

***HIBERNATE***

**Training Lab**

**XML Configuration & CRUD Operations**

|  |  |
| --- | --- |
| Document Code | 25e-BM/HR/HDCV/FSOFT |
| Version | 1.1 |
| Effective Date | 20/11/2012 |

**Hanoi, 06/2020**

RECORD OF CHANGES

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| No | Effective Date | Change Description | Reason | Reviewer | Approver |
| 1 | 15/06/2020 | Create Lab |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

Contents

**1. Problem Description 4**

**2. Technologies and Tools Used 4**

**3. Development Steps 5**

3.1, Project Directory Structure 5

3.2, Add jar Dependencies to pom.xml 6

3.3, Creating the JPA Entity Class (Persistent class) 7

3.4, ★Create a Hibernate configuration file - hibernate.cfg.xml 8

3.5, Create a Hibernate utility class 9

3.6, ★Implement CRUD operations 10

3.7, Create the class for testing 14

|  |  |
| --- | --- |
|  | **CODE: ORM.L.A101**  **TYPE: LONG**  **LOC: NA**  **DURATION: 480 Minutes (completed in 2 work days)** |

1. Problem Description

* We will show you step by step hibernate application to demonstrate the use of hibernate.cfg.xml configuration to connect database. We will define a mapping between Student Java class and database table using Hibernate ORM Framework.

**id**: int

**firstName**: String

**lastName**: String

**email**: String

**Class [Student.java]**

**id**: INT

**firstName**: VARCHAR

**lastName**: VARCHAR

**email**: VARCHAR

**Table [student]**

HIBERNATE MAPPING

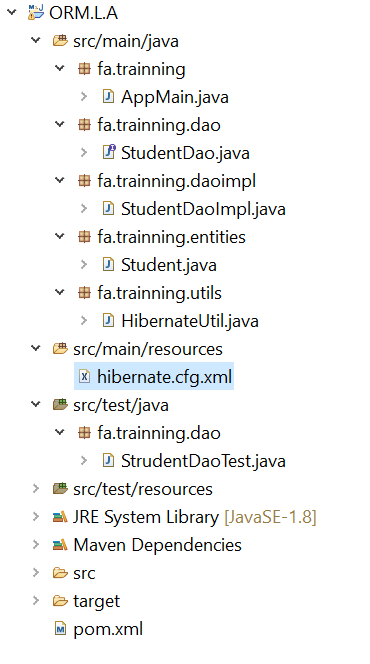
* CRUD operations are Create(save), Read(select), Update(update) and Delete(delete). Hibernate has Session interface which provides many APIs to perform operations with database.
* Demo how to using **@NamedQueries** Annotaion to query data.

2. Technologies and Tools Used

* Hibernate 5.4
* IDE - Eclipse
* Maven
* JavaSE 1.8
* MySQL

3. Development Steps

**3.1, Project Directory Structure**

**3.2, Add jar Dependencies to pom.xml**

Session factory configuration

Hibernate configuration

JPA Entity

<project

xmlns="http://maven.apache.org/POM/4.0.0"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0

http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<parent>

<groupId>hibernate</groupId>

<artifactId>hibernate-lab</artifactId>

<version>0.0.1-SNAPSHOT</version>

</parent>

<artifactId>hibernate-lab1</artifactId>

<properties>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

</properties>

<dependencies>

**<!-- https://mvnrepository.com/artifact/mysql/mysql-connector-java -->**

<dependency>

<groupId>mysql</groupId>

<artifactId>**mysql-connector-java**</artifactId>

<version>8.0.13</version>

</dependency>

**<!-- https://mvnrepository.com/artifact/org.hibernate/hibernate-core -->**

<dependency>

<groupId>org.hibernate</groupId>

<artifactId>**hibernate-core**</artifactId>

<version>5.4.18.Final</version>

</dependency>

</dependencies>

<build>

<sourceDirectory>src/main/java</sourceDirectory>

<plugins>

<plugin>

<artifactId>maven-compiler-plugin</artifactId>

<version>3.5.1</version>

<configuration>

<source>1.8</source>

<target>1.8</target>

</configuration>

</plugin>

</plugins>

</build>

</project>

**3.3, Creating the JPA Entity Class (Persistent class)**

Create a Student entity class under fa.trainning.entities package as follows.

