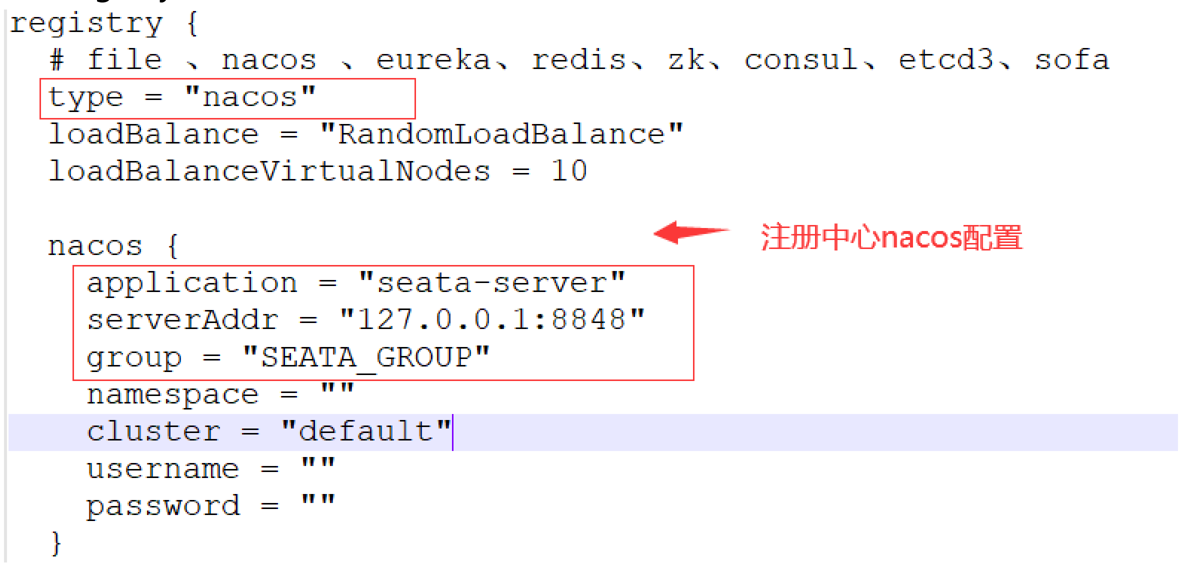
主讲老师：Fox

1. 启动Seata Server

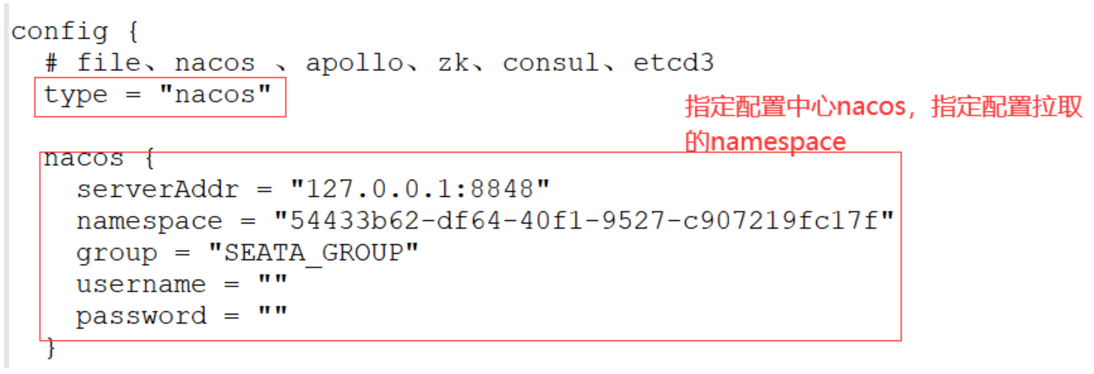
1.1 环境准备

1）指定nacos作为配置中心和注册中心

修改registry.conf文件



```
registry {
  # file 、nacos 、eureka、redis、zk、consul、etcd3、sofa
  type = "nacos"
  loadBalance = "RandomLoadBalance"
  loadBalanceVirtualNodes = 10

  nacos {
    application = "seata-server"
    serverAddr = "127.0.0.1:8848"
    group = "SEATA_GROUP"
    namespace = ""
    cluster = "default"
    username = ""
    password = ""
  }
```

