Struts 2 y Hibernate Integración

Hibernate es un Framework de alto rendimiento Objeto / Relacional persistencia y la consulta que está bajo la licencia de código abierto GNU Lesser General Public License (LGPL) y se puede descargar gratis. En este capítulo. vamos a aprender cómo lograr Struts 2 integración con Hibernate. .

Configuración de base de datos

Para este tutorial, voy a utilizar la base de datos MySQL "struts2_tutorial". Me conecto a esta base de datos en mi máquina utilizando el nombre de usuario "root" y sin contraseña.En primer lugar, es necesario ejecutar el siguiente script. Este script crea una nueva tabla llamada **estudiante** y crea unos registros en esta tabla:

```
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);
```

Configuración de Hibernate

Siguiente vamos a crear el hibernate.cfg.xml que es el archivo de configuración de la hibernación.

```
<?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 class="com.tutorialspoint.hibernate.Student" />
</session-factory>
</hibernate-configuration>
```

Vayamos a través del archivo de configuración de hibernación. En primer lugar, declaramos que estamos utilizando controlador MySQL. Luego declaramos la URL de JDBC para conectarse a la base de datos. Luego declaramos la conexión de nombre de usuario, contraseña y tamaño de la piscina. También indicamos que nos gustaría ver el SQL en el archivo de registro mediante la activación de "show_sql" true. Por favor, pasar por el tutorial de hibernación para entender lo que significan estas propiedades. Por último, hemos creado la clase de mapeo para com.tutorialspoint.hibernate.Student que crearemos en este capítulo.

Configuración envrionment

A continuación, tiene un montón de frascos para este proyecto. Se adjunta una captura de pantalla de la lista completa de los archivos JAR necesarios:

La mayoría de los archivos JAR se puede obtener como parte de su distribución de puntales. Si tiene un servidor de aplicaciones como glassfish, websphere o jboss instaló entonces usted puede conseguir la mayoría de los archivos jar restantes de la carpeta lib del servidor de aplicaciones. Si no puede descargar los archivos de forma individual:

- Hibernate archivos jar - Hibernate.org
- Struts hibernan plugin - Plugin de Struts hibernan
- JTA Files- archivos JTA
- Archivos dom4j - dom4j
- Archivos SLF4J - SLF4J

- archivos log4j - log4j

Resto de los archivos, debe ser capaz de obtener de su distribución struts2.

Hibernate Clases

Vamos ahora a crear clases Java necesarios para la integración de hibernación. Siguiendo el contenido de **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;
   }
}
```

Esta es una clase POJO que representa el **estudiante** tabla según la especificación de Hibernación. Tiene propiedades de identificación, nombre y apellido, que corresponden a los nombres de columna de la tabla de los estudiantes. Siguiente vamos a crear**StudentDAO.java** archivo de la siguiente manera:

```
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);
   }
}
```

La clase StudentDAO es la capa de acceso a datos para la clase del estudiante. Tiene métodos para mostrar todos los estudiantes y luego de salvar un nuevo expediente del estudiante.

Clase de Acción

Siguiendo archivo **AddStudentAction.java** define nuestra clase de la acción. Tenemos dos métodos de acción aquí - ejecutar () y listStudents (). El método execute () se utiliza para agregar el nuevo expediente académico. Utilizamos el método del dao save () para lograrlo.El otro método, listStudents () se utiliza para listar los estudiantes. Utilizamos el método de lista del dao para obtener la lista de todos los estudiantes.

