Active MQ 高级特性和用法（二）：消息的可靠性、通配符式订阅与死信队列

作者：DongXiaokai0819

原文链接：https://ld246.com/article/1583568767423

来源网站：链滴

Active MQ 高级特性和用法（二）：消息的可靠性、通配符式订阅与死信队列

一、消息的可靠性

消息发送成功后，接收端接收到了消息。然后进行处理，但是可能由于某种原因，高并发也好，IO阻也好，反正这条消息在接收端处理失败了。而点对点的特性是一条消息，只会被一个接收端给接收，要接收端A接收成功了，接收端B，就不可能接收到这条消息，如果是一些普通的消息还好，但是如果一些很重要的消息，比如说用户的支付订单，用户的退款，这些与金钱相关的，是必须保证成功的，么这个时候要怎么处理呢？

必须要保证消息的可靠性，除了**消息的持久化**，还包括两个方面，一是生**产者发送的消息可以被ActiveMQ收到**，二是**消费者确实收到了ActiveMQ发送的消息**

生产者端的可靠性

1、send()方法

在生产者端，我们会使用send() 方法向ActiveMQ发送消息，默认情况下，持久化消息以同步方式发，send() 方法会被阻塞，直到 broker 发送一个确认消息给生产者，这个确认消息表示broker已经成接收到消息，并且持久化消息已经把消息保存到二级存储中。

2、测试send()方法

```
//循环发送消息
for (int i = 0; i < SENDNUM; i++) {
    String msg = "发送消息" + i + " " + System.currentTimeMillis();
    TextMessage textMessage = session.createTextMessage(msg);
    System.out.println("标准用法:" + msg);
```

```
    messageProducer.send(textMessage);
}
```

    我们在send方法上打
个断点，可以看到send方法每执行一次，ActiveMQ管理控制台增加一条入队消息，数据库中增加一
消息。

3、事务消息

事务中消息（无论是否持久化），会进行异步发送，send() 方法不会被阻塞。但是commit 方法会被
塞，直到收到确认消息，表示broker已经成功接收到消息，并且持久化消息已经把消息保存到二级存
中。

4、测试事务消息

```
//循环发送消息
for (int i = 0; i < SENDNUM; i++) {
    String msg = "发送消息" + i + " " + System.currentTimeMillis();
    TextMessage textMessage = session.createTextMessage(msg);
    System.out.println("标准用法:" + msg);
    messageProducer.send(textMessage);
}
session.commit();
```

    我们在session.commit
)打一个断点，可以看到send方法每执行一次，ActiveMQ管理控制台和数据库中没有任何变化，只有
行完session.commit()后ActiveMQ管理控制台和数据库中才增加。

5、生产者端可靠性总结

    **非持久化又不在事务
的消息**，可能会有消息的丢失。为保证消息可以被ActiveMQ收到，我们应该采用**事务消息**或**持久化
息**。

消费者端的可靠性

ACK_MODE描述了Consumer与broker确认消息的方式(时机),比如当消息被Consumer接收之后,Con
umer将在何时确认消息。所以ack_mode描述的不是producer于broker之间的关系，而是customer
broker之间的关系。

对于broker而言，只有接收到ACK指令,才会认为消息被正确的接收或者处理成功了,通过ACK，可以在
onsumer与Broker之间建立一种简单的“担保”机制.

对**消息的确认**有4种机制(customer对broker进行消息确认)

1. AUTO_ACKNOWLEDGE = 1　自动确认
2. LIENT_ACKNOWLEDGE = 2　客户端手动确认
3. DUPS_OK_ACKNOWLEDGE = 3　自动批量确认
4. ESSION_TRANSACTED = 0　事务提交并确认

    在创建Session时设置

息的确认机制：

session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);

