目录

[producer 1](#_Toc13008)

[javax.jms.Destination 2](#_Toc1272)

[org.springframework.jms.core.JmsTemplate 2](#_Toc7676)

[consumer 3](#_Toc13932)

现在已经没那种学习的热情了，只要知道他是什么，如何使用就好。

我一直以为消息队列就是一个jar包，一套代码，但消息队列本身是一个软件，不是jar包，它需要安装。但为了使用这个软件，我们需要在pom里引入某个依赖，类比mysql于mysql-connector-java的关系。

其实光看名字也能看出含义来，存放消息的队列。消息是怎么来的呢，由生产者提供；消息用来干嘛呢，被消费者消费。一个提供，一个拿，并由队列保证顺序。

# producer

生产者就是个普通的java类，他怎么发消息呢，这里用spring举例，spring是真的强大，几乎所有工具都可以通过它来配置，极大提高开发效率。

首先，知道目的地[javax](eclipse-javadoc:%E2%98%82=BookCombinedBranch/D:\\/Maven\\/repo\\/m2\\/org\\/apache\\/geronimo\\/specs\\/geronimo-jms_1.1_spec\\/1.1.1\\/geronimo-jms_1.1_spec-1.1.1.jar<javax).[jms](eclipse-javadoc:%E2%98%82=BookCombinedBranch/D:\\/Maven\\/repo\\/m2\\/org\\/apache\\/geronimo\\/specs\\/geronimo-jms_1.1_spec\\/1.1.1\\/geronimo-jms_1.1_spec-1.1.1.jar<javax.jms).Destination

然后通过jms发送工具[org](eclipse-javadoc:%E2%98%82=BookCombinedBranch/D:\\/Maven\\/repo\\/m2\\/org\\/springframework\\/spring-jms\\/4.2.5.RELEASE\\/spring-jms-4.2.5.RELEASE.jar<org).[springframework](eclipse-javadoc:%E2%98%82=BookCombinedBranch/D:\\/Maven\\/repo\\/m2\\/org\\/springframework\\/spring-jms\\/4.2.5.RELEASE\\/spring-jms-4.2.5.RELEASE.jar<org.springframework).[jms](eclipse-javadoc:%E2%98%82=BookCombinedBranch/D:\\/Maven\\/repo\\/m2\\/org\\/springframework\\/spring-jms\\/4.2.5.RELEASE\\/spring-jms-4.2.5.RELEASE.jar<org.springframework.jms).[core](eclipse-javadoc:%E2%98%82=BookCombinedBranch/D:\\/Maven\\/repo\\/m2\\/org\\/springframework\\/spring-jms\\/4.2.5.RELEASE\\/spring-jms-4.2.5.RELEASE.jar<org.springframework.jms.core).JmsTemplate进行发送

/\*\*

\* 发送文本格式的消息

\* **@param** message

\*/

**public** **void** sendMessage(**final** String message) {

**try** {

jmsTemplate.send(destination, **new** MessageCreator() {

**public** Message createMessage(Session session) **throws** JMSException {

**return** session.createTextMessage(message);

}

});

} **catch**(Exception e) {

*logger*.error("BookExchangeProducer==>向队列{}发送消息失败，消息为：{}", destination, message);

e.printStackTrace();

