JAVA: EXTENDING THE POWER OF PL/SQL

Vadim Loevski

Summary

Oracle 8i introduced so many new features to the Oracle RDBMS that it will take some time for the Oracle developers to absorb them. It also created some fears among developers about the fate of the PL/SQL programming language, which for many years ruled the world of back-end development in Oracle. These fears are based on the introduction of JServer, which is an environment enabling the use of Java language to write Oracle stored procedures, functions and triggers. The goal of this presentation is

- to show how Java and PL/SQL can co-exist and even complement each other
- to show PL/SQL developers how they can use Java now in order to create functionality they were not able to implement in PL/SQL before (e.g. executing OS commands, advanced file handling routines, ZIP archiving etc).

This approach will help the traditional Oracle developers to make use of Java immediately, extending and enhancing the power of PL/SQL to easily perform tasks that were previously unachievable in PL/SQL.

Introduction

1999 was one of the most exciting years in the Oracle community. Undoubtedly, the main event was the release of the long awaited Oracle 8i database, which added so many new features to Oracle RDBMS, that it will take probably some time for Oracle developers to absorb them, let alone implementing them as part of their architectural solutions. The excitement was not the only thing that this new release brought into the Oracle World, it also created some fears among the traditional Oracle application developers about the fate of PL/SQL – the programming language, which for many years was ruling the world of back-end development. These fears were instantiated by the introduction of the JServer, which is effectively the environment enabling the use of Java language to write Oracle stored procedures, functions and triggers. This article will be interesting to PL/SQL developers who want to use some Java capabilities to extend the power of PL/SQL beyond its traditional limits. This article may not present any interest to pure Java developers since it does not intend to cover Java technology as such. It may interest those who sincerely want to understand the mentality of PL/SQL developers, who are believed, according to some very rough estimates to be the army of more than 150K individuals worldwide.

The future of PL/SQL

Oracle is still committed to PL/SQL. Last year I was at Oracle Open World in Los Angeles where I met the management of Oracle's PL/SQL development. I could not resist asking them a question if they felt any signs of PL/SQL group going the stagnation period because of Java. They laughed and reassured me that the PL/SQL group is quite big and getting bigger. They have huge plans and they are committed to deliver lots of new PL/SQL functionality in the upcoming releases of Oracle. This statement was very encouraging and I felt that the internal competition at Oracle between Java and PL/SQL groups is in fact a very healthy one in terms of speeding up the efforts

in PL/SQL development. I am pretty sure that we will see lots of wonderful additions to PL/SQL in the nearest future. The future of PL/SQL is still pretty bright because PL/SQL excelled as a procedural extension to SQL language and definitely, at this point, is the language of choice for development of Oracle client-server solutions. According to many reports it is a faster language for pure database applications, because of its better integration with SQL.
Java stored programs show unbeatable results when implementing pure computational tasks, but they leg behind in the area of traditional database applications and definitely cannot be used for database applications where the performance is crucial. I remember several articles that tried to convince the Oracle community in the opposite saying that Java is a faster then PL/SQL for database access. A particular memorable article comes to mind, written in one of the respectable Java journals where the author misled readers (hopefully unintentionally) when comparing the performance results of two programs written in Java and PL/SQL. I had a discussion with this author, and want to share with you an example that we discussed. So consider the code below:

```
CREATE OR REPLACE AND RESOLVE JAVA SOURCE NAMED "SqlTester" AS
  import java.lang.*;
  import java.sql.*;
  import oracle.sql.*;
  import oracle.jdbc.driver.*;

  public class SqlTester
  {
   private static Connection nullConnect;
   private static Connection getConnection (int arraySize)
   {
    try {
         Connection conn = new
         OracleDriver().defaultConnection();
         ((OracleConnection)conn).setDefaultRowPrefetch(arraySize);
         return conn;
        }
    catch (SQLException sqlError)
        {
         System.out.println("No Connection!");
         return nullConnect;
        }
   }

   private static Connection getConnection()
   {
    return getConnection(1);
   }

   public static int performReadTest(int arraySize)
   {
    String objectName;
    int objectCount = 0;
    try {
         Connection DBConnect = getConnection(arraySize);
         Statement sqlStmt = DBConnect.createStatement();
         String sql = "SELECT object_name FROM dba_objects";
         ResultSet dbResults = sqlStmt.executeQuery(sql);
         while (dbResults.next())
          {
                          objectName = dbResults.getString(1);
                             objectCount = objectCount + 1;
                                         }
         dbResults.close();
         sqlStmt.close();
        }
    catch (SQLException sqlError)
        {
```

```
        System.out.println("SQL Error!");
       }
    return objectCount;
   }

   public static int performReadTest()
   {
    return performReadTest(1);
   }
  }
/
CREATE OR REPLACE FUNCTION ReadTestJava (n NUMBER)
   RETURN NUMBER
AS
   LANGUAGE JAVA
      NAME 'SqlTester.performReadTest(int) return int';

/

CREATE OR REPLACE FUNCTION ReadTestPlSql
   RETURN NUMBER
IS
   nbr_objects   NUMBER := 0;
BEGIN
   FOR c1_rec IN ( SELECT object_name
                     FROM dba_objects)
   LOOP
      nbr_objects := nbr_objects + 1;
   END LOOP;

   RETURN nbr_objects;
END;
/
```

