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.

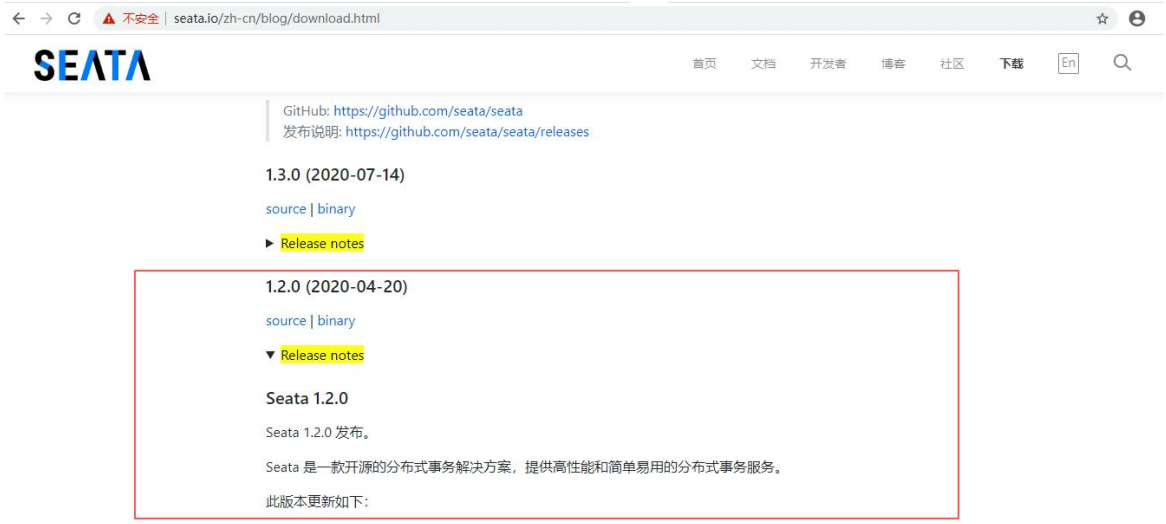
3. Seata 使用步骤

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

```
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
```