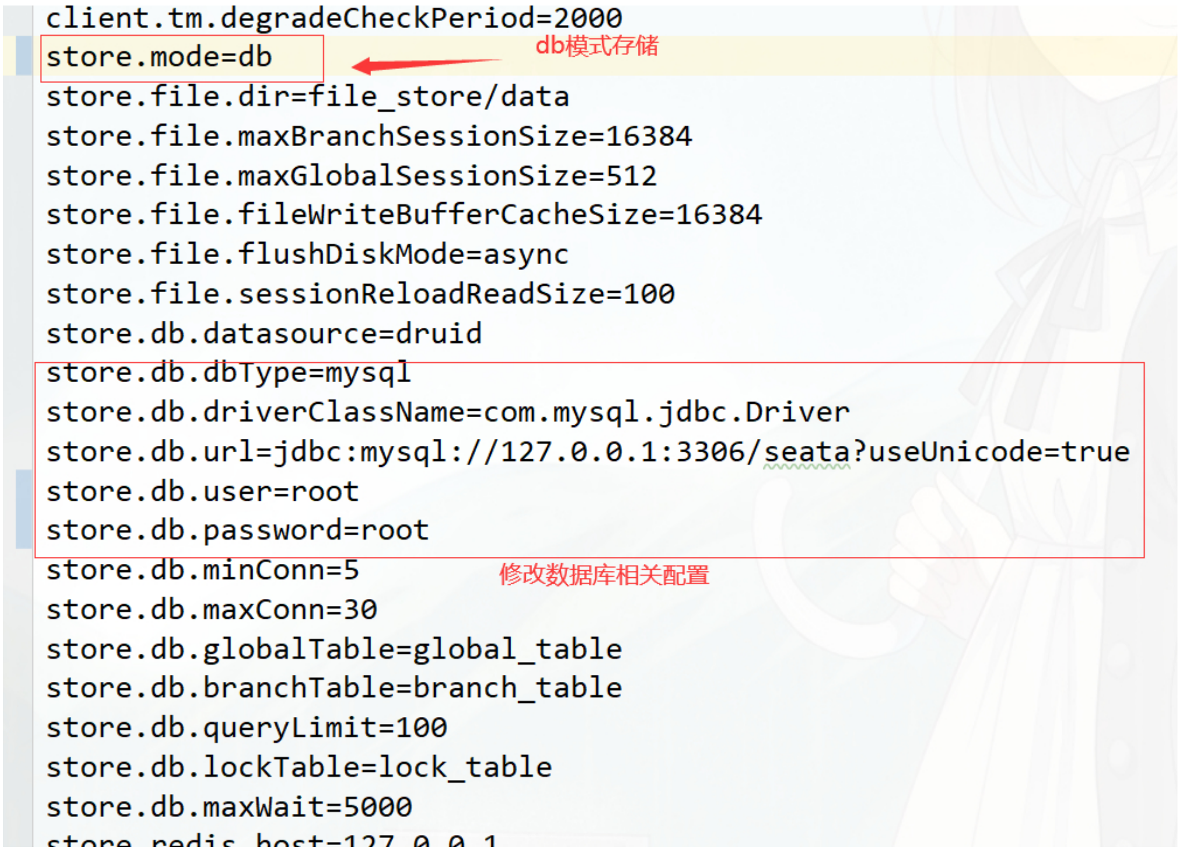


```
config {
  # file、nacos 、apollo、zk、consul、etcd3
  type = "nacos"

  nacos {
    serverAddr = "127.0.0.1:8848"
    namespace = "54433b62-df64-40f1-9527-c907219fc17f"
    group = "SEATA_GROUP"
    username = ""
    password = ""
  }
```

注意：客户端配置registry.conf使用nacos时也要注意group要和seata server中的group一致，默认group是"DEFAULT_GROUP"

2）同步seata server的配置到nacos

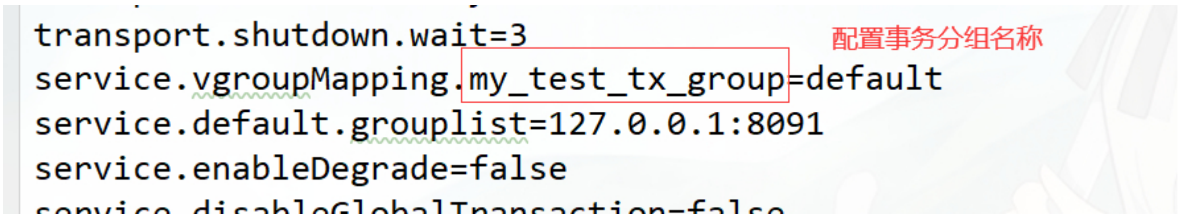
获取/seata/script/config-center/config.txt，修改配置信息



```
client.tm.degradeCheckPeriod=2000
store.mode=db
store.file.dir=file_store/data
store.file.maxBranchSessionSize=16384
store.file.maxGlobalSessionSize=512
store.file.fileWriteBufferCacheSize=16384
store.file.flushDiskMode=async
store.file.sessionReloadReadSize=100
store.db.datasource=druid
store.db.dbType=mysql
store.db.driverClassName=com.mysql.jdbc.Driver
store.db.url=jdbc:mysql://127.0.0.1:3306/seata?useUnicode=true
store.db.user=root
store.db.password=root
store.db.minConn=5
store.db.maxConn=30
store.db.globalTable=global_table
store.db.branchTable=branch_table
store.db.queryLimit=100
store.db.lockTable=lock_table
store.db.maxWait=5000
store.redis.host=127.0.0.1
```

配置事务分组， 要与客户端配置的事务分组一致

（客户端properties配置：spring.cloud.alibaba.seata.tx-service-group=my_test_tx_group）



```
transport.shutdown.wait=3
service.vgroupMapping.my_test_tx_group=default
service.default.grouplist=127.0.0.1:8091
service.enableDegrade=false
service.disableGlobalTransaction=false
```

