[**Spring整合JMS(二)——三种消息监听器**](http://elim.iteye.com/blog/1893676)

**博客分类：**

* [Spring](http://elim.iteye.com/category/156657)
* [Jms](http://elim.iteye.com/category/347649)

[MessageListener](http://www.iteye.com/blogs/tag/MessageListener)[SessionAwareMessageListener](http://www.iteye.com/blogs/tag/SessionAwareMessageListener)[MessageListenerAdapter](http://www.iteye.com/blogs/tag/MessageListenerAdapter)[Spring](http://www.iteye.com/blogs/tag/Spring)[JMS](http://www.iteye.com/blogs/tag/JMS)

**1.3     消息监听器MessageListener**

       在Spring整合JMS的应用中我们在定义消息监听器的时候一共可以定义三种类型的消息监听器，分别是MessageListener、SessionAwareMessageListener和MessageListenerAdapter。下面就分别来介绍一下这几种类型的区别。

**1.3.1**  **MessageListener**

MessageListener是最原始的消息监听器，它是JMS规范中定义的一个接口。其中定义了一个用于处理接收到的消息的onMessage方法，该方法只接收一个Message参数。我们前面在讲配置消费者的时候用的消息监听器就是MessageListener，代码如下：

**Java代码  [收藏代码](javascript:void())**

1. **import** javax.jms.JMSException;
2. **import** javax.jms.Message;
3. **import** javax.jms.MessageListener;
4. **import** javax.jms.TextMessage;
6. **public** **class** ConsumerMessageListener **implements** MessageListener {
8. **public** **void** onMessage(Message message) {
9. //这里我们知道生产者发送的就是一个纯文本消息，所以这里可以直接进行强制转换，或者直接把onMessage方法的参数改成Message的子类TextMessage
10. TextMessage textMsg = (TextMessage) message;
11. System.out.println("接收到一个纯文本消息。");
12. **try** {
13. System.out.println("消息内容是：" + textMsg.getText());
14. } **catch** (JMSException e) {
15. e.printStackTrace();
16. }
17. }
19. }

**1.3.2**  **SessionAwareMessageListener**

SessionAwareMessageListener是Spring为我们提供的，它不是标准的JMS MessageListener。MessageListener的设计只是纯粹用来接收消息的，假如我们在使用MessageListener处理接收到的消息时我们需要发送一个消息通知对方我们已经收到这个消息了，那么这个时候我们就需要在代码里面去重新获取一个Connection或Session。SessionAwareMessageListener的设计就是为了方便我们在接收到消息后发送一个回复的消息，它同样为我们提供了一个处理接收到的消息的onMessage方法，但是这个方法可以同时接收两个参数，一个是表示当前接收到的消息Message，另一个就是可以用来发送消息的Session对象。先来看一段代码：

**Java代码  [收藏代码](javascript:void())**

1. **package** com.tiantian.springintejms.listener;
3. **import** javax.jms.Destination;
4. **import** javax.jms.JMSException;
5. **import** javax.jms.Message;
6. **import** javax.jms.MessageProducer;
7. **import** javax.jms.Session;
8. **import** javax.jms.TextMessage;
10. **import** org.springframework.jms.listener.SessionAwareMessageListener;
12. **public** **class** ConsumerSessionAwareMessageListener **implements**
13. SessionAwareMessageListener<TextMessage> {
15. **private** Destination destination;
17. **public** **void** onMessage(TextMessage message, Session session) **throws** JMSException {
18. System.out.println("收到一条消息");
19. System.out.println("消息内容是：" + message.getText());
20. MessageProducer producer = session.createProducer(destination);
21. Message textMessage = session.createTextMessage("ConsumerSessionAwareMessageListener。。。");
22. producer.send(textMessage);
23. }
25. **public** Destination getDestination() {
26. returndestination;
27. }
29. **public** **void** setDestination(Destination destination) {
30. **this**.destination = destination;
31. }
33. }

       在上面代码中我们定义了一个SessionAwareMessageListener，在这个Listener中我们在接收到了一个消息之后，利用对应的Session创建了一个到destination的生产者和对应的消息，然后利用创建好的生产者发送对应的消息。

       接着我们在Spring的配置文件中配置该消息监听器将处理来自一个叫sessionAwareQueue的目的地的消息，并且往该MessageListener中通过set方法注入其属性destination的值为queueDestination。这样当我们的SessionAwareMessageListener接收到消息之后就会往queueDestination发送一个消息。

**Xml代码  [收藏代码](javascript:void())**

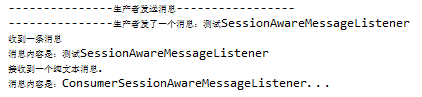
1. **<?xml** version="1.0" encoding="UTF-8"**?>**
2. **<beans** xmlns="http://www.springframework.org/schema/beans"
3. xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:context="http://www.springframework.org/schema/context"
4. xmlns:jms="http://www.springframework.org/schema/jms"
5. xsi:schemaLocation="http://www.springframework.org/schema/beans
6. http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
7. http://www.springframework.org/schema/context
8. http://www.springframework.org/schema/context/spring-context-3.0.xsd
9. http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
10. http://www.springframework.org/schema/jms http://www.springframework.org/schema/jms/spring-jms-3.0.xsd"**>**
12. **<context:component-scan** base-package="com.tiantian" **/>**
13. <!-- Spring提供的JMS工具类，它可以进行消息发送、接收等 -->
14. **<bean** id="jmsTemplate" class="org.springframework.jms.core.JmsTemplate"**>**
15. <!-- 这个connectionFactory对应的是我们定义的Spring提供的那个ConnectionFactory对象 -->
16. **<property** name="connectionFactory" ref="connectionFactory"**/>**
17. **</bean>**
19. <!-- 真正可以产生Connection的ConnectionFactory，由对应的 JMS服务厂商提供-->
20. **<bean** id="targetConnectionFactory" class="org.apache.activemq.ActiveMQConnectionFactory"**>**
21. **<property** name="brokerURL" value="tcp://localhost:61616"**/>**
22. **</bean>**
24. <!-- Spring用于管理真正的ConnectionFactory的ConnectionFactory -->
25. **<bean** id="connectionFactory" class="org.springframework.jms.connection.SingleConnectionFactory"**>**
26. <!-- 目标ConnectionFactory对应真实的可以产生JMS Connection的ConnectionFactory -->
27. **<property** name="targetConnectionFactory" ref="targetConnectionFactory"**/>**
28. **</bean>**
30. <!--这个是队列目的地-->
31. **<bean** id="queueDestination" class="org.apache.activemq.command.ActiveMQQueue"**>**
32. **<constructor-arg>**
33. **<value>**queue**</value>**
34. **</constructor-arg>**
35. **</bean>**
36. <!--这个是sessionAwareQueue目的地-->
37. **<bean** id="sessionAwareQueue" class="org.apache.activemq.command.ActiveMQQueue"**>**
38. **<constructor-arg>**
39. **<value>**sessionAwareQueue**</value>**
40. **</constructor-arg>**
41. **</bean>**
42. <!-- 消息监听器 -->
43. **<bean** id="consumerMessageListener" class="com.tiantian.springintejms.listener.ConsumerMessageListener"**/>**
44. <!-- 可以获取session的MessageListener -->
45. **<bean** id="consumerSessionAwareMessageListener" class="com.tiantian.springintejms.listener.ConsumerSessionAwareMessageListener"**>**
46. **<property** name="destination" ref="queueDestination"**/>**
47. **</bean>**
48. <!-- 消息监听容器 -->
49. **<bean** id="jmsContainer"        class="org.springframework.jms.listener.DefaultMessageListenerContainer"**>**
50. **<property** name="connectionFactory" ref="connectionFactory" **/>**
51. **<property** name="destination" ref="queueDestination" **/>**
52. **<property** name="messageListener" ref="consumerMessageListener" **/>**
53. **</bean>**
55. **<bean** id="sessionAwareListenerContainer"
56. class="org.springframework.jms.listener.DefaultMessageListenerContainer"**>**
57. **<property** name="connectionFactory" ref="connectionFactory" **/>**
58. **<property** name="destination" ref="sessionAwareQueue" **/>**
59. **<property** name="messageListener" ref="consumerSessionAwareMessageListener" **/>**
60. **</bean>**
61. **</beans>**

