Historija JDBC

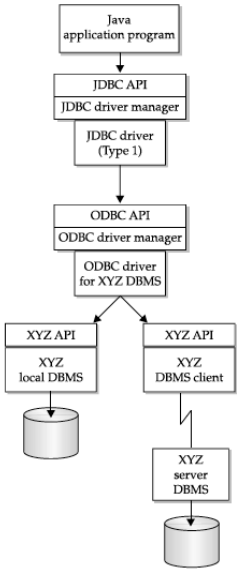
JDBC 1.0: osnovne funkcionalnosti za pristup podacima.

JDBC 2.0: batch operations, scrollable result set, updateable result set, connection pooling, distributed transactions, dana sources, rowset (disconnected result set), JNDI support.

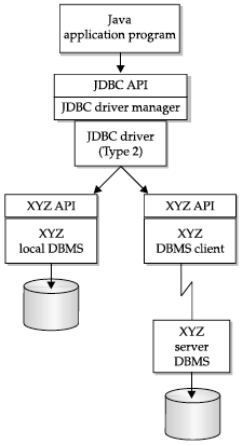
JDBC 3.0: object relational extensions, savepoints, cursor preservation, prepared statement metadata.

Implementacija JDBC

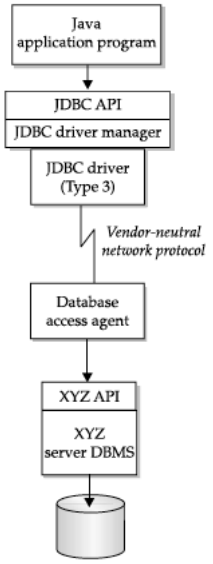
JDBC tip 1 drajvera:



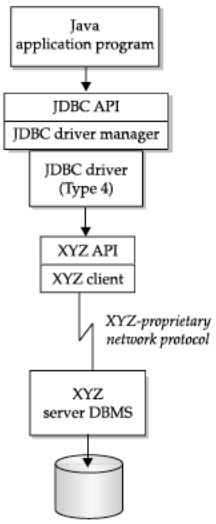
JDBC tip 2 drajvera:



JDBC tip 3 drajvera:



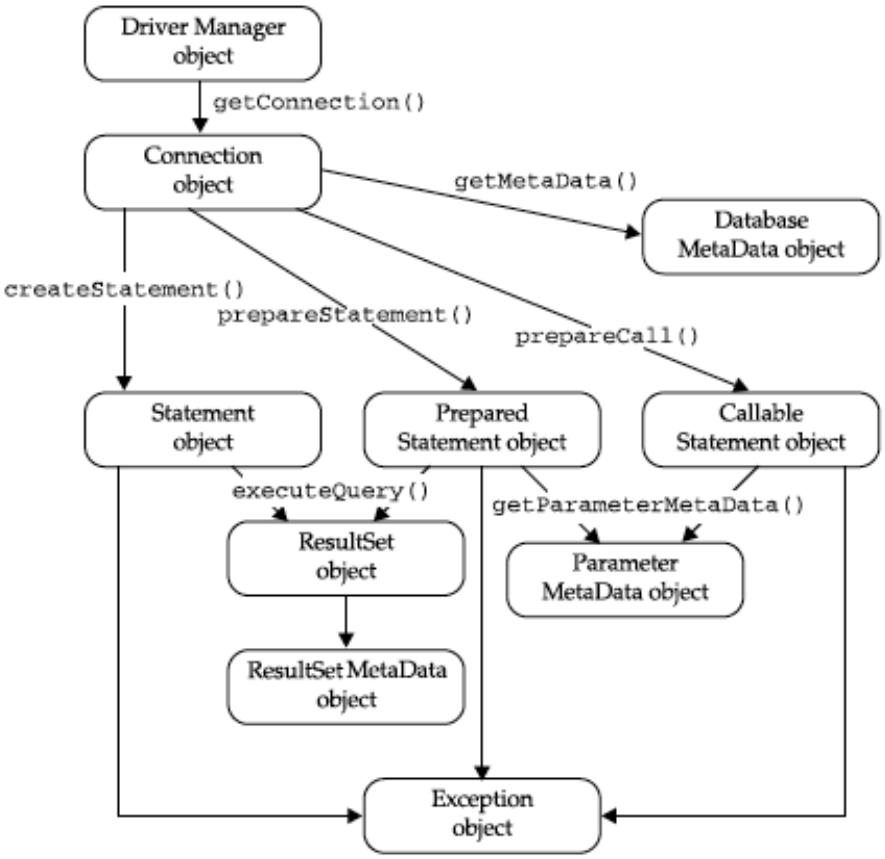
JDBC tip 4 drajvera:



DBMS API location	Database access: Via vendor-neutral (ODBC) API	Database access: Direct to vendor-proprietary API
Client-side	Type 1 driver	Type 2 driver
Server-side	Type 3 driver	Type 4 driver

JDBC API

Kratak pregled klasa i metoda za pristup bazi podataka.



Sve metode koje će biti opisane nalaze se u paketu koji treba biti uveden sa:

```
import java.sql.*;
```

Najpre je potrebno uspostaviti vezu sa bazom podataka, što uključuje dva koraka: 1. učitavanje drajvera i 2. pravljenje veze (konekcije). Učitavanje drajvera se obavlja sa:

```
Class.forName("jdbc.DriverXYZ");
```

Pri tom nije potrebno praviti instancu drajvera i registrovati je sa Driver Manager-om jer će to obaviti poziv Class.forName automatski. Nakon učitavanja drajvera moguće je napraviti vezu (Connection objekat) sa:

```
String url = "jdbc:odbc:XYZ";
Connection con = DriverManager.getConnection(url, "myLogin", "myPassword");
```

Objekat Connection koji se dobija kao rezultat poziva predstavlju otvorenu vezu koja se može koristiti za kreiranje JDBC iskaza i prosleđivanje SQL iskaza do ciljne DBMS. Da bi se DBMS prosledio željeni SQL iskaz neophodno je kreirati Statement objekat, za šta će biti iskorišćen prethodno kreiran Connection objekat.

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

U ovom trenutku postoji iskaz, ali on ne sadrži potrebni SQL koji bi trebao da se prosledi do ciljnog DBMS. Potrebno je definisati SQL iskaz koji želimo da bude izvršen na DMBS, a potom ga pokrenuti. Ukoliko SQL iskaz ne vraća rezultat (CREATE, INSERT, UPDATE, DELETE) onda se za njegovo izvršavanje koristi executeUpdate metoda (ova metoda vraća celobrojnu vrednost koja predstavlja koliko je redova promenjeno izvršavanjem SQL iskaza. U slučaju DDL naredbi povratna vrednost je 0).

```
String SQLStm = "DELETE FROM ...";
stmt.executeUpdate(SQLStm);
```

Ukoliko SQL iskaz vraća rezultat, što je u slučaju SELECT naredbe, onda je potrebno koristiti objekat za prihvatanje rezultata ResultSet:

```
String SQLStmQuery = "SELECT * ...";
ResultSet rs = stmt.executeQuery(SQLStmQuery);
```

Dobijeni rezultat se obrađuje red po red, a za dohvatanje vrednosti pojedinih kolona u posmatranom redu se koriste odgovarajuće get metode (getString, getInt, itd.). Obično se obrada vrši u petlji:

```
while (rs.next()) {
    String s = rs.getString("column1");
    float n = rs.getFloat("column2");
    int m = rs.getInt("column3");
    Boolean b = rs.getBoolean("column4");
}
```

U slučaju kada se želi obrada različita od obrade red po red, potrebno je definisati parametre kursora pri kreiranju Statement objekta:

```
Statement stmt = con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
                                                    ResultSet.CONCUR_READ_ONLY);
```

