Java DataBase Connectivity

connettore per database

In informatica **JDBC** (*Java DataBase Connectivity*)[1], è un connettore (*driver*) per database che consente l'accesso e la gestione della persistenza dei dati sulle basi di dati da qualsiasi programma scritto con il linguaggio di programmazione Java, indipendentemente dal tipo di DBMS utilizzato. È costituito da un'API object oriented orientata ai database relazionali, raggruppata nel package *java.sql*, che serve ai client per connettersi a un database fornendo i metodi per interrogare e modificare i dati.

La piattaforma Java 2 Standard Edition contiene le API JDBC, insieme all'implementazione di un bridge JDBC-ODBC, che permette di connettersi a database relazionali che supportino ODBC, che è in codice nativo e non in Java. Tipicamente ciascun DB ha il suo specifico driver JDBC per interfacciarsi con l'applicazione. Spesso i framework di persistenza in ambito Java (es. Hibernate) nella loro implementazione a più alto livello si interfacciano a più basso livello proprio con uno strato software JDBC.

Architettura

L'architettura di JDBC, così come quella di ODBC, prevede l'utilizzo di un “driver manager”, che espone alle applicazioni un insieme di interfacce standard e si occupa di caricare a “run-time” i driver opportuni per “pilotare” gli specifici DBMS. Le applicazioni Java utilizzano le "JDBC API" per parlare con il JDBC driver manager, mentre il driver manager usa le JDBC driver API per parlare con i singoli driver che pilotano i DBMS specifici. Esiste un driver particolare, il "JDBC-ODBC Bridge", che consente di interfacciarsi con qualsiasi driver ODBC in ambiente Windows.

Tipi di driver

Esistono driver free e commerciali per la maggior parte dei server di database relazionali. I driver possono essere di quattro tipi:

- Tipo 1, il JDBC-ODBC Bridge
- Tipo 2, API nativa
- Tipo 3, protocollo di rete
- Tipo 4, protocollo nativo

Panoramica dell'API

JDBC ammette che esistano diverse implementazioni e vengano utilizzate dalla stessa applicazione. L'API fornisce un meccanismo che carica dinamicamente i driver appropriati e li registra nel JDBC Driver Manager. Esso funge da fabbrica di connessioni.

Le connessioni JDBC supportano la creazione e l'esecuzione delle istruzioni. Esse possono essere comandi SQL come INSERT, UPDATE, DELETE, interrogazioni come SELECT o chiamate a stored procedure. I tipi di istruzioni supportati sono:

- *Statement* - l'istruzione viene inviata al database di volta in volta;
- *Prepared Statement* - l'istruzione viene compilata una sola volta, in modo che le chiamate successive siano più efficienti;
- *Callable Statement* - usati per chiamare le stored procedure.

I comandi di scrittura come INSERT, UPDATE e DELETE restituiscono un valore che indica quante righe sono state coinvolte (inserite, modificate, cancellate) nell'istruzione. Essi non restituiscono altre informazioni.

Le interrogazioni (query) restituiscono un result set (classe *ResultSet*). È possibile spostarsi nel result set riga per riga (tramite il metodo *next()*). Si può accedere alle colonne di ogni singola riga chiamandole per nome o per numero. Il result set può essere costituito da un numero qualsiasi di righe. Esso comprende dei metadati che indicano il nome, il tipo e le dimensioni delle colonne.

Esiste un'estensione di JDBC che permette, tra le altre cose, l'uso di result set scorribili e di cursori lato client. Si veda la documentazione di Sun Microsystems per maggiori informazioni.

Le eccezioni

Tutti i metodi delle API JDBC possono lanciare eccezioni, in quanto la connessione al DBMS in ogni momento può subire una interruzione o comunque si possono verificare errori nell'esecuzione dei comandi SQL. Tutte le eccezioni di JDBC derivano dalla classe *SQLException* e possono anche essere concatenate tra loro più eccezioni diverse. Ogni eccezione contiene un messaggio descrittivo, una stringa contenente lo stato SQL (conforme a quanto indicato nella specifica XOPEN SQL) e un intero contenente un codice errore addizionale specifico per il particolare driver o sorgente utilizzati.

