jms-mdb - JMS and MDB Sample Application

{scrollbar}

top
Asynchronous processing is often appropriate under requirements to maximize throughput possibly at the expense of latency. Java Message Service (JMS) supports a considerable range of asynchronous processing scenarios. Generally it is most appropriate when processing may be divided among a set of components that are under the control of one organization and have reasonably reliable inter-component communication. When the content or format of data to be transferred between components needs to be negotiated among several organizations or communication is less reliable web service solutions may be more appropriate. When the components are sufficiently lightweight or coupled a SEDA solution may be more appropriate.

Javaee message driven beans provide convenient support for components that need to process only one, independent message at a time. More complicated scenarios such as components that need to receive two or more messages to proceed or require in-order delivery are usually better handled with jms apis directly or message routing systems.

This sample provides a very simple example of an MDB supplied from a jms queue. There is a web app to feed the queue.

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 uses ActiveMQ as the default jms provider. JMS connectivity is provided through the J2CA connector provided by ActiveMQ. Deploying an instance of this resource adapter sets up connection factories and destinations for use by your application. By default Geronimo starts up an ActiveMQ message broker in the geronimo vm, but the resource adapter can be configured to connect to any ActiveMQ broker whether running inside geronimo or not.

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).

Application Overview application

The order processing application has a message queue for orders. Order requests can be generated and sent via the web application. When order requests are received on the order queue, a MDB will be triggered.

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 |-jms-mdb-ear-{version}.ear |- geronimo-activemq-ra-{geronimoVersion}.rar |- jms-mdb-ejb-{version}.jar |- jms-mdb-war-{version}.war |- index.jsp |- error.jsp |- WEB-INF |- web.xml |- classes |- META-INF |- 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.
Note the various @ActivationConfigProperty annotations. The property names are specific to the particular resource adapter used. Each resource adapter is required to specify an ActivationSpec javabean and the javabean properties are the allowable propertyNames for these annotations. The ActiveMQ ActivationSpec class is org.apache.activemq.ra.ActiveMQActivationSpec.

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{ public void onMessage(Message message) { TextMessage textMessage = (TextMessage) message; try { System.out.println("(mdb) Order Received \n"+ textMessage.getText()); } catch (JMSException e) { e.printStackTrace(); } } }

In this application there is a MDB that will listen on **OrderQueue**. The openejb-jar section of the geronimo plan indicates that the OrderRecvMDB MDB uses the **jms-resources** resource adapter instance.

xmlsolidopenejb-jar.xml <openejb-jar xmlns="http://openejb.apache.org/xml/ns/openejb-jar-2.2"> <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>

The connector section of the geronimo plan configures a resource adapter instance named "jms-resources" that has a ConnectionFactory to be used by the web client and the Order queue used by both client and mdb.

xmlsolidjms-resources.xml <connector xmlns="http://geronimo.apache.org/xml/ns/j2ee/connector-1.2"> <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> </connector> 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("(client) 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("(client) 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); } }

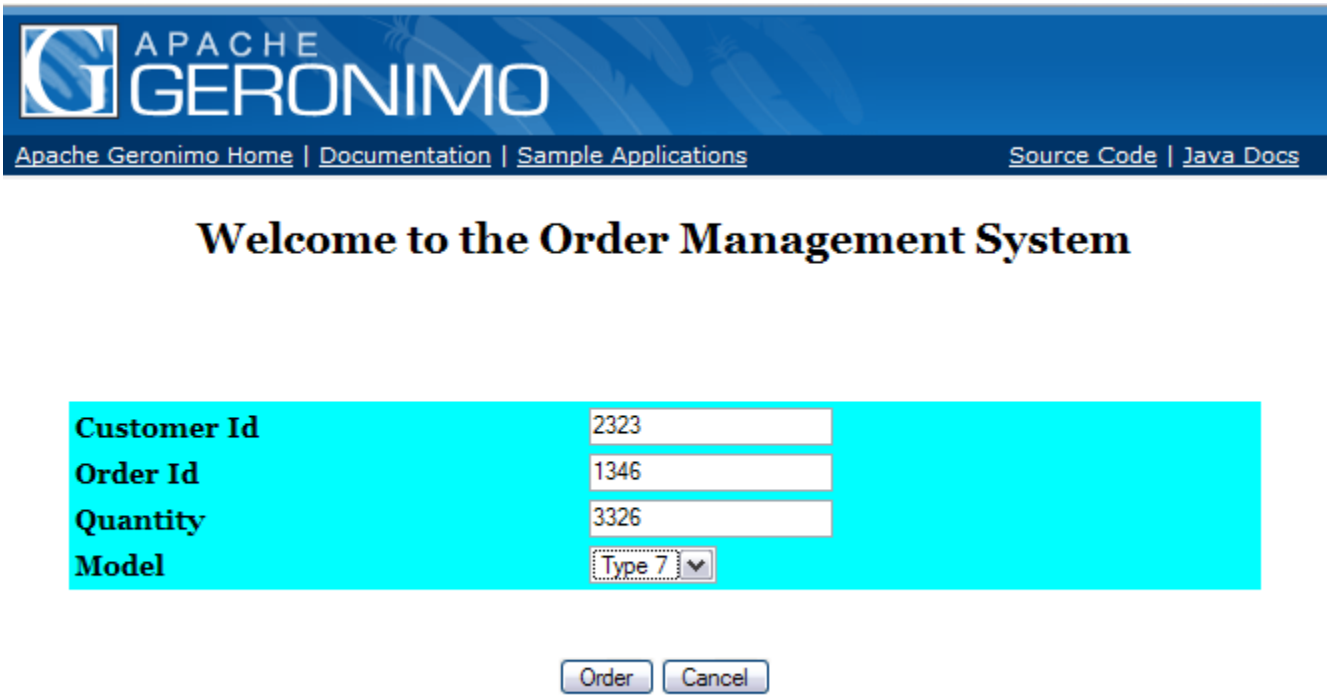
web.xml of the archive has the relevant configurations for the both queue connection factory and the queue, which is essential to refer to resources in a local enviroment.

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.

The geronimo plan does not need to include details about the web component as the annotations will help resolve the queue or connection factory references.

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.

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.