Lecture 11
S4 - Core Distributed Middleware Programming in JEE

presentation

DAD – Distributed Applications Development
Cristian Toma

D.I.C.E/D.E.I.C – Department of Economic Informatics & Cybernetics
www.dice.ase.ro
Agenda for Lecture 11

1. JMS – Java Message Service
2. JMS DEMO
3. Exchange Ideas
Java Message Service Overview

DAD Section 4 – JMS Architecture, MOM – Message-Oriented Middleware, JMS Messaging Models, JMS API, Synchronous & Asynchronous Queues, Durable and Non-Durable Topics
1. Middleware & JMS Technology Overview

Middleware Technologies:

Middleware can be grouped into the following categories:

- **Remote Procedure Call or RPC**-based middleware, which allows procedures in one application to call procedures in remote applications as if they were local calls. The middleware implements a linking mechanism that locates remote procedures and makes these transparently available to a caller. Traditionally, this type of middleware handled procedure-based programs; it now also includes object-based components.

- **Object Request Broker or ORB**-based middleware, which enables an application’s objects to be distributed and shared across heterogeneous networks.

- **Message Oriented Middleware or MOM**-based middleware, which allows distributed applications to communicate and exchange data by sending and receiving messages.

http://docs.oracle.com/cd/E19717-01/819-7759/index.html
1. Middleware & JMS Technology Overview

MOM & RPC Combined Systems:

http://docs.oracle.com/cd/E19717-01/819-7759/index.html
Clients A and B are message producers, sending messages to clients C, D, and E by way of two different kinds of destinations.

- Messaging between clients A, C, and D illustrates the point-to-point pattern. Using this pattern, a client sends a message to a queue destination from which only one receiver may get it. No other receiver accessing that destination can get that message.

- Messaging between clients B, E, and F illustrates the publish/subscribe pattern. Using this broadcast pattern, a client sends a message to a topic destination from which any number of consuming subscribers can retrieve it. Each subscriber gets its own copy of the message.

Message consumers in either domain can choose to get messages synchronously or asynchronously. Synchronous consumers make an explicit call to retrieve a message; asynchronous consumers specify a callback method that is invoked to pass a pending message. Consumers can also filter out messages by specifying selection criteria for incoming messages.
1. Middleware & JMS Technology Overview

JMS Messaging Patterns – Queue vs. Topics:

http://docs.oracle.com/cd/E19717-01/819-7759/index.html
1. JMS Technology Overview

JMS – Messaging Domains: **Point-to-Point (Queue Based) Messaging System**

There may be multiple Senders of messages, a best practice is to have a single Receiver for the messages.

1. JMS Technology Overview

JMS – Messaging Domains: Publish-Subscribe Based Messaging System

When multiple applications need to receive the same messages; there may be multiple Senders and multiple Receivers.

1. JMS Technology Overview

JMS – Messaging Domains: *Request-Reply Messaging System*

This is the standard synchronous object-messaging format. This messaging model is often defined as a subset of one of the other two models. JMS does not explicitly support Request-Reply Messaging, though it allows it in the context of the other methods.

To set up the request-reply pattern you need to do the following:
1. Create a temporary destination where the consumer can send replies.
2. In the message to be sent, set the JMSReplyTo field of the message header to that temporary destination.

When the message consumer processes the message, it examines the JMSReplyTo field of the message to determine if a reply is required and sends the reply to the specified destination.

The request-reply mechanism saves the producer the trouble of setting up an administered object for the reply destination and makes it easy for the consumer to respond to the request. This pattern is useful when the producer must be sure that a request has been handled before proceeding.

As the figure shows, MyTopicPublisher produces Msg1 to the destination MyTopic. MyTopicSubscriber1 and MyTopicSubscriber2 receive the message and send a reply to MyTempQueue, from where MyTQReceiver retrieves it. This pattern might be useful for an application that published pricing information to a large number of clients and which queued their (reply) orders for sequential processing.
1. **JMS Technology Overview**

**JMS Architecture**

JMS Service Providers implement the JMS interface on top of their messaging services. JMS defines Queues and Topics, but it does not require the provider to implement both.

The primary features of JMS are as follows:

- Connection Factories are used in JMS to create connections to a specific JMS provider.
- In JMS, both Publish-Subscribe Messaging and Point-To-Point are implemented and defined by separate interfaces so that a Provider does not have to support both.
- JMS defines the concept of a Topic or a Queue as the target for a Message. Topics are used for Publish-Subscribe Messaging. Queues are used for Point-to-Point Messaging.
- The Providers’ code is defined by interfaces in JMS, freeing the implementation from the limitations of sub-classing.
- JMS provides support for distributed transactions.

