# 1、Seata 1.3.0源码编译

## 1.1准备源码包

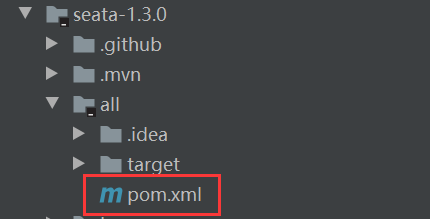
Github地址：

大小大概2.2M左右

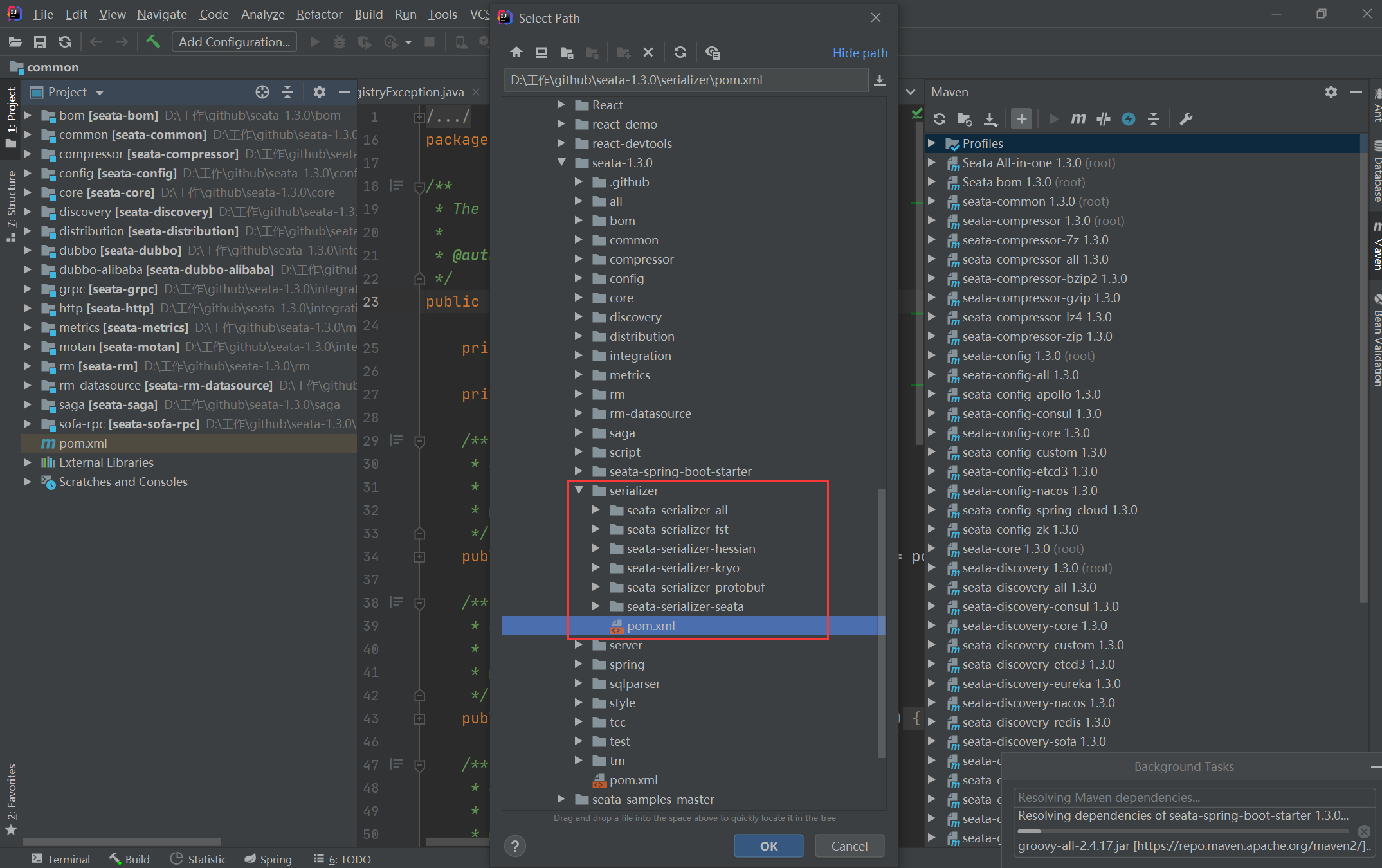


## **1.2导入到idea中**

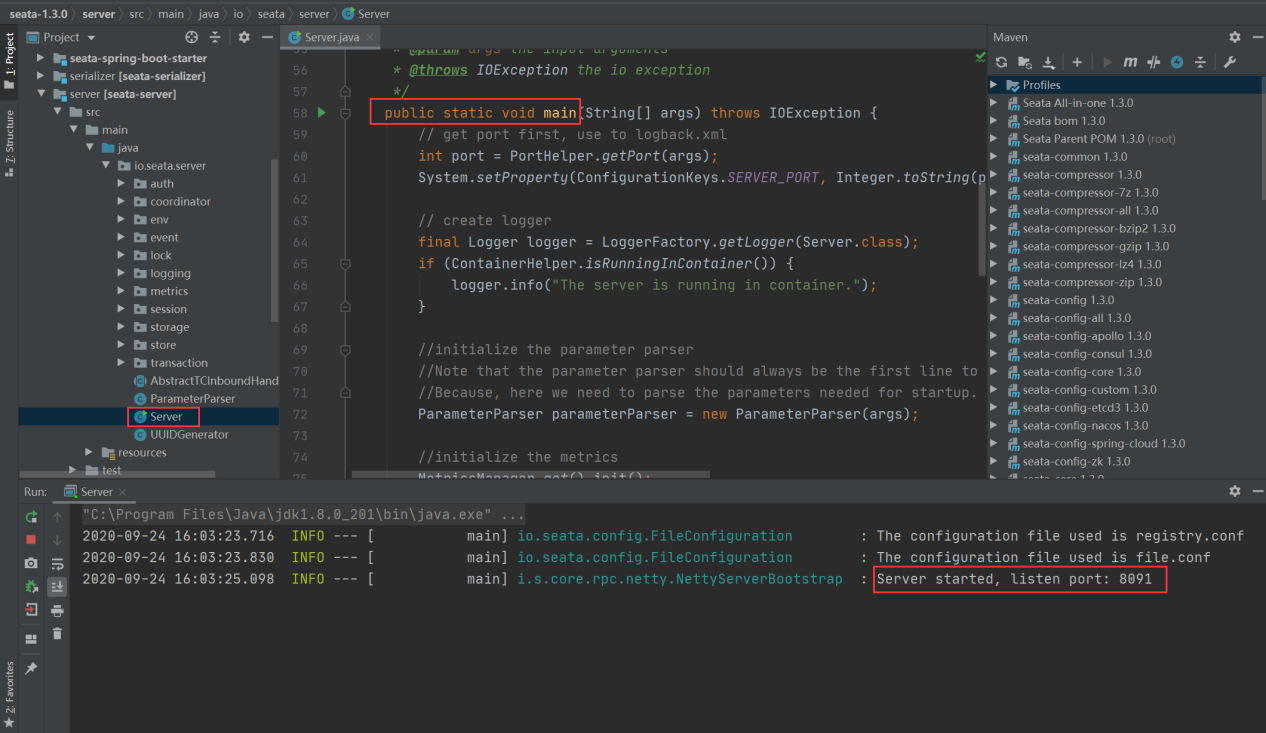
先将all模块中的pom.xml加载进来



## **1.3依次将各个模块中的pom文件加进来**



## **1.4找到server模块，运行Main方法**



# Windows配置Nacos

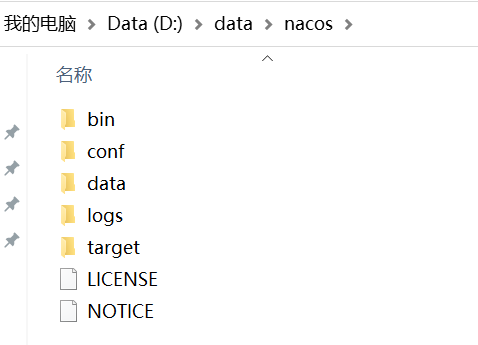
## 2.1 准备包

版本：1.3.2

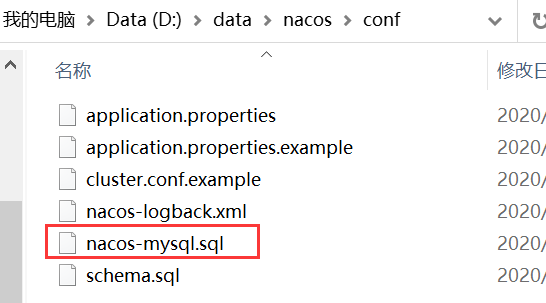
注意：这个版本的nacos连接mysql8，没有问题，很顺利！

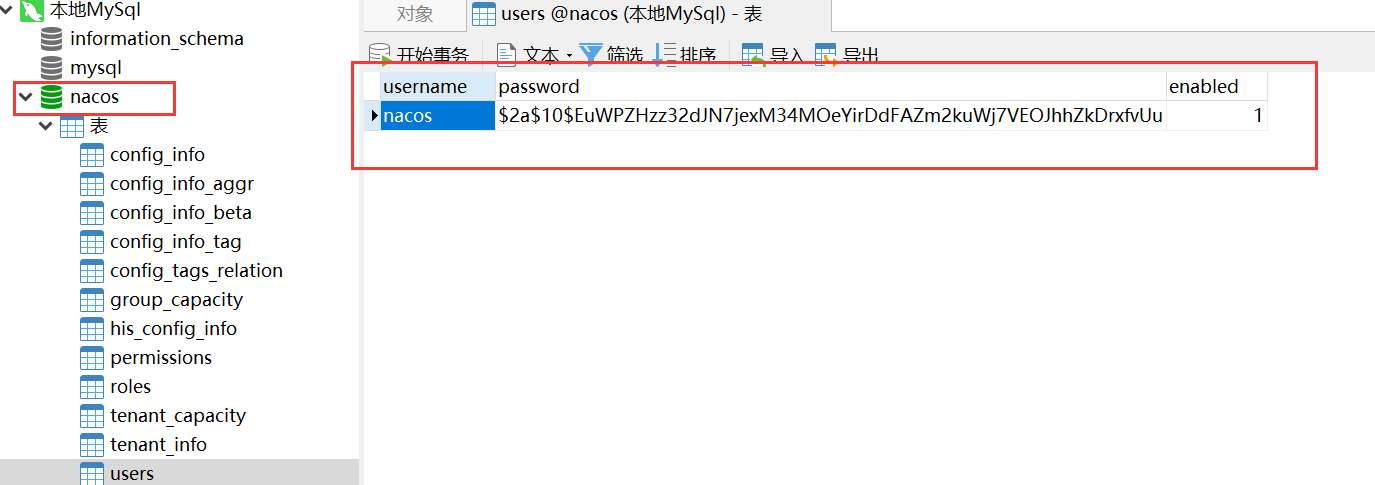


解压到指定路径



## 2.2 执行conf文件夹下的mysql脚本



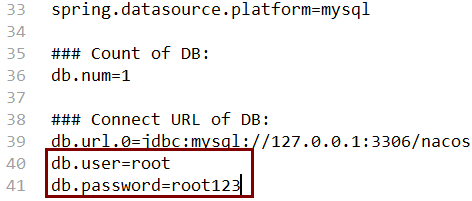


## 2.3 修改conf文件夹下application.properties

定位到mysql配置

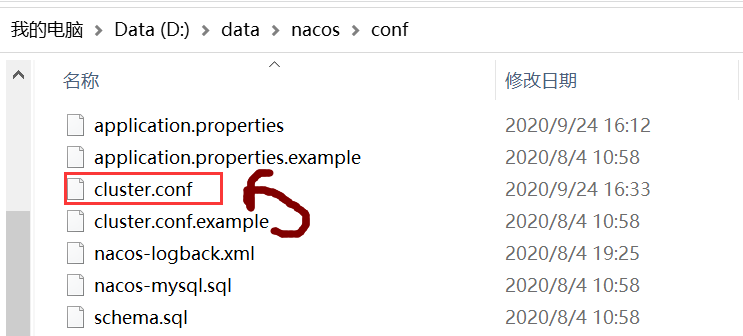


同时改下，用户名和密码，我们使用root账户



## 2.4 修改nacos集群模式为单机版

将conf文件夹下的cluster.conf.example复制一份出来，重命名为cluster.conf



对它的内容稍作修改，将ip换成自己的，将端口号8847改成8848

原来的内容：

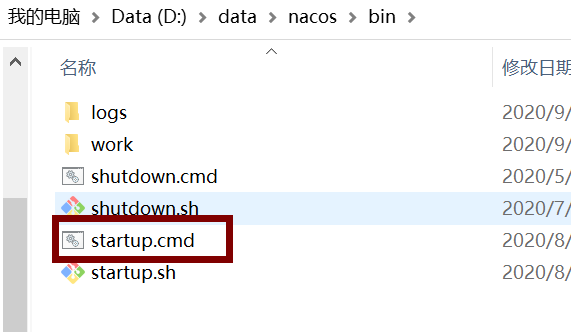
|  |
| --- |
| #it is ip  #example  192.168.16.101:8847  192.168.16.102  192.168.16.103 |

