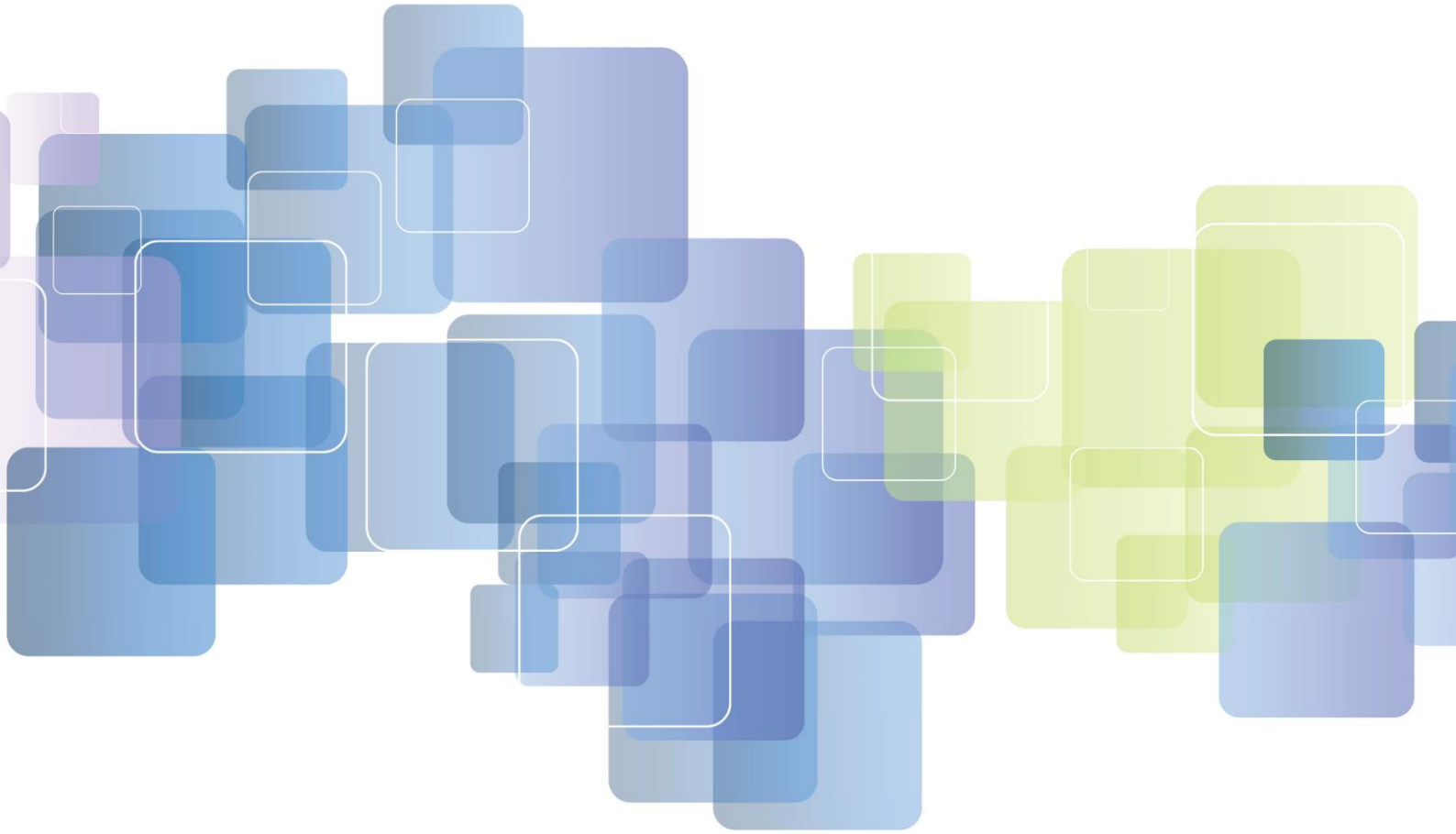
TecnetDati

Java e PL/SQL

Simona Rotolo



Con l'uscita di Oracle 8i è possibile costruire componenti software che accedono ai dati Oracle utilizzando in modo congiunto i linguaggi PL/Sql e Java. In tal modo è possibile sfruttare le capacità computazionali di Java e le facility del PL/Sql per l'elaborazione di insiemi consistenti di dati.
Le considerazioni che seguono vogliono dare una rassegna breve ma esaustiva sull'argomento, esaminando in modo approfondito due scenari e precisamente:

1. scrittura di componenti di business in Java che richiamano Stored Procedure (SP) per l'accesso ai dati
2. scrittura di SP che richiamano classi Java.

