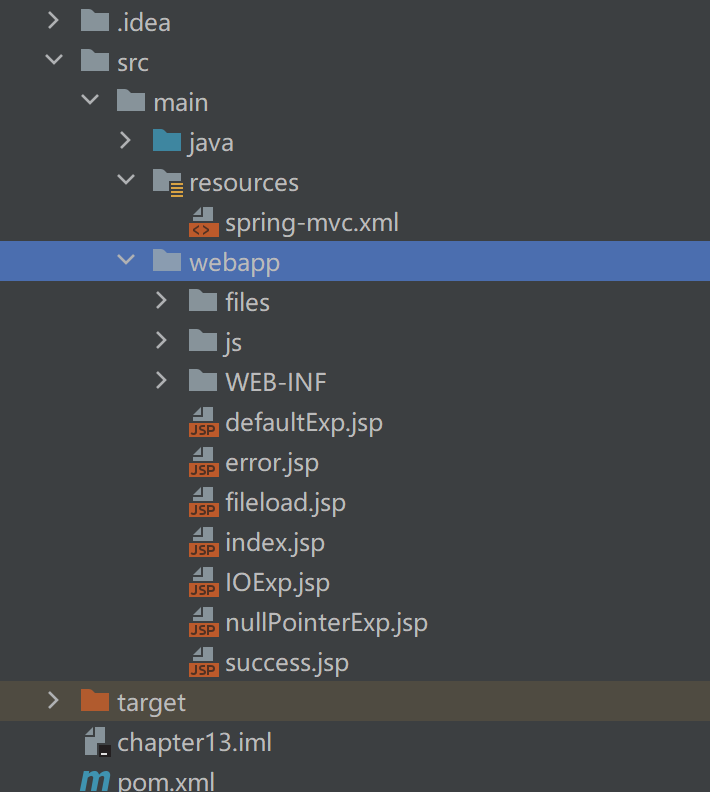
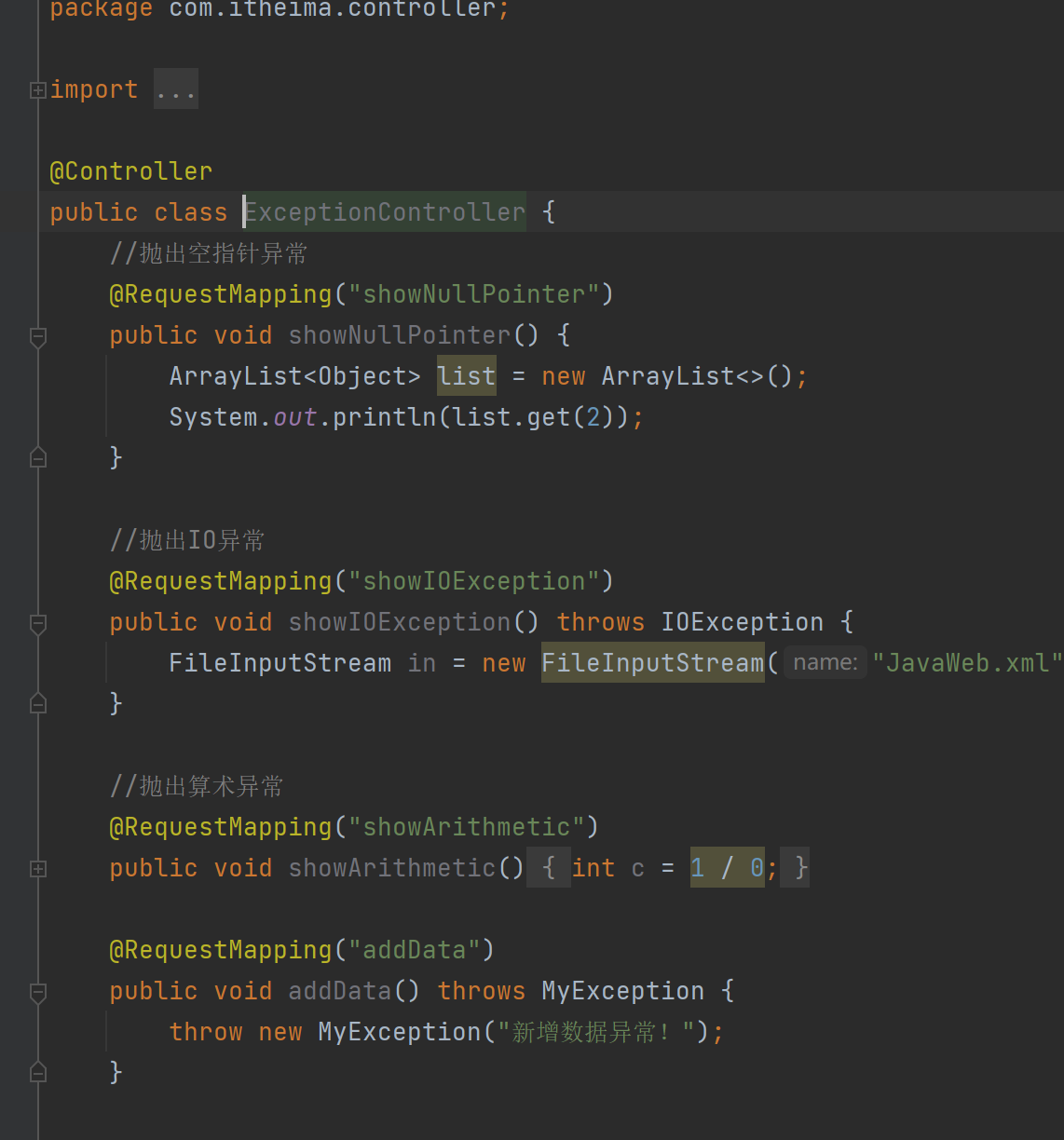
**简单异常处理器**

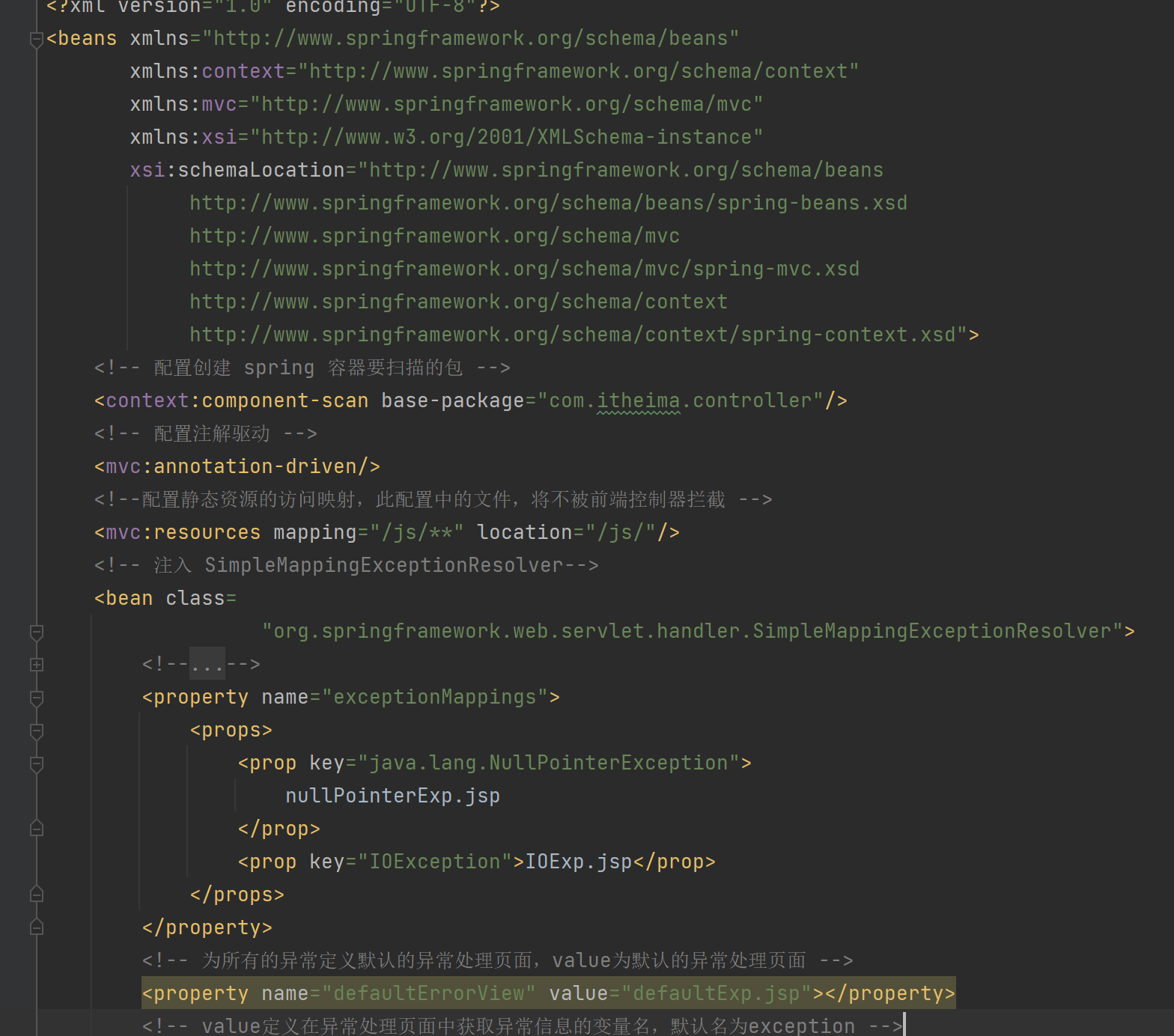
**（1）通过一个案例演示SimpleMappingExceptionResolver对异常的统一处理，案例具体实现步骤如下所示。在IDEA中创建一个名称为chapter13的Maven Web项目，并在项目chapter13中搭建好Spring MVC运行所需的环境。**



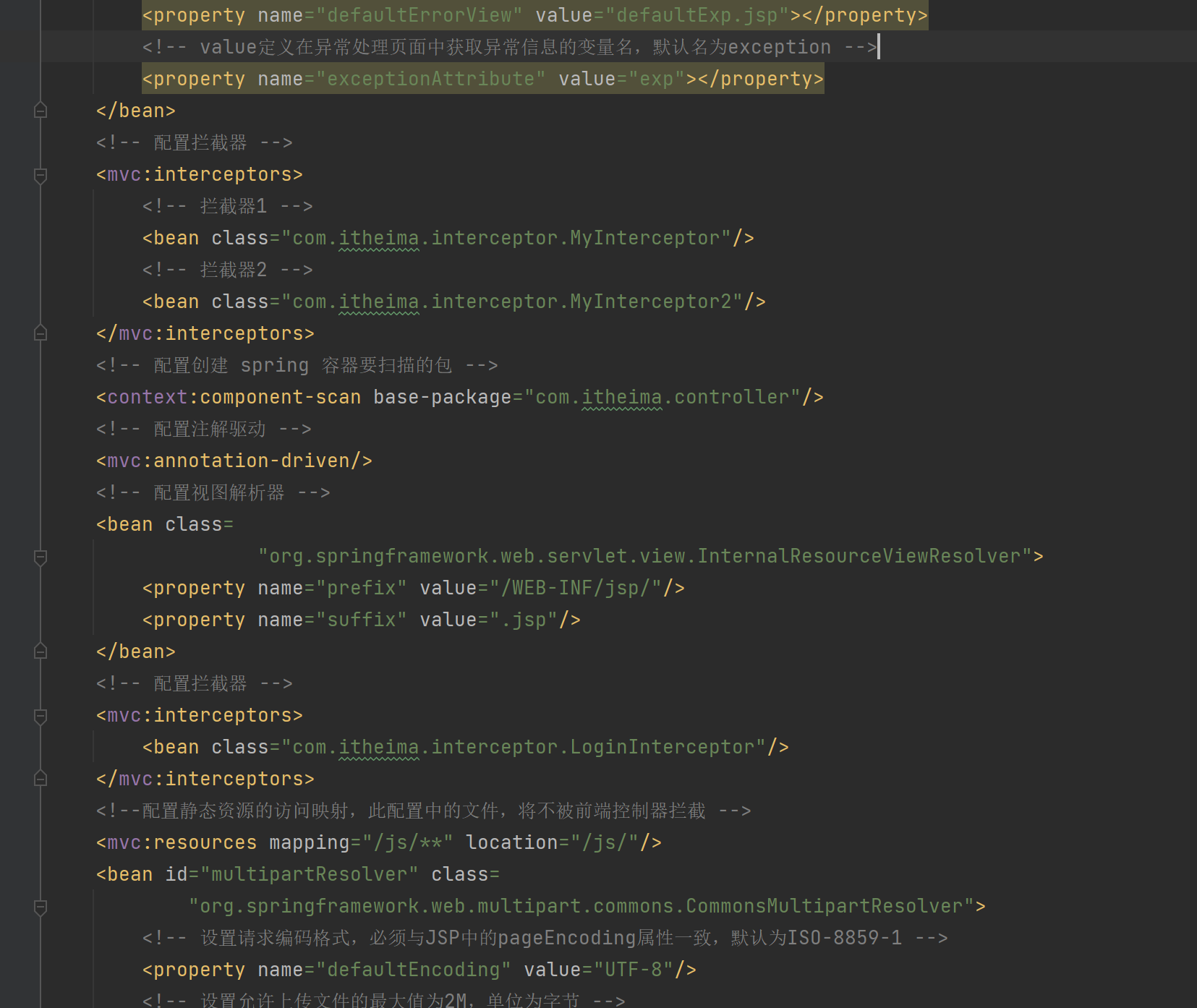
（2）**src\main\java目录下，创建一个名称为com.itheima.controller的包。在包中创建ExceptionController类，ExceptionController类的具体代码如下所示。**