To će omogućiti kretanje kroz rezultat ne samo next metodom, već i sa previous, absolute, relative metodama. Za dohvatanje se može koristiti naziv kolone kao "column1" (String), ili pak redni broj kolone (int):

```
String s = rs.getString(1);
```

Upotrebom rezultata moguće je vršiti promene u posmatranoj tabeli (ukoliko se radi o upitu koji odgovara pravilima ažurabilnog pogleda), ali je pre toga neophodno definisati tip rezultata (tip kursora). Za potrebe promena se koriste odgovarajuće update metode (updateString, updateFloat, updateInt, itd.)

```
Statement stmt = con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
                                  ResultSet.CONCUR_UPDATABLE);
ResultSet uprs = stmt.executeQuery(SQlString);
```

```
uprs.updateFloat("Column1", 102.45f);
uprs.updateRow();
```

U slučaju kada se u rezultat želi dodati novi red, onda se to radi sa

```
uprs.moveToInsertRow();
...
uprs.updateFloat("Column1", 102.45f);
uprs.insertRow();
```

Ukoliko se želi obrisati određeni red onda se to može uraditi sa

```
uprs.deleteRow();
```

U nekim slučajevima je potrebno izvršavati neki SQL iskaz više puta. U tim slučajevima je korisno napraviti pripremljeni SQL iskaz, tj. PreparedStatement objekat. Pripremljen iskaz se može koristiti bez parametara, mada je u najvećem broju slučajeva njegova upotreba upravo takva da se parametri koriste.

```
PreparedStatement updTab = con.prepareStatement(
    "UPDATE Tabela1 SET Atribute1 = ? WHERE Atribute2 LIKE ?");
updTab.setInt(1, 43);
updTab.setString(2, "TestString");
updTab.executeUpdate():
```

Kao što se iz prethodnog dela programa može videti, parametri su u SQL iskazu obeleženi znakom pitanja. Parametri moraju pre samog izvršavanja iskaza biti prosleđeni odgovarajućim set metodama (setInt, setString, itd.)

Function	Description
Scrollable cursor motion	
`previous()`	Moves cursor to previous row of query results
`beforeFirst()`	Moves cursor before the start of the results
`first()`	Moves cursor to first row of query results
`last()`	Moves cursor to last row of query results
`afterLast()`	Moves cursor past end of the results
`absolute()`	Moves cursor to absolute row number indicated
`relative()`	Moves cursor to relative row number indicated
Cursor position sensing	
`isFirst()`	Determines whether the current row is the first row of the result set
`isLast()`	Determines whether the current row is the last row of the result set
`isBeforeFirst()`	Determines whether the cursor is positioned before the beginning of the result set
`isAfterLast()`	Determines whether the cursor is positioned past the end of the result set
`moveToInsertRow()`	Moves cursor to "empty" row for inserting new data
`moveToCurrentRow()`	Moves cursor back to the current row before an insertion
Update a column of current row (via cursor)	
`updateInt()`	Updates an integer column value
`updateShort()`	Updates a short integer column value
`updateLong()`	Updates a long integer column value
`updateFloat()`	Updates a floating point column value
`updateDouble()`	Updates a double-precision floating point column value
`updateString()`	Updates a string column value
`updateBoolean()`	Updates a true/false column value
`updateDate()`	Updates a date column value
`updateTime()`	Updates a time column value
`updateTimestamp()`	Updates a timestamp column value
`updateByte()`	Updates a byte column value
`updateBytes()`	Updates a fixed-length or variable-length column value
`updateBigDecimal()`	Updates a `DECIMAL` or `NUMERIC` column value
`updateNull()`	Updates a column to a `NULL` value
`updateObject()`	Updates an arbitrary column value

