JAVA DATABASE CONECTIVITY

TOMA Ana

Universitatea Tehnică a Moldovei

Abstract: *În articolul dat este descris modul de utilizare a bibliotecii JDBC, arhitectura ei, tipurile de drivere utilizate. Sunt descrise funcționalitățile puse la dispoziția programatorilor de Java Database Connectivity pentru gestiunea informațiilor reținute într-o bază de date. Se relevă modul de utilizarea API-ului Java Database Connectivity pentru a realiza operații de bază cu informațiile stocate într-o bază de date MySQL din contextul unei aplicații Java. De asemenea, sunt prezentate modalități de folosire a unor mecanisme avansate oferite de Java Database Connectivity (obiecte deconectate) pentru optimizarea transferului de date între diferite niveluri ale unei aplicații.*

Cuvinte cheie: *baza de date, SQL, Java , MySQL, driver, maşină virtuală.*

1. Introducere în JDBC

JDBC (Java Database Connectivity) este o interfaţă standard SQL de acces la baze de date. JDBC este constituită dintr-un set de clase si interfețe scrise în Java, furnizând mecanisme standard pentru proiectanţii aplicaţiilor de baze de date. Folosind JDBC este usor sa transmitem secvente SQL catre baze de date relationale. Cu alte cuvinte, nu este necesar să scriem un program pentru a accesa o bază de date Oracle, alt program pentru a accesa o baza de date Sybase şi aşa mai departe. Este de ajuns să scriem un singur program folosind API-ul JDBC şi acesta va fi capabil să trimită secvenţe SQL bazei de date dorite. Bineînţeles, scriind codul sursă în Java, ne este asigurataă portabilitatea programului. Deci, iata doua motive puternice care fac combinaţia Java - JDBC demnă de luat în seamă.

Folosind JDBC, un program poate:

-Stabili conexiuni cu diferite sisteme de baze de date;

- Executa comenzi (SQL) pentru a crea, actualiza, manipula datele şi a recepţiona rezultate;
- Inspecta și manipula meta-datele bazei de date.

JDBC este doar un API care permite programului să interacționeze cu un sistem de gestiune a bazelor de date. Serverul de baze de date trebuie sa existe separat: Java DB (fost Apache Derby) sau MySQL.

2. Arhitectura JDBC

Interfețele şi clasele pentru JDBC se găsesc în pachetul java.sql. O aplicaţie JDBC utilizează unul sau mai multe drivere din pachetul java.sql care sunt utilizate de către clasa DriverManager. Driverele sunt specifice bazelor de date, deci pentru fiecare tip de bază de date se utilizează un driver special. În aceeaşi aplicaţie putem lucra cu baze de date diferite, deci implicit si cu mai multe drivere. Putem avea nevoie de un driver care comunica cu o bază de date Oracle aflata la distanță și de un altul care reprezintă legătura spre driverul ODBC local şi comunică cu un server SQL. In aplicaţii comunicaţia cu o bază de date necesită urmatorii pasi:

1. Se apelează la DriverManager cerand un driver specific pentru baza de date;
2. Driverul specific crează legatura cu baza de date şi returneaza un obiect de tip Connection;
3. Cu ajutorul obiectului de tip Connection se crează un obiect Statement care conţine şi o cerere SQL către baza de date;
4. Obiectul Statement returnează rezultatele într-un obiect ResultSet.

3. Drivere şi manageri de drivere

Driver - această interfață gestionează comunicațiile cu serverul de baze de date. Veți interacționa direct cu obiectele Driver foarte rar. În schimb, utilizați obiecte DriverManager, care gestionează obiecte de acest tip. Rezumă, de asemenea, detaliile asociate cu lucrul cu obiectele Driver.

Conexiune - această interfață gestionează cu toate metodele de contactare a unei baze de date. Obiectul de conexiune reprezintă contextul de comunicare, adică, toată comunicarea cu baza de date este numai prin obiectul de conectare.

