一、功能说明

该工具包包含spring-cloud-starter-alibaba-seata的集成, 并在该基础之上封装了对动态数据源/读写分离的支持。

背景

- 在微服务横行的当下, 分布式业务并发风险剧增;
- 急需一套相应的分布式场景解决方案保证高并发下的数据一致性。

二、工具使用

- 添加依赖: pom.xml

```
<dependency>
    <groupId>cn.gmlee</groupId>
    <artifactId>tools-seata</artifactId>
    <version>3.0.2</version>
</dependency>
```

- 应用配置: application.properties

```
\# Seata
seata.enableAutoDataSourceProxy=false
seata.txServiceGroup=SEATA_GROUP
seata.applicationId=seata-server

\# nacos registration center
seata.registry.type=nacos
seata.registry.nacos.application=seata-server
seata.registry.nacos.cluster=DEFAULT
seata.registry.nacos.group=SEATA_GROUP
seata.registry.nacos.username=nacos
seata.registry.nacos.password=nacos
seata.registry.nacos.server-addr=host:port

\# nacos config center
seata.config.type=nacos
seata.config.nacos.group=SEATA_GROUP
seata.config.nacos.username=nacos
seata.config.nacos.password=nacos
seata.config.nacos.server-addr=host:port
```

- 启用工具

> 在启动类上或其他配置类上扫描包: cn.gmlee.tools

```
@SpringBootApplication
@ComponentScan("cn.gmlee.tools")
public class XxxApp {
    public static void main(String[] args) {
        SpringApplication.run(XxxApp.class, args);
    }
}
```

- 开启全局事务: @GlobalTransactional

```
@GlobalTransactional
public void come(Tx tx) {
    // 本地仓储: 插入数据
    txMapper.insert(tx);
    // 远程调用: one服务
    AssertUtil.isOk(oneApi.come(MicroConstant.one), MicroConstant.one+": 远程服务异常");
    // 远程调用: two服务
    AssertUtil.isOk(twoApi.come(MicroConstant.two), MicroConstant.two+": 远程服务异常");
}
```

