Utilización de Oracle Java Procedures

Por Francisco Riccio 

Introducción

A partir de la versión Oracle Database 8i se tiene implementado una maquina virtual de Java como parte de los componentes internos de la base de datos. Los Java Procedures son librerías en Java registradas en la base de datos que se caracterizan por el buen desempeño que tienen sobre operaciones lógicas y matemáticas obteniendo mejor tiempo de respuesta versus el lenguaje PL/SQL en este aspecto. Dichas librerías Java se ejecutan en la máquina virtual Java de la base de datos.

La máquina virtual de Java instalada dentro de la base de datos tiene algunos limitantes versus el Java instalado en los sistemas operativos convencionales; algunos de estos son: ausencia de la librería Abstract Window Toolkit (AWT) y Swing para crear componentes gráficos; así mismo no está disponible actualizaciones del componente Java de la base de datos por separado. La versión Oracle Database 10g incluye la versión de Java 1.42 y la versión Oracle Database 11g incluye la versión de Java 1.5.

Oracle Database puede almacenar Java Source & Java Class y podría exponerlo como procedimientos, funciones o triggers de PL/SQL a las aplicaciones.

Este feature no está disponible en la edición XE.

Implementación

Es importante conocer que el componente de Java Virtual Machine instalado en la base de datos trabaja bajo un esquema de seguridad conocido como Java2 Security.

Desde Oracle Database 8i están disponible 2 tipos de roles para manejar la seguridad de las aplicaciones Java: JAVAUSERPRIV y JAVASYSPRIV (lleva todos los permisos Java disponibles en la base de datos). Oracle recomienda que evitemos el uso de estos roles y que mejor asignemos los permisos que necesitemos puntualmente. Los permisos que vamos asignando se guardan internamente en objetos llamados Permission los cuales son almacenados en las tablas: PolicyTable y PolicyTablePermission respectivamente.

Si requerimos de algunos permisos para que nuestros programas Java puedan trabajar de manera correcta lo hacemos a través del paquete DBMS_JAVA.

Los permisos más conocidos son java.io.FilePermission y el java.net.SocketPermission.

Java.io.FilePermission permite que nuestro código Java pueda realizar operaciones sobre archivos del sistema operativo. Por default está restringido.

```
SQL> execute
DBMS_JAVA.grant_permission('<usuario>','java.io.FilePermission','<directorio>|*','read,write,execute,delete');
```

Ejemplo:

```
SQL> execute DBMS_JAVA.grant_permission('SCOTT','java.io.FilePermission','*','read,write');
```

Java.net.SocketPermission, permite que nuestro programa pueda enviar tramas vía TCP/IP mediante sockets. Por default también está restringido.

```
SQL> execute
dbms_java.grant_permission('<usuario>','SYS:java.net.SocketPermission',<ip:puerto>','connect,resolve,accept,listen');
```

Ejemplo:

```
SQL> execute
dbms_java.grant_permission('SCOTT','SYS:java.net.SocketPermission','localhost:7000','connect,resolve,accept,listen');
```

Si deseamos otorgar permisos puntuales a un usuario para algunas operaciones restringidas en la base de datos debemos contamos con el rol JAVA_ADMIN.

Antes de realizar alguna implementación de Java Procedure primero debemos tener correctamente instalado la máquina virtual Java en nuestra base de datos.

```
SQL> select COMP_NAME,VERSION,STATUS from dba_registry where upper(comp_name) like '%JAVA%';
```

```
SQL> select COMP_NAME,VERSION,STATUS from dba_registry where upper(comp_name) like '%JAVA%';

COMP_NAME                                VERSION                        STATUS
---------------------------------------- ------------------------------ ---------------
JServer JAVA Virtual Machine             11.2.0.3.0                     VALID
Oracle Database Java Packages            11.2.0.3.0                     VALID
```

A continuación se presenta el desarrollo de cuatro diferentes ejemplos de cómo implementar un Java Procedure.