Instrucțiune - se utilizează obiectele create de această interfață pentru a trimite instrucțiunile SQL la baza de date. Unele interfețe derivate acceptă parametri în plus față de executarea procedurilor stocate.

ResultSet - aceste obiecte stochează datele preluate dintr-o bază de date după ce executați o interogare SQL folosind obiecte de declarație. Acționează ca un iterator pentru permite deplasarea prin datele sale (figura 1).

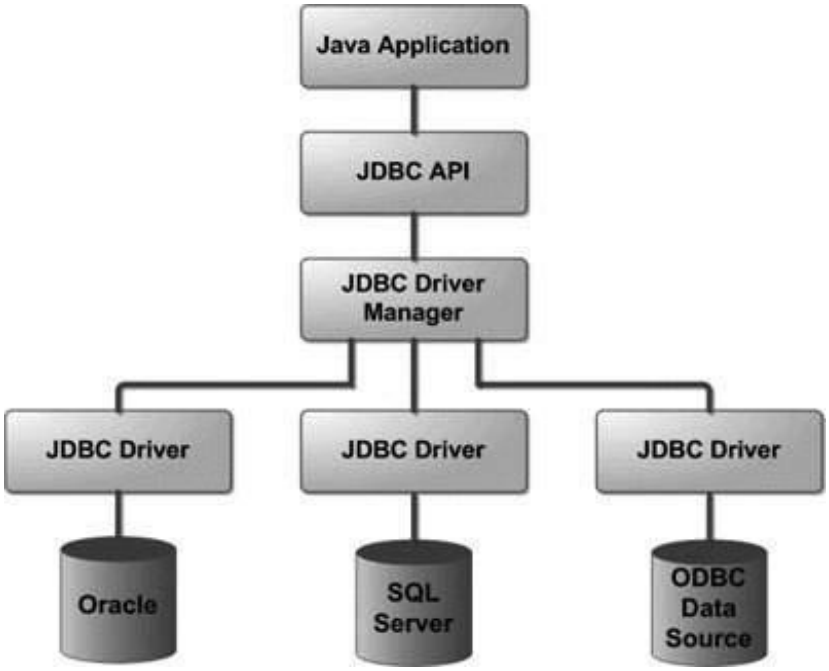

Fig. 1. Drivere JDBC

4. Conectarea la o baza de date

Pentru fiecare motor de baze de date, fabricantul pune la dispozitie o clasa driver. În cazul MySQL aceasta se numeşte com.mysql.jdbc.Driver, şi se găseşte în jar-ul mentionat mai sus. Încărcarea driver-ului în sistem se face printr-un apel: Class.forName("com.mysql.jdbc.Driver");

Pasul următor este obţinerea unei conexiuni (obiect Connection) la bază de date, cu precizarea:

-adresei server-ului ;

-numelui bazei de date;

-utilizatorului ;

-parolei:

String url = "jdbc:mysql://121.0.0.1:3306/mydatabase";

Connection conn = DriverManager.getConnection(url, name, pass).

Câmpul sub-protocol denumeşte tipul de driver ce trebuie folosit pentru realizarea conexiunii şi poate fi odbc, oracle, sybase, db2 etc. Identificatorul bazei de date este un indicator specific fiecărui driver care specifică baza de date cu care aplicaţia doreşte să interacţioneze. În funcţie de tipul driver-ului acest identificator poate include numele unei maşini gazdă, un număr de port, numele unui fişier sau al unui director: jdbc:odbc:testdb, jdbc:oracle:thin@persistentjava.com:1521:testdb, jdbc:sybase:testdb jdbc:db2:testdb. La primirea unui JDBC URL, DriverManager-ul va parcurge lista driver-elor înregistrate în memorie, pâna când unul dintre ele va recunoaşte URL-ul respectiv. Daca nu există nici unul potrivit, atunci va fi lansată o excepţie de tipul SQLException, cu mesajul ”no suitable driver”.