Pozivi uskladištenih procedura obavljaju se upotrebom CallableStatement objekta, koji poput PreparedStatement objekta može da prihvata ulazne parametre, ali takođe i izlazne odnosno ulazno-izlazne paramtere takođe.

```
CallableStatement cs = con.prepareCall("{call Procedura1}");
ResultSet rs = cs. executeUpdate ();
```

Primeri poziva bi mogli biti: {call procedura2(?, ?, ?)} ili prilikom poziva funkcije {? = call funkcija3(?)}. Numeracija pozicija za smeštanje vrednosti kreće uvek od 1. Pozicije za Izlazne parametre u slučaju procedura, odnosno rezultat u slučaju funkcija treba registrovati pozivom metode registerOutParameter.

Nakon kreiranja konekcije, odnodno Connection objekta, ona se nalazi u auto-commit modu rada, što znači da se svaki SQL iskaz posmatra kao zasebna transakcija. Ukoliko se želi postići da veći broj SQL iskaza predstavljaju jednu transakciju, onda je potrebno eksplicitno isključiti auto-commit režim rada. Međutim u tom slučaju je neophodno eksplicitno izvršiti commit u cilju završavanja transakcije.

```
con.setAutoCommit(false);
...
con.commit();
con.setAutoCommit(true);
```

Transakcijom se smatraju svi SQL iskazi u periodu između dva poziva metode commit. U slučajevima kada je to potrebno moguće je anulirati efekte tekuće transakcije pozivom metode rollback. Nivo izolacije transakcija se postavlja sa:

```
con.setTransactionIsolation(TRANSACTION_READ_COMMITTED);
```

Mogući nivoi izolacija su (ali dostupnost svakog od njih zavisi od konkrekte implementacije):

TRANSACTION_READ_UNCOMMITTED – Problemi: dirty reads, nonrepeatable reads, phantom reads

TRANSACTION_READ_COMMITTED – Problemi: nonrepeatable reads, phantom reads

TRANSACTION_REPEATABLE_READ – Problemi: phantom reads

TRANSACTION_SERIALIZABLE

Pri radu sa JDBC, pozivi metoda u slučaju greške mogu baciti izuzetke tipa SQLException, a detalji mogu biti pročitani getMessage (dohvata poruku koja opisuje izuzetak), getSQLState (dohvata vrednost za SQLSTATE, to je string od 5 karaktera), getErrorCode (dohvata DBMS specifičan kod greške), getNextException (prelazi na naredni izuzetak u seriji) metodama.

Obrada meta podataka je moguća i to na nivou baze (DatabaseMetaData), niovu rezultata (ResultSetMetaData) i na nivou parametara (ParameterMetaData) koji se koriste u pripremljenim iskazima (Prepared statement ili CallableStatement)

DatabaseMetaData

Function	Description
`getTables()`	Returns result set of table information of tables in database
`getColumns()`	Returns result set of column names and type info, given table name
`getPrimaryKeys()`	Returns result set of primary key info, given table name
`getProcedures()`	Returns result set of stored procedure info
`getProcedureColumns()`	Returns result set of info about parameters for a specific stored procedure

ResultSetMetaData

Function	Description
`getColumnCount()`	Returns number of query results columns
`getColumnName()`	Retrieves name of specified results column
`getColumnType()`	Retrieves data type of specified results column

ParameterMetaData

Function	Description
`getParameterClassName()`	Returns name of the class (data type) for specified parameter
`getParameterCount()`	Returns number of parameters in the statement
`getParameterMode()`	Returns mode (`IN`, `OUT`, `INOUT`) of parameter
`getParameterType()`	Returns SQL data type of specified parameter
`getParameterTypeName()`	Returns DBMS data type of specified parameter
`getPrecision()`	Returns precision of specified parameter
`getScale()`	Returns scale of specified parameter
`isNullable()`	Determines whether the specified parameter is nullable
`isSigned()`	Determines whether the specified parameter is a signed number