Enterprise JavaBeans

Layer 06:

Client
Agenda

- Show the process used by the client code to access an EJB Home object from within JNDI.
- Demonstrate the use of the `PortableRemoteObject` class.
- Discuss the `ServiceLocator` pattern to help reduce the coupling between the client and the EJB server.
I'll continue to use the same example that I used before, specifically the Book bean.
Home Object
The client must first access the Home Object for the bean whose services it wishes to access.

This means that the client must know the JNDI name of that bean.

We'll look at a J2EE pattern to help reduce this coupling.
Packages

The client will need access to the following packages:

- java.rmi
- javax.ejb
- javax.naming
- java.rmi
- javax.ejb
- javax.naming

The client will also need access to the client JAR file created at deployment time.
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 );
1. Object obj = ic.lookup("BookEJB");
2. BookHome home =
   (BookHome)PortableRemoteObject.narrow(obj, BookHome.class);
3. Book book = home.create("EJBs"," 1-56592-869-5 ");
4. // invoke business methods, handling all
   // appropriate exceptions
Service Locator
Accessing a directory via JNDI is similar in terms of processing requirements to accessing a database via JDBC. Each client will access JNDI the same way and ask for the same set of home objects. This results in excessive and unnecessary network traffic as well as an increased footprint due to redundant objects.
Purpose

• Place the common JNDI access and bean construction logic into a separate class to hide the details of those processes.
• Use of the **ServiceLocator** pattern serves to reduce the number of JNDI accesses and allows for added features such as caching of the home object.

• This can result in better performance and reduced network traffic.
Implementation

• A *ServiceLocator* is often implemented as a Singleton. This ensures that we only require one instance per JVM.

• Each client will need to know which class will act as the *ServiceLocator*. This information could be provided within a configuration file.
Summary

- Show the process used by the client code to access an EJB Home object from within JNDI.
- Demonstrate the use of the *PortableRemoteObject* class.
- Discuss the *ServiceLocator* pattern to help reduce the coupling between the client and the EJB server.
Next Steps

- Layer:06 has been posted to the course website.
- Make reading assignment