Seata分布式事务

一、安装与配置

1、拉取镜像

Shell 复制代码

```shell
docker pull seataio/seata-server:1.5.0
```

2、注册中心配置

Shell 复制代码

```shell
// 将下面两个配置文件放在conf当中
mkdir -p /mydata/seata/conf
mkdir -p /mydata/seata/logs
```

registry.conf　　Shell　复制代码

```shell
registry {
  # file 、nacos 、eureka、redis、zk、consul、etcd3、sofa
  # 默认为file  这里我是使用nacos为注册中心
  type = "nacos"

  nacos {
    application = "seata-server"
    # nacos的地址
    serverAddr = "120.79.132.251:8848"
    group = "SEATA_GROUP"
    # f81810bc-a134-4e55-955b-2d5c9c602826
    namespace = "f81810bc-a134-4e55-955b-2d5c9c602826"
    cluster = "default"
    username = "nacos"
    password = "nacos"
  }
  eureka {
    serviceUrl = "http://localhost:8761/eureka"
    application = "default"
    weight = "1"
  }
  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 = ""
    password = ""
  }
  consul {
    cluster = "default"
    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"
    }
  }

  config {
    # file、nacos 、apollo、zk、consul、etcd3
    type = "nacos"

    nacos {
      serverAddr = "120.79.132.251:8848"
      # f81810bc-a134-4e55-955b-2d5c9c602826
      namespace = "f81810bc-a134-4e55-955b-2d5c9c602826"
      group = "SEATA_GROUP"
      username = "nacos"
      password = "nacos"
      # dataId = "seataServer.properties"
    }
    consul {
      serverAddr = "127.0.0.1:8500"
    }
    apollo {
      appId = "seata-server"
      apolloMeta = "http://192.168.1.204:8801"
      namespace = "application"
    }
    zk {
      serverAddr = "127.0.0.1:2181"
      sessionTimeout = 6000
      connectTimeout = 2000
      username = ""
      password = ""
    }
    etcd3 {
      serverAddr = "http://localhost:2379"
    }
    file {
     # 配置file.conf的文件路径
      name = "file:/root/seata-config/file.conf"
    }
```

```
}
```

file.conf Shell 复制代码

```shell
## 事务日志存储，仅在seata服务器中使用
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 = "druid"
    ## mysql/oracle/postgresql/h2/oceanbase etc.
    dbType = "mysql"
    driverClassName = "com.mysql.jdbc.Driver"
    url = "jdbc:mysql://120.79.132.251:3306/seata"
    user = "mysql"
    password = "mysql"
    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
    }

  }
```

3、配置中心配置

3.1、修改配置文件

config.txt Properties 复制代码

```
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
# 修改事务分组
service.vgroupMapping.my_tx_group=default
service.default.grouplist=120.79.132.251:8091
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=druid
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

# 修改数据库模式
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

# 修改数据库连接
```

```
store.db.datasource=druid
store.db.dbType=mysql
store.db.driverClassName=com.mysql.jdbc.Driver
store.db.url=jdbc:mysql://120.79.132.251:3306/seata?useUnicode=true
store.db.user=root
store.db.password=root
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
```

3.2、使用脚本把配置更新到配置中心

nacos-config.sh Shell 复制代码

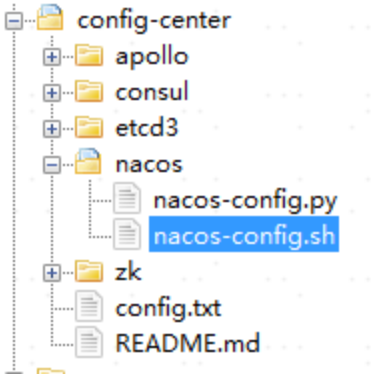
```shell
#!/usr/bin/env bash
# Copyright 1999-2019 Seata.io Group.
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at、
#
#      http://www.apache.org/licenses/LICENSE-2.0
#
# 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.

while getopts ":h:p:g:t:u:w:" opt
do
  case $opt in
  h)
    host=$OPTARG
    ;;
  p)
    port=$OPTARG
    ;;
  g)
    group=$OPTARG
    ;;
  t)
    tenant=$OPTARG
    ;;
  u)
    username=$OPTARG
    ;;
  w)
    password=$OPTARG
    ;;
  ?)
    echo " USAGE OPTION: $0 [-h host] [-p port] [-g group] [-t tenant] [-u username] [-w password] "
    exit 1
    ;;
  esac
done

```

```
if [[ -z ${host} ]]; then
    host=localhost
fi
if [[ -z ${port} ]]; then
    port=8848
fi
if [[ -z ${group} ]]; then
    group="SEATA_GROUP"
fi
if [[ -z ${tenant} ]]; then
    tenant=""
fi
if [[ -z ${username} ]]; then
    username=""
fi
if [[ -z ${password} ]]; then
    password=""
fi

nacosAddr=$host:$port
contentType="content-type:application/json;charset=UTF-8"

echo "set nacosAddr=$nacosAddr"
echo "set group=$group"

failCount=0
tempLog=$(mktemp -u)
function addConfig() {
  curl -X POST -H "${contentType}" "http://$nacosAddr/nacos/v1/cs/configs?dataId=$1&group=$group&content=$2&tenant=$tenant&username=$username&password=$password" >"${tempLog}" 2>/dev/null
  if [[ -z $(cat "${tempLog}") ]]; then
    echo " Please check the cluster status. "
    exit 1
  fi
  if [[ $(cat "${tempLog}") =~ "true" ]]; then
    echo "Set $1=$2 successfully "
  else
    echo "Set $1=$2 failure "
    (( failCount++ ))
  fi
}

count=0
for line in $(cat $(dirname "$PWD")/config.txt | sed s/[[:space:]]//g); do
  (( count++ ))
```

```
  key=${line%%=*}
    value=${line#*=}
  addConfig "${key}" "${value}"
done

echo "==========================================================================="
echo " Complete initialization parameters,  total-count:$count , failure-count:$failCount "
echo "==========================================================================="

if [[ ${failCount} -eq 0 ]]; then
  echo " Init nacos config finished, please start seata-server. "
else
  echo " init nacos config fail. "
fi
```

