Einführung in JDBC

IFIS - Universität zu Lübeck

18.04.2007

Was ist JDBC

- Standard für relationale DB: SQL
- Vor JDBC: “Open Database Connectivity” (ODBC), uniforme Schnittstelle für Zugriff
- JDBC: Programmierschnittstelle (Java-API) für Zugriff auf relationale Datenbanken
- Ermöglicht Java-Programmen, SQL Befehle an relationale Datenbanken zu schicken
- Beliebige DB, lokal oder über das Netz

Was ist JDBC (cont.)

- Sammlung von Klassen und Interfaces
- Für jedes DBMS: JDBC Treiber Klasse
- Low-level: ruft SQL direkt auf
- Grundlage für higher-level APIs
- JDBC is a trademarked name and is not an acronym; ... often thought of as standing for "Java Database Connectivity”

Unterstützte SQL-Befehle

- SQL ANSI SQL-2 Entry Level
- Nicht von allen JDBC-Treibern erfüllt
- Wichtigste Befehle:
  - SELECT
  - CREATE TABLE, ALTER TABLE, DROP TABLE
  - INSERT, UPDATE, DELETE
  - COMMIT, ROLLBACK

Ablaufschema

- Laden eines JDBC-Treibers
- Aufbau einer Verbindung zur Datenbank
- Aufbau eines SQL-Statements
- Senden des Statements an die Datenbank
- [Analyse der Struktur der Ergebnismenge]
- Laden der Rückgabedaten
- Weiterverarbeiten der Daten
- Beenden der Datenbankverbindung

Ein umfassendes Beispiel

- Java Programm betrachten, das über JDBC auf eine Datenbank zugreift und sie mit SQL manipuliert
- Für das Beispiel: hsqldb verwenden. Soll auch mit mySQL, DB2 usw. funktionieren
- Tabellen und Daten vom Programm erzeugt
- Gelesene Daten in System.out ausgeben

Bibliotheken verwenden

- Beim Programmaufruf: Treiber-Klassen (z.B: hsqldb.jar oder db2java.zip) in CLASSPATH aufnehmen. Kann in Startup-Skript (.bashrc, autoexec.bat) geschehen
- Im Source-Code: packages (oder einzelne Klassen) importieren (import java.sql.*;)

Schritt 1: Treiber laden

- Nur einmal am Anfang des Programms

```
Class.forName(nameDerTreiberKlasse);
```

- Beispiel: mit HSQLDB:
  Class.forName(“org.hsqldb.jdbcDriver”);
- DB2: COM.ibm.db2.jdbc.app.DB2Driver
- mySQL: org.gjt.mm.mysql.Driver

Schritt 2: Verbindung aufbauen

```
Connection con =
   DriverManager.getConnection(url, user, passwd);
```

- URL treiberabhängig. Besteht aus Protokol, Host (oder IP Adresse), Port, Parameter...
  - jdbc:hsqldb:hsql://localhost[:9001]
  - jdbc:db2:prakt, jdbc:db2://linse[:6789]/prakt
  - jdbc:mysql://myPC/myDB
- user, password können u.U. null sein

Schritt 3: SQL-Statement aufbauen

- SQL Befehle an DB: java.sql.Statement (Container für Befehlausführung)
- 3 Typen von Statements: Statement (ohne Parameter), PreparedStatement (mit/ohne IN-Parameter), CallableStatement
- In Bezug auf aktuelle Verbindung erzeugen

```
Statement stmt = con.createStatement();
```

Schritt 4: Senden des SQL-Statements an Datenbank

- SQL-Befehle als String an Statement übergeben. Je nach Art der Anfrage wird stmt.executeQuery(String arg) oder stmt.executeUpdate(String arg) ausgeführt
- [Es gibt noch execute, nur Sonderfälle]
- Bsp: String query = “select * from Artikel”; stmt.executeQuery(query);

Schritt 5: Ergebnis laden

- Ergebnis der Anfrage: java.sql.ResultSet

```
ResultSet res = stmt.executeQuery(query);
```

- In fetch-Schleife die Ergebnisse holen:

```
while (res.next()) {
  String n = res.getString("Name"); ...
}
```

- Alternative: über Index lesen: getInt(2);

Schritt 6: Weiterverarbeiten

- Ergebnisdaten vom Programm weiter verarbeitet: Berechnungen, Visualisierung...
- Bsp: {String n = res.getString("Name"); System.out.println(n);} im einfachsten Fall
- In vielen Fällen: Daten zwischenspeichern (z.B. in einer Collection). In Schleife: Zeile lesen, Objekt erzeugen, Objekt in Container stecken

Schritt 7: Verbindung beenden

- Resourcen freigeben. Verbindungen explizit schließen
- Empfehlenswert: Statements schließen mit stmt.close()
- Vebindung beenden mit con.close();

Exceptions behandeln

- Viele Methodenaufrufe: Exception
- Mit try/catch-Konstrukt abfangen & behandeln

```
try {
  Connection con =
    DriverManager.getConnection(url, user,
    passwd); } catch (SQLException e) {
 System.err.println(„Cannot connect: “+e);}
```

Beispiel: Tabelle erzeugen

```
package ifis.dbp03.database;
import java.sql.*;
public class CreateTable {
public static void main(String[] args) {
  try {
    Class.forName("org.hsqldb.jdbcDriver");
    String url = "jdbc:hsqldb:hsql://localhost";
    Connection con = DriverManager.getConnection(url, "sa", "");
    Statement stmt = con.createStatement();
    String s = "Create TABLE artikel(Nummer int, bez varchar(40))";
    stmt.executeUpdate(s);
    stmt.close();
    con.close();
  } catch (Exception e) {e.printStackTrace();}
}
}
```

