Integrating Foreign JMS Providers with BEA WebLogic Server

Greg Brail
Senior Software Engineer
BEA Systems, Inc.
Wednesday, March 5, 2003
9:45:00 AM - 10:45 AM
Tampa 1, 2 & 3
1097-90
Disclaimer

This information represents work in progress
This information is NOT a commitment by BEA
This information is subject to change
Learning Objectives

• As a result of this presentation, you will be able to:
  • Understand why BEA WebLogic Server can work with different JMS providers
  • Configure an EJB, servlet, or the Messaging Bridge to use various JMS providers
  • Use these features in a flexible and efficient way
Speaker’s Qualifications

- **Greg Brail** is a software developer at BEA Systems, Inc.

- **Greg Brail** works on the BEA messaging team, and has implemented some of the features described here

- **Greg Brail** has had more than ten years experience designing and using messaging systems at BEA, IBM, and Transarc
Presentation Agenda

- JMS Interoperability Concepts
- Message-Driven Beans
- The Messaging Bridge
- Using JMS in J2EE Components
- Foreign JMS Provider Definitions
JMS Interoperability Concepts

- To support any JMS provider, a client must:
  - Look up the `ConnectionFactory` and `Destination` objects using JNDI
  - Create all other JMS objects using the `ConnectionFactory` object
You need **four common properties** to connect to a JMS provider

- The “initial context factory” name
  - This is the name of the class that implements the JNDI directory
- The URL to reach the JNDI provider
- The name of the JMS connection factory object to look up in JNDI
- The name of the JMS destination object to look up in JNDI
Example: BEA WebLogic JMS

Create JNDI initial context:
Initial Context Factory = weblogic.jndi.WLInitialContextFactory
Provider URL = t3://localhost:7001/

Look up ConnectionFactory Object
Returns weblogic.jms.client.JMSConectionFactory

Look up Destination
Returns weblogic.jms.common.DestinationImpl

Create Connection and Session from factory

Create Consumer or Producer using Destination

WebLogic Server
Example: BEA WebLogic JMS

• When you configure a BEA WebLogic JMS connection factory, you specify “JNDI Name”
  • This is the name that you look up

• When you configure a WLS JMS queue or topic, you also specify a “JNDI Name”

• When using WLS JMS, from within the same cluster, you should not specify a URL
Example: IBM MQSeries JMS

Create JNDI initial context:
Initial Context Factory = com.sun.jndi.ldap.LdapCtxFactory
Provider URL = ldap://localhost:389/c=US,o=BEA/

Look up ConnectionFactory Object
Returns com.ibm.mq.jms.MQQueueConnectionFactory

Look up Destination
Returns com.ibm.mq.jms.MQQueue

Create Connection and Session from factory

Create Consumer or Producer using Destination

JNDI Provider (LDAP Directory)

MQ Queue Manager
A Message-Driven Bean (MDB) is

- An Enterprise Java Bean (EJB)
- Invoked every time a message is received from a JMS queue or topic
- The MDB is a full-fledged EJB that can use transactions, security, JDBC, call other EJBs, and so forth
Configuring an MDB

• Here’s what goes in weblogic-ejb-jar.xml:

```xml
<weblogic-enterprise-bean>
  <ejb-name>UseMQHere</ejb-name>
  <message-driven-descriptor>
    <pool>
      <max-beans-in-free-pool>4</max-beans-in-free-pool>
      <initial-beans-in-free-pool>1</initial-beans-in-free-pool>
    </pool>
    <destination-jndi-name>AnMQQueue</destination-jndi-name>
    <initial-context-factory>
      com.sun.jndi.ldap.LdapCtxFactory
    </initial-context-factory>
    <provider-url>
      ldap://bigserver/c=US,o=BEA,ou=JMSTeam
    </provider-url>
    <connection-factory-jndi-name>QCF</connection-factory-jndi-name>
  </message-driven-descriptor>
</weblogic-enterprise-bean>
```
• When using WLS JMS with an MDB:
  • Do not specify “initial context factory”
  • Do not specify connection factory name
  • Only specify URL if the MDB and JMS destination are in different clusters

```xml
<weblogic-enterprise-bean>
  <ejb-name>WLSJMSIsGreat</ejb-name>
  <message-driven-descriptor>
    <pool>
      <max-beans-in-free-pool>4</max-beans-in-free-pool>
      <initial-beans-in-free-pool>1</initial-beans-in-free-pool>
    </pool>
    <destination-jndi-name>AnMQQueue</destination-jndi-name>
  </message-driven-descriptor>
</weblogic-enterprise-bean>
```
MDBs and Transactions

