seata+nacos 实现 AT 模式分布式事务

作者：wenyl
原文链接：https://ld246.com/article/1635746329490
来源网站：链滴

1、AT模式简介

AT模式官网已经给出了很详细的介绍，可以直接看官网

http://seata.io/zh-cn/docs/dev/mode/at-mode.html

2、建立项目

涉及的代码过多，这里只对几个关键的步骤进行说明，完整代码可以到git上下载，AT模式在master支上，数据库脚本在script目录下

https://gitee.com/WylLoveX/seata.git

2.1、maven依赖

这里我们建立一个spring boot项目，基于2.2.5.RELEASE版本，引入seata和nacos需要的依赖。

spring-cloud-starter-alibaba-seata内部封装了seata分布式事务的XID的传递，引入直接使用，如不用这个组件，就只能自己解决XID传递的问题；

seata-spring-boot-starter的版本号和seata版本保持一致；

```
<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.2.5.RELEASE</version>
    <relativePath/> <!-- lookup parent from repository -->
</parent>
<dependencies>
```

```
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
            <version>2.2.5.RELEASE</version>
        </dependency>

        <dependency>
            <groupId>com.alibaba.cloud</groupId>
            <artifactId>spring-cloud-starter-alibaba-seata</artifactId>
            <version>2.2.5.RELEASE</version>
            <exclusions>
                <exclusion>
                    <groupId>io.seata</groupId>
                    <artifactId>seata-spring-boot-starter</artifactId>
                </exclusion>
            </exclusions>
        </dependency>
        <dependency>
            <groupId>io.seata</groupId>
            <artifactId>seata-spring-boot-starter</artifactId>
            <version>1.3.0</version>
            <exclusions>
                <exclusion>
                    <groupId>com.alibaba</groupId>
                    <artifactId>druid</artifactId>
                </exclusion>
            </exclusions>
        </dependency>

        <dependency>
            <groupId>com.alibaba</groupId>
            <artifactId>druid</artifactId>
            <version>1.1.21</version>
        </dependency>
        <dependency>
            <groupId>com.alibaba.cloud</groupId>
            <artifactId>spring-cloud-starter-alibaba-nacos-discovery</artifactId>
            <version>2.2.5.RELEASE</version>
        </dependency>
    </dependencies>
```

2.2、yml配置

先配置将服务注册到nacos，然后对seata进行配置，seata的tx-service-group的值，对应了seata事组的配置，例如，我这个服务的seata.tx-service-group的值为order_group，那就要和seata中的配对应

service.vgroupMapping.order_group=default

```
server:
  port: 8080
spring:
  datasource:
```

```
    url: jdbc:mysql://10.116.11.110:3306/order?useUnicode=true&characeterEncoding=utf-8&
erverTimezone=UTC&useSSL=false
    username: root
    password: root
    driver-class-name: com.mysql.jdbc.Driver
  application:
    name: order
  cloud:
    nacos:
      discovery:
        server-addr: 10.116.11.110:8848
logging:
  level:
    io:
      seata: debug
mybatis:
  # Mybatis配置Mapper路径
  mapper-locations: classpath:mapper/**/*.xml
seata:
  tx-service-group: order_group
  config:
    type: nacos
    nacos:
      server-addr: 10.116.11.110:8848
      group: SEATA_GROUP
```

2.3、数据源配置

然后我们需要设置mybatis使用seata的数据源代理

```
import com.alibaba.druid.pool.DruidDataSource;
import io.seata.rm.datasource.DataSourceProxy;
import org.apache.ibatis.session.SqlSessionFactory;
import org.mybatis.spring.SqlSessionFactoryBean;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.boot.context.properties.ConfigurationProperties;
import org.springframework.boot.jdbc.DataSourceBuilder;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.core.io.support.PathMatchingResourcePatternResolver;

import javax.sql.DataSource;

/**
 * @author Mr.Wen
 * @version 1.0
 * @date 2021-10-27 19:06
 */

@Configuration
public class DataSourceProxyConfig {

    @Value("${mybatis.mapper-locations}")
    private String mapperLocations;
```

```
    @Bean
    @ConfigurationProperties(prefix = "spring.datasource")
    public DataSource druidDatasource(){
        return new DruidDataSource();
    }
    @Bean
    public SqlSessionFactory sqlSessionFactory(DataSource datasource) throws Exception{
        SqlSessionFactoryBean sessionFactoryBean = new SqlSessionFactoryBean();
        sessionFactoryBean.setDataSource(new DataSourceProxy(datasource));
        sessionFactoryBean.setMapperLocations(new PathMatchingResourcePatternResolver().ge
Resources(mapperLocations));
        return sessionFactoryBean.getObject();
    }

}
```

