

## Spring Cloud

- 1 微服务是什么
- 2 Spring Cloud是什么
- 3 Spring Cloud Eureka
- 4 Spring Cloud Ribbon
- 5 Spring Cloud OpenFeign
- 6 Spring Cloud Hystrix
- 7 Spring Cloud Gateway
- 8 Spring Cloud Config
- 9 Spring Cloud Alibaba是什么
- 10 Spring Cloud Alibaba Nacos
- 11 Spring Cloud Alibaba Sentinel
- 12 Spring Cloud Alibaba Seata

首页 > Spring Cloud

## Seata: Spring Cloud Alibaba分布式事务组件 (非常详细)

< 上一节

下一节 >

随着业务的不断发展，单体架构已经无法满足我们的需求，分布式微服务架构逐渐成为大型互联网平台的首选，但所有使用分布式微服务架构的应用都必须面临一个十分棘手的问题，那就是“分布式事务”问题。

在分布式微服务架构中，几乎所有业务操作都需要多个服务协作才能完成。对于其中的某个服务而言，它的数据一致性可以由其自身数据库事务来保证，但从整个分布式微服务架构来看，其全局数据的一致性却是无法保证的。

例如，用户在某电商系统下单购买了一件商品后，电商系统会执行下 4 步：

1. 调用订单服务创建订单数据
2. 调用库存服务扣减库存
3. 调用账户服务扣减账户金额
4. 最后调用订单服务修改订单状态

为了保证数据的正确性和一致性，我们必须保证所有这些操作要么全部成功，要么全部失败，否则就可能出现类似于商品库存已扣减，但用户账户资金尚未扣减的情况。各服务自身的事务特性显然是无法实现这一目标的，此时，我们可以通过分布式事务框架来解决这个问题。

Seata 就是这样一个分布式事务处理框架，它是由阿里巴巴和蚂蚁金服共同开源的分布式事务解决方案，能够在微服务架构下提供高性能且简单易用的分布式事务服务。

## Seata 的发展历程

阿里巴巴作为国内最早一批进行应用分布式（微服务化）改造的企业，很早就遇到微服务架构下的分布式事务问题。

阿里巴巴对于分布式事务问题先后发布了以下解决方案：

- 2014 年，阿里中间件团队发布 TXC (Taobao Transaction Constructor)，为集团内应用提供分布式事务服务。
- 2016 年，TXC 在经过产品化改造后，以 GTS (Global Transaction Service) 的身份登陆阿里云，成为当时业界唯一一款云上分布式事务产品。在阿云里的公有云、专有云解决方案中，开始服务于众多外部客户。
- 2019 年起，基于 TXC 和 GTS 的技术积累，阿里中间件团队发起了开源项目 Fescar (Fast & EaSy Commit And Rollback, FESCAR)，和社区一起建设这个分布式事务解决方案。
- 2019 年 fescar 被重命名为了 seata (simple extensible autonomous transaction architecture)。
- TXC、GTS、Fescar 以及 seata 一脉相承，为解决微服务架构下的分布式事务问题交出了一份与众不同的答卷。

## 分布式事务相关概念

分布式事务主要涉及以下概念：

- **事务**：由一组操作构成的可靠、独立的工作单元，事务具备 ACID 的特性，即原子性、一致性、隔离性和持久性。
- **本地事务**：本地事务由本地资源管理器（通常指数据库管理系统 DBMS，例如 MySQL、Oracle 等）管理，严格地支持 ACID 特性，高效可靠。本地事务不具备分布式事务的处理能力，隔离的最小单位受限于资源管理器，即本地事务只能对自己数据库的操作进行控制，对于其他数据库的操作则无能为力。



- **全局事务**: 全局事务指的是一次性操作多个资源管理器完成的事务，由一组分支事务组成。
- **分支事务**: 在分布式事务中，就是一个个受全局事务管辖和协调的本地事务。

我们可以将分布式事务理解成一个包含了若干个分支事务的全局事务。全局事务的职责是协调其管辖的各个分支事务达成一致，要么一起成功提交，要么一起失败回滚。此外，通常分支事务本身就是一个满足 ACID 特性的本地事务。

## Seata 整体工作流程

Seata 对分布式事务的协调和控制，主要是通过 XID 和 3 个核心组件实现的。

### XID

XID 是全局事务的唯一标识，它可以在服务的调用链路中传递，绑定到服务的事务上下文中。

### 核心组件

Seata 定义了 3 个核心组件：

- TC (Transaction Coordinator) : 事务协调器，它是事务的协调者（这里指的是 Seata 服务器），主要负责维护全局事务和分支事务的状态，驱动全局事务提交或回滚。
- TM (Transaction Manager) : 事务管理器，它是事务的发起者，负责定义全局事务的范围，并根据 TC 维护的全局事务和分支事务状态，做出开始事务、提交事务、回滚事务的决议。
- RM (Resource Manager) : 资源管理器，它是资源的管理者（这里可以将其理解为各服务使用的数据库）。它负责管理分支事务上的资源，向 TC 注册分支事务，汇报分支事务状态，驱动分支事务的提交或回滚。

以上三个组件相互协作，TC 以 Seata 服务器 (Server) 形式独立部署，TM 和 RM 则是以 Seata Client 的形式集成在微服务中运行，其整体工作流程如下图。

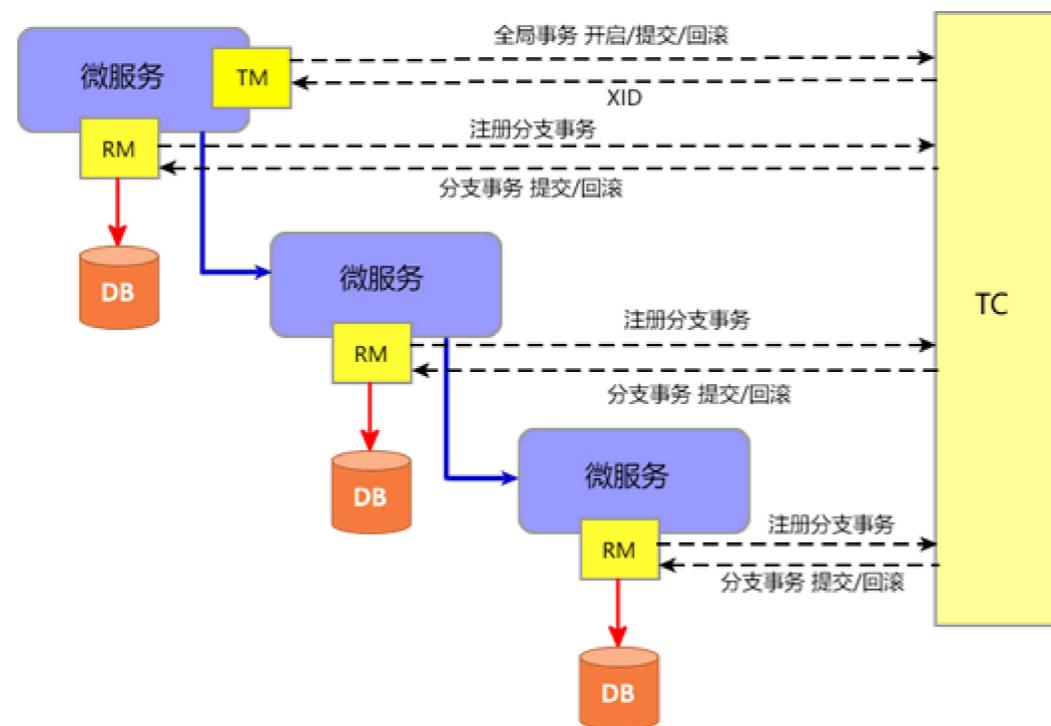


图1: Sentinel 的工作流程

Seata 的整体工作流程如下：

1. TM 向 TC 申请开启一个全局事务，全局事务创建成功后，TC 会针对这个全局事务生成一个全局唯一的 XID；
2. XID 通过服务的调用链传递到其他服务；



3. RM 向 TC 注册一个分支事务，并将其纳入 XID 对应全局事务的管辖；
4. TM 根据 TC 收集的各个分支事务的执行结果，向 TC 发起全局事务提交或回滚决议；
5. TC 调度 XID 下管辖的所有分支事务完成提交或回滚操作。

## Seata AT 模式

Seata 提供了 AT、TCC、SAGA 和 XA 四种事务模式，可以快速有效地对分布式事务进行控制。

在这四种事务模式中使用最多，最方便的就是 AT 模式。与其他事务模式相比，AT 模式可以应对大多数的业务场景，且基本可以做到无业务入侵，开发人员能够有更多的精力关注于业务逻辑开发。

### AT 模式的前提

任何应用想要使用 Seata 的 AT 模式对分布式事务进行控制，必须满足以下 2 个前提：

- 必须使用支持本地 ACID 事务特性的关系型数据库，例如 MySQL、Oracle 等；
- 应用程序必须是使用 JDBC 对数据库进行访问的 JAVA 应用。

此外，我们还需要针对业务中涉及的各个数据库表，分别创建一个 UNDO\_LOG（回滚日志）表。不同数据库在创建 UNDO\_LOG 表时会略有不同，以 MySQL 为例，其 UNDO\_LOG 表的创表语句如下：

```
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;
```

### AT 模式的工作机制

Seata 的 AT 模式工作时大致可以分为以两个阶段，下面我们就结合一个实例来对 AT 模式的工作机制进行介绍。

假设某数据库中存在一张名为 webset 的表，表结构如下。

| 列名   | 类型           | 主键 |
|------|--------------|----|
| id   | bigint(20)   | ✓  |
| name | varchar(255) |    |
| url  | varchar(255) |    |



在某次分支事务中，我们需要在 webset 表中执行以下操作。

```
update webset set url = 'c.biancheng.net' where name = 'C语言中文网';
```

## 一阶段

Seata AT 模式一阶段的工作流程如下图所示。

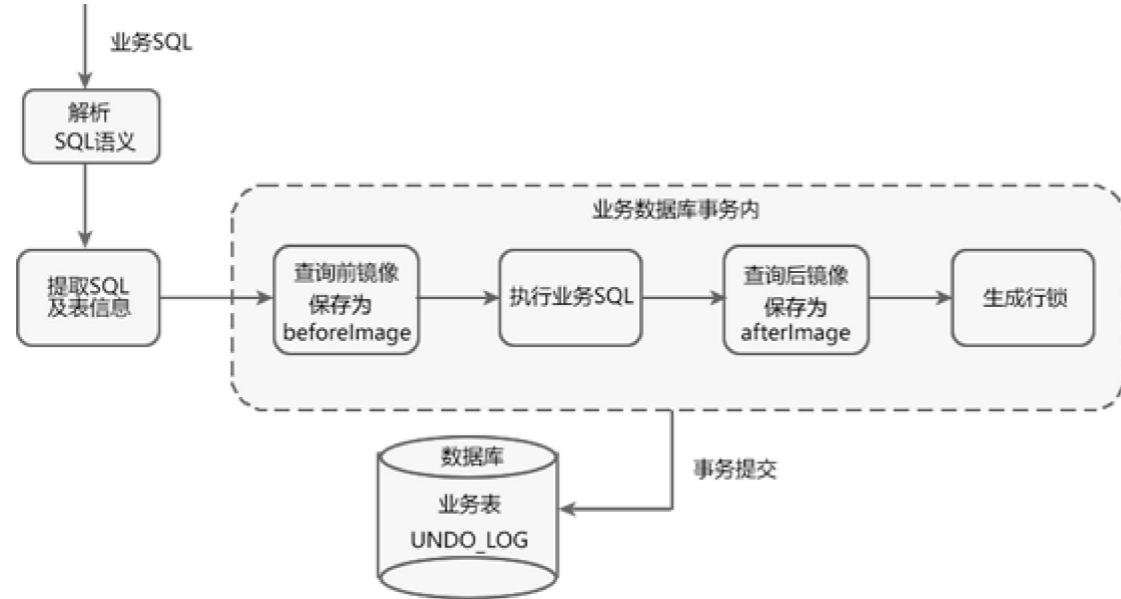


图2: Seata AT 模式一阶段

Seata AT 模式一阶段工作流程如下。

1. 获取 SQL 的基本信息: Seata 拦截并解析业务 SQL, 得到 SQL 的操作类型 (UPDATE) 、表名 (webset) 、判断条件 (where name = 'C语言中文网') 等相关信息。

2. 查询前镜像: 根据得到的业务 SQL 信息, 生成 “前镜像查询语句” 。

```
select id, name, url from webset where name='C语言中文网';
```

执行 “前镜像查询语句” , 得到即将执行操作的数据, 并将其保存为 “前镜像数据 (beforeImage) ” 。

| <b>id</b> | <b>name</b> | <b>url</b>    |
|-----------|-------------|---------------|
| 1         | C语言中文网      | biancheng.net |

3. 执行业务 SQL (update webset set url = 'c.biancheng.net' where name = 'C语言中文网') , 将这条记录的 url 修改为 c.biancheng.net。

4. 查询后镜像: 根据 “前镜像数据”的主键 (id : 1) , 生成 “后镜像查询语句” 。

```
select id, name, url from webset where id= 1;
```

执行 “后镜像查询语句” , 得到执行业务操作后的数据, 并将其保存为 “后镜像数据 (afterImage) ” 。



| <b>id</b> | <b>name</b> | <b>url</b>      |
|-----------|-------------|-----------------|
| 1         | C语言中文网      | c.biancheng.net |

5. 插入回滚日志：将前后镜像数据和业务 SQL 的信息组成一条回滚日志记录，插入到 UNDO\_LOG 表中，示例回滚日志如下。

```

01. {
02.     "@class": "io.seata.rm.datasource.undo.BranchUndoLog",
03.     "xid": "172.26.54.1:8091:5962967415319516023",
04.     "branchId": 5962967415319516027,
05.     "sqlUndoLogs": [
06.         "java.util.ArrayList",
07.         [
08.             {
09.                 "@class": "io.seata.rm.datasource.undo.SQLUndoLog",
10.                 "sqlType": "UPDATE",
11.                 "tableName": "webset",
12.                 "beforeImage": {
13.                     "@class": "io.seata.rm.datasource.sql.struct.TableRecords",
14.                     "tableName": "webset",
15.                     "rows": [
16.                         "java.util.ArrayList",
17.                         [
18.                             {
19.                                 "@class": "io.seata.rm.datasource.sql.struct.Row",
20.                                 "fields": [
21.                                     "java.util.ArrayList",
22.                                     [
23.                                         {
24.                                             "@class": "io.seata.rm.datasource.sql.struct.Field",
25.                                             "name": "id",
26.                                             "keyType": "PRIMARY_KEY",
27.                                             "type": -5,
28.                                             "value": [
29.                                                 "java.lang.Long",
30.                                                 1
31.                                             ]
32.                                         },
33.                                         {
34.                                             "@class": "io.seata.rm.datasource.sql.struct.Field",
35.                                             "name": "url",
36.                                             "keyType": "NULL",
37.                                             "type": 12,
38.                                             "value": "biancheng.net"
39.                                         }
40.                                     ]
41.                                 ]
42.                             }
43.                         ]
44.                     ]
45.                 }
46.             }
47.         ]
48.     ]
49. }
```



```

41.         ]
42.     }
43.     ]
44.   ]
45. },
46. "afterImage": {
47.   "@class": "io.seata.rm.datasource.sql.struct.TableRecords",
48.   "tableName": "webset",
49.   "rows": [
50.     "java.util.ArrayList",
51.     [
52.       {
53.         "@class": "io.seata.rm.datasource.sql.struct.Row",
54.         "fields": [
55.           "java.util.ArrayList",
56.           [
57.             {
58.               "@class": "io.seata.rm.datasource.sql.struct.Field",
59.               "name": "id",
60.               "keyType": "PRIMARY_KEY",
61.               "type": -5,
62.               "value": [
63.                 "java.lang.Long",
64.                 1
65.               ]
66.             },
67.             {
68.               "@class": "io.seata.rm.datasource.sql.struct.Field",
69.               "name": "url",
70.               "keyType": "NULL",
71.               "type": 12,
72.               "value": "c.biancheng.net"
73.             }
74.           ]
75.         ]
76.       }
77.     ]
78.   ]
79. }
80. ]
81. ]
82. ]
83. }

```

6. 注册分支事务，生成行锁：在这次业务操作的本地事务提交前，RM 会向 TC 注册分支事务，并针对主键 id 为 1 的记录生成行锁。



以上所有操作均在同一个数据库事务内完成，可以保证一阶段的操作的原子性。

7. 本地事务提交：将业务数据的更新和前面生成的 UNDO\_LOG 一并提交。

8. 上报执行结果：将本地事务提交的结果上报给 TC。

## 二阶段：提交

当所有的 RM 都将自己分支事务的提交结果上报给 TC 后，TM 根据 TC 收集的各个分支事务的执行结果，来决定向 TC 发起全局事务的提交或回滚。

若所有分支事务都执行成功，TM 向 TC 发起全局事务的提交，并批量删除各个 RM 保存的 UNDO\_LOG 记录和行锁；否则全局事务回滚。

## 二阶段：回滚

若全局事务中的任何一个分支事务失败，则 TM 向 TC 发起全局事务的回滚，并开启一个本地事务，执行如下操作。

1. 查找 UNDO\_LOG 记录：通过 XID 和分支事务 ID (Branch ID) 查找所有的 UNDO\_LOG 记录。

2. 数据校验：将 UNDO\_LOG 中的后镜像数据 (afterImage) 与当前数据进行比较，如果有不同，则说明数据被当前全局事务之外的动作所修改，需要人工对这些数据进行处理。

3. 生成回滚语句：根据 UNDO\_LOG 中的前镜像 (beforeImage) 和业务 SQL 的相关信息生成回滚语句：

```
update webset set url= 'biancheng.net' where id = 1;
```

4. 还原数据：执行回滚语句，并将前镜像数据、后镜像数据以及行锁删除。

5. 提交事务：提交本地事务，并把本地事务的执行结果（即分支事务回滚的结果）上报给 TC。

## 下载 Seata 服务器

1. 使用浏览器访问 "<https://github.com/seata/seata/releases/tag/v1.4.2>"，在 Seata Server 下载页面分别下载 "seata-server-1.4.2.zip"，如下图。



图3：Seata 服务器下载页面

2. 解压 seata-server-1.4.2.zip，其目录结构如下图。



图4: Seata Server 目录结构

Seata Server 目录中包含以下子目录:

- bin: 用于存放 Seata Server 可执行命令。
- conf: 用于存放 Seata Server 的配置文件。
- lib: 用于存放 Seata Server 依赖的各种 Jar 包。
- logs: 用于存放 Seata Server 的日志。

## Seata 配置中心

所谓“配置中心”，就像是一个“大衣柜”，内部存放着各种各样的配置文件，我们可以根据自己的需要从其中获取指定的配置文件，加载到对应的客户端中。

Seata 支持多种配置中心:

- nacos
- consul
- apollo
- etcd
- zookeeper
- file (读本地文件，包含 conf、properties、yml 等配置文件)

## Seata 整合 Nacos 配置中心

对于 Seata 来说，Nacos 是一种重要的配置中心实现。

Seata 整合 Nacos 配置中心的操作步骤十分简单，大致步骤如下。

### 添加 Maven 依赖

我们需要将 nacos-client 的 Maven 依赖添加到项目的 pom.xml 文件中:

01. <dependency>



```
02. <groupId>io.seata</groupId>
03. <artifactId>seata-spring-boot-starter</artifactId>
04. <version>最新版</version>
05. </dependency>
06. <dependency>
07.   <groupId>com.alibaba.nacos</groupId>
08.   <artifactId>nacos-client</artifactId>
09.   <version>1.2.0及以上版本</version>
10. </dependency>
```

在 Spring Cloud 项目中，通常只需要在 pom.xml 中添加 spring-cloud-starter-alibaba-seata 依赖即可，代码如下。

```
01. <!--引入 seata 依赖-->
02. <dependency>
03.   <groupId>com.alibaba.cloud</groupId>
04.   <artifactId>spring-cloud-starter-alibaba-seata</artifactId>
05. </dependency>
```

### Seata Server 配置

在 Seata Server 安装目录下的 config/registry.conf 中，将配置方式 (config.type) 修改为 Nacos，并对 Nacos 配置中心的相关信息进行配置，示例配置如下。

```
01. config {
02.   # Seata 支持 file、nacos、apollo、zk、consul、etcd3 等多种配置中心
03.   #配置方式修改为 nacos
04.   type = "nacos"
05.
06.   nacos {
07.     #修改为使用的 nacos 服务器地址
08.     serverAddr = "127.0.0.1:1111"
09.     #配置中心的命名空间
10.     namespace = ""
11.     #配置中心所在的分组
12.     group = "SEATA_GROUP"
13.     #Nacos 配置中心的用户名
14.     username = "nacos"
15.     #Nacos 配置中心的密码
16.     password = "nacos"
17.   }
18. }
```

### Seata Client 配置

我们可以在 Seata Client (即微服务架构中的服务) 中，通过 application.yml 等配置文件对 Nacos 配置中心进行配置，示例代码如下。

```
01. seata:
02.   config:
03.     type: nacos
```



```
04. nacos:  
05.   server-addr: 127.0.0.1:1111 # Nacos 配置中心的地址  
06.   group : "SEATA_GROUP" #分组  
07.   namespace: ""  
08.   username: "nacos" #Nacos 配置中心的用户名  
09.   password: "nacos" #Nacos 配置中心的密码
```

### 上传配置到 Nacos 配置中心

在完成了 Seata 服务端和客户端的相关配置后，接下来，我们就可以将配置上传的 Nacos 配置中心了，操作步骤如下。

1. 我们需要获取一个名为 config.txt 的文本文件，该文件包含了 Seata 配置的所有参数明细。

我们可以通过 [Seata Server 源码](#)/script/config-center 目录中获取 config.txt，然后根据自己需要修改其中的配置，如下图。



图5: config.txt

2. 在 /script/config-center/nacos 目录中，有以下 2 个 Seata 脚本：

- nacos-config.py: python 脚本。
- nacos-config.sh: 为 Linux 脚本，我们可以在 Windows 下通过 Git 命令，将 config.txt 中的 Seata 配置上传到 Nacos 配置中心。

在 seata-1.4.2\script\config-center\nacos 目录下，右键鼠标选择 Git Bush Here，并在弹出的 Git 命令窗口中执行以下命令，将 config.txt 中的配置上传到 Nacos 配置中心。

```
sh nacos-config.sh -h 127.0.0.1 -p 1111 -g SEATA_GROUP -u nacos -w nacos
```

Git 命令各参数说明如下：

- -h: Nacos 的 host，默认取值为 localhost
- -p: 端口号，默认取值为 8848
- -g: Nacos 配置的分组，默认取值为 SEATA\_GROUP
- -u: Nacos 用户名



- -w: Nacos 密码

## 验证 Nacos 配置中心

在以上所有步骤完成后，启动 Nacos Server，登陆 Nacos 控制台查看配置列表，结果如下图。

The screenshot shows the Nacos 2.0.3 configuration management interface. The left sidebar has sections like '配置管理', '历史版本', '监听查询', '服务管理', '权限控制', '命名空间', and '集群管理'. The main area is titled '配置管理 | .public' and shows a table of configuration items. The table columns are 'Data ID', 'Group', '归属应用', and '操作'. There are 10 items listed, all belonging to the 'SEATA\_GROUP'. The items are: transport.type, transport.server, transport.heartbeat, transport.enableClientBatchSendRequest, transport.threadFactory.boosThreadPrefix, transport.threadFactory.workerThreadPrefix, transport.threadFactory.serverExecutorThreadPrefix, transport.threadFactory.shareBoosWorker, transport.threadFactory.clientSelectorThreadPrefix, and transport.threadFactory.clientSelectorThreadSize. Each item has a red-bordered box around it. At the bottom of the table, there are buttons for '重置', '清除', '输出', '每页显示: 10', and a page navigation section from 1 to 9.

图6: Seata Nacos 配置中心

## Seata 注册中心

所谓“注册中心”，可以说是微服务架构中的“通讯录”，它记录了服务与服务地址的映射关系。

在分布式微服务架构中，各个微服务都可以将自己注册到注册中心，当其他服务需要调用某个服务时，就可以从这里找到它的服务地址进行调用，常见的服务注册中心有 Nacos、Eureka、zookeeper 等。

Seata 支持多种服务注册中心：

- eureka
- consul
- nacos
- etcd
- zookeeper
- sofa
- redis
- file (直连)



Seata 通过这些服务注册中心，我们可以获取 Seata Sever 的服务地址，进行调用。

## Seata 整合 Nacos 注册中心

对于 Seata 来说，Nacos 是一种重要的注册中心实现。

Seata 整合 Nacos 注册中心的步骤十分简单，步骤如下。

### 添加 Maven 依赖

将 nacos-client 的 Maven 依赖添加到项目的 pom.xml 文件中：

```
01. <dependency>
02.   <groupId>io.seata</groupId>
03.   <artifactId>seata-spring-boot-starter</artifactId>
04.   <version>最新版</version>
05. </dependency>
06. <dependency>
07.   <groupId>com.alibaba.nacos</groupId>
08.   <artifactId>nacos-client</artifactId>
09.   <version>1.2.0及以上版本</version>
10. </dependency>
```

在 Spring Cloud 项目中，通常只需要在 pom.xml 中添加 spring-cloud-starter-alibaba-seata 依赖即可，代码如下。

```
01. <!--引入 seata 依赖-->
02. <dependency>
03.   <groupId>com.alibaba.cloud</groupId>
04.   <artifactId>spring-cloud-starter-alibaba-seata</artifactId>
05. </dependency>
```

### Seata Server 配置注册中心

在 Seata Server 安装目录下的 config/registry.conf 中，将注册方式 (registry.type) 修改为 Nacos，并对 Nacos 注册中心的相关信息进行配置，示例配置如下。

```
01. registry {
02.   # Seata 支持 file、nacos、eureka、redis、zk、consul、etcd3、sofa 作为其注册中心
03.   # 将注册方式修改为 nacos
04.   type = "nacos"
05.
06.   nacos {
07.     application = "seata-server"
08.     # 修改 nacos 注册中心的地址
09.     serverAddr = "127.0.0.1:1111"
10.     group = "SEATA_GROUP"
11.     namespace = ""
12.     cluster = "default"
```



```
13.     username = ""
14.     password = ""
15. }
16. }
```

## Seata Client 配置注册中心

我们可以在 Seata Client 的 application.yml 中，对 Nacos 注册中心进行配置，示例配置如下。

```
01. seata:
02.   registry:
03.     type: nacos
04.   nacos:
05.     application: seata-server
06.     server-addr: 127.0.0.1:1111 # Nacos 注册中心的地址
07.     group : "SEATA_GROUP" #分组
08.     namespace: ""
09.     username: "nacos" #Nacos 注册中心的用户名
10.    password: "nacos" # Nacos 注册中心的密码
```

## 验证 Nacos 注册中心

在以上所有步骤完成后，先启动 Nacos Server 再启动 Seata Server，登录 Nacos 控制台查看服务列表，结果如下图。

The screenshot shows the Nacos 2.0.3 service management interface. The left sidebar has sections for Configuration Management, Service Management (with 'Service List' highlighted), Consumer List, Limit Control, and Namespace. The main area is titled 'Service List | public'. It includes search fields for 'Service Name' and 'Group Name', and a toggle switch for 'Empty Service'. A table lists services with columns: Service Name, Group Name, Cluster Count, Instance Count, Healthy Instance Count, Trigger Protection Threshold, and Operations. One row is selected, showing 'seata-server' under 'Service Name', 'SEATA\_GROUP' under 'Group Name', '1' under 'Cluster Count', '1' under 'Instance Count', '1' under 'Healthy Instance Count', and 'false' under 'Trigger Protection Threshold'. The 'Operations' column contains links for 'Details', 'Example Code', 'Consumers', and 'Delete'.

图7：Seata Nacos 注册中心

从图 9 可以看出，seata-server 服务已经注册到了 Nacos 注册中心。

## Seata 事务分组

事务分组是 Seata 提供的一种 TC (Seata Server) 服务查找机制。



Seata 通过事务分组获取 TC 服务，流程如下：

1. 在应用中配置事务分组。
2. 应用通过配置中心去查找配置： `service.vgroupMapping.{事务分组}`，该配置的值就是 TC 集群的名称。
3. 获得集群名称后，应用通过一定的前后缀 + 集群名称去构造服务名。
4. 得到服务名后，去注册中心去拉取服务列表，获得后端真实的 TC 服务列表。

下面我们以 Nacos 服务注册中心为例，介绍 Seata 事务的使用。

### Seata Server 配置

在 Seata Server 的 config/registry.conf 中，进行如下配置。

```
01. registry {  
02.     # file、nacos、eureka、redis、zk、consul、etcd3、sofa  
03.     type = "nacos"          # 使用 Nacos 作为注册中心  
04.     nacos {  
05.         serverAddr = "127.0.0.1:1111"    # Nacos 注册中心的地址  
06.         namespace = ""           # Nacos 命名空间 id，"" 为 Nacos 保留 public 空间控件，用户勿配置 namespace = "public"  
07.         cluster = "c.biancheng.net"      # seata-server 在 Nacos 的集群名  
08.     }  
09. }  
10. config {  
11.     # file、nacos、apollo、zk、consul、etcd3  
12.     type = "nacos"          # 使用 nacos 作为配置中心  
13.     nacos {  
14.         serverAddr = "localhost"  
15.         namespace = ""  
16.         cluster = "default"  
17.     }  
18. }
```

### Seata Client 配置

Seata Client 中 application.yml 的配置如下。

```
01. spring:  
02.     alibaba:  
03.         seata:  
04.             tx-service-group: service-order-group  # 事务分组名  
05.  
06.     seata:  
07.         registry:  
08.             type: nacos  # 从 Nacos 获取 TC 服务  
09.             nacos:  
10.                 server-addr: 127.0.0.1:1111  
11.         config:  
12.             type: nacos  # 使用 Nacos 作为配置中心  
13.             nacos:  
14.                 server-addr: 127.0.0.1:1111
```



15.

name

在以上配置中，我们通过 `spring.cloud.alibaba.seata.tx-service-group` 来配置 Seata 事务分组名，其默认取值为： `服务名-fescar-service-group`。

### 上传配置到 Nacos

将以下配置上传到 Nacos 配置中心。

```
service.vgroupMapping.service-order-group=c.biancheng.net
```

在以上配置中，

- `service-order-group`: 为事务分组的名称；
- `c.biancheng.net`: 为 TC 集群的名称。

### 获取事务分组

1. 先启动 Nacos，再启动 Seata Server，最后再启动 Seata Client。

2. Seata Client 在启动时，会从 `application.yml` 的配置中，根据 `spring.cloud.alibaba.seata.tx-service-group` 获取事务分组的名称：`service-order-group`。

### 获取 TC 集群名

使用事务分组名 “`service-order-group`” 拼接成 “`service.vgroupMapping.service-order-group`”，并从 Nacos 配置中心获取该配置的取值，这个值就是 TC 集群的名称：“`c.biancheng.net`”。

### 查找 TC 服务

根据 TC 集群名、Nacos 注册中心的地址 (`server-addr`) 以及命名空间 (`namespace`)，在 Nacos 注册中心找到真实的 TC 服务列表。

### 总结

通过事务分组获取服务名，共需要以下 3 步：

1. 服务启动时，从配置文件中获取服务分组的名称；
2. 从配置中心，通过事务分组名获取 TC 集群名；
3. 根据 TC 群组名以及其他信息构建服务名，获取真实的 TC 服务列表。

## 启动 Seata Server

### 1. 建表

Seata Server 共有以下 3 种存储模式 (`store.mode`)：

| 模式   | 说明   | 准备工作  |
|------|--|-------|
| file | 文件存储模式，默认存储模式；<br>该模式为单机模式，全局事务的会话信息在内存中读写，并持久化本地文件 <code>root.data</code> ，性能较高 | -     |
| db   | 数据库存储模式；<br>该模式为高可用模式，全局事务会话信息通过数据库共享，性能较低。                                      | 建数据库表 |



|       |  |                |
|-------|--|----------------|
| redis | 缓存处处模式;<br>Seata Server 1.3 及以上版本支持该模式，性能较高，但存在事务信息丢失风险, | 配置 redis 持久化配置 |
|-------|--|----------------|

在 db 模式下，我们需要针对全局事务的会话信息创建以下 3 张数据库表。

- 全局事务表，对应的表为：global\_table
- 分支事务表，对应的表为：branch\_table
- 全局锁表，对应的表为：lock\_table

在 MySQL 中，创建一个名为 seata 的数据库实例，并在该数据库内执行以下 SQL。

global\_table 的建表 SQL 如下。

```
-- ----- The script used when storeMode is 'db' -----
-- the table to store GlobalSession data
CREATE TABLE IF NOT EXISTS `global_table`
(
  `xid`          VARCHAR(128) NOT NULL,
  `transaction_id` BIGINT,
  `status`        TINYINT      NOT NULL,
  `application_id` VARCHAR(32),
  `transaction_service_group` VARCHAR(32),
  `transaction_name` VARCHAR(128),
  `timeout`       INT,
  `begin_time`    BIGINT,
  `application_data` VARCHAR(2000),
  `gmt_create`    DATETIME,
  `gmt_modified` DATETIME,
  PRIMARY KEY (`xid`),
  KEY `idx_gmt_modified_status` (`gmt_modified`, `status`),
  KEY `idx_transaction_id` (`transaction_id`)
) ENGINE = InnoDB
DEFAULT CHARSET = utf8;
```

branch\_table 的建表 SQL 如下。

```
-- the table to store BranchSession data
CREATE TABLE IF NOT EXISTS `branch_table`
(
  `branch_id`     BIGINT      NOT NULL,
  `xid`          VARCHAR(128) NOT NULL,
  `transaction_id` BIGINT,
  `resource_group_id` VARCHAR(32),
  `resource_id`    VARCHAR(256),
  `branch_type`   VARCHAR(8),
```



```
`status`          TINYINT,  
`client_id`      VARCHAR(64),  
`application_data`  VARCHAR(2000),  
`gmt_create`     DATETIME(6),  
`gmt_modified`   DATETIME(6),  
PRIMARY KEY (`branch_id`),  
KEY `idx_xid`(`xid`)  
) ENGINE = InnoDB  
DEFAULT CHARSET = utf8;
```

lock\_table 的建表 SQL 如下。

```
-- the table to store lock data  
CREATE TABLE IF NOT EXISTS `lock_table`  
(  
    `row_key`      VARCHAR(128) NOT NULL,  
    `xid`          VARCHAR(96),  
    `transaction_id` BIGINT,  
    `branch_id`    BIGINT      NOT NULL,  
    `resource_id`   VARCHAR(256),  
    `table_name`    VARCHAR(32),  
    `pk`           VARCHAR(36),  
    `gmt_create`    DATETIME,  
    `gmt_modified`  DATETIME,  
    PRIMARY KEY (`row_key`),  
    KEY `idx_branch_id`(`branch_id`)  
) ENGINE = InnoDB  
DEFAULT CHARSET = utf8;
```

## 2. 修改 Seata Server 配置

在 seata-server-1.4.2/conf/ 目录下的 registry.conf 中，将 Seata Server 的服务注册方式 (registry.type) 和配置方式 (config.type) 都修改为 Nacos，修改内容如下。

```
01. registry {  
02.     # file 、nacos 、eureka、redis、zk、consul、etcd3、sofa  
03.     # 将注册方式修改为 nacos  
04.     type = "nacos"  
05.  
06.     nacos {  
07.         application = "seata-server"  
08.         # 修改 nacos 的地址  
09.         serverAddr = "127.0.0.1:1111"  
10.         group = "SEATA_GROUP"  
11.         namespace = ""  
12.         cluster = "default"  
13.         username = ""
```



```
14.         password = ""
15.     }
16. }
17.
18. config {
19.     # file、nacos、apollo、zk、consul、etcd3
20.     #配置方式修改为 nacos
21.     type = "nacos"
22.
23.     nacos {
24.         #修改为使用的 nacos 服务器地址
25.         serverAddr = "127.0.0.1:1111"
26.         namespace = ""
27.         group = "SEATA_GROUP"
28.         username = "nacos"
29.         password = "nacos"
30.         #不使用 seataServer.properties 方式配置
31.         #dataId = "seataServer.properties"
32.     }
33. }
```

### 3. 将 Seata 配置上传到 Nacos

1) 下载并解压 Seata Server 的源码 seata-1.4.2.zip，然后修改 seata-1.4.2/script/config-center 目录下的 config.txt，修改内容如下。

```
#将 Seata Server 的存储模式修改为 db
store.mode=db
# 数据库驱动
store.db.driverClassName=com.mysql.cj.jdbc.Driver
# 数据库 url
store.db.url=jdbc:mysql://127.0.0.1:3306/seata?useSSL=false&characterEncoding=UTF-8&useUnicode=true&serverTimezone=UTC
# 数据库的用户名
store.db.user=root
# 数据库的密码
store.db.password=root
# 自定义事务分组
service.vgroupMapping.service-order-group=default
service.vgroupMapping.service-storage-group=default
service.vgroupMapping.service-account-group=default
```

2) 在 seata-1.4.2\script\config-center\nacos 目录下，右键鼠标选择 Git Bush Here，在弹出的 Git 命令窗口中执行以下命令，将 config.txt 中的配置上传到 Nacos 配置中心。

```
sh nacos-config.sh -h 127.0.0.1 -p 1111 -g SEATA_GROUP -u nacos -w nacos
```

3) 当 Git 命令窗口出现以下执行日志时，则说明配置上传成功。



```
=====
Complete initialization parameters, total-count:87 , failure-count:0
=====
Init nacos config finished, please start seata-server.
```

4) 使用浏览器访问 Nacos 服务器主页，查看 Nacos 配置列表，如下图。

| Data ID  | Group       | 操作                       |
|--|-------------|--------------------------|
| transport.type                                     | SEATA_GROUP | 详情   示例代码   编辑   删除   更多 |
| transport.server                                   | SEATA_GROUP | 详情   示例代码   编辑   删除   更多 |
| transport.heartbeat                                | SEATA_GROUP | 详情   示例代码   编辑   删除   更多 |
| transport.enableClientBatchSendRequest             | SEATA_GROUP | 详情   示例代码   编辑   删除   更多 |
| transport.threadFactory.bossThreadPrefix           | SEATA_GROUP | 详情   示例代码   编辑   删除   更多 |
| transport.threadFactory.workerThreadPrefix         | SEATA_GROUP | 详情   示例代码   编辑   删除   更多 |
| transport.threadFactory.serverExecutorThreadPrefix | SEATA_GROUP | 详情   示例代码   编辑   删除   更多 |
| transport.threadFactory.shareBossWorker            | SEATA_GROUP | 详情   示例代码   编辑   删除   更多 |
| transport.threadFactory.clientSelectorThreadPrefix | SEATA_GROUP | 详情   示例代码   编辑   删除   更多 |
| transport.threadFactory.clientSelectorThreadSize   | SEATA_GROUP | 详情   示例代码   编辑   删除   更多 |

图8: Seata 配置上传到 Nacos

注意：在使用 Git 命令将配置上传到 Nacos 前，应该先确保 Nacos 服务器已启动。

#### 4. 启动 Seata Server

双击 Seata Server 端 bin 目录下的启动脚本 seata-server.bat，启动 Seata Server。



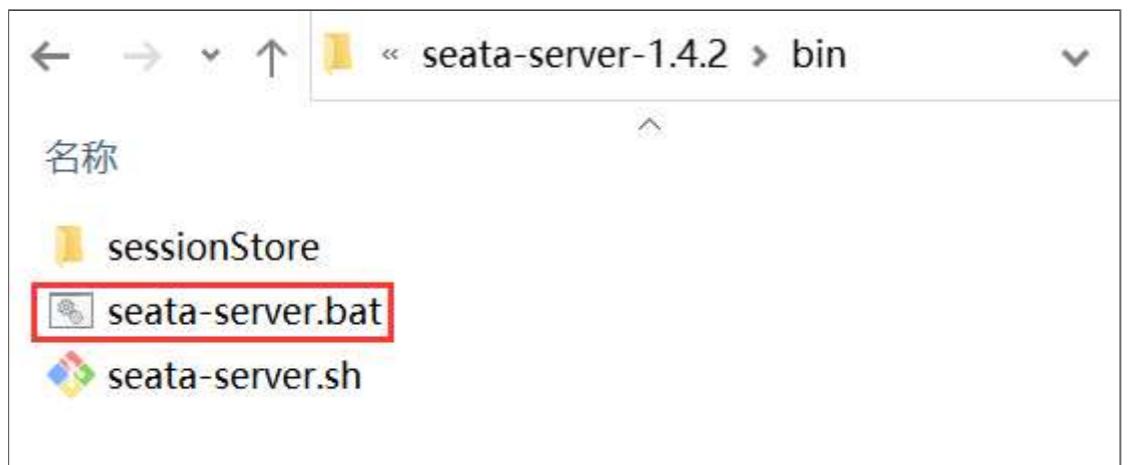


图9: Seata Server 启动脚本

Seata Server 启动日志如下。

```
16:52:48,549 |-INFO in ch.qos.logback.classic.LoggerContext[default] - Could NOT find resource [logback-test.xml]
16:52:48,549 |-INFO in ch.qos.logback.classic.LoggerContext[default] - Could NOT find resource [logback.groovy]
16:52:48,550 |-INFO in ch.qos.logback.classic.LoggerContext[default] - Found resource [logback.xml] at
[file:/C:/Users/79330/Desktop/seata-server-1.4.2/conf/logback.xml]
16:52:48,551 |-WARN in ch.qos.logback.classic.LoggerContext[default] - Resource [logback.xml] occurs multiple times on the classpath.
16:52:48,551 |-WARN in ch.qos.logback.classic.LoggerContext[default] - Resource [logback.xml] occurs at
[jar:file:/C:/Users/79330/Desktop/seata-server-1.4.2/lib/seata-server-1.4.2.jar!/logback.xml]
.....
SLF4J: A number (18) of logging calls during the initialization phase have been intercepted and are
SLF4J: now being replayed. These are subject to the filtering rules of the underlying logging system.
SLF4J: See also http://www.slf4j.org/codes.html#replay
16:52:49.003 INFO --- [           main] io.seata.config.FileConfiguration : The file name of the operation is
registry
16:52:49.008 INFO --- [           main] io.seata.config.FileConfiguration : The configuration file used is
C:/Users/79330/Desktop/seata-server-1.4.2/conf/registry.conf
16:52:51.063 INFO --- [           main] com.alibaba.druid.pool.DruidDataSource : {dataSource-1} initied
16:52:51.981 INFO --- [           main] i.s.core.rpc.netty.NettyServerBootstrap : Server started, listen port: 8091
```

## 业务系统集成 Seata

接下来，我们就以电商系统为例，来演示下业务系统是如何集成 Seata 的。

在电商系统中，用户下单购买一件商品，需要以下 3 个服务提供支持：

- Order (订单服务)：创建和修改订单。
- Storage (库存服务)：对指定的商品扣除仓库库存。
- Account (账户服务)：从用户帐户中扣除商品金额。

这三个微服务分别使用三个不同的数据库，架构图如下所示。

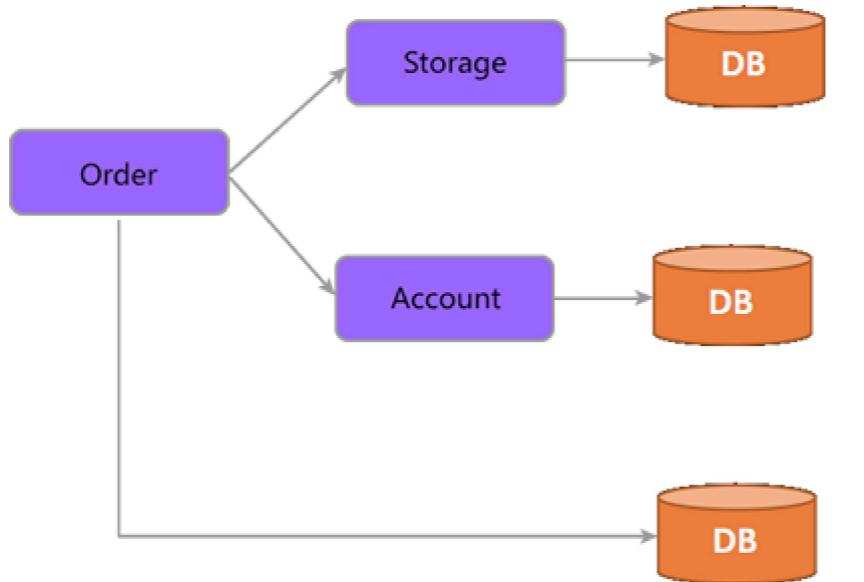


图10：电商系统架构图

当用户从这个电商网站购买了一件商品后，其服务调用步骤如下：

1. 调用 Order 服务，创建一条订单数据，订单状态为“未完成”；
2. 调用 Storage 服务，扣减商品库存；
3. 调用 Account 服务，从用户账户中扣除商品金额；
4. 调用 Order 服务，将订单状态修改为“已完成”。

#### 创建订单 (Order) 服务

1. 在 MySQL 数据库中，新建一个名为 seata-order 的数据库实例，并通过以下 SQL 语句创建 2 张表：t\_order（订单表）和 undo\_log（回滚日志表）。

```
-- 
-- Table structure for t_order
-- 

DROP TABLE IF EXISTS `t_order`;
CREATE TABLE `t_order` (
  `id` bigint NOT NULL AUTO_INCREMENT,
  `user_id` bigint DEFAULT NULL COMMENT '用户id',
  `product_id` bigint DEFAULT NULL COMMENT '产品id',
  `count` int DEFAULT NULL COMMENT '数量',
  `money` decimal(11,0) DEFAULT NULL COMMENT '金额',
  `status` int DEFAULT NULL COMMENT '订单状态: 0: 未完成; 1: 已完结',
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=32 DEFAULT CHARSET=utf8;

-- 
-- Table structure for undo_log
-- 

DROP TABLE IF EXISTS `undo_log`;
CREATE TABLE `undo_log` (
  `branch_id` bigint NOT NULL COMMENT 'branch transaction id',
  `xid` varchar(128) NOT NULL COMMENT 'global transaction id',
  `context` varchar(128) NOT NULL COMMENT 'undo_log context, such as serialization',
  `rollback_info` longblob NOT NULL COMMENT 'rollback info',
)
```



```

`log_status` int NOT NULL COMMENT '0:normal status,1:defense status',
`log_created` datetime(6) NOT NULL COMMENT 'create datetime',
`log_modified` datetime(6) NOT NULL COMMENT 'modify datetime',
UNIQUE KEY `ux_undo_log` (`xid`, `branch_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 ;

