Java and Web Services
Developer's Perspective on Oracle Database 10*g*

An Oracle White Paper
May 2005

ORACLE

Java and Web Services Developer's Perspective on Oracle Database 10*g*

Java and Web Services Developer's Perspective on Oracle Database 10*g*

INTRODUCTION – DATABASE MATTERS

Java and Web Services developers are in quest of portability and interoperability (standards support), faster time to market (productivity), sustaining unpredictable workload (performance and scalability), reliability (data integrity, high-availability), reach (reach-in and reach-out) and extensibility (risk reduction). Higher-level J2EE containers and OR mapping frameworks that shield the Java developers from the underlying database capabilities might give the illusion that RDBMS are interchangeable commodities, in other words, mere data repositories for persisting object states, but as you will see through this paper, database matters for building, deploying, and managing Java/J2EE applications and Web Services. I'll look at the Oracle Database 10*g* from the Java developers and Web Service developers' perspective. More specifically, how the Oracle JDBC drivers, the Java runtime in the database and Database Web Services framework address standard support, productivity, performance, scalability, load-balancing, reliability, reach and high-availability. The white paper "*What's New for Java, JDBC, and Database Web Services*"[1]gives an exhaustive overview of the new features in Oracle Database 10*g* Release 1 and Release 2.

STANDARD SUPPORT - PORTABILITY

Application portability across platforms and vendors is one of the motivations for choosing Java. On JDBC front, portability is achieved through the complete support for JDBC 3.0 and JDBC Rowset (i.e., JSR 114) specifications. The Java runtime in the database is modeled after the ANSI SQLJ Part-I specification, which defines "*SQL Routines and Types Using Java*". J2SE 1.4 compatibility ensures the portability of Java stored procedures and functions from foreign RDBMS to Oracle's robust and more scalable implementation. Web Services allow XML message-based, web standards based, and service centric applications to applications interactions. Oracle Database 10g Web services allows Web Services clients to invoke database operations such as PL/SQL procedures, Java methods in the database, SQL Queries and DML, Streams/AQ through standard web services mechanisms; conversely, external Web Services can be invoked from within the database, using standard Web services standards mechanisms: SOAP over HTTP, WSDL, and UDDI.

[1] http://www.oracle.com/technology/tech/java/jsp/index.html

REUSE - PRODUCTIVITY

One of the key benefits of Java is code and skills reuse, which results in productivity. For SQL data access front, standard JDBC 3.0 applications are reusable as is, with the Oracle Database 10*g* JDBC drivers. Further productivity is garnered when using the Oracle specific extensions to JDBC. The Java runtime in the Oracle Database 10g allows the implementation of procedures, functions and call-out capabilities – such as JDBC callout, RMI callout, HTTP callout, RMI/IIOP callout, JMS callout, SOAP/HTTP callout – just by reusing existing Java libraries, resulting in huge productivity gains for the Java developers. Most customers build Database Web Services by reusing existing PL/SQL stored procedures and functions, existing Java stored procedures and functions, existing Data Warehousing SQL queries. The Oracle Database Web Services white paper available on the Oracle Technology Network[2] provides more details on publishing database entities as Web services as well as consuming external Web Services from within the Oracle Database.

SCALABILITY AND LOAD-BALANCING

Inefficient JDBC connection pooling might clog database access; the Oracle JDBC 10*g* drivers bring scalability through Implicit Connection Cache (ICC) and Runtime Connection Load Balancing (RCLB). The ICC allows connection tagging/ stripping, connection attribute and connection weight based search and retrieval. RCLB is a powerful scalability feature, which routes JDBC connection requests based on RAC nodes workload. Unlike other RDBMS vendors, which use an external JDK VM or dedicated processes to run Java, the Oracle database[3] runs Java by using the same session architecture it uses to run any database operation (i.e., SQL, PL/SQL, etc), hence the same proven scalability. The current implementation of Database as Web Services provider utilizes the SOAP provider and JDBC connection pooling in the Oracle Application Server hence scale identically.

PERFORMANCE

Performance is a key requirement for Java applications and Web Services. The performance optimizations in Oracle Database 10*g* include the optimization of Java to SQL data conversion, the reduction of the drivers code path (i.e., the reduction of the number of Java methods called to satisfy a request), the reduction of data movement to/from the database and reusing Java objects.