By measuring the execution time of Java stored procedure and PL/SQL procedure the author came to the conclusion that Java Stored Procedures for database access are 20-40% faster than similar PL/SQL stored procedure. When I read this statement I was puzzled. Trying to understand how the author had arrived to this conclusion I looked into the source code provided in article (see above) and discovered a very important omission. For some reason the author of the article was comparing Java Stored Procedure, which used JDBC array processing with PL/SQL code with no array processing. We all know that array processing is a very important performance boosting technique and conducting a test without applying it to both Java and PL/SQL was a serious mistake. As it turned out to be the case the author of the article did not know about the existence of PL/SQL array processing, which was introduced in Oracle 8 as part of DBMS_SQL package. After rewriting the PL/SQL function with array processing, as follows:

```
CREATE OR REPLACE FUNCTION ReadTetPsql1 (p_array_size IN NUMBER)
   RETURN NUMBER
IS
   cur           NUMBER                  := DBMS_SQL.open_cursor;
   fdbk          NUMBER;
   obj_nm_tab    DBMS_SQL.varchar2_table;
   indx          NUMBER                  := p_array_size * (-1);
   nbr_objects   NUMBER                  := 0;
BEGIN
   DBMS_SQL.parse (
      cur,
      'select object_name from dba_objects',
      DBMS_SQL.native
   );
   DBMS_SQL.define_array (cur, 1, obj_nm_tab, p_array_size, indx);
   fdbk := DBMS_SQL.execute (cur);
```

```
   LOOP
      fdbk := DBMS_SQL.fetch_rows (cur);
      DBMS_SQL.column_value (cur, 1, obj_nm_tab);
      nbr_objects := nbr_objects + fdbk;
      EXIT WHEN fdbk != p_array_size;
   END LOOP;

   DBMS_SQL.close_cursor (cur);
   RETURN (nbr_objects);
EXCEPTION
   WHEN OTHERS
   THEN
      IF DBMS_SQL.is_open (cur)
      THEN
         DBMS_SQL.close_cursor (cur);
      END IF;

      RETURN (nbr_objects);
END;
/
```