package fa.trainning.entities;

import javax.persistence.Column;

import javax.persistence.Entity;

import javax.persistence.GeneratedValue;

import javax.persistence.GenerationType;

import javax.persistence.Id;

import javax.persistence.Table;

**@Entity**

**@Table**(name = "student")

@NamedNativeQueries({

@NamedNativeQuery(name = "**GET\_STUDENT\_BY\_ID**", query = "SELECT \* FROM student where id=:id")

})

public class Student {

**@Id**

@GeneratedValue(strategy = GenerationType.IDENTITY)

@Column(name = "id")

private int id;

@Column(name = "first\_name")

private String firstName;

@Column(name = "last\_name")

private String lastName;

@Column(name = "email")

private String email;

public Student() {

}

public Student(String firstName, String lastName, String email) {

this.firstName = firstName;

this.lastName = lastName;

this.email = email;

}

// getter setter

// …

}

**3.4, ★Create a Hibernate configuration file - hibernate.cfg.xml**

The configuration file contains information about the database and mapping file. Conventionally, its name should be hibernate.cfg.xml.

Let's create an XML file named as hibernate.cfg.xml under resources folder and write the following code in it.

<!DOCTYPE hibernate-configuration PUBLIC

"-//Hibernate/Hibernate Configuration DTD 3.0//EN"

"http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>

<session-factory>

<!-- JDBC Database connection settings -->

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

**<property name="connection.url">jdbc:mysql://localhost:3306/hibernatelab\_db?useSSL=false</property>**

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

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

<!-- JDBC connection pool settings ... using built-in test pool -->

<property name="connection.pool\_size">1</property>

<!-- Select our SQL dialect -->

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

<!-- Echo the SQL to stdout -->

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

<!-- Set the current session context -->

<property name="current\_session\_context\_class">thread</property>

<!-- Drop and re-create the database schema on startup -->

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

<!-- dbcp connection pool configuration -->

<property name="hibernate.dbcp.initialSize">5</property>

<property name="hibernate.dbcp.maxTotal">20</property>

<property name="hibernate.dbcp.maxIdle">10</property>

<property name="hibernate.dbcp.minIdle">5</property>

<property name="hibernate.dbcp.maxWaitMillis">-1</property>

<mapping class="**fa.trainning.entities.Student**" />

</session-factory>

</hibernate-configuration>

**3.5, Create a Hibernate utility class**

The bootstrapping API is quite flexible, but in most cases, it makes the most sense to think of it as a 3 steps process:

* Build the StandardServiceRegistry
* Build the Metadata
* Use those 2 to build the SessionFactory

package fa.trainning.utils;

import org.hibernate.SessionFactory;

import org.hibernate.boot.Metadata;

import org.hibernate.boot.MetadataSources;

import org.hibernate.boot.registry.StandardServiceRegistry;

import org.hibernate.boot.registry.StandardServiceRegistryBuilder;

public class HibernateUtil {

private static StandardServiceRegistry registry;

private static SessionFactory sessionFactory;

public static SessionFactory getSessionFactory() {

if (sessionFactory == null) {

try {

// Create registry

registry = new StandardServiceRegistryBuilder().configure().build();

// Create MetadataSources

MetadataSources sources = new MetadataSources(registry);

// Create Metadata

Metadata metadata = sources.getMetadataBuilder().build();

// Create SessionFactory

sessionFactory = metadata.getSessionFactoryBuilder().build();

} catch (Exception e) {

e.printStackTrace();

if (registry != null) {

StandardServiceRegistryBuilder.destroy(registry);

}

}

}

return sessionFactory;

}

public static void shutdown() {

if (registry != null) {

StandardServiceRegistryBuilder.destroy(registry);

}

}

}

**3.6, ★Implement** **CRUD operations**

Create a StudentDao interface and StudentDaoImpl class as follows

* Interface

package fa.trainning.dao;

import fa.trainning.entities.Student;

public interface StudentDao {

public void saveStudent(Student student);

public Student getStudent(int id);

public Student loadStudent(int id);

public Student getStudentUsingNamedQuery(int id);

public void saveOrUpdateStudent(Student student);

public void deleteStudent(int id);

public void removeStudent(int id);

