链滴

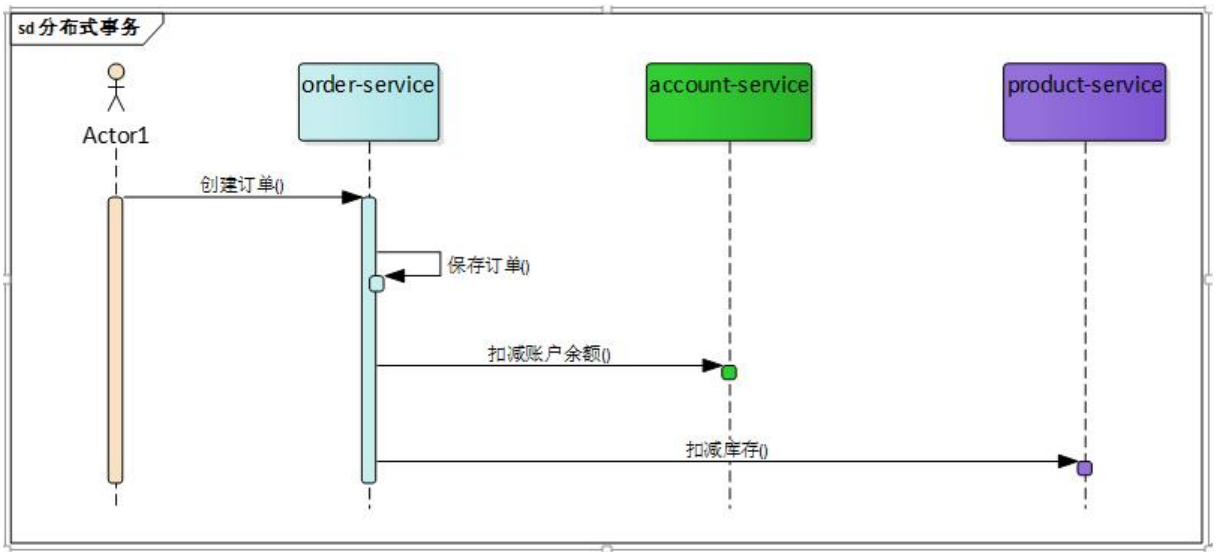
SpringCloud Alibaba 微服务实战七 - 分布式事务

作者：jianzh5
原文链接：https://ld246.com/article/1576734207438
来源网站：链滴
许可协议：署名-相同方式共享 4.0 国际 (CC BY-SA 4.0)

导读：本篇作为SpringCloud Alibaba微服务实战系列的第七篇，主要内容是使用Seata解决分布式务问题。系列文章，欢迎持续关注。

场景说明

订单服务order-service需要对外提供创建订单的接口，创建订单的业务逻辑如下：



先调用本地的orderService保存订单操作，然后通过feign调用远程的accout-service进行账户余额扣，最后再通过feign调用远程的product-service进行库存扣减操作。

关键的逻辑代码如下：

- OrderController对外提供创建订单的接口

```
@PostMapping("/order/create")
```

```
public ResultData<OrderDTO> create(@RequestBody OrderDTO orderDTO){
    log.info("create order:{}",orderDTO);
    orderDTO.setOrderNo(UUID.randomUUID().toString());
    orderDTO.setAmount(orderDTO.getPrice().multiply(new BigDecimal(orderDTO.getCount())))

    orderService.createOrder(orderDTO);
    return ResultData.success("create order success");
}
```

- OrderServiceImpl负责处理创建订单的业务逻辑

```
@Transactional(rollbackFor = RuntimeException.class)
@Override
public void createOrder(OrderDTO orderDTO) {
    Order order = new Order();
    BeanUtils.copyProperties(orderDTO,order);
    //本地存储Order
    this.saveOrder(order);
    //库存扣减
    productFeign.deduct(orderDTO.getProductCode(),order.getCount());
    //账户余额扣减
    accountFeign.reduce(orderDTO.getAccountCode(), orderDTO.getAmount());
}

@Transactional(rollbackFor = RuntimeException.class)
void saveOrder(Order order) {
    orderMapper.insert(order);
}
```

