**Spring MVC** in Java is one of the Most Popular Application Framework for building Java Enterprise Applications. Along the side,**Hibernate** is also one of the most popular **ORM Framework**, combining these two frameworks we can build Scalable, robust and industry standard **Enterprise applications** in Java. So today in this article, let’s discuss these two frameworks and build a complete CRUD Application.

This is web based maven project. Please refer the file 'CreateWebBasedMavenProjectInEclipse.docx' in this project to know more about creation of web based maven project.

**Note that this is xml based spring hibernate project. We have web config file web.xml ans spring configuration file spring-servlet.xml. (Location of these files are shown in screen shot of project structure below.)**

**Tools and Technologies Used:**

* Spring 4
* Hibernate 5
* MySQL
* Apache Tomcat 9
* STS i.e. Spring Tool Suite or Eclipse

Let’s first Create a Database for our application,

**NOTE**: We are using **MySQL** Database, you can use any other Database you want.

Execute the following command inside the MySQL,

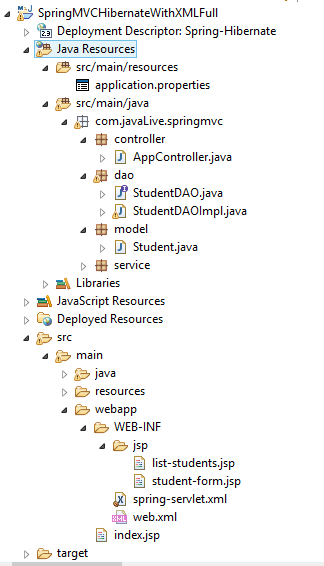
|  |  |
| --- | --- |
| 1 | create database javalive; |

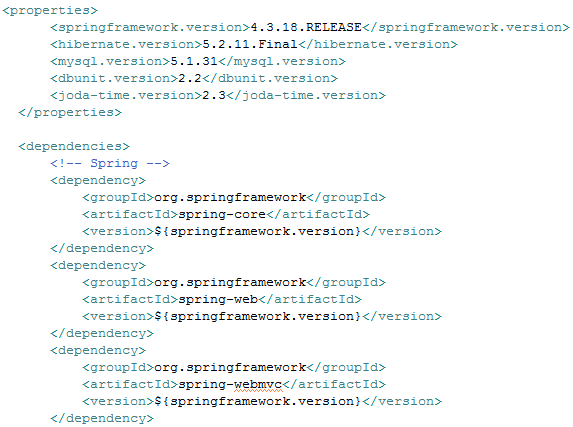
To **select the database**, execute the following command,

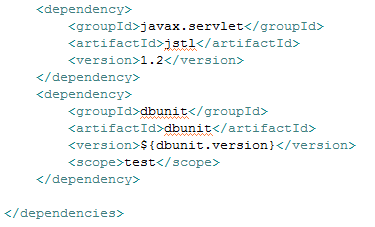
|  |  |
| --- | --- |
| 1 | use javalive; |

Let’s create a table inside the database, execute the following command,

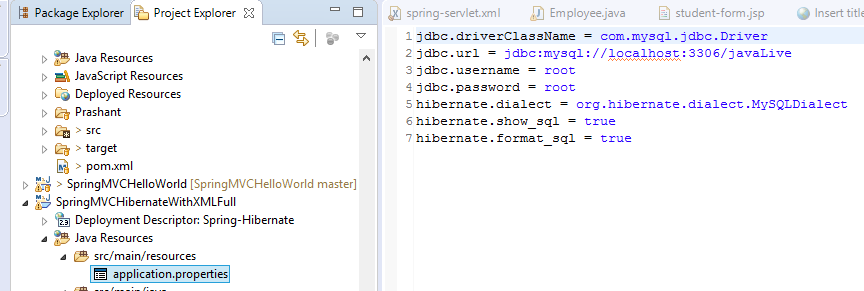
|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | create table student(         id int not null primary key auto\_increment,         name varchar(50) not null,         email varchar(50) not null,         gender varchar(50) not null,         hobbies varchar(50) not null,         city varchar(50) not null,         address varchar(50) not null    ); |

Next step is to create a **maven web based project** in Eclipse, and set up a structure like this,  
  
 





Next step is to create a **property file** inside the **resources** folder,



First, let’s talk about the **web.xml**

**web.xml** is a **Deployment Descriptor** which means, it is the entry point to our application, when we run the application, the web container will search for the web.xml and it will scan the entire file.

