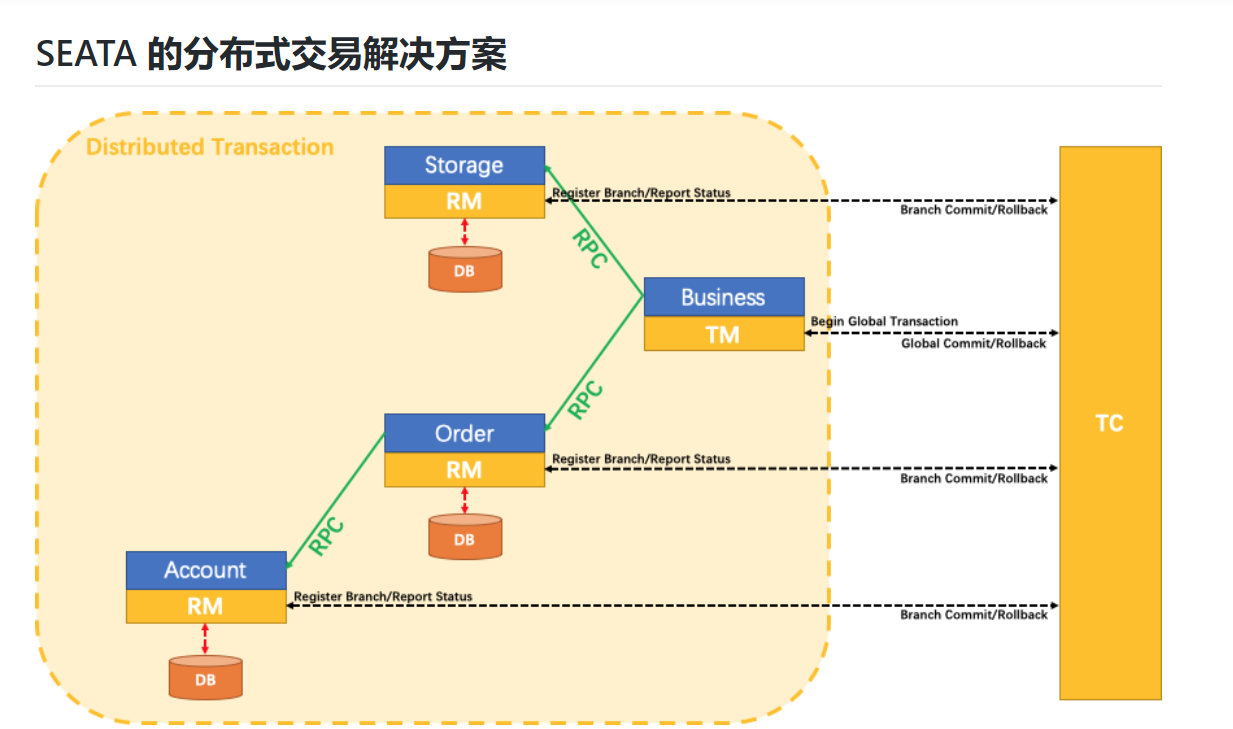
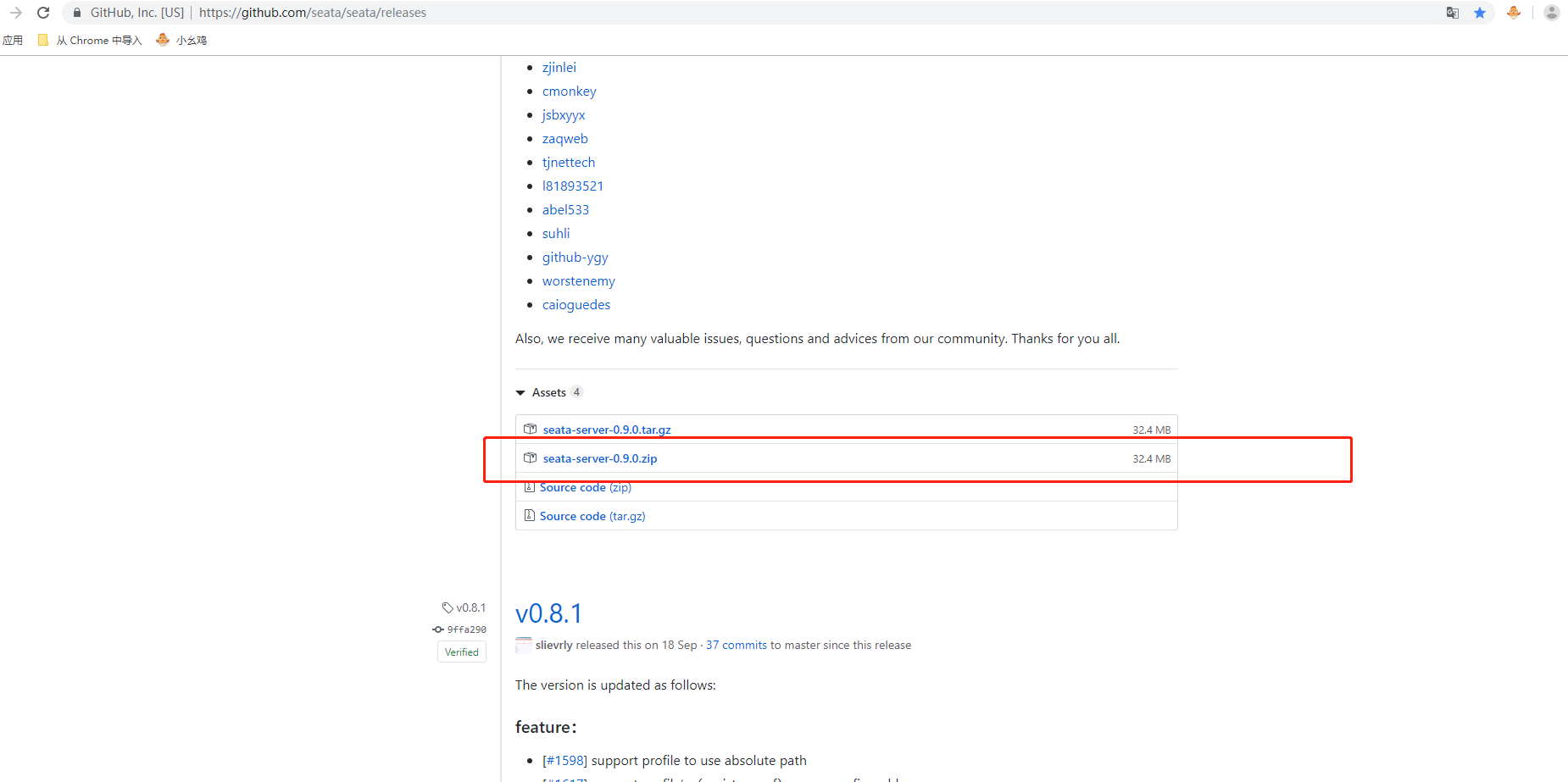
在工作中，我们用到了springcloud来实现微服务，目前有订单服务（order-service）、库存服务(storage-service)、账户服务(account-service),我们需要在下单服务中（business-service）中调用order-service和storage服务，而order-service中调用了account-service，这三个服务连接的数据库可能是不一样的，即每个服务都是单独的数据库，如果账户不足或者库存不够都不能下单成功，所以就需要用到分布式事务。



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。

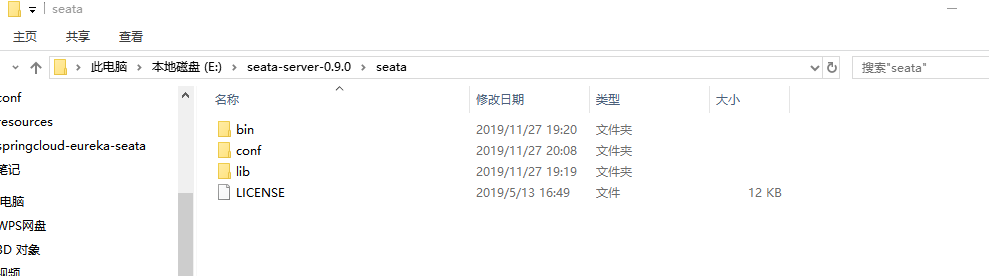
下载seata-server，<https://github.com/seata/seata/releases>，如图：



