Spring ORM Example using Hibernate

Spring ORM is a module of the Java Spring framework used to implement the ORM(Object Relational Mapping) Technique. It can be integrated with various mapping and persistence frameworks like Hibernate, Oracle Toplink, iBatis, etc. for database access and manipulation. This article covers an example of the integration of the Spring ORM module with Hibernate framework.

Prerequisites:

- Java basics
- Spring core
- Hibernate or any other ORM tool

Spring ORM provides various classes and interfaces for integrating Spring applications with Hibernate framework. Some useful classes in Spring ORM are:

- HibernateTemplate
- HibernateTransactionManager
- LocalSessionFactoryBean

HibernateTemplate is used to perform database operations. It provides various methods which facilitate the insertion, deletion, modification, and retrieval of data from the database. Useful methods of HibernateTemplate are as follows:

- *void clear()*
- *void delete(Object entity)*
- *<T> T get(Class<T> entityClass, Serializable id)*
- *<T> T load(Class<T> entityClass, Serializable id)*
- *<T> List<T> loadAll(Class<T> entityClass)*
- *Serializable save(Object entity)*
- *void saveOrUpdate(Object entity)*
- *void update(Object entity)*

HibernateTemplate requires an object of SessionFactory. **LocalSessionFactoryBean** is a class present in the Spring ORM module which provides the object of SessionFactory. LocalSessionFactoryBean takes the following properties:

- DataSource: contains information like driverClassName, URL, username, password, etc.
- HibernateProperties: used to set various hibernate properties like hibernate dialect, show SQL queries, etc.
- AnnotatedClasses/MappingResources: used to provide annotated beans or mapping resources for the entities based on which hibernate constructs the tables in the database.

HibernateTransactionManager is used to handle transactional logic when the application consists of data modification operations on the database.

Example

Given below is an example of Spring ORM with Hibernate using maven.

pom.xml

First, we need to add the following dependencies in pom.xml:

- Hibernate maven dependency
- Spring Core maven dependency
- Spring Context maven dependency
- Spring JDBC maven dependency
- Spring ORM maven dependency
- MySQL Connector Java maven dependency

XML

```
<?xml version="1.0" encoding="UTF-8"?>
<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>

    <groupId>org.example</groupId>
    <artifactId>SpringORM</artifactId>
    <version>1.0-SNAPSHOT</version>

    <properties>
        <maven.compiler.source>11</maven.compiler.source>
        <maven.compiler.target>11</maven.compiler.target>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    </properties>
    <dependencies>

        <!-- https://mvnrepository.com/artifact/org.springframework/spring-c
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-core</artifactId>
            <version>5.1.0.RELEASE</version>
        </dependency>

        <!-- https://mvnrepository.com/artifact/org.springframework/spring-c
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-context</artifactId>
            <version>5.1.0.RELEASE</version>
        </dependency>

        <!-- https://mvnrepository.com/artifact/org.springframework/spring-j
        <dependency>
             <groupId>org.springframework</groupId>
              <artifactId>spring-jdbc</artifactId>
              <version>5.1.0.RELEASE</version>
            </dependency>

        <!-- https://mvnrepository.com/artifact/mysql/mysql-connector-java -
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <version>8.0.12</version>
        </dependency>

        <!-- https://mvnrepository.com/artifact/org.hibernate/hibernate-core
        <dependency>
            <groupId>org.hibernate</groupId>
            <artifactId>hibernate-core</artifactId>
            <version>5.6.1.Final</version>
        </dependency>

        <!-- https://mvnrepository.com/artifact/org.springframework/spring-o
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-orm</artifactId>
            <version>5.1.0.RELEASE</version>
        </dependency>

    </dependencies>
</project>
```

Student.java

This example consists of a single bean-Student.

Java

```
package beans;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.Table;

@Entity
@Table(name="Student_Details")
public class Student {
    @Id
    @Column(name="Student_Id")
    private int id;
    @Column(name="Student_Name")
    private String name;

    public Student() {
    }

    public Student(int id, String name) {
        this.id = id;
        this.name = name;
    }

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

    public void setName(String name) {
        this.name = name;
    }

    public int getId() {
        return id;
    }

    public String getName() {
        return name;
    }

    @Override
    public String toString() {
        return "Student{" +
                "id=" + id +
                ", name='" + name + '\'' +
                '}';
    }
}
```

StudentDao.java

StudentDao is an interface that declares all the operations that can be performed on Student bean.

