

Spring ve Java Message Service (JMS)



Spring ve JMS Desteği

- Spring, JMS API'sinin **kullanımını kolaylaştıran** imkanlar sunmaktadır
- JMS resource'larının (ConnectionFactory, Connection) **açılması, kapatılması** gibi işler otomatik olarak ele alınmaktadır
- Checked JMS exception'larının **unchecked exception'lara dönüşümü** gerçekleştirilir

Spring ve JMS Desteği

- JMS işlemlerinin **TX içerisinde** yürütülmesini sağlar
- **Java nesneleri ile JMS mesajları** arasındaki dönüşümler yapılır
- **JMS destinasyonlarına** (queue, topic) erişmeyi ve bunları yönetmeyi kolaylaştırır

JMS ConnectionFactory

- javax.jms.**ConnectionFactory**, JMS API'nin ana giriş noktasıdır
- Message Broker veya Uygulama **sunucusu tarafından** yönetilmektedir
- **JNDI lookup** ile erişilebilir
- Spring'in JMS kabiliyetlerini kullanabilmek için öncelikle bir **ConnectionFactory konfigürasyonu** yapılmalıdır

Spring JMS ConnectionFactory

- **DelegatingConnectionFactory**
 - JNDI'dan elde edilmiş başka bir ConnectionFactory nesnesini referans olarak kabul eder
 - Bütün JMS çağrılarını **hedef ConnectionFactory nesnesine** delege eder

DelegatingConnection Factory Konfigürasyonu

```
<beans...>
```

```
<bean id="connectionFactory"  
class="org.springframework.jms.connection.DelegatingConnectionFactory">  
  <property name="targetConnectionFactory"  
            ref="targetConnectionFactory"/>  
</bean>
```

```
<bean id="targetConnectionFactory"  
class="org.springframework.jndi.JndiObjectFactoryBean">  
  <property name="jndiEnvironment">  
    <value>  
      java.naming.factory.initial=org.apache.activemq.jndi.ActiveMQInitialContextFactory  
      java.naming.provider.url=tcp://localhost:61616  
    </value>  
  </property>  
  <property name="jndiName" value="ConnectionFactory"/>  
</bean>
```

```
</beans>
```

