JMS and MDB sample application

{scrollbar}

top
Enterprise messaging has become an increasingly important component of loosely coupled, reliable enterprise frameworks. This is due in large part to the proliferation of enterprise applications and disparate enterprise resources, and the increasing need to integrate these applications into cohesive systems. Over the years Messaging and Message Oriented Middleware (MOM) has provided this integration proprietary manner. Introduction of Java Messaging Service (JMS) as a standard, eliminated many of the disadvantages in proprietary MOM based products. In addition,Message Driven Beans(MDBs) introduced together with Enterprise Java Beans 2.0 have served to get the best out of existing investments in J2EE application servers. Most of the J2EE application servers in modern era are acting as a MOM with a whole lot of value added services to JMS. As a J2EE 1.4 certified application server, Apache Geronimo comes into the party with support of JMS integrating with one of the best breed open source messaging frameworks, ActiveMQ. This article will provide you with a way to use JMS for your enterprise application scenario both as a local and remotely referred environments with Geronimo and ActiveMQ.

The company referred in this sample application sells one specific item in both retail and wholesale markets under different categories. All the placed orders in the application have to be authorized by a company sales employee before delivering goods to the customer. For the wholesale market, the company has placed their agents all over the country. They send their orders as a bunch at once, which is called a consignment. End users place their orders using the company web site while agents send their consignments with a special software installed in their premises. All the consignments must be approved by the company General Manager before it is handed over to a sales employee.

This is a typical application to use JMS as a solution because both consignment and order requests are processed in asynchronous manner.

After reading this article, you should be able to define Message Queues and their Connection Factories in Geronimo/ActiveMQ environment, send and receive messages using different kinds of applications in your Enterprise Application with ease.

This article is organized into following sections.

- Overview of JMS in Geronimo/ActiveMQ Enviroment
- Application Overview
- Configuring, Building and Deploying the Sample Application
- Testing of the Sample Application
- Summary

Overview of JMS in Geronimo/ActiveMQ Enviroment overview

Geronimo server comes with a JMS server and application components that can access JMS resources like connection factories, topics and queues from it. This JMS server is also known as message broker. The default message broker supported by Geronimo is ActiveMQ, usually does not need to be changed since it is a mature and feature-rich JMS product. This implementation uses inbuilt Derby database for the message persistent features.

ActiveMQ supports a large variety of transports (such as TCP, SSL, UDP, multicast, intra-JVM, and NIO) and client interactions (such as push, pull, and publish/subscribe). In the Geronimo context ActiveMQ supports MDBs, which are EJBs that consume JMS messages. It allows JMS applications to take J2EE specific features from Geronimo and application components such as JSPs, Servlets or EJBs utilizing JMS. Geronimo has implemented this JMS API in an abstract layer to support any JMS provider. It has achieved this feature by supporting J2EE Connector (JCA) specification. The JCA 1.5 specification details the contracts required between the application server and the driver supplied by ActiveMQ (resource adapter). Applications deployed in the Geronimo access ActiveMQ message broker only through this resource adapter(RA).

Application Overview application

Order processing application has two defined message queues to receive orders and consignments. Order requests can be generated and sent via the company's web application. When order requests are received to the order queue, a MDB will be triggered. It will carry out the next level of order request processing by saving those requests in to a server repository. Those saved order requests will be processed by a company employee later.

The company's sales agents are using the consignment sender application to send consignment (collection of orders) requests from their locations. First, they will prepare consignment as an XML file, then it will be passed as an application parameter. Consignment sender application will read the content of an XML file (with a consignment request) and send it to the consignment queue. General Manager in the company uses the consignment receiver application to find out the consignment requests. When a consignment request received to the consignment queue, consignment receiver listener application will download those requests to the General Manager's computer. He will then authorize it and hand it over to a sales employee for further processing.