配置参数同步到Nacos

shell:

```
sh ${SEATAPATH}/script/config-center/nacos/nacos-config.sh -h localhost -p 8848 -g SEATA_GROUP -t 5a3c7d6c-f497-4d68-a71a-2e5e3340b3ca
```

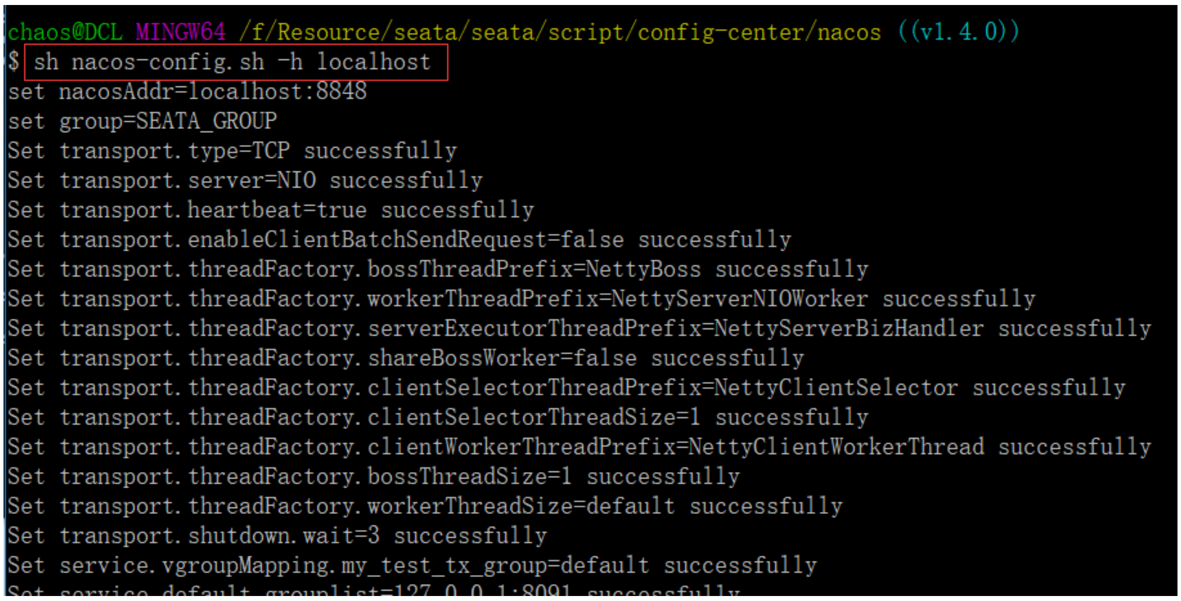
参数说明：

-h: host，默认值 localhost

-p: port，默认值 8848

-g: 配置分组，默认值为 'SEATA_GROUP'

-t: 租户信息，对应 Nacos 的命名空间ID字段, 默认值为空 ''



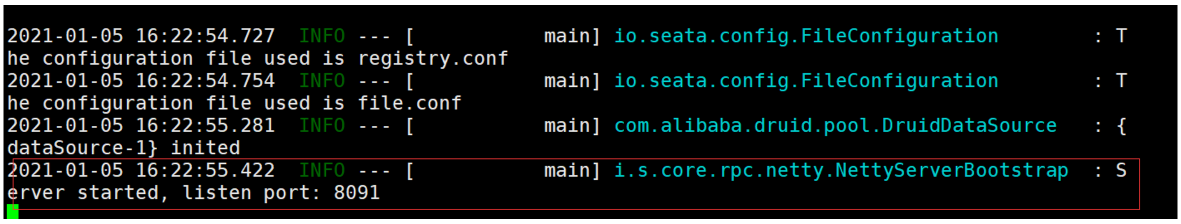
```
chaos@DCL MINGW64 /f/Resource/seata/seata/script/config-center/nacos ((v1.4.0))
$ sh nacos-config.sh -h localhost
set nacosAddr=localhost:8848
set group=SEATA_GROUP
Set transport.type=TCP successfully
Set transport.server=NIO successfully
Set transport.heartbeat=true successfully
Set transport.enableClientBatchSendRequest=false successfully
Set transport.threadFactory.bossThreadPrefix=NettyBoss successfully
Set transport.threadFactory.workerThreadPrefix=NettyServerNIOWorker successfully
Set transport.threadFactory.serverExecutorThreadPrefix=NettyServerBizHandler successfully
Set transport.threadFactory.shareBossWorker=false successfully
Set transport.threadFactory.clientSelectorThreadPrefix=NettyClientSelector successfully
Set transport.threadFactory.clientSelectorThreadSize=1 successfully
Set transport.threadFactory.clientWorkerThreadPrefix=NettyClientWorkerThread successfully
Set transport.threadFactory.bossThreadSize=1 successfully
Set transport.threadFactory.workerThreadSize=default successfully
Set transport.shutdown.wait=3 successfully
Set service.vgroupMapping.my_test_tx_group=default successfully
Set service.default.grouplist=127.0.0.1:8091 successfully
```

