Debugging PL/SQL and Java Stored Programs with JPDA

Christian Antognini
Trivadis AG
Zürich, Switzerland

Introduction

Once upon a time the only way to debug a program was to fill it with tracing instructions that send messages to the standard output or to a log file. For many years there has been a better, more efficient and more flexible way that helps a developer to find a bug: to use a debugger. Unfortunately too many developers are still using the old methods and, therefore, losing a lot of time debugging programs.

The aim of this paper is to describe a new debugging method, based on JPDA, available as of Oracle9*i* Release 2 and integrated in JDeveloper9*i*. This method enables seamless and remote debugging of PL/SQL and Java programs stored and run in the database.

Below I describe the operations or requirements necessary for a seamless debug of both programming languages. If only one of them is used, not all requirements or operations are necessary.

What Is JPDA?

The Java Platform Debugger Architecture (JPDA) is a Java multi-tiered debugging architecture that allows tools developers to create debugger applications which run portably across platforms, virtual machine (VM) implementations and SDK versions. It consists of the following three layers (Figure 1):

Figure 1 JDPA Overview

- Java Debug Interface (JDI) – Defines a high-level Java language interface which tool developers can use to write remote debugger applications which are usually part of an Integrated Development Environment (IDE).
- Java Debug Wire Protocol (JDWP) – Defines the communication between debuggee and debugger processes.
- Java VM Debug Interface (JVMDI) – Defines the debugging services a VM provides.

A Debugging Session Overview

Before looking at the details it is important to understand how a debugging session in an Oracle environment works and to know which components are involved. It is also important to note that also if JPDA has been specified for Java, Oracle uses it for the debugging of the PL/SQL code as well.

There are three main components (Figure 2):

- JDeveloper – The IDE that implements JDI. It communicates with the database via JDWP and Oracle Net.
- The Oracle9*i* Release 2 database – The remote VM that implements JVMDI and where the debugged stored program run.
- Client application – Any application (e.g. SQL*Plus, a J2EE container, a job started via DBMS_JOB, …) that uses a PL/SQL or Java stored program. It communicates with the database via Oracle Net.

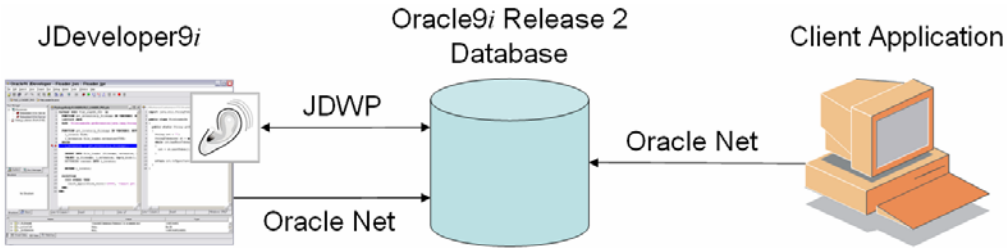

Figure 2 Components in an Oracle debugging environment

The main steps performed in debugging a stored program are the following:

1. In JDeveloper a project containing the Java classes to be debugged have to be configured. The PL/SQL programs need not reside on the development client; in fact they are directly loaded from the database via the Oracle Net connection.
2. In JDeveloper some break points should be defined. Of course you can also define them when the application is already running.
3. In JDeveloper a JPDA listener must be configured and started.
4. The client application is started and opens a connection to the database; a database session is created. By default a session is not in debug modus, therefore it has to be explicitly enabled. When it is enabled, the host where JDeveloper is running and the port where the listener is hearing must be specified. Thus the database can inform JDeveloper that a session is executing in debug modus.
5. When the client application executes a PL/SQL or Java stored program the debugging events are sent to JDeveloper. E.g. if a break point is reached the program is stopped and the control is passed to the IDE. Then the user can step through the code and/or inspect the variables to understand how the program behaves.

It is important to notice that JDeveloper does not start the client application but only waits for the JPDA events. Therefore the three components can reside on separate systems.

Requirements

JPDA has been introduced in the Java SDK version 1.3. Since JServer (the internal Oracle VM) uses this SDK as of Oracle9*i* Release 2 only, JPDA cannot be used in older releases. However to debug only PL/SQL code, no JServer needs to be installed in the database! Other debugging methods are also available in older releases, but it is not the aim of this paper to describe them.

To use JPDA with JDeveloper9*i* I suggest that, for stability reasons, you to use version 9.0.3.x.