I discovered that the result was quite opposite. PL/SQL was 30-40% faster than the equivalent Java stored procedure with the same array fetch size. You can run the tests in your to get similar results:

```
SQL> SET TIMING ON
SQL> REM
SQL> REM Java Stored Program with array fetch size 1000
SQL> REM
SQL> select readtestjava(1000) from dual;

READTESTJAVA(1000)
------------------
             12691

 real: 2053
SQL> REM
SQL> REM PL/SQL Stored Program with no array processing
SQL> REM
SQL> select readtestplsql from dual;

READTESTPLSQL
-------------
        12691

 real: 3235
SQL> REM
SQL> REM PL/SQL Stored Program with array fetch size 1000
SQL> REM
SQL> select readtestplsql1(1000) from dual;

READTESTPLSQL1(1000)
--------------------
               12691

 real: 1182
SQL>
```

Java is a great language and I sincerely enjoy writing Java programmes. But being also PL/SQL guy I would like also to add some observations that might be interesting for you:

1. PL/SQL has native SQL syntax support and in comparison with Java produces clean, easy-to-read code. This statement will seem very true for PL/SQL programmers, but Java programmers will definitely appreciate JDBC interface since it is the natural extension of the object model.
2. Java code for SQL is very verbose, which means that one line of PL/SQL implicit cursor will be equivalent to approximately 10 lines of JDBC code. This affects readability but again it is not an issue once JDBC syntax is understood and remembered.
3. However, a more important observation is the following. If your PL/SQL module uses %TYPE compiler directives to allow the variable types to be based on the types of columns in the table being queried, the following PL/SQL code does not have to be re-written if the table column is modified later on.

```
CREATE OR REPLACE FUNCTION get_object_id (
   p_owner          IN   VARCHAR2,
   p_object_name    IN   VARCHAR2,
   p_object_type    IN   VARCHAR2
   )
   RETURN NUMBER
IS
   l_object_id   all_objects.object_id%TYPE;
BEGIN
   FOR rec IN ( SELECT object_id
                  FROM all_objects
                 WHERE owner = p_owner
                   AND object_name = p_object_name
                   AND object_type = p_object_type)
   LOOP
      l_object_id := rec.object_id;
   END LOOP;
END;
```

   This is not the case for Java stored modules that have to be re-written to accommodate the change of the table column type and consequently reloaded into the database.

In my view PL/SQL as of Oracle 8.1.6 also presents a clear advantage in the following areas:

1. Native data types
2. Transaction control
3. BULK COLLECT operation (no direct equivalent in Java)

The first point is purely due to the fact that PL/SQL natively implements SQL data types. For example, for NUMBER datatype this implementation supports **native scale and precision semantics**. If you convert NUMBER into Java's 'double' or 'float' this may cause the loss of Oracle specific rounding semantics, since Java uses its own rounding approach. The only choice is to use Oracle extensions supplied in oracle.sql.NUMBER class, which means we will have to use method calls to achieve the functionality which are available in PL/SQL through native arithmetic.

In the area of transaction control (2) PL/SQL is the only language, which supports **autonomous transactions** the new feature introduced in Oracle 8i. Java stored procedures will have to call PL/SQL procedures that implement autonomous transactions.

BULK COLLECT operation (3) presents another area where PL/SQL is clearly ahead. This operation allows selecting the table rows into PL/SQL memory table in a single operation. This functionality could be programmed in Java as well, but BULK COLLECT is implemented in PL/SQL natively thus producing better performance results.

"Wait and See" Vs "Seize the Day"

Despite the fact that PL/SQL is a better choice for applications that require database access, the fact remains that **PL/SQL is no longer the only server-side language available in Oracle**. Sooner or later every PL/SQL developer will ask the question 'What do I do?'. I can see two obvious strategies in this area. The first and the simplest strategy is "wait and see", which is fairly self-explanatory and more than likely, it will be the most popular one, since it does not require any change on behalf of the traditional Oracle PL/SQL programmers. Really, it is very easy to wait and see who is going to win Java or PL/SQL and make a decision based on that result.

My personal preference is 'seize the day' strategy, which is the approach of taking the opportunity straight away and ripping the benefits of server-side Java in order to extend the functionality of PL/SQL. This approach will take PL/SQL beyond its design limits and it will also help PL/SQL developers to get familiar with Java immediately. This statement will probably require some extra explanations.

