[**Spring整合JMS(一)——基于ActiveMQ实现**](http://elim.iteye.com/blog/1893038)

**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)[整合](http://www.iteye.com/blogs/tag/%E6%95%B4%E5%90%88)[integrate](http://www.iteye.com/blogs/tag/integrate)[ActiveMQ](http://www.iteye.com/blogs/tag/ActiveMQ)

**1.1     JMS简介**

       JMS的全称是Java Message Service，即Java消息服务。它主要用于在生产者和消费者之间进行消息传递，生产者负责产生消息，而消费者负责接收消息。把它应用到实际的业务需求中的话我们可以在特定的时候利用生产者生成一消息，并进行发送，对应的消费者在接收到对应的消息后去完成对应的业务逻辑。对于消息的传递有两种类型，一种是点对点的，即一个生产者和一个消费者一一对应；另一种是发布/订阅模式，即一个生产者产生消息并进行发送后，可以由多个消费者进行接收。

**1.2     Spring整合JMS**

       对JMS做了一个简要介绍之后，接下来就讲一下Spring整合JMS的具体过程。JMS只是一个标准，真正在使用它的时候我们需要有它的具体实现，这里我们就使用Apache的activeMQ来作为它的实现。所使用的依赖利用Maven来进行管理，具体依赖如下：

**Xml代码  [收藏代码](javascript:void())**

1. **<dependencies>**
2. **<dependency>**
3. **<groupId>**junit**</groupId>**
4. **<artifactId>**junit**</artifactId>**
5. **<version>**4.10**</version>**
6. **<scope>**test**</scope>**
7. **</dependency>**
8. **<dependency>**
9. **<groupId>**org.springframework**</groupId>**
10. **<artifactId>**spring-context**</artifactId>**
11. **<version>**${spring-version}**</version>**
12. **</dependency>**
13. **<dependency>**
14. **<groupId>**org.springframework**</groupId>**
15. **<artifactId>**spring-jms**</artifactId>**
16. **<version>**${spring-version}**</version>**
17. **</dependency>**
18. **<dependency>**
19. **<groupId>**org.springframework**</groupId>**
20. **<artifactId>**spring-test**</artifactId>**
21. **<version>**${spring-version}**</version>**
22. **</dependency>**
23. **<dependency>**
24. **<groupId>**javax.annotation**</groupId>**
25. **<artifactId>**jsr250-api**</artifactId>**
26. **<version>**1.0**</version>**
27. **</dependency>**
28. **<dependency>**
29. **<groupId>**org.apache.activemq**</groupId>**
30. **<artifactId>**activemq-core**</artifactId>**
31. **<version>**5.7.0**</version>**
32. **</dependency>**
33. **</dependencies>**

**1.2.1**  **activeMQ准备**

       既然是使用的apache的activeMQ作为JMS的实现，那么首先我们应该到apache官网上下载activeMQ（<http://activemq.apache.org/download.html>），进行解压后运行其bin目录下面的activemq.bat文件启动activeMQ。

**1.2.2配置ConnectionFactory**

       ConnectionFactory是用于产生到JMS服务器的链接的，Spring为我们提供了多个ConnectionFactory，有SingleConnectionFactory和CachingConnectionFactory。SingleConnectionFactory对于建立JMS服务器链接的请求会一直返回同一个链接，并且会忽略Connection的close方法调用。CachingConnectionFactory继承了SingleConnectionFactory，所以它拥有SingleConnectionFactory的所有功能，同时它还新增了缓存功能，它可以缓存Session、MessageProducer和MessageConsumer。这里我们使用SingleConnectionFactory来作为示例。

**Xml代码  [收藏代码](javascript:void())**

1. **<bean** id="connectionFactory" class="org.springframework.jms.connection.SingleConnectionFactory"**/>**

       这样就定义好产生JMS服务器链接的ConnectionFactory了吗？答案是非也。Spring提供的ConnectionFactory只是Spring用于管理ConnectionFactory的，真正产生到JMS服务器链接的ConnectionFactory还得是由JMS服务厂商提供，并且需要把它注入到Spring提供的ConnectionFactory中。我们这里使用的是ActiveMQ实现的JMS，所以在我们这里真正的可以产生Connection的就应该是由ActiveMQ提供的ConnectionFactory。所以定义一个ConnectionFactory的完整代码应该如下所示：