改后现在的内容：

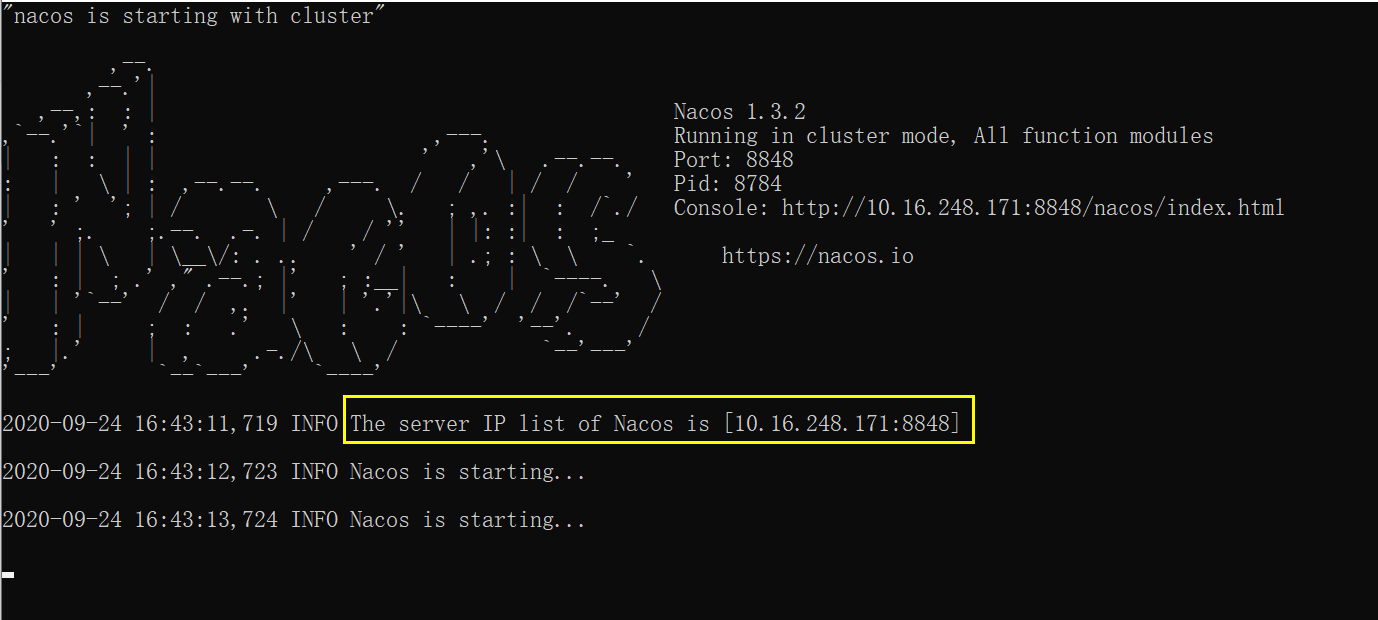
|  |
| --- |
| #it is ip  #example  10.16.248.171:8848 |

## 2.5 启动nacos

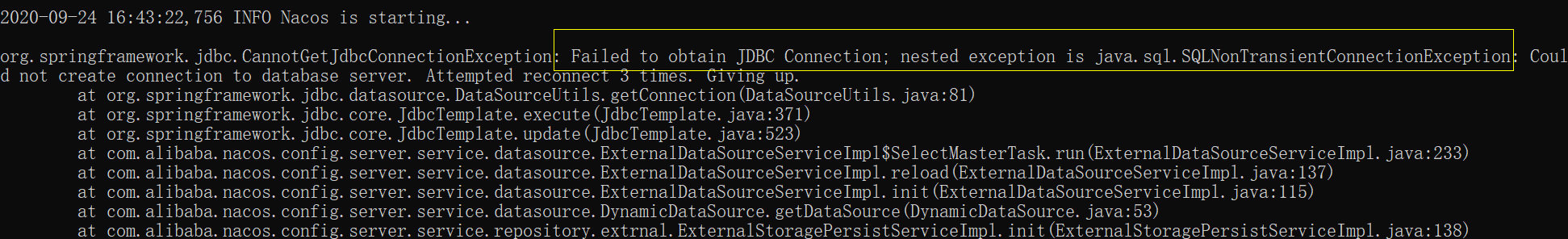
在bin目录下执行startup.cmd，如下：



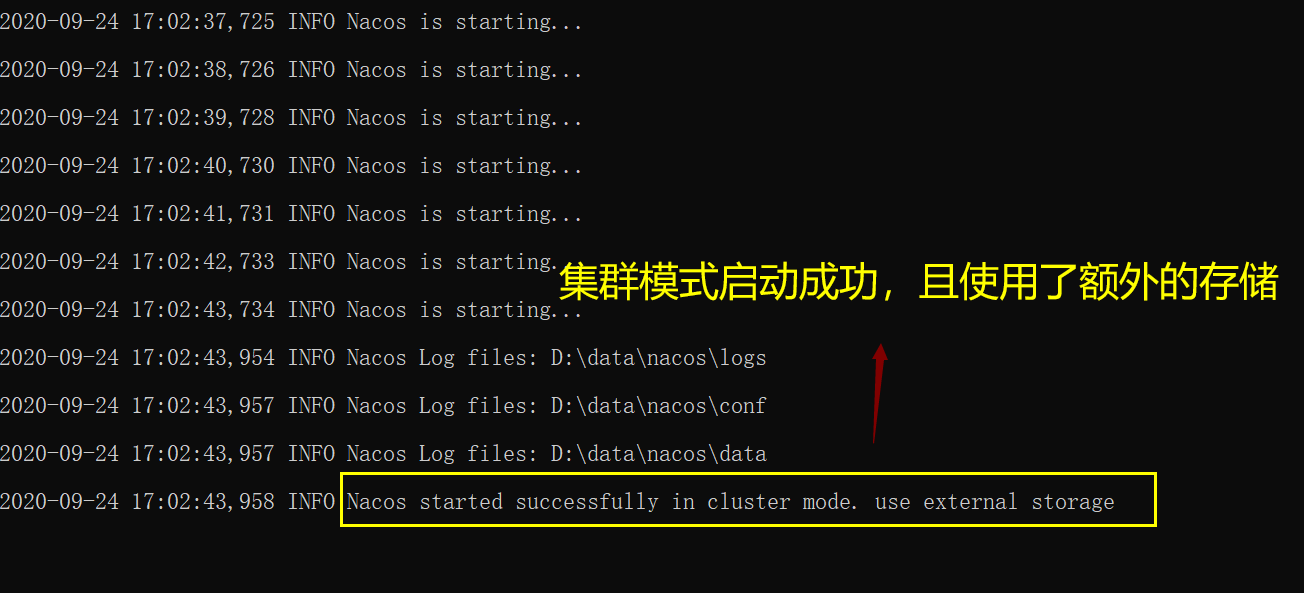
控制台输出结果如下：



如果mysql配置的有问题，你会看到下面输出了连不上的日志：



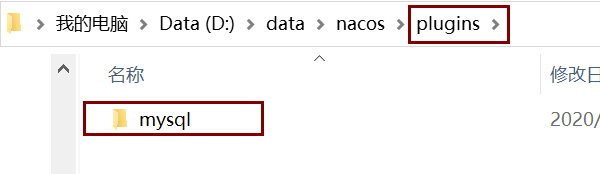
如果mysql配置没有问题，你会看到，success的字样：



## 2.6 解决nacos连不上mysql8数据库的问题

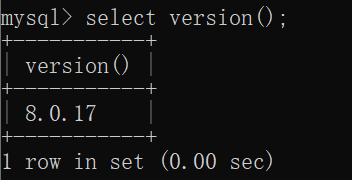
低版本的nacos存在无法连上mysql库的问题，如果遇到了，要么升级nacos版本，要么按照下面的步骤做！

### 2.6.1 在nacos安装目录下新建plugins/mysql文件夹



### 2.6.3 将mysql-connector-java-8.0.17.jar放进去

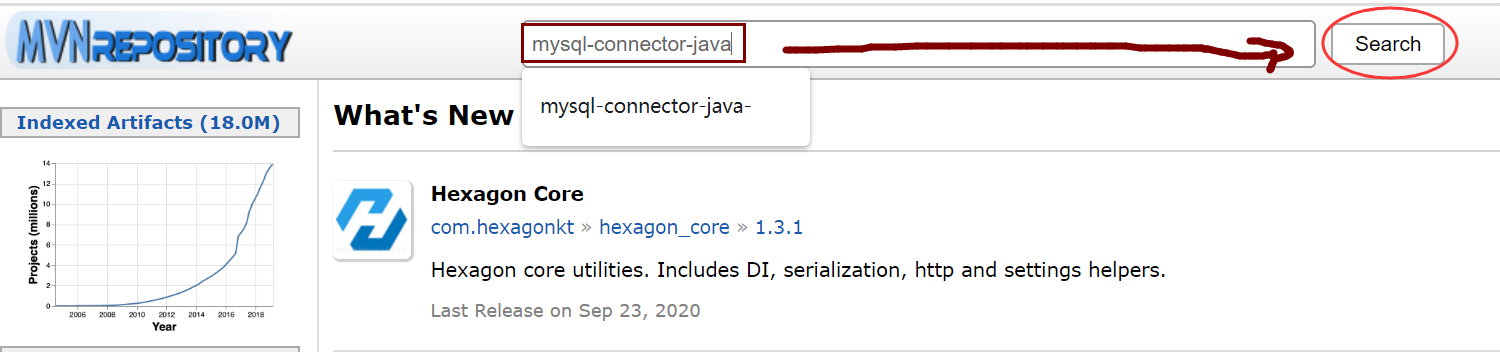
#### 2.6.3.1 先查看mysql的版本



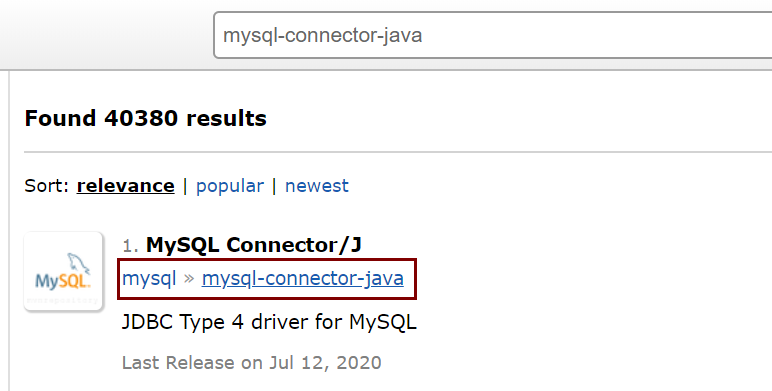
#### 2.6.3.2 去Maven公仓中查jar包

Maven地址：<https://mvnrepository.com/>

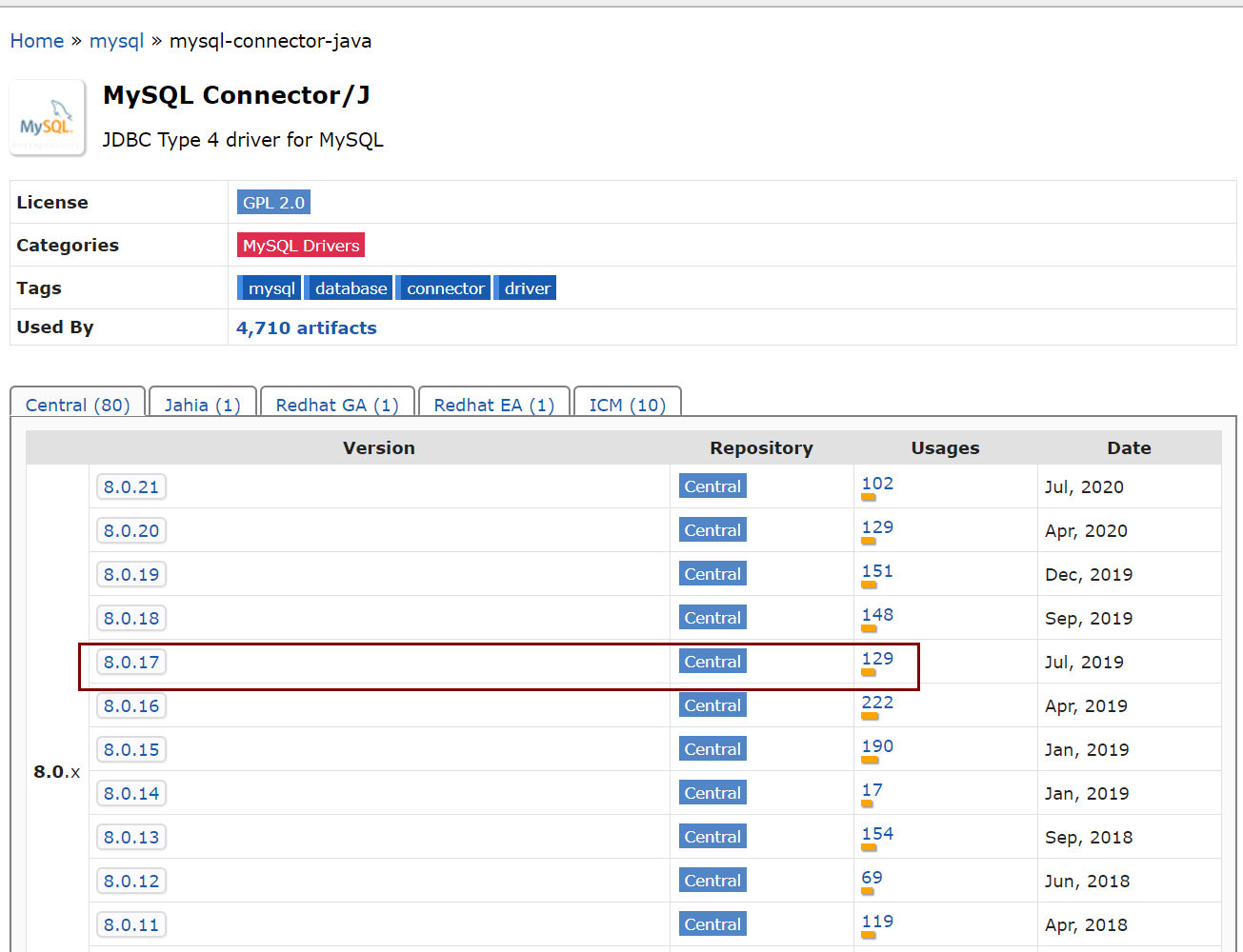
注意，第一次进去，需要进行图片验证，比如找汽车啊、打飞机啊、啊不，找飞机啊....



**图 1 搜索mysql驱动包**



**图2 搜索到的结果**



**图3 定位到合适的版本**

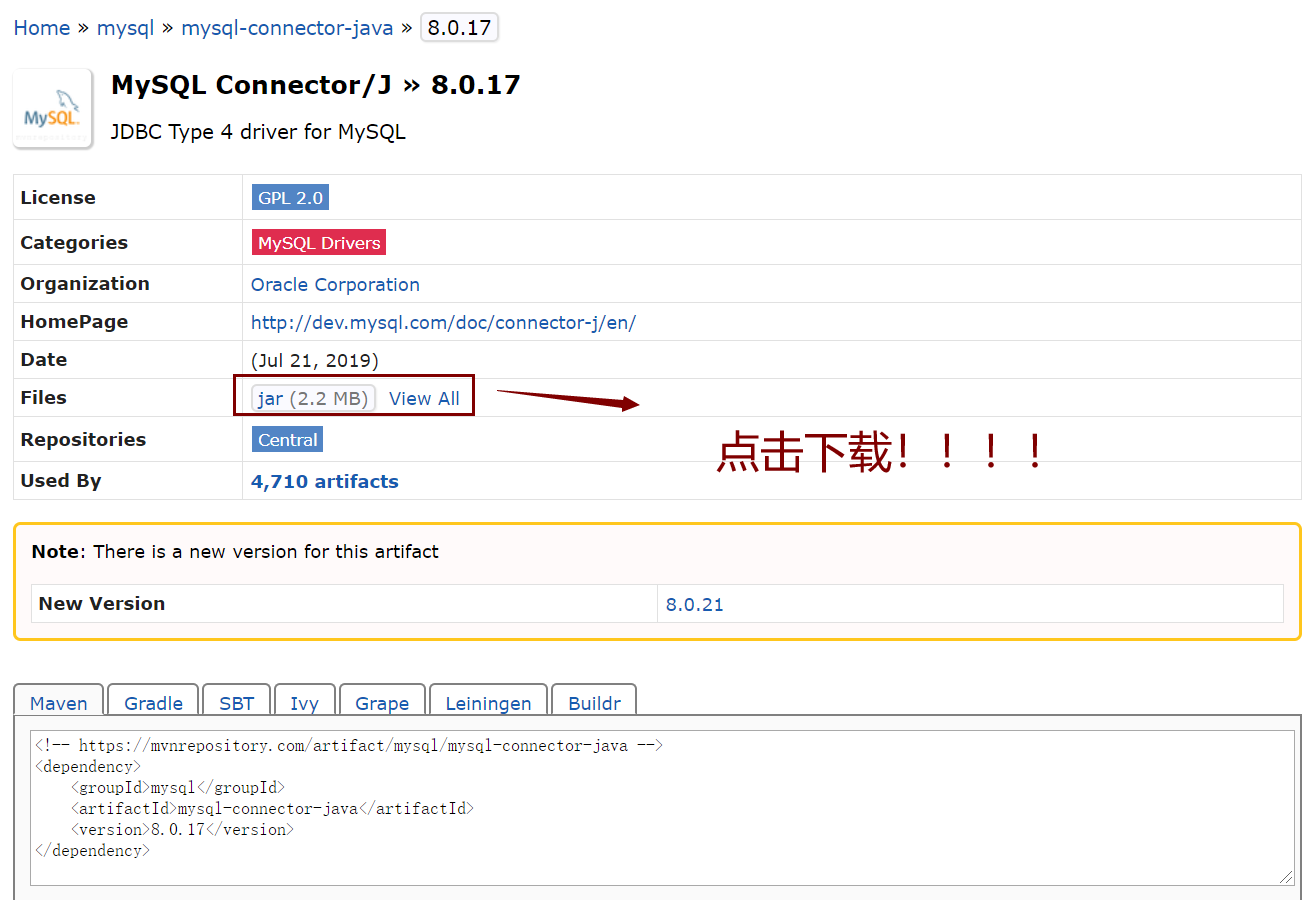
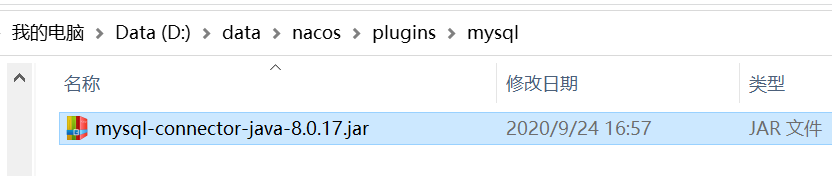


