18

Using the Java Naming and Directory Interface

A *naming service* maintains a set of bindings, which relate names to objects. The Java EE naming service is based on the Java Naming and Directory Interface (JNDI) API. The JNDI API allows application components and clients to look up distributed resources, services, and EJB components. For general information about the JNDI API, see `http://www.oracle.com/technetwork/java/jndi/index.html`. You can also see the JNDI tutorial at `http://download.oracle.com/javase/jndi/tutorial/`.

The following topics are addressed here:

- Accessing the Naming Context
- Configuring Resources
- Using a Custom `jndi.properties` File
- Mapping References

> **Note:** The Web Profile of the GlassFish Server supports the EJB 3.1 Lite specification, which allows enterprise beans within web applications, among other features. The full GlassFish Server supports the entire EJB 3.1 specification. For details, see JSR 318 (`http://jcp.org/en/jsr/detail?id=318`).

Accessing the Naming Context

The Oracle GlassFish Server provides a naming environment, or *context*, which is compliant with standard Java EE requirements. A `Context` object provides the methods for binding names to objects, unbinding names from objects, renaming objects, and listing the bindings. The `InitialContext` is the handle to the Java EE naming service that application components and clients use for lookups.

The JNDI API also provides subcontext functionality. Much like a directory in a file system, a subcontext is a context within a context. This hierarchical structure permits better organization of information. For naming services that support subcontexts, the `Context` class also provides methods for creating and destroying subcontexts.

The following topics are addressed here:

- Portable Global JNDI Names
- GlassFish Server V2 Vendor-Specific Global JNDI Names
- Disabling GlassFish Server V2 JNDI Names

- Accessing EJB Components Using the `CosNaming` Naming Context
- Accessing EJB Components in a Remote GlassFish Server
- Naming Environment for Lifecycle Modules

Note: Each resource within a server instance must have a unique name. However, two resources in different server instances or different domains can have the same name.

Portable Global JNDI Names

If an EJB component is a kind of session bean and it is deployed to any implementation supporting the EJB 3.1specification (for example, GlassFish Server 3.1.2), it automatically has one or more portable JNDI names defined based on the syntax in the specification. Note that this is true of existing EJB 3.0 and 2.x applications that are deployed to an implementation supporting EJB 3.1. No code changes are required to the bean class itself in order to have the portable global JNDI name automatically assigned when deployed to an EJB 3.1 container.

For more information, see the Java EE 6 Platform Specification, section EE.5.2.2, "Application Component Environment Namespaces" (`http://jcp.org/en/jsr/detail?id=316`), and the EJB 3.1 Specification, section 4.4, "Global JNDI Access" (`http://jcp.org/en/jsr/detail?id=318`).

If the `disable-nonportable-jndi-names` property is set to false (the default), a GlassFish Server V2-specific JNDI name is assigned in addition to a portable global JNDI name. For more information, see GlassFish Server V2 Vendor-Specific Global JNDI Names and Disabling GlassFish Server V2 JNDI Names.

GlassFish Server V2 Vendor-Specific Global JNDI Names

GlassFish Server v2 vendor-specific global JNDI names are assigned according to the following precedence rules:

1. A global JNDI name assigned in the `glassfish-ejb-jar.xml`, `glassfish-web.xml`, or `glassfish-application-client.xml` deployment descriptor file has the highest precedence. See Mapping References.
2. A global JNDI name assigned in a `mapped-name` element in the `ejb-jar.xml`, `web.xml`, or `application-client.xml` deployment descriptor file has the second highest precedence. The following elements have `mapped-name` subelements: `resource-ref`, `resource-env-ref`, `ejb-ref`, `message-destination`, `message-destination-ref`, `session`, `message-driven`, and `entity`.
3. A global JNDI name assigned in a `mappedName` attribute of an annotation has the third highest precedence. The following annotations have `mappedName` attributes: `@javax.annotation.Resource`, `@javax.ejb.EJB`, `@javax.ejb.Stateless`, `@javax.ejb.Singleton`, `@javax.ejb.Stateful`, and `@javax.ejb.MessageDriven`.
4. In most cases, a default global JNDI name is assigned (and recorded in the server log) if no name is assigned in deployment descriptors or annotations.
   - For a session or entity bean, a GlassFish Server V2-specific JNDI name is assigned as follows:
     - For an EJB 2.x dependency or a session or entity bean with a remote interface, the default is the fully qualified name of the home interface.