本地先保存，然后调用两个远程服务进行扣减操作。

- AccountServiceImpl扣减账户余额

```
@Transactional(rollbackFor = RuntimeException.class)
@Override
public void reduceAccount(String accountCode, BigDecimal amount) {
    Account account = accountMapper.selectByCode(accountCode);
    if(null == account){
        throw new RuntimeException("can't reduce amount,account is null");
    }
    BigDecimal subAmount = account.getAmount().subtract(amount);
    if(subAmount.compareTo(BigDecimal.ZERO) < 0){
        throw new RuntimeException("can't reduce amount,account'amount is less than reduce
mount");
    }
    account.setAmount(subAmount);
    accountMapper.updateById(account);
}
```

做些简单的校验，当账户余额不足的时候不允许扣减操作。

- ProductServiceImpl扣减产品库存

```
@Transactional(rollbackFor = RuntimeException.class)
```

```
@Override
public void deduct(String productCode, Integer deductCount) {
    Product product = productMapper.selectByCode(productCode);
    if(null == product){
        throw new RuntimeException("can't deduct product,product is null");
    }
    int surplus = product.getCount() - deductCount;
    if(surplus < 0){
        throw new RuntimeException("can't deduct product,product's count is less than deduct
ount");
    }
    product.setCount(surplus);
    productMapper.updateById(product);
}
```

做些简单的校验，当产品库存不足时不允许扣减操作。

order-service、product-service、account-service分属不同的服务,当其中一个服务抛出异常无法交时就会导致分布式事务，如当使用feign调用account-service执行扣减账户余额时，account-servie校验账户余额不足抛出异常，但是order-service的保存操作不会回滚；或者是前两步执行成功但是poduct-service校验不通过前面的操作也不会回滚，这就导致了数据不一致，也就是分布式事务问题！

Seata解决方案

在Springcloud Alibaba体系中使用Seata作为分布式事务解决方案，大家可以访问seata官网去了解情。

这次我们先使用Seata的file配置解决上面出现的问题，后面再来对其改造。

下载安装Seata Server。

- 从 Release 页面下载Seata Server
- 下载完成后直接启动Server端服务。

在Linux/Mac下

```
$ sh ./bin/seata-server.sh
```

在Windows下

```
bin\seata-server.bat
```

引入seata组件

```
<dependency>
    <groupId>com.alibaba.cloud</groupId>
    <artifactId>spring-cloud-alibaba-seata</artifactId>
</dependency>
```

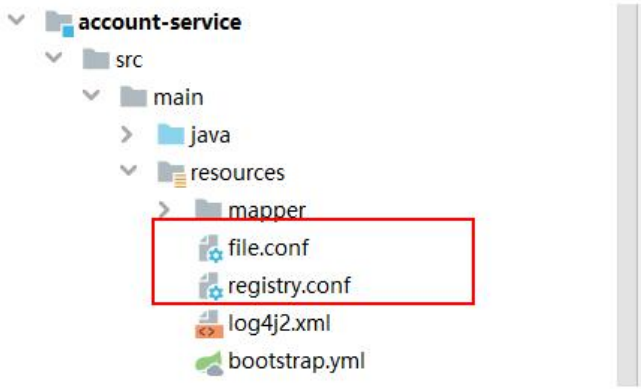
在配置文件中增加seata配置

```
spring:
  cloud:
    alibaba:
```

```
  seata:
    tx-service-group: ${spring.application.name}-seata
```

Seata Client 配置修改

- 将Seata Server 配置目录下的 registry.conf、file.conf 2个文件拷贝到微服务中的resources文件下



- 修改拷贝后的registry.conf

```
registry{
  type = "file"

  file {
    name = "file.conf"
  }
}

config{
  type = "file"

  file {
    name = "file.conf"
  }
}
```

