**简介**

     在前面一篇文章里讨论过几种应用系统集成的方式，发现实际上面向消息队列的集成方案算是一个总体比较合理的选择。这里，我们先针对具体的一个消息队列 Activemq的基本通信方式进行探讨。activemq是JMS消息通信规范的一个实现。总的来说，消息规范里面定义最常见的几种消息通信模式主要有 发布-订阅、点对点这两种。另外，通过结合这些模式的具体应用，我们在处理某些应用场景的时候也衍生出来了一种请求应答的模式。下面，我们针对这几种方式 一一讨论一下。

**基础流程**

    在讨论具体方式的时候，我们先看看使用activemq需要启动服务的主要过程。

    按照JMS的规范，我们首先需要获得一个JMS connection factory.，通过这个connection factory来创建connection.在这个基础之上我们再创建session, destination, producer和consumer。因此主要的几个步骤如下：

1. 获得JMS connection factory. 通过我们提供特定环境的连接信息来构造factory。

2. 利用factory构造JMS connection

3. 启动connection

4. 通过connection创建JMS session.

5. 指定JMS destination.

6. 创建JMS producer或者创建JMS message并提供destination.

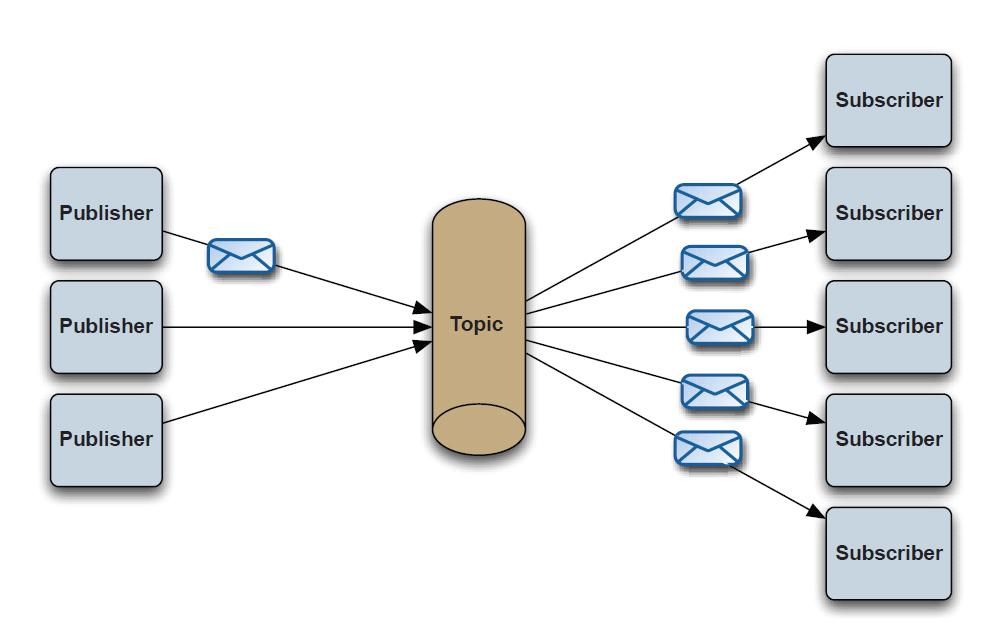
7. 创建JMS consumer或注册JMS message listener.

8. 发送和接收JMS message.

9. 关闭所有JMS资源，包括connection, session, producer, consumer等。

**publish-subscribe**

     发布订阅模式有点类似于我们日常生活中订阅报纸。每年到年尾的时候，邮局就会发一本报纸集合让我们来选择订阅哪一个。在这个表里头列了所有出版发行的报 纸，那么对于我们每一个订阅者来说，我们可以选择一份或者多份报纸。比如北京日报、潇湘晨报等。那么这些个我们订阅的报纸，就相当于发布订阅模式里的 topic。有很多个人订阅报纸，也有人可能和我订阅了相同的报纸。那么，在这里，相当于我们在同一个topic里注册了。对于一份报纸发行方来说，它和 所有的订阅者就构成了一个1对多的关系。这种关系如下图所示：



     现在，假定我们用前面讨论的场景来写一个简单的示例。我们首先需要定义的是publisher.