```java
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;
   }

}
```

Usted se dará cuenta de que estamos implementando la interfaz ModelDriven. Esto se utiliza cuando su clase de la acción se está ocupando de una clase modelo concreto (como estudiante) en lugar de las propiedades individuales (tales como nombre, apellido). La interfaz ModelAware requiere para implementar un método para devolver el modelo. En nuestro caso vamos a regresar el objeto "estudiante".

Crear archivos de vista

Vamos ahora a crear el **student.jsp** archivo de vista con el siguiente contenido:

```
<%@ 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>
```

El student.jsp es bastante sencillo. En la sección superior, tenemos un formulario que se somete a "addStudent.action". Toma en Nombre, Apellido y marcas. Debido a la acción addStudent está ligado a la "AddSudentAction" ModelAware, automáticamente un frijol estudiante se creará con los valores para Nombre, Apellido y marca de auto poblado.

En la sección inferior, vamos a través de la lista de los estudiantes (ver AddStudentAction.java). Nos recorrer la lista y mostrar los valores para el primer nombre, apellido y marcas en una tabla.

Configuración Struts

Pongamos todo junto usando **struts.xml** :

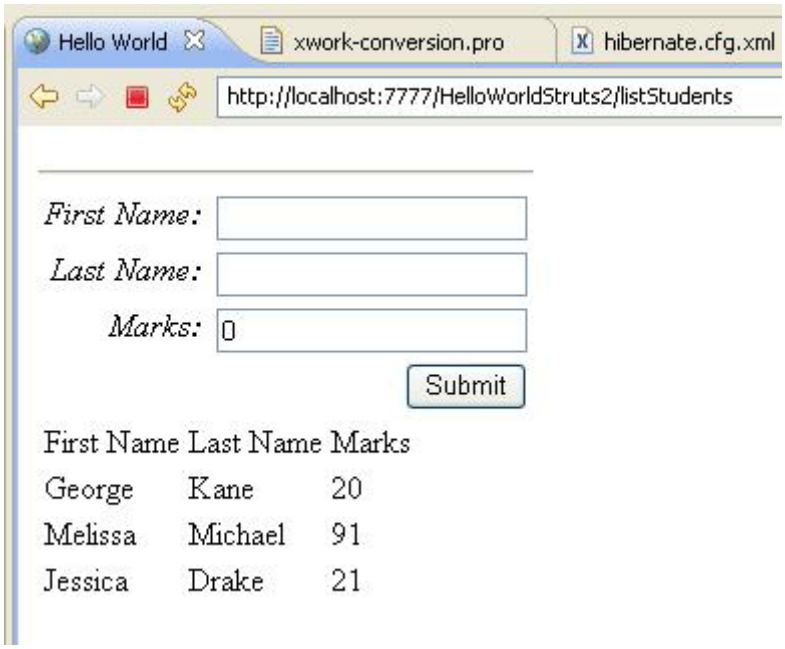
```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE struts PUBLIC
   "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
   "http://struts.apache.org/dtds/struts-2.0.dtd">


<struts>
```

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

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

      <action name="addStudent" method="execute"
         class="com.tutorialspoint.struts2.AddStudentAction">
         <result name="success" type="redirect">
               listStudents
         </result>
      </action>

      <action name="listStudents" method="listStudents"
         class="com.tutorialspoint.struts2.AddStudentAction">
         <result name="success">/students.jsp</result>
      </action>

</package>

</struts>
```

Lo importante a notar aquí es que nuestro paquete "myhibernate" extiende el paquete struts2 defecto llamado "hibernar-default". Entonces Declaramos dos acciones - addStudent y listStudents. addStudent llama al execute () en la clase AddStudentAction y luego a successs, se llama al método listStudents acción.

El método de acción listStudent llama a los listStudents () en la clase AddStudentAction y utiliza el student.jsp como la vista

Ahora haga clic derecho en el nombre del proyecto y haga clic en **Exportar> WAR** archivo para crear un archivo de Guerra. Entonces implementar esta WAR en el directorio webapps del Tomcat. Por último, iniciar el servidor Tomcat y tratar de acceso URL http: // localhost: 8080 / HelloWorldStruts2 / student.jsp. Esto le dará la siguiente pantalla:



En la sección superior, tenemos un formulario para introducir los valores de un nuevo expediente del estudiante y su sección inferior enumera los estudiantes en la base de datos. Seguir adelante y añadir un nuevo expediente del estudiante y pulse enviar. La pantalla se actualizará y le mostrará una lista actualizada cada vez que haga clic en Presentar.