Datenbankprogrammierung mit Java

Michael Dienert

27. März 2002

1 Java DataBase Connectivity

Eines der Hauptziele bei der Entwicklung von Java war die Plattformunabhängigkeit der Sprache. Und diese Vorgabe wollte man natürlich auch bei der Kommunikation zwischen einem Java-Programm und einer Datenbank einhalten.

Da SUN aber nicht wissen kann, mit welcher Datenbank auf welcher Plattform ein beliebiges Java-Programm zusammenspielen wird, haben die SUN-Entwickler die ***J**ava **D**ata**B**ase **C**onnectivity* definiert.

Alle "Klassen", die zusammen die JDBC bilden, sind im Paket `java.sql` zusammengefasst. Hier steht absichtlich "Klassen", denn eigentlich stehen im `java.sql` - Paket keine Klassen sondern Interfaces, also leere Klassen ohne Methoden und Eigenschaften. Und genau diese Interfaces ermöglichen es, eine *einzige, wohldefinierte* Schnittstelle zwischen Java und einer beliebigen Datenbank zu haben:

- Aus der Sicht eines Java-Programms stellen die JDBC Interfaces in `java.sql` eine *einheitliche* Fassade dar, die vor einer beliebigen Datenbank steht.
- Die Implementierung der Interfaces wickeln die eigentliche Kommunikation mit der Datenbank hinter den Kulissen ab.
- Diese Implementierung wird als JDBC-**Treiber** bezeichnet.
- Für jede Datenbank wird also ein spezieller JDBC-Treiber, sprich die Implementierung der `java.sql`-Interfaces, benötigt.

2 Der JDBC-Treiber

Das Wort *Treiber* für die tatsächliche Implementierung der Interfaces in `java.mysql` ist ein bisschen unglücklich gewählt. Eines der Interfaces heisst nämlich tatsächlich `Driver`, aber mit JDBC-Treiber ist nicht nur dieses eine Interface gemeint, sondern alle. Aus diesem Grund wäre die Bezeichnung *Bibliothek* viel exakter. Da aber in vielen JDBC-Büchern von einem JDBC-Treiber die Rede ist, sollte man den Begriff kennen.

Nun hängt ja wie gesagt, der Treiber von der verwendeten Datenbank ab. In allen folgenden Beispielen wird als Datenbank *my*SQL verwendet. Ein hierfür frei [1] erhältlicher Treiber ist zum Beipiel die Version von Mark Matthews, der `mm.mysql.jdbc`-Treiber (die `mm.mysql.jdbc`-Bibliothek).

2.1 Installation der `mm.mysql.jdbc`-Bibliothek

Der Java-Compiler `javac` und die Laufzeitumgebung `java` müssen beide Zugriff auf die Klassen der `mm.mysql.jdbc`-Bibliothek haben. Damit diese Bibliothek gefunden wird, hat man zwei Möglichkeiten:

1. Man installiert die Bibliothek in ein *beliebiges* Verzeichnis. Z.B. in ein Unterverzeichnis von `/usr/local/lib`. Damit javac und java die Klassen dort auch finden, muss man anschliessend den Pfad zu diesem Unterverzeichnis in die Umgebungsvariable `CLASSPATH` einfügen.
2. Man installiert die Bibliothek an eine Stelle, an der sie von `javac` und `java` auch ohne Angabe in CLASSPATH gefunden wird:

[1] Diese Bibliothek steht unter der *GNU Library General Public License*. Selbstverständlich sind die java-Quellen dabei. Sie sind sehr gut kommentiert, so dass ein Blick in den Quellcode der einzelnen Klassen sehr hilfreich sein kann.

`../lib/jre/ext`. Dabei steht `jre` für *Java Runtime Environment* und `ext` steht für *external*. In dieses Verzeichnis kommen also Klassenbibliotheken, die nicht von SUN sondern von anderen Anbietern stammen.

Auf dem Debian-Linux in R25 sieht der absolute Pfad so aus:

```
/usr/lib/j2re1.3/lib/ext/mm.mysql.jdbc-1.2c
```

Von diesen beiden Möglichkeiten ist die zweite die bessere: Änderungen an der Umgebungsvariablen CLASSPATH sollten bei den neuren Java-Versionen möglichst vermieden werden.

