服务开始调用账户，做扣减end");
44.
45.
46.         //修改订单状态，从零到1代表已经完成
47.         log.info("----->修改订单状态开始");
48.         orderDao.update(order.getUserId(), 0);
```

```

49.         log.info("----->修改订单状态结束");
50.
51.         log.info("----->下订单结束了");
52.
53.     }
54. }

```

5.2.8.3. StorageService

```

1. package cn.sitedev.springcloud.alibaba.service;
2.
3. import cn.sitedev.springcloud.alibaba.domain.CommonResult;
4. import org.springframework.cloud.openfeign.FeignClient;
5. import org.springframework.web.bind.annotation.PostMapping;
6. import org.springframework.web.bind.annotation.RequestParam;
7.
8.
9. @FeignClient(value = "seata-storage-service")
10. public interface StorageService {
11.     @PostMapping(value = "/storage/decrease")
12.     CommonResult decrease(@RequestParam("productId") Long productId,
13.         @RequestParam("count") Integer count);
14. }

```

5.2.8.4. AccountService

```

1. package cn.sitedev.springcloud.alibaba.service;
2.
3. import cn.sitedev.springcloud.alibaba.domain.CommonResult;
4. import org.springframework.cloud.openfeign.FeignClient;
5. import org.springframework.web.bind.annotation.PostMapping;
6. import org.springframework.web.bind.annotation.RequestParam;
7.
8. import java.math.BigDecimal;
9.
10. @FeignClient(value = "seata-account-service")
11. public interface AccountService {
12.     @PostMapping(value = "/account/decrease")
13.     CommonResult decrease(@RequestParam("userId") Long userId,
14.         @RequestParam("money") BigDecimal money);
15. }

```

5.2.9. Controller


```
1. package cn.sitedev.springcloud.alibaba.controller;
2.
3. import cn.sitedev.springcloud.alibaba.domain.CommonResult;
4. import cn.sitedev.springcloud.alibaba.domain.Order;
5. import cn.sitedev.springcloud.alibaba.service.OrderService;
6. import org.springframework.web.bind.annotation.GetMapping;
7. import org.springframework.web.bind.annotation.RestController;
8.
9. import javax.annotation.Resource;
10.
11. @RestController
12. public class OrderController {
13.     @Resource
14.     private OrderService orderService;
15.
16.     @GetMapping("/order/create")
17.     public CommonResult create(Order order) {
18.         orderService.create(order);
19.         return new CommonResult(200, "订单创建成功");
20.     }
21. }
```

5.2.10. config配置

5.2.10.1. MyBatisConfig

```
1. package cn.sitedev.springcloud.alibaba.config;
2.
3. import org.mybatis.spring.annotation.MapperScan;
4. import org.springframework.context.annotation.Configuration;
5.
6. @Configuration
7. @MapperScan({"cn.sitedev.springcloud.alibaba.dao"})
8. public class MyBatisConfig {
9.
10. }
```

5.2.10.2. DataSourceProxyConfig

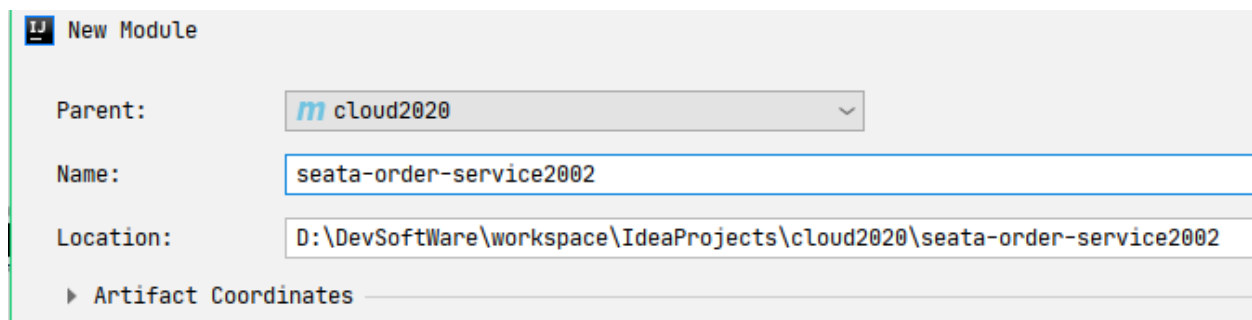
```
1. package cn.sitedev.springcloud.alibaba.config;
2.
3. import com.alibaba.druid.pool.DruidDataSource;
4. import io.seata.rm.datasource.DataSourceProxy;
5. import org.apache.ibatis.session.SqlSessionFactory;
6. import org.mybatis.spring.SqlSessionFactoryBean;
7. import org.mybatis.spring.transaction.SpringManagedTransactionFactory;
8. import org.springframework.beans.factory.annotation.Value;
9. import org.springframework.boot.context.properties.ConfigurationProperties;
10. import org.springframework.context.annotation.Bean;
11. import org.springframework.context.annotation.Configuration;
12. import org.springframework.core.io.support.PathMatchingResourcePatternResolver;
13.
14. import javax.sql.DataSource;
15.
16.
17. @Configuration
18. public class DataSourceProxyConfig {
19.
20.     @Value("${mybatis.mapperLocations}")
21.     private String mapperLocations;
22.
23.     @Bean
24.     @ConfigurationProperties(prefix = "spring.datasource")
25.     public DataSource druidDataSource() {
26.         return new DruidDataSource();
27.     }
28.
29.     @Bean
30.     public DataSourceProxy dataSourceProxy(DataSource dataSource) {
31.         return new DataSourceProxy(dataSource);
32.     }
33.
34.     @Bean
35.     public SqlSessionFactory sqlSessionFactoryBean(DataSourceProxy
dataSourceProxy) throws Exception {
36.         SqlSessionFactoryBean sqlSessionFactoryBean = new
SqlSessionFactoryBean();
37.         sqlSessionFactoryBean.setDataSource(dataSourceProxy);
38.         sqlSessionFactoryBean.setMapperLocations(new
PathMatchingResourcePatternResolver().getResources(mapperLocations));
39.         sqlSessionFactoryBean.setTransactionFactory(new
SpringManagedTransactionFactory());
40.         return sqlSessionFactoryBean.getObject();
41.     }
42.
43. }
```

5.2.11. 主启动

```
1. package cn.sitedev.springcloud.alibaba;
2.
3. import org.springframework.boot.SpringApplication;
4. import org.springframework.boot.autoconfigure.SpringBootApplication;
5. import org.springframework.boot.autoconfigure.jdbc.DataSourceAutoConfiguration;
6. import org.springframework.cloud.client.discovery.EnableDiscoveryClient;
7. import org.springframework.cloud.openfeign.EnableFeignClients;
8.
9. @EnableDiscoveryClient
10. @EnableFeignClients
11. @SpringBootApplication(exclude = DataSourceAutoConfiguration.class)//取消数据源
    自动创建的配置
12. public class SeataOrderMainApp2001 {
13.
14.     public static void main(String[] args) {
15.         SpringApplication.run(SeataOrderMainApp2001.class, args);
16.     }
17. }
```

5.3. 新建库存Storage-Module

5.3.1. 新建seata-order-service2002



5.3.2. POM

```
1. <?xml version="1.0" encoding="UTF-8"?>
2. <project xmlns="http://maven.apache.org/POM/4.0.0"
3.         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4.         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
5.     <parent>
6.         <artifactId>cloud2020</artifactId>
7.         <groupId>cn.sitedev.springcloud</groupId>
8.         <version>1.0-SNAPSHOT</version>
9.     </parent>
10.    <modelVersion>4.0.0</modelVersion>
11.
12.    <artifactId>seata-order-service2002</artifactId>
13.
14.    <dependencies>
15.        <!--nacos-->
16.        <dependency>
17.            <groupId>com.alibaba.cloud</groupId>
18.            <artifactId>spring-cloud-starter-alibaba-nacos-
discovery</artifactId>
19.        </dependency>
20.        <!--seata-->
21.        <dependency>
22.            <groupId>com.alibaba.cloud</groupId>
23.            <artifactId>spring-cloud-starter-alibaba-seata</artifactId>
24.            <exclusions>
25.                <exclusion>
26.                    <artifactId>seata-all</artifactId>
27.                    <groupId>io.seata</groupId>
28.                </exclusion>
29.            </exclusions>
30.        </dependency>
31.        <dependency>
32.            <groupId>io.seata</groupId>
33.            <artifactId>seata-all</artifactId>
34.            <version>0.9.0</version>
35.        </dependency>
36.        <!--feign-->
37.        <dependency>
38.            <groupId>org.springframework.cloud</groupId>
39.            <artifactId>spring-cloud-starter-openfeign</artifactId>
40.        </dependency>
41.        <dependency>
42.            <groupId>org.springframework.boot</groupId>
43.            <artifactId>spring-boot-starter-web</artifactId>
44.        </dependency>
45.        <dependency>
46.            <groupId>org.springframework.boot</groupId>
47.            <artifactId>spring-boot-starter-test</artifactId>
```

```
48.         <scope>test</scope>
49.     </dependency>
50.     <dependency>
51.         <groupId>org.mybatis.spring.boot</groupId>
52.         <artifactId>mybatis-spring-boot-starter</artifactId>
53.         <version>2.0.0</version>
54.     </dependency>
55.     <dependency>
56.         <groupId>mysql</groupId>
57.         <artifactId>mysql-connector-java</artifactId>
58.         <version>5.1.37</version>
59.     </dependency>
60.     <dependency>
61.         <groupId>com.alibaba</groupId>
62.         <artifactId>druid-spring-boot-starter</artifactId>
63.         <version>1.1.10</version>
64.     </dependency>
65.     <dependency>
66.         <groupId>org.projectlombok</groupId>
67.         <artifactId>lombok</artifactId>
68.         <optional>true</optional>
69.     </dependency>
70. </dependencies>
71.
72. </project>
```

5.3.3. YML

```
1.  server:
2.    port: 2002
3.
4.  spring:
5.    application:
6.      name: seata-storage-service
7.    cloud:
8.      alibaba:
9.        seata:
10.          tx-service-group: fsp_tx_group
11.      nacos:
12.        discovery:
13.          server-addr: localhost:8848
14.    datasource:
15.      driver-class-name: com.mysql.jdbc.Driver
16.      url: jdbc:mysql://localhost:3306/seata_storage
17.      username: root
18.      password: root
19.
20.  logging:
21.    level:
22.      io:
23.        seata: info
24.
25.  mybatis:
26.    mapperLocations: classpath:mapper/*.xml
```

5.3.4. file.conf

```
1.  transport {
2.      # tcp udt unix-domain-socket
3.      type = "TCP"
4.      #NIO NATIVE
5.      server = "NIO"
6.      #enable heartbeat
7.      heartbeat = true
8.      #thread factory for netty
9.      thread-factory {
10.         boss-thread-prefix = "NettyBoss"
11.         worker-thread-prefix = "NettyServerNIOWorker"
12.         server-executor-thread-prefix = "NettyServerBizHandler"
13.         share-boss-worker = false
14.         client-selector-thread-prefix = "NettyClientSelector"
15.         client-selector-thread-size = 1
16.         client-worker-thread-prefix = "NettyClientWorkerThread"
17.         # netty boss thread size,will not be used for UDT
18.         boss-thread-size = 1
19.         #auto default pin or 8
20.         worker-thread-size = 8
21.     }
22.     shutdown {
23.         # when destroy server, wait seconds
24.         wait = 3
25.     }
26.     serialization = "seata"
27.     compressor = "none"
28. }
29.
30. service {
31.     #vgroup->rgroup
32.     vgroup_mapping.fsp_tx_group = "default"
33.     #only support single node
34.     default.grouplist = "127.0.0.1:8091"
35.     #degrade current not support
36.     enableDegrade = false
37.     #disable
38.     disable = false
39.     #unit ms,s,m,h,d represents milliseconds, seconds, minutes, hours, days,
    default permanent
40.     max.commit.retry.timeout = "-1"
41.     max.rollback.retry.timeout = "-1"
42.     disableGlobalTransaction = false
43. }
44.
45. client {
46.     async.commit.buffer.limit = 10000
47.     lock {
48.         retry.internal = 10
```

```
49.     retry.times = 30
50. }
51. report.retry.count = 5
52. tm.commit.retry.count = 1
53. tm.rollback.retry.count = 1
54. }
55.
56. transaction {
57.     undo.data.validation = true
58.     undo.log.serialization = "jackson"
59.     undo.log.save.days = 7
60.     #schedule delete expired undo_log in milliseconds
61.     undo.log.delete.period = 86400000
62.     undo.log.table = "undo_log"
63. }
64.
65. support {
66.     ## spring
67.     spring {
68.         # auto proxy the DataSource bean
69.         datasource.autoproxy = false
70.     }
71. }
```

5.3.5. registry.conf


```
1. registry {
2.     # file 、nacos 、eureka、redis、zk
3.     type = "nacos"
4.
5.     nacos {
6.         serverAddr = "localhost:8848"
7.         namespace = ""
8.         cluster = "default"
9.     }
10.    eureka {
11.        serviceUrl = "http://localhost:8761/eureka"
12.        application = "default"
13.        weight = "1"
14.    }
15.    redis {
16.        serverAddr = "localhost:6381"
17.        db = "0"
18.    }
19.    zk {
20.        cluster = "default"
21.        serverAddr = "127.0.0.1:2181"
22.        session.timeout = 6000
23.        connect.timeout = 2000
24.    }
25.    file {
26.        name = "file.conf"
27.    }
28. }
29.
30. config {
31.     # file、nacos 、apollo、zk
32.     type = "file"
33.
34.     nacos {
35.         serverAddr = "localhost"
36.         namespace = ""
37.         cluster = "default"
38.     }
39.     apollo {
40.         app.id = "fescar-server"
41.         apollo.meta = "http://192.168.1.204:8801"
42.     }
43.     zk {
44.         serverAddr = "127.0.0.1:2181"
45.         session.timeout = 6000
46.         connect.timeout = 2000
47.     }
48.     file {
49.         name = "file.conf"
```

```
50.     }  
51. }
```

5.3.6. domain

5.3.6.1. CommonResult

```
1. package cn.sitedev.springcloud.alibaba.domain;  
2.  
3.  
4. import lombok.AllArgsConstructor;  
5. import lombok.Data;  
6. import lombok.NoArgsConstructor;  
7.  
8. @Data  
9. @AllArgsConstructor  
10. @NoArgsConstructor  
11. public class CommonResult<T> {  
12.     private Integer code;  
13.     private String message;  
14.     private T data;  
15.  
16.     public CommonResult(Integer code, String message) {  
17.         this(code, message, null);  
18.     }  
19. }
```

5.3.6.2. Storage

```
1. package cn.sitedev.springcloud.alibaba.domain;
2.
3. import lombok.Data;
4.
5. @Data
6. public class Storage {
7.
8.     private Long id;
9.
10.    // 产品id
11.    private Long productId;
12.
13.    //总库存
14.    private Integer total;
15.
16.    //已用库存
17.    private Integer used;
18.
19.    //剩余库存
20.    private Integer residue;
21. }
```

5.3.7. Dao接口及实现

5.3.7.1. StorageDao

```
1. package cn.sitedev.springcloud.alibaba.dao;
2.
3. import org.apache.ibatis.annotations.Mapper;
4. import org.apache.ibatis.annotations.Param;
5.
6. @Mapper
7. public interface StorageDao {
8.    //扣减库存信息
9.    void decrease(@Param("productId") Long productId, @Param("count") Integer
count);
10. }
```

5.3.7.2. StorageMapper.xml

resources文件夹下新建mapper文件夹后添加

```

1. <?xml version="1.0" encoding="UTF-8" ?>
2. <!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
   "http://mybatis.org/dtd/mybatis-3-mapper.dtd" >
3.
4.
5. <mapper namespace="cn.sitedev.springcloud.alibaba.dao.StorageDao">
6.
7.     <resultMap id="BaseResultMap"
   type="cn.sitedev.springcloud.alibaba.domain.Storage">
8.         <id column="id" property="id" jdbcType="BIGINT"/>
9.         <result column="product_id" property="productId" jdbcType="BIGINT"/>
10.        <result column="total" property="total" jdbcType="INTEGER"/>
11.        <result column="used" property="used" jdbcType="INTEGER"/>
12.        <result column="residue" property="residue" jdbcType="INTEGER"/>
13.    </resultMap>
14.
15.    <update id="decrease">
16.        UPDATE
17.            t_storage
18.        SET
19.            used = used + #{count}, residue = residue - #{count}
20.        WHERE
21.            product_id = #{productId}
22.    </update>
23.
24. </mapper>

```

5.3.8. Service接口及实现

5.3.8.1. StorageService

```

1. package cn.sitedev.springcloud.alibaba.service;
2.
3. public interface StorageService {
4.     // 扣减库存
5.     void decrease(Long productId, Integer count);
6. }

```

5.3.8.2. StorageServiceImpl

```
1. package cn.sitedev.springcloud.alibaba.service.impl;
2.
3. import cn.sitedev.springcloud.alibaba.dao.StorageDao;
4. import cn.sitedev.springcloud.alibaba.service.StorageService;
5. import org.slf4j.Logger;
6. import org.slf4j.LoggerFactory;
7. import org.springframework.stereotype.Service;
8.
9. import javax.annotation.Resource;
10.
11. @Service
12. public class StorageServiceImpl implements StorageService {
13.
14.     private static final Logger LOGGER =
15.         LoggerFactory.getLogger(StorageServiceImpl.class);
16.
17.     @Resource
18.     private StorageDao storageDao;
19.
20.     // 扣减库存
21.     @Override
22.     public void decrease(Long productId, Integer count) {
23.         LOGGER.info("----->storage-service中扣减库存开始");
24.         storageDao.decrease(productId, count);
25.         LOGGER.info("----->storage-service中扣减库存结束");
26.     }
27. }
```

5.3.9. Controller