       接着我们来做一个测试，测试代码如下：

**Java代码  [收藏代码](javascript:void())**

1. @RunWith(SpringJUnit4ClassRunner.**class**)
2. @ContextConfiguration("/applicationContext.xml")
3. **public** **class** ProducerConsumerTest {
5. @Autowired
6. **private** ProducerService producerService;
7. @Autowired
8. @Qualifier("sessionAwareQueue")
9. **private** Destination sessionAwareQueue;
11. @Test
12. **public** **void** testSessionAwareMessageListener() {
13. producerService.sendMessage(sessionAwareQueue, "测试SessionAwareMessageListener");
14. }
16. }

       在上述测试代码中，我们通过前面定义好的生产者往我们定义好的SessionAwareMessageListener监听的sessionAwareQueue发送了一个消息。程序运行之后控制台输出如下：



       这说明我们已经成功的往sessionAwareQueue发送了一条纯文本消息，消息会被ConsumerSessionAwareMessageListener的onMessage方法进行处理，在onMessage方法中ConsumerSessionAwareMessageListener就是简单的把接收到的纯文本信息的内容打印出来了，之后再往queueDestination发送了一个纯文本消息，消息内容是“ConsumerSessionAwareMessageListener…”，该消息随后就被ConsumerMessageListener处理了，根据我们的定义，在ConsumerMessageListener中也只是简单的打印了一下接收到的消息内容。

**1.3.3**  **MessageListenerAdapter**

MessageListenerAdapter类实现了MessageListener接口和SessionAwareMessageListener接口，它的主要作用是将接收到的消息进行类型转换，然后通过反射的形式把它交给一个普通的Java类进行处理。

       MessageListenerAdapter会把接收到的消息做如下转换：

       TextMessage转换为String对象；

       BytesMessage转换为byte数组；

       MapMessage转换为Map对象；

       ObjectMessage转换为对应的Serializable对象。

       既然前面说了MessageListenerAdapter会把接收到的消息做一个类型转换，然后利用反射把它交给真正的目标处理器——一个普通的Java类进行处理（如果真正的目标处理器是一个MessageListener或者是一个SessionAwareMessageListener，那么Spring将直接使用接收到的Message对象作为参数调用它们的onMessage方法，而不会再利用反射去进行调用），那么我们在定义一个MessageListenerAdapter的时候就需要为它指定这样一个目标类。这个目标类我们可以通过MessageListenerAdapter的构造方法参数指定，如：

**Xml代码  [收藏代码](javascript:void())**

1. <!-- 消息监听适配器 -->
2. **<bean** id="messageListenerAdapter" class="org.springframework.jms.listener.adapter.MessageListenerAdapter"**>**
3. **<constructor-arg>**
4. **<bean** class="com.tiantian.springintejms.listener.ConsumerListener"**/>**
5. **</constructor-arg>**
6. **</bean>**

       也可以通过它的delegate属性来指定，如：

**Xml代码  [收藏代码](javascript:void())**

1. <!-- 消息监听适配器 -->
2. **<bean** id="messageListenerAdapter" class="org.springframework.jms.listener.adapter.MessageListenerAdapter"**>**
3. **<property** name="delegate"**>**
4. **<bean** class="com.tiantian.springintejms.listener.ConsumerListener"**/>**
5. **</property>**
6. **<property** name="defaultListenerMethod" value="receiveMessage"**/>**
7. **</bean>**

       前面说了如果我们指定的这个目标处理器是一个MessageListener或者是一个SessionAwareMessageListener的时候Spring将直接利用接收到的Message对象作为方法参数调用它们的onMessage方法。但是如果指定的目标处理器是一个普通的Java类时Spring将利用Message进行了类型转换之后的对象作为参数通过反射去调用真正的目标处理器的处理方法，那么Spring是如何知道该调用哪个方法呢？这是通过MessageListenerAdapter的defaultListenerMethod属性来决定的，当我们没有指定该属性时，Spring会默认调用目标处理器的handleMessage方法。