**publisher**

     publisher是属于发布信息的一方，它通过定义一个或者多个topic，然后给这些topic发送消息。

    publisher的构造函数如下：

Java代码  [收藏代码](javascript:void())

1. public Publisher() throws JMSException {
2. factory = new ActiveMQConnectionFactory(brokerURL);
3. connection = factory.createConnection();
4. try {
5. connection.start();
6. } catch (JMSException jmse) {
7. connection.close();
8. throw jmse;
9. }
10. session = connection.createSession(false, Session.AUTO\_ACKNOWLEDGE);
11. producer = session.createProducer(null);
12. }

     我们按照前面说的流程定义了基本的connectionFactory, connection, session, producer。这里代码就是主要实现初始化的效果。

    接着，我们需要定义一系列的topic让所有的consumer来订阅，设置topic的代码如下：

Java代码  [收藏代码](javascript:void())

1. protected void setTopics(String[] stocks) throws JMSException {
2. destinations = new Destination[stocks.length];
3. for(int i = 0; i < stocks.length; i++) {
4. destinations[i] = session.createTopic("STOCKS." + stocks[i]);
5. }
6. }

     这里destinations是一个内部定义的成员变量Destination[]。这里我们总共定义了的topic数取决于给定的参数stocks。

     在定义好topic之后我们要给这些指定的topic发消息，具体实现的代码如下：

Java代码  [收藏代码](javascript:void())

1. protected void sendMessage(String[] stocks) throws JMSException {
2. for(int i = 0; i < stocks.length; i++) {
3. Message message = createStockMessage(stocks[i], session);
4. System.out.println("Sending: " + ((ActiveMQMapMessage)message).getContentMap() + " on destination: " + destinations[i]);
5. producer.send(destinations[i], message);
6. }
7. }
9. protected Message createStockMessage(String stock, Session session) throws JMSException {
10. MapMessage message = session.createMapMessage();
11. message.setString("stock", stock);
12. message.setDouble("price", 1.00);
13. message.setDouble("offer", 0.01);
14. message.setBoolean("up", true);
16. return message;
17. }

     前面的代码很简单，在sendMessage方法里我们遍历每个topic，然后给每个topic发送定义的Message消息。

    在定义好前面发送消息的基础之后，我们调用他们的代码就很简单了：

Java代码  [收藏代码](javascript:void())

1. public static void main(String[] args) throws JMSException {
2. if(args.length < 1)
3. throw new IllegalArgumentException();
5. // Create publisher
6. Publisher publisher = new Publisher();
8. // Set topics
9. publisher.setTopics(args);
11. for(int i = 0; i < 10; i++) {
12. publisher.sendMessage(args);
13. System.out.println("Publisher '" + i + " price messages");
14. try {
15. Thread.sleep(1000);
16. } catch(InterruptedException e) {
17. e.printStackTrace();
18. }
19. }
20. // Close all resources
21. publisher.close();
22. }

     调用他们的代码就是我们遍历所有topic，然后通过sendMessage发送消息。在发送一个消息之后先sleep1秒钟。要注意的一个地方就是我们使用完资源之后必须要使用close方法将这些资源关闭释放。close方法关闭资源的具体实现如下：

Java代码  [收藏代码](javascript:void())

1. public void close() throws JMSException {
2. if (connection != null) {
3. connection.close();
4. }
5. }

**consumer**

    Consumer的代码也很类似，具体的步骤无非就是1.初始化资源。 2. 接收消息。 3. 必要的时候关闭资源。

    初始化资源可以放到构造函数里面：

Java代码  [收藏代码](javascript:void())