```
1. package cn.sitedev.springcloud.alibaba.controller;
2.
3. import cn.sitedev.springcloud.alibaba.domain.CommonResult;
4. import cn.sitedev.springcloud.alibaba.service.StorageService;
5. import org.springframework.beans.factory.annotation.Autowired;
6. import org.springframework.web.bind.annotation.RequestMapping;
7. import org.springframework.web.bind.annotation.RestController;
8.
9. @RestController
10. public class StorageController {
11.
12.     @Autowired
13.     private StorageService storageService;
14.
15.     //扣减库存
16.     @RequestMapping("/storage/decrease")
17.     public CommonResult decrease(Long productId, Integer count) {
18.         storageService.decrease(productId, count);
19.         return new CommonResult(200, "扣减库存成功!");
20.     }
21. }
```

5.3.10. Config配置

5.3.10.1. MyBatisConfig

```
1. package cn.sitedev.springcloud.alibaba.config;
2.
3. import org.mybatis.spring.annotation.MapperScan;
4. import org.springframework.context.annotation.Configuration;
5.
6. @Configuration
7. @MapperScan({"cn.sitedev.springcloud.alibaba.dao"})
8. public class MyBatisConfig {
9.
10. }
```

5.3.10.2. DataSourceProxyConfig

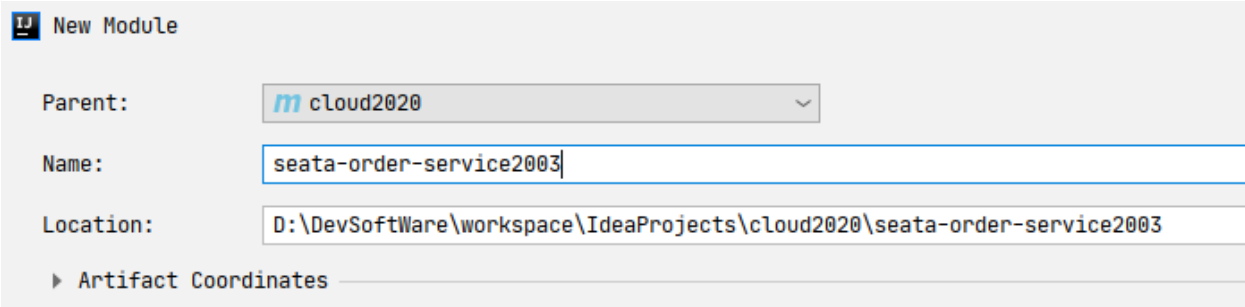
```
1. package cn.sitedev.springcloud.alibaba.config;
2.
3. import com.alibaba.druid.pool.DruidDataSource;
4. import io.seata.rm.datasource.DataSourceProxy;
5. import org.apache.ibatis.session.SqlSessionFactory;
6. import org.mybatis.spring.SqlSessionFactoryBean;
7. import org.mybatis.spring.transaction.SpringManagedTransactionFactory;
8. import org.springframework.beans.factory.annotation.Value;
9. import org.springframework.boot.context.properties.ConfigurationProperties;
10. import org.springframework.context.annotation.Bean;
11. import org.springframework.context.annotation.Configuration;
12. import org.springframework.core.io.support.PathMatchingResourcePatternResolver;
13.
14. import javax.sql.DataSource;
15.
16.
17. @Configuration
18. public class DataSourceProxyConfig {
19.
20.     @Value("${mybatis.mapperLocations}")
21.     private String mapperLocations;
22.
23.     @Bean
24.     @ConfigurationProperties(prefix = "spring.datasource")
25.     public DataSource druidDataSource() {
26.         return new DruidDataSource();
27.     }
28.
29.     @Bean
30.     public DataSourceProxy dataSourceProxy(DataSource dataSource) {
31.         return new DataSourceProxy(dataSource);
32.     }
33.
34.     @Bean
35.     public SqlSessionFactory sqlSessionFactoryBean(DataSourceProxy
dataSourceProxy) throws Exception {
36.         SqlSessionFactoryBean sqlSessionFactoryBean = new
SqlSessionFactoryBean();
37.         sqlSessionFactoryBean.setDataSource(dataSourceProxy);
38.         sqlSessionFactoryBean.setMapperLocations(new
PathMatchingResourcePatternResolver().getResources(mapperLocations));
39.         sqlSessionFactoryBean.setTransactionFactory(new
SpringManagedTransactionFactory());
40.         return sqlSessionFactoryBean.getObject();
41.     }
42.
43. }
```

5.3.11. 主启动

```
1. package cn.sitedev.springcloud.alibaba;
2.
3. import org.springframework.boot.SpringApplication;
4. import org.springframework.boot.autoconfigure.SpringBootApplication;
5. import org.springframework.boot.autoconfigure.jdbc.DataSourceAutoConfiguration;
6. import org.springframework.cloud.client.discovery.EnableDiscoveryClient;
7. import org.springframework.cloud.openfeign.EnableFeignClients;
8.
9. @SpringBootApplication(exclude = DataSourceAutoConfiguration.class)
10. @EnableDiscoveryClient
11. @EnableFeignClients
12. public class SeataStorageServiceApplication2002 {
13.     public static void main(String[] args) {
14.         SpringApplication.run(SeataStorageServiceApplication2002.class, args);
15.     }
16. }
```

5.4. 新建账户Account-Module

5.4.1. 新建seata-order-service2003



New Module

Parent: m cloud2020

Name: seata-order-service2003

Location: D:\DevSoftWare\workspace\IdeaProjects\cloud2020\seata-order-service2003

▶ Artifact Coordinates

5.4.2. POM


```
1. <?xml version="1.0" encoding="UTF-8"?>
2. <project xmlns="http://maven.apache.org/POM/4.0.0"
3.         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4.         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
5.     <parent>
6.         <artifactId>cloud2020</artifactId>
7.         <groupId>cn.sitedev.springcloud</groupId>
8.         <version>1.0-SNAPSHOT</version>
9.     </parent>
10.    <modelVersion>4.0.0</modelVersion>
11.
12.    <artifactId>seata-order-service2003</artifactId>
13.
14.    <dependencies>
15.        <!--nacos-->
16.        <dependency>
17.            <groupId>com.alibaba.cloud</groupId>
18.            <artifactId>spring-cloud-starter-alibaba-nacos-
discovery</artifactId>
19.        </dependency>
20.        <!--seata-->
21.        <dependency>
22.            <groupId>com.alibaba.cloud</groupId>
23.            <artifactId>spring-cloud-starter-alibaba-seata</artifactId>
24.            <exclusions>
25.                <exclusion>
26.                    <artifactId>seata-all</artifactId>
27.                    <groupId>io.seata</groupId>
28.                </exclusion>
29.            </exclusions>
30.        </dependency>
31.        <dependency>
32.            <groupId>io.seata</groupId>
33.            <artifactId>seata-all</artifactId>
34.            <version>0.9.0</version>
35.        </dependency>
36.        <!--feign-->
37.        <dependency>
38.            <groupId>org.springframework.cloud</groupId>
39.            <artifactId>spring-cloud-starter-openfeign</artifactId>
40.        </dependency>
41.        <dependency>
42.            <groupId>org.springframework.boot</groupId>
43.            <artifactId>spring-boot-starter-web</artifactId>
44.        </dependency>
45.        <dependency>
46.            <groupId>org.springframework.boot</groupId>
47.            <artifactId>spring-boot-starter-test</artifactId>
```

```
48.         <scope>test</scope>
49.     </dependency>
50. <dependency>
51.     <groupId>org.mybatis.spring.boot</groupId>
52.     <artifactId>mybatis-spring-boot-starter</artifactId>
53.     <version>2.0.0</version>
54. </dependency>
55. <dependency>
56.     <groupId>mysql</groupId>
57.     <artifactId>mysql-connector-java</artifactId>
58.     <version>5.1.37</version>
59. </dependency>
60. <dependency>
61.     <groupId>com.alibaba</groupId>
62.     <artifactId>druid-spring-boot-starter</artifactId>
63.     <version>1.1.10</version>
64. </dependency>
65. <dependency>
66.     <groupId>org.projectlombok</groupId>
67.     <artifactId>lombok</artifactId>
68.     <optional>true</optional>
69. </dependency>
70. </dependencies>
71.
72. </project>
```

5.4.3. YML

```
1.  server:
2.    port: 2003
3.
4.  spring:
5.    application:
6.      name: seata-account-service
7.    cloud:
8.      alibaba:
9.        seata:
10.          tx-service-group: fsp_tx_group
11.      nacos:
12.        discovery:
13.          server-addr: localhost:8848
14.    datasource:
15.      driver-class-name: com.mysql.jdbc.Driver
16.      url: jdbc:mysql://localhost:3306/seata_account
17.      username: root
18.      password: root
19.
20.  feign:
21.    hystrix:
22.      enabled: false
23.
24.  logging:
25.    level:
26.      io:
27.        seata: info
28.
29.  mybatis:
30.    mapperLocations: classpath:mapper/*.xml
```

5.4.4. file.conf