Java

```
package dao;

import beans.Student;
import java.util.List;

public interface StudentDao {
    public int insert(Student s);
    public void delete(int id);
    public void delete(Student s);
    public void update(Student s);
    public Student getStudent(int id);
    public List<Student> getAllStudents();
}
```

StudentDaoImpl.java

StudentDaoImpl is an implementing class of the StudentDao interface.

Java

```
package dao;

import beans.Student;
import org.springframework.orm.hibernate5.HibernateTemplate;
import org.springframework.transaction.annotation.Transactional;

import java.util.List;

public class StudentDaoImpl implements StudentDao{
    private HibernateTemplate hTemplate;

    public void sethTemplate(HibernateTemplate hTemplate) {
        this.hTemplate = hTemplate;
    }

    @Override
    @Transactional
    public int insert(Student s) {
        return (int) hTemplate.save(s);
    }

    @Override
    @Transactional
    public void delete(int id) {
        Student s=hTemplate.get(Student.class,id);
        hTemplate.delete(s);
    }

    @Override
    @Transactional
    public void delete(Student s) {
        hTemplate.delete(s);
    }

    @Override
    @Transactional
    public void update(Student s) {
        hTemplate.update(s);
    }

    @Override
    public Student getStudent(int id) {
        return hTemplate.get(Student.class,id);
    }

    @Override
    public List<Student> getAllStudents() {
        return hTemplate.loadAll(Student.class);
    }
}
```

ContextProvider.java

ContextProvider is a class that implements the Singleton design pattern to provide an ApplicationContext object.

Java

```java
package context;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class ContextProvider {
    private static ApplicationContext context;

    public static ApplicationContext provideContext()
    {
        if(context==null)
        {
            context=new ClassPathXmlApplicationContext("config.xml");
        }
        return context;
    }
}
```

config.xml

Configuration file for Spring. It defines the following beans:

XML

```xml
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:context="http://www.springframework.org/schema/context"
       xmlns:tx="http://www.springframework.org/schema/tx"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
                           https://www.springframework.org/schema/beans/spri
                           http://www.springframework.org/schema/context
                           http://www.springframework.org/schema/context/spr
                           http://www.springframework.org/schema/tx
                           http://www.springframework.org/schema/tx/spring-t
    <context:annotation-config/>
    <bean class="org.springframework.orm.hibernate5.HibernateTransactionMana
        <property name="sessionFactory" ref="sessionFactory"/>
    </bean>

    <tx:annotation-driven/>
    <bean class="org.springframework.jdbc.datasource.DriverManagerDataSource
        <property name="driverClassName" value="com.mysql.cj.jdbc.Driver"/>
        <property name="url" value="jdbc:mysql://localhost:3306/gfg?useSSL=f
        <property name="username" value="root"/>
        <property name="password" value="root"/>
    </bean>
    <bean class="org.springframework.orm.hibernate5.LocalSessionFactoryBean"
        <property name="dataSource" ref="ds"/>
        <property name="hibernateProperties" >
            <props>
                <prop key="hibernate.dialect">org.hibernate.dialect.MySQL8D
                <prop key="hibernate.show_sql">true</prop>
                <prop key="hibernate.format_sql">true</prop>
                <prop key="hibernate.hbm2ddl.auto">update</prop>
            </props>
        </property>
        <property name="annotatedClasses">
            <list>
                <value>beans.Student</value>
            </list>
        </property>
    </bean>
    <bean class="org.springframework.orm.hibernate5.HibernateTemplate" id="
        <property name="sessionFactory" ref="sessionFactory"/>
    </bean>
    <bean class="dao.StudentDaoImpl" id="stDao">
        <property name="hTemplate" ref="hTemplate"/>
    </bean>
</beans>
```

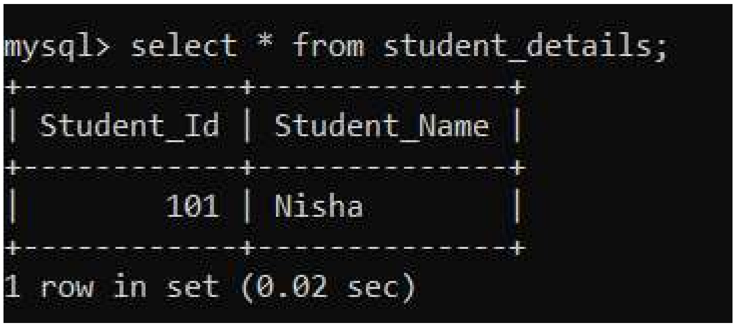
Inserting Student object into the database:

Insert.java

Java

```java
import beans.Student;
import context.ContextProvider;
import dao.StudentDao;
import org.springframework.context.ApplicationContext;

public class Insert {
    public static void main(String[] args) {

        ApplicationContext ctx= ContextProvider.provideContext();
        StudentDao studentDao=ctx.getBean("stDao",StudentDao.class);

        // insert
        Student s=new Student(101,"Nisha");
        studentDao.insert(s);

    }
}
```

Output:



```
mysql> select * from student_details;
+------------+--------------+
| Student_Id | Student_Name |
+------------+--------------+
|        101 | Nisha        |
+------------+--------------+
1 row in set (0.02 sec)
```

Updating Student details:

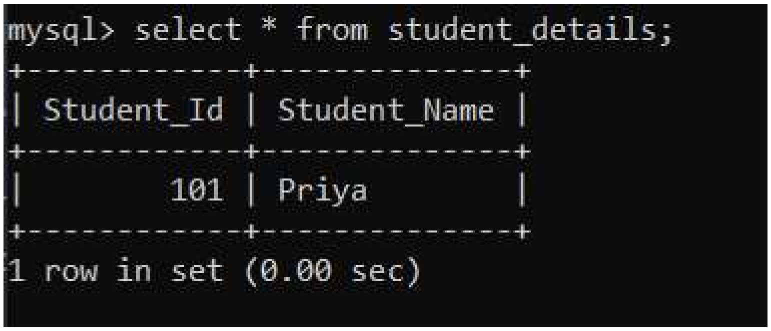
Update.java

Java

```java
import beans.Student;
import context.ContextProvider;
import dao.StudentDao;
import org.springframework.context.ApplicationContext;
```

```
public class Update {
    public static void main(String[] args) {

        ApplicationContext ctx= ContextProvider.provideContext();
        StudentDao studentDao=ctx.getBean("stDao",StudentDao.class);

        // update
        Student s=studentDao.getStudent(101);
        s.setName("Priya");
        studentDao.update(s);

    }
}
```

Output:



```
mysql> select * from student_details;
+------------+--------------+
| Student_Id | Student_Name |
+------------+--------------+
|        101 | Priya        |
+------------+--------------+
1 row in set (0.00 sec)
```

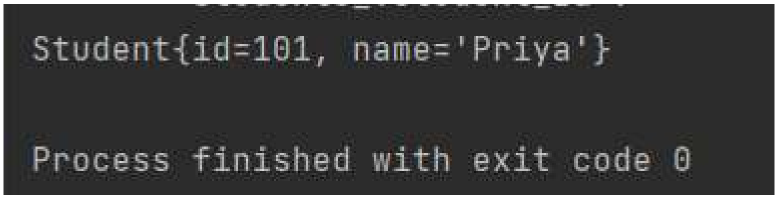
Retrieval of Student Object:

GetStudent.java

Java

```
import beans.Student;
import context.ContextProvider;
import dao.StudentDao;
import org.springframework.context.ApplicationContext;

public class GetStudent {
    public static void main(String[] args) {

        ApplicationContext ctx= ContextProvider.provideContext();
        StudentDao studentDao=ctx.getBean("stDao",StudentDao.class);
        // update
        Student s=studentDao.getStudent(101);
        System.out.println(s);

    }
}
```

Output:



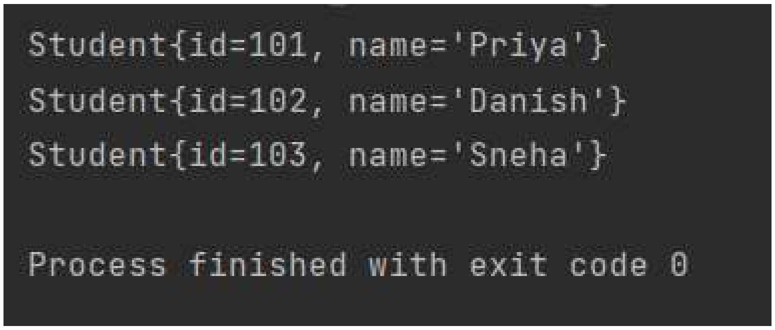
```
Student{id=101, name='Priya'}

Process finished with exit code 0
```

GetAllStudents.java

Java

```
import beans.Student;
import context.ContextProvider;
import dao.StudentDao;
import org.springframework.context.ApplicationContext;

import java.util.List;

public class GetAllStudents {
    public static void main(String[] args) {

        ApplicationContext ctx= ContextProvider.provideContext();
        StudentDao studentDao=ctx.getBean("stDao",StudentDao.class);

        studentDao.insert(new Student(102,"Danish"));
        studentDao.insert(new Student(103,"Sneha"));

        // update
        List<Student> students=studentDao.getAllStudents();
        for(Student s:students)
        {
            System.out.println(s);
        }

    }
}
```

Output:



```
Student{id=101, name='Priya'}
Student{id=102, name='Danish'}
Student{id=103, name='Sneha'}

Process finished with exit code 0
```

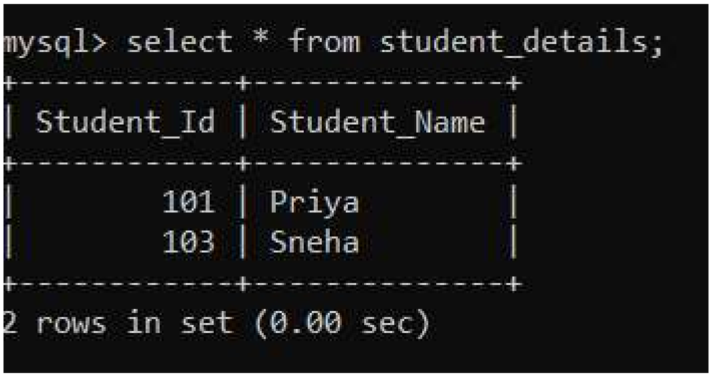
Deleting Student object from the database:

Delete.java

Java

```
import beans.Student;
import context.ContextProvider;
import dao.StudentDao;
import org.springframework.context.ApplicationContext;

import java.util.List;

public class Main {
    public static void Delete(String[] args) {

        ApplicationContext ctx= ContextProvider.provideContext();
        StudentDao studentDao=ctx.getBean("stDao",StudentDao.class);

        // delete
        studentDao.delete(102);
    }
}
```

Output:



```
mysql> select * from student_details;
+------------+--------------+
| Student_Id | Student_Name |
+------------+--------------+
|        101 | Priya        |
|        103 | Sneha        |
+------------+--------------+
2 rows in set (0.00 sec)
```

Last Updated : 31 Oct, 2022

3

Previous

Java - JPA vs Hibernate

Next

Hibernate - One-to-One Mapping

Share your thoughts in the comments

Add Your Comment

Similar Reads

- Difference Between Spring DAO vs Spring ORM vs Spring JDBC
- What is Spring Framework and Hibernate ORM?
- Hibernate - Difference Between ORM and JDBC
- Spring - Integration of Spring 4, Struts 2, and Hibernate
- Spring - ORM Framework
- Hibernate - Create Hibernate Configuration File with the Help of Plugin
- Spring MVC and Hibernate CRUD Example
- Spring Boot - Spring JDBC vs Spring Data JDBC
- Spring Boot - Validation using Hibernate Validator
- How to Create a Project using Spring MVC and Hibernate 5?

Complete Tutorials

- Java AWT Tutorial
- Spring MVC Tutorial
- Spring Tutorial
- Spring Boot Tutorial
- Java 8 Features - Complete Tutorial

P payalrath...

Article Tags : Java-Hibernate , Java-Spring , Technical Scripter 2022 , Java , Technical Scripter

Practice Tags : Java

Additional Information

GeeksforGeeks
Sanchhaya Education Private Limited
A-143, 9th Floor, Sovereign Corporate Tower, Sector-136, Noida, Uttar Pradesh - 201305

Company

About Us

Legal

Careers

In Media

Contact Us

Explore

Job-A-Thon Hiring Challenge

Hack-A-Thon

GfG Weekly Contest

Offline Classes (Delhi/NCR)

DSA in JAVA/C++

Advertise with us

GFG Corporate Solution

Placement Training Program

Apply for Mentor

Master System Design

Master CP

GeeksforGeeks Videos

Languages

Python

Java

C++

PHP

GoLang

SQL

R Language

Android Tutorial

Tutorials Archive

DSA

Data Structures

Algorithms

DSA for Beginners

Basic DSA Problems

DSA Roadmap

Top 100 DSA Interview Problems

DSA Roadmap by Sandeep Jain

All Cheat Sheets

Data Science & ML

Data Science With Python

Data Science For Beginner

Machine Learning Tutorial

ML Maths

Data Visualisation Tutorial

Pandas Tutorial

NumPy Tutorial

NLP Tutorial

Deep Learning Tutorial

HTML & CSS

HTML

CSS

Bootstrap

Tailwind CSS

SASS

LESS

Web Design

Python

Python Programming Examples

Django Tutorial

Python Projects

Python Tkinter

Computer Science

GATE CS Notes

Operating Systems

Computer Network

Database Management System