Fundamentals of Java Enterprise Components

Session: 4

Application Resources

Objectives

- Describe the different types of resources
- Describe the different methods through which these resources are introduced in the application
- Describe Java Naming Directory Interface (JNDI)
- Describe the usage of dependency injection and resource injection
- Explain packaging of enterprise applications
- Explain packaging of Web applications
- Describe Context Dependency Injection (CDI) in Java EE 7

Resource Creation

Resources are those components of the application which can be accessed by the core application code.

Resources in an application can be entities such as databases, mail services, and so on.

Various resources of the application can be located in geographically distinct locations.

JNDI (Java Naming and Directory Interface) is used to locate and identify different components of the application.

Java Naming and Directory Interface (JNDI) 1-3

- Binds the application resource with a name.
- Binds the name of an object with an object or a reference of an object in the directory.
- Organizes the naming space of the application into a hierarchy.
- Defines an initial context which specifies where to look up for objects.

Java Naming and Directory Interface (JNDI) 2-3

Following code defines the initial context of an application:

```
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NamingException;
…………..
////In main function
Hashtable env = new
Hashtable();env.put(Context.INITIAL_CONTEXT_FACTORY,
"com.sun.jndi.fscontext.RefFSContextFactory");Context
c = new InitialContext(env);…………
```

Java Naming and Directory Interface (JNDI) 3-3

JNDI uses the initial context of the application, which is the location where the mapping of the object name and its reference is stored.

The objects of the application are stored relative to the initial context.

Initial context of the application is set through Context and InitialContext classes.

Following are the methods of the Context class which are used for naming services:

- bind() – which binds a string or name to the object
- lookup() – which locates the object based on a string parameter
- unbind() – which removes the binding between the name and the object

Databases as Resources in Applications

Databases are widely used resources in enterprise applications.

Most popular data model of the databases is relational model, where data is stored in tables.

Java provides APIs such as Java Data Objects (JDO) and Java Database Connectivity (JDBC) to access and manipulate databases.

JDBC access database through DataSource objects, where each object has various properties such as location, user credentials, protocol, and so on.

The DataSource object which is accessing the database has to be registered with the naming service such as JNDI.

The naming service registers a JNDI name to the data source and provides a reference to the corresponding object.

Binding a DataSource Object with Database 1-2

An instance of DataSource object is created with the respective database driver.

- `com.microsoft.sqlserver.BasicDataSource ds = new com.microsoft.sqlserver.BasicDataSource();`

The DataSource object ds is set with properties such as database server name, database name, and other similar properties specific to the application as follows:

- `ds.setServerName("Europa");`
- `ds.setDatabaseName("Student");`
- `ds.setDescription("Student information for university");`

Binding a DataSource Object with Database 2-2

This DataSource object is bound with a JNDI name using the following statements:

```
• Context c = new InitialContext();
• c.bind("jndiTestDB",ds);
```

The DataSource object is deployed with the application.

Connections as Resources in Applications

Databases are accessed through Connection objects.

PooledConnection objects are a variant of Connection objects, which is a set of Connection objects.

A Connection object is destroyed after the connection with the database is closed.

In case of PooledConnection object, a Connection object is used from the pool and returned to the pool of Connections after the connection with the database is closed.

Database connectivity can also be established through Persistence API, which provides the object-relational mapping for data in a Java application.

Injection

The components in a Java application are distributed and each component uses various resources.

All the resources required by the application are not instantiated at compile time instead an annotation is added to the code.

Based on the annotations, the references and instances are generated only during runtime.

Process of injection of resources or dependencies into the application is managed by an API such as Context Dependency Injection (CDI). There are two variants of the injection process:

- Resource injection
- Dependency injection

Resource Injection 1-3

Enables developers to inject resources such as databases, connectors, and so on into container managed components such as Web components and bean components.

Resources must be defined in the JNDI namespace so that they can be injected into the component.

The javax.annotation.Resource annotation is used to declare a reference to a resource.

The annotation is added in the code as @Resource.

Resource Injection 2-3

A Resource annotation has five elements: name, type, authenticationType, shareable, mappedName, and description

- name field stores the JNDI name of the resource being accessed
- type implies the type of the resource
- authenticationType refers to the method of authentication used by the resource, if any
- shareable implies whether the resource can be shared with other components
- mappedName implies any system specific name to which the resource has to be mapped
- description describes the resource

Resource Injection 3-3

Variants of Resource Injection

Field based –The field declaration has to be preceded by the @Resource annotation.

Method based – The setter method of an attribute can be declared preceded by the @Resource annotation.

Class based - The @Resource annotation with appropriate name and type values applicable to the entire class.

Dependency Injection

Objects of different classes collaborate together during application execution.

There exists dependencies among the objects of the application. Dependency injection resolves these dependencies at runtime.

