Day03 RocketMQ笔记

**课程主题：**

**知识回顾：消息发送模式特点**

**1、消息是如何存储的？**

**2、如何实现顺序消息？**

**3、消息堆积了如何解决？**

**4、延时消息的应用场景及实现？**

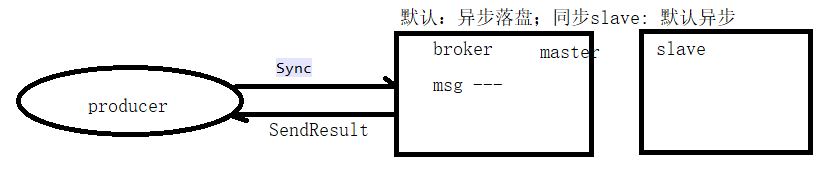
**5、如何实现消息去重？**

**6、如何保证消息的可靠性？（100%的可靠性）**

**7、rocketmq的事务消息是怎么回事？**

## 1、消息发送模式

### 1.1、同步发送

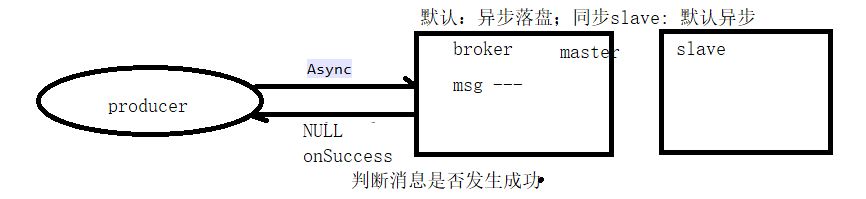


同步发送消息使用非常广泛，一般情况下，同步发送，消息可靠性非常高，因此一些重要的消息，不能丢失的消息使用同步发送。例如：

重要消息通知（金融），短信通知。

同步模式： 重试次数3次。

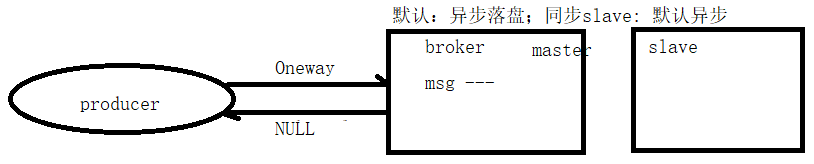
### 1.2、异步模式



异步模式发送消息：适用于对响应时间非常敏感的业务（要求响应时间非常迅速），不能容忍长时间的等待。项目为了提高吞吐能力，采用异步模式发送消息，异步方式消息可靠性也是ok的。

异步模式： 重试只有1次，重试一次的原因在于，异步模式发送必须要快，节省时间。

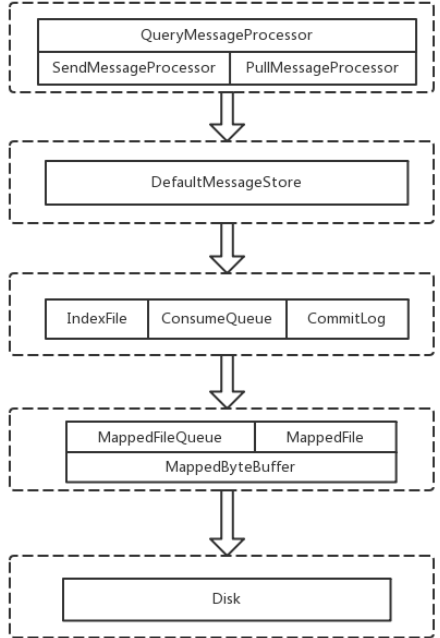
### 1.3、OneWay



特点：不关心消息发送的结果，有消息丢失。适用于消息不重要的场景。

## 2、消息存储

### 2.1、核心组件

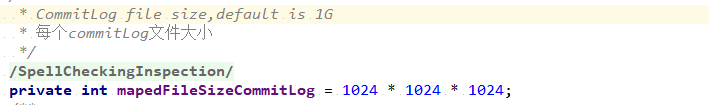


消息存储统一入口： DefaultMessageStore

IndexFile : 索引文件，hash slot , header,---- 运维

ConsumeQueue : 消息队列，此队列中存储的是offset (相对位移量)