**Xml代码  [收藏代码](javascript:void())**

1. <!-- 真正可以产生Connection的ConnectionFactory，由对应的 JMS服务厂商提供-->
2. **<bean** id="targetConnectionFactory" class="org.apache.activemq.ActiveMQConnectionFactory"**>**
3. **<property** name="brokerURL" value="tcp://localhost:61616"**/>**
4. **</bean>**
6. <!-- Spring用于管理真正的ConnectionFactory的ConnectionFactory -->
7. **<bean** id="connectionFactory" class="org.springframework.jms.connection.SingleConnectionFactory"**>**
8. <!-- 目标ConnectionFactory对应真实的可以产生JMS Connection的ConnectionFactory -->
9. **<property** name="targetConnectionFactory" ref="targetConnectionFactory"**/>**
10. **</bean>**

       ActiveMQ为我们提供了一个PooledConnectionFactory，通过往里面注入一个ActiveMQConnectionFactory可以用来将Connection、Session和MessageProducer池化，这样可以大大的减少我们的资源消耗。当使用PooledConnectionFactory时，我们在定义一个ConnectionFactory时应该是如下定义：

**Xml代码  [收藏代码](javascript:void())**

1. <!-- 真正可以产生Connection的ConnectionFactory，由对应的 JMS服务厂商提供-->
2. **<bean** id="targetConnectionFactory" class="org.apache.activemq.ActiveMQConnectionFactory"**>**
3. **<property** name="brokerURL" value="tcp://localhost:61616"**/>**
4. **</bean>**
6. **<bean** id="pooledConnectionFactory" class="org.apache.activemq.pool.PooledConnectionFactory"**>**
7. **<property** name="connectionFactory" ref="targetConnectionFactory"**/>**
8. **<property** name="maxConnections" value="10"**/>**
9. **</bean>**
11. **<bean** id="connectionFactory" class="org.springframework.jms.connection.SingleConnectionFactory"**>**
12. **<property** name="targetConnectionFactory" ref="pooledConnectionFactory"**/>**
13. **</bean>**

**1.2.3配置生产者**

配置好ConnectionFactory之后我们就需要配置生产者。生产者负责产生消息并发送到JMS服务器，这通常对应的是我们的一个业务逻辑服务实现类。但是我们的服务实现类是怎么进行消息的发送的呢？这通常是利用Spring为我们提供的JmsTemplate类来实现的，所以配置生产者其实最核心的就是配置进行消息发送的JmsTemplate。对于消息发送者而言，它在发送消息的时候要知道自己该往哪里发，为此，我们在定义JmsTemplate的时候需要往里面注入一个Spring提供的ConnectionFactory对象。

**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. **</bean>**

       在真正利用JmsTemplate进行消息发送的时候，我们需要知道消息发送的目的地，即destination。在Jms中有一个用来表示目的地的Destination接口，它里面没有任何方法定义，只是用来做一个标识而已。当我们在使用JmsTemplate进行消息发送时没有指定destination的时候将使用默认的Destination。默认Destination可以通过在定义jmsTemplate bean对象时通过属性defaultDestination或defaultDestinationName来进行注入，defaultDestinationName对应的就是一个普通字符串。在ActiveMQ中实现了两种类型的Destination，一个是点对点的ActiveMQQueue，另一个就是支持订阅/发布模式的ActiveMQTopic。在定义这两种类型的Destination时我们都可以通过一个name属性来进行构造，如：

**Xml代码  [收藏代码](javascript:void())**

1. <!--这个是队列目的地，点对点的-->
2. **<bean** id="queueDestination" class="org.apache.activemq.command.ActiveMQQueue"**>**
3. **<constructor-arg>**
4. **<value>**queue**</value>**
5. **</constructor-arg>**
6. **</bean>**
7. <!--这个是主题目的地，一对多的-->
8. **<bean** id="topicDestination" class="org.apache.activemq.command.ActiveMQTopic"**>**
9. **<constructor-arg** value="topic"**/>**
10. **</bean>**