3.3、执行脚本

使用git执行以下命令



Shell 复制代码

```
sh nacos-config.sh -h 120.79.132.251 -p 8848 -g SEATA_GROUP -t f81810bc-a134-4e55-955b-2d5c9c602826

参数说明:
-h: host,默认值localhost
-p: port,默认值8848
-g:配置分组,默认值为'SEATA_GROUP'
-t:租户信息，对应Nacos的命名空间ID字段，默认值为空
```

3.4、执行结果

4、创建容器

SQL 复制代码

```sql
docker run -d -p 8091:8091 \
-v /mydata/seata/conf/registry.conf:/seata-server/resources/registry.conf \
-v /mydata/seata/conf/file.conf:/seata-server/resources/file.conf \
-v /mydata/seata/logs:/root/logs \
-e SEATA_IP=120.79.132.251 \
-e SEATA_PORT=8091 \
--restart=always \
--name seata  seataio/seata-server:1.5.0
```

5、客户端配置

application.yml YAML 复制代码

```yaml
spring:
  datasource:
    type: com.alibaba.druid.pool.DruidDataSource
    driver-class-name: com.mysql.cj.jdbc.Driver
    username: root
    password: root
    url: jdbc:mysql://120.79.132.251:3306/seata_account?useUnicode=true&characterEncoding=UTF-8&useSSL=false&serverTimezone=Asia/Shanghai
  cloud:
    alibaba:
      seata:
        tx-service-group: seata-account_tx_group
    nacos:
      discovery:
        server-addr: 120.79.132.251:8848
  application:
    name: seata-account
  jackson:
    date-format: yyyy-MM-dd HH:mm:ss


mybatis-plus:
  mapper-locations: classpath:/mapper/**/*.xml
  global-config:
    db-config:
      id-type: auto
      logic-delete-value: 1
      logic-not-delete-value: 0
server:
  port: 8080
  servlet:
    context-path: /
logging:
  level:
    com.formssi.account: debug

seata:
  tx-service-group: my_tx_group #与服务端保持一致
  service:
    vgroup-mapping:
      my_tx_group: default
```

transport.threadFactory.workerThreadSize	SEATA_GROUP	
transport.shutdown.wait	SEATA_GROUP	
service.vgroupMapping.my_tx_group	SEATA_GROUP	
service.default.grouplist	SEATA_GROUP	
service.enableDegrade	SEATA_GROUP	
service.disableGlobalTransaction	SEATA_GROUP	
client.rm.asyncCommitBufferLimit	SEATA_GROUP	

6、加入数据表

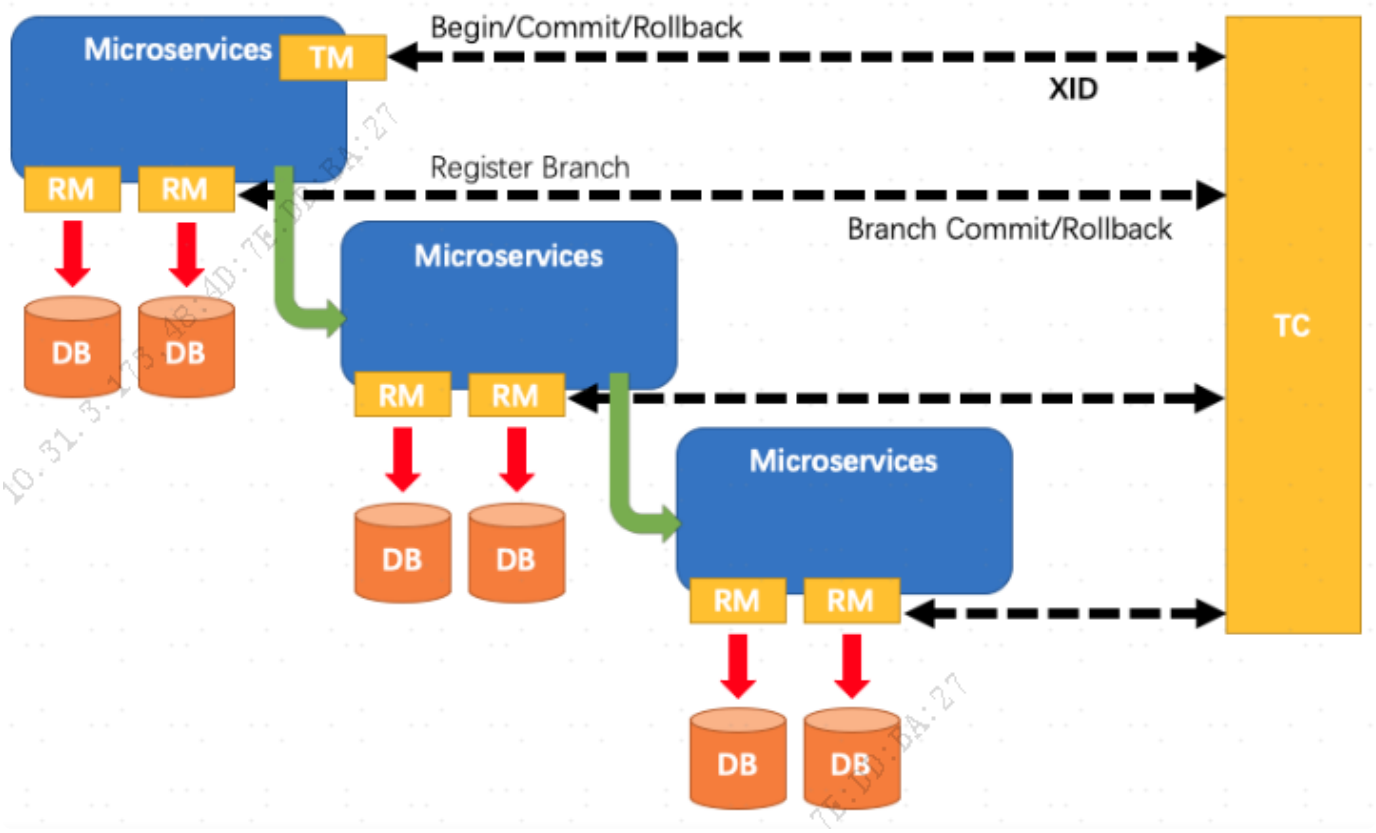
SQL 复制代码

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

-- the table to store BranchSession data
CREATE TABLE IF NOT EXISTS `branch_table`
(
    `branch_id`         BIGINT       NOT NULL,
    `xid`               VARCHAR(128) NOT NULL,
    `transaction_id`    BIGINT,
    `resource_group_id` VARCHAR(32),
    `resource_id`       VARCHAR(256),
    `branch_type`       VARCHAR(8),
    `status`            TINYINT,
    `client_id`         VARCHAR(64),
    `application_data`  VARCHAR(2000),
    `gmt_create`        DATETIME(6),
    `gmt_modified`      DATETIME(6),
    PRIMARY KEY (`branch_id`),
    KEY `idx_xid` (`xid`)
) ENGINE = InnoDB
  DEFAULT CHARSET = utf8;