Metoda folosită pentru realizarea unei conexiuni este getConnection din clasa DriverManager şi poate avea mai multe forme:

- Connection c = DriverManager.getConnection(url);
- Connection c = DriverManager.getConnection(url, username, password);
- Connection c = DriverManager.getConnection(url, dbproperies).

O conexiune va fi folosită pentru:

- crearea de secvențe SQL ce vor fi folosite pentru interogarea sau actualizarea bazei;
- aflarea unor informaţii legate de bază de date (meta-date).

Clasa Connection asigură suport pentru controlul tranzacțiilor din memorie către baza de date prin metodele commit, rollback, setAutoCommit.

5. Mapări

Maparea claselor Java pe tabele dintr-o bază relațională se realizează prin configurarea unui fișier XML sau prin folosirea de adnotări Java. Atunci când se alege prima variantă, Hibernate poate genera un schelet de cod pentru clasele persistente. Dacă se folosesc adnotările Java, acest lucru nu mai este necesar. Hibernate poate folosi oricare dintre cele două variante pentru a menţine schema bazei de date. Hibernate oferă posibilitatea de a realiza relații unu-la-unu sau mulți-la-mulți între clase. De asemenea, se pot realiza asocieri reflexive între un obiect și mai multe instanţe în acelasi timp, relaţie de tip unu-la-mulți. Configurările necesare pentru lucrul cu Hibernate se realizează în fişierul hibernate.cfg.xml. Acest fişier conţine informaţii pentru realizarea conexiunii JDBC precum:

1. propietăți – property,
2. connection.driver_class,
3. connection.url,
4. connection.username,
5. connection.password,
6. connection.pool_size - folosit pentru configurarea pool-ului,
7. dialect - specifică dialectul pe care Hibernate îl va converti,
8. hbm2ddl.auto - activează generarea schemei bazei de date direct în bază,
9. mapări persistente pentru clasele POJO – mapping,
10. class - numele clasei mapate referitor la pachetul în care se află,
11. resource - localizează clasa folosind java.lang.ClassLoader.

6. Exemplu de conectare la o bază de date

```
package jdbc;
import java.sql.*;
public class SimpleJDBC {
public static void main( String args[]){
        try{
            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
            String databaseName = "jdbc:odbc:Inventory";
            Connection con
=DriverManager.getConnection(databaseName,"username","password");
            Statement stmt = con.createStatement();
            ResultSet rs = stmt.executeQuery("select * from Inventory");
            while( rs.next()){
                System.out.println(rs.getString(1)+":"+rs.getFloat(2));
            }
        }
        catch( Exception e ){
            e.printStackTrace();
        }
    }
}
```

Concluzii

Prin JDBC se încearcă realizarea unei interfeţe între baze de date și Java. JDBC este folosit pentru a conecta un program Java pe ușa din spate la o baza de date, pentru a șterge sau pentru a introduce noi resurse. JDBC se găseşte în pachetul java.sql și reprezintă o unealtă indispensabilă uneori pentru construirea de aplicaţii pentru WEB**.**

Bibliografie

1. *Arhitectura JDBC* – Resursă electronica, [Regim de acces]: https://corejava25hours.com/2016/09/19/jdbc/
2. *The Java Database Connectivity* (JDBC) – Resursă electronică [Regim de acces]: http://www.oracle.com/technetwork/java/javase/jdbc/index.html
3. *JDBC – Introduction* – Resursă electronica,[Regim de acces]: https://www.tutorialspoint.com/jdbc/jdbc-introduction.htm
4. *JDBC - SQL Syntax* – Resursă electronica, [Regim de acces]: https://www.tutorialspoint.com/jdbc/jdbc-sql-syntax.htm
5. *JDBC - Database Connections* – Resursă electronica, [Regim de acces]: https://www.tutorialspoint.com/jdbc/jdbc-db-connections.htm