Spring Support

We fully support Spring for configuration of the JMS client side as well as for configuring the JMS Message Broker.
There is a great article on using Spring with ActiveMQ - I'd recommend reading it first.

Configuring the JMS client

To configure an ActiveMQ JMS client in Spring it is just a simple matter of configuring an instance of ActiveMQConnectionFactory within a standard Spring XML configuration file like any other bean. There are several examples and test cases available and this one shows how to construct an ActiveMQConnectionFactory in Spring which is then passed into a Spring JmsTemplate for use by some POJOs.

e.g. the following fragment of XML shows us creating a JMS connection factory for ActiveMQ connecting to a remote broker on a specific host name and port.

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

The following shows how to use Zeroconf to discover the available brokers to connect to.

```
<bean id="jmsFactory" class="org.apache.activemq.ActiveMQConnectionFactory">
    <property name="brokerURL">
      <value>zeroconf://_activemq.broker.development.</value>
    </property>
  </bean>
```

From 1.1 of ActiveMQ onwards you can also use JNDI to configure ActiveMQ within Spring. This example shows how to configure Spring using ActiveMQ's JNDI Support.

Using Spring

If you are using the new XML Schema-based configuration of Spring 2.0 you can embed the ActiveMQ broker XML inside any regular Spring.xml file without requiring the above factory bean. e.g. here is an example of a regular Spring XML file in Spring 2.0 which also configures a broker.

```
<beans
  xmlns="http://www.springframework.org/schema/beans"
  xmlns:amq="http://activemq.apache.org/schema/core"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans
/spring-beans-2.0.xsd
  http://activemq.apache.org/schema/core http://activemq.apache.org/schema/core/activemq-core.xsd">

  <amq:broker useJmx="false" persistent="false">
    <amq:transportConnectors>
      <amq:transportConnector uri="tcp://localhost:0" />
    </amq:transportConnectors>
  </amq:broker>

  <amq:connectionFactory id="jmsFactory" brokerURL="vm://localhost"/>
</beans>
```

This allows you to configure JMS artifacts like destinations and connection factories together with the entire broker.

Working with Spring's JmsTemplate

Spring supports a handy abstraction, JmsTemplate, which allows you to hide some of the lower level JMS details when sending messages etc.

Please be aware that there are a number of JmsTemplate Gotchas to be careful of.

One thing to bear in mind with JmsTemplate is that by default it will create a new connection, session, producer for each message sent - then close them all down again. This is very inefficient! It is done like this to work in EJB containers which tend to use a special ConnectionFactory which does pooling.

If you are not using a JCA container to manage your JMS connections, we recommend you use our pooling JMS connection provider, (org.apache.activemq.pool.PooledConnectionFactory) from the `activemq-pool` library, which will pool the JMS resources to work efficiently with Spring's JmsTemplate or with EJBs.

e.g.

```
<!-- a pooling based JMS provider -->
  <bean id="jmsFactory" class="org.apache.activemq.pool.PooledConnectionFactory" destroy-method="stop">
    <property name="connectionFactory">
      <bean class="org.apache.activemq.ActiveMQConnectionFactory">
        <property name="brokerURL">
          <value>tcp://localhost:61616</value>
        </property>
      </bean>
    </property>
  </bean>

  <!-- Spring JMS Template -->
  <bean id="myJmsTemplate" class="org.springframework.jms.core.JmsTemplate">
    <property name="connectionFactory">
      <ref local="jmsFactory"/>
    </property>
  </bean>
```

The `PooledConnectionFactory` supports the pooling of Connection, Session and MessageProducer instances so it can be used with tools like Camel and Spring's JmsTemplate and MessagListenerContainer . Connections, sessions and producers are returned to a pool after use so that they can be reused later without having to undergo the cost of creating them again.

Note: while the `PooledConnectionFactory` does allow the creation of a collection of active consumers, it does not 'pool' consumers. Pooling makes sense for connections, sessions and producers, which can be seldom-used resources, are expensive to create and can remain idle a minimal cost. Consumers, on the other hand, are usually just created at startup and left going, handling incoming messages as they come. When a consumer is complete, it's preferred to shut down it down rather than leave it idle and return it to a pool for later reuse: this is because, even if the consumer is idle, ActiveMQ will keep delivering messages to the consumer's prefetch buffer, where they'll get held up until the consumer is active again.

If you are creating a collection of consumers (for example, for multi-threaded message consumption), you should consider keeping a low prefetch value (e.g. 10 or 20), to ensure that all messages don't end up going to just one of the consumers.

We do also have a pooling JMS ConnectionFactory for use inside a JCA / MDB container (org.apache.activemq.ra.InboundConnectionProxyFactory), when using our JCA Resource Adapter which will reuse the same JMS connection/session which is being used for inbound messages.

Consuming JMS from inside Spring

Spring's MessagListenerContainer should be used for message consumption. This provides all the power of MDBs - efficient JMS consumption and pooling of the message listeners - but without requiring a full EJB container.

You can use the `activemq-pool` `org.apache.activemq.pool.PooledConnectionFactory` for efficient pooling of the connections and sessions for your collection of consumers, or you can use the Spring JMS `org.springframework.jms.connection.CachingConnectionFactory` to achieve the same effect.

More Information

Also check out the following blogs for information about using Spring JMS with ActiveMQ:

- Synchronous Request Response with ActiveMQ and Spring
- Using Spring to Send JMS Messages
- Using Spring to Receive JMS Messages
- Tuning JMS Message Consumption In Spring