Come accedere a Java dal PL/SQL

In questo capitolo vedremo come è facile costruire una classe java, compilarla e memorizzarla in un Database Oracle facendola diventare a tutti gli effetti uno tra i suoi oggetti, più precisamente una JAVA STORED PROCEDURE che può essere utilizzata all'interno di programmi PL/Sql.
Per effettuare le prove descritte di seguito assicurarsi di avere almeno una versione 1.1.5 del Java Development Kit(JDK).

La prima cosa da fare è creare una classe specifica Java che include il metodo che si intende utilizzare (questa può essere scritta sia in Java standard che in Java+SQLJ) .

Nel nostro esempio creeremo una classe che cancellerà un file, il cui nome è preso in input.

La classe Java.io.file fornisce una serie di metodi per gestire i files e le directories.

```
public class java.io.File {
public boolean delete();
public boolean mkdir ();
}
```

Richiamando il metodo booleano java (delete()) per cancellare un file, otterremo TRUE se il file è stato cancellato, FALSE se non l'ha trovato.

Non si può richiamare direttamente il metodo della classe java da PL/Sql in quanto:

- per eseguire un metodo Java è necessario creare (istanziare) un oggetto e non si può istanziare un oggetto java direttamente da Pl/Sql
- i datatype in Java non corrispondono direttamente a quelli PL/Sql.

Occorre pertanto costruire una nuova classe Java e memorizzarla nel database Oracle; questa nuova classe:

- istanzierà un oggetto della classe file
- invocherà il metodo delete sull'oggetto
- restituirà un valore che potrà essere interpretato mediante costrutti PL/Sql

Scriviamo la nostra classe:

```
import java.io.File;
public class Jdelete {
public static String delete (String fileName) {
File myFile = new File (fileName);
boolean retval = myFile.delete();
if (retval) return "file cancellato"; else return "file non trovato";
} //method delete
} //class JDelete
```

Il metodo Jdelete.delete semplicemente instanzia un oggetto file utilizzando il filename specificato, di modo che si possa chiamare il metodo delete su quel file.
Già nella definizione della classe si può notare la diversa sintassi tra Java e PL/Sql.

Una volta scritta la nostra classe possiamo compilarla, testarla ed infine caricarla sul nostro database.

Compilare la classe Java. A tal fine occorre andare nell'apposita directory e richiamare il comando "javac" con il nome della classe da compilare.

Es: c:\javac Jdelete.java

Prima di caricarla su database è sempre meglio compilare e testare la classe java utilizzando gli strumenti di sviluppo presenti sul client (per esempio Jdeveloper o JBuilder) in modo da verificare il buon funzionamento del codice Java.

Caricare infine la classe sul database utilizzando l'utility Oracle **loadjava** che si lancia dalla linea di comando del S.O.

D:\Java>loadjava -user scott/tiger -oci8 -resolve JDelete.class

E' anche possibile caricare la classe dall'applicazione Oracle utilizzando l'apposita built-in DBMS_JAVA, che non fa altro che eseguire al suo interno la LOADJAVA.
Per vedere se l'oggetto è stato caricato occorre interrogare la vista USER_OBJECTS

A questo punto occorre dichiarare una JAVA STORED PROCEDURE, specificando al suo interno la clausola LANGUAGE JAVA, che registra le informazioni essenziali sul metodo java quali il nome, i parametri ed eventuali valori di ritorno.

In questa particolare SP ci sarà il richiamo al metodo della nostra classe Java e il mapping tra i datatype dei parametri del metodo e i tipi di Oracle.

Creazione della stored procedure che richiamerà il metodo Java:

```
CREATE OR REPLACE FUNCTION fn_DeleteFile(file IN VARCHAR2)
RETURN varchar2
AS LANGUAGE JAVA

NAME 'JDelete.delete (java.lang.String)
return String';
/
```

La dichiarazione della stringa che descrive il metodo(clausola NAME), corrisponde all'invocazione del metodo stesso.
I parametri specificati nella lista della stored procedure devono rispecchiare i parametri del metodo; in questo esempio viene specificato come parametro il tipo.

Per poter eseguire la nostra procedura Java l'utente Oracle deve avere le abilitazioni necessarie per accedere o modificare i file del sistema operativo. Riportiamo di seguito il comando Oracle per dare l'abilitazione all'utente

```
GRANT JAVASYSPRIV TO user;
```

Una volta compilata, la procedura PL/Sql può essere inclusa in altre stored procedure o richiamata direttamente come illustrato nel seguente esempio:

```
SQL> select fn_deleteFile('c:\temp\Scartati.txt') from dual;

FN_DELETEFILE('C:\TEMP\SCARTATI.TXT')
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
file cancellato

SQL> select fn_deleteFile('c:\temp\Scartati.txt') from dual;

FN_DELETEFILE('C:\TEMP\SCARTATI.TXT')
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
file non trovato
```

Nota

La versione 9i di Oracle fornisce Jdeveloper per sviluppare, testare, debuggare ed effettuare il deploy delle Java stored procedure. Inoltre l'utility loadjava permette di pubblicare automaticamente le classi java creando le corrispondenti Call Spec.

