RabbitMQ 怎么对消息确认机制的

题目标签

- 题目难度：一般
- 知识点标签：RabbitMQ，RocketMQ，kafka
- 课程知识点：开源消息中间件RabbitMQ
- 课程时长：30分钟

问题回顾

在使用RabbitMQ的时候，我们可以通过消息持久化操作来解决因为服务器的异常奔溃导致的消息丢失，除此之外我们还会遇到一个问题，当消息的发布者在将消息发送出去之后，消息到底有没有正确到达broker代理服务器呢？如果不进行特殊配置的话，默认情况下发布操作是不会返回任何信息给生产者的，也就是默认情况下我们的生产者是不知道消息有没有正确到达broker的，如果在消息到达broker之前已经丢失的话，持久化操作也解决不了这个问题，因为消息根本就没到达代理服务器，你怎么进行持久化，那么这个问题该怎么解决呢？

解决方式

RabbitMQ为我们提供了两种方式：

1. 通过AMQP事务机制实现，这也是AMQP协议层面提供的解决方案；
2. 通过将channel设置成confirm模式来实现；

事务机制

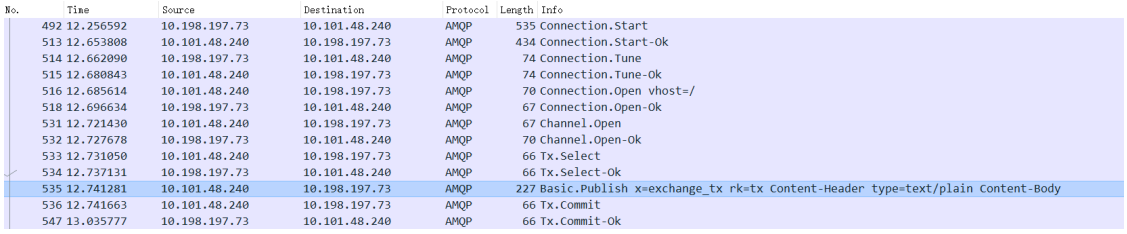
RabbitMQ中与事务机制有关的方法有三个：txSelect(), txCommit()以及txRollback(), txSelect用于将当前channel设置成transaction模式，txCommit用于提交事务，txRollback用于回滚事务，在通过txSelect开启事务之后，我们便可以发布消息给broker代理服务器了，如果txCommit提交成功了，则消息一定到达了broker了，如果在txCommit执行之前broker异常崩溃或者由于其他原因抛出异常，这个时候我们便可以捕获异常通过txRollback回滚事务了。

关键代码：

```
channel.txSelect();
channel.basicPublish(ConfirmConfig.exchangeName, ConfirmConfig.routingKey,
MessageProperties.PERSISTENT_TEXT_PLAIN, ConfirmConfig.msg_10B.getBytes());
channel.txCommit();
```

