

分布式事务解决方案 Seata

1. Seata 是什么

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

注：官方文档地址：<https://seata.io/zh-cn/index.html>

2. 为什么使用 Seata

拉勾教育实战项目，是典型的微服务架构，多数据源，因此好多业务操作，都是跨数据源操作，

传统的事务是不能解决跨数据源的，因此需要引入分布式事务解决。Seata 是国内阿里巴巴研发的一款高性能分布式事务中间件，只需要对应的方法上加上注解就可以解决分布式事务，不侵入业务代码，跟应用系统耦合低，使用简单，因此最终选用 Seata。

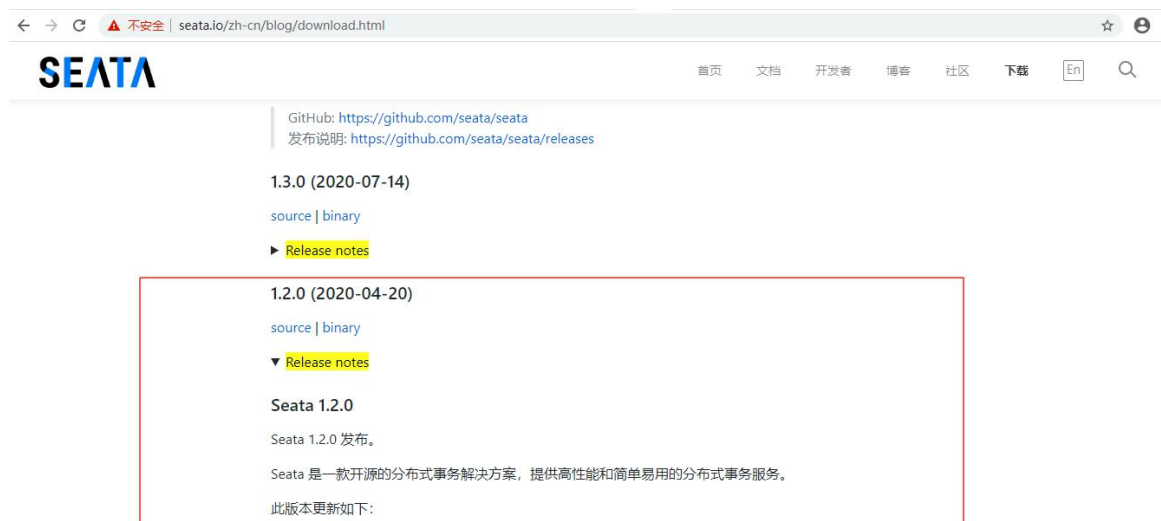
3. Seata 使用步骤

Seata 分服务端&客户端，两部分，首先需要安装 Seata 服务端，然后再配置客户端。

4. Seata 服务端安装

a. 下载 Seata

Seata 下载地址：<https://seata.io/zh-cn/docs/ops/deploy-guide-beginner.html>



目前最新版本是 1.3.0，最新版本相对来说不太稳定，因此我们往前选择一个版本，选择 1.2.0

b.上传到服务器

```
total 40350
drwxr-xr-x 5 root root    55 Apr 21 14:11 seata
-rw-r--r-- 1 root root 41301629 Jul 28 01:22 seata.zip
```

c.解压

```
[root@teacher3 seata]# unzip seata.zip
```

解压完以后看到的是这个目录

```
drwxr-xr-x 2 root root    70 Jul 28 15:21 bin
drwxr-xr-x 3 root root   141 Jul 28 15:14 conf
drwxr-xr-x 3 root root  8192 Apr 21 14:11 lib
-rw-r--r-- 1 root root 11365 May 13 2019 LICENSE
```

d.修改配置

进入 conf 目录

```
-rw-r--r-- 1 root root 11365 May 13 2019 LICENSE
[root@teacher3 seata]# cd conf
[root@teacher3 conf]# ll
total 24
-rw-r--r-- 1 root root 1238 Jul 28 15:14 file.conf
-rw-r--r-- 1 root root 2929 Apr 21 14:10 file.conf.example
-rw-r--r-- 1 root root 2152 Apr 21 14:10 logback.xml
drwxr-xr-x 3 root root    22 Apr 21 14:10 META-INF
-rw-r--r-- 1 root root 1324 Apr 21 14:10 README.md
-rw-r--r-- 1 root root 1327 Apr 21 14:10 README-zh.md
-rw-r--r-- 1 root root 1645 Jul 28 15:14 registry.conf
```

需要修改两个配置文件 file.conf 和 registry.conf

首先进入 file.conf

```

## transaction log store, only used in seata-server
store {
  ## store mode: file, db
  mode = "db"
  ## file store property
  file {
    ## store location dir
    dir = "sessionStore"
    # branch session size, if exceeded first try compress lockkey, still exceeded throws exceptions
    maxBranchSessionSize = 16384
    # globe session size, if exceeded throws exceptions
    maxGlobalSessionSize = 512
    # file buffer size, if exceeded allocate new buffer
    fileWriteBufferCacheSize = 16384
    # when recover batch read size
    sessionReLoadReadSize = 100
    # async, sync
    flushDiskMode = async
  }
  ## database store property
  db {
    ## the implement of javax.sql.DataSource, such as DruidDataSource(druid)/BasicDataSource(dbcp) etc.
    datasource = "druid"
    ## mysql/oracle/postgresql/h2/oceanbase etc.
    dbType = "mysql"
    driverClassName = "com.mysql.jdbc.Driver"
    url = "jdbc:mysql://172.16.86.47:3306/seata?characterEncoding=UTF-8&useUnicode=true&useSSL=false&serverTimezone=GMT"
    user = "edurw"
    password = "edurw"
    minConn = 5
    maxConn = 30
    globalTable = "global_table"
    branchTable = "branch_table"
  }
}

