Enterprise JavaBeans

Layer 05:

Deployment
Agenda

• Discuss the deployment descriptor including its structure and capabilities.

• Discuss JNDI as it pertains to EJB.
Example

- I'll continue to use the same example that I used before, specifically the **Book** bean.
JNDI is an acronym for **Java Naming and Directory Interface**.

- This is an API for accessing naming services such as:
  - COSNaming
  - LDAP
  - RMI Registry
Directories

• Conceptually, a directory service is similar to a file system. Rather than serving up files, directory services serve up objects.

• Each *entry* in a directory is associated with a *name*.

• We use JNDI to establish a uniform way of accessing the entries within directories.
Server Compliance

- To be compliant with the EJB 1.1 specification, each EJB server must provide JNDI access to its naming service.
  - Only lookup access is required; the vendor need not provide update capabilities.
  - Additional proprietary access mechanisms may be supplied, but must not be required.
The JNDI API is accessible from the `javax.naming` package.

We'll only use a small part of the API including:

- `InitialContext`
- `Context`
- `NamingException`
Connecting to a Directory

- As with databases, sockets, and other such objects, we first need to connect to the appropriate directory service.
- Each kind of naming service may have its own set of properties that can be used to configure this connection.
The **InitialContext** is used to provide a starting location in the directory hierarchy.

- The code on the following page demonstrates how to connect to the JBoss naming service.
1. Properties env = new Properties();
2. env.setProperty("java.naming.factory.initial",
   "org.jnp.interfaces.NamingContextFactory");
3. env.setProperty("java.naming.provider.url",
   "localhost:1099");
4. env.setProperty("java.naming.factory.url.pkgs",
   "org.jboss.naming");
5. InitialContext ic = new InitialContext( env );
To create an initial context, we provide information about the directory service we're using.

Other such information may be required by each individual vendor.

This data is provided as a **Hashtable** to the **InitialContext** constructor.
Deployment
Purpose

- **Deployment** is the process of making a server aware of some new component.
- This process is generally vendor-specific although the various high-level tasks to be performed are common to all implementations.
To deploy an EJB we need to complete the following steps:

1. Create a deployment descriptor.
2. Create a JAR file containing the necessary classes and the deployment descriptor.
3. Install the new bean into the server.
1. `<xml version="1.0" encoding="Cp1252"?>`

2. `<ejb-jar>`

3. `<description>`
   Contains all book-related EJBs.
   `</description>`

4. `<display-name>BookJAR</display-name>`

5. `<enterprise-beans>`

6. `<session>`

7. `<description>`
   Controls book processing.
   `</description>`
8.  <display-name>
    BookEJB
  </display-name>

9.  <ejb-name>
    BookEJB
  </ejb-name>

10.  <home>
    se554.layer05.ejb.BookHome
  </home>

11.  <remote>
    se554.layer05.ejb.Book
  </remote>
12.  <ejb-class>
    se554.layer05.ejb.BookBean
  </ejb-class>

13.  <session-type>
    Stateful
  </session-type>

14.  <transaction-type>
    Bean
  </transaction-type>

15.  </session>

16.  </enterprise-beans>

17.  </ejb-jar>
Environment
To make beans less brittle, we can take advantage of their environment.

Each bean class is given an area within the server's namespace by which they can access bean-specific data.

- Environment entries
- Bean references
- Resource factories
Beans often need to access configuration information. To avoid hard-coding this information in the bean, we use an environment entry. An environment entry is a value that's placed in the bean's namespace within the server's naming server.
Environment Entries
(2 of 2)

- To access an environment entry, an EJB will use JNDI.
- By default, all environment entries are placed under the context:
  \[ \text{java:comp/env} \]
- The bean provider needs to make sure that the entry is defined at deployment time.
Defining Environment Entries
(1 of 2)

- During the deployment process, the bean provider indicates that a bean requires an environment entry.
- They'll provide the entry's:
  - Name
  - Type
  - Value
Defining Environment Entries
(2 of 2)

1. <env-entry>
2.  <description>
    General description of the entry.
  </description>
3.  <env-entry-name>
    platypus
  </env-entry-name>
4.  <env-entry-type>
    java.lang.String
  </env-entry-type>
5.  <env-entry-value>
    some String value
  </env-entry-value>
6.   </env-entry>
Using Environment Entries

1. `Properties env = new Properties();`
2. `env.setProperty("java.naming.factory.initial", "org.jnp.interfaces.NamingContextFactory");`
3. `env.setProperty("java.naming.provider.url", "localhost:1099");`
4. `env.setProperty("java.naming.factory.url.pkgs", "org.jboss.naming");`
5. `InitialContext ic = new InitialContext(env);`
6. `Object obj = ic.lookup("java:comp/env/platypus");`
7. `String platypusString = (String)obj;`
Deployment Descriptor Entries

- There are other entries that we'll make within the deployment descriptor:
  - EJB references
  - Resource factories
Deployment
The actual deployment process is different for each EJB server, but they all share common characteristics:

- Create stubs and skeletons.
- Generate home and EJB objects.
- Install the home object into the naming service.
Summary

- Discuss the deployment descriptor including its structure and capabilities.
- Discuss JNDI as it pertains to EJB.
Next Steps

- Read chapter 10 in your text.
- The assignment for layer:05 is available on the course website.