  - For an EJB 3.0 dependency or a session bean with a remote interface, the default is the fully qualified name of the remote business interface.
  - If both EJB 2.x and EJB 3.0 remote interfaces are specified, or if more than one 3.0 remote interface is specified, there is no GlassFish Server V2-specific default. For an entity bean, a global JNDI name must be assigned.

- For all other component dependencies that must be mapped to global JNDI names, the default is the name of the dependency relative to `java:comp/env`. For example, in the `@Resource(name="jdbc/Foo") DataSource ds;` annotation, the global JNDI name is `jdbc/Foo`.

Disabling GlassFish Server V2 JNDI Names

The EJB 3.1 specification supported by GlassFish Server 3.1.2 defines portable EJB JNDI names for session beans. Because of this, there is less need to continue to use older vendor-specific JNDI names.

By default, GlassFish Server V2-specific JNDI names are applied automatically by GlassFish Server 3.1.2 for backward compatibility. However, this can lead to some ease-of-use issues. For example, deploying two different applications containing a remote EJB component that exposes the same remote interface causes a conflict between the default JNDI names.

The default handling of V2-specific JNDI names in GlassFish Server 3.1.2 can be managed by using the `asadmin` command:

```
asadmin> set server.ejb-container.property.disable-nonportable-jndi-names="true"
```

The `disable-nonportable-jndi-names` property is a boolean flag that can take the following values:

`false`
Enables the automatic use of GlassFish Server V2-specific JNDI names in addition to portable global JNDI names. This is the default setting.

`true`
Disables the automatic use of V2-specific JNDI names. In all cases, only portable global JNDI names are used.

Note that this setting applies to all session beans deployed to the server.

Accessing EJB Components Using the `CosNaming` Naming Context

The preferred way of accessing the naming service, even in code that runs outside of a Java EE container, is to use the no-argument `InitialContext` constructor. However, if EJB client code explicitly instantiates an `InitialContext` that points to the `CosNaming` naming service, it is necessary to set the `java.naming.factory.initial` property to `com.sun.jndi.cosnaming.CNCtxFactory` in the client JVM software when accessing EJB components. You can set this property using the `asadmin create-jvm-options` command, as follows:

```
asadmin> create-jvm-options
-Djava.naming.factory.initial=com.sun.jndi.cosnaming.CNCtxFactory
```

For details about `asadmin create-jvm-options`, see the *Oracle GlassFish Server Reference Manual*.

Or you can set this property in the code, as follows:

```
Properties properties = null;
      try {
            properties = new Properties();
            properties.put("java.naming.factory.initial",
                  "com.sun.jndi.cosnaming.CNCtxFactory");
            ...
          }
      ...
```

The `java.naming.factory.initial` property applies to only one instance. The property is not cluster-aware.

Accessing EJB Components in a Remote GlassFish Server

The recommended approach for looking up an EJB component in a remote GlassFish Server from a client that is a servlet or EJB component is to use the Interoperable Naming Service syntax. Host and port information is prepended to any global JNDI names and is automatically resolved during the lookup. The syntax for an interoperable global name is as follows:

`corbaname:iiop:`*`host`*`:`*`port`*`#a/b/`*`name`*

This makes the programming model for accessing EJB components in another GlassFish Server exactly the same as accessing them in the same server. The deployer can change the way the EJB components are physically distributed without having to change the code.

For Java EE components, the code still performs a `java:comp/env` lookup on an EJB reference. The only difference is that the deployer maps the `ejb-ref` element to an interoperable name in a GlassFish Server deployment descriptor file instead of to a simple global JNDI name.

For example, suppose a servlet looks up an EJB reference using `java:comp/env/ejb/Foo`, and the target EJB component has a global JNDI name of `a/b/Foo`.

