STRUTS 2 & HIBERNATE INTEGRATION

Hibernate is a high-performance Object/Relational persistence and query service which is licensed under the open source GNU Lesser General Public License *LGPL* and is free to download. In this chapter. we are going to learn how to achieve Struts 2 integration with Hibernate. If you are not familiar with Hibernate then you can check our Hibernate tutorial.

Database Setup

For this tutorial, I am going to use the "struts2_tutorial" MySQL database. I connect to this database on my machine using the username "root" and no password. First of all, you need to run the following script. This script creates a new table called **student** and creates few records in this table:

```
CREATE TABLE IF NOT EXISTS `student` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `first_name` varchar(40) NOT NULL,
  `last_name` varchar(40) NOT NULL,
  `marks` int(11) NOT NULL,
  PRIMARY KEY (`id`)
);

--
-- Dumping data for table `student`
--

INSERT INTO `student` (`id`, `first_name`, `last_name`, `marks`)
  VALUES(1, 'George', 'Kane', 20);
INSERT INTO `student` (`id`, `first_name`, `last_name`, `marks`)
  VALUES(2, 'Melissa', 'Michael', 91);
INSERT INTO `student` (`id`, `first_name`, `last_name`, `marks`)
  VALUES(3, 'Jessica', 'Drake', 21);
```

Hibernate Configuration

Next let us create the hibernate.cfg.xml which is the hibernate's configuration file.

```
<?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">

<hibernate-configuration>
<session-factory>
   <property name="hibernate.connection.driver_class">c
      om.mysql.jdbc.Driver
   </property>
   <property name="hibernate.connection.url">
      jdbc:mysql://www.tutorialspoint.com/struts_tutorial
   </property>
   <property name="hibernate.connection.username">root</property>
   <property name="hibernate.connection.password"></property>
   <property name="hibernate.connection.pool_size">10</property>
   <property name="show_sql">true</property>
   <property name="dialect">
      org.hibernate.dialect.MySQLDialect
   </property>
   <property name="hibernate.hbm2ddl.auto">update</property>
   <mapping  />
</session-factory>
</hibernate-configuration>
```

Let us go through the hibernate config file. First, we declared that we are using MySQL driver. Then

we declared the jdbc url for connecting to the database. Then we declared the connection's username, password and pool size. We also indicated that we would like to see the SQL in the log file by turning on "show_sql" to true. Please go through the hibernate tutorial to understand what these properties mean. Finally, we set the mapping class to com.tutorialspoint.hibernate.Student which we will create in this chapter.

Envrionment Setup

Next you need a whole lot of jars for this project. Attached is a screenshot of the complete list of JAR files required:

Most of the JAR files can be obtained as part of your struts distribution. If you have an application server such as glassfish, websphere or jboss installed then you can get the majority of the remaining jar files from the appserver's lib folder. If not you can download the files individually :

- Hibernate jar files - Hibernate.org
- Struts hibernate plugin - Struts hibernate plugin
- JTA files- JTA files
- Dom4j files - Dom4j
- SLF4J files - SLF4J
- log4j files - log4j

Rest of the files, you should be able to get from your struts2 distribution.

Hibernate Classes

Let us now create required java classes for the hibernate integration. Following the content of **Student.java**:

```
package com.tutorialspoint.hibernate;

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

@Entity
@Table(name="student")
public class Student {

   @Id
   @GeneratedValue
   private int id;
   @Column(name="last_name")
   private String lastName;
   @Column(name="first_name")
   private String firstName;
   private int marks;
   public int getId() {
    return id;
   }
   public void setId(int id) {
    this.id = id;
   }
   public String getLastName() {
      return lastName;
   }
   public void setLastName(String lastName) {
      this.lastName = lastName;
   }
   public String getFirstName() {
```

```
      return firstName;
   }
   public void setFirstName(String firstName) {
      this.firstName = firstName;
   }
   public int getMarks() {
      return marks;
   }
   public void setMarks(int marks) {
      this.marks = marks;
   }
}
```

This is a POJO class that represents the **student** table as per Hibernate specification. It has properties id, firstName and lastName which correspond to the column names of the student table. Next let us create **StudentDAO.java** file as follows:

```
package com.tutorialspoint.hibernate;

import java.util.ArrayList;
import java.util.List;

import org.hibernate.Session;
import org.hibernate.Transaction;

import com.googlecode.s2hibernate.struts2.plugin.\
                     annotations.SessionTarget;
import com.googlecode.s2hibernate.struts2.plugin.\
                     annotations.TransactionTarget;

public class StudentDAO {

   @SessionTarget
   Session session;

   @TransactionTarget
   Transaction transaction;

   @SuppressWarnings("unchecked")
   public List<Student> getStudents()
   {
      List<Student> students = new ArrayList<Student>();
      try
      {
         students = session.createQuery("from Student").list();
      }
      catch(Exception e)
      {
         e.printStackTrace();
      }
      return students;
   }

   public void addStudent(Student student)
   {
      session.save(student);
   }
}
```

The StudentDAO class is the data access layer for the Student class. It has methods to list all students and then to save a new student record.