3） 启动Seata Server

启动Seata Server命令

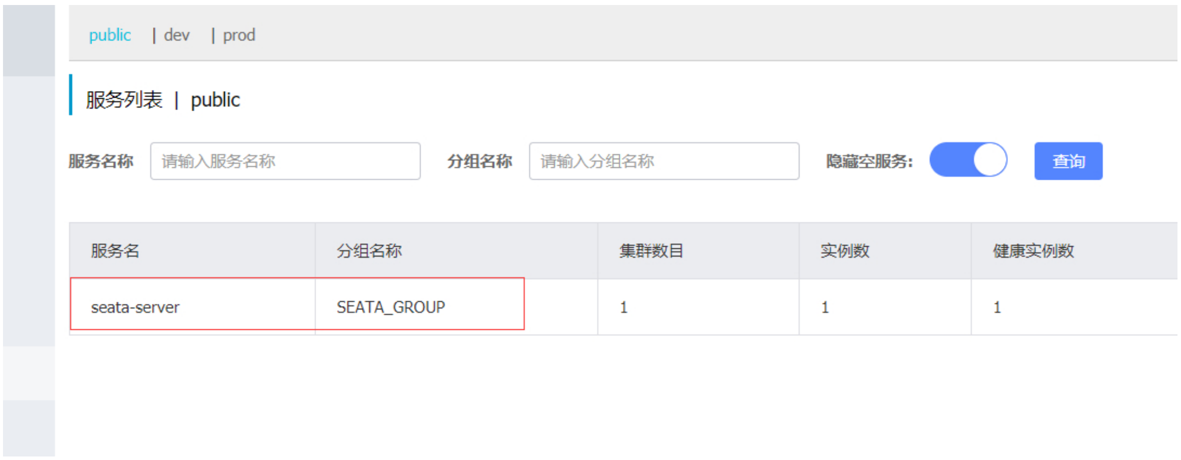
```
bin/seata-server.sh
```

启动成功，默认端口8091



```
2021-01-05 16:22:54.727  INFO --- [           main] io.seata.config.FileConfiguration        : T
he configuration file used is registry.conf
2021-01-05 16:22:54.754  INFO --- [           main] io.seata.config.FileConfiguration        : T
he configuration file used is file.conf
2021-01-05 16:22:55.281  INFO --- [           main] com.alibaba.druid.pool.DruidDataSource   : {
dataSource-1} inited
2021-01-05 16:22:55.422  INFO --- [           main] i.s.core.rpc.netty.NettyServerBootstrap  : S
erver started, listen port: 8091
```

在注册中心中可以查看到seata-server注册成功



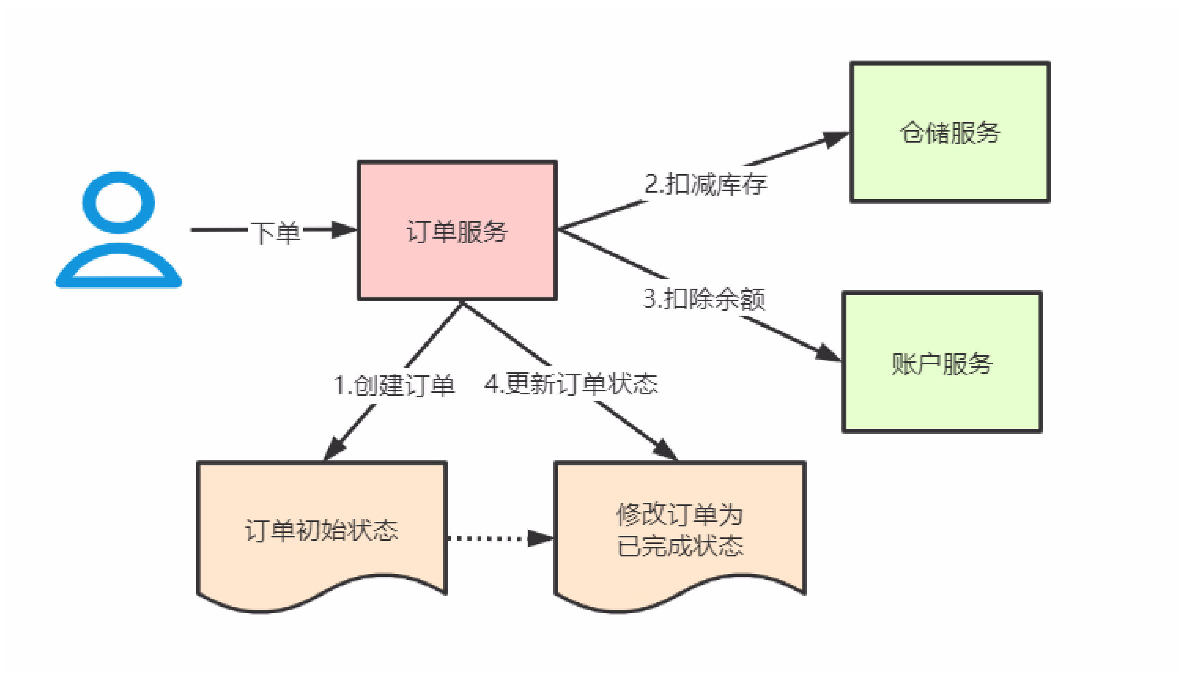
服务名	分组名称	集群数目	实例数	健康实例数
seata-server	SEATA_GROUP	1	1	1