```
1.  transport {
2.      # tcp udt unix-domain-socket
3.      type = "TCP"
4.      #NIO NATIVE
5.      server = "NIO"
6.      #enable heartbeat
7.      heartbeat = true
8.      #thread factory for netty
9.      thread-factory {
10.         boss-thread-prefix = "NettyBoss"
11.         worker-thread-prefix = "NettyServerNIOWorker"
12.         server-executor-thread-prefix = "NettyServerBizHandler"
13.         share-boss-worker = false
14.         client-selector-thread-prefix = "NettyClientSelector"
15.         client-selector-thread-size = 1
16.         client-worker-thread-prefix = "NettyClientWorkerThread"
17.         # netty boss thread size,will not be used for UDT
18.         boss-thread-size = 1
19.         #auto default pin or 8
20.         worker-thread-size = 8
21.     }
22.     shutdown {
23.         # when destroy server, wait seconds
24.         wait = 3
25.     }
26.     serialization = "seata"
27.     compressor = "none"
28. }
29.
30. service {
31.
32.     vgroup_mapping.fsp_tx_group = "default" #修改自定义事务组名称
33.
34.     default.grouplist = "127.0.0.1:8091"
35.     enableDegradate = false
36.     disable = false
37.     max.commit.retry.timeout = "-1"
38.     max.rollback.retry.timeout = "-1"
39.     disableGlobalTransaction = false
40. }
41.
42.
43. client {
44.     async.commit.buffer.limit = 10000
45.     lock {
46.         retry.internal = 10
47.         retry.times = 30
48.     }
49.     report.retry.count = 5
```

```
50.     tm.commit.retry.count = 1
51.     tm.rollback.retry.count = 1
52. }
53.
54. ## transaction log store
55. store {
56.     ## store mode: file、db
57.     mode = "db"
58.
59.     ## file store
60.     file {
61.         dir = "sessionStore"
62.
63.         # branch session size , if exceeded first try compress lockkey, still
        exceeded throws exceptions
64.         max-branch-session-size = 16384
65.         # globe session size , if exceeded throws exceptions
66.         max-global-session-size = 512
67.         # file buffer size , if exceeded allocate new buffer
68.         file-write-buffer-cache-size = 16384
69.         # when recover batch read size
70.         session.reload.read_size = 100
71.         # async, sync
72.         flush-disk-mode = async
73.     }
74.
75.     ## database store
76.     db {
77.         ## the implement of javax.sql.DataSource, such as
        DruidDataSource(druid)/BasicDataSource(dbcp) etc.
78.         datasource = "dbcp"
79.         ## mysql/oracle/h2/oceanbase etc.
80.         db-type = "mysql"
81.         driver-class-name = "com.mysql.jdbc.Driver"
82.         url = "jdbc:mysql://127.0.0.1:3306/seata"
83.         user = "root"
84.         password = "root"
85.         min-conn = 1
86.         max-conn = 3
87.         global.table = "global_table"
88.         branch.table = "branch_table"
89.         lock-table = "lock_table"
90.         query-limit = 100
91.     }
92. }
93. lock {
94.     ## the lock store mode: local、remote
95.     mode = "remote"
96.
```

```

97.     local {
98.         ## store locks in user's database
99.     }
100.
101.     remote {
102.         ## store locks in the seata's server
103.     }
104. }
105. recovery {
106.     #schedule committing retry period in milliseconds
107.     committing-retry-period = 1000
108.     #schedule asyn committing retry period in milliseconds
109.     asyn-committing-retry-period = 1000
110.     #schedule rollbacking retry period in milliseconds
111.     rollbacking-retry-period = 1000
112.     #schedule timeout retry period in milliseconds
113.     timeout-retry-period = 1000
114. }
115.
116. transaction {
117.     undo.data.validation = true
118.     undo.log.serialization = "jackson"
119.     undo.log.save.days = 7
120.     #schedule delete expired undo_log in milliseconds
121.     undo.log.delete.period = 86400000
122.     undo.log.table = "undo_log"
123. }
124.
125. ## metrics settings
126. metrics {
127.     enabled = false
128.     registry-type = "compact"
129.     # multi exporters use comma divided
130.     exporter-list = "prometheus"
131.     exporter-prometheus-port = 9898
132. }
133.
134. support {
135.     ## spring
136.     spring {
137.         # auto proxy the DataSource bean
138.         datasource.autoproxy = false
139.     }
140. }

```

5.4.5. registry.conf

```
1. registry {
2.     # file 、nacos 、eureka、redis、zk、consul、etcd3、sofa
3.     type = "nacos"
4.
5.     nacos {
6.         serverAddr = "localhost:8848"
7.         namespace = ""
8.         cluster = "default"
9.     }
10.    eureka {
11.        serviceUrl = "http://localhost:8761/eureka"
12.        application = "default"
13.        weight = "1"
14.    }
15.    redis {
16.        serverAddr = "localhost:6379"
17.        db = "0"
18.    }
19.    zk {
20.        cluster = "default"
21.        serverAddr = "127.0.0.1:2181"
22.        session.timeout = 6000
23.        connect.timeout = 2000
24.    }
25.    consul {
26.        cluster = "default"
27.        serverAddr = "127.0.0.1:8500"
28.    }
29.    etcd3 {
30.        cluster = "default"
31.        serverAddr = "http://localhost:2379"
32.    }
33.    sofa {
34.        serverAddr = "127.0.0.1:9603"
35.        application = "default"
36.        region = "DEFAULT_ZONE"
37.        datacenter = "DefaultDataCenter"
38.        cluster = "default"
39.        group = "SEATA_GROUP"
40.        addressWaitTime = "3000"
41.    }
42.    file {
43.        name = "file.conf"
44.    }
45. }
46.
47. config {
48.     # file、nacos 、apollo、zk、consul、etcd3
49.     type = "file"
```

```
50.
51.   nacos {
52.     serverAddr = "localhost"
53.     namespace = ""
54.   }
55.   consul {
56.     serverAddr = "127.0.0.1:8500"
57.   }
58.   apollo {
59.     app.id = "seata-server"
60.     apollo.meta = "http://192.168.1.204:8801"
61.   }
62.   zk {
63.     serverAddr = "127.0.0.1:2181"
64.     session.timeout = 6000
65.     connect.timeout = 2000
66.   }
67.   etcd3 {
68.     serverAddr = "http://localhost:2379"
69.   }
70.   file {
71.     name = "file.conf"
72.   }
73. }
```

5.4.6. domain

5.4.6.1. CommonResult


```
1. package cn.sitedev.springcloud.alibaba.domain;
2.
3.
4. import lombok.AllArgsConstructor;
5. import lombok.Data;
6. import lombok.NoArgsConstructor;
7.
8. @Data
9. @AllArgsConstructor
10. @NoArgsConstructor
11. public class CommonResult<T> {
12.     private Integer code;
13.     private String message;
14.     private T data;
15.
16.     public CommonResult(Integer code, String message) {
17.         this(code, message, null);
18.     }
19. }
```

5.4.6.2. Account

```
1. package cn.sitedev.springcloud.alibaba.domain;
2.
3. import lombok.AllArgsConstructor;
4. import lombok.Data;
5. import lombok.NoArgsConstructor;
6.
7. import java.math.BigDecimal;
8.
9. @Data
10. @AllArgsConstructor
11. @NoArgsConstructor
12. public class Account {
13.
14.     private Long id;
15.
16.     /**
17.      * 用户id
18.      */
19.     private Long userId;
20.
21.     /**
22.      * 总额度
23.      */
24.     private BigDecimal total;
25.
26.     /**
27.      * 已用额度
28.      */
29.     private BigDecimal used;
30.
31.     /**
32.      * 剩余额度
33.      */
34.     private BigDecimal residue;
35. }
```

5.4.7. Dao接口及实现

5.4.7.1. AccountDao

```

1. package cn.sitedev.springcloud.alibaba.dao;
2.
3. import org.apache.ibatis.annotations.Mapper;
4. import org.apache.ibatis.annotations.Param;
5.
6. import java.math.BigDecimal;
7.
8. @Mapper
9. public interface AccountDao {
10.
11.     /**
12.      * 扣减账户余额
13.      */
14.     void decrease(@Param("userId") Long userId, @Param("money") BigDecimal
money);
15. }

```

5.4.7.2. AccountMapper.xml

resources文件夹下新建mapper文件夹后添加

```

1. <?xml version="1.0" encoding="UTF-8" ?>
2. <!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
"http://mybatis.org/dtd/mybatis-3-mapper.dtd" >
3.
4. <mapper namespace="cn.sitedev.springcloud.alibaba.dao.AccountDao">
5.
6.     <resultMap id="BaseResultMap"
type="cn.sitedev.springcloud.alibaba.domain.Account">
7.         <id column="id" property="id" jdbcType="BIGINT"/>
8.         <result column="user_id" property="userId" jdbcType="BIGINT"/>
9.         <result column="total" property="total" jdbcType="DECIMAL"/>
10.        <result column="used" property="used" jdbcType="DECIMAL"/>
11.        <result column="residue" property="residue" jdbcType="DECIMAL"/>
12.    </resultMap>
13.
14.    <update id="decrease">
15.        UPDATE t_account
16.        SET
17.            residue = residue - #{money},used = used + #{money}
18.        WHERE
19.            user_id = #{userId};
20.    </update>
21.
22. </mapper>

```

5.4.8. Service接口及实现

5.4.8.1. AccountService