第一个参数:是否支持事务，如果为true，则会忽略第二个参数，自动被jms服务器设置为SESSION_TANSACTED

1、AUTO_ACKNOWLEDGE

● 客户端 **自动确认**机制。

● “同步”(receive)方法返回message给消息时会立即确认。

● 在"异步"(messageListener)方式中,将会首先调用listener.onMessage(message)，如果onMessag方法正常结束,消息将会正常确认。如果onMessage方法异常，将导致消费者要求ActiveMQ重发消。此外需要注意，消息的重发次数是有限制的，每条消息中都会包含“redeliveryCounter”计数器用来表示此消息已经被重发的次数，如果重发次数达到阀值，将导致broker端认为此消息无法消费,消息将会被删除或者迁移到"dead letter"通道中。

● 因此当我们使用messageListener方式消费消息时，可以在onMessage方法中使用try-catch,这样以在处理消息出错时记录一些信息，而不是让consumer不断去重发消息；如果你没有使用try-catch,有可能会因为异常而导致消息重复接收的问题,需要注意onMessage方法中逻辑是否能够兼容对重复息的判断。

1、LIENT_ACKNOWLEDGE

● 客户端 **手动确认**，这就意味着AcitveMQ将不会“自作主张”的为你ACK任何消息，开发者需要己择机确认。可以用方法： message.**acknowledge()**，或session.acknowledge()；效果一样。

● 如果忘记调用acknowledge方法，将会导致当consumer重启后，会接受到重复消息，因为对于broer而言，那些尚未真正ACK的消息被视为“未消费”。

● 我们可以在当前消息处理成功之后，立即调用message.acknowledge()方法来"逐个"确认消息，这可以尽可能的减少因网络故障而导致消息重发的个数；当然也可以处理多条消息之后，间歇性的调用aknowledge方法来一次确认多条消息，减少ack的次数来提升consumer的效率，不过需要自行权衡。

1、DUPS_OK_ACKNOWLEDGE

● 类似于AUTO_ACK确认机制，为自动批量确认而生，而且具有“延迟”确认的特点，ActiveMQ会据内部算法，在收到一定数量的消息自动进行确认。在此模式下，可能会出现重复消息，什么时候?consumer故障重启后，那些尚未ACK的消息会重新发送过来。

1、ESSION_TRANSACTED

● 当session使用事务时，就是使用此模式。当决定事务中的消息可以确认时，必须调用session.comit()方法，commit方法将会导致当前session的事务中所有消息立即被确认。在事务开始之后的任何机调用rollback()，意味着当前事务的结束，事务中所有的消息都将被重发。当然在commit之前抛出常，也会导致事务的rollback。

二、通配符式分层订阅

Wildcards 用来支持联合的名字分层体系（federated name hierarchies）。它不是JMS 规范的一部，而是ActiveMQ 的扩展。

ActiveMQ 支持以下三种wildcards：

- "." 用于作为路径上名字间的 **分隔符。**
- "*" 用于匹配路径上的 **任何名字。**
- ">" 用于递归地匹配 **任何以这个名字开始的**destination。

    订阅者可以明确地指定
estination 的名字来订阅消息，或者它也可以使用wildcards 来定义一个**分层的模式**来匹配它希望订的 destination。

通配符测试

    接下来，我们创建了
个topic模式的消费者，并启动。

```
//消费者01,匹配kk.开头的任何destination
destination = session.createTopic("kk.>");
//消费者02，匹配kk.vip.开头的任何destination
destination = session.createTopic("kk.vip.>");
//消费者03，匹配kk.vip.*.redis.cache
destination = session.createTopic("kk.vip.*.redis.cache");
```

    然后定义三个topic模
的生产者，来测试一下：

```
//生产者01，符合消费者1、2的匹配规则
destination = session.createTopic("kk.vip.program.thread");
//生产者02，符合消费者1、2、3的匹配规则
destination = session.createTopic("kk.vip.program.redis.cache");
//生产者03，符合消费者1的匹配规则
destination = session.createTopic("kk.public.arct.redis.cache");
```

