11

Native Oracle JVM Support for JNDI

This chapter describes Oracle JVM support for Java Naming and Directory Interface (JNDI). This chapter contains the following sections:

- Overview of Oracle JVM Support for JNDI
- Requirements for Oracle JVM Support for JNDI
- OJDS Command-Line Tools
- OJDS APIs and Classes

11.1 Overview of Oracle JVM Support for JNDI

Native Oracle JVM support for JNDI enables you to bind Oracle data source objects, which contain specific database connection information, by a name in a directory structure. You can use this name to retrieve the particular connection information to establish a connection within an application. You can also change the database connection properties and the actual source database without changing the application by changing only the associated object to which a specific name is resolved. This feature also provides a general purpose directory service for storing objects and object references.

The Oracle Java Directory Service (OJDS) package, `oracle.aurora.jndi.ojds` provides the APIs for implementing JNDI support.

Related Topics

- OJDS APIs and Classes

11.2 Requirements for Oracle JVM Support for JNDI

This section describes the implementation requirements for JNDI support in the Oracle JVM. This section is divided into the following sections:

- Namespace
- Oracle Java Directory Service JNDI Name Space Provider
- Namespace Browser

11.2.1 Namespace

The namespace is represented similarly as in the typical Unix File System structure. The root directory and the directory separator are represented by the slash symbol (/). The root directory is owned by `SYS` and only `SYS` can create new subdirectories under it.

The following two directories (`DirContexts`) are created during the installation process of OJDS:

- `/public` directory

  The `/public` directory is a public area for testing and any user can bind, delete, or lookup objects in this directory.

- `/etc` directory

  The `/etc` directory is an area for the deployment of all production type objects that a client may need and is protected from any update or removal. The `/etc` directory is writable only by the `SYS` user, but is readable by all users.

11.2.1.1 Object permissions

You can assign permissions to the objects stored in the directory structure. These permissions are a union of the following permissions:

- Read
- Write
- Execute

The following table describes the permissions that you can assign to the objects stored in the directory structure:

Action	Parent Context Permissions	Child (obj/ctx) Permissions
`bind`	Write	NA
`unbind`	Write	Write
`createSubcontext`	Write	NA
`getAttributes`	Read	Read
`rebind`	Write	Write
`destroySub context`	Write	Write
`list`	Read	Read
`listBindings`	Read	Read
`lookup`	Read	Read
`lookupLink`	Read	Read
`rename (target)`	Write	Write (if exists)
`rename (source)`	Read	Read

Note:

All parent contexts must have Execute permission for operations to succeed.

11.2.1.2 Persistent Storage Tables, Indexes, and Sequences

The database tables owned by `OJVMSYS` store the following details for each object:

- Namespace metadata
- Bound names
- Attributes
- Permissions
- Stored object representations

11.2.1.3 Initial Contexts and Permissions

The following table shows the contexts that are created by default at the time of installation:

Name	Owner	Read	Write	Execute
/	SYS	PUBLIC	SYS	PUBLIC
/public	SYS	PUBLIC	PUBLIC	PUBLIC
/etc	SYS	PUBLIC	SYS	PUBLIC

11.2.1.4 Object and Context Default Permissions

When a context is created or an object is bound to the OJDS, then the Read and Execute permissions are granted to the user or schema that creates the context.

11.2.2 Oracle Java Directory Service JNDI Name Space Provider

This section describes the following Oracle Java Directory Service concepts:

- Directory Context
- StateFactories
- ObjectFactories
- OJDS URL Support
- Client classpath

11.2.2.1 Directory Context

The Oracle Java Directory Service (OJDS) must implement the interface as specified by the `javax.naming.directory.DirContext` context. The `javax.naming.directory.DirContext` context, the `oracle.aurora.jndi.ojds.OjdsServerContext` context, and the `oracle.aurora.jndi.ojds.OjdsClientContext` context provide the methods for examining and updating attributes associated with the objects, and enables searches of the directory for server-side and client-side executions respectively.