-- the table to store lock data
CREATE TABLE IF NOT EXISTS `lock_table`
(
    `row_key`        VARCHAR(128) NOT NULL,
    `xid`            VARCHAR(128),
    `transaction_id` BIGINT,
```

```sql
    `branch_id`      BIGINT       NOT NULL,
    `resource_id`    VARCHAR(256),
    `table_name`     VARCHAR(32),
    `pk`             VARCHAR(36),
    `status`         TINYINT      NOT NULL DEFAULT '0' COMMENT '0:locked,1:rollbacking',
    `gmt_create`     DATETIME,
    `gmt_modified`   DATETIME,
    PRIMARY KEY (`row_key`),
    KEY `idx_status` (`status`),
    KEY `idx_branch_id` (`branch_id`)
) ENGINE = InnoDB
  DEFAULT CHARSET = utf8;

-- 业务数据库增加日志表
-- 注意此处0.7.0+ 增加字段 context
CREATE TABLE `undo_log` (
  `id` bigint(20) NOT NULL AUTO_INCREMENT,
  `branch_id` bigint(20) NOT NULL,
  `xid` varchar(100) NOT NULL,
  `context` varchar(128) NOT NULL,
  `rollback_info` longblob NOT NULL,
  `log_status` int(11) NOT NULL,
  `log_created` datetime NOT NULL,
  `log_modified` datetime NOT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `ux_undo_log` (`xid`,`branch_id`)
) ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT CHARSET=utf8;
```

二、Seata解决方案

1. Seata是一款开源的分布式事务解决方案，致力于提高性能和简单易用的分布式事务服务。为用户提供了AT、TCC、SAGA、XA事务模式
2. Seata分为三大模块
   a. TC（Trainsaction Coordinator）：事务协调器。维护全局事务的运行状态，负责协调并驱动全局事务的回滚或提交。
   b. TM（Trainsaction Manager）：事务发起器。控制全局事务的边界，负责开启一个全局事务，并最终发起全局提交或全局回滚的决策。
   c. RM（Resource Manager）：资源管理器。管理每个分支事务的资源，每一个RM都会作为一个分支事务注册在TC。



1、AT

1.1、执行流程

1. 第一阶段

a. 开启全局事务注解@GlobalTransactional，使该业务成为TM

i. TM会向TC申请一个全局事务ID，该内容会写入到**global_table**

```
ter : xid in RootContext[null] xid in HttpContext[null]
    : offer message: timeout=60000,transactionName=shopping(com.formssi.order.entity.OrderEntity)
    : write message:SeataMergeMessage timeout=60000,transactionName=shopping(com.formssi.order.entity.OrderEntity)
?true,isopen?true
    : io.seata.core.rpc.netty.TmNettyRemotingClient@22d5d139 msgId:10, body:MergeResultMessage xid=120.79.132.251:8091:272009956104871936;extraData=;

    : bind 120.79.132.251:8091:272009956104871936
on  : Begin new global transaction [120.79.132.251:8091:272009956104871936]
```

```
2022-05-23 06:30:27.245  INFO --- [ServerHandlerThread_1_43_500] i.s.s.coordinator.DefaultCoordinator     : Begin new global transaction
applicationId: seata-order,transactionServiceGroup: my_tx_group, transactionName: shopping(com.formssi.order.entity.OrderEntity),timeout:
60000,xid:120.79.132.251:8091:272009956104871936
```

global_table

	xid	transaction_id	status	application_id	transaction_service_group	transaction_name	timeout	begin_time
1	120.79.132.251:8091:272009956104871936	272,009,956,104,871,936	1	seata-order	my_tx_group	shopping(com.formssi.orc	60,000	1,653,287,427,237

b. 开启分支事务注解@Transactional，使该业务成为RM

i. RM会向TC申请分支事务ID，该内容会写入branch_table

```
    : offer message: xid=120.79.132.251:8091:272009956104871936,branchType=AT,resourceId=jdbc:mysql://120.79.132.251:3306/seata_order,lockKey=t_order:19
    : write message:SeataMergeMessage xid=120.79.132.251:8091:272009956104871936,branchType=AT,resourceId=jdbc:mysql://120.79.132.251:3306/seata_order,lockKey=t_order:1
?true,isopen?true
    : io.seata.core.rpc.netty.RmNettyRemotingClient@65062a05 msgId:13, body:MergeResultMessage BranchRegisterResponse: branchId=272009956352335873,result code =Success,
```

```
2022-05-23 06:30:27.292  INFO --- [batchLoggerPrint_1_1] i.s.c.r.p.server.BatchLogHandler         : SeataMergeMessage xid=120.79.132.251:
8091:272009956104871936,branchType=AT,resourceId=jdbc:mysql://120.79.132.251:3306/seata_order,lockKey=t_order:19
,clientIp:183.11.235.45,vgroup:my_tx_group
2022-05-23 06:30:27.324  INFO --- [ServerHandlerThread_1_44_500] i.seata.server.coordinator.AbstractCore  : Register branch successfully,
 xid = 120.79.132.251:8091:272009956104871936, branchId = 272009956352335873, resourceId = jdbc:mysql://120.79.132.251:3306/seata_order ,
lockKeys = t_order:19
```

branch_table

	branch_id	xid	transaction_id	resource_group_id	resource_id	lock_key	branch_type
1	272,010,015,458,467,841	120.79.132.251:8091:272009956104871936	272,009,956,104,871,936	[NULL]	jdbc:mysql://120.79.132.251:3306/seata_account	[NULL]	AT
2	272,009,956,352,335,873	120.79.132.251:8091:272009956104871936	272,009,956,104,871,936	[NULL]	jdbc:mysql://120.79.132.251:3306/seata_order	[NULL]	AT

c. 写入完成之后还会在lock_table记录被锁定的事务信息，主要是记录行数据信息

lock_table

	row_key	xid	transaction_id	branch_id	resource_id	table_name	pk
1	9.132.251:3306/seata_account^^^t_account^^^1	120.79.132..	272009956104871936	272010015458467	jdbc:mysql://120.79.132.251:3306/seata_account	t_account	1
2	jdbc:mysql://120.79.132.251:3306/seata_order^^^t_order^^^19	120.79.132..	272009956104871936	272009956352335	jdbc:mysql://120.79.132.251:3306/seata_order	t_order	19

d. 处理完上述信息，每个业务的数据库都会记录相对应的日志

