**[Spring整合JMS（三）——MessageConverter介绍](http://elim.iteye.com/blog/1900937)**

**博客分类：**

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

[Spring](http://www.iteye.com/blogs/tag/Spring)[jms](http://www.iteye.com/blogs/tag/jms)[integrate](http://www.iteye.com/blogs/tag/integrate)[messageConverter](http://www.iteye.com/blogs/tag/messageConverter)

**1.4     消息转换器MessageConverter**

MessageConverter的作用主要有两方面，一方面它可以把我们的非标准化Message对象转换成我们的目标Message对象，这主要是用在发送消息的时候；另一方面它又可以把我们的Message对象转换成对应的目标对象，这主要是用在接收消息的时候。

下面我们就拿发送一个对象消息来举例，假设我们有这样一个需求：我们平台有一个发送邮件的功能，进行发送的时候我们只是把我们的相关信息封装成一个JMS消息，然后利用JMS进行发送，在对应的消息监听器进行接收到的消息处理时才真正的进行消息发送。

假设我们有这么一个Email对象：

**Java代码  [收藏代码](javascript:void())**

1. **public** **class** Email **implements** Serializable {
3. **private** **static** **final** **long** serialVersionUID = -658250125732806493L;
5. **private** String receiver;
6. **private** String title;
7. **private** String content;
9. **public** Email(String receiver, String title, String content) {
10. **this**.receiver = receiver;
11. **this**.title = title;
12. **this**.content = content;
13. }
15. **public** String getReceiver() {
16. **return** receiver;
17. }
19. **public** **void** setReceiver(String receiver) {
20. **this**.receiver = receiver;
21. }
23. **public** String getTitle() {
24. **return** title;
25. }
27. **public** **void** setTitle(String title) {
28. **this**.title = title;
29. }
31. **public** String getContent() {
32. **return** content;
33. }
35. **public** **void** setContent(String content) {
36. **this**.content = content;
37. }
39. @Override
40. **public** String toString() {
41. StringBuilder builder = **new** StringBuilder();
42. builder.append("Email [receiver=").append(receiver).append(", title=")
43. .append(title).append(", content=").append(content).append("]");
44. **return** builder.toString();
45. }
47. }

       这个Email对象包含了一个简单的接收者email地址、邮件主题和邮件内容。我们在发送的时候就把这个对象封装成一个ObjectMessage进行发送。代码如下所示：

**Java代码  [收藏代码](javascript:void())**

1. **public** **class** ProducerServiceImpl **implements** ProducerService {
3. @Autowired
4. **private** JmsTemplate jmsTemplate;
6. **public** **void** sendMessage(Destination destination, **final** Serializable obj) {
7. jmsTemplate.send(destination, **new** MessageCreator() {
9. **public** Message createMessage(Session session) **throws** JMSException {
10. ObjectMessage objMessage = session.createObjectMessage(obj);
11. **return** objMessage;
12. }
14. });
15. }
17. }

       这是对应的在没有使用MessageConverter的时候我们需要new一个MessageCreator接口对象，然后在其抽象方法createMessage内部使用session创建一个对应的消息。在使用了MessageConverter的时候我们在使用JmsTemplate进行消息发送时只需要调用其对应的convertAndSend方法即可。如：

**Java代码  [收藏代码](javascript:void())**

1. **public** **void** sendMessage(Destination destination, **final** Serializable obj) {
2. //未使用MessageConverter的情况
3. /\*jmsTemplate.send(destination, new MessageCreator() {
5. public Message createMessage(Session session) throws JMSException {
6. ObjectMessage objMessage = session.createObjectMessage(obj);
7. return objMessage;
8. }
10. });\*/
11. //使用MessageConverter的情况
12. jmsTemplate.convertAndSend(destination, obj);
13. }

这样JmsTemplate就会在其内部调用预定的MessageConverter对我们的消息对象进行转换，然后再进行发送。

       这个时候我们就需要定义我们的MessageConverter了。要定义自己的MessageConverter很简单，只需要实现Spring为我们提供的MessageConverter接口即可。我们先来看一下MessageConverter接口的定义：

**Java代码  [收藏代码](javascript:void())**

1. **public** **interface** MessageConverter {
3. Message toMessage(Object object, Session session) **throws** JMSException, MessageConversionException;
5. Object fromMessage(Message message) **throws** JMSException, MessageConversionException;
7. }

       我们可以看到其中一共定义了两个方法fromMessage和toMessage，fromMessage是用来把一个JMS Message转换成对应的Java对象，而toMessage方法是用来把一个Java对象转换成对应的JMS Message。因为我们已经知道上面要发送的对象就是一个Email对象，所以在这里我们就简单地定义一个EmailMessageConverter用来把Email对象和对应的ObjectMessage进行转换，其代码如下：