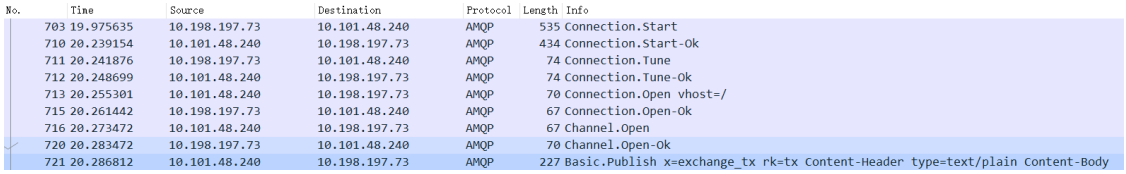
结果展示

- 带事务的



No.	Time	Source	Destination	Protocol	Length	Info
492	12.256592	10.198.197.73	10.101.48.240	AMQP	535	Connection.Start
513	12.653808	10.101.48.240	10.198.197.73	AMQP	434	Connection.Start-Ok
514	12.662090	10.198.197.73	10.101.48.240	AMQP	74	Connection.Tune
515	12.680843	10.101.48.240	10.198.197.73	AMQP	74	Connection.Tune-Ok
516	12.685614	10.101.48.240	10.198.197.73	AMQP	70	Connection.Open vhost=/
518	12.696634	10.198.197.73	10.101.48.240	AMQP	67	Connection.Open-Ok
531	12.721430	10.101.48.240	10.198.197.73	AMQP	67	Channel.Open
532	12.727678	10.198.197.73	10.101.48.240	AMQP	70	Channel.Open-Ok
533	12.731050	10.101.48.240	10.198.197.73	AMQP	66	Tx.Select
534	12.737131	10.198.197.73	10.101.48.240	AMQP	66	Tx.Select-Ok
535	12.741281	10.101.48.240	10.198.197.73	AMQP	227	Basic.Publish x=exchange_tx rk=tx Content-Header type=text/plain Content-Body
536	12.741663	10.101.48.240	10.198.197.73	AMQP	66	Tx.Commit
547	13.035777	10.198.197.73	10.101.48.240	AMQP	66	Tx.Commit-Ok

- 不带事务的



No.	Time	Source	Destination	Protocol	Length	Info
703	19.975635	10.198.197.73	10.101.48.240	AMQP	535	Connection.Start
710	20.239154	10.101.48.240	10.198.197.73	AMQP	434	Connection.Start-Ok
711	20.241876	10.198.197.73	10.101.48.240	AMQP	74	Connection.Tune
712	20.248699	10.101.48.240	10.198.197.73	AMQP	74	Connection.Tune-Ok
713	20.255301	10.101.48.240	10.198.197.73	AMQP	70	Connection.Open vhost=/
715	20.261442	10.198.197.73	10.101.48.240	AMQP	67	Connection.Open-Ok
716	20.273472	10.101.48.240	10.198.197.73	AMQP	67	Channel.Open
720	20.283472	10.198.197.73	10.101.48.240	AMQP	70	Channel.Open-Ok
721	20.286812	10.101.48.240	10.198.197.73	AMQP	227	Basic.Publish x=exchange_tx rk=tx Content-Header type=text/plain Content-Body

可以看到带事务的多了四个步骤：

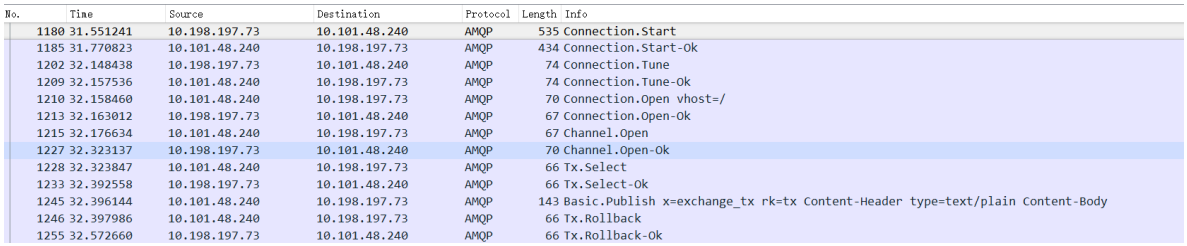
- client发送Tx.Select
- broker发送Tx.Select-Ok(之后publish)
- client发送Tx.Commit
- broker发送Tx.Commit-Ok

事务回滚

下面我们来看下事务回滚是什么样子的。关键代码如下：

```
try {
    channel.txSelect();
    channel.basicPublish(exchange, routingKey,
MessageProperties.PERSISTENT_TEXT_PLAIN, msg.getBytes());
    int result = 1 / 0;
    channel.txCommit();
} catch (Exception e) {
    e.printStackTrace();
    channel.txRollback();
}
```

数据展示



No.	Time	Source	Destination	Protocol	Length	Info
1180	31.551241	10.198.197.73	10.101.48.240	AMQP	535	Connection.Start
1185	31.770823	10.101.48.240	10.198.197.73	AMQP	434	Connection.Start-Ok
1202	32.148438	10.198.197.73	10.101.48.240	AMQP	74	Connection.Tune
1209	32.157536	10.101.48.240	10.198.197.73	AMQP	74	Connection.Tune-Ok
1210	32.158460	10.101.48.240	10.198.197.73	AMQP	70	Connection.Open vhost=/
1213	32.163012	10.198.197.73	10.101.48.240	AMQP	67	Connection.Open-Ok
1215	32.176634	10.101.48.240	10.198.197.73	AMQP	67	Channel.Open
1227	32.323137	10.198.197.73	10.101.48.240	AMQP	70	Channel.Open-Ok
1228	32.323847	10.101.48.240	10.198.197.73	AMQP	66	Tx.Select
1233	32.392558	10.198.197.73	10.101.48.240	AMQP	66	Tx.Select-Ok
1245	32.396144	10.101.48.240	10.198.197.73	AMQP	143	Basic.Publish x=exchange_tx rk=tx Content-Header type=text/plain Content-Body
1246	32.397986	10.101.48.240	10.198.197.73	AMQP	66	Tx.Rollback
1255	32.572660	10.198.197.73	10.101.48.240	AMQP	66	Tx.Rollback-Ok

代码中先是发送了消息至broker中但是这时候发生了异常，之后在捕获异常的过程中进行事务回滚。

总结

事务确实能够解决producer与broker之间消息确认的问题，只有消息成功被broker接受，事务提交才能成功，否则我们便可以在捕获异常进行事务回滚操作同时进行消息重发，但是使用事务机制的话会降低RabbitMQ的性能，那么有没有更好的方法既能保障producer知道消息已经正确送到，又能基本上不带来性能上的损失呢？从AMQP协议的层面看是没有更好的方法，但是RabbitMQ提供了一个更好的方案，即将channel信道设置成confirm模式。

存在的问题

生成者不知道消息是否真正到达broker，随后通过AMQP协议层面为我们提供了事务机制解决了这个问题，但是采用事务机制实现会降低RabbitMQ的消息吞吐量，那么有没有更加高效的解决方式呢

Confirm模式

producer端confirm模式的实现原理

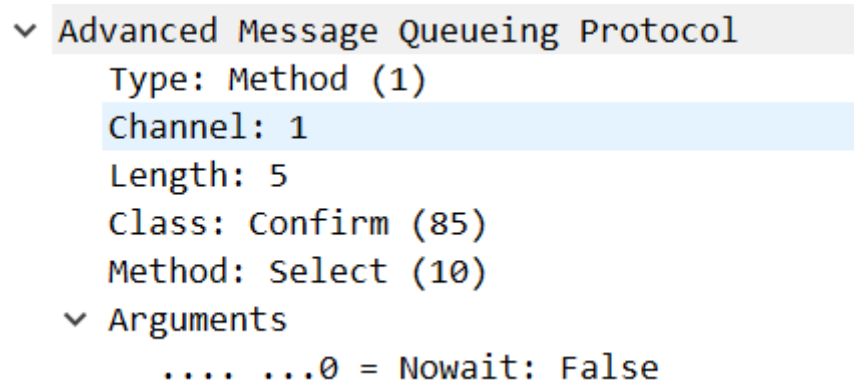
生产者将信道设置成confirm模式，一旦信道进入confirm模式，所有在该信道上面发布的消息都会被指派一个唯一的ID(从1开始)，一旦消息被投递到所有匹配的队列之后，broker就会发送一个确认给生产者（包含消息的唯一ID）,这就使得生产者知道消息已经正确到达目的队列了，如果消息和队列是可持久化的，那么确认消息会将消息写入磁盘之后发出，broker回传给生产者的确认消息中deliver-tag域包含了确认消息的序列号，此外broker也可以设置basic.ack的multiple域，表示到这个序列号之前的所有消息都已经得到了处理。

confirm模式最大的好处在于他是异步的，一旦发布一条消息，生产者应用程序就可以在等信道返回确认的同时继续发送下一条消息，当消息最终得到确认之后，生产者应用便可以通过回调方法来处理该确认消息，如果RabbitMQ因为自身内部错误导致消息丢失，就会发送一条nack消息，生产者应用程序同样可以在回调方法中处理该nack消息。