Even though pure computational Java runs faster outside the database, when you add SQL data access (JDBC), data traffic and network overhead, the resulting Java application runs faster in the database. Here follows a complete code sample that you can run in your environment, which demonstrates what I have just declared;

---

[2] http://www.oracle.com/technology/tech/webservices/database.html
[3] http://www.oracle.com/technology/tech/java/jsp/index.html

the timing figures will be different on your platform but I am confident that the trend is consistent (i.e., JDBC applications run faster in the database).

Running JDBC Applications Faster in the Database

One of the benefit of Java is the availability of a HUGE set of JDBC programs that you can reuse however, few changes are required for running a standard JDBC application in the database. In summary you need to address JDBC connection, loading and compiling the JDBC application in the database -- or compiling first then loading the resulting class in the database; that's it!

Environment and Set Up

We'll run the Workers.Java as stand-alone JDBC application. The platform is a Windows based laptop equipped with 1GB RAM which hosts the JDBC application, an Oracle database 10.1.0.2 and the corresponding JDBC drivers.

Setting Up the Database Table

The following sql script will prepare a table to be used by your code sample

```
rem
rem Workers.sql
rem
connect scott/tiger
rem
rem Set up the Workers table
rem
drop table workers;
create table workers (wid int, wname varchar2(20),
     wposition varchar2(25), wsalary int);

insert into workers values (103, 'Adams Tagnon',
'Janitor', 10000);
insert into workers values (201, 'John Doe',
'Secretary', 20000);
insert into workers values (323, 'Racine Johnson',
'Junior Staff Member', 30000);
insert into workers values (418, 'Abraham Wilson',
'Senior Staff Member', 40000);
insert into workers values (521, 'Jesus Nucci',
'Engineer', 50000);
insert into workers values (621, 'Jean Francois',
'Engineer', 60000);
commit;
```

At this stage, the "*workers*" table has been created and populated.

The JDBC Application

Workers.java

The following code sample retrieves a specific worker from the workers table then updates its position and salary.

```
/*
 *  Adapted from existing JDBC demo.
 */

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

public class Workers
{

  public static void main (String args []) throws SQLException
{

     String name = null;
     String pos = null;
     int sal;
     int id;
     long t0,t1;
     Connection conn = null;
     Statement stmt = null;
     PreparedStatement pstmt = null;

     if ( args.length < 1 ) {
      System.err.println("Usage: Java Workers <wid> <new
position>
                                                <new   salary>");
      System.exit(1);
      }

     // Get parameters value
     id = Integer.parseInt(args[0]);
     pos = args[1];
     sal = Integer.parseInt(args[2]);

       /*
       * Where is your code running: in the database or
outside?
       */
       if (System.getProperty("oracle.jserver.version") !=
null)
       {
       /*
       * You are in the database, already connected, use the
default
       * connection
       */
       conn =
DriverManager.getConnection("jdbc:default:connection:");
       System.out.println ("Running in OracleJVM,  in the
database!");
       }
       else
       {
       /*
       * You are not in the database, you need to connect to
       * the database
       */

       DriverManager.registerDriver(new
oracle.jdbc.OracleDriver());
       conn = DriverManager.getConnection("jdbc:oracle:thin:",
                                          "scott", "tiger");
        System.out.println ("Running in JDK VM, outside the
database!");
        }
```

```
      /*
       * Turning off Auto-commit, not supported in the Database
       */
       conn.setAutoCommit (false);

       // Start timing
          t0=System.currentTimeMillis();

      /*
       * find the name of the workers given his id number
       */

       // create statement
          stmt = conn.createStatement();

// find the name of the worker
   ResultSet rset = stmt.executeQuery(
     "SELECT WNAME FROM workers WHERE wid = " + id);

// retrieve and print the result (we are only expecting 1 row
   while (rset.next())
   {
    name = rset.getString(1);
   }

// return the name of the worker with the given worker number
   System.out.println ("Worker Name: "+ name);
  /*
   * update the position and salary of the retrieved worker
   */

// prepare the update statement
   pstmt = conn.prepareStatement("UPDATE WORKERS SET WPOSITION
   = ?, " + " WSALARY = ? WHERE WNAME = ?");

// set up bind values and execute the update
   pstmt.setString(1, pos);
   pstmt.setInt(2, sal);
   pstmt.setString(3, name);
   pstmt.execute();

// double-check (retrieve) the updated position and salary
   rset = stmt.executeQuery(
   "SELECT WPOSITION, WSALARY FROM WORKERS WHERE WNAME = '"
    + name + "'");
   while (rset.next())
   {
     pos = rset.getString ("wposition");
     sal = rset.getInt ("wsalary");
   }
   System.out.println ("Worker: Id = " + id + ", Name = " +
   name +  ", Position = " + pos + ", Salary = " + sal);

// Close the ResultSet
   rset.close();

// Close the Statement
   stmt.close();

// Stop timing
   t1=System.currentTimeMillis();
   System.out.println ("===> Duration: "+(int)(t1-t0)+ " ms");
```

```
// Close the connection
   conn.close();
   }
 }
```