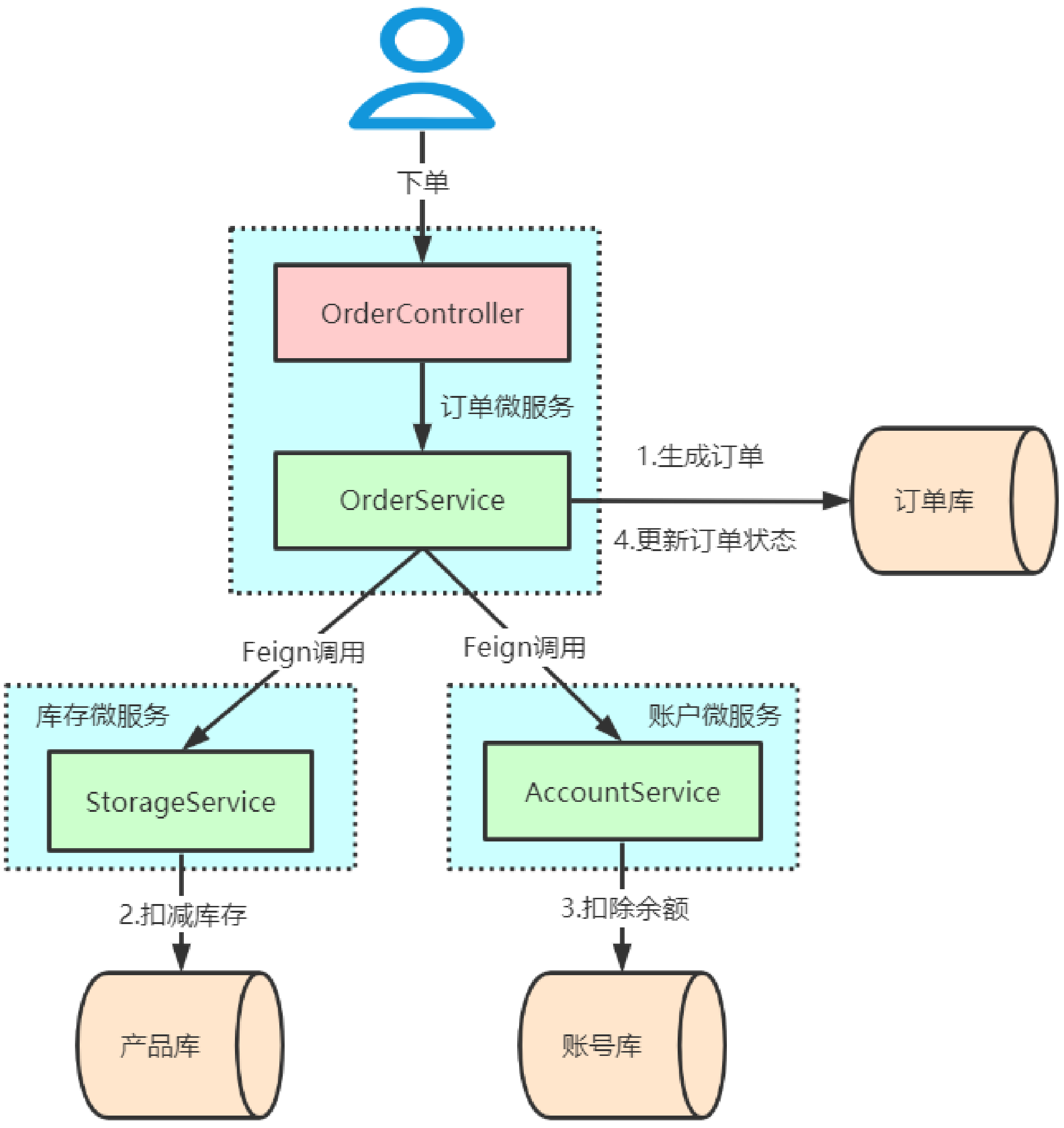
2. Seata如何整合到Spring Cloud微服务

业务场景：

用户下单，整个业务逻辑由三个微服务构成：

- 仓储服务：对给定的商品扣除库存数量。
- 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。


下单
订单服务
2.扣减库存
仓储服务
3.扣除余额
账户服务
1.创建订单
4.更新订单状态
订单初始状态
修改订单为已完成状态


下单
OrderController
订单微服务
OrderService
1.生成订单
4.更新订单状态
订单库
Feign调用
Feign调用
库存微服务
StorageService
账户微服务
AccountService
2.扣减库存
3.扣除余额
产品库
账号库

