Hibernate - Examples

Previous Page Next Page

Let us now take an example to understand how we can use Hibernate to provide Java persistence in a standalone application. We will go through the different steps involved in creating a Java Application using Hibernate technology.

Create POJO Classes

The first step in creating an application is to build the Java POJO class or classes, depending on the application that will be persisted to the database. Let us consider our **Employee** class with **getXXX** and **setXXX** methods to make it JavaBeans compliant class.

A POJO (Plain Old Java Object) is a Java object that doesn't extend or implement some specialized classes and interfaces respectively required by the EJB framework. All normal Java objects are POJO.

When you design a class to be persisted by Hibernate, it is important to provide JavaBeans compliant code as well as one attribute, which would work as index like **id** attribute in the Employee class.

```
public class Employee {
   private int id;
   private String firstName;
   private String lastName;
   private int salary;

   public Employee() {}
   public Employee(String fname, String lname, int salary) {
      this.firstName = fname;
      this.lastName = lname;
      this.salary = salary;
   }

   public int getId() {
      return id;
   }

   public void setId( int id ) {
      this.id = id;
   }
```

```
   public String getFirstName() {
      return firstName;
   }

   public void setFirstName( String first_name ) {
      this.firstName = first_name;
   }

   public String getLastName() {
      return lastName;
   }

   public void setLastName( String last_name ) {
      this.lastName = last_name;
   }

   public int getSalary() {
      return salary;
   }

   public void setSalary( int salary ) {
      this.salary = salary;
   }
}
```

Create Database Tables

Second step would be creating tables in your database. There would be one table corresponding to each object, you are willing to provide persistence. Consider above objects need to be stored and retrieved into the following RDBMS table –

```
create table EMPLOYEE (
   id INT NOT NULL auto_increment,
   first_name VARCHAR(20) default NULL,
   last_name  VARCHAR(20) default NULL,
   salary     INT  default NULL,
   PRIMARY KEY (id)
);
```

Create Mapping Configuration File

This step is to create a mapping file that instructs Hibernate how to map the defined class or classes to the database tables.

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

<hibernate-mapping>
   <class name = "Employee" table = "EMPLOYEE">

      <meta attribute = "class-description">
         This class contains the employee detail.
```

```
      </meta>

      <id name = "id" type = "int" column = "id">
         <generator class="native"/>
      </id>

      <property name = "firstName" column = "first_name" type = "string"/>
      <property name = "lastName" column = "last_name" type = "string"/>
      <property name = "salary" column = "salary" type = "int"/>

   </class>
</hibernate-mapping>
```

You should save the mapping document in a file with the format <classname>.hbm.xml. We saved our mapping document in the file Employee.hbm.xml. Let us see little detail about the mapping document –

The mapping document is an XML document having <hibernate-mapping> as the root element which contains all the <class> elements.

The **<class>** elements are used to define specific mappings from a Java classes to the database tables. The Java class name is specified using the **name** attribute of the class element and the database table name is specified using the **table** attribute.

The **<meta>** element is optional element and can be used to create the class description.

The **<id>** element maps the unique ID attribute in class to the primary key of the database table. The **name** attribute of the id element refers to the property in the class and the **column** attribute refers to the column in the database table. The **type** attribute holds the hibernate mapping type, this mapping types will convert from Java to SQL data type.

The **<generator>** element within the id element is used to generate the primary key values automatically. The **class** attribute of the generator element is set to **native** to let hibernate pick up either **identity, sequence** or **hilo** algorithm to create primary key depending upon the capabilities of the underlying database.

The **<property>** element is used to map a Java class property to a column in the database table. The **name** attribute of the element refers to the property in the class and the **column** attribute refers to the column in the database table. The **type** attribute holds the hibernate mapping type, this mapping types will convert from Java to SQL data type.

There are other attributes and elements available, which will be used in a mapping document and I would try to cover as many as possible while discussing other Hibernate related topics.

Create Application Class

Finally, we will create our application class with the main() method to run the application. We will use this application to save few Employee's records and then we will apply CRUD operations on those records.

