1. RocketMQ 消息中间件存在的意义
2. 实现异步调用 对模块调用进行解耦 (无序等待 被调用模块返回结果再其他业务)
3. 削峰限流 ，大量的请求直接 发送到mq ，而不是直接发送到各个模块给服务造成瞬间的压力

不适用场景 ： 强一致性 一直等待结果,但是能保证最终一致性

1. RocketMQ 和 RabbitMQ的对比

RabbitMQ 只有主从，分发接收接收消息是微秒级的 ，但是吞吐量较低 万级的

RocketMQ 功能较为完善 ，支持分布式集群扩展性较好 吞吐量 10万 级的

1. RocketMQ 做的事情
   1. 接收消息
   2. 分发消息
   3. 存储消息

1. RocketMQ 的启动
   1. 启动nameServer 相当于注册中心,并根据topic定位到broker（路由消息的角色）
   2. 启动 broker 物理存储消息的地方

如果是集群 一台服务器对应一个broker ，包含多个topic每个topic包含8个队列

* 1. 启动管理控制台(可选，不影响使用) 从git上下载依赖项目rocketMQ

修改配置文件 连接 nameServer服务

1. **核心概念** 
   1. RocketMQ、kafaka 不像其他 mq 实现了消息规范，只是实现了生产消费的功能

ActiveMQ 实现了 java jms的消息规范

RabbitMQ 实现了 AMQP 消息规范

* 1. nameServer 相等于 消息的注册中心存储与broker的映射关系，根据topic找到broker
  2. 持久化机制 ： 默认持久化3 天
  3. Topic 表示不同类型的消息 (消息一级分类)

Topic存储在broker中，topic中又包含4个读队列 和 4个写队列 读写分离

* 1. Tag ：消息二级分类

Topic中又包含 tag ，表示topic某一消息类型下的二级分类， 用于定位某个消息时使用

* 1. 生产者组 和 消费者组的概念

如果是集群的话， 可以讲不同节点的相同模块划分到一个组中 。

注意 ： 如果生产者组 和 消费者组是可以重名的。重名并不表示生产者 和消费 者是同一个组

生产者组： 使用场景(再使用到分布式事务时 回调时需要使用到)

当其中一个节点在提交事务出问题了 ， MQ就会让这个组中的其他节点来做 剩下的事情

1. DefaultMQProducer(“组名”);

消费者组： 使用场景(消费者组中的所有成员，一起消费一个topic)

不同的方式即使用不同的 消费者监听

1. 第一种消费方式，按照块平均分(推荐使用)

topic中4个读对列 ，假如一个消费者组中有 2 个节点 那么4/2 =2每个节点 会分配两个2对列 。 那么在消费消息时每个节点只会重固定的队列中取消息

1. 第二种消费方式，顺序平均分(推荐使用)

topic中4个读对列 ，假如一个消费者组 中有 2 个节点 , 按照顺序取消息

1. 第三种消费方式，broker 就近机房分配

如果在消费者跟 broker在同一个局域网内 则消费者就到就近的broker中取

队列，进行消费

1. 第四种消费方式 指定特定的broker进行消息消费(基本不用)

不能实现高可用

1. 第五种消费方式 指定队列的方式(基本不用)

不能实现高可用

1. 第五种消费方式 Hash环分配

* 1. 广播(订阅)

集群环境下默认负载均衡只消费一次消息 ， 还有一种广播机制 所有消费者节点都消费

在消费端的 消费者组中设置属性

Consumer.setMessageModel(broadcasting); [ˈbrɔːdkæstɪŋ]

* 1. 重试机制 (针对的是消费者)

1. 如果消费者 处理业务时失败 ，可以不 return concumer\_success;

而返回 return reconcumer\_later; mq会把消息存至重发队列中。 接着MQ 会 等待一段时间 会重新推送消息 给消费者, 如果每次都失败 推送消息的间 隔时间会不断延长 。（默认重发 16次 如果还没有被消费 那么消息会进 入死信队列）

默认每次间隔时间的配置 和 消息延迟推送是一样的



1. 如果消费者接收完消息后 ，挂了 并没有返回 。 MQ默认认为没有消费成 功会继续发送消息 。知道 return concumer\_success;
2. 也可以自定义重试次数， 如果超过重试次数也会进死信队列

* 1. 死信队列

A、3中状态 (默认值是2)

2： 既不能写入数据 也不能读数据

4: 可以读数据 ， 不能写数据

6: 既能读 也能死

死信队列默认是不能 读和写入数据的， 需要重新设置值 。 如果消息写入死信队列 也可以监听该队列 ，去消费消息做一些人工操作

**发送消息的过程(默认20个生产线程，每5线程对应一个队列)** ：

生产者连接nameSertver并发送某个topic消息先到 nameServer 服务 ，然后 nameServer分发消息存储到 broker中的topic的写对列。写消息到队列 时默认是随 机的 ，rocketMQ默认会让消息均匀的分布在不同的写队列上 。 如果想发 送有序 的消息 在发送消息时指定消息进入哪一个队列。默认对数据进行 持久化默认 3 天清除。

