enterprise java beans

layer:08

persistence
Agenda

- Discuss "finder" methods.
- Describe DataSource resources.
- Describe bean-managed persistence.
- Describe container-managed persistence.
Finder Methods
• Entity beans are typically used to refer to existing data.

• In order to retrieve that data, we construct finder methods that are responsible for locating and retrieving that data using an appropriate select statement.
Finder methods are declared by:

- Providing a method `find<method>` in the home interface.
- Declare a corresponding `ejbFind<method>` in the bean class.
- As with all other methods with these relationships, the arguments must match between the methods.
Home interface

- Each finder method has the following declaration in the home interface:
  - A name typically called `find<method>`.
  - A return type of the remote interface of the entity bean.
  - A `throws` clause including both `FinderException` and `RemoteException`. 
Bean Class

• Each finder method has the following definition in the bean class:
  – A name typically called `ejbFind<method>`.
  – A return type of the primary key type of the bean or a return type of `Collection` or `Enumeration`.
  – A `throws` clause including `FinderException`. 
Home Interface

1. `public interface BookHome
   extends EJBObjec {`

2.  `public Book findByPrimaryKey(BookKey key)
       throws FinderException, RemoteException;`

3.  `public Collection findByAuthor(String name)
       throws RemoteException;`

4.  `}`
1. public interface BookEJB implements EntityBean {

2.     public BookKey ejbFindByPrimaryKey(BookKey key)
            throws FinderException, RemoteException {

3.         }

4.     public Collection ejbFindByAuthor(String name)
            throws RemoteException {

5.         }

6.     }

Bean Class
Single-Row Finders
Finding (1 of 6)

- The client invokes the `find<method>` on the home interface.
The home object delegates the call to the `ejbFind<method>` of an arbitrary bean instance in the free pool.

The process does not activate the bean or make sure that the bean does whatever it needs to do in the method to prepare itself to access the database.
The bean instance performs an appropriate **select** against the underlying backing store.

The result of this **select** will be used to construct a primary key.
The result of the select is used to construct a primary key object.

This object is returned to the home object.
• The home object creates a new EJBObj ect and installs the key inside of it.
The home object return the EJBOBJECT to the client.
The client can now use that EJBOBJECT to perform business processing.
Multi-Row Finders

Client
2. find()
7. EJBOBJECT

Home
3. ejbFind()
5. return keys

EJBOBJECT (key)
6a. newInstance()
6b. newInstance()
4. select...

Database

Bean Instance

Free Pool
Finding
(1 of 6)

- The client invokes the `find<method>` on the home interface.
The home object delegates the call to the `ejbFind<method>` of an arbitrary bean instance in the free pool.

The process does not activate the bean or make sure that the bean does whatever it needs to do in the method to prepare itself to access the database.
The bean instance performs an appropriate `select` against the underlying backing store.

The result of this `select` will be used to construct a `Collection` or `Enumeration` of primary keys.
The result of the `select` is used to construct a `Collection` or `Enumeration` of primary key objects.

This `Collection` or `Enumeration` is returned to the home object.
Finding
(5 of 6)

- For each key within the Collection or Enumeration, the home object creates a new EJBObject and installs the key inside of it.
- This new EJBObject is then placed into a new Collection or Enumeration.
Finding
(6 of 6)

- The home object returns the Collection or Enumeration of EJBOBJECTS to the client.
- The client can now use those EJBOBJECTS to perform business processing.
Bean-Managed Persistence
In bean-managed persistence (BMP) the developer writes all of the code needed to access the database and perform SQL queries. This results in more code to maintain, but tighter control over what that code does.
For a bean to provide BMP it must provide the queries for each of the following methods:

- `ejbCreate()`
- `ejbFind<method>`
- `ejbLoad()`
- `ejbRemove()`
- `ejbStore()`
Under EJB 1.1, the only mechanism by which a bean may use JDBC to access a database is through a **DataSource**.

