

Deployment of EJB

Middleware Technologies

Unit – 3

Roy Antony Arnold G
Lecturer
Panimalar Engineering College
Chennai, Tamilnadu, India

The Deployment Descriptor Class

- This is the base class used by both **SessionDescriptor** and the **EntityDescriptor** classes.
- This class is the main way in which information is communicated **from the EJB developer to the deployer and to the container** in which the bean will be deployed.
- The bean developer uses the “**setter**” methods of this class to initialize the various properties of this class, and the deployment environment uses the “**getter**” methods to read these values at deployment time.
- All currently available EJB tools provide GUI tools to allow the developer and/or deployer to generate DeploymentDescriptors and their associated classes via pointing and clicking.

DeploymentDescriptor Class

```
public class javax.ejb.deployment.DeploymentDescriptor extends
java.lang.Object implements java.io.Serializable {
    protected int versionNumber;
    public DeploymentDescriptor();
    public AccessControlEntry[] getAccessControlEntries();
    public AccessControlEntry getAccessControlEntries(int index);
    public Name getBeanHomeName();
    public ControlDescriptor[] getControlDescriptors();
    public ControlDescriptor getControlDescriptors(int index);
    public String getEnterpriseBeanClassName();
    public Properties getEnvironmentProperties();
    public String getHomeInterfaceClassName();
    public boolean getReentrant();
    public String getRemoteInterfaceClassName();
    public boolean isReentrant();
```

Contd...

```
    public void setAccessControlEntries(AccessControlEntry values[]);
    public void setAccessControlEntires(int index, AccessControlEntry value);
    public void setBeanHomeName(Name value);
    public void setControlDescriptors(ControlDescriptor value[]);
    public void setControlDescriptors(int index, ControlDescriptor value);
    public void setEnterpriseBeanClassName(String value);
    public void setEnvironmentProperties(Properties value);
    public void setHomeInterfaceClassName(String value);
    public void setReentrant(boolean value);
    public void setRemoteInterfaceClassName(String value);
}
```

Contd...

- The deployment descriptor inherits from `java.lang.Object` and that it is serializable.
- **Serialization** is important for deployment descriptors for shipping the final product and later for deployment of the bean.
- The EJB specification indicates that a container should provide support for versioning. *For example, if a new version of a bean is installed, an older version should also continue to run.*
- The **version number** provides a mechanism for discriminating among otherwise-identical beans.
- The `getBeanHomeName()` method returns a `javax.naming.Name` object representing the name that the container should make available for clients that wish to use the bean.
- The container is responsible for placing this name into the JNDI namespace. Clients then use the JNDI `InitialContext.lookup()` method, with this name as the argument to find the bean's home interface.

Contd...

- Environment properties carry the information such as the name of the JDBC driver and the name of the JDBC data source.
- Properties are useful for holding information that may be modified at deployment time. *(this resembles the *.ini files in windows)*
- The `getReentrant()` method, which returns a Boolean value indicating whether the bean is reentrant.
- There are few important points about reentrancy:
 - Only Entity beans can be reentrant; Session beans can never be reentrant.
 - Reentrancy introduces the possibility of several sorts of concurrency-related problems. **Think twice before using it. Then think again. Then find another way to do it.**

The AccessControlEntry Class

- An `AccessControlEntry` is another class in the `javax.ejb.deployment` package; the purpose of this class is to pair a given method in your bean with a list of identities.

```
public class javax.ejb.deployment.AccessControlEntry extends
java.lang.Object implements java.io.Serializable {
    public AccessControlEntry();
    public AccessControlEntry(Method method);
    public AccessControlEntry(Method method, Identity identities[]);
    public Identity[] getAllowedIdentities();
    public Identity getAllowedIdentities(int index);
    public Method getMethod();
    public void setAllowedIdentities(Identity values[]);
    public void setAllowedIdentities(int index, Identity value);
    public void setMethod(Method value);
}
```

Contd...

- Note that the class consists mainly of **getter** and **setter** methods (known as **accessor** and **mutator** methods) that allow you to associate a set of identities with a particular method.
- If you set up an access control entry for a given method, then only the accounts that correspond to the specified identities will be allowed to invoke that method.
- Typically, the Identity objects correspond to roles rather than to the identities of particular users.
- At deployment time, the **EJB deployer** establishes **ACL** that relate sets of actual accounts to the roles provided in the deployment descriptor.
- If a user does not map to one of these identities, a **SecurityException** will be thrown, when he attempts to execute the method.

The ControlDescriptor Class

```
public class javax.ejb.deployment.ControlDescriptor extends
    java.lang.Object implements java.io.Serializable {
    public final static int CLIENT_IDENTITY;
    public final static int SPECIFIED_IDENTITY;
    public final static int SYSTEM_IDENTITY;
    public final static int TRANSACTION_READ_COMMITTED;
    public final static int TRANSACTION_READ_UNCOMMITTED;
    public final static int TRANSACTION_REPEATABLE_READ;
    public final static int TRANSACTION_SERIALIZABLE;
    public final static int TX_BEAN_MANAGED;
    public final static int TX_MANDATORY;
    public final static int TX_NOT_SUPPORTED;
    public final static int TX_SUPPORTS;
    public final static int TX_REQUIRED;
    public final static int TX_REQUIRES_NEW;
```

Contd...