```

这里改为db

这里配置我们的数据库

需要先创建 Seata 数据库，一下是 sql 脚本

1. CREATE DATABASE IF NOT EXISTS `seata` /*!40100 DEFAULT CHARACTER SET utf8 COLLATE utf8_unicode_ci */;
2. USE `seata`;
3. -- MySQL dump 10.13 Distrib 8.0.18, for macos10.14 (x86_64)
4. --
5. -- Host: 127.0.0.1 Database: seata
6. -- -----
7. -- Server version 5.7.29-log
- 8.
9. /*!40101 SET @OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT */;
10. /*!40101 SET
11. @OLD_CHARACTER_SET_RESULTS=@@CHARACTER_SET_RESULTS */;
12. /*!40101 SET @OLD_COLLATION_CONNECTION=@@COLLATION_CONNECTION
13. */;
14. /*!50503 SET NAMES utf8 */;
15. /*!40103 SET @OLD_TIME_ZONE=@@TIME_ZONE */;
16. /*!40103 SET TIME_ZONE='+00:00' */;
17. /*!40014 SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS,
18. UNIQUE_CHECKS=0 */;
19. /*!40014 SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS,
20. FOREIGN_KEY_CHECKS=0 */;
21. /*!40101 SET @OLD_SQL_MODE=@@SQL_MODE,
22. SQL_MODE='NO_AUTO_VALUE_ON_ZERO' */;
23. /*!40111 SET @OLD_SQL_NOTES=@@SQL_NOTES, SQL_NOTES=0 */;
- 24.
25. --
26. -- Table structure for table `branch_table`
27. -- seata server 端使用 的数据库表
28. -- seata server 端使用 的数据库表
- 29.
30. DROP TABLE IF EXISTS `branch_table`;
31. /*!40101 SET @saved_cs_client = @@character_set_client */;
32. /*!50503 SET character_set_client = utf8mb4 */;
33. CREATE TABLE `branch_table` (
34. `branch_id` bigint(20) NOT NULL,

```

30. `xid` varchar(128) NOT NULL,
31. `transaction_id` bigint(20) DEFAULT NULL,
32. `resource_group_id` varchar(32) DEFAULT NULL,
33. `resource_id` varchar(256) DEFAULT NULL,
34. `lock_key` varchar(128) DEFAULT NULL,
35. `branch_type` varchar(8) DEFAULT NULL,
36. `status` tinyint(4) DEFAULT NULL,
37. `client_id` varchar(64) DEFAULT NULL,
38. `application_data` varchar(2000) DEFAULT NULL,
39. `gmt_create` datetime DEFAULT NULL,
40. `gmt_modified` datetime DEFAULT NULL,
41. PRIMARY KEY (`branch_id`),
42. KEY `idx_xid` (`xid`)
43.) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
44. /*!40101 SET character_set_client = @saved_cs_client */;
45.
46. --
47. -- Dumping data for table `branch_table`
48. --
49.
50. LOCK TABLES `branch_table` WRITE;
51. /*!40000 ALTER TABLE `branch_table` DISABLE KEYS */;
52. /*!40000 ALTER TABLE `branch_table` ENABLE KEYS */;
53. UNLOCK TABLES;
54.
55. --
56. -- Table structure for table `global_table`
57. --
58.
59. DROP TABLE IF EXISTS `global_table`;
60. /*!40101 SET @saved_cs_client = @@character_set_client */;
61. /*!50503 SET character_set_client = utf8mb4 */;
62. CREATE TABLE `global_table` (
63. `xid` varchar(128) NOT NULL,
64. `transaction_id` bigint(20) DEFAULT NULL,
65. `status` tinyint(4) NOT NULL,
66. `application_id` varchar(32) DEFAULT NULL,
67. `transaction_service_group` varchar(32) DEFAULT NULL,
68. `transaction_name` varchar(128) DEFAULT NULL,
69. `timeout` int(11) DEFAULT NULL,
70. `begin_time` bigint(20) DEFAULT NULL,
71. `application_data` varchar(2000) DEFAULT NULL,
72. `gmt_create` datetime DEFAULT NULL,
73. `gmt_modified` datetime DEFAULT NULL,
74. PRIMARY KEY (`xid`),
75. KEY `idx_gmt_modified_status` (`gmt_modified`,`status`),
76. KEY `idx_transaction_id` (`transaction_id`)
77.) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
78. /*!40101 SET character_set_client = @saved_cs_client */;
79.
80. --