       接下来我们来看一个示例，假设我们有一个普通的Java类ConsumerListener，其对应有两个方法，handleMessage和receiveMessage，其代码如下：

**Java代码  [收藏代码](javascript:void())**

1. **package** com.tiantian.springintejms.listener;
3. **public** **class** ConsumerListener {
5. **public** **void** handleMessage(String message) {
6. System.out.println("ConsumerListener通过handleMessage接收到一个纯文本消息，消息内容是：" + message);
7. }
9. **public** **void** receiveMessage(String message) {
10. System.out.println("ConsumerListener通过receiveMessage接收到一个纯文本消息，消息内容是：" + message);
11. }
13. }

       假设我们要把它作为一个消息监听器来监听发送到adapterQueue的消息，这个时候我们就可以定义一个对应的MessageListenerAdapter来把它当做一个MessageListener使用。

**Xml代码  [收藏代码](javascript:void())**

1. <!-- 消息监听适配器 -->
2. **<bean** id="messageListenerAdapter" class="org.springframework.jms.listener.adapter.MessageListenerAdapter"**>**
3. **<property** name="delegate"**>**
4. **<bean** class="com.tiantian.springintejms.listener.ConsumerListener"**/>**
5. **</property>**
6. **<property** name="defaultListenerMethod" value="receiveMessage"**/>**
7. **</bean>**

       当然，有了MessageListener之后我们还需要配置其对应的MessageListenerContainer，这里配置如下：

**Xml代码  [收藏代码](javascript:void())**

1. <!-- 消息监听适配器对应的监听容器 -->
2. **<bean** id="messageListenerAdapterContainer" class="org.springframework.jms.listener.DefaultMessageListenerContainer"**>**
3. **<property** name="connectionFactory" ref="connectionFactory"**/>**
4. **<property** name="destination" ref="adapterQueue"**/>**
5. **<property** name="messageListener" ref="messageListenerAdapter"**/>**<!-- 使用MessageListenerAdapter来作为消息监听器 -->
6. **</bean>**
8. <!-- 用于测试消息监听适配器的队列目的地 -->
9. **<bean** id="adapterQueue" class="org.apache.activemq.command.ActiveMQQueue"**>**
10. **<constructor-arg>**
11. **<value>**adapterQueue**</value>**
12. **</constructor-arg>**
13. **</bean>**

       在上面的MessageListenerAdapter中我们指定了其defaultListenerMethod属性的值为receiveMessage，所以当MessageListenerAdapter接收到消息之后会自动的调用我们指定的ConsumerListener的receiveMessage方法。

       针对于上述代码我们定义测试代码如下：

**Java代码  [收藏代码](javascript:void())**

1. **package** com.tiantian.springintejms.test;
3. **import** javax.jms.Destination;
5. **import** org.junit.Test;
6. **import** org.junit.runner.RunWith;
7. **import** org.springframework.beans.factory.annotation.Autowired;
8. **import** org.springframework.beans.factory.annotation.Qualifier;
9. **import** org.springframework.test.context.ContextConfiguration;
10. **import** org.springframework.test.context.junit4.SpringJUnit4ClassRunner;
12. **import** com.tiantian.springintejms.service.ProducerService;
14. @RunWith(SpringJUnit4ClassRunner.**class**)
15. @ContextConfiguration("/applicationContext.xml")
16. **public** **class** ProducerConsumerTest {
18. @Autowired
19. @Qualifier("adapterQueue")
20. **private** Destination adapterQueue;
22. @Test
23. **public** **void** testMessageListenerAdapter() {
24. producerService.sendMessage(adapterQueue, "测试MessageListenerAdapter");
25. }
27. }

       这时候我们会看到控制台输出如下：

http://dl2.iteye.com/upload/attachment/0086/1540/ed428c5f-4b65-31fa-bc6c-6b5940df5b99.png  
        如果我们不指定MessageListenerAdapter的defaultListenerMethod属性，那么在运行上述代码时控制台会输出如下结果：

http://dl2.iteye.com/upload/attachment/0086/1542/d0270ffa-b3e3-3405-8181-83e9cd80574b.png  
        MessageListenerAdapter除了会自动的把一个普通Java类当做MessageListener来处理接收到的消息之外，**其另外一个主要的功能是可以自动的发送返回消息**。