1. public Consumer() throws JMSException {
2. factory = new ActiveMQConnectionFactory(brokerURL);
3. connection = factory.createConnection();
4. connection.start();
5. session = connection.createSession(false, Session.AUTO\_ACKNOWLEDGE);
6. }

     接收和处理消息的方法有两种，分为同步和异步的，一般同步的方式我们是通过MessageConsumer.receive()方法来处理接收到的消 息。而异步的方法则是通过注册一个MessageListener的方法，使用 MessageConsumer.setMessageListener()。这里我们采用异步的方式实现：

Java代码  [收藏代码](javascript:void())

1. public static void main(String[] args) throws JMSException {
2. Consumer consumer = new Consumer();
3. for (String stock : args) {
4. Destination destination = consumer.getSession().createTopic("STOCKS." + stock);
5. MessageConsumer messageConsumer = consumer.getSession().createConsumer(destination);
6. messageConsumer.setMessageListener(new Listener());
7. }
8. }
10. public Session getSession() {
11. return session;
12. }

     在前面的代码里我们先找到同样的topic,然后遍历所有的topic去获得消息。对于消息的处理我们专门通过Listener对象来负责。

    Listener对象的职责很简单，主要就是处理接收到的消息：

Java代码  [收藏代码](javascript:void())

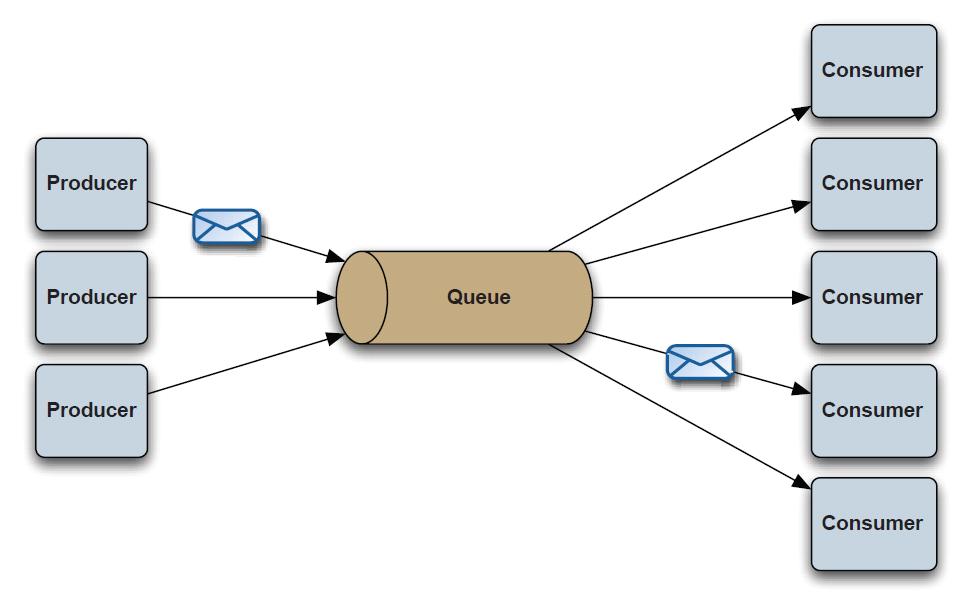
1. public class Listener implements MessageListener {
3. public void onMessage(Message message) {
4. try {
5. MapMessage map = (MapMessage)message;
6. String stock = map.getString("stock");
7. double price = map.getDouble("price");
8. double offer = map.getDouble("offer");
9. boolean up = map.getBoolean("up");
10. DecimalFormat df = new DecimalFormat( "#,###,###,##0.00" );
11. System.out.println(stock + "\t" + df.format(price) + "\t" + df.format(offer) + "\t" + (up?"up":"down"));
12. } catch (Exception e) {
13. e.printStackTrace();
14. }
15. }
17. }

    它实现了MessageListener接口，里面的onMessage方法就是在接收到消息之后会被调用的方法。

    现在，通过实现前面的publisher和consumer我们已经实现了pub-sub模式的一个实例。仔细回想它的步骤的话，主要就是要两者设定一个 共同的topic，有了这个topic之后他们可以实现一方发消息另外一方接收。另外，为了连接到具体的message server,这里是使用了连接tcp://localhost:16161作为定义ActiveMQConnectionFactory的路径。在 publisher端通过session创建producer，根据指定的参数创建destination，然后将消息和destination作为 producer.send()方法的参数发消息。在consumer端也要创建类似的connection, session。通过session得到destination，再通过session.createConsumer(destination)来得到 一个MessageConsumer对象。有了这个MessageConsumer我们就可以自行选择是直接同步的receive消息还是注册 listener了。

**p2p**

    p2p的过程则理解起来更加简单。它好比是两个人打电话，这两个人是独享这一条通信链路的。一方发送消息，另外一方接收，就这么简单。在实际应用中因为有多个用户对使用p2p的链路，它的通信场景如下图所示：



    我们再来看看一个p2p的示例：

    在p2p的场景里，相互通信的双方是通过一个类似于队列的方式来进行交流。和前面pub-sub的区别在于一个topic有一个发送者和多个接收者，而在p2p里一个queue只有一个发送者和一个接收者。

**发送者**

    和前面的示例非常相似，我们构造函数里需要初始化的内容基本上差不多：

Java代码  [收藏代码](javascript:void())

1. public Publisher() throws JMSException {
2. factory = new ActiveMQConnectionFactory(brokerURL);
3. connection = factory.createConnection();
4. connection.start();
5. session = connection.createSession(false, Session.AUTO\_ACKNOWLEDGE);
6. producer = session.createProducer(null);
7. }

     发送消息的方法如下：

Java代码  [收藏代码](javascript:void())

1. public void sendMessage() throws JMSException {
2. for(int i = 0; i < jobs.length; i++)
3. {
4. String job = jobs[i];
5. Destination destination = session.createQueue("JOBS." + job);
6. Message message = session.createObjectMessage(i);
7. System.out.println("Sending: id: " + ((ObjectMessage)message).getObject() + " on queue: " + destination);
8. producer.send(destination, message);
9. }
10. }

     这里我们定义了一个jobs的数组，通过遍历这个数组来创建不同的job queue。这样就相当于建立了多个点对点通信的链路。

    消息发送者的启动代码如下：

Java代码  [收藏代码](javascript:void())

1. public static void main(String[] args) throws JMSException {
2. Publisher publisher = new Publisher();
3. for(int i = 0; i < 10; i++) {
4. publisher.sendMessage();
5. System.out.println("Published " + i + " job messages");
6. try {
7. Thread.sleep(1000);
8. } catch (InterruptedException x) {
9. e.printStackTrace();
10. }
11. }
12. publisher.close();
13. }

     我们在这里发送10条消息，当然，在每个sendMessage的方法里实际上是针对每个queue发送了10条。

**接收者**

     接收者的代码很简单，一个构造函数初始化所有的资源：

Java代码  [收藏代码](javascript:void())

1. public Consumer() throws JMSException {
2. factory = new ActiveMQConnectionFactory(brokerURL);
3. connection = factory.createConnection();
4. connection.start();
5. session = connection.createSession(false, Session.AUTO\_ACKNOWLEDGE);
6. }

    还有一个就是注册消息处理的对象：

Java代码  [收藏代码](javascript:void())

1. public static void main(String[] args) throws JMSException {
2. Consumer consumer = new Consumer();
3. for (String job : consumer.jobs) {
4. Destination destination = consumer.getSession().createQueue("JOBS." + job);
5. MessageConsumer messageConsumer = consumer.getSession().createConsumer(destination);
6. messageConsumer.setMessageListener(new Listener(job));
7. }
8. }
10. public Session getSession() {
11. return session;
12. }

