CSC584 Enterprise Programming

CHAPTER 8 – PACKAGING AND DEPLOYMENT OF ENTERPRISE APPLICATION

Chapter 8
Outline

Packaging and Deployment of Enterprise Application

- Packaging components
- Packaging Java EE applications – EJB modules, Web modules
- Deployment descriptors
- Deployment tools

Packaging components

- A Java EE applications delivered in a Java Archive (JAR) file, a Web Archive (WAR) file, or an Enterprise Archive (EAR) file.
- A WAR or EAR file is a standard JAR (.jar) file with a .war or .ear extension.
- Using JAR, WAR, and EAR files and modules makes it possible to assemble a number of different Java EE applications using some of the same components.
- No extra coding is needed; it is only a matter of assembling (or packaging) various Java EE modules into Java EE JAR, WAR, or EAR files

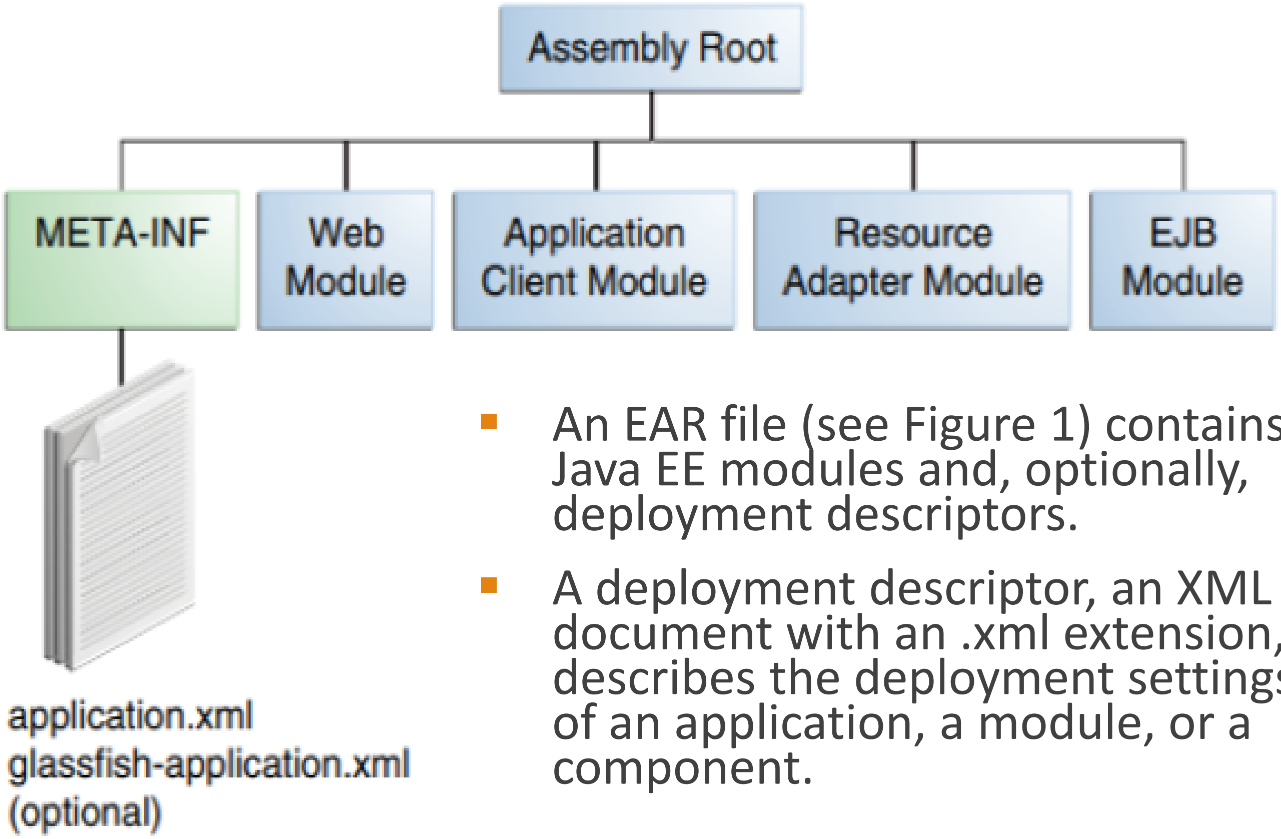
Packaging Web Applications

- The Java EE specification defines how the web application can be archived into a **web application archive (WAR)**
- **WAR files** are
  - Java archives with a **.war extension**
  - Packaged using the same specification as zip files
  - Understood by all Java EE compliant application servers
- WAR files can be directly deployed in servlet containers such as Glassfish or Tomcat