图4 下载驱动包



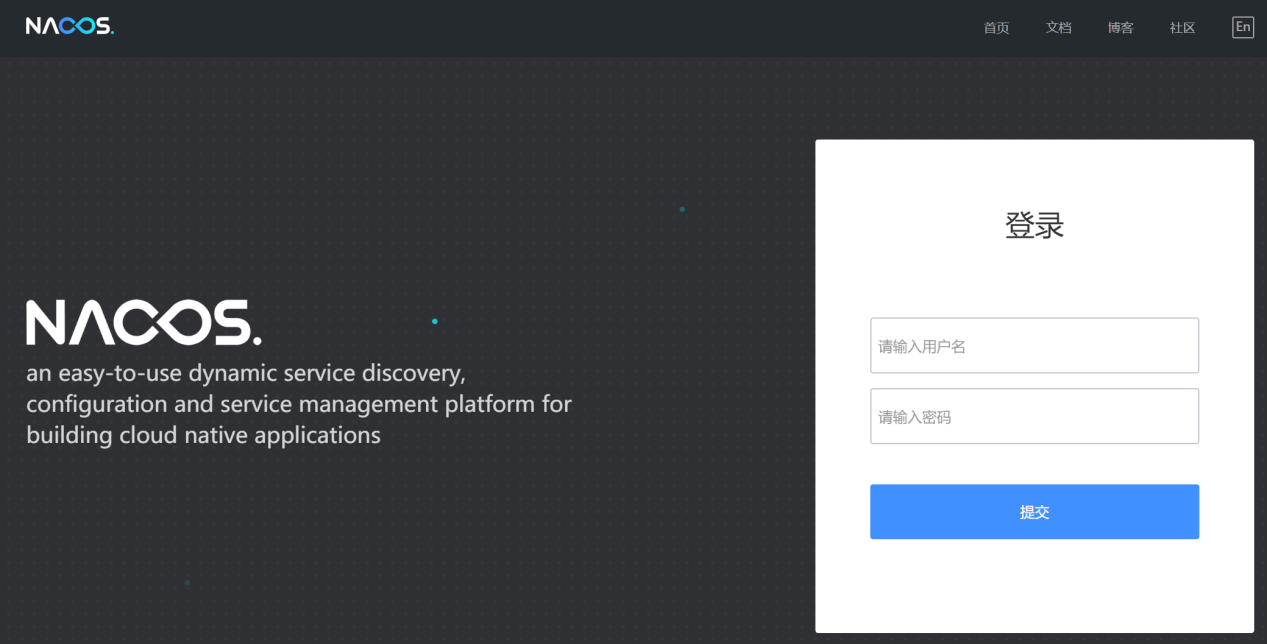
下完后，放到mysql文件夹下，如下：



最后，重启一次nacos即可！

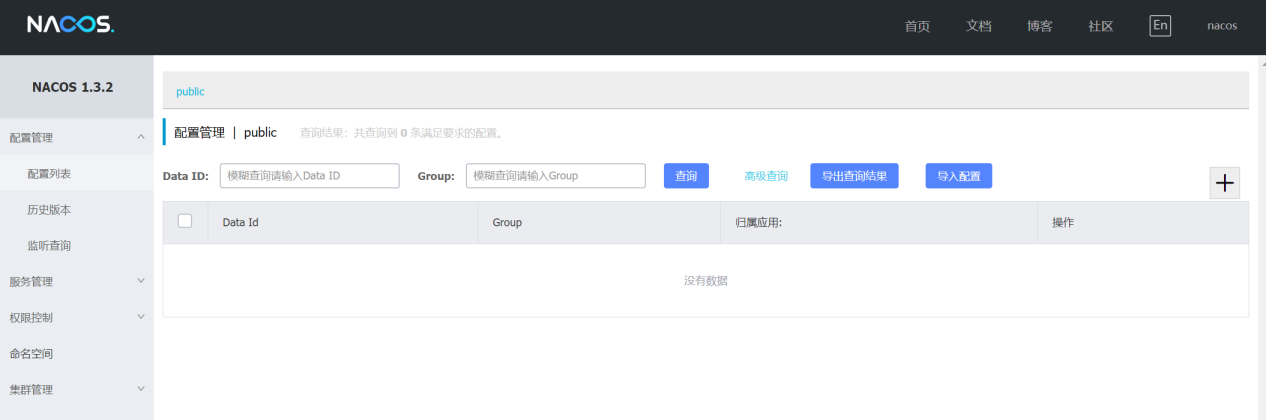
## 2.7 启动成功后，访问Nacos页面

浏览器输入: <http://10.16.248.171:8848/nacos/index.html>

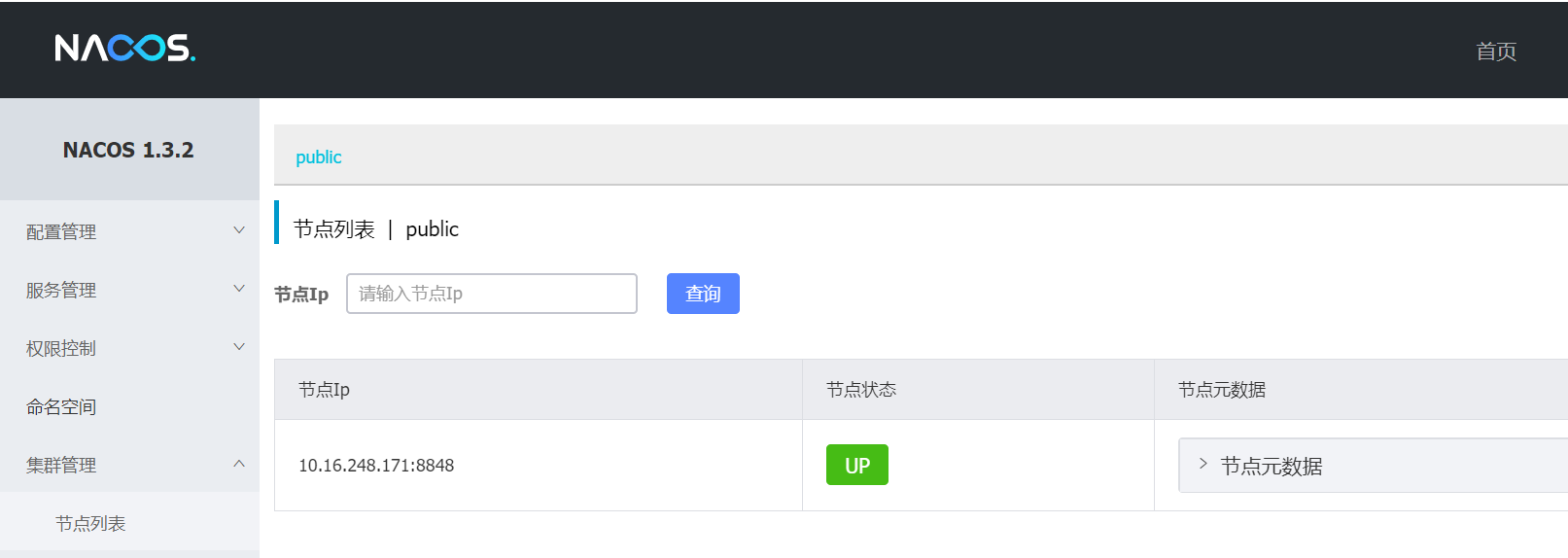


默认用户：nacos

默认密码：nacos

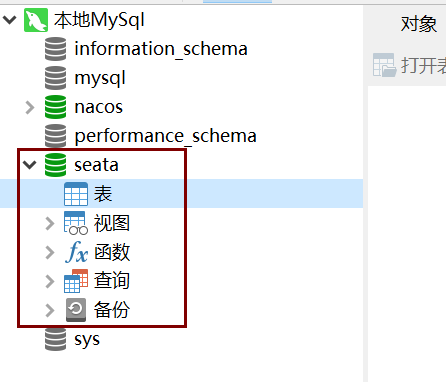


查看nacos节点



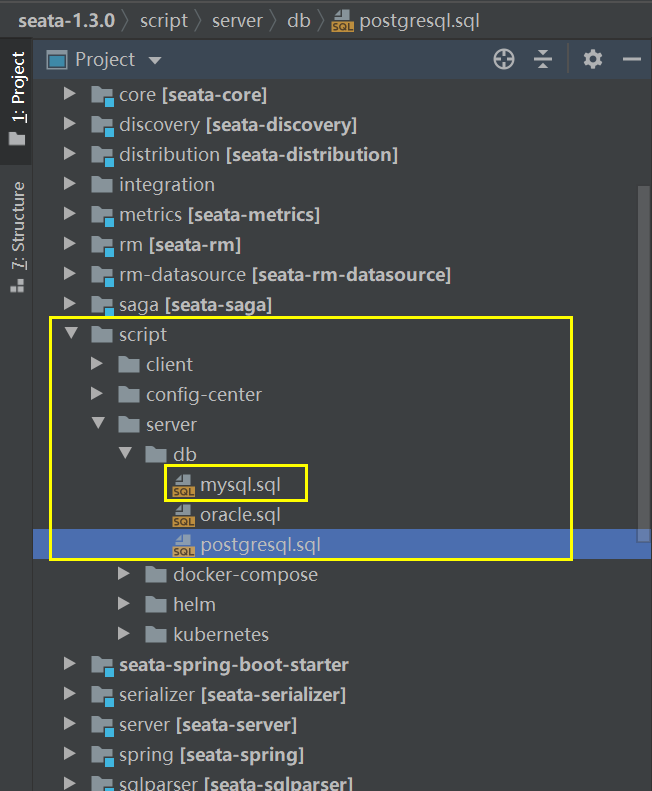
# Seata 1.3.0整合nacos 1.3.2

## 3.1 建库（Seata）

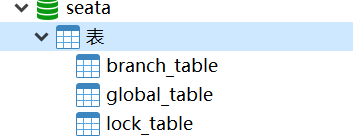


## 3.2 建表（seata）

从seata 1.3.0源码中，找到script目录，找到db下面mysql的脚本如下：



建好后，刷新，如下：



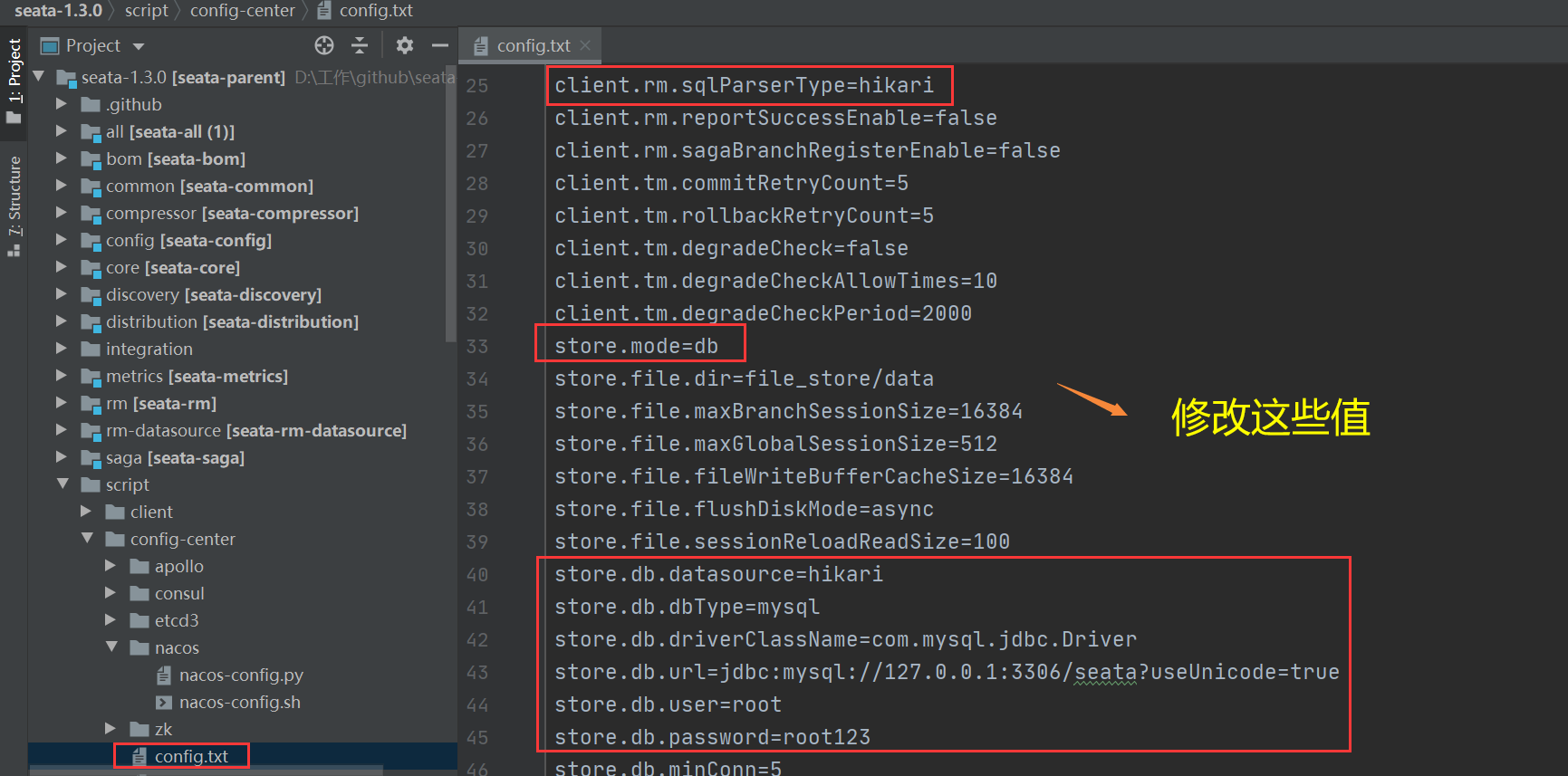
## 3.3 向nacos服务添加配置

### 3.3.1 没添加之前



### 3.3.2 找到config.txt文件

修改如下内容：

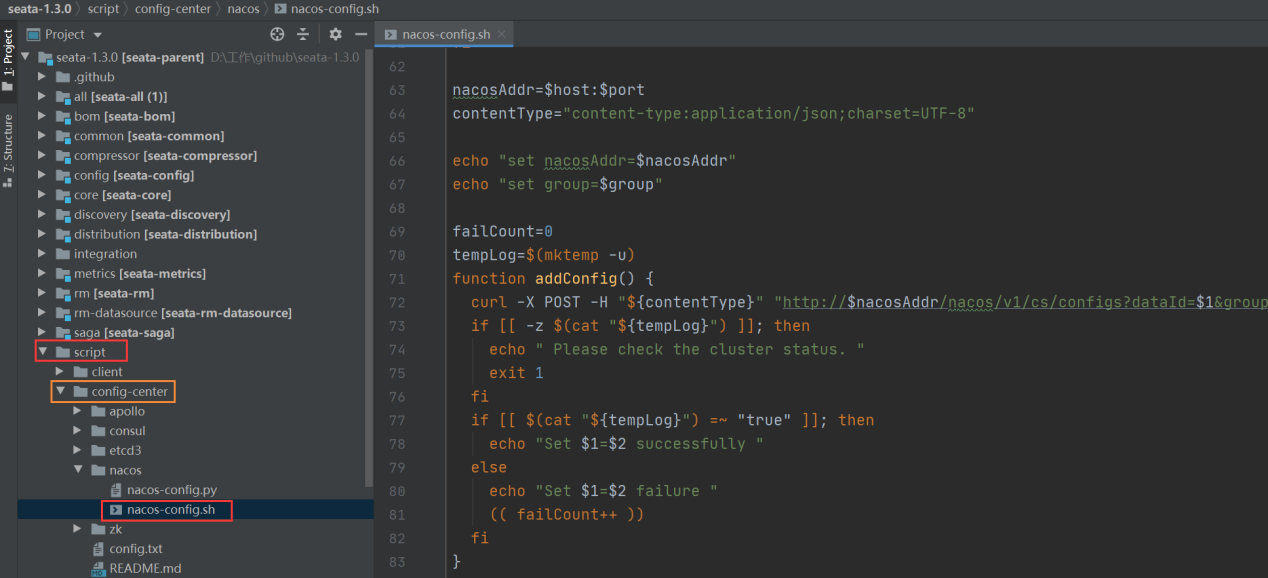


具体内容如下：

|  |
| --- |
| transport.type=TCP transport.server=NIO transport.heartbeat=true transport.enableClientBatchSendRequest=false transport.threadFactory.bossThreadPrefix=NettyBoss transport.threadFactory.workerThreadPrefix=NettyServerNIOWorker transport.threadFactory.serverExecutorThreadPrefix=NettyServerBizHandler transport.threadFactory.shareBossWorker=false transport.threadFactory.clientSelectorThreadPrefix=NettyClientSelector transport.threadFactory.clientSelectorThreadSize=1 transport.threadFactory.clientWorkerThreadPrefix=NettyClientWorkerThread transport.threadFactory.bossThreadSize=1 transport.threadFactory.workerThreadSize=default transport.shutdown.wait=3 service.vgroupMapping.seata\_example\_tx\_group=default service.default.grouplist=127.0.0.1:8091 service.enableDegrade=false service.disableGlobalTransaction=false client.rm.asyncCommitBufferLimit=10000 client.rm.lock.retryInterval=10 client.rm.lock.retryTimes=30 client.rm.lock.retryPolicyBranchRollbackOnConflict=true client.rm.reportRetryCount=5 client.rm.tableMetaCheckEnable=false client.rm.sqlParserType=hikari client.rm.reportSuccessEnable=false client.rm.sagaBranchRegisterEnable=false client.tm.commitRetryCount=5 client.tm.rollbackRetryCount=5 client.tm.degradeCheck=false client.tm.degradeCheckAllowTimes=10 client.tm.degradeCheckPeriod=2000 store.mode=db store.file.dir=file\_store/data store.file.maxBranchSessionSize=16384 store.file.maxGlobalSessionSize=512 store.file.fileWriteBufferCacheSize=16384 store.file.flushDiskMode=async store.file.sessionReloadReadSize=100 store.db.datasource=hikari store.db.dbType=mysql store.db.driverClassName=com.mysql.jdbc.Driver store.db.url=jdbc:mysql://127.0.0.1:3306/seata?useUnicode=true store.db.user=root store.db.password=root123 store.db.minConn=5 store.db.maxConn=30 store.db.globalTable=global\_table store.db.branchTable=branch\_table store.db.queryLimit=100 store.db.lockTable=lock\_table store.db.maxWait=5000 store.redis.host=127.0.0.1 store.redis.port=6379 store.redis.maxConn=10 store.redis.minConn=1 store.redis.database=0 store.redis.password=null store.redis.queryLimit=100 server.recovery.committingRetryPeriod=1000 server.recovery.asynCommittingRetryPeriod=1000 server.recovery.rollbackingRetryPeriod=1000 server.recovery.timeoutRetryPeriod=1000 server.maxCommitRetryTimeout=-1 server.maxRollbackRetryTimeout=-1 server.rollbackRetryTimeoutUnlockEnable=false client.undo.dataValidation=true client.undo.logSerialization=jackson client.undo.onlyCareUpdateColumns=true server.undo.logSaveDays=7 server.undo.logDeletePeriod=86400000 client.undo.logTable=undo\_log client.log.exceptionRate=100 transport.serialization=seata transport.compressor=none metrics.enabled=false metrics.registryType=compact metrics.exporterList=prometheus metrics.exporterPrometheusPort=9898 |

