Leading Open Source Middleware

J2EE Programmer's Guide

JOnAS Team ( )

- Fev 2008 -

Table of Contents

The target audience for this guide is the application component provider, i.e., the person in charge of developing the software components on the server side (the business tier).

Chapter 1. Principles

JOnAS supports two types of J2EE application components: Enterprise Beans and Web components . In addition to providing guides for construction of application components, guides are supplied for application assembly, deployment, and administration.

1.1. Enterprise Bean Creation

The individual in charge of developing Enterprise Beans should consult the Enterprise Beans Programmer's Guide [ejb2_programmer_guide.html] for instructions on how to perform the following tasks:

1. Write the source code for the beans.

2. Specify the deployment descriptor.

3. Bundle the compiled classes and the deployment descriptor into an EJB JAR file.

This JOnAS documentation provides guides for developing the three types of enterprise bean:

- Session beans [ejb2_programmer_guide.html#ejb2.session]

- Entity beans [ejb2_programmer_guide.html#ejb2.entity]

- Message driven beans [ejb2_programmer_guide.html#ejb2.mdb]

Deployment descriptor specification is presented in the Defining the Deployment Descriptor [ejb2_programmer_guide.html#ejb2.deploy] chapter.

More specific issues related to transaction behaviour, the Enterprise Bean environment, or security service, are presented in the corresponding chapters: Transactional behaviour [ejb2_programmer_guide.html#ejb2.transaction] , Enterprise Bean Environment [ejb2_programmer_guide.html#ejb2.environment] , Security Management [ejb2_programmer_guide.html#ejb2.security] .

Principles and tools for providing EJB JAR files are presented in the chapters EJB Packaging [ejb2_programmer_guide.html#ejb2.packaging] and Deployment and Installation Guide [ejb2_programmer_guide.html#ejb2.deploy] .

1.2. Web Components Creation

Web designers in charge of JSP pages and software developers providing servlets can consult the Web Application Programmer's Guide.

The Developing Web Components [web_pg.html#web.component] guide explains how to construct Web components, as well as how to access Enterprise Beans from within the Web Components.

Deployment descriptor specification is presented in the Defining the Web Deployment Descriptor [web_pg.html#web.deployment] chapter.

Web components can be used as Web application components or as J2EE application components. In both cases, a WAR file will be created, but the content of this file is different in the two situations. In the first case, the WAR contains the Web components and the Enterprise Beans. In the second case, the WAR does not contain the Enterprise Beans. The EJB JAR file containing the Enterprise Beans is packed together with the WAR file containing the Web components, into an EAR file. Principles and tools for providing WAR files are presented in WAR Packaging [web_pg.html#web.packaging] and the Deployment and Installation Guide [Deployer.html#Deployer] .

1.3. Application Deployer and Administrator

JOnAS provides tools for the deployment and administration of Enterprise Beans (EJB JARs), Web applications (WARs), and J2EE applications (EARs).

The Deployment and Installation Guide [Deployer.html#Deployer] covers issues related to the deployment of application components.

The Administration Guide [Admin.html#Admin] presents information about how to manage the JOnAS server and the JOnAS services that allow deployment of the different types of application components: EJB Container service, Web Container service, and EAR service.

1.4. J2EE Application Assembler

The application assembler in charge of assembling the application components already bundled in EJB JAR files and WAR files into a J2EE EAR file, can obtain useful information from the J2EE Application Assembler's Guide [eardeploy.html].

Chapter 2. JOnAS Class Loader

2.1. JOnAS class loader hierarchy

This section describes a new and important key feature of the J2EE integration: the class loader hierarchy in JOnAS.

2.1.1. Understanding class loader hierarchy

An application is deployed by its own class loader. This means, for example, that if a WAR and an EJB JAR are deployed separately, the classes contained in the two archives are loaded with two separate classloaders with no hierarchy between them. Thus, the EJBs from within the JAR will not be visible to the Web components in the WAR. This is not acceptable in cases where the Web components of an application need to reference and use some of the EJBs (this concerns local references in the same JVM).

For this reason, prior to EAR files, when a Web application had to be deployed using EJBs, the EJB JAR had to be located in the WEB-INF/lib directory of the Web application.

Currently, with the J2EE integration and the use of the EAR packaging, class visibility problems no longer exist and the EJB JAR is no longer required in the WEB-INF/lib directory.

The following sections describe the JOnAS class loader hierarchy and explain the mechanism used to locate the referenced classes.

2.1.2. Commons class loader

The `commons` class loader is a JOnAS-specific class loader that will load all classes required to start the JOnAS server. This class loader has the system class loader as parent class loader. The commons class loader adds all the common libraries required to start the JOnAS server (J2EE apps, commons logging, objectweb components, etc.); it also loads the classes located in `XTRA_CLASSPATH` .

