Hibernate Vs JDBC

Author: Dipti Phutela, Mindfire Solutions
www.mindfiresolutions.com

Table of Contents

1. Introduction to JDBC

JDBC stands for **Java Database Connectivity** allows developers to connect, query and update a database using the Structured Query Language. JDBC API standard provides Java developers to interact with different RDBMS and access table data through Java application without learning RDBMS details and using Database Specific JDBC Drivers.

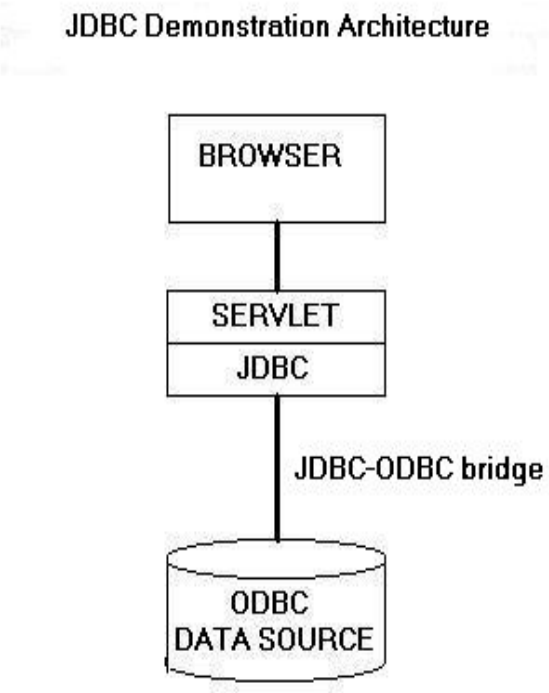
2. JDBC Architecture

JDBC makes the interaction with RDBMS simple and intuitive. When a Java application needs to access database:

- open connection to database,
- use JDBC driver to send SQL queries to database,
- process the results that are returned, and
- close the connection.

JDBC uses **two architectures** to communicate with database:

1) The driver connects to database and executes SQL statements. Results are sent back from driver to driver manager and finally to the application.
2) The JDBC driver communicates with ODBC driver. ODBC driver executes SQL query and then results are sent back to JDBC driver to driver manager and then to application.



3. Interaction with RDBMS

General steps:

1) Load the RDBMS specific JDBC driver because this driver actually communicates with the database.
2) Open the connection to database which is then used to send SQL statements and get results back.
3) Create JDBC Statement object. This object contains SQL query.
4) Execute statement which returns resultset(s). ResultSet contains the tuples of database table as a result of SQL query.
5) Process the result set.
6) Close the connection.

Example: Retrieve list of employees from Employee table using JDBC.

```
String url =  "jdbc:odbc:" + dbName;
List<EmployeeBean> employeeList = new ArrayList<EmployeeBean>();

/* load the jdbc-odbc driver */
class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

/* Open a connection to database */
Connection con = DriverManager.getConnection(url);

/* create Statement object */
Statement stmt = con.createStatement();

/* execute statement */
ResultSet rs = stmt.executeQuery("SELECT * FROM Sells");
while ( rs.next() )
{
      EmployeeBean eb = new Employeebean();
      eb.setName(rs.getString("name"));
      eb.setSalary(rs.getFloat("salary"));
      employeeList.add(eb);
}
```

4. Introduction to Hibernate

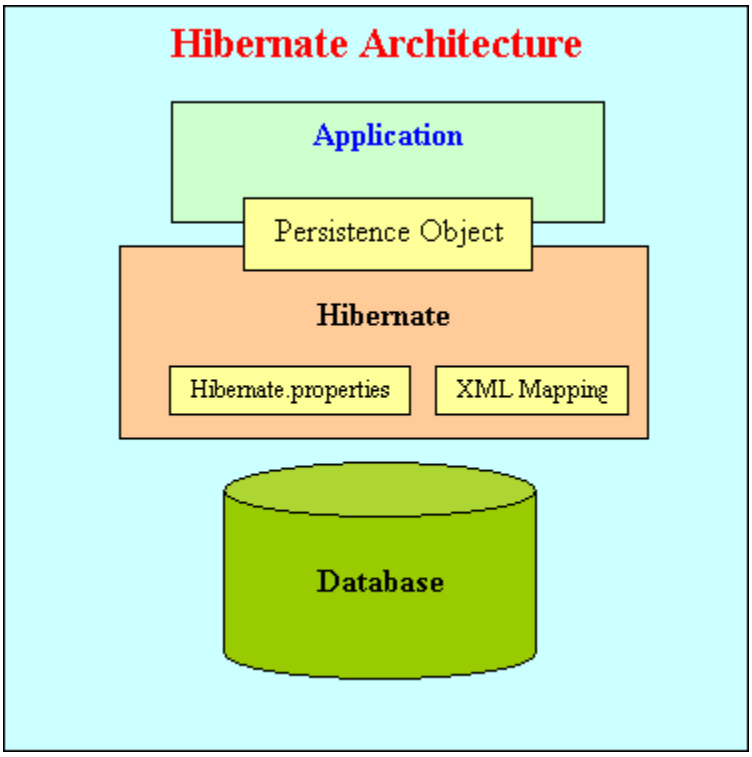
Hibernate is an **Object-Relational Mapping (ORM) solution** for JAVA. It is a powerful, high performance object/relational persistence and query service. It allows us to develop persistent classes following object-oriented idiom – including association, inheritance and polymorphism.

5. Hibernate Architecture

Hibernate:

1) itself opens connection to database,
2) converts HQL (Hibernate Query Language) statements to database specific statement,
3) receives result set,
4) then performs mapping of these database specific data to Java objects which are directly used by Java application.

Hibernate uses the database specification from Hibernate Properties file. Automatic mapping is performed on the basis of the properties defined in hbm XML file defined for particular Java object.



6. Hibernate communication with RDBMS

General steps:

1. Load the Hibernate configuration file and create configuration object. It will automatically load all hbm mapping files.
2. Create session factory from configuration object
3. Get one session from this session factory.
4. Create HQL query.
5. Execute query to get list containing Java objects.

Example: Retrieve list of employees from Employee table using Hibernate.

```
/* Load the hibernate configuration file */
Configuration cfg = new Configuration();
cfg.configure(CONFIG_FILE_LOCATION);

/* Create the session factory */
SessionFactory sessionFactory = cfg.buildSessionFactory();

/* Retrieve the session */
Session session = sessionFactory.openSession();

/* create query */
Query query = session.createQuery("from  EmployeeBean");

/* execute query and get result in form of Java objects */
List<EmployeeBean> finalList = query.list();
```

EmployeeBean.hbm.xml File

```
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE hibernate-mapping PUBLIC
    "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
    "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">

<hibernate-mapping>
      <class name="com.mf.bean.EmployeeBean"
            table="t_employee">

            <id name="id" type="string" unsaved-value="null">
                  <column name="id" sql-type="varchar(32)" not-null="true"/>
                  <generator class="uuid"/>
            </id>

            <property name="name">
                  <column name="name" />
            </property>

            <property name="salary">
                  <column name="salary" />
            </property>
      </class>
</hibernate-mapping>
```

7. JDBC Vs Hibernate

7.1 Why is Hibernate better than JDBC

1) Relational Persistence for JAVA
Working with both Object-Oriented software and Relational Database is complicated task with JDBC because there is mismatch between how data is represented in objects versus relational database. So with JDBC, developer has to write code to map an object model's data representation to a relational data model and its corresponding database schema. Hibernate is flexible and powerful ORM solution to map Java classes to database tables. Hibernate itself takes care of this mapping using XML files so developer does not need to write code for this.

2) Transparent Persistence
The automatic mapping of Java objects with database tables and vice versa is called Transparent Persistence. Hibernate provides transparent persistence and developer does not need to write code explicitly to map database tables tuples to application objects during interaction with RDBMS. With JDBC this conversion is to be taken care of by the developer manually with lines of code.

3) Support for Query Language
JDBC supports only native Structured Query Language (SQL). Developer has to find out the efficient way to access database, i.e to select effective query from a number of queries to perform same task. Hibernate provides a powerful query language Hibernate Query Language (independent from type of database) that is expressed in a familiar SQL like syntax and includes full support for polymorphic queries. Hibernate also supports native SQL statements. It also selects an effective way to perform a database manipulation task for an application.

4) Database Dependent Code
Application using JDBC to handle persistent data (database tables) having database specific code in large amount. The code written to map table data to application objects and vice versa is actually to map table fields to object properties. As table changed or database changed then it's essential to change object structure as well as to change code written to map table-to-object/object-to-table. Hibernate provides this mapping itself. The actual mapping between tables and application objects is done in XML files. If there is change in Database or in any table then the only need to change XML file properties.

5) Maintenance Cost
With JDBC, it is developer's responsibility to handle JDBC result set and convert it to Java objects through code to use this persistent data in application. So with JDBC, mapping between Java objects and database tables is done manually. Hibernate reduces lines of code by maintaining object-table mapping itself and returns result to application in form of Java objects. It relieves programmer from manual handling of persistent data, hence reducing the development time and maintenance cost.

6) Optimize Performance
Caching is retention of data, usually in application to reduce disk access. Hibernate, with Transparent Persistence, cache is set to application work space. Relational tuples are moved to this cache as a result of query. It improves performance if client application reads same data many

times for same write. Automatic Transparent Persistence allows the developer to concentrate more on business logic rather than this application code. With JDBC, caching is maintained by hand-coding.

7) Automatic Versioning and Time Stamping
By database versioning one can be assured that the changes done by one person is not being roll backed by another one unintentionally. Hibernate enables developer to define version type field to application, due to this defined field Hibernate updates version field of database table every time relational tuple is updated in form of Java class object to that table. So if two users retrieve same tuple and then modify it and one user save this modified tuple to database, version is automatically updated for this tuple by Hibernate. When other user tries to save updated tuple to database then it does not allow to save it because this user does not has updated data. In JDBC there is no check that always every user has updated data. This check has to be added by the developer.

8) Open-Source, Zero-Cost Product License
Hibernate is an open source and free to use for both development and production deployments.

9) Enterprise-Class Reliability and Scalability
Hibernate scales well in any environment, no matter if use it in-house Intranet that serves hundreds of users or for mission-critical applications that serve hundreds of thousands. JDBC can not be scaled easily.

7.2 Disadvantages of Hibernate

1) Steep learning curve.

2) Use of Hibernate is an overhead for the applications which are :
- simple and use one database that never change
- need to put data to database tables, no further SQL queries
- there are no objects which are mapped to two different tables

Hibernate increases extra layers and complexity. So for these types of applications JDBC is the best choice.

3) Support for Hibernate on Internet is not sufficient.

4) Anybody wanting to maintain application using Hibernate will need to know Hibernate.

5) For complex data, mapping from Object-to-tables and vise versa reduces performance and increases time of conversion.

6) Hibernate does not allow some type of queries which are supported by JDBC. For example It does not allow to insert multiple objects (persistent data) to same table using single query. Developer has to write separate query to insert each object.