Better JMS Transport for CXF Webservice using Apache Camel

Better JMS Transport for CXF Webservice using Apache Camel

Configuring JMS in Apache CXF before Version 2.1.3 is possible but not really easy or nice. This article shows how to use Apache Camel to provide a better JMS Transport for CXF.

Update: Since CXF 2.1.3 there is a new way of configuring JMS (Using the JMSConfigFeature). It makes JMS config for CXF as easy as with Camel. Using Camel for JMS is still a good idea if you want to use the rich feature of Camel for routing and other Integration Scenarios that CXF does not support.

You can find the original announcement for this Tutorial and some additional info on Christian Schneider´s Blog

So how to connect Apache Camel and CXF

The best way to connect Camel and CXF is using the Camel transport for CXF. This is a camel module that registers with cxf as a new transport. It is quite easy to configure.

```
<bean class="org.apache.camel.component.cxf.transport.CamelTransportFactory">
  <property name="bus" ref="cxf" />
  <property name="camelContext" ref="camelContext" />
  <property name="transportIds">
    <list>
      <value>http://cxf.apache.org/transports/camel</value>
    </list>
  </property>
</bean>
```

This bean registers with CXF and provides a new transport prefix camel:// that can be used in CXF address configurations. The bean references a bean cxf which will be already present in your config. The other refrenceis a camel context. We will later define this bean to provide the routing config.

How is JMS configured in Camel

In camel you need two things to configure JMS. A ConnectionFactory and a JMSComponent. As ConnectionFactory you can simply set up the normal Factory your JMS provider offers or bind a JNDI ConnectionFactory. In this example we use the ConnectionFactory provided by ActiveMQ.

```
<bean id="jmsConnectionFactory" class="org.apache.activemq.ActiveMQConnectionFactory">
  <property name="brokerURL" value="tcp://localhost:61616" />
</bean>
```

Then we set up the JMSComponent. It offers a new transport prefix to camel that we simply call jms. If we need several JMSComponents we can differentiate them by their name.

```
<bean id="jms" class="org.apache.camel.component.jms.JmsComponent">
  <property name="connectionFactory" ref="jmsConnectionFactory" />
  <property name="useMessageIDAsCorrelationID" value="true" />
</bean>
```

You can find more details about the JMSComponent at the Camel Wiki. For example you find the complete configuration options and a JNDI sample there.

Setting up the CXF client

We will configure a simple CXF webservice client. It will use stub code generated from a wsdl. The webservice client will be configured to use JMS directly. You can also use a direct: Endpoint and do the routing to JMS in the Camel Context.

```
<client id="CustomerService" xmlns="http://cxf.apache.org/jaxws" xmlns:customer="http://customerservice.example.com/"
  serviceName="customer:CustomerServiceService"
  endpointName="customer:CustomerServiceEndpoint"
  address="camel:jms:queue:CustomerService"
  serviceClass="com.example.customerservice.CustomerService">
</client>
```

We explicitly configure serviceName and endpointName so they are not read from the wsdl. The names we use are arbitrary and have no further function but we set them to look nice. The serviceclass points to the service interface that was generated from the wsdl. Now the important thing is address. Here we tell cxf to use the camel transport, use the JmsComponent who registered the prefix "jms" and use the queue "CustomerService".

Setting up the CamelContext

As we do not need additional routing an empty CamelContext bean will suffice.

```
<camelContext id="camelContext" xmlns="http://activemq.apache.org/camel/schema/spring">
</camelContext>
```

Running the Example

- Download the example project here
- Follow the readme.txt

Conclusion

As you have seen in this example you can use Camel to connect services to JMS easily while being able to also use the rich integration features of Apache Camel.