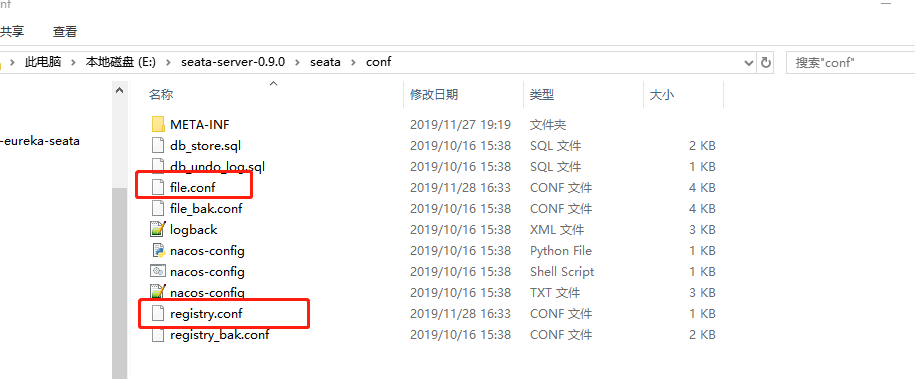


开启springcloud-seata的demo：

1. 解压seata-server-0.9.0.zip到E盘下面



1. 依次修改E:\seata-server-0.9.0\seata\conf文件夹下的file.conf和registry.conf文件



file.conf：



该文件中主要修改图中地方：



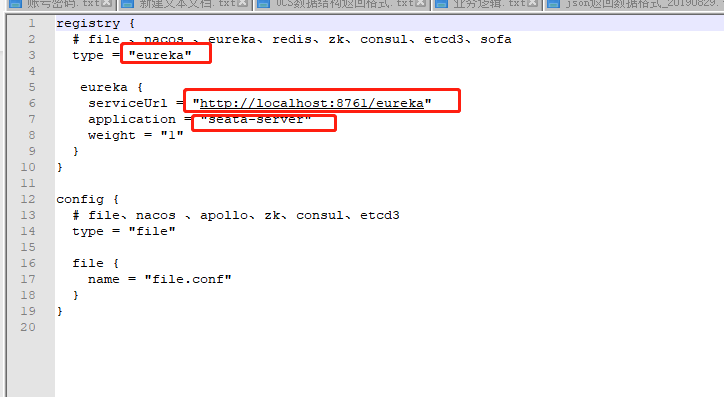
registry.conf:



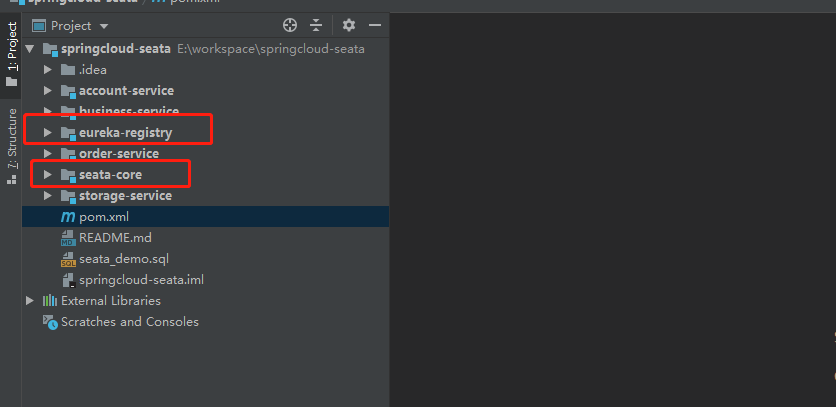
**registry.type = "eureka"  #注册发现中心为eureka**

**registry.eureka.serviceUrl = "${你自己的eureka地址}"**

**registry.eureka.application = "${注册到eureka中的应用名称}"**



1. 开始撸代码了，工程结构图如下：



主pom.xml文件：

<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.xcxcxcxcx</groupId>

<artifactId>springcloud-seata</artifactId>

<packaging>pom</packaging>

<version>1.0-SNAPSHOT</version>

<!-- 父pom必须要加上parent -->

<parent>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>

<version>2.1.5.RELEASE</version>

<relativePath/> <!-- lookup parent from repository -->

</parent>

<modules>

<module>order-service</module>

<module>account-service</module>

<module>storage-service</module>

<module>business-service</module>

<module>eureka-registry</module>

<module>seata-core</module>

</modules>

<dependencies>

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-dependencies</artifactId>

<version>Finchley.RELEASE</version>

<type>pom</type>

<scope>import</scope>

</dependency>

<dependency>

<groupId>com.google.code.gson</groupId>

<artifactId>gson</artifactId>

<version>2.8.5</version>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-test</artifactId>

<scope>test</scope>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

<version>2.1.5.RELEASE</version>

<exclusions>

<exclusion>

<groupId>org.springframework</groupId>

<artifactId>spring-core</artifactId>

</exclusion>

</exclusions>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-core</artifactId>

<version>5.1.7.RELEASE</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>5.1.7.RELEASE</version>

</dependency>

<dependency>

<groupId>org.apache.commons</groupId>

<artifactId>commons-lang3</artifactId>

<version>3.7</version>

</dependency>

<dependency>

<groupId>commons-collections</groupId>

<artifactId>commons-collections</artifactId>

<version>3.2.2</version>

</dependency>

<dependency>

<groupId>com.alibaba</groupId>

<artifactId>fastjson</artifactId>

<version>1.2.60</version>

</dependency>

</dependencies>

<dependencyManagement>

<dependencies>

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-dependencies</artifactId>

<version>Finchley.RELEASE</version>

<type>pom</type>

<scope>import</scope>

</dependency>

</dependencies>

</dependencyManagement>

<build>

<resources>

<resource>

<directory>src/main/resources</directory>

<includes>

<include>\*\*/\*</include>

</includes>

<filtering>true</filtering>

</resource>

</resources>

<plugins>

<plugin>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-maven-plugin</artifactId>

</plugin>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-compiler-plugin</artifactId>

<version>3.8.0</version>

<configuration>

<source>1.8</source>

<target>1.8</target>

</configuration>

</plugin>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-pmd-plugin</artifactId>

<version>3.8</version>

<configuration>

<sourceEncoding>UTF-8</sourceEncoding>

<minimumPriority>2</minimumPriority>

<printFailingErrors>true</printFailingErrors>

<rulesets>

<ruleset>rulesets/java/ali-comment.xml</ruleset>

<ruleset>rulesets/java/ali-concurrent.xml</ruleset>

<ruleset>rulesets/java/ali-constant.xml</ruleset>

<ruleset>rulesets/java/ali-exception.xml</ruleset>

<ruleset>rulesets/java/ali-flowcontrol.xml</ruleset>

<ruleset>rulesets/java/ali-naming.xml</ruleset>

<ruleset>rulesets/java/ali-oop.xml</ruleset>

<ruleset>rulesets/java/ali-orm.xml</ruleset>

<ruleset>rulesets/java/ali-other.xml</ruleset>

<ruleset>rulesets/java/ali-set.xml</ruleset>

</rulesets>

</configuration>

<executions>

<execution>

<phase>verify</phase>

<goals>

<goal>check</goal>

</goals>

</execution>

</executions>

<dependencies>

<dependency>

<groupId>com.alibaba.p3c</groupId>

<artifactId>p3c-pmd</artifactId>

<version>1.3.6</version>

</dependency>

</dependencies>

</plugin>

<plugin>

<artifactId>maven-resources-plugin</artifactId>

<configuration>

<encoding>utf-8</encoding>

<useDefaultDelimiters>true</useDefaultDelimiters>

<nonFilteredFileExtensions>

<nonFilteredFileExtension>xls</nonFilteredFileExtension>

<nonFilteredFileExtension>xlsx</nonFilteredFileExtension>

<nonFilteredFileExtension>datx</nonFilteredFileExtension>

</nonFilteredFileExtensions>

</configuration>

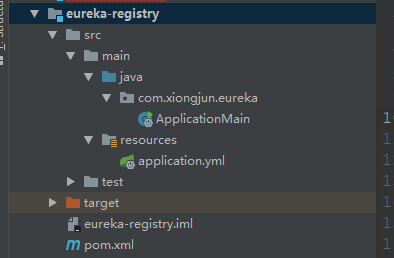
</plugin>

</plugins>

</build>

</project>

eureka-registry是服务注册中心，



pom.xml如下：

<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

<parent>

<groupId>com.xcxcxcxcx</groupId>

<artifactId>springcloud-seata</artifactId>

<version>1.0-SNAPSHOT</version>

</parent>

<modelVersion>4.0.0</modelVersion>

<artifactId>eureka-registry</artifactId>

<dependencies>

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-netflix-eureka-server</artifactId>

