springmvc

**第二天**

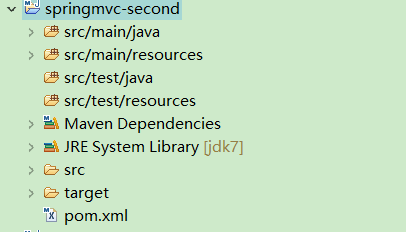
# 学习目标

* 掌握springmvc高级参数绑定
* 掌握@RequestMapping注解作用
* 掌握Controller方法返回值
* 掌握springmvc的统一异常处理
* 掌握图片上传
* 掌握springmvc与json交互
* 掌握springmvc支持的RESTful风格
* 了解springmvc拦截器

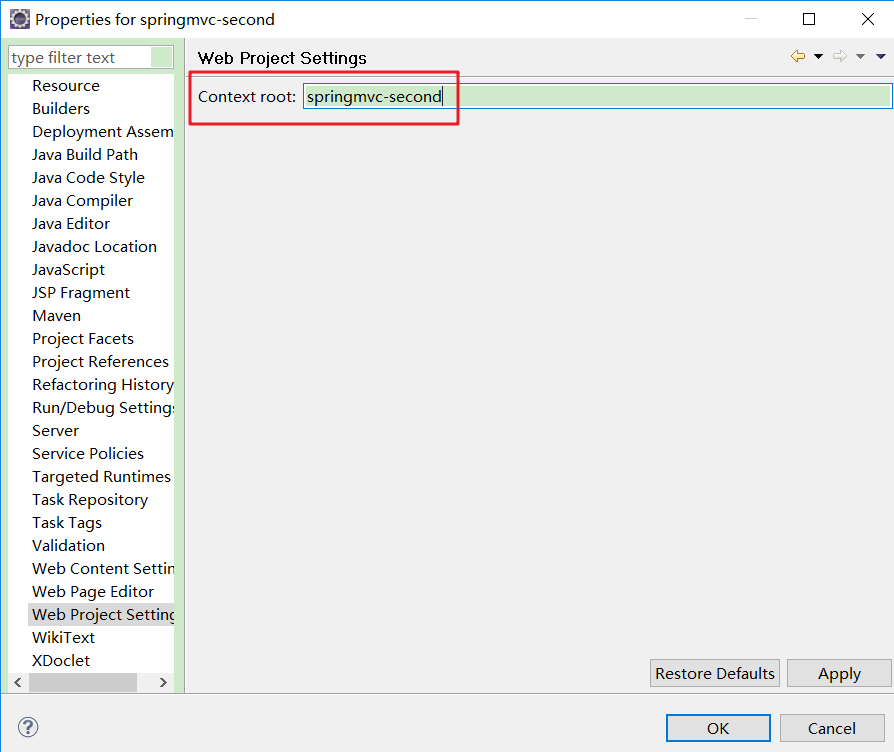
# 高级参数绑定

## 准备环境

### 复制ssm项目，修改名称为springmvc-second

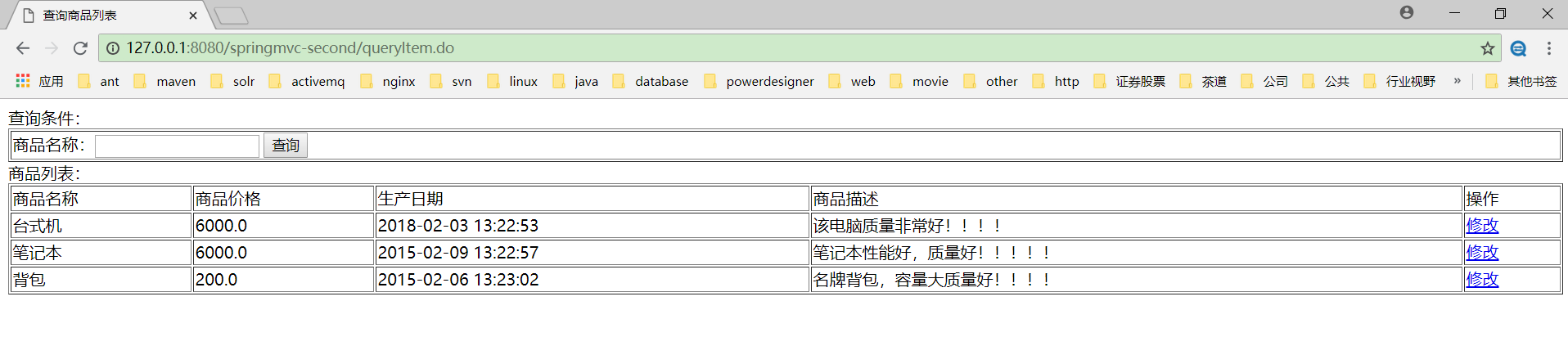


### 修改项目的上下文路径



### 修改pom.xml文件中的项目信息

|  |
| --- |
| <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>cn.itheima</groupId>  <artifactId>springmvc-second</artifactId>  <version>0.0.1-SNAPSHOT</version>  <packaging>war</packaging>  <name>springmvc-second Maven Webapp</name>  <url>http://maven.apache.org</url>  <properties>  <!-- spring版本号 -->  <spring.version>4.3.8.RELEASE</spring.version>  <!-- aspectj版本号 -->  <aspectj.version>1.6.12</aspectj.version>  <!-- jstl标签版本 -->  <jstl.version>1.2</jstl.version>  <!-- mybatis版本号 -->  <mybatis.version>3.4.5</mybatis.version>  <!-- mybatis-spring整合包版本 -->  <mybatis.spring.version>1.3.1</mybatis.spring.version>  <!-- mysql版本号 -->  <mysql.version>5.1.30</mysql.version>  <!-- dbcp数据源连接池jar包 -->  <dbcp.version>1.2.2</dbcp.version>  <!-- log4j日志包版本 -->  <slf4j.version>1.7.7</slf4j.version>  <log4j.version>1.2.17</log4j.version>  <!-- commons-lang版本 -->  <commons.lang.version>2.6</commons.lang.version>  </properties>  <dependencies>  <!-- springmvc依赖包 -->  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-core</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-tx</artifactId>  <version>${spring.version}</version>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-jdbc</artifactId>  <version>${spring.version}</version>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-webmvc</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-context-support</artifactId>  <version>${spring.version}</version>  </dependency>  <dependency>  <groupId>org.aspectj</groupId>  <artifactId>aspectjweaver</artifactId>  <version>${aspectj.version}</version>  </dependency>  <!