无序发送：默认消息保存在不同的队列中，无法保证有序

有序发送：

1. 保证生产的消息进入同一个队列



使用 new SelectMessageQueueByHash(msg , ”id标识”)

参数1： 消息内容

参数2： 消息标识 ， mq会对这个这个标识进行hash运算来选择消息进入哪个队列， 如果期 望一组消息进入同一个队列那么 id标识必须一样

1. 消费端改成每个队列只能使用单线程消费,使用有序消费监听可以实现

使用 MessageListnerOrderly()

延迟发送(本质是延迟消费)：

在发送消息时设置 延迟级别(rocket不能随意设置延 迟时间，只能设置对应 的级别 。每个级别 对应不同的时间 。 如果提供的级别中没有想要的 可以 在broker配置文件中配置)

延迟发送时消息会立即进入broker ，然后延迟消费 如果是延迟发送，如果 集群 挂了消息不能被持久化就没了



Message msg = new Message(topic, tag, 消息内容);

msg.setDelayTimeLevel(3) //延迟3秒

应用场景 ： 如下订单 30分钟以内支付

当下完订单的同时，可以发送一条延迟30分钟修改订单状态的消息

如果30分钟还未支付， 消费端消费消息修改数据库将状态置为过期作废

批量发送：一次连接发送很多消息 数据总大小最好不要超过1M，超过4M报错

send时 发送 msg集合

缺点 ： 不支持延迟 不支持事务 不支持过滤

消费者过滤消息：通过 tag实现 (topic下的二级分类)

A、生产端：在创建消息时，既指定topic 又指定tag

Message msg = new Message(topic\_01, tag\_01, 消息内容);

B、消费端： 设置消费者 设置 consumer.subscribe(topic\_01 ,tag\_01);

那么消费端只能 消费topic主题 ， tag\_01的消息

如果想监听多个 tag 可以使用 ||

subscribe(topic\_01 ,”tag\_01 || tag\_02” );

**消费消息的过程(默认20个消费线程，每5线程对应一个队列)：**

消费者连接nameSertver ，告诉namaServer要消费哪个消息 ， 然 后namaServer会 从broker中的topic读对列获取数据然后push给消费者 。 当消费者消费完消息后 return success 后，消费才能消费完毕 。如果消费端消费了消息 但是报错导致没有 return success； broker中还会有这条消息

如果是集群的话，找不到nameServer会切换namServer

消费者如果是集群 如何保证按顺序消费消息呢 ？

设置某个读队列只能被特定的消费线程消费消息 ， 这样就保证了消费顺序

1. RocketMQ 使用
   1. Topic ： 生产 和 消费应对用同一个topic
   2. 过滤器

**生产者 ： 使用 producer.send 发送消息**

1. 发送消息的选择器
2. 无序存储到队列 ： 默认 使用 send(msg)





1. 有序存储到队列 ： new SelectMessageQueueByHash(msg, “标识”);



C、批量发送 send方法传 msg 集合

**消费者： 创建监听 接收消息 (集群数不要大于 topic中的读队列数)**

1. 消费模式 （推 + 拉）
   1. 消费者客户端 定时发送 pull 请求到nameServer ，并保持5s长连接 去拉取数据如果有数据，nameServer会推送 。如果没有拉到数据 会继续重 试3次 。 如果还是没有消息则断开长连接
   2. 接收消息的不同监听
2. 无序监听： MessageListnerConcurrently()
3. 有序监听：MessageListnerOrderly()
4. 指定消息监听 接收push过来的消息
5. 消费完 一定 return custumer\_success ; 不然broker不会清除消息
6. RocketMQ实现分布式事务(保证最终一致性，不能保证强一致性)
7. 分布式事务实现的方式