The following table describes the JNDI properties that you can use for creating a context or using a context:

Package Name	Description
`java.naming.factory.initial`	Specifies what class to use to create initial contexts for the application. The `oracle.aurora.jndi.ojds` package defines the `oracle.aurora.jndi.ojds.OjdsInitialContextFactory` for use with this property to create `InitialDirContext`.
`java.naming.security.principal`	Specifies the user ID for creating a database connection. You must specify the value for this property.
`java.naming.security.credentials`	Specifies the password for creating a database connection. You must specify the value for this property.
`java.naming.provider.url`	Specifies the connection URL for creating a database connection. This property is optional.

Package Name	Description
`java.naming.factory.url.pkgs`	Is a colon separated list of URL handlers for specific JNDI implementations. The `oracle.aurora.jndi.ojds.OjdsURLContextFactory` class returns a context based on an OJDS URL.

11.2.2.2 StateFactories

A `StateFactory` transforms a Java object into an object that can be stored in the implementing JNDI provider. The OJDS converts all the objects to bind to a serialized object. OJDS follows the specifications of the `java.io.Serializable` interface and the Java Object Serialization Specification for this conversion. Once serialized, the object is stored in the OJDS persistent store. No external `StateFactories` are supported for OJDS.

11.2.2.3 ObjectFactories

An `ObjectFactory` takes objects stored in the implementing JNDI provider and converts them to back into Java objects.The OJDS does not support external `ObjectFactories`. The serialized objects are created from their binary form that are retrieved from the OJDS persistent store. After an object is deserialized, OJDS handles the object in one of the following ways:

- If the object is a `Context`, then the `connection` and the `env` fields are set and a `DirContext` is returned.
- If the object is a `javax.naming.Reference`, then you can use the `DirectoryManager.getObjectInstance` method to create the object.
- If the object is neither a `Context` nor a `javax.naming.Reference`, then the object is returned as it is to the user.

The retrieved bytes specifying an object must conform to the `java.io.Serializable` interface standards. If the class implementing the object changes on the client, then the deserialization of the object can fail. So, you must be careful to maintain compatibility between the object bytes and the class or object stream deserializing the object bytes.

11.2.2.4 OJDS URL Support

The OJDS supports a URL specified in the following format:

```
ojds://jdbc_connection_url/path…/object
```

In the preceding syntax:

- `jdbc_connection_url` is one of the supported JDBC connection URLs. You must specify the `jdbc_connection_url` in the URL to connect to the directory.

Note:

The OJDS provider supports both the thin and OCI URLs for a JDK-based external client. For example, you can use the following URLs for thin driver and OCI driver respectively:

```
thin:localhost:5521:mysid
oci:22.133.242:5521:mysid
```

However, OJDS URL support in the server is only for thin connection type. You must set a value for `Context.SECURITY_PRINCIPAL` and `Context.SECURITY_CREDENTIALS` to complete the URL connection.

- `path` is a slash[1]-separated list similar to a Unix type file system. This represents nodes in the Directory tree.
- `object` is the actual terminal object name in the context. If the object is omitted, then the path terminates in a slash (/). In such a case, a `DirContext` is returned representing this path as the root.

Example

The following code snippet shows how to look up for the object `myobj` of type `MyObj` in the directory `/one/two` using the OCI driver connected as user `HR`:

```
import javax.naming.*;
.
.
.
Hashtable env = new Hashtable();
env.put(“java.naming.security.principal",“HR");
@ env.put(“java.naming.security.credentials","<password>");
MyObj obj = (MyObj)(new InitialDirContext(env)).lookup(ojds://
oci:host1:5521:mysid/one/two/myobj");
```

11.2.2.5 Client classpath

You must add the `$ORACLE_HOME/jdbc/lib/ojdbc6.jar` and `$ORACLE_HOME/javavm/lib/aurora.zip` jar files to the classpath for a JDK client to use the OJDS.

11.2.3 Namespace Browser

The namespace browser enables browsing permissions and properties of objects stored in the OJDS. The existing `ojvmjava` utility is enhanced to support the operations as described in the following table:

Command Name	Description
`ls`	Lists the contents of a context similar to Unix `ls` command.
`rm`	Removes the context or an object.
`mkdir`	Creates a context in the OJDS.
`chown`	Changes the owner of the given context, object, and so on.

[1] The slash symbol (\)

Command Name	Description
`chmod`	Changes rights on objects or contexts.
`cd`	Changes the working context.
`pwd`	Lists the current working context.
`ln`	Refers to the same object by using different names, similar to a symbolic link in Unix.
`mv`	Changes or rebinds old names of a context (or object), to a new name.
`bind`	Binds an object reference or naming context into the JNDI namespace.
`bindds`	Binds a Data Source object to a given context.
`bindurl`	Binds a URL object to the given context.

Related Topics

- The ojvmjava Tool

11.3 OJDS Command-Line Tools

The enhanced `ojvmjava` commands enable you to manipulate and browse the OJDS. This section describes the following commands:

- ls Command
- cd Command
- pwd Command
- chown Command
- mkdir Command
- rm Command
- ln Command
- mv Command
- chmod Command
- bind Command
- bindds Command
- bindurl Command

11.3.1 ls Command

The `ls` command displays the contents of a context.

Syntax

```
ls [options] [context1] [context2] [obj|context]...
```

Options

The following table describes the `ls` command options:

Option	Description
`Context obj`	Specifies the name of the context or object to be listed
`-l`	Shows contents in long format including name, creation time, owner, rights, and so on. Shows the class of an object.
`dir`	Shows only contexts, similar to the Unix `ls -d` command
`ldir`	Shows contents in long format like the `-l` command, but ignores bound objects, similar to Unix –ld
`R`	Recursively lists though child contexts

Example

The following command displays contents of the root Context in short format:

```
$ ls /

etc/      public/
```

The following command displays contents of the root Context in long format:

```
$ ls -l
Read      Write    Exec        Owner    Date       Time      Type       Name
PUBLIC    SYS      PUBLIC      SYS      Dec 14     14:59     Context    etc
PUBLIC    PUBLIC   PUBLIC      SYS      Dec 14     14:59     Context    public
```

11.3.2 cd Command

The `cd` command changes the working context. This command is similar to the Unix `cd` command to change directories.

Example

The following command changes the context to root Context

```
$ cd /
```

11.3.3 pwd Command

The pwd command lists the current working context.

Example

If the current context is `/test/alpha`, then the output of the `pwd` command is the following:

```
$ pwd

/test/alpha/
```

11.3.4 chown Command

The `chown` command changes the ownership of the context or the object.

Note:

You can change ownership of a context or an object only if you are the `SYS` user.

Syntax

```
chown [options] {user | role} <object name>
```

Options

The following table describes the `chown` command options:

Option	Description
`User role`	Specifies the name of the user or role to become the owner
`<object name>`	Specifies the name of the context or object to be changed
`-R`	Recursively changes ownership of the following: • Context • All the subcontexts in the context • All the objects that are contained in the context and subcontexts

Example

The following command makes `HR` the owner of the `/alpha/beta/gamma` command:

```
$ chown HR /alpha/beta/gamma
```

11.3.5 mkdir Command

The `mkdir` command creates a context with the given name.

Note:

You must have the Write permission for the target context to create the new context.

Options

The following table describes the `mkdir` command options:

Option	Description
`<name>`	Specifies the name of the context to be created
`-path | -p`	Creates intermediate contexts if they do not exist

Example

The following command creates a Context called `/test/alpha`, where the `/test` context exists already:

```
mkdir /test/alpha
```

The following command creates a Context called `/test/alpha/beta/gamma`, where the `/test/alpha/beta` context does not exist:

```
$ mkdir -path /test/alpha/beta/gamma
```

11.3.6 rm Command

The `rm` command is analogous to the `rm` UNIX shell command. This command removes an object or a context, including its contents.