    测试结果

kk.public.arct.redis.cache	1	3	0	Send To Active Subscribers Active Producers Delete
kk.vip.program.redis.cache	3	3	0	Send To Active Subscribers Active Producers Delete
kk.vip.program.thread	2	6	0	Send To Active Subscribers Active Producers Delete

三、死信队列DLQ（Dead Letter Queue）

死信队列是用来保存处理失败或者过期的消息的一种特殊的普通队列。

    当一个消息被重发超
最大重发次数（缺省为6次，消费者端可以修改）时，会给broker发送一个"有毒标记 “，这个消息被为是有问题，这时broker将这个消息发送到死信队列，以便后续处理。

    消息在什么情况下会被
发?

1. 事务会话被回滚。

2. 事务会话在提交之前关闭。

3. 会话使用CLIENT_ACKNOWLEDGE模式，并且Session.recover()被调用。

4. 自动应答失败

在配置文件(activemq.xml)来调整死信发送策略：

```
<policyEntry queue=">" producerFlowControl="true" memoryLimit="1mb">
  <!-- Use VM cursor for better latency
       For more information, see:

       http://activemq.apache.org/message-cursors.html

  <pendingQueuePolicy>
    <vmQueueCursor/>
  </pendingQueuePolicy>
  -->
    <deadLetterStrategy>
        <!--
         queuePrefix:设置死信队列前缀
         useQueueForQueueMessages: 设置使用队列保存死信，
         可以设置useQueueForTopicMessages，使用Topic来保存死信
        -->
        <individualDeadLetterStrategy   queuePrefix="DLQ." useQueueForQueueMessages="true" />
    </deadLetterStrategy>
</policyEntry>
```

```
<policyEntry queue=">">
  <deadLetterStrategy>
   <!--queuePrefix:设置死信队列前缀 -->
   <!--useQueueForQueueMessages:设置使用队列保存死信 -->
   <!--可以设置useQueueForQueueMessages，使用Topic来保存死信 -->
   <individualDeadLetterStrategy queuePrefix="DLQ." useQueueForQueueMessages="true" p
ocessExpired="false"/>
    <!--是否丢弃过期消息-->
    <!--<sharedDeadLetterStrategy processExpired="false" />-->
   </deadLetterStrategy>
</policyEntry>
```

制造死信队列

    测试死信队列发送端：

```
public class DlqProducer {

   //默认连接用户名
   private static final String USERNAME
         = ActiveMQConnection.DEFAULT_USER;
   //默认连接密码
   private static final String PASSWORD
         = ActiveMQConnection.DEFAULT_PASSWORD;
   //默认连接地址
   private static final String BROKEURL
         = ActiveMQConnection.DEFAULT_BROKER_URL;
   //发送的消息数量
   private static final int SENDNUM = 1;

   public static void main(String[] args) {
      //不是直接使用接口，而是使用Active MQ所提供的工厂，已便于我们进行更多配置信息。
```

```
        ActiveMQConnectionFactory connectionFactory;
        ActiveMQConnection connection = null;
        Session session;
        ActiveMQDestination destination;
        MessageProducer messageProducer;

        connectionFactory
                = new ActiveMQConnectionFactory(USERNAME,PASSWORD,BROKEURL);
        try {
            connection = (ActiveMQConnection) connectionFactory.createConnection();
            connection.start();

            session = connection.createSession(true,Session.AUTO_ACKNOWLEDGE);

            destination = (ActiveMQDestination) session.createQueue("TestDlq2");
            messageProducer = session.createProducer(destination);
            for(int i=0;i<SENDNUM;i++){
                String msg = "发送消息"+i+" "+System.currentTimeMillis();
                TextMessage message = session.createTextMessage(msg);

                System.out.println("发送消息:"+msg);
                messageProducer.send(message);
            }
            session.commit();

        } catch (JMSException e) {
            e.printStackTrace();
        }finally {
            if(connection!=null){
                try {
                    connection.close();
                } catch (JMSException e) {
                    e.printStackTrace();
                }
            }
        }
    }
}
```

