not-in-svn - Inbound JCA example

{scrollbar}

top
The J2EE Connector Architecture (JCA) provides a Java technology solution to the problem of connectivity between the many application containers and today's enterprise information systems (EIS). In the 1.5 release of the JCA specification, the architecture defines an inbound communication model, whereby the EIS initiates all communication to an EJB application. The mechanism allows the inbound resource adapters to invoke Enterprise Java Beans (EJB).

Overview overview

This document will describe how to deploy an EJB application, so that it can receive inbound events from a JCA Adapter. It will first describe the key parts of the Resource Adapter that pertain to the inbound communication model, as well as the Geronimo-specific plans that will deploy the Resource Adapter stand-alone. It will then describe the Enterprise Application that will be executed by the EIS, via the inbound communication mechanism.

Resource Adapter adapter

The code described in this section are classes that will be deployed as part of a resource adapter. For the most part, there is nothing Geronimo-specific about these files. However, the end of this section will discuss how to deploy the resource adapter into a Geronimo container.

In the JCA 1.5 specification, inbound adapters can support custom messaging formats. This allows a Message-Driven Bean (MDB) to implement any interface that has been defined by the inbound Resource Adapter. In the previous version of the JCA specification, Java Message Service (JMS) messages was the only mechanism for inbound communication, requiring the MDB to implement the interface javax.jms.MessageListener. With JCA 1.5, however, any interface definition can be used by a MDB to handle inbound requests.

Interface Definition

For this example, the interface **com.sample.connector.EventListener** interface has been defined. This interface will be implemented by any MDB that will be called by the inbound Resource Adapter.

solidEventListener.java package com.sample.connector; import java.util.List; import javax.resource.ResourceException; /** * An interface allowing Events to be Handled. Message-driven Beans implement * this interface, in order to receive inbound-events from our EIS via the JCA adapter. */ public interface EventListener { /** * Method to be called when a Event is handled. * * @param eventName the name of the event * @param paramList the parameters sent to the event * * @throws ResourceException if an error occurs */ void handleEvent(String eventName, List paramList) throws ResourceException; }

ActivationSpec implementation

Inbound resource adapters use implementations of the interface **javax.resource.spi.ActivationSpec**. The interface itself has no methods, but the class implementing the interface

- must be a JavaBean, providing both getter and setter methods to the bean's fields
- must be serializable

According the the API documentation, "the ActivationSpec implementation will hold the activation configuration information for a message endpoint". The message endpoint, in this case, will be an MDB in our enterprise application. Our ActivationSpec implementation is the class **com.sample.connector. EventActivationSpec.** Our example's ActivationSpec will store all the data required so that a connection can be opened, in order for the remote EIS system to call into the application container (machineName, port, user name, user password, and event pattern).

solidEventActivationSpec.java /* * EventActivationSpec.java */ package com.sample.connector; import java.io.Serializable; import javax.resource.spi. ActivationSpec; import javax.resource.spi.InvalidPropertyException; import javax.resource.spi.ResourceAdapter; /** * Implementation of the <code>ActivationSpec</code> interface, which allows EIS Events * to be exposed to message-drive beans. This <code>ActivationSpec</code> is used to * support inbound connection from our EIS to a message-driven bean. This will open * an EIS connection and use its event mechanism provided by the EIS-specific connection * class. The connection to the EIS will be held open while the application containing * the message-driven bean is available. */ public class EventActivationSpec implements ActivationSpec, Serializable { private ResourceAdapter resourceAdapter; private String serverName; private Integer portNumber; private String userName; private String password; private String eventPatterns; //A comma-delimeted string of patterns /** * Creates a new instance of the EventActivationSpec class. */ public EventActivationSpec() { } /** * No validation is performed */ public void validate() throws InvalidPropertyException { } //javadoc inherited public ResourceAdapter getResourceAdapter() { return resourceAdapter; } //javadoc inherited public void setResourceAdapter(ResourceAdapter resourceAdapter) { this.resourceAdapter = resourceAdapter; } /** * Getter method for the ServerName attribute. This allows the server name * to be defined against a Connection pool * * @return a <code>String</code> containing the server name value */ public String getServerName() { return serverName; } /** * Setter method for the ServerName attribute. This allows the server name to be * defined against a connection pool/. * * @param serverName the <code>String</code> that will be defined against this attribute */ public void setServerName(String serverName) { this.serverName = serverName; } /** * Getter method for the PortNumber attribute. This allows the port number * to be defined against a Connection pool * * @return a <code>Integer</code> containing the server port value */ public Integer getPortNumber() { return portNumber; } /** * Setter method for the PortNumber attribute. This allows the server port to be * defined against a connection pool/. * * @param portNumber the <code>Integer< /code> that will be defined against this attribute */ public void setPortNumber(Integer portNumber) { this.portNumber = portNumber; } /** * Getter method for the UserName attribute. This allows the user name * to be defined against a Connection pool * * @return a <code>String</code> containing the user name value */ public String getUserName() { return userName; } /** * Setter method for the UserName attribute. This allows the user name to be * defined against a connection pool/. * * @param userName the <code>String</code> that will be defined against this attribute */ public void setUserName(String userName) { this.userName = userName; } /** * Getter method for the Password attribute. This allows the password * to be defined against a Connection pool * * @return a <code>String</code> containing the password value */ public String getPassword() { return password; } /** * Setter method for the Password attribute. This allows the password to be * defined against a connection pool/. * * @param password the <code>String</code> that will be