在channel 被设置成 confirm 模式之后，所有被 publish 的后续消息都将被 confirm（即 ack） 或者被 nack一次。但是没有对消息被 confirm 的快慢做任何保证，并且同一条消息不会既被 confirm又被 nack 。

开启confirm模式的方法

生产者通过调用channel的confirmSelect方法将channel设置为confirm模式，如果没有设置no-wait标志的话，broker会返回confirm.select-ok表示同意发送者将当前channel信道设置为confirm模式(从目前RabbitMQ最新版本3.6来看，如果调用了channel.confirmSelect方法，默认情况下是直接将no-wait设置成false的，也就是默认情况下broker是必须回传confirm.select-ok的)。



编码部分

第一种

- 普通confirm模式

简单　每发送一条消息后，调用waitForConfirms()方法，等待服务器端confirm。实际上是一种串行confirm了。。
关键代码如下：

```
channel.basicPublish(ConfirmConfig.exchangeName, ConfirmConfig.routingKey,
MessageProperties.PERSISTENT_TEXT_PLAIN, ConfirmConfig.msg_10B.getBytes());
if(!channel.waitForConfirms()){
    System.out.println("send message failed.");
}
```

第二种

- 批量confirm模式

稍微复杂一点　每发送一批消息后，调用waitForConfirms()方法，等待服务器端confirm。

客户端程序需要定期（每隔多少秒）或者定量（达到多少条）或者两则结合起来publish消息，然后等待服务器端confirm, 相比普通confirm模式，批量极大提升confirm效率，但是问题在于一旦出现confirm返回false或者超时的情况时，客户端需要将这一批次的消息全部重发，这会带来明显的重复消息数量，并且，当消息经常丢失时，批量confirm性能应该是不升反降的。
关键代码：

```
channel.confirmSelect();
for(int i=0;i<batchCount;i++){
    channel.basicPublish(ConfirmConfig.exchangeName, ConfirmConfig.routingKey,
MessageProperties.PERSISTENT_TEXT_PLAIN, ConfirmConfig.msg_10B.getBytes());
}
if(!channel.waitForConfirms()){
    System.out.println("send message failed.");
}
```

第三种

- 异步confirm模式

最复杂 提供一个回调方法，服务端confirm了一条或者多条消息后Client端会回调这个方法。

Channel对象提供的ConfirmListener()回调方法只包含deliveryTag（当前Chanel发出的消息序号），我们需要自己为每一个Channel维护一个unconfirm的消息序号集合，每publish一条数据，集合中元素加1，每回调一次handleAck方法，unconfirm集合删掉相应的一条（multiple=false）或多条（multiple=true）记录。从程序运行效率上看，这个unconfirm集合最好采用有序集合SortedSet存储结构。实际上，SDK中的waitForConfirms()方法也是通过SortedSet维护消息序号的。
关键代码：