To debug a program stored in the database the user that connects it must have the system privileges DEBUG CONNECT SESSION and DEBUG ANY PROCEDURE. Notice that the object privilege DEBUG, which is documented in “Oracle9*i* SQL Reference Release 2”, is not implemented yet.

The PL/SQL and Java stored programs must be compiled in debug modus. The next two sections explain how to do that.

Compiling PL/SQL in Debug Modus

To compile a PL/SQL stored program in debug modus there are basically two possibilities:

- Recompile it with the statement “ALTER <program> COMPILE DEBUG”.
- Before creating or recompiling it set the parameter PLSQL_DEBUG at session level with the statement “ALTER SESSION SET PLSQL_DEBUG = TRUE”. Since the statement sets the default compilation method to session level, all subsequent (re)compilation will use the specified method.

If you use JDeveloper to deploy the code in the database you can set the check box “Generate PL/SQL Debug Information” in the “Database Connections” preference panel under “Tools → Preferences”.

Notice that you cannot compile a native compiled stored program in debug modus.

Compiling Java in Debug Modus

To compile a Java stored program in debug modus there are basically two possibilities:

- Specify the flag “-g” during the compilation with javac.
- Before creating or recompiling it with the statement “ALTER JAVA SOURCE <class> COMPILE” set the option DEBUG via the procedure SET_COMPILER_OPTION of the package DBMS_JAVA. Notice that this procedure creates the table JAVA$OPTIONS to store the options used during the compilation. You can remove the table at any time. Of course this means you will lose the options stored in it. The following example shows that before calling SET_COMPILER_OPTION the table JAVA$OPTIONS does not exist.

```
SQL> SELECT * FROM java$options;
SELECT * FROM java$options
              *
ERROR at line 1:
ORA-00942: table or view does not exist

SQL> BEGIN
  2    dbms_java.set_compiler_option(
  3      what       => 'Property',
  4      optionname => 'debug',
  5      value      => 'true');
  6  END;
  7  /

SQL> SELECT * FROM java$options;

WHAT          OPT        VALUE
------------- ---------- ----------
Property      debug      true

SQL> ALTER JAVA SOURCE "Property" COMPILE;
```

If you use JDeveloper to deploy the Java code in the database you can set the check box “Include Debug Information” in the Compiler preference panel under “Project → Project Settings”.

Step 1 Project Setup

Figure 3 shows the Navigator when a minimal setup has been performed, i.e.:

- A project (JPDA.jpr), containing the Java stored program (Property.java) that will be debugged, has been created.
- As written above the PL/SQL stored program (package SYSTEM_PKG) does not reside on the client where JDeveloper is installed. Therefore a database connection (TESTDEBUG_A920) must be configured to extract it directly from the database when needed.

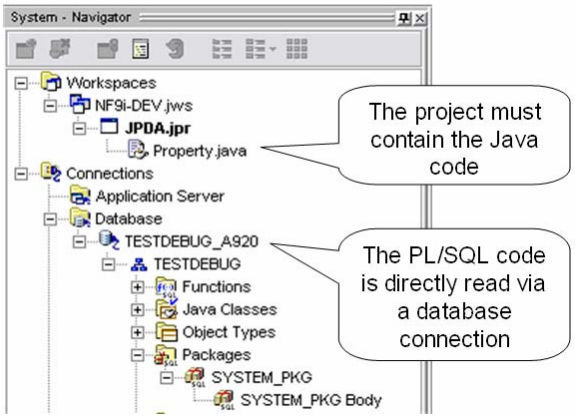

Figure 3 The Navigator, project and database connection

Step 2 Defining Breakpoints

To define breakpoints open the stored program from the Navigator and click on the left of the code (where there are the line numbers). Figure 4 shows two breakpoints, the first defined at line 11 in the PL/SQL code and the second at line 6 in the Java code.

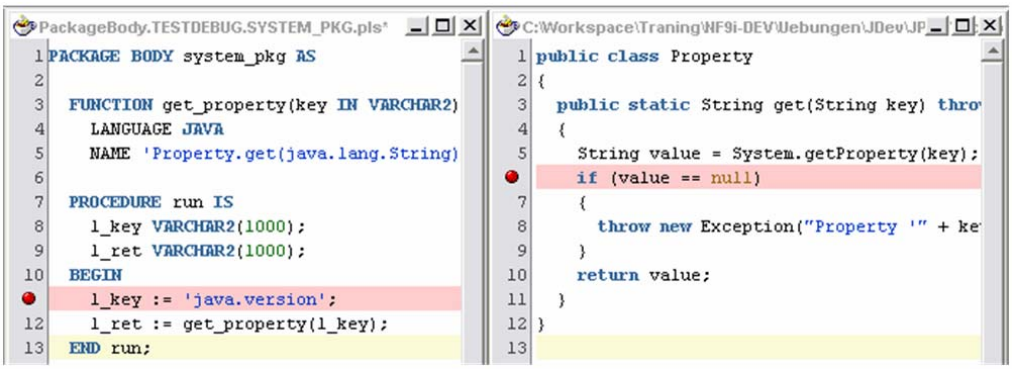

Figure 4 Breakpoints in the PL/SQL and Java code editors

Step 3 Configuring and Starting JPDA Listener

The configuration, at project level, is performed in the “Remote” preference panel under “Project → Project Settings” (Figure 5). The check box “Remote Debugging” and the radio button “Listen for JPDA” must be set. If PL/SQL stored programs are debugged the database connection must be specified too.

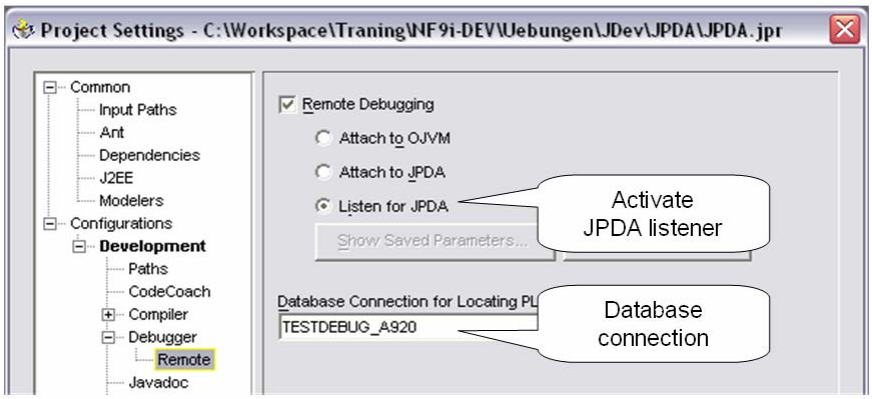

Figure 5 Remote debugging configuration

Once the remote debugging has been activated, when the debugging is started (e.g. with Shift+F9) no execution takes place. Instead the JPDA listener is started.

During the listener startup the user is asked to enter the port number used by the listener (Figure 6). Of course you should choose a free port for it. If you want to bypass this dialog, i.e. in order to always use the same port, select the check box “Save parameters”.

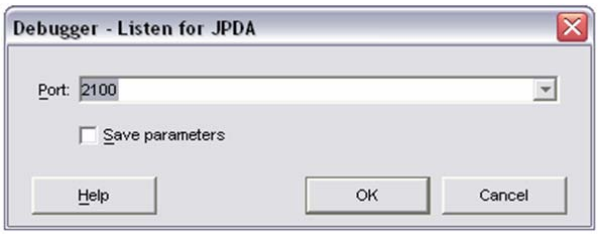

Figure 6 Listener port configuration

In the Run Manager it is possible to check whether the listener is running. In Figure 7 shows that a listener is running on port 2100.

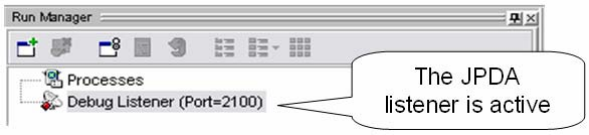

Figure 7 Running listener in Run Manager

Step 4 Enabling Debug Modus in Session

With the package DBMS_DEBUG_JDWP, which is not documented in the server documentation yet (only on OTN you can find some scattered information about it), it is possible to enable and disable the debug modus. The most important procedures that it contains are the following:

- CONNECT_TCP – Enables the debug modus in the current session.
- DISCONNECT – Disable the debug modus in the current session.

With the procedure CONNECT_TCP the location of the JPDA listener must be specified, i.e. the host and port number must be passed as a parameter. When the database and JDeveloper runs on the same system the following call can be used.

```
dbms_debug_jdwp.connect_tcp('localhost', 2100)
```

Notice that both procedures have a parameter SESSION_ID and SESSION_SERIAL; i.e. a user with the necessary privileges should be able to enable the debug modus in any session. Unfortunately this feature is not implemented yet ☹. This means that the debug modus can only be activated from the session that should be debugged. Therefore you should usually foresee this possibility in your application, i.e. define a parameter or property that you can use to activate the debug modus. Another possibility is to use a logon trigger.