```
: Flushing UNDO LOG: {"@class":"io.seata.rm.datasource.undo.BranchUndoLog","xid":"120.79.132.251:8091:272009956104871936","branchId":272009956352335873,"sqlUndoLogs
: <==    Updates: 1
```

undo_log

	id	branch_id	xid	context	rollback_info	log_status
1	12	272,009,956,352,335,873	120.79.132.251:8091:272009956104871936	serializer=jackson	{"@class":"io.seata.rm.datasourc... [1309]	0

undo_log

	id	branch_id	xid	context	rollback_info	log_status
1	13	272,010,015,458,467,841	120.79.132.251:8091:272009956104871936	serializer=jackson	{"@class":"io.seata.rm.datasourc... [1133]	0

2. 第二阶段

a. 全局提交，删除相关日志信息

```
: unbind 120.79.132.251:8091:272009956104871936
: Suspending current transaction,xid = 120.79.132.251:8091:272009956104871936
: [120.79.132.251:8091:272009956104871936] commit status: Committed
: io.seata.core.rpc.netty.RmNettyRemotingClient@65062a05 msgId:59, body:xid=120.79.132.251:8091:272009956104871936,branchId=272009956352335873,branchType=AT,resourceId=
: rm client handle branch commit process:xid=120.79.132.251:8091:272009956104871936,branchId=272009956352335873,branchType=AT,resourceId=jdbc:mysql://120.79.132.251:330
: Branch committing: 120.79.132.251:8091:272009956104871936 272009956352335873 jdbc:mysql://120.79.132.251:3306/seata_order null
: Branch commit result: PhaseTwo_Committed
: branch commit result:xid=120.79.132.251:8091:272009956104871936,branchId=272009956352335873,branchStatus=PhaseTwo_Committed,result code =Success,getMsg =null
: write message:xid=120.79.132.251:8091:272009956104871936,branchId=272009956352335873,branchStatus=PhaseTwo_Committed,result code =Success,getMsg =null, channel:[id: 0
: batch delete undo log size 1
```

b. 全局回滚，通过undo日志对数据进行回滚并删除日志信息

```
: rm handle branch rollback process:xid=120.79.132.251:8091:272033508220739584,branchId=272033508661141505,branchType=AT,resourceId=jdbc:mysql://120.79.132.251:3306/seata_o
: Branch Rollbacking: 120.79.132.251:8091:272033508220739584 272033508661141505 jdbc:mysql://120.79.132.251:3306/seata_order
: xid 120.79.132.251:8091:272033508220739584 branch 272033508661141505, undo_log deleted with GlobalFinished
: Branch Rollbacked result: PhaseTwo_Rollbacked
: [120.79.132.251:8091:272033508220739584] rollback status: Rollbacked
```

1.2、代码实现

AT模式代码实现 Java 复制代码

```java
@PostMapping("/shopping")
public R shopping(@RequestBody OrderEntity orderEntity){
    orderService.shopping(orderEntity);
    return R.ok();
}

/**
 * 主分支TM
 * @param orderEntity
*/
@GlobalTransactional
@Override
public void shopping(OrderEntity orderEntity) {
    log.info("开启第一个事务");
    createOrder(orderEntity);
    log.info("开启第二个事务");

 accountFeignService.deduct(orderEntity.getUserId(),orderEntity.getAmount
());
}

/**
 * 子分支RM
 * 创建订单
 * @param orderEntity
*/
@Transactional
@Override
public boolean createOrder(OrderEntity orderEntity) {
    orderEntity.setOrderNo(UUID.randomUUID().toString());
    orderDao.insert(orderEntity);
    //模拟出错
    if(orderEntity.getUserId()==0){
        log.info("模拟业务出错",1/0);
    }
    return true;
}

/**
 * 子分支RM
 * 更新账户
 * @param map
*/
@Transactional
@Override
```

```java
public void deductAccount(HashMap<String, Object> map){
    accountDao.deductAccount((Integer) map.get("amount"),
                                new QueryWrapper<AccountEntity>
().eq("user_id",map.get("userId")));
    //模拟出错
    if(Integer.parseInt(map.get("userId").toString())==2){
        log.info("模拟业务出错",1/0);
    }
}
```

1.3、总结

1. 会在第一阶段就提交事务
2. 一阶段提交事务之后，如果有其他事务来修改数据，AT模式可以确保回滚的数据正确；因为在回滚数据时，会先校验redo（修改后的数据）的数据和当前的数据是否一致，如果一致直接回滚，不一致再根据undo里的数据进行处理。

2、TCC

2.1、执行流程

1. 第一阶段

a. 通过@GlobalTransactional生成全局事务ID

global_table

	xid	transaction_id	stat	application	transaction_service_group	transaction_name
1	120.79.132.251:8091:272659766134710272	272,659,766,134,710,272	1	seata-order	my_tx_group	tccShopping(com.formssi

b. 通过@Transactional生成分支事务ID

branch_table

	branch_id	xid	transaction_id	resource	resource_id	branch	client_id	app
1	8,210,857,426,945	120.79.132..	2,658,202,338,795,520	[NULL]	TccOrderService	TCC	0	seata-order:183.11.235.45:2452	{"actior
2	8,262,959,071,233	120.79.132..	2,658,202,338,795,520	[NULL]	TccAccountService	TCC	0	seata-account:183.11.235.45:2451	{"actior

c. try

i. 使用@TwoPhaseBusinessAction注解标记try方法，用@BusinessActionContextParameter标记要传入下个阶段的参数

ii. 业务会先进入try方法执行预处理业务，完成try阶段就会释放锁资源

tcc_order 输入一个 SQL 表达式来过滤结果 (使用 Ctrl+Space)

	id	order_no	user_id	amount	order_status	remark
1	56	1675d70b2ba34ff08bd043f369759b47	1	100	0	[NULL]

tcc_account 输入一个 SQL 表达式来过滤结果 (使用 Ctrl+Space)

	id	user_id	balance	frozen	remark
1	1	1	2,900	100	[NULL]
2	2	2	2,000	0	[NULL]

2. 第二阶段

a. commit

i. 第一阶段结束后，seata会根据branch_table中记录的信息调用comitMethod，并且把上下文信息传递到commitMethod中

tcc_order 输入一个 SQL 表达式来过滤结果 (使用 Ctrl+Space)

	id	order_no	user_id	amount	order_status	remark
1	56	1675d70b2ba34ff08bd043f369759b47	1	100	1	[NULL]

tcc_account 输入一个 SQL 表达式来过滤结果 (使用 Ctrl+Space)

	id	user_id	balance	frozen	remark
1	1	1	2,900	0	[NULL]
2	2	2	2,000	0	[NULL]

b. cancel（这个过程会出现一个问题）

i. 在第一阶段出现错误后，seata会根据branch_table中的信息调用cancelMethod，对数据进行回滚

2.2、代码实现

TCC模式代码实现 Java 复制代码

