Proyecto TIC: Desarrollo e implantación

Persistence in Enterprise Applications Java EE

Xavi Piñol
xavi.pinyol@gmail.com

Index

Index

Persistence. Introduction.

Persistence, in computer science, is an adjective describing data that outlives the process that created it. Java persistence could be defined as storing anything to any level of persistence using the Java programming language.

There are many ways to make data persist in Java, including (to name a few): JDBC, serialization, file IO, JCA, object databases, and XML databases. However, the majority of data is persisted in databases, specifically relational databases.

Persistence. Introduction.

Most things that you do on a computer or web site that involve storing data, involve accessing a relational database. Relational databases are the standard mode of persistent storage for most industries, from banking to manufacturing.

There are many things that can be stored in databases with Java. Java data includes strings, numbers, date and byte arrays, images, XML, and Java objects. Many Java applications use Java objects to model their application data.

Persistence. Introduction.

Because Java is an Object Oriented language, storing Java objects is a natural and common approach to persisting data from Java.

This subject is focused on storing Java objects to relational databases.

Index

JDBC. Introduction.

JDBC is a Java database connectivity technology (Java Standard Edition platform) from Oracle Corporation. This technology is an API for the Java programming language that defines how a client may access a database. It provides methods for querying and updating data in a database. JDBC is oriented towards relational databases.

JDBC. Introduction.

JDBC is similar to Open DataBase Connectivity (ODBC), but is designed specifically for Java programs, whereas ODBC is language-independent.

To connect with individual databases requires drivers for each database. The JDBC driver gives out the connection to the database and implements the protocol for transferring the query and result between client and database.

JDBC. Introduction.

- A JDBC driver is a software component enabling a Java application to interact with a database. It's a .jar file.
- The JDBC driver is normally given by the Database vendor. Third party drivers are also available.
- Programmers will use java.sql and javax.sql packages no matter which driver is going to be used.

Index

JDBC. JDBC Drivers types.

There are commercial and free drivers available for most relational database servers. These drivers fall into one of the following types:

- Type 1 JDBC-ODBC bridge driver
- Type 2 Native-API driver
- Type 3 Network Protocol driver
- Type 4 Thin driver

JDBC. JDBC Drivers types.

Type 1. JDBC-ODBC bridge driver

The JDBC-ODBC bridge driver uses ODBC driver to connect to the database. The JDBC-ODBC bridge driver converts JDBC method calls into the ODBC function calls. This is now discouraged because of thin driver.

JDBC. JDBC Drivers types.

Advantages:

Easy to use.

Can be easily connected to any database.

Disadvantages:

Performance degraded because JDBC method call is converted into the ODBC function calls.

The ODBC driver needs to be installed on the client machine.

JDBC. JDBC Drivers types.

Type 2. Native-API driver

The Native API driver uses the client-side libraries of the database. The driver converts JDBC method calls into native calls of the database API. It is not written entirely in java.

JDBC. JDBC Drivers types.

Advantage:

performance upgraded than JDBC-ODBC bridge driver.

Disadvantages:

The Native driver needs to be installed on the each client machine.

The Vendor client library needs to be installed on client machine..

JDBC. JDBC Drivers types.

Type 3. Network Protocol driver

The Network Protocol driver uses middleware (application server) that converts JDBC calls directly or indirectly into the vendor-specific database protocol. It is fully written in java.

JDBC. JDBC Drivers types.

Advantage:

No client side library is required because of application server that can perform many tasks like auditing, load balancing, logging etc.

Disadvantages:

Network support is required on client machine.

Requires database-specific coding to be done in the middle tier.

Maintenance of Network Protocol driver becomes costly because it requires database-specific coding to be done in the middle tier.

JDBC. JDBC Drivers types.

Type 4. Thin driver

The thin driver converts JDBC calls directly into the vendor-specific database protocol. That is why it is known as thin driver. It is fully written in Java language.

JDBC. JDBC Drivers types.

Advantage:

Better performance than all other drivers.

No software is required at client side or server side.

Disadvantage:

Drivers depends on the Database.

Index

JDBC. Executing SQL statments.

In case that there is a change of the relational database (e.g. changing from PostgreSQL to MySQL), in theory, the code will stay unaffected. Unfortunately the relational databases have defined different SQL dialects.

- Data types are different
- Unique identifiers are different
- and so on.

JDBC. Executing SQL statments.

There are 5 steps to connect any java application with the database in java using JDBC. They are as follows:

- Register the driver class
- Creating connection
- Creating statement
- Executing queries
- Closing connection

JDBC. Executing SQL statments.

Register the driver class

The forName() method of Class class is used to register the driver class. This method is used to dynamically load the driver class.

Syntax of forName() method

```
public static void forName(String
className)throws ClassNotFoundException
```

JDBC. Executing SQL statments.

Examples to register differents drivers class

```
Class.forName("oracle.jdbc.driver.OracleDriver");
    Class.forName("com.mysql.jdbc.Driver");
    Class.forName("org.postgresql.Driver");
```

JDBC. Executing SQL statments.

Create the connection object

The getConnection() method of DriverManager class is used to establish connection with the database.

Syntax of getConnection() method

public static Connection getConnection(String url)throws SQLException

public static Connection getConnection(String url,String name,String password) throws SQLException

JDBC. Executing SQL statments.

Example to establish connection with the Oracle database

```
Connection con = DriverManager.getConnection
("jdbc:oracle:thin:@localhost:1521:xe", "system",
"password");
```

JDBC. Executing SQL statments.