Ejemplo 1:

Realizaremos una función en Java que permita mostrar la fecha que se encuentra en el servidor de base de datos

a) Creamos el archivo Fecha.java con el siguiente contenido:

```
public class Fecha {

 public static java.sql.Date getFecha(){

       return new java.sql.Date(new java.util.Date().getTime());

 }

}
```

Opcional: Podemos compilar el programa en el sistema operativo:

java -c Fecha.java

Se creará el archivo: Fecha.class como producto de la compilación.

b) Registro del programa Java en la base de datos:

loadjava -user <usuario>/<password> <archivo.java|class|jar>

Podemos cargar directamente la fuente del programa (*.java) o el código compilado (*.class). Si cargamos el código fuente no es necesario cargar el código compilado.

Si lo ejecutamos nuevamente el comando loadjava reemplazará la librería anterior por la nueva que estemos cargando.

Nota 1:Si deseamos eliminar la librería cargada ejecutamos:

dropjava -user <usuario>/<password> <archivo.java|class|jar>

Nota 2: Si deseamos validar el registro de los programas Java en la base de datos consultamos el siguiente query.

SQL> select owner,object_name, object_type from dba_objects;

La información de las librerías de Java creadas y cargadas dentro de la base de datos se lleva almacenada en tablas internas creadas por Oracle en cada esquema, una de estas son: JAVA$OPTION y CREATE$JAVALOBTABLE.

Nota 3: Las librerías Java almacenadas en la base de datos se les conoce con el término LIBUNITS, el cual es como un análogo al término DLL en lenguaje C.

c) Publicación.

La publicación se puede realizar a través de Stored Procedures, Funciones o Triggers.

```
create or replace function Fecha return Date
IS LANGUAGE JAVA NAME 'Fecha.getFecha() return java.sql.Date';
/
```

Nota: La equivalencia del campo DATE de base de datos es la clase java.sql.Date en Java.

Utilizamos la función:

```
SQL> select Fecha from dual;
```

Ejemplo 2:

Crearemos una función en Java que consulte la fecha de la base de datos en un formato determinado y lo devuelva como un varchar.

a) Creamos el archivo BD.java con el siguiente contenido:

```
import java.sql.*;

import java.util.*;


public class BD

{


 public static String getFecha()

 {

  Connection cn = null;

  PreparedStatement stmt = null;

  String fecha=null;

  try

  {

   Properties prop = new Properties();

   prop.setProperty("user","sys");

   prop.setProperty("password","oracle");

   prop.setProperty("internal_logon","sysdba");

   Class.forName("oracle.jdbc.OracleDriver");

   cn=DriverManager.getConnection("jdbc:oracle:thin:@"+"127.0.0.1:1521/PRD",prop);

   stmt=cn.prepareStatement("select to_char(sysdate,'DD-MM-YYYY') as fecha_BD from
dual");

   ResultSet rs=stmt.executeQuery();
```

```
   rs.next();
   fecha=rs.getString("fecha_BD");
   rs.close();
   stmt.close();
  }
  catch(Exception e){
   System.out.println(e.getMessage());
  }
  finally{
   if (cn!=null){
    stmt=null;
    cn=null;
   }
  }
  return fecha;
 }

}
```

b) Registro del programa Java en la base de datos:

```
loadjava -user friccio/oracle BD.java
```

c) Publicación

```
create or replace function FechaBD return varchar
IS LANGUAGE JAVA NAME 'BD.getFecha() return String';
```

```
/
```

Utilizamos la función:

```
select FechaBD from dual;
```

Ejemplo 3:

Crearemos un procedimiento en Java que almacene el contenido de un BLOB en un archivo del sistema operativo.