2.4、undo_log

AT模式下，每个服务对应的库中，都要建立undo_log表

```
-- for AT mode you must to init this sql for you business database. the seata server not need it

CREATE TABLE IF NOT EXISTS `undo_log`
(
    `branch_id`     BIGINT       NOT NULL COMMENT 'branch transaction id',
    `xid`           VARCHAR(128) NOT NULL COMMENT 'global transaction id',
    `context`       VARCHAR(128) NOT NULL COMMENT 'undo_log context,such as serialization',
    `rollback_info` LONGBLOB     NOT NULL COMMENT 'rollback info',
    `log_status`    INT(11)      NOT NULL COMMENT '0:normal status,1:defense status',
    `log_created`   DATETIME(6)  NOT NULL COMMENT 'create datetime',
    `log_modified`  DATETIME(6)  NOT NULL COMMENT 'modify datetime',
    UNIQUE KEY `ux_undo_log` (`xid`, `branch_id`)
) ENGINE = InnoDB
  AUTO_INCREMENT = 1
  DEFAULT CHARSET = utf8 COMMENT ='AT transaction mode undo table';
```

2.5、项目启动

启动需要几个注解，因为我们自己建立了数据源，所以就直接剔除spring boot的数据源自动配置

```
/**
 * @author Mr.Wen
 * @version 1.0
 * @date 2021-10-21 16:59
 */
@SpringBootApplication(exclude = DataSourceAutoConfiguration.class)
@EnableDiscoveryClient
@Import(DataSourceProxyConfig.class)
public class OrderApplication {
    public static void main(String[] args){
        SpringApplication.run(OrderApplication.class);
```

```
    }
}
```

3、使用

事务发起端添加全局事务注解@GlobalTransactional，同时注意不要捕获异常，否则事务不会回滚

事务发起端：

```
@Override
@GlobalTransactional
public int insert(OrderForGoods orderForGoods){
    int ret = orderForGoodsMapper.insert(orderForGoods);
    productServiceApi.reduceStock( id: "1", num: 1);
    bankServiceApi.reduceAccount( id: "1", num: 10);
    return ret;
}
```

事务参与：

```
@Service
public class BankUserServiceImpl implements BankUserService {
    @Resource
    private BankUserMapper bankUserMapper;

    @Override
    @Transactional
    public int reduceAccount(String id,Integer num) {
        int i=1/0;
        return bankUserMapper.reduceAccount(id,num);
    }
}
```

```
@Service
public class ProductServiceImpl implements ProductService {
    @Resource
    private ProductMapper productMapper;

    @Override
    @Transactional(rollbackFor = Exception.class)
    public int reduceStock(String id, Integer num) {
        return productMapper.reduceStock(id,num);
    }
}
```

、

4、注意

这个模式的核心是undo_log表，这个表记录的操作的数据在事务提交前的状态和事务提交后的状态
本地事务会将undo_log和我们的数据库操作一起提交，所以，我们在3中的全局事务发起者中打个断
，会发现事务已经是提交了的，出错回滚的话，其实并不是我们理解的rollback，因为我们修改数据

事务已经提交了，seata的回滚就是读取undo_log表的操作日志，找到对应数据的事务提交前的记录将他重新设置为事务提交前的样子。全局事务回滚或者提交后，会删除undo_log中对应的记录。

分布式事务如果回滚失败，可以检查下XID是否传递成功。

原文链接：seata+nacos 实现 AT 模式分布式事务