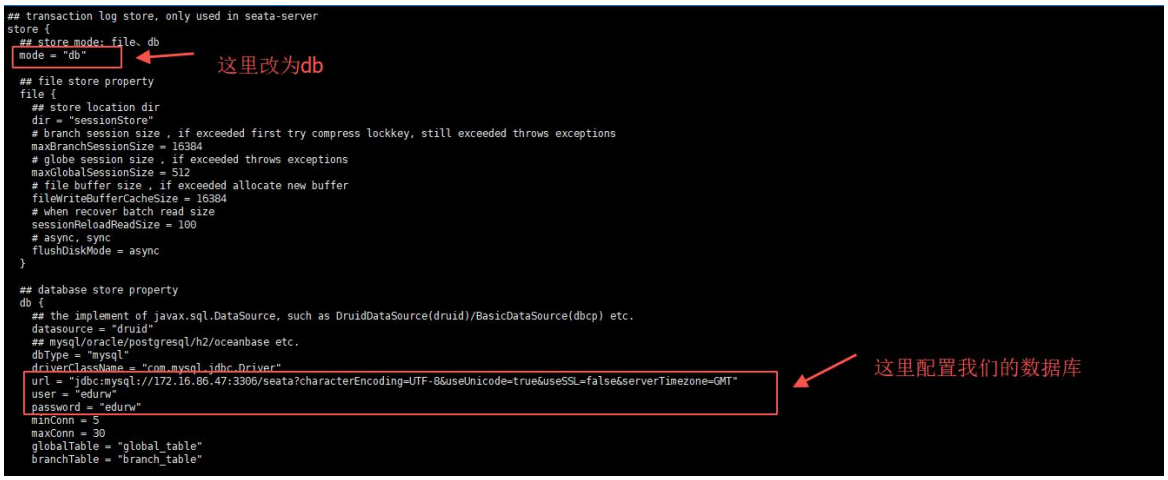
d.修改配置

进入 conf 目录

```
[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



需要先创建 Seata 数据库， 一下是 sql 脚本

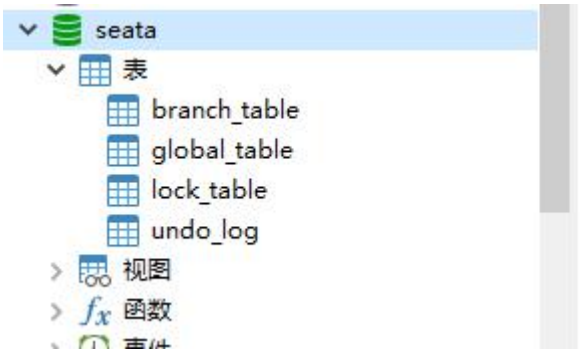
```
1. CREATE DATABASE  IF NOT EXISTS `seata` /*!40100 DEFAULT CHARACTER SET utf8 COLLATE utf8_unicode_ci */;
2. USE `seata`;
3. -- MySQL dump 10.13  Distrib 8.0.18, for macos10.14 (x86_64)
4. --
5. -- Host: 127.0.0.1    Database: seata
6. -- ------------------------------------------------------
7.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

```
  `xid` varchar(128) NOT NULL,
  `transaction_id` bigint(20) DEFAULT NULL,
  `resource_group_id` varchar(32) DEFAULT NULL,
  `resource_id` varchar(256) DEFAULT NULL,
  `lock_key` varchar(128) DEFAULT NULL,
  `branch_type` varchar(8) DEFAULT NULL,
  `status` tinyint(4) DEFAULT NULL,
  `client_id` varchar(64) DEFAULT NULL,
  `application_data` varchar(2000) DEFAULT NULL,
  `gmt_create` datetime DEFAULT NULL,
  `gmt_modified` datetime DEFAULT NULL,
  PRIMARY KEY (`branch_id`),
  KEY `idx_xid` (`xid`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
/*!40101 SET character_set_client = @saved_cs_client */;

--
-- Dumping data for table `branch_table`
--

LOCK TABLES `branch_table` WRITE;
/*!40000 ALTER TABLE `branch_table` DISABLE KEYS */;
/*!40000 ALTER TABLE `branch_table` ENABLE KEYS */;
UNLOCK TABLES;

--
-- Table structure for table `global_table`
--

DROP TABLE IF EXISTS `global_table`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
/*!50503 SET character_set_client = utf8mb4 */;
CREATE TABLE `global_table` (
  `xid` varchar(128) NOT NULL,
  `transaction_id` bigint(20) DEFAULT NULL,
  `status` tinyint(4) NOT NULL,
  `application_id` varchar(32) DEFAULT NULL,
  `transaction_service_group` varchar(32) DEFAULT NULL,
  `transaction_name` varchar(128) DEFAULT NULL,
  `timeout` int(11) DEFAULT NULL,
  `begin_time` bigint(20) DEFAULT NULL,
  `application_data` varchar(2000) DEFAULT NULL,
  `gmt_create` datetime DEFAULT NULL,
  `gmt_modified` datetime DEFAULT NULL,
  PRIMARY KEY (`xid`),
  KEY `idx_gmt_modified_status` (`gmt_modified`,`status`),
  KEY `idx_transaction_id` (`transaction_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
/*!40101 SET character_set_client = @saved_cs_client */;

--
```

```
-- Dumping data for table `global_table`
--

LOCK TABLES `global_table` WRITE;
/*!40000 ALTER TABLE `global_table` DISABLE KEYS */;
/*!40000 ALTER TABLE `global_table` ENABLE KEYS */;
UNLOCK TABLES;

--
-- Table structure for table `lock_table`
--

DROP TABLE IF EXISTS `lock_table`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
/*!50503 SET character_set_client = utf8mb4 */;
CREATE TABLE `lock_table` (
  `row_key` varchar(128) NOT NULL,
  `xid` varchar(96) DEFAULT NULL,
  `transaction_id` mediumtext,
  `branch_id` mediumtext,
  `resource_id` varchar(256) DEFAULT NULL,
  `table_name` varchar(32) DEFAULT NULL,
  `pk` varchar(36) DEFAULT NULL,
  `gmt_create` datetime DEFAULT NULL,
  `gmt_modified` datetime DEFAULT NULL,
  PRIMARY KEY (`row_key`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
/*!40101 SET character_set_client = @saved_cs_client */;

--
-- Dumping data for table `lock_table`
--

LOCK TABLES `lock_table` WRITE;
/*!40000 ALTER TABLE `lock_table` DISABLE KEYS */;
/*!40000 ALTER TABLE `lock_table` ENABLE KEYS */;
UNLOCK TABLES;

--
-- Table structure for table `undo_log`
--

DROP TABLE IF EXISTS `undo_log`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
/*!50503 SET character_set_client = utf8mb4 */;
CREATE TABLE `undo_log` (
  `id` bigint(20) NOT NULL AUTO_INCREMENT,
  `branch_id` bigint(20) NOT NULL,
  `xid` varchar(100) NOT NULL,
  `context` varchar(128) NOT NULL,
  `rollback_info` longblob NOT NULL,
```

```
  `log_status` int(11) NOT NULL,
  `log_created` datetime NOT NULL,
  `log_modified` datetime NOT NULL,
  `ext` varchar(100) DEFAULT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `ux_undo_log` (`xid`,`branch_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
/*!40101 SET character_set_client = @saved_cs_client */;

--
-- Dumping data for table `undo_log`
--

LOCK TABLES `undo_log` WRITE;
/*!40000 ALTER TABLE `undo_log` DISABLE KEYS */;
/*!40000 ALTER TABLE `undo_log` ENABLE KEYS */;
UNLOCK TABLES;
/*!40103 SET TIME_ZONE=@OLD_TIME_ZONE */;

/*!40101 SET SQL_MODE=@OLD_SQL_MODE */;
/*!40014 SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS */;
/*!40014 SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS */;
/*!40101 SET CHARACTER_SET_CLIENT=@OLD_CHARACTER_SET_CLIENT */;
/*!40101 SET CHARACTER_SET_RESULTS=@OLD_CHARACTER_SET_RESULTS */;
/*!40101 SET COLLATION_CONNECTION=@OLD_COLLATION_CONNECTION */;
/*!40111 SET SQL_NOTES=@OLD_SQL_NOTES */;

-- Dump completed on 2020-05-12 14:43:04

