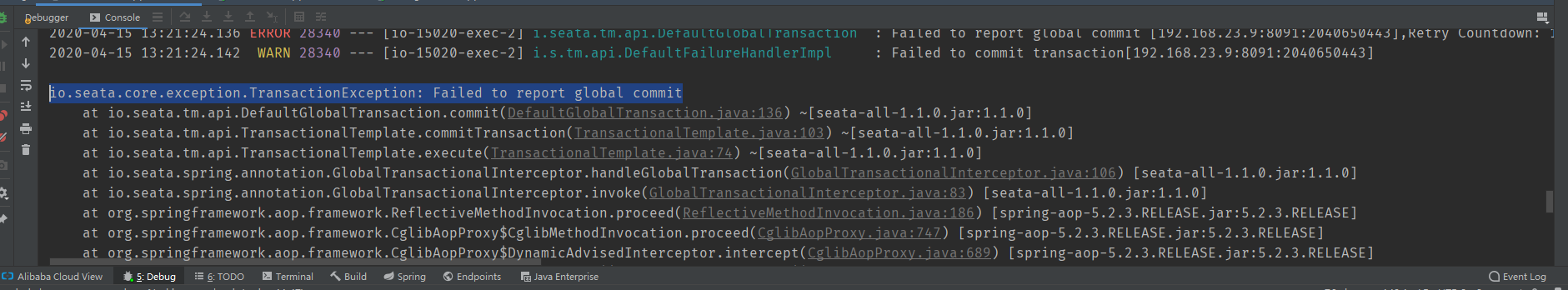
1、TM 向 TC 申请开启一个全局事务，全局事务创建成功并生成一个全局唯一的 XID。

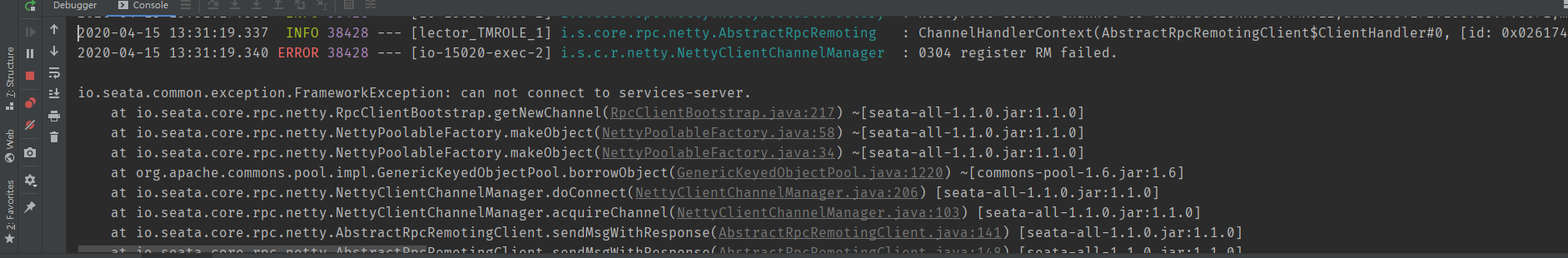
测试情景一：一段时间内所有服务正常运行，开启全局事务时TM宕机

结论:客户端报错io.seata.core.exception.TransactionException: Failed to report global commit,全局事务未开启,程序终止,各数据库数据没有改变



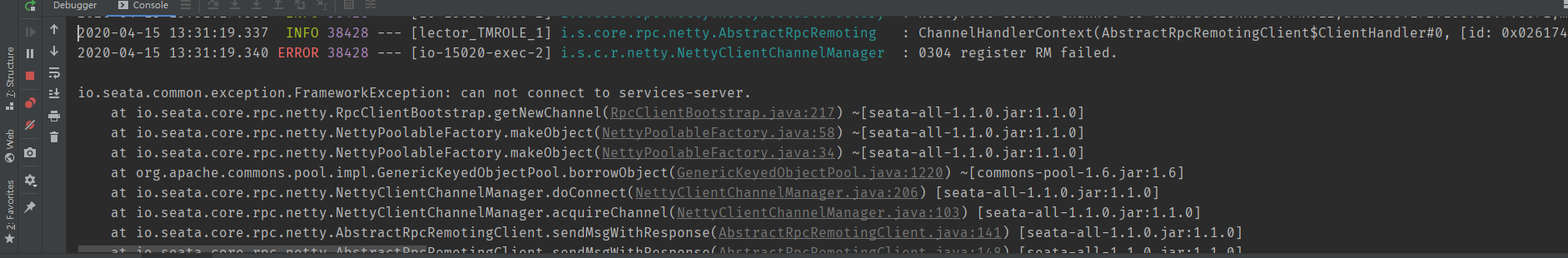
测试情景二：一段时间内所有服务正常运行，当TM 向TC 申请XID 时，其中一个TC宕机：

结论:客户端报错io.seata.common.exception.FrameworkException: can not connect to services-server. 全局事务未受到影响，另外一个TC 负责发放XID，全局事务提交，业务数据正常改变



测试情景三： 一段时间内所有服务正常运行，当TM 向TC 申请XID 时，所有TC宕机

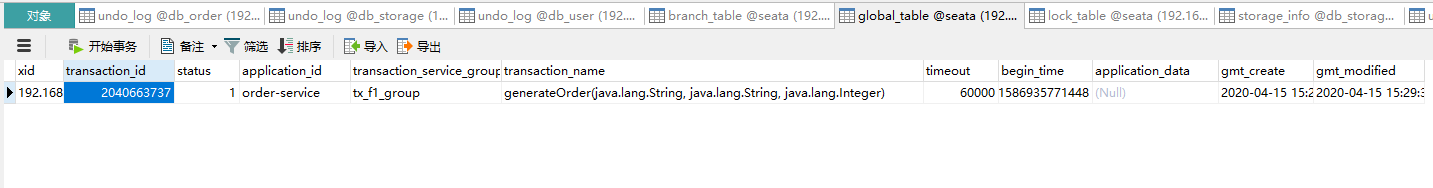
结论:客户端报错io.seata.common.exception.FrameworkException: can not connect to services-server. ,程序终止,各数据库数据没有改变



2、RM 向 TC 注册分支事务，将其纳入 XID 对应全局事务的管辖。

测试情景四：一段时间内所有服务正常运行，在全局事务已开启、RM分支事务未注册到TC的情况下，TM宕机

结论：程序终结，业务数据没有变化，global\_table 表中出现一条数据



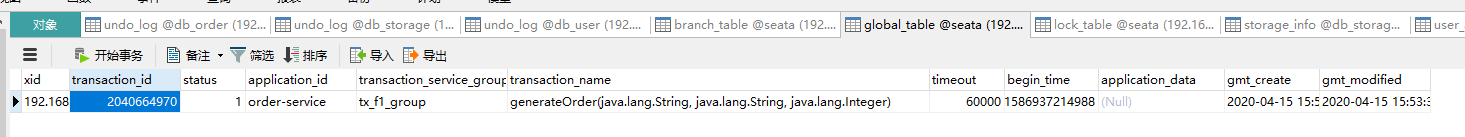
TM 重启后数据被清除

测试情景五：一段时间内所有服务正常运行，在全局事务已开启、RM分支事务未注册到TC的情况下，其中一个TC宕机

结论：剩余的TC完成RM分支事务的注册，并提交全局事务

测试情景六：一段时间内所有服务正常运行，在全局事务已开启、RM分支事务未注册到TC的情况下，所有TC宕机

结论：客户端报错io.seata.common.exception.FrameworkException: can not connect to services-server. ,程序终止，global\_table 表中出现一条数据



任意一个TC重启后数据被清除

3、RM 向 TC 注册分支事务，将其纳入 XID 对应全局事务的管辖。

测试情景七：一段时间内所有服务正常运行，在全局事务已开启、分支事务A注册到TC并提交的情况下，RM宕机

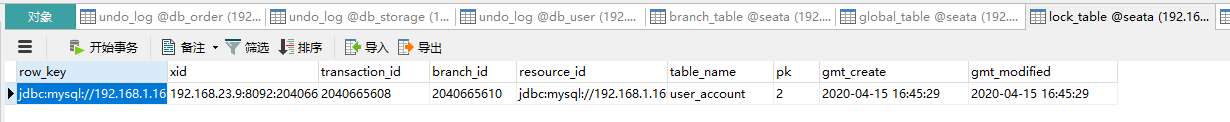
结论：全局事务和分支事务全部回滚，业务数据没有改变

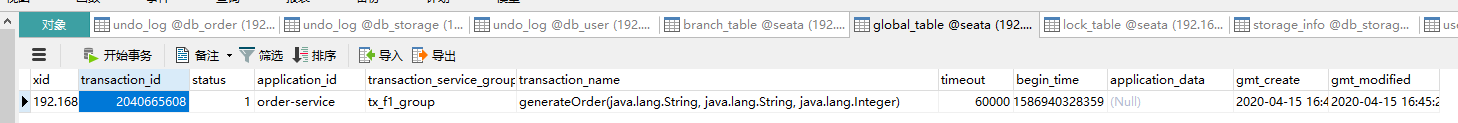
测试情景八：一段时间内所有服务正常运行，在全局事务已开启、分支事务A注册到TC并提交的情况下，其中一个TC宕机

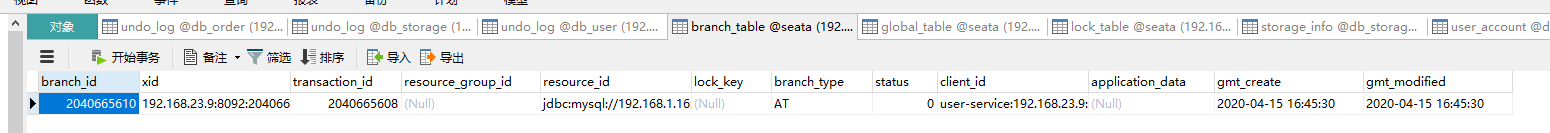
结论：全局事务和分支事务全部提交。

测试情景九：一段时间内所有服务正常运行，在全局事务已开启、分支事务A注册到TC并提交的情况下，所有TC宕机

结论：业务数据库出现脏数据的情况，Seata相关数据库表记录了回滚信息。



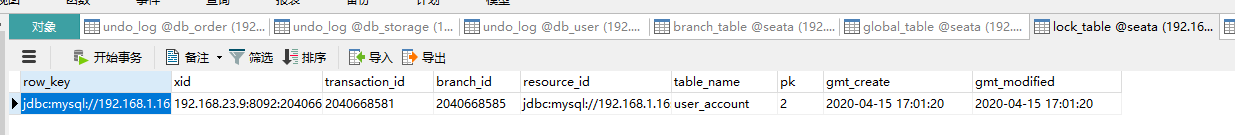
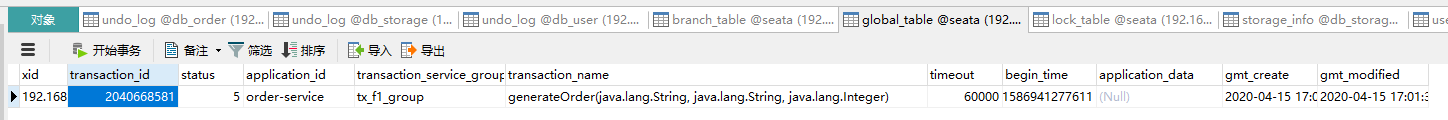
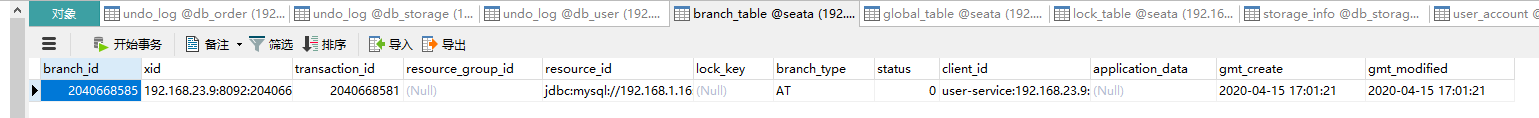
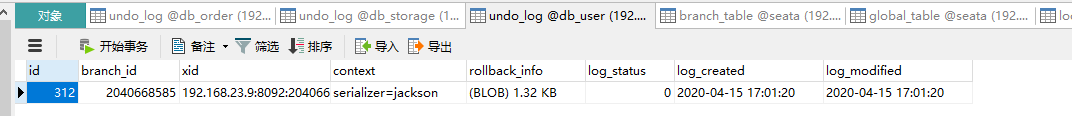