Dealing with JDBC Connections

To run Java in the database, you need first to be already connected to a database session therefore, the server-side JDBC connection, is in reality just a handle to the session itself. A specific connect string syntax is required in this case. However, because OracleJVM does not yet support resolving JDBC connect strings through data-sources, you have to use the traditional DriverManager syntax:

```
Connection conn =
DriverManager.getConnection("jdbc:default:connection:");
or
Connection conn =
DriverManager.getConnection ("jdbc:oracle:kprb:");
or
OracleDriver t2server = new OracleDriver();
Connection conn = t2server.defaultConnection();
```

The following code snippet deals with JDBC connect string whether the program is running outside or within the database:

```
Connection conn = null;
       ...
       ...

/*
 * Where is your code running (in the database or
outside)?
 */
  if (System.getProperty("oracle.jserver.version") !=
null)
  {
   /*
    * You are in the database, already connected, use
the default
    * connection
    */
   conn =
DriverManager.getConnection("jdbc:default:connection:");
   System.out.println ("Running in OracleJVM,  in the
database!");
   }
   else
   {
   /*
    * You are not in the database, you need to connect
to
    * the database using either the traditional Driver
Manager
    * syntax (showed below), or JDBC datasources syntax
(not showed)
    */
```

```
    DriverManager.registerDriver(new
oracle.jdbc.OracleDriver());
    conn =
DriverManager.getConnection("jdbc:oracle:thin:",
                                        "scott",
"tiger");
    System.out.println ("Running in JDK VM, outside the
database!");
    }
```

Running the JDBC Application as Stand-alone Client

Step#1: compile
With proper classpath end environments settings:

```
javac Workers.java
```

Step#2: invoke

```
C:\My_Data>java Workers 103 "Manager" 13000
Running in JDK VM, outside the database!
Worker Name: Adams Tagnon
Worker: Id = 103, Name = Adams Tagnon, Position = Manager,
Salary = 13000
====> Duration: 390 Milliseconds
```

For consistency, invoke again

```
C:\My_Data>java Workers 201 "Director " 23000
Running in JDK VM, outside the database!
Worker Name: John Doe
Worker: Id = 201, Name = John Doe, Position = Director , Salary
= 23000
====> Duration: 421 Milliseconds

C:\My_Data>
```

It takes the JDBC program, in average 400 Milliseconds to perform this database operation when it runs as stand-alone application, outside the database. Let's see it performs when it runs within the database. I'll show you the various ways of invoking Java in the database.

Running the JDBC Program in the Database

There are many ways to create Java in the database, which are beyond this article (see the Oracle Database 10g Java Developers Guide for more details). Here is the easiest way using a SQL*Plus session

```
rem
rem Create Java class Workers.java, directly in the
database
rem  assuming we are still connected as scott/tiger
```

```
rem

create or replace java source named Workers as

/*
 * copy-and-paste here the Workers.java
 */

 ...
 ...

 /

show errors;

alter java source Workers compile;
show errors;
```

At this stage, assuming there are no errors, the Workers.class has been created in the scott schema, and ready to run.

Invoking Java in the Database

The top-level call to Java in the database is generally performed using a user-defined PL/SQL wrapper (a.k.a. Call Spec). It hides the implementation language of the stored procedures --could be Java, PL/SQL or any other supported language; it can be invoked by SQL, JDBC, and middle-tier Java/J2EE (through JDBC).