Create the Statement object

The createStatement() method of Connection interface is used to create statement. The object of statement is responsible to execute queries with the database.

Syntax of createStatement() method

public Statement createStatement()throws SQLException

JDBC. Executing SQL statments.

Example to create the statement object

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

JDBC. Executing SQL statments.

Execute the query

The executeQuery() method of Statement interface is used to execute queries to the database. This method returns the object of ResultSet that can be used to get all the records of a table.

Initially the cursor is just before the first record. If there isn't more records the .next() method returns “false”. The driver will load in memory blocks of records instead of loading all the records of the query.

JDBC. Executing SQL statments.

The executeUpdate() method of Statement interface is used to execute updates to the database.

Syntax of executeQuery() method

public ResultSet executeQuery(String sql)throws SQLException

JDBC. Executing SQL statments.

Example to execute query

```
ResultSet rs=stmt.executeQuery("select * from emp");
while(rs.next())
{
    System.out.println(rs.getInt(1)+" "+rs.getString(2));
}
```

JDBC. Executing SQL statments.

Prepared Statements.

Prepared statement or parameterized statement is a feature used to execute the same or similar database statements repeatedly with high efficiency. Typically used with SQL statements such as queries or updates, the prepared statement takes the form of a template into which certain constant values are substituted during each execution.

JDBC. Executing SQL statments.

Syntax of Prepared Statements.

public PreparedStatement prepareStatement(String query)throws SQLException{}

Example of Prepared Statements.

java.sql.PreparedStatement stmt = connection.prepareStatement("SELECT * FROM users WHERE USERNAME = ? AND ROOM = ?");

stmt.setString(1, username);

stmt.setInt(2, roomNumber);

stmt.executeQuery();

JDBC. Executing SQL statments.

Close the connection object

By closing connection object statement and ResultSet will be closed automatically. The close() method of Connection interface is used to close the connection.

Syntax of close() method

```
public void close()throws SQLException
```

JDBC. Executing SQL statments.

Example to close connection Java EE 6.0

```
try {
 con = ConnectionManager.getConnection();
 << Queries or update >>
} catch (Exception e) { e.printStackTrace(System.err);}
 finally {
  Try { if (con != null)  con.close(); }
 catch (Exception e) { e.printStackTrace(System.err);}
}
```

JDBC. Executing SQL statments.

Example to close connection Java EE 7.0

```
try (Connection con  =
ConnectionManager.getConnection())
 {
    << Queries or update >>
} catch (Exception e)
{e.printStackTrace(System.err); }
```

JDBC. Executing SQL statments.

SQL Exception

public class SQLException

extends Exception

implements Iterable<Throwable>

An exception that provides information on a database access error or other errors.

JDBC. Executing SQL statments.

Each SQLException provides several kinds of information:

- a string describing the error.
- a "SQLstate" string.
- an integer error code that is specific to each vendor.
- a chain to a next Exception. This can be used to provide additional error information.
- the causal relationship, if any for this SQLException.

Index

SQL types and Java.

Java types	SQL types
boolean	BIT
byte	TINYINT
short	SMALLINT
int	INTEGER
long	BIGINT
float	REAL
double	DOUBLE
java.math.BigDecimal	NUMERIC
String	VARCHAR
byte[]	VARBINARY
java.sql.Date	DATE
java.sql.Time	TIME
java.sql.Timestamp	TIMESTAMP

Index

Data sources.

Datasource is a name given to the connection set up to a Database from an applications server. A DataSource object has properties that identify and describe the data source it represents (such as username, password, host, database name, port number, ...). Also, a DataSource object works with a Java Naming and Directory Interface (JNDI) naming service and can be created, deployed, and managed separately from the applications that use it.

Data sources.

An application does not need to hardcode driver information, as it does with the DriverManager. A programmer can choose a logical name for the data source and register the logical name with a JNDI naming service. The application uses the logical name, and the JNDI naming service will supply the DataSource object associated with the logical name. The DataSource object can then be used to create a connection to the data source it represents.

Data sources

Using a DataSource object is the preferred alternative to using the DriverManager for establishing a connection to a data source. They are similar to the extent that the DriverManager class and DataSource interface both have methods for creating a connection, methods for getting and setting a timeout limit for making a connection, and methods for getting and setting a stream for logging.

Data sources.

Example to Data source.

Note: JNDI “ java:jboss/PostgresXA”

```
DataSource ds = (DataSource)
cxt.lookup( "java:jboss/PostgresXA");
Connection connection = ds.getConnection();
```

Index

Connection pool

Problems:

- Applications server is receiving a lot of HTTP requests per minute.
- A request to create a connection to the data base it takes time.
- If the data base doesn't accept more connections it returns an error.

Solution: connection pool

Connection pool

A connection pool is a cache of database connections maintained by the Applications server so that the connections can be reused when future requests to the database are required. Connection pools are used to enhance the performance of executing commands on a database. Opening and maintaining a database connection for each user, especially requests made to a dynamic database-driven website application, is costly and wastes resources.

Connection pool

In connection pooling, after a connection is created, it is placed in the pool and it is used again so that a new connection does not have to be established. If all the connections are being used, a new connection is made and is added to the pool. Connection pooling also cuts down on the amount of time a user must wait to establish a connection to the database.

Connection pool

Applications servers Administrators can configure connection pools with restrictions on the numbers of minimum connections, maximum connections and idle connections to optimize the performance of pooling in specific problem contexts and in specific environments.

Connection pool

Data base server goes down.

GetConnection() method can check if the connection is working properly, depends on the driver vendor.

Application server must be reinitialized.