Note:

To remove an object, you must have the Write right for the context that contains the object.

Options

The following table describes the `rm` command options:

Option	Description
`<Object>`	Specifies the name of the context or the object to be removed
`-recurse | -r`	Assumes a context and removes the context and its contents recursively

Examples

The following command removes the object `/test/bank`:

```
rm /test/bank
```

The following command removes the context `/test/release3` and its contents:

```
rm -r /test/release3
```

11.3.7 ln Command

The `ln` command is analogous to the UNIX `ln` command. A link is a synonym for a context or an object. When you move a context or an object, the reference to the context or object may become invalid. The `ln` command creates a link with the old name and makes the object accessible by both the old and the new name.

Syntax

```
ln [-symbolic | -s] <object> <link>
```

Options

The following table describes the `ln` command options:

Option	Description
`-s`	Create a soft link of `<object>` to `<link>`
`<object>`	Specifies the name of a context or an object

Option	Description
`<link>`	Specifies the name of the synonym to which you link a context or an object

Example

The following command preserves access to the `old` object even after the name of the object is changed to `new`:

```
$ mv old new
$ ln new old
```

11.3.8 mv Command

The `mv` command changes the name (or rebinds old names) of a context or an object to a new name.

Syntax

```
mv <old> <new>
```

Example

The following command changes the name of the context `/test/foo` to `/test/bar`:

```
$ mv /test/foo /test/bar
```

11.3.9 chmod Command

The `chmod` command is analogous to the `chmod` UNIX shell command. This command changes the rights of a user or a role on a context or an object.

Note:

You can change the rights on an object only if you are the `SYS` user or the owner of the object.

Syntax

```
chmod [options] {+|-} {r|w|x} {<user> | <role>, ...} <objectname>
```

Options

The following table describes the `chmod` command options:

Option	Description
`+/-rwx`	Add (+) or remove (-) read, write, or execute
`<user> | <role>`	The user or role whose rights are added or removed
`<objectname>`	The context or object for which rights are changed
`-R`	Changes rights recursively

Example

The following example changes rights for the `/alpha/beta/gamma` context to `HR` and `NANCY`:

```
$ chmod +x HR,NANCY /alpha/beta/gamma
```

Note:

The schemas are separated by only a comma.

The following example removes the Write rights of `HR` for the `/alpha/beta/gamma` context:

```
$ chmod -w HR /alpha/beta/gamma
```

Related Topics

- Object permissions

11.3.10 bind Command

The `bind` command binds an object reference or context into the JNDI namespace.

Syntax

```
bind <objectname> [options] [-context] [-rebind] {-class | -c <classname>} [-
factory | -f <factory>] [-location | -l <URL>][-string <type_name>

<string_value> [-string <type_name> <string_value> ...]] [-binary <type_name>
<string_value> [-binary <type_name> <string_value> ...]]
```

Options

The following table describes the `bind` command options:

Option	Description
`<objectname>`	Name object is to be bound to
`-context`	The object to be bound is a Context or InitialContext
`-rebind`	If the JNDI name already exists, replaces the object that it is bound to with this object
`-class <classname>`	Specifies the class name for the bound object
`-factory <factory>`	Specifies the factory class name for creating the object. JNDI uses this for creating the object.
`-location <URL>`	Specifies the factory location if the default location is not used. This takes a JNDI URL.
`-string <type_name> <string_value>`	Specifies a String reference attribute for the object by the type name and value.
`-binary <type_name> <string_value>`	Specifies a Binary reference attribute for the object by the type and a binary value. The given Hexidecimal string value is converted into binary.

Example

The following binds an object reference into the name space. A string and binary attribute is supplied to the reference.

```
bind /tmp/myprinter  -class gen.Inkjet  -factory gen.InkjetFactory  -string PRINTERNAME
co2  -binary DPI  0X12C
```

11.3.11 bindds Command

This command binds a `DataSource` object in the JNDI namespace. This command binds general, XA, or pooled data sources depending on specified options.

