7

Server-Side Internal Driver

This chapter covers the following topics:

7.1 Overview of the Server-Side Internal Driver

The server-side internal driver is intrinsically tied to Oracle Database and to the embedded Java Virtual Machine, also known as Oracle Java Virtual Machine (Oracle JVM). The driver runs as part of the same process as the Database. It also runs within the default session, the same session in which the Oracle JVM was started. Each Oracle JVM session has a single implicit native connection to the Database session in which it exists. This connection is conceptual and is not a Java object. It is an inherent aspect of the session and cannot be opened or closed from within the JVM.

The server-side internal driver is optimized to run within the database server and provide direct access to SQL data and PL/SQL subprograms on the local database. The entire JVM operates in the same address space as the database and the SQL engine. Access to the SQL engine is a function call. This enhances the performance of your Java Database Connectivity (JDBC) applications and is much faster than running a remote Oracle Net call to access the SQL engine.

The server-side internal driver supports the same features, application programming interfaces (APIs), and Oracle extensions as the client-side drivers. This makes application partitioning very straightforward. For example, if you have a Java application that is data-intensive, then you can easily move it into the database server for better performance, without having to modify the application-specific calls.

7.2 Connecting to the Database

As described in the preceding section, the server-side internal driver runs within a default session. Therefore, you are already connected. There are two methods to access the default connection:

- Use the `OracleDataSource.getConnection` method, with any of the following forms as the URL string:
  - `jdbc:oracle:kprb`
  - `jdbc:default:connection`
  - `jdbc:oracle:kprb:`
  - `jdbc:default:connection:`

- Use the Oracle-specific `defaultConnection` method of the `OracleDriver` class.

Using `defaultConnection` is generally recommended.

> **Note:**
>
> You are no longer required to register the `OracleDriver` class for connecting with the server-side internal driver.

Connecting with the OracleDriver Class defaultConnection Method

The `defaultConnection` method of the `oracle.jdbc.OracleDriver` class is an Oracle extension and always returns the same connection object. Even if you call this method multiple times, assigning the resulting connection object to different variable names, then only a single connection object is reused.

You need *not* include a connection string in the `defaultConnection` call. For example:

```
import java.sql.*;
import oracle.jdbc.*;

class JDBCConnection
{
  public static Connection connect() throws SQLException
  {
    Connection conn = null;
    try {
      // connect with the server-side internal driver

         conn = ora.defaultConnection();
      }

    } catch (SQLException e) {...}
    return conn;
  }
}
```

Note that there is no `conn.close` call in the example. When JDBC code is running inside the target server, the connection is an implicit data channel, not an explicit connection instance as from a client. It should *not* be closed.

OracleDriver has a static variable to store a default connection instance. The method `OracleDriver.defaultConnection` returns this default connection instance if the connection exists and is not closed. Otherwise, it creates a new, open instance and stores it in the static variable and returns it to the caller.

Typically, you should use the `OracleDriver.defaultConnection` method. This method is faster and uses less resources. Java stored procedures should be carefully written. For example, to close statements before the end of each call.

Typically, you should not close the default connection instance because it is a single instance that can be stored in multiple places, and if you close the instance, each would become unusable. If it is closed, a later call to the `OracleDriver.defaultConnection` method gets a new, open instance.

Connecting with the OracleDataSource.getConnection Method

To connect to the internal server connection from code that is running within the target server, you can use the `OracleDataSource.getConnection` method with either of the following URLs:

```
OracleDataSource ods = new OracleDataSource();
ods.setURL("jdbc:oracle:kprb");
Connection conn = ods.getConnection();
```

or:

```
OracleDataSource ods = new OracleDataSource();
ods.setURL("jdbc:default:connection");
Connection conn = ods.getConnection();
```

Any user name or password you include in the URL is ignored in connecting to the default server connection.

The `OracleDataSource.getConnection` method returns a new Java `Connection` object every time you call it. The fact that `OracleDataSource.getConnection` returns a new connection object every time you call it is significant if you are working with object maps or type maps. A type map is associated with a specific `Connection` object and with any state that is part of the object. If you want to use multiple type maps as part of your program, then you can call `getConnection` to create a new `Connection` object for each type map.

> **Note:**
>
> Although the `OracleDataSource.getConnection` method returns a new object every time you call it, it does not create a new database connection every time. They all utilize the same implicit native connection and share the same session state, in particular, the local transaction.

7.3 About Session and Transaction Context

The server-side driver operates within a default session and default transaction context. The default session is the session in which the JVM was started. In effect, you are already connected to the database on the server. This is different from the client-side where there is no default session. You must explicitly connect to the database.

Auto-commit mode is disabled in the server. You must manage transaction `COMMIT` and `ROLLBACK` operations explicitly by using the appropriate methods on the connection object:

```
conn.commit();
```

or:

```
conn.rollback();
```

> **Note:**
>
> As a best practice, it is recommended not to commit or rollback a transaction inside the server.

