Unit3: The WebLogic EJB Deployer

By
C.Narendra

The manual modification of the deployment descriptor in a text editor .The type of deployment descriptor you produce depends on whether you are creating a standard EJB JAR file or a deployable EJB JAR file.

Step1: Creating a Standard EJB JAR File

If you are creating a standard EJB JAR file, you only need to modify the single deployment descriptor contained in the EJB JAR file produced by the Bean Provider. When modifying this deployment descriptor, use the syntax described in the EJB 1.1 Specification produced by Sun Microsystems, Inc.

Step 2:Creating a Deployable EJB JAR File

If you are creating a deployable EJB JAR file, you need to do the following:
Modify the deployment descriptor, as described in the section Creating a Standard EJB JAR File.
Create a WebLogic Enterprise extended deployment descriptor file, which is also created as an XML file, and which specifies the information described in the section Step 3: Create the WebLogic EJB extensions to the deployment descriptor DTD..

Step 3: Create the WebLogic EJB extensions to the deployment descriptor DTD.

For an EJB application to be deployable in the WLE environment, you need to create a file containing the WebLogic EJB extensions to the deployment descriptor

DTD. This file specifies the following run time and configuration information for the EJB application:

1. Custom application startup and shutdown properties
2. Registration of the application's home interfaces
3. Persistence information

Specifying the WebLogic EJB Extensions DTD

The file that includes the WebLogic EJB extensions to the deployment descriptor DTD must specify the following DTD reference at the beginning:

<!DOCTYPE weblogic-ejb-extensions SYSTEM "weblogic-ejb-extensions.dtd" >

Specifying the Module Initializer Class

Server application startup and shutdown can be handled automatically by the EJB container. However, if you have special server initialization or shutdown requirements, use the module-initializer-class-name element in the WebLogic EJB extensions to the deployment descriptor DTD to specify the name of the module initializer object.

The syntax for specifying a module initializer object for handling the initialize and release methods is similar to that for CORBA and RMI. However, with the CORBA and RMI models, the startup and shutdown information is specified in an XML file and is serialized by the buildjavaserver command.

For EJB deployment, XML elements for startup and shutdown procedures are specified together with the other elements in the WebLogic EJB extensions to the deployment descriptor DTD, and you process them using the ejbc command.

The WLE EJB container parses the XML at run time and performs the startup and shutdown processing. See the section Step 5: Configure the EJB application. for a complete description of startup and shutdown handling in WLE.

XML DTD Syntax

<!ELEMENT module-initializer-class (module-initializer-class-name*))>
<!ELEMENT module-initializer-class-name (#PCDATA)>
Example

```
<weblogic-ejb-extensions>.                                                    .
        <module-initializer-class>
                <moduleinitializer-class-name>ServerImpl
                </moduleinitializer-class-name>
        </module-initializer-class>
</weblogic-ejb-extensions>
```

Registering Names for the EJB Home Classes

A name for the EJB home class must be registered in the global WLE JNDI namespace. This allows Java clients to perform a lookup on the JNDI name for the EJB home and gain access to the object.

The name for the EJB home class can be different than the <ejb-name> element specified in the standard EJB XML. The <ejb-name> in the standard deployment descriptor must be unique only among the names of the EJBs in the same EJB JAR file.

However, the JNDI name must be unique among all global home or factory names in a WLE domain; this includes EJB homes, CORBA factories, and RMI named objects.

XML DTD Syntax

<!ELEMENT jndi-name (#PCDATA)>
Example

```
<weblogic-ejb-extensions>
  <weblogic-version>
    WebLogic Enterprise Server 5.0
  </weblogic-version>
  <weblogic-enterprise-bean>
    <ejb-name>
      oracle
    </ejb-name>
    <weblogic-deployment-params>

      <max-beans-in-free-pool>
        20
      </max-beans-in-free-pool>
      <max-beans-in-cache>
        1000
      </max-beans-in-cache>
      <idle-timeout-seconds>
        5
      </idle-timeout-seconds>
      <!-- JNDI name that is associated with this EJB;
           used for lookup -->
      <jndi-name>
        oracle.OracleHome
      </jndi-name>
              .
              .
              .
</weblogic-ejb-extensions>
```

Specifying Persistence Information

The WLE EJB container provides container-managed persistence. The code for implementing the persistence is generated by the ejbc command based on the deployment descriptors.