Eine Bibliothek besteht naturgemäss aus vielen Dateien. Um diesen Haufen handlicher zu machen, wurde die `mm.mysql.jdbc`-Bibliothek mit dem Linux-Kommando **`tar`** in eine Archivdatei gesteckt. Mit der Kommandozeile

```
tar -xvf mm.mysql.jdbc-1.2c.tar
```

lässt sich dieses Archiv wieder auspacken.

3 Die Klassen der jdbc-Bibliothek

3.1 `Driver` und `DriverManager`

Das Interface `Driver` ist zusammen mit der Klasse `DriverManager` für den Aufbau der Verbindung mit einer Datenbank verantwortlich.

Warum gibt es da nun einen `Driver` und auch noch einen `DriverManager`? Stellen Sie sich einfach vor, dass Ihr jdbc-Java-Programm sehr anspruchsvoll ist und nicht nur mit einer Datenbank zusammenarbeitet, sondern mit mehreren. Abb. 1 zeigt nun die Rolle von `DriverManager`. Für jede dieser Datenbanken benötigt man einen eigenen Treiber (= Driver). Der `DriverManager` wählt nun je nach Zugriff den richtigen Treiber zur richtigen Datenbank aus.

Wie geschieht aber nun die Auswahl? Im Beispielprogramm im Anhang A wird ein `Connection`-Objekt mit folgender Zeile erzeugt:

```
Connection dbVerbindung
        = DriverManager.getConnection(dbURL);
```

Der Anfang des Strings `dbURL` hat dabei folgenden Wert:

```
dbURL="jdbc:mysql://localhost/cd_datab..."
```

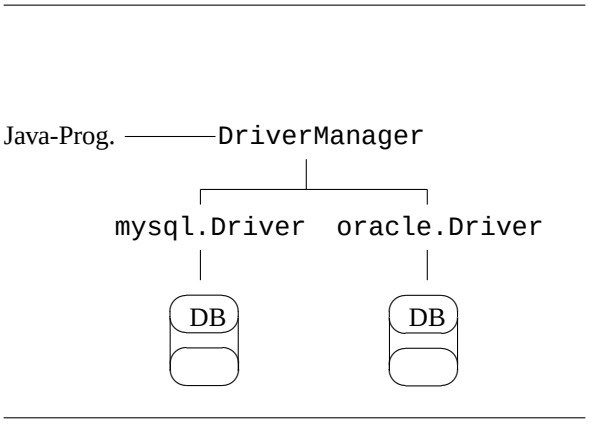

Abbildung 1: Verschiedene Datenbanken und ihre Treiber

Und in allgemeiner Form sieht er so aus:

```
dbURL="jdbc:subprotocol://host:port/dbname..."
```

Der `DriverManager` erkennt also am Wert von *subprotocol* mit welcher Datenbank er Kontakt aufnehmen soll und nimmt dazu den entsprechenden Treiber.

Die Methode `getConnection(dbURL)` liefert ein Objekt der Klasse `Connection` zurück. Dieses Objekt repräsentiert die Verbindung mit der Datenbank.

Nun braucht man nur noch einen Mechanismus, mit dem man die verschiedenen `Driver`-Objekte beim `DriverManager` anmeldet. Das geht aber ganz einfach, das erledigt nämlich der Konstruktor der Klasse `Driver`. Und wie man ja weiss, wird der Konstruktor immer dann automatisch ausgeführt, wenn ein Objekt der Klasse gebildet wird. Alles was man also tun muss, ist ein Objekt der Klasse `Driver` zu bilden.

Das erledigt die folgende Zeile:

```
Class.forName("org.gjt.mm.mysql.Driver")
                          .newInstance();
```

Durch die Methode `newInstance()` wird ein (namenloses) Objekt der Klasse gebildet, die im durch den String

```
"org.gjt.mm.mysql.Driver"
```

angegebenen Pfad liegt. Dieser Pfad zur Klasse `Driver` liegt dabei unterhalb des oben angegebenen Pfads

```
../lib/jre/ext/jdbc-Bibliothek
```

Natürlich hätte man das Driver-Objekt auch herkömmlich mit `new` bilden können. Der gezeigte Ansatz mit der Methode `forName(String)` ist aber universeller, da man den Treiber und die Datenbank wechseln kann, ohne dass man sein Programm neu compilieren muss.