     当我们用于处理接收到的消息的方法的返回值不为空的时候，Spring会自动将它封装为一个JMS Message，然后自动进行回复。那么这个时候这个回复消息将发送到哪里呢？这主要有两种方式可以指定。  
       第一，可以通过发送的Message的setJMSReplyTo方法指定该消息对应的回复消息的目的地。这里我们把我们的生产者发送消息的代码做一下修改，在发送消息之前先指定该消息对应的回复目的地为一个叫responseQueue的队列目的地，具体代码如下所示：

**Java代码  [收藏代码](javascript:void())**

1. **package** com.tiantian.springintejms.service.impl;
3. **import** javax.jms.Destination;
4. **import** javax.jms.JMSException;
5. **import** javax.jms.Message;
6. **import** javax.jms.Session;
7. **import** javax.jms.TextMessage;
9. **import** org.springframework.beans.factory.annotation.Autowired;
10. **import** org.springframework.beans.factory.annotation.Qualifier;
11. **import** org.springframework.jms.core.JmsTemplate;
12. **import** org.springframework.jms.core.MessageCreator;
13. **import** org.springframework.stereotype.Component;
15. **import** com.tiantian.springintejms.service.ProducerService;
17. @Component
18. **public** **class** ProducerServiceImpl **implements** ProducerService {
20. @Autowired
21. **private** JmsTemplate jmsTemplate;
23. @Autowired
24. @Qualifier("responseQueue")
25. **private** Destination responseDestination;
27. **public** **void** sendMessage(Destination destination, **final** String message) {
28. System.out.println("---------------生产者发送消息-----------------");
29. System.out.println("---------------生产者发了一个消息：" + message);
30. jmsTemplate.send(destination, **new** MessageCreator() {
31. **public** Message createMessage(Session session) **throws** JMSException {
32. TextMessage textMessage = session.createTextMessage(message);
33. textMessage.setJMSReplyTo(responseDestination);
34. **return** textMessage;
35. }
36. });
37. }
39. }

       接着定义一个叫responseQueue的队列目的地及其对应的消息监听器和监听容器。

**Xml代码  [收藏代码](javascript:void())**

1. <!-- 用于测试消息回复的 -->
2. **<bean** id="responseQueue" class="org.apache.activemq.command.ActiveMQQueue"**>**
3. **<constructor-arg>**
4. **<value>**responseQueue**</value>**
5. **</constructor-arg>**
6. **</bean>**
8. <!-- responseQueue对应的监听器 -->
9. **<bean** id="responseQueueListener" class="com.tiantian.springintejms.listener.ResponseQueueListener"**/>**
11. <!-- responseQueue对应的监听容器 -->
12. **<bean** id="responseQueueMessageListenerContainer" class="org.springframework.jms.listener.DefaultMessageListenerContainer"**>**
13. **<property** name="connectionFactory" ref="connectionFactory"**/>**
14. **<property** name="destination" ref="responseQueue"**/>**
15. **<property** name="messageListener" ref="responseQueueListener"**/>**
16. **</bean>**

       ResponseQueueListener的定义如下所示：

**Java代码  [收藏代码](javascript:void())**

1. **public** **class** ResponseQueueListener **implements** MessageListener {
3. **public** **void** onMessage(Message message) {
4. **if** (message **instanceof** TextMessage) {
5. TextMessage textMessage = (TextMessage) message;
6. **try** {
7. System.out.println("接收到发送到responseQueue的一个文本消息，内容是：" + textMessage.getText());
8. } **catch** (JMSException e) {
9. e.printStackTrace();
10. }
11. }
12. }
14. }

        接着把我们接收消息的ConsumerListener的receiveMessage方法改为如下：

**Java代码  [收藏代码](javascript:void())**

1. /\*\*
2. \* 当返回类型是非null时MessageListenerAdapter会自动把返回值封装成一个Message，然后进行回复
3. \* @param message
4. \* @return
5. \*/
6. **public** String receiveMessage(String message) {
7. System.out.println("ConsumerListener通过receiveMessage接收到一个纯文本消息，消息内容是：" + message);
8. **return** "这是ConsumerListener对象的receiveMessage方法的返回值。";
9. }

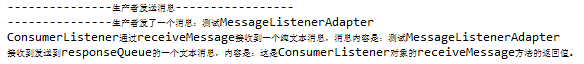
        我们可以看到在上述负责接收消息的receiveMessage方法有一个非空的返回值。

       接着我们运行我们的测试代码，利用生产者往我们定义好的MessageListenerAdapter负责处理的adapterQueue目的地发送一个消息。测试代码如下所示：

**Java代码  [收藏代码](javascript:void())**

1. @RunWith(SpringJUnit4ClassRunner.**class**)
2. @ContextConfiguration("/applicationContext.xml")
3. **public** **class** ProducerConsumerTest {
5. @Autowired
6. **private** ProducerService producerService;
8. @Qualifier("adapterQueue")
9. @Autowired
10. **private** Destination adapterQueue;
12. @Test
13. **public** **void** testMessageListenerAdapter() {
14. producerService.sendMessage(adapterQueue, "测试MessageListenerAdapter");
15. }
17. }

       运行上述测试代码之后，控制台输出如下：

  
        这说明我们的生产者发送消息被MessageListenerAdapter处理之后，MessageListenerAdapter确实把监听器的返回内容封装成一个Message往原Message通过setJMSReplyTo方法指定的回复目的地发送了一个消息。对于MessageListenerAdapter对应的监听器处理方法返回的是一个null值或者返回类型是void的情况，MessageListenerAdapter是不会自动进行消息的回复的，有兴趣的网友可以自己测试一下。

       第二，通过MessageListenerAdapter的defaultResponseDestination属性来指定。这里我们也来做一个测试，首先维持生产者发送消息的代码不变，即发送消息前不通过Message的setJMSReplyTo方法指定消息的回复目的地；接着我们在定义MessageListenerAdapter的时候通过其defaultResponseDestination属性指定其默认的回复目的地是“defaultResponseQueue”，并定义defaultResponseQueue对应的消息监听器和消息监听容器。

**Xml代码  [收藏代码](javascript:void())**

1. <!-- 消息监听适配器 -->
2. **<bean** id="messageListenerAdapter" class="org.springframework.jms.listener.adapter.MessageListenerAdapter"**>**
3. <!-- **<constructor-arg>**
4. **<bean** class="com.tiantian.springintejms.listener.ConsumerListener"**/>**
5. **</constructor-arg>** --**>**
6. **<property** name="delegate"**>**
7. **<bean** class="com.tiantian.springintejms.listener.ConsumerListener"**/>**
8. **</property>**
9. **<property** name="defaultListenerMethod" value="receiveMessage"**/>**
10. **<property** name="defaultResponseDestination" ref="defaultResponseQueue"**/>**
11. **</bean>**
13. <!-- 消息监听适配器对应的监听容器 -->
14. **<bean** id="messageListenerAdapterContainer" class="org.springframework.jms.listener.DefaultMessageListenerContainer"**>**
15. **<property** name="connectionFactory" ref="connectionFactory"**/>**
16. **<property** name="destination" ref="adapterQueue"**/>**
17. **<property** name="messageListener" ref="messageListenerAdapter"**/>**<!-- 使用MessageListenerAdapter来作为消息监听器 -->
18. **</bean>**
20. !-- 默认的消息回复队列 --**>**
21. **<bean** id="defaultResponseQueue" class="org.apache.activemq.command.ActiveMQQueue"**>**
22. **<constructor-arg>**
23. **<value>**defaultResponseQueue**</value>**
24. **</constructor-arg>**
25. **</bean>**
27. <!-- defaultResponseQueue对应的监听器 -->
28. **<bean** id="defaultResponseQueueListener" class="com.tiantian.springintejms.listener.DefaultResponseQueueListener"**/>**
30. <!-- defaultResponseQueue对应的监听容器 -->
31. **<bean** id="defaultResponseQueueMessageListenerContainer" class="org.springframework.jms.listener.DefaultMessageListenerContainer"**>**
32. **<property** name="connectionFactory" ref="connectionFactory"**/>**
33. **<property** name="destination" ref="defaultResponseQueue"**/>**
34. **<property** name="messageListener" ref="defaultResponseQueueListener"**/>**
35. **</bean>**

       DefaultResponseQueueListener的代码如下所示：

**Java代码  [收藏代码](javascript:void())**

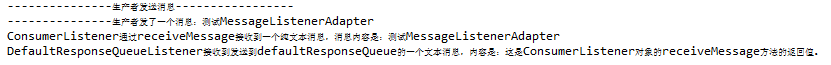
1. **package** com.tiantian.springintejms.listener;
3. **import** javax.jms.JMSException;
4. **import** javax.jms.Message;
5. **import** javax.jms.MessageListener;
6. **import** javax.jms.TextMessage;
8. **public** **class** DefaultResponseQueueListener **implements** MessageListener {
10. **public** **void** onMessage(Message message) {
11. **if** (message **instanceof** TextMessage) {
12. TextMessage textMessage = (TextMessage) message;
13. **try** {
14. System.out.println("DefaultResponseQueueListener接收到发送到defaultResponseQueue的一个文本消息，内容是：" + textMessage.getText());
15. } **catch** (JMSException e) {
16. e.printStackTrace();
17. }
18. }
19. }
21. }

       这时候运行如下测试代码：

**Java代码  [收藏代码](javascript:void())**

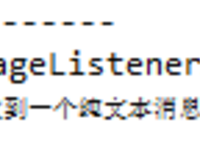
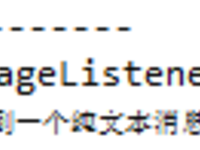
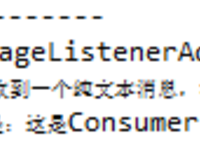
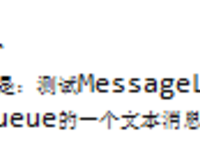
1. @Test
2. **public** **void** testMessageListenerAdapter() {
3. producerService.sendMessage(adapterQueue, "测试MessageListenerAdapter");
4. }

       控制台将输出如下内容：

  
        这说明MessageListenerAdapter会自动把真正的消息处理器返回的非空内容封装成一个Message发送回复消息到通过defaultResponseDestination属性指定的默认消息回复目的地。

       既然我们可以通过两种方式来指定MessageListenerAdapter自动发送回复消息的目的地，那么当我们两种方式都指定了而且它们的目的地还不一样的时候会怎么发送呢？是两个都发还是只发其中的一个呢？关于这部分的测试我这里就不赘述了，有兴趣的网友可以自己进行。这里我可以直接的告诉大家，当两种方式都指定了消息的回复目的地的时候使用发送消息的setJMSReplyTo方法指定的目的地将具有较高的优先级，MessageListenerAdapter将只往该方法指定的消息回复目的地发送回复消息。

**附：**

* [Spring整合JMS——基于ActiveMQ实现（一）](http://elim.iteye.com/blog/1893038)
* [Spring整合JMS（三）——MessageConverter介绍](http://elim.iteye.com/blog/1900937)
* [Spring整合JMS（四）——事务管理](http://elim.iteye.com/blog/1983532)
* [](http://dl2.iteye.com/upload/attachment/0086/1538/bea547c4-869a-30f0-bef2-aacdcf62d3f3.png)
* 大小: 4.9 KB
* [](http://dl2.iteye.com/upload/attachment/0086/1540/ed428c5f-4b65-31fa-bc6c-6b5940df5b99.png)
* 大小: 4.5 KB
* [](http://dl2.iteye.com/upload/attachment/0086/1542/d0270ffa-b3e3-3405-8181-83e9cd80574b.png)
* 大小: 4.4 KB
* [](http://dl2.iteye.com/upload/attachment/0086/1544/0f9140c6-5670-3aad-9cda-8d98b890bafa.png)
* 大小: 7.1 KB
* [](http://dl2.iteye.com/upload/attachment/0086/1546/d5f016aa-a09b-302a-8eb9-c688027662dd.png)
* 大小: 7.8 KB