启动任意一个TC 节点后数据回滚成功

测试情景十：一段时间内所有服务正常运行，在全局事务已开启、分支事务A注册到TC并提交的情况下，TM报错，在回滚过程中分支事务A所在的服务宕机。

结论：业务数据库出现脏数据，Seata相关数据库表记录了回滚信息。

Seata服务端以一秒一次的频率持续请求连接RM，服务重启后数据回滚

4、TM 向 TC 发起针对 XID 的全局提交或回滚决议。

测试情景十一：段时间内所有服务正常运行，在全局事务已开启、分支事务注册到TC并提交的情况下，TM 宕机：

结论：全局提交。

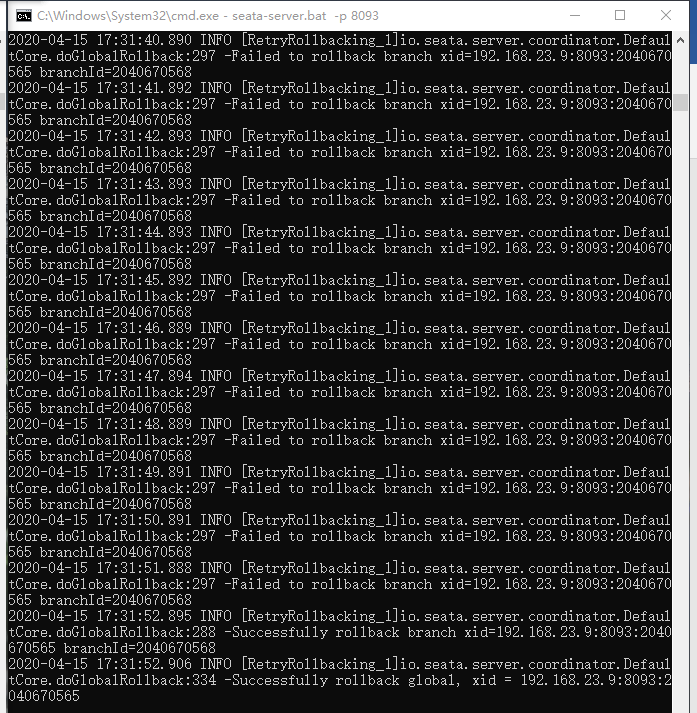
5、TC 调度 XID 下管辖的全部分支事务完成提交或回滚请求。

步骤外情景：

测试情景十二：回滚时当前数据与后置镜像数据不同：

结论：官方文档说 有详细的文档说明执行策略，没有找到

默认情况下，Server 不断显示回滚失败



手动处理数据后回滚成功。

发生脏数据时，其他程序也可以修改该数据。

# 七、测试

对Seata的服务端修改完成后，测试Seata AT模式是否在分布式情境下保证了数据的一致性。

单点模式下：

情景1、没有异常抛出的时候

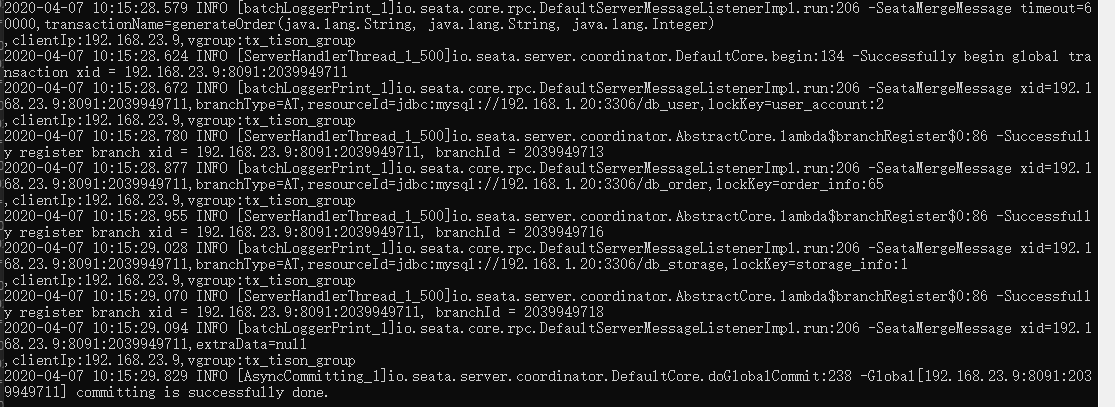
所有库中的 undo 表各出现一条数据，全局提交后被删除

Global表出一条数据，全局提交后删除

Branch 表出现三条数据，本地提交后删除

Lock表出现三条数据，提交后删除

服务端输出



输出全局事务ID、本地事务ID，最后输出 committing is successfully done.表示事务提交成功

情景2、抛出异常，回滚成功

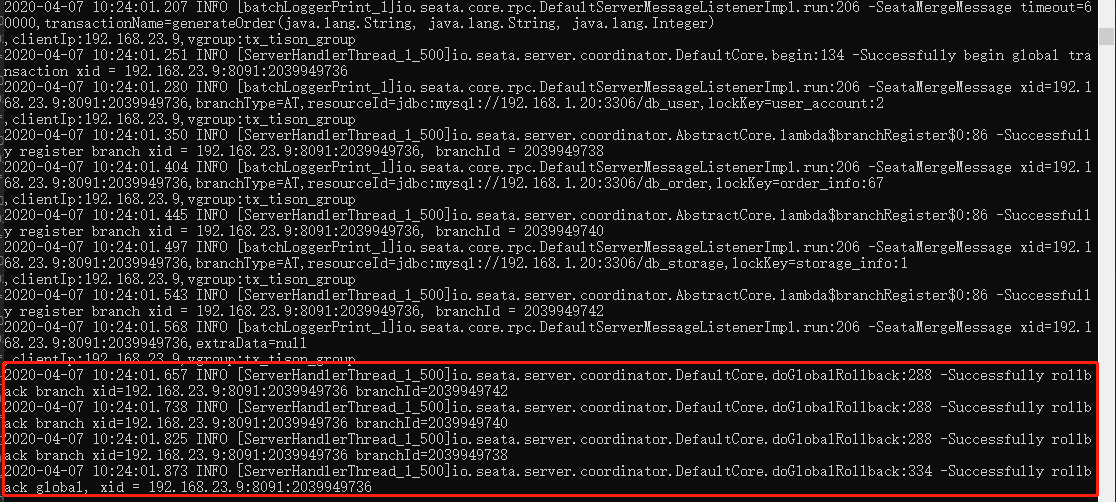
所有库中的 undo 表各出现一条数据，全局提交后被删除

Global表出一条数据，全局提交后删除

Branch 表出现三条数据，本地提交后删除

Lock表出现三条记录，随后删除

服务端输出



本地服务回滚成功，随后全局提交



情景3、不抛出异常，成功提交。

所有库中的 undo 表各出现一条数据，全局提交后被删除

Global表出一条数据，全局提交后删除

Branch 表出现三条数据，本地提交后删除

Lock表出现三条记录，随后删除

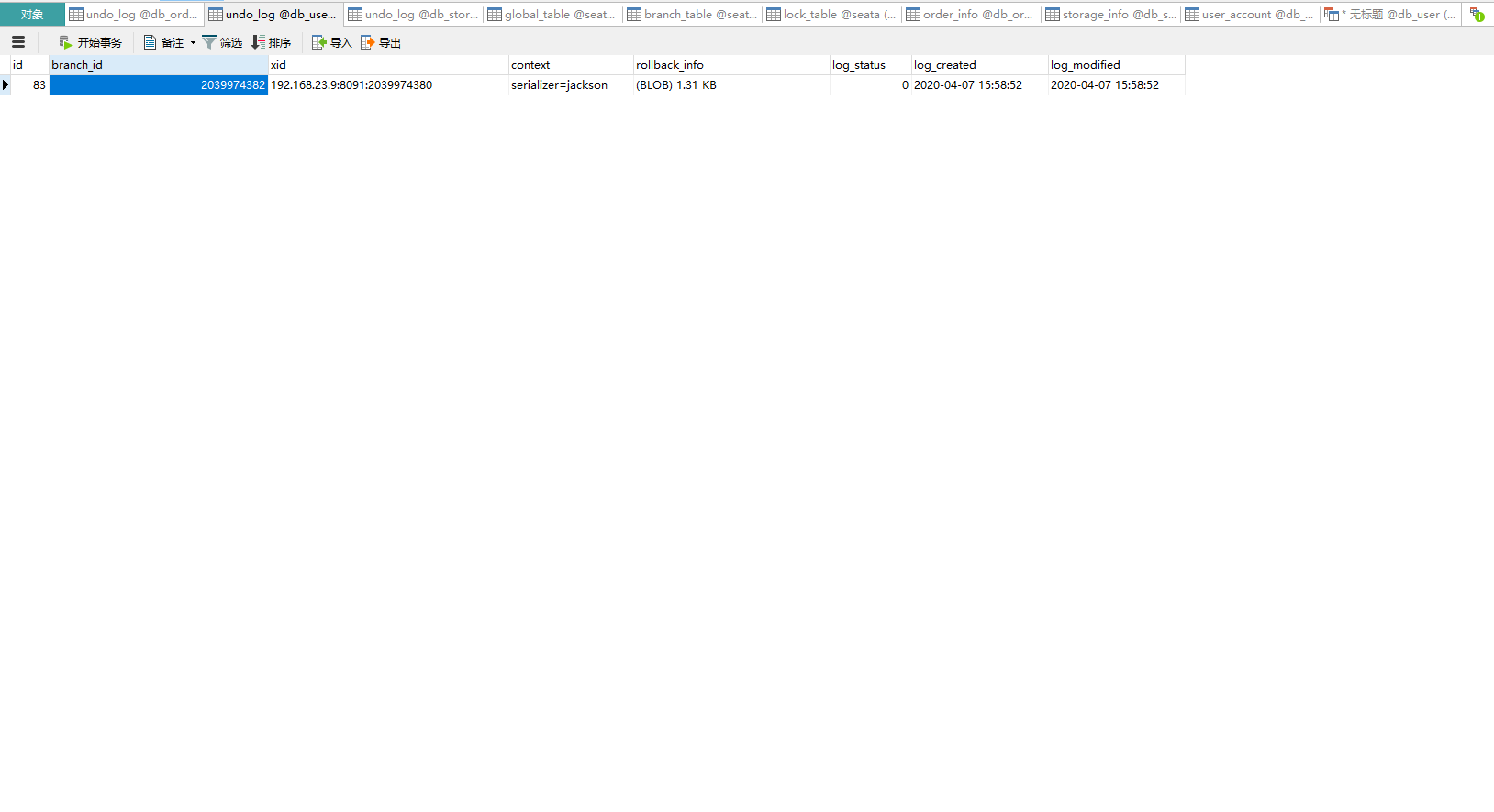
当程序出现异常，如果对异常进行了处理（try、catch操作），全局事务不会回滚，成功提交。