Dependency injection is implemented with @Inject annotation.

The javax.xml.ws.WebServiceRef annotation is used to define dependency injection in Web services.

Context Dependency Injection (CDI)

Service provided by Java platform to provide dependency injection in both enterprise and Web applications.

An integration library which integrates JSF components and managed bean components.

Binds context to the objects of the application; based on the context appropriate objects are injected into the application.

Annotations prompt the inject points of the objects into the application.

Features of CDI

Dependency injection is type safe as it does not resolve the dependencies based on the name of the object.

CDI is integrated with Expression Language (EL) which enables direct access of components from JavaServerFaces and JavaServerPages.

It enables integration of third party service with the Java environment.

It allows loosely coupled components in the application.

CDI provides for an event-notification model.

A Web conversation scope is defined in CDI apart from session, request, and application scope.

Packaging Applications

Java applications are packaged as archive files.

- Standard Java applications are packaged as Java Archive (JAR)files.
- Enterprise applications are packaged as Enterprise Archive (EAR) files.
- Web applications are packaged as Web Archive (WAR) files.

The archive files are packaged with deployment descriptors.

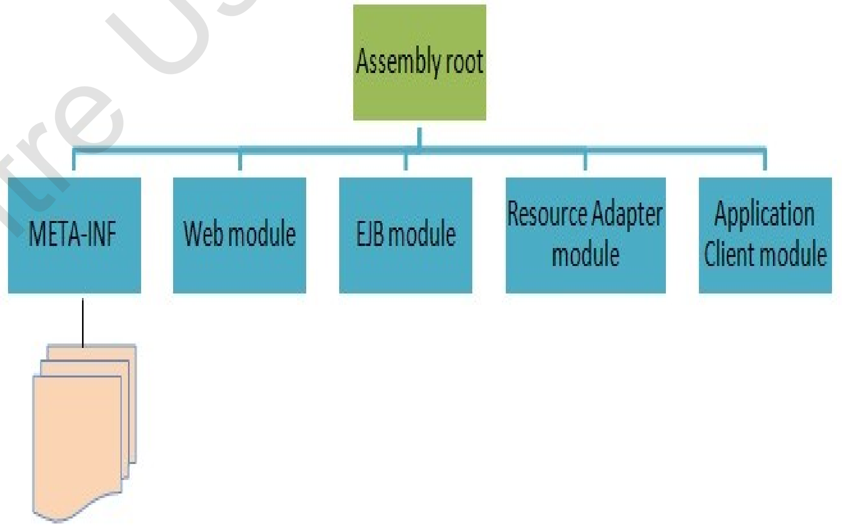
- Deployment descriptors are .xml files which are declarative.
- Deployment descriptor is used at runtime to deploy the application.

Structure of a Generic EAR File

The META-INF files comprises metadata pertaining to the deployment of the application as deployment descriptors, library files, and so on.

Modules deployed in Java EE application:

- EJB module
- Web module
- Resource adapter module
- Application client module



Components of Generic EAR File

Types of deployment descriptors

- Java EE deployment descriptor
- Runtime deployment descriptor

Types of application modules

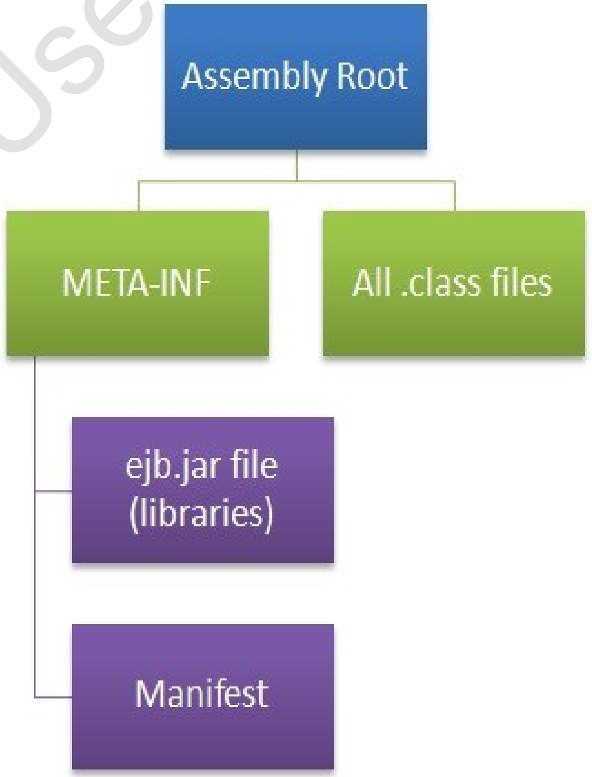
- EJB module
- Web module
- Resource Adapter module
- Application client module

Packaging Enterprise Beans as Jar Files

Enterprise beans are Java classes which implement the business logic of the enterprise application.