The activation of the debug modus can be seen in the Run Manager. Notice the difference between Figure 7 and 8. In the latter the process “JPDA.jpr” (the name of the project) is active. This means that a connection from the database was received for this project.

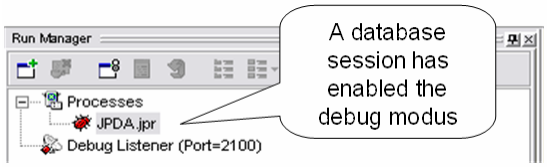

Figure 8 The database session has announced itself to JDeveloper

Step 5 Running the Application

Run the application as usual. When a breakpoint is reached the application is stopped and the control is given to JDeveloper. The very nice feature of this tool is that both PL/SQL and Java are supported. Therefore during a debug session you can transparently step from one language to the other and you can inspect both and modify the variables in the same way. Unfortunately for PL/SQL stored programs not all data types are supported. In fact, as shown in Figure 9, for object types (variable L_OBJECT_TYPE) the type is “OPAQUE” and the attributes cannot be inspected. Since XMLType is an object type, variables of this type (variable L_XMLTYPE) cannot be inspected as well. Notice that records, collections and multi-level collections defined in PL/SQL are supported (variables L_VECTOR and L_ARRAY). In addition the content of a CLOB cannot be completely displayed and the content of a LONG is cut to 32760 characters.

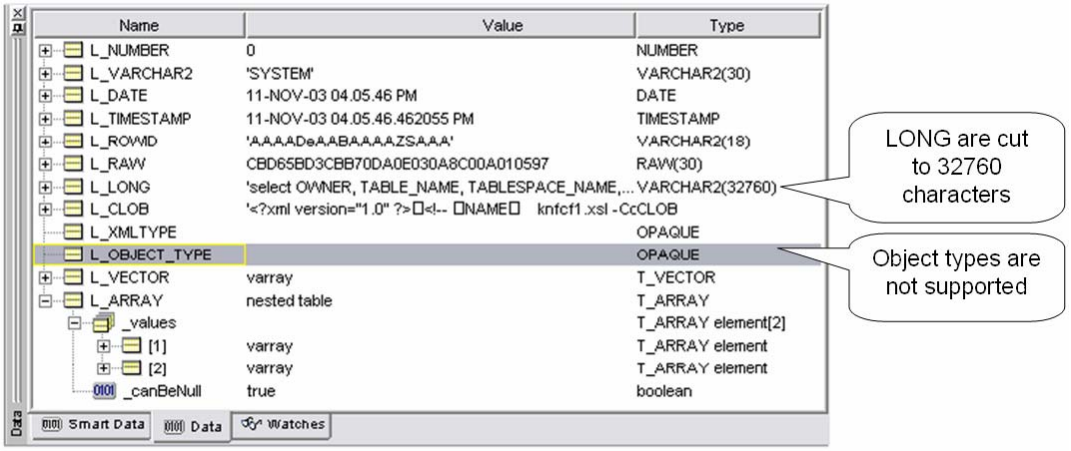

Figure 9 Which data types are supported?

For each run, for stability reasons, I suggest you create a new database connection. If you fail to do so, sometimes JDeveloper freezes. If you want to use the same database connection you should at least close the debugging session with the procedure DISCONNECT from the DBMS_DEBUG_JDWP package. Note that JDeveloper can close the debugging session too.

Conclusion

The integration of JPDA in the database and in JDeveloper is a powerful tool not only for Java developers but also for PL/SQL developers. Of course, thanks to the seamless debugging of PL/SQL and Java stored programs, you get the greatest advantage when you are using both programming languages at the same time.

Some enhancements are desirable; nevertheless, the current versions of the database and of JDeveloper are ready to be used (as I write this, the stable release of the database is 9.2.0.4 and from JDeveloper is 9.0.3.3 is now available).

If you are interested in knowing more about other Oracle9*i* features I would be pleased to welcome you to one of our “New Features Oracle9*i* for Developers” courses.

I hope you don’t have to debug your applications too often, but if you really need to, please, don’t use DBMS_OUTPUT, System.out or any other tracing tool! A debugger allows you to spend your valuable time writing challenging code and not by adding boring tracing instructions to it.