**HIBERNATE AND JPA**

**What is JPA?**

A JPA (Java Persistence API) is a specification of Java which is used to access, manage, and persist data between Java object and relational database.

It is considered as a standard approach for Object Relational Mapping.

JPA can be seen as a bridge between object-oriented domain models and relational database systems. Being a specification, JPA doesn't perform any operation by itself. Thus, it requires implementation.

So, ORM tools like Hibernate, TopLink, and iBatis implements JPA specifications for data persistence.

**What is Hibernate?**

A Hibernate is a Java framework which is used to store the Java objects in the relational database system. It is an open-source, lightweight, ORM (Object Relational Mapping) tool.

Hibernate is an implementation of JPA. So, it follows the common standards provided by the JPA.

**Need of JPA**

As we have seen so far, JPA is a specification. It provides common prototype and functionality to ORM tools. By implementing the same specification, all ORM tools (like Hibernate, TopLink, iBatis) follows the common standards. In the future, if we want to switch our application from one ORM tool to another, we can do it easily.

**Hibernate**

This hibernate tutorial provides in-depth concepts of Hibernate Framework with simplified examples. It was started in 2001 by Gavin King as an alternative to EJB2 style entity bean.

**Hibernate Framework**

Hibernate is a Java framework that simplifies the development of Java application to interact with the database. It is an open source, lightweight, ORM (Object Relational Mapping) tool. Hibernate implements the specifications of JPA (Java Persistence API) for data persistence.

**ORM Tool**

An ORM tool simplifies the data creation, data manipulation and data access. It is a programming technique that maps the object to the data stored in the database.

**What is JPA?**

Java Persistence API (JPA) is a Java specification that provides certain functionality and standard to ORM tools. The javax.persistence package contains the JPA classes and interfaces.

**Advantages of Hibernate Framework**

Following are the advantages of hibernate framework:

1) Open Source and Lightweight

Hibernate framework is open source under the LGPL license and lightweight.

2) Fast Performance

The performance of hibernate framework is fast because cache is internally used in hibernate framework. There are two types of cache in hibernate framework first level cache and second level cache. First level cache is enabled by default.

3) Database Independent Query

HQL (Hibernate Query Language) is the object-oriented version of SQL. It generates the database independent queries. So you don't need to write database specific queries. Before Hibernate, if database is changed for the project, we need to change the SQL query as well that leads to the maintenance problem.

4) Automatic Table Creation

Hibernate framework provides the facility to create the tables of the database automatically. So, there is no need to create tables in the database manually.

5) Simplifies Complex Join

Fetching data from multiple tables is easy in hibernate framework.

6) Provides Query Statistics and Database Status

Hibernate supports Query cache and provide statistics about query and database status.

* Hibernate Architecture
* Elements of Hibernate Architecture
* SessionFactory
* Session
* Transaction
* ConnectionProvider
* TransactionFactory
* The Hibernate architecture includes many objects such as persistent object, session factory, transaction factory, connection factory, session, transaction etc.

The Hibernate architecture is categorized in four layers.

* Java application layer
* Hibernate framework layer
* Backhand api layer
* Database layer
* Elements of Hibernate Architecture

For creating the first hibernate application, we must know the elements of Hibernate architecture. They are as follows:

**SessionFactory**

The SessionFactory is a factory of session and client of ConnectionProvider. It holds second level cache (optional) of data. The org.hibernate.SessionFactory interface provides factory method to get the object of Session.

**Session**

The session object provides an interface between the application and data stored in the database. It is a short-lived object and wraps the JDBC connection. It is factory of Transaction, Query and Criteria. It holds a first-level cache (mandatory) of data. The org.hibernate.Session interface provides methods to insert, update and delete the object. It also provides factory methods for Transaction, Query and Criteria.

**Transaction**

The transaction object specifies the atomic unit of work. It is optional. The org.hibernate.Transaction interface provides methods for transaction management.

**ConnectionProvider**

It is a factory of JDBC connections. It abstracts the application from DriverManager or DataSource. It is optional.

**TransactionFactory**

It is a factory of Transaction. It is optional.

• Hibernate Example using XML in Eclipse

• Example to create the Hibernate Application in Eclipse IDE

• Create the java project

• Add jar files for hibernate

• Create the Persistent class

• Create the mapping file for Persistent class

• Create the Configuration file

• Create the class that retrieves or stores the persistent object

• Run the application

Here, we are going to create a simple example of hibernate application using eclipse IDE. For creating the first hibernate application in Eclipse IDE, we need to follow the following steps:

• Create the java project

• Add jar files for hibernate

• Create the Persistent class

• Create the mapping file for Persistent class

• Create the Configuration file

• Create the class that retrieves or stores the persistent object

• Run the application

1) Create the java project

Create the java project by File Menu - New - project - java project. Now specify the project name e.g. firsthb then next - finish.

2) Add jar files for hibernate

To add the jar files Right click on your project - Build path - Add external archives. Now select all the jar files as shown in the image given below then click open.

Download the required jar file

In this example, we are connecting the application with oracle database. So you must add the ojdbc14.jar file.

download the ojdbc14.jar file

3) Create the Persistent class

Here, we are creating the same persistent class which we have created in the previous topic. To create the persistent class, Right click on src - New - Class - specify the class with package name (e.g. com.javatpoint.mypackage) - finish .

Employee.java

1. package com.javatpoint.mypackage;

2.

3. public class Employee {

4. private int id;

5. private String firstName,lastName;

6.

7. public int getId() {

8. return id;

9. }

10. public void setId(int id) {

11. this.id = id;

12. }

13. public String getFirstName() {

14. return firstName;

15. }

16. public void setFirstName(String firstName) {

17. this.firstName = firstName;

18. }

19. public String getLastName() {

20. return lastName;

21. }

22. public void setLastName(String lastName) {

23. this.lastName = lastName;

24. }

25. }

4) Create the mapping file for Persistent class

Here, we are creating the same mapping file as created in the previous topic. To create the mapping file, Right click on src - new - file - specify the file name (e.g. employee.hbm.xml) - ok. It must be outside the package.

employee.hbm.xml

1. <?xml version='1.0' encoding='UTF-8'?>

2. <!DOCTYPE hibernate-mapping PUBLIC

3. "-//Hibernate/Hibernate Mapping DTD 5.3//EN"

4. "http://hibernate.sourceforge.net/hibernate-mapping-5.3.dtd">

5.

6. <hibernate-mapping>

7. <class name="com.javatpoint.mypackage.Employee" table="emp1000">

8. <id name="id">

9. <generator class="assigned"></generator>

10. </id>

11.

12. <property name="firstName"></property>

13. <property name="lastName"></property>

14.

15. </class>

16.

17. </hibernate-mapping>

5) Create the Configuration file

The configuration file contains all the informations for the database such as connection\_url, driver\_class, username, password etc. The hbm2ddl.auto property is used to create the table in the database automatically. We will have in-depth learning about Dialect class in next topics. To create the configuration file, right click on src - new - file. Now specify the configuration file name e.g. hibernate.cfg.xml.

hibernate.cfg.xml

1. <?xml version='1.0' encoding='UTF-8'?>

2. <!DOCTYPE hibernate-configuration PUBLIC

3. "-//Hibernate/Hibernate Configuration DTD 5.3//EN"

4. "http://hibernate.sourceforge.net/hibernate-configuration-5.3.dtd">

5.

6. <hibernate-configuration>

7.

8. <session-factory>

9. <property name="hbm2ddl.auto">update</property>

10. <property name="dialect">org.hibernate.dialect.Oracle9Dialect</property>

11. <property name="connection.url">jdbc:oracle:thin:@localhost:1521:xe</property>

12. <property name="connection.username">system</property>

13. <property name="connection.password">oracle</property>

14. <property name="connection.driver\_class">oracle.jdbc.driver.OracleDriver</property>

15. <mapping resource="employee.hbm.xml"/>

16. </session-factory>

17.

18. </hibernate-configuration>

6) Create the class that retrieves or stores the persistent object

In this class, we are simply storing the employee object to the database.

1. package com.javatpoint.mypackage;

2.

3. import org.hibernate.Session;

4. import org.hibernate.SessionFactory;

5. import org.hibernate.Transaction;

6. import org.hibernate.boot.Metadata;

7. import org.hibernate.boot.MetadataSources;

8. import org.hibernate.boot.registry.StandardServiceRegistry;

9. import org.hibernate.boot.registry.StandardServiceRegistryBuilder;

10.

11. public class StoreData {

12.

13. public static void main( String[] args )

14. {

15. StandardServiceRegistry ssr = new StandardServiceRegistryBuilder().configure("hibernate.cfg.xml").build();

16. Metadata meta = new MetadataSources(ssr).getMetadataBuilder().build();

17.

18. SessionFactory factory = meta.getSessionFactoryBuilder().build();

19. Session session = factory.openSession();

20. Transaction t = session.beginTransaction();

21.

22. Employee e1=new Employee();

23. e1.setId(1);

24. e1.setFirstName("Gaurav");

25. e1.setLastName("Chawla");

26.

27. session.save(e1);

28. t.commit();

29. System.out.println("successfully saved");

30. factory.close();

31. session.close();

32. }

33. }

7) Run the application

Before running the application, determine that directory structure is like this.

To run the hibernate application, right click on the StoreData class - Run As - Java Application.