The persistence store can be a flat file or it can be a database managed with a JDBC connection pool. For the EJB state to fully cooperate in a WLE global transaction, configure the EJB to use the JDBC-managed database store provided in WLE. Use file-based persistence only during development and prototyping.

The standard deployment descriptor created by the Bean Provider normally specifies:

The fields in the EJB that are to be persistent, via the cmp-field element

Information about the primary key

However, you, as the deployer, need to specify additional information for mapping an EJB to its persistent store via the WebLogic EJB extensions to the deployment descriptor DTD.

File-based Persistence

The following code shows the WebLogic EJB extensions to the deployment descriptor DTD for specifying file-based persistence:

```
<!-- Persistence store descriptor. Specifies what type of persistence store
EJB container should use to store state of bean. -->

<!ELEMENT persistence-store-descriptor (description?,(persistence-store-file |
persistence-store-jdbc)?)>

<!--Persistence store using file. Bean is serialized to a file.Mainly used to store
state of Stateful Session Beans.-->

<!ELEMENT persistence-store-file (description?,persistence-store-directory-root?)>

<!--Root directory on File system for storing files per bean.-->

<!ELEMENT persistence-store-directory-root (#PCDATA)>
```

The information supplied for the persistence-store-directory-root element is used by the EJB container to store all instances of the EJB, with the ejb-name element converted to a directory name.

Database-stored Persistence

The following code shows the WebLogic EJB extensions to the deployment descriptor DTD for specifying a JDBC connection for database-stored persistence:

```
<!-- Persistence store is any RDBMS. JDBC driver is used to talk to database.
Required for CMP.-->
```

```
<!ELEMENT persistence-store-jdbc (description?, pool-name, user?, password?,
driver-url?, driver-class-name?, table-name, attribute-map,finder-descriptor*)>

<!-- Required for CMP -->
<!ELEMENT pool-name (#PCDATA)>

<!-- Ignored in WebLogic Enterprise Server as this is part of connection pool
 setup at startup -->
<!ELEMENT user (#PCDATA)>

<!-- Ignored in WebLogic Enterprise Server as this is part of connection pool
setup at startup -->

<!ELEMENT password (#PCDATA)>

<!-- Ignored in WebLogic Enterprise Server as this is part of connection pool
setup at startup -->

<!ELEMENT driver-url (#PCDATA)>

<!-- Ignored in WebLogic Enterprise Server as this is part of connection
poolsetup at startup -->

<!ELEMENT driver-class-name (#PCDATA)>

<!-- Required for CMP -->
<!ELEMENT table-name (#PCDATA)>

<!-- Required for CMP -->
<!ELEMENT attribute-map (description?, attribute-map-entry+)>

<!-- Required for CMP -->
<!ELEMENT attribute-map-entry (bean-field-name, table-column-name)>

<!-- Required for CMP -->
<!ELEMENT bean-field-name (#PCDATA)>

<!-- Required for CMP -->
<!ELEMENT table-column-name (#PCDATA)>

<!-- Required for CMP -->
<!ELEMENT finder-descriptor (description?, method?, query-grammar?)>
```

```
<!-- Required for CMP -->
<!ELEMENT query-grammar (#PCDATA)>
```

The EJB instances are stored in a database that has been previously set up with a JDBC connection pool, which is identified by the pool-name element. The table-name and attribute-map elements map the EJB fields to the appropriate table columns in the database.

Finder descriptors are the WLE implementation of the EJB find methods.