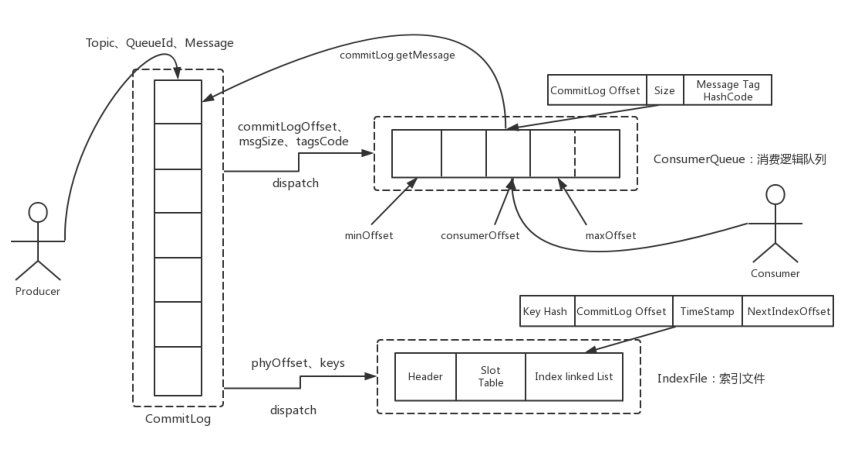
Commitlog ： 真正用来存储消息文件，此文件在顺序存储，在磁盘上一个连续的地址，commitLog默认大小：1G,如果超过1G大小，将会重新再次创建一个commitLog文件存储消息。



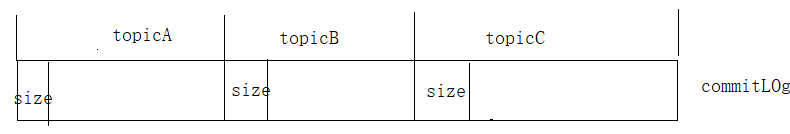
mappedFile – commitLog

mappedByteBuffer : 内存 --- 虚拟内存 （1.5G --- 2.5G） --- mmap 虚拟内存映射技术，通过此方式把commitlog文件在磁盘上进行顺序存储。

### 2.2、存储原理



1、commitlog是真正存储消息文件。commitLog中存储多个topic, topic下面有很多消息。



CommigLog命名： 根据字节数量进行命名。

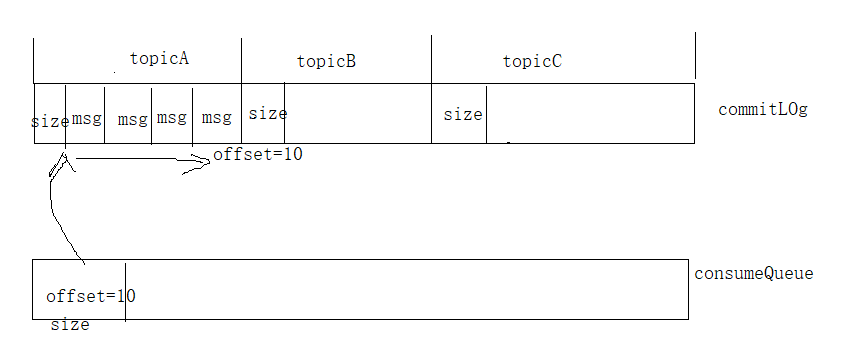
00000000000000000000 (0)

00000000001073741825 (1) – 物理位置偏移量

00000000001073741911（中间状态） (2) – 物理位置偏移量

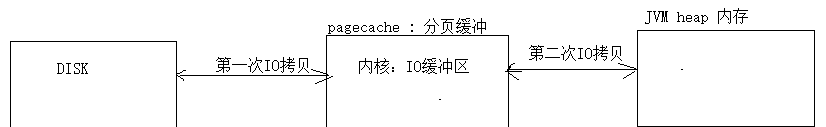
相对offset: 1073741911 – 1073741825 = 86 (offset)

ConsumeQueue: offset存储队列，消息者消费消息，根据offset读取commlog文件



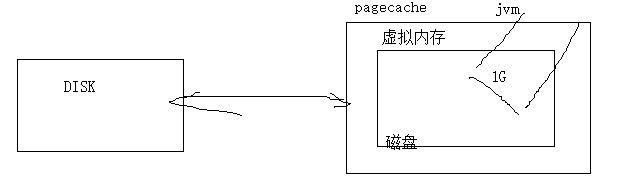
IndexFile: 根据ID查询消息，IndexFile可以很方便的去查询消息，操作消息。（运维）

### 2.3、Linux IO操作



以上就是linux x86架构模式下标准的read,write模式。涉及到2次的IO拷贝，性能相对低下，最关键是如此模式下文件存储，在磁盘存储数据分布上是散乱的，不是顺序存储，因此在读取数据的时候也是非常慢对的。

