Spring and JMS

- JmsTemplate
- Sending and Receiving Messages

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Spring and JMS

- Spring provides support for integrating JMS into Spring applications:
    - **JmsTemplate** type that handles boilerplate JMS coding
    - **Unchecked exceptions** for JMS exception handling
    - **MessageConverters** so you can centralize converting domain objects to/from JMS messages
    - **Gateway** classes that you can subclass in a similar fashion to JDBC DAO support classes
    - **POJO-based** alternative to message-driven beans for receiving JMS messages

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

The JmsTemplate Type

- This is the workhorse type for Spring programs that work with JMS

org.springframework.jms.core.JmsTemplate
convertAndSend(dest:Destination, obj:Object):void convertAndSend(obj:Object):void getDeliveryMode():int getMessageConverter():MessageConverter getPriority():int getReceiveTimeout():long getTimeToLive():long receive():Message receive(dest:Destination):Message receiveAndConvert():Object receiveAndConvert(dest:Destination):Object send(dest:Destination, creator:MessageCreator):void send(creator:MessageCreator):void setDeliveryMode(mode:int):void setDeliveryPersistent(b:boolean):void setPriority(priority:int):void setReceiveTimeout(timeout:long):void setTimeToLive(tol:long):void . . .

This notion of a "template" is a common Spring pattern that's also used with JDBC, Hibernate, iBATIS and other types of backends. The template essentially automates repetetive coding, making the Spring application simpler.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Accessing JNDI Resources

- JMS applications use **connection factories** and **destinations** (queues or topics) that are defined administratively and have JNDI names
- Spring applications can use the **jee:jndi-lookup** element to lookup the JMS artifacts so that Spring beans can reference them

```
1    <jee:jndi-lookup id="jmsConnectionFactory"
2      resource-ref="false"
3      jndi-name="jms/purchaseQCF" />
4
5    <jee:jndi-lookup id="purchaseQueue"
6       resource-ref="false"
7      jndi-name="jms/purchaseQ" />
```

If you create a local resource reference, then you specify resource-ref="true" - that would be the case if you created the resource reference in the web.xml for a Web application.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Configuring the JmsTemplate

- You can inject a JmsTemplate into a Spring bean
- You must configure the template with the JMS connection factory

```
1    <bean id="jmsTemplate"
2      class="org.springframework.jms.core.JmsTemplate">
3      <property name="connectionFactory"
4       ref="jmsConnectionFactory" />
5    </bean>
6
7    <bean id="myJmsSender" class="beans.MyJmsSender">
8      <property name="template" ref="jmsTemplate" />
9      <property name="destination" ref="purchaseQueue" />
10   </bean>
```


Here we show configuring a Spring bean that uses an injected template. Note how we also inject the JMS destination that we looked up previously.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

JmsTemplate Examples

MyJmsSender.java

MyJmsReceiver.java

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

```
1    package beans;
2
3    import javax.jms.Destination;
4    import javax.jms.JMSException;
5    import javax.jms.Message;
6    import javax.jms.ObjectMessage;
7    import javax.jms.Session;
8
9    import org.springframework.jms.core.JmsTemplate;
10   import org.springframework.jms.core.MessageCreator;
11
12   import domain.Purchase;
13
14   public class MyJmsSender
15   {
16       private JmsTemplate template;
17       private Destination destination;
18
19       public Destination getDestination()
20       {
21           return destination;
22       }
23
24       public void setDestination(Destination destination)
25       {
26           this.destination = destination;
27       }
28
29       public JmsTemplate getTemplate()
30       {
31           return template;
32       }
33
34       public void setTemplate(JmsTemplate template)
35       {
36           this.template = template;
37       }
38
39       public void sendPurchase(final Purchase p)
40       {
41           class MyMessageCreator implements MessageCreator
42           {
43               public Message createMessage(Session session)
44                   throws JMSException
45               {
```