**web.xml**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | <?xml version="1.0" encoding="UTF-8"?>  <web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xmlns="http://java.sun.com/xml/ns/javaee" xmlns:web="http://java.sun.com/xml/ns/javaee"  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd"  id="WebApp\_ID" version="2.5">  <welcome-file-list>  <welcome-file>index.jsp</welcome-file>  </welcome-file-list>  <display-name>Spring-Hibernate</display-name>  <servlet>  <servlet-name>spring</servlet-name>  <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>  <load-on-startup>1</load-on-startup>  </servlet>  <servlet-mapping>  <servlet-name>spring</servlet-name>  <url-pattern>/</url-pattern>  </servlet-mapping>  </web-app> |

Next step is to **configure the Spring and Hibernate** inside the **spring-servlet.xml** file

**spring-servlet.xml**

|  |  |
| --- | --- |
|  | <?xml version="1.0" encoding="UTF-8"?>  <beans:beans xmlns="http://www.springframework.org/schema/mvc"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:beans="http://www.springframework.org/schema/beans"  xmlns:context="http://www.springframework.org/schema/context"  xmlns:aop="http://www.springframework.org/schema/aop" xmlns:jee="http://www.springframework.org/schema/jee"  xmlns:lang="http://www.springframework.org/schema/lang" xmlns:p="http://www.springframework.org/schema/p"  xmlns:mvc="http://www.springframework.org/schema/mvc" xmlns:tx="http://www.springframework.org/schema/tx"  xmlns:util="http://www.springframework.org/schema/util"  xsi:schemaLocation="http://www.springframework.org/schema/lang http://www.springframework.org/schema/lang/spring-lang-3.1.xsd  http://www.springframework.org/schema/jee http://www.springframework.org/schema/jee/spring-jee-3.1.xsd  http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop-3.1.xsd  http://www.springframework.org/schema/mvc http://www.springframework.org/schema/mvc/spring-mvc.xsd  http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd  http://www.springframework.org/schema/util http://www.springframework.org/schema/util/spring-util-3.1.xsd  http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-3.1.xsd  http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context.xsd">    <!-- Add support for component scanning -->  <context:component-scan base-package="com.javaLive" />  <context:property-placeholder location="classpath:jdbc.properties" />    <!-- Add support for conversion, formatting and validation support -->  <mvc:annotation-driven />    <!-- Define Spring MVC view resolver -->  <beans:bean id="jspViewResolver"  class="org.springframework.web.servlet.view.InternalResourceViewResolver">  <beans:property name="viewClass"  value="org.springframework.web.servlet.view.JstlView" />  <beans:property name="prefix" value="/WEB-INF/jsps/" />  <beans:property name="suffix" value=".jsp" />  </beans:bean>    <!-- Step 1: Define Database Datasource/connection pool -->  <beans:bean id="dataSource"  class="org.springframework.jdbc.datasource.DriverManagerDataSource">  <beans:property name="driverClassName" value="${jdbc.driverName}" />  <beans:property name="url" value="${jdbc.databaseurl}" />  <beans:property name="username" value="${jdbc.username}" />  <beans:property name="password" value="${jdbc.password}" />  </beans:bean>    <!-- Step 2: Setup Hibernate session factory -->  <beans:bean id="sessionFactory"  class="org.springframework.orm.hibernate5.LocalSessionFactoryBean">  <beans:property name="dataSource" ref="dataSource" />  <beans:property name="packagesToScan" value="com.javaLive.entity" />  <beans:property name="hibernateProperties">  <beans:props>  <beans:prop key="hibernate.dialect">${jdbc.dialect}</beans:prop>  </beans:props>  </beans:property>  </beans:bean>    <!-- Step 3: Setup Hibernate transaction manager -->  <beans:bean id="transactionManager"  class="org.springframework.orm.hibernate5.HibernateTransactionManager">  <beans:property name="sessionFactory" ref="sessionFactory" />  </beans:bean>    <!-- Step 4: Enable configuration of transactional behavior based on annotations -->  <tx:annotation-driven transaction-manager="transactionManager" />    </beans:beans> |

Now, create a **Model** class inside the **entity package,**

We will annotate this class with annotation **@Entity**, It defines that a class can be mapped to a table. And that is it, **it is just a marker,**

**Student.java**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92 | package com.javaLive.entity;    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")  public class Student {    @Id  @GeneratedValue(strategy = GenerationType.IDENTITY)  @Column(name = "id")  private Integer id;    @Column(name = "name")  private String name;    @Column(name = "email")  private String email;    @Column(name = "gender")  private String gender;    @Column(name = "hobbies")  private String hobbies;    @Column(name = "address")  private String address;    @Column(name = "city")  private String city;    public Integer getId() {  return id;  }    public void setId(Integer id) {  this.id = id;  }    public String getName() {  return name;  }    public void setName(String name) {  this.name = name;  }    public String getEmail() {  return email;  }    public void setEmail(String email) {  this.email = email;  }    public String getGender() {  return gender;  }    public void setGender(String gender) {  this.gender = gender;  }    public String getHobbies() {  return hobbies;  }    public void setHobbies(String hobbies) {  this.hobbies = hobbies;  }    public String getAddress() {  return address;  }    public void setAddress(String address) {  this.address = address;  }    public String getCity() {  return city;  }    public void setCity(String city) {  this.city = city;  }  } |

Next step is to create an **interface**, **StudentDAO** inside the **DAO** **package**,

**StudentDAO.java**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | package com.javaLive.dao;    import java.util.List;    import com.javaLive.entity.Student;    public interface StudentDAO {    public void saveStudentObj(Student studentObj);    public List<Student> getAllStudents();    public Student getStudentObj(int theId);    public void removeStudentObj(int theId);  } |

Now let’s create an **implementation** class **StudentDAOImpl**, inside the **DAO package**,

We will annotate this class with **@Repository**, It is **Persistence layer(Data Access Layer)** of application which used to get data from the database. i.e. **all the Database related operations are done by the repository.**  
  
  
  
**@Repository**: You need to use this annotation within the persistence layer, **this acts like database repository.**

**StudentDAOImpl.java**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42 | package com.javaLive.dao;    import java.util.List;    import org.hibernate.Session;  import org.hibernate.SessionFactory;  import org.springframework.beans.factory.annotation.Autowired;  import org.springframework.stereotype.Repository;    import com.javaLive.entity.Student;    @Repository  public class StudentDAOImpl implements StudentDAO {    @Autowired  SessionFactory sessionFactory;    @Override  public void saveStudentObj(Student studentObj) {  sessionFactory.getCurrentSession().saveOrUpdate(studentObj);  }    @Override  public List<Student> getAllStudents() {  Session currentSession = sessionFactory.getCurrentSession();  List<Student> theQuery = currentSession.createQuery("from Student").list();  return theQuery;  }    @Override  public Student getStudentObj(int theId) {  Student studentObj = (Student) sessionFactory.getCurrentSession().get(Student.class, theId);  return studentObj;  }    @Override  public void removeStudentObj(int theId) {  Student studentObj = (Student) sessionFactory.getCurrentSession().get(Student.class, theId);  sessionFactory.getCurrentSession().delete(studentObj);  }    } |

So now call these methods from the service layer, let’s create an **interface, StudentService** inside the **service package**

**StudentService.java**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | package com.javaLive.service;    import java.util.List;    import com.javaLive.entity.Student;    public interface StudentService {    public void saveStudentObj(Student studentObj);    public List<Student> getAllStudents();    public Student getStudentObj(int theId);    public void removeStudentObj(int theId);    } |

So now let’s create an **implementation** class, **StudentServiceImpl,** inside the **service package,**

We will annotate this class with **@Service,**All **business logic** is here i.e. Data related calculations and all. This annotation of business layer in which our user not directly call the persistence method so it will call this method using this annotation. **It will request @Repository as per user request**

***What’s special about @Service?***

Apart from the fact that it is used to indicate that it’s holding the business logic, there’s no noticeable specialty that this annotation provides, but who knows, spring may add some additional exceptional in future.  
**StudentServiceImpl.java**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42 | package com.javaLive.service;    import java.util.List;    import org.springframework.beans.factory.annotation.Autowired;  import org.springframework.stereotype.Service;  import org.springframework.transaction.annotation.Transactional;    import com.javaLive.dao.StudentDAO;  import com.javaLive.entity.Student;    @Service  public class StudentServiceImpl implements StudentService {    @Autowired  StudentDAO studentDAO;    @Override  @Transactional  public void saveStudentObj(Student studentObj) {  studentDAO.saveStudentObj(studentObj);  }    @Override  @Transactional  public List<Student> getAllStudents() {  return studentDAO.getAllStudents();  }    @Override  @Transactional  public Student getStudentObj(int theId) {  return studentDAO.getStudentObj(theId);  }    @Override  @Transactional  public void removeStudentObj(int theId) {  studentDAO.removeStudentObj(theId);  }    } |

Finally, let’s create a controller **StudentController**, inside the**controller package**,

We will annotate this class with **@Controller:** where your **request** **mapping from presentation page** done i.e. Presentation layer won’t go to any other file it goes directly to **@Controller** class and checks for requested path in **@RequestMapping** annotation which written before method calls if necessary.

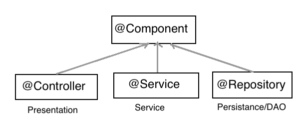
***What’s special about @Controller?***

We cannot switch this annotation with any other like **@Service** or **@Repository**, even though they look same. The dispatcher scans the classes annotated with **@Controller** and detects **@RequestMapping** annotations within them. We can only use **@RequestMapping** on **@Controller**annotated classes.

**StudentController.java**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49 | package com.javaLive.controller;    import org.springframework.beans.factory.annotation.Autowired;  import org.springframework.stereotype.Controller;  import org.springframework.ui.Model;  import org.springframework.web.bind.annotation.ModelAttribute;  import org.springframework.web.bind.annotation.RequestMapping;  import org.springframework.web.bind.annotation.RequestParam;    import com.javaLive.entity.Student;  import com.javaLive.service.StudentService;    @Controller  @RequestMapping(value = "/student")  public class StudentController {    @Autowired  StudentService studentService;    @RequestMapping("/showFormForAdd")  public String showAddForm(Model model) {  model.addAttribute("student", new Student());  return "student-form";  }    @RequestMapping("/saveProcess")  public String saveStudentObj(@ModelAttribute("student") Student studentObj) {  studentService.saveStudentObj(studentObj);  return "redirect:/student/list";  }    @RequestMapping("/list")  public String listOfStudents(Model model) {  model.addAttribute("studentsList", studentService.getAllStudents());  return "list-students";  }    @RequestMapping("/displayUpdateForm")  public String showUpdateForm(@RequestParam("studentId") int theId, Model model) {  model.addAttribute("student", studentService.getStudentObj(theId));  return "student-form";  }    @RequestMapping("/displayDeleteForm")  public String deleteStudentObj(@RequestParam("studentId") int theId, Model model) {  studentService.removeStudentObj(theId);  return "redirect:/student/list";  }  } |

**Purpose of each annotation:**

[](https://bushansirgur.in/wp-content/uploads/2018/06/FFIdh.png)

1) **@Controller** : Classes annotated with this, are intended to receive a request from the client side. The first request comes to the Dispatcher Servlet, from where it passes the request to the particular controller using the value of @RequestMapping annotation.

2) **@Service** : Classes annotated with this, are intended to manipulate data, that we receive from a client or fetch from the database. All the manipulation with data should be done in this layer.

3) **@Repository** : Classes annotated with this, are intended to connect with database. It can also be considered as DAO(Data Access Object) layer. This layer should be restricted to CRUD (create, retrieve, update, delete) operations only. If any manipulation is required, data should be sent be send back to @Service layer.

If we interchange their place(use @Repository in place of @Controller), our application will work fine.

**The main purpose of using three different @annotations is to provide better Modularity to the Enterprise application.**

So now, let’s create a folder called **jsps**inside the**WEB-INF** to keep all the**JSP** files,

inside the jsps, create **list-students.jsp,** this will be our landing page for our application, here we will display the student records

**list-students.jsp**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54 | <%@ page language="java" contentType="text/html; charset=ISO-8859-1"  pageEncoding="ISO-8859-1"%>  <%@ taglib uri="http://www.springframework.org/tags/form" prefix="form"%>  <%@ taglib uri="http://www.springframework.org/tags" prefix="spring"%>  <%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>  <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">  <html>  <head>  <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">  <title>Insert title here</title>  </head>  <body>  <h1 align="center">List Students</h1>  <button  onclick="window.location.href = 'showFormForAdd'; return false;">Add  Student</button>    <hr />    <center>  <table border="1">  <tr>  <th>Name</th>  <th>Email</th>  <th>Gender</th>  <th>Hobbies</th>  <th>City</th>  <th>Address</th>  <th>Action</th>  </tr>  <c:forEach items="${studentsList}" var="s">  <c:url var="updateLink" value="/student/displayUpdateForm.html">  <c:param name="studentId" value="${s.id}" />  </c:url>    <c:url var="deleteLink" value="/student/displayDeleteForm.html">  <c:param name="studentId" value="${s.id}" />  </c:url>  <tr>  <td>${s.name}</td>  <td>${s.email}</td>  <td>${s.gender}</td>  <td>${s.hobbies}</td>  <td>${s.city}</td>  <td>${s.address}</td>  <td><a href="${updateLink}">Update</a> | <a  href="${deleteLink}"  onclick="if(!(confirm('Are you sure want to delete this Student permanently?'))) return false">Delete</a></td>  </tr>  </c:forEach>  </table>  </center>  </body>  </html> |

Next step is to create **student-form.jsp** inside the jsps folder to enter the student details,

**student-form.jsp**

|  |  |
| --- | --- |
|  | <%@ page language="java" contentType="text/html; charset=ISO-8859-1"  pageEncoding="ISO-8859-1"%>  <%@ taglib uri="http://www.springframework.org/tags/form" prefix="form"%>  <%@ taglib uri="http://www.springframework.org/tags" prefix="spring"%>  <%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>  <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">  <html>  <head>  <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">  <title>Add Student</title>  </head>  <body>  <h1>Provide Student Information</h1>    <hr />    <form:form action="saveProcess.html" modelAttribute="student">  <table>  <form:hidden path="id" />  <form:hidden path="" value="${student.hobbies}" id="hobbies" />  <tr>  <td><b>Full Name:</b></td>  <td><form:input path="name" /></td>  </tr>  <tr>  <td><b>Email:</b></td>  <td><form:input path="email" /></td>  </tr>  <tr>  <td><b>Gender:</b></td>  <td><form:radiobutton path="gender" value="Male" />Male&nbsp;&nbsp;<form:radiobutton  path="gender" value="Female" />Female</td>  </tr>  <tr>  <td><b>Hobbies:</b></td>  <td><form:checkbox path="hobbies" value="Cricket" />Cricket <form:checkbox  path="hobbies" value="Football" />Football <form:checkbox  path="hobbies" value="Volleyball" />Volleyball <form:checkbox  path="hobbies" value="Hockey" />Hockey</td>  </tr>  <tr>  <td><b>City:</b></td>  <td><form:select path="city">  <form:option value="0">--Select--</form:option>  <form:option value="Banglore">Banglore</form:option>  <form:option value="Hubli">Hubli</form:option>  <form:option value="Davangere">Davangere</form:option>  <form:option value="Mysore">Mysore</form:option>  <form:option value="Belgaum">Belgaum</form:option>  <form:option value="Tumkur">Tumkur</form:option>  <form:option value="Chitradurga">Chitradurga</form:option>  </form:select></td>  </tr>  <tr>  <td><b>Address:</b></td>  <td><form:textarea path="address" /></td>  </tr>  <tr>  <td></td>  <td><input type="submit" value="Save" /> <input type="reset"  value="Reset" /></td>  </tr>  </table>  </form:form>    <hr />    <a href="list">Back to List</a>  <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.1.1/jquery.min.js"></script>  <script type="text/javascript">  $(document).ready(function() {  var hobbies = $("#hobbies").val().split(",");  var $checkboxes = $("input[type=checkbox]");  $checkboxes.each(function(idx, element) {  if (hobbies.indexOf(element.value) != -1) {  element.setAttribute("checked", "checked");  $("#hobbies").val("");  } else {  element.removeAttribute("checked");  }  });  });  </script>  </body>  </html> |

Finally, let’s create a **welcome file** inside the **WebContent folder**. Here we will just forward the request a specific controller

**index.jsp**

|  |  |
| --- | --- |
| 1 | <% response.sendRedirect("student/list"); %> |