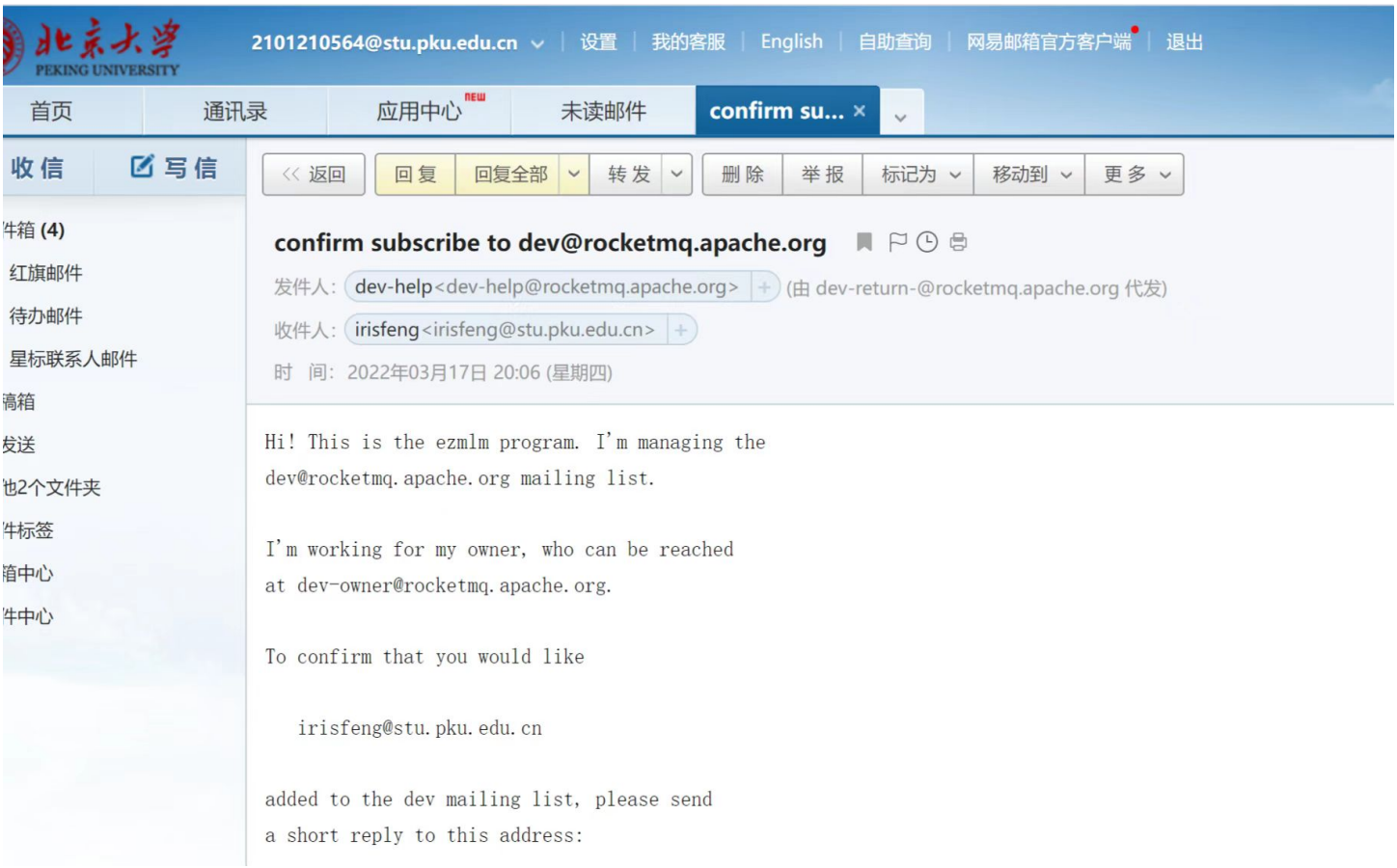
订阅



思考题

从发送端、消费端、服务端各个角度描述RocketMQ是如何保证消息不丢的?

丢失消息情景:

1. 生产者发消息到Broker时
2. Broker主从同步
3. Broker存储消息时
4. 消费者消费消息时
5. 整个MQ服务宕机

发送端:

方式一:采用同步发送

三种发送消息的方式

1. 同步发送
2. 异步发送

3. 单向发送

由于同步和异步方式均需要Broker返回确认信息，单向发送只管发，不需要Broker返回确认信息，所以单向发送并不知道消息是不是发送成功，单向发送不能保证消息不丢失。produce要想发消息时保证消息不丢失，可以采用同步发送的方式去发消息，send消息方法只要不抛出异常，就代表发送成功。

方式二：采用事务消息

RocketMQ的事务消息机制就是为了保证零丢失来设计的

消费端：

方式一：RockerMQ默认提供了At least Once机制保证消息可靠消费

Consumer先pull 消息到本地，消费完成后，才向服务器返回ack。

通常消费消息的ack机制一般分为两种思路：

1、先提交后消费；

2、先消费，消费成功后再提交；

思路一可以解决重复消费的问题但是会丢失消息，因此Rocketmq默认实现的是思路二，由各自consumer业务方保证幂等来解决重复消费问题。

方式二：消费消息重试机制

正常情况下，rocketMq拉取消息后，执行业务逻辑。一旦执行成功，将会返回一个ACK响应给 Broker，这时MQ就会修改offset，将该消息标记为已消费，不再往其他消费者推送消息。

如果出现消费超时(默认15分钟)、拉取消息后消费者服务宕机等消费失败的情况，此时的Broker由于没有等到消费者返回的ACK，会向同一个消费者组中的其他消费者间隔性的重发消息，直到消息返回成功（默认是重复发送16次，若16次还是没有消费成功，那么该消息会转移到死信队列,人工处理或是单独写服务处理这些死信消息）

在Broker的这种重新推送机制下，正常同步消费是不会丢消息的，但是异步消费就不一定。所以，为了保证消费的可靠性，消费端尽量不要使用异步消费机制。

服务端：

方式一：使用同步刷盘机制

同步刷盘机制，只有在消息真正持久化至磁盘后，RocketMQ的Broker端才会真正地返回给Producer端一个成功的ACK响应，保证了消息可靠性，但影响了性能。异步刷盘则能够充分利用OS的PageCache的优势，只要消息写入PageCache即可将成功的ACK返回给Producer端，消息刷盘采用后台异步线程提交的方式进行，提高了MQ的性能和吞吐量，但是可能会丢消息。点击查看配置方式

方式二：使用同步复制机制

同步复制是等Master和Slave都写入消息成功后才反馈给客户端写入成功的状态。在同步复制下，如果Master节点故障，Slave上有全部的数据备份，这样容易恢复数据。但是同步复制会增大数据写入的延迟，降低系统的吞吐量。异步复制是只要master写入消息成功，就反馈给客户端写入成功的状态。速度快，同样可能丢消息！