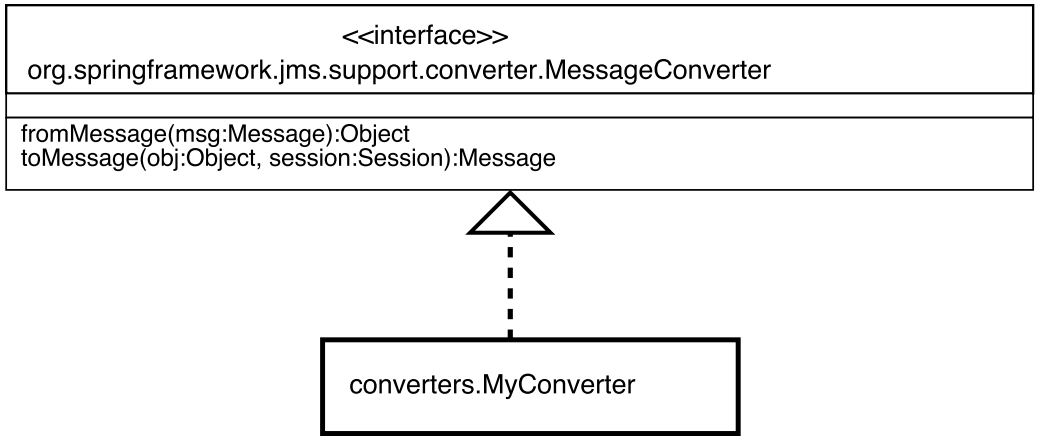
```
46                  ObjectMessage msg = session.createObjectMessage();
47                  msg.setObject(p);
48
49                  return msg;
50              }
51          }
52
53          MyMessageCreator mmc = new MyMessageCreator();
54          template.send(destination, mmc);
55      }
56
57
58
59  }
```

```
1    package beans;
2
3    import javax.jms.Destination;
4    import javax.jms.JMSException;
5    import javax.jms.ObjectMessage;
6
7    import org.springframework.jms.core.JmsTemplate;
8
9    import domain.Purchase;
10
11   public class MyJmsReceiver
12   {
13       private JmsTemplate template;
14       private Destination destination;
15
16       public Destination getDestination()
17       {
18           return destination;
19       }
20
21       public void setDestination(Destination destination)
22       {
23           this.destination = destination;
24       }
25
26       public JmsTemplate getTemplate()
27       {
28           return template;
29       }
30
31       public void setTemplate(JmsTemplate template)
32       {
33           this.template = template;
34       }
35
36       public Purchase receivePurchase()
37           throws JMSException
38       {
39           ObjectMessage msg = (ObjectMessage) template.receive(destination);
40           return (Purchase)msg.getObject();
41       }
42
43   }
```

2 - 6C : MyJmsReceiver.java

Converting Messages

- Often, programs need to convert domain objects to/from JMS messages (e.g. MapMessage)
- You can centralize such processing by writing a class that implements the **MessageConverter** interface and then call the template's **convertAndSend** and **receiveAndConvert** methods



In the example in this chapter, a converter isn't very helpful, since the example uses the JMS ObjectMessage type - that's easy to convert to/from a domain class like Purchase. But if you use other JMS message types, then writing a converter can save you from having to duplicate the code in several places.

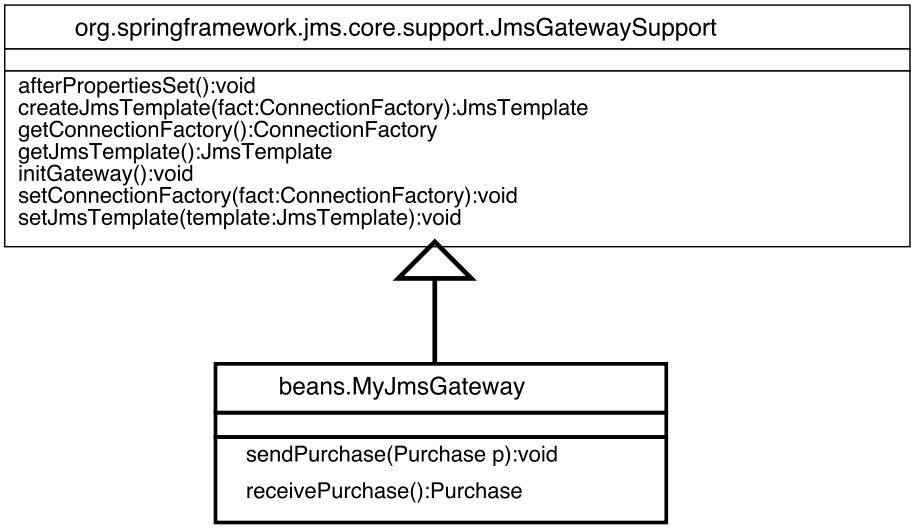
You must configure the converter in the Spring configuration file:

```
<bean id="myConverter"
   class="converters.MyConverter />
<bean id="jmsTemplate"
  class="org.springframework.jms.core.JmsTemplate">
    <property name="connectionFactory" ref="jmsConnectionFactory" />
    <property name="messageConverter" ref="myConverter" />
</bean>
```

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Subclassing JmsGatewaySupport

- In a fashion similar to DAO support classes, Spring provides the **JmsGatewaySupport** superclass that composes a JmsTemplate
- Using this superclass makes configuration a bit easier since it provides the template



You could configure the "gateway" in Spring.xml:

```
<bean id="myGateway"
  class="beans.MyJmsGateway" />
<property name="connectionFactory"
  ref="jmsConnectionFactory" / >
</bean>
```

Note that you don't need to explicitly configure the template!

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

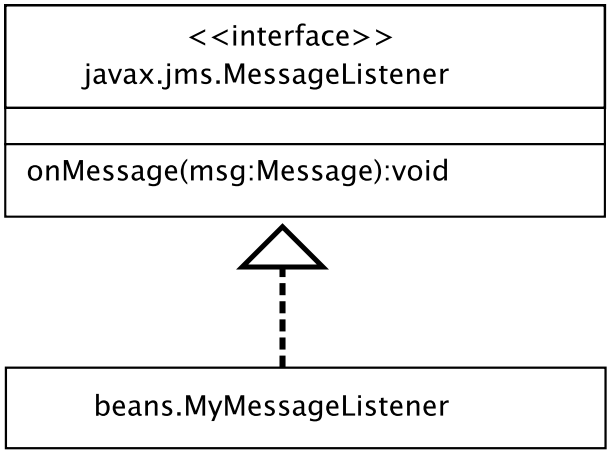
Receiving Messages

- There are three basic ways to act as a JMS receiver:
    - Use native JMS API with or without Spring
    - Use a message-driven Enterprise JavaBean
    - Use a Spring message-driven POJO
- In this chapter, we've already covered the first option and will now cover the third

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

The MessageListener Interface

- Spring beans that implement this interface can act as the receiver of JMS messages
- You must configure a Spring **message listener container** that actually receives the JMS messages and then calls the listener's **onMessage** method



Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Configuring a Message Listener Container

- Spring provides three types of message-listener containers
- Note that some application servers may require using one of the types exclusively

```
SimpleMessageListenerContainer

DefaultMessageListenerContainer

ServerSessionMessageListenerContainer
```


SimpleMessageListenerContainer provides the basic support to listen for JMS messages and then call your listener.

DefaultMessageListenerContainer does everything that SimpleMessageListenerContainer does, plus supports JMS transactions.

ServerSessionMessageListenerContainer does everything that DefaultMessageListenerContainer does, plus supports dynamic management of JMS sessions.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

POJO Listener Example

MyMessageListener.java

```
<!-- note that class name continues on two lines
   in this listing so it fits on the printed page -->
<bean
 class="org.springframework.jms.listener.
    DefaultMessageListenerContainer">
  <property name="connectionFactory"
     ref="jmsConnectionFactory" />
  <property name="destination" ref="purchaseQueue" />
  <property name="messageListener"
     ref="myMessageListener" />
</bean>

<bean id="myMessageListener"
  class="beans.MyMessageListener" />
```


This page shows a fragment of the spring.xml file to configure the message-driven POJO.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

```
1    package beans;
2
3    import java.io.FileWriter;
4    import java.io.IOException;
5    import java.io.PrintWriter;
6
7    import javax.jms.JMSException;
8    import javax.jms.Message;
9    import javax.jms.MessageListener;
10   import javax.jms.ObjectMessage;
11
12   import domain.Purchase;
13
14   public class MyMessageListener implements MessageListener
15   {
16
17       public void onMessage(Message m)
18       {
19           System.out.println("In onMessage");
20           ObjectMessage msg = (ObjectMessage)m;
21           try
22           {
23               Purchase p = (Purchase)msg.getObject();
24
25               PrintWriter out = new PrintWriter(
26                   new FileWriter("c:/temp/purchase.log"));
27
28               out.println("Date: " + p.getDate());
29               out.println("Amount: " + p.getAmount());
30               out.println("Reason: " + p.getReason());
31               out.close();
32           }
33           catch (JMSException e)
34           {
35               e.printStackTrace();
36           }
37           catch (IOException e)
38           {
39               e.printStackTrace();
40           }
41       }
42
43   }
```

Chapter Summary

In this chapter, you learned:

- About Spring's support for JMS applications
- How to code and configure Spring applications that send and receive JMS messages

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum