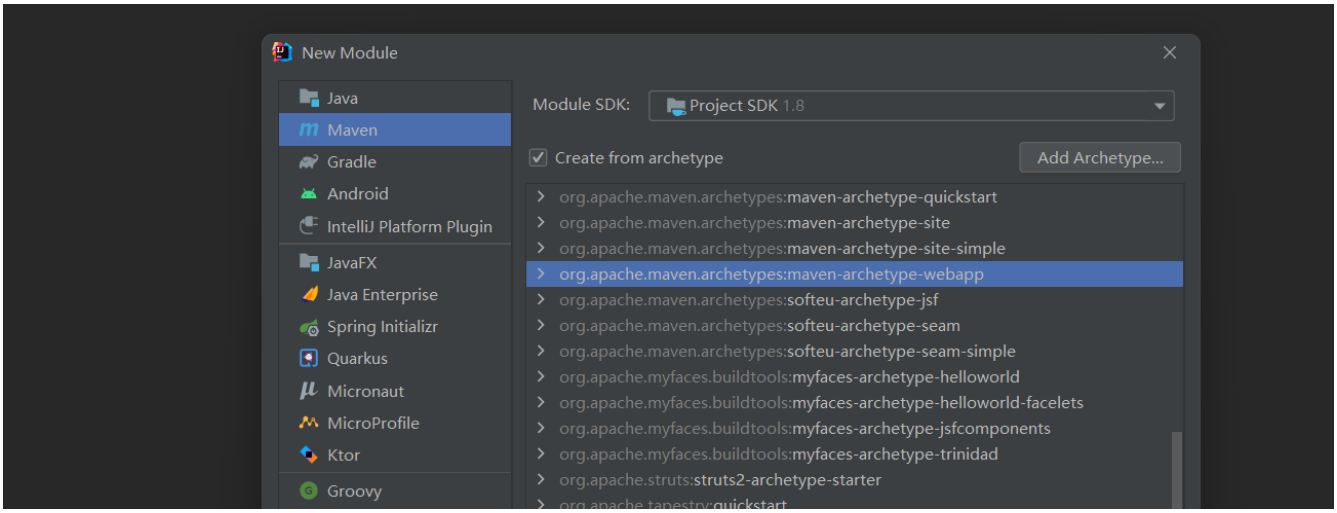
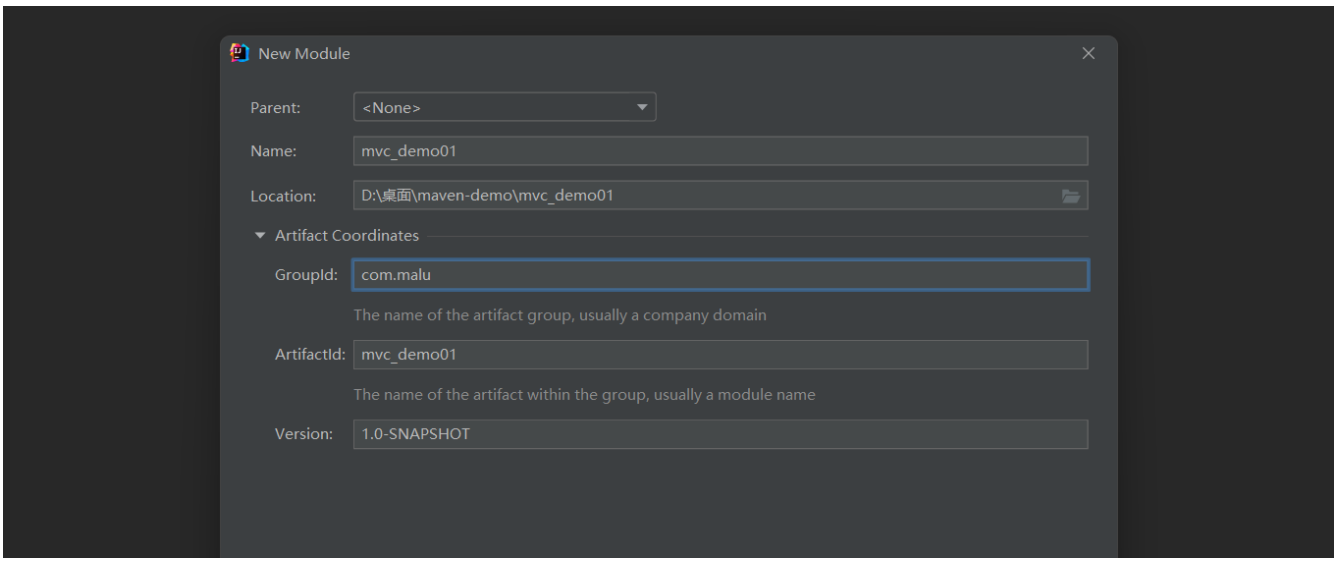
1, SpringMVC简介

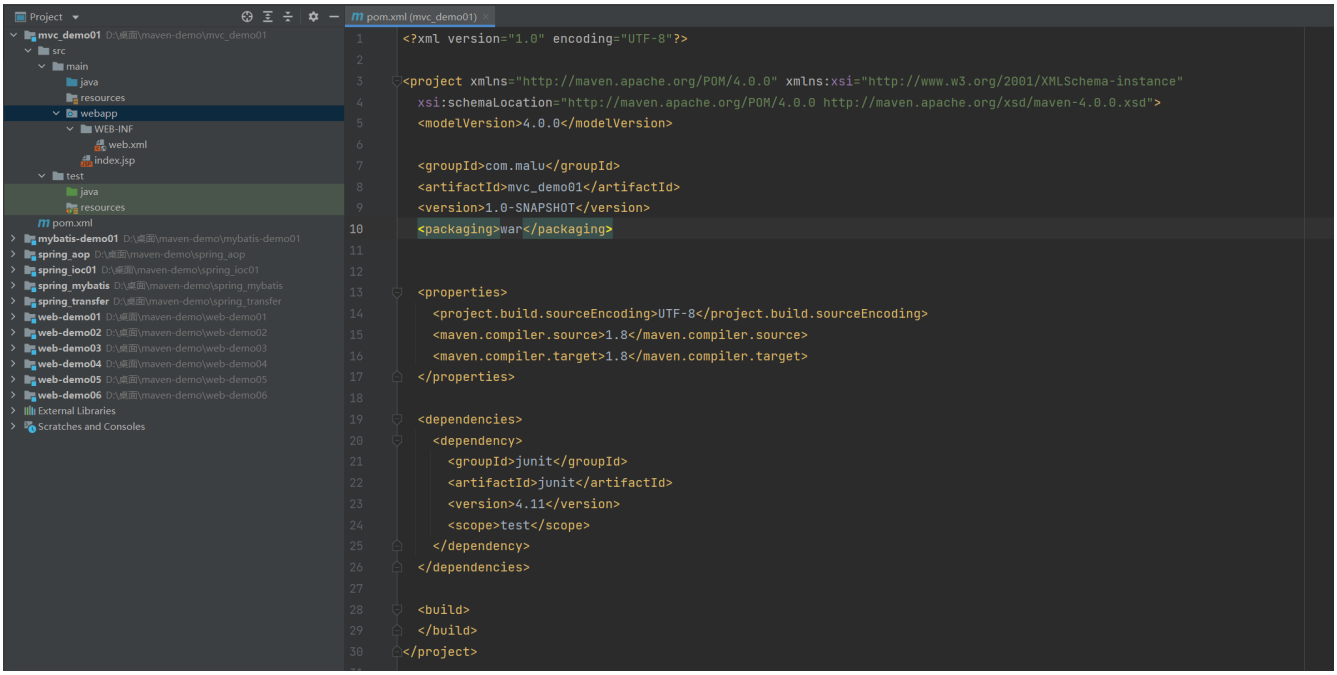
SpringMVC是一个基于MVC模式的轻量级Web框架，是Spring框架的一个模块，和Spring可以直接整合使用。SpringMVC代替了Servlet技术，它通过一套注解，让一个简单的Java类成为处理请求的控制器，而无须实现任何接口。