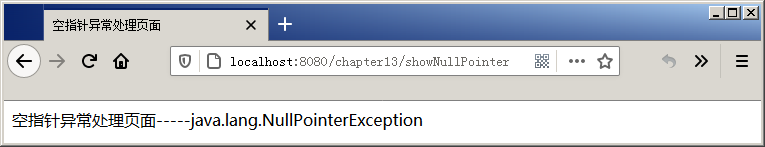
（3）**程序执行文件ExceptionController.java中的任意一个方法时，都会抛出异常。在异常发生时，如果要跳转到指定的处理页面，则需要在Spring MVC的配置文件spring-mvc.xml中使用SimpleMappingExceptionResolver指定异常和异常处理页面的映射关系。Spring MVC配置文件的部分配置如下所示。**



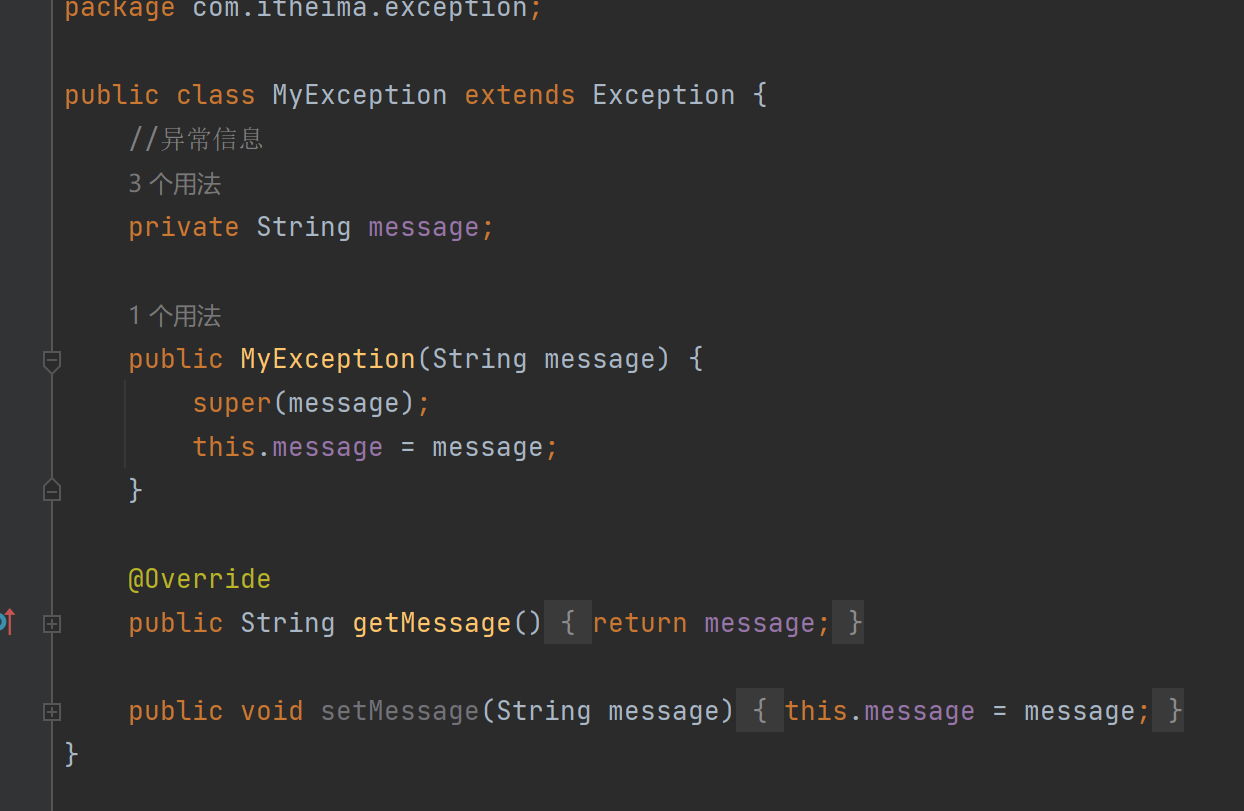
（4）**在文件spring-mvc.xml中，已经指定了异常类别对应的异常处理页面，接下来创建这些异常处理页面。在此不对异常处理页面做太多处理，只在页面中展示对应的异常信息。**



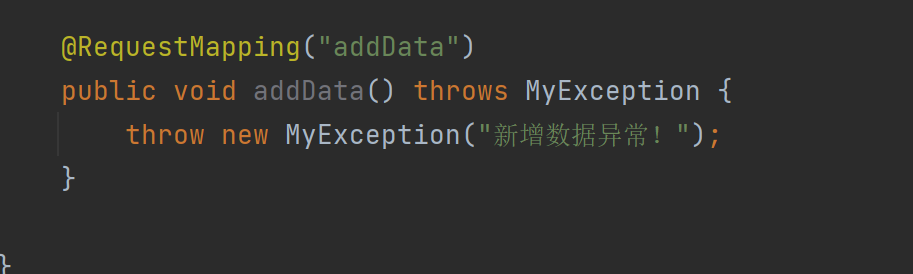
（5）**启动chapter13项目，在浏览器中访问地址http://localhost:8080/chapter13/showNullPointer，程序将执行showNullPointer()方法。方法执行后页面显示效果如图所示。**

****

**自定义异常处理器**

**（1）通过一个案例演示自定义异常处理器分类别处理自定义异常和系统自带的异常，案例具体实现步骤如下所示。在src\main\java目录，创建一个路径为com.itheima.exception的包，并在包中创建自定义异常类MyException。** 

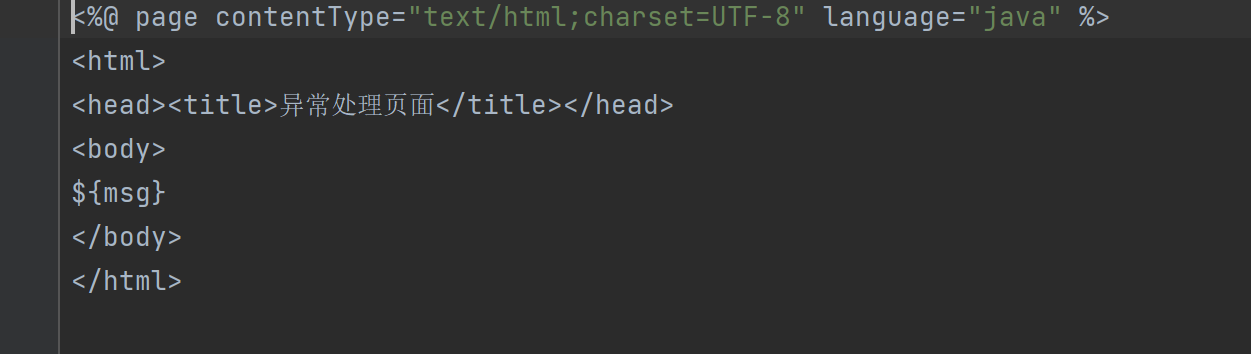
**（2）修改文件ExceptionController.java，在ExceptionController类中，新增方法addData()用于抛出自定义异常，addData()方法的具体代码如下所示。**



**（3）在com.itheima.controller包下，创建名称为MyExceptionHandler的自定义异常处理器。在MyExceptionHandler类中重写resolveException()方法，用于判断当前异常是自定义异常还是系统自带的异常，根据异常的种类不同，resolveException()方法返回不同的异常信息。使用自定义异常处理器，需要先将自定义异常处理器注册到Spring MVC中。**



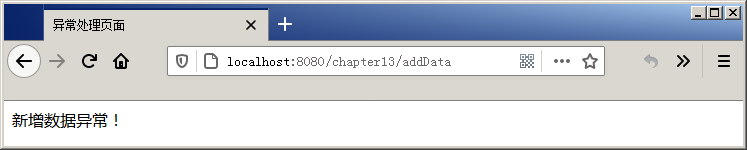
**（4）在src\main\webapp目录下，创建一个名称为error.jsp的文件，用作异常处理页面。本案例不对异常处理页面进行过多处理，只将异常信息打印在页面上。**



**（5）启动chapter13项目，在浏览器中访问地址http://localhost:8080/chapter13/showNullPointer，程序将执行showNullPointer()方法，方法执行后页面显示效果如图所示。**

****

**（6）在浏览器中访问地址http://localhost:8080/chapter13/addData，程序将执行addData()方法，方法执行后页面显示效果如图所示。**

****

**异常处理注解**

**（1）接下来通过一个案例演示使用注解实现异常的分类处理，具体实现步骤如下所示。**

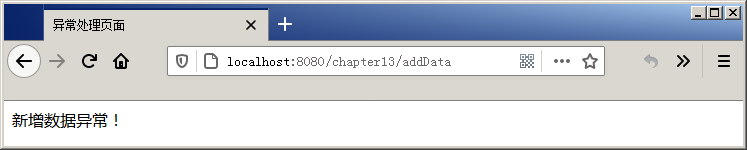
**在com.itheima.controller包下，创建名称为ExceptionAdvice的异常处理器。ExceptionAdvice类中定义2个处理不同异常的方法，其中doMyException()方法用来处理Handler执行时抛出的自定义异常， doOtherException()方法用来处理Handler执行时抛出的系统异常。**



**（2）启动chapter13项目，在浏览器中访问地址http://localhost:8080/chapter13/showNullPointer，程序将执行showNullPointer()方法，方法执行后页面显示效果如图所示。**

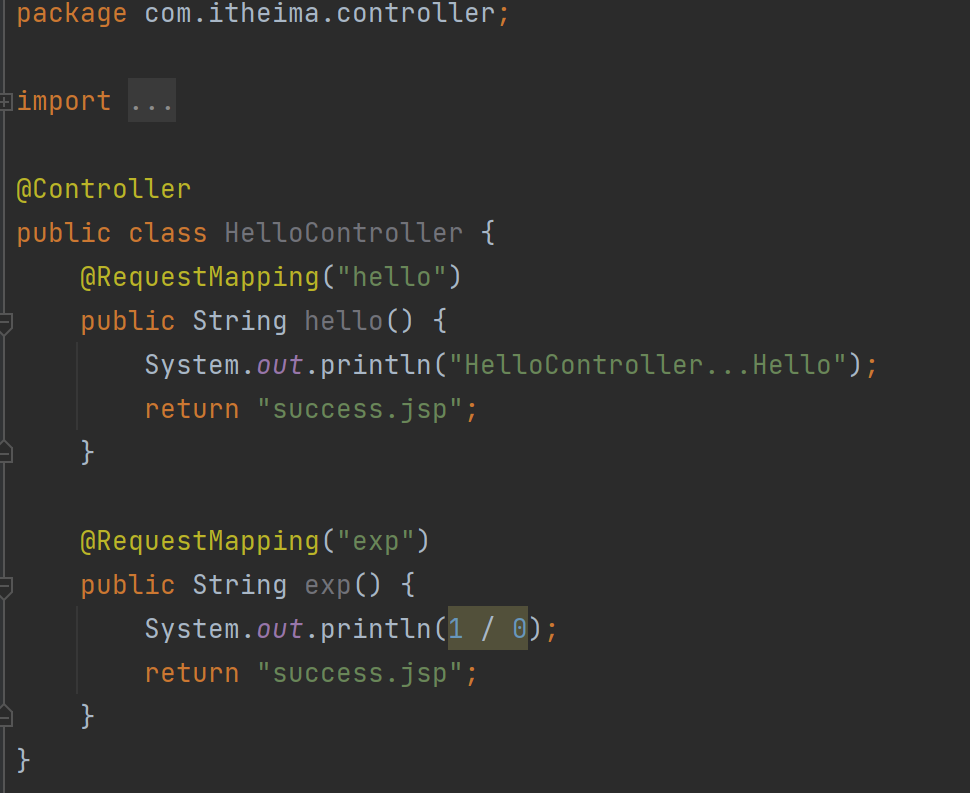
****

**（3）在浏览器中访问地址http://localhost:8080/chapter13/addData，程序将执行addData()方法，方法执行后页面显示效果如图所示。**

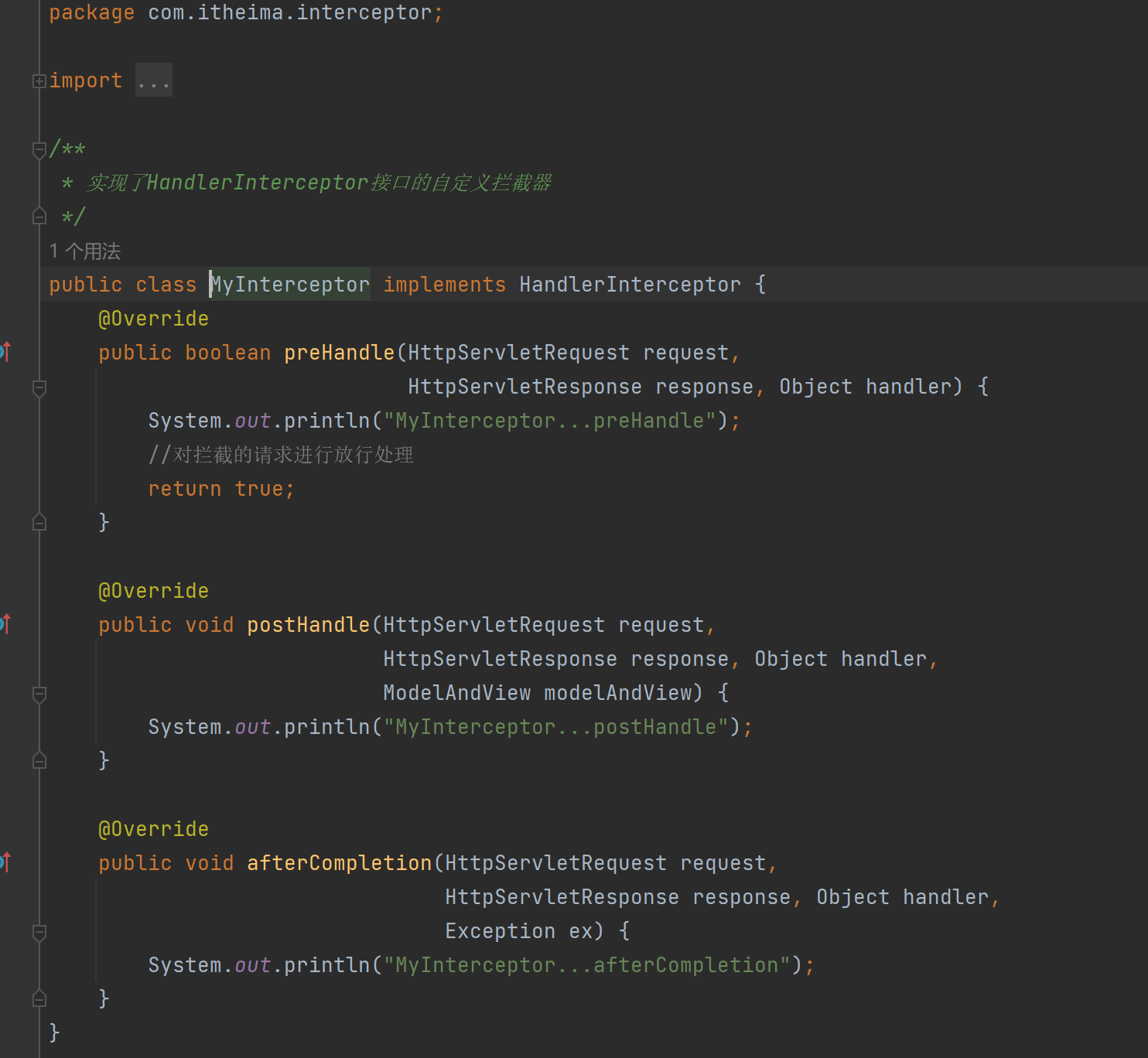
****

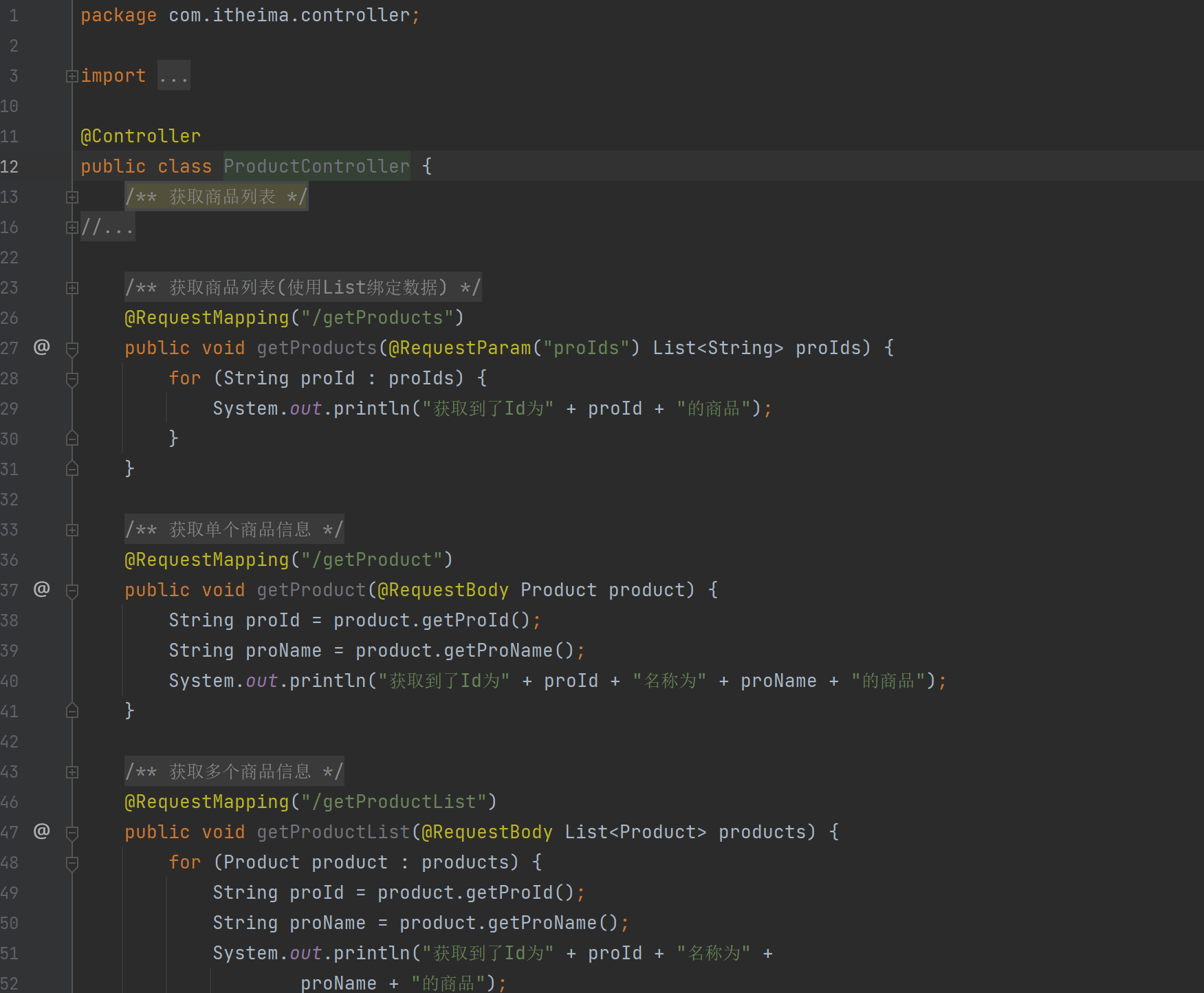
**拦截器**

**（1）通过一个案例演示单个拦截器的执行流程，案例具体实现步骤如下所示。在chapter13项目的com.itheima.controller包下，创建名称为HelloController的控制器类，在HelloControlle类中定义2个方法，其中hello()方法用于正常处理客户端的请求，exp()方法被调用时产生异常。**

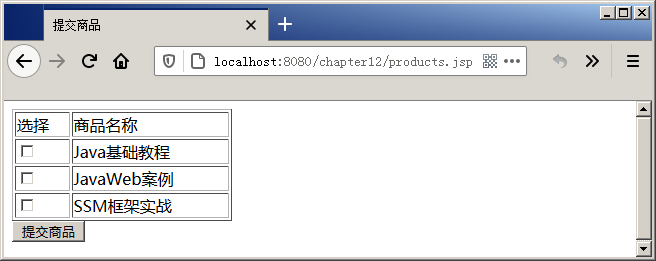


**（2）在src\main\java目录下，创建一个路径为com.itheima.interceptor的包，并在包中创建实现HandlerInterceptor接口的拦截器MyInterceptor。在MyInterceptor类中重写的3个方法内编写输出语句来验证方法的执行情况。**

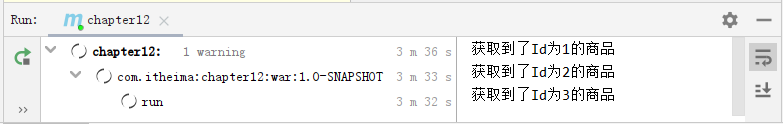


**（3）在com.itheima.controller包中创建一个商品处理器类ProductController，在ProductController类中定义getProducts() 方法，用于接收表单提交的商品id。**

**（4）启动chapter12项目，在浏览器中访问提交商品页面products.jsp，访问地址为http://localhost:8080/chapter12/products.jsp，products.jsp显示效果如图所示。**

****

**（5）勾选products.jsp显示效果图中所示的全部复选框，然后单击“提交商品”按钮，控制台打印信息如下图所示。**

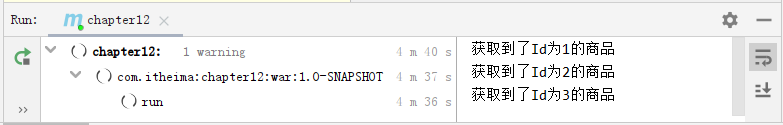
****

**集合绑定**

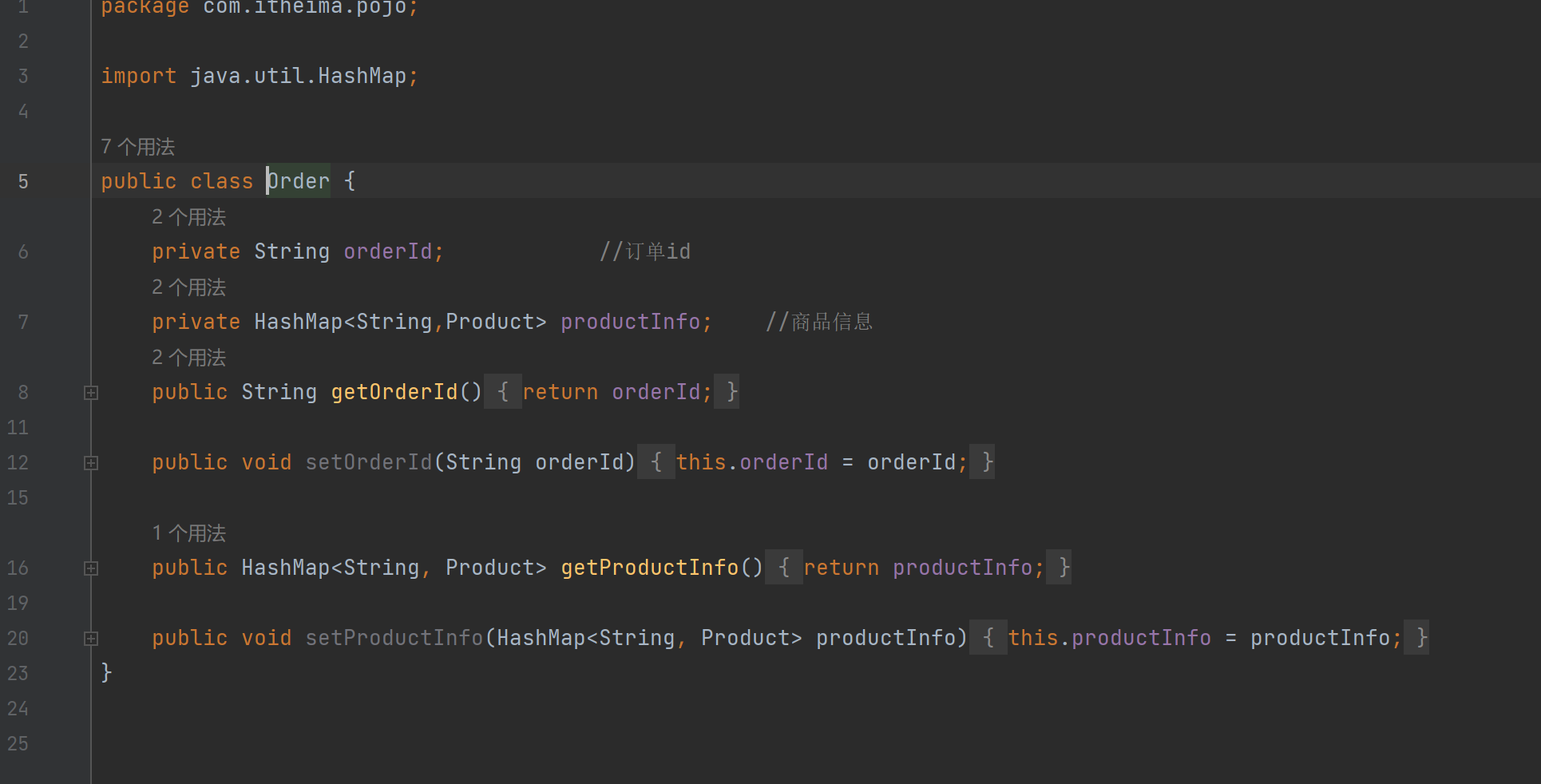
**（1）使用集合数据绑定来批量提交商品案例，具体实现步骤如下所示。修改ProductController.java类的getProducts()方法，让getProducts( )方法使用List类型来接受客户端的请求参数。**



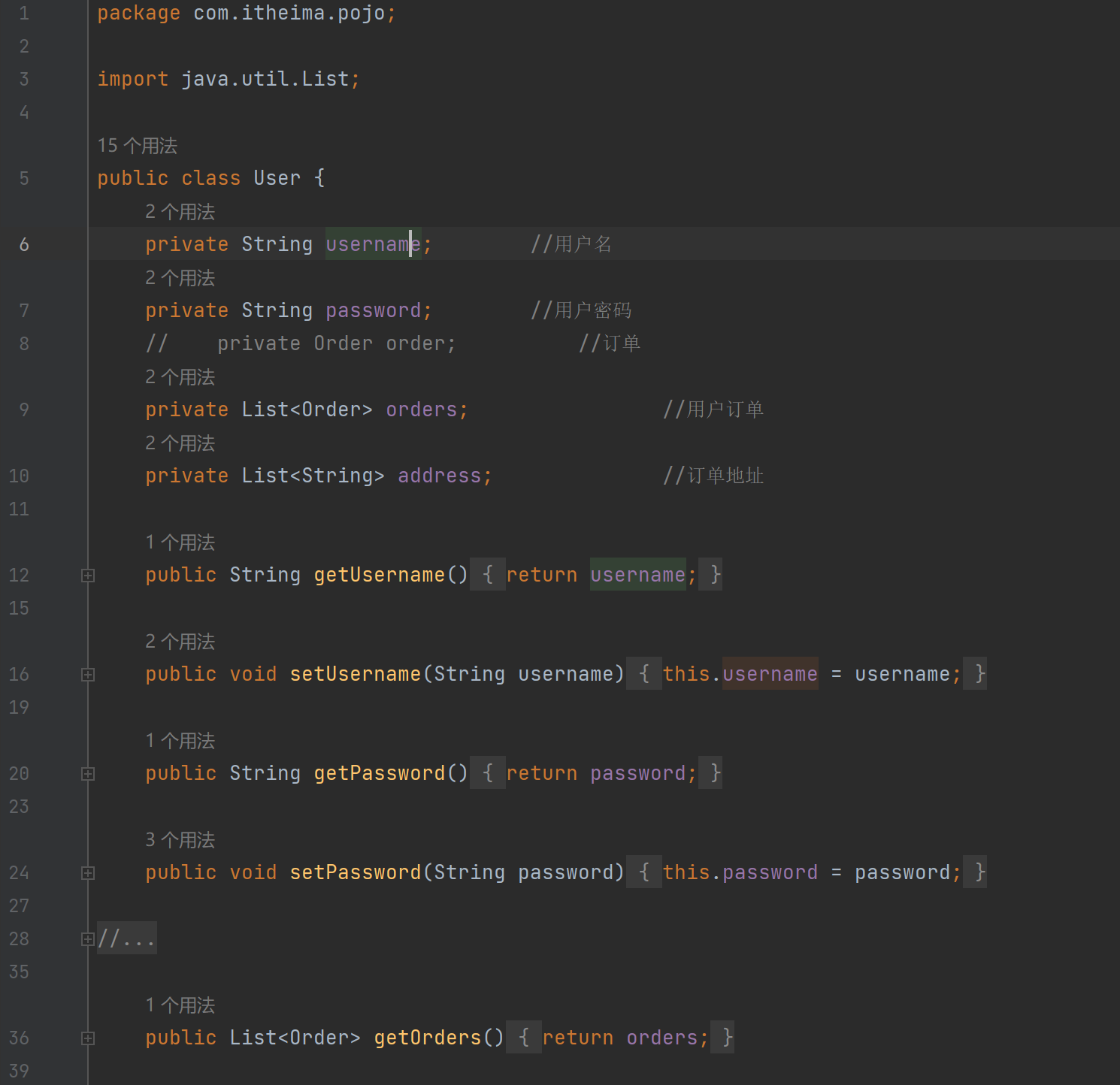
**（2）启动chapter12项目，在浏览器中访问地址http://localhost:8080/chapter12/products.jsp，勾选products.jsp页面表单的所有复选框，然后单击“提交商品”按钮，控制台打印信息如图所示。**

****

**复杂POJO绑定**

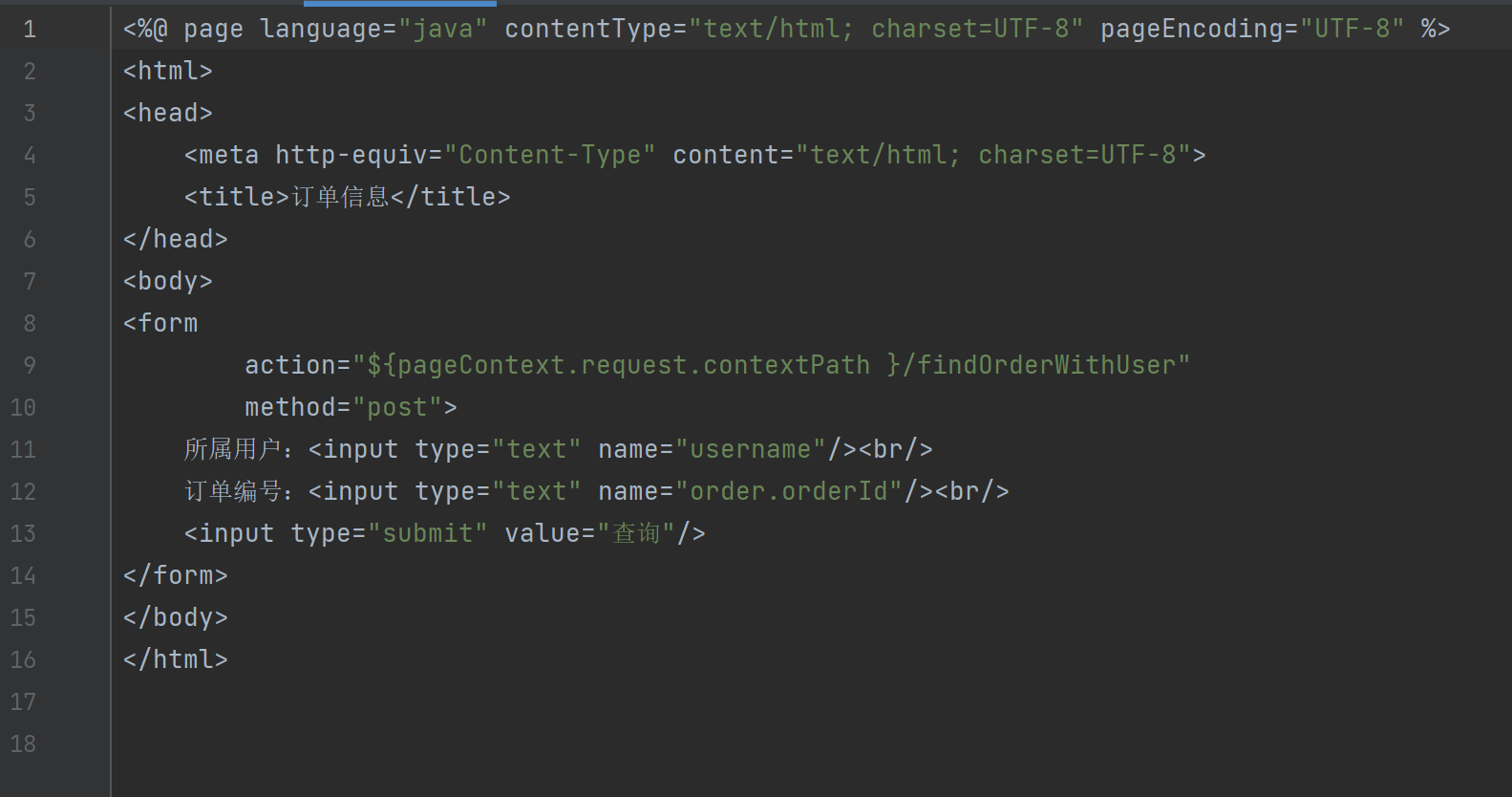
**（1）通过一个获取用户订单信息的案例，演示复杂POJO中对象类型的数据绑定，案例具体实现步骤如下。在com.itheima.pojo包中创建一个订单类Order，用于封装订单信息。**

**（2）修改User.java类，在User类中新增Order类型的属性order，并定义相应的getter和setter方法。**

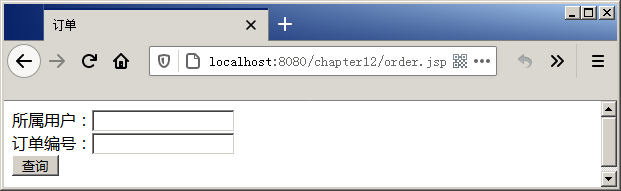
**（3）在UserController.java类中定义方法findOrderWithUser( )，用于获取客户端请求中的User信息。**



**（4）在项目的src\main\webapp目录下，创建一个订单信息文件order.jsp，在order.jsp文件中创建一个表单，表单中包含用户名和订单编号。表单提交时将用户名和订单编号信息发送到处理器.**

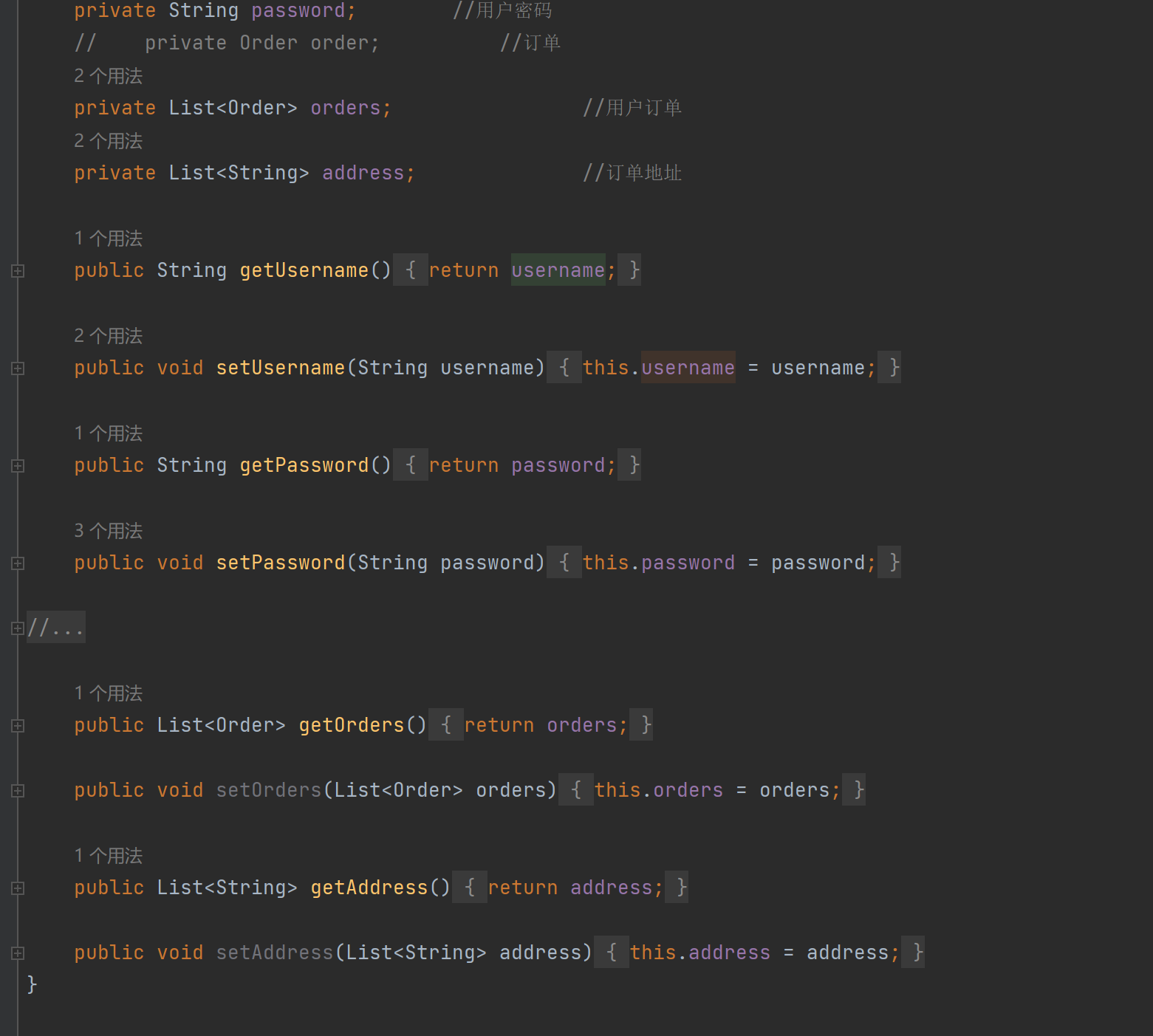


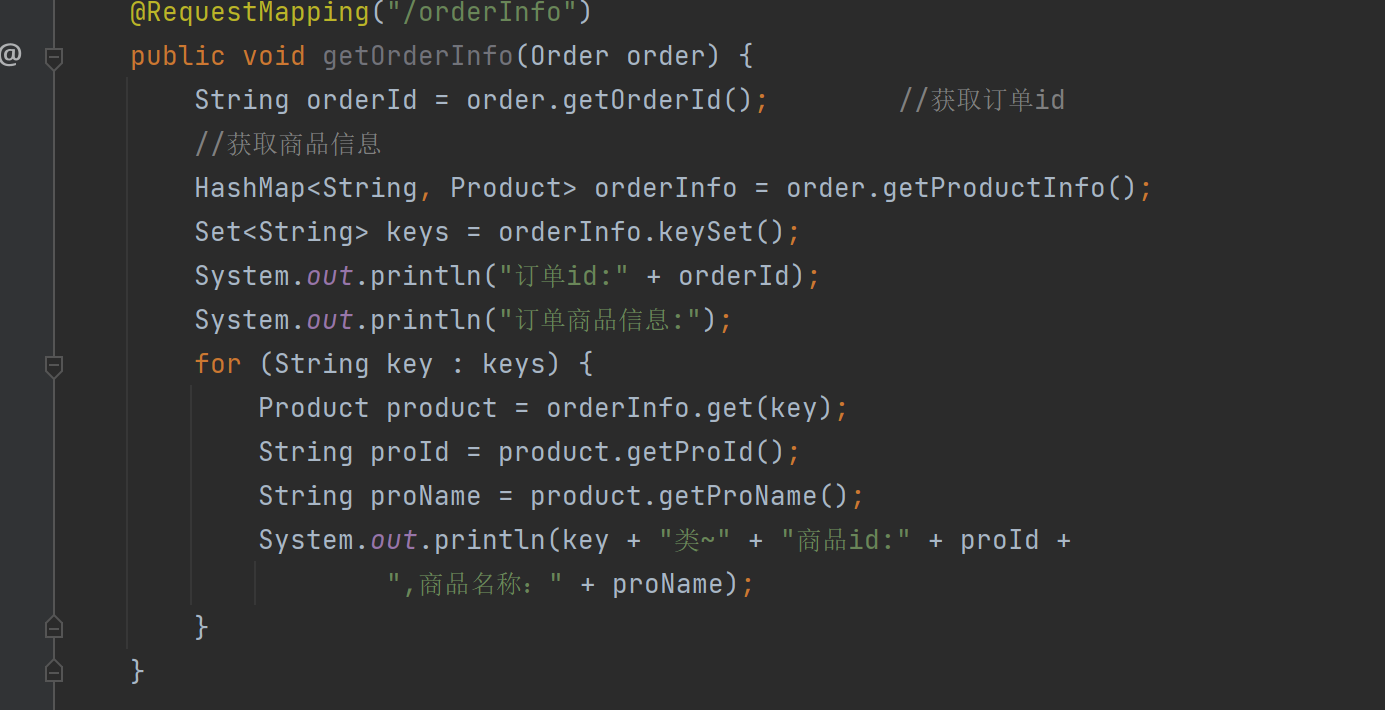
**（5）启动chapter12项目，在浏览器中访问订单页面order.jsp，访问地址为http://localhost:8080/chapter12/order.jsp，order.jsp的显示效果如图所示。**

****

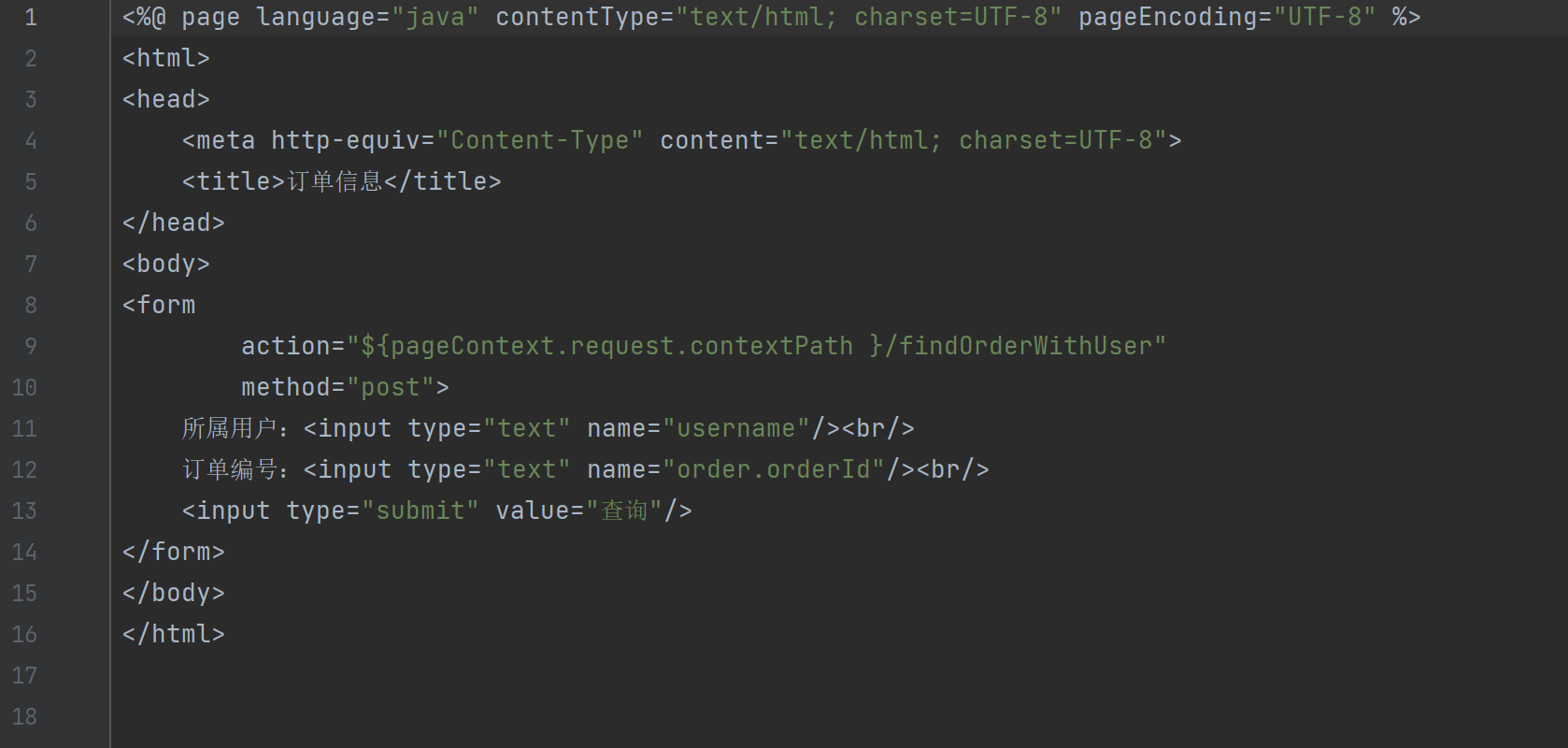
**复杂POJO绑定2**

**（1）通过一个获取用户订单信息的例子，演示复杂POJO中属性为List类型的数据绑定。**

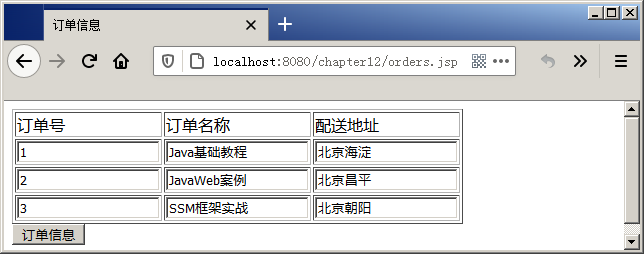


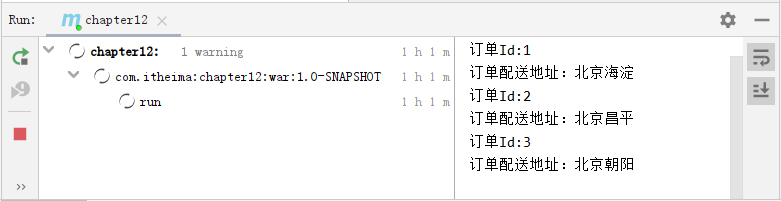
**（2）在com.itheima.controller包中创建一个订单处理器类OrderController，在OrderController类中定义showOrders( ) 方法，用于展示用户的订单信息。**

**（3）在项目的src\main\webapp目录下，创建一个订单信息文件orders.jsp，在orders.jsp中创建一个表单用于提交用户的订单信息。表单提交时，表单数据分别封装到User的订单属性orders和地址属性address中。**

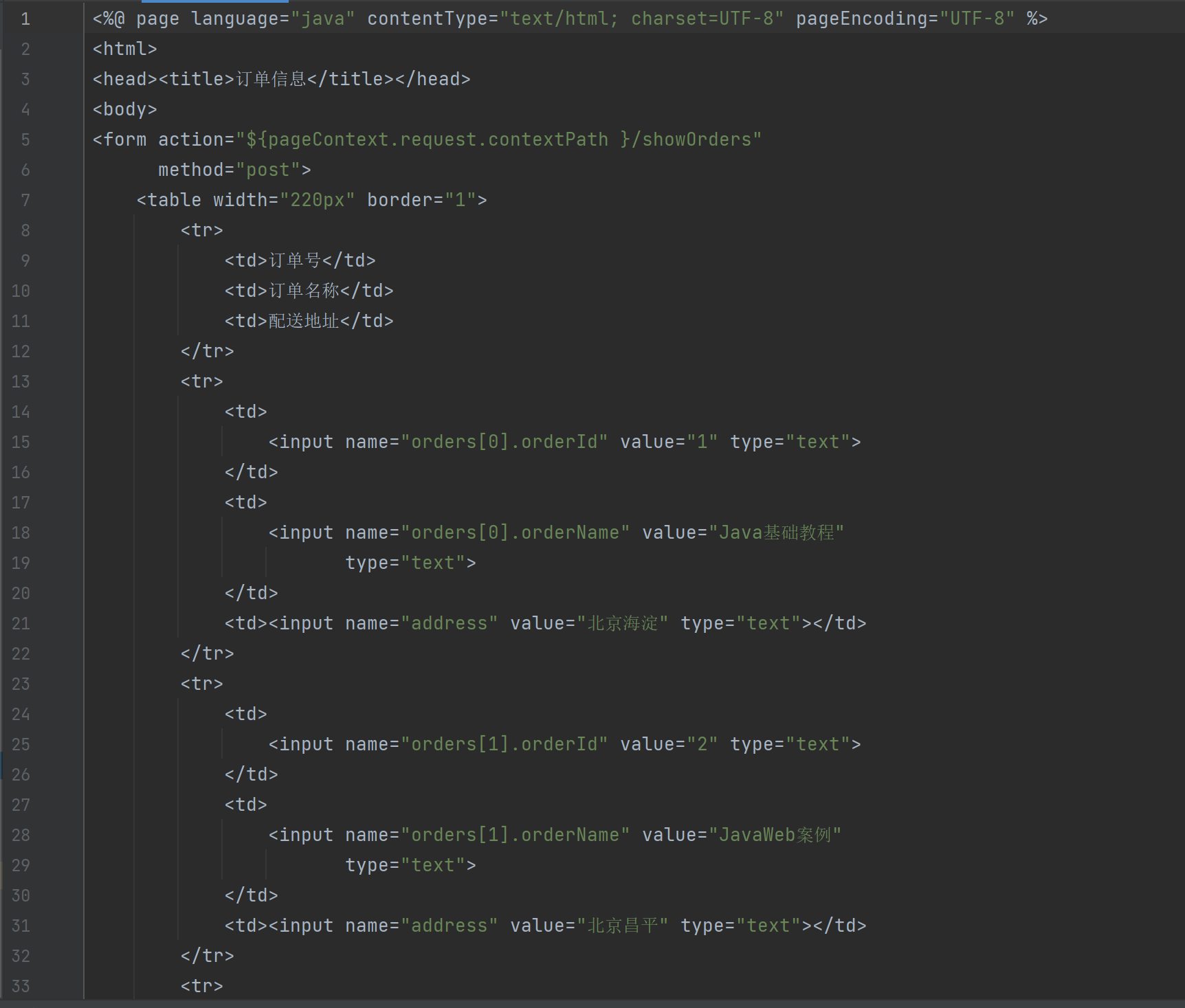


**（4）启动chapter12项目，在浏览器中访问订单信息页面orders.jsp，访问地址为http://localhost:8080/chapter12/orders.jsp，orders.jsp显示效果如图所示。**

****

**（5）在orders.jsp显示效果图所示的页面中，单击左下角“订单信息”按钮，orders.jsp表单中的订单信息发送到服务器端的showOrders()方法进行处理，控制台打印信息如图所示。**

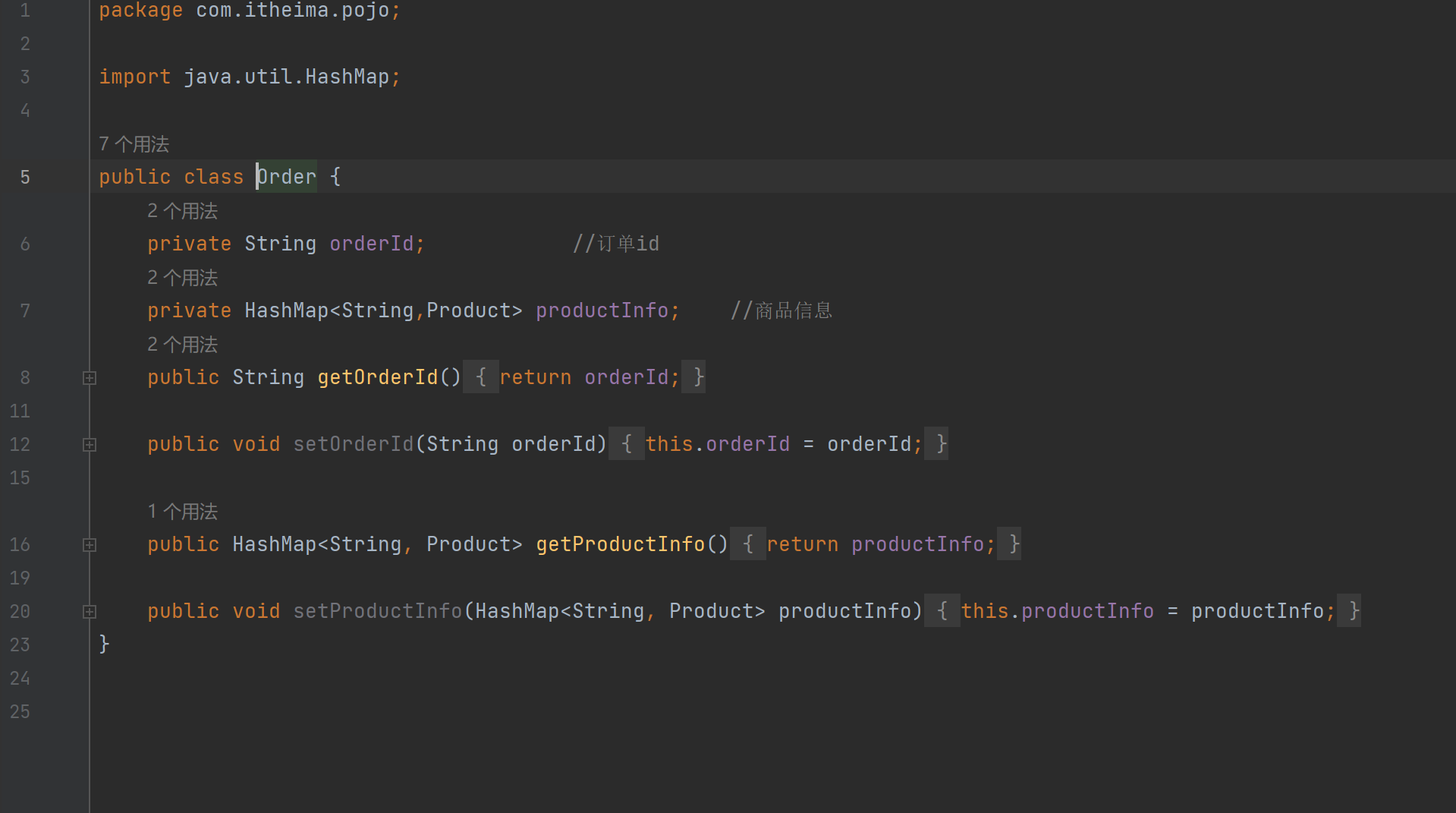
**复杂POJO绑定3**

**（1）通过一个获取订单信息的案例，演示复杂POJO中属性为Map类型的数据绑定，具体实现如下。修改Order.java类，在Order类中新增HashMap类型的属性productInfo，用于封装订单中的商品信息，其中productInfo的键用来存放商品的类别，productInfo的值用来存放商品类别对应的商品。**

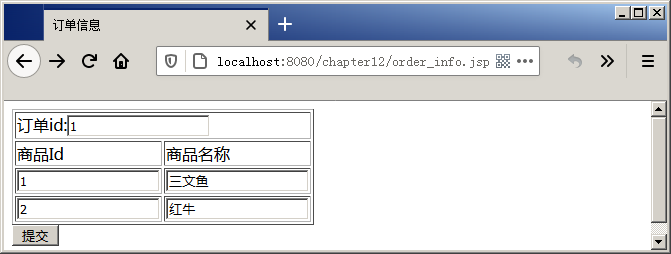
**（2）修改OrderController.java类，在OrderController类中新增getOrderInfo()方法，用于获取客户端提交的订单信息，并将获取到的订单信息打印在控制台。**

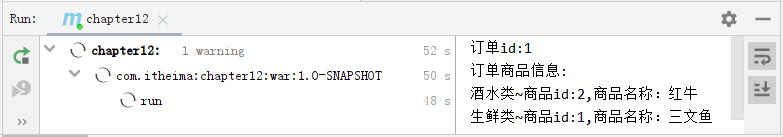


**（3）在项目的src\main\webapp目录下，创建一个订单信息页面order\_info.jsp，在order\_info.jsp中创建一个表单用于提交订单信息。表单提交时，表单数据分别封装到Order的orderId属性和商品信息属性productInfo中。**



**（4）启动chapter12项目，在浏览器中访问订单信息页面order\_info.jsp，访问地址为http://localhost:8080/chapter12/order\_info.jsp，order\_info.jsp显示效果如图所示。**

****

**（5）在order\_info.jsp显示效果图所示的页面中，单击左下角“提交”按钮，order\_info.jsp表单中的订单信息发送到服务器端的getOrderInfo()方法进行处理，控制台打印信息如图所示。**

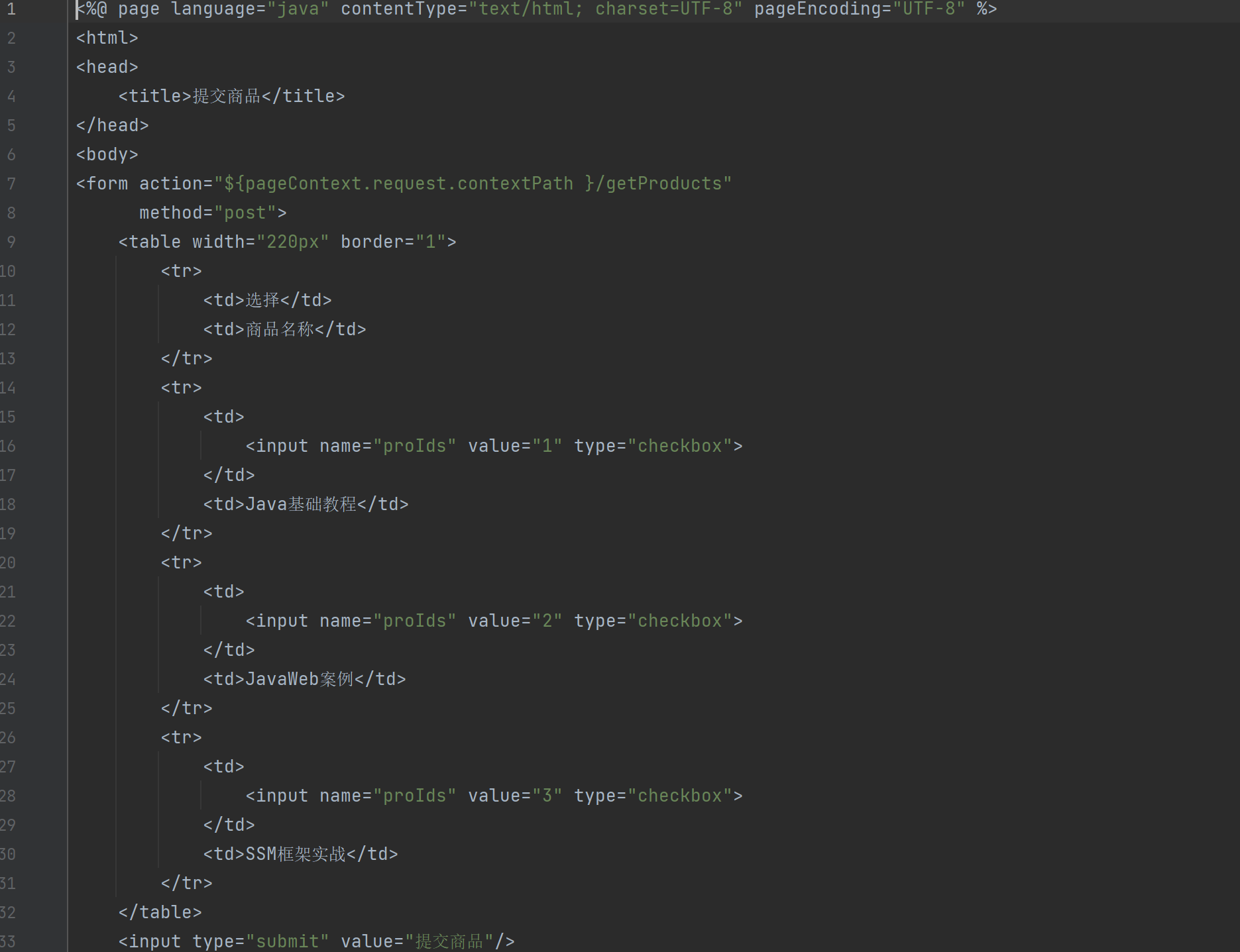
**JSON数据绑定**

**（1）通过一个异步提交商品信息案例，演示Spring MVC中的JSON数据绑定。**



**（2）在项目中导入jQuery文件。**

**（3）在项目的src\main\webapp目录下，创建一个商品信息页面product.jsp，在product.jsp中创建一个表单用于填写商品信息，表单提交时，表单发送异步请求将表单的商品信息发送到处理器。**



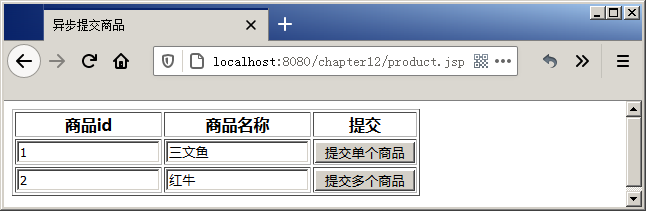
**（4）修改ProductController.java类，在ProductController类中新增getProduct()方法和getProductList()方法，分别用于获取客户端提交的单个商品信息和多个商品信息。由于客户端发送的是JSON格式的数据，此时，在处理器中无法直接使用方法形参接收数据，完成数据的自动绑定。对此，可以使用Spring MVC提供的@RequestBody注解。**



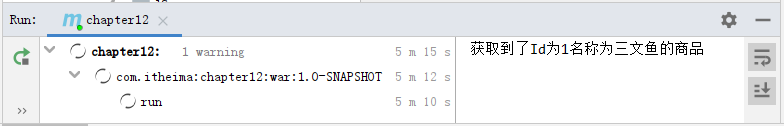
**（5）在项目的web.xml文件中配置的DispatcherServlet会拦截所有URL，导致项目中的静态资源（如css、jsp、js等）也被DispatcherServlet拦截。如果想放行静态资源，可以在Spring MVC的配置文件中进行静态资源配置。**



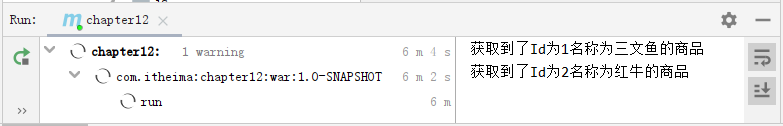
**（6）启动chapter12项目，在浏览器中访问商品信息页面product.jsp，访问地址为http://localhost:8080/chapter12/product.jsp，product.jsp的显示效果如图所示。**

****

**（7）在product.jsp的显示效果图所示的页面中，单击右侧“提交单个商品”按钮，product.jsp表单中的单个商品信息以JSON格式异步发送到服务器端getProduct()方法中。提交单个商品时控制台打印信息如图所示。**

****

**（8）在product.jsp的显示效果图所示的页面中，单击“提交多个商品”按钮，product.jsp表单中的2个商品信息以JSON格式异步发送到服务器端getProductList()方法中。提交多个商品时控制台打印信息如图所示。**

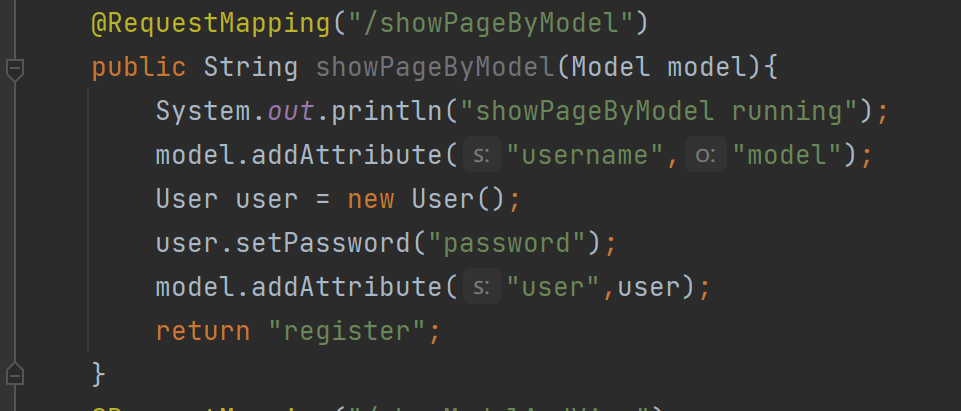
****

**返回值为void类型的页面跳转**

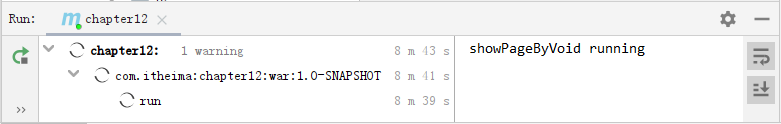
**（1）通过案例演示返回值为void类型的页面跳转，案例具体实现步骤如下。在项目的com.itheima.controller包下创建一个页面跳转类PageController，在PageController类中定义方法showPageByVoid()，用于测试Spring MVC方法返回值为void的页面跳转。**



**（2）PageController.java类中的方法showPageByVoid()处理请求。**

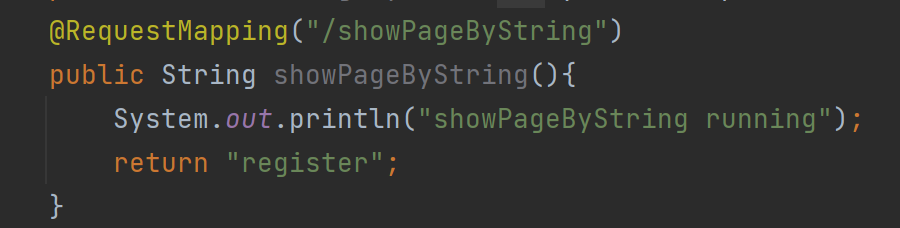


**（3）在项目webapp/WEB-INF文件夹下创建名称为pages的文件夹，将前面的User.java类移动到pages文件下。启动chapter12项目，在浏览器中访问地址http://localhost:8080/chapter12/register。访问后，控制台打印信息如图所示。**

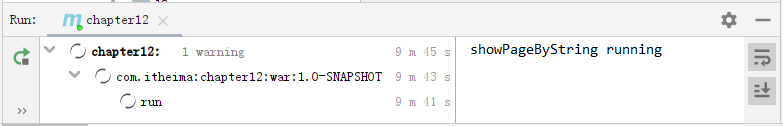
****

**返回值为String类型的页面跳转**

**（1）返回值为String类型时，不携带数据页面跳转相对比较简单，接下来通过一个案例演示返回值为String类型时，不携带数据的页面跳转，案例具体实现步骤如下。修改文件PageController.java，新增showPageByString()方法，用于测试返回值为String类型的页面跳转。**



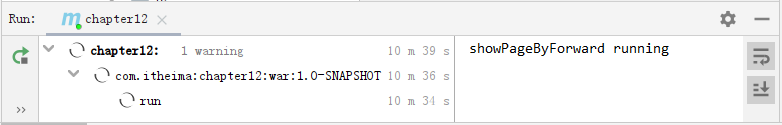
**（2）启动chapter12项目，在浏览器中访问地址http://localhost:8080/chapter12/showPageByString。访问后，控制台打印信息如图所示。**

****

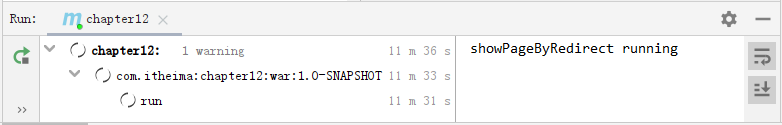
**（3）通过一个案例演示返回指定前缀的字符串的页面跳转，具体实现步骤如下。修改文件PageController.java，新增showPageByForward()方法和showPageByRedirect()方法，分别用于测试方法执行后转发和重定向的页面跳转。**



**（4）启动chapter12项目，在浏览器中访问访问地址http://localhost:8080/chapter12/showPageByForward。访问后，控制台打印信息如图所示。**

****

**（5）在浏览器中访问地址http://localhost:8080/chapter12/showPageByRedirect。访问后，控制台打印信息如图所示。**

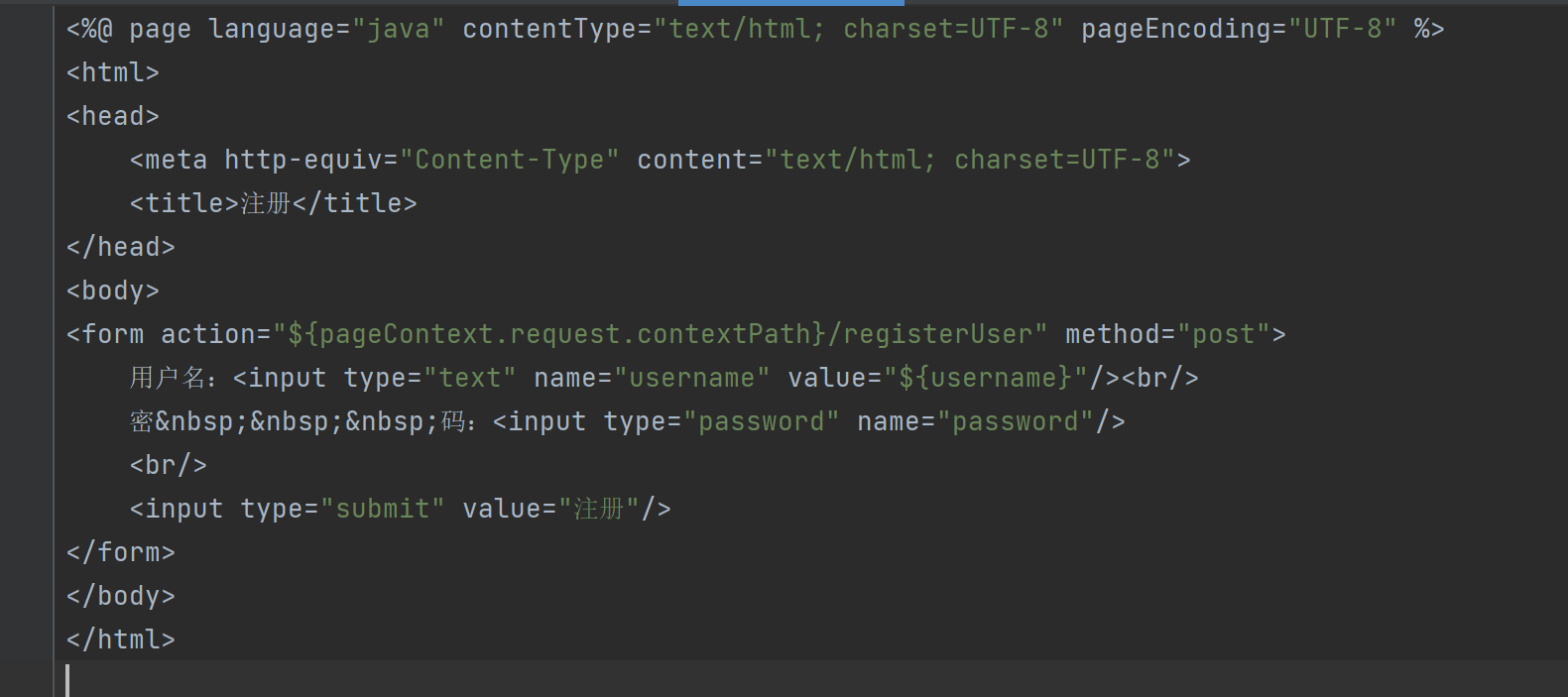
****

**返回值为String类型的页面跳转2**

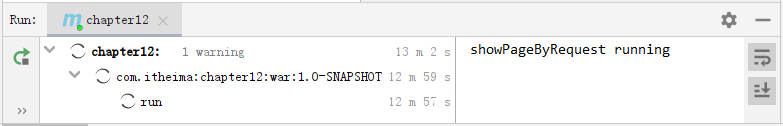
**（1）通过一个案例演示携带数据的页面转发，该案例使用HttpServletRequest类型形参和Model类型形参进行数据传递，案例具体实现步骤如下。修改文件PageController.java，新增showPageByRequest()方法和showPageByModel()方法，showPageByRequest()方法使用HttpServletRequest传递数据，showPageByModel()方法使用Model传递数据，两个方法都使用字符串指定跳转的页面。**