```

```

81. -- Dumping data for table `global_table`
82. --
83.
84. LOCK TABLES `global_table` WRITE;
85. /*!40000 ALTER TABLE `global_table` DISABLE KEYS */;
86. /*!40000 ALTER TABLE `global_table` ENABLE KEYS */;
87. UNLOCK TABLES;
88.
89. --
90. -- Table structure for table `lock_table`
91. --
92.
93. DROP TABLE IF EXISTS `lock_table`;
94. /*!40101 SET @saved_cs_client = @@character_set_client */;
95. /*!50503 SET character_set_client = utf8mb4 */;
96. CREATE TABLE `lock_table` (
97.   `row_key` varchar(128) NOT NULL,
98.   `xid` varchar(96) DEFAULT NULL,
99.   `transaction_id` mediumtext,
100.  `branch_id` mediumtext,
101.  `resource_id` varchar(256) DEFAULT NULL,
102.  `table_name` varchar(32) DEFAULT NULL,
103.  `pk` varchar(36) DEFAULT NULL,
104.  `gmt_create` datetime DEFAULT NULL,
105.  `gmt_modified` datetime DEFAULT NULL,
106.  PRIMARY KEY (`row_key`)
107. ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
108. /*!40101 SET character_set_client = @saved_cs_client */;
109.
110. --
111. -- Dumping data for table `lock_table`
112. --
113.
114. LOCK TABLES `lock_table` WRITE;
115. /*!40000 ALTER TABLE `lock_table` DISABLE KEYS */;
116. /*!40000 ALTER TABLE `lock_table` ENABLE KEYS */;
117. UNLOCK TABLES;
118.
119. --
120. -- Table structure for table `undo_log`
121. --
122.
123. DROP TABLE IF EXISTS `undo_log`;
124. /*!40101 SET @saved_cs_client = @@character_set_client */;
125. /*!50503 SET character_set_client = utf8mb4 */;
126. CREATE TABLE `undo_log` (
127.   `id` bigint(20) NOT NULL AUTO_INCREMENT,
128.   `branch_id` bigint(20) NOT NULL,
129.   `xid` varchar(100) NOT NULL,
130.   `context` varchar(128) NOT NULL,
131.   `rollback_info` longblob NOT NULL,

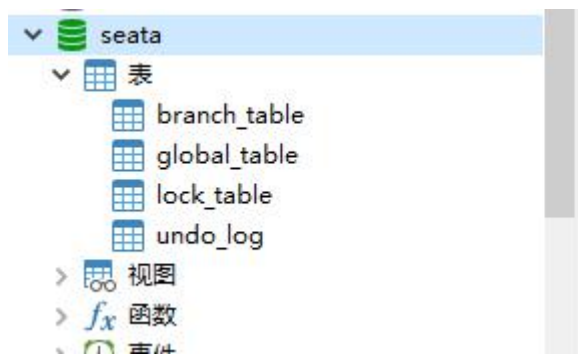
```

```

132. `log_status` int(11) NOT NULL,
133. `log_created` datetime NOT NULL,
134. `log_modified` datetime NOT NULL,
135. `ext` varchar(100) DEFAULT NULL,
136. PRIMARY KEY (`id`),
137. UNIQUE KEY `ux_undo_log` (`xid`,`branch_id`)
138. ) ENGINE=InnoDB DEFAULT CHARSET=utf8;
139. /*!40101 SET character_set_client = @saved_cs_client */;
140.
141. --
142. -- Dumping data for table `undo_log`
143. --
144.
145. LOCK TABLES `undo_log` WRITE;
146. /*!40000 ALTER TABLE `undo_log` DISABLE KEYS */;
147. /*!40000 ALTER TABLE `undo_log` ENABLE KEYS */;
148. UNLOCK TABLES;
149. /*!40103 SET TIME_ZONE=@OLD_TIME_ZONE */;
150.
151. /*!40101 SET SQL_MODE=@OLD_SQL_MODE */;
152. /*!40014 SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS */;
153. /*!40014 SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS */;
154. /*!40101 SET CHARACTER_SET_CLIENT=@OLD_CHARACTER_SET_CLIENT */;
155. /*!40101 SET CHARACTER_SET_RESULTS=@OLD_CHARACTER_SET_RESULTS */;
156. /*!40101 SET COLLATION_CONNECTION=@OLD_COLLATION_CONNECTION */;
157. /*!40111 SET SQL_NOTES=@OLD_SQL_NOTES */;
158.
159. -- Dump completed on 2020-05-12 14:43:04
160.

```

先创建 Seata 数据库，然后再修改 file.conf 配置文件。Seata 数据库是为了处理分布式事务需要存储的跟业务无关的数据。



然后打开 registry.conf 文件

```
-rw-r--r-- 1 root root 1327 Apr 21 14:10 README-zh.md
-rw-r--r-- 1 root root 1645 Jul 28 15:14 registry.conf
[root@teacher3 conf]# more registry.conf
```

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

注册中心

```
nacos {
  application = "seata-server"
  serverAddr = "localhost"
  namespace = ""
  cluster = "default"
  username = ""
  password = ""
}
```

```
eureka {
  serviceUrl = "http://172.16.86.47:8761/eureka"
  application = "seata-server-dev"
  weight = "1"
}
```

注册中心地址

```
redis {
  serverAddr = "localhost:6379"
  db = 0
  password = ""
  cluster = "default"
  timeout = 0
}
```

```
zk {
  cluster = "default"
  serverAddr = "127.0.0.1:2181"
  sessionTimeout = 6000
  connectTimeout = 2000
  username = ""
}
```

根据我们的具体情况，选择对应的配置中心，这里我们选用的是 eureka.

e.启动 Seata 服务

先进入到 bin 目录

```
[root@teacher3 seata]# cd bin
[root@teacher3 bin]# pwd
/apps/seata/seata/bin
[root@teacher3 bin]# ll
total 64
-rw----- 1 root root 49457 Jul 28 16:50 nohup.out
-rw-r--r-- 1 root root 3648 Apr 21 14:10 seata-server.bat
-rwxr--r-- 1 root root 4175 Apr 21 14:10 seata-server.sh
[root@teacher3 bin]#
```

执行启动命令

```
-rw-r--r-- 1 root root 49457 Jul 28 16:50 nohup.out
-rw-r--r-- 1 root root 3648 Apr 21 14:10 seata-server.bat
-rwxr--r-- 1 root root 4175 Apr 21 14:10 seata-server.sh
[root@teacher3 bin]# nohup ./seata-server.sh -h localhost -p 9092 -m db &
```

注：-h 是客户端连接的 ip -p 是端口 -m 是用数据存储用数据库

```
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.
2020-07-28 17:02:19.768 INFO [main]com.netflix.discovery.DiscoveryClient.getAndStoreFullRegistry:1056 -The response status is 200
2020-07-28 17:02:19.772 INFO [main]com.netflix.discovery.DiscoveryClient.initScheduledTasks:1270 -Starting heartbeat executor: renew interval is: 30
2020-07-28 17:02:19.775 INFO [main]com.netflix.discovery.InstanceInfoReplicator.<init>:60 -InstanceInfoReplicator onDemand update allowed rate per min is 4
2020-07-28 17:02:19.782 INFO [main]com.netflix.discovery.DiscoveryClient.<init>:449 -DiscoveryClient initialized at timestamp:156692693779 with initial instances count: 14
2020-07-28 17:02:19.782 INFO [main]com.netflix.discovery.DiscoveryClient.notify:1395 -Saw local status change event StatusChangeEvent [timestamp:156692693782, current-IP, previous-STARTING]
2020-07-28 17:02:19.783 INFO [DiscoveryClient-InstanceInfoReplicator-0]com.netflix.discovery.DiscoveryClient.register:826 -DiscoveryClient_SEATA-SERVER-DEV/172.16.181.31:seata-server-dev:9092
: registering service...
2020-07-28 17:02:19.814 INFO [DiscoveryClient-InstanceInfoReplicator-0]com.netflix.discovery.DiscoveryClient.register:835 -DiscoveryClient_SEATA-SERVER-DEV/172.16.181.31:seata-server-dev:9092
- registration status: 204
```