Invoking Java through Call Spec involves two steps:

(i) publishing the Java method to SQL through a PL/SQL wrapper (i.e., Call Spec)

(ii) invoking the method through the PL/SQL wrapper

*Step#1: publishing the PL/SQL wrapper (from a SQL*plus session)*

In general, Call Spec parameters and the arguments of the published Java methods must correspond one to one however, this rule does not apply to the `main()` method which only has an array of `String` as parameter (`String[]`). This `Array of String` can only be mapped parameters of `CHAR or VARCHAR2` types. As a consequence of parameter mapping with `main()`, both worker id and worker salary, which holds int

values are mapped to `VARCHAR2.`

```
SQL>create or replace procedure WorkerSP (wid IN
varchar2, wpos IN
    varchar2, wsal IN varchar2) as
    language java name
'Workers.main(java.lang.String[])';
/
Procedure created.

SQL> show errors;
```

```
No errors.
```

*Step#2: invoking the Java method through the PL/SQL wrapper (from a SQL*Plus session)*

```
C:\My_Data>sqlplus scott/tiger

SQL*Plus: Release 10.1.0.2.0 - Production on Wed Feb 2 17:10:08
2005

Copyright (c) 1982, 2004, Oracle.  All rights reserved.


Connected to:
Oracle Database 10g Release 10.1.0.2.0 - Production

SQL> set serveroutput on
SQL> call dbms_java.set_output(50000);

Call completed.

SQL> call WorkerSp('621', 'Senior VP', '650000');
Running in OracleJVM,  in the database!
Worker Name: Jean Francois
Worker: Id = 621, Name = Jean Francois, Position = Senior VP,
Salary = 650000
====> Duration: 12 Milliseconds

Call completed.

SQL> call WorkerSp('201', 'Senior VP', '650000');
Running in OracleJVM,  in the database!
Worker Name: John Doe
Worker: Id = 201, Name = John Doe, Position = Senior VP, Salary
= 650000
====> Duration: 10 Milliseconds

Call completed.

SQL>
```

The elapsed time is consistently around 10 to 12 milliseconds, for this particular application, JDBC runs almost *40 times faster* in the database than outside. However, not all JDBC applications are suitable for being executed in the database, try your own JDBC application and choose the most efficient place to run it!

RELIABILITY – DATA INTEGRITY

Data integrity is a universal requirement. By design Java in the Oracle database runs in the same address space as SQL and PL/SQL while preserving data integrity through security mechanisms. Like any code running in the database, Java in the Oracle database benefits from a rich set of security mechanisms including: user-authentication, database-schema security, login-user security, and effective-user security. In addition, the Java VM in the database adds class-resolution security and Java 2 security. For more details on the security mechanism in the OracleJVM, read the related article[4] on the Oracle Technology Network.

[4] http://www.oracle.com/technology/oramag/oracle/03-jul/o43devjvm.html

RELIABILITY – HIGH-AVAILABILITY

In multi-instance RAC and Grid environments, database connections are distributed across all nodes. When an instance goes down, due to an instance or host failure, it brings down the connections associated with the instance or host in question; this could potentially result in bad connections being returned from the cache. Although instance or node failure might not be avoidable, high-availability mechanisms can palliate the impacts for Java applications and Web Services. The Oracle Database 10g JDBC furnishes Fast Connection Failover (FCF) which enables an automatic detect-and-fix mechanism to handle any instance or host failures in a RAC environment. This mechanism performs fast shutdown of connections in the connection cache, when RAC instance/node failures are detected, preventing bad or invalid connections being handed out to applications. FCF is enabled on a cache enabled DataSource, by simply flipping the DataSource property *astConnectionFailoverEnabled* to *true*. FCF is described in greater details in the "*What's New for Java, JDBC, and Database Web Services* white paper. The current implementation of Database Web Services leverages the Web Services framework in the Oracle Application server (i.e., Database as Web Services Provider) and the JDBC connection pool; it therefore benefits from the JDBC Fast Connection Fail-Over.