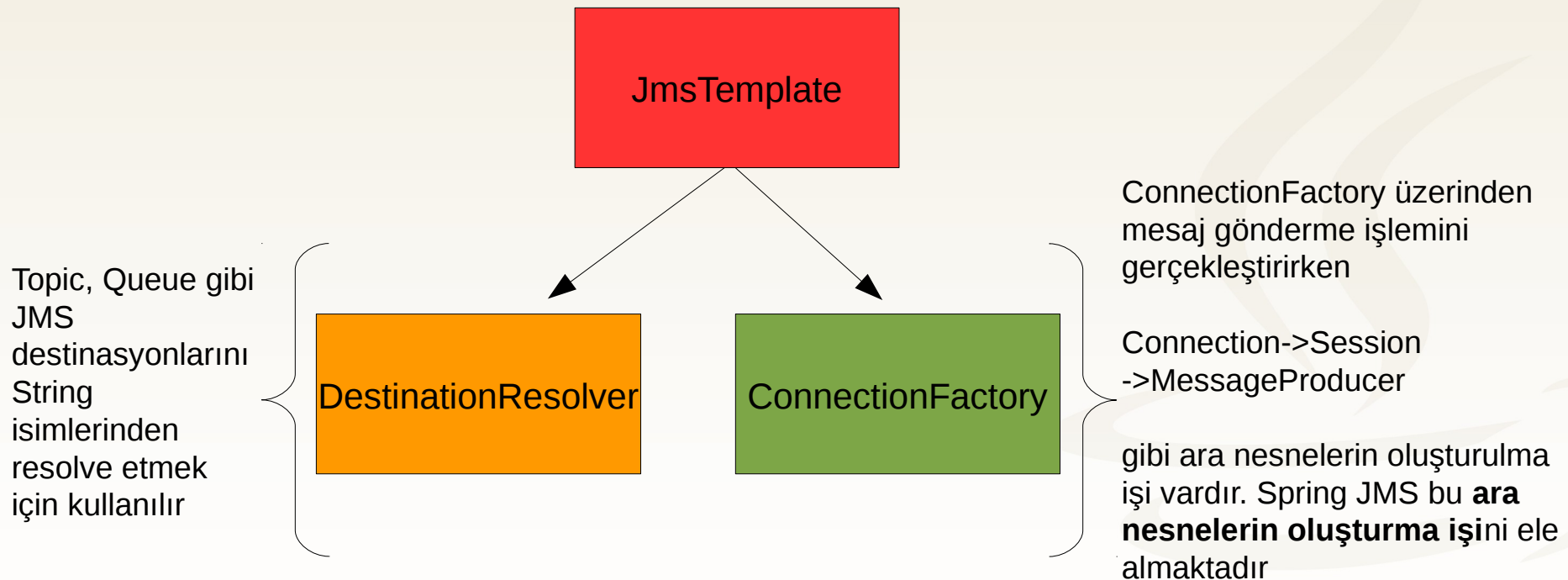
JmsTemplate ve Senkron Mesajlaşma

- **JmsTemplate ile senkron biçimde mesaj gönderim ve alım işlemleri yapılabilir**
- **Connection, Session oluşturma, kapama gibi resource yönetim işlemleri JmsTemplate tarafından otomatik yürütülmektedir**
- **Geliştirici sadece mesaj gönderme ve alma işlemine odaklanmaktadır**

JmsTemplate Konfigürasyonu

JmsTemplate **thread safe**'tir

deliveryMode, priority, time-to-live, receiveTimeout gibi QoS parametreleri bean properties olarak set edilebilmektedir



Bazı JMS provider'lar QoS değerlerini ConnectionFactory düzeyinde yönetebilirler. Bu durumda **isExplicitQoSEnabled=true** yapılmalıdır

JmsTemplate ile Mesaj Gönderimi

Mesajın gönderileceği Queue veya Topic javax.jms.Destination tipinde bir nesne olmalıdır

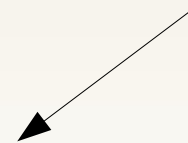
Mesajın destinasyonu olarak queue veya topic ismi verilebilir, bu durumda Queue/Topic nesnesine JNDI ile lookup yapılacaktır

```
jmsTemplate.send("myQueue", new MessageCreator() {  
  
    @Override  
    public Message createMessage(Session session)  
        throws JMSEException {  
  
        return session.createTextMessage("Hello there!");  
  
    }  
});
```

MessageCreator callback yardımı ile Session üzerinden JMS mesajı üretilir

JmsTemplate ile Mesaj Gönderimi

```
jmsTemplate.convertAndSend("myQueue", "Hello there!");
```



MessageConverter yardımı ile Java nesneleri ile JMS mesajları arasında çevrim yapmak da mümkündür

String - TextMessage,

byte[] - BytesMessage

java.util.Map - MapMessage

Serializable – ObjectMessage

JmsTemplate default durumda **SimpleMessageConverter** kullanır

JmsTemplate ile Mesaj Gönderimi

```
jmsTemplate.convertAndSend("myQueue", "Hello there!",  
    new MessagePostProcessor() {  
        @Override  
        public Message postProcessMessage(Message message)  
            throws JMSEException {  
  
            message.setJMSDeliveryMode(DeliveryMode.NON_PERSISTENT);  
  
            message.setIntProperty("messageNumber", 1);  
  
            return message;  
        }  
    });
```

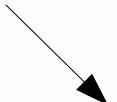


MessagePostProcessor ile mesaj çevriminden sonra header, properties bilgilerinin set edilmesi gibi ilave işlemler de yapılabilir

Session & Producer Callback Arayüzleri

```
jmsTemplate.execute(new SessionCallback<Object>() {  
  
    @Override  
    public Object doInJms(Session session) throws JMSException {  
        Queue queue = session.createQueue("myQueue");  
        MessageProducer producer = session.createProducer(queue);  
        TextMessage message = session.createTextMessage("Hello there!");  
        producer.send(message);  
        producer.close();  
        return null;  
    }  
});
```

```
jmsTemplate.execute("myQueue", new ProducerCallback<Object>() {  
  
    @Override  
    public Object doInJms(Session session, MessageProducer producer) throws JMSException {  
        TextMessage message = session.createTextMessage("Hello there!");  
        producer.send(message);  
        return null;  
    }  
});
```

 **SessionCallback** ve **ProducerCallback** arayüzleri vasıtası ile Session ve MessageProducer nesnelerine **doğrudan müdahale** etmek de mümkündür

JmsTemplate ile Senkron Mesaj Alımı

- **JmsTemplate.receive(..)** ile gerçekleştirilir
- Mesaj alımı boyunca **caller thread blocked** vaziyettedir
- Eğer karşı taraftan gelen mesaj yoksa **belirsiz süreli beklemelere** neden olabilir
- JmsTemplate'in **receiveTimeout** property değeri ile belirsiz süreli beklemelerin önüne geçilebilir