3.2 `Connection` und `Statement`

Objekte der Klasse `Connection` repräsentieren die Verbindung mit der Datenbank.

Um über diese Verbindung nun SQL-Befehle (SQL-Statements) an die Datenbank senden zu können, gibt es die Methoden

- `executeUpdate()`
- `executeQuery()`
- `execute()`

Dies sind Methoden der Klasse `Statement`. Um sie aufrufen zu können, braucht man zunächst also ein Statement-Objekt, das mit der Zeile

```
Statement sqlStatement
     = dbVerbindung.createStatement();
```

erzeugt wird. `createStatement()` ist eine Methode eines Objekts der Klasse `Connection`.

3.3 `ResultSet`

Zum Schluss noch eine wichtige Klasse, die Klasse `ResultSet`. Objekte dieser Klasse nehmen die Daten einer SQL-Abfrage auf, ResultSet-Objekte sind folglich wie eine Tabelle (Zeilen und Spalten) aufgebaut.

Mit der Methode next() wird zur nächsten Zeile weitergeschaltet.

Mit get-Methoden für alle Java-Datentypen können anschliessend die Spaltenwerte ausgelesen werden. Dabei findet unter Umständen eine Typenwandlung statt: wird z.B. eine Spalte, die numerische Werte enthält mit `getString()` ausgelesen, wird der numerische Wert in einen String umgewandelt. Umgekehrt geht das natürlich nicht.

Alle diese get-Methoden erwarten *genau einen* Parameter. Dieser kann von Typ `String` oder `int` sein und repräsentiert den Spaltennamen oder die Spaltennummer in der Tabelle.

A Beispielprogramm

```
import java.sql.*;

class Connect {

    public static void main(String[] args) {

        String dbURL; //die jdbc:URL fuer diese Datenbank
        String user = "nobody"; //wer loggt in die datenbank ein
        String hostURL  = "localhost"; //auf welchem host laeuft der server
        String password  = "neu"; //passwort
        String dataBase = "cd_datab"; //welche datenbank soll benutzt werden

        dbURL =
            "jdbc:mysql://" + hostURL+ "/"+ dataBase+"?" + "user=" + user + "&" + "password="+password;

        //dbURL = "jdbc:mysql://localhost/cd_datab?user=nobody&password=neu";

        try {
            // Treiber laden; der treiber wird  in folgendem pfad gesucht:
            //absoluter pfad:   /usr/lib/j2re1.3/lib/ext/mm.mysql.jdbc-1.2c/org/gjt/mm/mysql

            Class.forName("org.gjt.mm.mysql.Driver").newInstance();

            Connection dbVerbindung = DriverManager.getConnection(dbURL);   // verbindungsobjekt holen

            Statement sqlStatement = dbVerbindung.createStatement();        //SQL-statement-objekt erzeugen

            sqlStatement.executeUpdate("Update Artist"+
                                       " set ArtistName="+
                                       "'Bernhard Potschka'"+
                                       " where ArtistID = 3");

            ResultSet ergebnisMenge;                                        //ResultSet-objekt deklarieren

            //SQL-statement absetzen; diese erzeugt ResultSet-objekt
            ergebnisMenge = sqlStatement.executeQuery
                ("SELECT ArtistID, ArtistName FROM Artist");

            System.out.println("\n\nErgebnis der Abfrage:\n");

            int i = 1;
            while(ergebnisMenge.next() ){
                int id = ergebnisMenge.getInt("ArtistID");
                String musiker = ergebnisMenge.getString("ArtistName");
                System.out.println(i+" ID :"+id+"   "+ musiker);
                i++;
            }

            System.out.println("\n\n");

            dbVerbindung.close();                                           //datenbank wieder schliessen

        }

        catch(Exception e ) {e.printStackTrace();}
    }
}
```

Abbildung 2: Ein einfaches Beispielprogramm zur JDBC-Schnittstelle