7.4 Testing JDBC on the Server

Almost any JDBC program that can run on a client can also run on the server. All the programs in the `samples` directory can be run on the server, with only minor modifications. Usually, these modifications concern only the connection statement.

Consider the following code fragment which obtains a connection to a database:

```
ods.setUrl(
"jdbc:oracle:oci:@(DESCRIPTION=
  (ADDRESS=(PROTOCOL=TCP)(HOST=cluster_alias)
    (PORT=5221))
    (CONNECT_DATA=(SERVICE_NAME=orcl)))");
ods.setUser("HR");
ods.setPassword("hr");
Connection conn = ods.getConnection();
```

We can modify this code fragment for use in the server-side internal driver. In the server-side internal driver, no user, password, or database information is necessary. For the connection statement, you use:

```
ods.setUrl(
"jdbc:oracle:kprb:@");
Connection conn = ods.getConnection();
```

However, the most convenient way to get a connection is to call the `OracleDriver.defaultConnection` method, as follows:

```
Connection conn = OracleDriver.defaultConnection();
```

7.5 Loading an Application into the Server

When loading an application into the server, you can load `.class` files that you have already compiled on the client or you can load `.java` source files and have them automatically compiled on the server.

7.5.1 Using the Loadjava Utility

You can use the `loadjava` utility to load your files. You can either specify source file names on the command line or put the files into a Java Archive (JAR) file and specify the JAR file name on the command line.

The `loadjava` script, which runs the actual utility, is in the `bin` directory in your Oracle home. This directory should already be in your path once Oracle has been installed.

> **Note:**
>
> The `loadjava` utility supports compressed files.

Loading Class Files into the Server

Consider a case where you have the following three class files in your application: `Foo1.class`, `Foo2.class`, and `Foo3.class`. Each class is written into its own class schema object in the server.

You can load the class files using the default JDBC Oracle Call Interface (OCI) driver in the following ways:

- Specifying the individual class file names, as follows:

```
loadjava -user HR Foo1.class Foo2.class Foo3.class
Password: password
```

- Specifying the class file names using a wildcard, as follows:

```
loadjava -user HR Foo*.class
Password: password
```

- Specifying a JAR file that contains the class files, as follows:

```
loadjava -user HR Foo.jar
Password: password
```

You can load the files using the JDBC Thin driver, as follows:

```
loadjava -thin -user HR@localhost:5221:orcl Foo.jar
Password: password
```

> **Note:**
>
> Starting from Oracle Database 12*c* Release 1 (12.1), JDK 6, and JDK 7 are supported. However, only one of the JVMs will be active at a given time.
>
> Ensure that your classes are not compiled using a newer version of JDK than the active runtime version on the server.

Loading Source Files into the Server

If you enable the `loadjava -resolve` option when loading a `.java` source file, then the server-side compiler will compile your application as it is loaded, resulting in both a source schema object for the original source code and one or more class schema objects for the compiled output.

If you do not specify `-resolve`, then the source is loaded into a source schema object without any compilation. In this case, however, the source is implicitly compiled the first time an attempt is made to use a class defined in the source.

For example, run `loadjava` as follows to load and compile `Foo.java`, using the default JDBC OCI driver:

```
loadjava -user HR -resolve Foo.java
Password: password
```

Or, use the following command to load using the JDBC Thin driver:

```
loadjava -thin -user HR@localhost:5221:orcl -resolve Foo.java
Password: password
```

Either of these will result in appropriate class schema objects being created in addition to the source schema object.

> **Note:**
>
> Oracle generally recommends compiling source on the client, whenever possible, and loading the `.class` files instead of the source files into the server.

> **See Also:**
>
> *Oracle Database Java Developer's Guide*

7.5.2 Using the JVM Command Line

You can also use the JVM command-line option to load your files. The command-line interface to Oracle JVM is analogous to using the JDK or JRE shell commands. You can:

- Use the standard `-classpath` syntax to indicate where to find the classes to load
- Set the system properties by using the standard `-D` syntax

The interface is a PL/SQL function that takes a string (`VARCHAR2`) argument, parses it as a command-line input and if it is properly formed, runs the indicated Java method in Oracle JVM. To do this, PL/SQL package `DBMS_JAVA` provides the following functions:

- `runjava`

  You can use the `runjava` function in the following way:

  ```
  FUNCTION runjava(cmdline VARCHAR2) RETURN VARCHAR2;
  ```

- `runjava_in_current_session`

  You can use the `runjava_in_current_session` function in the following way:

  ```
  FUNCTION runjava_in_current_session(cmdline VARCHAR2) RETURN VARCHAR2;
  ```

> **Note:**
>
> Starting with Oracle Database 11*g* Release 1, there is a just-in-time (JIT) compiler for Oracle JVM environment. A JIT compiler for Oracle JVM enables much faster execution because the JIT compiler uses advanced techniques as compared to the old Native compiler and compiles dynamically generated code. Unlike the old Native compiler, the JIT compiler does not require a C compiler. It is enabled without the support of any plug-ins.