Spring Framework
Module 10 – JMS, EJB

Evgeniy Krivosheev
Last update: March, 2012

Содержание

- Поддержка JMS в Spring
- Поддержка EJB в Spring

Spring :: JMS :: JmsTemplate

Для работы с **JMS** используется класс `JmsTemplate` для API 1.1 -вспомогательный класс, для доступа к **JMS** ;

Spring :: JMS :: JmsTemplate

Для `JmsTemplate` необходима ссылка на `ConnectionFactory`

`ConnectionFactory`:

- Объект из контракта JMS;
- Получается из JNDI;
- Используется клиентским приложением для создания соединения с JMS-провайдером;
- Включает в себя множество конфигурационных параметров;

Spring :: JMS :: JmsTemplate

LUXOFT TRAINING

`Destination`:

- Объект из контракта JMS;
- Получается из JNDI;
- Часто известен только во время выполнения;

`DestinationResolver`:

- Объект из Spring API;
- Для определения `Destination` во время выполнения;

Spring :: JMS :: JmsTemplate

Для асинхронного получения сообщений необходим `MessageListenerContainer`

- Объект из Spring API;
- Посредник между получателем и отправителем сообщений;

Spring :: JMS :: Message Listener

LUXOFT TRAINING

- регистрация и получение сообщений;
- управление ресурсами;
- участие в транзакциях;
- обработка исключений;
- простейшая реализация – `SimpleMessageListenerContainer`

Spring :: JMS :: Пример

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 :: Пример

LUXOFT TRAINING

Отправка сообщения

```
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 :: Пример

LUXOFT TRAINING

Синхронное получение сообщения

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

Spring :: JMS :: Пример

LUXOFT TRAINING

Асинхронное получение сообщения

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

Spring :: Поддержка EJB

LUXOFT TRAINING

- *Spring* рассматривается как альтернатива *EJB ;*
- Но эти технологии можно использовать совместно ;
- *Spring* упрощает работу с *EJB ;*

Spring :: Поддержка EJB

LUXOFT TRAINING

- При работе с *EJB* возникает проблема тестирования ;
- Необходимо получать бины из *JNDI* и вызывать метод *create() ;*
- *Spring* упрощает процесс тестирования ;
- Позволяет декларативно конфигурировать объекты ;

Spring :: Поддержка EJB

Spring Framework Поддерживает:

- *EJB 2.x ;*
- *EJB 3.x ;*
- При обращении к *EJB*, нет необходимости точно знать версию – Spring сам определит с какой версией *EJB* работает ;
- Для *EJB 3*, можно использовать не специфичный для *EJB* способ вызова, а получить объект из *JNDI ;*

Spring :: EJB :: Пример

LUXOFT TRAINING

Бизнес-интерфейс

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

Контроллер

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

Spring :: EJB :: Пример

LUXOFT TRAINING

Кофигурация *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 :: Пример

LUXOFT TRAINING

Кофигурация *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 :: Пример

LUXOFT TRAINING

- Необходимо реализовать *local* и *remote interface* и класс, реализующий *SessionBean* и *MyComponent* (бизнес-интерфейс) ;
- Плюс связать контроллер и *EJB ;*

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

Spring :: EJB :: Достоинства

LUXOFT TRAINING

- При необходимости заменить *EJB* на *POJO* или *mock* объект необходимо изменить только конфигурацию, не меняя *Java* код ;
- Также нет необходимости писать код поиска в *JNDI* и другой «служебный код» ;
- Однотипность обращения к *local* и *remote EJB* - нет необходимости обрабатывать *RemoteException* в бизнес-методах для *remote EJB ;*
- При возникновении *RemoteException* во время вызова удаленного бина, выбрасывается *non-checked RemoteAccessException ;*

Spring :: EJB :: Особенности

- *Proxy*-класс для *EJB - singleton* (нет необходимости в *prototype*) ;
- Многие реализации контейнеров бинов пре-инициализируют *singleton'ы ;*
- Может быть попытка создать *proxy* объект для которого еще не создан *EJB ;*
- *Объект создается один раз в методе init(), а потом кэшируется ;*
- Решение – *НЕ* пре-инициализировать объекты, а создавать их по первому запросу ;
- Это задается атрибутом *lazy-init ;*
- *Транзакции обеспечиваются J2EE контейнером, Spring* обеспечивает только вызов методов *EJB ;*

LUXOFT TRAINING

Вопросы!?