2, 入门案例

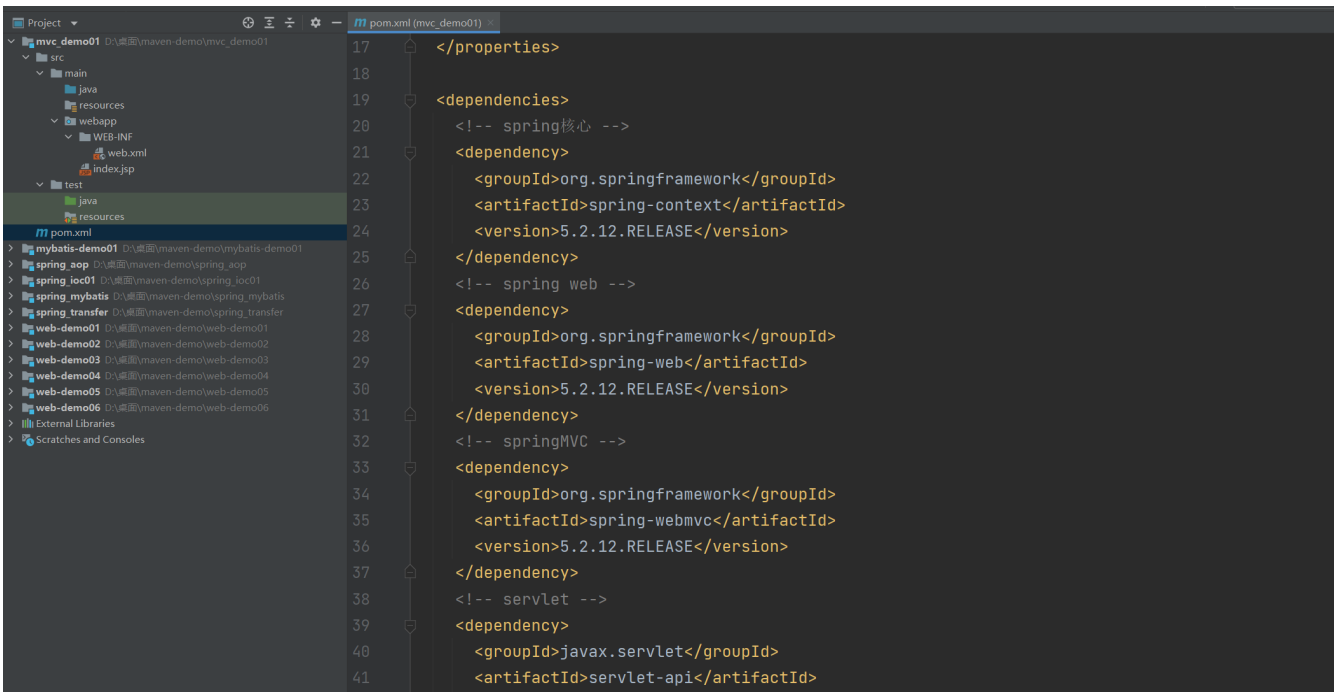
使用maven创建web项目，补齐包结构：







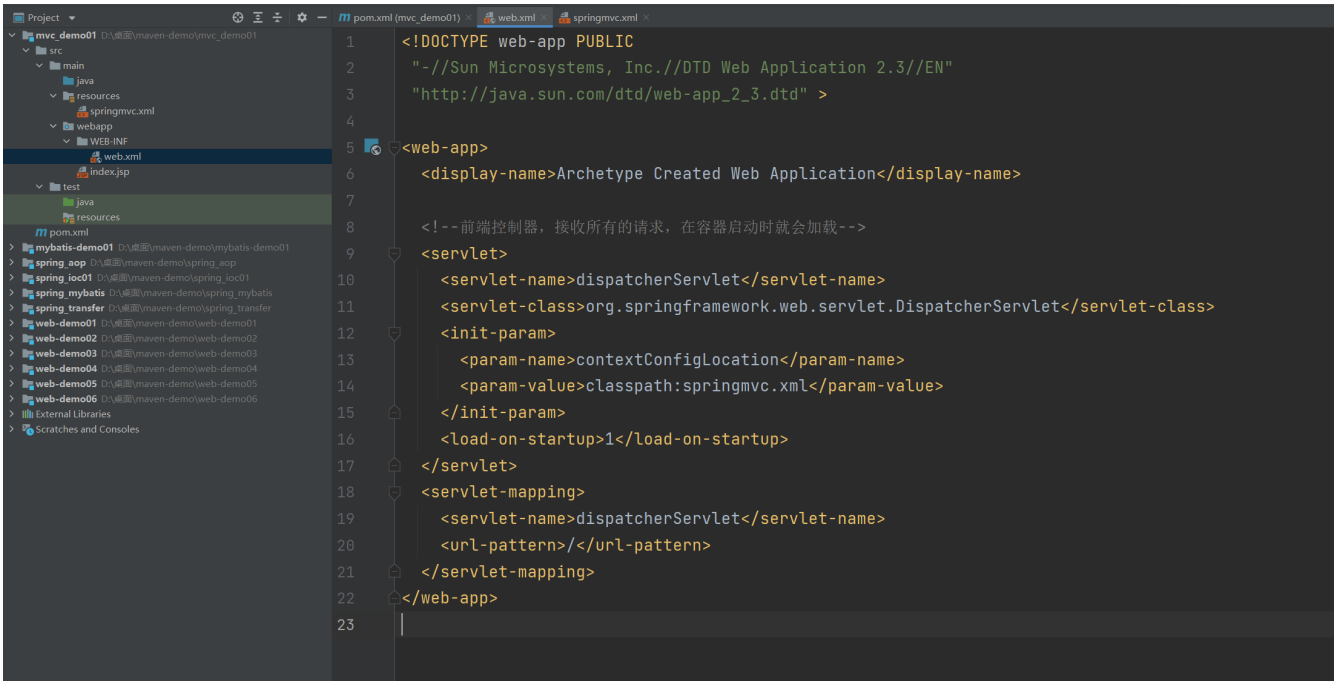
引入相关依赖和tomcat插件：




```
<dependencies>
    <!-- spring核心 -->
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-context</artifactId>
      <version>5.2.12.RELEASE</version>
    </dependency>
    <!-- spring web -->
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-web</artifactId>
```

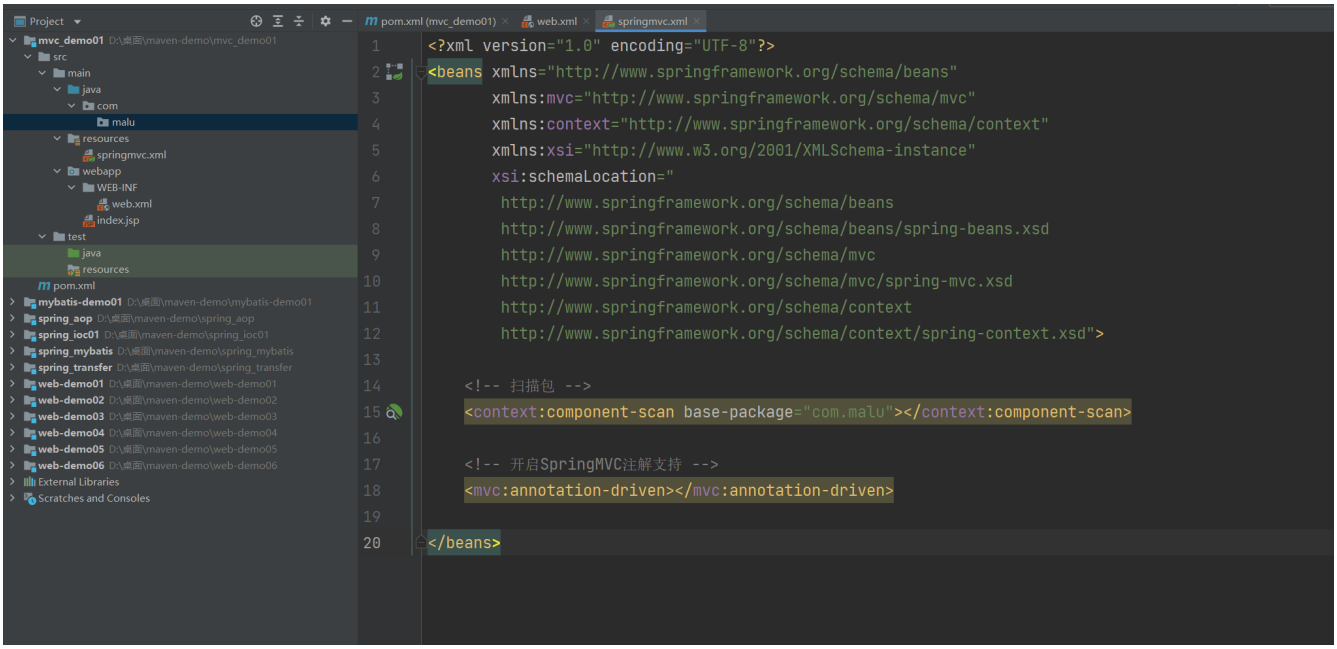
```
      <version>5.2.12.RELEASE</version>
    </dependency>
    <!-- springMVC -->
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-webmvc</artifactId>
      <version>5.2.12.RELEASE</version>
    </dependency>
    <!-- servlet -->
    <dependency>
      <groupId>javax.servlet</groupId>
      <artifactId>servlet-api</artifactId>
      <version>2.5</version>
      <scope>provided</scope>
    </dependency>
    <!-- jsp -->
    <dependency>
      <groupId>javax.servlet.jsp</groupId>
      <artifactId>jsp-api</artifactId>
      <version>2.0</version>
      <scope>provided</scope>
    </dependency>
  </dependencies>

  <build>
    <plugins>
      <!-- tomcat插件 -->
      <plugin>
        <groupId>org.apache.tomcat.maven</groupId>
        <artifactId>tomcat7-maven-plugin</artifactId>
        <version>2.1</version>
        <configuration>
          <port>8080</port>
          <path>/</path>
          <uriEncoding>UTF-8</uriEncoding>
          <server>tomcat7</server>
          <systemProperties>
            <java.util.logging.SimpleFormatter.format>%1$tH:%1$tM:%1$tS %2$s%n%4$s:
%5$s%6$s%n
            </java.util.logging.SimpleFormatter.format>
          </systemProperties>
        </configuration>
      </plugin>
    </plugins>
  </build>
```