```
import java.util.List;
import java.util.Date;
import java.util.Iterator;

import org.hibernate.HibernateException;
import org.hibernate.Session;
import org.hibernate.Transaction;
import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;

public class ManageEmployee {
   private static SessionFactory factory;
   public static void main(String[] args) {

      try {
         factory = new Configuration().configure().buildSessionFactory();
      } catch (Throwable ex) {
         System.err.println("Failed to create sessionFactory object." + ex);
         throw new ExceptionInInitializerError(ex);
      }

      ManageEmployee ME = new ManageEmployee();

      /* Add few employee records in database */
      Integer empID1 = ME.addEmployee("Zara", "Ali", 1000);
      Integer empID2 = ME.addEmployee("Daisy", "Das", 5000);
      Integer empID3 = ME.addEmployee("John", "Paul", 10000);

      /* List down all the employees */
      ME.listEmployees();

      /* Update employee's records */
      ME.updateEmployee(empID1, 5000);

      /* Delete an employee from the database */
      ME.deleteEmployee(empID2);

      /* List down new list of the employees */
      ME.listEmployees();
   }

   /* Method to CREATE an employee in the database */
   public Integer addEmployee(String fname, String lname, int salary){
      Session session = factory.openSession();
      Transaction tx = null;
      Integer employeeID = null;

      try {
         tx = session.beginTransaction();
         Employee employee = new Employee(fname, lname, salary);
         employeeID = (Integer) session.save(employee);
         tx.commit();
```

```
      } catch (HibernateException e) {
         if (tx!=null) tx.rollback();
         e.printStackTrace();
      } finally {
         session.close();
      }
      return employeeID;
   }

   /* Method to  READ all the employees */
   public void listEmployees( ){
      Session session = factory.openSession();
      Transaction tx = null;

      try {
         tx = session.beginTransaction();
         List employees = session.createQuery("FROM Employee").list();
         for (Iterator iterator = employees.iterator(); iterator.hasNext();){
            Employee employee = (Employee) iterator.next();
            System.out.print("First Name: " + employee.getFirstName());
            System.out.print("  Last Name: " + employee.getLastName());
            System.out.println("  Salary: " + employee.getSalary());
         }
         tx.commit();
      } catch (HibernateException e) {
         if (tx!=null) tx.rollback();
         e.printStackTrace();
      } finally {
         session.close();
      }
   }

   /* Method to UPDATE salary for an employee */
   public void updateEmployee(Integer EmployeeID, int salary ){
      Session session = factory.openSession();
      Transaction tx = null;

      try {
         tx = session.beginTransaction();
         Employee employee = (Employee)session.get(Employee.class, EmployeeID);
         employee.setSalary( salary );
                 session.update(employee);
         tx.commit();
      } catch (HibernateException e) {
         if (tx!=null) tx.rollback();
         e.printStackTrace();
      } finally {
         session.close();
      }
   }

   /* Method to DELETE an employee from the records */
   public void deleteEmployee(Integer EmployeeID){
      Session session = factory.openSession();
      Transaction tx = null;

      try {
         tx = session.beginTransaction();
         Employee employee = (Employee)session.get(Employee.class, EmployeeID);
         session.delete(employee);
         tx.commit();
```

```
      } catch (HibernateException e) {
         if (tx!=null) tx.rollback();
         e.printStackTrace();
      } finally {
         session.close();
      }
   }
}
```

Compilation and Execution

Here are the steps to compile and run the above mentioned application. Make sure, you have set PATH and CLASSPATH appropriately before proceeding for the compilation and execution.

- Create hibernate.cfg.xml configuration file as explained in configuration chapter.
- Create Employee.hbm.xml mapping file as shown above.
- Create Employee.java source file as shown above and compile it.
- Create ManageEmployee.java source file as shown above and compile it.
- Execute ManageEmployee binary to run the program.

You would get the following result, and records would be created in the EMPLOYEE table.

```
$java ManageEmployee
.......VARIOUS LOG MESSAGES WILL DISPLAY HERE........

First Name: Zara  Last Name: Ali  Salary: 1000
First Name: Daisy  Last Name: Das  Salary: 5000
First Name: John  Last Name: Paul  Salary: 10000
First Name: Zara  Last Name: Ali  Salary: 5000
First Name: John  Last Name: Paul  Salary: 10000
```

If you check your EMPLOYEE table, it should have the following records –

```
mysql> select * from EMPLOYEE;
+----+------------+-----------+--------+
| id | first_name | last_name | salary |
+----+------------+-----------+--------+
| 29 | Zara       | Ali       |   5000 |
| 31 | John       | Paul      |  10000 |
+----+------------+-----------+--------+
2 rows in set (0.00 sec

mysql>
```

Previous Page Next Page

Enter email for newsletter go