**Java代码  [收藏代码](javascript:void())**

1. **import** javax.jms.JMSException;
2. **import** javax.jms.Message;
3. **import** javax.jms.ObjectMessage;
4. **import** javax.jms.Session;
6. **import** org.springframework.jms.support.converter.MessageConversionException;
7. **import** org.springframework.jms.support.converter.MessageConverter;
9. **public** **class** EmailMessageConverter **implements** MessageConverter {
11. **public** Message toMessage(Object object, Session session)
12. **throws** JMSException, MessageConversionException {
13. **return** session.createObjectMessage((Serializable) object);
14. }
16. **public** Object fromMessage(Message message) **throws** JMSException,
17. MessageConversionException {
18. ObjectMessage objMessage = (ObjectMessage) message;
19. **return** objMessage.getObject();
20. }
22. }

       这样当我们利用JmsTemplate的convertAndSend方法发送一个Email对象的时候就会把对应的Email对象当做参数调用我们定义好的EmailMessageConverter的toMessage方法。

       定义好我们的EmailMessageConverter之后就需要指定我们用来发送Email对象的JmsTemplate对象的messageConverter为EmailMessageConverter，这里我们在Spring的配置文件中定义JmsTemplate bean的时候就指定：

**Xml代码  [收藏代码](javascript:void())**

1. <!-- Spring提供的JMS工具类，它可以进行消息发送、接收等 -->
2. **<bean** id="jmsTemplate" class="org.springframework.jms.core.JmsTemplate"**>**
3. <!-- 这个connectionFactory对应的是我们定义的Spring提供的那个ConnectionFactory对象 -->
4. **<property** name="connectionFactory" ref="connectionFactory"**/>**
5. <!-- 消息转换器 -->
6. **<property** name="messageConverter" ref="emailMessageConverter"**/>**
7. **</bean>**
8. <!-- 类型转换器 -->
9. **<bean** id="emailMessageConverter" class="com.tiantian.springintejms.converter.EmailMessageConverter"**/>**

       到此我们的MessageConverter就定义好了，也能够进行使用了，接着我们来进行测试一下，定义测试代码如下所示：

**Java代码  [收藏代码](javascript:void())**

1. @Test
2. **public** **void** testObjectMessage() {
3. Email email = **new** Email("zhangsan@xxx.com", "主题", "内容");
4. producerService.sendMessage(destination, email);
5. }

       上面destination对应的接收处理的MessageListener方法如下所示：

**Java代码  [收藏代码](javascript:void())**

1. **public** **class** ConsumerMessageListener **implements** MessageListener {
3. **public** **void** onMessage(Message message) {
5. **if** (message **instanceof** ObjectMessage) {
6. ObjectMessage objMessage = (ObjectMessage) message;
7. **try** {
8. Object obj = objMessage.getObject();
9. Email email = (Email) obj;
10. System.out.println("接收到一个ObjectMessage，包含Email对象。");
11. System.out.println(email);
12. } **catch** (JMSException e) {
13. e.printStackTrace();
14. }
15. }
16. }
18. }

       之前说了MessageConverter有两方面的功能，除了把Java对象转换成对应的Jms Message之外还可以把Jms Message转换成对应的Java对象。我们看上面的消息监听器在接收消息的时候接收到的就是一个Jms Message，如果我们要利用MessageConverter来把它转换成对应的Java对象的话，只能是我们往里面注入一个对应的MessageConverter，然后在里面手动的调用，如：

**Java代码  [收藏代码](javascript:void())**

1. **public** **class** ConsumerMessageListener **implements** MessageListener {
3. **private** MessageConverter messageConverter;
5. **public** **void** onMessage(Message message) {
7. **if** (message **instanceof** ObjectMessage) {
8. ObjectMessage objMessage = (ObjectMessage) message;
9. **try** {
10. /\*Object obj = objMessage.getObject();
11. Email email = (Email) obj;\*/
12. Email email = (Email) messageConverter.fromMessage(objMessage);
13. System.out.println("接收到一个ObjectMessage，包含Email对象。");
14. System.out.println(email);
15. } **catch** (JMSException e) {
16. e.printStackTrace();
17. }
19. }
20. }
22. **public** MessageConverter getMessageConverter() {
23. **return** messageConverter;
24. }
26. **public** **void** setMessageConverter(MessageConverter messageConverter) {
27. **this**.messageConverter = messageConverter;
28. }
30. }