REACH AND EXTENSIBILITY

Database Applications that are designed to offer specific services can be exposed to remote applications as Web Services and therefore reachable by non-connected applications and heterogeneous environments; see the white paper "*Web Services in Oracle Database 10g, and Beyond*" for more details. Similarly, database applications might need to interact with external systems and components such as Enterprise Java Beans (i.e., RMI/IIOP Callout), Web components (i.e., HTTP Callout), and Web Services (SOAP/HTTP callout[5]), and so on. Embedding a Java runtime in the database allows the extension of the capabilities of the database by simply using standards or third party Java libraries. The TECSIS use case[6] is an illustration of custom enterprise integration, using Java in the database.

Bridging the Database and EJBs (RMI/IIOP Callout)

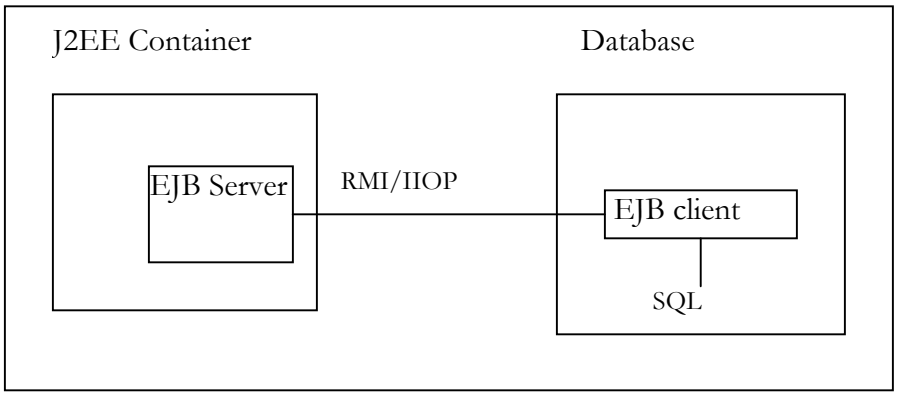
Enterprise Java Beans are the archetype of J2EE programming model where the infrastructure services such as transaction and security management (i.e. behavior) are declaratively specified. These are generally complex or pure computational logic that implement a business function such as Tax calculation. Why call an EJB from within the database? Assume a batch job iterates through customers requests (i.e., result set) for quotes (i.e., car rental, loans, etc). They take lot of input parameters and produce output the rate proposals. For each request, or group of

---

5 http://www.oracle.com/technology/sample_code/tech/java/jsp/samples/wsclient/Readme.html

6 http://www.oracle.com/technology/tech/java/jsp/pdf/JavaDB_Use_Case.pdf

requests, a batch job running in the database invokes a stateless EJB that implements a rate engine. Rate engines are complex and compute intensive. In this example, the database and the middle-tier co-operate to accomplish the business service.



Integrating the Database and Web Components (HTTP Callout)

Database Applications might also cooperate with Web components, using HTTP Callout, to accomplish a business services. Assume a product catalog (or rates) that do not change so frequently, are cached in a middle-tier application for faster search and retrieval. A Java stored procedure which implements HTTP callout, is used as a database trigger and attached to the product table. Upon changes to the database table, the triggers fires an HTTP call to a Java ServerPage which forces the middle-tier application to refresh its cache with fresher values from the database. You can see this as a poor man's cache invalidation mechanism. The "*Database-Assisted Web Publishing using Java Stored Procedure*" code sample[7] gives you a complete step-by-step implementation of such application.

CONCLUSIONS

This paper describes how Java, JDBC and Web Services in the Oracle Database 10*g* address the requirements of Java developers and Web Service developers, in terms of standard support, productivity, performance, scalability, load-balancing, reliability, reach, and high-availability. As a result, Java and Web Services developer get portability, interoperability, faster time to market, sustainability of unpredictable workloads, reliability, and risk reduction. An exhaustive list of new Java, JDBC and Web Services features in Oracle Database 10g can be found in the related white paper, on the Oracle Technology Network[8].

---

[7] http://www.oracle.com/technology/sample_code/tech/java/jsp/samples/jwcache/Readme.html

[8] http://www.oracle.com/technology/tech/java/jsp/index.html

Java and Web Services Developers Perspective on the Oracle database 10g
May 2005
Author: Kuassi Mensah

Oracle Corporation
World Headquarters
500 Oracle Parkway
Redwood Shores, CA 94065
U.S.A.

Worldwide Inquiries:
Phone: +1.650.506.7000
Fax: +1.650.506.7200
oracle.com