-- JSTL标签类 -->  <dependency>  <groupId>jstl</groupId>  <artifactId>jstl</artifactId>  <version>${jstl.version}</version>  </dependency>  <!-- mybatis核心包 -->  <dependency>  <groupId>org.mybatis</groupId>  <artifactId>mybatis</artifactId>  <version>${mybatis.version}</version>  </dependency>  <!-- mybatis-spring整合包 -->  <dependency>  <groupId>org.mybatis</groupId>  <artifactId>mybatis-spring</artifactId>  <version>${mybatis.spring.version}</version>  </dependency>  <!-- 导入Mysql数据库链接jar包 -->  <dependency>  <groupId>mysql</groupId>  <artifactId>mysql-connector-java</artifactId>  <version>${mysql.version}</version>  </dependency>  <!-- 导入dbcp数据源连接池jar包 -->  <dependency>  <groupId>commons-dbcp</groupId>  <artifactId>commons-dbcp</artifactId>  <version>${dbcp.version}</version>  </dependency>  <!-- 日志文件管理包 -->  <dependency>  <groupId>log4j</groupId>  <artifactId>log4j</artifactId>  <version>${log4j.version}</version>  </dependency>  <dependency>  <groupId>org.slf4j</groupId>  <artifactId>slf4j-api</artifactId>  <version>${slf4j.version}</version>  </dependency>  <dependency>  <groupId>org.slf4j</groupId>  <artifactId>slf4j-log4j12</artifactId>  <version>${slf4j.version}</version>  </dependency>  <dependency>  <groupId>commons-lang</groupId>  <artifactId>commons-lang</artifactId>  <version>${commons.lang.version}</version>  </dependency>  <!-- jsp依赖包，只在编译时需要 -->  <dependency>  <groupId>javax.servlet</groupId>  <artifactId>jsp-api</artifactId>  <version>2.0</version>  <scope>provided</scope>  </dependency>  </dependencies>  <build>  <finalName>ssm</finalName>  <plugins>  <plugin>  <groupId>org.apache.tomcat.maven</groupId>  <artifactId>tomcat7-maven-plugin</artifactId>  <version>2.1</version>  <configuration>  <!-- tomcat 的端口号 -->  <port>8080</port>  <!-- 访问应用的路径 -->  <path>/springmvc-second</path>  <!-- URL按UTF-8进行编码，解决中文参数乱码 -->  <uriEncoding>UTF-8</uriEncoding>  <!-- tomcat名称 -->  <server>tomcat7</server>  </configuration>  </plugin>  </plugins>  </build>  </project> |