       当我们使用MessageListenerAdapter来作为消息监听器的时候，我们可以为它指定一个对应的MessageConverter，这样Spring在处理接收到的消息的时候就会自动地利用我们指定的MessageConverter对它进行转换，然后把转换后的Java对象作为参数调用指定的消息处理方法。这里我们再把前面讲解MessageListenerAdapter时定义的MessageListenerAdapter拿来做一个测试，我们指定它的MessageConverter为我们定义好的EmailMessageConverter。

**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. **<property** name="messageConverter" ref="emailMessageConverter"**/>**
8. **</bean>**
9. <!-- 消息监听适配器对应的监听容器 -->
10. **<bean** id="messageListenerAdapterContainer" class="org.springframework.jms.listener.DefaultMessageListenerContainer"**>**
11. **<property** name="connectionFactory" ref="connectionFactory"**/>**
12. **<property** name="destination" ref="adapterQueue"**/>**
13. **<property** name="messageListener" ref="messageListenerAdapter"**/>**<!-- 使用MessageListenerAdapter来作为消息监听器 -->
14. **</bean>**

       然后在我们的真正用于处理接收到的消息的ConsumerListener中添加一个receiveMessage方法，添加后其代码如下所示：

**Java代码  [收藏代码](javascript:void())**

1. **public** **class** ConsumerListener {
3. **public** **void** receiveMessage(String message) {
4. System.out.println("ConsumerListener通过receiveMessage接收到一个纯文本消息，消息内容是：" + message);
5. }
7. **public** **void** receiveMessage(Email email) {
8. System.out.println("接收到一个包含Email的ObjectMessage。");
9. System.out.println(email);
10. }
12. }

       然后我们定义如下测试代码：

**Java代码  [收藏代码](javascript:void())**

1. @Test
2. **public** **void** testObjectMessage() {
3. Email email = **new** Email("zhangsan@xxx.com", "主题", "内容");
4. producerService.sendMessage(adapterQueue, email);
5. }

       因为我们给MessageListenerAdapter指定了一个MessageConverter，而且是一个EmailMessageConverter，所以当MessageListenerAdapter接收到一个消息后，它会调用我们指定的MessageConverter的fromMessage方法把它转换成一个Java对象，根据定义这里会转换成一个Email对象，然后会把这个Email对象作为参数调用我们通过defaultListenerMethod属性指定的默认处理器方法，根据定义这里就是receiveMessage方法，但是我们可以看到在ConsumerListener中我们一共定义了两个receiveMessage方法，因为是通过转换后的Email对象作为参数进行方法调用的，所以这里调用的就应该是参数类型为Email的receiveMessage方法了。上述测试代码运行后会输出如下结果：  
http://dl2.iteye.com/upload/attachment/0086/6469/ee3a2dc6-82ce-3d4c-a7a5-30f5adca1499.png  
        说到这里可能有读者就会有疑问了，说我们在之前讲解MessageListenerAdapter的时候不是没有指定对应的MessageConverter，然后发送了一个TextMessage，结果Spring还是把它转换成一个String对象，调用了ConsumerListener参数类型为String的receiveMessage方法吗？那你这个MessageConverter在MessageListenerAdapter进行消息接收的时候也没什么用啊。

       其实还是有用的，在我们使用MessageListenerAdapter时，在对其进行初始化也就是调用其构造方法时，它会默认new一个Spring已经为我们实现了的MessageConverter——SimpleMessageConverter作为其默认的MessageConverter，这也就是为什么我们在使用MessageListenerAdapter的时候不需要指定MessageConverter但是消息还是会转换成对应的Java对象的原因。所以默认情况下我们使用MessageListenerAdapter时其对应的MessageListener的处理器方法参数类型必须是一个普通Java对象，而不能是对应的Jms Message对象。

       那如果我们在处理Jms Message的时候想使用MessageListenerAdapter，然后又希望处理最原始的Message，而不是经过MessageConverter进行转换后的Message该怎么办呢？这个时候我们只需要在定义MessageListenerAdapter的时候指定其MessageConverter为空就可以了。

**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. **<property** name="messageConverter"**>**
8. **<null/>**
9. **</property>**
10. **</bean>**

       那么这个时候我们的真实MessageListener的处理器方法参数类型就应该是Jms Message或对应的Jms Message子类型了，不然就会调用不到对应的处理方法了。这里因为我们发送的是一个ObjectMessage，所以这里就添加一个对应的参数类型为ObjectMessage的receiveMessage方法了。

**Java代码  [收藏代码](javascript:void())**

1. **public** **void** receiveMessage(ObjectMessage message) **throws** JMSException {
2. System.out.println(message.getObject());
3. }

