Spring Framework
Module 10 – JMS, EJB

Evgeniy Krivosheev
Last update: March, 2012

Contents

- JMS support in Spring
- EJB support Spring

Spring :: JMS :: JmsTemplate

LUXOFT TRAINING

The `JmsTemplate` class is used to work with **JMS** API 1.1 – subclass for accessing **JMS** ;

Spring :: JMS :: JmsTemplate

The `JmsTemplate` requires a reference to `ConnectionFactory`

`ConnectionFactory`:

- Is a part of JMS specification;
- Comes from JNDI;
- Used by client the application to create connections with JMS provider;
- Encapsulates various configuration parameters;

Spring :: JMS :: JmsTemplate

LUXOFT TRAINING

`Destination`:

- Object from JMS specification;
- Retrieved in JNDI;
- Is only known at runtime;

`DestinationResolver`:

- Object from Spring API;
- For defining `Destination` at runtime;

Spring :: JMS :: JmsTemplate

For asynchronous message reception you need `MessageListenerContainer`

- Object from Spring API;
- An intermediary between receiver and messaging provider;

Spring :: JMS :: Message Listener

LUXOFT TRAINING

- Registering and receiving messages;
- Resource management;
- Participating in transactions;
- Handling exceptions;
- The simplest implementation is `SimpleMessageListenerContainer`

Spring :: JMS :: Example

LUXOFT TRAINING

```
<bean id="connectionFactory"
      class="org.apache.activemq.
                    ActiveMQConnectionFactory">
    <property name="brokerURL"
              value="tcp://localhost:61616"/>
</bean>
<bean id="jmsTemplate"
      class="org.springframework.jms.core.JmsTemplate">
    <property name="connectionFactory">
        <ref local="connectionFactory"/>
    </property>
</bean>
```

Spring :: JMS :: Example

LUXOFT TRAINING

Sending a message

```
JmsTemplate template =
            (JmsTemplate) ctx.getBean("jmsTemplate");
Destination destination =
            (Destination) ctx.getBean("destination");
template.send(destination, new MessageCreator() {
        public Message createMessage(Session session)
                                        throws JMSException {
            return session.createTextMessage("Hello World");
        }
});
```

Spring :: JMS :: Example

LUXOFT TRAINING

Synchronous message reception

```
JmsTemplate template =
            (JmsTemplate) ctx.getBean("jmsTemplate");
Destination destination =
            (Destination) ctx.getBean("destination");
Message msg = template.receive(destination);
```

Spring :: JMS :: Example

LUXOFT TRAINING

Asynchronous message reception

```
public class ExampleListener
                        implements MessageListener{
    public void onMessage(Message message) {
        try {
            ((TextMessage) message).getText();
        } catch (JMSException ex) {
            throw new RuntimeException(ex);
        }
    }
}
```

Spring :: EJB support

- *Spring* is often considered as *EJB* replacement*;*
- However, these technologies can be combined;
- *Spring* facilitates working with *EJB ;*

Spring :: EJB support

- During the work with *EJB* test problems appear;
- One need to obtain beans from *JNDI* and invoke *create()* method*;*
- *Spring* facilitates testing procedure;
- Allows for declarative object configuration;

Spring :: EJB support

Spring Framework supports:

- *EJB 2.x ;*
- *EJB 3.x ;*
- If accessing *EJB,* there is no need to exactly know the version, because Spring will automatically detect the *EJB* version;
- For *EJB 3*, you can retrieve an object from *JNDI* instead of using *EJB* specific invoke method*;*

Spring :: EJB ::Example

LUXOFT TRAINING

Business Interface

```
public interface MyComponent {
    ...
}
```

Controller

```
...
public class private MyComponent myComponent;
public void setMyComponent(MyComponent myComponent) {
    this.myComponent = myComponent;
}
```

Spring :: EJB :: Example

LUXOFT TRAINING

Configuring *local interface*

```
<jee:local-slsb id="myComponent" jndi-name="ejb/myBean"
                business-interface="com.mycom.MyComponent"/>
<bean id="myController" class="com.mycom.myController">
    <property name="myComponent" ref="myComponent"/>
</bean>
```

Spring :: EJB :: Example

LUXOFT TRAINING

Configuring *remote interface*

```
<jee:remote-slsb id="myComponent" jndi-name="ejb/myBean"
                 business-interface="com.mycom.MyComponent"/>
<bean id="myController" class="com.mycom.myController">
    <property name="myComponent" ref="myComponent"/>
</bean>
```

Spring :: EJB :: Example

LUXOFT TRAINING

- It is necessary to implement *local* and *remote interface* and class that implements *SessionBean* and *MyComponent* (business interface) ;
- And bind controller and *EJB ;*

```
...
public class private MyComponent myComponent;
public void setMyComponent(MyComponent myComponent) {
    this.myComponent = myComponent;
}
```

Spring :: EJB :: Benefits

LUXOFT TRAINING

- If we want to change *EJB* for *POJO* or *mock* object we can only change the configuration without changing *Java* code ;
- Additionally, we don't have to write *JNDI* lookup or other plumbing code;
- Uniform accessing to *local* and *remote EJB*: there is no need to handle *RemoteException* in business methods for *remote EJB ;*
- If *RemoteException* is thrown when invoking remote bean, the *non-checked RemoteAccessException* will be thrown*;*

Spring :: EJB :: Features

- *Proxy* class for *EJB* is *singleton* (there is no need in *prototype*) ;
- Many implementations of bean containers pre-instantiate s*ingletons ;*
- There may be an attempt to create a *proxy* object before the target *EJB* is loaded *;*
- *Object is only created once in init() method and then cached ;*
- The solution is NOT pre-instantiate objects, but allow it to be created on first use ;
- This is controlled via the *lazy-init* attribute*;*
- *Transaction are provided through J2EE container, while Spring* only provides invoking *EJB* methods*;*

Any questions!?