Mmap内存映射技术：pagecache:mmap直接把pagecache映射到用户态（jvm）地址空间中，这样就是实现0拷贝。



## 3、顺序消息

### 3.1、应用场景

思考：在什么样场景下，才需要发送顺序消息，且消费也必须按照消息顺序进行消费？？？

例如：

1、创建订单 --- 发送消息

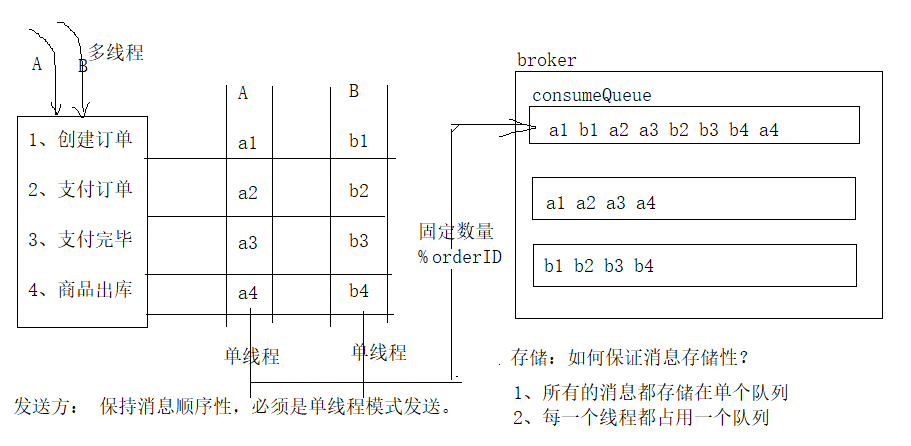
2、支付订单 --- 发送消息

3、支付完毕 --- 发送消息

4、商品出库 --- 发送消息

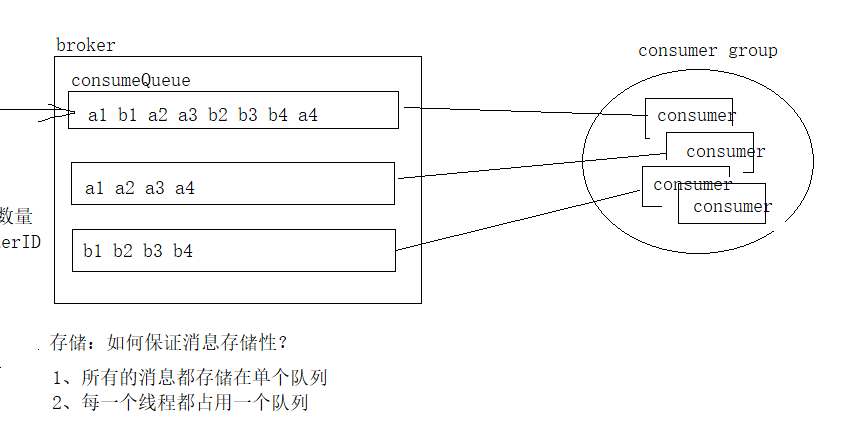
以上业务在项目中具有严格顺序性，不能发送乱序。乱序后业务就出现问了.

### 3.2、顺序消息实现



消息发送方，消息存储保证消息具有顺序性。

### 3.3、顺序消费



### 3.4、代码实现

*//重新写了消息发送策略*SendResult sendResult = producer.send(msg, **new** MessageQueueSelector() {  
 *//根据orderID发送顺序消息* @Override  
 **public** MessageQueue select(List<MessageQueue> mqs, Message msg, Object arg) {  
 Long id = (Long) arg; *//根据订单id选择发送queue*

*// A线程：11111 （4个消息） B线程： 11112 （4个消息）* **long** index = id % mqs.size();   
 **return** mqs.get((**int**) index);  
 }  
}, orderList.get(i).getOrderId());*//订单id*