• When an MDB has in its descriptor files
  • A “transaction-type” of “Container”
  • A “trans-attribute” of “Required”

• Then the MDB’s “onMessage” method and the JMS provider are part of the same two-phase commit transaction
• In BEA WebLogic Server 6.1, MDBs support two-phase commit transactions with WebLogic JMS only

• In BEA WebLogic Server 7.0, MDBs support two phase commit transactions with foreign JMS providers that support XA
The Messaging Bridge

• The Messaging Bridge is a feature that forwards messages
  • From one JMS queue or topic (the “source”)
  • To another (the “target”)
  • Either one may be any JMS provider
The Messaging Bridge

• An instance of the Bridge moves messages between two “Bridge Destinations”

• Each Bridge Destination is configured in using the **four common properties**:
  • Initial Context Factory
  • Connection URL
  • Connection Factory JNDI Name
  • Destination JNDI Name
• Here’s an example of the config.xml:

```xml
<Application Name="jms-xa-adp"
    Path="D:\weblogic\server\lib"
    StagingMode="nostage" TwoPhase="true">
    <ConnectorComponent Name="jms-xa-adp" Targets="myserver" URI="jms-xa-adp.rar"/>
</Application>
<JMSBridgeDestination ConnectionFactoryJNDIName="XAQCF"
    ConnectionURL="file:/D:/JNDI/" DestinationJNDIName="Test1"
    InitialContextFactory="com.sun.fscontext.ReffFSContextFactory"
    Name="MQInput"/>
<JMSBridgeDestination
    ConnectionFactoryJNDIName="weblogic.jms.XAConnectionFactory"
    DestinationJNDIName="Test2" Name="WLSOutput"/>
<MessagingBridge Name="MQToWLSBridge" SourceDestination="MQInput"
    TargetDestination="WLSOutput"/>
```
Using JMS In J2EE Components

- Inside an EJB or a servlet, use JMS by including `resource-ref` elements in the deployment descriptors

- For instance, inside ejb-jar.xml:

```xml
<resource-ref>
  <res-ref-name>jms/QCF</res-ref-name>
  <res-type>javax.jms.QueueConnectionFactory</res-type>
  <res-auth>Container</res-auth>
  <res-sharing-scope>Shareable</res-sharing-scope>
</resource-ref>

<resource-env-ref>
  <resource-env-ref-name>jms/MYQUEUE</resource-env-ref-name>
  <resource-env-ref-type>javax.jms.Queue</resource-env-ref-type>
</resource-env-ref>
```
Using the `resource-ref`

- And in `weblogic-ejb-jar.xml`:

  ```xml
  <reference-descriptor>
    <resource-description>
      <res-ref-name>.jms/QCF</res-ref-name>
      <jndi-name>JmsFactories.CF</jndi-name>
    </resource-description>
  </reference-descriptor>
  <resource-env-description>
    <res-env-ref-name>.jms/MYQUEUE</res-env-ref-name>
    <jndi-name>JmsQueues.FooQueue</jndi-name>
  </resource-env-description>
  ```
Using the resource-ref

• In this example, we have:
  • Created two JNDI entries that may be looked up inside the EJB or servlet:
    – java:comp/env/jms/QCF
    – java:comp/env/jms/MYQUEUE
  • When looked up, these will actually return:
    – JmsFactories.CF
    – JmsQueues.FooQueue
resource-ref Example

WLS Server “MyServer”

EJB “MySessionBean”

- **resource-ref** “jms/QCF”
  JNDI Name: java:comp/env/jms/QCF

- **resource-env-ref** “jms/MYQUEUE”
  JNDI Name: java:comp/env/jms/MYQUEUE

JMS Connection Factory “CF”