Java is not only a language with all its syntactical originality. It is an entire whole programming environment that consists of myriads of libraries or class libraries in Java terminology. These libraries contain lots of very useful functionality that can be used by PL/SQL as well as by Java itself. This may seem even more useful if I say that Java community is very large and there are even more class libraries that can be downloaded from the Internet and loaded into Oracle database and potentially used by PL/SQL developers from within the context of PL/SQL programs. In addition Oracle Corporation itself implemented a lot of useful Java class libraries that became part of Jserver and are included into JDeveloper libraries.

Where to start?

Looking through the Oracle-related mailing lists, frequently-asked questions and newsgroups it is not very hard to realise that the main challenges developers of Oracle-based applications face when they attempt to implement the requirements that traditionally do not fall into the area of RDBMS. These challenges are very well known and it will not be a huge discovery if we mention among them such tasks as sending mail messages from PL/SQL, deleting files from the file system, extracting data from the database tables and saving the data in ZIP archive files, executing operating system commands from PL/SQL stored programmes. The list of these challenges could be continued spanning through the number of pages still far from being complete. PL/SQL never had straightforward built-in functionality to allow the above mentioned tasks to be programmed easily. PL/SQL was designed as a language that is strictly focused on the database applications, giving the procedural power to SQL the standard language that became the intrinsic part of many relational databases.

So, what were the choices for the developers, who tried execute operating system command from PL/SQL or the file handling routines such as move file and delete file, before Oracle 8I. The only approach available in Oracle 7 was based on the usage of Oracle database pipes. The database pipe is a mechanism similar to UNIX pipes through which a process (let's say PL/SQL procedure) could communicate with another process (e.g., UNIX process written in PRO*C), which in turn could execute an OS command performing the required function. Oracle 8 introduced external libraries – SO or DLL libraries that could be utilised directly from PL/SQL. The main disadvantages of these cumbersome techniques were application architecture complexity, non-portability (native libraries and programs), application deployment difficulties (executable code residing outside the database) and poor maintainability.
Java in the database is a big achievement, which opens the opportunity to develop applications that are purely server-side based and 100% portable.

What is the best place to start for PL/SQL developer who wants to get familiar with Java technology and to write simple extensions to traditional PL/SQL? Well, here is the list of things that PL/SQL developers can implement easily after reading a short introduction to Java:

- Extensions to UTL_FILE. Finally we can implement the set of file handling functions that were not implemented as part of the famous UTL_FILE package (e.g. delete, copy and move file, create directory etc)

- Write utilities based on standard Java classes. For instance, this could be ZIP archiving utility, hash-encoding etc

- Recently a friend of mine asked me how to convert EST into GMT. There is no easy way to do it in PL/SQL. NEW_TIME function does work but in the opposite direction (GMT to EST). Now we can create NEW_TIME utility that works in all directions using Java classes stored in Oracle 8i. Actually, there are other Java date/time classes that maybe useful for PL/SQL developers.

This list maybe be continued but it is more than enough for a start. Let's implement some of the above mentioned utilities.

Let's get to business

The main complexity traditional PL/SQL developers will encounter when starting to familiarize themselves with Java technology is the concept of Object Oriented Design. I agree that to understand object orientation will require some efforts and it will be a steep learning curve for the majority of us, but the good news is that Java does not force you to write object oriented programs. A Java class can be written by using so called static methods only. These methods are very similar to PL/SQL procedures and functions. Also, PL/SQL programs can only call these static methods unless PL/SQL program is a method of Oracle's object type.

The next complexity is to familiarize with Java class libraries. There are so many of them that it will require some time to find the class you will want to utilise from your PL/SQL program.