       刚刚讲到Spring已经为我们实现了一个简单的MessageConverter，即org.springframework.jms.support.converter.SimpleMessageConverter，其实Spring在初始化JmsTemplate的时候也指定了其对应的MessageConverter为一个SimpleMessageConverter，所以如果我们平常没有什么特殊要求的时候可以直接使用JmsTemplate的convertAndSend系列方法进行消息发送，而不必繁琐的在调用send方法时自己new一个MessageCreator进行相应Message的创建。

这里我们也来看一下SimpleMessageConverter的定义，如果觉得它不能满足你的要求，那我们可以对它里面的部分方法进行重写，或者是完全实现自己的MessageConverter。

**Java代码  [收藏代码](javascript:void())**

1. **public** **class** SimpleMessageConverter **implements** MessageConverter {
3. **public** Message toMessage(Object object, Session session) **throws** JMSException, MessageConversionException {
4. **if** (object **instanceof** Message) {
5. **return** (Message) object;
6. }
7. **else** **if** (object **instanceof** String) {
8. **return** createMessageForString((String) object, session);
9. }
10. **else** **if** (object **instanceof** **byte**[]) {
11. **return** createMessageForByteArray((**byte**[]) object, session);
12. }
13. **else** **if** (object **instanceof** Map) {
14. **return** createMessageForMap((Map) object, session);
15. }
16. **else** **if** (object **instanceof** Serializable) {
17. **return** createMessageForSerializable(((Serializable) object), session);
18. }
20. **else** {
21. **throw** **new** MessageConversionException("Cannot convert object of type [" +
22. ObjectUtils.nullSafeClassName(object) + "] to JMS message. Supported message " +
23. "payloads are: String, byte array, Map<String,?>, Serializable object.");
24. }
25. }
27. **public** Object fromMessage(Message message) **throws** JMSException, MessageConversionException {
28. **if** (message **instanceof** TextMessage) {
29. **return** extractStringFromMessage((TextMessage) message);
30. }
31. **else** **if** (message **instanceof** BytesMessage) {
32. **return** extractByteArrayFromMessage((BytesMessage) message);
33. }
34. **else** **if** (message **instanceof** MapMessage) {
35. **return** extractMapFromMessage((MapMessage) message);
36. }
37. **else** **if** (message **instanceof** ObjectMessage) {
38. **return** extractSerializableFromMessage((ObjectMessage) message);
39. }
40. **else** {
41. **return** message;
42. }
43. }
45. **protected** TextMessage createMessageForString(String text, Session session) **throws** JMSException {
46. **return** session.createTextMessage(text);
47. }
49. **protected** BytesMessage createMessageForByteArray(**byte**[] bytes, Session session) **throws** JMSException {
50. BytesMessage message = session.createBytesMessage();
51. message.writeBytes(bytes);
52. **return** message;
53. }
55. **protected** MapMessage createMessageForMap(Map<?, ?> map, Session session) **throws** JMSException {
56. MapMessage message = session.createMapMessage();
57. **for** (Map.Entry entry : map.entrySet()) {
58. **if** (!(entry.getKey() **instanceof** String)) {
59. **throw** **new** MessageConversionException("Cannot convert non-String key of type [" +
60. ObjectUtils.nullSafeClassName(entry.getKey()) + "] to JMS MapMessage entry");
61. }
62. message.setObject((String) entry.getKey(), entry.getValue());
63. }
64. **return** message;
65. }
67. **protected** ObjectMessage createMessageForSerializable(Serializable object, Session session) **throws** JMSException {
68. **return** session.createObjectMessage(object);
69. }

72. **protected** String extractStringFromMessage(TextMessage message) **throws** JMSException {
73. **return** message.getText();
74. }
76. **protected** **byte**[] extractByteArrayFromMessage(BytesMessage message) **throws** JMSException {
77. **byte**[] bytes = **new** **byte**[(**int**) message.getBodyLength()];
78. message.readBytes(bytes);
79. **return** bytes;
80. }
82. **protected** Map extractMapFromMessage(MapMessage message) **throws** JMSException {
83. Map<String, Object> map = **new** HashMap<String, Object>();
84. Enumeration en = message.getMapNames();
85. **while** (en.hasMoreElements()) {
86. String key = (String) en.nextElement();
87. map.put(key, message.getObject(key));
88. }
89. **return** map;
90. }
92. **protected** Serializable extractSerializableFromMessage(ObjectMessage message) **throws** JMSException {
93. **return** message.getObject();
94. }
96. }

**附：**

* [Spring整合JMS——基于ActiveMQ实现（一）](http://elim.iteye.com/blog/1893038)
* [Spring整合JMS（二）——三种消息监听器](http://elim.iteye.com/blog/1893676)
* [Spring整合JMS（四）——事务管理](http://elim.iteye.com/blog/1983532)