**（2）修改文件User.java，在register.jsp的表单中添加value属性，用于接收转发传递过来的数据。**



**（3）启动chapter12项目，在浏览器中访问地址http://localhost:8080/chapter12/showPageByRequest。访问后，控制台打印信息如图所示。**

****

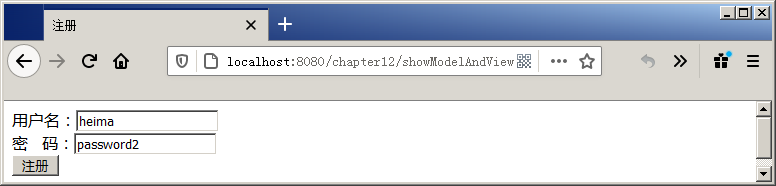
**（4）在浏览器中访问地址http://localhost:8080/chapter12/showPageByModel。访问后，控制台打印信息如图所示。**

****

**返回值为ModelAndView类型的页面跳转**

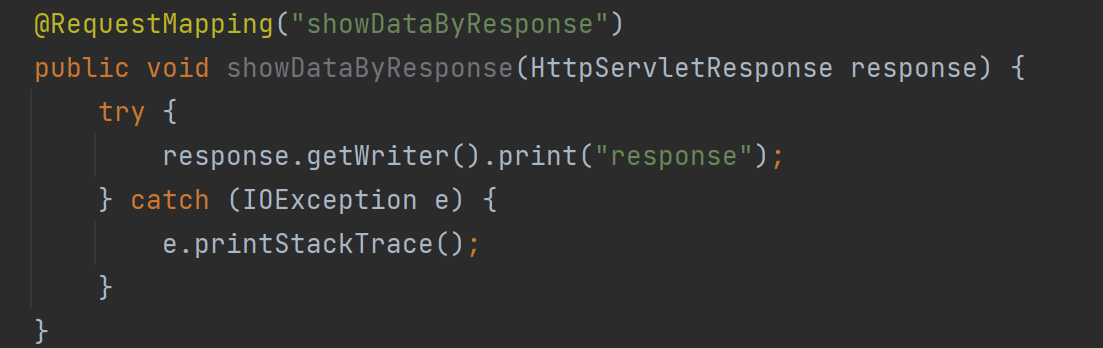
**（1）通过一个案例演示返回值为ModelAndView类型的页面跳转，案例具体实现步骤如下。** 

**（2）启动chapter12项目，在浏览器中访问地址http://localhost:8080/chapter12/showModelAndView。访问后，浏览器页面进行跳转，跳转的页面如图所示。**

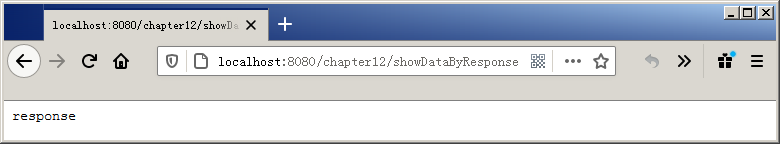
****

**普通字符串的回写**

**（1）通过HttpServletResponse输出数据的案例，演示普通字符串的回写，案例具体实现步骤如下。**

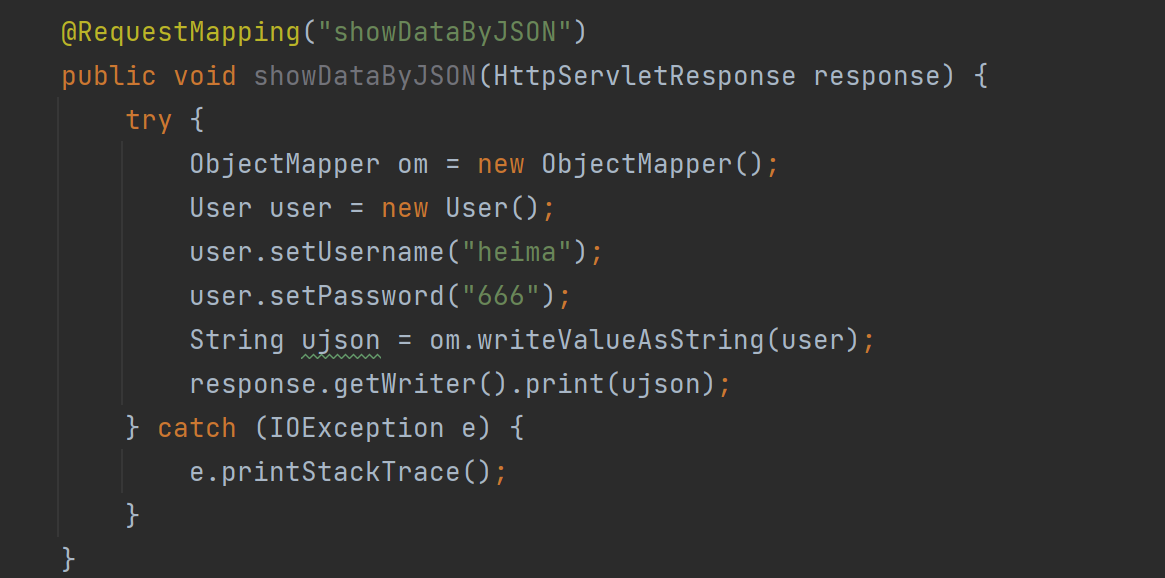


**（2）启动chapter12项目，在浏览器中访问地址http://localhost:8080/chapter12/showDataByResponse。访问后，浏览器页面不跳转，页面显示效果如图所示。**

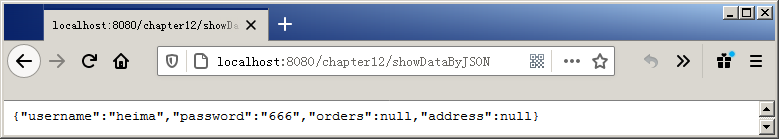
****

**JSON数据的回写**

**（1）修改文件DataController.java，在DataController类中新增showDataByJSON()方法，用于将对象转换成JSON数据并写入输出流中完成回写。**

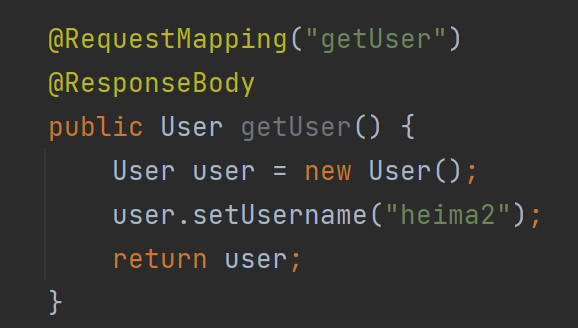


**（2）启动chapter12项目，在浏览器中访问地址http://localhost:8080/chapter12/showDataByJSON。访问后，页面显示效果如图所示。**

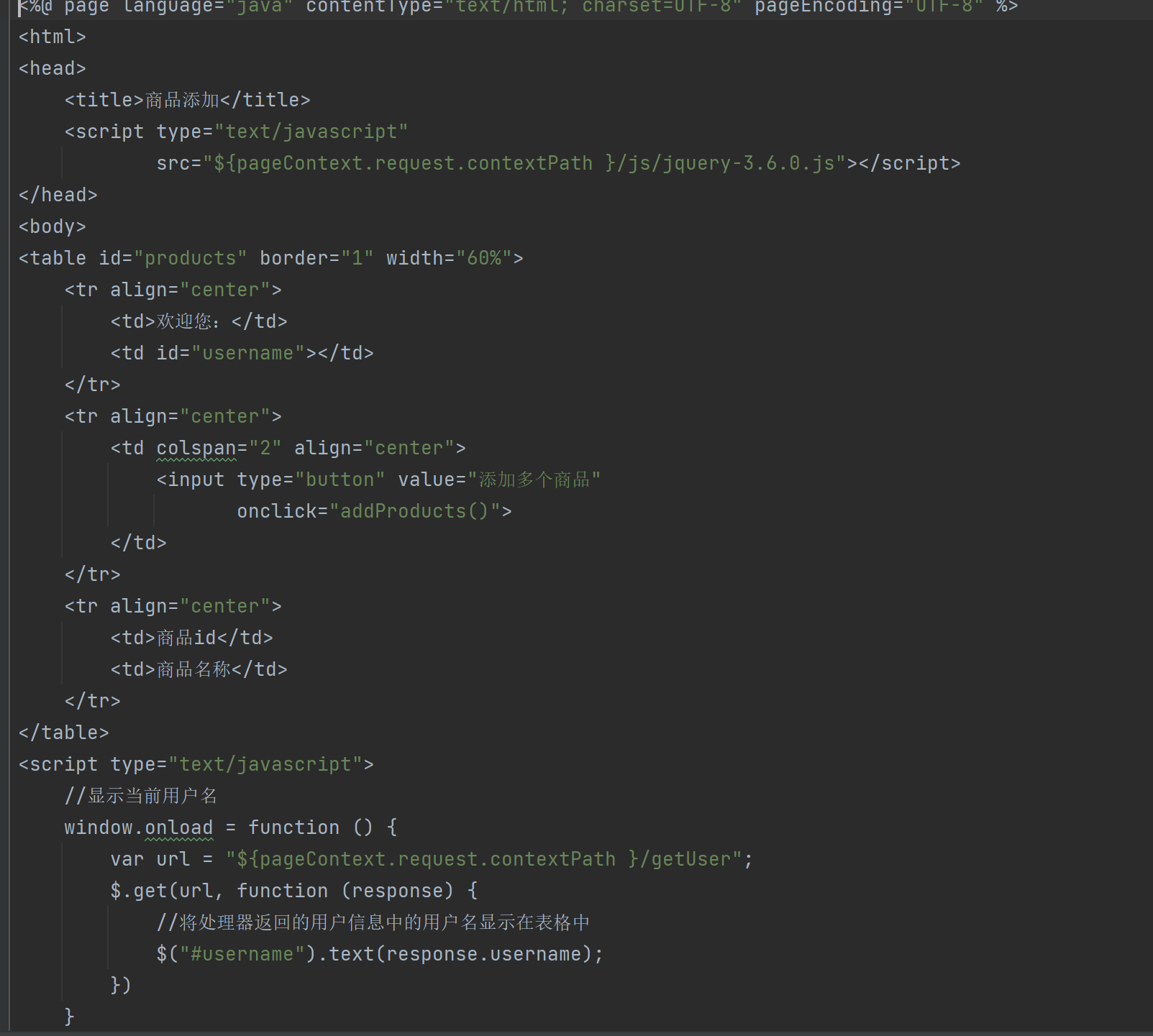
****

**JSON数据的回写2**

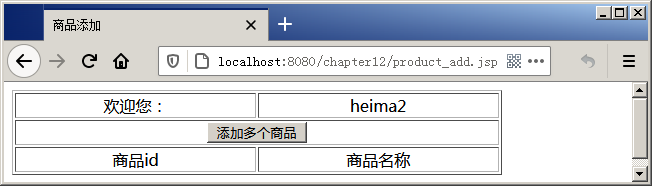
**（1）修改文件DataController.java，在DataController类中新增getUser()方法，用于返回JSON类型的User信息；新增addProducts()方法用于返回JSON类型的Product列表信息。**



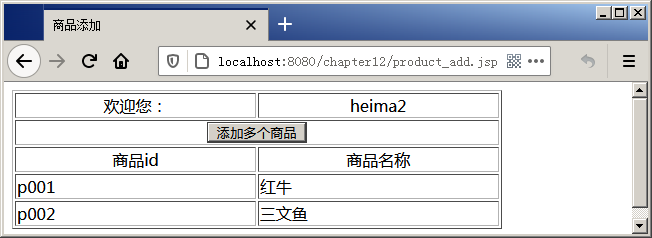
**（2）在项目的src\main\webapp目录下，创建一个商品添加页面product\_add.jsp，在product\_add.jsp中创建一个表格，用于显示用户信息和添加商品信息。**



**（3）启动chapter12项目，在浏览器中访问商品添加页面product\_add.jsp，访问地址为http://localhost:8080/chapter12/product\_add.jsp，product\_add.jsp页面显示效果如图所示。**

****

**（4）单击product\_add.jsp页面显示效果图所示的“添加多个商品”按钮，product\_add.jsp页面显示效果如图所示。**

****