## 4、消息堆积

消息堆积：

消费者出现了问题，没有及时消费消息，导致大量消息积压。

消费者数量少，消息发送比较快，导致消费者来不及消费消息，导致消息积压。

RocketMQ单机消息堆积能力： 10w

解决方案：

消息不重要： 直接跳过大量积压的消息，直接抛弃掉，从后面开始消费。



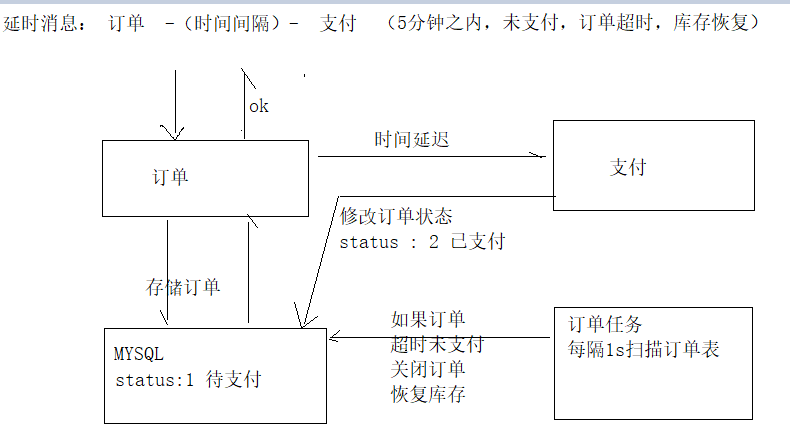
重要的消息： 修复消费者，扩容（扩展消费队列，扩展消费者数量）



## 5、延迟消息

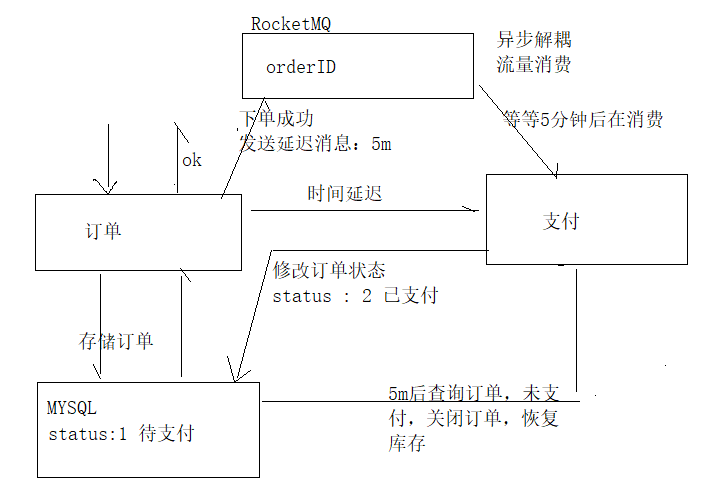
延时消息就是消费发送到broker服务器，此消息必须等待延时时间到了以后才能被消费者消费，此消息就是延时消息。

### 5.1、应用场景



以前方案： 解决订单超时未支付的问题。以上方案对数据库压力非常大。系统性能很差。

解决方案： 使用延迟消息

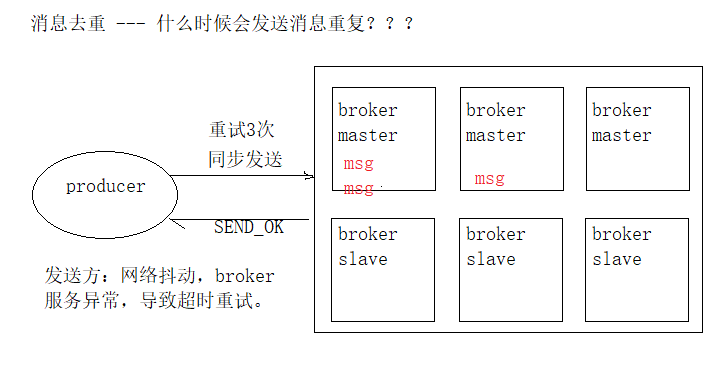


### 5.2、代码案例

**for** (**int** i = 0; i < totalMessagesToSend; i++) {  
 Message message = **new** Message(**"TestTopic"**, (**"Hello scheduled message "** + i).getBytes());  
 *// 设置延时等级3,这个消息将在10s之后发送(现在只支持固定的几个时间,详看delayTimeLevel)* message.setDelayTimeLevel(3);  
 *// 发送消息* producer.send(message);  
}