三、工具原理

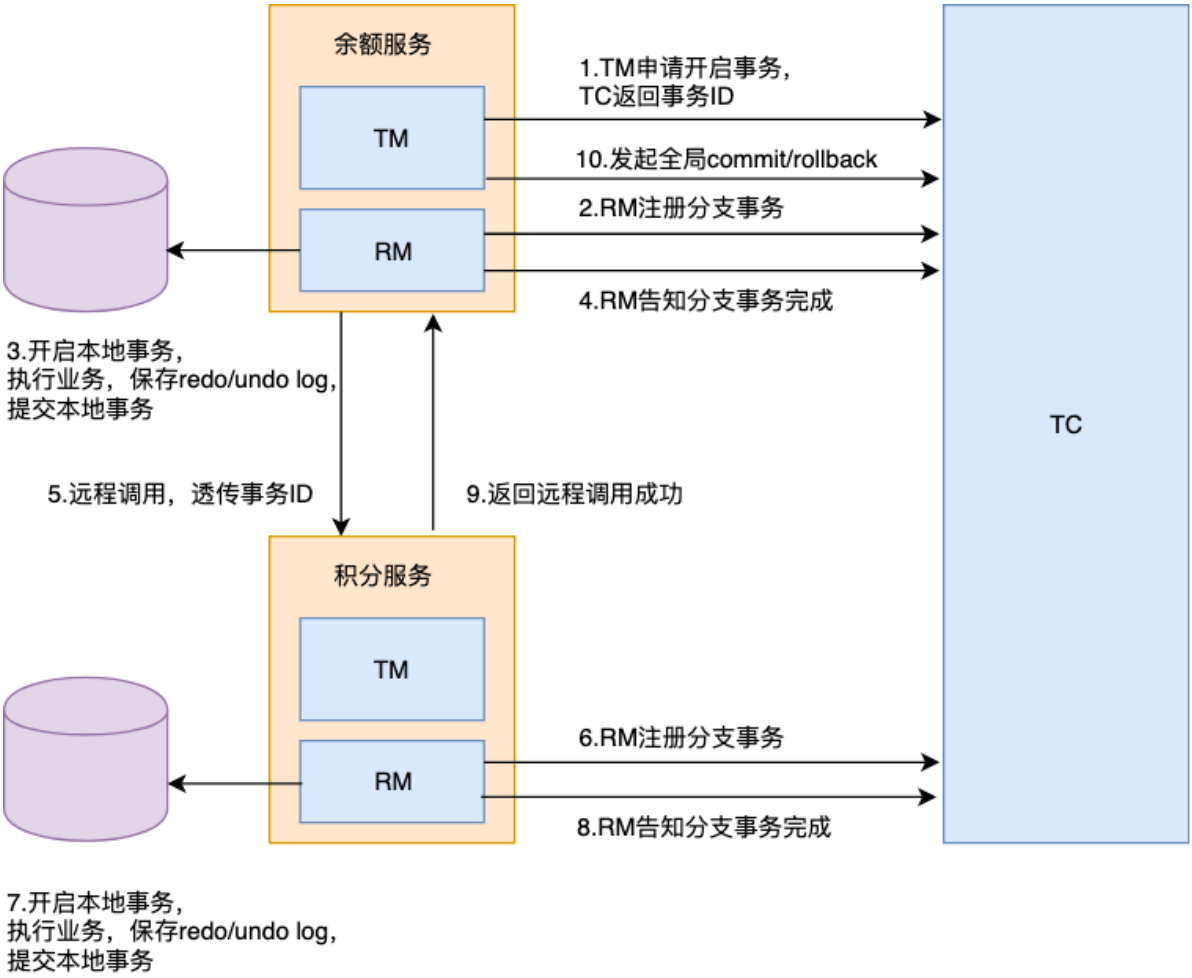
逻辑

- 基于tools-mysql创建动态数据源
- 将数据源托管给seata数据源DataSourceProxy管理 **以下以AT模式继续表述:**
- 第一阶段: 给需要全局事务的业务方法创建全局事务并提交到TC并获得资源全局独占锁

> 1. 获得全局事务独占锁后创建本地事务
> 2. 执行本地事务并与记录回滚日志(**undolog**)一并提交
> 3. 本地事务提交后即释放本地所锁和连接 (意味着后续操作均为异步)

- 第二阶段: 异步处理全局事务并释放全局独占锁

> 1. 该阶段异步处理
> 2. 提交: 如全局事务中所有分支事务正常则通过TC通知所有分支清理undolog
> 3. 回滚: 如全局事务中有任何分支事务异常、超时, 则通过TC通知其他分支根据undolog进行反向补偿



代码

1. AT模式(**推荐**): 代码无侵入, 需要全局事务的业务方法只需使用@GlobalTransactional修饰/声明
2. TCC模式: 在业务接口上使用@LocalTCC、在业务接口方法上使用@TwoPhaseBusinessAction (其方法需要传递BusinessActionContext)
3. Saga模式: 代码侵入较大, 如需参考请查阅: https://cloud.tencent.com/developer/article/1769276
4. XA模式: 二阶段阻塞提交, 性能较差, 不推荐使用, 除非有强一致性要求。

数据

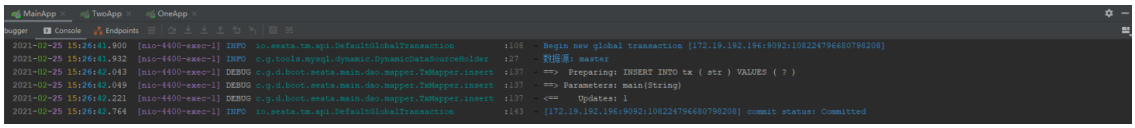
- 工具产生的数据, 均存在于db中, 可以通过配置切换到file, redis等其他存储服务
- 工具中微服务的数据库仅限: Oracle、Mysql、DB2等实现了XA协议接口的关系型数据库

四、使用示例

4.1 提交案例

- seata-main: 开启了全局事务的微服务

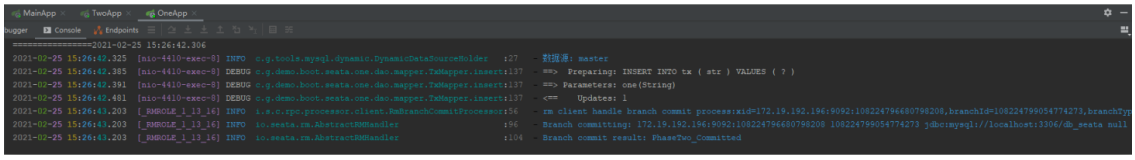
```
@GlobalTransactional
public void come(Tx tx) {
    // 本地仓储: 插入数据
    txMapper.insert(tx);
    // 远程调用: seata-one服务
    AssertUtil.isOk(oneApi.come(MicroConstant.one), MicroConstant.one+": 远程服务异常");
    // 远程调用: seata-two服务
    AssertUtil.isOk(twoApi.come(MicroConstant.two), MicroConstant.two+": 远程服务异常");
}
```