<version>2.0.2.RELEASE</version>

</dependency>

</dependencies>

</project>

配置application.yml如下：

server:

port: 8761

eureka:

instance:

hostname: localhost

prefer-ip-address: true

client:

registerWithEureka: false

fetchRegistry: false

serviceUrl:

defaultZone: http://${eureka.instance.hostname}:${server.port}/eureka/

spring:

application:

name: eureka-registry

ApplicationMain.java:

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.cloud.netflix.eureka.server.EnableEurekaServer;

@SpringBootApplication

@EnableEurekaServer

public class ApplicationMain {

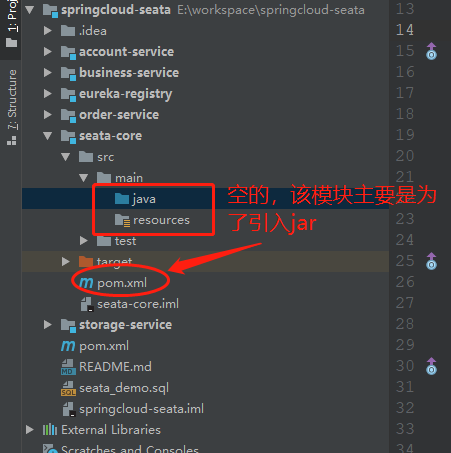
public static void main(String[] args) {

SpringApplication.run(ApplicationMain.class, args);

}

}

seata-core模块主要是为了存放引入seata需要的jar



pom.xml:

<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

<parent>

<artifactId>springcloud-seata</artifactId>

<groupId>com.xcxcxcxcx</groupId>

<version>1.0-SNAPSHOT</version>

</parent>

<modelVersion>4.0.0</modelVersion>

<artifactId>seata-core</artifactId>

<dependencies>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

<exclusions>

<exclusion>

<artifactId>spring-web</artifactId>

<groupId>org.springframework</groupId>

</exclusion>

</exclusions>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-test</artifactId>

<scope>test</scope>

</dependency>

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

<scope>runtime</scope>

</dependency>

<dependency>

<groupId>com.alibaba</groupId>

<artifactId>druid-spring-boot-starter</artifactId>

<version>1.1.21</version>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-data-jpa</artifactId>

</dependency>

<dependency>

<groupId>org.mybatis.spring.boot</groupId>

<artifactId>mybatis-spring-boot-starter</artifactId>

<version>2.1.1</version>

</dependency>

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-openfeign</artifactId>

<exclusions>

<exclusion>

<artifactId>spring-web</artifactId>

<groupId>org.springframework</groupId>

</exclusion>

</exclusions>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-web</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>

<version>2.1.4.RELEASE</version>

</dependency>

<!-- https://mvnrepository.com/artifact/com.alibaba.cloud/spring-cloud-alibaba-seata -->

<dependency>

<groupId>com.alibaba.cloud</groupId>

<artifactId>spring-cloud-alibaba-seata</artifactId>

<version>2.1.1.RELEASE</version>

</dependency>

<dependency>

<groupId>io.seata</groupId>

<artifactId>seata-all</artifactId>

<version>0.9.0</version>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-maven-plugin</artifactId>

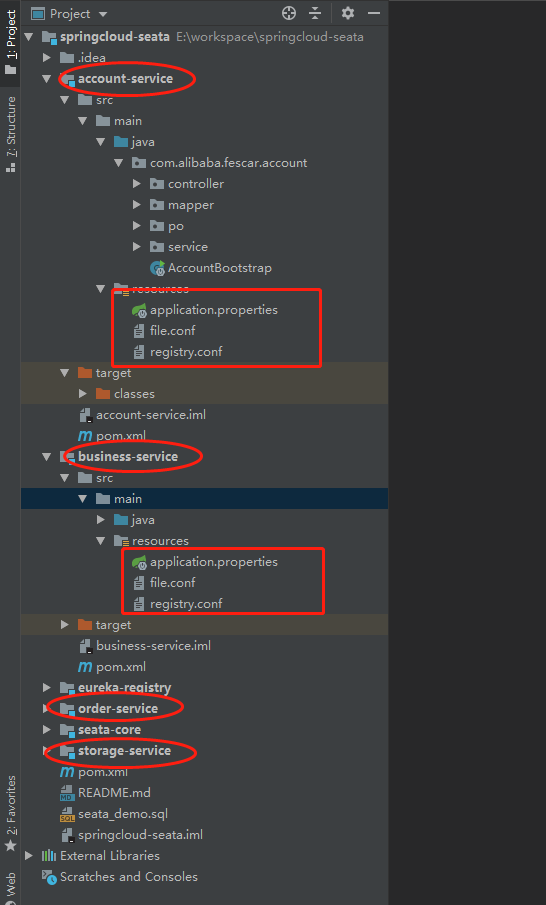
</plugin>

</plugins>

</build>

</project>

修改demo中的配置，以springcloud-seata/account-service的配置文件为例，其他的module配置均一样(4个module的application.properties、file.conf和registry.conf都一样)



application.properties:

spring.application.name=account-service#服务名称

server.port=11111#服务端口号

### datasource mysql

spring.datasource.type=com.zaxxer.hikari.util.DriverDataSource

spring.datasource.url=jdbc:mysql://10.10.1.12:3306/xxl\_crud?useUnicode=true&characterEncoding=UTF-8

spring.datasource.username=root

spring.datasource.password=ubt83474428!@

spring.datasource.tomcat.driver-class-name=com.mysql.jdbc.Driver

spring.datasource.tomcat.initial-size=5

spring.datasource.tomcat.min-idle=5

spring.datasource.tomcat.max-active=20

spring.datasource.tomcat.max-wait=60000

spring.datasource.tomcat.time-between-eviction-runs-millis=60000

spring.datasource.tomcat.min-evictable-idle-time-millis=300000

spring.datasource.tomcat.validation-query=SELECT 1 FROM DUAL

spring.datasource.tomcat.test-while-idle=true

spring.datasource.tomcat.test-on-borrow=false

spring.datasource.tomcat.test-on-return=false

spring.datasource.dbcp2.pool-prepared-statements=true

logging.level.io.seata=debug

spring.cloud.alibaba.seata.tx-service-group=my\_test\_tx\_group

#跟seata-server中的file.conf文件中vgroup\_mapping.my\_test\_tx\_group=

"seata-server"保持一致

#eureka

#注册中心服务运行的IP

eureka.instance.hostname=127.0.0.1

#注册中心服务运行的端口

eureka.instance.port=8761

#eureka注册中心的地址

eureka.client.serviceUrl.defaultZone=http://${eureka.instance.hostname}:${eureka.instance.port}/eureka/

#以IP地址注册到服务中心，相互注册使用IP地址

eureka.instance.prefer-ip-address=true

#本服务注册到eureka的信息格式

eureka.instance.instance-id=${spring.cloud.client.ip-address}:${spring.application.name}:${server.port}

file.conf:(主要修改下面标注的地方)

transport {

# tcp udt unix-domain-socket

type = "TCP"

#NIO NATIVE

server = "NIO"

#enable heartbeat

heartbeat = true

#thread factory for netty

thread-factory {

boss-thread-prefix = "NettyBoss"

worker-thread-prefix = "NettyServerNIOWorker"

server-executor-thread-prefix = "NettyServerBizHandler"

share-boss-worker = false

client-selector-thread-prefix = "NettyClientSelector"

client-selector-thread-size = 1

client-worker-thread-prefix = "NettyClientWorkerThread"

# netty boss thread size,will not be used for UDT

boss-thread-size = 1

#auto default pin or 8

worker-thread-size = 8

}

shutdown {

# when destroy server, wait seconds

wait = 3

}

serialization = "seata"

compressor = "none"

}

service {

#vgroup->rgroup

vgroup\_mapping.my\_test\_tx\_group="seata-server"

#only support single node

seata-server.grouplist = "127.0.0.1:8091"

#degrade current not support

enableDegrade = false

#disable

disable = false

#unit ms,s,m,h,d represents milliseconds, seconds, minutes, hours, days, default permanent

max.commit.retry.timeout = "-1"

max.rollback.retry.timeout = "-1"

disableGlobalTransaction = false

}

client {

async.commit.buffer.limit = 10000

lock {

retry.internal = 10

retry.times = 30

}

report.retry.count = 5

tm.commit.retry.count = 1

tm.rollback.retry.count = 1

}

transaction {

undo.data.validation = true

undo.log.serialization = "jackson"

undo.log.save.days = 7

#schedule delete expired undo\_log in milliseconds

undo.log.delete.period = 86400000

undo.log.table = "undo\_log"