```

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。



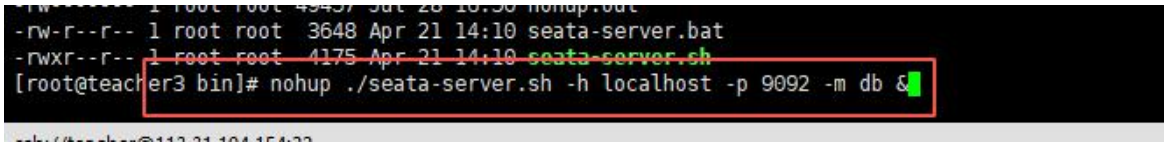
然后打开 registry.conf 文件

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

执行启动命令



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

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

```
<!--seata-->
  <dependency>
 <groupId>io.seata</groupId>
 <artifactId>seata-all</artifactId>
 <version>1.2.0</version>
</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

```
  transport {
# tcp udt unix-domain-socket
type = "TCP"
#NIO NATIVE
server = "NIO"
#enable heartbeat
heartbeat = true
# the client batch send request enable
enableClientBatchSendRequest = true
 #thread factory for netty
 threadFactory {
  bossThreadPrefix = "NettyBoss"
  workerThreadPrefix = "NettyServerNIOWorker"
  serverExecutorThread-prefix = "NettyServerBizHandler"
  shareBossWorker = false
  clientSelectorThreadPrefix = "NettyClientSelector"
  clientSelectorThreadSize = 1
  clientWorkerThreadPrefix = "NettyClientWorkerThread"
  # netty boss thread size,will not be used for UDT
  bossThreadSize = 1
  #auto default pin or 8
  workerThreadSize = "default"
 }
 shutdown {
  # when destroy server, wait seconds
  wait = 3
 }
 serialization = "seata"
 compressor = "none"
}
service {
 #这里是重点
 # 设置事务组映射关系 这里不知道能否从注册中心获取
```

```
  vgroupMapping.my_tx_group = "seata-server-dev"
  seata-server-dev.grouplist = "113.31.104.154:9092"
  #degrade, current not support
  enableDegrade = false
  #disable seata
  disableGlobalTransaction = false
}

client {
  rm {
    asyncCommitBufferLimit = 10000
    lock {
      retryInterval = 10
      retryTimes = 30
      retryPolicyBranchRollbackOnConflict = true
    }
    reportRetryCount = 5
    tableMetaCheckEnable = false
    reportSuccessEnable = false
  }
  tm {
    commitRetryCount = 5
    rollbackRetryCount = 5
  }
  undo {
    dataValidation = true
    logSerialization = "jackson"
    logTable = "undo_log"
  }
  log {
    exceptionRate = 100
  }
}
```

```
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

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

  nacos {
    serverAddr = "localhost"
    namespace = "public"
    cluster = "default"
  }
  eureka {
    serviceUrl = "http://113.31.104.154:8761/eureka/"
    application = "seata-server-dev"
    weight = "1"
  }
  redis {
    serverAddr = "localhost:6379"
    db = "0"
  }
  zk {
    cluster = "default"
    serverAddr = "127.0.0.1:2181"
    session.timeout = 6000
    connect.timeout = 2000
  }
  consul {
```

```
    cluster = "seata-server"
    serverAddr = "127.0.0.1:8500"
  }
  etcd3 {
    cluster = "default"
    serverAddr = "http://localhost:2379"
  }
  sofa {
    serverAddr = "127.0.0.1:9603"
    application = "default"
    region = "DEFAULT_ZONE"
    datacenter = "DefaultDataCenter"
    cluster = "default"
    group = "SEATA_GROUP"
    addressWaitTime = "3000"
  }
  file {
    name = "file.conf"
  }
}

# 配置信息来源
config {
  # file、nacos 、apollo、zk、consul、etcd3
  type = "file"

  nacos {
    serverAddr = "localhost"
    namespace = "public"
    cluster = "default"
  }
  consul {
    serverAddr = "http://127.0.0.1:8500"
  }
  apollo {
    app.id = "seata-server"
    apollo.meta = "http://192.168.1.204:8801"
  }
  zk {
    serverAddr = "127.0.0.1:2181"
    session.timeout = 6000
    connect.timeout = 2000
  }
  etcd3 {
    serverAddr = "http://localhost:2379"
  }
  file {
    name = "file.conf"
  }
}
```

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

  nacos {
    serverAddr = "localhost"
    namespace = "public"
    cluster = "default"
  }
  eureka {
    serviceUrl = "http://113.31.104.154:8761/eureka/"
    application = "seata-server-dev"
    weight = "1"
  }
  redis {
    serverAddr = "localhost:6379"
    db = "0"
  }
  zk {
    cluster = "default"
    serverAddr = "127.0.0.1:2181"
    session.timeout = 6000
```

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。

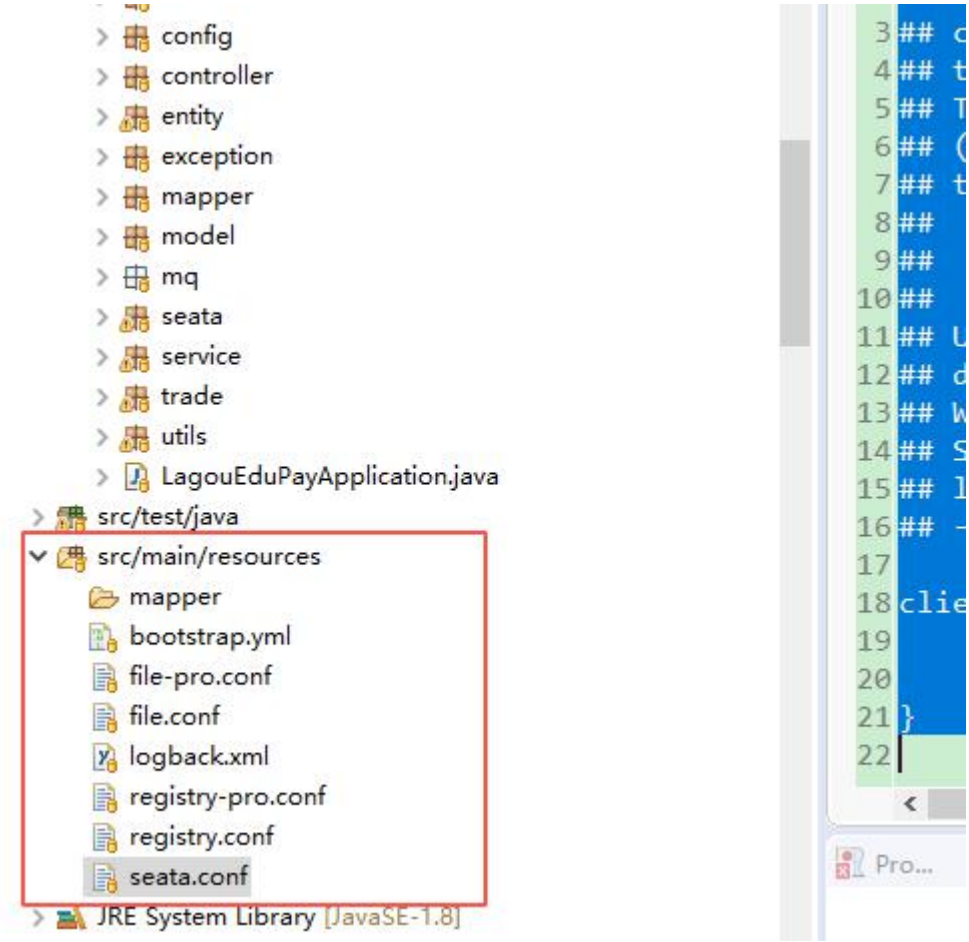
再来添加 seata.conf 配置

```
## ---------------------------------------------------------------------------
## Licensed to the Apache Software Foundation (ASF) under one or more
## contributor license agreements.  See the NOTICE file distributed with
## this work for additional information regarding copyright ownership.
## The ASF licenses this file to You under the Apache License, Version 2.0
## (the "License"); you may not use this file except in compliance with
## the License.  You may obtain a copy of the License at
##
##      http://www.apache.org/licenses/LICENSE-2.0
##
## Unless required by applicable law or agreed to in writing, software
## distributed under the License is distributed on an "AS IS" BASIS,
## WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
## See the License for the specific language governing permissions and
## limitations under the License.
## ---------------------------------------------------------------------------

client {
    application.id = edu-pay-server
    transaction.service.group = my_tx_group
}