- 修改file.conf

```
service {
  #vgroup->rgroup
  vgroup_mapping.account-service-seata = "default"
  default.grouplist = "localhost:8091"
  enableDegrade = false
  disable = false
  max.commit.retry.timeout = "-1"
  max.rollback.retry.timeout = "-1"
}

support {
  ## spring
  spring {
    # auto proxy the DataSource bean
    datasource.autoproxy = true
  }
}
```

主要修改如下三处：

service.vgroup_mapping.后面的值修改为配置文件spring.cloud.alibaba.seata.tx-service-group属性

service.default.grouplist=修改为Seata Server的ip:端口

support.spring.datasource.autoproxy的值修改为true，开启datasource自动代理

生成undo_log表

在微服务的业务库下执行如下语句，生成undo_log表

```
-- 此脚本必须初始化在你当前的业务数据库中，用于AT 模式XID记录。与server端无关（注：业务数库）
drop table `undo_log`;
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,
  `ext` varchar(100) DEFAULT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `ux_undo_log` (`xid`,`branch_id`)
) ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT CHARSET=utf8;
```

开启全局事务

在分布式事务方法入口添加注解@GlobalTransactional，这里只需要在createOrder方法上添加此注即可！

```
@GlobalTransactional(name = "TX_ORDER_CREATE")
@Override
public void createOrder(OrderDTO orderDTO) {
    Order order = new Order();
    BeanUtils.copyProperties(orderDTO,order);
    //本地存储Order
    this.saveOrder(order);
    log.info("ORDER XID is: {}", RootContext.getXID());
    //账户余额扣减
    accountFeign.reduce(orderDTO.getAccountCode(), orderDTO.getAmount());
    //库存扣减
    productFeign.deduct(orderDTO.getProductCode(),orderDTO.getCount());
}
```

在代码中可以使用RootContext.getXID()获取全局xid

启动服务

服务正常启动后在Seata Server控制台可以看到注册信息

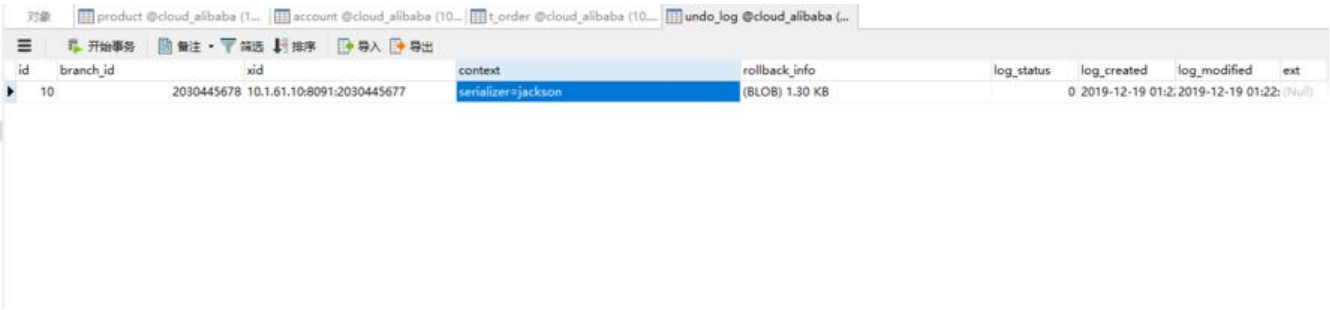
```
2019-12-19 09:29:11.236 INFO [NettyServerNIOWorker_7_8]io.seata.core.rpc.DefaultServerMessageListenerImpl.onRegTmMessage:133
-checkAuth for client:127.0.0.1:52887 vgroup:order-service-seata ok
2019-12-19 09:29:11.245 INFO [ServerHandlerThread_53_500]io.seata.core.rpc.DefaultServerMessageListenerImpl.onRegRmMessage:11
6 -rm register success,message:RegisterRMRequest{resourceIds='jdbc:mysql://10.0.10.48:3306/cloud_alibaba', applicationId='ord
er-service', transactionServiceGroup='order-service-seata'},channel:[id: 0x236f5248, L:/127.0.0.1:8091 - R:/127.0.0.1:52888]
2019-12-19 09:29:17.471 INFO [NettyServerNIOWorker_1_8]io.seata.core.rpc.DefaultServerMessageListenerImpl.onRegTmMessage:133
-checkAuth for client:127.0.0.1:52918 vgroup:product-service-seata ok
2019-12-19 09:29:17.477 INFO [ServerHandlerThread_54_500]io.seata.core.rpc.DefaultServerMessageListenerImpl.onRegRmMessage:11
6 -rm register success,message:RegisterRMRequest{resourceIds='jdbc:mysql://10.0.10.48:3306/cloud_alibaba', applicationId='pro
duct-service', transactionServiceGroup='product-service-seata'},channel:[id: 0x5d818a17, L:/127.0.0.1:8091 - R:/127.0.0.1:529
17]
2019-12-19 09:29:22.180 INFO [ServerHandlerThread_55_500]io.seata.core.rpc.DefaultServerMessageListenerImpl.onRegRmMessage:11
6 -rm register success,message:RegisterRMRequest{resourceIds='jdbc:mysql://10.0.10.48:3306/cloud_alibaba', applicationId='acc
ount-service', transactionServiceGroup='account-service-seata'},channel:[id: 0x13a8abd7, L:/127.0.0.1:8091 - R:/127.0.0.1:529
49]
2019-12-19 09:29:24.515 INFO [NettyServerNIOWorker_4_8]io.seata.core.rpc.DefaultServerMessageListenerImpl.onRegTmMessage:133
-checkAuth for client:127.0.0.1:52960 vgroup:account-service-seata ok
```

接口测试

改造完成后对接口进行测试，如果其他服务抛出异常会看到如下错误日志，再结合数据库数据观察是正常回滚

```
2019-12-19 09:15:39,500 INFO  OrderServiceImpl:44 - ORDER XID is: 10.1.61.10:8091:2030445670
2019-12-19 09:15:39,565 INFO  RmMessageListener:66 - onMessage:xid=10.1.61.10:8091:2030445670,branchId=2030445671,branchType=AT,
 resourceId=jdbc:mysql://10.0.10.48:3306/cloud_alibaba,applicationData=null
2019-12-19 09:15:39,566 INFO  AbstractRMHandler:122 - Branch Rollbacking: 10.1.61.10:8091:2030445670 2030445671 jdbc:mysql://10.0.10
.48:3306/cloud_alibaba
2019-12-19 09:15:39,577 INFO  AbstractUndoLogManager:332 - xid 10.1.61.10:8091:2030445670 branch 2030445671, undo_log deleted with
 GlobalFinished
2019-12-19 09:15:39,578 INFO  AbstractRMHandler:130 - Branch Rollbacked result: PhaseTwo_Rollbacked
2019-12-19 09:15:39,581 INFO  DefaultGlobalTransaction:185 - [10.1.61.10:8091:2030445670] rollback status:Rollbacked
```

执行过程中我们通过debug可以发现undo_log表会不断插入数据，在执行后又会被删除。



通过上面几步我们使用Seata实现了分布式事务，保证了数据的一致性，最后说一句Seata真香，你们不要感受一下。

至此本期的“SpringCloud Alibaba微服务实战七 - 分布式事务”篇也就该结束啦，咱们下期有缘再！

再见之前让我在求一波关注吧，O(∩_∩)O哈哈~！

系列文章

- SpringCloud Alibaba微服务实战六 - 配置隔离
- SpringCloud Alibaba微服务实战五 - 限流熔断
- SpringCloud Alibaba微服务实战四 - 版本管理
- SpringCloud Alibaba微服务实战三 - 服务调用
- SpringCloud Alibaba微服务实战二 - 服务注册
- SpringCloud Alibaba微服务实战一 - 基础环境准备