The ControlDescriptor has three concerns:

1. The method's transaction attribute
2. The method's isolation level
3. The method's "run-as" mode

```
public ControlDescriptor();
public ControlDescriptor(Method method);
public int getIsolationLevel();
public Method getMethod();
public Identity getRunAsIdentity();
public int getRunAsMode();
public int getTransactionAttribute();
public void setIsolationLevel(int value);
public void setMethod(Method value);
public void setRunAsIdentity(Identity value);
public void setRunAsMode(int value);
public void setTransactionAttribute(int value);
}
```

"Run-as" Modes

- "Run-as" modes specify the identity that the bean should have at runtime.
- It is important when you set up an access control scheme for your beans and need to call one bean from another.
- The run-as mode specifies the type of identity that the calling bean should pass to the called bean.
- Three run-as modes exist:
 - **CLIENT_IDENTITY**, which means that the bean will use the identity of the client as its own identity when it makes calls to other beans.
 - **SYSTEM_IDENTITY**, which means that the bean will use the identity of a "system" account when invoking other beans. Generally, the bean deployer must configure this account in the container.
 - **SPECIFIED_IDENTITY**, which means that the bean will use specific identity. This identity is found in the **RunAsIdentity** property, which is applicable only if the bean is using **SPECIFIED_IDENTITY**.

The SessionDescriptor Class

```
public class javax.ejb.deployment.SessionDescriptor extends
    javax.ejb.deployment.DeploymentDescriptor {
    public final static int STATEFUL_SESSION;
    public final static int STATELESS_SESSION;
    public SessionDescriptor();
    public int getSessionTimeout();
    public int getStateManagementType();
    public void setSessionTimeout(int value);
    public void setStateManagementType(int value);
}
```

The return value of the **SessionTimeout** property represents the number of seconds that a bean can be inactive before the container is allowed to destroy it.

The EntityDescriptor Class

```
public class javax.ejb.deployment.EntityDescriptor extends
    javax.ejb.deployment.DeploymentDescriptor() {
    public EntityDescriptor();
    public Field[] getContainerManagedFields();
    public Field getContainerManagedFields (int index);
    public String getPrimaryKeyClassname();
    public void setContainerManagedFields (Field values[]);
    public void setContainerManagedFields (int index, Field
    value);
    public void setPrimaryKeyClassName(String value);
}
```

Using Roles at Runtime

- **isCallerInRole()** can be used to allow a method to respond differently based on the role of the user who invokes it.
- When the method is invoked by a **normal user**, the method could display information about the user's account.
- When the method is invoked by a **super user**, it might display information about all accounts.
- Following code can be written:

```
Identity su = new MyIdentity("superuser");
if (ctx.isCallerInRole(su)) {
    // return a list of all accounts
}
else {
    // return only the caller's account info
}
```

The DeploymentDescriptor

- Setting up ACL is a two-part process:
 - When the bean is written, **AccessControlEntry** objects need to be established for the various categories of users that are allowed to access the methods.
 - At deployment time, these categories of users must be mapped to actual users.
- To take care of the first part of this process, add the following code to the deployment descriptor

```
(accessControlEntries DEFAULT [superusers myPeople]
myProtectedMethod [superusers]); end accessControlEntries
```
- Only members of the **superusers** group will be allowed to access these methods.
- If a client who is not a member of one of these groups attempts to invoke one of the bean's methods, a **SecurityException** will be thrown.
- When distributing your beans, you should **include a description of the roles supported**. This precaution will allow the deployer to make informed decisions about which users should be in which roles.

Setting up Access Control Lists

- In WebLogic, there are two ways of setting up ACLs:
 - You can set them up in the **weblogic.properties** file.
 - You can create an authentication database and set up your user identities there.
- Ex: for **weblogic.properties** file sets up the superusers & mypeople groups:

```
weblogic.password.superuser1=superpassword1
weblogic.password.superuser2=superpassword2
weblogic.security.group.superusers=superuser1, superuser2
weblogic.password.myUser=myPassword
weblogic.password.myUser2=myPassword2
weblogic.security.group.mypeople=myUser,myUser2
```
- To create the user id WebLogic, use the following syntax

```
weblogic.password.<userid>=<password>
```
- To create the group in WebLogic, use the following syntax

```
weblogic.security.group.<groupname>=<userid1>,...<useridn>
```

Container-Managed Finder Methods

- Finder methods for container-managed Entity beans are created at deployment time, and that the **EJB specification does not mandate any particular format** that the bean developer must use to describe the finder methods to the bean deployer.
- An SQL-like syntax is probably the least ambiguous way to describe these finder methods.
- A sample SQL statement to implement:
`select * from accounts where date_today – date_due > $argument`
- As a bean deployer, you should provide a full and complete description of exactly how to implement your finder methods, if you use container-managed entity beans.

Other Deployment Issues

- **Caching Issues**
 - The number of Entity beans to maintain in the pool
 - The number of instances of a session bean to allow before the container begins to passivate beans.
 - A time interval after which inactive beans will be passivated by the container*Consider implementation's documentation for more information on the configuration parameters available.*
- **Persistent Storage**
 - The container may require the deployer to create a particular database table to be used for persistent storage of Entity EJBs.
 - It also may require information about the mechanism employed to passivate stateful Session EJBs temporarily.
- **Properties**
 - If environment properties must be set for the bean, the deployer should examine the properties and ensure that they are set correctly.

Other Deployment Issues – Other Administrative Issues

- Don't Reboot
- Watch for Churning
- Adjust Timeout value of Session Bean Appropriately
- Monitor the Number of Aborted Transactions
- Watch for Security Exceptions
- Be Parsimonious with Permissions
- Be Parsimonious with Services
- Check for Default Accounts and Obsolete Accounts
- Know your System