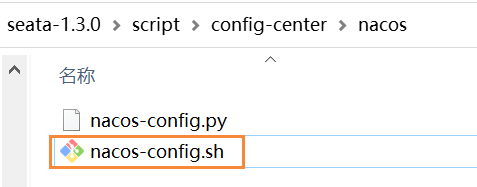
### 3.3.3 找到nacos-config.sh脚本

在源码中，定位到script/config-center/nacos/包，找到nacos-config.sh文件



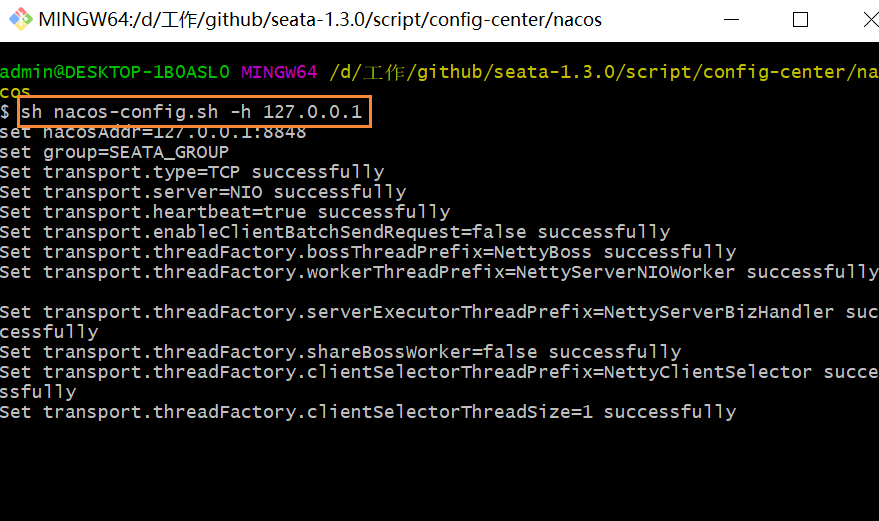
### 3.3.4 执行nacos-config.sh脚本

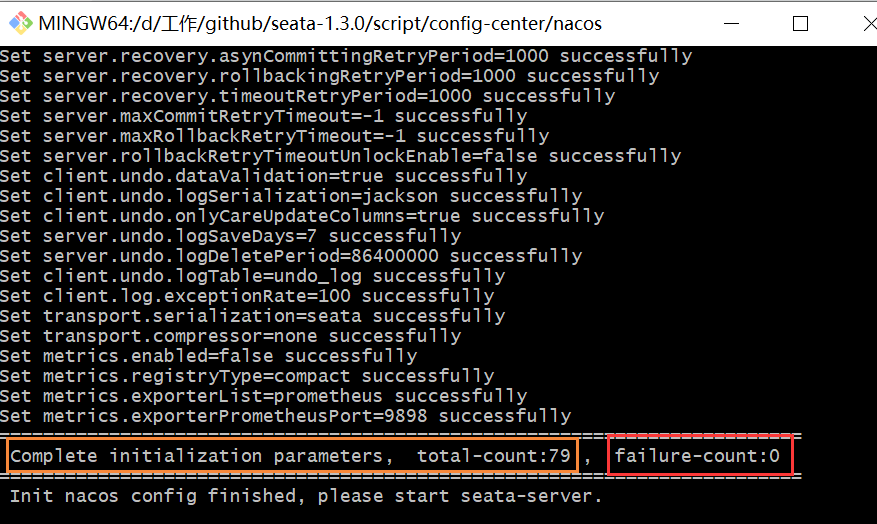
在当前根路径下，打开git bash，执行该脚本



|  |
| --- |
| **sh nacos-config.sh -h 127.0.0.1** |

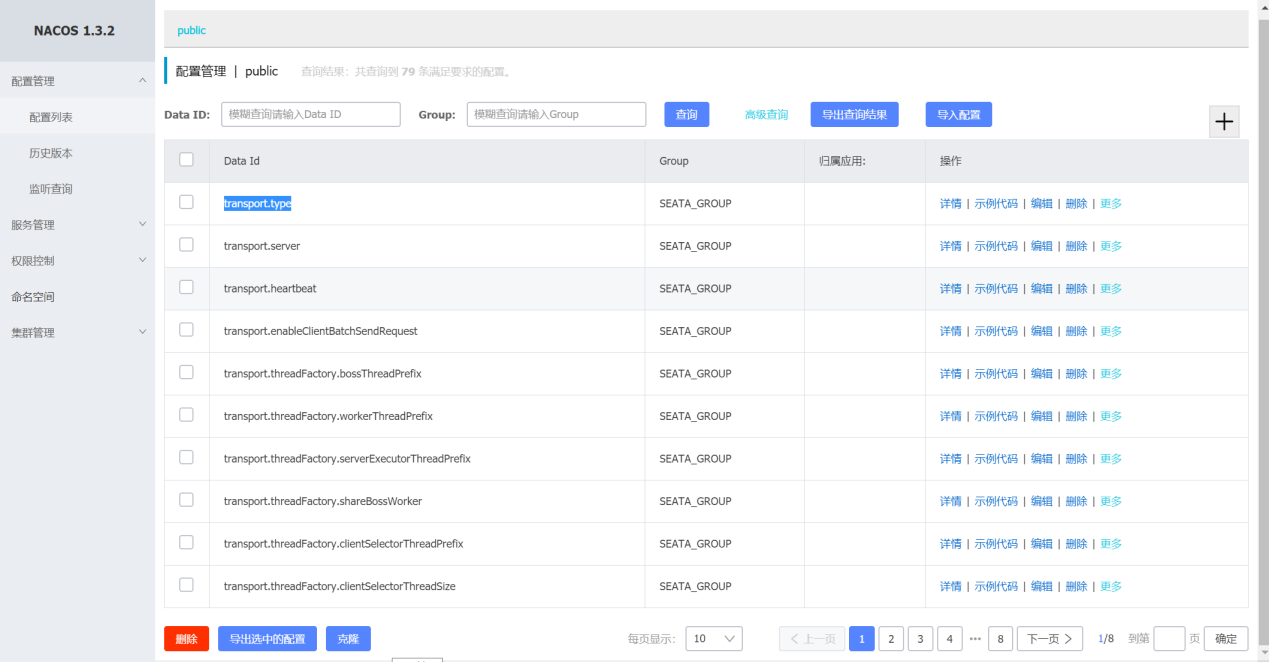
执行完成之后，会将config.txt中配置的信息推送到nacos服务中





看到上面这段输出，说明nacos的配置已经全部设置成功了

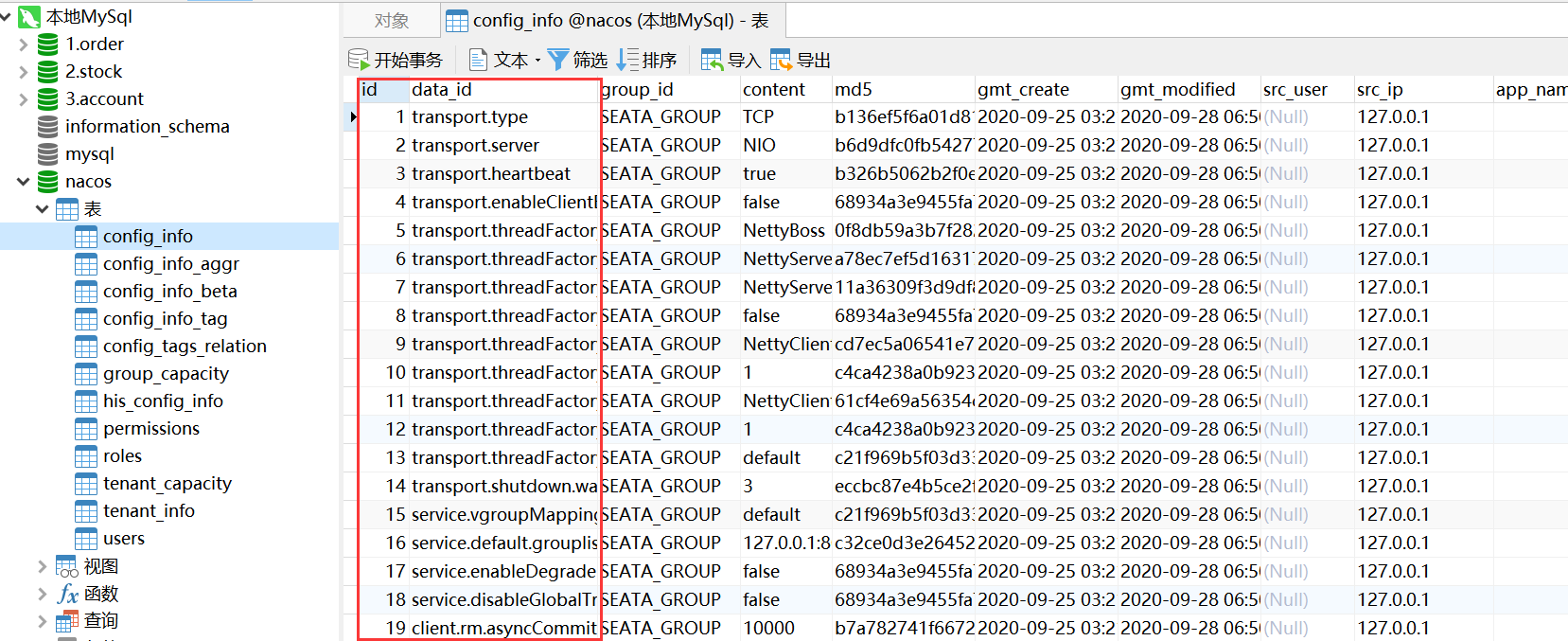
### 3.3.5 刷新页面，查看Nacos配置







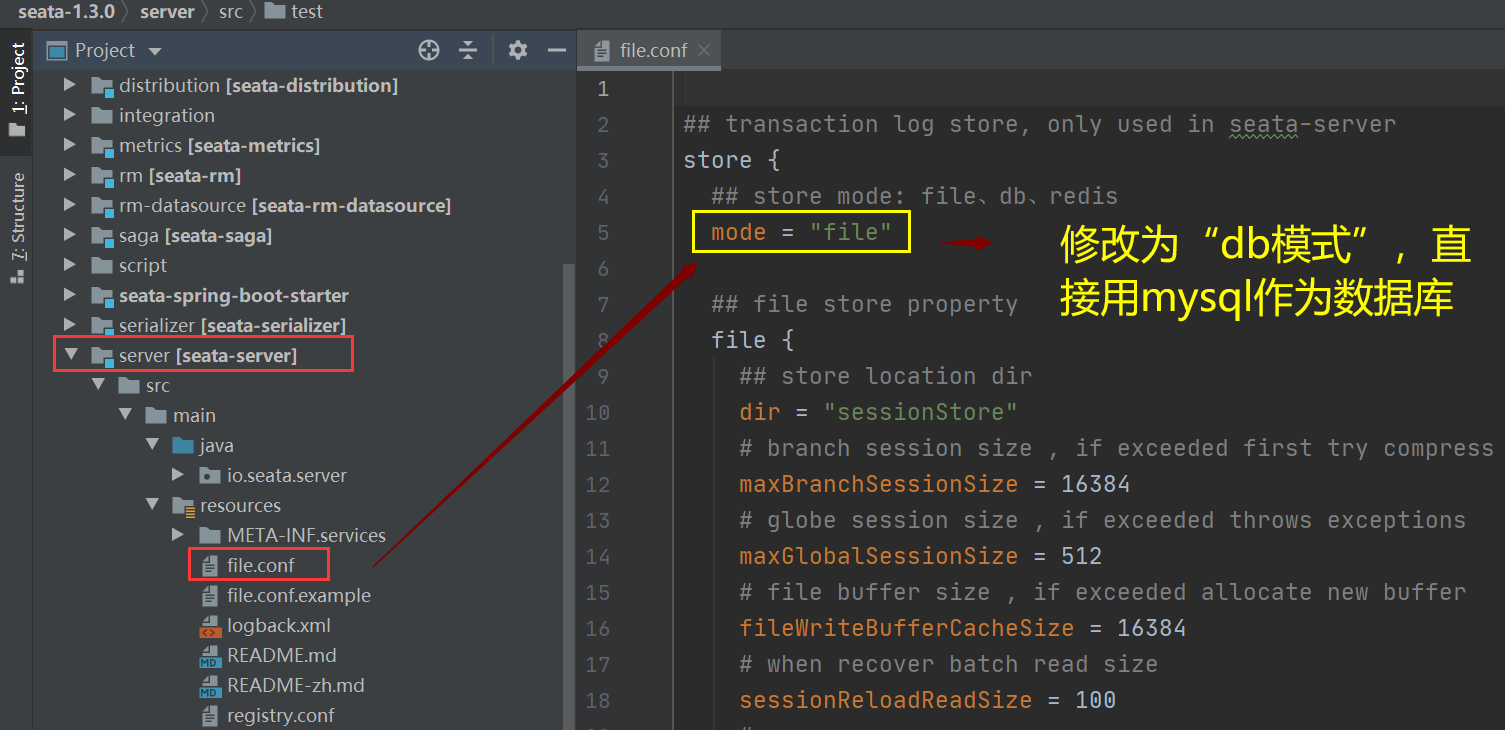
可以看到，对应的nacos库中的config\_info表中已经有配置数据了



## 3.4 修改seata配置（file.conf）

数据存储配置

**三种模式，默认是存在磁盘文件中，其他分别是存在数据库和redis缓存库中；**

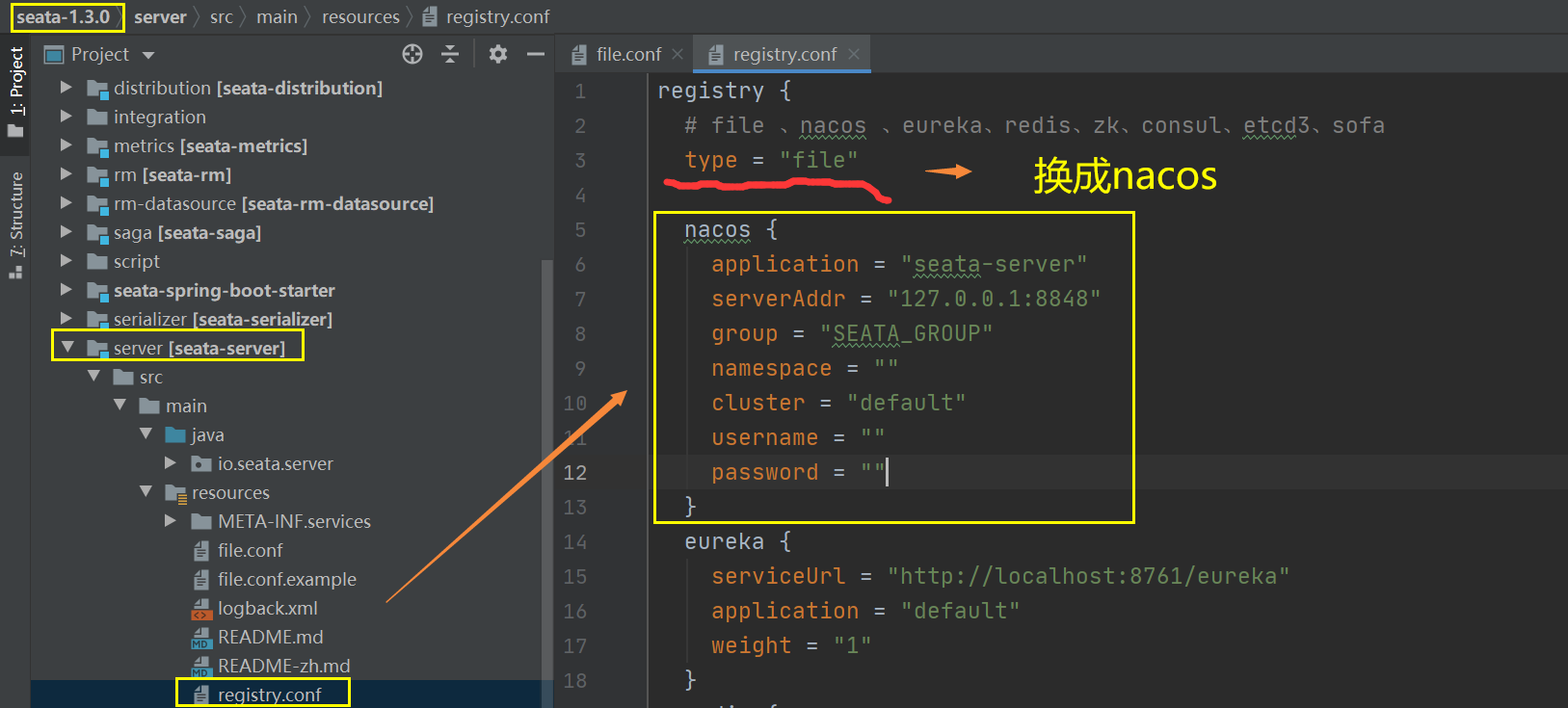


除了改这个地方外，还要把db对应的配置项改了，最终修改后的内容如下：

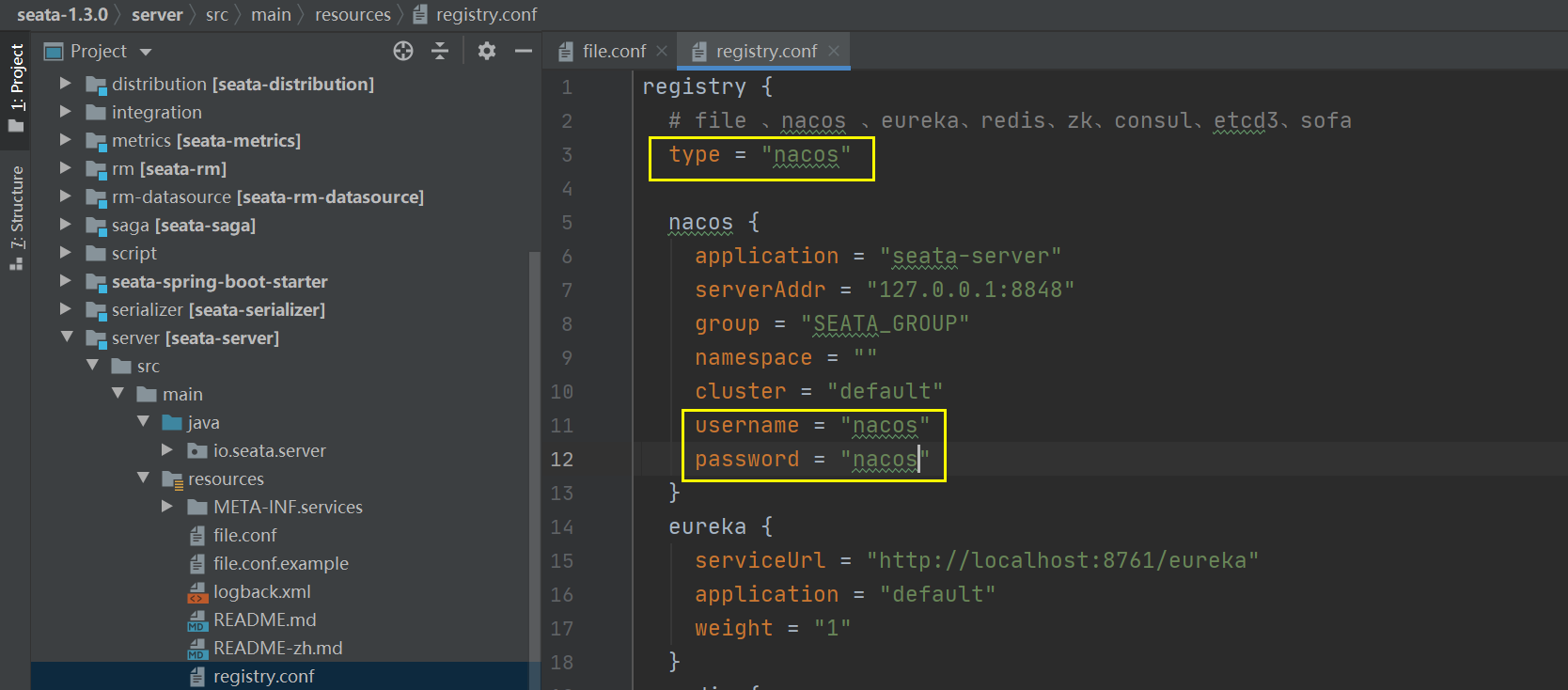
|  |
| --- |
| transport {  # tcp udt unix-domain-socket  type = "TCP"  #NIO NATIVE  server = "NIO"  #enable heartbeat  heartbeat = true  # the client batch send request enable  enableClientBatchSendRequest = false  #thread factory for netty  threadFactory {  bossThreadPrefix = "NettyBoss"  workerThreadPrefix = "NettyServerNIOWorker"  serverExecutorThreadPrefix = "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" }  ## transaction log store, only used in seata-server store {  ## store mode: file、db、redis  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)/HikariDataSource(hikari) etc.  datasource = "hikari"  ## mysql/oracle/postgresql/h2/oceanbase etc.  dbType = "mysql"  driverClassName = "com.mysql.jdbc.Driver"  url = "jdbc:mysql://127.0.0.1:3306/seata"  user = "root"  password = "root123"  minConn = 5  maxConn = 30  globalTable = "global\_table"  branchTable = "branch\_table"  lockTable = "lock\_table"  queryLimit = 100  maxWait = 5000  }   ## redis store property  redis {  host = "127.0.0.1"  port = "6379"  password = ""  database = "0"  minConn = 1  maxConn = 10  queryLimit = 100  }  }  ## server configuration, only used in server side server {  recovery {  #schedule committing retry period in milliseconds  committingRetryPeriod = 1000  #schedule asyn committing retry period in milliseconds  asynCommittingRetryPeriod = 1000  #schedule rollbacking retry period in milliseconds  rollbackingRetryPeriod = 1000  #schedule timeout retry period in milliseconds  timeoutRetryPeriod = 1000  }  undo {  logSaveDays = 7  #schedule delete expired undo\_log in milliseconds  logDeletePeriod = 86400000  }  #check auth  enableCheckAuth = true  #unit ms,s,m,h,d represents milliseconds, seconds, minutes, hours, days, default permanent  maxCommitRetryTimeout = "-1"  maxRollbackRetryTimeout = "-1"  rollbackRetryTimeoutUnlockEnable = false }  ## metrics configuration, only used in server side metrics {  enabled = false  registryType = "compact"  # multi exporters use comma divided  exporterList = "prometheus"  exporterPrometheusPort = 9898 } |