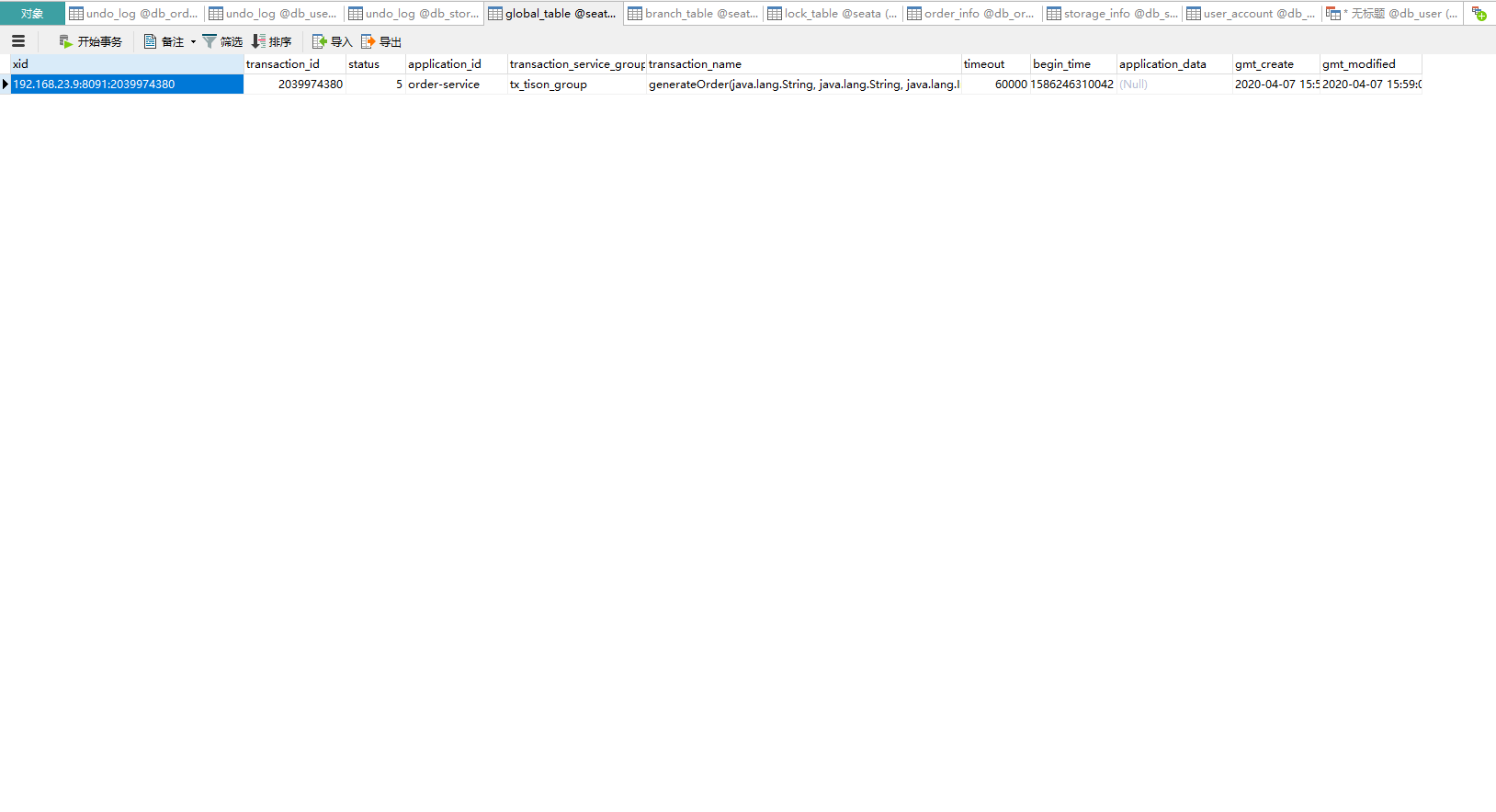


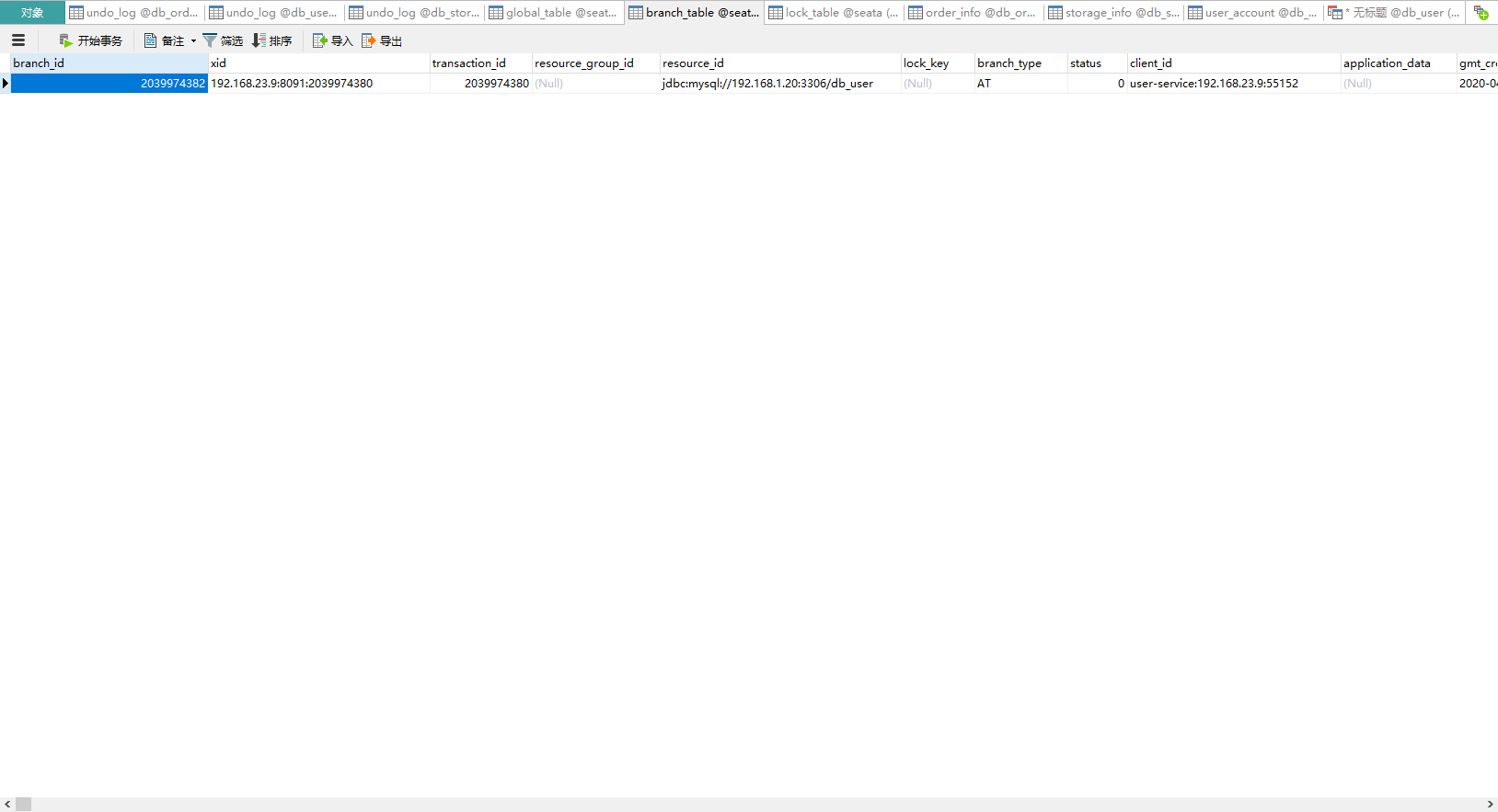
情景4、回滚时业务服务宕机

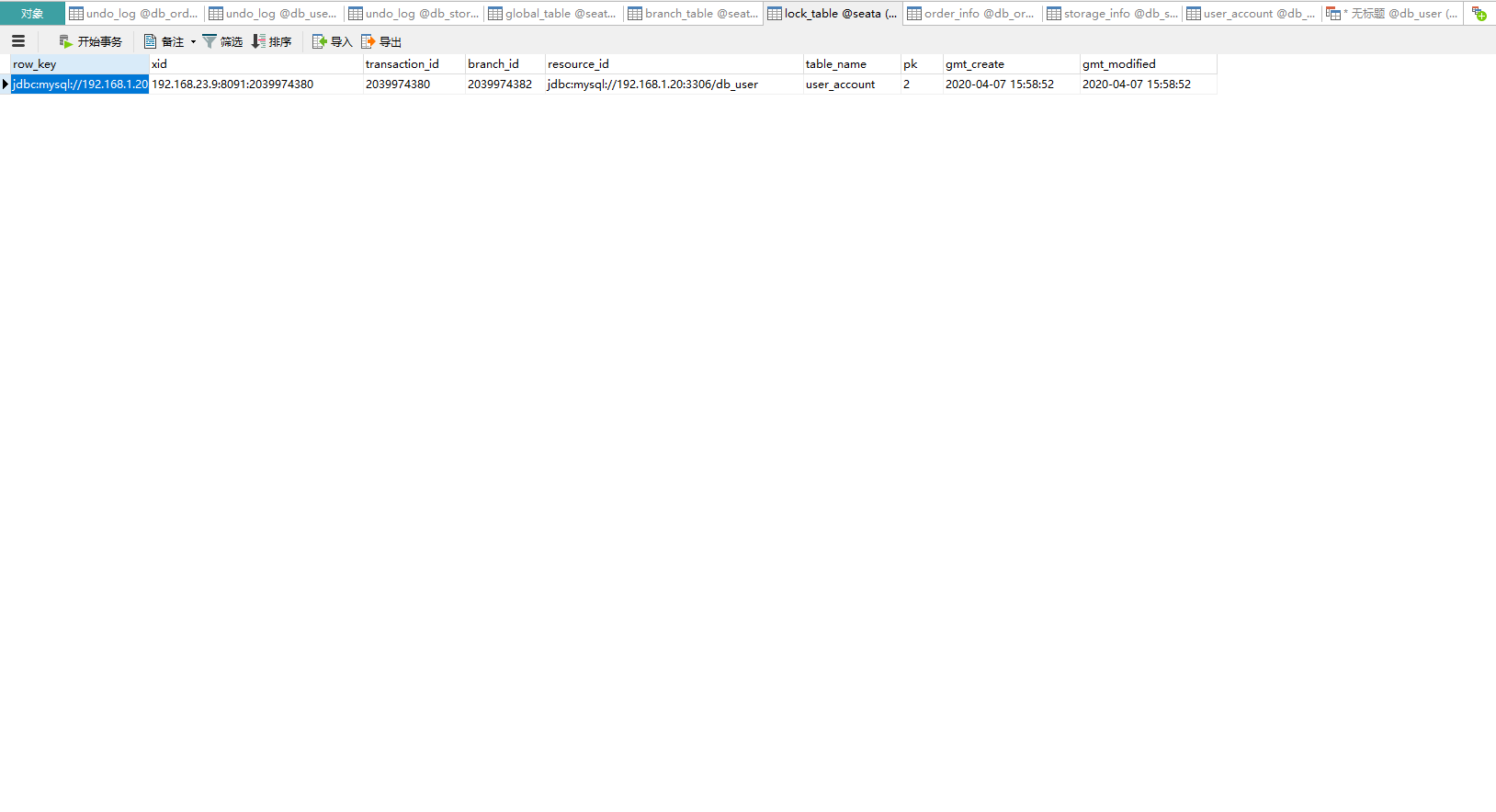
回滚时出现宕机现象，导致事务回滚失败，

对应业务数据库undo 表、seata数据库global\_table、branch\_table、lock\_table中出现数据









执行sql

select CAST(rollback\_info AS CHAR(10000) CHARACTER SET utf8) from undo\_log

可看到 rollback\_info 字段中的数据即为 数据修改前后的记录

{

"@class":"io.seata.rm.datasource.undo.BranchUndoLog",

"xid":"192.168.23.9:8091:2039974380",

"branchId":2039974382,

"sqlUndoLogs":[

"java.util.ArrayList",

[

{

"@class":"io.seata.rm.datasource.undo.SQLUndoLog",

"sqlType":"UPDATE",

"tableName":"user\_account",

"beforeImage":{

"@class":"io.seata.rm.datasource.sql.struct.TableRecords",

"tableName":"user\_account",

"rows":[

"java.util.ArrayList",

[

{

"@class":"io.seata.rm.datasource.sql.struct.Row",

"fields":[

"java.util.ArrayList",

[

{

"@class":"io.seata.rm.datasource.sql.struct.Field",

"name":"id",

"keyType":"PRIMARY\_KEY",

"type":4,

"value":2

},

{

"@class":"io.seata.rm.datasource.sql.struct.Field",

"name":"user\_id",

"keyType":"NULL",

"type":12,

"value":"1002"

},

{

"@class":"io.seata.rm.datasource.sql.struct.Field",

"name":"money",

"keyType":"NULL",

"type":4,

"value":97500

}

]

]

}

]

]

},

"afterImage":{

"@class":"io.seata.rm.datasource.sql.struct.TableRecords",

"tableName":"user\_account",

"rows":[

"java.util.ArrayList",

[

{

"@class":"io.seata.rm.datasource.sql.struct.Row",

"fields":[

"java.util.ArrayList",

[

{

"@class":"io.seata.rm.datasource.sql.struct.Field",

"name":"id",

"keyType":"PRIMARY\_KEY",

"type":4,

"value":2

},

{

"@class":"io.seata.rm.datasource.sql.struct.Field",

"name":"user\_id",

"keyType":"NULL",

"type":12,

"value":"1002"

},

{

"@class":"io.seata.rm.datasource.sql.struct.Field",

"name":"money",

"keyType":"NULL",

"type":4,

"value":96500

}

]

]

}

]

]

}

}

]

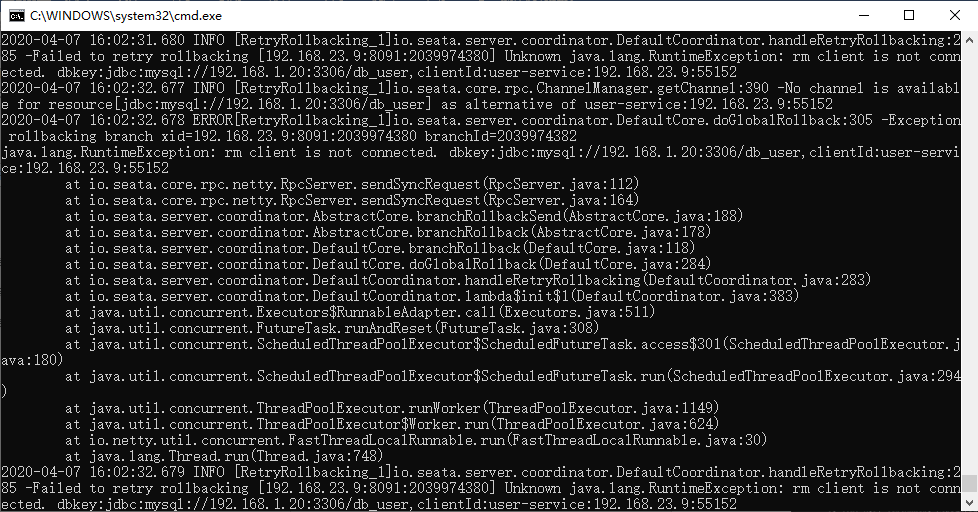
]

}

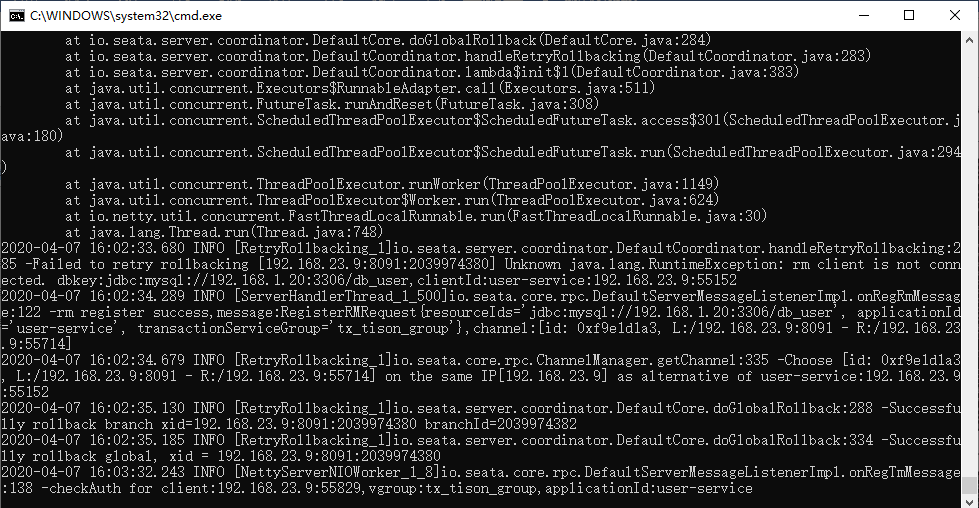
其中，beforeImage代表修改之前的数据快照，afterImage修改之后的数据快照

