在分布式消息队列中，目前唯一提供**完整的**事务消息的，只有 RocketMQ 。关于这一点，还是可以鼓吹下的。

可能会有胖友怒喷艿艿，RabbitMQ 和 Kafka 也有事务消息啊，也支持发送事务消息的发送，以及后续的事务消息的 commit提交或 rollbackc 回滚。但是要考虑一个极端的情况，在**本地数据库事务已经提交**的时时候，如果因为网络原因，又或者崩溃等等意外，导致事务消息没有被 commit ，最终导致这条事务消息丢失，分布式事务出现问题。

相比来说，RocketMQ 提供事务回查机制，如果应用超过一定时长未 commit 或 rollback 这条事务消息，RocketMQ 会主动回查应用，询问这条事务消息是 commit 还是 rollback ，从而实现事务消息的状态最终能够被 commit 或是 rollback ，达到最终事务的一致性。

这也是为什么艿艿在上面专门加粗“**完整的**”三个字的原因。可能上述的描述，对于绝大多数没有了解过分布式事务的胖友，会比较陌生，所以推荐阅读如下文章：

* [《阿里云消息队列 MQ —— 事务消息》](https://links.jianshu.com/go?to=https%3A%2F%2Fwww.alibabacloud.com%2Fhelp%2Fzh%2Fdoc-detail%2F43348.htm)

虽然说 RabbitMQ、Kafka 并未提供完整的事务消息，但是社区里，已经基于它们之上拓展，提供了事务回查的功能。例如说：[Myth](https://links.jianshu.com/go?to=https%3A%2F%2Fdromara.org%2Fzh-cn%2Fdocs%2Fmyth%2Findex.html) ，采用消息队列解决分布式事务的开源框架, 基于 Java 语言来开发（JDK1.8），支持 Dubbo，Spring Cloud，Motan 等 RPC 框架进行分布式事务。

下面，我们来搭建一个 RocketMQ 定时消息的使用示例。考虑方便，我们直接复用[快速入门]小节的项目，复制原producer项目成 [sca-stream-rocketmq-producer-transaction]发送**事务消息**，继续使用 [sca-stream-rocketmq-consumer] 消费消息。

**8.1 复制项目**

从 [sca-stream-rocketmq-producer] 复制出 [sca-stream-rocketmq-producer-transaction]来发送**事务消息**。

**8.2 配置文件**

修改 [application.yml](https://links.jianshu.com/go?to=https%3A%2F%2Fgithub.com%2FYunaiV%2FSpringBoot-Labs%2Fblob%2Fmaster%2Flabx-06-spring-cloud-stream-rocketmq%2Flabx-06-sca-stream-rocketmq-producer-transaction%2Fsrc%2Fmain%2Fresources%2Fapplication.yml) 配置文件，添加 transactional 配置项为 true，设置 Producer 发送**事务**消息。完整配置如下：

spring:

application:

name: stream-rocketmq-producer-application

cloud:

# Spring Cloud Stream 配置项，对应 BindingServiceProperties 类

stream:

# Binding 配置项，对应 BindingProperties Map

bindings:

erbadagang-output:

destination: ERBADAGANG-TOPIC-01 # 目的地。这里使用 RocketMQ Topic

content-type: application/json # 内容格式。这里使用 JSON

trek-output:

destination: TREK-TOPIC-01 # 目的地。这里使用 RocketMQ Topic

content-type: application/json # 内容格式。这里使用 JSON

# Spring Cloud Stream RocketMQ 配置项

rocketmq:

# RocketMQ Binder 配置项，对应 RocketMQBinderConfigurationProperties 类

binder:

name-server: 101.133.227.13:9876 # RocketMQ Namesrv 地址

# RocketMQ 自定义 Binding 配置项，对应 RocketMQBindingProperties Map

bindings:

erbadagang-output:

# RocketMQ Producer 配置项，对应 RocketMQProducerProperties 类

producer:

group: test # 生产者分组

sync: true # 是否同步发送消息，默认为 false 异步。

transactional: true # 是否发送事务消息，默认为 false。

server:

port: 18080

**8.3 Demo01Controller**

修改 [Demo01Controller]类，增加发送**事务**消息的 HTTP 接口。代码如下：

/\*\*

\* 事务消息

\*

\* @return

\*/

@GetMapping("/send\_transaction")

public boolean sendTransaction() {

// 创建 Message

Demo01Message message = new Demo01Message()

.setId(new Random().nextInt());

// 创建 Spring Message 对象

Args args = new Args().setArgs1(1).setArgs2("2");

Message<Demo01Message> springMessage = MessageBuilder.withPayload(message)

.setHeader("args", JSON.toJSONString(args)) // <X>

.build();

// 发送消息

return mySource.erbadagangOutput().send(springMessage);

}

public static class Args {

private Integer args1;

private String args2;

public Integer getArgs1() {

return args1;

}

public Args setArgs1(Integer args1) {

this.args1 = args1;

return this;

}

public String getArgs2() {

return args2;

}

public Args setArgs2(String args2) {

this.args2 = args2;

return this;

}

@Override

public String toString() {

return "Args{" +

"args1=" + args1 +

", args2='" + args2 + '\'' +

'}';

}

}

因为 Spring Cloud Stream 在设计时，并没有考虑事务消息，所以我们只好在 <X> 处，通过 Header 传递参数。

又因为 Header 后续会被转换成 String 类型，导致我们无法获得正确的真实的原始参数，所以这里我们先使用 JSON 将 args 参数序列化成字符串，这样后续我们可以使用 JSON 反序列化回来。

**8.4 TransactionListenerImpl**

创建 [TransactionListenerImpl]类，实现 MQ 事务的监听。代码如下：

package com.erbadagang.springcloudalibaba.stream.rocketmq.producer.message;

import com.alibaba.fastjson.JSON;

import com.erbadagang.springcloudalibaba.stream.rocketmq.producer.controller.Demo01Controller;

import org.apache.rocketmq.spring.annotation.RocketMQTransactionListener;

import org.apache.rocketmq.spring.core.RocketMQLocalTransactionListener;

import org.apache.rocketmq.spring.core.RocketMQLocalTransactionState;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import org.springframework.messaging.Message;

/\*\*

\* @description 实现 MQ 事务的监听。

\* @ClassName: TransactionListenerImpl

\* @author: 郭秀志 jbcode@126.com

\* @date: 2020/8/7 9:21

\* @Copyright:

\*/

@RocketMQTransactionListener(txProducerGroup = "test")

public class TransactionListenerImpl implements RocketMQLocalTransactionListener {

private Logger logger = LoggerFactory.getLogger(getClass());

@Override

public RocketMQLocalTransactionState executeLocalTransaction(Message msg, Object arg) {

// 从消息 Header 中解析到 args 参数，并使用 JSON 反序列化

Demo01Controller.Args args = JSON.parseObject(msg.getHeaders().get("args", String.class),

Demo01Controller.Args.class);

// ... local transaction process, return rollback, commit or unknown

logger.info("[executeLocalTransaction][执行本地事务，消息：{} args：{}]", msg, args);

return RocketMQLocalTransactionState.UNKNOWN;

}

@Override

public RocketMQLocalTransactionState checkLocalTransaction(Message msg) {

// ... check local transaction status and return rollback, commit or unknown

logger.info("[checkLocalTransaction][回查消息：{}]", msg);

return RocketMQLocalTransactionState.COMMIT;

}

}

① 在类上，添加 [@RocketMQTransactionListener](https://links.jianshu.com/go?to=https%3A%2F%2Fgithub.com%2Fapache%2Frocketmq-spring%2Fblob%2Fmaster%2Frocketmq-spring-boot%2Fsrc%2Fmain%2Fjava%2Forg%2Fapache%2Frocketmq%2Fspring%2Fannotation%2FRocketMQTransactionListener.java) 注解，声明监听器的是生产者分组是 "test" 的 Producer 发送的事务消息。因为 RocketMQ 是回查（请求）指定指定生产分组下的 Producer，从而获得事务消息的状态，所以一定要正确设置。

② 实现 [RocketMQLocalTransactionListener](https://links.jianshu.com/go?to=https%3A%2F%2Fgithub.com%2Fapache%2Frocketmq%2Fblob%2Fmaster%2Fclient%2Fsrc%2Fmain%2Fjava%2Forg%2Fapache%2Frocketmq%2Fclient%2Fproducer%2FTransactionListener.java) 接口，实现执行本地事务和检查本地事务的方法。

③ 实现 #executeLocalTransaction(...) 方法，实现执行本地事务。

* 注意，这是一个**模板方法**。在调用这个方法之前，Spring Cloud Alibaba Stream RocketMQ 已经使用 Producer 发送了一条**事务**消息，本方法里面我们使用消息内容进行本地数据库事务操作。然后根据该方法执行的返回的 [RocketMQLocalTransactionState](https://links.jianshu.com/go?to=https%3A%2F%2Fgithub.com%2Fapache%2Frocketmq-spring%2Fblob%2F3f89080df8f797cad0d1f9eb8badb5050b09a553%2Frocketmq-spring-boot%2Fsrc%2Fmain%2Fjava%2Forg%2Fapache%2Frocketmq%2Fspring%2Fcore%2FRocketMQLocalTransactionState.java) 结果，提交还是回滚该事务消息。
* 这里，我们为了模拟需要 RocketMQ 回查 Producer 来获得事务消息的状态，所以返回了 RocketMQLocalTransactionState.UNKNOWN——未知状态。

④ 实现 #checkLocalTransaction(...) 方法，检查本地事务。

* 在事务消息长事件未被提交或回滚时，RocketMQ 会回查事务消息对应的生产者分组下的 Producer ，获得事务消息的状态。此时，该方法就会被调用。
* 这里，我们直接返回 RocketMQLocalTransactionState.COMMIT 提交状态。

一般来说，有两种方式实现本地事务回查时，返回事务消息的状态。

**第一种**，通过 msg 消息，获得某个业务上的标识或者编号，然后去数据库中查询业务记录，从而判断该事务消息的状态是提交还是回滚。

**第二种**，记录 msg 的事务编号，与事务状态到数据库中。

* 第一步，在 #executeLocalTransaction(...) 方法中，先存储一条 id 为 msg 的事务编号，状态为 RocketMQLocalTransactionState.UNKNOWN 的记录。
* 第二步，调用带有**事务的**业务 Service 的方法。在该 Service 方法中，在逻辑都执行成功的情况下，更新 id 为 msg 的事务编号，状态变更为 RocketMQLocalTransactionState.COMMIT 。这样，我们就可以伴随这个事务的提交，更新 id 为 msg 的事务编号的记录的状为 RocketMQLocalTransactionState.COMMIT ，美滋滋。。
* 第三步，要以 try-catch 的方式，调用业务 Service 的方法。如此，如果发生异常，回滚事务的时候，可以在 catch 中，更新 id 为 msg 的事务编号的记录的状态为 RocketMQLocalTransactionState.ROLLBACK 。😭 极端情况下，可能更新失败，则打印 error 日志，告警知道，人工介入。
* 如此三步之后，我们在 #executeLocalTransaction(...) 方法中，就可以通过查找数据库，id 为 msg 的事务编号的记录的状态，然后返回。

**相比来说**，倾向第一种，实现更加简单通用，对于业务开发者，更加友好。和有几个朋友沟通了下，但他们是采用第二种。

**8.5 简单测试**

① 执行 **Consumer**Application，启动**消费者**的实例。  
② 执行 **Producer**Application，启动**生产者**的实例。

之后，请求 [http://127.0.0.1:18080/demo01/send\_transaction](https://links.jianshu.com/go?to=http%3A%2F%2F127.0.0.1%3A18080%2Fdemo01%2Fsend_transaction) 接口，发送**事务**消息。IDEA 控制台输出日志如下：

// ProduerApplication 控制台

// ### TransactionListenerImpl 执行 executeLocalTransaction 方法，先执行本地事务的逻辑

2020-08-07 09:53:13.715 INFO 5656 --- [io-18080-exec-5] c.e.s.s.r.p.m.TransactionListenerImpl : [executeLocalTransaction][执行本地事务，消息：GenericMessage [payload=byte[16], headers={args={"args1":1,"args2":"erbadagang神车"}, rocketmq\_TOPIC=ERBADAGANG-TOPIC-01, rocketmq\_FLAG=0, rocketmq\_TRANSACTION\_ID=C0A82B7C161818B4AAC2210545790002, id=06cfec82-f118-78df-5f8c-c6d273b69581, contentType=application/json, timestamp=1596765193715}] args：Args{args1=1, args2='erbadagang神车'}]

// ### Producer 发送事务消息成功，但是因为 executeLocalTransaction 方法返回的是 UNKOWN 状态，所以事务消息并未提交或者回滚

// ### RocketMQ Broker 在发送事务消息 38 秒后，发现事务消息还未提交或是回滚，所以回查 Producer 。此时，checkLocalTransaction 方法返回 COMMIT ，所以该事务消息被提交。

2020-08-07 09:53:51.368 INFO 5656 --- [pool-1-thread-1] c.e.s.s.r.p.m.TransactionListenerImpl : [checkLocalTransaction][回查消息：GenericMessage [payload=byte[16], headers={rocketmq\_QUEUE\_ID=3, TRANSACTION\_CHECK\_TIMES=1, rocketmq\_BORN\_TIMESTAMP=1596765193593, args={"args1":1,"args2":"erbadagang神车"}, rocketmq\_TOPIC=ERBADAGANG-TOPIC-01, rocketmq\_FLAG=0, rocketmq\_MESSAGE\_ID=6585E30D00002A9F000000000005005F, rocketmq\_TRANSACTION\_ID=C0A82B7C161818B4AAC2210545790002, rocketmq\_SYS\_FLAG=0, id=e3304269-c7da-b3d5-864a-2c0fb8a713e5, CLUSTER=DefaultCluster, rocketmq\_BORN\_HOST=103.3.96.229, contentType=application/json, timestamp=1596765231368}]]

// ConsumerApplication 控制台

// ### 事务消息被提交，所以该消息被 Consumer 消费

2020-08-07 09:53:51.602 INFO 8832 --- [MessageThread\_1] c.e.s.s.r.c.listener.Demo01Consumer

作者：梅西爱骑车  
链接：https://www.jianshu.com/p/3dca1dabda45  
来源：简书  
著作权归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。