## 3.5 修改seata配置（registry.conf）

**Seata服务注册配置**



注册中心换成nacos后，由于本机的nacos是单个节点，所以serverAddr的值不用改，唯一需要改的就是username和password，改后的内容如下：

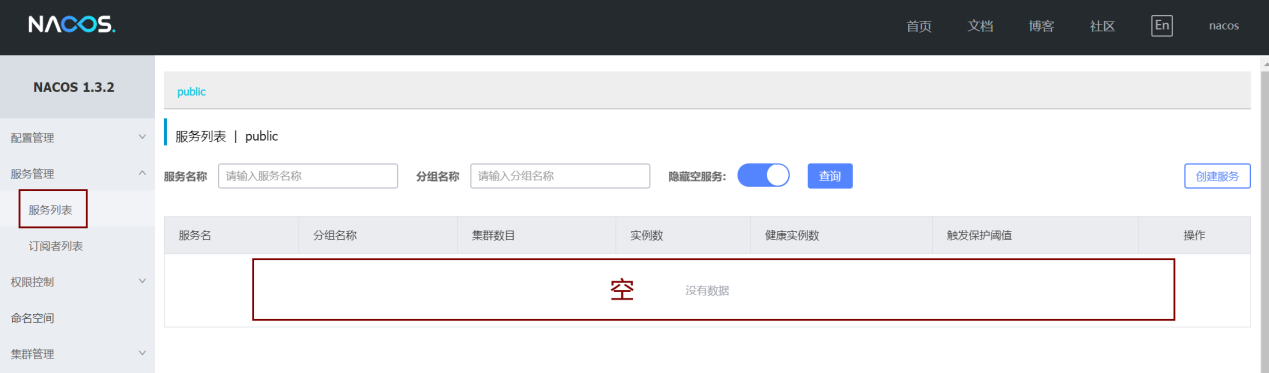


配置中心换成nacos后，由于本机的nacos是单个节点，所以serverAddr的值不用改，唯一需要改的就是username和password，改后的内容如下：

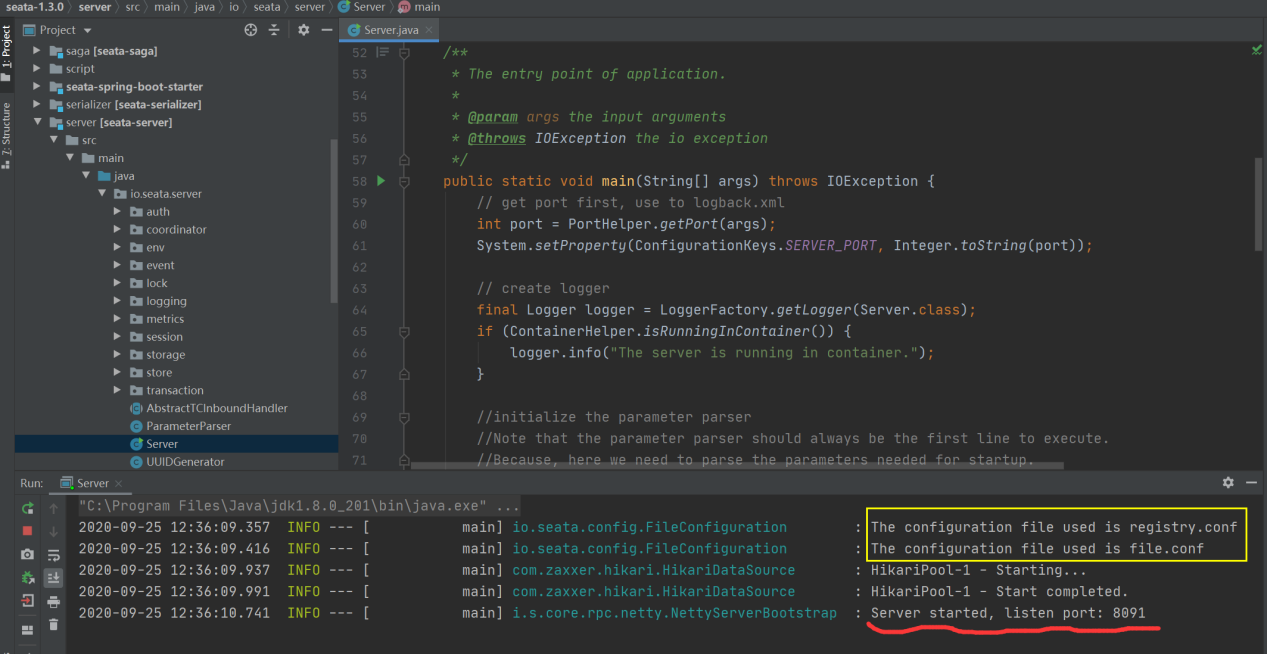
## 3.6 启动seata服务

### 3.6.1 没启动之前，查看下nacos服务列表

由于目前还没有任何服务与nacos服务互通（注册），所以，在nacos控制台中的服务列表中显示空



### 3.6.2 启动seata服务



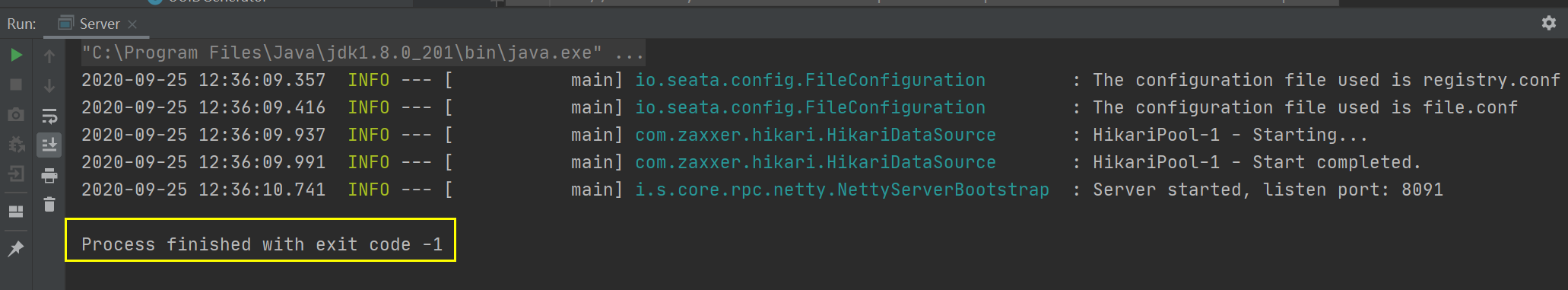
### 3.6.3 启动之后，查看下nacos服务列表



**点击详情，进服务详情页面：**



### 3.6.4 停掉seata-server实例



### 3.6.5 观察nacos服务列表

Nacos目前支持临时实例使用心跳上报方式维持活性，发送心跳的周期默认是**5秒**，Nacos服务端会在**15秒**没收到心跳后将实例设置为不健康，在**30秒**没收到心跳时将这个临时实例摘除。