## 数组参数类型

### 需求

实现商品数据的批量删除。

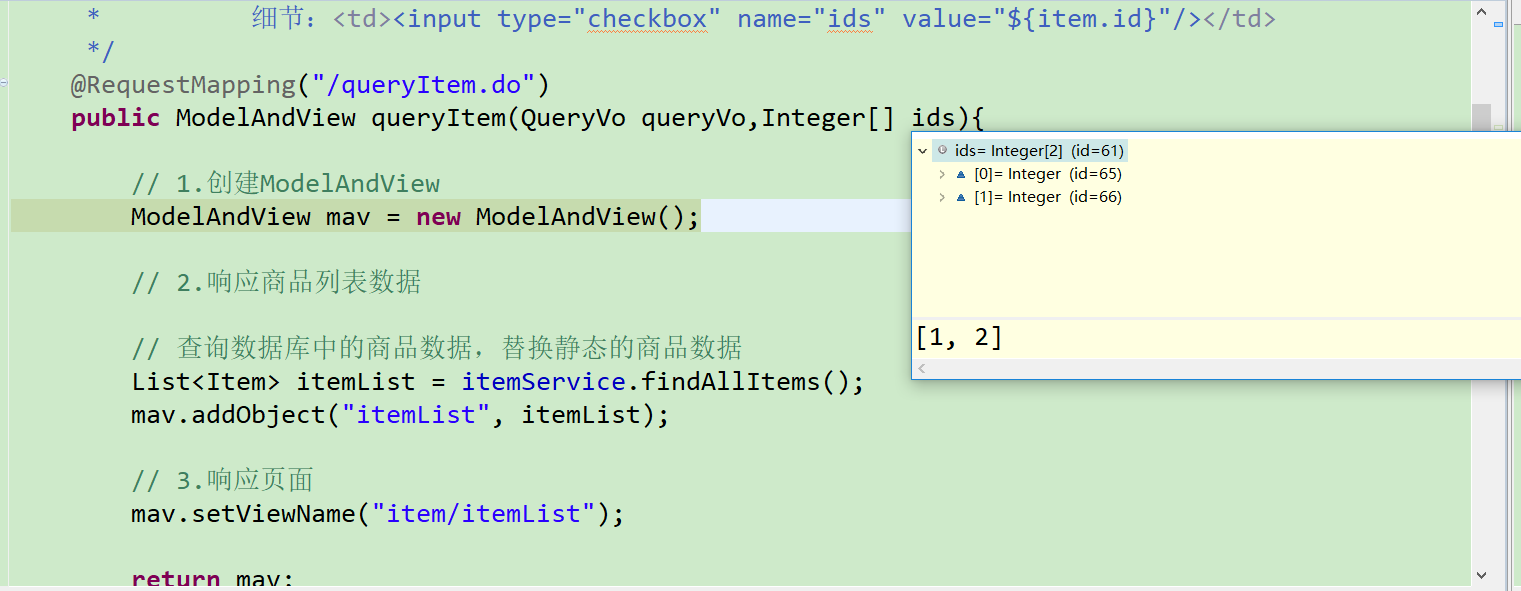
### 需求实现

#### 修改商品列表页面，增加复选框

|  |
| --- |
| 商品列表：<input type=*"submit"* name=*"批量删除商品"* value=*"批量删除"*/>  <table width=*"100%"* border=*1*>  <tr>  <td>商品id</td>  <td>商品名称</td>  <td>商品价格</td>  <td>生产日期</td>  <td>商品描述</td>  <td>操作</td>  </tr>  <c:forEach items=*"*${itemList}*"* var=*"item"*>  <tr>  <td><input type=*"checkbox"* name=*"ids"* value=*"*${item.id}*"*/></td>  <td>${item.name }</td>  <td>${item.price }</td>  <td><fmt:formatDate value=*"*${item.createtime}*"*  pattern=*"yyyy-MM-dd HH:mm:ss"* /></td>  <td>${item.detail }</td>  <td><a  href=*"*${pageContext.request.contextPath }*/queryItemById.do?id=*${item.id}*"*>修改</a></td>  </tr>  </c:forEach>  </table> |

#### 在queryItem方法中，增加形参数组

|  |
| --- |
| /\*\*  \* 学习高级参数绑定：  \* 1.数组参数类型：增加形参ids  \* 细节：<td><input type="checkbox" name="ids" value="${item.id}"/></td>  \*/  @RequestMapping("/queryItem.do")  **public** ModelAndView queryItem(QueryVo queryVo,Integer[] ids){  // 1.创建ModelAndView  ModelAndView mav = **new** ModelAndView();    // 2.响应商品列表数据    // 查询数据库中的商品数据，替换静态的商品数据  List<Item> itemList = itemService.findAllItems();  mav.addObject("itemList", itemList);    // 3.响应页面  mav.setViewName("item/itemList");    **return** mav;    } |