- seata-one: 分支1事务所在的微服务

> 此处省略feign部分, 完整代码请进 实验室 查看: https://github.com/Jason8080/demo/tree/master/boot-seata

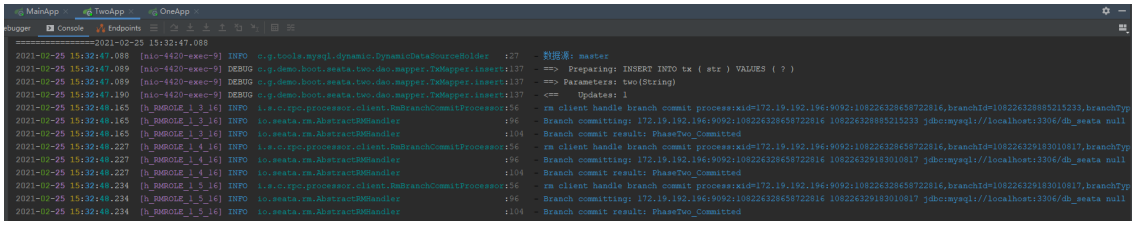
```
public void come(Tx tx) {
    txMapper.insert(tx);
}
```



- seata-two: 分支2事务所在的微服务

> 此处省略feign部分, 完整代码请进 实验室 查看: https://github.com/Jason8080/demo/tree/master/boot-seata

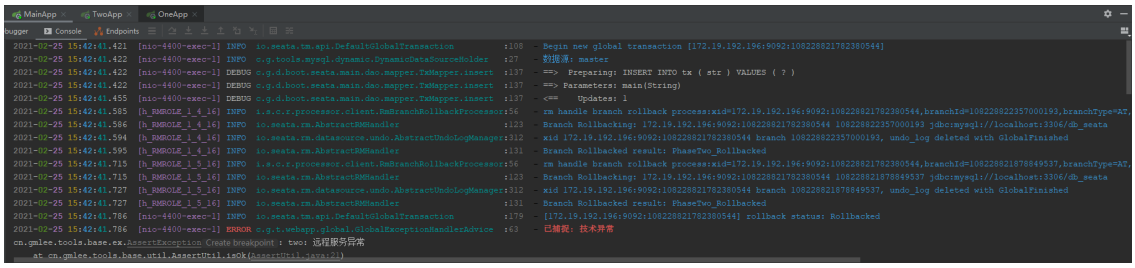
```
public void come(Tx tx) {
    txMapper.insert(tx);
}
```



4.2 回滚案例

- seata-main: 开启了全局事务的微服务

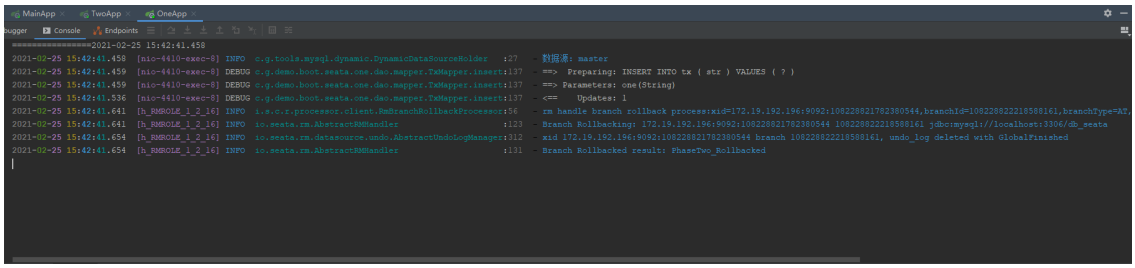
```
@GlobalTransactional
public void come(Tx tx) {
    // 本地仓储: 插入数据
    txMapper.insert(tx);
    // 远程调用: seata-one服务
    AssertUtil.isOk(oneApi.come(MicroConstant.one), MicroConstant.one+": 远程服务异常");
    // 远程调用: seata-two服务
    AssertUtil.isOk(twoApi.come(MicroConstant.two), MicroConstant.two+": 远程服务异常");
}
```



- seata-one: 分支1事务所在的微服务

> 此处省略feign部分, 完整代码请进 实验室 查看: https://github.com/Jason8080/demo/tree/master/boot-seata

```
public void come(Tx tx) {
    txMapper.insert(tx);
}
```



- seata-two: 分支2事务所在的微服务

> 此处省略feign部分, 完整代码请进 实验室 查看: https://github.com/Jason8080/demo/tree/master/boot-seata

```
public void come(Tx tx) {
    txMapper.insert(tx);
    int i = 1/0;
}
```