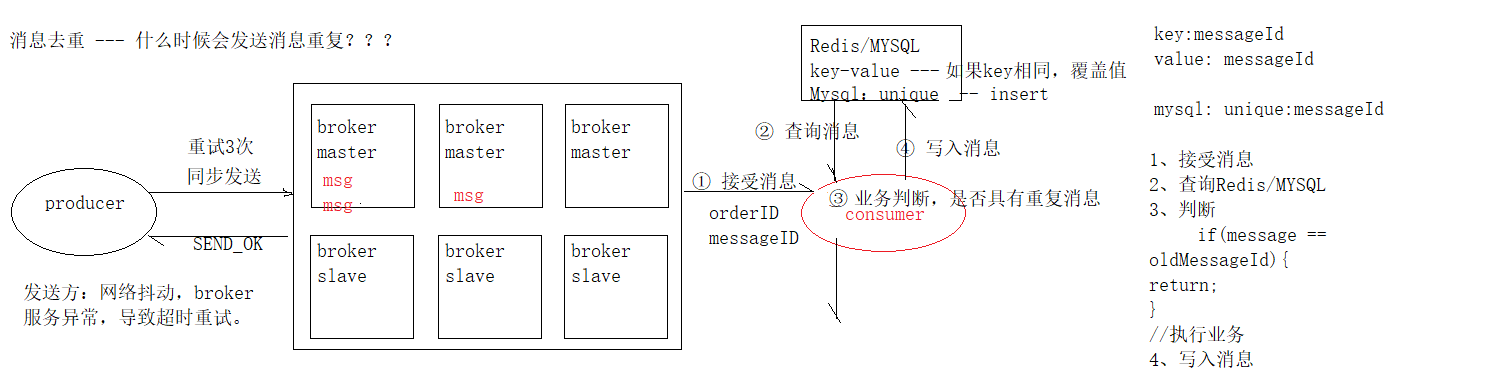
## 6、消息去重

### 6.1、消息重复发送场景



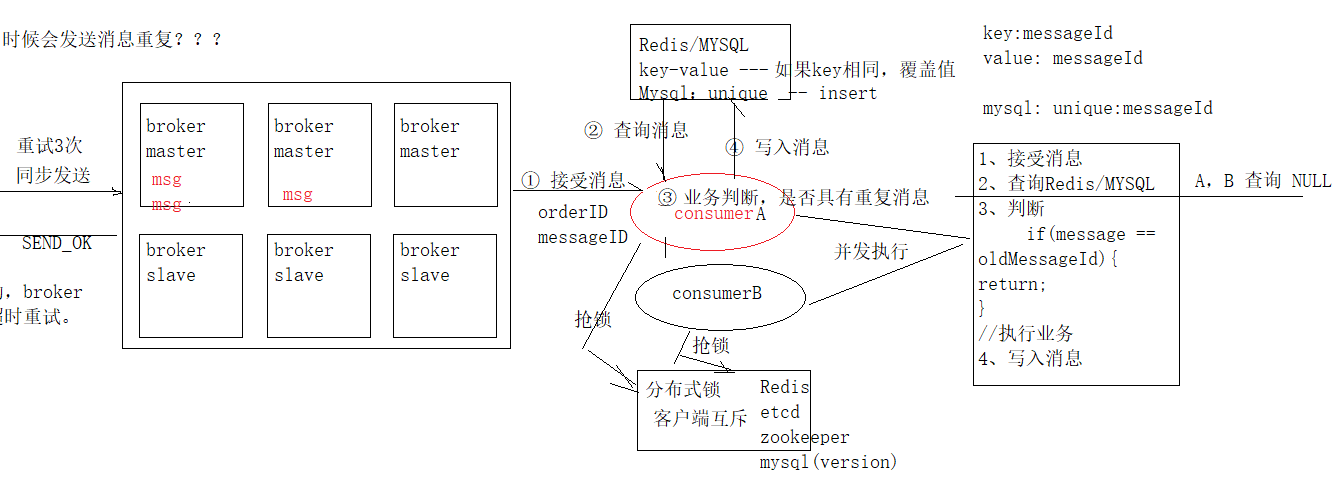
退款消息： 不解决，出现重复退款……

### 6.2、解决方案（单机）



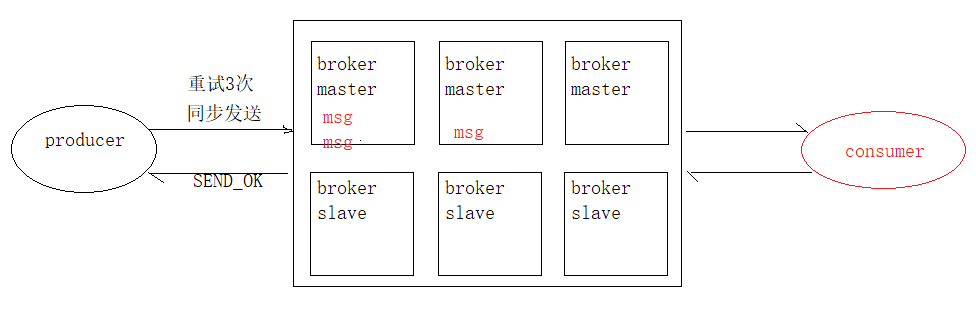
以上模式只有一个消费情况下去重是ok的，但是多个消费者还能实现消息去重吗？？

### 6.3、分布式集群



## 7、消息可靠性

### 7.1、消息丢失阶段



消息生产阶段：producer通过网络发送消息，由于网络延迟不可达，网络抖动，导致消息发送失败。

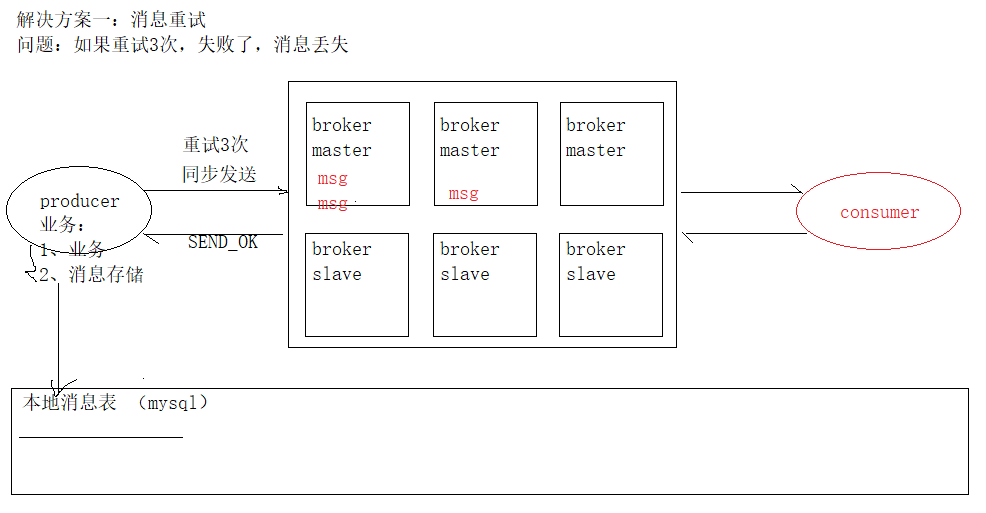
消息存储阶段：

producer把消息刚刚发送到broker服务器内存中，此时broker进程宕机了，消息还没有来得及落盘（消息可能丢失）-- broker宕机时候，自动shutdown,--commitlog.commit;

服务器宕机，操作系统宕机了，消息一定会丢失

消费者阶段： 消费失败

### 7.2、发送阶段



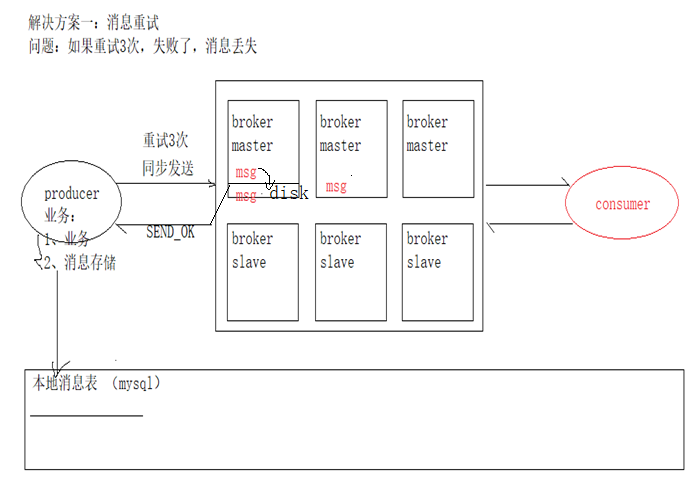
### 7.3、存储阶段

断电（可以通过落盘，同步slave方式解决），操作系统宕机都会存在消息丢失；因为broker服务器默认情况下，从内存到磁盘是异步落盘，也就是所当发生消息到内存中，还没有来得及落盘，此时服务器宕机，会造成消息丢失。