## List参数类型

### 需求

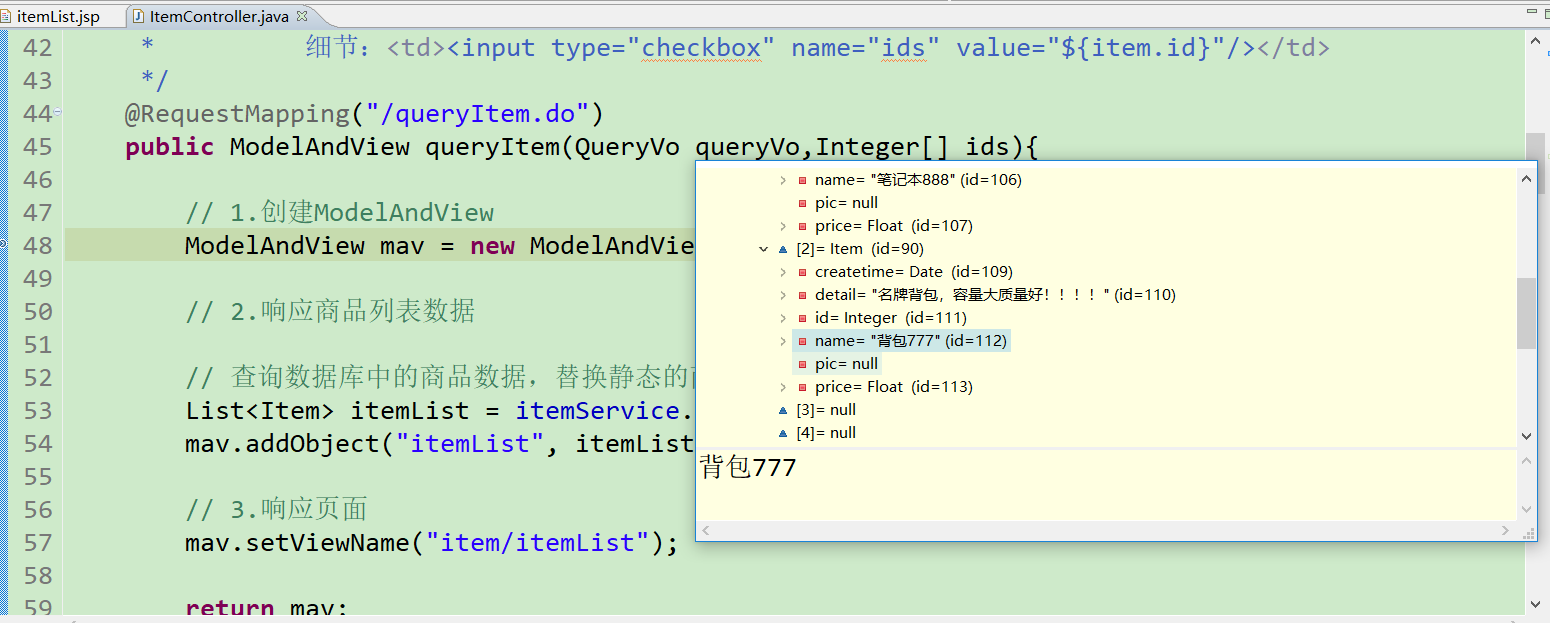
批量修改商品数据

### 需求实现

#### 修改商品列表页面，增加商品的批量修改功能

|  |
| --- |
| <p/><p/>  <!-- 批量修改商品 -->  <form action=*"*${pageContext.request.contextPath}*/queryItem.do"* method=*"post"*>  商品列表批量修改：<input type=*"submit"* value=*"批量修改"*/>  <table width=*"100%"* border=*1*>  <tr>  <td>商品名称</td>  <td>商品价格</td>  <td>生产日期</td>  <td>商品描述</td>  </tr>  <c:forEach items=*"*${itemList}*"* var=*"item"* varStatus=*"vs"*>  <tr>  <!--  itemList：要绑定的集合list属性  itemList[0]:要绑定商品集合中的的第一个商品对象  itemList[0].id：要绑定的第一个商品对象的id属性    varStatus：当前遍历的状态  vs.index：当前遍历的索引  -->  <input type=*"hidden"* name=*"itemList[*${vs.index}*].id"* value=*"*${item.id}*"*/>  <td><input type=*"text"* name=*"itemList[*${vs.index}*].name"* value=*"*${item.name }*"*/></td>  <td><input type=*"text"* name=*"itemList[*${vs.index}*].price"* value=*"*${item.price }*"*/></td>  <td>  <input type=*"text"* name=*"itemList[*${vs.index}*].createtime"* value=*'*<fmt:formatDate value="${item.createtime}"  pattern=*"yyyy-MM-dd HH:mm:ss"* />*'*/>  </td>  <td><input type=*"text"* name=*"itemList[*${vs.index}*].detail"* value=*"*${item.detail }*"*/></td>  </tr>  </c:forEach>  </table>  </form> |