Come Accedere a PL/SQL da Java

Abbiamo visto come accedere a Java da PL/Sql, altrettanto facile è richiamare delle stored procedure Pl/Sql all'interno del codice Java.

Occore innanzitutto creare una procedura, funzione o package e memorizzarla sul database Oracle.

Nel nostro esempio, definiamo una semplice procedura che ricevendo in input il nominativo di un impiegato, ne restituisce il codice con cui è stato registrato.

```
CREATE OR REPLACE PROCEDURE PR_IMPIEGATO (P_NOME IN VARCHAR2,
P_COGNOME IN VARCHAR2, P_COD_IMPIEGATO OUT VARCHAR2)IS
BEGIN

SELECT cod_impiegato
INTO p_cod_impiegato
WHERE nome= p_nome
AND cognome= p_cognome

  EXCEPTION
    WHEN OTHERS THEN
      P_COD_IMPIEGATO:='ERRORE';
END PR_PROVA;
/
```

Definiamo ora una classe Java che richiamerà al suo interno la procedura appena definita.

```
import.java.sql.*;

public class EstrazioneImpiegati extends Objects {

 private Connection dbCon;

 public static void main (String args[]){
  try{

     Class.forName("oracle.jdbc.driver.OracleDriver");
     // Istruzione per stabilire una connessione con il database
     // nel quale risiede la nostra stored procedure
     dbCon=
     DriverManager.getConnection
     ("jdbc:oracle:thin:@host:porta:SID",user,psw);

     // Istruzione per richiamare la stored procedure
```

```
        CallableStatement cs=conn.prepareCall("{call
         pr_impiegato(?,?,?)}");

        // definizione dei parametri in Input e Output
        cs.setString(1, "Paolo");
        cs.setString(2, "Rossi");
        cs.registerOutParameter(3,OracleTypes.VARCHAR2);
        cs.execute();

        // stampa a video del parametro di Output
        System.out.println(cs.getString(3));

        //chiusura connessione
        conn.close();

    //Gestione errori
        } catch(SQLException e){
        System.out.println(e.getMessage());
        }
      }
}
```

Poniamo ora l'attenzione su alcune istruzioni utilizzate all'interno del codice della nostra classe java.

- metodo prepareCall per richiamare la stored procedure. Nella call alla procedura occorre passare tanti ? quanti sono i suoi parametri sia in input che in output

  CallableStatemets cs=conn.prepareCall ("{call pr_impiegato(?,?,?)}");

- metodo setString per settare i parametri da passare in input, in base al tipo e alla loro posizione nella chiamata alla procedura (Call)

  cs.setString(1, "Paolo");
  cs.setString(2, "Rossi");

- metodo registerOutParameter per dichiarare il tipo dei parametri in output e la loro posizione nella call

  cs.registerOutParameter(3,OracleTypes.VARCHAR2);

Vediamo ora come può essere letto da java il risultato di una procedura che restituisce un resultset di dati.

Definiamo la procedura PL/Sql che restituisce in output un cursore

```
PACKAGE Pg_dataAssunzioneIS

- - dichiarazione del tipo cursore - -
TYPE CURSORE IS REF CURSOR;

- - dichiarazione procedura - -
PROCEDURE pr_GetDataAssunzione(CUR OUT CURSORE);
END Pg_dataAssunzione;
/
CREATE OR REPLACE PACKAGE BODY Pg_dataAssunzioneIS AS

- - codice della procedura - -
PROCEDURE Pr_GetDataAssunzione (CUR OUT CURSORE) IS
BEGIN
- - cursore da restituire - -
  OPEN CUR FOR
                SELECT TO_CHAR(DATA_ASSUNZIONE, 'dd/mm/yyyy
                hh24:mi:ss')
                 FROM IMPIEGATI
                 ORDER BY 1;

END;
END Pg_dataAssunzioneIS;
/
```

In questo caso occorrerà modificare la nostra classe java utilizzando le seguenti istruzioni per ospitare e leggere il parametro di output della stored procedure PL/Sql.