NetBeans WAR files

- To make a WAR for your NetBeans project, right click on the project node and select **Build Project**.
- The WAR file will be placed in the **“dist” sub-directory** of your project folder
- The following video shows how to generate WAR file:
  - https://www.youtube.com/watch?v=Gkm3qxSEqP0

Figure 1 EAR File Structure



- An EAR file (see Figure 1) contains Java EE modules and, optionally, deployment descriptors.
- A deployment descriptor, an XML document with an .xml extension, describes the deployment settings of an application, a module, or a component.

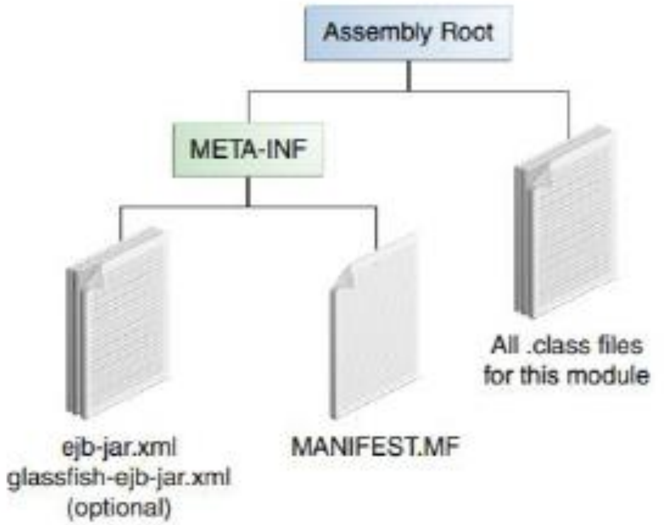
Java EE modules are of the following types

- EJB modules, which **contain class files for enterprise beans** and, optionally, an EJB deployment descriptor. EJB modules are packaged as JAR files with a .jar extension.
- Web modules, which contain **servlet class files, web files, supporting class files, GIF and HTML files,** and, optionally, a web application deployment descriptor. Web modules are packaged as JAR files with a .war (web archive) extension.
- Application client modules, which contain **class files and, optionally, an application client deployment descriptor**. Application client modules are packaged as JAR files with a .jar extension.

Packaging Java EE applications – EJB modules

- An EJB JAR file is portable and can be used for various applications.
- To assemble a Java EE application, package one or more modules, such as EJB JAR files, into an EAR file, the archive file that holds the application.
- When deploying the EAR file that contains the enterprise bean's EJB JAR file, you also deploy the enterprise bean to GlassFish Server.
- Figure 2 shows the contents of an EJB JAR file

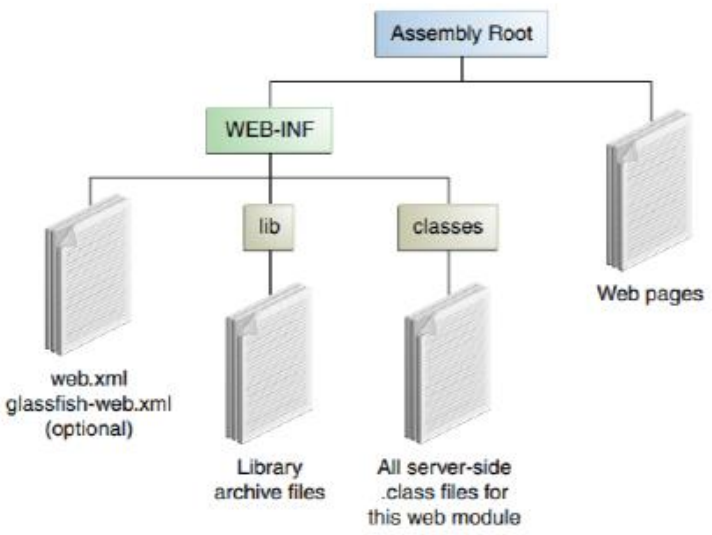
Figure -2 Structure of an Enterprise Bean JAR



Packaging Java EE applications – Web modules

- A web module can be deployed as an unpacked file structure or can be packaged in a JAR file known as a Web Archive (WAR) file.
- Because the contents and use of WAR files differ from those of JAR files, WAR file names use a .war extension.
- The web module just described is portable; you can deploy it into any web container that conforms to the Java Servlet specification.