     具体注册的对象处理方法和前面还是类似，实现MessageListener接口就可以了。

Java代码  [收藏代码](javascript:void())

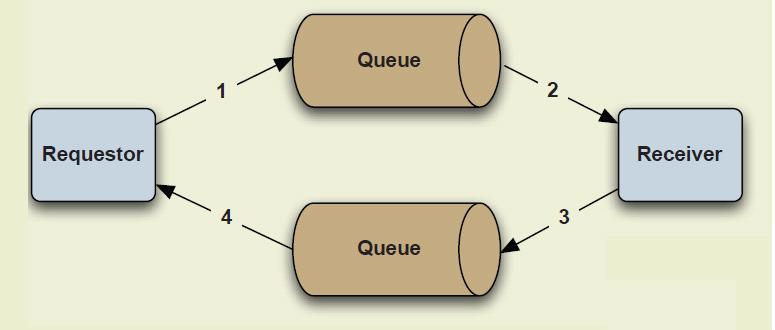
1. import javax.jms.Message;
2. import javax.jms.MessageListener;
3. import javax.jms.ObjectMessage;
5. public class Listener implements MessageListener {
7. private String job;
9. public Listener(String job) {
10. this.job = job;
11. }
13. public void onMessage(Message message) {
14. try {
15. //do something here
16. System.out.println(job + " id:" + ((ObjectMessage)message).getObject());
17. } catch (Exception e) {
18. e.printStackTrace();
19. }
20. }
22. }

     这里代码和前面pub-sub的具体实现代码非常相似，就不再赘述。

     现在如果我们比较一下pub-sub和p2p模式的具体实现步骤的话，我们会发现他们基本的处理流程都是类似的，除了在pub-sub中要通过 createTopic来设置topic，而在p2p中要通过createQueue来创建通信队列。他们之间存在着很多的重复之处，在具体的开发过程 中，我们是否可以进行一些工程上的优化呢？别急，后面我们会讨论到的。

**request-response**

    和前面两种方式比较起来，request-response的通信方式很常见，但是不是默认提供的一种模式。在前面的两种模式中都是一方负责发送消息而另 外一方负责处理。而我们实际中的很多应用相当于一种一应一答的过程，需要双方都能给对方发送消息。于是请求-应答的这种通信方式也很重要。它也应用的很普 遍。

     请求-应答方式并不是JMS规范系统默认提供的一种通信方式，而是通过在现有通信方式的基础上稍微运用一点技巧实现的。下图是典型的请求-应答方式的交互过程：

     在JMS里面，如果要实现请求/应答的方式，可以利用JMSReplyTo和JMSCorrelationID消息头来将通信的双方关联起来。另外，QueueRequestor和TopicRequestor能够支持简单的请求/应答过程。

    現在，如果我们要实现这么一个过程，在发送请求消息并且等待返回结果的client端的流程如下：

Java代码  [收藏代码](javascript:void())

1. // client side
2. Destination tempDest = session.createTemporaryQueue();
3. MessageConsumer responseConsumer = session.createConsumer(tempDest);
4. ...
6. // send a request..
7. message.setJMSReplyTo(tempDest)
8. message.setJMSCorrelationID(myCorrelationID);
10. producer.send(message);

     client端创建一个临时队列并在发送的消息里指定了发送返回消息的destination以及correlationID。那么在处理消息的server端得到这个消息后就知道该发送给谁了。Server端的大致流程如下：

Java代码  [收藏代码](javascript:void())

1. public void onMessage(Message request) {
3. Message response = session.createMessage();
4. response.setJMSCorrelationID(request.getJMSCorrelationID())
6. producer.send(request.getJMSReplyTo(), response)
7. }

    这里我们是用server端注册MessageListener，通过设置返回信息的CorrelationID和JMSReplyTo将信息返回。

    以上就是发送和接收消息的双方的大致程序结构。具体的实现代码如下：

**Client:**

Java代码  [收藏代码](javascript:void())

1. public Client() {
2. ActiveMQConnectionFactory connectionFactory = new ActiveMQConnectionFactory("tcp://localhost:61616");
3. Connection connection;
4. try {
5. connection = connectionFactory.createConnection();
6. connection.start();
7. Session session = connection.createSession(transacted, ackMode);
8. Destination adminQueue = session.createQueue(clientQueueName);
10. //Setup a message producer to send message to the queue the server is consuming from
11. this.producer = session.createProducer(adminQueue);
12. this.producer.setDeliveryMode(DeliveryMode.NON\_PERSISTENT);
14. //Create a temporary queue that this client will listen for responses on then create a consumer
15. //that consumes message from this temporary queue...for a real application a client should reuse
16. //the same temp queue for each message to the server...one temp queue per client
17. Destination tempDest = session.createTemporaryQueue();
18. MessageConsumer responseConsumer = session.createConsumer(tempDest);
20. //This class will handle the messages to the temp queue as well
21. responseConsumer.setMessageListener(this);
23. //Now create the actual message you want to send
24. TextMessage txtMessage = session.createTextMessage();
25. txtMessage.setText("MyProtocolMessage");
27. //Set the reply to field to the temp queue you created above, this is the queue the server
28. //will respond to
29. txtMessage.setJMSReplyTo(tempDest);
31. //Set a correlation ID so when you get a response you know which sent message the response is for
32. //If there is never more than one outstanding message to the server then the
33. //same correlation ID can be used for all the messages...if there is more than one outstanding
34. //message to the server you would presumably want to associate the correlation ID with this
35. //message somehow...a Map works good
36. String correlationId = this.createRandomString();
37. txtMessage.setJMSCorrelationID(correlationId);
38. this.producer.send(txtMessage);
39. } catch (JMSException e) {
40. //Handle the exception appropriately
41. }
42. }

    这里的代码除了初始化构造函数里的参数还同时设置了两个destination，一个是自己要发送消息出去的destination，在 session.createProducer(adminQueue);这一句设置。另外一个是自己要接收的消息destination, 通过Destination tempDest = session.createTemporaryQueue(); responseConsumer = session.createConsumer(tempDest); 这两句指定了要接收消息的目的地。这里是用的一个临时队列。在前面指定了返回消息的通信队列之后，我们需要通知server端知道发送返回消息给哪个队 列。于是txtMessage.setJMSReplyTo(tempDest);指定了这一部分，同时 txtMessage.setJMSCorrelationID(correlationId);方法主要是为了保证每次发送回来请求的server端能 够知道对应的是哪个请求。这里一个请求和一个应答是相当于对应一个相同的序列号一样。

    同时，因为client端在发送消息之后还要接收server端返回的消息，所以它也要实现一个消息receiver的功能。这里采用实现MessageListener接口的方式：

Java代码  [收藏代码](javascript:void())

1. public void onMessage(Message message) {
2. String messageText = null;
3. try {
4. if (message instanceof TextMessage) {
5. TextMessage textMessage = (TextMessage) message;
6. messageText = textMessage.getText();
7. System.out.println("messageText = " + messageText);
8. }
9. } catch (JMSException e) {
10. //Handle the exception appropriately
11. }
12. }

**Server:**

     这里server端要执行的过程和client端相反，它是先接收消息，在接收到消息后根据提供的JMSCorelationID来发送返回的消息：

Java代码  [收藏代码](javascript:void())

1. public void onMessage(Message message) {
2. try {
3. TextMessage response = this.session.createTextMessage();
4. if (message instanceof TextMessage) {
5. TextMessage txtMsg = (TextMessage) message;
6. String messageText = txtMsg.getText();
7. response.setText(this.messageProtocol.handleProtocolMessage(messageText));
8. }
10. //Set the correlation ID from the received message to be the correlation id of the response message
11. //this lets the client identify which message this is a response to if it has more than
12. //one outstanding message to the server
13. response.setJMSCorrelationID(message.getJMSCorrelationID());
15. //Send the response to the Destination specified by the JMSReplyTo field of the received message,
16. //this is presumably a temporary queue created by the client
17. this.replyProducer.send(message.getJMSReplyTo(), response);
18. } catch (JMSException e) {
19. //Handle the exception appropriately
20. }
21. }