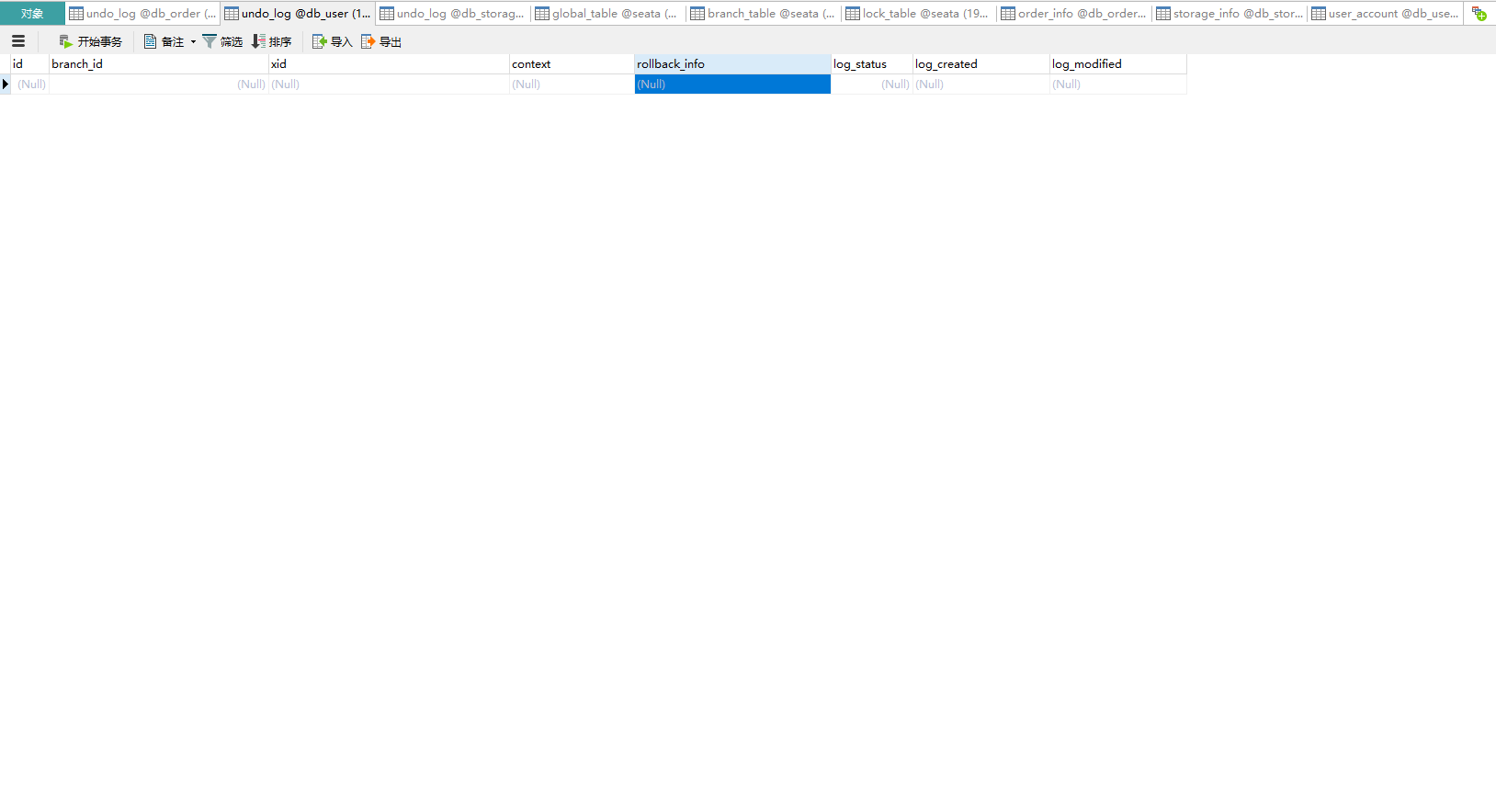
同时，Seata 会以大约一秒一次的频率不断尝试恢复进行回滚，重启后恢复。这

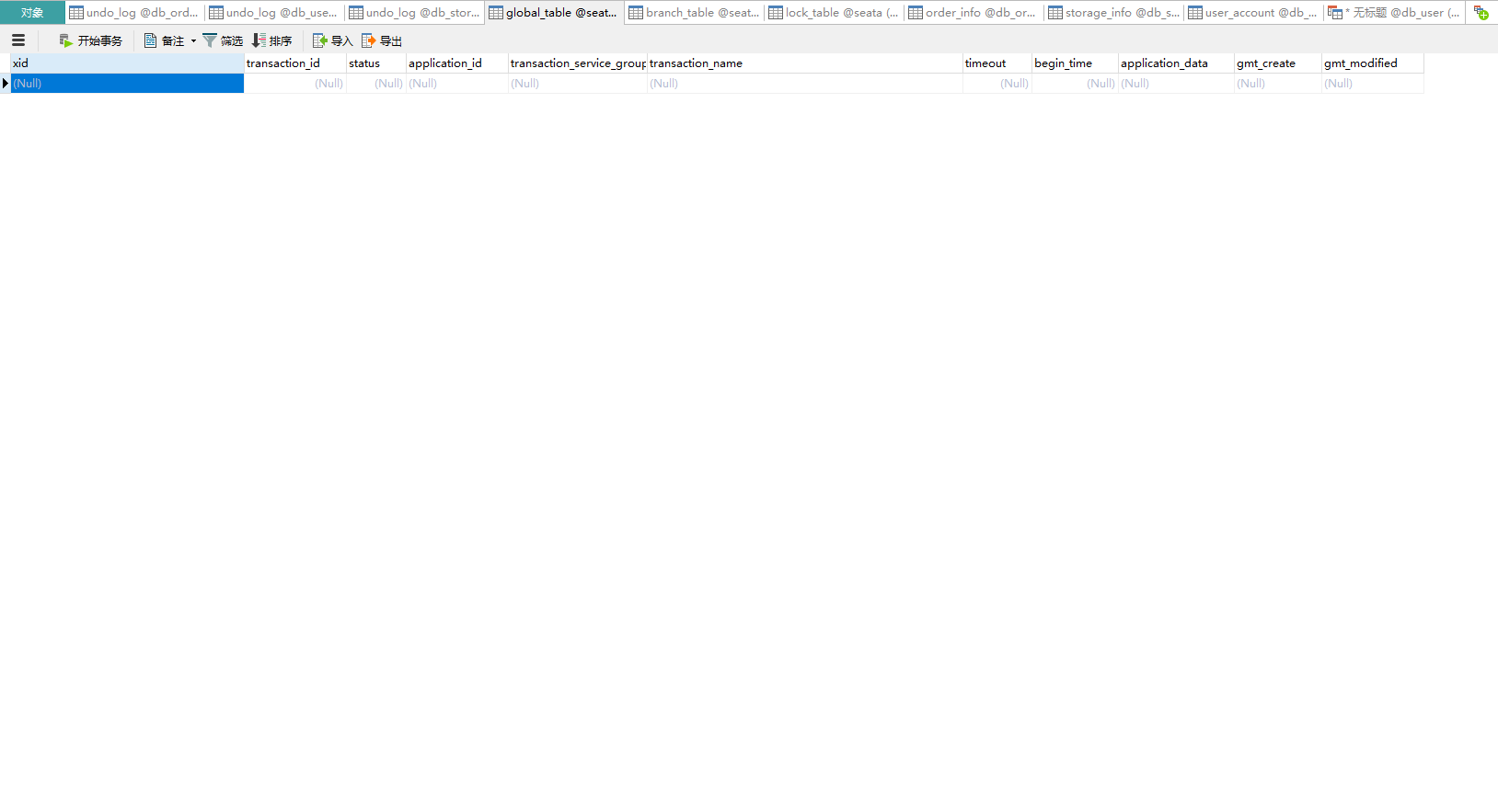
服务端输出如下

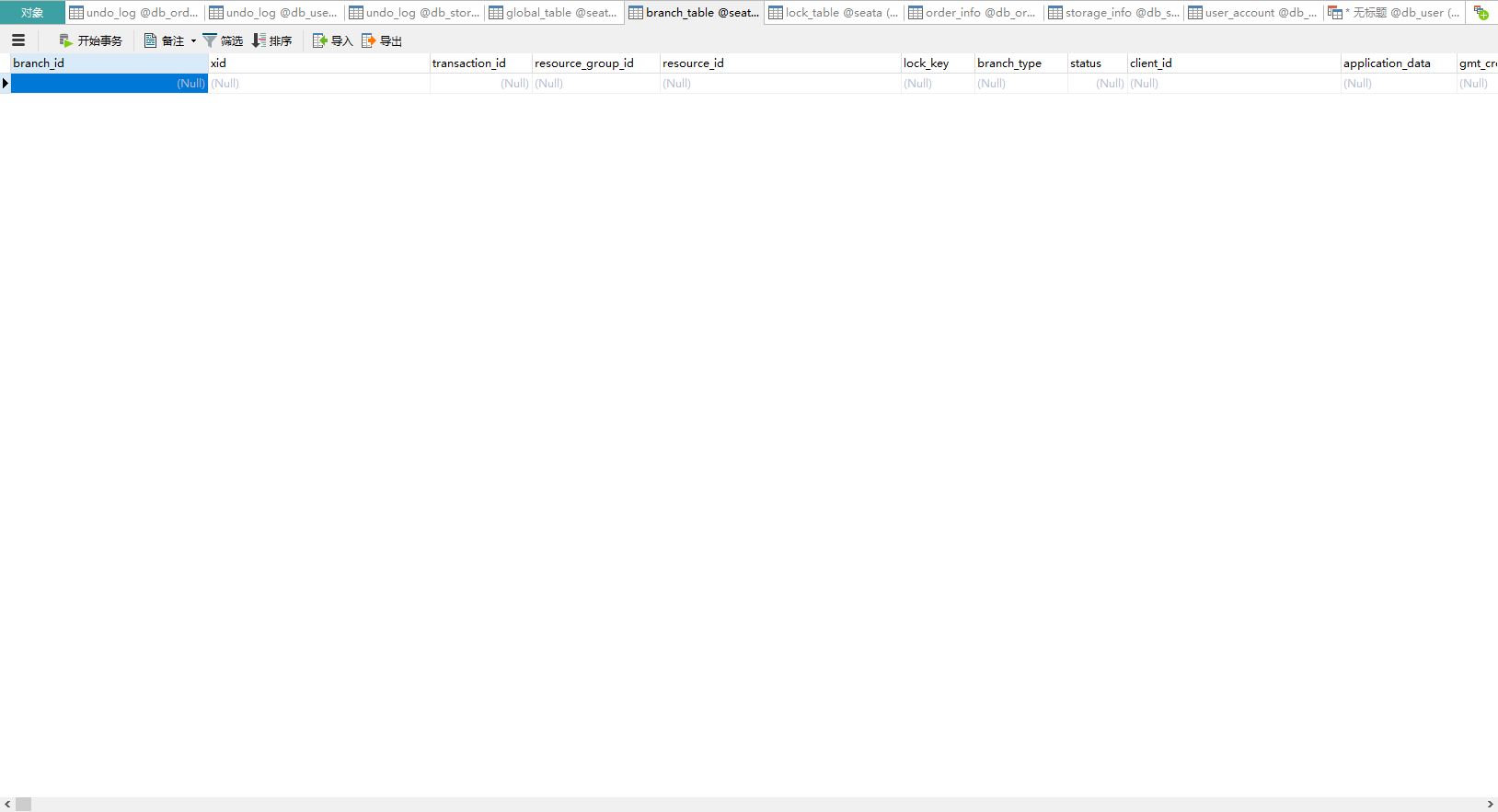


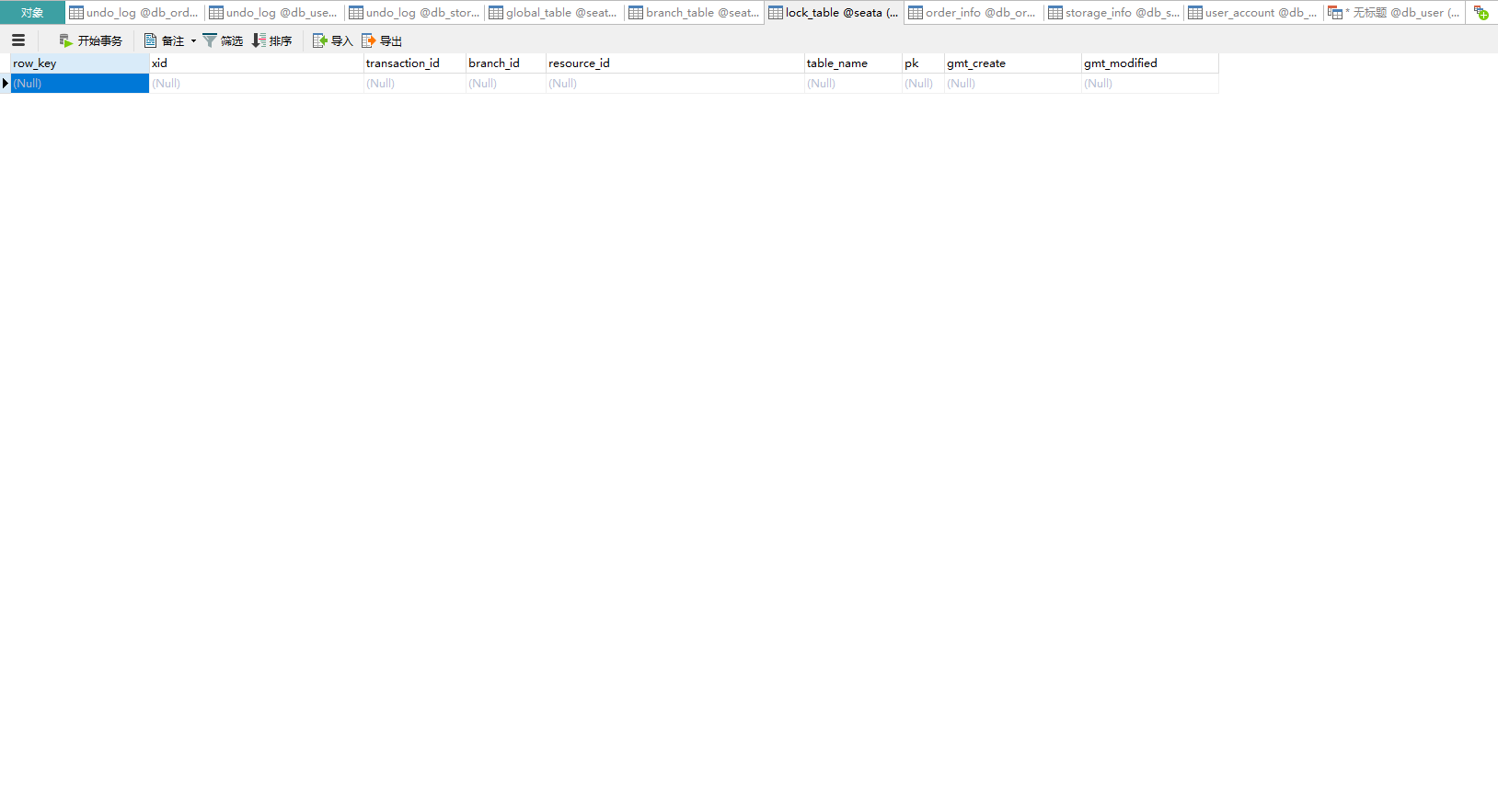
重启服务后，seata自动执行回滚，undo数据库中记录的信息也被Seata删除。