停掉seata-server服务，过一会后，可以看到如下情况：



### 3.6.6 再观察nacos服务列表

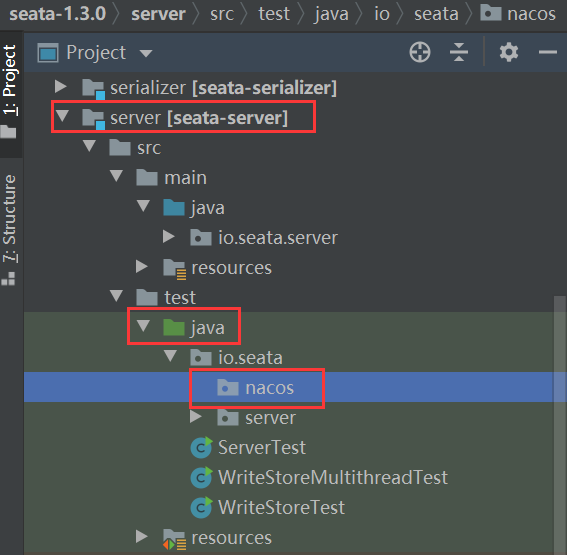
再过一会儿，可以看到如下情况：



# Nacos配置的动态管理

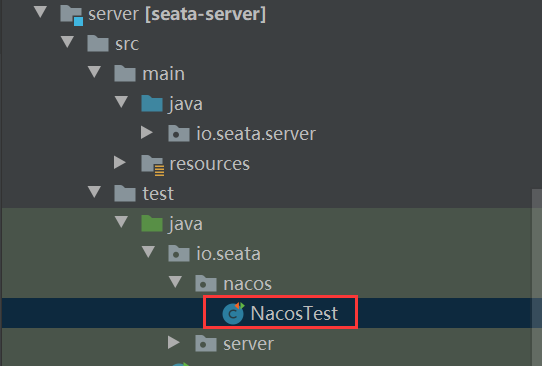
## 4.1 新建nacos测试包

在seata源码包下面的server模块，找到test包，在io.seata包下创建test包



## 4.2 创建nacos测试类

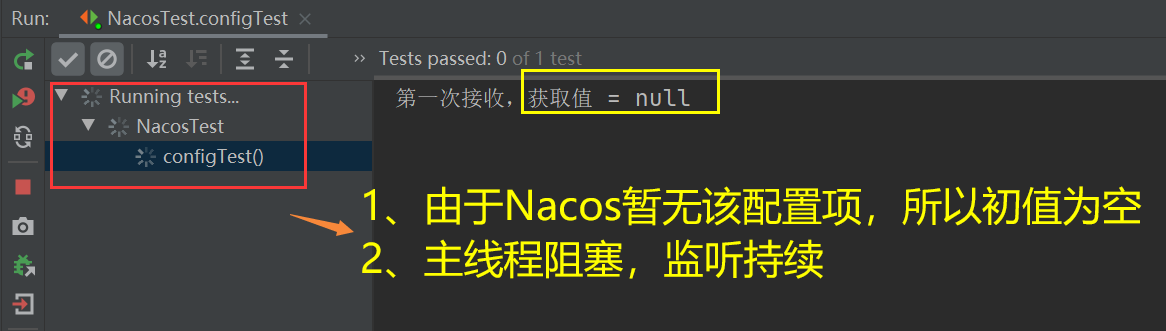
在io.seata.test包下创建NacosTest测试类



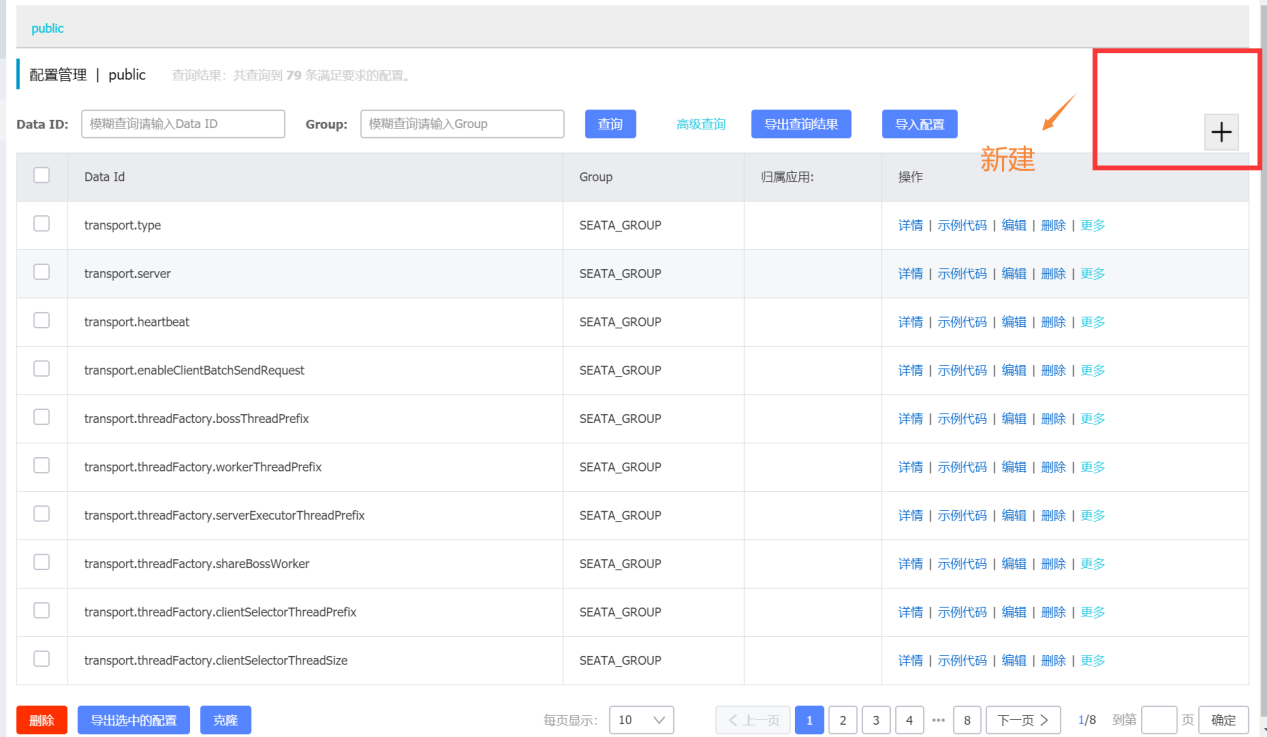
代码如下：

|  |
| --- |
| package io.seata.nacos;  import com.alibaba.nacos.api.NacosFactory;  import com.alibaba.nacos.api.config.ConfigService;  import com.alibaba.nacos.api.config.listener.Listener;  import com.alibaba.nacos.api.exception.NacosException;  import org.junit.jupiter.api.Test;  import java.io.IOException;  import java.util.Properties;  import java.util.concurrent.Executor;  /\*\*  \* <p>Nacos服务注册发现中心，配置动态管理功能测试</p>  \*  \* @author appleyk  \* @version V.0.1.1  \* @blob https://blog.csdn.net/appleyk  \* @github https://github.com/kobeyk  \* @date created on 14:06 2020/9/25  \*/  public class NacosTest {  @Test  public void configTest() throws NacosException, IOException {  String serverAddr = "127.0.0.1";  String dataId = "config.port";  String group = "CONFIG\_GROUP";  Properties properties = new Properties();  properties.put("serverAddr",serverAddr);  ConfigService configService = NacosFactory.createConfigService(properties);  String value = configService.getConfig(dataId, group, 5000);  System.out.println("第一次接收，获取值 = "+value);  configService.addListener(dataId, group, new Listener() {  @Override  public Executor getExecutor() {  return null;  }  @Override  public void receiveConfigInfo(String content) {  // 当Nacos服务中心，对应dataId的配置发生变化时，会触发这个方法  System.out.println("当前时间："+System.currentTimeMillis()+",当前值 = "+content);  }  });  // 使主线程阻塞，便于看到监听的config信息的变化  int n = System.in.read();  }    } |

## 4.3 启动nacos测试方法



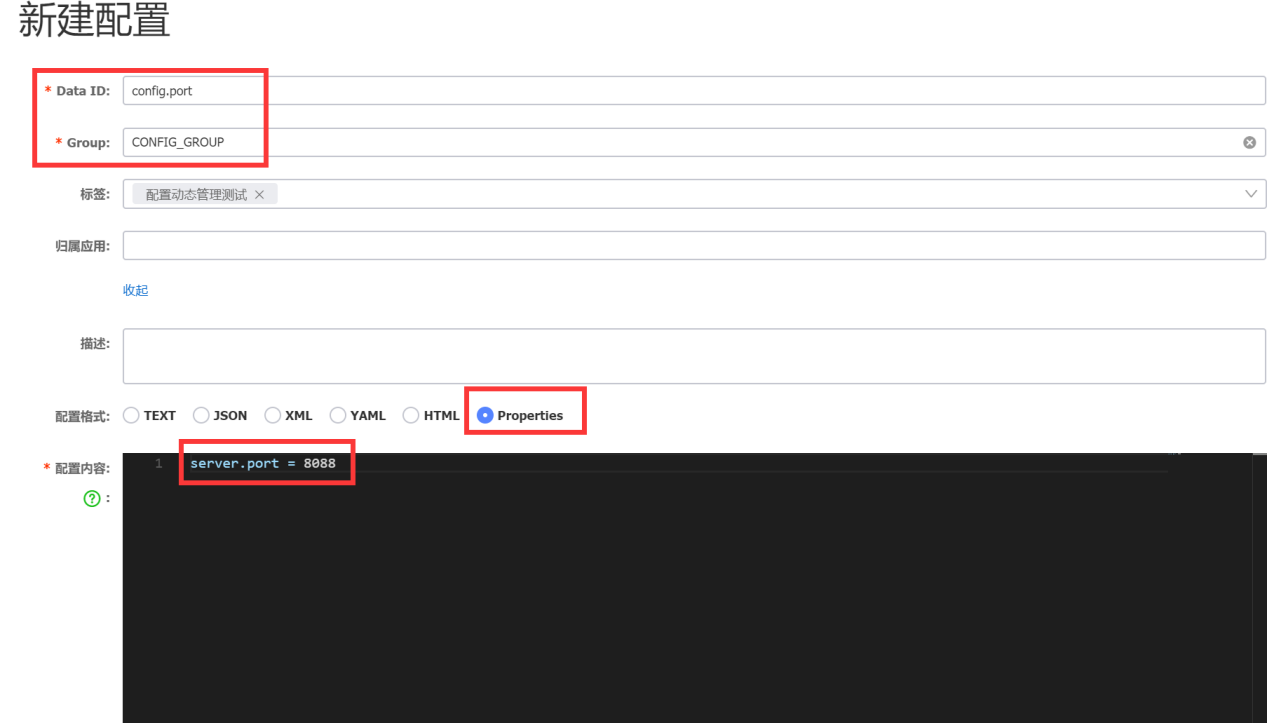
## 4.4 Nacos创建新配置项



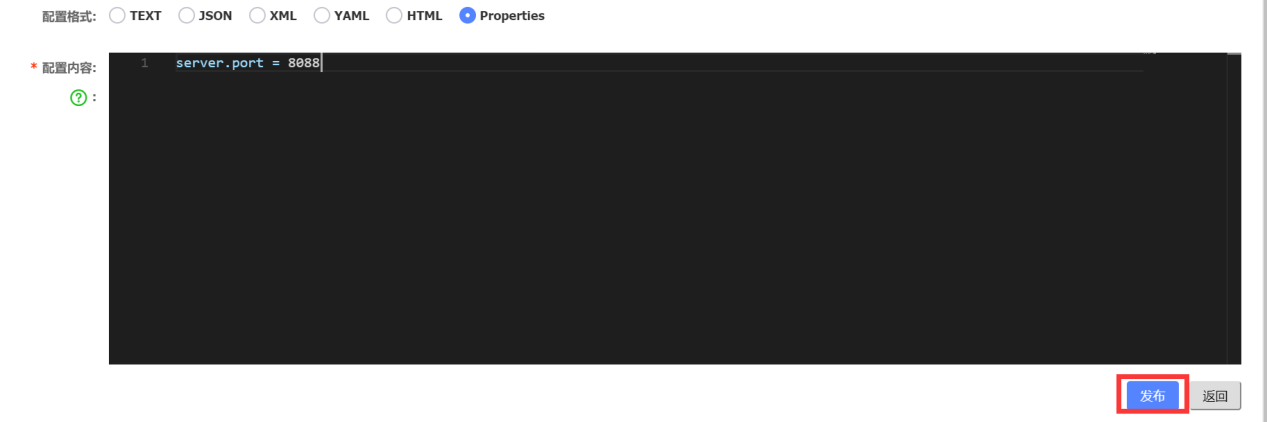
dataId ： config.port

group ：CONFIG\_GROUP

配置内如如下：



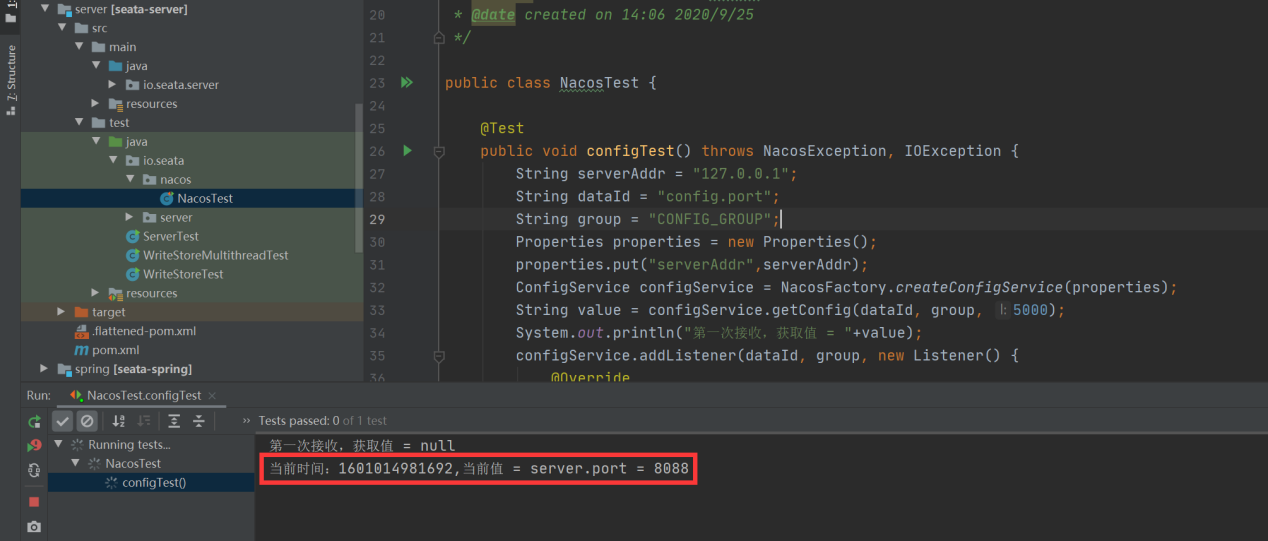
点击发布



点击确定



## 4.5 回到IDEA控制台，查看效果



立竿见影，有没有！！！！！！，而且实现起来很简单！！！