看到这里 说明 Seata 服务端启动成功。

5. Seata 客户端使用

a.pom 引入依赖

```
1.      <!--seata-->
2.      <dependency>
3.      <groupId>io.seata</groupId>
4.      <artifactId>seata-all</artifactId>
5.      <version>1.2.0</version>
6.      </dependency>
```

b.添加配置文件

客户端也需要配置 file.conf 和 registry.conf, 还有 seata.conf 配置文件
首先是 file.conf 配置

```
1.  transport {
2.  # tcp udt unix-domain-socket
3.  type = "TCP"
4.  #NIO NATIVE
5.  server = "NIO"
6.  #enable heartbeat
7.  heartbeat = true
8.  # the client batch send request enable
9.  enableClientBatchSendRequest = true
10. #thread factory for netty
11. threadFactory {
12.  bossThreadPrefix = "NettyBoss"
13.  workerThreadPrefix = "NettyServerNIOWorker"
14.  serverExecutorThread-prefix = "NettyServerBizHandler"
15.  shareBossWorker = false
16.  clientSelectorThreadPrefix = "NettyClientSelector"
17.  clientSelectorThreadSize = 1
18.  clientWorkerThreadPrefix = "NettyClientWorkerThread"
19.  # netty boss thread size,will not be used for UDT
20.  bossThreadSize = 1
21.  #auto default pin or 8
22.  workerThreadSize = "default"
23. }
24. shutdown {
25.  # when destroy server, wait seconds
26.  wait = 3
27. }
28. serialization = "seata"
29. compressor = "none"
30. }
31. service {
32. #这里是重点
33. # 设置事务组映射关系 这里不知道能否从注册中心获取
```



```
34. vgroupMapping.my_tx_group = "seata-server-dev"
35. seata-server-dev.grouplist = "113.31.104.154:9092"
36. #degrade, current not support
37. enableDegrade = false
38. #disable seata
39. disableGlobalTransaction = false
40. }
41.
42. client {
43.   rm {
44.     asyncCommitBufferLimit = 10000
45.     lock {
46.       retryInterval = 10
47.       retryTimes = 30
48.       retryPolicyBranchRollbackOnConflict = true
49.     }
50.     reportRetryCount = 5
51.     tableMetaCheckEnable = false
52.     reportSuccessEnable = false
53.   }
54.   tm {
55.     commitRetryCount = 5
56.     rollbackRetryCount = 5
57.   }
58.   undo {
59.     dataValidation = true
60.     logSerialization = "jackson"
61.     logTable = "undo_log"
62.   }
63.   log {
64.     exceptionRate = 100
65.   }
66. }
```

```

28   serialization = "seata"
29   compressor = "none"
30 }
31 service {
32   #这里是重点
33   # 设置事务组映射关系 这里不知道能否从注册中心获取
34   vgroupMapping.my_tx_group = "seata-server-dev"
35   seata-server-dev.grouplist = "113.31.104.154:9092"
36   #degrade, current not support
37   enableDegrade = false
38   #disable seata
39   disableGlobalTransaction = false
40 }
41
42 client {
43   rm {
44     asyncCommitBufferLimit = 10000
45     lock {
46       retryInterval = 10
47       retryTimes = 30
48       retryPolicyBranchRollbackOnConflict = true

```

修改 vgroupMapping.my_tx_group 这个值是我们服务端配置的名字
seata-server-dev.grouplist 是启动 Seata 服务时候指定的 ip+端口

再来修改 registry.conf

```

1. registry {
2.   # file 、nacos 、eureka、redis、zk、consul、etcd3、sofa
3.   type = "eureka"
4.
5.   nacos {
6.     serverAddr = "localhost"
7.     namespace = "public"
8.     cluster = "default"
9.   }
10.  eureka {
11.    serviceUrl = "http://113.31.104.154:8761/eureka/"
12.    application = "seata-server-dev"
13.    weight = "1"
14.  }
15.  redis {
16.    serverAddr = "localhost:6379"
17.    db = "0"
18.  }
19.  zk {
20.    cluster = "default"
21.    serverAddr = "127.0.0.1:2181"
22.    session.timeout = 6000
23.    connect.timeout = 2000
24.  }
25.  consul {

```

```
26.   cluster = "seata-server"
27.   serverAddr = "127.0.0.1:8500"
28. }
29. etcd3 {
30.   cluster = "default"
31.   serverAddr = "http://localhost:2379"
32. }
33. sofa {
34.   serverAddr = "127.0.0.1:9603"
35.   application = "default"
36.   region = "DEFAULT_ZONE"
37.   datacenter = "DefaultDataCenter"
38.   cluster = "default"
39.   group = "SEATA_GROUP"
40.   addressWaitTime = "3000"
41. }
42. file {
43.   name = "file.conf"
44. }
45. }
46.
47. # 配置信息来源
48. config {
49.   # file、nacos、apollo、zk、consul、etcd3
50.   type = "file"
51.
52.   nacos {
53.     serverAddr = "localhost"
54.     namespace = "public"
55.     cluster = "default"
56.   }
57.   consul {
58.     serverAddr = "http://127.0.0.1:8500"
59.   }
60.   apollo {
61.     app.id = "seata-server"
62.     apollo.meta = "http://192.168.1.204:8801"
63.   }
64.   zk {
65.     serverAddr = "127.0.0.1:2181"
66.     session.timeout = 6000
67.     connect.timeout = 2000
68.   }
69.   etcd3 {
70.     serverAddr = "http://localhost:2379"
71.   }
72.   file {
73.     name = "file.conf"
74.   }
75. }
```

```

1 registry {
2   # file , nacos , eureka, redis, zk, consul, etcd3, sofa
3   type = "eureka"
4
5   nacos {
6     serverAddr = "localhost"
7     namespace = "public"
8     cluster = "default"
9   }
10  eureka {
11    serviceUrl = "http://113.31.104.154:8761/eureka/"
12    application = "seata-server-dev"
13    weight = "1"
14  }
15  redis {
16    serverAddr = "localhost:6379"
17    db = "0"
18  }
19  zk {
20    cluster = "default"
21    serverAddr = "127.0.0.1:2181"
22    session timeout = 60000

```

这个配置跟之前服务端的配置差不多。也是修改这两个地方。

再来添加 seata.conf 配置

```

1. ## -----
2. ## Licensed to the Apache Software Foundation (ASF) under one or more
3. ## contributor license agreements. See the NOTICE file distributed with
4. ## this work for additional information regarding copyright ownership.
5. ## The ASF licenses this file to You under the Apache License, Version 2.0
6. ## (the "License"); you may not use this file except in compliance with
7. ## the License. You may obtain a copy of the License at
8. ##
9. ## http://www.apache.org/licenses/LICENSE-2.0
10. ##
11. ## Unless required by applicable law or agreed to in writing, software
12. ## distributed under the License is distributed on an "AS IS" BASIS,
13. ## WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
14. ## See the License for the specific language governing permissions and
15. ## limitations under the License.
16. ## -----
17.
18. client {
19.   application.id = edu-pay-server
20.   transaction.service.group = my_tx_group
21. }
22.

```

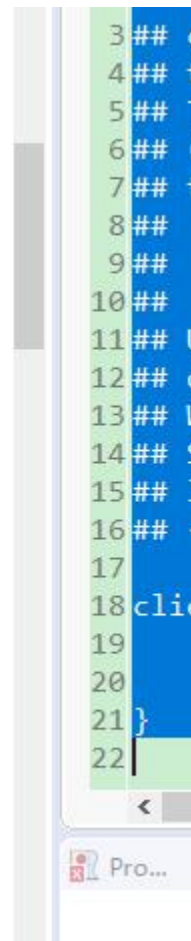
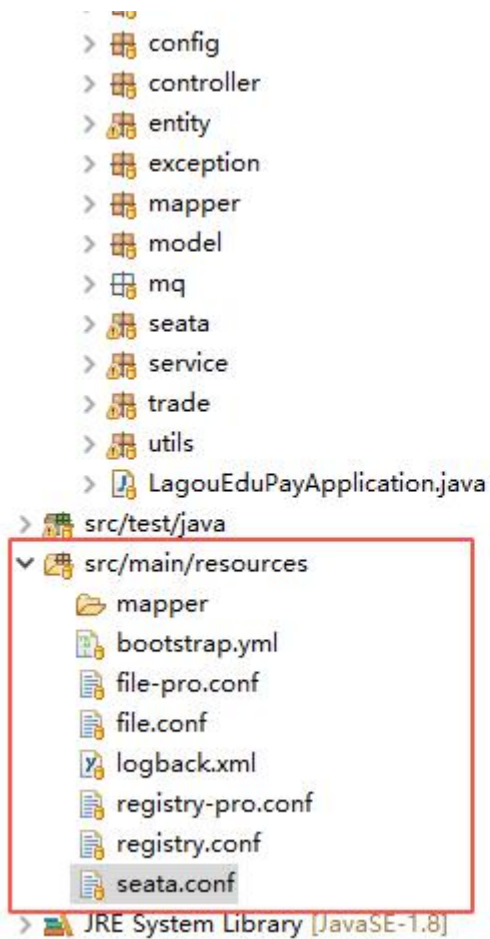
```

17
18 client {
19     application.id = edu-pay-server
20     transaction.service.group = my_tx_group
21 }
22 |

```

application.id 是客户端项目的名字
下面那个是自定义的一个事务分组

c.配置文件放置路径



把以上配置文件放到项目的 resources 目录下。

d.修改启动类

启动的时候，排除DataSource的自动注入
原因：需要对DataSource进行自动代理

```
17
18 /**
19  * @Description:(支付服务)
20  * @author: ma wei long
21  * @date: 2020年6月17日 上午10:47:47
22  */
23 @SpringBootApplication(exclude = {
24     DataSourceAutoConfiguration.class
25 })
26 @EnableDiscoveryClient
27 @MapperScan("com.lagou.edu.pay.mapper")
28 @ComponentScan("com.lagou.edu")
29 @EnableFeignClients("com.lagou.edu")
30 @EnableRetry
31 @Slf4j
32 @EnableAutoDataSourceProxy
33 public class LagouEduPayApplication implements DisposableBean {
34
35     private static ConfigurableApplicationContext ctx;
36
37     public static void main(String[] args) {
```

按照标红的地方修改

e.添加 Seata 相关类

```
1. package com.lagou.edu.pay.seata;
2.
3. import javax.sql.DataSource;
4.
5. import org.springframework.beans.factory.annotation.Value;
6. import org.springframework.boot.context.properties.ConfigurationProperties;
7. import org.springframework.context.annotation.Bean;
8. import org.springframework.context.annotation.Configuration;
9. import org.springframework.context.annotation.Primary;
10.
11. import com.alibaba.druid.pool.DruidDataSource;
12.
13. import io.seata.rm.datasource.DataSourceProxy;
14. import io.seata.spring.annotation.GlobalTransactionScanner;
15.
16. /**
17.  * @Description:(数据源代理)
18.  * @author: ma wei long
19.  * @date: 2020年7月28日 上午12:24:01
20.  */
21. @Configuration
22. public class DataSourceConfiguration {
23.
24.     @Value("${spring.cloud.alibaba.seata.tx-service-group}")
25.     private String group;
26.     @Value("${spring.application.name}")
27.     private String appName;
28.
```

```

        // 创建FescarXidFilter拦截器对象, 将事务ID注册到上下文对象中
29.     @Bean
30.     public FescarXidFilter fescarXidFilter(){
31.         return new FescarXidFilter();
32.     }
33.     // 全局的扫描
34.     @Bean
35.     public GlobalTransactionScanner globalTransactionScanner(){
36.         GlobalTransactionScanner scanner = new GlobalTransactionScanner(appName,group);
37.         return scanner;
38.     }
39.     // 数据源的代理
40.     @Bean
41.     @ConfigurationProperties(prefix = "spring.datasource")
42.     public DataSource druidDataSource(){
43.         DruidDataSource druidDataSource = new DruidDataSource();
44.         return druidDataSource;
45.     }
46.
47.     @Primary
48.     @Bean("dataSource")
49.     public DataSourceProxy dataSource(DataSource druidDataSource){
50.         return new DataSourceProxy(druidDataSource);
51.     }
52.
53. // @Bean
54. // public SqlSessionFactory sqlSessionFactory(DataSourceProxy dataSourceProxy)throws
Exception{
55. //     SqlSessionFactoryBean sqlSessionFactoryBean = new SqlSessionFactoryBean();
56. //     sqlSessionFactoryBean.setDataSource(dataSourceProxy);
57. //     sqlSessionFactoryBean.setMapperLocations(new
PathMatchingResourcePatternResolver()
58. //         .getResources("classpath*:/mapper/*.xml"));
59. //     sqlSessionFactoryBean.setTransactionFactory(new
SpringManagedTransactionFactory());
60. //     return sqlSessionFactoryBean.getObject();
61. // }
62.
63. }
64.
65.

```

```

1. package com.lagou.edu.pay.seata;
2. import java.io.IOException;
3.
4. import javax.servlet.FilterChain;
5. import javax.servlet.ServletException;
6. import javax.servlet.http.HttpServletRequest;
7. import javax.servlet.http.HttpServletResponse;
8.

```



```

9. import org.apache.commons.lang.StringUtils;
10. import org.springframework.web.filter.OncePerRequestFilter;
11.
12. import io.seata.core.context.RootContext;
13. import lombok.extern.slf4j.Slf4j;
14.
15. /**
16.  * @author: ma wei long
17.  * @date: 2020 年 7 月 28 日 上午 12:27:34
18.  */
19. @Slf4j
20. public class FescarXidFilter extends OncePerRequestFilter {
21.
22.     @Override
23.     protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response,
FilterChain filterChain) throws ServletException, IOException {
24.         String xid = RootContext.getXID();
25.         String restXid = request.getHeader("Fescar-Xid");
26.         boolean bind = false;
27.         if(StringUtils.isBlank(xid)&&StringUtils.isNotBlank(restXid)){
28.             RootContext.bind(restXid);
29.             bind = true;
30.             if (logger.isDebugEnabled()) {
31.                 logger.debug("bind[" + restXid + "] to RootContext");
32.             }
33.         }
34.         try{
35.             filterChain.doFilter(request, response);
36.         } finally {
37.             if (bind) {
38.                 String unbindXid = RootContext.unbind();
39.                 if (logger.isDebugEnabled()) {
40.                     logger.debug("unbind[" + unbindXid + "] from RootContext");
41.                 }
42.                 if (!restXid.equalsIgnoreCase(unbindXid)) {
43.                     logger.warn("xid in change during http rest from " + restXid + " to " +
unbindXid);
44.                     if (unbindXid != null) {
45.                         RootContext.bind(unbindXid);
46.                         logger.warn("bind [" + unbindXid + "] back to RootContext");
47.                     }
48.                 }
49.             }
50.         }
51.     }
52. }

```

```

1. package com.lagou.edu.pay.seata;
2.
3. import org.apache.commons.lang3.StringUtils;
4. import org.springframework.stereotype.Component;
5.
6. import feign.RequestInterceptor;
7. import feign.RequestTemplate;
8. import io.seata.core.context.RootContext;
9. /**
10.  * @author: ma wei long
11.  * @date: 2020 年 7 月 28 日 上午 12:28:08
12.  */
13. @Component
14. public class RequestHeaderInterceptor implements RequestInterceptor {
15.
16.     @Override
17.     public void apply(RequestTemplate template) {
18.         String xid = RootContext.getXID();
19.         if(StringUtils.isNotBlank(xid)){
20.             template.header("Fescar-Xid",xid);
21.         }
22.     }
23. }

```

事务ID (Fescar-Xid)
事务的控制有这个事务的唯一ID控制的

f.application.yml 配置文件修改

```

1 spring:
2   application:
3     name: edu-pay-boot
4   cloud:
5     alibaba:
6       seata:
7         tx-service-group: my_tx_group
8   config:
9     discovery:
10      enabled: true
11      service-id: edu-config-boot
12      name: ${spring.application.name}
13      profile: @profile@
14   main:
15     allow-bean-definition-overriding: true
16
17 ribbon:
18   ###指的是建立连接所用的时间，适用于网络状况正常的情况下，两端连接所用的时间。
19   ReadTimeout: 10000
20   ###指的是建立连接后从服务器读取到可用资源所用的时间。
21   ConnectTimeout: 10000
22   #注册到Eureka服务中心
23   eureka:
24     client:
25       service-url:
26         # 注册到集群，就把多个EurekaServer地址使用逗号连接起来即可；注册到单实例（非集群模式），那就写一个就ok
27         defaultZone: @eureka-host@

```

g.分布式事务注解使用

```
/**
 * @author: ma wei long
 * @date: 2020年7月28日 上午11:11:07
 */
@Override
@Transactional(name="cancelPayOrder_tx",rollbackFor = Exception.class)
public void cancelPayOrder(CancelPayOrderDTO cancelPayOrderDTO) {
    PayOrder payOrderDB = this.getId(cancelPayOrderDTO.getOrderID());
    ValidateUtils.notNull(payOrderDB, ResultCode.ALERT_ERROR.getState(), "查询支付订单信息为空:orderId:" + cancelPayOrderDTO.getOrderID());

    if(!payOrderDB.getStatus().equals(Status.NOT_PAY.getCode())){
        //支付订单已经终态 直接返回
        Log.warn("支付订单已经终态 payOrderDB:{", JSON.toJSONString(payOrderDB));
        return;
    }
    ValidateUtils.isTrue(payOrderService.updateStatusInvalid(payOrderDB, "支付订单更新为失效头账");
    ResponseDTO<?> resp = userCourseOrderRemoteService.updateOrderStatus(payOrderDB.getGoodsOrderNo(), UserCourseOrderStatus.CANCEL);
    ValidateUtils.isTrue(resp.isSuccess(), resp.getState(),resp.getMessage());
    //如果是活动商品还原库存
    ResponseDTO<?> respStock = activityCourseRemoteService.updateActivityCourseStock(payOrderDB.getProductID(),payOrderDB.getGoodsOrderNo());
    ValidateUtils.isTrue(respStock.isSuccess(), respStock.getState(),respStock.getMessage());
}

/**
 * @author: ma wei long
 * @date: 2020年7月28日 上午11:11:07
 */
```