```

```
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) {                    事务ID（Fescar-Xid）
18.         String xid = RootContext.getXID();                           事务的控制有这个事务的唯一ID控制的
19.         if(StringUtils.isNotBlank(xid)){
20.             template.header("Fescar-Xid",xid);
21.         }
22.     }
23. }
```

f.application.yml 配置文件修改

```yaml
spring:
  application:
    name: edu-pay-boot
  cloud:
    alibaba:
        seata:
            tx-service-group: my_tx_group
    config:
      discovery:
        enabled: true
        service-id: edu-config-boot
      name: ${spring.application.name}
      profile: @profile@
  main:
    allow-bean-definition-overriding: true

ribbon:
###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。
 ReadTimeout: 10000
###指的是建立连接后从服务器读取到可用资源所用的时间。
 ConnectTimeout: 10000
#注册到Eureka服务中心
eureka:
  client:
    service-url:
      # 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
      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
@GlobalTransactional(name="cancelPayOrder_tx",rollbackFor = Exception.class)
public void cancelPayOrder(CancelPayOrderDTO cancelPayOrderDTO) {
    PayOrder payOrderDB = this.getById(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失败");
    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.getGoodsC
    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.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: CompositePropertySo
2020-07-28 17:45:29.533 [restartedMain] INFO  com.lagou.edu.pay.LagouEduPayApplication:646 - No active profile set, falling back to default pr
2020-07-28 17:45:32.973 [restartedMain] WARN  o.springframework.boot.actuate.endpoint.EndpointId:131 - Endpoint ID 'bus-env' contains invalid
2020-07-28 17:45:32.983 [restartedMain] WARN  o.springframework.boot.actuate.endpoint.EndpointId:131 - Endpoint ID 'bus-refresh' contains inva
2020-07-28 17:45:33.516 [restartedMain] INFO  o.s.d.r.config.RepositoryConfigurationDelegate:244 - Multiple Spring Data modules found, enteri
2020-07-28 17:45:33.521 [restartedMain] INFO  o.s.d.r.config.RepositoryConfigurationDelegate:126 - Bootstrapping Spring Data repositories in D
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.springframework.boot.actuate.endpoint.EndpointId:131 - Endpoint ID 'service-registry' contains
2020-07-28 17:45:35.169 [restartedMain] INFO  o.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 e
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$BeanPostProcessorChecker:330 - Bean 'dataSourceConfigu
2020-07-28 17:45:35.466 [restartedMain] INFO  io.seata.config.FileConfiguration:121 - The configuration file used is registry.conf
2020-07-28 17:45:35.499 [restartedMain] INFO  io.seata.config.FileConfiguration:121 - The configuration file used is file.conf
2020-07-28 17:45:35.514 [restartedMain] INFO  i.seata.spring.annotation.GlobalTransactionScanner:162 - Initializing Global Transaction Clients
2020-07-28 17:45:35.866 [restartedMain] INFO  io.seata.core.rpc.netty.AbstractRpcRemotingClient:147 - RpcClientBootstrap has started
2020-07-28 17:45:35.873 [restartedMain] INFO  i.seata.spring.annotation.GlobalTransactionScanner:170 - Transaction Manager Client is initiali
2020-07-28 17:45:35.944 [restartedMain] INFO  io.seata.rm.datasource.AsyncWorker:125 - Async Commit Buffer Limit: 10000
2020-07-28 17:45:35.945 [restartedMain] INFO  io.seata.rm.datasource.xa.ResourceManagerXA:40 - ResourceManagerXA init ...
2020-07-28 17:45:35.975 [restartedMain] INFO  io.seata.core.rpc.netty.AbstractRpcRemotingClient:147 - RpcClientBootstrap has started
2020-07-28 17:45:35.975 [restartedMain] INFO  i.seata.spring.annotation.GlobalTransactionScanner:175 - Resource Manager is initialized. applic
2020-07-28 17:45:35.976 [restartedMain] INFO  i.seata.spring.annotation.GlobalTransactionScanner:179 - Global Transaction Clients are initiali
2020-07-28 17:45:36.191 [restartedMain] INFO  o.s.c.s.PostProcessorRegistrationDelegate$BeanPostProcessorChecker:330 - Bean 'org.springframewo
2020-07-28 17:45:36.437 [restartedMain] INFO  o.s.c.s.PostProcessorRegistrationDelegate$BeanPostProcessorChecker:330 - Bean 'org.springframewo
2020-07-28 17:45:36.831 [restartedMain] INFO  o.s.c.s.PostProcessorRegistrationDelegate$BeanPostProcessorChecker:330 - Bean 'integrationDispos
2020-07-28 17:45:36.966 [restartedMain] INFO  o.s.c.s.PostProcessorRegistrationDelegate$BeanPostProcessorChecker:330 - Bean 'org.springframewo
```