```java
/**
 * TCC模式
 * @param tccOrderEntity
 * @return
 */
@PostMapping("/tcc/shopping")
public R tccShopping(@RequestBody TccOrderEntity tccOrderEntity){
    orderService.tccShopping(tccOrderEntity);
    return R.ok();
}

/**
 * 主分支TM
 * @param orderEntity
 */
@GlobalTransactional
@Override
public void tccShopping(TccOrderEntity orderEntity) {
    log.info("开启第一个事务");
    orderEntity.setOrderNo(UUID.randomUUID().toString().replace("-","""));
    boolean prepare = tccOrderService.prepare(new BusinessActionContext(), orderEntity);
    if(prepare){
        log.info("开启第二个事务");
 accountFeignService.tccDeduct(orderEntity.getUserId(),orderEntity.getAmount(),orderEntity.getErrorPhase());
    }
}

@LocalTCC
public interface TccOrderService {
    @TwoPhaseBusinessAction(name = "TccOrderService",
                            commitMethod = "commit",
                            rollbackMethod = "rollback")
    public boolean prepare(BusinessActionContext actionContext,
                           @BusinessActionContextParameter(paramName = "order")TccOrderEntity tccOrderEntity);

    public boolean commit(BusinessActionContext actionContext);

    public boolean rollback(BusinessActionContext actionContext);
}

```

```
41  /**
42   * 生成中间状态订单
43   * @param actionContext
44   * @param tccOrderEntity
45   * @return
46   */
47  @Override
48  public boolean prepare(BusinessActionContext actionContext,
    TccOrderEntity tccOrderEntity) {
49      log.info("订单业务预备阶段{}",actionContext);
50      tccOrderEntity.setOrderStatus(0);
51      tccOrderDao.insert(tccOrderEntity);
52      //模拟出错
53      if(tccOrderEntity.getErrorPhase()==1){
54          log.info("订单业务预备阶段出错",1/0);
55      }
56      return true;
57  }
58
59  /**
60   * 更新中间状态订单
61   * @param actionContext
62   * @return
63   */
64  @Override
65  public boolean commit(BusinessActionContext actionContext) {
66      log.info("订单业务提交阶段{}",actionContext);
67      TccOrderEntity entity = new TccOrderEntity();
68      JSONObject order =
    JSONObject.parseObject(actionContext.getActionContext("order").toString(
    ));
69      entity.setOrderStatus(1);
70      entity.setOrderNo(order.get("orderNo").toString());
71      tccOrderDao.update(entity,new QueryWrapper<TccOrderEntity>
    ().eq("order_no",entity.getOrderNo()));
72      log.info("参数{}",actionContext.getActionContext().get("order"));
73      //模拟出错
74      if(Integer.parseInt(order.get("errorPhase").toString())==3){
75          try {
76              System.out.println(1/0);
77          }catch (Exception ex){
78              log.error("订单业务提交阶段业务出错");
79              //                  try {
80              //                      TransactionManager manager =
    TransactionManagerHolder.get();
81              //                      GlobalStatus status =
    manager.rollback(actionContext.getXid());
82              //                      log.info("回滚状态{}",status);
```

```
83                //                } catch (TransactionException e) {
84                //                    e.printStackTrace();
85                //                }
86                //
GlobalTransactionContext.reload(actionContext.getXid()).rollback();
87            }
88        }
89        return true;
90    }
91
92    /**
93     * 删除中间态订单
94     * @param actionContext
95     * @return
96     */
97    @Override
98    public boolean rollback(BusinessActionContext actionContext) {
99        log.info("订单业务回滚阶段{}",actionContext);
100       JSONObject order =
JSONObject.parseObject(actionContext.getActionContext("order").toString(
));
101       TccOrderEntity entity = new TccOrderEntity();
102       entity.setOrderStatus(2);
103       tccOrderDao.update(entity,new QueryWrapper<TccOrderEntity>
().eq("order_no",order.get("orderNo")));
104       //        tccOrderDao.delete(new QueryWrapper<TccOrderEntity>
().eq("order_no",order.get("orderNo")));
105       //模拟出错
106       if(Integer.parseInt(order.get("errorPhase").toString())==5){
107           log.info("订单业务回滚阶段业务出错",1/0);
108       }
109       return true;
110   }
111
112
113   @LocalTCC
114   public interface TccAccountService {
115
116       @TwoPhaseBusinessAction(name = "TccAccountService",
117               commitMethod = "commit",
118               rollbackMethod = "rollback")
119       public boolean prepare(BusinessActionContext actionContext,
120                              @BusinessActionContextParameter(paramName =
"map") HashMap<String, Object> map);
121
122       public boolean commit(BusinessActionContext actionContext);
123
124       public boolean rollback(BusinessActionContext actionContext);
```