#### 修改QueryVo包装类型，增加商品集合属性list



注意事项：使用List参数类型绑定，List需要作为pojo的属性出现，不能直接在方法的形参中使用List，否则不能完成绑定。

# @RequestMapping注解

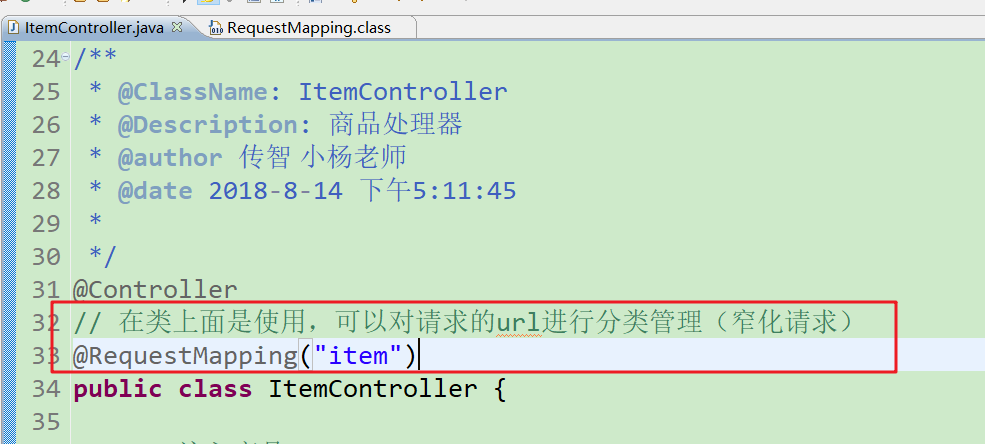
## 作用一：配置请求的url

|  |
| --- |
| /\*\*  \* 学习@RequestMapping注解  \* 作用一：配置请求的url  \* 属性：  \* value：是一个数组，可以配置多个url  \* 细节："/"和".do"可以省略  \*/  //@RequestMapping(value={"/queryItem.do","/queryItemAAA.do"})  @RequestMapping(value={"queryItem"})  **public** ModelAndView queryItem(QueryVo queryVo,Integer[] ids){  // 1.创建ModelAndView  ModelAndView mav = **new** ModelAndView();    // 2.响应商品列表数据    // 查询数据库中的商品数据，替换静态的商品数据  List<Item> itemList = itemService.findAllItems();  mav.addObject("itemList", itemList);    // 3.响应页面  mav.setViewName("item/itemList");    **return** mav;    } |

## 作用二：限制http请求方法

|  |
| --- |
| /\*\*  \* 学习@RequestMapping注解  \* 作用一：配置请求的url  \* 属性：  \* value：是一个数组，可以配置多个url  \* 细节："/"和".do"可以省略  \* 作用二：限制http请求方法  \* 属性：  \* method：限制http请求方法  \* 细节：  \* RequestMethod.POST：值允许post请求  \* RequestMethod.POST,RequestMethod.GET：只允许post/get请求  \*/  //@RequestMapping(value={"/queryItem.do","/queryItemAAA.do"})  //@RequestMapping(value={"queryItem"})  //@RequestMapping(value={"queryItem"},method={RequestMethod.POST})  @RequestMapping(value={"queryItem"},method={RequestMethod.*POST*,RequestMethod.*GET*})  **public** ModelAndView queryItem(QueryVo queryVo,Integer[] ids){  // 1.创建ModelAndView  ModelAndView mav = **new** ModelAndView();    // 2.响应商品列表数据    // 查询数据库中的商品数据，替换静态的商品数据  List<Item> itemList = itemService.findAllItems();  mav.addObject("itemList", itemList);    // 3.响应页面  mav.setViewName("item/itemList");    **return** mav;    } |