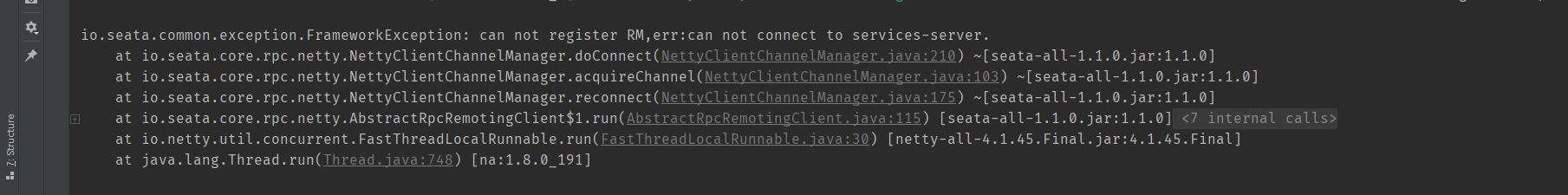




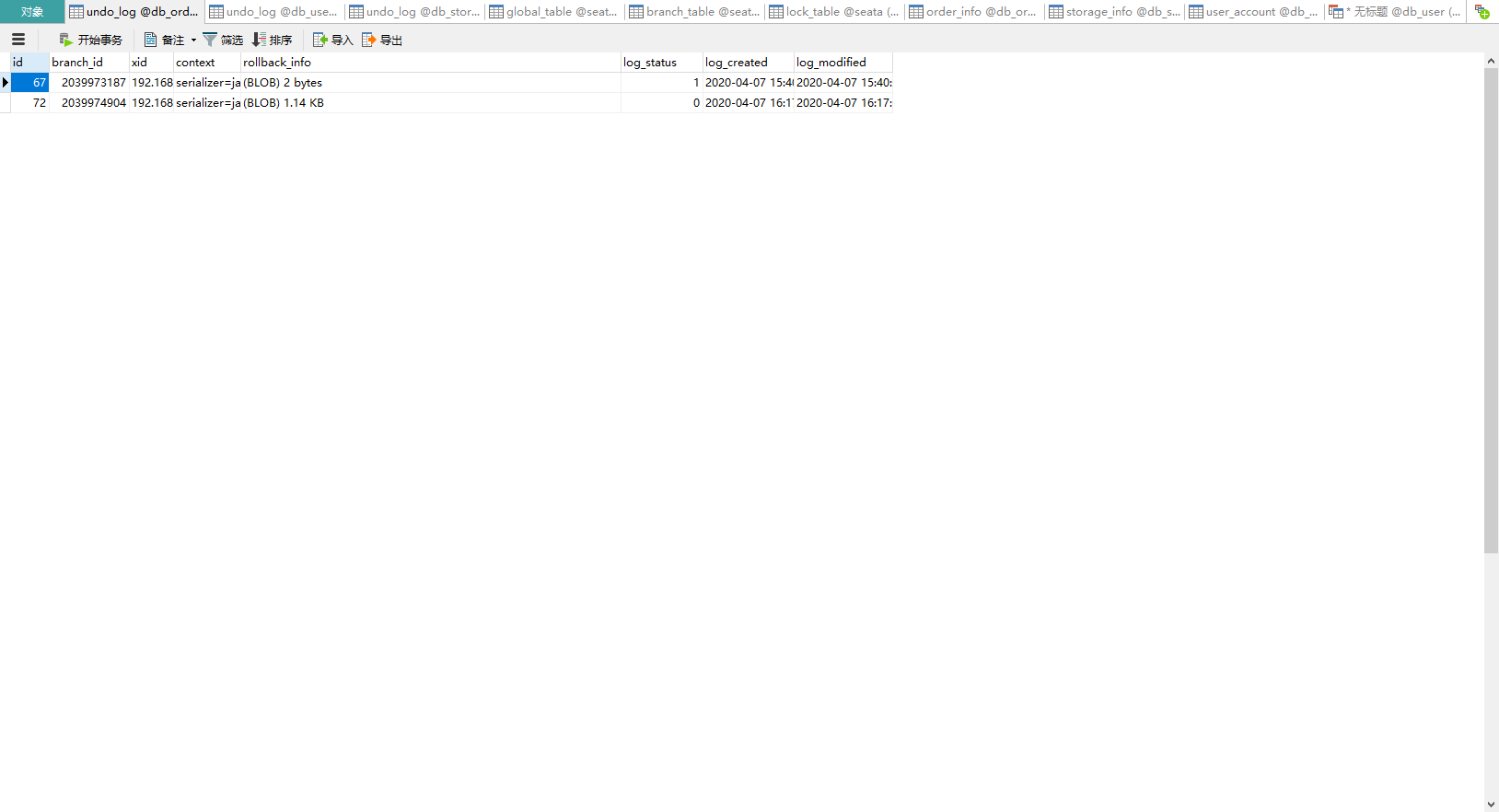


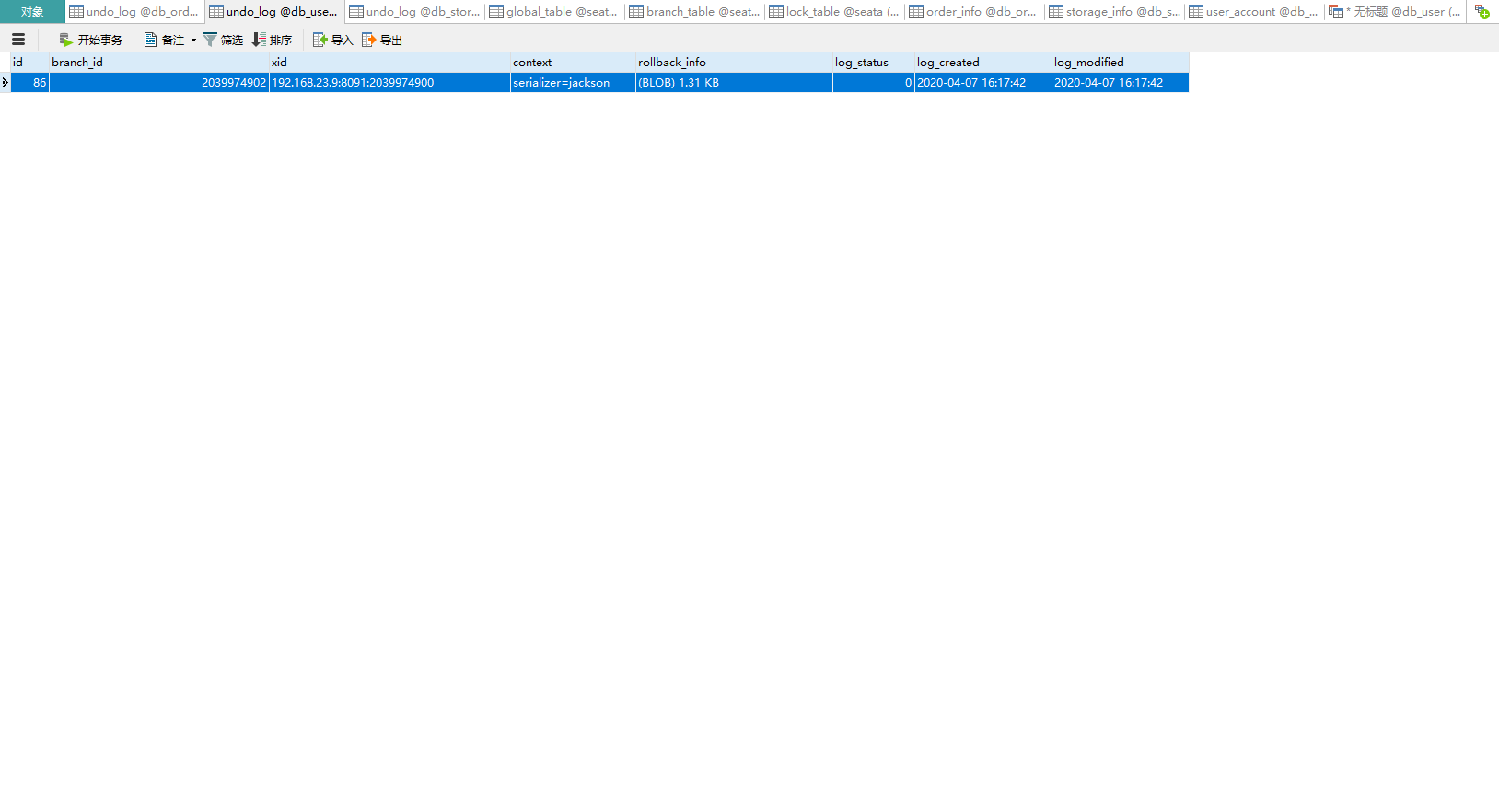
情景5、回滚时Seata服务端宕机

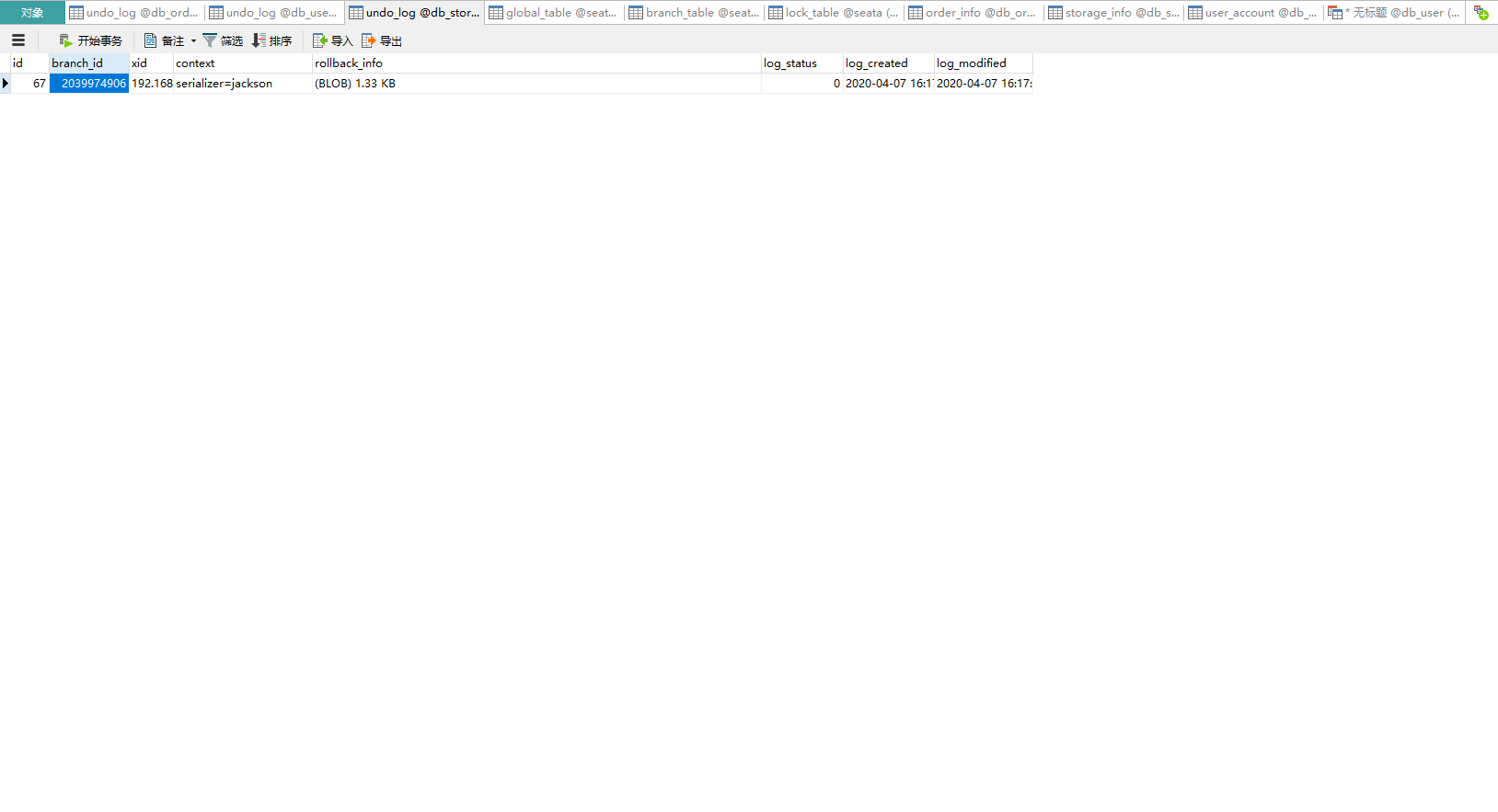
事务需要回滚时，Seata 服务端宕机，各服务均抛出无法连接server 的异常