@GlobalTransactional 这个注解是**控制全局事务**的，在相应的业务方法上添加这个注解就可以了。

h.启动客户端服务

```
2020-07-28 17:45:29.493 [restartedMain] INFO o.s.c.c.c.Client.ConfigServicePropertySourceLocator:149 - Located environment: name=edu-pay-boot,
2020-07-28 17:45:29.494 [restartedMain] INFO o.s.c.b.c.PropertySourceBootstrapConfiguration:98 - Located property source: CompositePropertySource{
2020-07-28 17:45:29.533 [restartedMain] INFO com.lagou.edu.pay.LagouEduPayApplication:646 - No active profile set, falling back to default p
2020-07-28 17:45:32.973 [restartedMain] WARN o.s.springframework.boot.actuate.endpoint.EndpointId:131 - Endpoint ID 'bus-env' contains invalid
2020-07-28 17:45:32.983 [restartedMain] WARN o.s.springframework.boot.actuate.endpoint.EndpointId:131 - Endpoint ID 'bus-refresh' contains invalid
2020-07-28 17:45:33.516 [restartedMain] INFO o.s.d.r.config.RepositoryConfigurationDelegate:244 - Multiple Spring Data modules found, entering
2020-07-28 17:45:33.521 [restartedMain] INFO o.s.d.r.config.RepositoryConfigurationDelegate:126 - Bootstrapping Spring Data repositories in
2020-07-28 17:45:33.575 [restartedMain] INFO o.s.d.r.config.RepositoryConfigurationDelegate:182 - Finished Spring Data repository scanning in
2020-07-28 17:45:33.968 [restartedMain] WARN o.s.springframework.boot.actuate.endpoint.EndpointId:131 - Endpoint ID 'service-registry' contains
2020-07-28 17:45:35.169 [restartedMain] INFO o.s.springframework.cloud.context.scope.GenericScope:294 - BeanFactory id=3a50dfcb-b330-3309-af9f
2020-07-28 17:45:35.200 [restartedMain] INFO o.s.i.c.DefaultConfiguringBeanFactoryPostProcessor:193 - No bean named 'errorChannel' has been
2020-07-28 17:45:35.218 [restartedMain] INFO o.s.i.c.DefaultConfiguringBeanFactoryPostProcessor:280 - No bean named 'taskScheduler' has been
2020-07-28 17:45:35.248 [restartedMain] INFO o.s.i.c.DefaultConfiguringBeanFactoryPostProcessor:431 - No bean named 'integrationHeaderChannel
2020-07-28 17:45:35.291 [restartedMain] INFO o.s.c.s.PostProcessorRegistrationDelegate$BeanPostProcessorChecker:330 - Bean 'dataSourceConfig
2020-07-28 17:45:35.466 [restartedMain] INFO io.seata.config.FileConfiguration:121 - The configuration file used is registry.conf
2020-07-28 17:45:35.499 [restartedMain] INFO io.seata.config.FileConfiguration:121 - The configuration file used is file.conf
2020-07-28 17:45:35.514 [restartedMain] INFO i.seata.spring.annotation.GlobalTransactionScanner:162 - Initializing Global Transaction Client
2020-07-28 17:45:35.866 [restartedMain] INFO io.seata.core.rpc.netty.AbstractRpcRemotingClient:147 - RpcClientBootstrap has started
2020-07-28 17:45:35.873 [restartedMain] INFO i.seata.spring.annotation.GlobalTransactionScanner:170 - Transaction Manager Client is initiali
2020-07-28 17:45:35.944 [restartedMain] INFO io.seata.rm.datasource.AsyncWorker:125 - Async Commit Buffer Limit: 10000
2020-07-28 17:45:35.945 [restartedMain] INFO io.seata.rm.datasource.xa.ResourceManagerXA:40 - ResourceManagerXA init ...
2020-07-28 17:45:35.975 [restartedMain] INFO io.seata.core.rpc.netty.AbstractRpcRemotingClient:147 - RpcClientBootstrap has started
2020-07-28 17:45:35.975 [restartedMain] INFO i.seata.spring.annotation.GlobalTransactionScanner:175 - Resource Manager is initialized, appli
2020-07-28 17:45:35.976 [restartedMain] INFO i.seata.spring.annotation.GlobalTransactionScanner:179 - Global Transaction Clients are initial
2020-07-28 17:45:36.191 [restartedMain] INFO o.s.c.s.PostProcessorRegistrationDelegate$BeanPostProcessorChecker:330 - Bean 'org.springframework
2020-07-28 17:45:36.437 [restartedMain] INFO o.s.c.s.PostProcessorRegistrationDelegate$BeanPostProcessorChecker:330 - Bean 'org.springframework
2020-07-28 17:45:36.831 [restartedMain] INFO o.s.c.s.PostProcessorRegistrationDelegate$BeanPostProcessorChecker:330 - Bean 'integrationDispo
2020-07-28 17:45:36.966 [restartedMain] INFO o.s.c.s.PostProcessorRegistrationDelegate$BeanPostProcessorChecker:330 - Bean 'org.springframework
```

可以看到客户端的服务已经读取了 配置文件

```
om.netflix.discovery.DiscoveryClient:1056 - The response status is 200
om.netflix.discovery.DiscoveryClient:1317 - Not registering with Eureka server per configuration
om.netflix.discovery.DiscoveryClient:449 - Discovery Client initialized at timestamp 1595929546027 with initial instances count: 15
o.seata.core.rpc.netty.NettyClientChannelManager:99 - will connect to 113.31.104.154:9092
o.seata.core.rpc.netty.RmRpcClient:149 - RM will register jdbc:mysql://113.31.119.154:3306/edu_pay
o.seata.core.rpc.netty.NettyPoolableFactory:56 - NettyPool create channel to transactionRole:RMROLE,address:113.31.104.154:9092,msg:< Register
o.seata.core.rpc.netty.RmRpcClient:167 - register RM success. server version:1.2.0,channel:[id: 0xa9349a1c, L:/10.71.9.61:52618 - R:/113.31.1
o.seata.core.rpc.netty.NettyPoolableFactory:81 - register success, cost 170 ms, version:1.2.0,role:RMROLE,channel:[id: 0xa9349a1c, L:/10.71.9
o.seata.core.rpc.netty.RmRpcClient:206 - will register resourceId:jdbc:mysql://113.31.119.154:3306/edu_pay
s.s.a.datasource.SeataAutoDataSourceProxyCreator:45 - Auto proxy of [dataSource]
s.scheduling.concurrent.ThreadPoolTaskScheduler:171 - Initializing ExecutorService 'taskScheduler'

s.b.d.autoconfigure.OptionalLiveReloadServer:57 - LiveReload server is running on port 35729
```