Application contents

The core of the order placement application will be deployed as an EAR to the application server. Overview of the contents of EAR is given in the following depiction.

java |-Order.ear |- geronimo-activemq-ra-2.0-SNAPSHOT.rar |- jms-mdb-sample-ejb-2.0-SNAPSHOT.jar |-META-INF |- openejb-jar.xml |- jms-mdb-sample-ejb-2.0-SNAPSHOT.war |- index.jsp |- error.jsp |- WEB-INF |- web.xml |- classes |- META-INF |- application.xml |- geronimo-application.xml

MDB Implementation

The Message-Driven Bean uses the @MessageDriven annotation to replace the declaration of this MDB in the ejb-jar.xml file. By providing the annotation with further information it knows to look for a destination (in this case it happens to be a queue) to process. So this MDB will sit there and process messages passed into the 'OrderQueue.' The end result is that is echoes this message to the screen.

javasolidOrderRecvMDB.java // // MessageDrivenBean that listens to items on the // 'OrderQueue' queue and processes them accordingly. // @MessageDriven(activationConfig = { @ActivationConfigProperty(propertyName="destinationType", propertyValue="javax.jms.Queue"), @ActivationConfigProperty(propertyName="destination", propertyValue="OrderQueue") }) public class OrderRecvMDB implements MessageListener{ private static final String ORDER_MGMT_INFO = "order_mgmt.properties"; private static final String ORDER_REPO = "order.repo"; public OrderRecvMDB() { } public void onMessage(Message message) { TextMessage textMessage = (TextMessage) message; try { System.out.println("Order Received \n"+ textMessage.getText()); } catch (JMSException e) { e.printStackTrace(); } } }

In this application there is a MDB that will listen on **OrderQueue**. **openejb-jar.xml** tells Geronimo that there is a MDB which is associated with the **jms-resources** JMS Resource Group. It links **OrderRecvMDB** with **OrderQueue** via **CommonConnectionFactory**.

xmlsolidopenejb-jar.xml <openejb-jar xmlns="http://www.openejb.org/xml/ns/openejb-jar-2.1" xmlns:naming="http://geronimo.apache.org/xml/ns/naming-1.1" xmlns:security="http://geronimo.apache.org/xml/ns/security-1.1" xmlns:sys="http://geronimo.apache.org/xml/ns/deployment-1.2"> <sys:environment> <sys:moduleId> <sys:groupId>${pom.groupId}</sys:groupId> <sys:artifactId>${pom.artifactId}</sys:artifactId> <sys:version>${version}</sys:version> <sys:type>jar</sys:type> </sys:moduleId> <sys:dependencies> <sys:dependency> <sys:groupId>org.apache.geronimo.configs</sys:groupId> <sys:artifactId>activemq-broker</sys:artifactId> <sys:type>car</sys:type> </sys:dependency> </sys:dependencies> <sys:hidden-classes/> <sys:non-overridable-classes/> </sys:environment> <enterprise-beans> <message-driven> <ejb-name>OrderRecvMDB</ejb-name> <resource-adapter> <resource-link>jms-resources</resource-link> </resource-adapter> </message-driven> </enterprise-beans> </openejb-jar>

geronimo-application.xml and **application.xml** define the main components of the EAR. Both EJB component and Web archive information are given in these files as usual. This **geronimo-application.xml** also includes a section for defining a JMS queue and a common queue connection factory to access it. This is used for deploying the geronimo-activemq-ra.rar that is embedded in the ear.

xmlsolidgeronimo-application.xml <application xmlns="http://geronimo.apache.org/xml/ns/j2ee/application-1.1"> <environment xmlns="http://geronimo.apache.org/xml/ns/deployment-1.2"> <moduleId> <groupId>${pom.groupId}</groupId> <artifactId>${pom.artifactId}</artifactId> <version>${version}</version> <type>ear</type> </moduleId> </environment> <module> <connector>geronimo-activemq-ra-2.0-SNAPSHOT.rar</connector> <connector xmlns="http://geronimo.apache.org/xml/ns/j2ee/connector-1.2"> <dep:environment xmlns:dep="http://geronimo.apache.org/xml/ns/deployment-1.2"> <dep:moduleId> <dep:groupId>${pom.groupId}</dep:groupId> <dep:artifactId>jms-resources</dep:artifactId> <dep:version>${version}</dep:version> <dep:type>rar</dep:type> </dep:moduleId> <dep:dependencies> <dep:dependency> <dep:groupId>org.apache.geronimo.configs</dep:groupId> <dep:artifactId>activemq-broker</dep:artifactId> <dep:type>car</dep:type> </dep:dependency> </dep:dependencies> </dep:environment> <resourceadapter> <resourceadapter-instance> <resourceadapter-name>jms-resources</resourceadapter-name> <nam:workmanager xmlns:nam="http://geronimo.apache.org/xml/ns/naming-1.2"> <nam:gbean-link>DefaultWorkManager</nam:gbean-link> </nam:workmanager> </resourceadapter-instance> <outbound-resourceadapter> <connection-definition> <connectionfactory-interface>javax.jms.ConnectionFactory</connectionfactory-interface> <connectiondefinition-instance> <name>CommonConnectionFactory</name> <implemented-interface>javax.jms.QueueConnectionFactory</implemented-interface> <implemented-interface>javax.jms.TopicConnectionFactory</implemented-interface> <connectionmanager> <xa-transaction> <transaction-caching/> </xa-transaction> <single-pool> <match-one/> </single-pool> </connectionmanager> </connectiondefinition-instance> </connection-definition> </outbound-resourceadapter> </resourceadapter> <adminobject> <adminobject-interface>javax.jms.Queue</adminobject-interface> <adminobject-class>org.apache.activemq.command.ActiveMQQueue</adminobject-class> <adminobject-instance> <message-destination-name>OrderQueue</message-destination-name> <config-property-setting name="PhysicalName">OrderQueue</config-property-setting> </adminobject-instance> </adminobject> <adminobject> <adminobject-interface>javax.jms.Topic</adminobject-interface> <adminobject-class>org.apache.activemq.command.ActiveMQTopic</adminobject-class> </adminobject> </connector> </module> </application> xmlsolidapplication.xml <?xml version="1.0" encoding="UTF-8"?> <application xmlns="http://java.sun.com/xml/ns/javaee" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/application_5.xsd" version="5"> <description>Geronimo Sample EAR for jms-mdb-sample</description> <display-name>Geronimo Sample EAR for jms-mdb-sample</display-name> <module> <connector>geronimo-activemq-ra-2.0-SNAPSHOT.rar</connector> </module> <module> <ejb>jms-mdb-sample-ejb-2.0-SNAPSHOT.jar</ejb> </module> <module> <web> <web-uri>jms-mdb-sample-war-2.0-SNAPSHOT.war</web-uri> <context-root>/order</context-root> </web> </module> </application>

Client Implementation

The **OrderSenderServlet.java** servlet will parse the web form, create a message, and send that message to the OrderQueue via the CommonConnectoryFactory.

Please note that Geronimo ignores the 'mappedName' configuration attribute for @Resource. Instead, use 'name' when annotating. javasolidOrderSenderServlet.java public class OrderSenderServlet extends HttpServlet { @Resource(name="CommonConnectionFactory") private ConnectionFactory factory; @Resource(name="OrderQueue") private Queue receivingQueue; public void init() throws ServletException { super.init(); } protected void doGet(HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException { manageOrders(req,res); } protected void doPost(HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException { doGet(req,res); } private void manageOrders (HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException{ String path = "/error.jsp"; Connection connection = null; MessageProducer messageProducer = null; Session sess = null; try { String customerId = req.getParameter("customerId"); String orderId = req.getParameter("orderId"); String qty = req.getParameter("quantity"); String model = req.getParameter("model"); if(!customerId.equals("") && !orderId.equals("") && !qty.equals("")){ System.out.println("Start Sending Order Request"); // creating online order request String orderRequest = "<Order orderId=\""+orderId+"\" custId=\""+customerId+"\" qty=\""+qty+"\" model=\""+model+"\"/>" ; connection = factory.createConnection(); sess = connection.createSession (false, Session.AUTO_ACKNOWLEDGE); path = "/index.jsp"; TextMessage msg = sess.createTextMessage("<OrderId=" + orderId + " CustomerId=" + customerId + " Quantity=" + qty + " Model=" + model + ">"); messageProducer = sess.createProducer(receivingQueue); messageProducer.send(msg); System.out.println("Order Request Send"); } else{ String error = ""; if(customerId.equals("")){ error = "Customer Id Cannot be Empty"; }else if(orderId.equals("")){ error = "Order Id Cannot be Empty"; }else if(qty.equals("")){ error = "Quantity Cannot be Empty"; } req.setAttribute("error",error); } } catch (Exception e) { System.out.println("Error "+e); e.printStackTrace(); } finally { try { if(messageProducer != null) messageProducer.close(); if(sess != null)sess.close(); if (connection!= null)connection.close(); } catch (JMSException e) { e.printStackTrace(); } } getServletContext().getRequestDispatcher(path).forward(req,res); } }

