**SpringMVCDemo测试代码**

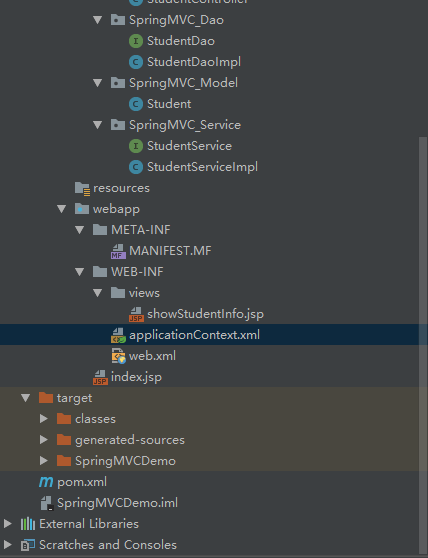
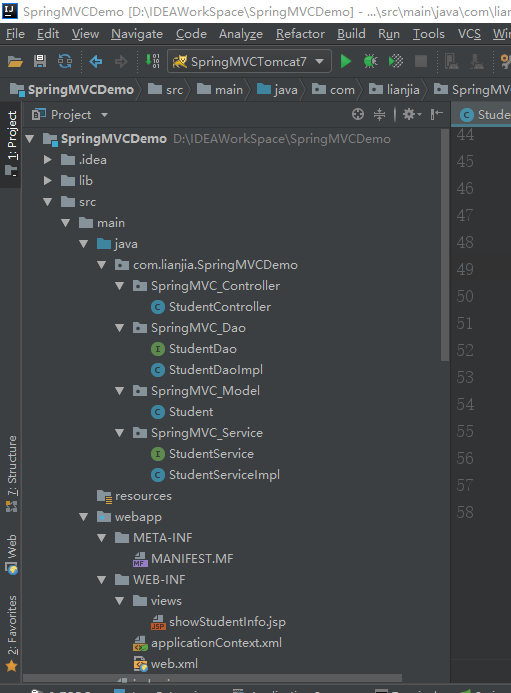
一，环境搭载

       WIN10，idea,  java 8 , maven 3.5 , tomcat 7,

       环境搭载并测试成功！

二，创建maven项目，SpringMVCDemo

       创建如下目录：



三，每个文件的代码；

pom.xml文件配置：

<?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>com.lianjia.SpringMVCDemo</groupId>  
 <artifactId>SpringMVCDemo</artifactId>  
 <version>1.0-SNAPSHOT</version>  
 <packaging>war</packaging>  
  
 <name>SpringMVCDemo Maven Webapp</name>  
 <!-- FIXME change it to the project's website -->  
 <url>http://www.example.com</url>  
  
 <properties>  
 <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  
 <maven.compiler.source>1.7</maven.compiler.source>  
 <maven.compiler.target>1.7</maven.compiler.target>  
 <!--Spring版本号-->  
 <spring.version>3.2.9.RELEASE</spring.version>  
 </properties>  
  
 <dependencies>  
 <!--单元测试依赖-->  
 <dependency>  
 <groupId>junit</groupId>  
 <artifactId>junit</artifactId>  
 <version>4.11</version>  
 <scope>test</scope>  
 </dependency>  
<!--spring单元测试依赖-->  
 <dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-test</artifactId>  
 <version>${spring.version}</version>  
 <scope>test</scope>  
 </dependency>  
<!--springMVC核心包-->  
 <dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-webmvc</artifactId>  
 <version>${spring.version}</version>  
 </dependency>  
<!--spring核心包-->  
 <dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-core</artifactId>  
 <version>${spring.version}</version>  
 </dependency>  
  
 <dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-beans</artifactId>  
 <version>${spring.version}</version>  
 </dependency>  
  
 <dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-context</artifactId>  
 <version>${spring.version}</version>  
 </dependency>  
  
 <dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-context-support</artifactId>  
 <version>${spring.version}</version>  
 </dependency>  
  
 <dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-aop</artifactId>  
 <version>${spring.version}</version>  
 </dependency>  
  
 <dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-aspects</artifactId>  
 <version>${spring.version}</version>  
 </dependency>  
  
 <dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-tx</artifactId>  
 <version>${spring.version}</version>  
 </dependency>  
  
 <dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-web</artifactId>  
 <version>${spring.version}</version>  
 </dependency>  
  
 <dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-jdbc</artifactId>  
 <version>${spring.version}</version>  
 </dependency>  
<!--AOP begin-->  
 <dependency>  
 <groupId>org.aspectj</groupId>  
 <artifactId>aspectjrt</artifactId>  
 <version>1.8.13</version>  
 </dependency>  
  