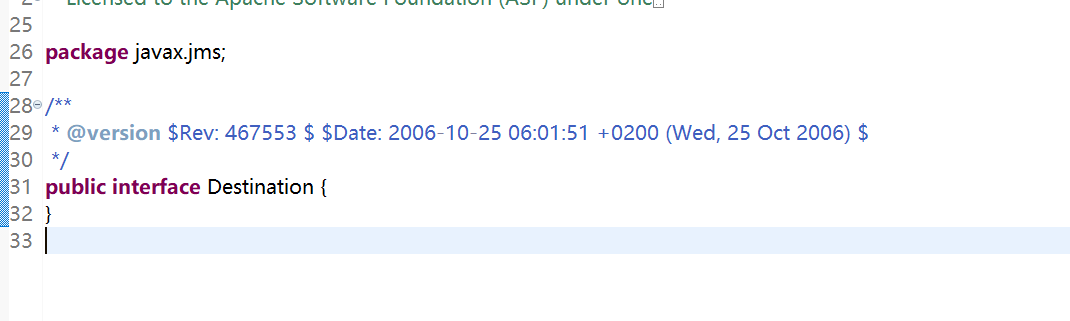
}

}

这个是发送文本格式的消息，还能发送map，Object，Byte，Stream。

我大概看了一下send方法的源码，各种内部类传参，是真的绕。反正，有了这三要素，我们就能发消息了。下面讲解一下[javax](eclipse-javadoc:%E2%98%82=BookCombinedBranch/D:\\/Maven\\/repo\\/m2\\/org\\/apache\\/geronimo\\/specs\\/geronimo-jms_1.1_spec\\/1.1.1\\/geronimo-jms_1.1_spec-1.1.1.jar<javax).[jms](eclipse-javadoc:%E2%98%82=BookCombinedBranch/D:\\/Maven\\/repo\\/m2\\/org\\/apache\\/geronimo\\/specs\\/geronimo-jms_1.1_spec\\/1.1.1\\/geronimo-jms_1.1_spec-1.1.1.jar<javax.jms).Destination和[org](eclipse-javadoc:%E2%98%82=BookCombinedBranch/D:\\/Maven\\/repo\\/m2\\/org\\/springframework\\/spring-jms\\/4.2.5.RELEASE\\/spring-jms-4.2.5.RELEASE.jar<org).[springframework](eclipse-javadoc:%E2%98%82=BookCombinedBranch/D:\\/Maven\\/repo\\/m2\\/org\\/springframework\\/spring-jms\\/4.2.5.RELEASE\\/spring-jms-4.2.5.RELEASE.jar<org.springframework).[jms](eclipse-javadoc:%E2%98%82=BookCombinedBranch/D:\\/Maven\\/repo\\/m2\\/org\\/springframework\\/spring-jms\\/4.2.5.RELEASE\\/spring-jms-4.2.5.RELEASE.jar<org.springframework.jms).[core](eclipse-javadoc:%E2%98%82=BookCombinedBranch/D:\\/Maven\\/repo\\/m2\\/org\\/springframework\\/spring-jms\\/4.2.5.RELEASE\\/spring-jms-4.2.5.RELEASE.jar<org.springframework.jms.core).JmsTemplate

## [javax](eclipse-javadoc:%E2%98%82=BookCombinedBranch/D:\\/Maven\\/repo\\/m2\\/org\\/apache\\/geronimo\\/specs\\/geronimo-jms_1.1_spec\\/1.1.1\\/geronimo-jms_1.1_spec-1.1.1.jar<javax).[jms](eclipse-javadoc:%E2%98%82=BookCombinedBranch/D:\\/Maven\\/repo\\/m2\\/org\\/apache\\/geronimo\\/specs\\/geronimo-jms_1.1_spec\\/1.1.1\\/geronimo-jms_1.1_spec-1.1.1.jar<javax.jms).Destination



这个接口的定义是如此的强悍！

Destination，目的地，就代表某一个消息队列。以ActiveMQ举例，一般将这个类org.apache.activemq.command.ActiveMQQueue的实例赋给Destination接口。ActiveMQQueue本身不复杂的，但是继承的ActiveMQDestination很复杂。算了，不想懂那么多。

## [org](eclipse-javadoc:%E2%98%82=BookCombinedBranch/D:\\/Maven\\/repo\\/m2\\/org\\/springframework\\/spring-jms\\/4.2.5.RELEASE\\/spring-jms-4.2.5.RELEASE.jar<org).[springframework](eclipse-javadoc:%E2%98%82=BookCombinedBranch/D:\\/Maven\\/repo\\/m2\\/org\\/springframework\\/spring-jms\\/4.2.5.RELEASE\\/spring-jms-4.2.5.RELEASE.jar<org.springframework).[jms](eclipse-javadoc:%E2%98%82=BookCombinedBranch/D:\\/Maven\\/repo\\/m2\\/org\\/springframework\\/spring-jms\\/4.2.5.RELEASE\\/spring-jms-4.2.5.RELEASE.jar<org.springframework.jms).[core](eclipse-javadoc:%E2%98%82=BookCombinedBranch/D:\\/Maven\\/repo\\/m2\\/org\\/springframework\\/spring-jms\\/4.2.5.RELEASE\\/spring-jms-4.2.5.RELEASE.jar<org.springframework.jms.core).JmsTemplate

这个很复杂。

为了实例化它，需要这么一长串配置。

<!-- 定义ReDelivery(重发机制)机制 ，重发时间间隔是100毫秒，最大重发次数是3次 http://www.kuqin.com/shuoit/20140419/339344.html -->

<bean id=*"activeMQRedeliveryPolicy"* class=*"org.apache.activemq.RedeliveryPolicy"*>

<!--是否在每次尝试重新发送失败后,增长这个等待时间 -->

<property name=*"useExponentialBackOff"* value=*"true"*></property>

<!--重发次数,默认为6次 这里设置为5次 -->

<property name=*"maximumRedeliveries"* value=*"5"*></property>

<!--重发时间间隔,默认为5秒 -->

<property name=*"initialRedeliveryDelay"* value=*"5000"*></property>

<!--第一次失败后重新发送之前等待500毫秒,第二次失败再等待500 \* 2毫秒,这里的2就是value -->

<property name=*"backOffMultiplier"* value=*"2"*></property>

<!--最大传送延迟，只在useExponentialBackOff为true时有效（V5.5），假设首次重连间隔为10ms，倍数为2，那么第

二次重连时间间隔为 20ms，第三次重连时间间隔为40ms，当重连时间间隔大的最大重连时间间隔时，以后每次重连时间间隔都为最大重连时间间隔。 -->

<property name=*"maximumRedeliveryDelay"* value=*"1000"*></property>

</bean>

<!-- 使用pool进行链接 -->

<!-- 真正可以产生Connection的ConnectionFactory，由对应的 JMS服务厂商提供-->

<bean id=*"targetConnectionFactory"* class=*"org.apache.activemq.ActiveMQConnectionFactory"*>

