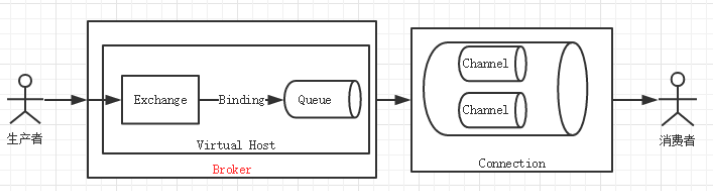
[RabbitMQ事务和Confirm发送方消息确认](https://www.cnblogs.com/vipstone/p/9350075.html)



## 一、事务使用

正常情况下，如果消息经过交换器进入队列就可以完成消息的持久化，但如果消息在没有到达broker之前出现意外，那就造成消息丢失，有没有办法可以解决这个问题？

RabbitMQ有两种方式来解决这个问题：

通过AMQP提供的事务机制实现；

使用发送者确认模式实现；

事务的实现主要是对信道（Channel）的设置，主要的方法有三个：

channel.txSelect()声明启动事务模式；

channel.txComment()提交事务；

channel.txRollback()回滚事务；

从上面的可以看出事务都是以tx开头的，tx应该是transaction extend（事务扩展模块）的缩写.

===CODE

// 创建连接

ConnectionFactory factory = new ConnectionFactory();

factory.setUsername("guest");

factory.setPassword("guest");

factory.setHost("localhost");

factory.setPort(5672);

Connection conn = factory.newConnection();

// 创建信道

Channel channel = conn.createChannel();

// 声明队列

channel.queueDeclare("tq1", true, false, false, null);

String message = String.format("时间 => %s", new Date().getTime());

try {

channel.txSelect();// 声明事务

// 发送消息

channel.basicPublish("", "tq1", MessageProperties.PERSISTENT\_TEXT\_PLAIN, message.getBytes("UTF-8"));

channel.txCommit(); // 提交事务

System.out.println("提交成功");

} catch (Exception e) {

// TODO: handle exception

channel.txRollback();

System.out.println("提交失败"+e);

}finally {

channel.close();

conn.close();

}

上就完成了事务的交互流程，如果其中任意一个环节出现问题，就会抛出IoException移除，这样用户就可以拦截异常进行事务回滚，或决定要不要重复消息。

那么，既然已经有事务了，没什么还要使用发送方确认模式呢，原因是因为事务的性能是非常差的。事务模式的性能要差很多，那有没有既能保证消息的可靠性又能兼顾性能的解决方案呢？那就是接下来要讲的**Confirm发送方确认模式**。

## 二、Confirm发送方确认模式

Confirm发送方确认模式使用和事务类似，也是通过设置Channel进行发送方确认的。

Confirm的三种实现方式：

方式一：channel.waitForConfirms()普通发送方确认模式；

方式二：channel.waitForConfirmsOrDie()批量确认模式；

方式三：channel.addConfirmListener()异步监听发送方确认模式；

### 方式一：普通Confirm模式

channel.confirmSelect();// 开启发送方确认模式

// 发送消息

channel.basicPublish("", "tq1", null, message.getBytes("UTF-8"));

if(channel.waitForConfirms()) {

System.out.println("发送成功");

}else {

System.out.println("发送失败");

}

我们只需要在推送消息之前，channel.confirmSelect()声明开启发送方确认模式，再使用channel.waitForConfirms()等待消息被服务器确认即可。

### 方式二：批量Confirm模式

channel.confirmSelect();// 开启发送方确认模式

// 发送消息

for (int i = 0; i < 10000; i++) {

channel.basicPublish("", "tq1", null, message.getBytes("UTF-8"));

}

channel.waitForConfirmsOrDie();

////直到所有信息都发布，只要有一个未确认就会IOException

System.out.println("全部执行完成"+(end-start));

channel.waitForConfirmsOrDie()，使用同步方式等所有的消息发送之后才会执行后面代码，只要有一个消息未被确认就会抛出IOException异常。

### 方式三：异步Confirm模式

channel.confirmSelect();

// 发送消息

long start = System.currentTimeMillis();

for (int i = 0; i <10000; i++) {

channel.basicPublish("", "tq1", null, message.getBytes("UTF-8"));

}

channel.addConfirmListener(new ConfirmListener() {

public void handleNack(long deliveryTag, boolean multiple) throws IOException {

// TODO Auto-generated method stub

System.out.println("消息未确认"+deliveryTag);

}

public void handleAck(long deliveryTag, boolean multiple) throws IOException {

// TODO Auto-generated method stub

System.out.println("消息已经确认"+deliveryTag+" "+multiple);

}

});

异步模式的优点，就是执行效率高，不需要等待消息执行完，只需要监听消息即可.

可以看出，代码是异步执行的，消息确认有可能是批量确认的，是否批量确认在于返回的multiple的参数，此参数为boolean值，如果true表示批量执行了deliveryTag这个值以前的所有消息，如果为false的话表示单条确认。

Confirm性能测试

测试前提：与事务一样，我们发送1w条消息。

方式一：Confirm普通模式

执行花费时间：2253s

执行花费时间：2018s

执行花费时间：2043s

方式二：Confirm批量模式

执行花费时间：1576s

执行花费时间：1400s

执行花费时间：1374s

方式三：Confirm异步监听方式

执行花费时间：1498s

执行花费时间：1368s

执行花费时间：1363s

总结

综合总体测试情况来看：Confirm批量确定和Confirm异步模式性能相差不大，Confirm模式要比事务快10倍左右。