}

support {

## spring

spring {

# auto proxy the DataSource bean

datasource.autoproxy = false

}

}

registry.conf:

registry {

# file 、nacos 、eureka、redis、zk、consul、etcd3、sofa

type = "file"

file {

name = "file.conf"

}

}

config {

# file、nacos 、apollo、zk、consul、etcd3

type = "file"

file {

name = "file.conf"

}

}

服务启动类修改AccountBootstrap.java

import com.alibaba.druid.pool.DruidDataSource;

import io.seata.rm.datasource.DataSourceProxy;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.boot.context.properties.ConfigurationProperties;

import org.springframework.cloud.client.discovery.EnableDiscoveryClient;

import org.springframework.cloud.openfeign.EnableFeignClients;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Primary;

import javax.sql.DataSource;

/\*\*

\* @author XCXCXCXCX

\* @since 1.0

\*/

@SpringBootApplication

@EnableDiscoveryClient

@EnableFeignClients

public class AccountBootstrap {

public static void main(String[] args) {

SpringApplication.run(AccountBootstrap.class, args);

}

//特别要注意下面两个方法，必须要加上，否则注册不到seata-server服务中去，也达不到分布式事务的效果。四个服务，每个都得加。

@Bean

@ConfigurationProperties(prefix = "spring.datasource")

public DataSource dataSource() {

DruidDataSource druidDataSource = new DruidDataSource();

return druidDataSource;

}

@Primary

@Bean("dataSourceProxy")

public DataSourceProxy dataSourceProxy(DataSource dataSource) {

return new DataSourceProxy(dataSource);

}

}

按照account-service同样修改business-sevice、order-service、storage-service,此刻代码修改完毕。

1. 执行sql文件：

DROP TABLE IF EXISTS `account\_tbl`;

CREATE TABLE `account\_tbl` (

`id` int(11) NOT NULL AUTO\_INCREMENT,

`user\_id` varchar(255) DEFAULT NULL,

`money` int(11) DEFAULT '0',

PRIMARY KEY (`id`)

) ENGINE=InnoDB AUTO\_INCREMENT=5 DEFAULT CHARSET=utf8;

INSERT INTO `account\_tbl` VALUES ('4', 'U100001', '999');

-- ----------------------------

-- Table structure for order\_tbl

-- ----------------------------

DROP TABLE IF EXISTS `order\_tbl`;

CREATE TABLE `order\_tbl` (

`id` int(11) NOT NULL AUTO\_INCREMENT,

`user\_id` varchar(255) DEFAULT NULL,

`commodity\_code` varchar(255) DEFAULT NULL,

`count` int(11) DEFAULT '0',

`money` int(11) DEFAULT '0',

PRIMARY KEY (`id`)

) ENGINE=InnoDB AUTO\_INCREMENT=27 DEFAULT CHARSET=utf8;

-- ----------------------------

-- Table structure for storage\_tbl

-- ----------------------------

DROP TABLE IF EXISTS `storage\_tbl`;

CREATE TABLE `storage\_tbl` (

`id` int(11) NOT NULL AUTO\_INCREMENT,

`commodity\_code` varchar(255) DEFAULT NULL,

`count` int(11) DEFAULT '0',

PRIMARY KEY (`id`),

UNIQUE KEY `commodity\_code` (`commodity\_code`)

) ENGINE=InnoDB AUTO\_INCREMENT=5 DEFAULT CHARSET=utf8;

-- ----------------------------

-- Records of storage\_tbl

-- ----------------------------

INSERT INTO `storage\_tbl` VALUES ('4', 'C00321', '100');

-- ----------------------------

-- Table structure for undo\_log

-- ----------------------------

DROP TABLE IF EXISTS `undo\_log`;

CREATE TABLE `undo\_log` (

`id` bigint(20) NOT NULL AUTO\_INCREMENT,

`branch\_id` bigint(20) NOT NULL,

`xid` varchar(100) NOT NULL,

`context` varchar(128) NOT NULL,

`rollback\_info` longblob NOT NULL,

`log\_status` int(11) NOT NULL,

`log\_created` datetime NOT NULL,

`log\_modified` datetime NOT NULL,

`ext` varchar(100) DEFAULT NULL,

PRIMARY KEY (`id`),

UNIQUE KEY `ux\_undo\_log` (`xid`,`branch\_id`)

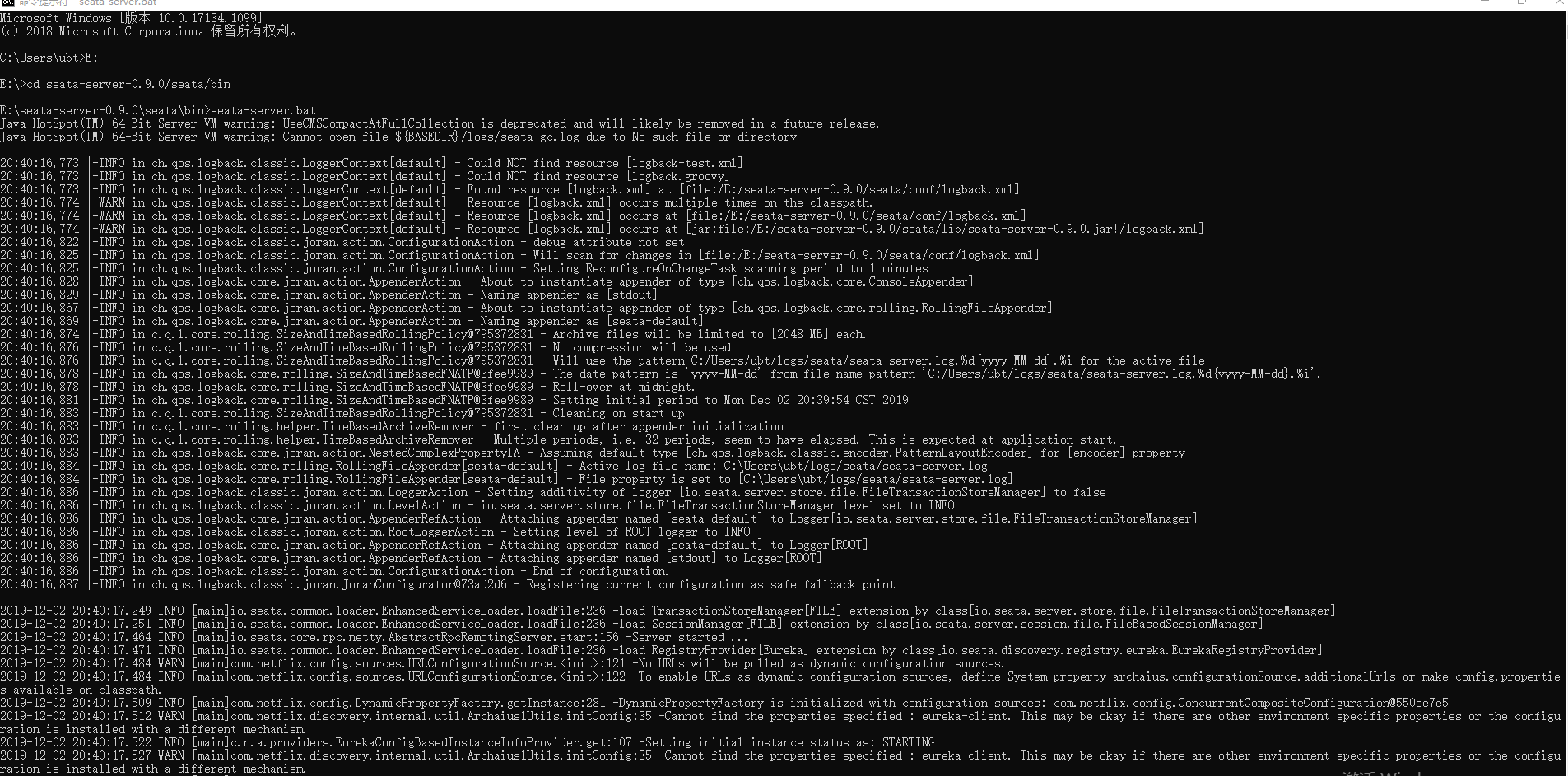
) ENGINE=InnoDB AUTO\_INCREMENT=1 DEFAULT CHARSET=utf8;