       假设我们定义了一个ProducerService，里面有一个向Destination发送纯文本消息的方法sendMessage，那么我们的代码就大概是这个样子：

**Java代码  [收藏代码](javascript:void())**

1. **package** com.tiantian.springintejms.service.impl;
3. **import** javax.annotation.Resource;
4. **import** javax.jms.Destination;
5. **import** javax.jms.JMSException;
6. **import** javax.jms.Message;
7. **import** javax.jms.Session;
9. **import** org.springframework.jms.core.JmsTemplate;
10. **import** org.springframework.jms.core.MessageCreator;
11. **import** org.springframework.stereotype.Component;
13. **import** com.tiantian.springintejms.service.ProducerService;
15. @Component
16. **public** **class** ProducerServiceImpl **implements** ProducerService {
18. **private** JmsTemplate jmsTemplate;
20. **public** **void** sendMessage(Destination destination, **final** String message) {
21. System.out.println("---------------生产者发送消息-----------------");
22. System.out.println("---------------生产者发了一个消息：" + message);
23. jmsTemplate.send(destination, **new** MessageCreator() {
24. **public** Message createMessage(Session session) **throws** JMSException {
25. **return** session.createTextMessage(message);
26. }
27. });
28. }
30. **public** JmsTemplate getJmsTemplate() {
31. returnjmsTemplate;
32. }
34. @Resource
35. **public** **void** setJmsTemplate(JmsTemplate jmsTemplate) {
36. **this**.jmsTemplate = jmsTemplate;
37. }
39. }

       我们可以看到在sendMessage方法体里面我们是通过jmsTemplate来发送消息到对应的Destination的。到此，我们生成一个简单的文本消息并把它发送到指定目的地Destination的生产者就配置好了。

**1.2.4配置消费者**

生产者往指定目的地Destination发送消息后，接下来就是消费者对指定目的地的消息进行消费了。那么消费者是如何知道有生产者发送消息到指定目的地Destination了呢？这是通过Spring为我们封装的消息监听容器MessageListenerContainer实现的，它负责接收信息，并把接收到的信息分发给真正的MessageListener进行处理。每个消费者对应每个目的地都需要有对应的MessageListenerContainer。对于消息监听容器而言，除了要知道监听哪个目的地之外，还需要知道到哪里去监听，也就是说它还需要知道去监听哪个JMS服务器，这是通过在配置MessageConnectionFactory的时候往里面注入一个ConnectionFactory来实现的。所以我们在配置一个MessageListenerContainer的时候有三个属性必须指定，一个是表示从哪里监听的ConnectionFactory；一个是表示监听什么的Destination；一个是接收到消息以后进行消息处理的MessageListener。Spring一共为我们提供了两种类型的MessageListenerContainer，SimpleMessageListenerContainer和DefaultMessageListenerContainer。

SimpleMessageListenerContainer会在一开始的时候就创建一个会话session和消费者Consumer，并且会使用标准的JMS MessageConsumer.setMessageListener()方法注册监听器让JMS提供者调用监听器的回调函数。它不会动态的适应运行时需要和参与外部的事务管理。兼容性方面，它非常接近于独立的JMS规范，但一般不兼容Java EE的JMS限制。

大多数情况下我们还是使用的DefaultMessageListenerContainer，跟SimpleMessageListenerContainer相比，DefaultMessageListenerContainer会动态的适应运行时需要，并且能够参与外部的事务管理。它很好的平衡了对JMS提供者要求低、先进功能如事务参与和兼容Java EE环境。

**定义处理消息的MessageListener**

       要定义处理消息的MessageListener我们只需要实现JMS规范中的MessageListener接口就可以了。MessageListener接口中只有一个方法onMessage方法，当接收到消息的时候会自动调用该方法。

**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** ConsumerMessageListener **implements** MessageListener {
10. **public** **void** onMessage(Message message) {
11. //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
12. TextMessage textMsg = (TextMessage) message;
13. System.out.println("接收到一个纯文本消息。");
14. **try** {
15. System.out.println("消息内容是：" + textMsg.getText());
16. } **catch** (JMSException e) {
17. e.printStackTrace();
18. }
19. }
21. }