环境准备：

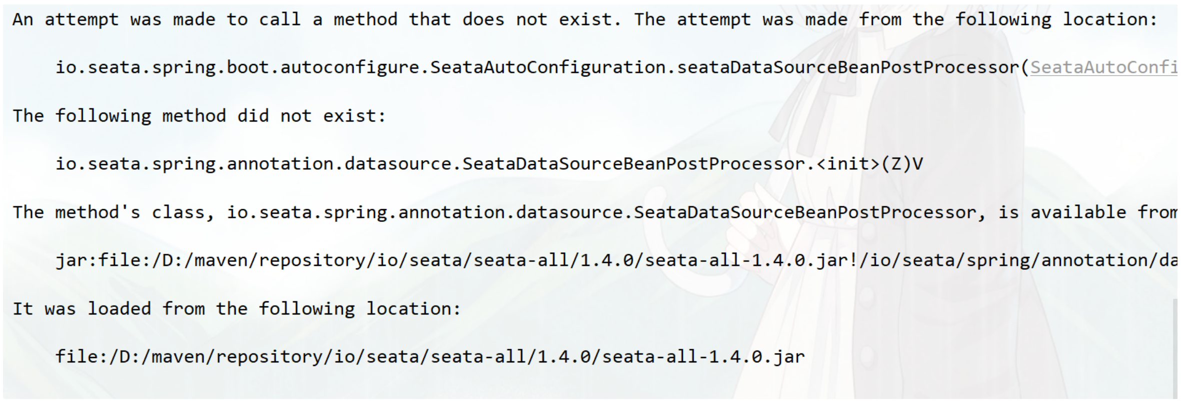
seata： v1.4.0

spring cloud&spring cloud alibaba:

```
<spring-cloud.version>Greenwich.SR3</spring-cloud.version>
<spring-cloud-alibaba.version>2.1.1.RELEASE</spring-cloud-alibaba.version>
```

注意版本选择问题：

spring cloud alibaba 2.1.2 及其以上版本使用seata1.4.0会出现如下异常 （支持seata 1.3.0)



```
An attempt was made to call a method that does not exist. The attempt was made from the following location:

    io.seata.spring.boot.autoconfigure.SeataAutoConfiguration.seataDataSourceBeanPostProcessor(SeataAutoConfi

The following method did not exist:

    io.seata.spring.annotation.datasource.SeataDataSourceBeanPostProcessor.<init>(Z)V

The method's class, io.seata.spring.annotation.datasource.SeataDataSourceBeanPostProcessor, is available from

    jar:file:/D:/maven/repository/io/seata/seata-all/1.4.0/seata-all-1.4.0.jar!/io/seata/spring/annotation/da

It was loaded from the following location:

    file:/D:/maven/repository/io/seata/seata-all/1.4.0/seata-all-1.4.0.jar
```

2.1 导入依赖

```
<!-- seata-->
<dependency>
 <groupId>com.alibaba.cloud</groupId>
 <artifactId>spring-cloud-starter-alibaba-seata</artifactId>
 <exclusions>
 <exclusion>
 <groupId>io.seata</groupId>
 <artifactId>seata-all</artifactId>
 </exclusion>
  </exclusions>
</dependency>
<dependency>
  <groupId>io.seata</groupId>
  <artifactId>seata-all</artifactId>
  <version>1.4.0</version>
</dependency>

<!--nacos 注册中心-->
<dependency>
  <groupId>com.alibaba.cloud</groupId>
  <artifactId>spring-cloud-starter-alibaba-nacos-discovery</artifactId>
</dependency>

<dependency>
```

```xml
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-starter-openfeign</artifactId>
 </dependency>

 <dependency>
  <groupId>com.alibaba</groupId>
  <artifactId>druid-spring-boot-starter</artifactId>
  <version>1.1.21</version>
 </dependency>

 <dependency>
  <groupId>mysql</groupId>
  <artifactId>mysql-connector-java</artifactId>
  <scope>runtime</scope>
  <version>8.0.16</version>
 </dependency>

 <dependency>
  <groupId>org.mybatis.spring.boot</groupId>
  <artifactId>mybatis-spring-boot-starter</artifactId>
  <version>2.1.1</version>
 </dependency>
```

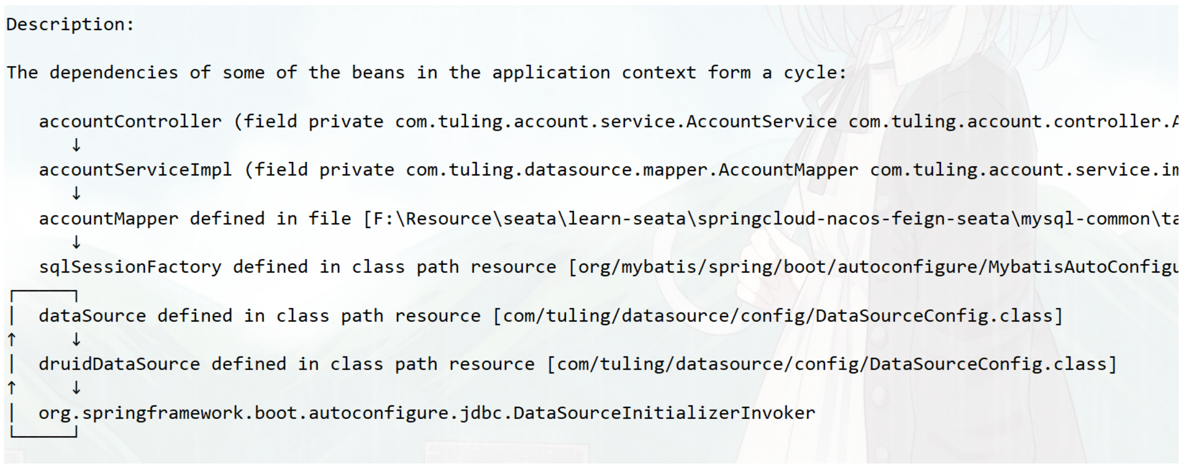
2.2 微服务对应数据库中添加undo_log表

```sql
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,
  PRIMARY KEY (`id`),
  UNIQUE KEY `ux_undo_log` (`xid`,`branch_id`)
 ) ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT CHARSET=utf8;
```