Esempi

Il metodo Class.forName() carica la classe del driver JDBC. La linea seguente carica il driver per *mioDbms* nell'applicazione.

```
Class.forName( "com.mioDbms.mioDriver" );
```

Poi, il metodo DriverManager.getConnection() crea una connessione.

```
Connection conn = DriverManager.getConnection(
    "jdbc:mioDbms:altri dati utili per il driver",
    "mioLogin",
    "miaPassword" );
```

La stringa da utilizzare dipende dal driver JDBC che useremo. Inizia sempre con "jdbc:", il resto varia a seconda del prodotto scelto. Una volta stabilita la connessione, occorre passare una istruzione.

```
Statement stmt = conn.createStatement();
stmt.executeUpdate( "INSERT INTO miaTabella( nome ) VALUES ( 'andrea' ) " );
```

I dati vengono prelevati dal database col classico meccanismo delle query. L'esempio sottostante mostra come creare ed eseguire un'interrogazione:

```
Statement stmt = conn.createStatement();
ResultSet rs = stmt.executeQuery( "SELECT * FROM miaTabella" );
while ( rs.next() ) {
   int numeroColonne = rs.getMetaData().getColumnCount();
   for ( int i = 1 ; i <= numeroColonne ; i++ ) {
      // I numeri di colonna iniziano da 1.
      // Vi sono diversi metodi che convertono il valore di una colonna in un certo tipo.
      // Vedi la documentazione per una lista delle conversioni valide.
      System.out.println( "COLONNA " + i + " = " + rs.getObject(i) );
   }
}
rs.close();
stmt.close();
```

È raro, però, che un programmatore Java scriva codice in questo stile. Il modo più diffuso è inserire la logica del database in una classe differente e passare le stringhe SQL già elaborate (magari derivanti anch'esse da un'altra classe) e la connessione ai metodi che ne hanno bisogno.

Un esempio di Prepared Statement. Si utilizza la connessione dell'esempio precedente.

```
 try {

    ResultSet rs;
    PreparedStatement ps = conn.prepareStatement(
             "SELECT i.*, j.* FROM Omega i, Zappa j WHERE i = ?
AND j = ?" );
    // Nel prepared statement ps, i punti di domanda denotano le
variabili in input,
    // che possono essere passate attraverso una lista di
parametri, per esempio.

    // Il codice seguente sostituisce i punti di domanda con
stringhe o interi.
    // Il primo parametro indica la posizione in cui il valore va
inserito,
    // il secondo parametro è il valore da inserire.
    ps.setString(1, "Poor Yorick");
    ps.setInt(2, 8008);

    // Il ResultSet rs riceve la risposta del database.
    rs = ps.executeQuery();
    while ( rs.next() ) {
       int numeroColonne = rs.getMetaData().getColumnCount();
        for ( int i = 1 ; i <= numeroColonne ; i++ ) {
           System.out.println( "COLONNA " + i + " = " +
rs.getObject(i) );
     }

    rs.close();
    ps.close();

 } catch (SQLException e) {
    // gestione delle eccezioni
 }
```

Seguono alcuni esempi di conversioni tra il DBMS e Java.

Tipi Oracle	Metodi Java
CHAR	setString()
VARCHAR2	setString()
LONG	setString()
NUMBER	setBigDecimal()
	setBoolean()
	setByte()
	setShort()
	setInt()
	setLong()
	setFloat()
	setDouble()
INTEGER	setInt()
FLOAT	setDouble()
CLOB	setClob()
BLOB	setBlob()
RAW	setBytes()
LONGRAW	setBytes()
DATE	setDate()
	setTime()
	setTimestamp()

Chiusura delle connessioni

Il rilascio delle risorse allocate durante le operazioni su database, in particolare l'oggetto *connection*, è particolarmente critica, in quanto il numero totale delle connessioni disponibili è limitato e normalmente la connessione al DB non viene rilasciata automaticamente quando non è più utilizzata.

Per essere sicuri che una connessione sia chiusa correttamente, anche in caso di eccezione, conviene utilizzare la *finally*. Inoltre bisogna prestare particolare attenzione a non sollevare ulteriori eccezioni nel blocco finally:

```
Connection con=null;
try {
  con = DriverManager.getConnection ( URL, “”, “”);
  // utilizzo la connessione
  ...
} catch(Exception e){
  ...
} finally {
  if (con != null) con.close();
}
```

Note

1. ^ Java SE Technologies - Database

Voci correlate

- DBMS
- ODBC
- SQL
- Cursore (basi di dati)

Altri progetti

- Wikimedia Commons contiene immagini o altri file su **JDBC**

Collegamenti esterni

- (EN) *Sun tutorial* , su *java.sun.com*.
- (EN) *Sintassi delle API in Java 2 1.4.1* , su *java.sun.com*.
- (EN) *Duke's Bakery - A JDBC Order Entry Prototype* , su *java.sun.com*.
- (EN) *DBAccessor: A JDBC Wrapper Package* , su *java.sun.com*.
- (EN) *List of Java Open Source Databases with JDBC drivers* , su *java-source.net*.

Portale Informatica: accedi alle voci di Wikipedia che trattano di Informatica