第一种方案： 同步落盘方式

// 发送消息 和 落盘是同步模式，也就是所发送消息必须等待消息落盘成功后，才认为此消息发送成功，否则认为此消息发送失败

flushDiskType = SYNC\_FLUSH // 同步落盘



第二种解决方案： master ,slave 同时落盘，才认为消息发送成功

#master节点落盘

flushDiskType = SYNC\_FLUSH // 同步落盘

brokerRole = SYNC\_MASTER

#slave节点刷盘

flushDiskType = SYNC\_FLUSH // 同步落盘

brokerRole = slave

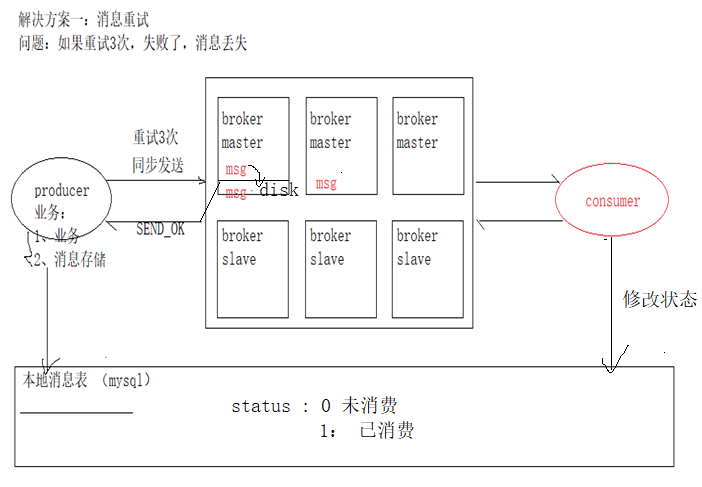
### 7.4、接受阶段

解决方案一： 重试，默认重试16次。

解决方案二： 重试16次，还是失败，如何解决？？？

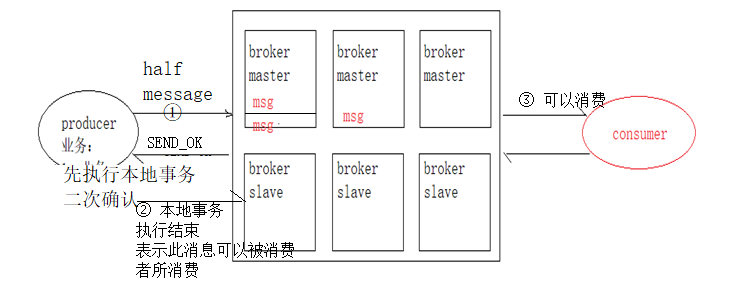
人工解决 ： 运维，console 人工投递

定时任务： 本地消息表进行重新投递，注意重复消息处理



## 8、事务消息

### 8.1、半消息机制

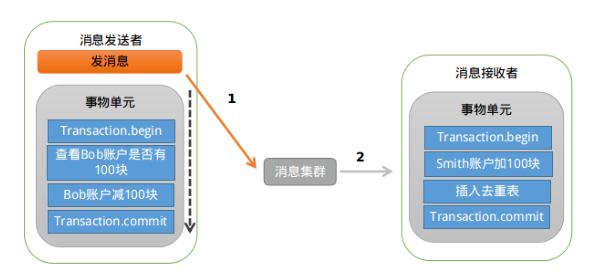


发送消息暂时不能被消费者消费，只有等到本地事务执行结束，二次确认消息（实际上修改消息状态），消费者就可以消费消息。通过这样方式，可以确保

Producer服务，consumer服务数据保持一致性。也就是说让producer业务，consumer业务要么都成功，要么都失败。

### 8.2、事务消息场景

事务场景一： 消息发送者： A银行系统 消费者： B银行系统



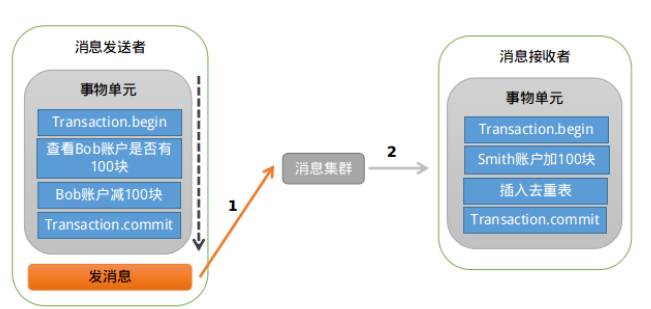
1、发送消息成功