```
125     }
126
127
128     /**
129      * 冻结账户金额
130      * @param actionContext
131      * @param map
132      * @return
133      */
134     @Override
135     public boolean prepare(BusinessActionContext actionContext,
        HashMap<String, Object> map) {
136         log.info("账户业务预备阶段{}",actionContext);
137
         tccAccountDao.deductAccount(Integer.parseInt(map.get("amount").toString
        ()),
138                                     new QueryWrapper<TccAccountEntity>()
139
        .eq("user_id",Integer.parseInt(map.get("userId").toString())));
140         //模拟出错
141         if(Integer.parseInt(map.get("errorPhase").toString())==2){
142             log.info("账户业务预备阶段业务出错",1/0);
143         }
144         return true;
145     }
146
147     /**
148      * 解除冻结账户信息
149      * @param actionContext
150      * @return
151      */
152     @Override
153     public boolean commit(BusinessActionContext actionContext) {
154         log.info("账户业务提交阶段{}",actionContext);
155         JSONObject map =
        JSONObject.parseObject(actionContext.getActionContext("map").toString())
        ;
156         int userId = Integer.parseInt(map.get("userId").toString());
157         int amount = Integer.parseInt(map.get("amount").toString());
158         tccAccountDao.commitAccount(amount,new
        QueryWrapper<TccAccountEntity>().eq("user_id",userId));
159         //模拟出错
160         if(Integer.parseInt(map.get("errorPhase").toString())==4){
161             log.info("账户业务提交阶段业务出错",1/0);
162         }
163         return true;
164     }
165
```

```java
/**
 * 回滚冻结金额
 * @param actionContext
 * @return
 */
@Override
public boolean rollback(BusinessActionContext actionContext) {
    log.info("账户业务回滚阶段{}",actionContext);
    JSONObject map = JSONObject.parseObject(actionContext.getActionContext("map").toString());
    int userId = Integer.parseInt(map.get("userId").toString());
    int amount = Integer.parseInt(map.get("amount").toString());
    tccAccountDao.rollbackAcount(amount,new QueryWrapper<TccAccountEntity>().eq("user_id",userId));
    //模拟出错
    if(Integer.parseInt(map.get("errorPhase").toString())==6){
        log.info("账户业务回滚阶段业务出错",1/0);
    }
    return true;
}
```

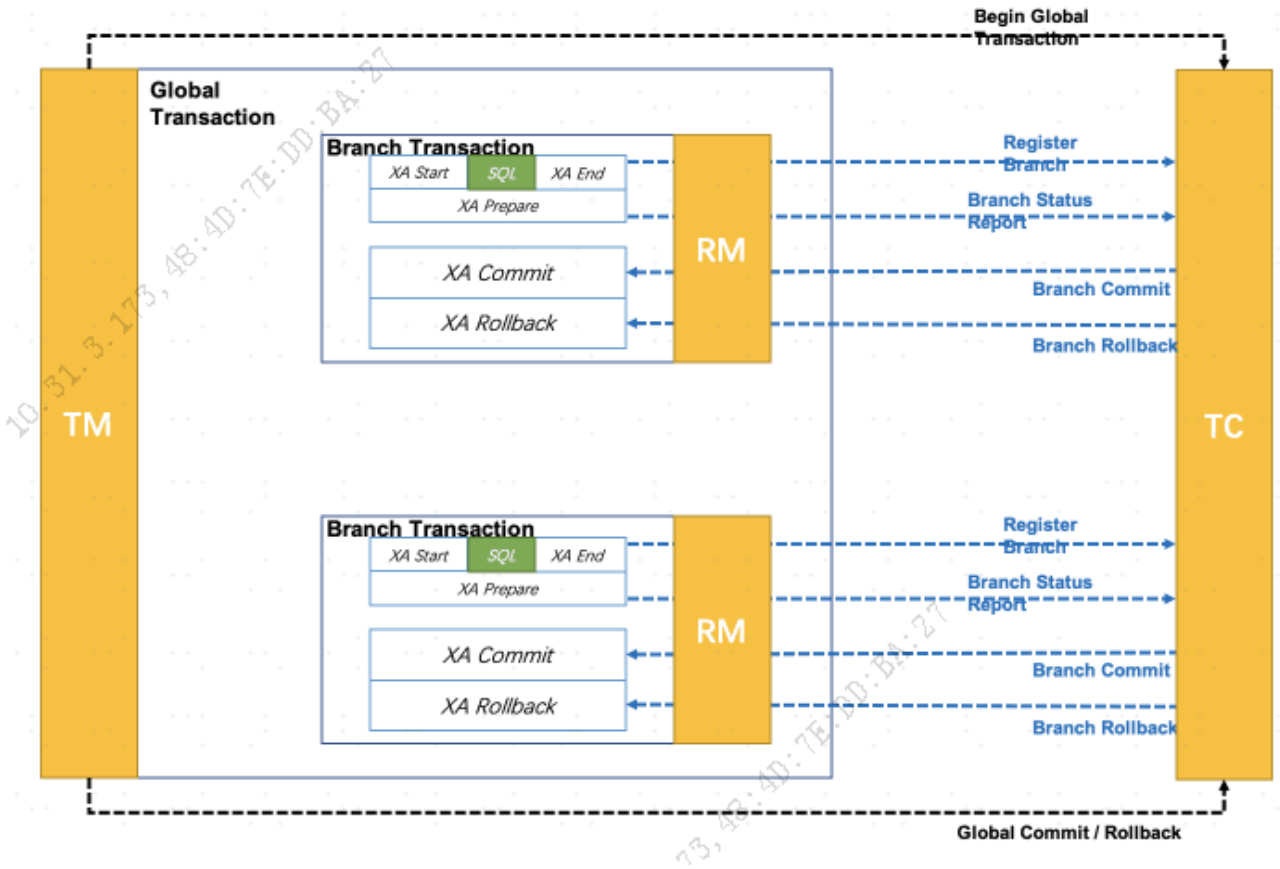
2.3、总结

1. 只要成功执行一个RM的commit，就会把该RM的事务信息删除，如果第二个RM的commit出现问题，也没办法回滚之前的业务
2. 只要有RM在commit阶段出现问题，seata会一直重试commit，不会进入回滚阶段
3. 无法手动回滚
4. 每个阶段执行后不锁资源

3、XA

3.1、基本概念

在 Seata 定义的分布式事务框架内，利用事务资源（数据库、消息服务等）对 XA 协议的支持，以 XA 协议的机制来管理分支事务的一种事务模式。（和AT模式类似，只需要XA代理数据源）



4、Saga

https://github.com/seata/seata-samples/tree/master/saga

4.1、基本概念

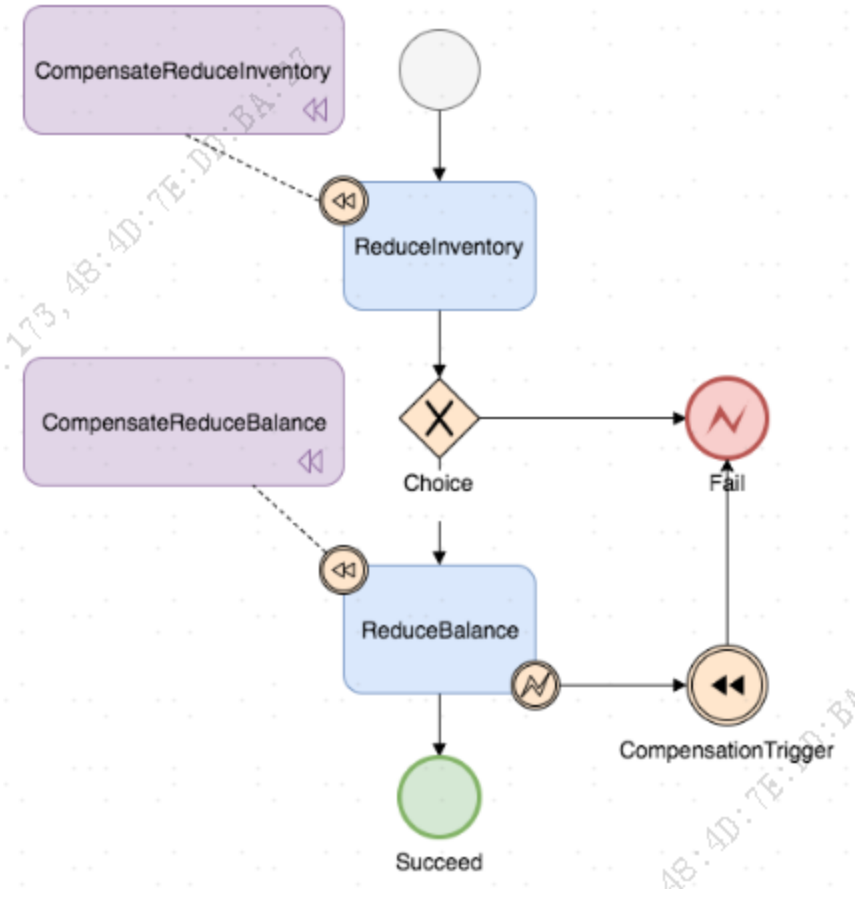
saga模式是seata提供的长事务解决方案，基于状态机引擎实现

机制：

1. 通过状态图来定义服务的调用并生成json状态语言定义文件
2. 状态图中一个节点可以是调用一个服务，节点可以配置补偿节点
3. 状态图 json 由状态机引擎驱动执行，当出现异常时状态引擎反向执行已成功节点对应的补偿节点将事务回滚
4. 可以实现服务编排要求，支持单项选择、并发、子流程、参数转换、参数映射、服务执行状态判断、异常捕获等功能

4.2、执行流程

1. 调用状态机start方法（ReduceInventory）
2. 构建状态实例，并写入到seata_state_inst
3. 执行状态实例中的服务
    a. 执行成功，进入下一个节点（ReduceBalance）
        i. 在ReduceBalance节点出现问题，会触发CompensationTrigger，把XID对应的事务都标记为补偿触发状态，TC根据状态实例找到所有需要补偿的状态，执行补偿方法
    b. 执行失败，进入补偿节点（CompensateReduceInventory）



reduce.json JSON 复制代码

```
1  {
2      "Name": "reduceInventoryAndBalance",//定义状态机的名字
3      "Comment": "reduce inventory then reduce balance in a transaction",
4      "StartState": "ReduceInventory",//定义状态机的初始状态，也就是状态机开始运行时第一个执行的状态
5      "Version": "0.0.1",
6      //下面是状态的定义
7      "States": {
8          "ReduceInventory": {…},
30         //选择状态，该状态里面有一个判断，可以根据判断结果执行不同的状态
31         "ChoiceState":{…},
41         "ReduceBalance": {…},
72         "CompensateReduceInventory": {…},
80         "CompensateReduceBalance": {…},
88         //补偿触发器状态，该状态表示状态机进入补偿，接下来要开始执行各个状态的补偿状态了
89         "CompensationTrigger": {…},
93         //下面两个状态是终止状态
94         "Succeed": {…},
97         "Fail": {…}
102    }
103 }
104
```

table SQL 复制代码

```sql
-- seata_order.seata_state_inst definition
-- 保存状态实例
CREATE TABLE `seata_state_inst` (
  `id` varchar(48) NOT NULL COMMENT 'id',
  `machine_inst_id` varchar(128) NOT NULL COMMENT 'state machine instance id',
  `name` varchar(128) NOT NULL COMMENT 'state name',
  `type` varchar(20) DEFAULT NULL COMMENT 'state type',
  `service_name` varchar(128) DEFAULT NULL COMMENT 'service name',
  `service_method` varchar(128) DEFAULT NULL COMMENT 'method name',
  `service_type` varchar(16) DEFAULT NULL COMMENT 'service type',
  `business_key` varchar(48) DEFAULT NULL COMMENT 'business key',
  `state_id_compensated_for` varchar(50) DEFAULT NULL COMMENT 'state compensated for',
  `state_id_retried_for` varchar(50) DEFAULT NULL COMMENT 'state retried for',
  `gmt_started` timestamp(3) NOT NULL DEFAULT CURRENT_TIMESTAMP(3) ON UPDATE CURRENT_TIMESTAMP(3) COMMENT 'start time',
  `is_for_update` tinyint(1) DEFAULT NULL COMMENT 'is service for update',
  `input_params` blob COMMENT 'input parameters',
  `output_params` blob COMMENT 'output parameters',
  `status` varchar(2) NOT NULL COMMENT 'status(SU succeed|FA failed|UN unknown|SK skipped|RU running)',
  `excep` blob COMMENT 'exception',
  `gmt_end` datetime DEFAULT NULL COMMENT 'end time',
  PRIMARY KEY (`id`,`machine_inst_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;


-- seata_order.seata_state_machine_def definition
-- 保存状态机定义
CREATE TABLE `seata_state_machine_def` (
  `id` varchar(32) NOT NULL COMMENT 'id',
  `name` varchar(128) NOT NULL COMMENT 'name',
  `tenant_id` varchar(32) NOT NULL COMMENT 'tenant id',
  `app_name` varchar(32) NOT NULL COMMENT 'application name',
  `type` varchar(20) DEFAULT NULL COMMENT 'state language type',
  `comment_` varchar(255) DEFAULT NULL COMMENT 'comment',
  `ver` varchar(16) NOT NULL COMMENT 'version',
  `gmt_create` timestamp(3) NOT NULL DEFAULT CURRENT_TIMESTAMP(3) ON UPDATE CURRENT_TIMESTAMP(3) COMMENT 'create time',
  `status` varchar(2) NOT NULL COMMENT 'status(AC:active|IN:inactive)',
  `content` blob COMMENT 'content',
```

```sql
  `recover_strategy` varchar(16) DEFAULT NULL COMMENT 'transaction recover strategy(compensate|retry)',
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;


-- seata_order.seata_state_machine_inst definition
-- 保存状态机实例
CREATE TABLE `seata_state_machine_inst` (
  `id` varchar(128) NOT NULL COMMENT 'id',
  `machine_id` varchar(32) NOT NULL COMMENT 'state machine definition id',
  `tenant_id` varchar(32) NOT NULL COMMENT 'tenant id',
  `parent_id` varchar(128) DEFAULT NULL COMMENT 'parent id',
  `gmt_started` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP COMMENT 'start time',
  `business_key` varchar(48) DEFAULT NULL COMMENT 'business key',
  `start_params` blob COMMENT 'start parameters',
  `gmt_end` datetime DEFAULT NULL COMMENT 'end time',
  `excep` blob COMMENT 'exception',
  `end_params` blob COMMENT 'end parameters',
  `status` varchar(2) DEFAULT NULL COMMENT 'status(SU succeed|FA failed|UN unknown|SK skipped|RU running)',
  `compensation_status` varchar(2) DEFAULT NULL COMMENT 'compensation status(SU succeed|FA failed|UN unknown|SK skipped|RU running)',
  `is_running` tinyint(1) DEFAULT NULL COMMENT 'is running(0 no|1 yes)',
  `gmt_updated` datetime NOT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `unikey_buz_tenant` (`business_key`,`tenant_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

实现代码 Java 复制代码

```java
@Autowired
StateMachineEngine stateMachineEngine;
@GetMapping("/create")
public String create() {
    log.info("=========开始创建订单============");
    Map<String, Object> startParams = new HashMap<>(3);
    //唯一健
    String businessKey = String.valueOf(System.currentTimeMillis());
    startParams.put("businessKey", businessKey);
    startParams.put("count", 10);
    startParams.put("amount", new BigDecimal("400"));

    //同步执行
    StateMachineInstance inst =

 stateMachineEngine.startWithBusinessKey("reduceInventoryAndBalance", null,
                                                    businessKey, startParams);

    if(ExecutionStatus.SU.equals(inst.getStatus())){
        log.info("创建订单成功,saga transaction execute Succeed. XID: " + inst.getId());
        return "创建订单成功";
    }else{
        log.info("创建订单失败 ,saga transaction execute failed. XID: " + inst.getId());
        return "创建订单失败";
    }
}

public interface InventoryAction {
    boolean reduce(String businessKey, int count);
    boolean compensateReduce(String businessKey);
}


public interface BalanceAction {
    boolean reduce(String businessKey, BigDecimal amount, Map<String, Object> params);
    boolean compensateReduce(String businessKey, Map<String, Object> params);
}

```

出现的问题：

io.seata.core.exception.RmTransactionException

io.seata.core.exception.RmTransactionException: Response[TransactionException[Could not register branch into global session xid = 120.79.132.251:8091:272390470565371904 status = Rollbac]

超时时间设置太短，导致注册分支事务的时候，全局事务已经进入第二阶段

client.tm.defaultGlobalTransactionTimeout设置超时时间60000

老是序列化出错

```
com.fasterxml.jackson.databind.exc.InvalidDefinitionException: Cannot construct instance of `java.time.LocalDateTime` (no Creators, like default constructor, exist): cannot deser
 at [Source: (byte[])"{"@class":"io.seata.rm.datasource.undo.BranchUndoLog","xid":"47.106.75.115:8091:273398326018191360","branchId":273398346297651200,"sqlUndoLogs":["java.util.
    at com.fasterxml.jackson.databind.exc.InvalidDefinitionException.from(InvalidDefinitionException.java:67) ~[jackson-databind-2.11.4.jar:2.11.4]
    at com.fasterxml.jackson.databind.DeserializationContext.reportBadDefinition(DeserializationContext.java:1615) ~[jackson-databind-2.11.4.jar:2.11.4]
    at com.fasterxml.jackson.databind.DatabindContext.reportBadDefinition(DatabindContext.java:400) ~[jackson-databind-2.11.4.jar:2.11.4]
    at com.fasterxml.jackson.databind.DeserializationContext.handleMissingInstantiator(DeserializationContext.java:1077) ~[jackson-databind-2.11.4.jar:2.11.4]
    at com.fasterxml.jackson.databind.deser.BeanDeserializerBase.deserializeFromObjectUsingNonDefault(BeanDeserializerBase.java:1332) ~[jackson-databind-2.11.4.jar:2.11.4]
```

Spring Cloud Alibaba Version	Sentinel Version	Nacos Version	RocketMQ Version	Dubbo Version	Seata Version
2.2.7.RELEASE*	1.8.1	2.0.3	4.6.1	2.7.13	1.3.0
2.2.6.RELEASE	1.8.1	1.4.2	4.4.0	2.7.8	1.3.0