    前面，在replyProducer.send()方法里，message.getJMSReplyTo()就得到了要发送消息回去的destination。

    另外，设置这些发送返回信息的replyProducer的信息主要在构造函数相关的方法里实现了：

Java代码  [收藏代码](javascript:void())

1. public Server() {
2. try {
3. //This message broker is embedded
4. BrokerService broker = new BrokerService();
5. broker.setPersistent(false);
6. broker.setUseJmx(false);
7. broker.addConnector(messageBrokerUrl);
8. broker.start();
9. } catch (Exception e) {
10. //Handle the exception appropriately
11. }
13. //Delegating the handling of messages to another class, instantiate it before setting up JMS so it
14. //is ready to handle messages
15. this.messageProtocol = new MessageProtocol();
16. this.setupMessageQueueConsumer();
17. }
19. private void setupMessageQueueConsumer() {
20. ActiveMQConnectionFactory connectionFactory = new ActiveMQConnectionFactory(messageBrokerUrl);
21. Connection connection;
22. try {
23. connection = connectionFactory.createConnection();
24. connection.start();
25. this.session = connection.createSession(this.transacted, ackMode);
26. Destination adminQueue = this.session.createQueue(messageQueueName);
28. //Setup a message producer to respond to messages from clients, we will get the destination
29. //to send to from the JMSReplyTo header field from a Message
30. this.replyProducer = this.session.createProducer(null);
31. this.replyProducer.setDeliveryMode(DeliveryMode.NON\_PERSISTENT);
33. //Set up a consumer to consume messages off of the admin queue
34. MessageConsumer consumer = this.session.createConsumer(adminQueue);
35. consumer.setMessageListener(this);
36. } catch (JMSException e) {
37. //Handle the exception appropriately
38. }
39. }

    总体来说，整个的交互过程并不复杂，只是比较繁琐。对于请求/应答的方式来说，这种典型交互的过程就是Client端在设定正常发送请求的Queue同时 也设定一个临时的Queue。同时在要发送的message里头指定要返回消息的destination以及CorelationID，这些就好比是一封 信里面所带的回执。根据这个信息人家才知道怎么给你回信。对于Server端来说则要额外创建一个producer，在处理接收到消息的方法里再利用 producer将消息发回去。这一系列的过程看起来很像http协议里面请求-应答的方式，都是一问一答。

**一些应用和改进**

    回顾前面三种基本的通信方式，我们会发现，他们都存在着一定的共同点，比如说都要初始化ConnectionFactory, Connection, Session等。在使用完之后都要将这些资源关闭。如果每一个实现它的通信端都这么写一通的话，其实是一种简单的重复。从工程的角度来看是完全没有必要 的。那么，我们有什么办法可以减少这种重复呢？

    一种简单的方式就是通过工厂方法封装这些对象的创建和销毁，然后简单的通过调用工厂方法的方式得到他们。另外，既然基本的流程都是在开头创建资源在结尾销 毁，我们也可以采用Template Method模式的思路。通过继承一个抽象类，在抽象类里提供了资源的封装。所有继承的类只要实现怎么去使用这些资源的方法就可以了。Spring中间的 JMSTemplate就提供了这种类似思想的封装。具体的实现可以参考[这篇文章](http://zorro.blog.51cto.com/2139862/831986)。

**总结**

     activemq默认提供了pub-sub, p2p这两种通信的方式。同时也提供了一些对request-response方式的支持。实际上，不仅仅是activemq，对于所有其他实现JMS规 范的产品都能够提供类似的功能。这里每种方式都不太复杂，主要是创建和管理资源的步骤显得比较繁琐。