---

1. JMS Technology Overview

JMS Basic Elements of an application

http://docs.oracle.com/cd/E19717-01/819-7759/index.html
1. JMS Technology Overview

JMS Basic Elements of an application

To Use Administered Objects as Destinations:

1. The administrator creates a physical destination on the broker.
2. The administrator creates a destination administered object and configures it by specifying the name of the physical destination to which it corresponds and its type: queue or topic.
3. The message producer looks up the destination administered object using a JNDI lookup call.
4. The message producer sends a message to the destination.
5. The message consumer looks up the destination administered object where it expects to get messages.
6. The message consumer gets the message from the destination.

The process of using connection factory administered objects is similar. The administrator creates and configures a connection factory administered object using administration tools. The client looks up the connection factory object and uses it to create a connection.

Although the use of administered objects adds a couple of steps to the messaging process, it also adds robustness and portability to messaging applications.
1. JMS Technology Overview

JMS API

1. JMS Technology Overview

JMS Message Consumption Types

Sun Message Consumption Types:

“Messaging products are inherently asynchronous in that no fundamental timing dependency exists between the production and the consumption of a message. However, the JMS Specification uses this term in a more precise sense. Messages can be consumed in either of two ways:

* Synchronously. A subscriber or a receiver explicitly fetches the message from the destination by calling the receive method. The receive method can block until a message arrives or can time out if a message does not arrive within a specified time limit.

* Asynchronously. A client can register a message listener with a consumer. A message listener is similar to an event listener. Whenever a message arrives at the destination, the JMS provider delivers the message by calling the listener's onMessage method, which acts on the contents of the message.”
1. JMS Technology Overview

JMS Durable versus Non-Durable Topic Subscription

http://docs.oracle.com/cd/E19717-01/819-7759/index.html
Fact: DAD middleware is based on MOM

In few samples it is simple to remember: MOM – Message-Oriented Middleware implemented using JMS – Java Message Service for:

- Synchronous and Asynchronous Queues
- Non-durable and Durable Topics
JMS - Java Message Service DEMO
2. JMS DEMO


2. JMS DEMO

JMS DEMO – JBOSS 4 Directories Distribution
JMS DEMO – JBOSS 5 / 6 AS Directories Distribution

+ bin/ - contains start scripts and run.jar
+ client/ - client jars
+ docs/ - docs, schemas/dtds, examples
+ lib/ - core bootstrap jars, different with the introduction of the micro-container and breakup of jboss-common.
+ server/ - contains the same server configuration directories.

**+ default/ configuration**

+ conf/ - contains server configuration files used when starting the server.
  
  changes in here are detected on restarting your server.
  
  # bootstrap-beans.xml - new mc kernel bootstrap configuration
  # jax-ws-catalog.xml - oasis catalog driven schema/dtd namespace conf.
  # jbossjta-properties.xml - new JBossTS properties
  # jboss-service.xml - legacy static mbeans for compatibility
  # jndi.properties - the same jndi props
  # log4j.xml - the same log4j config
  # login-config.xml - the same jaas login config
  # props/ - the same default jaas login properties files
  # standardjaws.xml - obsolete cmp config
  # standardjbosscmp-jdbc.xml - the same cmp2 config
  # standardjboss.xml - the same ejb2 config
  # xmdesc/ - legacy xmbean descriptors
### 2. JMS DEMO

**JMS DEMO – JBOSS 5 / 6 AS Directories Distribution**

<table>
<thead>
<tr>
<th>Directory</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ default/configuration</td>
<td>+ conf/ - contains server configuration files used when starting the server. Changes in here are detected on restarting your server.</td>
</tr>
<tr>
<td></td>
<td>+ data/ - contains hypersonic local database, transactions, xml-bean conf. files.</td>
</tr>
<tr>
<td></td>
<td>+ deploy/ - this is where services and your java applications are deployed. You can deploy an application on the JBoss application server by simply copying the application's (WAR, EAR or JAR files) into this directory.</td>
</tr>
<tr>
<td></td>
<td>+ deployers/ - new vdf deployers</td>
</tr>
<tr>
<td></td>
<td># bsh-deployer - beanshell deployer</td>
</tr>
<tr>
<td></td>
<td># ejb3.deployer - ejb3 deployers</td>
</tr>
<tr>
<td></td>
<td># jboss-aop-jboss5.deployer - aspect deployer</td>
</tr>
<tr>
<td></td>
<td># jboss-jca.deployer - JCA deployers</td>
</tr>
<tr>
<td></td>
<td># jbossweb.deployer - war deployers</td>
</tr>
<tr>
<td></td>
<td># jbossws.deployer - web services deployers</td>
</tr>
<tr>
<td></td>
<td># ear-deployer-beans.xml - ear deployers</td>
</tr>
<tr>
<td></td>
<td># ejb-deployer-beans.xml - ejb2.x deployers</td>
</tr>
<tr>
<td></td>
<td># metadata-beans.xml - metadata handlers</td>
</tr>
<tr>
<td></td>
<td># security-deployer-beans.xml - security deployers</td>
</tr>
<tr>
<td></td>
<td># profileservice-beans.xml.bak - an example of the repository based profile service</td>
</tr>
<tr>
<td></td>
<td>+ lib/ - the JBoss AS static library files shared by the services and app. in the respective conf.</td>
</tr>
</tbody>
</table>
# 2. JMS DEMO

## JBOSS 5/6 versus 7 Versions

<table>
<thead>
<tr>
<th>Java EE Platform Technology</th>
<th>Java Enterprise Edition 6</th>
<th>JBoss Application Server 7.0.0.Final</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Full Profile</td>
<td>Web Profile</td>
</tr>
<tr>
<td><strong>Web Application Technologies</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>JSR 315: Java Servlet 3.0</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>JSR 314: JavaServer Faces(JSF) 2.0</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>JSR 245: JavaServer Pages 2.2 and Expression Language (EL 1.2)</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>JSR-52: A Standard Tag Library for JavaServer Pages 1.2</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>JSR-45: Debugging Support for Other Languages 1.0</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td><strong>Enterprise Application Technologies</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>JSR 299: Contexts and Dependency Injection for the Java EE Platform 1.0</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>JSR 330: Dependency Injection for Java</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>JSR 318: Enterprise JavaBeans 3.1</td>
<td>X</td>
<td>X (Lite)</td>
</tr>
<tr>
<td>JSR 317: Java Persistence API 2.0</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>JSR 250: Common Annotations for the Java Platform 1.1</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>JSR 907: Java Transaction API (JTA) 1.1</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>JSR 303: Bean Validation 1.0</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>JSR 322: Java EE Connector Architecture 1.6</td>
<td>X</td>
<td>--</td>
</tr>
<tr>
<td>JSR 914: Java Message Service (JMS) API 1.1</td>
<td>X</td>
<td>--</td>
</tr>
<tr>
<td>JSR 919: JavaMail 1.4</td>
<td>X</td>
<td>--</td>
</tr>
<tr>
<td><strong>Web Services Technologies</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>JSR 311: JAX-RS: The Java API for RESTful Web Services 1.1</td>
<td>X</td>
<td>--</td>
</tr>
<tr>
<td>JSR 109: Implementing Enterprise Web Services 1.3</td>
<td>X</td>
<td>--</td>
</tr>
<tr>
<td>JSR 224: Java API for XML-Based Web Services (JAX-WS) 2.2</td>
<td>X</td>
<td>--</td>
</tr>
<tr>
<td>JSR 181: Web Services Metadata for the Java Platform</td>
<td>X</td>
<td>--</td>
</tr>
<tr>
<td>JSR 101: Java APIs for XML based RPC 1.1</td>
<td>X</td>
<td>--</td>
</tr>
<tr>
<td>JSR 93: Java API for XML Registries 1.0 (JAXR) 1.0*</td>
<td>X</td>
<td>--</td>
</tr>
<tr>
<td><strong>Management and Security Technologies</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>JSR 196: Java Authentication Service Provider Interface for Containers 1.0</td>
<td>X</td>
<td>--</td>
</tr>
<tr>
<td>JSR 115: Java Authorization Contract for Containers 1.4</td>
<td>X</td>
<td>--</td>
</tr>
<tr>
<td>JSR 77: J2EE Management 1.1*</td>
<td>X</td>
<td>--</td>
</tr>
<tr>
<td>JSR 86: Java EE Application Deployment 1.2*</td>
<td>X</td>
<td>--</td>
</tr>
</tbody>
</table>
2. JMS DEMO

JMS DEMO – JBOSS Clustering Architecture

2. JMS DEMO

JMS DEMO – JBOSS Clustering Architecture

Section Conclusions

JMS – Java Message Service is JEE implementation for MOM – Message Oriented Middleware

The chosen Java Web & Application Server with JMS implementation for testing is JBOSS

JBOSS architecture and directories structure

JBOSS compile, deploy and testing procedures for JMS samples

JBOSS 5 is using the “unified name service” accessed through JNDI with jndi.properties file/properties

JBOSS 5 samples for synchronous and asynchronous queues and durable/non-durable topics

JBOSS JMS could be “combined” with EJB-MDB (Message Driven Bean), RPC/RMI (Remote Method Invocation), Web-JSP-Bean, etc.
Communicate & Exchange Ideas

Distributed Application Development
Questions & Answers!

But wait...
There’s More!
Thanks!

DAD – Distributed Application Development
End of Lecture 11