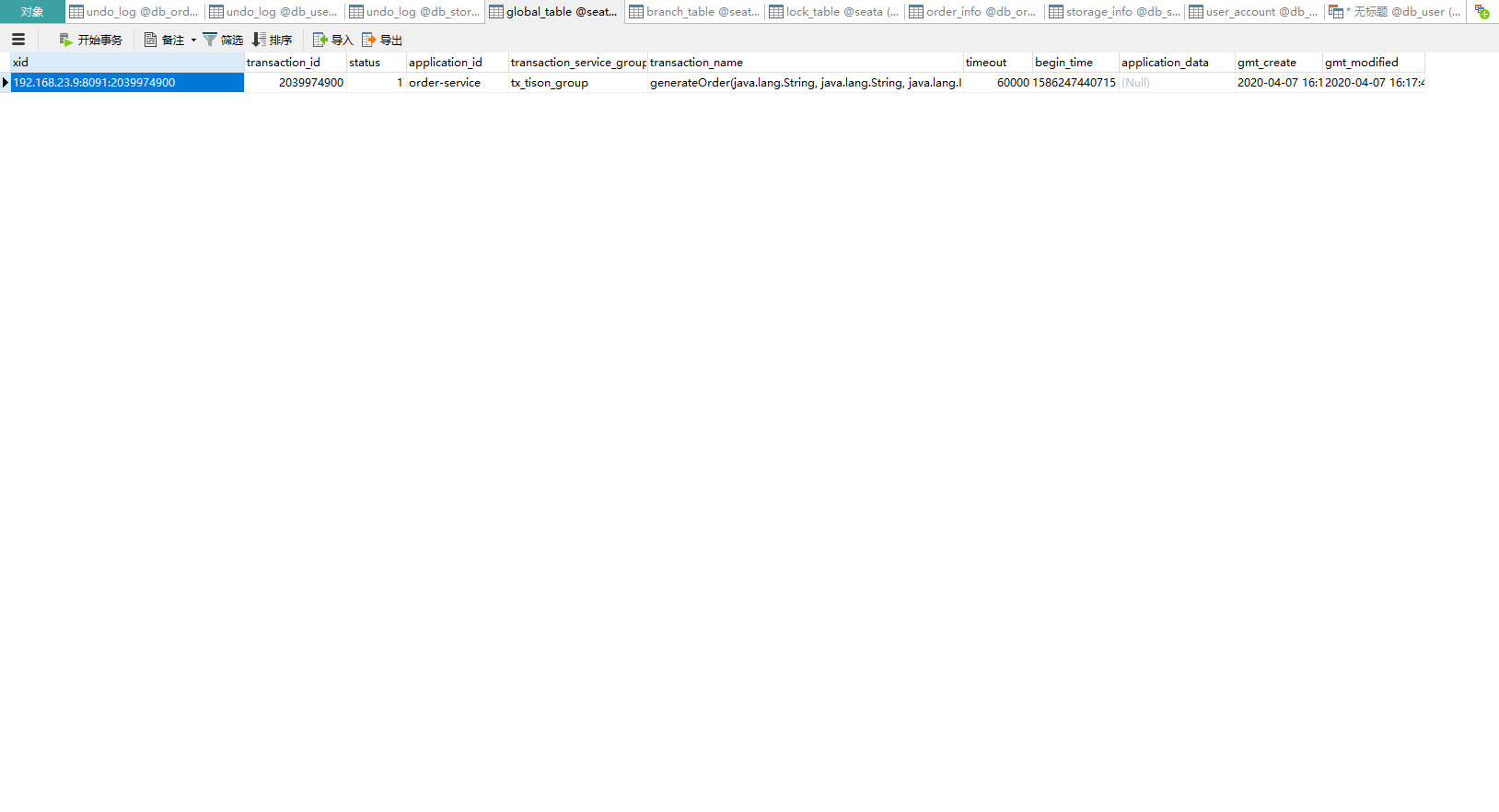


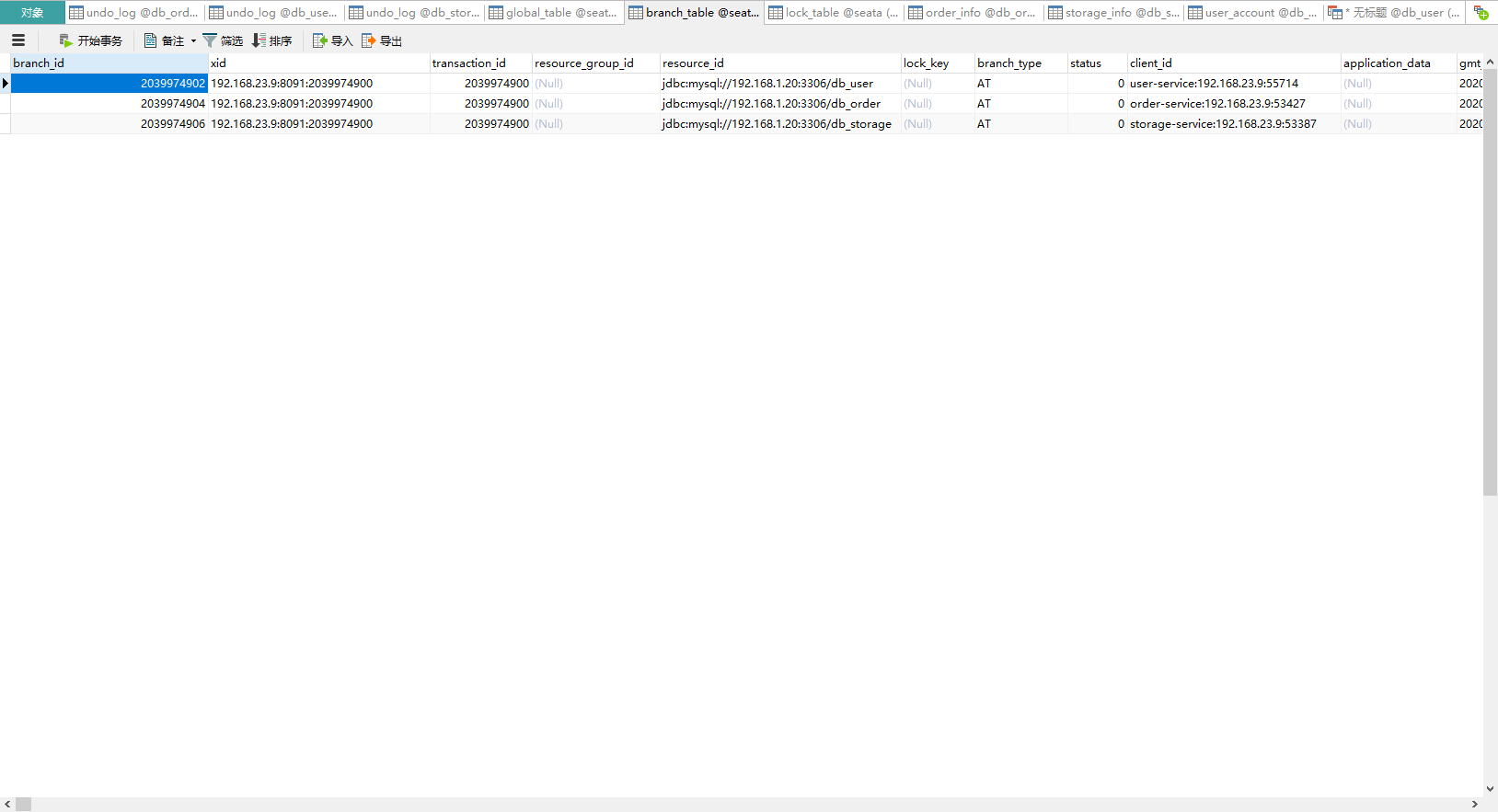
各业务数据库、seata数据库三张表出现数据

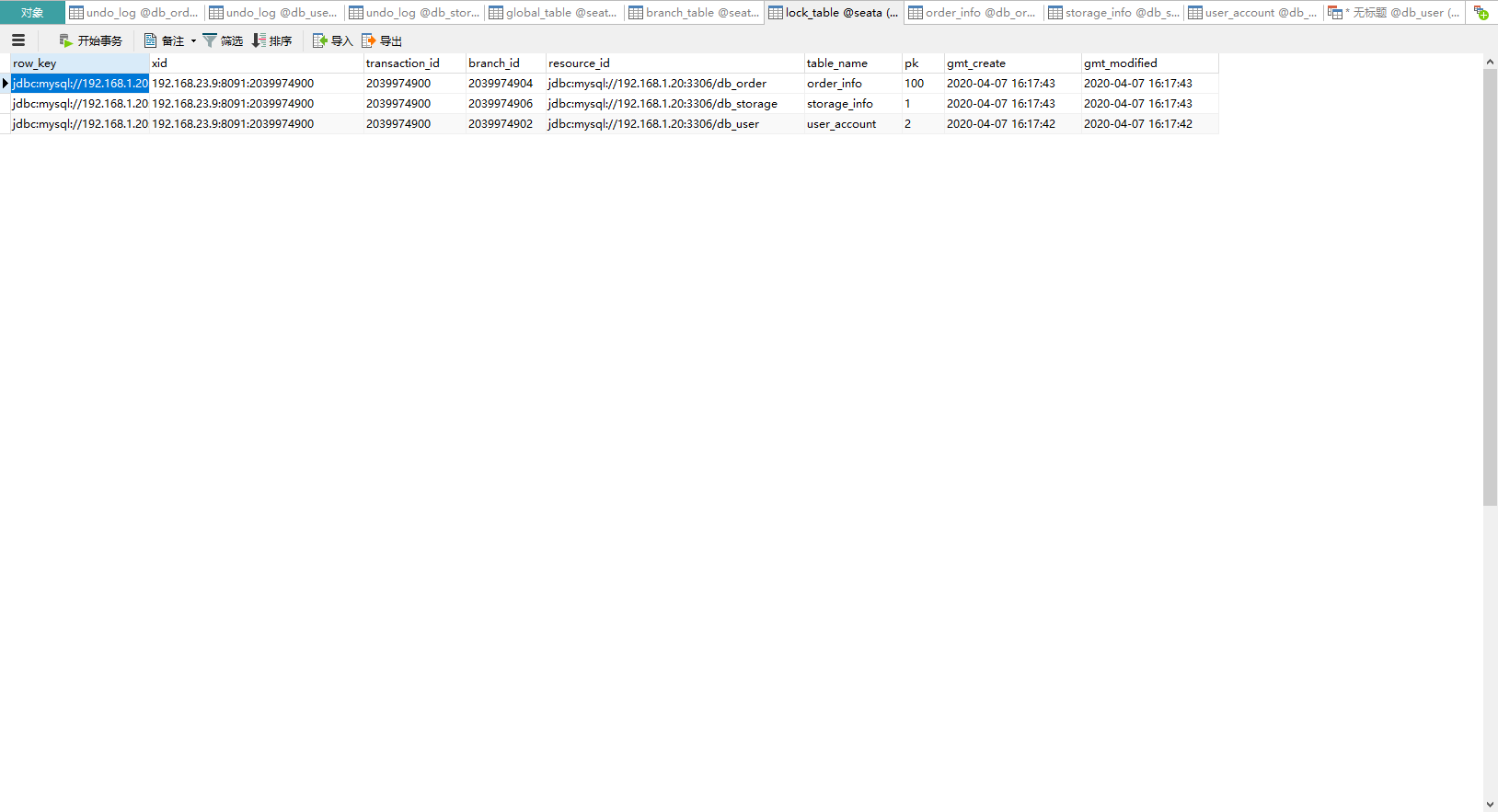










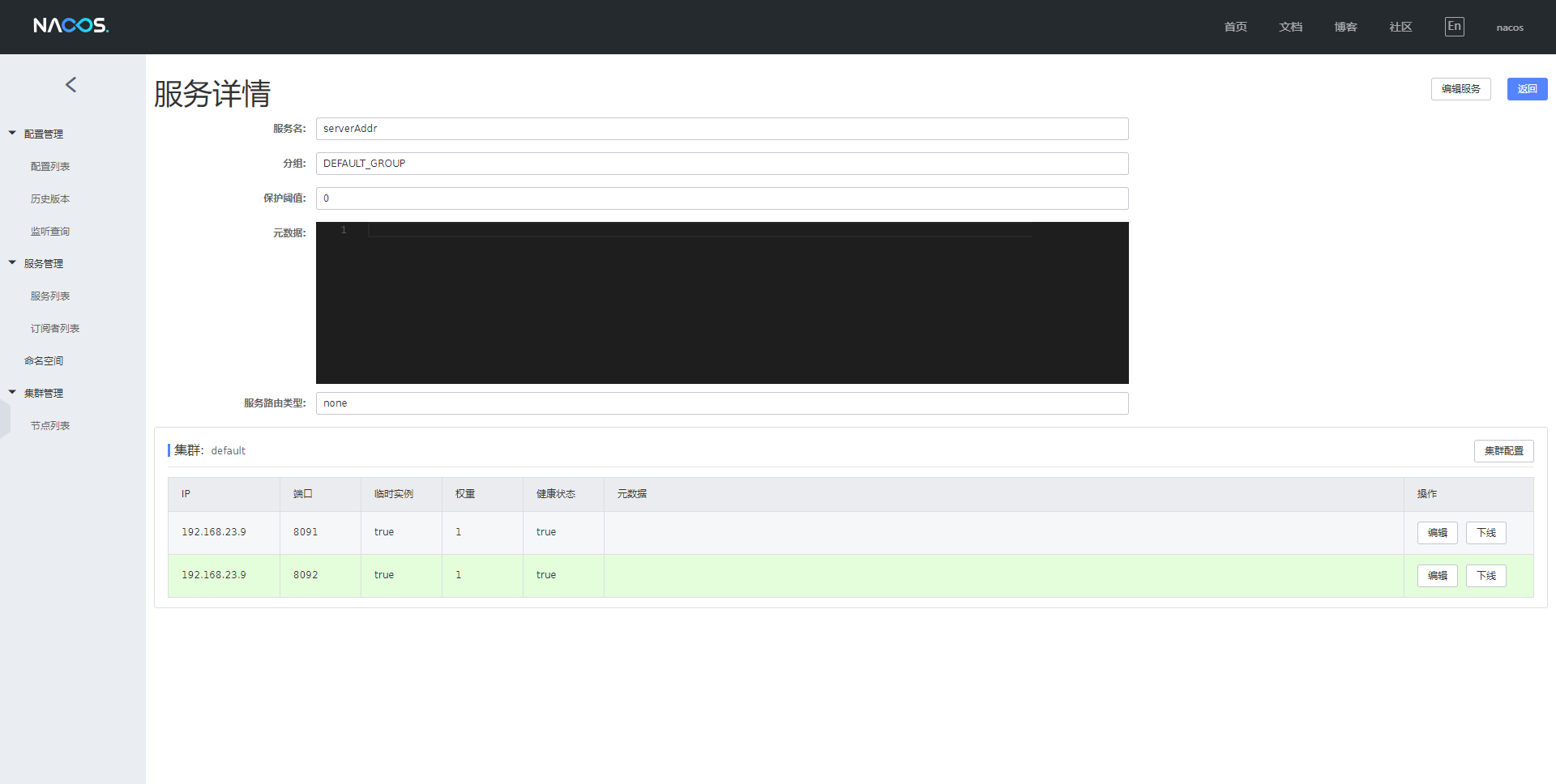


重启Seata服务端后，服务端自动进行回滚。

集群模式下

Seata的高可用并没有开源，但是Seata TC 自身支持多点部署

1、启动多个Seata TC 并注册到Nacos



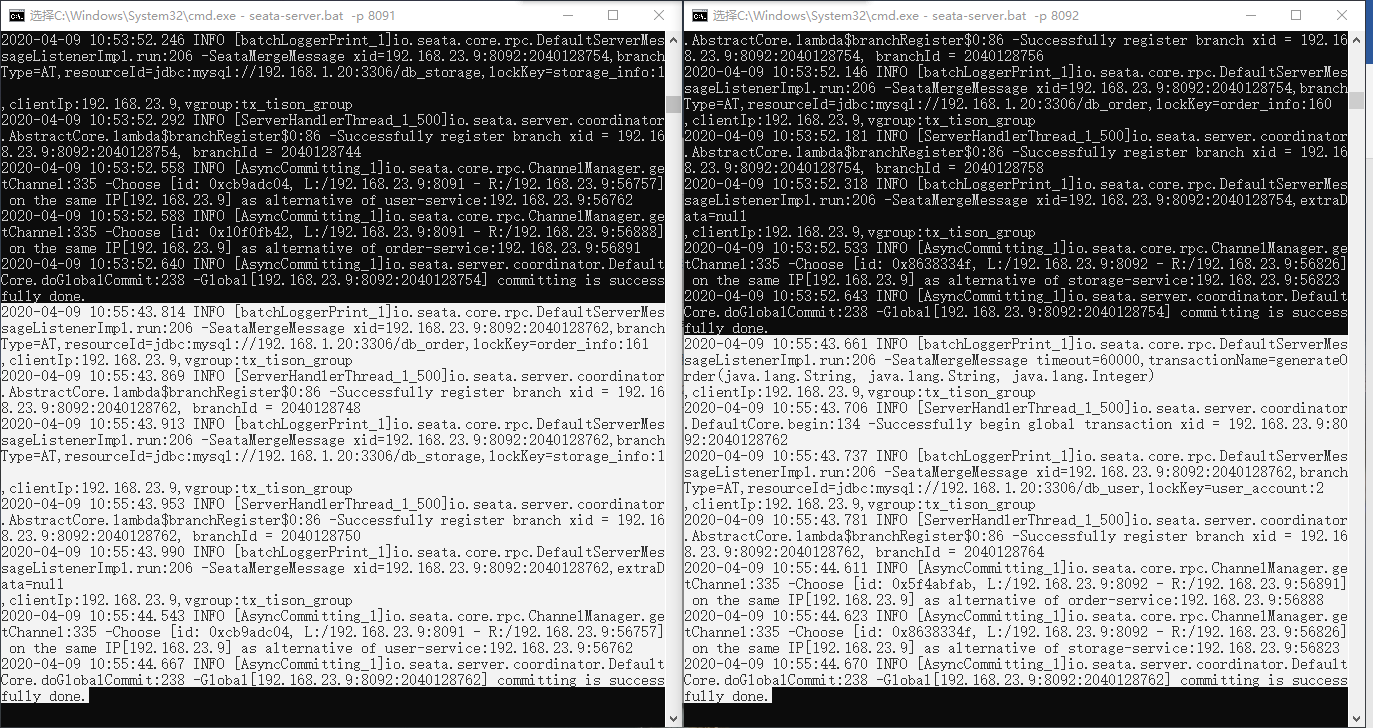
这里本地注册两个TC

2、请求接口查看结果

高可用测试结果

情景1、成功提交

数据库表现与单点模式一致

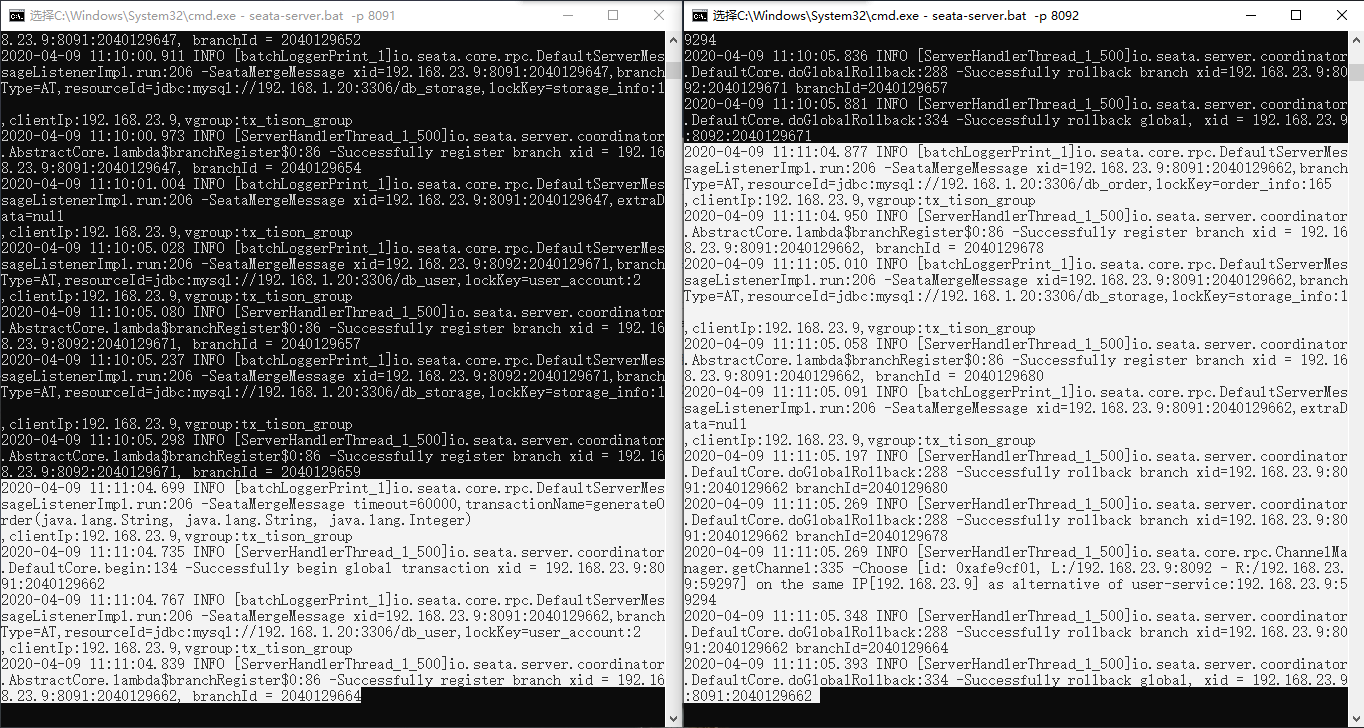


可以看到两个服务端同时对一个全局事务进行操作