2.3 微服务需要使用seata DataSourceProxy代理自己的数据源

```java
/**
 * @author Fox
 *
 * 需要用到分布式事务的微服务都需要使用seata DataSourceProxy代理自己的数据源
 */
@Configuration
@MapperScan("com.tuling.datasource.mapper")
public class MybatisConfig {

  /**
  * 从配置文件获取属性构造datasource，注意前缀，这里用的是druid，根据自己情况配置,
  * 原生datasource前缀取"spring.datasource"
  *
  * @return
  */
  @Bean
  @ConfigurationProperties(prefix = "spring.datasource")
  public DataSource druidDataSource() {
  DruidDataSource druidDataSource = new DruidDataSource();
  return druidDataSource;
  }

  /**
  * 构造datasource代理对象，替换原来的datasource
  * @param druidDataSource
  * @return
  */
  @Primary
  @Bean("dataSource")
  public DataSourceProxy dataSourceProxy(DataSource druidDataSource) {
  return new DataSourceProxy(druidDataSource);
  }


  @Bean(name = "sqlSessionFactory")
  public SqlSessionFactory sqlSessionFactoryBean(DataSourceProxy dataSourceProxy) throws Exception {
  SqlSessionFactoryBean factoryBean = new SqlSessionFactoryBean();
  //设置代理数据源
  factoryBean.setDataSource(dataSourceProxy);
  ResourcePatternResolver resolver = new PathMatchingResourcePatternResolver();
  factoryBean.setMapperLocations(resolver.getResources("classpath*:mybatis/**/*-mapper.xml"));

  org.apache.ibatis.session.Configuration configuration=new org.apache.ibatis.session.Configuration();
  //使用jdbc的getGeneratedKeys获取数据库自增主键值
  configuration.setUseGeneratedKeys(true);
  //使用列别名替换列名
  configuration.setUseColumnLabel(true);
  //自动使用驼峰命名属性映射字段，如userId ---> user_id
  configuration.setMapUnderscoreToCamelCase(true);
  factoryBean.setConfiguration(configuration);

  return factoryBean.getObject();
  }

 }
```

注意： 启动类上需要排除DataSourceAutoConfiguration，否则会出现循环依赖的问题



```
Description:

The dependencies of some of the beans in the application context form a cycle:

   accountController (field private com.tuling.account.service.AccountService com.tuling.account.controller.A
      ↓
   accountServiceImpl (field private com.tuling.datasource.mapper.AccountMapper com.tuling.account.service.im
      ↓
   accountMapper defined in file [F:\Resource\seata\learn-seata\springcloud-nacos-feign-seata\mysql-common\ta
      ↓
   sqlSessionFactory defined in class path resource [org/mybatis/spring/boot/autoconfigure/MybatisAutoConfigu
┌─────┐
|  dataSource defined in class path resource [com/tuling/datasource/config/DataSourceConfig.class]
↑     ↓
|  druidDataSource defined in class path resource [com/tuling/datasource/config/DataSourceConfig.class]
↑     ↓
|  org.springframework.boot.autoconfigure.jdbc.DataSourceInitializerInvoker
└─────┘
```

启动类排除DataSourceAutoConfiguration.class

```java
@SpringBootApplication(scanBasePackages = "com.tuling",exclude = DataSourceAutoConfiguration.class)
public class AccountServiceApplication {

 public static void main(String[] args) {
 SpringApplication.run(AccountServiceApplication.class, args);
 }

}
```

4. 添加seata的配置

1）将registry.conf文件拷贝到resources目录下，指定注册中心和配置中心都是nacos

```
registry {
 # file 、nacos 、eureka、redis、zk、consul、etcd3、sofa
 type = "nacos"

nacos {
serverAddr = "192.168.65.232:8848"
namespace = ""
cluster = "default"
group = "SEATA_GROUP"
 }
}

config {
 # file、nacos 、apollo、zk、consul、etcd3、springCloudConfig
 type = "nacos"

 nacos {
 serverAddr = "192.168.65.232:8848"
 namespace = "29ccf18e-e559-4a01-b5d4-61bad4a89ffd"
 group = "SEATA_GROUP"
 }
}
```

> 在 `*org.springframework.cloud:spring-cloud-starter-alibaba-seata* 的 `*org.springframework.cloud.alibaba.seata.GlobalTransactionAutoConfiguration*` 类中，默认会使用 `*${spring.application.name}-seata-service-group* 作为服务名注册到 Seata Server上，如果和service.vgroup_mapping配置不一致，会提示 `*no available server to connect* 错误
>
> 也可以通过配置 `*spring.cloud.alibaba.seata.tx-service-group* 修改后缀，但是必须和 `*file.conf* 中的配置保持一致

2）在yml中指定事务分组（和配置中心的service.vgroup_mapping 配置一一对应）

```
spring:
 application:
 name: account-service
 cloud:
 nacos:
 discovery:
 server-addr: 127.0.0.1:8848
 alibaba:
 seata:
  tx-service-group:
  my_test_tx_group # seata 服务事务分组
```

> 参考源码：
> io.seata.core.rpc.netty.NettyClientChannelManager#getAvailServerList
> 》NacosRegistryServiceImpl#lookup

> 》String clusterName = getServiceGroup(key); #获取seata server集群名称
> 》List<Instance> firstAllInstances = getNamingInstance().getAllInstances(getServiceName(), getServiceGroup(), clusters)

spring cloud alibaba 2.1.4 之后支持yml中配置seata属性，可以用来替换registry.conf文件

配置支持实现在seata-spring-boot-starter.jar中，也可以引入依赖

```xml
<dependency>
 <groupId>io.seata</groupId>
 <artifactId>seata-spring-boot-starter</artifactId>
 <version>1.4.0</version>
</dependency>
```

在yml中配置

```yaml
seata:
 # seata 服务分组，要与服务端nacos-config.txt中service.vgroup_mapping的后缀对应
 tx-service-group: my_test_tx_group
 registry:
 # 指定nacos作为注册中心
 type: nacos
 nacos:
 server-addr: 127.0.0.1:8848
 namespace: ""
  group: SEATA_GROUP

  config:
  # 指定nacos作为配置中心
  type: nacos
  nacos:
  server-addr: 127.0.0.1:8848
  namespace: "54433b62-df64-40f1-9527-c907219fc17f"
  group: SEATA_GROUP
```

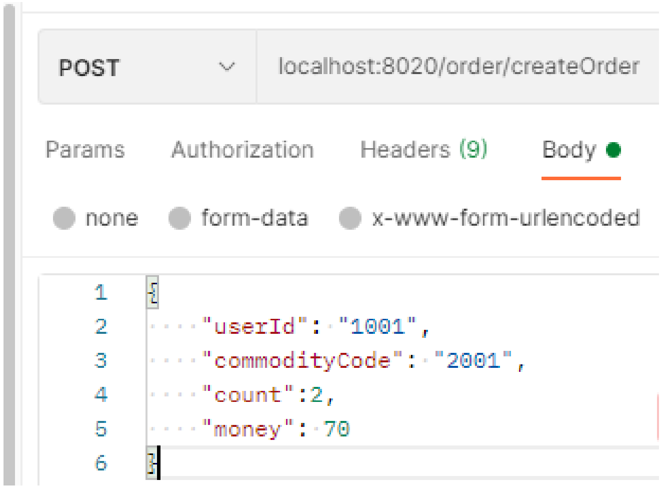
3） 在事务发起者中添加@GlobalTransactional注解

核心代码

```java
@Override
//@Transactional
@GlobalTransactional(name="createOrder")
public Order saveOrder(OrderVo orderVo){
 log.info("=============用户下单=================");
 log.info("当前 XID: {}", RootContext.getXID());

// 保存订单
Order order = new Order();
 order.setUserId(orderVo.getUserId());
 order.setCommodityCode(orderVo.getCommodityCode());
 order.setCount(orderVo.getCount());
 order.setMoney(orderVo.getMoney());
 order.setStatus(OrderStatus.INIT.getValue());

 Integer saveOrderRecord = orderMapper.insert(order);
 log.info("保存订单{}", saveOrderRecord > 0 ? "成功" : "失败");

 //扣减库存
 storageFeignService.deduct(orderVo.getCommodityCode(),orderVo.getCount());

 //扣减余额
 accountFeignService.debit(orderVo.getUserId(),orderVo.getMoney());

 //更新订单
 Integer updateOrderRecord = orderMapper.updateOrderStatus(order.getId(),OrderStatus.SUCCESS.getValue());
 log.info("更新订单id:{} {}", order.getId(), updateOrderRecord > 0 ? "成功" : "失败");

 return order;

}
```

4）测试分布式事务是否生效

用户下单账户余额不足，库存是否回滚



localhost:8020/order/createOrder

```
{
    "userId": "1001",
    "commodityCode": "2001",
    "count":2,
    "money": 70
}
```

文档：14-1 Spring Cloud Alibaba整合Seata实...

链接：http://note.youdao.com/noteshare?id=173e30d8c458359fb8756be9a15a891d&sub=620FF34B3AB749A093E78BC5AB6D5483