a) Creamos el archivo Java_BLOB.java con el siguiente contenido:

```
CREATE OR REPLACE JAVA SOURCE NAMED "Java_BLOB" AS


 import java.lang.*;

 import java.sql.*;

 import oracle.sql.*;

 import java.io.*;


 public class Java_BLOB

 {


  public static void crearArchivoBinario(String parchivo, BLOB pBlob) throws Exception

  {

   //Crea un archivo en una ruta específica.

   File archivo = new File(parchivo);

   FileOutputStream outStream = new FileOutputStream(archivo);

   //Conseguimos el contenido binario.

   InputStream inStream = pBlob.getBinaryStream();
```

```
  //Obtenemos un arreglo de bytes que más adelante llevara el contenido del BLOB.
  int size = pBlob.getBufferSize();
  byte[] buffer = new byte[size];
  int length = -1;
  //Leyendo la información y escribiéndolo en el archivo.
  while ((length = inStream.read(buffer)) != -1)
  {
   outStream.write(buffer, 0, length);
   outStream.flush();
  }
  inStream.close();
  outStream.close();
 }

};
/
```

Nota 1: En esta ocasión se ha utilizado la sintaxis: CREATE OR REPLACE JAVA SOURCE NAMED, el cual ya realiza los pasos de compilación y publicación de la librería.

Nota 2: Si deseamos compilar de manera específica y ver sus errores ejecutamos los siguientes pasos:

SQL> ALTER java source "<Nombre Librería>" compile;

SQL> show errors java source "<Nombre Librería>"

Ejemplo:

```
SQL> ALTER java source "Java BLOB" compile;

Java altered.

SQL> show errors java source "Java_BLOB"
No errors.
```

b) Permisos:

Debido a que la aplicación Java escribirá en un directorio del sistema operativo necesitamos garantizar el permiso a la máquina virtual Java a un usuario específico de la base de datos a dicha acción.

SQL> grant JAVASYSPRIV to FRICCIO;

SQL> execute dbms_java.grant_permission('FRICCIO','java.io.FilePermission','*','read ,write, execute, delete');

Al usuario le asignamos el rol JAVASYSPRIV porque esperamos que el usuario pueda ejecutar operaciones sobre cualquier directorio del sistema operativo siempre y cuando el usuario tenga permisos en el mismo a nivel de plataforma.

c) Publicación:

create or replace procedure SPU_Java_BLOB (parchivo varchar, pblob blob)

IS LANGUAGE JAVA NAME 'Java_BLOB.crearArchivoBinario(java.lang.String, oracle.sql.BLOB)';

/

Utilizamos la función:

set serveroutput on

call dbms_java.set_output(2000);

declare

v_blob blob;

```
 photo BLOB default EMPTY_BLOB() ;
 v_raw raw(123);
 v_totalBytes number ;
begin
 /*El valor a ingresar se encuentra en formato Hexadecimal,
 teniendo su representacion en ASCII como @*/
 v_raw:='40';
 v_totalBytes:=UTL_RAW.length(v_raw);
 /*Se crea un BLOB temporal*/
 dbms_lob.createtemporary(v_blob,TRUE);
 /*Se escribe informacion en el BLOB temporal*/
 DBMS_LOB.WRITE(v_blob,v_totalBytes,1,v_raw);
 /*Se escribe el contenido del BLOB temporal a disco*/
 SPU_Java_BLOB('/tmp/binario.txt',v_blob);
end;
/
```

Ejemplo 4:

En este ejemplo se creará una función en Java que permita ejecutar un comando en el Sistema Operativo y dicha función esté disponible desde PL/SQL.

a) Creamos el archivo OS.java con el siguiente contenido:

```
import java.io.*;


public class OS
{


 public static String cmd(String pcomando){
  String output = "";
  String[] programa = null;
  String s="";
  if (System.getProperty("os.name").toLowerCase().indexOf("windows")==-1)
  {
   programa = new String[3];
   programa[0]="/bin/sh";
   programa[1]="-c";
   programa[2]=pcomando;
  }
  else
  {
   programa = new String[4];
   programa[0]="c:\\windows\\system32\\cmd.exe";
   programa[1]="/y";
   programa[2]="/c";
```

```
   programa[3]=pcomando;
  }
  Runtime objRt=Runtime.getRuntime();
  try{
   Process objProceso = objRt.exec(programa);
   BufferedReader stdInput = new BufferedReader(new
InputStreamReader(objProceso.getInputStream()));
   while ((s = stdInput.readLine()) != null) {
    output = output + s;
    System.out.println(output);
   }
   objProceso.waitFor();
  }
  catch(Exception e){
   System.out.println(e.getMessage());
  }
  return output;
 }

}
```

b) Registro del programa Java en la base de datos:

```
loadjava -user friccio/oracle OS.class
```

c) Publicación:

```
set serveroutput on
call dbms_java.set_output(2000);
declare
 v_programa varchar(255);
begin
 v_programa:=OS_BD('/bin/touch /tmp/reporte.txt');
 dbms_output.put_line(v_programa);
end;
/
```