    测试死信队列的接收端

    通过配置重发策略来造死信队列

```
public class DlqConsumer {

    private static final String USERNAME
            = ActiveMQConnection.DEFAULT_USER;//默认连接用户名
    private static final String PASSWORD
            = ActiveMQConnection.DEFAULT_PASSWORD;//默认连接密码
    private static final String BROKEURL
            = ActiveMQConnection.DEFAULT_BROKER_URL;//默认连接地址

    public static void main(String[] args) {
        ActiveMQConnectionFactory connectionFactory;
```

```
        ActiveMQConnection connection = null;
        Session session;
        ActiveMQDestination destination;

        MessageConsumer messageConsumer;//消息的消费者

        //实例化连接工厂
        connectionFactory
                = new ActiveMQConnectionFactory(USERNAME,PASSWORD,BROKEURL);
        //配置策略
        RedeliveryPolicy redeliveryPolicy = new RedeliveryPolicy();
        //限制了重发次数为1
        redeliveryPolicy.setMaximumRedeliveries(1);

        try {
            //通过连接工厂获取连接
            connection = (ActiveMQConnection) connectionFactory.createConnection();
            //启动连接
            connection.start();
            // 拿到消费者端重复策略map
            RedeliveryPolicyMap redeliveryPolicyMap
                    = connection.getRedeliveryPolicyMap();
            //创建session
            session
                    = connection.createSession(false,
                    Session.AUTO_ACKNOWLEDGE);
            destination = (ActiveMQDestination) session.createQueue("TestDlq2");
            // 将消费者端重发策略配置给消费者
            redeliveryPolicyMap.put(destination,redeliveryPolicy);
            //创建消息消费者
            messageConsumer = session.createConsumer(destination);
            messageConsumer.setMessageListener(new MessageListener() {
                public void onMessage(Message message) {
                    try {
                        System.out.println("Accept msg : "
                                +((TextMessage)message).getText());
                    } catch (JMSException e) {
                        e.printStackTrace();
                    }
                    throw new RuntimeException("test");
                }
            });
        } catch (JMSException e) {
            e.printStackTrace();
        }
    }
}
```

消费死信队列

消费死信队列：

```
public class ProcessDlqConsumer {
```

```
    private static final String USERNAME
            = ActiveMQConnection.DEFAULT_USER;//默认连接用户名
    private static final String PASSWORD
            = ActiveMQConnection.DEFAULT_PASSWORD;//默认连接密码
    private static final String BROKEURL
            = ActiveMQConnection.DEFAULT_BROKER_URL;//默认连接地址

    public static void main(String[] args) {
        ConnectionFactory connectionFactory;//连接工厂
        Connection connection = null;//连接

        Session session;//会话 接受或者发送消息的线程
        Destination destination;//消息的目的地

        MessageConsumer messageConsumer;//消息的消费者

        //实例化连接工厂
        connectionFactory = new ActiveMQConnectionFactory(ProcessDlqConsumer.USERNAME

                ProcessDlqConsumer.PASSWORD, ProcessDlqConsumer.BROKEURL);

        try {
            //通过连接工厂获取连接
            connection = connectionFactory.createConnection();
            //启动连接
            connection.start();
            //创建session
            session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);
            //创建一个连接HelloWorld的消息队列
            //destination = session.createTopic("TestDlq");
            destination = session.createQueue("DLQ.>");

            //创建消息消费者
            messageConsumer = session.createConsumer(destination);
            messageConsumer.setMessageListener(new MessageListener() {
                public void onMessage(Message message) {
                    try {
                        System.out.println("Accept DEAD msg : "
                                +((TextMessage)message).getText());
                    } catch (JMSException e) {
                        e.printStackTrace();
                    }
                }
            });
        } catch (JMSException e) {
            e.printStackTrace();
        }
    }
}
```