JmsTemplate ile Senkron Mesaj Alımı

```
javax.jms.Message msg = jmsTemplate.receive("myQueue");
```

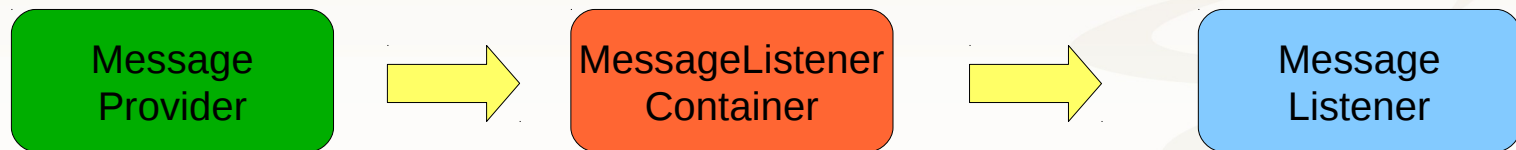
```
java.lang.Object msg =  
    jmsTemplate.receiveAndConvert("myQueue");
```

Asenkron Mesaj Alımı

- Asenkron mesaj alımı için ise **javax.jms.MessageListener** arayüzünü implement eden sınıflar yazılıp bean olarak tanımlanabilir
- Ya da **@JmsListener** anotasyonuna sahip metotlar ile de asenkron mesaj alımı gerçekleştirilebilir

Asenkron Mesaj Alımı

- Asenkron mesaj alımı arka planda **MessageListenerContainer** vasıtası ile gerçekleşir
- Bu **bean** mesaj alımı, JMS resource'larının yönetimi, hataların ele alınması, TX yönetimi gibi işlemleri yürütür



- Mesajı alır ve **MessageListener**'i invoke eder

MessageListener Container

- **MessageListenerContainer** arayüzünün iki farklı gerçekleştirimi vardır
 - **SimpleMessageListenerContainer**
 - Standalone ve test ortamları içindir
 - Sabit sayıda JMS Session ve consumer oluşturur
 - **DefaultMessageListenerContainer**
 - Gerçek ortamda bu kullanılır
 - Lokal TX yönetimi yapabilir
 - Global TX yönetimine dahil olma özelliği de vardır

MessageListener ile Asenkron Mesaj Alımı

Mesaj alımı yapacak sınıf MessageListener arayüzünü implement etmeli ve bean olarak tanımlanmalıdır

@Component

```
public class ExampleMessageListener implements MessageListener {  
  
    public void onMessage(Message message) {  
        if (message instanceof TextMessage) {  
            try {  
                System.out.println(((TextMessage) message).getText());  
            } catch (JMSEException ex) {  
                throw new RuntimeException(ex);  
            }  
        } else {  
            throw new IllegalArgumentException("Message must be of type TextMessage");  
        }  
    }  
}
```

MessageListener ile Asenkron Mesaj Alımı

```
<beans...>
```

```
<bean id="jmsContainer"  
class="org.springframework.jms.listener.DefaultMessageListenerContainer">  
  
    <property name="connectionFactory" ref="connectionFactory"/>  
  
    <property name="destinationResolver" ref="destinationResolver"/>  
  
    <property name="messageListener" ref="exampleMessageListener" />  
  
    <property name="destinationName" value="myTopic" />  
  
    <property name="pubSubDomain" value="true" />  
  
</bean>  
</beans>
```

MessageListener arayüzünü implement eden sınıf bean olarak tanımlanmış olmalıdır

MessageListenerAdapter ile Asenkron Mesaj Alımı

```
public class MessageReceiver {  
    public void receive(Message message) {  
        try {  
            TextMessage textMessage = (TextMessage) message;  
            System.out.println(">>>Message received :" +  
                               textMessage.getText());  
        } catch (JMSException ex) {  
            throw new RuntimeException(ex);  
        }  
    }  
}
```

```
<beans>  
  <bean id="messageReceiver" class="com.example.MessageReceiver"/>  
  
  <bean id="messageListener"  
        class="org.springframework.jms.listener.adapter.MessageListenerAdapter">  
    <property name="delegate" ref="messageReceiver"/>  
    <property name="defaultListenerMethod" value="receive"/>  
  </bean>  
</beans>
```

MessageListenerAdapter yardımı ile normal bir sınıf da MessageListener'a dönüştürülebilir