JMS Destination “FooQueue”
• Then, you can send a JMS message like this:

```java
public void sendAMessage() {
    InitialContext c = new InitialContext();
    Queue q = c.lookup("java.comp/env/jms/MYQUEUE");
    QueueConnectionFactory qcf = c.lookup("java:comp/env/jms/QCF");
    c.close();
    QueueConnection conn = qcf.createQueueConnection();
    try {
        QueueSession session = conn.createQueueSession(false, 0);
        QueueSender sender = session.createQueueSender(q);
        TextMessage msg = session.createTextMessage("Hello, World!");
        sender.send(msg);
    } finally {
        conn.close();
    }
}
```
Why use a resource-ref?

• Why should you use a resource-ref?
  • Ensures application portability
  • Change JMS objects by changing descriptors
    – No re-compilation required

• New features in WebLogic Server 8.1:
  • Automatic pooling of Connection, Session, and MessageProducer objects
  • Automatic re-connection after failure
  • Automatic transaction enlistment
J2EE Components & Transactions

- If a JMS resource defined using a `resource-ref` is used inside a JTA transaction, it becomes part of the transaction
  - A new feature in WebLogic Server 8.1
  - Requires XA support in the JMS provider
  - This does not happen in 7.0 and earlier
Using JMS In J2EE Components

• **But**, a `resource-ref` just lets you bind to a JNDI name
  • There is no place to specify the “initial context factory” and URL

• In BEA WebLogic Server 7.0 and before, this means you can’t use it with a foreign JMS provider

• In BEA WebLogic Server 8.1, you can, using a **Foreign JMS Server** definition
Foreign JMS Server Definitions

• A Foreign JMS Server Definition makes a sort of “symbolic link” between:
  • A JNDI object in another JNDI directory, like a JMS connection factory or destination object
  • A JNDI name in the JNDI name space for your WebLogic Server cluster

• You can set these up:
  • In the console, under “JMS”
  • In config.xml
Here’s an example of what goes in config.xml:

```xml
<ForeignJMSServer ConnectionURL="file:/D:/JNDI/"
    InitialContextFactory="com.sun.fscontext.ReffFSContextFactory"
    JNDIProperties="" Name="MQ" Targets="myserver">
    <ForeignJMSConnectionFactory LocalJNDIName="MQObjects.XAQCF"
        Name="XAQCF" RemoteJNDIName="XAQCF"/>
    <ForeignJMSDestination LocalJNDIName="MQObjects.TEST_QUEUE_1"
        Name="MQTestQ" RemoteJNDIName="TEST_QUEUE_1"/>
</ForeignJMSServer>
```
• By using a Foreign JMS Server definition in 8.1, you move all those JNDI parameters into one place

• You can share one definition between:
  • EJBs
  • servlets
  • Messaging Bridges

• You can change them without recompiling or changing deployment descriptors
Foreign JMS Provider Example

WLS Server “MyServer”

EJB “MySessionBean”

resource-ref “jms/QCF”
JNDI Name: java:comp/env/jms/QCF

resource-env-ref “jms/MYQUEUE”
JNDI Name: java:comp/env/jms/MYQUEUE

Foreign JMS Factory “LocalCF”
Foreign JMS Dest. “LocalQ”

WLS Server “YourServer”

JMS Connection Factory “CF”

JMS Destination “FooQueue”
There are four JNDI-related parameters you need to configure to use WebLogic Server with a foreign JMS provider:

- The “initial context factory” name
- The URL of the JNDI provider
- The name of the JMS connection factory object
- The name of the JMS destination object

These can be used in various places:

- Message-driven beans
- The Messaging Bridge (WLS 7.0 and up)
- Foreign JMS Server definitions (WLS 8.1)
Presentation Summary

• To maximize the flexibility of your J2EE applications:
  • Use *resource-ref* elements to move JMS names out of the source code and in to the deployment descriptors
  • Use Foreign JMS Server definitions to move the names yet further out of the descriptors and in to config.xml
  • This lets you change things at runtime
Q&A
Thank You!