2、本地事务执行失败

导致什么现象？？？？

本地事务执行失败，数据回滚，导致没有减100，但是B银行系统接受到消息，执行加100操作，导致2个服务数据不一致性。

事务场景二： 消息发送者： A银行系统 消费者： B银行系统



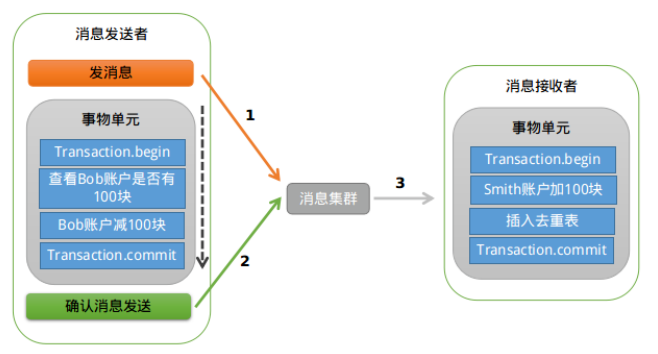
存在问题： 本地事务执行成功，发送消息失败了？？？

1、本地事务减100

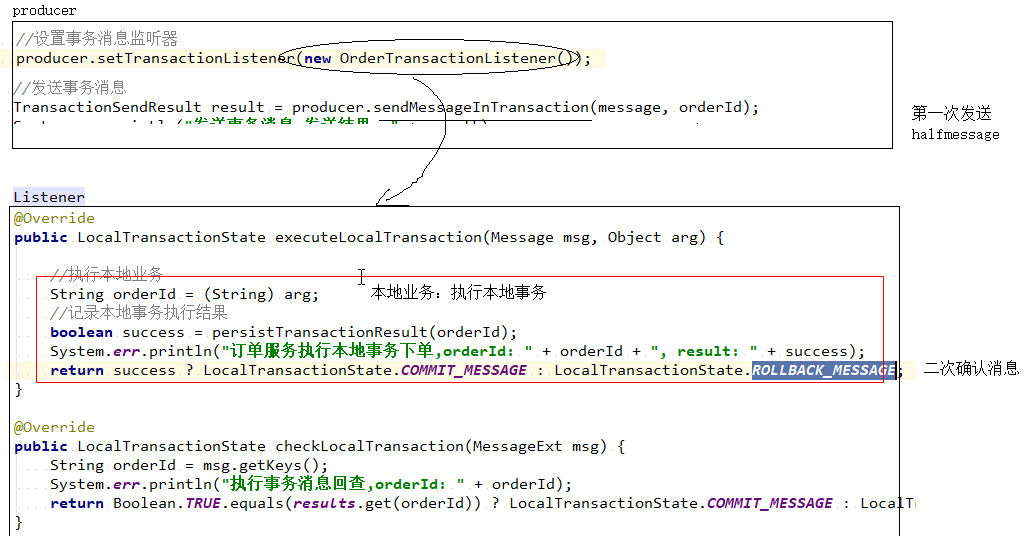
2、发送消息失败，导致b银行系统没有加100

导致2个系统数据不一致性。

半消息机制解决2个服务之间数据一致性问题：



### 8.3、代码实例



## 9、课后问题解答

### 问题一：RocketMQ存储消息量级是消息的条数？还是字节数？

答案：堆积能力指的是的消息的条数。但是注意这里所谓的单机10w级别的消息量级的堆积能力只是一个估算数字，也就是说rocketmq吹牛的一个数字，具体情况必须看服务器硬件配置环境，以及其他因素的影响。

Consume Queue中存储单元是一个20字节定长的二进制数据，顺序写顺序读

IndexFile 2000w 索引链表。

### 问题二：RocketMQ 是如何知道消息存储在哪个commigLog文件的？？

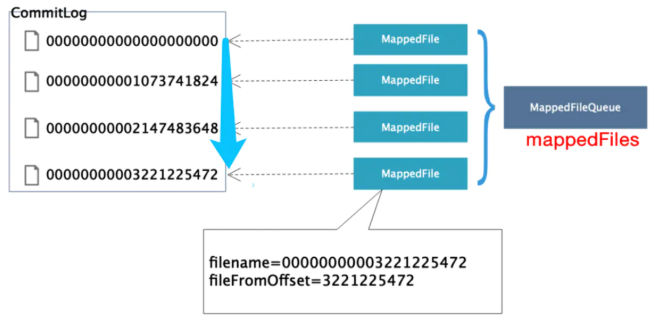
答案：

index 和 ComsumerQueue 中都有消息对应的物理偏移量，通过物理偏移量就可以计算出该消息位于哪个 CommitLog 文件上。

已知条件：

1、每个文件大小

2、相对位置偏移量



很明显：

Xx > Offset > 0 就在第一个文件中。 ----------- 偏移量，就可以计算出此消息到底在哪个文件。当然消息的大小也是可以设置的