The finder-descriptor elements are pairs of method signatures and expressions. You specify a method signature in the EJBHome interface, and you specify the method's expression in the deployment descriptor via the query-grammar element.

The finder methods return an enumeration of EJBs. For more information about specifying finder descriptors, see the *EJB XML Reference* in the WebLogic Enterprise online documentation.

Example

The following WebLogic EJB extensions to the deployment descriptor DTD fragment specify automatic saving using JDBC persistence of two fields of an entity bean (accountId and salary) to a database table (emp) using a connection pool (pool1),

shown in boldface type:

```
<weblogic-ejb-extensions>
  <weblogic-version>
    WebLogic Enterprise Server 5.0
  </weblogic-version>
  <weblogic-enterprise-bean>
    <ejb-name>
      oracle
    </ejb-name>
    <weblogic-deployment-params>

      <max-beans-in-free-pool>
        20
```

```
    </max-beans-in-free-pool>
    <max-beans-in-cache>
      1000
    </max-beans-in-cache>
    <idle-timeout-seconds>
      5
    </idle-timeout-seconds>
    <!-- JNDI name that is associated with this EJB;used for lookup -->
    <jndi-name>
      oracle.OracleHome
    </jndi-name>

    <!-- This is CMP EJB. Specify persistence information -->
    <persistence-store-descriptor>
      <persistence-store-jdbc>
        <!-- Pool name is looked up by the EJB source -->
        <pool-name>
          jdbc/pool1
        </pool-name>
      <!-- *** DATABASE INFORMATION SPECIFIC TO INSTALLATION SITE ***
-->
        <!-- Default user URL is for Oracle 8i-->
        <user>
          scott
        </user>

        <!-- Default password URL is for Oracle 8i-->
        <password>
          tiger
        </password>
        <!-- Default driver URL is for Oracle 8i,and default instance Beq-Local is
used  -->
        <driver-url>
          jdbc:weblogic:oracle:Beq-Local
        </driver-url>
        <table-name>
          emp
        </table-name>

        <!-- CMP Fields and database table column mapping-->
        <attribute-map>
          <attribute-map-entry>
            <bean-field-name>
```

```
            accountId
          </bean-field-name>
          <table-column-name>
            empno
          </table-column-name>
        </attribute-map-entry>
        <attribute-map-entry>
          <bean-field-name>
            salary
          </bean-field-name>
          <table-column-name>
            sal
          </table-column-name>
        </attribute-map-entry>
      </attribute-map>

      <!-- Finder Specifications -->
      <finder-descriptor>
        <method>
          <ejb-name>
          oracle
          </ejb-name>
          <method-name>
            findAccount(double salaryEqual)
          </method-name>
        </method>
        <query-grammar>
          (= salary $salaryEqual)
        </query-grammar>
      </finder-descriptor>
    </persistence-store-jdbc>
  </persistence-store-descriptor>
 </weblogic-deployment-params>
</weblogic-enterprise-bean>
</weblogic-ejb-extensions>
```

For more information about JDBC connections, see *Using the JDBC Drivers* in the WebLogic Enterprise online documentation.

Step 4: Produce the deployable EJB JAR file.

In this step, you package the deployment descriptor, the compiled files for the EJB classes, the WLE extensions to the deployment descriptor DTD, and any

additional required classes into a deployable EJB JAR file. You can do this using the ejbc command, as in the following **example:**

java com.beasys.ejb.utils.ejbc -validate -i DDfile -x WLEXfile archive-file

The ejbc command performs the following steps:

Parses the standard EJB deployment descriptor and WebLogic Enterprise extended deployment descriptor XML files, which are represented, respectively, as DDfile and WLEXfile in the preceding command.

Checks the deployment descriptors for semantic consistency, and writes any inconsistencies to standard output.

Generates the wrapper Java classes and compiles them. This is performed for each EJB in the deployment descriptor.

Packages the XML deployment descriptors and the generated class files into a deployable EJB JAR file. The command-line argument archive-files specifies the files that are archived into the EJB JAR file.

If you have multiple bean packages meant to be assembled as a deployable unit, the bean packages must be specified in a single deployment descriptor. For more information about the ejbc command, see the *Command Reference* in the WebLogic Enterprise online documentation.

Step 5: Configure the EJB application.

In the SERVERS section of the UBBCONFIG file, the administrator uses the MODULES keyword to identify the deployed EJB JAR files.

Optionally, a related set of startup arguments can be specified for each EJB JAR file.

For information about configuring the EJB container, configuring the WLE EJB server process, and specifying values in the UBBCONFIG file, see *Creating a Configuration File* in the WebLogic Enterprise online documentation.

Step 6: Specify the module initializer object in the WebLogic EJB extensions to the deployment descriptor DTD.

Server application startup and shutdown can be handled automatically by the EJB container. However, if you have special server initialization or shutdown requirements, you need to implement an application entity called the module initializer object.

The module initializer object implements operations that execute the following tasks:

Performing basic module initialization (or EJB JAR file deployment) operations, which may include allocating resources needed by the EJB JAR file.

Performing basic server application initialization operations, which may include registering homes or factories managed by the server application and allocating resources needed by the server application.

Performing server process shutdown and cleanup procedures when the server application has finished servicing requests.

Note: For EJBs, the scope of the module initializer object is at the EJB JAR file level and not of the entire server application, as with the Server object and WLE CORBA applications.

EJB Roles

An interesting feature of the *EJB Specification* is that its requirements are largely divided into different 'roles.' When reading the Specification, it is therefore quite clear to whom the various requirements correspond.

Component provider

The component provider [EJB2.0 25.1] is a software developer who authors EJBs. These EJBs may be destined for a particular application, or they may be

general-purpose or part of a software library. The EJB author may not know the names of the EJBs with which it is to interact although, of course, their method specifications must be known. Similarly, the EJB author may not know the names of external resources like databases.

The Specification indicates how the component provider uses the XML deployment descriptor to indicate that these EJBs and resources are required, and the names by which it has referenced them.

Application assembler

The application assembler [EJB2.0 25.2] is someone who builds an application out of component EJBs. The same person or people may be operating in this role as are fulfilling the component provider role, but the Specification is quite clear that the two are distinct jobs.

The application assembler resolves references between EJBs, unifies the references to external resources, and packages the components into a single file ready for deployment.

It is slightly confusing for the newcomer to EJB development that most of the graphical tools used for building EJB applications don't distinguish between what they are doing in the provider role versus in the assembler role.

It often appears that the developer is supplying redundant information or information that is duplicated; this is because the deployment descriptor being built does support the notion of a separate provider and assembler, even if they are in practice the same person.

Deployer

The deployer [EJB2.0 25.3] takes a file from the application assembler that contains the EJBs and other components required for the application and installs it on the server, making appropriate customizations for the site.

These customizations include replacing database references with real database information, mapping the security roles in the application onto real users and groups in the organization, and supplying values for other configuration parameters.

Server vendor

The server vendor supplies EJB server products. An EJB developer should not assume that the sections in the *EJB Specification* that deal with the responsibilities of the server vendor are not of interest. On the contrary: The server sets the operating environment of the EJB, and this section of the *EJB Specification* gives the developer an insight into what environment the EJB can expect.

Administrator

An administrator [EJB2.0 25.4] is a person responsible for the maintenance and performance tuning of an application.

Tool provider

A tool provider authors EJB packaging and deployment tools.

Pros and Cons of using EJB :

- The first option is likely to give better performance at runtime because invoking a Web service is slower than invoking an EJB.
- The first option can propagate context whereas a Web service invocation does not propagate context in the same way.
- The second option does not involve creating any custom code.
- The second option may not be possible for some EJB interface definitions, as generating an EJB service has limitations.
- See the Rational Application Developer documentation here: Limitations of Web services
- The second option may result in an interface change and hence a change to the SCA consumer.