```

2. 在主工程 spring-cloud-alibaba-demo 下，创建一个名为 spring-cloud-alibaba-seata-order-8005 的 Spring Boot 模块，并在其 pom.xml 中添加以下依赖，内容如下。

```

01. <?xml version="1.0" encoding="UTF-8"?>
02. <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 https://maven.apache.org/xsd/maven-4.0.0.xsd">
03.   <modelVersion>4.0.0</modelVersion>
04.   <parent>
05.     <groupId>net.biancheng.c</groupId>
06.     <version>1.0-SNAPSHOT</version>
07.     <artifactId>spring-cloud-alibaba-demo</artifactId>
08.   </parent>
09.
10.
11.   <groupId>net.biancheng.c</groupId>
12.   <artifactId>spring-cloud-alibaba-seata-order-8005</artifactId>
13.   <version>0.0.1-SNAPSHOT</version>
14.   <name>spring-cloud-alibaba-seata-order-8005</name>
15.   <description>Demo project for Spring Boot</description>
16.   <properties>
17.     <java.version>1.8</java.version>
18.     <seata.version>1.4.2</seata.version>
19.   </properties>
20.   <dependencies>
21.     <!--nacos 服务注册中心-->
22.     <dependency>
23.       <groupId>com.alibaba.cloud</groupId>
24.       <artifactId>spring-cloud-starter-alibaba-nacos-discovery</artifactId>
25.       <exclusions>
26.         <exclusion>
27.           <groupId>org.springframework.cloud</groupId>
28.           <artifactId>spring-cloud-starter-netflix-ribbon</artifactId>
29.         </exclusion>
30.       </exclusions>
31.     </dependency>
32.     <dependency>
33.       <groupId>org.springframework.boot</groupId>
34.       <artifactId>spring-boot-starter-web</artifactId>
35.     </dependency>
36.     <!--引入 OpenFeign 的依赖-->
37.     <dependency>

```



```
38.      <groupId>org.springframework.cloud</groupId>
39.      <artifactId>spring-cloud-starter-openfeign</artifactId>
40.    </dependency>
41.  <dependency>
42.    <groupId>org.springframework.cloud</groupId>
43.    <artifactId>spring-cloud-loadbalancer</artifactId>
44.  </dependency>
45.  <!--引入 seata 依赖-->
46.  <dependency>
47.    <groupId>com.alibaba.cloud</groupId>
48.    <artifactId>spring-cloud-starter-alibaba-seata</artifactId>
49.  </dependency>
50.
51.  <dependency>
52.    <groupId>org.springframework.boot</groupId>
53.    <artifactId>spring-boot-devtools</artifactId>
54.    <scope>runtime</scope>
55.    <optional>true</optional>
56.  </dependency>
57.  <dependency>
58.    <groupId>org.projectlombok</groupId>
59.    <artifactId>lombok</artifactId>
60.    <optional>true</optional>
61.  </dependency>
62.  <dependency>
63.    <groupId>org.springframework.boot</groupId>
64.    <artifactId>spring-boot-starter-test</artifactId>
65.    <scope>test</scope>
66.  </dependency>
67.  <dependency>
68.    <groupId>net.biancheng.c</groupId>
69.    <artifactId>spring-cloud-alibaba-api</artifactId>
70.    <version>${project.version}</version>
71.  </dependency>
72.
73.  <dependency>
74.    <groupId>junit</groupId>
75.    <artifactId>junit</artifactId>
76.    <version>4.12</version>
77.  </dependency>
78.  <dependency>
79.    <groupId>mysql</groupId>
80.    <artifactId>mysql-connector-java</artifactId>
81.    <version>8.0.19</version>
82.  </dependency>
83.
```



```
84.     <dependency>
85.         <groupId>ch.qos.logback</groupId>
86.         <artifactId>logback-core</artifactId>
87.
88.     </dependency>
89.     <dependency>
90.         <groupId>org.mybatis.spring.boot</groupId>
91.         <artifactId>mybatis-spring-boot-starter</artifactId>
92.         <version>2.2.0</version>
93.     </dependency>
94.     <!--添加 Spring Boot 的监控模块-->
95.     <!-- https://mvnrepository.com/artifact/org.springframework.boot/spring-boot-starter-actuator -->
96.     <dependency>
97.         <groupId>org.springframework.boot</groupId>
98.         <artifactId>spring-boot-starter-actuator</artifactId>
99.     </dependency>
100.    <!--SpringCloud alibaba sentinel -->
101.    <dependency>
102.        <groupId>com.alibaba.cloud</groupId>
103.        <artifactId>spring-cloud-starter-alibaba-sentinel</artifactId>
104.    </dependency>
105.    <!--配置中心 nacos-->
106.    <dependency>
107.        <groupId>com.alibaba.cloud</groupId>
108.        <artifactId>spring-cloud-starter-alibaba-nacos-config</artifactId>
109.    </dependency>
110.
111.    </dependencies>
112.
113.    <build>
114.        <plugins>
115.            <!--mybatis自动生成代码插件-->
116.            <plugin>
117.                <groupId>org.mybatis.generator</groupId>
118.                <artifactId>mybatis-generator-maven-plugin</artifactId>
119.                <version>1.4.0</version>
120.                <configuration>
121.                    <configurationFile>src/main/resources/mybatis-generator/generatorConfig.xml</configurationFile>
122.                    <verbose>true</verbose>
123.                    <!-- 是否覆盖, true表示会替换生成的JAVA文件, false则不覆盖 -->
124.                    <overwrite>true</overwrite>
125.                </configuration>
126.                <dependencies>
127.                    <!--mysql驱动包-->
128.                    <dependency>
129.                        <groupId>mysql</groupId>
```



```
130.          <artifactId>mysql-connector-java</artifactId>
131.          <version>8.0.19</version>
132.        </dependency>
133.        <dependency>
134.          <groupId>org.mybatis.generator</groupId>
135.          <artifactId>mybatis-generator-core</artifactId>
136.          <version>1.4.0</version>
137.        </dependency>
138.      </dependencies>
139.    </plugin>
140.    <plugin>
141.      <groupId>org.springframework.boot</groupId>
142.      <artifactId>spring-boot-maven-plugin</artifactId>
143.    </plugin>
144.  </plugins>
145. </build>
146. </project>
```

3. 在 spring-cloud-alibaba-seata-order-8005 的类路径 (/resources 目录) 下, 创建一个配置文件 bootstrap.yml, 配置内容如下。

```
01. spring:
02.   cloud:
03.     ## Nacos认证信息
04.     nacos:
05.       config:
06.         username: nacos
07.         password: nacos
08.         context-path: /nacos
09.         server-addr: 127.0.0.1:1111 # 设置配置中心服务端地址
10.       namespace: # Nacos 配置中心的namespace。需要注意, 如果使用 public 的 namcespace , 请不要填写这个值, 直接留空即可
```

4. 在 spring-cloud-alibaba-seata-order-8005 的类路径 (/resources 目录) 下, 创建一个配置文件 application.yml, 配置内容如下。

```
01. spring:
02.   application:
03.     name: spring-cloud-alibaba-seata-order-8005 #服务名
04.   #数据源配置
05.   datasource:
06.     driver-class-name: com.mysql.jdbc.Driver #数据库驱动
07.     name: defaultDataSource
08.     url: jdbc:mysql://localhost:3306/seata_order?serverTimezone=UTC #数据库连接地址
09.     username: root #数据库的用户名
10.     password: root #数据库密码
11.
12.   cloud:
13.     nacos:
```



```
14.     discovery:
15.         server-addr: 127.0.0.1:1111 #nacos 服务器地址
16.         namespace: public #nacos 命名空间
17.         username:
18.         password:
19.     sentinel:
20.         transport:
21.             dashboard: 127.0.0.1:8080 #Sentinel 控制台地址
22.             port: 8719
23.
24.     alibaba:
25.         seata:
26.             #自定义服务群组，该值必须与 Nacos 配置中的 service.vgroupMapping. {my-service-group}=default 中的 {my-service-group} 相同
27.             tx-service-group: service-order-group
28.     server:
29.         port: 8005 #端口
30.     seata:
31.         application-id: ${spring.application.name}
32.         #自定义服务群组，该值必须与 Nacos 配置中的 service.vgroupMapping. {my-service-group}=default 中的 {my-service-group} 相同
33.         tx-service-group: service-order-group
34.     service:
35.         grouplist:
36.             #Seata 服务器地址
37.             seata-server: 127.0.0.1:8091
38.     # Seata 的注册方式为 nacos
39.     registry:
40.         type: nacos
41.     nacos:
42.         server-addr: 127.0.0.1:1111
43.     # Seata 的配置中心为 nacos
44.     config:
45.         type: nacos
46.     nacos:
47.         server-addr: 127.0.0.1:1111
48.
49.     feign:
50.         sentinel:
51.             enabled: true #开启 OpenFeign 功能
52.
53.     management:
54.         endpoints:
55.             web:
56.                 exposure:
57.                     include: "*"
58.
59. ##### MyBatis 配置 #####
```



```
60. mybatis:  
61.     # 指定 mapper.xml 的位置  
62.     mapper-locations: classpath:mybatis/mapper/*.xml  
63.     #扫描实体类的位置,在此处指明扫描实体类的包,在 mapper.xml 中就可以不写实体类的全路径名  
64.     type-aliases-package: net.biancheng.c.entity  
65.     configuration:  
66.         #默认开启驼峰命名法, 可以不用设置该属性  
67.         map-underscore-to-camel-case: true
```

5. 在 net.biancheng.c.entity 包下, 创建一个名为 Order 的实体类, 代码如下。

```
01. package net.biancheng.c.entity;  
02.  
03. import java.math.BigDecimal;  
04.  
05. public class Order {  
06.     private Long id;  
07.  
08.     private Long userId;  
09.  
10.    private Long productId;  
11.  
12.    private Integer count;  
13.  
14.    private BigDecimal money;  
15.  
16.    private Integer status;  
17.  
18.    public Long getId() {  
19.        return id;  
20.    }  
21.  
22.    public void setId(Long id) {  
23.        this.id = id;  
24.    }  
25.  
26.    public Long getUserId() {  
27.        return userId;  
28.    }  
29.  
30.    public void setUserId(Long userId) {  
31.        this.userId = userId;  
32.    }  
33.  
34.    public Long getProductId() {
```



```
35.         return productId;
36.     }
37.
38.     public void setProductId(Long productId) {
39.         this.productId = productId;
40.     }
41.
42.     public Integer getCount() {
43.         return count;
44.     }
45.
46.     public void setCount(Integer count) {
47.         this.count = count;
48.     }
49.
50.     public BigDecimal getMoney() {
51.         return money;
52.     }
53.
54.     public void setMoney(BigDecimal money) {
55.         this.money = money;
56.     }
57.
58.     public Integer getStatus() {
59.         return status;
60.     }
61.
62.     public void setStatus(Integer status) {
63.         this.status = status;
64.     }
65. }
```

6. 在 net.biancheng.c.mapper 包下，创建一个名为 OrderMapper 的 Mapper 接口，代码如下。

```
01. package net.biancheng.c.mapper;
02.
03. import net.biancheng.c.entity.Order;
04. import org.apache.ibatis.annotations.Mapper;
05. import org.apache.ibatis.annotations.Param;
06.
07. @Mapper
08. public interface OrderMapper {
09.     int deleteByPrimaryKey(Long id);
10.
11.     int insert(Order record);
12.
```



```

13. int create(Order order);
14.
15. int insertSelective(Order record);
16.
17. //2 修改订单状态，从零改为1
18. void update(@Param("userId") Long userId, @Param("status") Integer status);
19.
20. Order selectByPrimaryKey(Long id);
21.
22. int updateByPrimaryKeySelective(Order record);
23.
24. int updateByPrimaryKey(Order record);
25. }

```

7. 在 /resources/mybatis/mapper 目录下，创建一个名为 OrderMapper.xml 的 MyBatis 映射文件，代码如下。

```

01. <?xml version="1.0" encoding="UTF-8"?>
02. <!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN" "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
03. <mapper namespace="net.biancheng.c.mapper.OrderMapper">
04.     <resultMap id="BaseResultMap" type="net.biancheng.c.entity.Order">
05.         <id column="id" jdbcType="BIGINT" property="id"/>
06.         <result column="user_id" jdbcType="BIGINT" property="userId"/>
07.         <result column="product_id" jdbcType="BIGINT" property="productId"/>
08.         <result column="count" jdbcType="INTEGER" property="count"/>
09.         <result column="money" jdbcType="DECIMAL" property="money"/>
10.         <result column="status" jdbcType="INTEGER" property="status"/>
11.     </resultMap>
12.     <sql id="Base_Column_List">
13.         id
14.         , user_id, product_id, count, money, status
15.     </sql>
16.     <select id="selectByPrimaryKey" parameterType="java.lang.Long" resultMap="BaseResultMap">
17.         select
18.             <include refid="Base_Column_List"/>
19.         from t_order
20.         where id = #{id, jdbcType=BIGINT}
21.     </select>
22.     <delete id="deleteByPrimaryKey" parameterType="java.lang.Long">
23.         delete
24.         from t_order
25.         where id = #{id, jdbcType=BIGINT}
26.     </delete>
27.     <insert id="insert" parameterType="net.biancheng.c.entity.Order">
28.         insert into t_order (id, user_id, product_id,
29.                             count, money, status)
30.         values (#{id, jdbcType=BIGINT}, #{userId, jdbcType=BIGINT}, #{productId, jdbcType=BIGINT},

```



```
31.          #{count, jdbcType=INTEGER}, #{money, jdbcType=DECIMAL}, #{status, jdbcType=INTEGER})
32.      </insert>
33.      <insert id="insertSelective" parameterType="net.biancheng.c.entity.Order">
34.          insert into t_order
35.              <trim prefix "(" suffix ")" suffixOverrides="">
36.                  <if test="id != null">
37.                      id,
38.                  </if>
39.                  <if test="userId != null">
40.                      user_id,
41.                  </if>
42.                  <if test="productId != null">
43.                      product_id,
44.                  </if>
45.                  <if test="count != null">
46.                      count,
47.                  </if>
48.                  <if test="money != null">
49.                      money,
50.                  </if>
51.                  <if test="status != null">
52.                      status,
53.                  </if>
54.              </trim>
55.              <trim prefix="values (" suffix ")" suffixOverrides="">
56.                  <if test="id != null">
57.                      #{id, jdbcType=BIGINT},
58.                  </if>
59.                  <if test="userId != null">
60.                      #{userId, jdbcType=BIGINT},
61.                  </if>
62.                  <if test="productId != null">
63.                      #{productId, jdbcType=BIGINT},
64.                  </if>
65.                  <if test="count != null">
66.                      #{count, jdbcType=INTEGER},
67.                  </if>
68.                  <if test="money != null">
69.                      #{money, jdbcType=DECIMAL},
70.                  </if>
71.                  <if test="status != null">
72.                      #{status, jdbcType=INTEGER},
73.                  </if>
74.              </trim>
75.      </insert>
76.      <insert id="create" parameterType="net.biancheng.c.entity.Order">
```



```

77.         insert into t_order (user_id, product_id,
78.                             count, money, status)
79.             values (#{userId, jdbcType=BIGINT}, #{productId, jdbcType=BIGINT},
80.                     #{count, jdbcType=INTEGER}, #{money, jdbcType=DECIMAL}, #{status, jdbcType=INTEGER})
81.
82.     </insert>
83.     <update id="updateByPrimaryKeySelective" parameterType="net.biancheng.c.entity.Order">
84.         update t_order
85.         <set>
86.             <if test="userId != null">
87.                 user_id = #{userId, jdbcType=BIGINT},
88.             </if>
89.             <if test="productId != null">
90.                 product_id = #{productId, jdbcType=BIGINT},
91.             </if>
92.             <if test="count != null">
93.                 count = #{count, jdbcType=INTEGER},
94.             </if>
95.             <if test="money != null">
96.                 money = #{money, jdbcType=DECIMAL},
97.             </if>
98.             <if test="status != null">
99.                 status = #{status, jdbcType=INTEGER},
100.            </if>
101.        </set>
102.        where id = #{id, jdbcType=BIGINT}
103.    </update>
104.    <update id="updateByPrimaryKey" parameterType="net.biancheng.c.entity.Order">
105.        update t_order
106.        set user_id      = #{userId, jdbcType=BIGINT},
107.            product_id   = #{productId, jdbcType=BIGINT},
108.            count        = #{count, jdbcType=INTEGER},
109.            money        = #{money, jdbcType=DECIMAL},
110.            status       = #{status, jdbcType=INTEGER}
111.        where id = #{id, jdbcType=BIGINT}
112.    </update>
113.
114.    <update id="update">
115.        update t_order
116.        set status = 1
117.        where user_id = #{userId}
118.            and status = #{status};
119.    </update>
120. </mapper>

