Section-End Project - Lesson 4

Hibernate

We will write a simple hibernate application to get started with hibernate.

Before starting, you should:

* Download hibernate4 distribution (with required jars). Refer to <http://javajee.com/downloading-hibernate-4-jar-files>.
* Install any database server (eg. MySQL) and download its jdbc driver jar file.

****Follow the below steps:****

****Step 1:****Create an eclipse project and add below jars to classpath:

* All jars under lib\required.
* Your jdbc driver jar (This will be specific to the database you are using. I am using mysql, hence using mysql-connector-java-5.1.26-bin.jar inside the zip version of the downloaded connector.)

ECLIPSE-NOTE: You can add jars to classpath in eclipse by right clicking your Java project in eclipse and selecting Build Path > Configure Build Path. Now select Library tab, click add external jars and add jars file by file or folder by folder.

****Step 2:**** Create a simple java class for a ‘User’ with two fields, an Id and a name. Generate setters and getters. You can generate getters and setters by right clicking over a field, select source, select generate getters and setters. We will use @Entity annotation (ijavax.persistence.Entity) over the class to tell hibernate to treat this as our entity class that needs to be saved and then @Id (javax.persistence.Id) over the id field to tell hibernate to make this field as the primary key of the table. Note that we import from javax.persistence package, not from org.hibernate.annotations. Our sample class will look like:

package com.javajee.hiberex.dto;

import javax.persistence.Entity;

import javax.persistence.Id;

@Entity

public class User {

               @Id

               int id;

               private String name;

               public int getId() {

                              return id;

               }

               public void setId(int id) {

                              this.id = id;

               }

               public String getName() {

                              return name;

               }

               public void setName(String name) {

                              this.name = name;

               }

}

****Step 3:****Create hibernate configuration xml file with the default name hibernate.cfg.xml in the src folder. By using the default name, you don’t have to tell hibernate about this file, but hibernate will search and find automatically. To make things simple we can open a configuration xml file that comes with the distribution, copy its contents and modify it to suit our needs. Go within the subdirectories of project folder and search for hibernate.cfg.xml. Check if tags for properties like connection.driver\_class,connection.url, connection.username, connection.password, dialect, hbm2ddl.auto and mapping are present in the one you select.Now copy the contents of the example file to our hibernate.cfg.xml we just created and start modifying as mentioned in next steps.

ECLIPSE-NOTE: You can create a new file in eclipse by right clicking, selecting create new > file, and give a name and click finish.

****Step 4:**** First, configure the database specific properties according to the database you use. You can find the driver by expanding your driver jar under ‘Referenced Libraries’ in the package explorer in eclipse. Below are the details I have updated for my MySQL:

 <!-- Database connection settings -->

<property name="connection.driver\_class">com.mysql.jdbc.Driver</property>

<property name="connection.url">jdbc:mysql://localhost:3306/hiberdb</property>

<property name="connection.username">root</property>

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

3306 is the default port for MySQL if you have not specified a different one while installing MySQL. ****Hibernate will not create the database, so you should create the database (eg. hiberdb in this case)****

****Step 5:****Set the correct dialect for your database. Even among databases that use SQL, there might be some database specific syntaxes. SQL dialect tells hibernate which of that syntax to use. Expand the hibernate jar under ‘Referenced Libraries’ in the package explorer in eclipse, check under org.hibernate.dialect for your database specific classname and update it for dialect property.

<property name="dialect">org.hibernate.dialect.MySQL5Dialect</property>

****Step 6:****Set show\_sql property to true. If this property is set to true, hibernate will print all sql statements it executes in the console. It would be good to see as we are learning. If you taken the file I have taken from the project folder, you will not see this and hence might need to add it to the file.

<property name="show\_sql">true</property>

****Step 7:**** Now set the value of hbm2ddl.auto to create. The values for this property are validate, update, create, create-drop.  ‘validate’ will validate the schema that you have, and makes no changes to the database, ‘update’ will update the database tables if tables are already there in the database, ‘create’ will create new database tables, removing existing tables if any, and ‘create-drop’ will create new database tables and drops when the sessionFactory is destroyed.