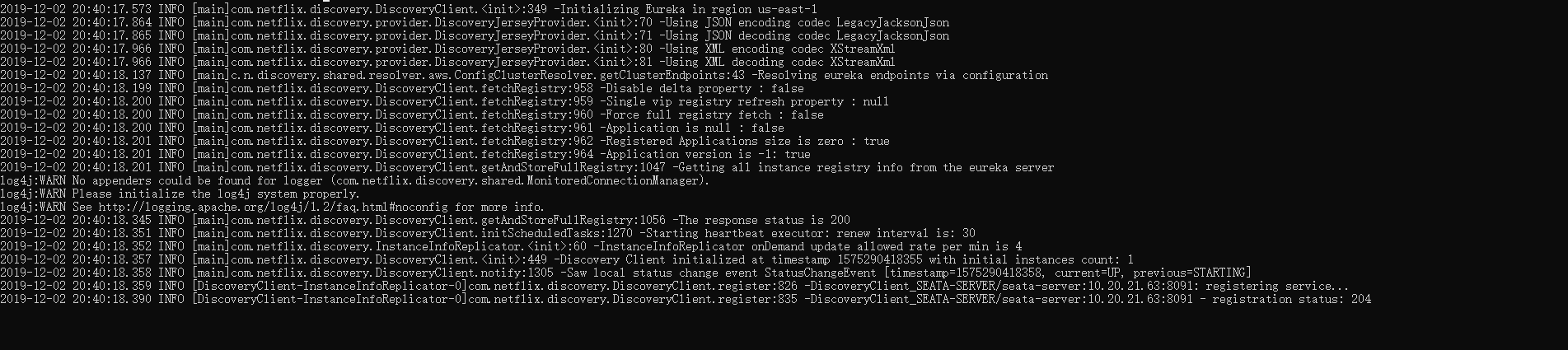
1. 开始跑工程了：

首先启动注册中心eureka-registry，

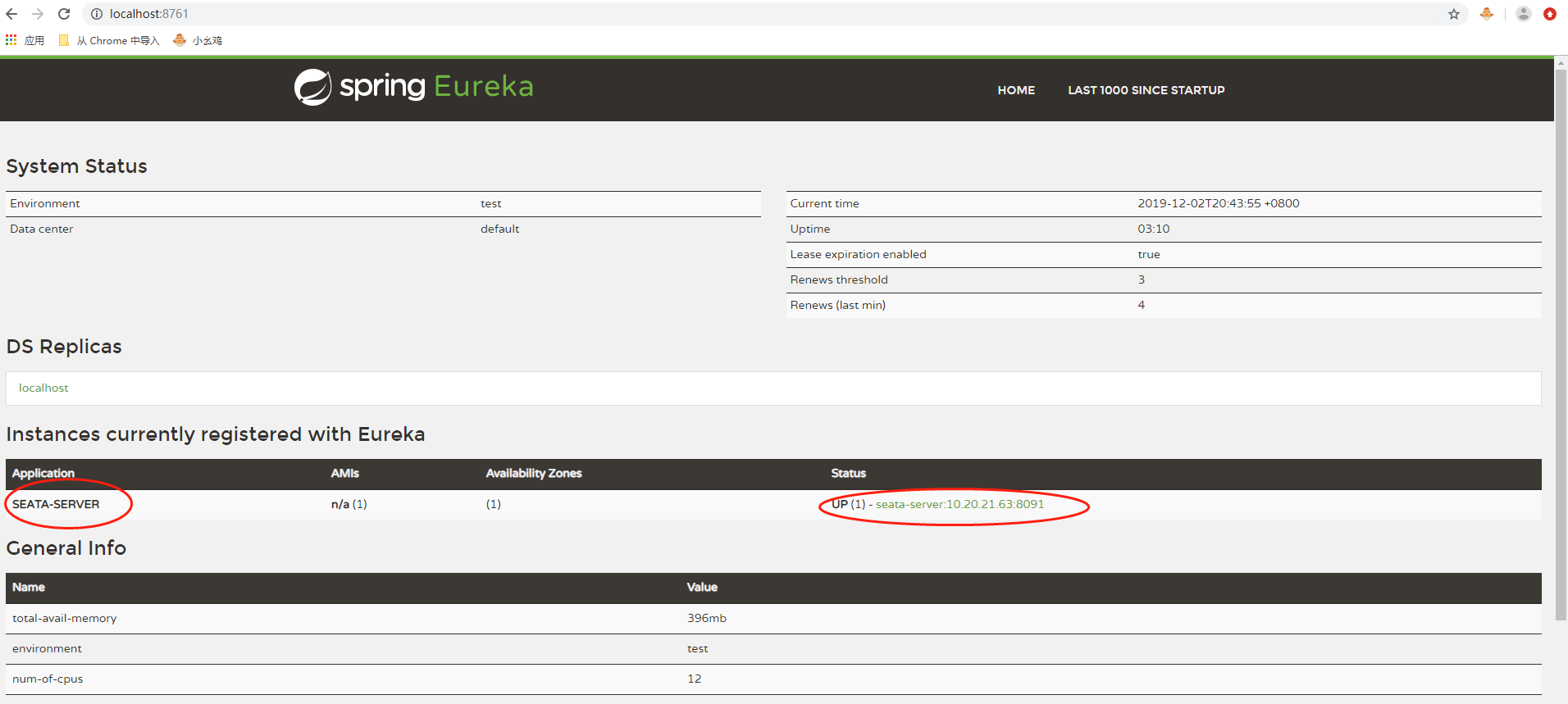
再启动seata-server，

windows下：打开cmd然后进入E:\seata-server-0.9.0\seata\bin文件夹下，输 入seata-server.bat执行：





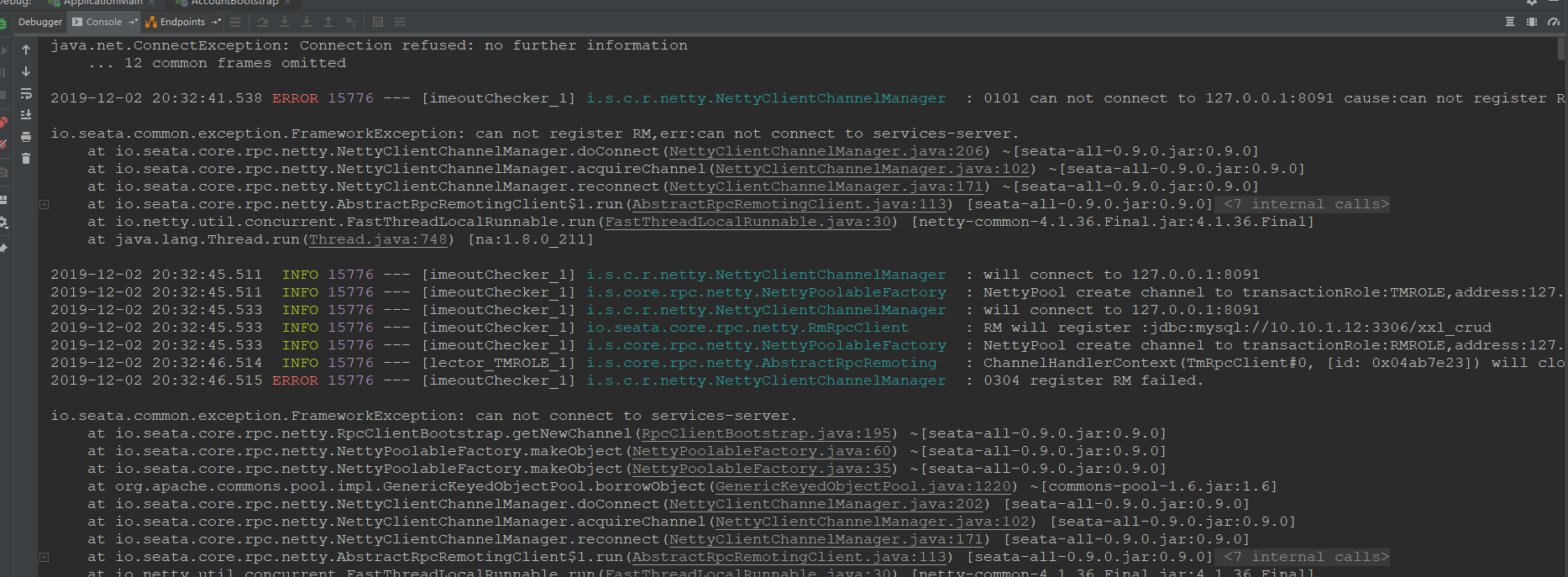
出现上图的结果表示seata-service启动成功，可以在服务注册中心验证，在浏览 器中输入[http://localhost:8761](http://localhost:8761/)，会看到有seata-server的服务：