defined against this attribute */ public void setPassword(String password) { this.password = password; } /** * Returns the event patterns that will be used when subscribing to Events. This string can contain * comma-delimited value, in order to subscribe to multiple Events. * * @return a <code>String</code> containing the event patterns that will be subscribed */ public String getEventPatterns() { return eventPatterns; } /** * Defines the event patterns that will be subscribed to by the message-driven beans. The value can * contain comma-delimited patterns, such that multiple events can be subscribed to in the single bean. * * @param eventPatterns a <code>String</code> containing the single pattern or comma-delimited patterns */ public void setEventPatterns (String eventPatterns) { this.eventPatterns = eventPatterns; } }

Common Deployment Descriptor

In the JCA deployment descriptor (ra.xml), there is a section that allows the adapter to associate an interface with the ActivationSpec implementation. This is required, so that the application container can ensure that mandatory properties are defined in the MDB that will be called by the adapter.

solidra.xml <?xml version="1.0" encoding="UTF-8"?> <connector xmlns="http://java.sun.com/xml/ns/j2ee" xmlns:xsi="http://www.w3.org/2001 /XMLSchema-instance" xmlns:schemaLocation="http://java.sun.com/xml/ns/j2ee http://java.sun.com/xml/ns/j2ee/connector_1_5.xsd" version="1.5"> <display-name>EIS Connector</display-name> <vendor-name>My Component</vendor-name> <eis-type>My Remote Server</eis-type> <resourceadapter-version>1.0</resourceadapter-version> <resourceadapter> <resourceadapter-class>com.sample.connector.MyResourceAdapter< /resourceadapter-class> <!-- snip --> <inbound-resourceadapter> <messageadapter> <messagelistener> <messagelistener-type>com.sample.connector. EventListener</messagelistener-type> <activationspec> <activationspec-class>com.sample.connector.EventActivationSpec</activationspec-class> <required-config-property> <config-property-name>ServerName</config-property-name> </required-config-property> <required-config-property> <config-property-name>PortNumber</config-property-name> </required-config-property> <required-config-property> <config-property-name>UserName</config-property-name> </required-config-property> <required-config-property> <config-property-name>Password</config-property-name> </required-config-property> <required-config-property> <config-property-name>EventPatterns</config-property-name> </required-config-property> </activationspec> < /messagelistener> </inbound-resourceadapter> <security-permission> <description>Permissions allowed to the EIS Connector</description> <security-permission-spec/> </security-permission> </resourceadapter> </connector>

Once an ActivationSpec has been implemented, the Resource Adapter's **endpointActivation** method can be updated. The application container will call this method on the Resource Adapter when a MDB is deployed into the environment. This allows the inbound adapter to open any connections. The method's signature is

public void endpointActivation(MessageEndpointFactory endpointFactory, ActivationSpec spec) throws ResourceException

The endpointFactory provides the ability to create new instances of MDBs, and the spec will be the implementation of ActivationSpec, which will contain all the required properties. The implementation of this method will be specific to the mechanism used by the EIS to support messaging. Typically, the resource adapter will define implementations of the **javax.resource.spi.work.Work** interface, which will service the inbound requests without directly creating new Threads.