```

8. 在 net.biancheng.c.service 包下，创建一个名为 OrderService 的 Service 接口，代码如下。



```
01. package net.biancheng.c.service;
02.
03. import net.biancheng.c.entity.Order;
04.
05. public interface OrderService {
06.     /**
07.      * 创建订单数据
08.      * @param order
09.     */
10.     CommonResult create(Order order);
11. }
```

9. 在 net.biancheng.c.service 包下，创建一个名为 StorageService 的接口，代码如下。

```
01. package net.biancheng.c.service;
02.
03. import net.biancheng.c.entity.CommonResult;
04. import org.springframework.cloud.openfeign.FeignClient;
05. import org.springframework.web.bind.annotation.PostMapping;
06. import org.springframework.web.bind.annotation.RequestParam;
07.
08. @FeignClient(value = "spring-cloud-alibaba-seata-storage-8006")
09. public interface StorageService {
10.     @PostMapping(value = "/storage/decrease")
11.     CommonResult decrease(@RequestParam("productId") Long productId, @RequestParam("count") Integer count);
12. }
```

10. 在 net.biancheng.c.service 包下，创建一个名为 AccountService 的接口，代码如下。

```
01. package net.biancheng.c.service;
02.
03. import net.biancheng.c.entity.CommonResult;
04. import org.springframework.cloud.openfeign.FeignClient;
05. import org.springframework.web.bind.annotation.PostMapping;
06. import org.springframework.web.bind.annotation.RequestParam;
07.
08. import java.math.BigDecimal;
09.
10. @FeignClient(value = "spring-cloud-alibaba-seata-account-8007")
11. public interface AccountService {
12.     @PostMapping(value = "/account/decrease")
13.     CommonResult decrease(@RequestParam("userId") Long userId, @RequestParam("money") BigDecimal money);
14. }
```

11. 在 net.biancheng.c.service.impl 包下，创建 OrderService 接口的实现类 OrderServiceImpl，代码如下。



```
01. package net.biancheng.c.service.impl;
02.
03. import io.seata.spring.annotation.GlobalTransactional;
04. import lombok.extern.slf4j.Slf4j;
05. import net.biancheng.c.entity.Order;
06. import net.biancheng.c.mapper.OrderMapper;
07. import net.biancheng.c.service.AccountService;
08. import net.biancheng.c.service.OrderService;
09. import net.biancheng.c.service.StorageService;
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 OrderMapper orderMapper;
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.      * 简单说：下订单->扣库存->减余额->改订单状态
27.     */
28.     @Override
29.     @GlobalTransactional(name = "fsp-create-order", rollbackFor = Exception.class)
30.     public CommonResult create(Order order) {
31.         log.info("----->开始新建订单");
32.         //1 新建订单
33.         order.setUserId(new Long(1));
34.         order.setStatus(0);
35.         orderMapper.create(order);
36.         //2 扣减库存
37.         log.info("----->订单服务开始调用库存服务，开始扣减库存");
38.         storageService.decrease(order.getProductId(), order.getCount());
39.         log.info("----->订单微服务开始调用库存，扣减库存结束");
40.         //3 扣减账户
41.         log.info("----->订单服务开始调用账户服务，开始从账户扣减商品金额");
42.         accountService.decrease(order.getUserId(), order.getMoney());
43.         log.info("----->订单微服务开始调用账户，账户扣减商品金额结束");
44.         //4 修改订单状态，从零到1, 1代表已经完成
45.         log.info("----->修改订单状态开始");
46.         orderMapper.update(order.getUserId(), 0);
```



```
47.     log.info("----->修改订单状态结束");
48.     log.info("----->下订单结束了----->");
49.     return new CommonResult(200, "订单创建成功");
50. }
51. }
```

12. 在 net.biancheng.c.controller 包下，创建一个名为 OrderController 的 Controller，代码如下。

```
01. package net.biancheng.c.controller;
02.
03. import net.biancheng.c.entity.CommonResult;
04. import net.biancheng.c.entity.Order;
05. import net.biancheng.c.service.OrderService;
06. import org.springframework.beans.factory.annotation.Autowired;
07. import org.springframework.web.bind.annotation.GetMapping;
08. import org.springframework.web.bind.annotation.PathVariable;
09. import org.springframework.web.bind.annotation.RestController;
10.
11. import java.math.BigDecimal;
12.
13. @RestController
14. public class OrderController {
15.     @Autowired
16.     private OrderService orderService;
17.
18.     @GetMapping("/order/create/{productId}/{count}/{money}")
19.     public CommonResult create(@PathVariable("productId") Integer productId, @PathVariable("count") Integer count
20.         , @PathVariable("money") BigDecimal money) {
21.         Order order = new Order();
22.         order.setProductId(Integer.valueOf(productId).longValue());
23.         order.setCount(count);
24.         order.setMoney(money);
25.         return orderService.create(order);
26.     }
27. }
```

13. 主启动类代码如下。

```
01. package net.biancheng.c;
02.
03. import org.springframework.boot.SpringApplication;
04. import org.springframework.boot.autoconfigure.SpringBootApplication;
05. import org.springframework.cloud.client.discovery.EnableDiscoveryClient;
06. import org.springframework.cloud.openfeign.EnableFeignClients;
07.
08. @EnableDiscoveryClient
```



```

9. @EnableFeignClients
10. @SpringBootApplication(scanBasePackages = "net.biancheng")
11. public class SpringCloudAlibabaSeataOrder8005Application {
12.
13.     public static void main(String[] args) {
14.         SpringApplication.run(SpringCloudAlibabaSeataOrder8005Application.class, args);
15.     }
16. }

```

## 搭建库存 (Storage) 服务

- 在 MySQL 数据库中，新建一个名为 seata-storage 的数据库实例，并通过以下 SQL 语句创建 2 张表：t\_storage（库存表）和 undo\_log（回滚日志表）。

```

-- -----
-- Table structure for t_storage
-- -----
DROP TABLE IF EXISTS `t_storage`;
CREATE TABLE `t_storage` (
    `id` bigint NOT NULL AUTO_INCREMENT,
    `product_id` bigint DEFAULT NULL COMMENT '产品id',
    `total` int DEFAULT NULL COMMENT '总库存',
    `used` int DEFAULT NULL COMMENT '已用库存',
    `residue` int DEFAULT NULL COMMENT '剩余库存',
    PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=2 DEFAULT CHARSET=utf8;

-- -----
-- Records of t_storage
-- -----
INSERT INTO `t_storage` VALUES ('1', '1', '100', '0', '100');

-- Table structure for undo_log
-- -----
DROP TABLE IF EXISTS `undo_log`;
CREATE TABLE `undo_log` (
    `branch_id` bigint NOT NULL COMMENT 'branch transaction id',
    `xid` varchar(128) NOT NULL COMMENT 'global transaction id',
    `context` varchar(128) NOT NULL COMMENT 'undo_log context, such as serialization',
    `rollback_info` longblob NOT NULL COMMENT 'rollback info',
    `log_status` int NOT NULL COMMENT '0:normal status,1:defense status',
    `log_created` datetime(6) NOT NULL COMMENT 'create datetime',
    `log_modified` datetime(6) NOT NULL COMMENT 'modify datetime',
    UNIQUE KEY `ux_undo_log` (`xid`,`branch_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COMMENT='AT transaction mode undo table';

```

- 在主工程 spring-cloud-alibaba-demo 下，创建一个名为 spring-cloud-alibaba-seata-storage-8006 的 Spring Boot 模块，并在其 pom.xml 中添加



以下依赖，内容如下。

```
01. <?xml version="1.0" encoding="UTF-8"?>
02. <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
03.   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">
04.   <modelVersion>4.0.0</modelVersion>
05.   <parent>
06.     <groupId>net.biancheng.c</groupId>
07.     <version>1.0-SNAPSHOT</version>
08.     <artifactId>spring-cloud-alibaba-demo</artifactId>
09.   </parent>
10.
11.   <groupId>net.biancheng.c</groupId>
12.   <artifactId>spring-cloud-alibaba-seata-storage-8006</artifactId>
13.   <version>0.0.1-SNAPSHOT</version>
14.   <name>spring-cloud-alibaba-seata-storage-8006</name>
15.   <description>Demo project for Spring Boot</description>
16.   <properties>
17.     <java.version>1.8</java.version>
18.     <seata.version>1.4.2</seata.version>
19.   </properties>
20.   <dependencies>
21.     <!--nacos 服务注册中心-->
22.     <dependency>
23.       <groupId>com.alibaba.cloud</groupId>
24.       <artifactId>spring-cloud-starter-alibaba-nacos-discovery</artifactId>
25.       <exclusions>
26.         <exclusion>
27.           <groupId>org.springframework.cloud</groupId>
28.           <artifactId>spring-cloud-starter-netflix-ribbon</artifactId>
29.         </exclusion>
30.       </exclusions>
31.     </dependency>
32.     <dependency>
33.       <groupId>org.springframework.boot</groupId>
34.       <artifactId>spring-boot-starter-web</artifactId>
35.     </dependency>
36.     <!--引入 OpenFeign 的依赖-->
37.     <dependency>
38.       <groupId>org.springframework.cloud</groupId>
39.       <artifactId>spring-cloud-starter-openfeign</artifactId>
40.     </dependency>
41.     <dependency>
42.       <groupId>org.springframework.cloud</groupId>
43.       <artifactId>spring-cloud-loadbalancer</artifactId>
44.     </dependency>
45.     <!--seata 依赖-->
```



```
46. <dependency>
47.     <groupId>com.alibaba.cloud</groupId>
48.     <artifactId>spring-cloud-starter-alibaba-seata</artifactId>
49. </dependency>
50. <dependency>
51.     <groupId>org.springframework.boot</groupId>
52.     <artifactId>spring-boot-devtools</artifactId>
53.     <scope>runtime</scope>
54.     <optional>true</optional>
55. </dependency>
56. <dependency>
57.     <groupId>org.projectlombok</groupId>
58.     <artifactId>lombok</artifactId>
59.     <optional>true</optional>
60. </dependency>
61. <dependency>
62.     <groupId>org.springframework.boot</groupId>
63.     <artifactId>spring-boot-starter-test</artifactId>
64.     <scope>test</scope>
65. </dependency>
66. <dependency>
67.     <groupId>net.biancheng.c</groupId>
68.     <artifactId>spring-cloud-alibaba-api</artifactId>
69.     <version>${project.version}</version>
70. </dependency>
71.
72. <dependency>
73.     <groupId>junit</groupId>
74.     <artifactId>junit</artifactId>
75.     <version>4.12</version>
76. </dependency>
77. <dependency>
78.     <groupId>mysql</groupId>
79.     <artifactId>mysql-connector-java</artifactId>
80.     <version>8.0.19</version>
81. </dependency>
82.
83. <dependency>
84.     <groupId>ch.qos.logback</groupId>
85.     <artifactId>logback-core</artifactId>
86.
87. </dependency>
88. <dependency>
89.     <groupId>org.mybatis.spring.boot</groupId>
90.     <artifactId>mybatis-spring-boot-starter</artifactId>
91.     <version>2.2.0</version>
```



```
92.     </dependency>
93.     <!--添加 Spring Boot 的监控模块-->
94.     <!-- https://mvnrepository.com/artifact/org.springframework.boot/spring-boot-starter-actuator -->
95.     <dependency>
96.         <groupId>org.springframework.boot</groupId>
97.         <artifactId>spring-boot-starter-actuator</artifactId>
98.     </dependency>
99.     <!--SpringCloud alibaba sentinel -->
100.    <dependency>
101.        <groupId>com.alibaba.cloud</groupId>
102.        <artifactId>spring-cloud-starter-alibaba-sentinel</artifactId>
103.    </dependency>
104.    <!--配置中心 nacos-->
105.    <dependency>
106.        <groupId>com.alibaba.cloud</groupId>
107.        <artifactId>spring-cloud-starter-alibaba-nacos-config</artifactId>
108.    </dependency>
109. </dependencies>
110.
111. <build>
112.     <plugins>
113.         <!--mybatis自动生成代码插件-->
114.         <plugin>
115.             <groupId>org.mybatis.generator</groupId>
116.             <artifactId>mybatis-generator-maven-plugin</artifactId>
117.             <version>1.4.0</version>
118.             <configuration>
119.                 <configurationFile>src/main/resources/mybatis-generator/generatorConfig.xml</configurationFile>
120.                 <verbose>true</verbose>
121.                 <!-- 是否覆盖, true表示会替换生成的JAVA文件, false则不覆盖 -->
122.                 <overwrite>true</overwrite>
123.             </configuration>
124.             <dependencies>
125.                 <!--mysql驱动包-->
126.                 <dependency>
127.                     <groupId>mysql</groupId>
128.                     <artifactId>mysql-connector-java</artifactId>
129.                     <version>8.0.19</version>
130.                 </dependency>
131.                 <dependency>
132.                     <groupId>org.mybatis.generator</groupId>
133.                     <artifactId>mybatis-generator-core</artifactId>
134.                     <version>1.4.0</version>
135.                 </dependency>
136.             </dependencies>
137.         </plugin>
```



```
138. <plugin>
139.     <groupId>org.springframework.boot</groupId>
140.     <artifactId>spring-boot-maven-plugin</artifactId>
141.   </plugin>
142. </plugins>
143. </build>
144. </project>
```

3. 在 spring-cloud-alibaba-seata-storage-8006 的类路径 (/resources 目录) 下, 创建一个配置文件 bootstrap.yml, 配置内容如下。

```
01. spring:
02.   cloud:
03.     ## Nacos认证信息
04.     nacos:
05.       config:
06.         username: nacos
07.         password: nacos
08.         context-path: /nacos
09.         server-addr: 127.0.0.1:1111 # 设置配置中心服务端地址
10.        namespace: # Nacos 配置中心的namespace。需要注意, 如果使用 public 的 namcespace , 请不要填写这个值, 直接留空即可
```