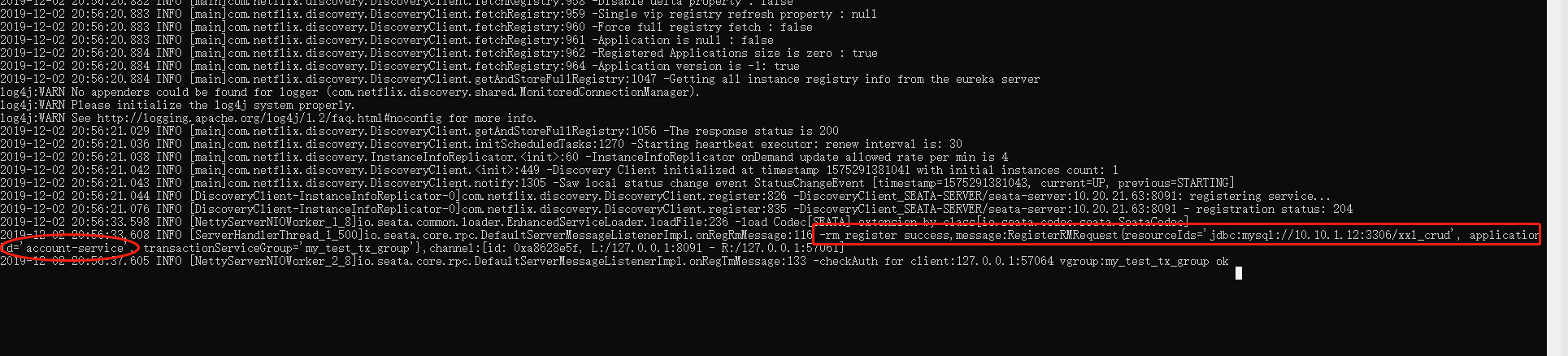
验证成功。

Linux下启动命令：sh seata-server.sh -p 8091 -h 127.0.0.1 -m file

然后依次启动服务：account-server、storage-service、order-service、 business-service。如果seata-service没启动成功，启动业务服务的 时候，虽然能启动成功，但是会报错：



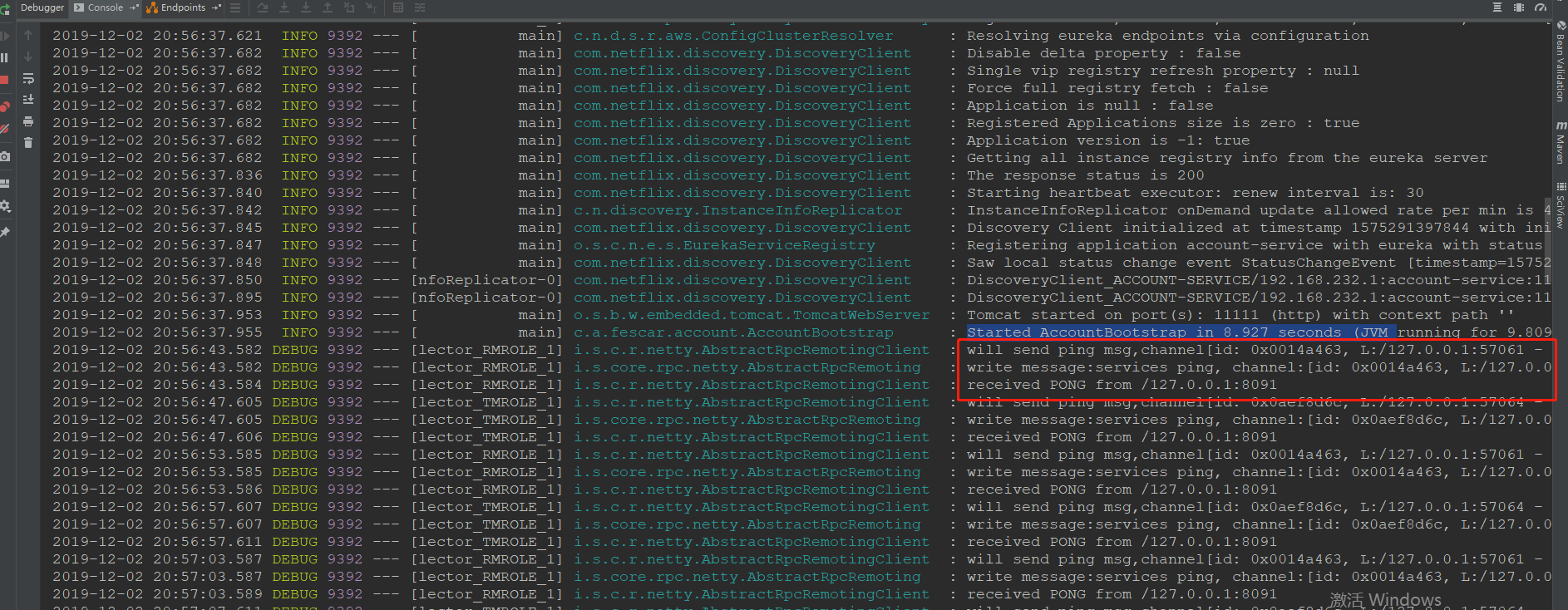
启动业务服务成功时，会注册时seata-server服务中，同时 seata-service会有日志打印：



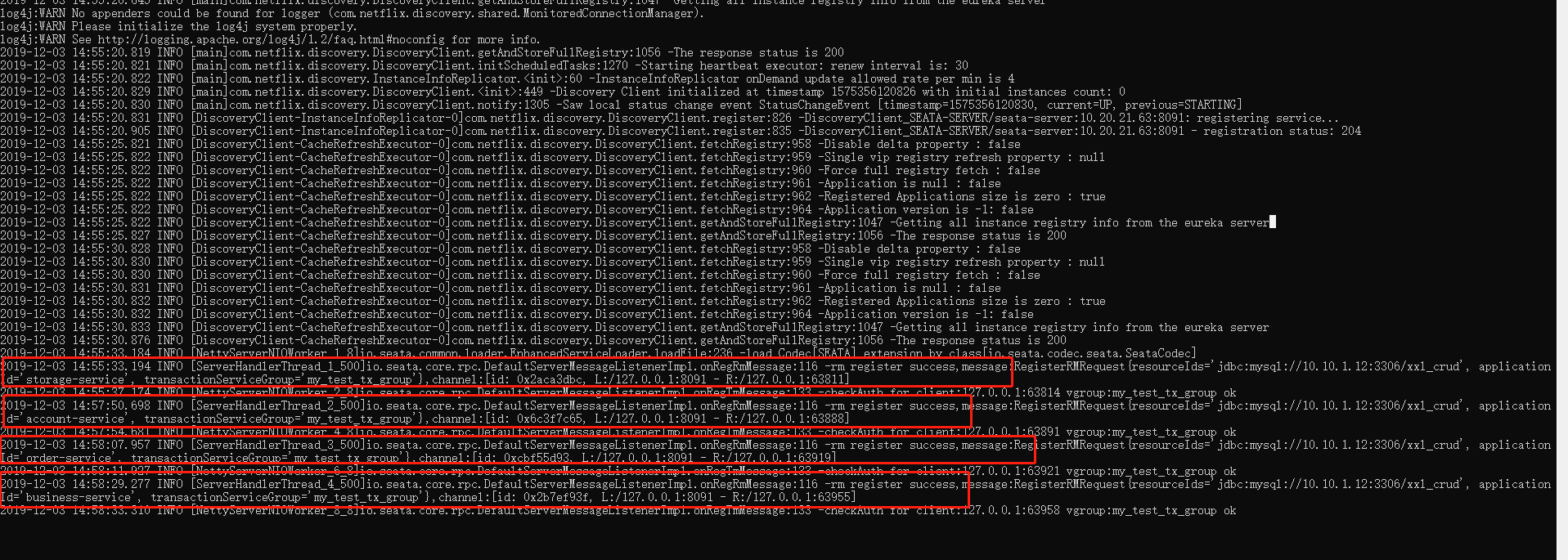
出现了上图中标记的日志才算业务服务启动成功（4个服务都是）

2019-12-02 20:56:33.608 INFO [ServerHandlerThread\_1\_500]io.seata.core.rpc.DefaultServerMessageListenerImpl.onRegRmMessage:116 -rm register success,message:RegisterRMRequest{resourceIds='jdbc:mysql://10.10.1.12:3306/xxl\_crud', applicationId='account-service', transactionServiceGroup='my\_test\_tx\_group'},channel:[id: 0xa8628e5f, L:/127.0.0.1:8091 - R:/127.0.0.1:57061]

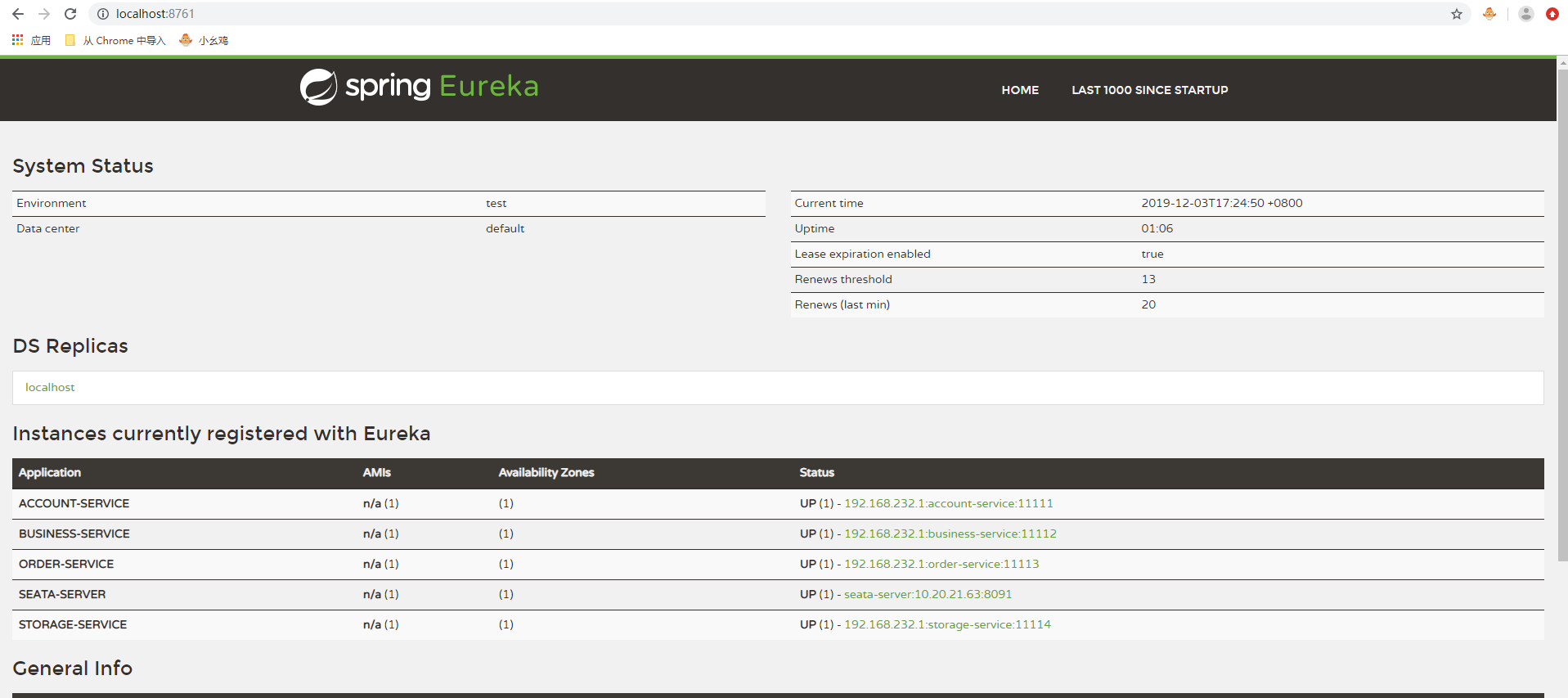
业务服务启动的时候也同样有日志显示注册到seata-service成功：



四个服务启动成功之后日志如下：



查看注册中心：



1. 开始验证分布式事务

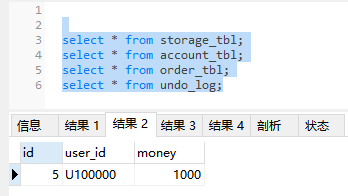
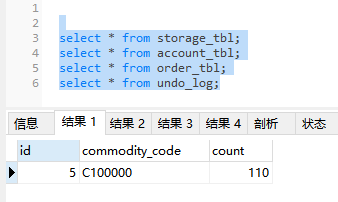
首先查询数据库的初始数据：

select \* from storage\_tbl;

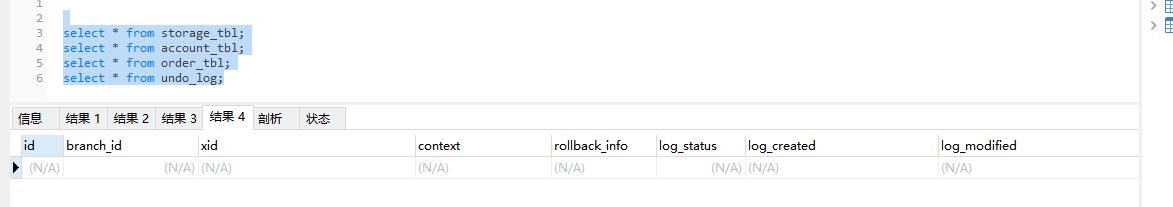
select \* from account\_tbl;

select \* from order\_tbl;

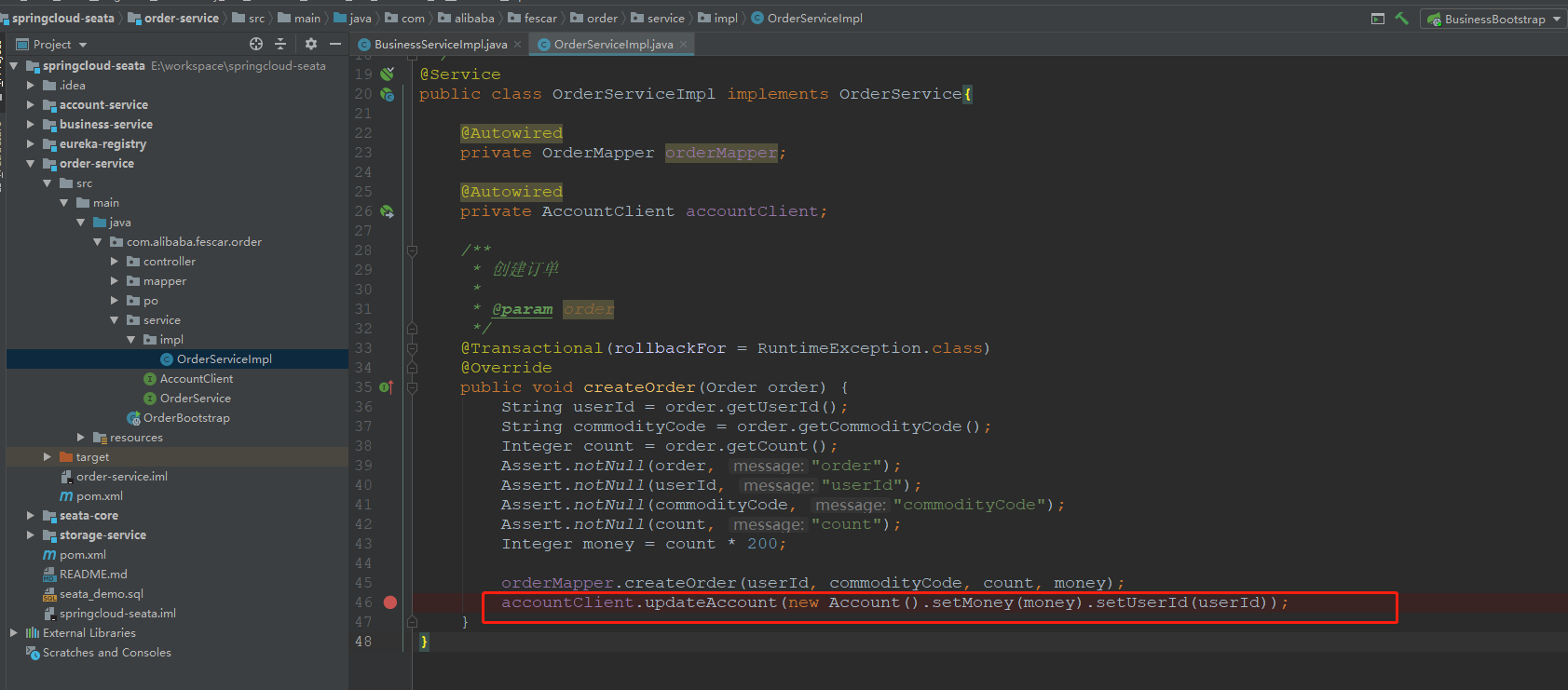
select \* from undo\_log;