在web.xml中配置前端控制器DispatcherServlet



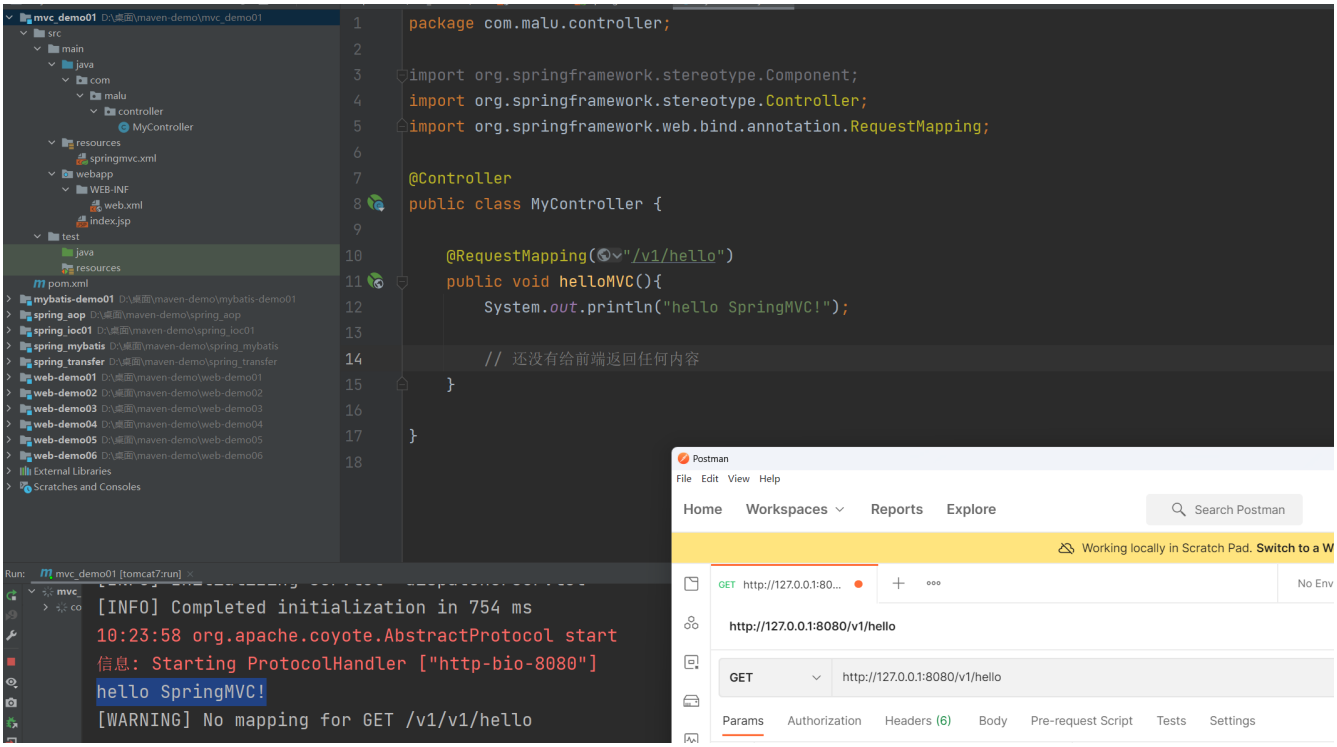

```
<!DOCTYPE web-app PUBLIC
 "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
 "http://java.sun.com/dtd/web-app_2_3.dtd" >

<web-app>
  <display-name>Archetype Created Web Application</display-name>

  <!--前端控制器，接收所有的请求，在容器启动时就会加载-->
  <servlet>
    <servlet-name>dispatcherServlet</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
    <init-param>
      <param-name>contextConfigLocation</param-name>
      <param-value>classpath:springmvc.xml</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>dispatcherServlet</servlet-name>
    <url-pattern>/</url-pattern>
  </servlet-mapping>
</web-app>
```

编写SpringMVC核心配置文件springmvc.xml，该文件和Spring配置文件写法一样。



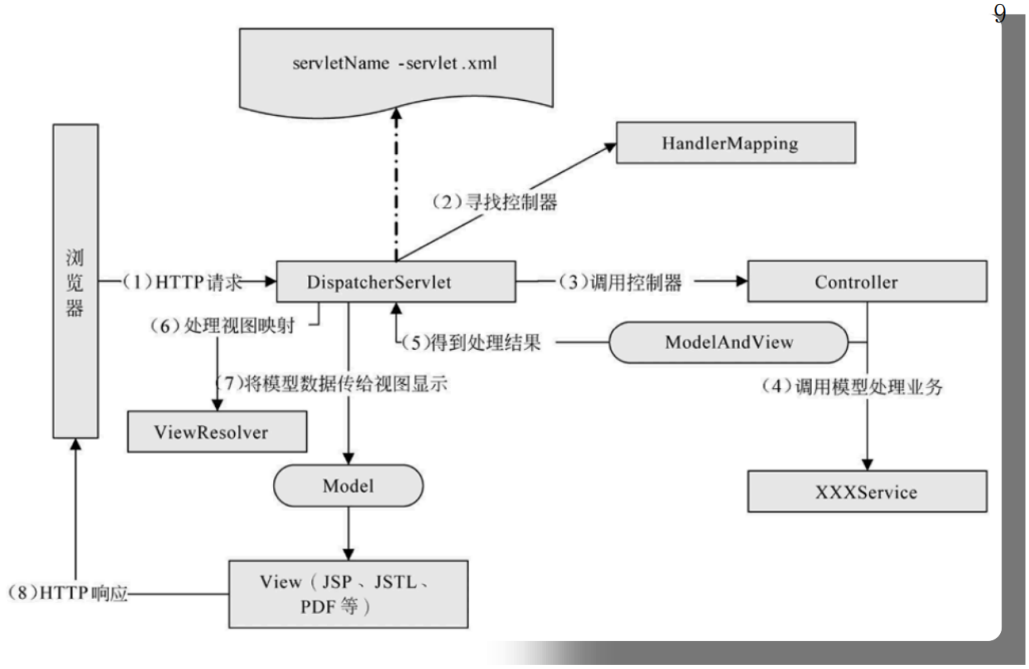

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:mvc="http://www.springframework.org/schema/mvc"
       xmlns:context="http://www.springframework.org/schema/context"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/mvc
        http://www.springframework.org/schema/mvc/spring-mvc.xsd
        http://www.springframework.org/schema/context
        http://www.springframework.org/schema/context/spring-context.xsd">

    <!-- 扫描包 -->
    <context:component-scan base-package="com.malu"></context:component-scan>

    <!-- 开启SpringMVC注解支持 -->
    <mvc:annotation-driven></mvc:annotation-driven>

</beans>
```

编写控制器：




```java
@Controller
public class MyController {

    @RequestMapping("/v1/hello")
    public void helloMVC(){
        System.out.println("hello SpringMVC!");