Beispiel: Daten einfügen & lesen

```
String s = "insert into artikel values(1, 'Milch')";
 int count = stmt.executeUpdate(s);
 System.out.println("Number of rows changed: "+count);
 s = "insert into artikel values(2, 'Zucker')";
 count = stmt.executeUpdate(s);
 s = "select * from artikel";
 ResultSet rs = stmt.executeQuery(s);
 while (rs.next()) {
   int num = rs.getInt(1);
   String bez = rs.getString(2);
   System.out.println("Nummer: "+num);
   System.out.println("Bezechnung: "+bez);
 }
```

PreparedStatement

- Abfrageschema festlegen, bei Bedarf Parameter einsetzen
- Performanter als Statement (vorkompiliert)

```
String s = "insert into artikel values(?, ?)";
PreparedStatement ps = con.prepareStatement(s);
ps.setInt(1, 30);
ps.setString(2, "Reis");
ps.executeUpdate();
```

Zuordnung Java-Klassen zu SQL-Datentypen

- CHAR, VARCHAR, LONGVARCHAR: String
- NUMERIC, DECIMAL: java.math.BigDecimal
- BIT: boolean; TINYINT: byte; SMALLINT, short; INTEGER: int, BIGINT: long
- REAL: float, DOUBLE: double
- DATE: java.sql.Date
- TIME: java.sql.Time
- TIMESTAMP: java.sql.Timestamp

Zugriff auf JDBC-Datentypen

- ResultSet: getXXX (getBigDecimal, getShort, getInt, getString...; XXX: Name des Java-Typs, nicht des JDBC-Typs)
- 1 Parameter: Spaltenname oder -nummer (erste Spalte: 1,...)
- PreparedStatement: setXXX (setByte, setFloat, setTimestamp...)
- 2 Parameter: Index, Wert

Analyse der Ergebnismenge

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

ResultSetMetaData md = rs.getMetaData();

- Anzahl, Namen, Typ... der Spalten
  - getColumnCount()
  - getColumnName(int col)
  - getColumnType(int col)
  - isNullable(int col)

Metadaten über Datenbank

- Mit Hilfe des aktuellen Connection-Objekts erzeugt: DatabaseMetaData meta = con.getMetaData()
- Liste aller Tabellen [getTables(...)], aller Spalten [getColumns(...)] in einem Katalog; Liste der SQL-Schlüsselworte; Name, Version des JDBC-Treibers; Unterstützung bestimmter Features (z.B. subselect)...

Transaktionen

- Statements zusammen ausführen
- Alle ausgeführt oder zurückgesetzt
- JDBC Voreinstellung: autocommit Modus (COMMIT nach jedem SQL-Befehl)
- Transaktion Unterstützung: Methoden in Connection: setAutoCommit(boolean), commit(), rollback()

Beispiel: Transaktion

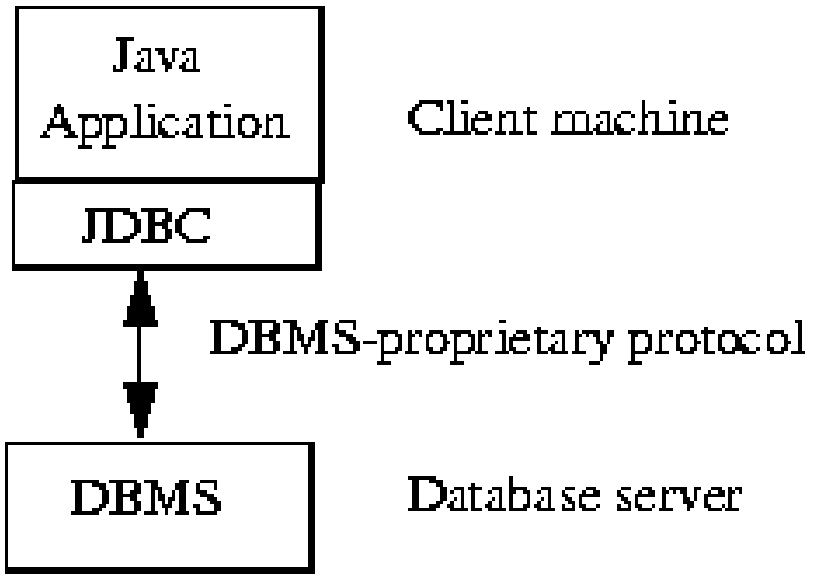
```
String s1 = "insert into address values(?,?,?);
String s2 = "insert into person values(?,?,?,?);
PreparedStatement addr = con.prepareStatement(s1);
PreparedStatement pers = con.prepareStatement(s2);
try {
  con.setAutoCommit(false);
  addr.setString(...); ...;
  addr.executeUpdate();
  pers.setInt(...); ...;
  pers.executeUpdate();
  con.commit();
} catch (SQLException e) {
  con.rollback();
}
```

Anwendungsarchitekturen

- 2-Schichten- (Client-DBMS) vs. 3-Schichten-Architektur (Client-Application Server-DBMS)
- 2-Schichten: Client sendet SQL an DBMS über JDBC, erhält ResultSet zurück, verarbeitet Daten und zeigt in (G)UI an
- 3-Schichten: DB-Zugriff durch AppServer, Client kommuniziert mit DBMS indirekt

2-Schichten Architektur

- Client: Java Applet oder Application
- UI & business logic beim Client
- Client schickt SQL an DB, manipuliert Daten direkt



3-Schichten-Modell

- Client - AppServer über higher-level API (Server setzt Befehle in SQL um)
- Zugriffskontrolle
- Performance