Nota 1: Es importante conocer que si realizamos un export (exp) de esquema o full solo estaríamos llevando las clases java mientras los permisos Java deben ser obtenidos aparte mediante un script. El resultado del script y del export serán las dos fuentes para iniciar una restauración. El export datapump incluye ambas fuentes.

Nota 2: **My Oracle Support (MOS) Nota: 183825.1 (How to Backup and Restore Java Classes and Privileges only)**, cuenta con un script que nos permite obtener todos los permisos registrados en la base datos. El script se presenta a continuación:

```
spool setjvmprivs.sql
set echo off
set feedback off
set heading off
set linesize 80
set pagesize 1000
column stmt format a70 word_wrapped
select 'exec '||stmt
from (select seq, 'dbms_java.grant_permission('''||grantee||''','''||
```

```
        type_schema||':'||type_name||'','''||name||''','''||action||
        ''');' stmt
    from dba_java_policy
    where grantee not in ('JAVADEBUGPRIV', 'JAVASYSPRIV', 'JAVAUSERPRIV',
                'JAVA_ADMIN', 'JAVA_DEPLOY', 'SYS', 'PUBLIC') and
        type_name!='oracle.aurora.rdbms.security.PolicyTablePermission'
    union all
    select seq,'dbms_java.grant_policy_permission('''||a.grantee||''','''||
        u.name||''','''||permition||''','''||action||''');' stmt
    from sys.user$ u,
        (select seq, grantee,
            to_number(substr(name,1,instr(name,':')-1)) userid,
            substr(name,instr(name,':')+1,instr(name,'#') -
                instr(name,':')-1) permition,
            substr(name,instr(name,'#')+1 ) action
        from dba_java_policy
        where grantee not in ('JAVADEBUGPRIV', 'JAVASYSPRIV',
                'JAVAUSERPRIV', 'JAVA_ADMIN', 'JAVA_DEPLOY',
                'SYS', 'PUBLIC') and
            type_name =
              'oracle.aurora.rdbms.security.PolicyTablePermission') a
    where u.user#=userid) order by seq;
column stmt clear
set pagesize 24
set heading on
```

```
spool off
```

Nota 3: También si deseamos consultar las clases Java que tenemos registrado en la base de datos consultamos la vista **DBA_JAVA_CLASSES**.

Nota 4: JVM JIT es un compilador diseñado para la máquina virtual Java el cual transparentemente selecciona procedimientos Java para compilarlos en código nativo, en ventaja de conseguir mejores tiempos de respuesta. Este compilador está disponible a partir de la versión Oracle Database 11gR1.

Conclusión

Oracle Database 8i y las siguientes versiones nos dan la posibilidad de escribir ciertas rutinas en lenguaje Java aprovechando al máximo las dificultades que podríamos tener en el lenguaje PL/SQL al ser Java un lenguaje de programación más completo.

Sobre la seguridad, Oracle nos entrega un completo control sobre los permisos que puede ejecutar cada usuario sobre las librerías de Java registrada.

Debemos ser conscientes que si bien las librerías Java tienen mejor performance en la resolución de cálculos matemático y lógicos, en ningún momento sustituye al lenguaje PL/SQL, ya que dicho lenguaje provee mejor tiempo en la manipulación de la información dentro de la base de datos.

Publicado por Ing. Francisco Riccio. Es un IT Specialist en IBM Perú e instructor de cursos oficiales de certificación Oracle. Está reconocido por Oracle como un Oracle ACE y certificado en productos de Oracle Application & Base de Datos.

e-mail: francisco@friccio.com

web: www.friccio.com