        // 还没有给前端返回任何内容
    }

}
```

3, 执行流程



SpringMVC的组件：

- DispatcherServlet：前端控制器，接受所有请求，调用其他组件。
- HandlerMapping：处理器映射器，根据配置找到方法的执行链。
- HandlerAdapter：处理器适配器，根据方法类型找到对应的处理器。
- ViewResolver：视图解析器，找到指定视图。

组件的工作流程：

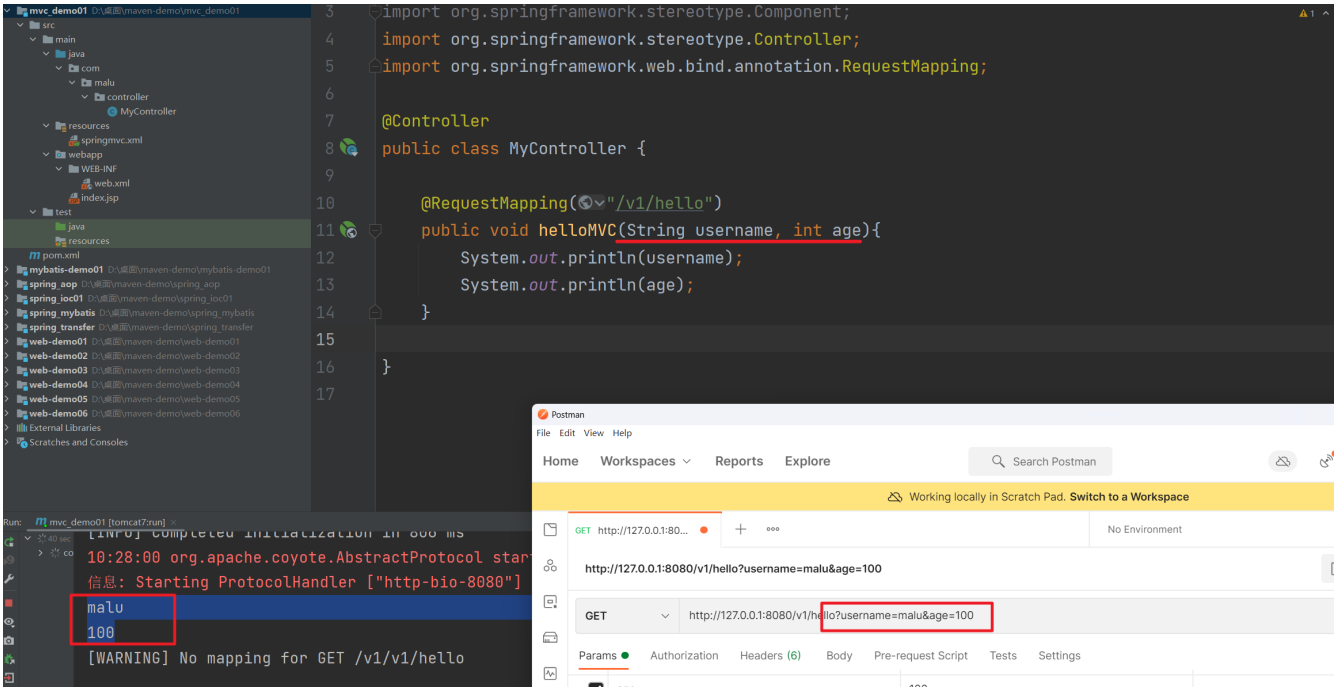
- 客户端将请求发送给前端控制器。
- 前端控制器将请求发送给处理器映射器，处理器映射器根据路径找到方法的执行链，返回给前端控制器。
- 前端控制器将方法的执行链发送给处理器适配器，处理器适配器根据方法类型找到对应的处理器。
- 处理器执行方法，将结果返回给前端控制器。
- 前端控制器将结果发送给视图解析器，视图解析器找到视图文件位置。
- 视图渲染数据并将结果显示到客户端。

4, 参数获取（封装为简单数据类型）

在Servlet中我们通过 request.getParameter(name) 获取请求参数。该方式存在两个问题：

- 请求参数较多时会出现代码冗余。
- 与容器紧耦合。

而SpringMVC支持参数注入的方式用于获取请求数据，即将请求参数直接封装到方法的参数当中。用法如下：

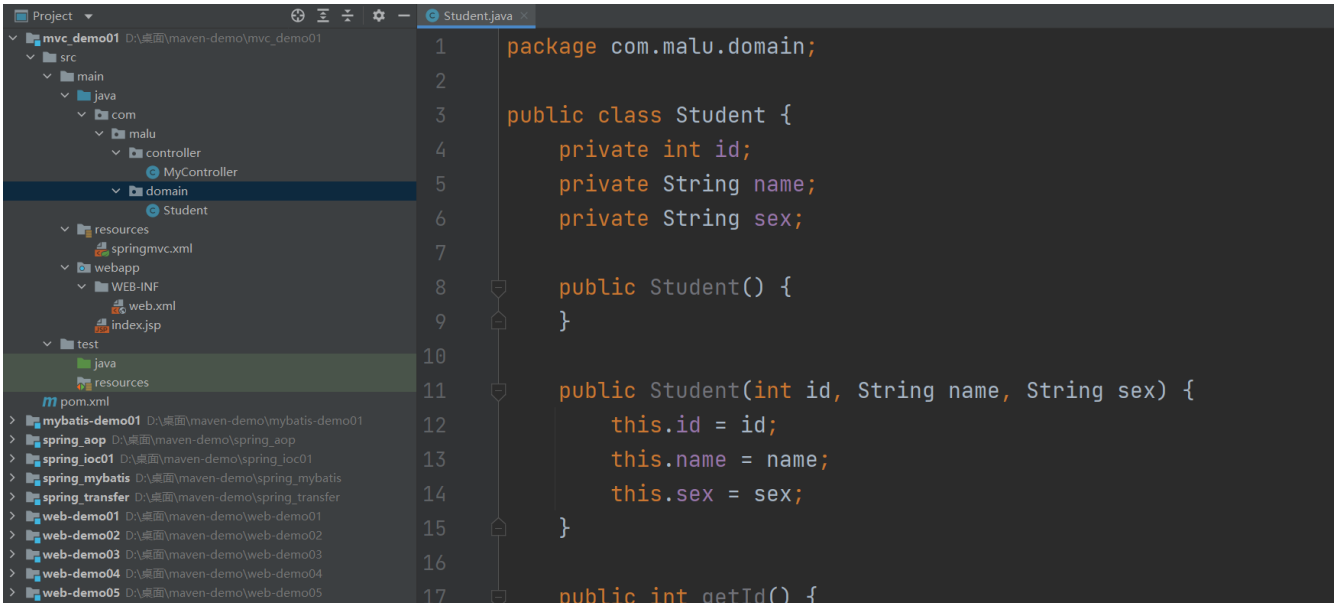



```
http://127.0.0.1:8080/v1/hello?username=malu&age=100
```

```
@Controller
public class MyController {

    @RequestMapping("/v1/hello")
    public void helloMVC(String username, int age){
        System.out.println(username);
        System.out.println(age);
    }