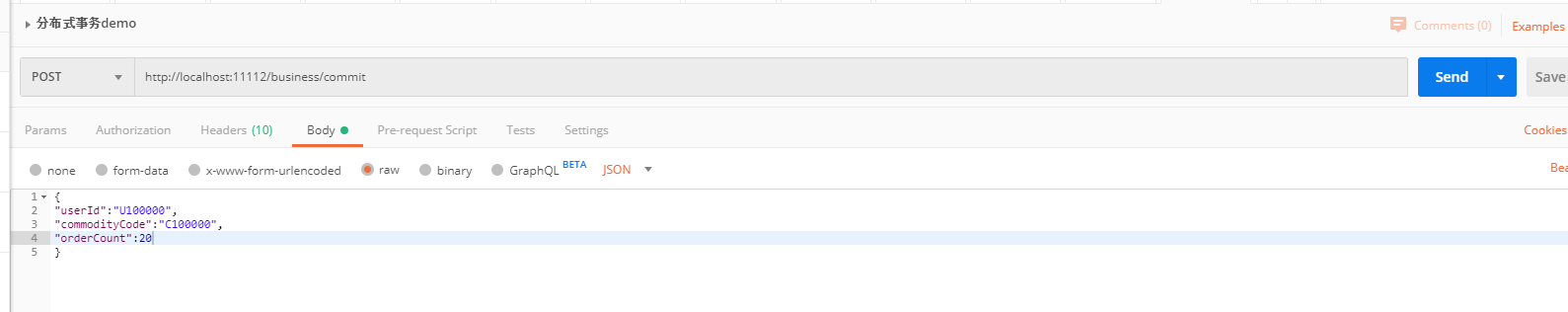




在程序中打断点：



用postman调服务：



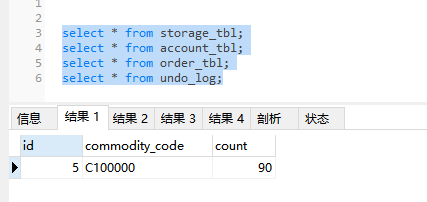
看查询数据库：

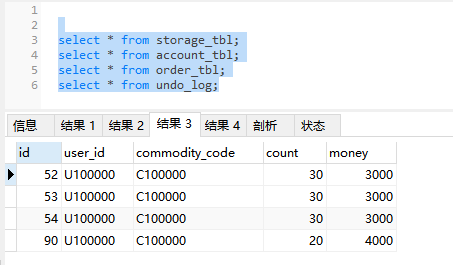
select \* from storage\_tbl;

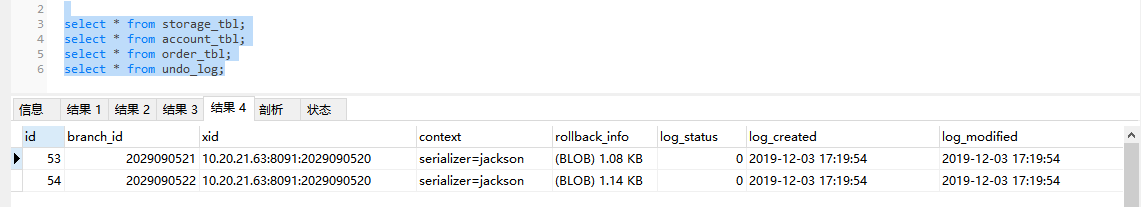
select \* from account\_tbl;

select \* from order\_tbl;

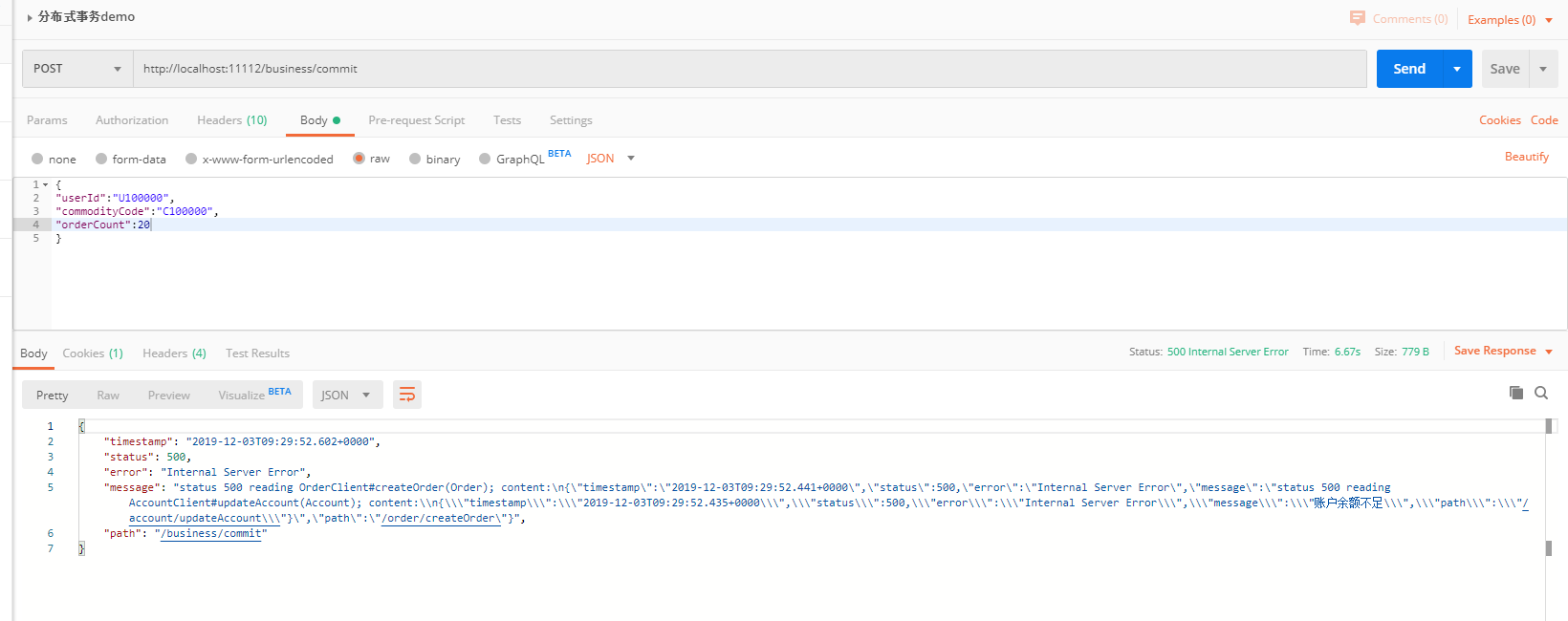
select \* from undo\_log;



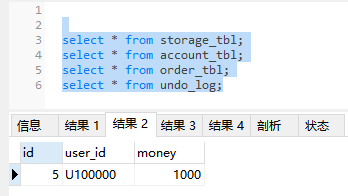
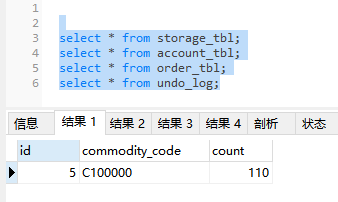




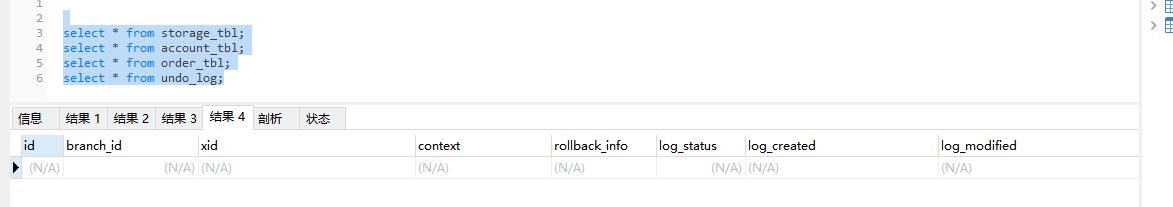
代码执行完之后，查看返回结果



再次查询数据库：







可以看到返回结果为账户余额不足，执行完数据库的数据和执行之前保持一致，表示事务成功回滚，到此springcloud-seata的分布式事务Demo完成。

注意：

Seata(AT模式)的默认全局事务隔离级别时读未提交****Read Uncommitted）****，如果在特点场景下要求全局的读已提交****（Read Committed）****，可以通过sql的select for update。

Seata参考文档：

<http://seata.io/zh-cn/docs/user/quickstart.html>