- A **DataSource** is similar to a **Driver** except that it provides more functionality:
  - connection pooling
  - distributed transactions
**DataSource**

(2 of 3)

- **DataSource** references are defined to the server in a server-specific way.
- This is sometimes part of deployment, but is more often part of the server configuration.
  - For instance, in JBoss we need to add information to the *jboss.jcml* configuration file.
- Our beans will use JNDI to locate this resource factory.
The **DataSource** is an interface within the `javax.sql` package.
Locating the DataSource
(1 of 2)

- The first step to using the DataSource is to use JNDI to locate its definition within the naming service.
- Once we've got the DataSource reference we can use it to create Connection references.
- We can then use the Connection to issue queries against the database.
Locating the DataSource

(2 of 2)

• Since the **DataSource** can be accessed and used throughout the life of the entity bean, it is reasonable to place the code for accessing the **DataSource** within the `setEntityContext()` method.
1. public void
   setEntityContext(EntityContext ctx) {
2.     this.context = context;

3.     try {
4.         // TODO: configure the properties
5.     }
6.     catch (Exception e) {
7.         throw new EJBException( e.getMessage() );
8.     }
try {
    this.dataSource = (DataSource)initialContext.lookup(dsName);
}

catch (Exception e) {
    throw new EJBException(e.getMessage());
}
1. public Integer
ejbCreate(int id, String name,
        int age, boolean available)
    throws CreateException {
 2. try {
 3.     // TODO: configure the bean's internal
 4.     // data. if any problems occurred
 5.     // with the validations, throw a
 6.     // CreateException.
 7. }
 8. catch (Exception e) {
 9.     throw new CreateException(e.getMessage());
10. }

11. Connection conn = null;
12. PreparedStatement stmt = null;

13. try {
14.    conn = dataSource.getConnection();
15.    stmt = conn.prepareStatement(EJB_CREATE);
16.   // TODO: set the appropriate parameters
17.    stmt.executeUpdate();
18. }
19.    catch (Exception e) {
20.        throw new EJBException( e.getMessage() );
21.    }
22.    finally {
23.        try {
24.            // TODO: close the statement and
25.            // connection.
26.        }
27.        catch (SQLException sqle) {
28.            }
29.     }
32. return new Integer( id );
33. }
1. public Integer
   ejbFindByPrimaryKey(Integer id)
   throws FinderException {
2.     Connection conn = null;
3.     PreparedStatement stmt = null;
4.     ResultSet rs = null;

5.     try {
6.         conn = dataSource.getConnection();
7.         stmt = conn.
8.             prepareStatement( EJB_FIND_BY_PK );
9.         stmt.setInt( 1, id.intValue() );
10.        rs = stmt.executeQuery();
if ( rs.next() ) {
    return id;
} catch (Exception e) {
    throw new EJBException( e.getMessage() );
}
19. try {
20.     // TODO: close the resultset, statement
21.     // and connection references.
22. }
23. catch (SQLException sqle) {
24. }
25. }
26. }
ejbLoad

1. public void ejbLoad() {
2.    Connection conn = null;
3.    PreparedStatement stmt = null;
4.    ResultSet rs = null;

5.    // TODO: use the database to load the
6.    // bean's internal state. use the
7.    // same techniques described in
8.    // ejbCreate() and
9.    // ejbFindByPrimaryKey()
10. }
ejbRemove

1. public void ejbRemove() {
2.    Connection conn = null;
3.    PreparedStatement stmt = null;
4.    // TODO: write the query to issue a
5.    // delete against the database.
6.    }

1. public void ejbStore() {
2.     Connection conn = null;
3.     PreparedStatement stmt = null;
4.     // TODO: write the query to issue an
5.     // update against the database.
6. }
Container-Managed Persistence
In container-managed persistence, the bean developer ceases to manage the database. Instead, the container generates a Data Access Object to interact with the database on behalf of the EJB. The cost of a container is often proportional to its CMP capabilities.
Bean Responsibilities