- impostare la variabile di output col tipo cursore
  `cs.registerOutParameter(1,OracleTypes.Cursor)`

- eseguire la procedura e memorizzare il cursore restituito nel resultset di java
  ```
  cs.execute();
  ResultSet rset= (ResultSet)cs.getObject(1);
  ```

- scorrere il cursore per leggere i dati:
  ```
  while (rset.next()){
                  sData= rset.getString(1);
          }
  ```

Criteri di scelta tra Java e PL/SQL

La scelta di Oracle di inserire Java all'interno del Database Server e nell'Application Server è stata dettata dalla necessità di:

- Fornire una piattaforma veloce e scalabile.
- Consentire la gestione di quantità notevoli di dati tramite tecnologie non completamente compatibili con PL/SQL quali XML, CORBA e EJB.
- Aggiungere funzionalità al server: la presenza di java fornisce una serie di servizi/programmi aggiuntivi che possono essere eseguiti direttamente sul server con un incremento di performance rispetto ai tradizionali programmi java

PL/SQL è ottimizzato per l'utilizzo di SQL: dal momento che utilizza lo stesso datatype system è, per efficienza e sicurezza, più adatto per scrivere applicazioni che utilizzano in maniera intensiva SQL ma, a differenza di Java, non supporta costrutti come ereditarietà, polimorfismo e modelli di componenti che sono propri di ambienti orientati agli oggetti.

Principalmente PL/SQL risulta migliore nei seguenti casi:

- il datatype system è più facile da utilizzare. Le operazioni numeriche di PL/SQL utilizzano il formato di Oracle che ha una sintassi naturale e precisa, mentre nel codice Java è necessario includere operazioni di conversione dei formati che riducono le performance.
- Per applicazioni che utilizzano intensamente SQL, il PL/SQL è molto più veloce di Java, soprattutto quando un ingente numero di dati necessita di essere convertito da types SQL a types per Java.
- Per applicazioni come i Triggers, che sono caratterizzati più dall'alta velocità di avvio dell'interprete PL/SQL o della java virtual machine che dalla velocità di esecuzione dell'applicazione, PL/SQL è più adatto. Questo perché invocare la VM di PL/SQL è molto più rapido che invocare la JVM.
- PL/SQL è ottimizzato per le operazioni di concatenamento delle stringhe.
- PL/SQL è molto più veloce negli accessi al database.

Java, invece, è ottimizzato per operazioni computazionali e per la gestione di oggetti distribuiti: è un linguaggio orientato agli oggetti con un system type molto ricco, modelli dei componenti e altre facility, quali, ad esempio, il supporto per CORBA e EJB. Fornisce la possibilità di utilizzare array multidimensionali e dal momento che è object-oriented è molto pratico per scrivere gerarchie di classi (ereditarietà).
La JVM è ottimizzata per eseguire programmi con poco SQL e la gestione dei datatypes, anche se meno precisa, è molto più veloce. Fornisce la possibilità di utilizzare tecnologie distribuite come servizi CORBA e EJB da utilizzare al posto delle stored procedure. CORBA e EJB possono venire utilizzati nell'architettura di applicazioni per i seguenti motivi:

- integrazioni eterogenee: dal momento che CORBA isola il client dall'implementazione sul server, esso fornisce un meccanismo efficiente per integrare applicazioni scritte in linguaggi diversi e di distribuire le applicazioni su servers differenti ottimizzando la scalabilità e le performances.
- system type eterogeneo: dal momento che con le stored procedure, client e server comunicano usando il datatype system, i clients e servers CORBA comunicano utilizzando IDL type system. E gli EJB clients e server utilizzano l'RMI

type system. Entrambi sono molto più flessibili di un SQL type system. Forniscono facilita' di gestione per le istanze di classi, sottoclassi, array multidimensionali, eccezioni attraverso clients e servers.

Un approccio Java consente di adottare soluzioni più modulari per la definizione delle interfacce delle applicazioni e per i componenti sul server. Questo tipo di approccio rende molto più semplice mantenere, estendere e gestire l'applicazione.
Java e' ottimizzato per operazioni computazionali, cioè per applicazioni che contengono codice complesso e operazioni CPU based. In tal caso Java è molto più veloce che PL/SQL.
Per esempio Java ha un supporto nativo per il floating point dove, invece, PL/SQL utilizza quello di SQL.

In fase di disegno delle componenti si dovrà quindi valutare, in base ai concetti espressi in precedenza, quale delle due tecnologie è più adatta alla funzionalità che si vuole implementare.

Bibliografia

Oracle8i, *Java Stored Procedures Developer's Guide*
Steven Feuerstein, *Calling Java from PL/S*

Tecnet Dati s.r.l.
C.so Svizzera 185 –
10149 - Torino (TO), Italia
Tel.: +39 011 7718090 Fax.: +39 011 7718092
P.I. 05793500017 C.F. 09205650154
www.tecnetdati.com