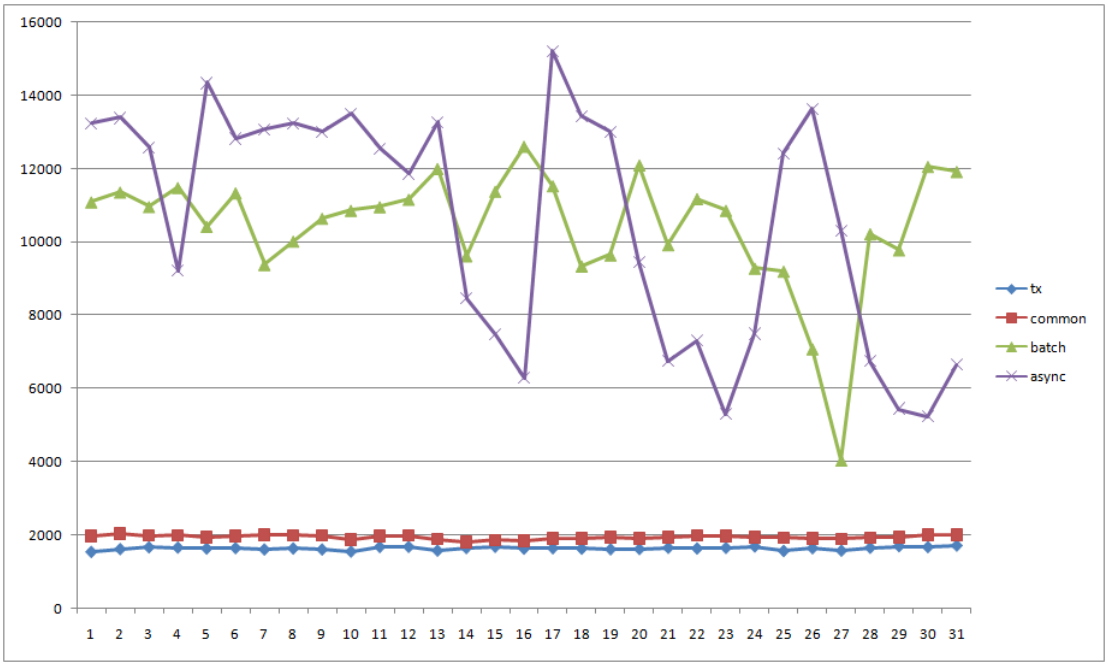
```
 SortedSet<Long> confirmSet = Collections.synchronizedSortedSet(new
TreeSet<Long>());
 channel.confirmSelect();
        channel.addConfirmListener(new ConfirmListener() {
            public void handleAck(long deliveryTag, boolean multiple) throws
IOException {
                if (multiple) {
                    confirmSet.headSet(deliveryTag + 1).clear();
                } else {
                    confirmSet.remove(deliveryTag);
                }
            }
            public void handleNack(long deliveryTag, boolean multiple) throws
IOException {
                System.out.println("Nack, SeqNo: " + deliveryTag + ", multiple:
" + multiple);
                if (multiple) {
                    confirmSet.headSet(deliveryTag + 1).clear();
                } else {
                    confirmSet.remove(deliveryTag);
                }
            }
        });

        while (true) {
            long nextSeqNo = channel.getNextPublishSeqNo();
```

```
            channel.basicPublish(ConfirmConfig.exchangeName,
ConfirmConfig.routingKey, MessageProperties.PERSISTENT_TEXT_PLAIN,
ConfirmConfig.msg_10B.getBytes());
            confirmSet.add(nextSeqNo);
        }
```

性能测试

Client端机器和RabbitMQ机器配置：CPU:24核，2600MHZ, 64G内存，1TB硬盘。
Client端发送消息体大小10B，线程数为1即单线程，消息都持久化处理（deliveryMode:2）。
分别采用事务模式、普通confirm模式，批量confirm模式和异步confirm模式进行producer实验，比对各个模式下的发送性能。



发送平均速率：

- 事务模式（tx）：1637.484
- 普通confirm模式(common)：1936.032
- 批量confirm模式(batch)：10432.45
- 异步confirm模式(async)：10542.06

总结

可以看到事务模式性能是最差的，普通confirm模式性能比事务模式稍微好点，但是和批量confirm模式还有异步confirm模式相比，还是小巫见大巫。批量confirm模式的问题在于confirm之后返回false之后进行重发这样会使性能降低，异步confirm模式(async)编程模型较为复杂，至于采用哪种方式，那是仁者见仁智者见智了。