(1 of 4)

When using CMP the following rules need to be followed by the bean developer:

- The `ejbCreate()` method's job is to set the values of the bean fields. The container will retrieve these fields during the insert.
- The `ejbCreate()` methods must return `null`. It's the responsibility of the container to generate the primary key.
When using CMP the following rules need to be followed by the bean developer:

- The fields of the bean class must be **public**. It's the responsibility of the container to read and write their values.
- The fields of the primary key must be **public**. It's the container's responsibility to read and write these fields.
Bean Responsibilities
(3 of 4)

- When using CMP the following rules need to be followed by the bean developer:
  - The `ejbLoad()` is called as a notification that the data has been read from the database.
  - The `ejbStore()` is called as a notification that the data is about to be saved to the database.
Bean Responsibilities
(4 of 4)

- When using CMP the following rules need to be followed by the bean developer:
  - The `ejbRemove()` method can be empty. The container will take care of removing the data from the database.
Deployment
**Entity Beans**

- Like session beans, an entity bean's structure and key properties are recorded in its deployment descriptor.
- In addition to many of the properties required by session beans, entity beans also have information about their primary keys and any container-managed fields.
ejb-jar.xml

1. <entity>
2.   <display-name>RiderEJB</display-name>
3.   <ejb-name>RiderEJB</ejb-name>
4.   <home>server.RiderHome</home>
5.   <remote>server.Rider</remote>
6.   <ejb-class>server.RiderBean</ejb-class>
7.   <persistence-type>Bean</persistence-type>
8.   <prim-key-class>
      java.lang.Integer
   </prim-key-class>
9.   </entity>
Container-Managed Persistence

- Container-managed persistence is reflected in the deployment descriptor.
- The persistence-type tag will be given a value of Container.
- For each field to be persisted, there will be a `cmp-field` tag.
1.  <entity>
2.   <display-name>RiderEJB</display-name>
3.   <ejb-name>RiderEJB</ejb-name>
4.   <home>server.RiderHome</home>
5.   <remote>server.Rider</remote>
6.   <ejb-class>server.RiderBean</ejb-class>
7.   <persistence-type>
      Container
   </persistence-type>
8.   <prim-key-class>
     java.lang.Integer
   </prim-key-class>
9.   <primkey-field>riderID</primkey-field>
```
10.  <cmp-field>
11.   <field-name>riderId</field-name>
12.  </cmp-field>
13.  <cmp-field>
14.   <field-name>riderName</field-name>
15.  </cmp-field>
16.  <primkey-field>riderId</primkey-field>
```
Resource Factories

- Resource factories such as **DataSource** references are defined to the server, but reference using JNDI.
- The bean uses an internal name to refer to the actual JNDI name.
- This is called a *resource reference*.
1. `<resource-ref>`
2. `<res-ref-name>jdbc/se554</res-ref-name>`
3. `<res-type>javax.sql.DataSource</res-type>`
4. `<res-auth>Container</res-auth>`
5. `</resource-ref>`
EJB References

• Sometimes one bean needs to refer to another.
  – One J2EE pattern, *Session Façade*, is built around this precept.

• To allow the bean to reference one another, we need to declare an EJB reference within a bean's namespace.
1. `<ejb-ref>`
2. `<ejb-ref-name>ejb/horse</ejb-ref-name>`
3. `<ejb-ref-type>Entity</ejb-ref-type>`
4. `<home>server.HorseHome</home>`
5. `<remote>server.Horse</remote>`
6. `<ejb-link>HorseEJB</ejb-link>`
7. `</ejb-ref>`
Summary

- Discuss "finder" methods.
- Describe DataSource resources.
- Describe bean-managed persistence.
- Describe container-managed persistence.
- Describe how resource and EJB references are recorded in the deployment descriptor.
Next Steps

- Make reading assignment.
- Layer:08 homework will be available on the course website.