## 作用三：在类上面使用，对请求的url进行分类管理（窄化请求）

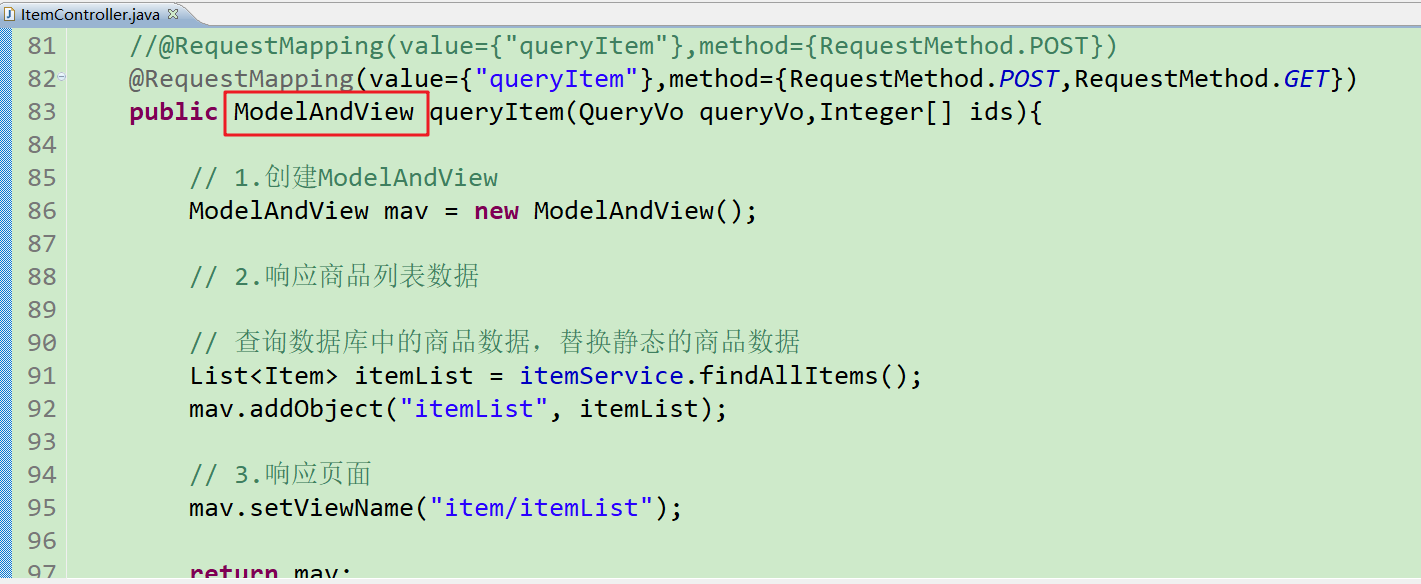


# Controller方法返回值

返回值类型：ModelAndView、Void、String

## 返回ModelAndView

作用：封装响应的模型数据；封装响应的视图。

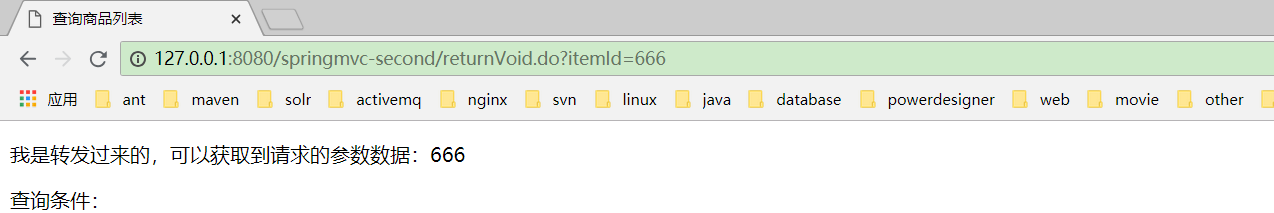


## 返回Void

说明：返回void，使用request转发；使用response重定向。

### request转发

|  |
| --- |
| /\*\*  \* Controller方法返回void学习  \*/  @RequestMapping("/returnVoid.do")  **public** **void** controllerReturnVoid(HttpServletRequest request,  HttpServletResponse response){    /\*\*  \* 1.request转发：  \* 转发是一次请求，可以获取到request中的数据  \* 转发浏览器的地址栏不会发生改变  \* 转发只能在项目内部实现  \*/  **try** {  request.getRequestDispatcher("queryItem.do").  forward(request, response);  } **catch** (Exception e) {  e.printStackTrace();  }    } |



### response重定向

|  |
| --- |
| /\*\*  \* Controller方法返回void学习  \*/  @RequestMapping("/returnVoid.do")  **public** **void** controllerReturnVoid(HttpServletRequest request,  HttpServletResponse response){    /\*\*  \* 1.request转发：  \* 转发是一次请求，可以获取到request中的数据  \* 转发浏览器的地址栏不会发生改变  \* 转发只能在项目内部实现  \*/  /\* try {  request.getRequestDispatcher("queryItem.do").  forward(request, response);  } catch (Exception e) {  e.printStackTrace();  }\*/    /\*\*  \* 2.response重定向：  \* 两次请求，不能获取到旧的request中的数据  \* 重定向浏览器地址栏改变成目标url  \* 重定向可以在项目内部实现；也可以重定向到别的项目  \*/  **try** {  // 1.项目内部重定向  //response.sendRedirect("queryItem.do");    // 2.重定向到别的项目  response.sendRedirect("https://www.cnblogs.com");  } **catch** (IOException e) {  e.printStackTrace();  }    } |