```
1. package cn.sitedev.springcloud.alibaba.service;
2.
3. import org.springframework.web.bind.annotation.RequestParam;
4.
5. import java.math.BigDecimal;
6.
7. public interface AccountService {
8.
9.     /**
10.      * 扣减账户余额
11.      */
12.     void decrease(@RequestParam("userId") Long userId, @RequestParam("money")
13.         BigDecimal money);
14. }
```

5.4.8.2. AccountServiceImpl

```
1. package cn.sitedev.springcloud.alibaba.service;
2.
3. import cn.sitedev.springcloud.alibaba.dao.AccountDao;
4. import org.slf4j.Logger;
5. import org.slf4j.LoggerFactory;
6. import org.springframework.stereotype.Service;
7.
8. import javax.annotation.Resource;
9. import java.math.BigDecimal;
10. import java.util.concurrent.TimeUnit;
11.
12. /**
13.  * 账户业务实现类
14.  */
15. @Service
16. public class AccountServiceImpl implements AccountService {
17.
18.     private static final Logger LOGGER =
19.         LoggerFactory.getLogger(AccountServiceImpl.class);
20.
21.     @Resource
22.     AccountDao accountDao;
23.
24.     /**
25.      * 扣减账户余额
26.      */
27.     @Override
28.     public void decrease(Long userId, BigDecimal money) {
29.         LOGGER.info("----->account-service中扣减账户余额开始");
30.         try {
31.             TimeUnit.SECONDS.sleep(20);
32.         } catch (InterruptedException e) {
33.             e.printStackTrace();
34.         }
35.         accountDao.decrease(userId, money);
36.         LOGGER.info("----->account-service中扣减账户余额结束");
37.     }
38. }
```

5.4.9. Controller

```

1. package cn.sitedev.springcloud.alibaba.controller;
2.
3. import cn.sitedev.springcloud.alibaba.domain.CommonResult;
4. import cn.sitedev.springcloud.alibaba.service.AccountService;
5. import org.springframework.web.bind.annotation.RequestMapping;
6. import org.springframework.web.bind.annotation.RequestParam;
7. import org.springframework.web.bind.annotation.RestController;
8.
9. import javax.annotation.Resource;
10. import java.math.BigDecimal;
11.
12. @RestController
13. public class AccountController {
14.
15.     @Resource
16.     AccountService accountService;
17.
18.     /**
19.      * 扣减账户余额
20.      */
21.     @RequestMapping("/account/decrease")
22.     public CommonResult decrease(@RequestParam("userId") Long userId,
23.     @RequestParam("money") BigDecimal money) {
24.         accountService.decrease(userId, money);
25.         return new CommonResult(200, "扣减账户余额成功!");
26.     }
27. }

```

5.4.10. Config配置

5.4.10.1. MyBatisConfig

```

1. package cn.sitedev.springcloud.alibaba.config;
2.
3. import org.mybatis.spring.annotation.MapperScan;
4. import org.springframework.context.annotation.Configuration;
5.
6. @Configuration
7. @MapperScan({"cn.sitedev.springcloud.alibaba.dao"})
8. public class MyBatisConfig {
9.
10. }

```

5.4.10.2. DataSourceProxyConfig

```
1. package cn.sitedev.springcloud.alibaba.config;
2.
3. import com.alibaba.druid.pool.DruidDataSource;
4. import io.seata.rm.datasource.DataSourceProxy;
5. import org.apache.ibatis.session.SqlSessionFactory;
6. import org.mybatis.spring.SqlSessionFactoryBean;
7. import org.mybatis.spring.transaction.SpringManagedTransactionFactory;
8. import org.springframework.beans.factory.annotation.Value;
9. import org.springframework.boot.context.properties.ConfigurationProperties;
10. import org.springframework.context.annotation.Bean;
11. import org.springframework.context.annotation.Configuration;
12. import org.springframework.core.io.support.PathMatchingResourcePatternResolver;
13.
14. import javax.sql.DataSource;
15.
16.
17. @Configuration
18. public class DataSourceProxyConfig {
19.
20.     @Value("${mybatis.mapperLocations}")
21.     private String mapperLocations;
22.
23.     @Bean
24.     @ConfigurationProperties(prefix = "spring.datasource")
25.     public DataSource druidDataSource() {
26.         return new DruidDataSource();
27.     }
28.
29.     @Bean
30.     public DataSourceProxy dataSourceProxy(DataSource dataSource) {
31.         return new DataSourceProxy(dataSource);
32.     }
33.
34.     @Bean
35.     public SqlSessionFactory sqlSessionFactoryBean(DataSourceProxy
dataSourceProxy) throws Exception {
36.         SqlSessionFactoryBean sqlSessionFactoryBean = new
SqlSessionFactoryBean();
37.         sqlSessionFactoryBean.setDataSource(dataSourceProxy);
38.         sqlSessionFactoryBean.setMapperLocations(new
PathMatchingResourcePatternResolver().getResources(mapperLocations));
39.         sqlSessionFactoryBean.setTransactionFactory(new
SpringManagedTransactionFactory());
40.         return sqlSessionFactoryBean.getObject();
41.     }
42.
43. }
```

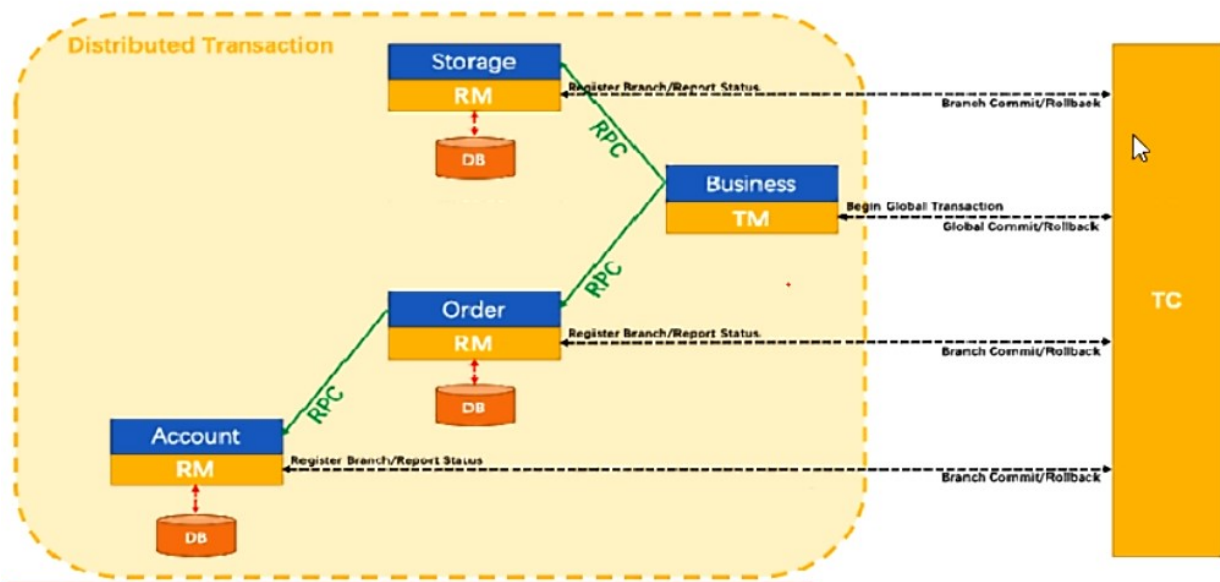
5.4.11. 主启动

```
1. package cn.sitedev.springcloud.alibaba;
2.
3. import org.springframework.boot.SpringApplication;
4. import org.springframework.boot.autoconfigure.SpringBootApplication;
5. import org.springframework.boot.autoconfigure.jdbc.DataSourceAutoConfiguration;
6. import org.springframework.cloud.client.discovery.EnableDiscoveryClient;
7. import org.springframework.cloud.openfeign.EnableFeignClients;
8.
9. @SpringBootApplication(exclude = DataSourceAutoConfiguration.class)
10. @EnableDiscoveryClient
11. @EnableFeignClients
12. public class SeataAccountMainApp2003 {
13.     public static void main(String[] args) {
14.         SpringApplication.run(SeataAccountMainApp2003.class, args);
15.     }
16. }
```

6. 测试

6.1. 下订单->减库存->扣余额->改（订单）状态

SEATA 的分布式交易解决方案



我们只需要使用一个@GlobalTransactional 注解在业务方法上

6.2. 数据库初始情况

保存 查询创建工具 美化 SQL 代码段 文本 导出结果

本地连接 运行已选择的 停

```

1 select * from seata_order.t_order;
2
3 select * from seata_storage.t_storage;
4
5 select * from seata_account.t_account;

```

信息 结果 1 剖析 状态

id	user_id	product_id	count	money	status
(N/A)	(N/A)	(N/A)	(N/A)	(N/A)	(N/A)

保存 查询创建工具 美化 SQL 代码段 文本

本地连接 运行已

```

1 select * from seata_order.t_order;
2
3 select * from seata_storage.t_storage;
4
5 select * from seata_account.t_account;

```

信息 结果 1 剖析 状态

id	product_id	total	used	residue
1	1	100	0	100

保存 查询创建工具 美化 SQL 代码段

本地连接

```

1 select * from seata_order.t_order;
2
3 select * from seata_storage.t_storage;
4
5 select * from seata_account.t_account;

```

信息 结果 1 剖析 状态

id	user_id	total	used	residue
1	1	1000	0	1000

6.3. 正常下单

1) seata-order-service2001模块OrderServiceImpl去除seata事务

```

1. public class OrderServiceImpl implements OrderService {
2.     @Override
3.     // @GlobalTransactional(name = "fsp-create-order", rollbackFor =
    Exception.class)
4.     public void create(Order order) {

```



```
Window Help    cloud2020 - OrderServiceImpl.java [seata-order-service2001]
ccountService  SeataAccountMainApp2003
AccountServiceImpl.java x OrderServiceImpl.java x
27      @Override
28      // @GlobalTransactional(name = "fsp-create-order", rollbackFor = Exception
    .class)
29      public void create(Order order) {
30          log.info("----->开始新建订单");
```

2) seata-order-service2003模块AccountServiceImpl去除超时代码

```
1. public class AccountServiceImpl implements AccountService {
2.     @Override
3.     public void decrease(Long userId, BigDecimal money) {
4.
5.         LOGGER.info("----->account-service中扣减账户余额开始");
6.         // try {
7.         //     TimeUnit.SECONDS.sleep(20);
8.         // } catch (InterruptedException e) {
9.         //     e.printStackTrace();
10.        // }
11.        accountDao.decrease(userId, money);
12.        LOGGER.info("----->account-service中扣减账户余额结束");
13.    }
```



```
Window Help    cloud2020 - AccountServiceImpl.java [seata-order-service2003]
decrease      SeataAccountMainApp2003
AccountServiceImpl.java x OrderServiceImpl.java x
24      */
25      @Override
26      public void decrease(Long userId, BigDecimal money) {
27
28          LOGGER.info("----->account-service中扣减账户余额开始");
29          try {
30              // TimeUnit.SECONDS.sleep(20);
31              // } catch (InterruptedException e) {
32              //     e.printStackTrace();
33              // }
34          accountDao.decrease(userId, money);
35          LOGGER.info("----->account-service中扣减账户余额结束");
36      }
37  }
```

3) 浏览器访问<http://localhost:2001/order/create?userId=1&productId=1&count=10&money=100>



4) 查看数据库记录

```

1 select * from seata_order.t_order;
2
3 select * from seata_storage.t_storage;
4
5 select * from seata_account.t_account;

```

id	user_id	product_id	count	money	status
15	1	1	10	100	1

```

1 select * from seata_order.t_order;
2
3 select * from seata_storage.t_storage;
4
5 select * from seata_account.t_account;

```

id	product_id	total	used	residue
1	1	100	10	90

```

1 select * from seata_order.t_order;
2
3 select * from seata_storage.t_storage;
4
5 select * from seata_account.t_account;

```

id	user_id	total	used	residue
1	1	1000	100	900

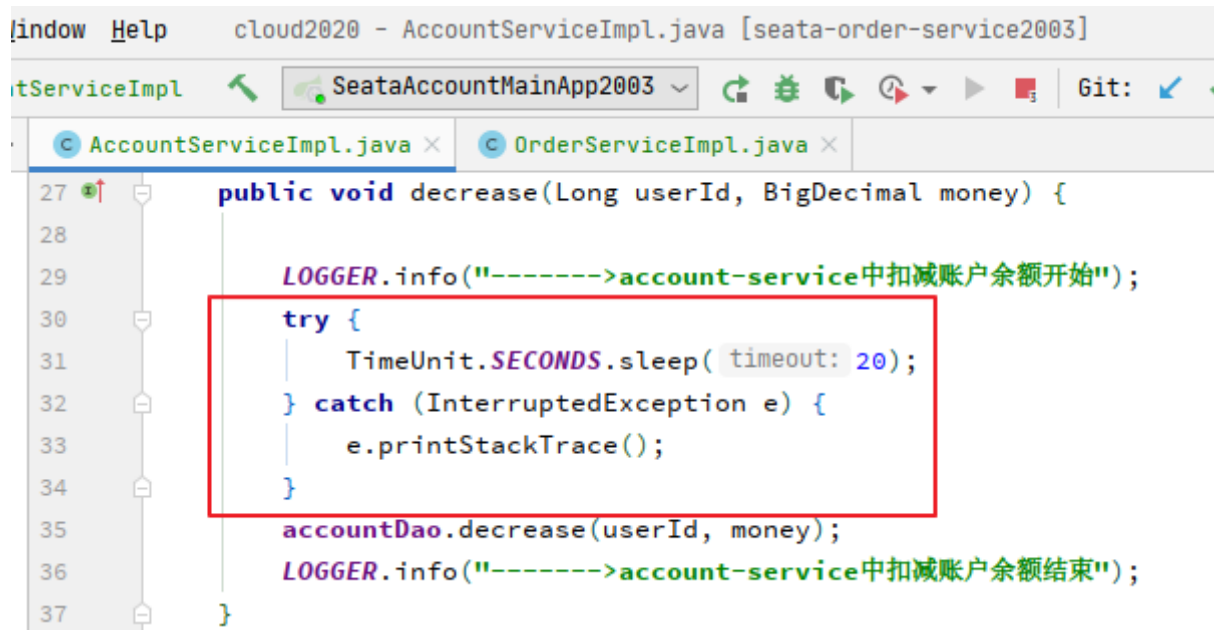
6.4. 超时异常，没加@GlobalTransactional

1) seata-order-service2003模块AccountServiceImpl添加超时代码:

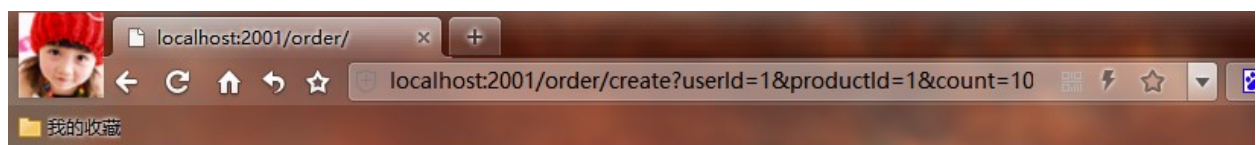
```

1. public class AccountServiceImpl implements AccountService {
2.     @Override
3.     public void decrease(Long userId, BigDecimal money) {
4.
5.         LOGGER.info("----->account-service中扣减账户余额开始");
6.         try {
7.             TimeUnit.SECONDS.sleep(20);
8.         } catch (InterruptedException e) {
9.             e.printStackTrace();
10.        }
11.        accountDao.decrease(userId, money);
12.        LOGGER.info("----->account-service中扣减账户余额结束");
13.    }

```



2) 浏览器访问<http://localhost:2001/order/create?userId=1&productId=1&count=10&money=100>



Whitelabel Error Page

This application has no explicit mapping for /error, so you are seeing this as a fallback.

Sun Jul 12 16:10:13 CST 2020

There was an unexpected error (type=Internal Server Error, status=500).

Read timed out executing POST http://seata-account-service/account/decrease?userId=1&money=100

3) 查看数据库记录:

```

1 select * from seata_order.t_order;
2
3 select * from seata_storage.t_storage;
4
5 select * from seata_account.t_account;

```

信息	结果 1		剖析	状态		
	id	user_id	product_id	count	money	status
▶	15	1	1	10	100	1
	16	1	1	10	100	0

本地连接

```

1 select * from seata_order.t_order;
2
3 select * from seata_storage.t_storage;
4
5 select * from seata_account.t_account;

```

信息	结果 1	剖析	状态		
id	product_id	total	used	residue	
1	1	100	20	80	

本地连接

```

1 select * from seata_order.t_order;
2
3 select * from seata_storage.t_storage;
4
5 select * from seata_account.t_account;

```

信息	结果 1	剖析	状态	
id	user_id	total	used	residue
1	1	1000	200	800

4) 故障情况:

- 当库存和账户余额扣减后，订单状态并没有设置为已经完成，没有从零改为1
- 而且由于feign的重试机制，账户余额还有可能被多次扣减
- feign的超时时间默认为1s

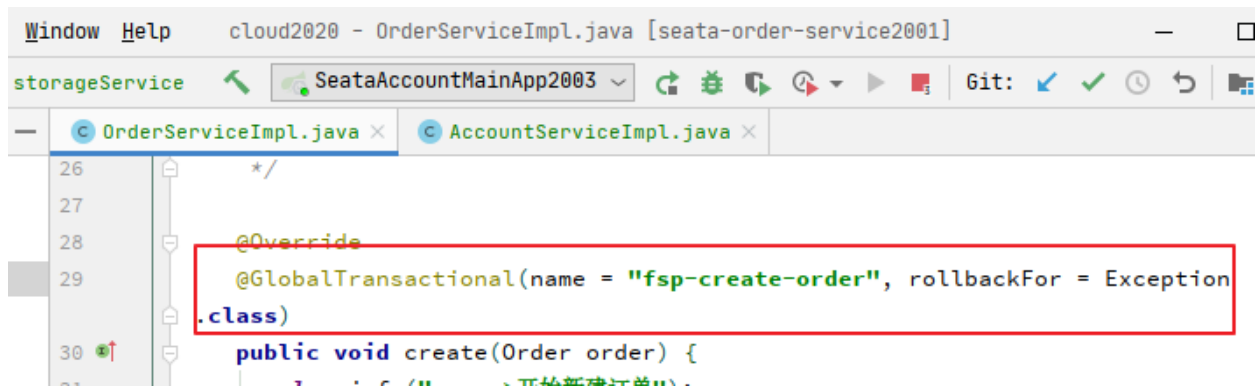
6.5. 超时异常，添加@GlobalTransactional

1) seata-order-service2001模块OrderServiceImpl添加seata事务