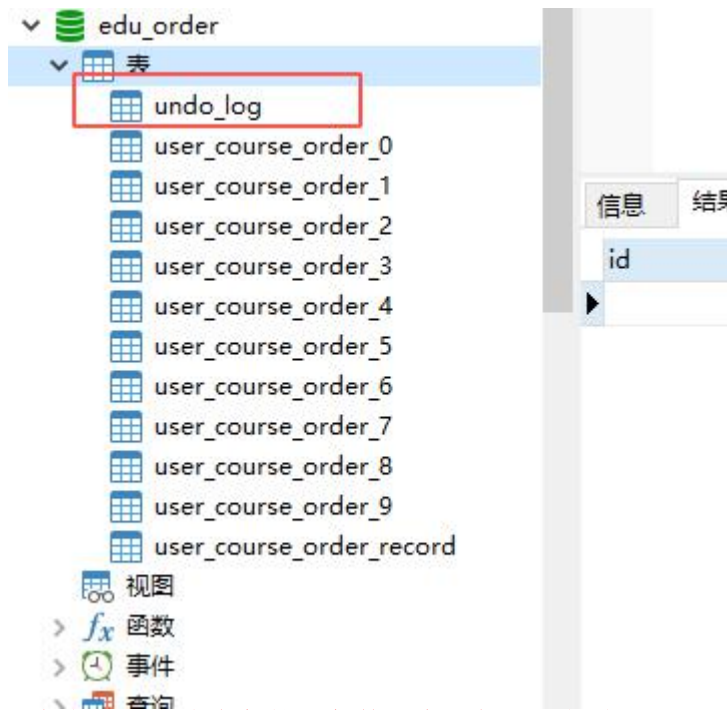
可以看到 Seata 客户端已经成功连接到 Seata 服务端

-RM register success

```
2020-07-28 17:42:35.442 INFO [NettyServerNIOWorker-1] io.seata.core.rpc.netty.AbstractRpcRemotingServer.handleDisconnect:259 -remove channel:[id: 0x91bf3104, L:/172.16.181.31:9092 ! R:/124.193.79.2:52391]context:RpcContext(applicationId='edu-order-server', transactionServiceGroup='my_tx_group', clientId='edu-order-server:124.193.79.2:52391', channel=[id: 0x91bf3104, L:/172.16.181.31:9092 ! R:/124.193.79.2:52391], resourceSets=null)
2020-07-28 17:42:35.442 INFO [NettyServerNIOWorker-1] io.seata.core.rpc.netty.AbstractRpcRemotingServer.handleDisconnect:259 -remove channel:[id: 0xa8290201, L:/172.16.181.31:9092 ! R:/124.193.79.2:52392]context:RpcContext(applicationId='edu-order-server', transactionServiceGroup='my_tx_group', clientId='edu-order-server:124.193.79.2:52392', channel=[id: 0xa8290201, L:/172.16.181.31:9092 ! R:/124.193.79.2:52392], resourceSets=null)
2020-07-28 17:45:52.121 INFO [ServerHandlerThread-1] io.seata.core.rpc.DefaultServerMessageListenerImpl.onRegRmMessage:127 -RM register success,message:RegisterRmRequest(resourceIds='jdbc:mysql://113.31.119.154:3306/edu_pay', applicationId='edu-pay-boot', transactionServiceGroup='my_tx_group'),channel:[id: 0x7fe4839a, L:/172.16.181.31:9092 ! R:/124.193.79.2:52618]
2020-07-28 17:45:52.210 INFO [ServerHandlerThread-1] io.seata.core.rpc.DefaultServerMessageListenerImpl.onRegRmMessage:127 -RM register success,message:RegisterRmRequest(resourceIds='jdbc:mysql://113.31.119.154:3306/edu_pay', applicationId='edu-pay-boot', transactionServiceGroup='my_tx_group'),channel:[id: 0x7fe4839a, L:/172.16.181.31:9092 ! R:/124.193.79.2:52618]
2020-07-28 17:46:05.301 INFO [ServerHandlerThread-1] io.seata.core.rpc.DefaultServerMessageListenerImpl.onRegRmMessage:127 -RM register success,message:RegisterRmRequest(resourceIds='jdbc:mysql://113.31.119.154:3306/edu_pay', applicationId='edu-pay-boot', transactionServiceGroup='my_tx_group'),channel:[id: 0x7fe4839a, L:/172.16.181.31:9092 ! R:/124.193.79.2:52618]
2020-07-28 17:46:40.280 INFO [NettyServerNIOWorker-1] io.seata.core.rpc.DefaultServerMessageListenerImpl.onRegTmMessage:153 -TM register success,message:RegisterTmRequest(applicationId='edu-pay-boot', transactionServiceGroup='my_tx_group'),channel:[id: 0x2537e83a, L:/172.16.181.31:9092 ! R:/124.193.79.2:52692]
2020-07-28 17:47:09.703 INFO [AsyncResolver-bootstrap-executor-0] c.n.discovery.shared.resolver.aws.ConfigClusterResolver.getClusterEndpoints:43 -Resolving eureka endpoints via configuration
```

Seata 服务端可以看到 客户端的信息。说明客户端&服务端 连接成功。

i.客户端数据库创建 undo_log 表



所有参与分布式事务的业务数据库，都需要创建一张 **undo_log** 表，分布式事务中间件事务回滚的时候需要用。

同理其他客户端也按照这个步骤配置即可