Tcc 两阶段提交 代码耦合较高 ，不利于后期的功能扩展及维护

Seata

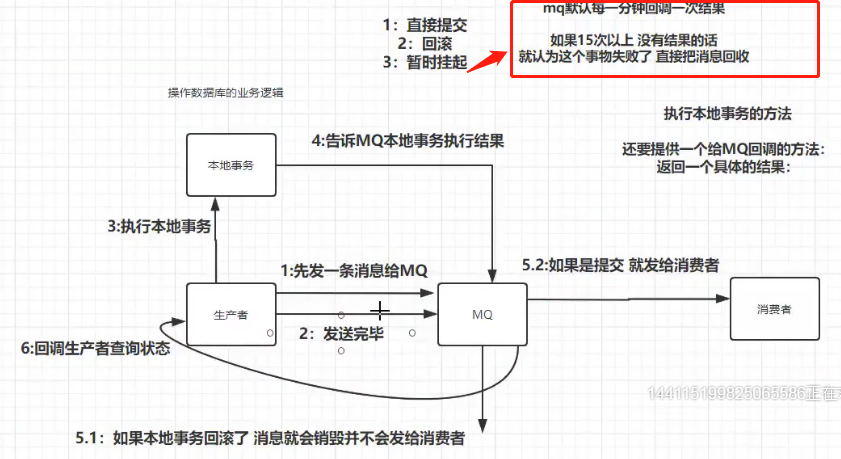
MQ实现

1. 强一直性 和 最终一致性的区别

强一致性 ： 如转账 要么同时都成功要么同时都失败

最终一致性 ： 要么都成功要么都失败 但是并不能保证同时

1. RocketMQ 本身提供了 分布式事务api 来实现事务， mq充当的角色 如果本地 事务提交完毕， 发送消息给消费者处理其他事务 而这条消息是绝对不会丢失的
2. 原理



1. 先发送一条事务的消息给 mq ，这条消费只有被确认后才能被消费
2. 发送完毕后 ，MQ调用本地业务处理mysql事务操作

实现 Transactionlistner接口 实现 excuteLoalTransaction() 方法

执行本地事务

1. 如果事务执行成功，在excuteLoalTransaction返回事务处理成功的标识

QM 会将事先发送的消息发送push给消费者进行消费

1. 如果 事务未提交 回滚了

return 回滚; 将事先发送的消息存至死信队列

1. 另外 MQ 还有一个占时挂起的功能

如果本地业务执行很久没有提交事务，或者没有明确的结果 可以开启挂起功能， 等确认事务了 再发送消息

实现 Transactionlistner接口 实现 checkLoalTransaction() 方法

自己根据业务检查执行业务结果是否发生改变 ,然后再确认是否提交

1. 注意 ： mq 只能确保 消息不丢失被消费 ，并不能保证消费端的事务是否能处理成功 。 至于消费端处理事务是否成功 需要通过业务代码取实现
2. 为什么先发送一条处理的消息 ，再执行本地的事务，而不是本地事务执行完毕再发送消息 ？

因为如果先处理本地事务 ，如果本地事务处理完毕数据库的值已经更新了， mq挂了 不能发送消息 那么实现不了分布式事务了

如果先发送消息 ， 不消费 ，消息已经到了 MQ中 . 如果本地事务回滚 就确认事务 时不成功 ，消息发送直接进死信队列

如果本地事务确认成功，即使mq挂了重启后 消息还在也能被消费者消费到

1. 如何让 springboot 的事务 和 rocketMQ的事务结合一起用 ？   
    A、配置事务监听重写两个方法
2. 处理本地事务 ： 可以 依赖 service ，service执行业务的方法并开启事务

@Autowrited

Service service; 处理业务返回结果

1. 执行本地事务校验的回调方法

B、service

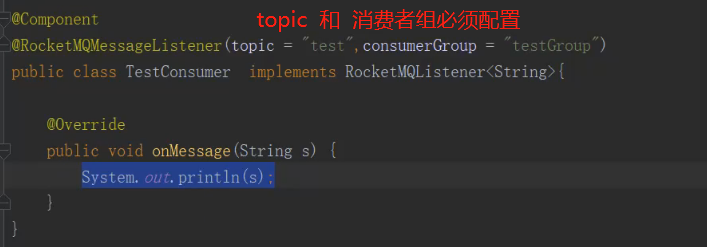
1. **整合springboot**
2. 生产者 ： 在controller 或者 service 业务中 发消息

@Autowrited

注入 ： RocketMQTemplete rocketMQTemplete;

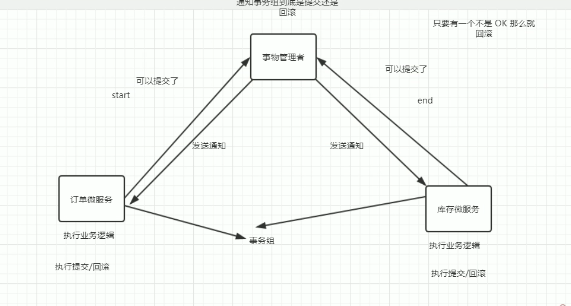
rocketMQTemplete.converAndSend(topic, msg);

1. 消费者 ：只需配置 topic 和 组即可



1. 手写分布式事务

性能高 ，但业务侵入性较强



1. 引入 事务管理者的概念(netty或socket服务)

事务管理者 中 定一个 map< ”事务组id”,事务组map >（是个事务组id对应一个map） 作为个个事务组

1. 定义注解@Tranction 两个属性 isStart isFalse
   1. 标注 isStart 的方法表示事务的开始
   2. 标注 isFlase 的方法表示事务的结束
2. 将connection的自动提交 改为手动提交, 并挂起 （通过java 的 condition 和 lock实现）
3. 一次执行 当前业务对数据库的每一次操作 并将当前执行是否成功的状态提交给事务组

（netty服务）

5当标注@Tranction(isEnd =true) 的方法 执行结束后 ， 通知挂起的操作 然后提交

提交 包括两种： 判断事务组中的状态是否包含 错误的 ，如果包含 全部rollback ，如果全部成功则提交到数据库

1. 判断完毕 后，将通知挂起的事务 根据之前是否提交成功的状态 决定是否提交或回滚