<!-- ActiveMQ服务地址 -->

<property name=*"brokerURL"* value=*"${jms.brokerUrl}"* />

<property name=*"userName"* value=*"${jms.username}"* />

<property name=*"password"* value=*"${jms.password}"* />

<property name=*"redeliveryPolicy"* ref=*"activeMQRedeliveryPolicy"* />

</bean>

<bean id=*"pooledConnectionFactory"* class=*"org.apache.activemq.pool.PooledConnectionFactory"*>

<property name=*"connectionFactory"* ref=*"targetConnectionFactory"*/>

<property name=*"maxConnections"* value=*"10"*/>

</bean>

<!-- 配置spring管理真正ConnectionFactory的ConnectionFactory，相当于spring对ConnectionFactory的一层封装 -->

<bean id=*"springConnectionFactory"* class=*"org.springframework.jms.connection.SingleConnectionFactory"*>

<property name=*"targetConnectionFactory"* ref=*"pooledConnectionFactory"*/>

</bean>

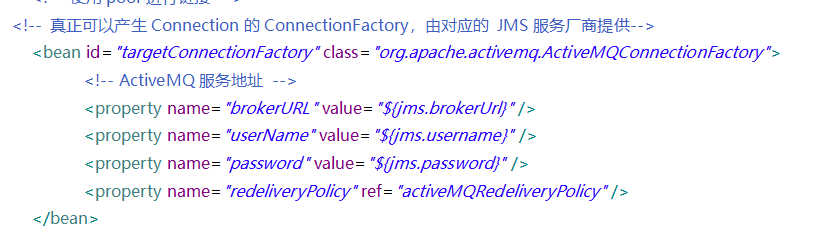
<!-- Spring提供的JMS工具类，它可以进行消息发送、接收等 -->

<bean id=*"jmsTemplate"* class=*"org.springframework.jms.core.JmsTemplate"*>

<!-- 这个connectionFactory对应的是我们定义的Spring提供的那个ConnectionFactory对象 -->

<property name=*"connectionFactory"* ref=*"springConnectionFactory"* />

</bean>



反正，至少要知道，连接消息队列，需要有地址，用户名，密码。

# consumer

消费者，消费消息，它也是个很普通的java类。

用spring的话让消费者实现这个接口会方便很多。

[org](eclipse-javadoc:%E2%98%82=BookCombinedBranch/D:\\/Maven\\/repo\\/m2\\/org\\/springframework\\/spring-jms\\/4.2.5.RELEASE\\/spring-jms-4.2.5.RELEASE.jar<org).[springframework](eclipse-javadoc:%E2%98%82=BookCombinedBranch/D:\\/Maven\\/repo\\/m2\\/org\\/springframework\\/spring-jms\\/4.2.5.RELEASE\\/spring-jms-4.2.5.RELEASE.jar<org.springframework).[jms](eclipse-javadoc:%E2%98%82=BookCombinedBranch/D:\\/Maven\\/repo\\/m2\\/org\\/springframework\\/spring-jms\\/4.2.5.RELEASE\\/spring-jms-4.2.5.RELEASE.jar<org.springframework.jms).[listener](eclipse-javadoc:%E2%98%82=BookCombinedBranch/D:\\/Maven\\/repo\\/m2\\/org\\/springframework\\/spring-jms\\/4.2.5.RELEASE\\/spring-jms-4.2.5.RELEASE.jar<org.springframework.jms.listener).SessionAwareMessageListener<M extends Message>

消费者和生产者一样，都要知道目的地是什么，照着这个配就得了，很多东西其实我都看不懂。有多个消费者的话，就定义多个org.springframework.jms.listener.DefaultMessageListenerContainer，名字随便起。

<!-- 配置自定义监听：MessageListener -->

<bean id=*"bookOrderDataQueueListener"* class=*"com.jiatu.activemq.consumer.BookOrderDataQueueListener"*/>

<!-- 将连接工厂、目标对了、自定义监听注入jms模板 -->

<bean id=*"sessionAwareListenerContainer"* class=*"org.springframework.jms.listener.DefaultMessageListenerContainer"*>

<property name=*"connectionFactory"* ref=*"springConnectionFactory"*/>

<property name=*"destination"* ref=*"bookOrderDataQueue"*/>

<property name=*"messageListener"* ref=*"bookOrderDataQueueListener"*/>

<!--应答模式是 INDIVIDUAL\_ACKNOWLEDGE-->

<property name=*"sessionAcknowledgeMode"* value=*"4"*/>

</bean>

# ReDelivery

消息重发。重发分两种场景，一种是发送失败了，需要重发，还有一种是发送成功了，但是业务处理失败了，需要重新发送，再处理一次。

这里讲业务处理失败后，消息如何重新发送。