最终成功提交

情景2、成功回滚

数据库表现与单点模式一致



可以看到两个服务端同时对一个全局事务执行回滚

情景3、TM 报错，回滚时业务RM停止，导致回滚失败

与单点模式表现一致

情景4、回滚时负责开启全局事务的TC宕机

事务回滚成功



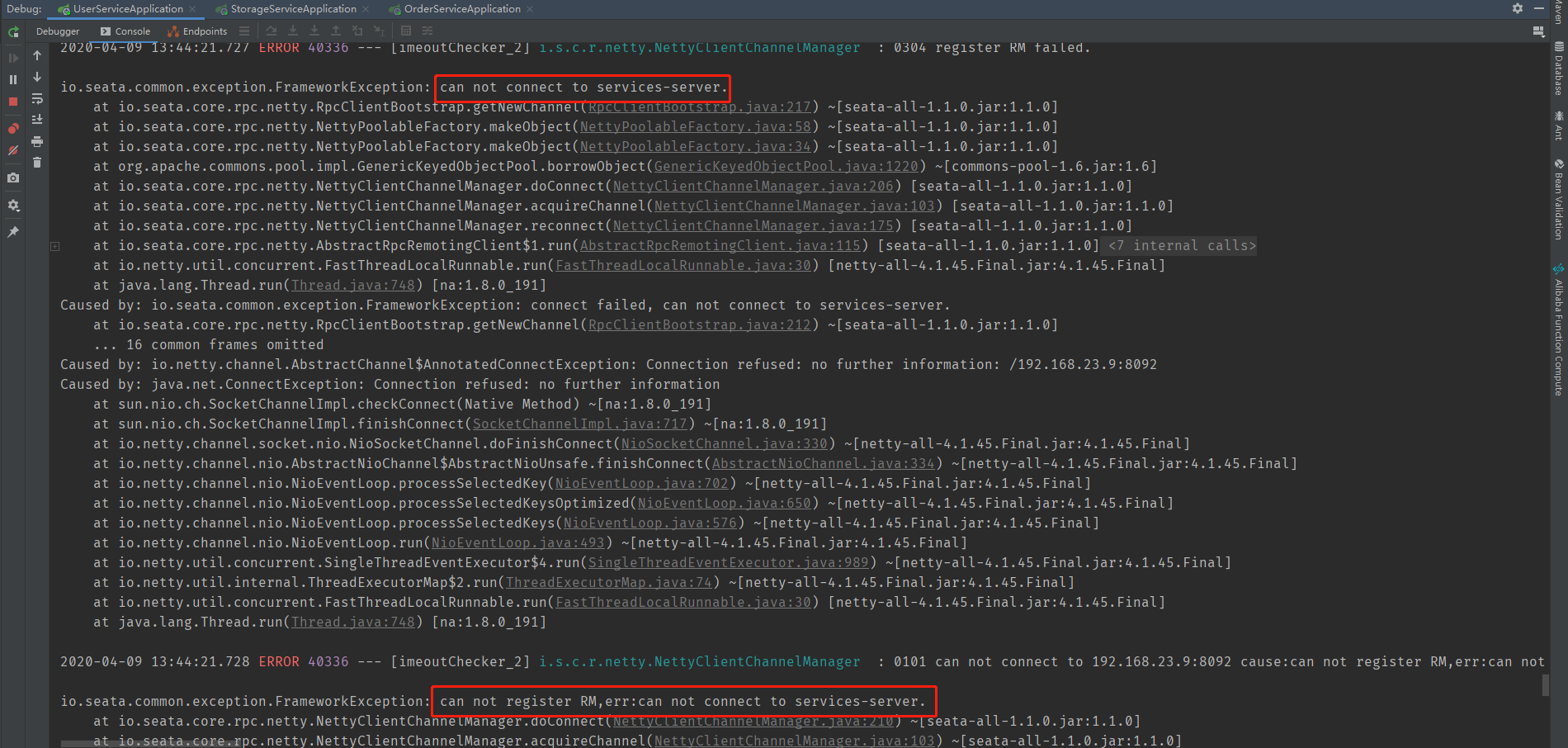
情景5、回滚时除负责开启全局事务的TC外的TC宕机



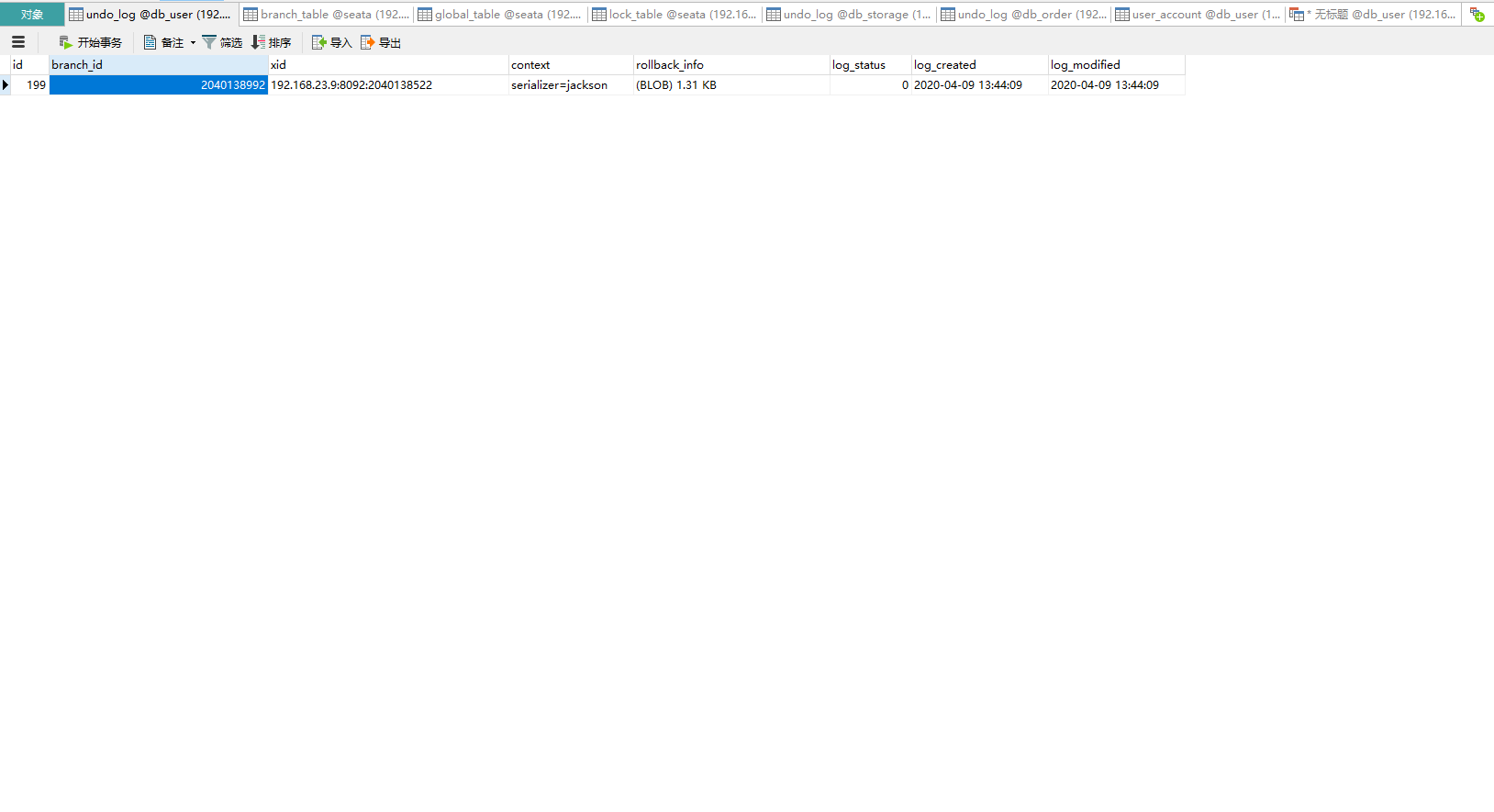
事务成功回滚

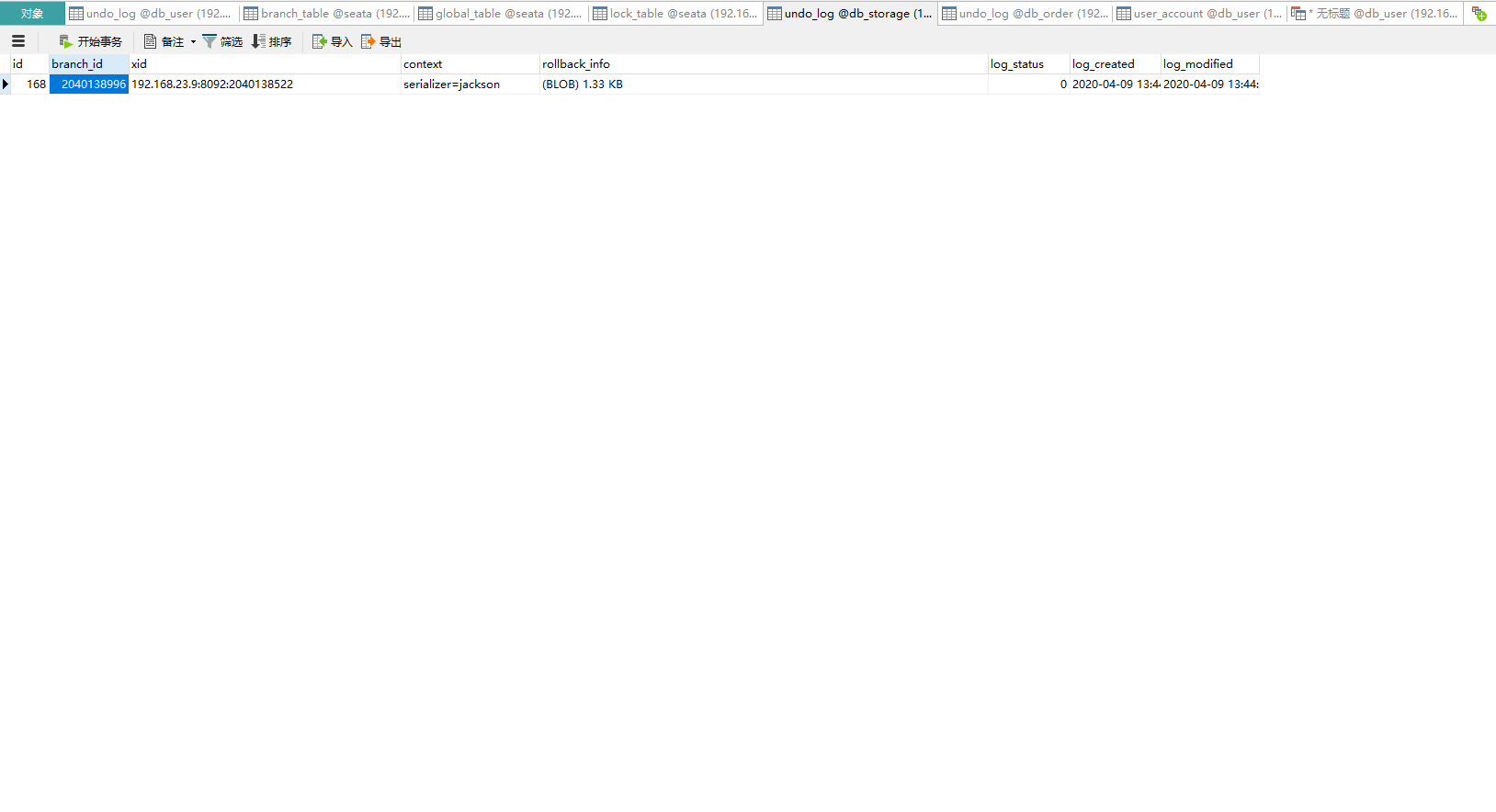
情景6、全部TC 宕机

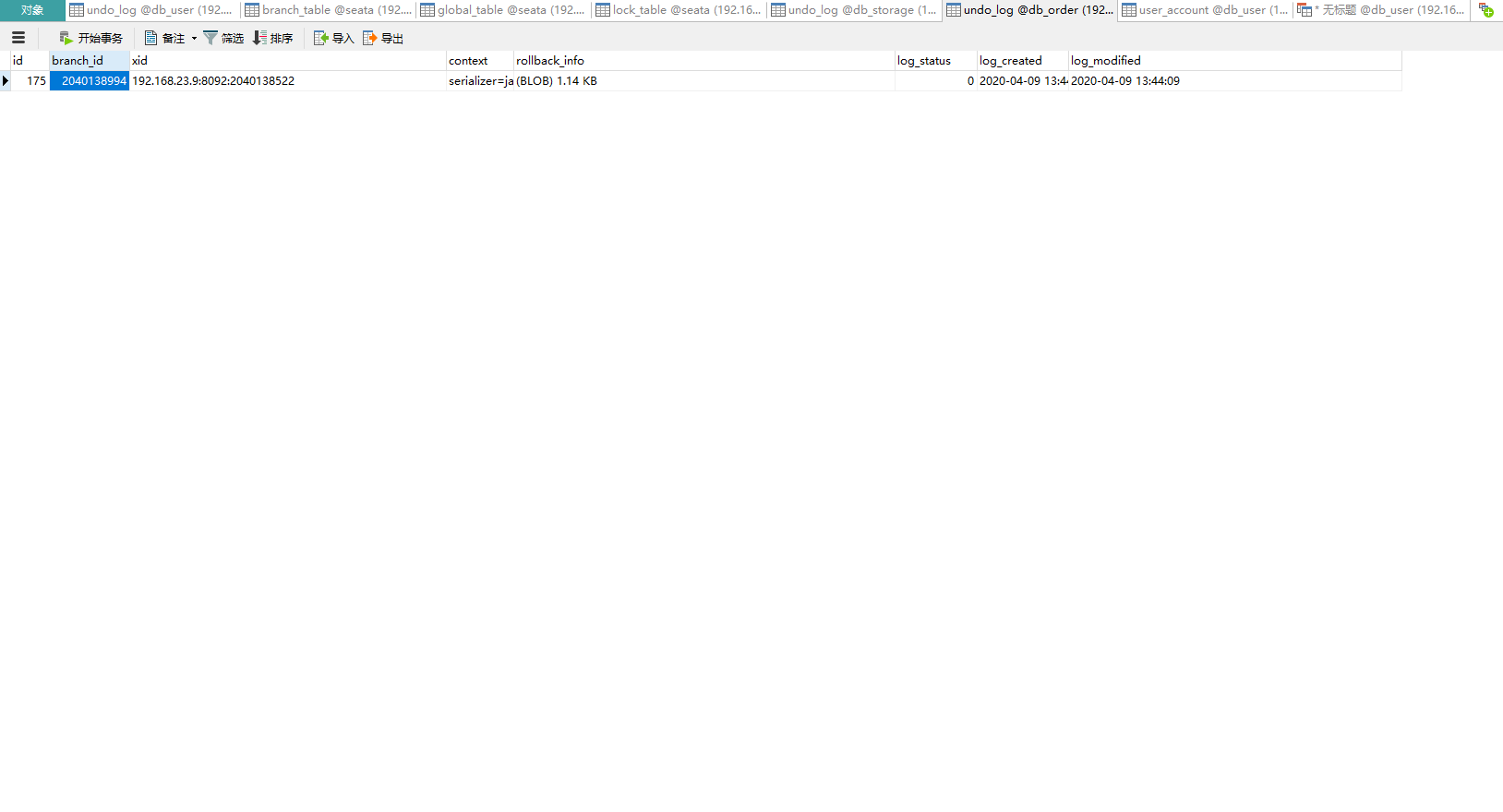
微服务端报错无法连接TC 和无法注册 RM

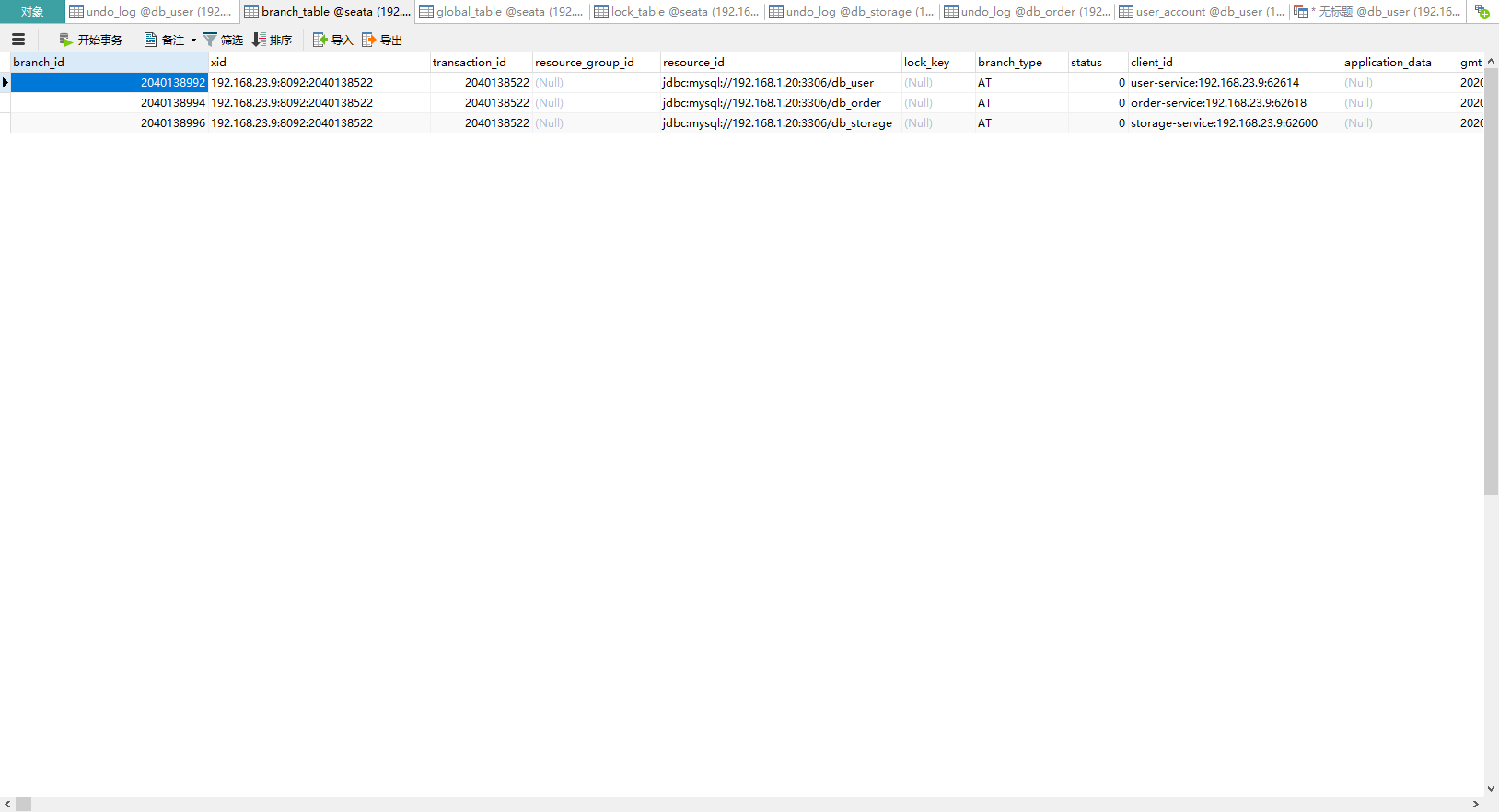


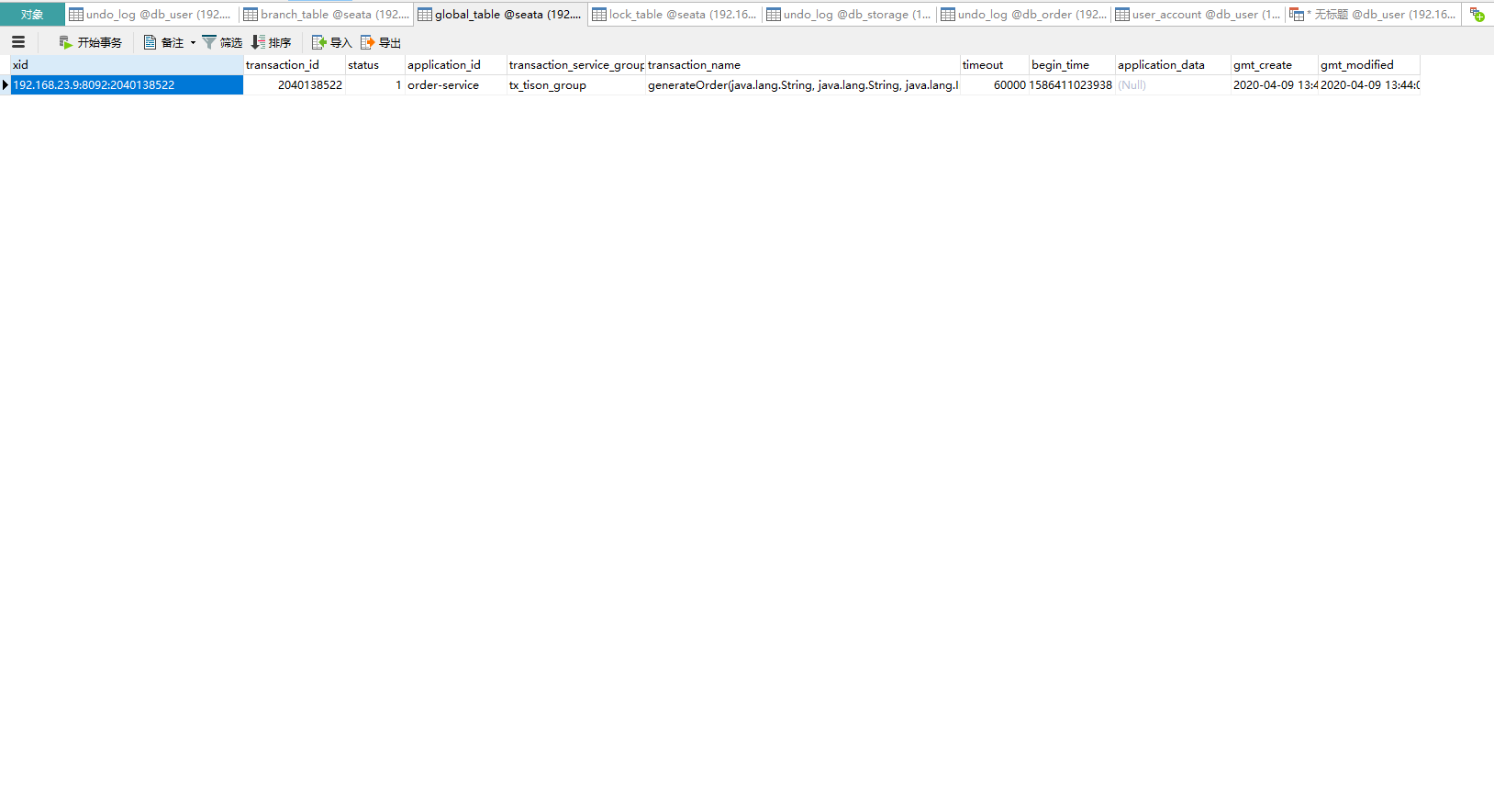