}
```

5, 参数获取（封装为对象类型）

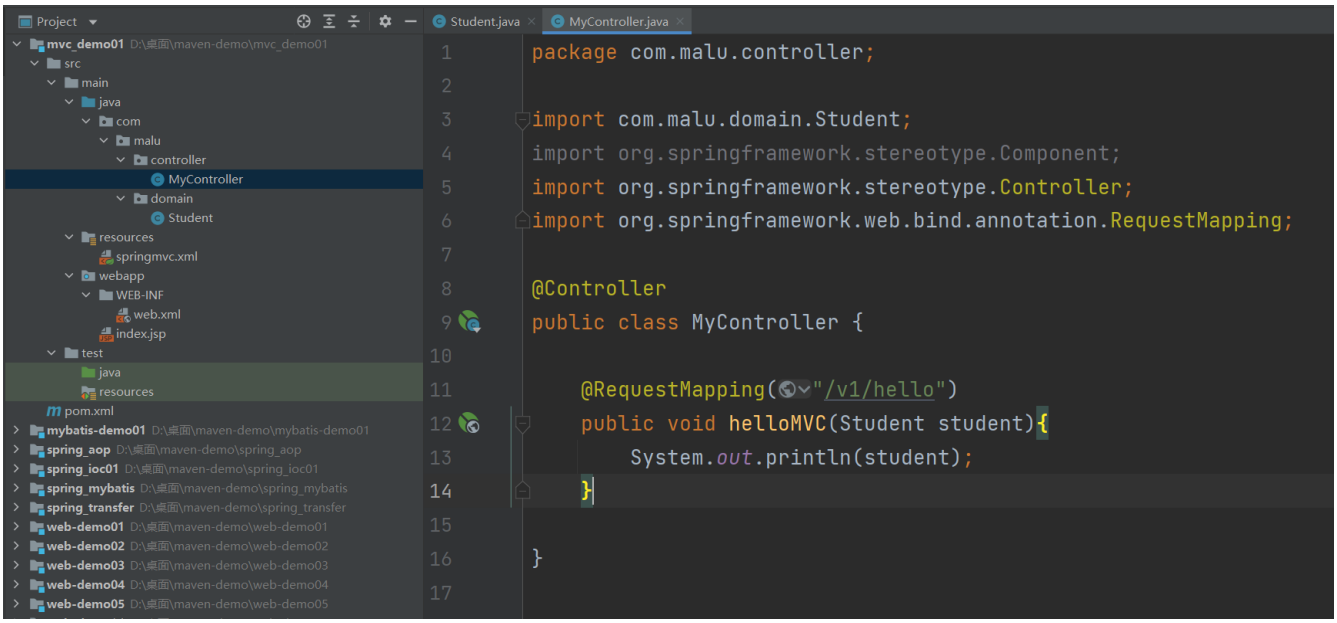
SpringMVC支持将参数直接封装为对象，编写实体类：



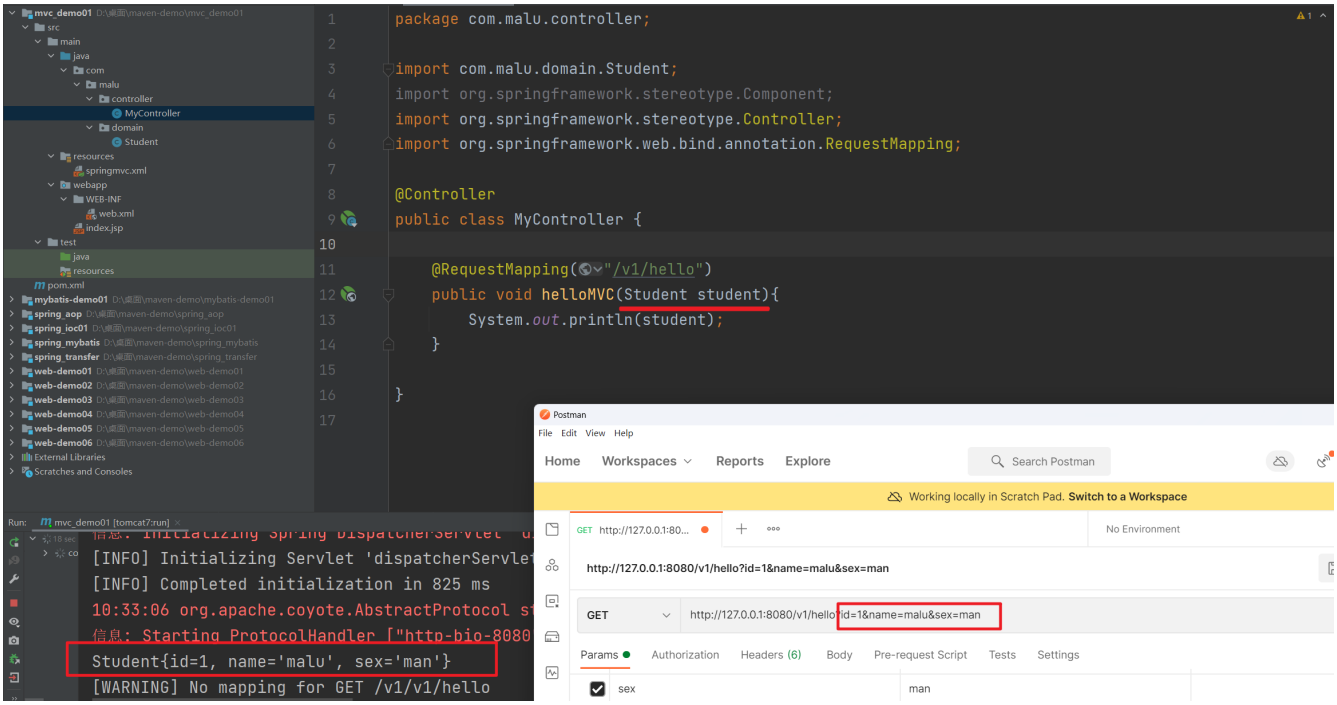

```java
package com.malu.domain;

public class Student {
    private int id;
    private String name;
    private String sex;

    public Student() {
    }

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

    public int getId() {
        return id;
    }

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

    public String getName() {
        return name;
    }

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

    public String getSex() {
        return sex;
    }
```

```java
    public void setSex(String sex) {
        this.sex = sex;
    }

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

编写控制器：



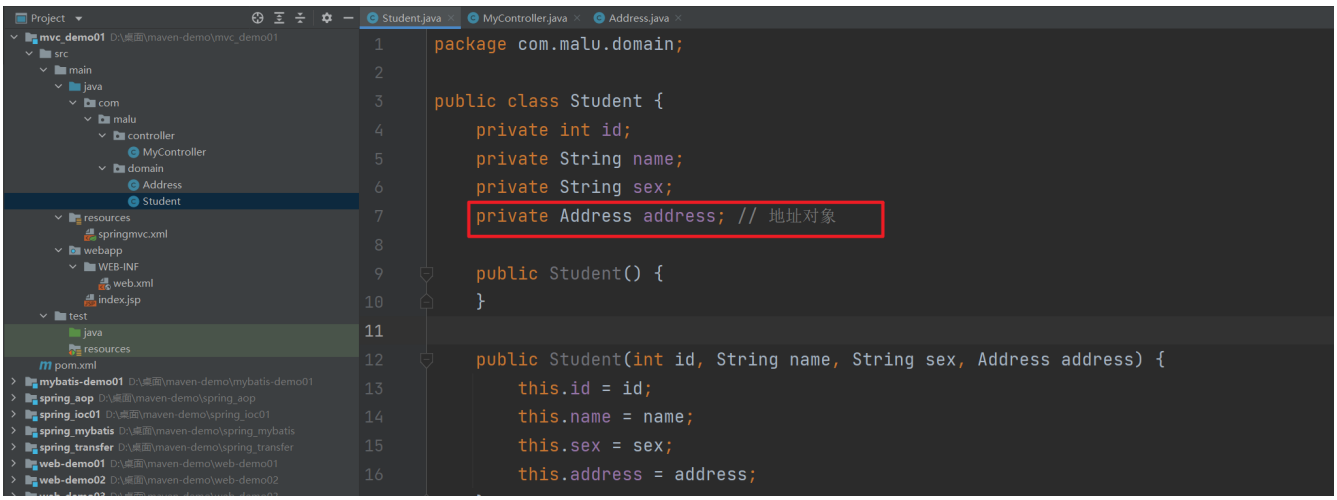
访问该方法时，请求参数名和方法参数的属性名相同，即可完成自动封装。



封装关联对象，编写实体类：


```java
package com.malu.domain;

public class Address {
    private String info; //地址信息
    private String postcode; //邮编

    public String getInfo() {
        return info;
    }

    public void setInfo(String info) {
        this.info = info;
    }
```

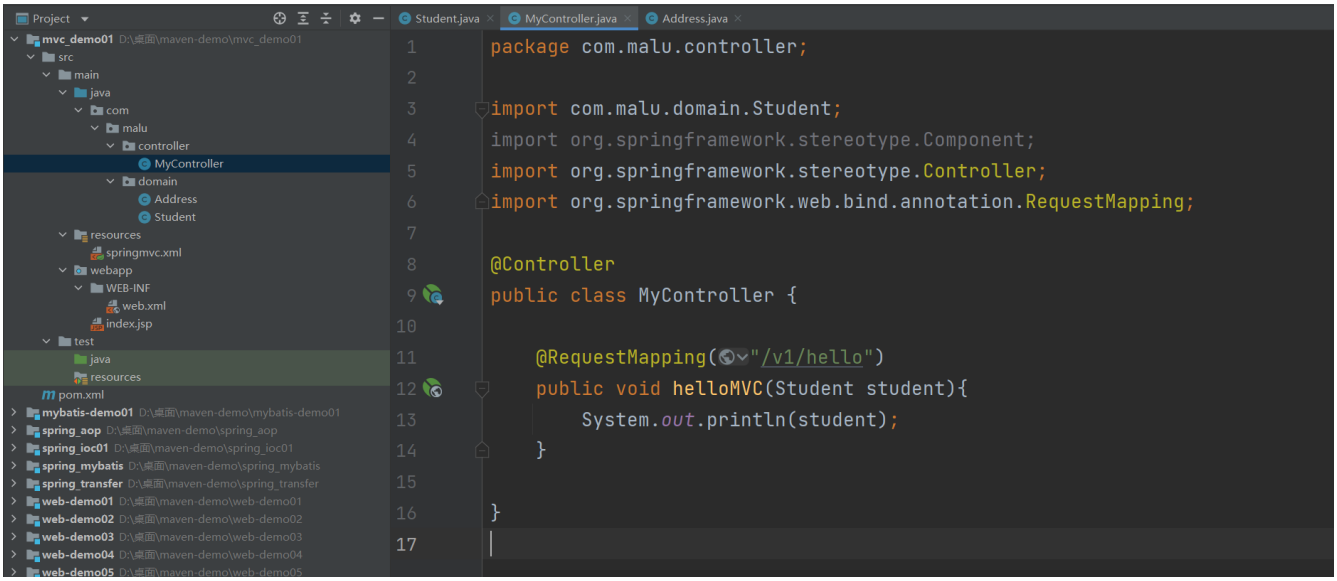
```java
    public String getPostcode() {
        return postcode;
    }

    public void setPostcode(String postcode) {
        this.postcode = postcode;
    }

    @Override
    public String toString() {
        return "Address{" +
                "info='" + info + '\'' +
                ", postcode='" + postcode + '\'' +
                '}';
    }
}
```




```java
package com.malu.domain;

public class Student {
    private int id;
    private String name;
    private String sex;
    private Address address; // 地址对象

    public Student() {
    }

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

    public int getId() {
```