Let's implement a Java stored procedure that allows executing operating system commands. This could be easily done via exec method of Runtime class defined as part of java.lang class library. This library is one of the core Java libraries and it is distributed with any implementation of Java Virtual Machine (JVM) including Oracle JVM, which is part of JServer. This Java stored procedure is defined as the following DDL statement:

```
CREATE OR REPLACE AND RESOLVE JAVA SOURCE NAMED " UTLcmd" AS
import java.lang.Runtime;
public class UTLcmd
{
  public static void execute (String command)
  {
   try
      {
       Runtime rt = java.lang.Runtime.getRuntime();
       rt.exec(command);
      }
   catch(Exception e)
   {
    System.out.println(e.getMessage());
    return;
   }
  }
}
```

```
/
```

The next step is to allow invocation of Java stored procedures from PL/SQL. We need to publish Java to PL/SQL by creating a PL/SQL program, which is a proxy for a Java method that we want to call ('execute' static method in UTLcmd class mentioned above). This is also frequently called Java call specification (call-spec). There is a new SQL DDL syntax introduced specifically to write call-specs. This is reasonably well described in the relevant Oracle 8I documentation. The new DDL accomplishes the following: mapping Java to PL/SQL datatypes, mapping Java parameter modes (in/out) to PL/SQL. The following is a DDL statement required to call UTLcmd.execute method from any PL/SQL block.

```
/
CREATE OR REPLACE PACKAGE UTLcmd IS
  PROCEDURE execute (cmd IN VARCHAR2) AS LANGUAGE JAVA NAME
           'UTLcmd.execute(java.lang.String)';
END;
/
BEGIN
  UTLcmd.execute
   ('cmd /c copy C:\TEMP\TEMP\MYTEXT.TXT C:\TEMP\TEMP\YOUR.TXT');
END;
/
```

This particular call-spec was created as part of the package specification, bit it would be also possible to create it as a standalone PL/SQL procedure.

How many times PL/SQL developers complained about the lack of file-handling routines supplied by Oracle as part of UTL_FILE package? Java opened the opportunity to overcome this obvious deficiency. The following Java source and call-spec implement some routines required for more complex file operations.

```
create or replace and resolve java source named "UTLFile" as
import java.io.File;
import java.io.*;
public class UTLFile {
 public static int delete (String sfilename)
 {
  File sf = new File(sfilename);
  if (sf.delete()) return 1;
  return 0;
 }
// Creates the directory named by sfilename, including any
// necessary but nonexistent parent directories.

 public static int mkdir (String sfilename)
 {
  File sf = new File(sfilename);
  if (sf.mkdirs()) return 1;
  return 0;
 }
//   Renames the file denoted by sfilename.
//

 public static int rename (String sfilename, String dfilename)
 {
  File sf = new File(sfilename);
  File df = new File(dfilename);
  if (sf.renameTo(df)) return 1;
  return 0;
 }
// Copy file
//
```

```
 public static int copy (String sfilename, String dfilename)
 {
  try {
       byte[] buf = new byte[4096];
       int retval;
       FileInputStream   is = new FileInputStream  (sfilename);
       FileOutputStream  os = new FileOutputStream (dfilename);
       while ((retval = is.read(buf, 0, 4096)) > -1)
       {
        os.write(buf, 0, retval);
       }
       is.close();
       os.close();
       return 1;
      }
   catch (Exception e)
      {
       return 0;
      }
 }
// Copy file
//
 public static int move (String sfilename, String dfilename)
 {
   if (copy(sfilename, dfilename) == 1)
   {
    File f = new File(sfilename);
    if (delete(sfilename) == 1) return 1;
   }
   return 0;
 }
}
/
CREATE OR REPLACE PACKAGE UTLFile IS
  FUNCTION delete (sfilename IN VARCHAR2) RETURN NUMBER AS LANGUAGE JAVA
     NAME 'UTLFile.delete(java.lang.String) return int';
  FUNCTION mkdir (sfilename IN VARCHAR2) RETURN NUMBER AS LANGUAGE JAVA
     NAME 'UTLFile.mkdir(java.lang.String) return int';
  FUNCTION rename(sfilename IN VARCHAR2, dfilename IN VARCHAR2)
    RETURN NUMBER AS LANGUAGE JAVA NAME
     'UTLFile.rename(java.lang.String, java.lang.String) return int';
  FUNCTION copy(sfilename IN VARCHAR2, dfilename IN VARCHAR2)
    RETURN NUMBER AS LANGUAGE JAVA NAME
     'UTLFile.copy(java.lang.String, java.lang.String) return int';
  FUNCTION move(sfilename IN VARCHAR2, dfilename IN VARCHAR2)
    RETURN NUMBER AS LANGUAGE JAVA NAME
     'UTLFile.move(java.lang.String, java.lang.String) return int';
END;
/
```

The only word of caution is related to Java security. Since the file-handling operations will obviously require some access to the disk storage, we will need to grant some privileges to the user requiring this access. There is nothing new in this requirement. We all remember UTL_FILE_DIR initialisation parameter, which is required before UTL_FILE package can be used. UTL_FILE_DIR parameter still needs to be defined for Java stored procedures. In addition JServer security requires JAVASYSPRIV or JAVAUSERPRIU roles to be granted to the user:. The difference between these two roles is the following. JAVAUSERPRIV is required if the user intends only to read files/directories. JAVASYSPRIV must be granted if the user will need to create/delete files or directories

Creating ZIP archives from PL/SQL programs was a dream. It is very easy to implement this functionality now. Java class library java.util.zip contains all the necessary methods required for

building quite sophisticated ZIP archiving utilities. The following is an example of Java stored procedure that reads file form the hard disk and puts into archive file after compressing it.

```
CREATE OR REPLACE AND RESOLVE JAVA SOURCE NAMED "UTLZip" AS
import java.util.zip.*;
import java.io.*;
public class UTLZip
{
 public static void compressFile(String infilename, String outfilename)
 {
  try
    {
     FileOutputStream fout = new FileOutputStream(outfilename);
     ZipOutputStream zout = new ZipOutputStream(fout);
     ZipEntry ze = new ZipEntry((new File(infilename)).getName());
     try
       {
        FileInputStream fin = new FileInputStream(infilename);
        zout.putNextEntry(ze);
        copy(fin, zout);
        zout.closeEntry();
        fin.close();
       }
     catch (IOException ie)
     {
      System.out.println("IO Exception occurred: " + ie);
     }
     zout.close();
    }
  catch(Exception e)
    {
     System.out.println("Exception occurred: " + e);
    }
 }

 public static void copy(InputStream in, OutputStream out)
      throws IOException
 {
  byte[] buffer = new byte[4096];
  while (true) {
    int bytesRead = in.read(buffer);
    if (bytesRead == -1) break;
    out.write(buffer, 0, bytesRead);
  }
 }
}

/
CREATE OR REPLACE PACKAGE UTLZip
IS
   PROCEDURE compressFile (p_in_file IN VARCHAR2, p_out_file IN VARCHAR2)
   AS
      LANGUAGE JAVA
         NAME 'UTLZip.compressFile(java.lang.String,
                   java.lang.String)';
END;
/

exec dbms_java.set_output(1000000);
exec file_zip('/home/oracle/of.lst','/home/oracle/of.zip')
```

Conclusion

This article demonstrated lots of examples of Java stored programs callable from PL/SQL. Java created huge opportunities for PL/SQL developers who really want to extend PL/SQL language capabilities. Just think of the different application functions, which we could easily implement in PL/SQL using Java stored programs. For instance we can select multiple LOBs (e.g. MS Word documents) from the database, ZIP them into one archive file and email to the nominated addressee and all this can be 100% transaction driven.
At the same time PL/SQL developers must be ready to challenges as well. Java technology could present a steep learning curve for PL/SQL developers that are not ready to grasp OO concepts. My advice would be to start writing simple classes and not get intimidated by the complexities of Java. It is a great language and you will be rewarded in the long run for learning it.