Figure-3 Web Module Structure



Deployment Descriptor

- In a java web application a file named web.xml is known as deployment descriptor.
- It is a xml file and <web-app> is the root element for it.
- When a request comes web server uses web.xml file to map the URL of the request to the specific code that handle the request.

How your web server finds the servlet

- Web server reads ***XML*** files that tell it about the servlets
- Each web application must have a Deployment Descriptor (DD) file, WEB-INF/web.xml file
- The root element of the DD file is **web-app**
  - This element has a bunch of "boilerplate" attributes
  - You don't have to know what any of it means
- The rest of the XML file gives names for the servlet, provides parameters, etc.
  - We'll cover only the most essential parts

Three names

- Every servlet has three names:
  - The “real” name, given to it by the programmer
  - An “internal” name, used only within the **web.xm**l file
  - The name that the user (client) knows it by
- The **<servlet>** element associates the fully-qualified class name with the internal name
- The **<servlet-mapping>** element associates the internal name with the name known to the client
- The reason for all this is to increase security by hiding the real name from the user

web.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation= http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd  version="2.4">

<servlet>

    <servlet-name>Some internal name</servlet-name>

    <servlet-class>com.example.web.MyServlet</servlet-class>

</servlet>

<servlet-mapping>

    <servlet-name>Some internal name</servlet-name>

    <url-pattern>/NameSeenByUser.do</url-pattern>

</servlet-mapping>
```

The Servlet

```
public class MyServlet extends HttpServlet {

    public void doPost(HttpServletRequest request,
    HttpServletResponse response)
    throws IOException, ServletException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        String value = request.getParameter("name");
        out.println("<html><body>I got: " + name + " = " +
        value + "</body></html>");
    }
}
```

Flow

1. The user submits an HTML form
2. Web server finds the servlet based on the URL and the deployment descriptor (web.xml) and passes the request to the servlet
3. The servlet computes a response

   Either:

      The servlet writes an HTML page containing the response

   Or:

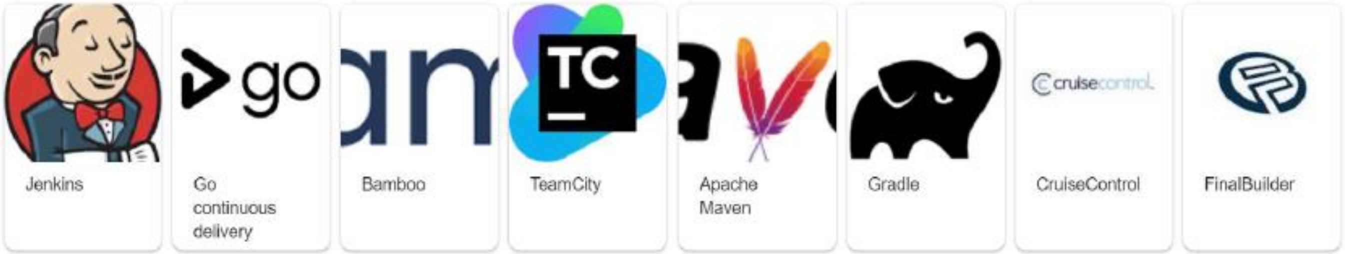
      The servlet forwards the response to the JSP

      The JSP embeds the response in an HTML page
4. Web server returns the HTML page to the user

Deployment tools

- A Java EE application is packaged into one or more standard units for deployment to any Java EE platform-compliant system. Each unit contains
  - A functional component or components, such as an enterprise bean, web page, servlet, or applet
  - An optional deployment descriptor that describes its content
- Once a Java EE unit has been produced, it is ready to be deployed. Deployment typically involves using a platform's deployment tool to specify location-specific information, such as a list of local users who can access it and the name of the local database.
- Once deployed on a local platform, the application is ready to run.

Examples of deployment tools


Jenkins
Go continuous delivery
Bamboo
TeamCity
Apache Maven
Gradle
CruiseControl
FinalBuilder