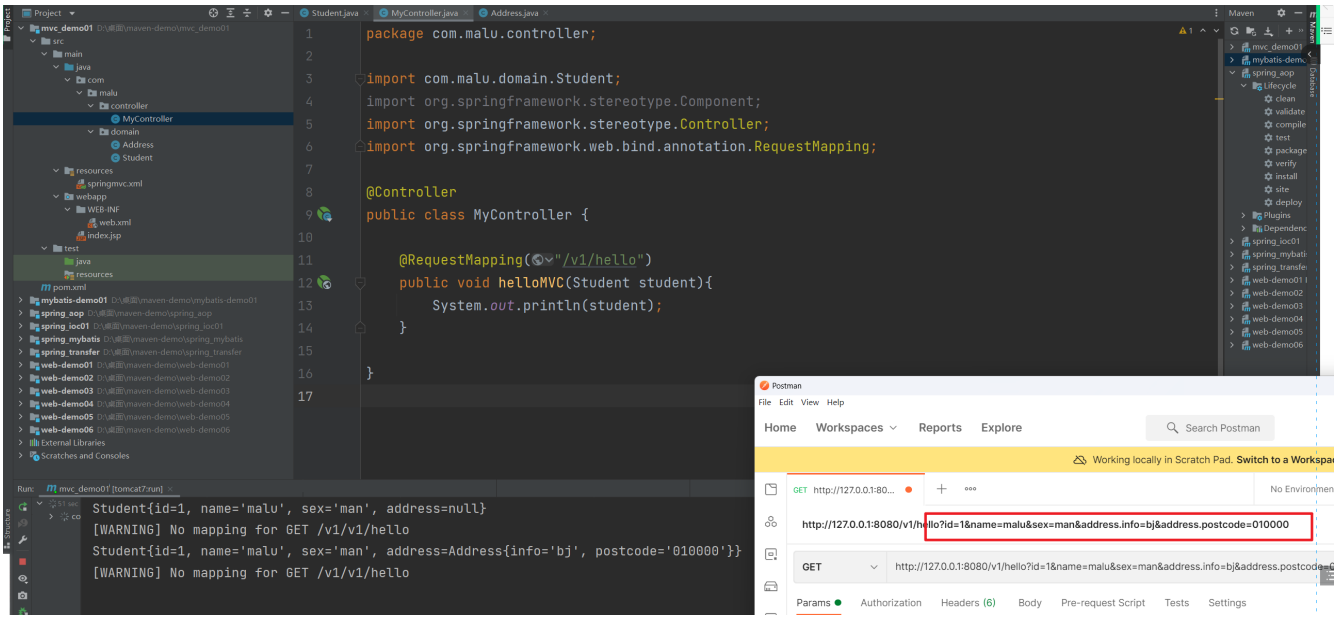
```java
        return id;
    }

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

    public String getName() {
        return name;
    }

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

    public String getSex() {
        return sex;
    }

    public void setSex(String sex) {
        this.sex = sex;
    }

    public Address getAddress() {
        return address;
    }

    public void setAddress(Address address) {
        this.address = address;
    }

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

编写控制器方法:

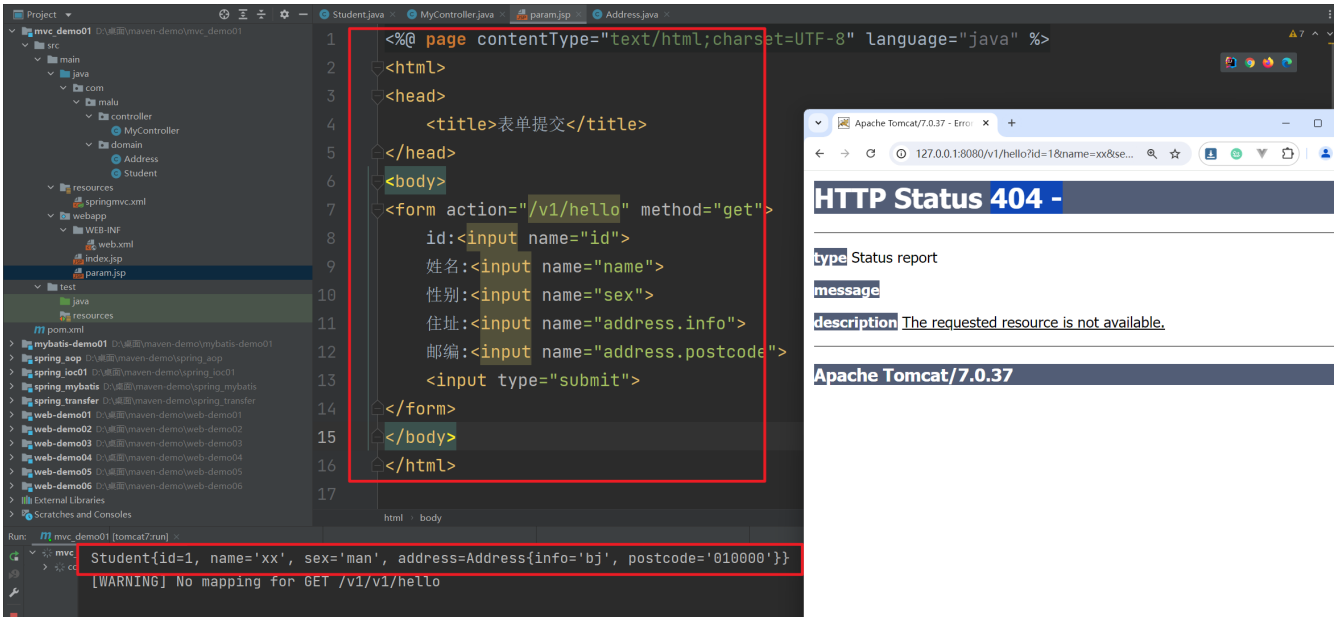


访问该方法时，请求参数名和方法参数的属性名相同，即可完成自动封装。