Undo表出现回滚数据

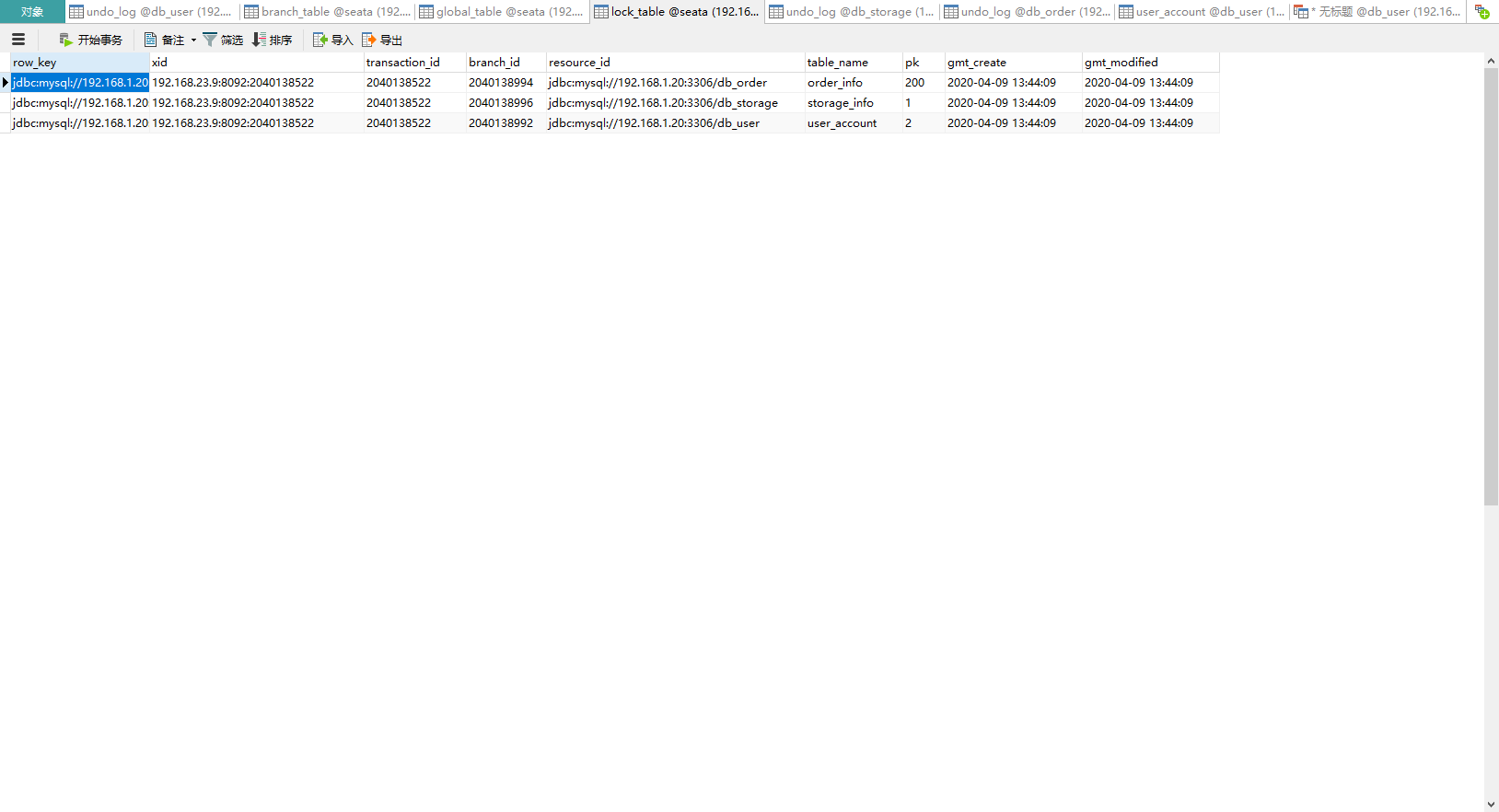










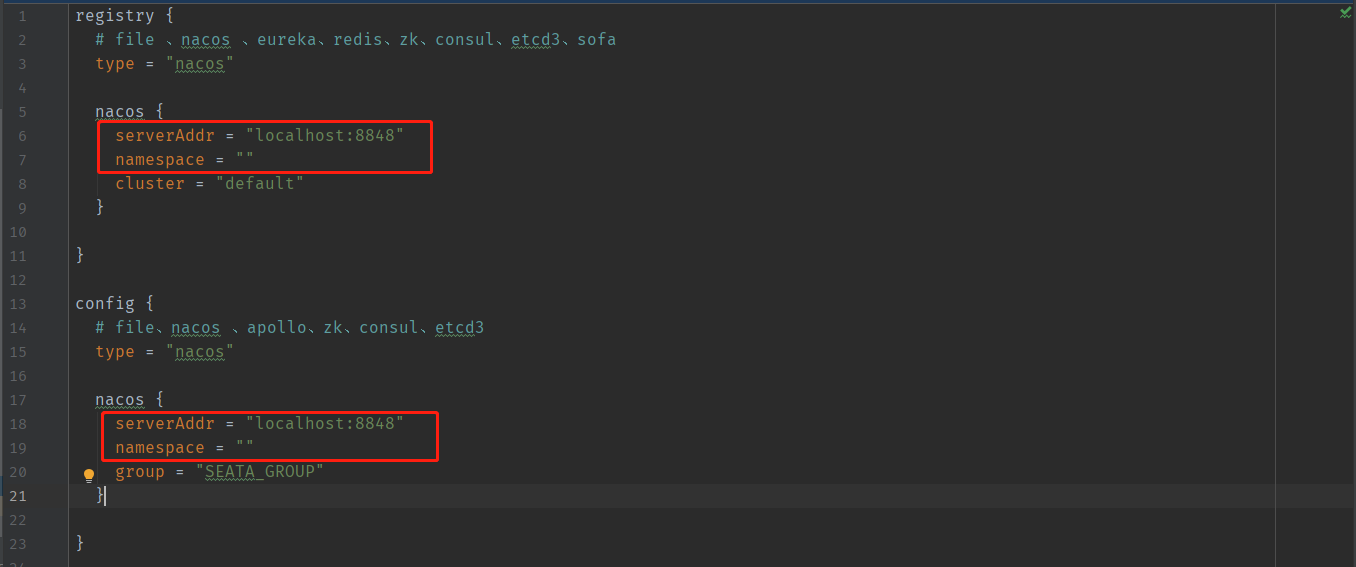


随意重启任意一个TC，自动进行回滚

七、如何使用

1、配置

修改文件中的nocos 地址



将Seata目录下register.conf文件复制到参与全局事务的各角色（TM,RM）项目resources 目录下

2、在所有业务数据库执行 db\_undo\_log.sql 脚本

3、新建名为“seata”的数据库，执行db\_store.sql 脚本

4、在所有参与全局事务的模块启动类之前加上注解@EnableAutoDataSourceProxy

5、启动Seata Server /bin目录下的启动脚本

6、在需要开启全局事务的Servvice 或 Controller 上 添加 @GlobalTransactional注解

原理、 测试 使用 分开