Action Class

Following file **AddStudentAction.java** defines our action class. We have two action methods here - execute and listStudents. The execute method is used to add the new student record. We use the dao's save method to achieve this. The other method, listStudents is used to list the students. We use the dao's list method to get the list of all students.

```
package com.tutorialspoint.struts2;

import java.util.ArrayList;
import java.util.List;

import com.opensymphony.xwork2.ActionSupport;
import com.opensymphony.xwork2.ModelDriven;
import com.tutorialspoint.hibernate.Student;
import com.tutorialspoint.hibernate.StudentDAO;

public class AddStudentAction extends ActionSupport
            implements ModelDriven<Student>{

   Student student  = new Student();
   List<Student> students = new ArrayList<Student>();
   StudentDAO dao = new StudentDAO();
   @Override
   public Student getModel() {
      return student;
   }

   public String execute()
   {
      dao.addStudent(student);
      return "success";
   }

   public String listStudents()
   {
      students = dao.getStudents();
      return "success";
   }

   public Student getStudent() {
      return student;
   }

   public void setStudent(Student student) {
      this.student = student;
   }

   public List<Student> getStudents() {
      return students;
   }

   public void setStudents(List<Student> students) {
      this.students = students;
   }

}
```

You will notice that we are implementing the ModelDriven interface. This is used when your action class is dealing with a concrete model class *suchasStudent* as opposed to individual properties *suchasfirstName*, *lastName*. The ModelAware interface requires you to implement a method to return the model. In our case we are returning the "student" object.

Create view files

Let us now create the **student.jsp** view file with the following content:

```
<%@ page contentType="text/html; charset=UTF-8"%>
<%@ taglib prefix="s" uri="/struts-tags"%>
<html>
<head>
<title>Hello World</title>
<s:head />
</head>
```

```
<body>
   <s:form action="addStudent">
   <s:textfield name="firstName" label="First Name"/>
   <s:textfield name="lastName" label="Last Name"/>
   <s:textfield name="marks" label="Marks"/>
   <s:submit/>
   <hr/>
   <table>
      <tr>
         <td>First Name</td>
         <td>Last Name</td>
         <td>Marks</td>
      </tr>
      <s:iterator value="students">
         <tr>
            <td><s:property value="firstName"/></td>
            <td><s:property value="lastName"/></td>
            <td><s:property value="marks"/></td>
          </tr>
      </s:iterator>
   </table>
   </s:form>
</body>
</html>
```

The student.jsp is pretty straightforward. In the top section, we have a form that submits to "addStudent.action". It takes in firstName, lastName and marks. Because the addStudent action is tied to the ModelAware "AddSudentAction", automatically a student bean will be created with the values for firstName, lastName and marks auto populated.

At the bottom section, we go through the students list *seeAddStudentAction. java*. We iterate through the list and display the values for first name, last name and marks in a table.

Struts Configuration

Let us put it all together using **struts.xml**:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE struts PUBLIC
   "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
   "http://struts.apache.org/dtds/struts-2.0.dtd">

<struts>
   <constant name="struts.devMode" value="true" />

   <package name="myhibernate" extends="hibernate-default">

      <action name="addStudent" method="execute"
         >
         <result name="success" type="redirect">
               listStudents
         </result>
      </action>

      <action name="listStudents" method="listStudents"
         >
         <result name="success">/students.jsp</result>
      </action>

</package>

</struts>
```

The important thing to notice here is that our package "myhibernate" extends the struts2 default package called "hibernate-default". We then declare two actions - addStudent and listStudents. addStudent calls the execute on the AddStudentAction class and then upon successs,it calls the listStudents action method.

The listStudent action method calls the listStudents on the AddStudentAction class and uses the student.jsp as the view

Now right click on the project name and click **Export > WAR** File to create a War file. Then deploy this WAR in the Tomcat's webapps directory. Finally, start Tomcat server and try to access URL http://localhost:8080/HelloWorldStruts2/student.jsp. This will give you following screen:

In the top section, we get a form to enter the values for a new student record and the bottom section lists the students in the database. Go ahead and add a new student record and press submit. The screen will refresh and show you an updated list every time you click Submit.