```

1. public class OrderServiceImpl implements OrderService {
2.     @Override
3.     @GlobalTransactional(name = "fsp-create-order", rollbackFor =
4.         Exception.class)
5.     public void create(Order order) {

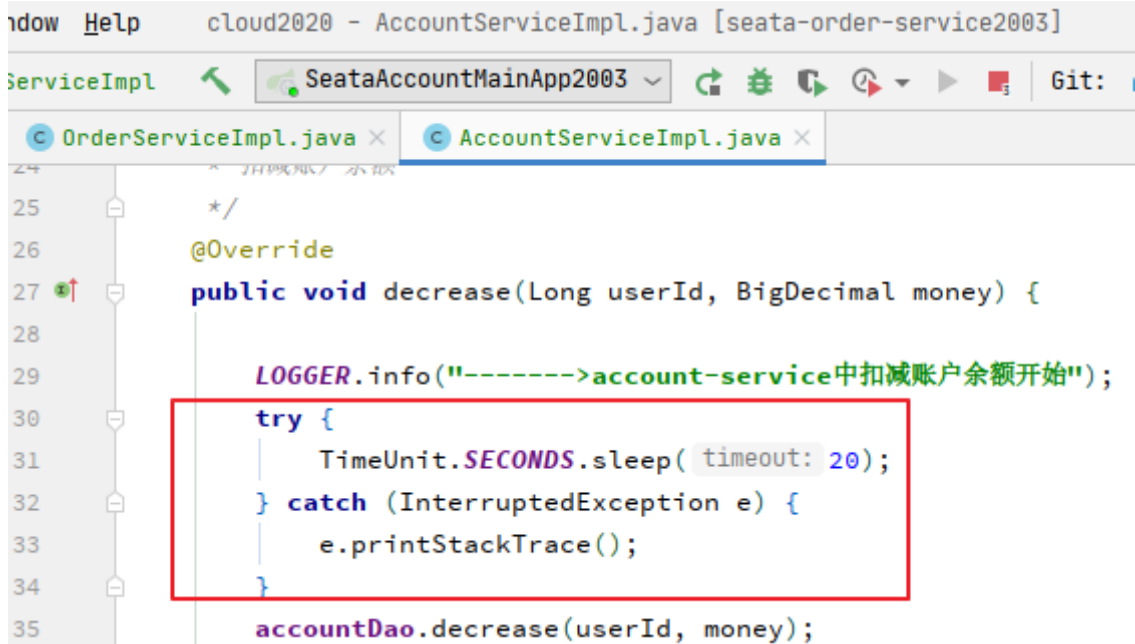
```



```
Window Help    cloud2020 - OrderServiceImpl.java [seata-order-service2001]
storageService SeataAccountMainApp2003
OrderServiceImpl.java x AccountServiceImpl.java x
26  */
27
28  @Override
29  @GlobalTransactional(name = "fsp-create-order", rollbackFor = Exception
30  .class)
31  public void create(Order order) {
```

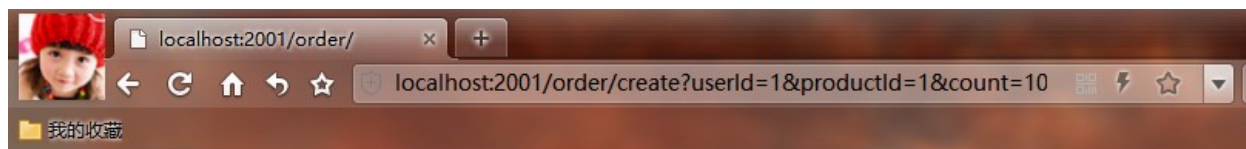
2) seata-order-service2003模块AccountServiceImpl添加超时代码

```
1. public class AccountServiceImpl implements AccountService {
2.     public void decrease(Long userId, BigDecimal money) {
3.
4.         LOGGER.info("----->account-service中扣减账户余额开始");
5.         try {
6.             TimeUnit.SECONDS.sleep(20);
7.         } catch (InterruptedException e) {
8.             e.printStackTrace();
9.         }
10.        accountDao.decrease(userId, money);
11.        LOGGER.info("----->account-service中扣减账户余额结束");
12.    }
```



```
Window Help    cloud2020 - AccountServiceImpl.java [seata-order-service2003]
ServiceImpl SeataAccountMainApp2003
OrderServiceImpl.java x AccountServiceImpl.java x
25  */
26  @Override
27  public void decrease(Long userId, BigDecimal money) {
28
29      LOGGER.info("----->account-service中扣减账户余额开始");
30      try {
31          TimeUnit.SECONDS.sleep(timeout: 20);
32      } catch (InterruptedException e) {
33          e.printStackTrace();
34      }
35      accountDao.decrease(userId, money);
```

3) 浏览器访问<http://localhost:2001/order/create?userId=1&productId=1&count=10&money=100>



Whitelabel Error Page

This application has no explicit mapping for /error, so you are seeing this as a fallback.

Sun Jul 12 16:15:42 CST 2020

There was an unexpected error (type=Internal Server Error, status=500).

Read timed out executing POST <http://seata-account-service/account/decrease?userId=1&money=100>

4) 下单后数据库数据并没有任何改变, 记录都添加不进来

```
1 select * from seata_order.t_order;
2
3 select * from seata_storage.t_storage;
4
5 select * from seata_account.t_account;
```

信息	结果 1	剖析	状态			
	id	user_id	product_id	count	money	status
▶	15	1	1	10	100	1
	16	1	1	10	100	0

```
1 select * from seata_order.t_order;
2
3 select * from seata_storage.t_storage;
4
5 select * from seata_account.t_account;
```

信息	结果 1	剖析	状态		
	id	product_id	total	used	residue
▶	1	1	100	20	80