<property name="hbm2ddl.auto">create</property>

****Step 8:****We have to configure the mapping classes next. Here we have to configure all the entity model classes that we have created.

<mapping class="com.javajee.hiberex.dto.User"/>

Final hibernate.cfg.xml will look as:

<?xml version='1.0' encoding='utf-8'?>

<!DOCTYPE hibernate-configuration PUBLIC

"-//Hibernate/Hibernate Configuration DTD//EN"

"[http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd](http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd" \t "https://javajee.com/_blank)">

<hibernate-configuration>

<session-factory>

<property name="hibernate.connection.driver\_class">com.mysql.jdbc.Driver</property>

<property name="hibernate.connection.url">jdbc:mysql://localhost:3306/hiberdb</property>

<property name="hibernate.connection.username">root</property>

<property name="hibernate.connection.password"></property>

<property name="hibernate.dialect">org.hibernate.dialect.MySQL5Dialect</property>

<property name="show\_sql">true</property>

<property name="hibernate.hbm2ddl.auto">create</property>

<mapping class="com.javajee.hiberex.dto.User" />

</session-factory>

</hibernate-configuration>

****Step 9:**** Create a test class (UserTest) with a main method that creates an object of the User class and save it to database using hibernate. To save an object with hibernate, we have to first create a session factory, then create a session from that session factory and finally use the session to save the model object within a transaction.

    Configuration configuration = new Configuration();

    configuration.configure();

    ServiceRegistry serviceRegistry = new StandardServiceRegistryBuilder().applySettings(configuration.getProperties()).build();

****SessionFactory sessionFactory**** = configuration.buildSessionFactory(serviceRegistry);

****Session session**** = sessionFactory.openSession();

    session.beginTransaction();

    //any operation

    session.getTransaction().commit();

If you have seen hibernate3 code you might have created a SessionFactory as:

SessionFactory sessionFactory = new Configuration().configure().****buildSessionFactory()****;

However buildSessionFactory() from the type Configuration is deprecated in hibernate 4 and we need to use buildSessionFactory(ServiceRegistry) method instead as above.

You can also add code to get the result from database and verify. Similar to saving, we have to create a session from the session factory and finally get data using the session.get method. A session factory is usually created once and reused throughout the application as it is a costly process to create a session factory. We need to pass the class object of the entity class and primary key to the session.get method.

The final class will look as below:

import org.hibernate.Session;

import org.hibernate.SessionFactory;

import org.hibernate.cfg.Configuration;

import org.hibernate.service.ServiceRegistry;

import com.javajee.hiberex.dto.User;

public class UserTest {

  public static void main(String[] args) {

    User u = new User();

    u.setId(1);

    u.setName("Heartin");

    Configuration configuration = new Configuration();

    configuration.configure();

    ServiceRegistry serviceRegistry = new StandardServiceRegistryBuilder().applySettings(configuration.getProperties()).build();

    SessionFactory sessionFactory = configuration.buildSessionFactory(serviceRegistry);

    Session session = sessionFactory.openSession();

    session.beginTransaction();

    session.save(u);

    session.getTransaction().commit();

    // Below code get data back from db and print it.

    u = null;

    session = sessionFactory.openSession();

    session.beginTransaction();

    u = (User) session.get(User.class, 1);

    session.getTransaction().commit();

    System.out.println("Name retrieved is:" + u.getName());

  }

}

****Step 10:**** Run the test class and see the result in the console. You can also verify the result by examining the data in the corresponding table in the database.

****Step 11:**** Now, if you run again and again, the current schema in the database will be deleted and current data will be inserted. Change the property hbm2ddl.auto in the hibernate.cfg.xml from ‘create’ to ‘update’ as:

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

and then run the same test class again and again. Now you will get an exception:

org.hibernate.exception.ConstraintViolationException due to com.mysql.jdbc.exceptions.jdbc4.MySQLIntegrityConstraintViolationException: Duplicate entry '1' for key 'PRIMARY'.

This is because since you use update, the schema will only be create if it doesn’t exist and will not be created every time, and hence adding same key twice will throw exception.

****\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*****