> **Note:**
>
> Oracle JVM supports only kprb drivers and thin drivers.

Syntax

```
bindds <object_name> [options] [-help | -h] [-describe | -d] [-version | -v] [-
dstype <datasource>]

[-host <hostname> -port <portnum> -sid <SID> -driver <driver_type>] [-url
<db_url>]

[-g | -grant {<user> | <role>} [,{<user> | <role>}]...] [-recursiveGrant | -rg
{<user> |<role>}

[,{<user> | <role>}]...] [-rebind] [-user | -u <user>]
```

Options

The following table describes the `bindds` command options:

Option	Description
`<objectname>`	Specifies the name to which the object is to be bound
`-help`	Specifies the help message
`-describe`	Summarizes the tools operation
`-version`	Specifies the version number
`-dstype <type>`	Specifies the data source type from one of the following types: • None for `OracleDataDource` • `xa` for `OracleXADatasource` • `pool` for `OracleConnectionPoolDataSource`
`-host <hostname> -port <portnum> -sid <SID> -driver <drv_type>`	Specify the location of the database and driver type for the connection to the database. You can alternatively specify this information in a URL format within the `-url` option. The default value for the `-sid` option is `ORCL`. The `–driver` option can have the following values: `thin`, `oci`, or `kprb`.
`-url <db_url>`	This JDBC URL specifies the location of the database.

Option	Description
`-grant <user|role>, <user|role>…`	Grants Read and Execute rights to the sequence of `<user>` and `<role>` names. When rebinding, replace the existing users or roles that have read or execute rights with the `<user>` and `<role>` names.
`-recursiveGrant <user|role>, <user|role>….`	Recursively grants Read and Execute permission to the designated object and all the contexts in which the object exists. If the context has a permission level of SYS, the grant for that context is ignored.
`-rebind`	If the `DataSource` object already exists, then you must specify this option to overwrite the existing data source with this new object. Otherwise, no bind occurs for this option.
`-user <user>`	Specifies the user name for connecting to the database. Stores the user name within the `DataSource` object. If you do not supply a user name within the JNDI Context while creating the database connection, then this user name is used.

Example

The following example binds the `ds1` data source into the namespace:

```
bindds /test/ds1 -host localhost -port 1522 -sid orcl -driver thin
bindds /test/ds1 -url jdbc:oracle:thin:@localhost:1522:orcl
```

The example uses the JDBC thin driver with a general data source, that is, `OracleDataSource`.

11.3.12 bindurl Command

The `bindurl` command binds a URL object in the namespace.

Options

The following table describes the `bindurl` command options:

Option	Description
`<objectname>`	Specifies the name of the object to be bound
`-help`	Specifies the help message
`-describe`	Summarizes the tools operations
`-version`	Prints the version of the `bindurl` command
`-rebind`	If the JNDI name already exists, then you must specify this option to overwrite the existing JNDI name with this new object. Otherwise, no bind occurs for this option.
`-grant <user|role>, <user|role>…`	Grants Read and Execute rights to the sequence of `<user>` and `<role>` names. When rebinding, you can replace the existing users or roles that have read or execute rights with the `<user>` and `<role>` names.
`-recursiveGrant <user|role>, <user|role>...`	Recursively grants Read and Execute permission to the designated object and to all contexts within which the object exists. If the context has a permission level of `SYS`, then the grant for that context is ignored.

Example

The following example binds the URL string `http://www.oracle.com` to a URL reference `/test/myURL` within the namespace:

```
bindurl /test/myURL http://www.oracle.com -rebind
```

The `-rebind` option is used to make sure that if the `/test/myURL` reference previously exists, then it is rebound with the string `http://www.oracle.com`.

11.4 OJDS APIs and Classes

This section describes the following OJDS APIs and classes:

- oracle.aurora.jndi.ojds.OjdsClientContext
- oracle.aurora.jndi.ojds.OjdsServerContext
- oracle.aurora.jndi.ojds.OjdsInitialContextFactory
- oracle.aurora.jndi.ojds.OjdsURLContextFactory
- oracle.aurora.jndi.ojds.OjdsURLContext

11.4.1 oracle.aurora.jndi.ojds.OjdsClientContext

This class implements the `javax.naming.directory.DirContext` interface. It establishes connection with the database and performs all functions required to support the OJDS. It supports all the methods described in `[DirContext]` except the following methods:

- `getSchema`
- `getSchemaClassDefintion`
- `modifyAttributes`
- `search`

This class is created automatically when an `InitialDirContext` is created on a JAVA JDK based client. It provides the communication and object transport between the client application and the OJDS.

You must set the following JNDI properties to specific values to complete a connection:

- `java.naming.factory.initial` to `oracle.aurora.jndi.ojds.OjdsIntialContextFactory`
- `java.naming.security.principal` to the name of the connection schema
- `java.naming.security.credentials` to the schema password
- `java.naming.provider.url` to a valid OJDS URL

You can set these properties as shown in the following code snippet:

```
Hashtable env = new Hashtable();
env.put("java.naming.factory.initial",
"oracle.aurora.jndi.ojds.OjdsInitialContextFactory");
env.put(Context.SECURITY_PRINCIPAL, "HR");
env.put(Context.SECURITY_CREDENTIALS, "<password>");
env.put(Context.PROVIDER_URL,"ojds://thin:localhost:5521:j3");
```

11.4.2 oracle.aurora.jndi.ojds.OjdsServerContext

This class implements the `javax.naming.directory.DirContext` interface. It uses the internal database connection to communicate with the OJDS persistent store. It supports all the methods described in `[DirContext]` except for the following methods:

- getSchema
- getSchemaClassDefintion
- modifyAttributes
- search

This class is created automatically when an `InitialDirContext` is created in a database resident application. It uses the database internal JDBC driver to communicate with the OJDS persistent store.

The four environment properties for the `OjdsClientContext` are ignored for `OjdsServerContext` because the application runs as the `login` schema. The connection is made with the kprb [JDBC] internal driver. If the Java stored procedure requires access outside the server, then you must use the OJDS `URLContext` as the value of the `java.naming.provider.url` property.

Related Topics

- oracle.aurora.jndi.ojds.OjdsInitialContextFactory

11.4.3 oracle.aurora.jndi.ojds.OjdsInitialContextFactory

This class implements the `javax.naming.spi.InitialContextFactory` interface. The JNDI `InitialContext` or `InitialDriContext` classes create either an `OjdsClientContext` or an `OjdsServerContext` depending on the execution environment.

11.4.4 oracle.aurora.jndi.ojds.OjdsURLContextFactory

This class supports OJDS style URLs. Depending on the method provided to the URL, this method can return a `DirContext` or an instance of an object stored in the OJDS.

11.4.5 oracle.aurora.jndi.ojds.OjdsURLContext

This class is an extension of the `oracle.aurora.jndi.ojds.OjdsClientContext.` It supports extraction of connection information from an OJDS URL and making a connection to the OJDS. It supports the same interfaces as `oracle.aurora.jndi.ojds.OjdsClientContext` class.

You must set the following parameters to use the `OjdsURLContext` class:

- `javax.naming.security.principal` to the connection schema
- `javax.naming.security.credentials` to the password of the connection schema
- `javax.naming.factory.initial` to `oracle.aurora.jndi.ojds.OjdsInitialContextFactory`

You can set these properties as shown in the following code snippet:

```
Hashtable env = new Hashtable();
env.put("java.naming.factory.initial",
"oracle.aurora.jndi.ojds.OjdsInitialContextFactory");
env.put(Context.SECURITY_PRINCIPAL, "HR");
env.put(Context.SECURITY_CREDENTIALS, "<password>");
DirContext dir =
(new InitialContext(env)).lookup("ojds://thin:localhost:5521:j3/public./mydir");
```