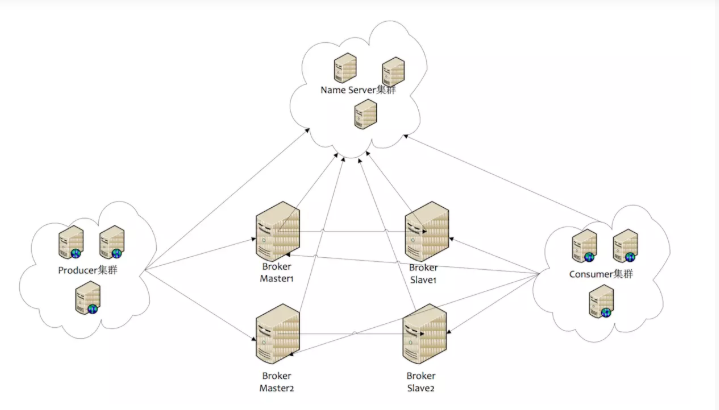
**1. RocketMQ集群部署结构**



#### 1) Name Server

Name Server是一个几乎无状态节点，可集群部署，节点之间无任何信息步。

#### 2) Broker

Broker部署相对复杂，Broker分为Master与Slave，一个Master可以对应多个Slave，但是一个Slave只能对应一个Master，Master与Slave的对应关系通过指定相同的Broker Name，不同的Broker Id来定义，BrokerId为0表示Master，非0表示Slave。Master也可以部署多个。

每个Broker与Name Server集群中的所有节点建立长连接，定时(每隔30s)注册Topic信息到所有Name Server。Name Server定时(每隔10s)扫描所有存活broker的连接，如果Name Server超过2分钟没有收到心跳，则Name Server断开与Broker的连接。

#### 3) Producer

Producer与Name Server集群中的其中一个节点(随机选择)建立长连接，定期从Name Server取Topic路由信息，并向提供Topic服务的Master建立长连接，且定时向Master发送心跳。Producer完全无状态，可集群部署。

Producer每隔30s（由ClientConfig的pollNameServerInterval）从Name server获取所有topic队列的最新情况，这意味着如果Broker不可用，Producer最多30s能够感知，在此期间内发往Broker的所有消息都会失败。

Producer每隔30s（由ClientConfig中heartbeatBrokerInterval决定）向所有关联的broker发送心跳，Broker每隔10s中扫描所有存活的连接，如果Broker在2分钟内没有收到心跳数据，则关闭与Producer的连接。

#### 4) Consumer

Consumer与Name Server集群中的其中一个节点(随机选择)建立长连接，定期从Name Server取Topic路由信息，并向提供Topic服务的Master、Slave建立长连接，且定时向Master、Slave发送心跳。Consumer既可以从Master订阅消息，也可以从Slave订阅消息，订阅规则由Broker配置决定。

Consumer每隔30s从Name server获取topic的最新队列情况，这意味着Broker不可用时，Consumer最多最需要30s才能感知。

Consumer每隔30s（由ClientConfig中heartbeatBrokerInterval决定）向所有关联的broker发送心跳，Broker每隔10s扫描所有存活的连接，若某个连接2分钟内没有发送心跳数据，则关闭连接；并向该Consumer Group的所有Consumer发出通知，Group内的Consumer重新分配队列，然后继续消费。

当Consumer得到master宕机通知后，转向slave消费，slave不能保证master的消息100%都同步过来了，因此会有少量的消息丢失。但是一旦master恢复，未同步过去的消息会被最终消费掉。

消费者对列是消费者连接之后（或者之前有连接过）才创建的。我们将原生的消费者标识由 {IP}@{消费者group}扩展为 {IP}@{消费者group}{topic}{tag}，（例如[xxx.xxx.xxx.xxx@mqtest\_producer-group\_2m2sTest\_tag-zyk](http://xxx.xxx.xxx.xxx@mqtest_producer-group_2m2sTest_tag-zyk" \t "_blank)）。任何一个元素不同，都认为是不同的消费端，每个消费端会拥有一份自己消费对列（默认是broker对列数量\*broker数量）。新挂载的消费者对列中拥有commitlog中的所有数据。