EJBs are either packaged in JAR or WAR files.

The figure shows the hierarchy according to which the enterprise beans are organized when packaged in a EAR file.



Packaging Enterprise Beans as WAR Files 1-2

- Web applications use beans to implement the business logic of the application.
- The Web components of the application make the enterprise beans compatible to be accessed over Internet.
- The beans of the applications are packaged as .jar file, which together with other components of the Web application are packaged as .war (Web Archive) files.

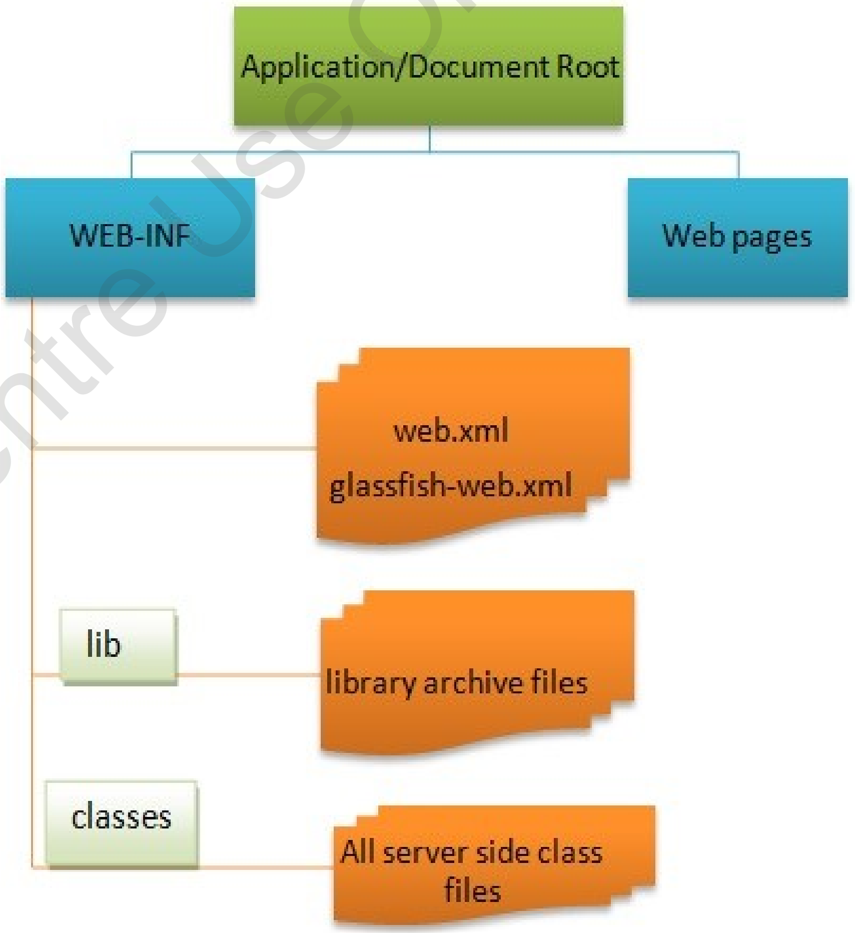
Packaging Enterprise Beans as WAR Files 2-2

The deployment of an application requires that all the components of the application follow certain standard location policy. Following are some of the locations for different components of the application:

- The enterprise bean class files stored in a WAR module are placed in the WEB-INF/classes directory.
- Bean components if provided as .jar files, must be placed in the WEB-INF/lib directory of the WAR module.
- The ejb-jar.xml file is a deployment descriptor which is not mandatory to be created for application deployment. It has to be located in WEB-INF directory.

Packaging Web Archives

- Web applications are packaged as Web archives with a standard structure which enables easy deployment of the application.
- The right branch has all the Web pages and the left branch has all the directories and libraries.
- The classes directory includes all the bean classes, server side classes, servlets, and other utility classes.
- The lib directory contains all the beans and library files used in the application.
- Web archive have two deployment descriptors – web.xml, ejb-jar.xml



Packaging Resource Adapter Archives

Resource adapters are required in an application to make accessible the different types of resources used in an application.

They can be packaged as a part of an enterprise archive file or as a separate .rar file.

Summary

- An application has various components such as database, mail service providers, messaging services, and so on. All these are collectively termed as resources in the application.
- Java does not instantiate the resources during compilation. In order to keep the application loosely coupled, the references to the resource instances are instantiated at runtime.
- Instead of injecting the resources into the application, annotations are added at places where the resource reference is required.
- Java implements dependency and resource injection through Context Dependency Injection (CDI) service.
- Enterprise applications are packaged as EAR files and Web applications are packaged as WAR files.
- The archive files may or may not have deployment descriptors. Each archive file has a specific structure. Different files are arranged in various directories of the archive files according to a standard structure.