可以看到客户端的服务已经读取了 配置文件

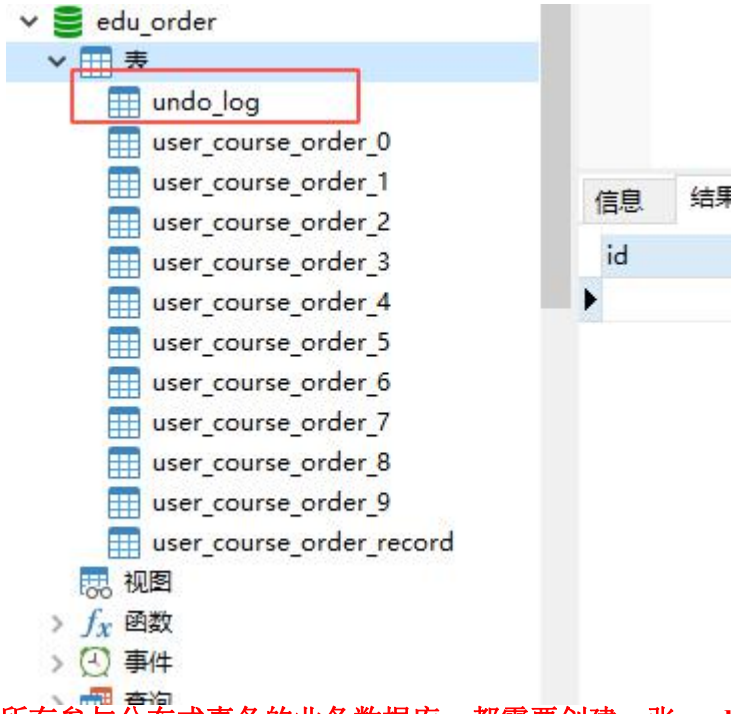
```
om.netflix.discovery.DiscoveryClient:1056 - The response status is 200
om.netflix.discovery.DiscoveryClient:1317 - Not registering with Eureka server per configuration
om.netflix.discovery.DiscoveryClient:449 - Discovery Client initialized at timestamp 1595929546027 with initial instances count: 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:/113.31.10
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
i.s.s.a.datasource.SeataAutoDataSourceProxyCreator:45 - Auto proxy of [dataSource]
i.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

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