 <dependency>  
 <groupId>org.aspectj</groupId>  
 <artifactId>aspectjrt</artifactId>  
 <version>1.8.13</version>  
 </dependency>  
  
 <dependency>  
 <groupId>cglib</groupId>  
 <artifactId>cglib</artifactId>  
 <version>3.2.5</version>  
 </dependency>  
 <!--AOP end-->  
  
  
 <!--json依赖-->  
  
 <dependency>  
 <groupId>org.codehaus.jackson</groupId>  
 <artifactId>jackson-core-asl</artifactId>  
 <version>1.5.2</version>  
 </dependency>  
 <dependency>  
 <groupId>org.codehaus.jackson</groupId>  
 <artifactId>jackson-mapper-asl</artifactId>  
 <version>1.5.2</version>  
 </dependency>  
  
  
 </dependencies>  
  
 <build>  
 <finalName>SpringMVCDemo</finalName>  
 <pluginManagement><!-- lock down plugins versions to avoid using Maven defaults (may be moved to parent pom) -->  
 <plugins>  
 <plugin>  
 <artifactId>maven-clean-plugin</artifactId>  
 <version>3.0.0</version>  
 </plugin>  
 <!-- see http://maven.apache.org/ref/current/maven-core/default-bindings.html#Plugin\_bindings\_for\_war\_packaging -->  
 <plugin>  
 <artifactId>maven-resources-plugin</artifactId>  
 <version>3.0.2</version>  
 </plugin>  
 <plugin>  
 <artifactId>maven-compiler-plugin</artifactId>  
 <version>3.7.0</version>  
 </plugin>  
 <plugin>  
 <artifactId>maven-surefire-plugin</artifactId>  
 <version>2.20.1</version>  
 </plugin>  
 <plugin>  
 <artifactId>maven-war-plugin</artifactId>  
 <version>3.2.0</version>  
 </plugin>  
 <plugin>  
 <artifactId>maven-install-plugin</artifactId>  
 <version>2.5.2</version>  
 </plugin>  
 <plugin>  
 <artifactId>maven-deploy-plugin</artifactId>  
 <version>2.8.2</version>  
 </plugin>  
 </plugins>  
 </pluginManagement>  
 </build>  
</project>

web.xml文件内容：

<?xml version="1.0" encoding="UTF-8"?>  
<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/web-app\_4\_0.xsd"  
 version="4.0">  
 <display-name>springmvcdemo</display-name>  
 <servlet>  
 <servlet-name>spring</servlet-name>  
 <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>  
 <init-param>  
 <param-name>contextConfigLocation</param-name>  
 <param-value>/WEB-INF/applicationContext.xml</param-value>  
 </init-param>  
 <load-on-startup>1</load-on-startup>  
 </servlet>  
 <servlet-mapping>  
 <servlet-name>spring</servlet-name>  
 <url-pattern>/</url-pattern>  
 </servlet-mapping>  
  
</web-app>

applicationContext.xml文件内容：

<?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:mvc="http://www.springframework.org/schema/mvc"  
 xsi:schemaLocation="http://www.springframework.org/schema/beans  
 http://www.springframework.org/schema/beans/spring-beans-3.0.xsd  
 http://www.springframework.org/schema/context  
 http://www.springframework.org/schema/context/spring-context-3.0.xsd http://www.springframework.org/schema/mvc http://www.springframework.org/schema/mvc/spring-mvc.xsd">  
 <!--扫描所有的 spring包下的文件；-->  
 <!--当然需要在spring配置文件里面配置一下自动扫描范围 <context:component-scan base-package="\*"/>  
 \*代表你想要扫描的那些包的目录所在位置。Spring 在容器初始化时将自动扫描 base-package 指定的包及其子包下的所有的.class文件，  
 所有标注了 @Repository 的类都将被注册为 Spring Bean。-->  
 <context:component-scan base-package="com.lianjia.SpringMVCDemo" />  
 <!--新增加的两个配置，这个是解决406问题的关键--><!-- mvc注解驱动(可代替注解适配器与注解映射器的配置)，默认加载很多参数绑定方法（实际开发时使用） -->    
 <context:annotation-config />  
 <mvc:annotation-driven />  
 <!--end-->  
  
 <bean class="org.springframework.web.servlet.mvc.annotation.DefaultAnnotationHandlerMapping" />  
 <bean class="org.springframework.web.servlet.mvc.annotation.AnnotationMethodHandlerAdapter" />  
  