Equally important is to implement the **endpointDeactivation** method. This allows the adapter to free any resources that were created when the endpoint was activated.

Geronimo Deployment Descriptor

Finally, we've reached the Geronimo-specific details. The Geronimo Deployment Descriptor provides a section that defines specific inbound Resource Adapters.

solidgeronimo-ra.xml <?xml version="1.0" encoding="UTF-8"?> <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>com.sample</dep:groupId> <dep: artifactId>myConnector</dep:artifactId> <dep:version>1.0</dep:version> <dep:type>rar</dep:type> </dep:moduleId> <dep:dependencies/> <dep:hidden-classes/> <dep:non-overridable-classes/> </dep:environment> <resourceadapter> <resourceadapter-instance> <resourceadapter-name>MyInboundEvents</resourceadapter-name> <workmanager> <gbean-link>DefaultWorkManager</gbean-link> </workmanager> </resourceadapter-instance> </resourceadapter> </connector>

When deploying the adapter's RAR file with the above deployment plan, a single inbound adapter will be created, named **MyInboundEvents**.

EJB Application application

At this point, we've defined and deployed our resource adapter into the Geronimo server instance. Now, an EJB application can be written and deployed that contains a Message-Driven Bean that will be called by the inbound Resource Adapter when it receives a message from the remote EIS.

In this example, a Message-Driven Bean implements the EventListener interface that was defined by the Resource Adapter.

solidEventBean.java package com.sample.eventDemo.message; import com.sample.connector.EventListener; import java.util.List; import javax.ejb. ActivationConfigProperty; import javax.ejb.MessageDriven; import javax.resource.ResourceException; /** * Message-driven Bean, listening for Events */ @MessageDriven(name="EventBean", messageListenerInterface=EventListener.class, activationConfig={ @ActivationConfigProperty(propertyName=" ServerName",propertyValue="fozzie"), @ActivationConfigProperty(propertyName="PortNumber",propertyValue="40100"), @ActivationConfigProperty (propertyName="UserName",propertyValue="user"), @ActivationConfigProperty(propertyName="Password",propertyValue="pwd"), @ActivationConfigProperty(propertyName="EventPatterns",propertyValue="myPattern") }) public class EventBean implements EventListener { /** * Method that will be called by the inbound Event handler */ public void handleEvent(String eventName, List paramList) throws ResourceException { System. out.println("In MDB: " + eventName + " with params " + paramList.toString()); } }

The bean doesn't do much, but it will show that the bean is being executed. However, it is using the ActivationConfigProperty annotation to define the properties required by the inbound resource adapter. These values will be defined against the appropriate ActivationSpec (in this case, the EventActivationSpec), which will be passed to the resource adapter's endpointActivation method.

Geronimo Deployment Descriptor

The final piece is to associate the MDB with the inbound Resource Adapter MyInboundEvents that was defined when the Resource Adapter was deployed into the Geronimo container. This is done in the EJB's Geronimo Deployment Descriptor, **openejb-jar.xml**.

solidopenejb-jar.xml <?xml version="1.0" encoding="UTF-8"?> <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>com.sample.connectorDemo</sys:groupId> <sys:artifactId>SampleEventHandler</sys:artifactId> <sys:version>1.1</sys:version> <sys:type>car</sys:type> </sys:moduleId> <sys:dependencies> <sys:dependency> <sys:groupId>com.sample</sys:groupId> <sys:artifactId>myConnector</sys:artifactId> <sys:version>1.0</sys:version> <sys:type>rar</sys:type> </sys:dependency> </sys:dependencies> <sys:hidden-classes/> <sys:non-overridable-classes/> </sys:environment> <enterprise-beans> <message-driven> <ejb-name>EventBean</ejb-name> <resource-adapter> <resource-link>MyInboundEvents</resource-link> </resource-adapter> </message-driven> </enterprise-beans> </openejb-jar>

In this file, we're associating the EventBean with the MyInboundEvents adapter.