4. 在 spring-cloud-alibaba-seata-storage-8006 的类路径 (/resources 目录) 下, 创建一个配置文件 application.yml, 配置内容如下。

```
01. spring:
02.   application:
03.     name: spring-cloud-alibaba-seata-storage-8006
04.
05.   datasource:
06.     driver-class-name: com.mysql.jdbc.Driver
07.     name: defaultDataSource
08.     url: jdbc:mysql://localhost:3306/seata_storage?serverTimezone=UTC
09.     username: root
10.     password: root
11.
12.   cloud:
13.     nacos:
14.       discovery:
15.         server-addr: 127.0.0.1:1111
16.         namespace: public
17.         username:
18.         password:
19.       sentinel:
20.         transport:
21.           dashboard: 127.0.0.1:8080
22.           port: 8719
23.
```



```
24.     alibaba:
25.         seata:
26.             tx-service-group: service-storage-group
27.
28.     server:
29.         port: 8006
30.     seata:
31.         application-id: ${spring.application.name}
32.         tx-service-group: service-storage-group
33.
34.     service:
35.         grouplist:
36.             seata-server: 127.0.0.1:8091
37.
38.     registry:
39.         type: nacos
40.     nacos:
41.         server-addr: 127.0.0.1:1111
42.
43.     config:
44.         type: nacos
45.     nacos:
46.         server-addr: 127.0.0.1:1111
47.
48.     feign:
49.         sentinel:
50.             enabled: true
51.
52.     management:
53.         endpoints:
54.             web:
55.                 exposure:
56.                     include: "*"
57. ##### MyBatis 配置 #####
58. mybatis:
59.     # 指定 mapper.xml 的位置
60.     mapper-locations: classpath:mybatis/mapper/*.xml
61.     #扫描实体类的位置,在此处指明扫描实体类的包,在 mapper.xml 中就可以不写实体类的全路径名
62.     type-aliases-package: net.biancheng.c.entity
63.     configuration:
64.         #默认开启驼峰命名法, 可以不用设置该属性
65.         map-underscore-to-camel-case: true
```

5. 在 net.biancheng.c.entity 包下, 创建一个名为 Storage 的实体类, 代码如下。

```
01. package net.biancheng.c.entity;
```



```
02.  
03. public class Storage {  
04.     private Long id;  
05.  
06.     private Long productId;  
07.  
08.     private Integer total;  
09.  
10.    private Integer used;  
11.  
12.    private Integer residue;  
13.  
14.    public Long getId() {  
15.        return id;  
16.    }  
17.  
18.    public void setId(Long id) {  
19.        this.id = id;  
20.    }  
21.  
22.    public Long getProductId() {  
23.        return productId;  
24.    }  
25.  
26.    public void setProductId(Long productId) {  
27.        this.productId = productId;  
28.    }  
29.  
30.    public Integer getTotal() {  
31.        return total;  
32.    }  
33.  
34.    public void setTotal(Integer total) {  
35.        this.total = total;  
36.    }  
37.  
38.    public Integer getUsed() {  
39.        return used;  
40.    }  
41.  
42.    public void setUsed(Integer used) {  
43.        this.used = used;  
44.    }  
45.  
46.    public Integer getResidue() {  
47.        return residue;
```



```
48.     }
49.
50.     public void setResidue(Integer residue) {
51.         this.residue = residue;
52.     }
53. }
```

6. 在 net.biancheng.c.mapper 包下，创建一个名为 StorageMapper 的接口，代码如下。

```
01. package net.biancheng.c.mapper;
02.
03. import net.biancheng.c.entity.Storage;
04. import org.apache.ibatis.annotations.Mapper;
05.
06. @Mapper
07. public interface StorageMapper {
08.     Storage selectByProductId(Long productId);
09.
10.    int decrease(Storage record);
11. }
```

7. 在 /resources/mybatis/mapper 目录下，创建一个名为 StorageMapper.xml 的 MyBatis 映射文件，代码如下。

```
01. <?xml version="1.0" encoding="UTF-8"?>
02. <!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN" "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
03. <mapper namespace="net.biancheng.c.mapper.StorageMapper">
04.     <resultMap id="BaseResultMap" type="net.biancheng.c.entity.Storage">
05.         <id column="id" jdbcType="BIGINT" property="id"/>
06.         <result column="product_id" jdbcType="BIGINT" property="productId"/>
07.         <result column="total" jdbcType="INTEGER" property="total"/>
08.         <result column="used" jdbcType="INTEGER" property="used"/>
09.         <result column="residue" jdbcType="INTEGER" property="residue"/>
10.     </resultMap>
11.     <sql id="Base_Column_List">
12.         id
13.         , product_id, total, used, residue
14.     </sql>
15.     <update id="decrease" parameterType="net.biancheng.c.entity.Storage">
16.         update t_storage
17.         <set>
18.             <if test="total != null">
19.                 total = #{total, jdbcType=INTEGER},
20.             </if>
21.             <if test="used != null">
22.                 used = #{used, jdbcType=INTEGER},
23.             </if>
```



```
24.     <if test="residue != null">
25.         residue = #{residue, jdbcType=INTEGER},
26.     </if>
27.     </set>
28.     where product_id = #{productId, jdbcType=BIGINT}
29. </update>
30.
31. <select id="selectByProductId" parameterType="java.lang.Long" resultMap="BaseResultMap">
32.     select
33.     <include refid="Base_Column_List"/>
34.     from t_storage
35.     where product_id = #{productId, jdbcType=BIGINT}
36. </select>
37. </mapper>
```

8. 在 net.biancheng.c.service 包下，创建一个名为 StorageService 的 Service 接口，代码如下。

```
01. package net.biancheng.c.service;
02.
03. public interface StorageService {
04.     int decrease(Long productId, Integer count);
05. }
```

9. 在 net.biancheng.c.service.impl 包下，创建 StorageService 的实现类 StorageServiceImpl，代码如下。

```
01. package net.biancheng.c.service.impl;
02.
03. import lombok.extern.slf4j.Slf4j;
04. import net.biancheng.c.entity.Storage;
05. import net.biancheng.c.mapper.StorageMapper;
06. import net.biancheng.c.service.StorageService;
07. import org.springframework.stereotype.Service;
08.
09. import javax.annotation.Resource;
10.
11. @Service
12. @Slf4j
13. public class StorageServiceImpl implements StorageService {
14.     @Resource
15.     StorageMapper storageMapper;
16.
17.     @Override
18.     public int decrease(Long productId, Integer count) {
19.         log.info("----->storage-service中扣减库存开始");
20.         log.info("----->storage-service 开始查询商品是否存在");
21.         Storage storage = storageMapper.selectByProductId(productId);
```



```
22.     if (storage != null && storage.getResidue().intValue() >= count.intValue()) {
23.         Storage storage2 = new Storage();
24.         storage2.setProductId(productId);
25.         storage.setUsed(storage.getUsed() + count);
26.         storage.setResidue(storage.getTotal().intValue() - storage.getUsed());
27.         int decrease = storageMapper.decrease(storage);
28.         log.info("----->storage-service 扣减库存成功");
29.         return decrease;
30.     } else {
31.         log.info("----->storage-service 库存不足, 开始回滚!");
32.         throw new RuntimeException("库存不足, 扣减库存失败!");
33.     }
34. }
35. }
```

10. 在 net.biancheng.c.controller 包下，创建一个名为 StorageController 的 Controller 类，代码如下。

```
01. package net.biancheng.c.controller;
02.
03. import lombok.extern.slf4j.Slf4j;
04. import net.biancheng.c.entity.CommonResult;
05. import net.biancheng.c.service.StorageService;
06. import org.springframework.beans.factory.annotation.Value;
07. import org.springframework.web.bind.annotation.*;
08.
09. import javax.annotation.Resource;
10.
11. @RestController
12. @Slf4j
13. public class StorageController {
14.     @Resource
15.     private StorageService storageService;
16.     @Value("${server.port}")
17.     private String serverPort;
18.
19.     @PostMapping(value = "/storage/decrease")
20.     CommonResult decrease(@RequestParam("productId") Long productId, @RequestParam("count") Integer count) {
21.         int decrease = storageService.decrease(productId, count);
22.         CommonResult result;
23.         if (decrease > 0) {
24.             result = new CommonResult(200, "from mysql, serverPort: " + serverPort, decrease);
25.         } else {
26.             result = new CommonResult(505, "from mysql, serverPort: " + serverPort, "库存扣减失败");
27.         }
28.         return result;
29.     }
}
```



30. }

11. 主启动类的代码如下。

```
01. package net.biancheng.c;
02.
03. import org.springframework.boot.SpringApplication;
04. import org.springframework.boot.autoconfigure.SpringBootApplication;
05.
06. import org.springframework.cloud.client.discovery.EnableDiscoveryClient;
07. import org.springframework.cloud.openfeign.EnableFeignClients;
08.
09. @EnableDiscoveryClient
10. @EnableFeignClients
11. @SpringBootApplication(scanBasePackages = "net.biancheng")
12. public class SpringCloudAlibabaSeataStorage8006Application {
13.
14.     public static void main(String[] args) {
15.         SpringApplication.run(SpringCloudAlibabaSeataStorage8006Application.class, args);
16.     }
17.
18. }
```

### 搭建账户 (Account) 服务

1. 在 MySQL 数据库中，新建一个名为 seata-account 的数据库实例，并通过以下 SQL 语句创建 2 张表：t\_account (账户表) 和 undo\_log (回滚日志表)。

```
DROP TABLE IF EXISTS `t_account`;
CREATE TABLE `t_account` (
    `id` bigint NOT NULL AUTO_INCREMENT COMMENT 'id',
    `user_id` bigint DEFAULT NULL COMMENT '用户id',
    `total` decimal(10, 0) DEFAULT NULL COMMENT '总额度',
    `used` decimal(10, 0) DEFAULT NULL COMMENT '已用余额',
    `residue` decimal(10, 0) DEFAULT '0' COMMENT '剩余可用额度',
    PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=2 DEFAULT CHARSET=utf8;

-- 
-- Records of t_account
-- 

INSERT INTO `t_account` VALUES ('1', '1', '1000', '0', '1000');

-- 
-- Table structure for undo_log
-- 

DROP TABLE IF EXISTS `undo_log`;
```



```

CREATE TABLE `undo_log` (
  `branch_id` bigint NOT NULL COMMENT 'branch transaction id',
  `xid` varchar(128) NOT NULL COMMENT 'global transaction id',
  `context` varchar(128) NOT NULL COMMENT 'undo_log context, such as serialization',
  `rollback_info` longblob NOT NULL COMMENT 'rollback info',
  `log_status` int NOT NULL COMMENT '0:normal status,1:defense status',
  `log_created` datetime(6) NOT NULL COMMENT 'create datetime',
  `log_modified` datetime(6) NOT NULL COMMENT 'modify datetime',
  UNIQUE KEY `ux_undo_log` (`xid`, `branch_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

```

2. 在主启动类 spring-cloud-alibaba-seata-account-8007 下，创建一个名为 spring-cloud-alibaba-seata-account-8007 的 Spring Boot 模块，并在其 pom.xml 中添加以下依赖，代码如下。

```

01. <?xml version="1.0" encoding="UTF-8"?>
02. <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
03.   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">
04.   <modelVersion>4.0.0</modelVersion>
05.   <parent>
06.     <groupId>net.biancheng.c</groupId>
07.     <version>1.0-SNAPSHOT</version>
08.     <artifactId>spring-cloud-alibaba-demo</artifactId>
09.   </parent>
10.
11.   <groupId>net.biancheng.c</groupId>
12.   <artifactId>spring-cloud-alibaba-seata-account-8007</artifactId>
13.   <version>0.0.1-SNAPSHOT</version>
14.   <name>spring-cloud-alibaba-seata-account-8007</name>
15.   <description>Demo project for Spring Boot</description>
16.   <properties>
17.     <java.version>1.8</java.version>
18.     <seata.version>1.4.2</seata.version>
19.   </properties>
20.   <dependencies>
21.     <!--nacos 服务注册中心-->
22.     <dependency>
23.       <groupId>com.alibaba.cloud</groupId>
24.       <artifactId>spring-cloud-starter-alibaba-nacos-discovery</artifactId>
25.       <exclusions>
26.         <exclusion>
27.           <groupId>org.springframework.cloud</groupId>
28.           <artifactId>spring-cloud-starter-netflix-ribbon</artifactId>
29.         </exclusion>
30.       </exclusions>
31.     </dependency>
32.     <dependency>

```



```
33.      <groupId>org.springframework.boot</groupId>
34.      <artifactId>spring-boot-starter-web</artifactId>
35.    </dependency>
36.    <!--引入 OpenFeign 的依赖-->
37.    <dependency>
38.      <groupId>org.springframework.cloud</groupId>
39.      <artifactId>spring-cloud-starter-openfeign</artifactId>
40.    </dependency>
41.    <dependency>
42.      <groupId>org.springframework.cloud</groupId>
43.      <artifactId>spring-cloud-loadbalancer</artifactId>
44.    </dependency>
45.    <!--seata-->
46.    <dependency>
47.      <groupId>com.alibaba.cloud</groupId>
48.      <artifactId>spring-cloud-starter-alibaba-seata</artifactId>
49.    </dependency>
50.
51.    <dependency>
52.      <groupId>org.springframework.boot</groupId>
53.      <artifactId>spring-boot-devtools</artifactId>
54.      <scope>runtime</scope>
55.      <optional>true</optional>
56.    </dependency>
57.    <dependency>
58.      <groupId>org.projectlombok</groupId>
59.      <artifactId>lombok</artifactId>
60.      <optional>true</optional>
61.    </dependency>
62.    <dependency>
63.      <groupId>org.springframework.boot</groupId>
64.      <artifactId>spring-boot-starter-test</artifactId>
65.      <scope>test</scope>
66.    </dependency>
67.    <dependency>
68.      <groupId>net.biancheng.c</groupId>
69.      <artifactId>spring-cloud-alibaba-api</artifactId>
70.      <version>${project.version}</version>
71.    </dependency>
72.
73.    <dependency>
74.      <groupId>junit</groupId>
75.      <artifactId>junit</artifactId>
76.      <version>4.12</version>
77.    </dependency>
78.    <dependency>
```



```
79.      <groupId>mysql</groupId>
80.      <artifactId>mysql-connector-java</artifactId>
81.      <version>8.0.19</version>
82.    </dependency>
83.
84.    <dependency>
85.      <groupId>ch.qos.logback</groupId>
86.      <artifactId>logback-core</artifactId>
87.
88.    </dependency>
89.    <dependency>
90.      <groupId>org.mybatis.spring.boot</groupId>
91.      <artifactId>mybatis-spring-boot-starter</artifactId>
92.      <version>2.2.0</version>
93.    </dependency>
94.    <!--添加 Spring Boot 的监控模块-->
95.    <!-- https://mvnrepository.com/artifact/org.springframework.boot/spring-boot-starter-actuator -->
96.    <dependency>
97.      <groupId>org.springframework.boot</groupId>
98.      <artifactId>spring-boot-starter-actuator</artifactId>
99.    </dependency>
100.   <!--SpringCloud alibaba sentinel -->
101.   <dependency>
102.     <groupId>com.alibaba.cloud</groupId>
103.     <artifactId>spring-cloud-starter-alibaba-sentinel</artifactId>
104.   </dependency>
105.   <dependency>
106.     <groupId>com.alibaba.cloud</groupId>
107.     <artifactId>spring-cloud-starter-alibaba-nacos-config</artifactId>
108.   </dependency>
109. </dependencies>
110.
111. <build>
112.   <plugins>
113.     <!--mybatis自动生成代码插件-->
114.     <plugin>
115.       <groupId>org.mybatis.generator</groupId>
116.       <artifactId>mybatis-generator-maven-plugin</artifactId>
117.       <version>1.4.0</version>
118.       <configuration>
119.         <configurationFile>src/main/resources/mybatis-generator/generatorConfig.xml</configurationFile>
120.         <verbose>true</verbose>
121.         <!-- 是否覆盖, true表示会替换生成的JAVA文件, false则不覆盖 -->
122.         <overwrite>true</overwrite>
123.       </configuration>
124.     </dependencies>
```



```
125.         <!--mysql驱动包-->
126.         <dependency>
127.             <groupId>mysql</groupId>
128.             <artifactId>mysql-connector-java</artifactId>
129.             <version>8.0.19</version>
130.         </dependency>
131.         <dependency>
132.             <groupId>org.mybatis.generator</groupId>
133.             <artifactId>mybatis-generator-core</artifactId>
134.             <version>1.4.0</version>
135.         </dependency>
136.     </dependencies>
137.     </plugin>
138.     <plugin>
139.         <groupId>org.springframework.boot</groupId>
140.         <artifactId>spring-boot-maven-plugin</artifactId>
141.     </plugin>
142. </plugins>
143. </build>
144. </project>
```

3. 在 spring-cloud-alibaba-seata-account-8007 的类路径 (/resources 目录) 下, 创建一个配置文件 bootstrap.yml, 配置内容如下。

```
01. spring:
02.   cloud:
03.     ## Nacos认证信息
04.     nacos:
05.       config:
06.         username: nacos
07.         password: nacos
08.         context-path: /nacos
09.         server-addr: 127.0.0.1:1111 # 设置配置中心服务端地址
10.       namespace: # Nacos 配置中心的namespace。需要注意, 如果使用 public 的 namespace , 请不要填写这个值, 直接留空即可
```

4. 在 spring-cloud-alibaba-seata-account-8007 的类路径 (/resources 目录) 下, 创建一个配置文件 application.yml, 配置内容如下。

```
01. spring:
02.   application:
03.     name: spring-cloud-alibaba-seata-account-8007
04.
05.   datasource:
06.     driver-class-name: com.mysql.cj.jdbc.Driver
07.     name: defaultDataSource
08.     url: jdbc:mysql://localhost:3306/seata_account?serverTimezone=UTC
09.     username: root
10.     password: root
```



```
11.
12.   cloud:
13.     nacos:
14.       discovery:
15.         server-addr: 127.0.0.1:1111
16.         namespace: public
17.       username:
18.       password:
19.     sentinel:
20.       transport:
21.         dashboard: 127.0.0.1:8080
22.         port: 8719
23.
24.   alibaba:
25.     seata:
26.       tx-service-group: service-account-group
27.
28.   server:
29.     port: 8007
30.   seata:
31.     application-id: ${spring.application.name}
32.     tx-service-group: service-account-group
33.
34.   service:
35.     grouplist:
36.       seata-server: 127.0.0.1:8091
37.
38.   registry:
39.     type: nacos
40.   nacos:
41.     server-addr: 127.0.0.1:1111
42.
43.   config:
44.     type: nacos
45.   nacos:
46.     server-addr: 127.0.0.1:1111
47.
48.   feign:
49.     sentinel:
50.       enabled: true
51.
52.   management:
53.     endpoints:
54.       web:
55.         exposure:
56.           include: "*"
```



```
57.  
58. ##### MyBatis 配置 #####  
59. mybatis:  
60.   # 指定 mapper.xml 的位置  
61.   mapper-locations: classpath:mybatis/mapper/*.xml  
62.   #扫描实体类的位置,在此处指明扫描实体类的包,在 mapper.xml 中就可以不写实体类的全路径名  
63.   type-aliases-package: net.biancheng.c.entity  
64. configuration:  
65.   #默认开启驼峰命名法, 可以不用设置该属性  
66.   map-underscore-to-camel-case: true
```

5. 在 net.biancheng.c.entity 包下, 创建一个名为 Account 的实体类, 代码如下。

```
01. package net.biancheng.c.entity;  
02.  
03. import java.math.BigDecimal;  
04.  
05. public class Account {  
06.     private Long id;  
07.  
08.     private Long userId;  
09.  
10.    private BigDecimal total;  
11.  
12.    private BigDecimal used;  
13.  
14.    private BigDecimal residue;  
15.  
16.    public Long getId() {  
17.        return id;  
18.    }  
19.  
20.    public void setId(Long id) {  
21.        this.id = id;  
22.    }  
23.  
24.    public Long getUserId() {  
25.        return userId;  
26.    }  
27.  
28.    public void setUserId(Long userId) {  
29.        this.userId = userId;  
30.    }  
31.  
32.    public BigDecimal getTotal() {  
33.        return total;
```



```
34.     }
35.
36.     public void setTotal(BigDecimal total) {
37.         this.total = total;
38.     }
39.
40.     public BigDecimal getUsed() {
41.         return used;
42.     }
43.
44.     public void setUsed(BigDecimal used) {
45.         this.used = used;
46.     }
47.
48.     public BigDecimal getResidue() {
49.         return residue;
50.     }
51.
52.     public void setResidue(BigDecimal residue) {
53.         this.residue = residue;
54.     }
55. }
```

6. 在 net.biancheng.c.mapper 包下，创建一个名为 AccountMapper 的接口，代码如下。

```
01. package net.biancheng.c.mapper;
02.
03. import net.biancheng.c.entity.Account;
04. import org.apache.ibatis.annotations.Mapper;
05.
06. import java.math.BigDecimal;
07.
08. @Mapper
09. public interface AccountMapper {
10.     Account selectById(Long userId);
11.
12.     int decrease(Long userId, BigDecimal money);
13. }
```

7. 在 /resources/mybatis/mapper 目录下，创建一个名为 AccountMapper.xml 的 MyBatis 映射文件，代码如下。

```
01. <?xml version="1.0" encoding="UTF-8"?>
02. <!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN" "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
03. <mapper namespace="net.biancheng.c.mapper.AccountMapper">
04.     <resultMap id="BaseResultMap" type="net.biancheng.c.entity.Account">
05.         <id column="id" jdbcType="BIGINT" property="id"/>
```



```

06.      <result column="user_id" jdbcType="BIGINT" property="userId"/>
07.      <result column="total" jdbcType="DECIMAL" property="total"/>
08.      <result column="used" jdbcType="DECIMAL" property="used"/>
09.      <result column="residue" jdbcType="DECIMAL" property="residue"/>
10.    </resultMap>
11.    <sql id="Base_Column_List">
12.      id
13.      , user_id, total, used, residue
14.    </sql>
15.    <select id="selectByUserId" resultType="net.biancheng.c.entity.Account">
16.      select
17.        <include refid="Base_Column_List"/>
18.      from t_account
19.      where user_id = #{userId, jdbcType=BIGINT}
20.    </select>
21.    <update id="decrease">
22.      UPDATE t_account
23.      SET residue = residue - #{money},
24.          used     = used + #{money}
25.      WHERE user_id = #{userId};
26.    </update>
27.  </mapper>

```

8. 在 net.biancheng.c.service 包下，创建一个名为 AccountService 的接口，代码如下。

```

01. package net.biancheng.c.service;
02. import org.springframework.web.bind.annotation.RequestParam;
03. import java.math.BigDecimal;
04.
05. public interface AccountService {
06.
07.     /**
08.      * 扣减账户余额
09.      *
10.      * @param userId 用户id
11.      * @param money 金额
12.      */
13.     int decrease(@RequestParam("userId") Long userId, @RequestParam("money") BigDecimal money);
14. }

```

9. 在 net.biancheng.c.service.impl 包下，创建 AccountService 的实现类 AccountServiceImpl，代码如下。

```

01. package net.biancheng.c.service.impl;
02.
03. import lombok.extern.slf4j.Slf4j;
04. import net.biancheng.c.entity.Account;

```



```
05. import net.biancheng.c.mapper.AccountMapper;
06. import net.biancheng.c.service.AccountService;
07. import org.springframework.stereotype.Service;
08.
09. import javax.annotation.Resource;
10. import java.math.BigDecimal;
11.
12. @Service
13. @Slf4j
14. public class AccountServiceImpl implements AccountService {
15.
16.     @Resource
17.     AccountMapper accountMapper;
18.
19.     @Override
20.     public int decrease(Long userId, BigDecimal money) {
21.
22.         log.info("----->account-service 开始查询账户余额");
23.         Account account = accountMapper.selectByUserId(userId);
24.         log.info("----->account-service 账户余额查询完成, " + account);
25.         if (account != null && account.getResidue().intValue() >= money.intValue()) {
26.             log.info("----->account-service 开始从账户余额中扣钱! ");
27.             int decrease = accountMapper.decrease(userId, money);
28.             log.info("----->account-service 从账户余额中扣钱完成");
29.             return decrease;
30.         } else {
31.             log.info("账户余额不足, 开始回滚! ");
32.             throw new RuntimeException("账户余额不足!");
33.         }
34.     }
35.
36. }
```

#### 10. 主启动类代码如下。

```
01. package net.biancheng.c;
02.
03. import org.springframework.boot.SpringApplication;
04. import org.springframework.boot.autoconfigure.SpringBootApplication;
05. import org.springframework.cloud.client.discovery.EnableDiscoveryClient;
06. import org.springframework.cloud.openfeign.EnableFeignClients;
07.
08. @EnableDiscoveryClient
09. @EnableFeignClients
10. @SpringBootApplication(scanBasePackages = "net.biancheng")
11. public class SpringCloudAlibabaSeataAccount8007Application {
```



```
12.  
13.     public static void main(String[] args) {  
14.         SpringApplication.run(SpringCloudAlibabaSeataAccount8007Application.class, args);  
15.     }  
16. }
```

11. 依次启动 Nacos Server (集群) 和 Seata Server (集群) , 最后启动 spring-cloud-alibaba-seata-order-8005, 当控制台出现以下日志时, 说明该服务已经成功连接上 Seata Server (TC) 。

```
2021-11-25 15:16:27.389 INFO 19564 --- [ restartedMain] com.zaxxer.hikari.HikariDataSource      : defaultDataSource - Start completed.  
2021-11-25 15:16:27.553 INFO 19564 --- [ restartedMain] i.s.c.r.netty.NettyClientChannelManager : will connect to 172.30.194.1:8091  
2021-11-25 15:16:27.553 INFO 19564 --- [ restartedMain] i.s.c.rpc.netty.RmNettyRemotingClient    : RM will register :jdbc:mysql://localhost:3306/seata_order  
2021-11-25 15:16:27.557 INFO 19564 --- [ restartedMain] i.s.core.rpc.netty.NettyPoolableFactory : NettyPool create channel to transactionRole:RMROLE, address:172.30.194.1:8091, msg:< RegisterRMRequest{resourceIds='jdbc:mysql://localhost:3306/seata_order', applicationId='spring-cloud-alibaba-seata-order-8005', transactionServiceGroup='service-order-group'} >  
2021-11-25 15:16:28.699 INFO 19564 --- [ restartedMain] i.s.c.rpc.netty.RmNettyRemotingClient    : register RM success. client version:1.3.0, server version:1.4.2, channel:[id: 0xc6da1cb4, L:/172.30.194.1:49945 - R:/172.30.194.1:8091]  
2021-11-25 15:16:28.707 INFO 19564 --- [ restartedMain] i.s.core.rpc.netty.NettyPoolableFactory : register success, cost 78 ms, version:1.4.2, role:RMROLE, channel:[id: 0xc6da1cb4, L:/172.30.194.1:49945 - R:/172.30.194.1:8091]
```

12. 启动 spring-cloud-alibaba-seata-storage-8006, 当控制台出现以下日志时, 说明该服务已经成功连接上 Seata Server (TC) 。

```
2021-11-25 15:16:28.621 INFO 14772 --- [ restartedMain] com.zaxxer.hikari.HikariDataSource      : defaultDataSource - Start completed.  
2021-11-25 15:16:28.969 INFO 14772 --- [ restartedMain] i.s.c.r.netty.NettyClientChannelManager : will connect to 172.30.194.1:8091  
2021-11-25 15:16:28.970 INFO 14772 --- [ restartedMain] i.s.c.rpc.netty.RmNettyRemotingClient    : RM will register :jdbc:mysql://localhost:3306/seata_storage  
2021-11-25 15:16:28.974 INFO 14772 --- [ restartedMain] i.s.core.rpc.netty.NettyPoolableFactory : NettyPool create channel to transactionRole:RMROLE, address:172.30.194.1:8091, msg:< RegisterRMRequest{resourceIds='jdbc:mysql://localhost:3306/seata_storage', applicationId='spring-cloud-alibaba-seata-storage-8006', transactionServiceGroup='service-storage-group'} >  
2021-11-25 15:16:30.171 INFO 14772 --- [ restartedMain] i.s.c.rpc.netty.RmNettyRemotingClient    : register RM success. client version:1.3.0, server version:1.4.2, channel:[id: 0x7311ae2a, L:/172.30.194.1:52026 - R:/172.30.194.1:8091]  
2021-11-25 15:16:30.182 INFO 14772 --- [ restartedMain] i.s.core.rpc.netty.NettyPoolableFactory : register success, cost 174 ms, version:1.4.2, role:RMROLE, channel:[id: 0x7311ae2a, L:/172.30.194.1:52026 - R:/172.30.194.1:8091]
```

13. 启动 spring-cloud-alibaba-seata-account-8007, 当控制台出现以下日志时, 说明该服务已经成功连接上 Seata Server (TC) 。

```
2021-11-25 15:16:29.914 INFO 8616 --- [ restartedMain] com.zaxxer.hikari.HikariDataSource      : defaultDataSource - Start completed.  
2021-11-25 15:16:30.253 INFO 8616 --- [ restartedMain] i.s.c.r.netty.NettyClientChannelManager : will connect to 172.30.194.1:8091  
2021-11-25 15:16:30.253 INFO 8616 --- [ restartedMain] i.s.c.rpc.netty.RmNettyRemotingClient    : RM will register
```



```
: jdbc:mysql://localhost:3306/seata_account
2021-11-25 15:16:30.257 INFO 8616 --- [ restartedMain] i.s.core.rpc.netty.NettyPoolableFactory : NettyPool create channel to
transactionRole:RMROLE, address:172.30.194.1:8091, msg:< RegisterRMRequest{resourceIds='jdbc:mysql://localhost:3306/seata_account',
applicationId='spring-cloud-alibaba-seata-account-8007', transactionServiceGroup='service-account-group'} >
2021-11-25 15:16:31.930 INFO 8616 --- [ restartedMain] i.s.c.rpc.netty.RmNettyRemotingClient : register RM success. client
version:1.3.0, server version:1.4.2, channel:[id: 0xa57ead6d, L:/172.30.194.1:52057 - R:/172.30.194.1:8091]
2021-11-25 15:16:31.941 INFO 8616 --- [ restartedMain] i.s.core.rpc.netty.NettyPoolableFactory : register success, cost 114 ms,
version:1.4.2, role:RMROLE, channel:[id: 0xa57ead6d, L:/172.30.194.1:52057 - R:/172.30.194.1:8091]
```

14. 使用浏览器访问 “<http://localhost:8005/order/createByAnnotation/1/2/20>” , 结果如下。

```
{"code":200,"message":"订单创建成功","data":null}
```

15. 执行以下 SQL 语句, 查询 seata\_order 数据库中的 t\_order 表。

```
SELECT * FROM seata_order.t_order;
```

结果如下。

| id | user_id | product_id | count | money | status |
|----|---------|------------|-------|-------|--------|
| 1  |         | 1          | 2     | 20    | 1      |

从上表可以看出, 已经创建一条订单数据, 且订单状态 (status) 已修改为 “已完成” 。

16. 执行以下 SQL 语句, 查询 seata\_storage 数据库中的 t\_storage 表。

```
SELECT * FROM seata_storage.t_storage;
```

结果如下。

| id | product_id | total | used | residue |
|----|------------|-------|------|---------|
| 1  | 1          | 100   | 2    | 98      |

从上表可以看出, 商品库存已经扣减 2 件, 仓库中商品储量从 100 件减少到了 98 件。

17. 执行以下 SQL 语句, 查询 seata\_account 数据库中的 t\_account 表,

```
SELECT * FROM seata_account.t_account;
```

结果如下。



| <b>id</b> | <b>user_id</b> | <b>total</b> | <b>used</b> | <b>residue</b> |
|-----------|----------------|--------------|-------------|----------------|
| 1         | 1              | 1000         | 20          | 980            |

从上表可以看出，账户余额已经扣减，金额从 1000 元减少到了 980 元。

18. 使用浏览器访问 “<http://localhost:8005/order/createByAnnotation/1/10/1000>” , 结果如下。

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

Thu Nov 25 15:27:03 CST 2021
There was an unexpected error (type=Internal Server Error, status=500).
[500] during [POST] to [http://spring-cloud-alibaba-seata-account-8007/account/decrease?userId=1&money=1000]
[AccountService#decrease(Long, BigDecimal)]: [{"timestamp": "2021-11-25T07:27:03.512+00:00", "status": 500, "error": "Internal Server
Error", "trace": "java.lang.RuntimeException: 账户余额不足! \r\n\tat net.biancheng.c.service.impl.AccountServiceImpl.decrease(... (5673
bytes)"}]
feign.FeignException$InternalServerError: [500] during [POST] to [http://spring-cloud-alibaba-seata-account-8007/account/decrease?
userId=1&money=1000] [AccountService#decrease(Long, BigDecimal)]: [{"timestamp": "2021-11-
25T07:27:03.512+00:00", "status": 500, "error": "Internal Server Error", "trace": "java.lang.RuntimeException: 账户余额不足! \r\n\tat
net.biancheng.c.service.impl.AccountServiceImpl.decrease(... (5673 bytes)"}]
```

注：在本次请求中，用户购买 10 件商品（商品 ID 为 1），商品总价为 1000 元，此时用户账户余额为 980 元，因此账户服务会抛出“账户余额不足！”的运行时异常。

19. 再次查询 t\_order、t\_storage 和 t\_account 表，结果如下。

| t_order (订单表) |           |                |                   |              |              |               |
|---------------|-----------|----------------|-------------------|--------------|--------------|---------------|
|               | <b>id</b> | <b>user_id</b> | <b>product_id</b> | <b>count</b> | <b>money</b> | <b>status</b> |
| 1             | 1         | 1              | 1                 | 2            | 20           | 1             |
| 2             | 2         | 1              | 1                 | 10           | 1000         | 0             |

| t_storage (仓储表) |           |                   |              |             |                |
|-----------------|-----------|-------------------|--------------|-------------|----------------|
|                 | <b>id</b> | <b>product_id</b> | <b>total</b> | <b>used</b> | <b>residue</b> |
| 1               | 1         | 1                 | 100          | 12          | 88             |

| t_account (账户表) |           |                |              |             |                |
|-----------------|-----------|----------------|--------------|-------------|----------------|
|                 | <b>id</b> | <b>user_id</b> | <b>total</b> | <b>used</b> | <b>residue</b> |
| 1               | 1         | 1              | 1000         | 20          | 980            |

图11：分布式事务问题

从上图可以看出：

- t\_order (订单表)：订单服务生成了一条订单数据 (id 为 2)，但订单状态 (status) 为未完成 (0)；
- t\_storage (库存表)：商品库存已扣减；
- t\_account (账户表)：账户金额不足，账户 (Account) 服务出现异常，进而导致账户金额并未扣减。



## @GlobalTransactional 注解

在分布式微服务架构中，我们可以使用 Seata 提供的 @GlobalTransactional 注解实现分布式事务的开启、管理和控制。

当调用 @GlobalTransaction 注解的方法时，TM 会先向 TC 注册全局事务，TC 生成一个全局唯一的 XID，返回给 TM。

@GlobalTransactional 注解既可以在类上使用，也可以在类方法上使用，该注解的使用位置决定了全局事务的范围，具体关系如下：

- 在类中某个方法使用时，全局事务的范围就是该方法以及它所涉及的所有服务。
- 在类上使用时，全局事务的范围就是这个类中的所有方法以及这些方法涉及的服务。

接下来，我们就使用 @GlobalTransactional 注解对业务系统进行改造，步骤如下。

1. 在 spring-cloud-alibaba-seata-order-8005 的 net.biacheng.c 包下的 OrderController 中，添加一个名为 createByAnnotation 的方法，代码如下。

```
01. /**
02. * 使用 @GlobalTransactional 注解对分布式事务进行管理
03. * @param productId
04. * @param count
05. * @param money
06. * @return
07. */
08. @GetMapping("/order/createByAnnotation/{productId}/{count}/{money}")
09. @GlobalTransactional(name = "c-biancheng-net-create-order", rollbackFor = Exception.class)
10. public CommonResult createByAnnotation(@PathVariable("productId") Integer productId, @PathVariable("count") Integer count
11.     , @PathVariable("money") BigDecimal money) {
12.     Order order = new Order();
13.     order.setProductId(Integer.valueOf(productId).longValue());
14.     order.setCount(count);
15.
16.     order.setMoney(money);
17.     return orderService.create(order);
18. }
```

从以上代码可以看出，添加的 createByAnnotation() 方法与 create() 方法无论是参数还是代码逻辑都一摸一样，唯一的不同就是前者标注了 @GlobalTransactional 注解。

2. 将数据恢复到浏览器访问 “<http://localhost:8005/order/createByAnnotation/1/2/20>” 之后。

3. 重启订单 (Order) 服务、库存 (Storage) 服务和账户 (Account) 服务，并使用浏览器访问 “<http://localhost:8005/order/createByAnnotation/1/10/1000>” ，结果如下。

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

Thu Nov 25 15:27:03 CST 2021  
There was an unexpected error (type=Internal Server Error, status=500).  
[500] during [POST] to [http://spring-cloud-alibaba-seata-account-8007/account/decrease?userId=1&money=1000]



```
[AccountService#decrease(Long, BigDecimal)]: [{"timestamp": "2021-11-25T07:27:03.512+00:00", "status": 500, "error": "Internal Server Error", "trace": "java.lang.RuntimeException: 账户余额不足! \r\n\tat net.biancheng.c.service.impl.AccountServiceImpl.decrease(... (5673 bytes)]  
feign.FeignException$InternalServerError: [500] during [POST] to [http://spring-cloud-alibaba-seata-account-8007/account/decrease?userId=1&money=1000] [AccountService#decrease(Long, BigDecimal)]: [{"timestamp": "2021-11-25T07:27:03.512+00:00", "status": 500, "error": "Internal Server Error", "trace": "java.lang.RuntimeException: 账户余额不足! \r\n\tat net.biancheng.c.service.impl.AccountServiceImpl.decrease(... (5673 bytes)]
```

在本次请求中，用户购买商品的总价为 1000 元，但用户账户余额只有 980 元，因此账户服务会抛出运行时异常，异常信息为“账户余额不足！”。

#### 4. spring-cloud-alibaba-seata-order-8005 控制台输出如下（标红部分为回滚日志）。

```
2021-11-25 15:26:57.586 INFO 19564 --- [nio-8005-exec-1] n.b.c.service.impl.OrderServiceImpl : ----->开始新建订单  
2021-11-25 15:26:58.276 INFO 19564 --- [nio-8005-exec-1] n.b.c.service.impl.OrderServiceImpl : ----->订单服务开始调用库存服务，  
开始扣减库存  
2021-11-25 15:26:58.413 WARN 19564 --- [nio-8005-exec-1] c.l.c.ServiceInstanceListSupplierBuilder : LoadBalancerCacheManager not  
available, returning delegate without caching.  
2021-11-25 15:27:00.705 INFO 19564 --- [nio-8005-exec-1] n.b.c.service.impl.OrderServiceImpl : ----->订单微服务开始调用库存，扣  
减库存结束  
2021-11-25 15:27:00.705 INFO 19564 --- [nio-8005-exec-1] n.b.c.service.impl.OrderServiceImpl : ----->订单服务开始调用账户服务，  
开始从账户扣减商品金额  
2021-11-25 15:27:00.723 WARN 19564 --- [nio-8005-exec-1] c.l.c.ServiceInstanceListSupplierBuilder : LoadBalancerCacheManager not  
available, returning delegate without caching.  
2021-11-25 15:27:03.665 INFO 19564 --- [h_RMROLE_1_1_16] i.s.c.r.p.c.RmBranchRollbackProcessor : rm handle branch rollback  
process:xid=172.30.194.1:8091:2702361983450404762,branchId=2702361983450404764,branchType=AT,resourceId=jdbc:mysql://localhost:3306/seata_order,applicationData=null  
2021-11-25 15:27:03.670 INFO 19564 --- [h_RMROLE_1_1_16] io.seata.rm.AbstractRMHandler : Branch Rollbacking:  
172.30.194.1:8091:2702361983450404762 2702361983450404764 jdbc:mysql://localhost:3306/seata_order  
2021-11-25 15:27:03.738 INFO 19564 --- [h_RMROLE_1_1_16] i.s.r.d.undo.AbstractUndoLogManager : xid  
172.30.194.1:8091:2702361983450404762 branch 2702361983450404764, undo_log deleted with GlobalFinished  
2021-11-25 15:27:03.742 INFO 19564 --- [h_RMROLE_1_1_16] io.seata.rm.AbstractRMHandler : Branch Rollbacked result:  
PhaseTwo_Rollbacked  
2021-11-25 15:27:03.817 INFO 19564 --- [nio-8005-exec-1] i.seata.tm.api.DefaultGlobalTransaction :  
[172.30.194.1:8091:2702361983450404762] rollback status: Rollbacked  
2021-11-25 15:27:03.853 ERROR 19564 --- [nio-8005-exec-1] o.a.c.c.C.[.[].[dispatcherServlet] : Servlet.service() for servlet  
[dispatcherServlet] in context with path [] threw exception [Request processing failed; nested exception is  
feign.FeignException$InternalServerError: [500] during [POST] to [http://spring-cloud-alibaba-seata-account-8007/account/decrease?userId=1&money=1000] [AccountService#decrease(Long, BigDecimal)]: [{"timestamp": "2021-11-25T07:27:03.512+00:00", "status": 500, "error": "Internal Server Error", "trace": "java.lang.RuntimeException: 账户余额不足! \r\n\tat net.biancheng.c.service.impl.AccountServiceImpl.decrease(... (5673 bytes)] with root cause
```

#### 5. spring-cloud-alibaba-seata-storage-8006 控制台输出如下。（标红部分为回滚日志）

```
2021-11-25 15:26:59.315 INFO 14772 --- [nio-8006-exec-1] n.b.c.service.impl.StorageServiceImpl : ----->storage-service中扣减库  
存开始
```



```

2021-11-25 15:26:59.316 INFO 14772 --- [nio-8006-exec-1] n.b.c.service.impl.StorageServiceImpl : ----->storage-service 开始查询商品是否存在
2021-11-25 15:27:00.652 INFO 14772 --- [nio-8006-exec-1] n.b.c.service.impl.StorageServiceImpl : ----->storage-service 扣减库存成功
2021-11-25 15:27:03.568 INFO 14772 --- [h_RMROLE_1_1_16] i.s.c.r.p.c.RmBranchRollbackProcessor : rm handle branch rollback
process:xid=172.30.194.1:8091:2702361983450404762,branchId=2702361983450404769,branchType=AT,resourceId=jdbc:mysql://localhost:3306/seata_storage,applicationData=null
2021-11-25 15:27:03.572 INFO 14772 --- [h_RMROLE_1_1_16] io.seata.rm.AbstractRMHandler : Branch Rollbacking:
172.30.194.1:8091:2702361983450404762 2702361983450404769 jdbc:mysql://localhost:3306/seata_storage
2021-11-25 15:27:03.631 INFO 14772 --- [h_RMROLE_1_1_16] i.s.r.d.undo.AbstractUndoLogManager : xid
172.30.194.1:8091:2702361983450404762 branch 2702361983450404769, undo_log deleted with GlobalFinished
2021-11-25 15:27:03.635 INFO 14772 --- [h_RMROLE_1_1_16] io.seata.rm.AbstractRMHandler : Branch Rollbacked result:
PhaseTwo_Rollbacked

```

6. spring-cloud-alibaba-seata-account-8007 控制台输出如下 (标红部分为回滚日志)。

```

2021-11-25 15:27:03.366 INFO 8616 --- [nio-8007-exec-1] n.b.c.service.impl.AccountServiceImpl : ----->account-service 开始查询账户余额
2021-11-25 15:27:03.484 INFO 8616 --- [nio-8007-exec-1] n.b.c.service.impl.AccountServiceImpl : ----->account-service 账户余额查询完成, net.biancheng.c.entity.Account@2a95537f
2021-11-25 15:27:03.485 INFO 8616 --- [nio-8007-exec-1] n.b.c.service.impl.AccountServiceImpl : 账户余额不足, 开始回滚!
2021-11-25 15:27:03.499 ERROR 8616 --- [nio-8007-exec-1] o.a.c.c.C.[.].[dispatcherServlet] : Servlet.service() for servlet [dispatcherServlet] in context with path [] threw exception [Request processing failed; nested exception is java.lang.RuntimeException: 账户余额不足!] with root cause
java.lang.RuntimeException: 账户余额不足!

```

7. 到 MySQL 数据库中，再次查询 t\_order、t\_storage 和 t\_account 表，结果如下。

| t_order (订单表) |    |         |            |       |       |        |
|---------------|----|---------|------------|-------|-------|--------|
|               | id | user_id | product_id | count | money | status |
| 1             | 1  | 1       | 1          | 1     | 2     | 20     |

| t_storage (仓储表) |    |            |       |      |         |
|-----------------|----|------------|-------|------|---------|
|                 | id | product_id | total | used | residue |
| 1               | 1  | 1          | 100   | 2    | 98      |

| t_account (账户表) |    |         |       |      |         |
|-----------------|----|---------|-------|------|---------|
|                 | id | user_id | total | used | residue |
| 1               | 1  | 1       | 1000  | 20   | 980     |

图12: Seata 事务回滚

从图 12 可以看出，这次并没有出现分布式事务造成的数据不一致问题。

## 推荐阅读

[Go语言输出正弦函数（Sin）图像](#)

[符号——链接的粘合剂](#)

[Qt QLineEdit单行输入框用法详解](#)

[C语言fscanf\(\): 从文件中格式化读取数据](#)

[CSS :nth-of-type\(\)伪类的应用场景](#)

[Java字符流的使用：字符输入/输出流、字符文件和字符缓冲区的输入/](#)

[Gateway: Spring Cloud API网关组件（非常详细）](#)

[MySQL企业版与社区版的区别](#)

[《Web前端开发实训案例教程（初级）》PDF下载（高清完整版）](#)

[Java三元运算符（三目运算符）](#)

精美而实用的网站，分享优质编程教程，帮助有志青年。千锤百炼，只为大作；精益求精，处处斟酌；这种教程，看一眼就倾心。

[关于网站](#) | [联系我们](#) | [网站地图](#)

Copyright ©2012-2023 biancheng.net

*biancheng.net*