The `ejb-ref` element in `glassfish-web.xml` looks like this:

```
<ejb-ref>
   <ejb-ref-name>ejb/Foo</ejb-ref-name>
   <jndi-name>corbaname:iiop:host:port#a/b/Foo</jndi-name>
</ejb-ref>
```

The code looks like this:

```
Context ic = new InitialContext();
Object o = ic.lookup("java:comp/env/ejb/Foo");
```

For a client that doesn't run within a Java EE container, the code just uses the interoperable global name instead of the simple global JNDI name. For example:

```
Context ic = new InitialContext();
Object o = ic.lookup("corbaname:iiop:host:port#a/b/Foo");
```

Objects stored in the interoperable naming context and component-specific (`java:comp/env`) naming contexts are transient. On each server startup or application reloading, all relevant objects are re-bound to the namespace.

Naming Environment for Lifecycle Modules

Lifecycle listener modules provide a means of running short or long duration tasks based on Java technology within the GlassFish Server environment, such as instantiation of singletons or RMI servers. These modules are automatically initiated at server startup and are notified at various phases of the server life cycle. For details about lifecycle modules, see Developing Lifecycle Listeners.

The configured properties for a lifecycle module are passed as properties during server initialization (the `INIT_EVENT`). The initial JNDI naming context is not available until server initialization is complete. A lifecycle module can get the `InitialContext` for lookups using the method `LifecycleEventContext.getInitialContext()` during, and only during, the `STARTUP_EVENT`, `READY_EVENT`, or `SHUTDOWN_EVENT` server life cycle events.

Configuring Resources

The GlassFish Server exposes special resources in the naming environment.

- External JNDI Resources
- Custom Resources
- Built-in Factories for Custom Resources
- Using Application-Scoped Resources

External JNDI Resources

An external JNDI resource defines custom JNDI contexts and implements the javax.naming.spi.InitialContextFactory interface. There is no specific JNDI parent context for external JNDI resources, except for the standard `java:comp/env/`.

Create an external JNDI resource in one of these ways:

- To create an external JNDI resource using the Administration Console, open the Resources component, open the JNDI component, and select External Resources. For details, click the Help button in the Administration Console.
- To create an external JNDI resource, use the `asadmin create-jndi-resource` command. For details, see the *Oracle GlassFish Server Reference Manual*.

Custom Resources

A custom resource specifies a custom server-wide resource object factory that implements the javax.naming.spi.ObjectFactory interface. There is no specific JNDI parent context for external JNDI resources, except for the standard `java:comp/env/`.

Create a custom resource in one of these ways:

- To create a custom resource using the Administration Console, open the Resources component, open the JNDI component, and select Custom Resources. For details, click the Help button in the Administration Console.
- To create a custom resource, use the `asadmin create-custom-resource` command. For details, see the *Oracle GlassFish Server Reference Manual*.

Built-in Factories for Custom Resources

The GlassFish Server provides built-in factories for the following types of custom resources:

- JavaBeanFactory
- PropertiesFactory
- PrimitivesAndStringFactory
- URLFactory

Template `glassfish-resources.xml` files for these built-in factories and a `README` file are available at *as-install*`/lib/install/templates/resources/custom/`. For more information about the `glassfish-resources.xml` file, see the *Oracle GlassFish Server Application Deployment Guide*.

JavaBeanFactory

To create a custom resource that provides instances of a JavaBean class, follow these steps:

1. Set the custom resource's factory class to `org.glassfish.resources.custom.factory.JavaBeanFactory`.
2. Create a property in the custom resource for each setter method in the JavaBean class.

   For example, if the JavaBean class has a method named `setAccount`, specify a property named `account` and give it a value.
3. Make sure the JavaBean class is accessible to the GlassFish Server.

   For example, you can place the JavaBean class in the *as-install*`/lib` directory.

PropertiesFactory

To create a custom resource that provides properties to applications, set the custom resource's factory class to `org.glassfish.resources.custom.factory.PropertiesFactory`, then specify one or both of the following:

- Create a property in the custom resource named `org.glassfish.resources.custom.factory.PropertiesFactory.fileName` and specify as its value the path to a properties file or an XML file.

  The path can be absolute or relative to *as-install*. The file must be accessible to the GlassFish Server.

  If an XML file is specified, it must match the document type definition (DTD) specified in the API definition of java.util.Properties (`http://download.oracle.com/javase/6/docs/api/java/util/Properties.html`).
- Create the desired properties directly as properties of the custom resource.