```
1 select * from seata_order.t_order;
2
3 select * from seata_storage.t_storage;
4
5 select * from seata_account.t_account;
```

信息	结果 1	剖析	状态		
	id	user_id	total	used	residue
▶	1	1	1000	200	800

7. Seata之原理简介

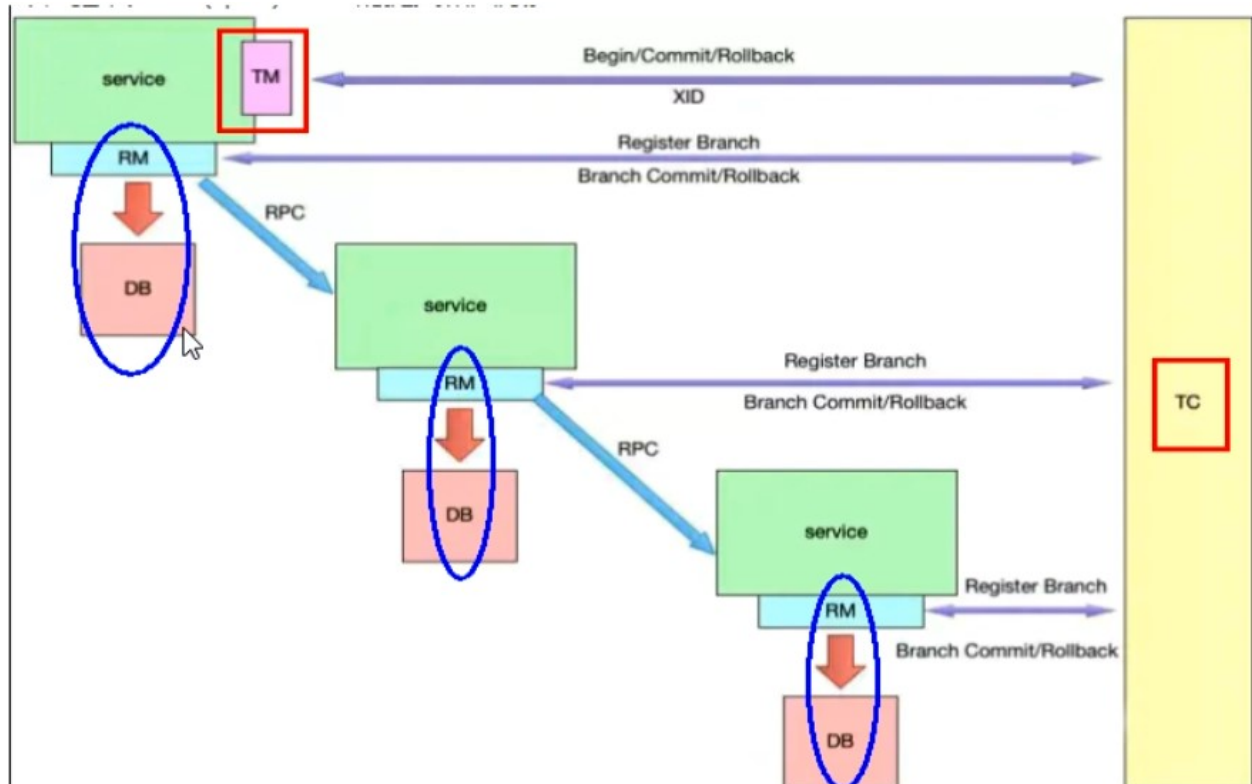
7.1. Seata

2019年1月份蚂蚁金服和阿里巴巴共同开源的分布式事务解决方案

Simple Extensible Autonomous Transaction Architecture,简单可扩展自治事务框架

2020年初, 参加工作后用1.0以后的版本

7.2. 再看TC/TM/RM三大组件



7.2.1. 分布式事务的执行流程

TM开启分布式事务(TM向TC注册全局事务记录)

换业务场景, 编排数据库, 服务等事务内资源 (RM向TC汇报资源准备状态)

TM结束分布式事务, 事务一阶段结束 (TM通知TC提交/回滚分布式事务)

TC汇总事务信息, 决定分布式事务是提交还是回滚

TC通知所有RM提交/回滚资源, 事务二阶段结束。

7.3. AT模式如何做到对业务的无侵入

7.3.1. 是什么

<http://seata.io/zh-cn/docs/overview/what-is-seata.html>

Seata 是一款开源的分布式事务解决方案，致力于提供高性能和简单易用的分布式事务服务。
Seata 将为用户提供了 AT、TCC、SAGA 和 XA 事务模式，为用户打造一站式的分布式解决方案。

AT 模式

前提

- 基于支持本地 ACID 事务的关系型数据库。
- Java 应用，通过 JDBC 访问数据库。

整体机制

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。

7.3.2. 一阶段加载

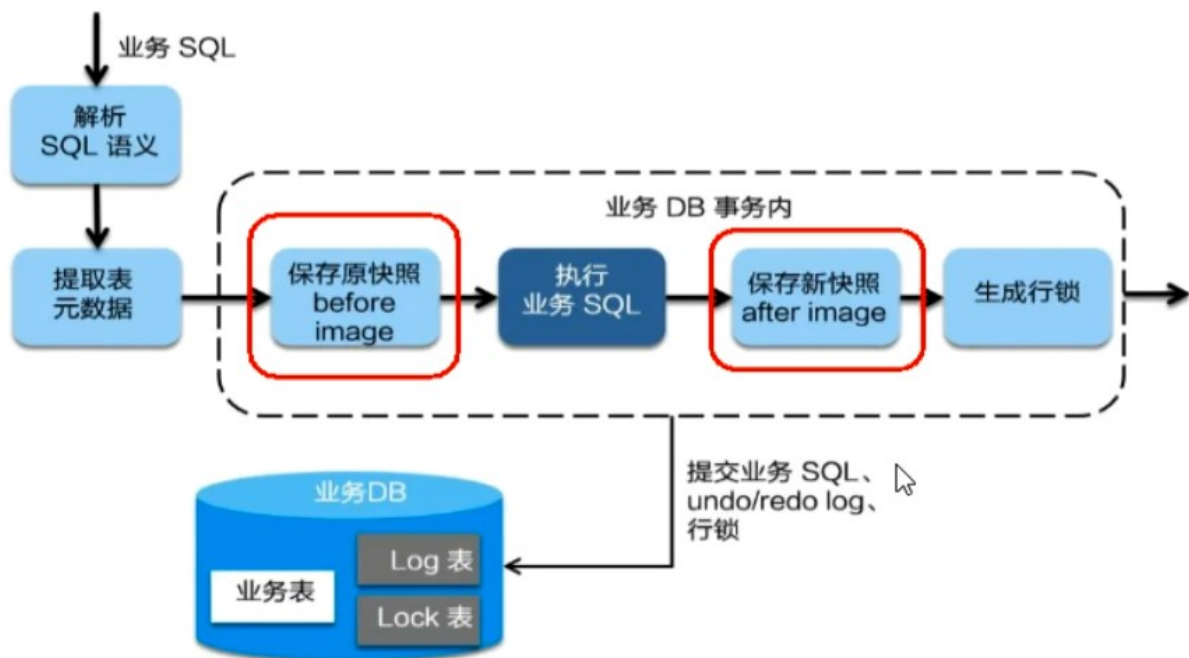
在一阶段，Seata会拦截"业务SQL"，

1.解析SQL语义，找到"业务SQL"要更新的业务数据，在业务数据被更新前，将其保存成"before image"，

2.执行“业务SQL”更新业务数据，在业务数据更新之后，

3.其保存成"after image"，最后生成行锁

以上操作全部在一个数据库事务内完成，这样保证了一阶段操作的原子性。

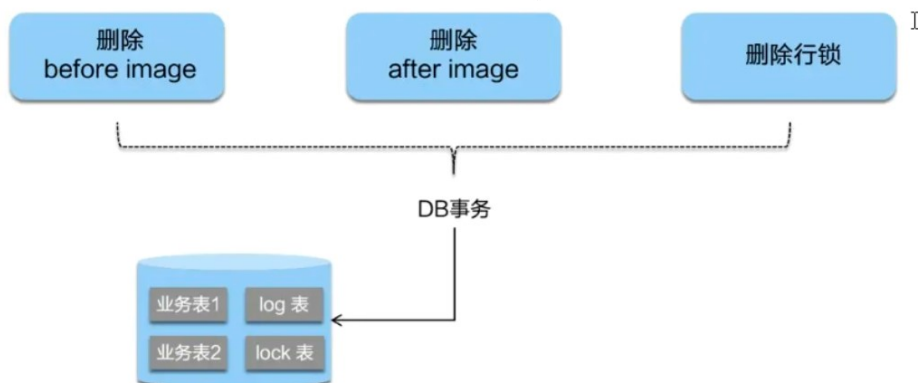


7.3.3. 二阶段提交

二阶段如是顺利提交的话，因为“业务SQL”在一阶段已经提交至数据库，所以 Seata框架只需将阶段保存快照数据和行锁删掉，完成数据清理即可

二阶段如是顺利提交的话，

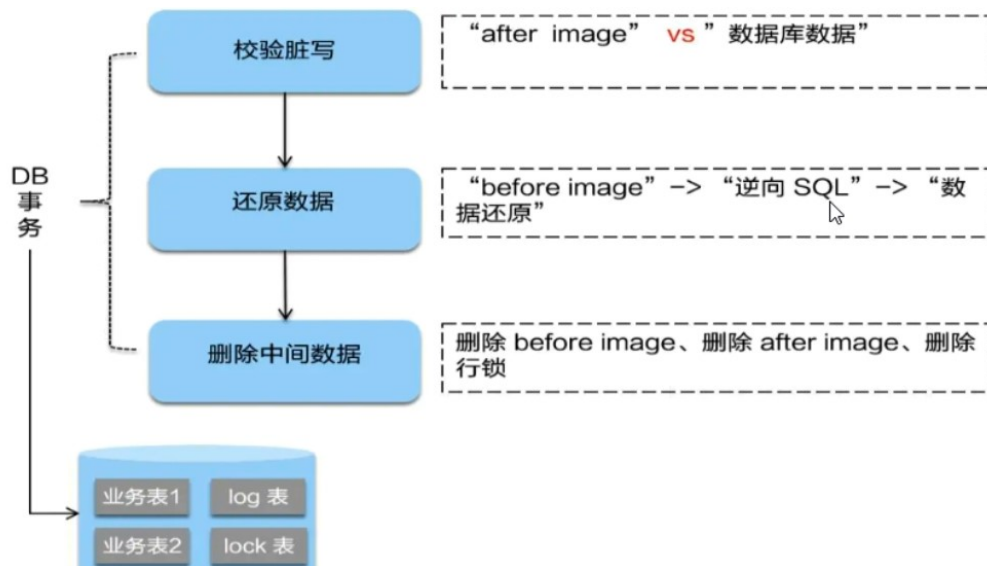
因为“业务 SQL”在一阶段已经提交至数据库，所以Seata框架只需将一阶段保存的快照数据和行锁删掉，完成数据清理即可。



7.3.4. 二阶段回滚

二阶段如果是回滚的话，Seata就需要回滚一阶段已经执行的“业务SQL”，还原业务数据。

回滚方式便是用“before image”还原业务数据；但在还原前要首先要校验脏写，对比“数据库当前业务数据”和“after image”，如果两份数据完全一致就说明没有脏写，可以还原业务数据，如果不一致就说明有脏写，出现脏写就需要转人工处理。



7.4. 补充