 <bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">  
 <property name="prefix" value="/WEB-INF/views/" />  
 <property name="suffix" value=".jsp" />  
 </bean>  
 <!--自己后加的，该BeanPostProcessor将自动对标注@Autowired的bean进行注入-->  
 <bean class="org.springframework.beans.factory.annotation.AutowiredAnnotationBeanPostProcessor"></bean>  
  
 <bean id="restTemplate" class="org.springframework.web.client.RestTemplate">  
 <property name="messageConverters">  
 <list>  
 <!--<ref bean="stringHttpMessageConverter"/>-->  
 <bean class="org.springframework.http.converter.ByteArrayHttpMessageConverter"/>  
 <bean class="org.springframework.http.converter.ResourceHttpMessageConverter"/>  
 <bean class="org.springframework.http.converter.xml.SourceHttpMessageConverter"/>  
 <bean class="org.springframework.http.converter.support.AllEncompassingFormHttpMessageConverter"/>  
 <bean class="org.springframework.http.converter.json.MappingJackson2HttpMessageConverter"/>  
 </list>  
 </property>  
 </bean>  
  
</beans>

<!-- mvc注解驱动(可代替注解适配器与注解映射器的配置)，默认加载很多参数绑定方法（实际开发时使用） -->

***Model层***

sutdent.java代码内容：

package com.lianjia.SpringMVCDemo.SpringMVC\_Model;  
  
import java.io.Serializable;  
  
/\*\*  
 \* @Aauthor:  
 \* @Date:2018/5/14 17:47  
 \* @Description:mvc中的m-model，描述数据结构模型；  
 \*/  
public class Student implements Serializable {  
 private static final long serialVersionUID = 1L;  
  
 private Integer id;  
 private String name;  
 /\*设置Student类的成员变量，并且设置get和set方法；\*/  
 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;  
 }  
 /\*这里一定要重写toString方法，返回数据；\*/  
 public String toString(){  
 return "Student [id = " + id +  
 ", name = " + name + "]";  
 }  
}

***Dao层***

StudentDao.java文件内容;

package com.lianjia.SpringMVCDemo.SpringMVC\_Dao;  
  
import com.lianjia.SpringMVCDemo.SpringMVC\_Model.Student;  
import java.util.List;  
  
/\*\*  
 \* @Aauthor:  
 \* @Date:2018/5/14 16:07  
 \* @Description:用户dao接口；  
 \*/  
  
public interface StudentDao {  
 /\*提供一个存放Student类的方法接口，用list模拟数据库存放；\*/  
 public List<Student> getAllStudents();  
}

StudentDaoImpl.java文件内容：

package com.lianjia.SpringMVCDemo.SpringMVC\_Dao;  
  
import com.lianjia.SpringMVCDemo.SpringMVC\_Model.Student;  
import org.springframework.stereotype.Repository;  
  
import java.util.ArrayList;  
import java.util.List;  
  
/\*\*  
\* @Aauthor:  
 \* @Date:2018/5/14 16:07  
 \* @Description:实现StudentDao接口；

 \*/  
 /\*Repository注解就是将数据访问层（DAO层）的类标识为Spring bean，只需将该注解标注在DAO类上即可；\*/  
 @Repository  
 public class StudentDaoImpl implements StudentDao{  
/\*用List模拟数据库存储数据类；\*/  
 @Override  
 public List<Student> getAllStudents(){  
List<Student> students = new ArrayList<Student>();  
  
 Student stu1 = new Student();  
 stu1.setId(1);  
 stu1.setName("zhang san");  
 students.add(stu1);  
  
  
 Student stu2 = new Student();  
 stu2.setId(1);  
 stu2.setName("li si");  
 students.add(stu2);  
return students;  
 }  
}  
***Service层***

StudentService.java文件内容：

package com.lianjia.SpringMVCDemo.SpringMVC\_Service;  
  
import com.lianjia.SpringMVCDemo.SpringMVC\_Model.Student;  
  
import java.util.List;  
  
/\*\*  
 \* @Aauthor:  
 \* @Date:2018/5/14 16:10  
 \* @Description: 提供StudentService接口；  
 \*/  
public interface StudentService {  
 public List<Student> getAllStudents();  
}

StudentServiceImpl.java文件内容：

package com.lianjia.SpringMVCDemo.SpringMVC\_Service;  
  
import com.lianjia.SpringMVCDemo.SpringMVC\_Dao.StudentDao;  
import com.lianjia.SpringMVCDemo.SpringMVC\_Model.Student;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Service;  
  
import java.util.List;  
  