  If both the `fileName` property and other properties are specified, the resulting property set is the union. If the same property is defined in the file and directly in the custom resource, the value of the latter takes precedence.

PrimitivesAndStringFactory

To create a custom resource that provides Java primitives to applications, follow these steps:

1. Set the custom resource's factory class to `org.glassfish.resources.custom.factory.PrimitivesAndStringFactory`.

2. Set the custom resource's resource type to one of the following or its fully qualified wrapper class name equivalent:
   - int
   - long
   - double
   - float
   - char
   - short
   - byte
   - boolean
   - String
3. Create a property in the custom resource named value and give it the value needed by the application.

   For example, If the application requires a double of value 22.1, create a property with the name value and the value 22.1.

URLFactory

To create a custom resource that provides URL instances to applications, follow these steps:

1. Set the custom resource's factory class to org.glassfish.resources.custom.factory.URLObjectFactory.
2. Choose which of the following constructors to use:
   - URL(protocol, host, port, file)
   - URL(protocol, host, file)
   - URL(spec)
3. Define properties according to the chosen constructor.

   For example, for the first constructor, define properties named protocol, host, port, and file. Example values might be http, localhost, 8085, and index.html, respectively.

   For the third constructor, define a property named spec and assign it the value of the entire URL.

Using Application-Scoped Resources

You can define an application-scoped JNDI or other resource for an enterprise application, web module, EJB module, connector module, or application client module by supplying a glassfish-resources.xml deployment descriptor file. For details, see "Application-Scoped Resources" in the *Oracle GlassFish Server Application Deployment Guide*.

Using a Custom jndi.properties File

To use a custom jndi.properties file, JAR it and place it in the *domain-dir*/lib directory. This adds the custom jndi.properties file to the Common class loader. For more information about class loading, see Class Loaders.

For each property found in more than one `jndi.properties` file, the Java EE naming service either uses the first value found or concatenates all of the values, whichever makes sense.

Mapping References

The following XML elements in the GlassFish Server deployment descriptors map resource references in application client, EJB, and web application components to JNDI names configured in GlassFish Server:

- `resource-env-ref` - Maps the `@Resource` or `@Resources` annotation (or the `resource-env-ref` element in the corresponding Java EE XML file) to the absolute JNDI name configured in GlassFish Server.
- `resource-ref` - Maps the `@Resource` or `@Resources` annotation (or the `resource-ref` element in the corresponding Java EE XML file) to the absolute JNDI name configured in GlassFish Server.
- `ejb-ref` - Maps the `@EJB` annotation (or the `ejb-ref` element in the corresponding Java EE XML file) to the absolute JNDI name configured in GlassFish Server.

  JNDI names for EJB components must be unique. For example, appending the application name and the module name to the EJB name is one way to guarantee unique names. In this case, `mycompany.pkging.pkgingEJB.MyEJB` would be the JNDI name for an EJB in the module `pkgingEJB.jar`, which is packaged in the `pkging.ear` application.

These elements are part of the `glassfish-web.xml`, `glassfish-application-client.xml`, `glassfish-ejb-jar.xml`, and `glassfish-application.xml` deployment descriptor files. For more information about how these elements behave in each of the deployment descriptor files, see "Elements of the GlassFish Server Deployment Descriptors" in the *Oracle GlassFish Server Application Deployment Guide*.

The rest of this section uses an example of a JDBC resource lookup to describe how to reference resource factories. The same principle is applicable to all resources (such as JMS destinations, JavaMail sessions, and so on).

The `@Resource` annotation in the application code looks like this:

```
@Resource(name="jdbc/helloDbDs") javax.sql.DataSource ds;
```

This references a resource with the JNDI name of `java:jdbc/helloDbDs`. If this is the JNDI name of the JDBC resource configured in the GlassFish Server, the annotation alone is enough to reference the resource.

However, you can use a GlassFish Server specific deployment descriptor to override the annotation. For example, the `resource-ref` element in the `glassfish-web.xml` file maps the `res-ref-name` (the name specified in the annotation) to the JNDI name of another JDBC resource configured in GlassFish Server.

```
<resource-ref>
   <res-ref-name>jdbc/helloDbDs</res-ref-name>
   <jndi-name>jdbc/helloDbDataSource</jndi-name>
</resource-ref>
```