JMS Namespace Desteği

- JMS namespace kabiliyeti ile bean konfigürasyonları daha kolay biçimde gerçekleştirilebilir

```
<jms:listener-container connection-factory="connectionFactory"
    task-executor="taskExecutor"
    destination-resolver="destinationResolver"
    transaction-manager="transactionManager"
    concurrency="10">
```

```
    <jms:listener destination="queue.orders" ref="orderService"
method="placeOrder"/>
```

```
    <jms:listener destination="queue.logs" ref="confirmationLogger"
method="log"/>
```

```
</jms:listener-container>
```

Sıradan herhangi bir bean message listener olarak tanımlanabilir

@JmsListener ile Asenkron Mesaj Alımı

- `<jms:annotation-driven/>` namespace elemanı ile **@JmsListener** anotasyonuna sahip metotların JMS mesajlarını alması sağlanır

```
<beans ...>  
    <jms:annotation-driven />  
</beans>
```

- Java tabanlı konfigürasyonda ise bu namespace elemanının karşılığı **@EnableJms**'tir

@JmsListener ile Asenkron Mesaj Alımı

@Component

```
public class CustomMessageHandler {
```

```
    @JmsListener(destination="myQueue")
```

```
    public void handleMessage(Message message) {
```

```
        //...
```

```
    }
```

```
    @JmsListener(destination="myTopic", id="customMessageListenerContainer")
```

```
    public void handleMessage2(Message message) {
```

```
        //...
```

```
    }
```

```
}
```

Jms:annotation-driven elamanı arka planda default bir MessageListenerContainer bean'i tanımlayıp @JmsListener anotasyonuna sahip metotların mesajları handle etmesini sağlar

İstenirse default MessageListenerContainer yerine uygulama içerisinde tanımlanmış custom bir messageListenerContainer bean'i de kullanılabilir

JMS ve Transaction Yönetimi

- JMS mesaj alma ve gönderme işlemlerini **TX içerisinde** gerçekleştirmek mümkündür
- Spring bu iş için **JmsTransactionManager** sınıfını sunar
- Bu sınıf ile **lokal TX yönetimi** gerçekleştirilir

JMS ve Transaction Yönetimi

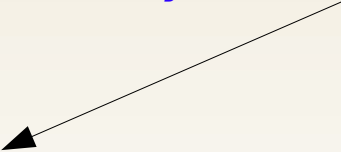
```
<bean id="transactionManager"
class="org.springframework.jms.connection.JmsTransactionManager">

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

</bean>

<bean id="connectionFactory"
class="org.springframework.jms.connection.DelegatingConnectionFactory">
    <property name="targetConnectionFactory"
                ref="targetConnectionFactory"/>
</bean>

<bean id="targetConnectionFactory"
class="org.springframework.jndi.JndiObjectFactoryBean">
    <property name="jndiEnvironment">
        <value>
            java.naming.factory.initial=org.apache.activemq.jndi.ActiveMQInitialContextFactory
            java.naming.provider.url=tcp://localhost:61616
        </value>
    </property>
    <property name="jndiName" value="ConnectionFactory"/>
</bean>
```



JMS ve Transaction Yönetimi

```
<bean id="jmsContainer"  
class="org.springframework.jms.listener.DefaultMessageListenerContainer">  
  
    <property name="connectionFactory" ref="connectionFactory"/>  
    <property name="destination" ref="destination"/>  
    <property name="messageListener" ref="messageListener" />  
  
    <property name="sessionTransacted" value="true"/>  
  
    <property name="transactionManager"  
ref="transactionManager" />  
  
</bean>
```

MessageListenerContainer'ın mesaj alımları sırasında TX'e participate edebilmesi için sessionTransacted=true tanımlanmalıdır

JMS ve Transaction Yönetimi

- JMS işlemlerinin harici Hibernate, JPA veya JDBC **transaction**'larına da dahil olmaları mümkündür
- **TransactionManager** bean konfigürasyonu olarak DataSource/JPA/HibernateTransactionManager'dan uygun olanı kullanılmalıdır
- Bu durumda JDBC ve ORM işlemleri ile JMS işlemleri **aynı local TX içerisinde** çalışmış olacaklardır

JMS ve Global Transaction Yönetimi

- **JtaTransactionManager** ile global TX'e de dahil olunabilir
- Bunun için JMS **ConnectionFactory** nesnesinin **XA kabiliyetine** sahip olması gerekir

İletişim

- **Harezmi** Bilişim Çözümleri
- Kurumsal Java Eğitimleri
- <http://www.java-egitimleri.com>
- info@java-egitimleri.com