## 三、 Rocketmq如何支持分布式事务消息

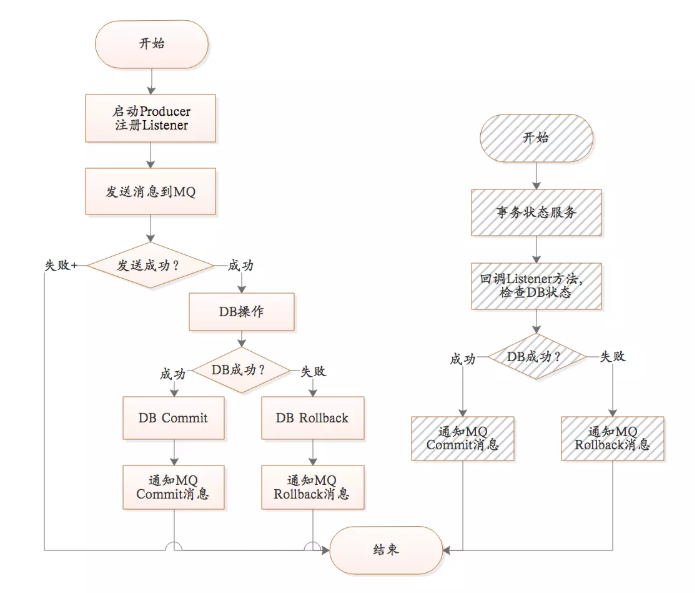
#### 场景

A（存在DB操作）、B（存在DB操作）两方需要保证分布式事务一致性，通过引入中间层MQ，A和MQ保持事务一致性（异常情况下通过MQ反查A接口实现check），B和MQ保证事务一致（通过重试），从而达到最终事务一致性。

**原理：大事务 = 小事务 + 异步**

#### 1. MQ与DB一致性原理（两方事务）

**流程图**



上图是RocketMQ提供的保证MQ消息、DB事务一致性的方案。

MQ消息、DB操作一致性方案：

1)发送消息到MQ服务器，此时消息状态为SEND\_OK。此消息为consumer不可见。

2)执行DB操作；DB执行成功Commit DB操作，DB执行失败Rollback DB操作。

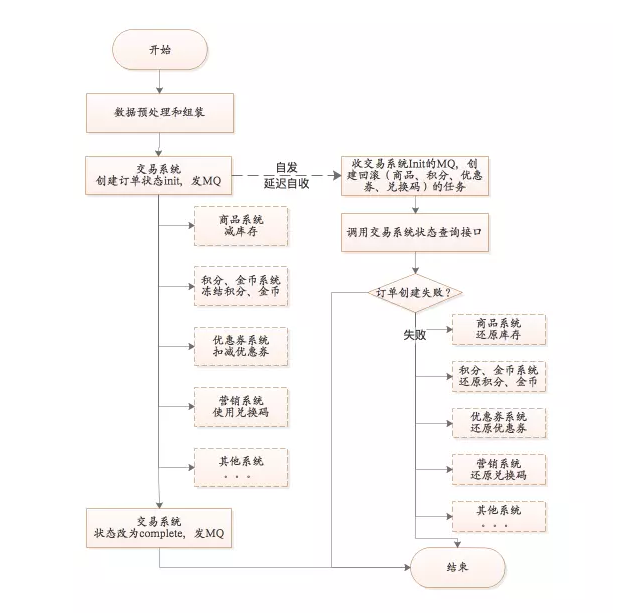
3)如果DB执行成功，回复MQ服务器，将状态为COMMIT\_MESSAGE；如果DB执行失败，回复MQ服务器，将状态改为ROLLBACK\_MESSAGE。注意此过程有可能失败。

4)MQ内部提供一个名为“事务状态服务”的服务，此服务会检查事务消息的状态，如果发现消息未COMMIT，则通过Producer启动时注册的TransactionCheckListener来回调业务系统，业务系统在checkLocalTransactionState方法中检查DB事务状态，如果成功，则回复COMMIT\_MESSAGE，否则回复ROLLBACK\_MESSAGE。

说明：上面以DB为例，其实此处可以是任何业务或者数据源。 以上SEND\_OK、COMMIT\_MESSAGE、ROLLBACK\_MESSAGE均是client jar提供的状态，在MQ服务器内部是一个数字

TransactionCheckListener 是在消息的commit或者rollback消息丢失的情况下才会回调（上图中灰色部分）。这种消息丢失只存在于断网或者rocketmq集群挂了的情况下。当rocketmq集群挂了，如果采用异步刷盘，存在1s内数据丢失风险，异步刷盘场景下保障事务没有意义。所以如果要核心业务用Rocketmq解决分布式事务问题，建议选择同步刷盘模式。

#### 2. 多系统之间数据一致性（多方事务）



当需要保证多方（超过2方）的分布式一致性，上面的两方事务一致性（通过Rocketmq的事务性消息解决）已经无法支持。这个时候需要引入TCC模式思想（Try-Confirm-Cancel，不清楚的自行百度）。

以上图交易系统为例：

1）交易系统创建订单（往DB插入一条记录），同时发送订单创建消息。通过RocketMq事务性消息保证一致性

2）接着执行完成订单所需的同步核心RPC服务（非核心的系统通过监听MQ消息自行处理，处理结果不会影响交易状态）。执行成功更改订单状态，同时发送MQ消息。

3）交易系统接受自己发送的订单创建消息，通过定时调度系统创建延时回滚任务（或者使用RocketMq的重试功能，设置第二次发送时间为定时任务的延迟创建时间。在非消息堵塞的情况下，消息第一次到达延迟为1ms左右，这时可能RPC还未执行完，订单状态还未设置为完成，第二次消费时间可以指定）。延迟任务先通过查询订单状态判断订单是否完成，完成则不创建回滚任务，否则创建。 PS：多个RPC可以创建一个回滚任务，通过一个消费组接受一次消息就可以；也可以通过创建多个消费组，一个消息消费多次，每次消费创建一个RPC的回滚任务。 回滚任务失败，通过MQ的重发来重试。

以上是交易系统和其他系统之间保持最终一致性的解决方案。

#### 3.案例分析

##### 1) 单机环境下的事务示意图

**如下为A给B转账的例子。**

| **步骤** | **动作** |
| --- | --- |
| 1 | 锁定A的账户 |
| 2 | 锁定B的账户 |
| 3 | 检查A账户是否有1元 |
| 4 | A的账户扣减1元 |
| 5 | 给B的账户加1元 |
| 6 | 解锁B的账户 |
| 7 | 解锁A的账户 |

**以上过程在代码层面甚至可以简化到在一个事物中执行两条sql语句。**

##### 2) 分布式环境下事务

**和单机事务不同，A、B账户可能不在同一个DB中，此时无法像在单机情况下使用事物来实现。此时可以通过一下方式实现，将转账操作分成两个操作。**

a) A账户

| **步骤** | **动作** |
| --- | --- |
| 1 | 锁定A的账户 |
| 2 | 检查A账户是否有1元 |
| 3 | A的账户扣减1元 |
| 4 | 解锁A的账户 |

b) MQ消息  
A账户数据发生变化时，发送MQ消息，MQ服务器将消息推送给转账系统，转账系统来给B账号加钱。

c) B账户

| **步骤** | **动作** |
| --- | --- |
| 1 | 锁定B的账户 |
| 2 | 给B的账户加1元 |
| 3 | 解锁B的账户 |

## 四、 顺序消息

### 1. 顺序消息缺陷

发送顺序消息无法利用集群Fail Over特性消费顺序消息的并行度依赖于队列数量队列热点问题，个别队列由于哈希不均导致消息过多，消费速度跟不上，产生消息堆积问题遇到消息失败的消息，无法跳过，当前队列消费暂停。

### 2. 原理

produce在发送消息的时候，把消息发到同一个队列（queue）中,消费者注册消息监听器为MessageListenerOrderly，这样就可以保证消费端只有一个线程去消费消息。

注意：把消息发到同一个队列（queue），不是同一个topic，默认情况下一个topic包括4个queue

### 3. 扩展

可以通过实现发送消息的对列选择器方法，实现部分顺序消息。

举例：比如一个数据库通过MQ来同步，只需要保证每个表的数据是同步的就可以。解析binlog，将表名作为对列选择器的参数，这样就可以保证每个表的数据到同一个对列里面，从而保证表数据的顺序消费

## 五、 最佳实践

### 1. Producer

#### 1) Topic

一个应用尽可能用一个Topic，消息子类型用tags来标识，tags可以由应用自由设置。只有发送消息设置了tags，消费方在订阅消息时，才可以利用tags 在broker做消息过滤。

#### 2) key

每个消息在业务层面的唯一标识码，要设置到 keys 字段，方便将来定位消息丢失问题。服务器会为每个消息创建索引(哈希索引)，应用可以通过 topic，key来查询这条消息内容，以及消息被谁消费。由于是哈希索引，请务必保证key 尽可能唯一，这样可以避免潜在的哈希冲突。

//订单Id

String orderId= "20034568923546";

message.setKeys(orderId);

#### 3) 日志

消息发送成功或者失败，要打印消息日志，务必要打印 send result 和key 字段。

#### 4) send

send消息方法，只要不抛异常，就代表发送成功。但是发送成功会有多个状态，在sendResult里定义。

SEND\_OK：消息发送成功

FLUSH\_DISK\_TIMEOUT：消息发送成功，但是服务器刷盘超时，消息已经进入服务器队列，只有此时服务器宕机，消息才会丢失

FLUSH\_SLAVE\_TIMEOUT：消息发送成功，但是服务器同步到Slave时超时，消息已经进入服务器队列，只有此时服务器宕机，消息才会丢失

SLAVE\_NOT\_AVAILABLE：消息发送成功，但是此时slave不可用，消息已经进入服务器队列，只有此时服务器宕机，消息才会丢失

### 2. Consumer

#### 1) 幂等

RocketMQ使用的消息原语是At Least Once，所以consumer可能多次收到同一个消息，此时务必做好幂等。

#### 2) 日志

消费时记录日志，以便后续定位问题。

#### 3) 批量消费

尽量使用批量方式消费方式，可以很大程度上提高消费吞吐量。

## 六、 参考资料

### 1. 文档

RocketMQ\_design.pdf  
RocketMQ\_experience.pdf

### 2. 博客

分布式开放消息系统(RocketMQ)的原理与实践

[http://www.jianshu.com/p/453c6e7ff81c](https://www.jianshu.com/p/453c6e7ff81c)

RocketMQ事务消费和顺序消费详解

<http://www.cnblogs.com/520playboy/p/6750023.html>

ZeroCopy

<http://www.linuxjournal.com/article/6345>

IO方式的性能数据

<http://stblog.baidu-tech.com/?p=851>

作者：彦帧  
链接：https://www.jianshu.com/p/2838890f3284  
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。