```
http://127.0.0.1:8080/v1/hello?
id=1&name=malu&sex=man&address.info=bj&address.postcode=010000
```



我们也可以使用表单发送带有参数的请求：

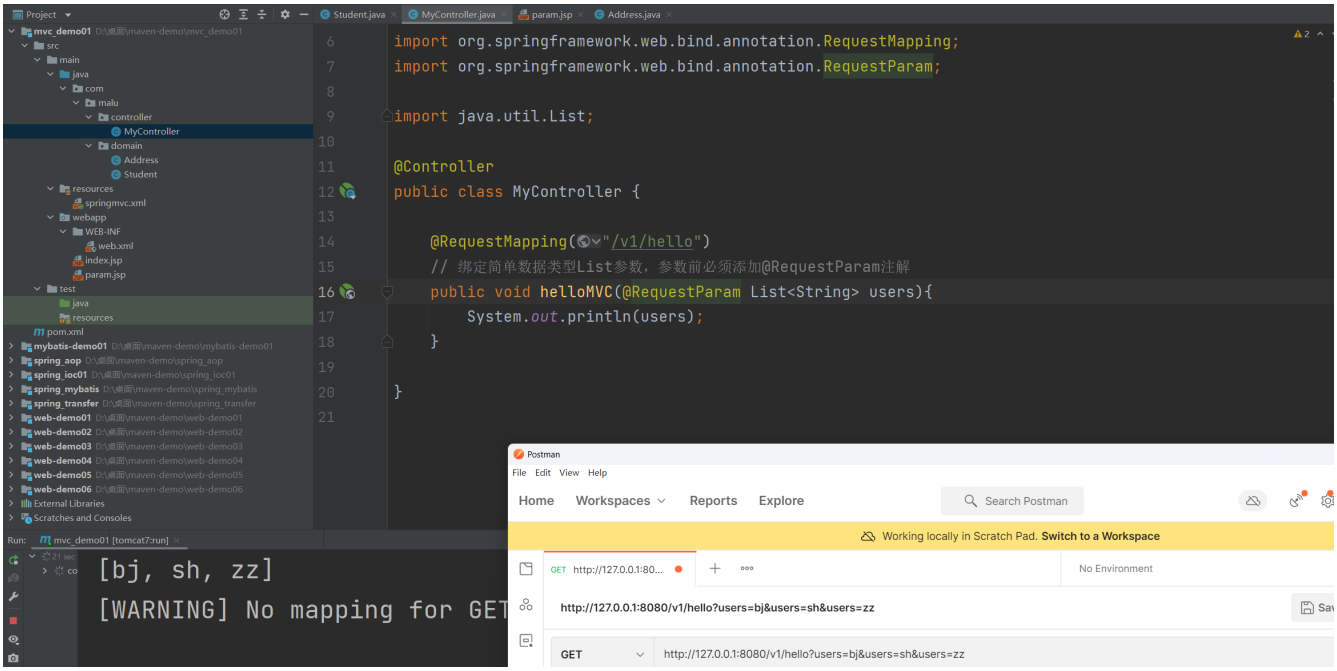


表单代码：

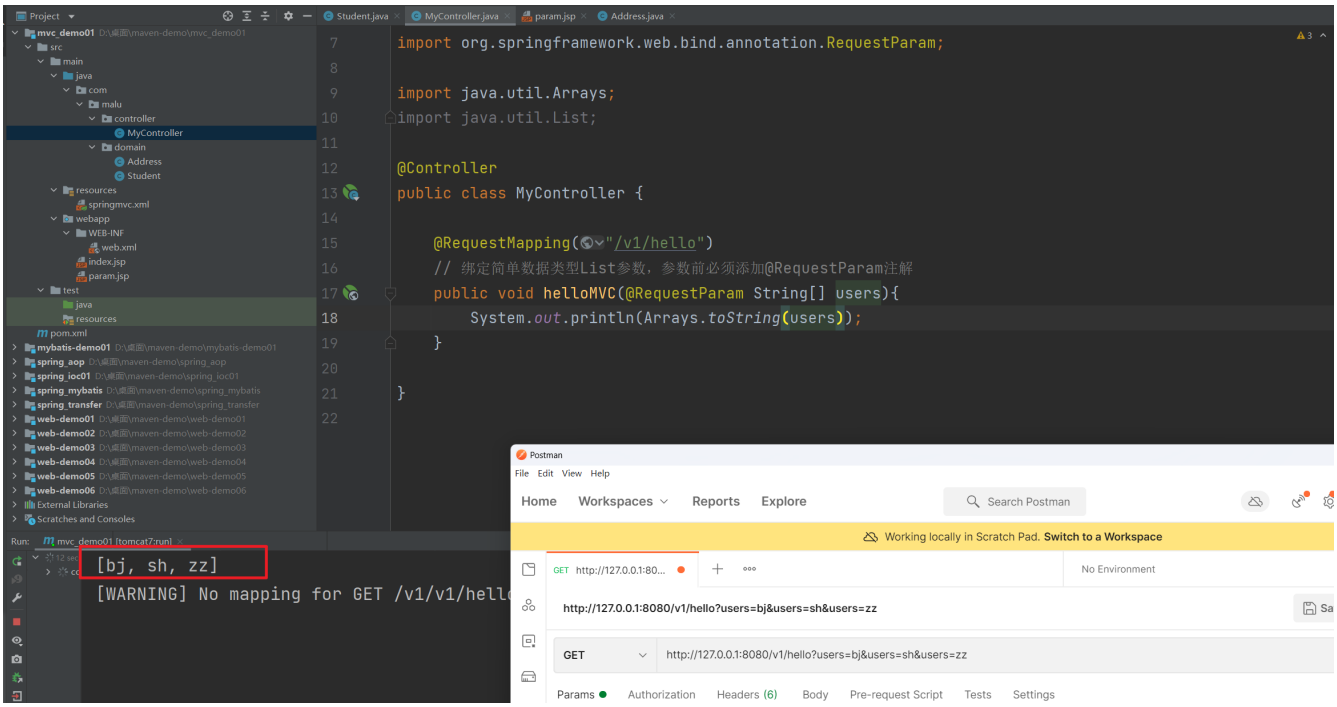
```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>表单提交</title>
</head>
<body>
<form action="/v1/hello" method="get">
    id:<input name="id">
    姓名:<input name="name">
    性别:<input name="sex">
    住址:<input name="address.info">
    邮编:<input name="address.postcode">
    <input type="submit">
</form>
</body>
</html>
```

6, 参数获取（封装为集合类型）

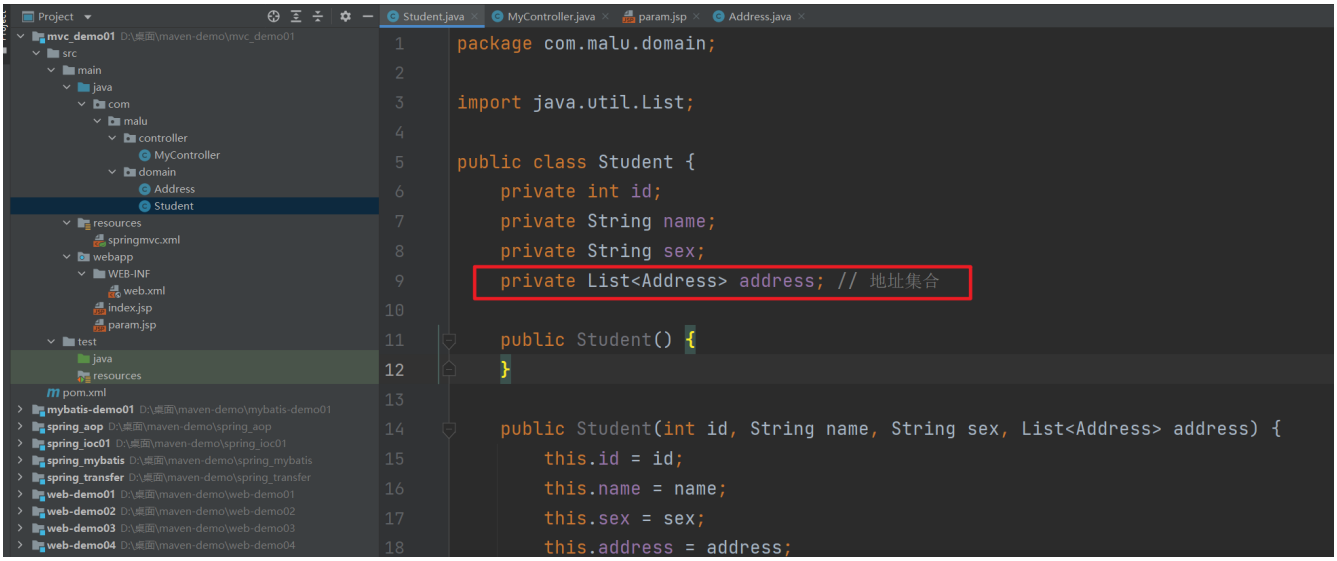
SpringMVC支持将参数封装为List或Map集合，封装成List集合，封装为简单数据类型集合，所谓的简单类型就是字符串+基本数据类型，编写控制器：



也可以绑定数组类型：



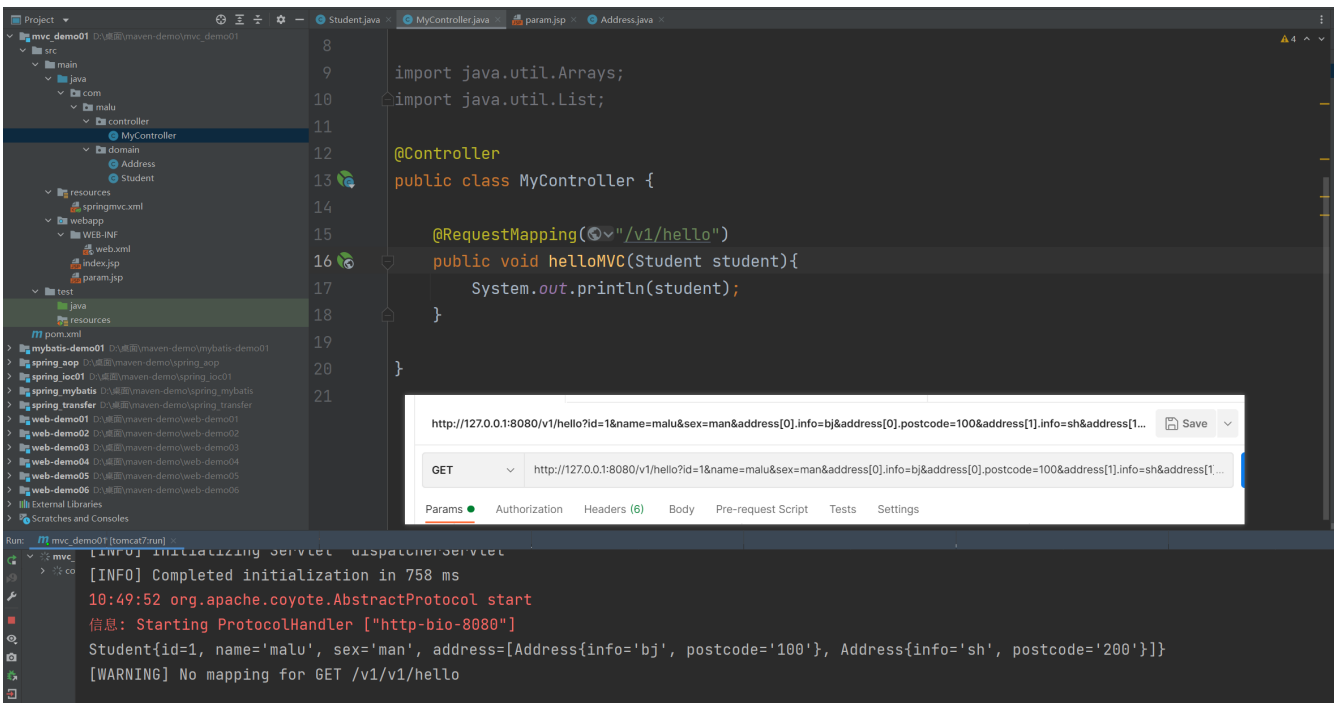
封装为对象类型集合，SpringMVC不支持将参数封装为对象类型的List集合，但可以封装到有List属性的对象中。编写实体类： List




```java
package com.malu.domain;

import java.util.List;

public class Student {
    private int id;
    private String name;
    private String sex;
    private List<Address> address; // 地址集合

    public Student() {
    }

    public Student(int id, String name, String sex, List<Address> address) {
        this.id = id;
        this.name = name;
        this.sex = sex;
        this.address = address;
    }

    public int getId() {
        return id;
    }

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

    public String getName() {
        return name;
    }

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

    public String getSex() {
```

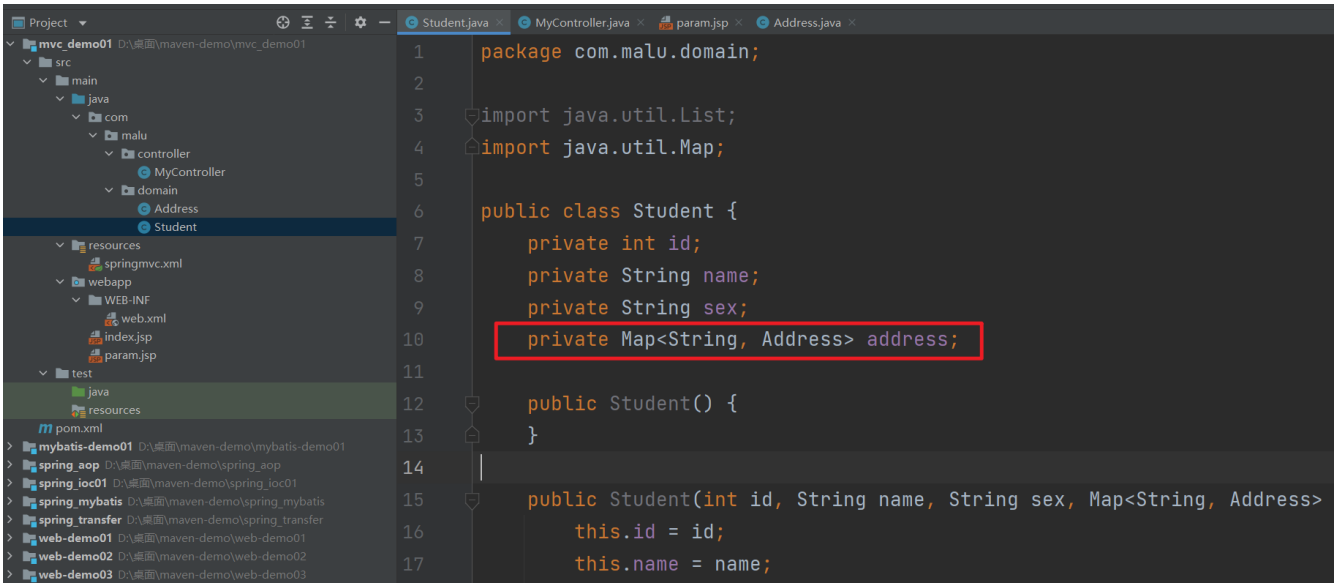
```java
        return sex;
    }

    public void setSex(String sex) {
        this.sex = sex;
    }

    public List<Address> getAddress() {
        return address;
    }

    public void setAddress(List<Address> address) {
        this.address = address;
    }

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

编写控制器：



```
http://127.0.0.1:8080/v1/hello?id=1&name=malu&sex=man&address[0].info=bj&address[0].postcode=100&address[1].info=sh&address[1].postcode=200
```

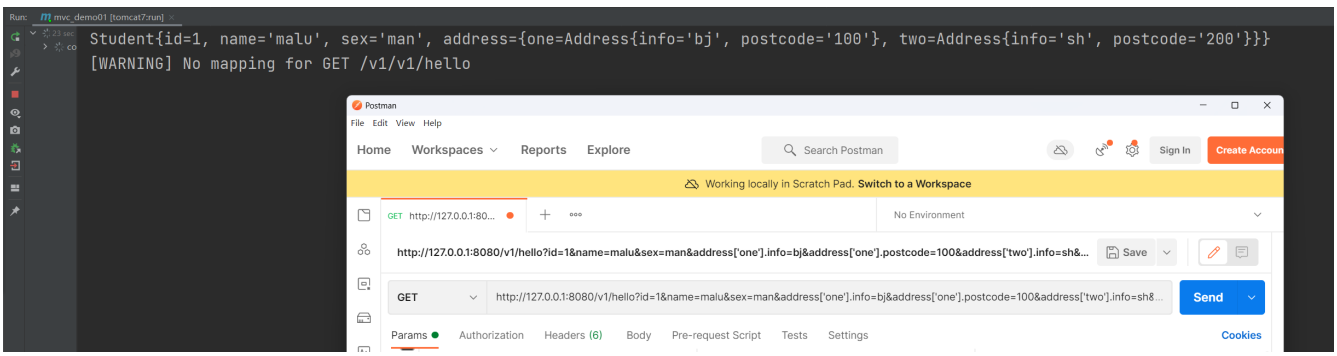
封装为Map集合，同样，SpringMVC要封装Map集合，需要封装到有Map属性的对象中。编写实体类：




```
package com.malu.domain;

import java.util.List;
import java.util.Map;

public class Student {
    private int id;
    private String name;
    private String sex;
    private Map<String, Address> address;

    public Student() {
    }

    public Student(int id, String name, String sex, Map<String, Address> address) {
        this.id = id;
        this.name = name;
        this.sex = sex;
        this.address = address;
    }

    public int getId() {
        return id;
    }

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

```java
    public String getName() {
        return name;
    }

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

    public String getSex() {
        return sex;
    }

    public void setSex(String sex) {
        this.sex = sex;
    }

    public Map<String, Address> getAddress() {
        return address;
    }

    public void setAddress(Map<String, Address> address) {
        this.address = address;
    }

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

编写控制器：



```
http://127.0.0.1:8080/v1/hello?
id=1&name=malu&sex=man&address['one'].info=bj&address['one'].postcode=100&address['two'].in
fo=sh&address['two'].postcode=200
```

7, 参数获取（Servlet原生对象获取参数）

8, 参数获取（自定义参数类型转化器）

9, 参数获取（编码过滤器）

10, 处理响应（配置视图解析器）

11, 处理响应（控制器方法的返回值）

12, 处理响应（request域设置数据）

13, 处理响应（session域设置数据）

14, 处理响应（context域设置数据）

15, 处理响应（请求转发与重定向）

16, @Controller

17, @RequestMapping

18, @RequestParam

19, @RequestHeader,@CookieValue

20, @SessionAttributes

21, @ModelAttribute

22, RESTful风格支持