web.xml of the archive has the relevant configurations for the both queue connection factory and the queue, which is essential to refer to resources in a local enviroment.

xmlsolidweb.xml <?xml version="1.0" encoding="UTF-8"?> <web-app version="2.5" xmlns="http://java.sun.com/xml/ns/javaee" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">

<description>JMS Servlet Sample</description> <servlet> <servlet-name>OrderSenderServlet</servlet-name> <servlet-class>org.apache.geronimo.samples.order.OrderSenderServlet</servlet-class> <load-on-startup>0</load-on-startup> </servlet> <servlet-mapping> <servlet-name>OrderSenderServlet</servlet-name> <url-pattern>/order</url-pattern> </servlet-mapping> <resource-ref> <res-ref-name>CommonConnectionFactory</res-ref-name> <res-type>javax.jms.QueueConnectionFactory</res-type> <res-auth>Container</res-auth> <es-sharing-scope>Shareable</res-sharing-scope> </resource-ref> <message-destination-ref> <message-destination-ref-name>OrderQueue</message-destination-ref-name> <message-destination-type>javax.jms.Queue</message-destination-type> <message-destination-usage>Produces</message-destination-usage> <message-destination-link>OrderQueue</message-destination-link> </message-destination-ref> <welcome-file-list> <welcome-file>/index.jsp</welcome-file> </welcome-file-list> </web-app>
Please note that this web application supports Servlet 2.5 specification. Some of the configurations in older versions (2.4) are slightly different than given in the above web.xml.

geronimo-web.xml is not necessary in this case as the annotations will help resolve the queue or connection factory references.

Tools used

The tools used for developing and building the order placement application are:

Apache Maven 2

Maven is a popular open source build tool for enterprise Java projects, designed to take much of the hard work out of the build process. Maven uses a declarative approach, where the project structure and contents are described, rather than the task-based approach used in Ant or in traditional make files, for example. This helps enforce company-wide development standards and reduces the time needed to write and maintain build scripts. The declarative, lifecycle-based approach used by Maven 1 is, for many, a radical departure from more traditional build techniques, and Maven 2 goes even further in this regard. Maven 2 can be download from the following URL:
http://maven.apache.org

Configuring, Building and Deploying the Sample Application configure

Download the order processing application from the following link:
jms-mdb-sample

After decompressing the given file, the **jms-mdb-sample** directory will be created.

Source Code

You can checkout the source code of this sample from SVN:

svn checkout http://svn.apache.org/repos/asf/geronimo/samples/trunk/samples/jms-mdb-sample

Building

The **jms-mdb-sample** folder will already contain an ear file ready to be deployed. However, you can still play with the source and build it yourself.

Use a command prompt to navigate into the **jms-mdb-sample** directory and just give **mvn install site** command to build. It will overwrite the **jms-mdb-sample-ear-2.0-SNAPSHOT.ear** under the **jms-mdb-sample** folder.