The JARs loaded by the `commons` class loader are located under the `JONAS_ROOT/lib/commons` and `JONAS_BASE/lib/commons` directories. You can extend this class loader by adding your own JARs inside these directories. If you are using a Jetty packaging, `JONAS_ROOT/lib/jetty` will be loaded too.

Note that the lib/ext extension mechanism is now deprecated. You should place additional JARs directly in the classloader directory (`commons` , `apps` , `tools`) under `JONAS_ROOT` and/or `JONAS_BASE` . You should now use the `JONAS_BASE/lib/commons` if you want to extend the `commons` ClassLoader, `JONAS_BASE/lib/apps` for `apps` ClassLoader and `JONAS_BASE/lib/tools` for `tools` ClassLoader. Notice that the `JONAS_BASE` extension directories are always loaded after `JONAS_ROOT` directories, so if the same class is in `JONAS_ROOT/lib/commons` and in `JONAS_BASE/lib/commons` , the one located in `JONAS_ROOT/lib/commons` will be used.

To have a library available for each component running inside JOnAS, add the required JAR files in the `JONAS_ROOT/lib/commons` directory or in the `JONAS_BASE/lib/commons` . All jars in subordinate directories will also be loaded. If a specific jar is needed only for a web application (i.e., need to use a version of a jar file which is different than a version loaded by JOnAS), change the compliance of the web application classloader to the java 2 delegation model. Refer to the following: WEB class loader. It is also possible to use the extension mechanism, which is described in the section dependencies of the J2EE specification (section 8.1.1.28).

2.1.3. Application class loader

The `application` class loader is a JOnAS-specific class loader that will load all application classes required by the user applications. This implies that this loader will load all single RAR files.

Thus, all applications have the visibility of the resource adapters classes. This class loader has the `commons` class loader as parent class loader.

The JARs loaded by the `application` class loader are located under the `JONAS_ROOT/lib/apps` directory, under `JONAS_ROOT/lib/catalina/common/lib` directory (`CATALINA_HOME/common/lib` if you are not using the Tomcat package) and under `JONAS_BASE/lib/apps` directory. You can extend this class loader by adding your own JARs inside these directories.

2.1.4. Tools class loader

The `tools` class loader is a JOnAS-specific class loader that will load all classes for which applications do not require visibility. (User applications will not have the ability to load the classes packaged in the `tools` class loader). For example, it includes the jakarta velocity and digester components. This class loader has the `commons` class loader as parent class loader.

The JARs loaded by the `tools` class loader are located under the `JONAS_ROOT/lib/tools` directory and under the `JONAS_BASE/lib/tools` directory. You can extend this class loader by adding your own JARs inside these directories.

2.1.5. Tomcat class loader

The `tomcat` class loader is a class loader that will load all classes of the tomcat server (`CATALINA_HOME/server/lib` directory). The classes of the common directory of tomcat (`CATALINA_HOME/common/lib` directory) are loaded by the `application` classloader and not by this `tomcat` classloader. Applications have the visibility of the classes and not the `server` classes. To have the visibility of the `server` classes, the context must have the privileged attribute set to `true` . This class loader has the `application` class loader as parent class loader.

The JARs loaded by the `tomcat` class loader are located under the `JONAS_ROOT/lib/catalina/server/lib` directory (if using Tomcat packaging, unless these libs are located under `CATALINA_HOME/server/lib`). You can extend this class loader by adding your own JARs inside this directory.

2.1.6. JOnAS class loaders

The JOnAS class loader hierarchy that allows the deployment of EAR applications without placing the EJB JAR in the WEB-INF/lib directory consists of the following:

2.1.6.1. EAR class loader

The EAR class loader is responsible for loading the EAR application. There is only one EAR class loader per EAR application. This class loader is the child of the application class loader, thus making JOnAS classes visible to it.

2.1.6.2. EJB class loader

The EJB class loader is responsible for loading all the EJB JARs of the EAR application, thus all the EJBs of the same EAR application are loaded with the same EJB classloader. This class loader is the child of the EAR class loader.

2.1.6.3. WEB class loader

The WEB class loader is responsible for loading the Web components. There is one WEB class loader per WAR file, and this class loader is the child of the EJB class loader. Using this class loader hierarchy (the EJB class loader is the parent of the WEB class loader) eliminates the problem of visibility between classes when a WEB component tries to reference EJBs; the classes loaded with the EJB class loader are definitely visible to the classes loaded by its child class loader (WEB class loader).

The compliance of the class loader of the web application to the java 2 delegation model can be changed by using the jonas-web.xml file. This is described in the section Defining the Web Deployment Descriptor [j2eeprogrammerguide.html#j2ee.pg.classloader].

If the java2-delegation-model element is set to false, the class loader of the web application looks for the class in its own repository before asking its parent class loader.

2.2. Conclusion

The resulting JOnAS class loader hierarchy is as follows: