4
Java Installation and Configuration

This chapter describes how to install and configure Oracle JVM. It also describes how to enable the Java client. This chapter covers the following topics:

- About Initializing a Java-Enabled Database
- Configuring Oracle JVM
- The DBMS_JAVA Package
- Enabling the Java Client
- Two-Tier Duration for Java Session State
- About Setting System Properties

4.1 Initializing a Java-Enabled Database

If you install Oracle Database with Oracle JVM option, then the database is Java-enabled. That is, it is ready to run Java stored procedures and Java Database Connectivity (JDBC).

This section contains the following topics:

- Configuring the Oracle JVM Option within the Oracle Database Template
- Modifying an Existing Oracle Database to Include Oracle JVM

4.1.1 Configuring the Oracle JVM Option within the Oracle Database Template

Configure Oracle JVM option within the database template. This is the recommended method for Java installation.

The Database Configuration Assistant enables you to create database templates for defining what each database instance installation will contain. Choose Oracle JVM option to have the Java platform installed within your database.

4.1.2 Modifying an Existing Oracle Database to Include Oracle JVM

If you have already installed Oracle Database without Oracle JVM, then you can add Java to your database through the modify mode of the Database Configuration Assistant of Oracle Database. The modify mode enables you to choose the features, such as Oracle JVM, that you would like to install on top of an existing Oracle Database instance.

4.2 Configuring Oracle JVM

Before you install Oracle JVM as part of your standard Oracle Database installation, you must ensure that the configuration requirements for Oracle JVM are fulfilled. The main configuration for Java classes within Oracle Database includes configuring the:

- Java memory requirements

You must have at least 50 MB of `JAVA_POOL_SIZE` and 96 MB of `SHARED_POOL_SIZE`.

> **Note:**
>
> Oracle recommends that you increase the `JAVA_POOL_SIZE` and `SHARED_POOL_SIZE` values when using large Java applications, or when a large number of users are running Java in the database.

- Database processes

  You must decide whether to use dedicated server processes or shared server processes for your database server.

> **Note:**
>
> Oracle recommends that you use dedicated servers. Shared server incurs extra Java states save in the database session, in SGA, at the end of the Java call.

Related Topics

- About Java Memory Usage

4.3 The DBMS_JAVA Package

Installing Oracle JVM creates the `DBMS_JAVA` PL/SQL package. The `DBMS_JAVA` package functions can be used by both Database server and Database clients. The corresponding Java class, `DbmsJava`, provides methods for accessing database functionality from Java.

Related Topics

- DBMS_JAVA Package

4.4 Enabling the Java Client

To run Java between the client and server, your must perform the following activities:

1. Installing Java SE on the Client
2. Setting Up Environment Variables

4.4.1 Installing Java SE on the Client

The client requires Java Development Kit (JDK) 11 or later. To confirm the version of JDK you are using, run the following commands on the command line:

```
$ which java
/usr/local/jdk11.0.18/bin/java

$ which javac
/usr/local/jdk11.0.18/bin/javac

$ java -version
java version "11.0.18"
```

4.4.2 Setting Up Environment Variables

After installing JDK on your client, add the directory path to the following environment variables:

- `$JAVA_HOME`

  This variable must be set to the top directory of the installed JDK base.

- `$PATH`

  This variable must include `$JAVA_HOME/bin`.

- `$LD_LIBRARY_PATH`

  This variable must include `$JAVA_HOME/lib`.

JAR Files Necessary for Java Clients

To ensure that the Java client successfully communicates with the server, include the following files in the `CLASSPATH`:

> **Note:**
>
> Specifics of `CLASSPATH` requirements may vary for Oracle JVMs running on different platforms. You must ensure that all elements of `CLASSPATH`, as defined in the script for Oracle JVM utilities, are present.

- For JDK 8, include `$JAVA_HOME/lib/dt.jar`
- For JRE 8, include `$JAVA_HOME/lib/rt.jar`
- For any interaction with JDBC, include `$ORACLE_HOME/jdbc/lib/ojdbc8.jar`
- For any client that uses SSL, include `$ORACLE_HOME/jlib/jssl-1_2.jar` and `$ORACLE_HOME/jlib/javax-ssl-1_2.jar`
- For any client that uses the Java Transaction API (JTA) functionality, include `$ORACLE_HOME/jlib/jta.jar`
- For any client that uses the Java Naming and Directory Interface (JNDI) functionality, include `$ORACLE_HOME/jlib/jndi.jar`

Server Application Development on the Client

If you develop and compile your server applications on the client and want to use the same Java Archive (JAR) files that are loaded on the server, then include `$ORACLE_HOME/lib/aurora.zip` in `CLASSPATH`. This is not required for running Java clients.

4.5 Two-Tier Duration for Java Session State

Java session state is split into two tiers. One tier has a longer duration and it encompasses the duration of the other tier. The duration of the shorter tier is the same as before, that is, it starts when a Java method is invoked and ends when JVM exits. The duration of the longer tier starts when a Java method is invoked in the RDBMS session for the first time. This session lasts until the RDBMS session ends or the session is explicitly terminated by a call to the function

`dbms_java.endsession_and_related_state`. This is addressed by the addition of the following two PL/SQL functions to the `DBMS_JAVA` package, which account for the two kinds of Java session duration:

- `FUNCTION endsession RETURN VARCHAR2;`

  This function clears any Java session state remaining from previous execution of Java in the current RDBMS session. The return value is a message indicating the action taken.

- `FUNCTION endsession_and_related_state RETURN VARCHAR2;`

  This function clears any Java session state remaining from previous execution of Java in the current RDBMS session and all supporting data related to running Java, such as property settings and output specifications. The return value is a message indicating the action taken.

Most of the values associated with running Java remain in the shorter tier. The values that can be useful for multiple invocations of JVM have been moved to the longer tier. For example, the system property values established by `dbms_java.set_property` and the output redirection specifications.

Related Topics

- About Setting System Properties
- About Redirecting Output on the Server

4.6 About Setting System Properties

Within an RDBMS session you can maintain a set of values that are added to the system properties whenever a Java session is started in the RDBMS session. This set of values remains valid for the duration of the longer tier of Java session state, which is typically the same as the duration of the RDBMS session.

There is a set of PL/SQL functions in the `DBMS_JAVA` package for setting, retrieving, removing and displaying key value pairs in an internal, RDBMS session duration table, where both elements of a pair are strings (`VARCHAR2`) and there is at most one pair for a given key. These functions are as follows:

- set_property
- get_property
- remove_property
- show_property

set_property

This function establishes a value for a system property that is then used for the duration of the current RDBMS session, whenever a Java session is initialized. The first argument is the name of the property and the second is the value to be established for it. The return value for `set_property` is null unless there is some error. For example, if an attempt is made to set a value for a prescribed property, then an error message is returned.

```
FUNCTION set_property(name VARCHAR2, value VARCHAR2) RETURN VARCHAR2;
```

get_property

This function returns any value previously established by `set_property`. It returns null if there is no such value.

```
FUNCTION get_property(name VARCHAR2) RETURN VARCHAR2;
```

remove_property

This function removes any value previously established by `set_property`. The return value is null unless an error occurred, in which case an error message is returned.

```
FUNCTION remove_property(name VARCHAR2) RETURN VARCHAR2;
```

show_property

This function displays a message of the form `name = value` for the input name, or for all established property bindings, if name is null. The return value for this function is null on successful completion, otherwise it is an error message. The output is displayed to wherever you have currently directed your Java output.

```
FUNCTION show_property(name VARCHAR2) RETURN VARCHAR2;
```

Before initializing the Java session, the values from this table are added to the set of default system property values already maintained by Oracle JVM. When you run a Java method by using the command-line interface, the values determined by the `-D` option, if present, override the values set in the table. As soon as you terminate the Java session, the values established by the `-D` option become obsolete and the keys are set to the original values as present in the table.

Related Topics

- Two-Tier Duration for Java Session State

4.6.1 Oracle JVM-Specific System Properties

Beginning with Oracle Database Release 23ai, Oracle JVM recognizes a few more additional system properties to accommodate support for Java modules.

This section describes these properties:

oracle.aurora.addmods

The value of this property is a comma-delimited list of modules to be added to the upcoming Oracle JVM session. You must specify this property prior to any use of Java in the RDBMS session, that is, prior to the start of the Oracle JVM session. The use of this property is roughly equivalent to specifying the `--add-modules` option of the client-side Java command.

oracle.aurora.addmods.from

The value of this property is a comma-delimited list of classes, whose associated modules are to be added to the upcoming Oracle JVM session. You must specify this property prior to any use of Java in the RDBMS session, that is, prior to the start of the Oracle JVM session.

If a class in the list is modularized itself, then you must specify the name of the class as `<module_name///<class_name>`. For example, if class C is in module M1, and module M2 is added to class C by `loadjava`, then, if the system property `oracle.aurora.addmods.from` is specified as `M1///C`, modules M1 and M2 both are added to the root set of modules, when the upcoming Oracle JVM session is initialized.

Using the `loadjava --add-modules` option provides better Oracle JVM session start-up performance than setting session properties. Adding modules with system properties also disables the use of any module hotloading that is specified when the module was loaded by `loadjava`.