       有了MessageListener之后我们就可以在Spring的配置文件中配置一个消息监听容器了。

**Xml代码  [收藏代码](javascript:void())**

1. <!--这个是队列目的地-->
2. **<bean** id="queueDestination" class="org.apache.activemq.command.ActiveMQQueue"**>**
3. **<constructor-arg>**
4. **<value>**queue**</value>**
5. **</constructor-arg>**
6. **</bean>**
7. <!-- 消息监听器 -->
8. **<bean** id="consumerMessageListener" class="com.tiantian.springintejms.listener.ConsumerMessageListener"**/>**
10. <!-- 消息监听容器 -->
11. **<bean** id="jmsContainer"        class="org.springframework.jms.listener.DefaultMessageListenerContainer"**>**
12. **<property** name="connectionFactory" ref="connectionFactory" **/>**
13. **<property** name="destination" ref="queueDestination" **/>**
14. **<property** name="messageListener" ref="consumerMessageListener" **/>**
15. **</bean>**

       我们可以看到我们定义了一个名叫queue的ActiveMQQueue目的地，我们的监听器就是监听了发送到这个目的地的消息。

       至此我们的生成者和消费者都配置完成了，这也就意味着我们的整合已经完成了。这个时候完整的Spring的配置文件应该是这样的：

**Xml代码  [收藏代码](javascript:void())**

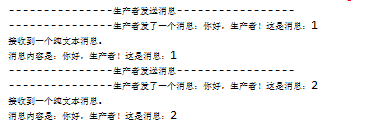
1. **<?xml** version="1.0" encoding="UTF-8"**?>**
2. **<beans** xmlns="http://www.springframework.org/schema/beans"
3. xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:context="http://www.springframework.org/schema/context"
4. xmlns:jms="http://www.springframework.org/schema/jms"
5. xsi:schemaLocation="http://www.springframework.org/schema/beans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

       接着我们来测试一下，看看我们的整合是否真的成功了，测试代码如下：

**Java代码  [收藏代码](javascript:void())**

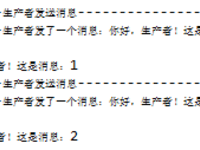
1. **package** com.tiantian.springintejms.test;
3. **import** javax.jms.Destination;
5. **import** org.junit.Test;
6. **import** org.junit.runner.RunWith;
7. **import** org.springframework.beans.factory.annotation.Autowired;
8. **import** org.springframework.beans.factory.annotation.Qualifier;
9. **import** org.springframework.test.context.ContextConfiguration;
10. **import** org.springframework.test.context.junit4.SpringJUnit4ClassRunner;
11. **import** com.tiantian.springintejms.service.ProducerService;
13. @RunWith(SpringJUnit4ClassRunner.**class**)
14. @ContextConfiguration("/applicationContext.xml")
15. **public** **class** ProducerConsumerTest {
17. @Autowired
18. **private** ProducerService producerService;
19. @Autowired
20. @Qualifier("queueDestination")
21. **private** Destination destination;
23. @Test
24. **public** **void** testSend() {
25. **for** (**int** i=0; i<2; i++) {
26. producerService.sendMessage(destination, "你好，生产者！这是消息：" + (i+1));
27. }
28. }
30. }

       在上面的测试代码中我们利用生产者发送了两个消息，正常来说，消费者应该可以接收到这两个消息。运行测试代码后控制台输出如下：



       看，控制台已经进行了正确的输出，这说明我们的整合确实是已经成功了。

**附：**

* [Spring整合JMS（二）——消息监听器](http://elim.iteye.com/blog/1893676)
* [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/0086/1032/bc449d33-84f8-3742-bca5-0251132e810f.png)
* 大小: 5 KB
* [SpringInteJMS\_项目源码\_.zip](http://dl.iteye.com/topics/download/a3f9e775-db87-3cb2-9467-e7045a3be2a7) (70.2 KB)
* 下载次数: 3413