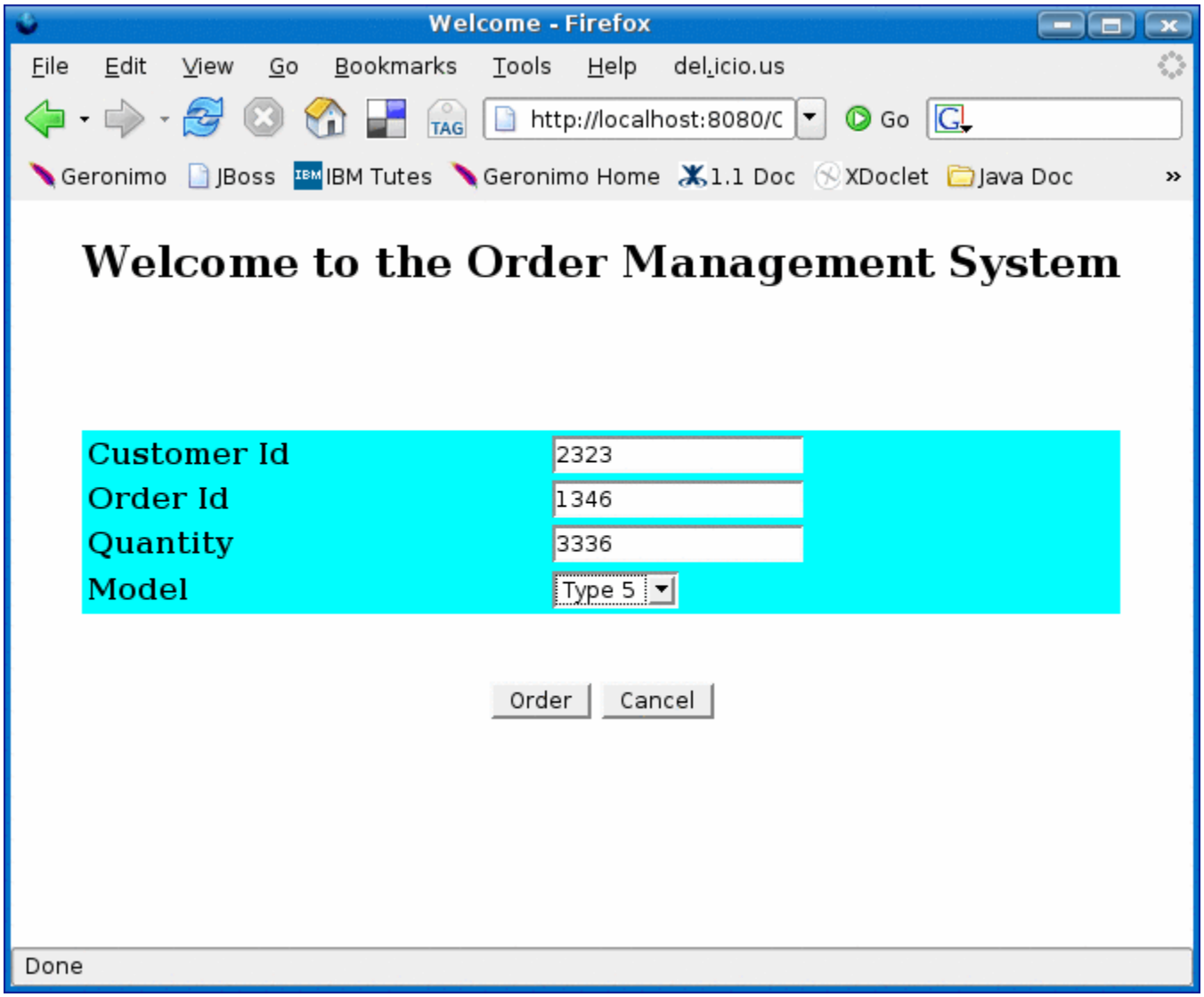
Deploying

Deploying Order processing sample application is pretty much the same as the deployment of JMS resources.

1. Navigate to **Deploy New** from the **Console Navigation** panel.
2. Load **jms-mdb-sample-ear-2.0-SNAPSHOT.ear** from **jms-mdb-sample** folder in to the **Archive** input box.
3. Press **Install** button to deploy application in the server.

Testing of the Sample Application testing

To test the sample web application open a browser and type http://localhost:8080/order. It will forward you in to the Order Management Welcome page. Then user has to fill the necessary information for the order placement and submit it.



After processing an order you will see the message printed to your console.

Summary summary

This article has demonstrated the use of JMS features in Apache Geronimo with the ActiveMQ JMS server. It provides a hypothetical example which extensively used JMS features.

Some of the highlights of this article : -

- Define JMS connection factories and related queues in a Geronimo enviroment.
- Message Driven Beans are the components listening on JMS queues providing by the J2EE container.