/\*\*  
 \* @Aauthor:  
 \* @Date:2018/5/14 16:10  
 \* @Description:实现了StudentService接口；  
 \*/  
/\*Service注解，创建了StudentService的bean；  
这样在StudentController中使用StudentService的时候，就可以直接把StudentService注入到Controller层；  
进行调用的时候，会声明一个service对象，并且在其上面用@Resource进行注解；  
\* \*/  
@Service  
public class StudentServiceImpl implements StudentService {  
 @Autowired //自动装载；  
 StudentDao dao;  
  
 @Override  
 public List<Student> getAllStudents() {  
 return dao.getAllStudents();  
 }  
  
}

***Controller层：***

StudentController.java

package com.lianjia.SpringMVCDemo.SpringMVC\_Controller;  
  
import com.lianjia.SpringMVCDemo.SpringMVC\_Model.Student;  
import com.lianjia.SpringMVCDemo.SpringMVC\_Service.StudentService;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Controller;  
import org.springframework.ui.Model;  
import org.springframework.web.bind.annotation.RequestMapping;  
import org.springframework.web.bind.annotation.RequestMethod;  
import org.springframework.web.bind.annotation.ResponseBody;  
  
import java.util.List;  
  
/\*\*  
 \* @Aauthor:  
 \* @Date:2018/5/14 16:03  
 \* @Description:控制层，action模块；  
 \*/  
/\*Controller注解Action，在springMVC开发中直接注解Controller  
\* Spring会自动创建一个StudentController类的对象\*/  
  
@Controller  
@RequestMapping("/studentModule")  
//http://localhost:8080/studentModule/getStudentInfo  
//访问的地址；  
/\*RequestMapping是一个用来处理请求地址映射的注解，可用于类或方法上。  
用于类上，表示类中的所有响应请求的方法都是以该地址作为父路径。  
默认的是value的值\*/  
  
public class StudentController {  
 @Autowired //自动注入，自动装配；  
  
 StudentService managerService;  
  
 @RequestMapping(value = "/getStudentInfo", method = RequestMethod.GET)  
 /\*public String getStudentInfo(Model model){  
 List<Student> allStudents = managerService.getAllStudents();//用List模拟数据库存储；  
 model.addAttribute("students", allStudents);  
 return "showStudentInfo";//这里是字符串的话，直接跳转到showStudentInfo.jsp页面了；  
 }\*/  
  
 /\*@Responsebody 注解表示该方法的返回的结果直接写入 HTTP 响应正文（ResponseBody）中，  
 一般在异步获取数据时使用，通常是在使用 @RequestMapping 后，返回值通常解析为跳转路径，  
 加上 @Responsebody 后返回结果不会被解析为跳转路径，而是直接写入HTTP 响应正文中。  
 作用： 该注解用于将Controller的方法返回的对象，通过适当的HttpMessageConverter转换为  
 指定格式后，写入到Response对象的body数据区。  
 使用时机： 返回的数据不是html标签的页面，而是其他某种格式的数据时（如json、xml等）使用；\*/  
  
 @ResponseBody  
 public List<Student> getStudentInfo(){  
 List<Student> allStudents = managerService.getAllStudents();  
 System.out.println("result返回的是："+allStudents);  
 return allStudents;  
 }  
 /\*直接将获取到的数据返回；\*/  
}

四，配置tomcat7，之后启动项目。

jdk版本一定要对~

在弹出的浏览器地址栏中输入<http://localhost:8080/studentModule/getStudentInfo>回车访问；

即可出现如下界面；



注意：若StudentController.java中直接使用

showStudentInfo渲染页面的时候，

showStudentInfo.jsp文件内容是：

<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>  
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt"%>  
<html>  
<head>  
 <title>Spring MVC Demo</title>  
</head>  
   
<body>  
 <h2>All Students</h2>  
   
 <table border="1">  
 <tr>  
 <th>Student Id</th>  
 <th>Student Name</th>  
 </tr>  
 <c:forEach items="${students}" var="student">  
 <tr>  
 <td>${student.id}</td>  
 <td>${student.name}</td>  
 </tr>  
 </c:forEach>  
 </table>  
   
</body>  
</html>

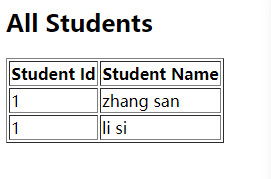
注意这里需要下载两个jar包，放到tomcat的lib文件夹下：

jstl-1.2.jar

standard.jar

然后同样输入<http://localhost:8080/studentModule/getStudentInfo>回车访问；

即可出现如下界面：



则代码测试成功！！！