}

* Implementation class

**C**reate (save an entity)

package fa.trainning.daoimpl;

import org.hibernate.Session;

import fa.trainning.entities.Student;

public class StudentDaoImpl implements StudentDao {

private Session session = HibernateUtil.getSessionFactory().openSession();

@Override

**public void saveStudent(Student student) {**

// Save the student object

session.save(student);

}

}

**R**ead (get an object or list of objects)

package fa.trainning.daoimpl;

import org.hibernate.Session;

import fa.trainning.entities.Student;

public class StudentDaoImpl implements StudentDao {

private Session session = HibernateUtil.getSessionFactory().openSession();

@Override

**public Student getStudent(int id) {**

// Get Student entity using get() method

Student student = session.get(Student.class, id);

// Return

return student;

}

@Override

**public Student loadStudent(int id) {**

// Get Student entity using load() method

Student student = session.load(Student.class, id);

// Return

return student;

}

@Override

**public Student** getStudentUsingNamedQuery**(int id) {**

// Executing named query

List<Student> students = session.createNamedQuery("**GET\_STUDENT\_BY\_ID**").

setParameter("id", id).getResultList;

Student student = null;

if (null != students) {

student = students.get(0);

}

// Return

return student;

}

}

**U**pdate

package fa.trainning.daoimpl;

import org.hibernate.Session;

import fa.trainning.entities.Student;

public class StudentDaoImpl implements StudentDao {

private Session session = HibernateUtil.getSessionFactory().openSession();

**public void saveOrUpdateStudent(Student student) {**

// Save the student object

session.saveOrUpdate(student);

}

}

**D**elete

* Using Session.delete() method:

package fa.trainning.daoimpl;

import org.hibernate.Session;

import fa.trainning.entities.Student;

public class StudentDaoImpl implements StudentDao {

private static Session session = HibernateUtil.getSessionFactory().openSession();

**public void deleteStudent(int id) {**

// Delete a persistent object

Student student = session.get(Student.class, id);

if (student != null) {

session.delete(student);

}

}

}

* Using Session.remove() method:

package fa.trainning.daoimpl;

import org.hibernate.Session;

import fa.trainning.entities.Student;

public class StudentDaoImpl implements StudentDao {

private static Session session = HibernateUtil.getSessionFactory().openSession();

**public void removeStudent(int id) {**

// Delete a persistent object

Student student = session.get(Student.class, id);

if (student != null) {

session.remove(student);

}

}

}

**3.7, Create the class for testing**

Here is class which is used to connect MySQL database and persist Student object in database table.

* Main class (test for **C**reate)

package fa.trainning;

import java.util.List;

import org.hibernate.Session;

import org.hibernate.Transaction;

import fa.trainning.entities.Student;

import fa.trainning.utils.HibernateUtil;

import fa.trainning.dao.StudentDao;

public class App {

public static void main(String[] args) {

// Create new objects: DAO and Student

StudentDao studentDao = new StudentDao();

Student student = new Student("Anna", "Hook", "anna@fsoft.com.vn");

// Create Transaction

Transaction transaction = null;

try (**Session session = HibernateUtil.getSessionFactory().openSession()**) {

// Start a transaction

transaction = session.**beginTransaction**();

// TODO: add you code here... **BEGIN**

// Ex: save the student objects

studentDao.saveStudent(student);

// TODO: add you code here... **END**

// Commit transaction

transaction.**commit**();

} catch (Exception e) {

if (transaction != null) {

transaction.rollback();

}

e.printStackTrace();

}

}

}

* JUnitTest class class (test for **R**ead)

package fa.trainning.dao;

import org.junit.Assert;

import org.junit.Test;

import org.junit.runner.RunWith;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.test.context.ContextConfiguration;

import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;

import fa.trainning.entities.Student;

import fa.trainning.dao.StudentDao;

@RunWith(SpringJUnit4ClassRunner.class)

public class StudentDaoTest {

@Autowired

StudentDao studentDao;

**@Test**

public void **getStudentTest** (){

Student student = studentDao.**getStudent**(1);

Assert.assertNotNull(student);

Assert.assertEquals("Anna", student.getFirstName());

Assert.assertEquals("Hook", student.getLastName());

Assert.assertEquals("anna@fsoft.com.vn", student.getEmail());

}

}

**-- THE END --**