### response响应数据

|  |
| --- |
| /\*\*  \* Controller方法返回void学习  \*/  @RequestMapping("/returnVoid.do")  **public** **void** controllerReturnVoid(HttpServletRequest request,  HttpServletResponse response){    /\*\*  \* 1.request转发：  \* 转发是一次请求，可以获取到request中的数据  \* 转发浏览器的地址栏不会发生改变  \* 转发只能在项目内部实现  \*/  /\* try {  request.getRequestDispatcher("queryItem.do").  forward(request, response);  } catch (Exception e) {  e.printStackTrace();  }\*/    /\*\*  \* 2.response重定向：  \* 两次请求，不能获取到旧的request中的数据  \* 重定向浏览器地址栏改变成目标url  \* 重定向可以在项目内部实现；也可以重定向到别的项目  \*/  /\*try {  // 1.项目内部重定向  //response.sendRedirect("queryItem.do");    // 2.重定向到别的项目  response.sendRedirect("https://www.cnblogs.com");  } catch (IOException e) {  e.printStackTrace();  }\*/    /\*\*  \* 3.response响应数据  \*/  **try** {  response.setCharacterEncoding("utf-8");  response.setContentType("application/json;charset=utf-8");  response.getWriter().write("json数据");  } **catch** (IOException e) {  e.printStackTrace();  }    } |

## 返回String

### 返回响应的视图



### forward转发

|  |
| --- |
| /\*\*  \* controller方法返回String  \*/  @RequestMapping("/returnString.do")  **public** String controllerReturnString(){    /\*\*  \* 1.forward转发  \* 转发格式：forward+":"+目标url  \* 细节：请求的url后缀是不能省略（forward:queryItem）  \*/  **return** "forward:queryItem.do";    } |

### redirect重定向

|  |
| --- |
| /\*\*  \* controller方法返回String  \*/  @RequestMapping("/returnString.do")  **public** String controllerReturnString(){    /\*\*  \* 1.forward转发  \* 转发格式：forward+":"+目标url  \* 细节：请求的url后缀是不能省略（forward:queryItem）  \*/  //return "forward:queryItem.do";    /\*\*  \* 2.redirect重定向  \* 重定向的格式：redirect+":"+目标url  \*  \* 细节：请求的url后缀是不能省略（redirect:queryItem）  \*/  **return** "redirect:queryItem.do";    } |

# 统一异常处理

## 异常介绍

异常，就是不正常。异常可以分类为预期的异常和运行时的异常。预期的异常可以通过捕获进行处理（try...catch...）。运行时的异常只能通过规范代码的编写和增加测试来减少。

## 异常处理思路

请求流程：客户端（浏览器）->前端控制器（DispatcherServlet）->表现层（controller）->业务层（service）->持久层（dao）。

处理思路：从dao层开始，每一层发生异常都向上一层抛出，一直抛到前端控制器，前端控制器需要调用异常处理器组件包装异常，提供一个友好的异常页面。

客户端（浏览器）

前端控制器：DispatcherServlet

表现层：controller

业务层：service

持久层：dao

请求流程

从dao开始，每一层发生异常都向上一层抛出，一直抛到前端控制器

异常处理：

HandlerExceptionResolver

前端控制器请求异常处理器组件包装异常

## 异常处理案例演示

### 自定义系统异常

|  |
| --- |
| **package** cn.itheima.ssm.exception;  /\*\*  \* **@ClassName**: SsmException  \* **@Description**: 自定义系统异常  \* **@author** 传智 小杨老师  \* **@date** 2018-8-16 下午4:45:45  \*  \*/  **public** **class** SsmException **extends** Exception {    //异常消息  **private** String message;    **public** SsmException(){    }  /\*\*  \* **@param** message  \*/  **public** SsmException(String message) {  **super**();  **this**.message = message;  }  /\*\*  \* **@return** the message  \*/  **public** String getMessage() {  **return** message;  }  /\*\*  \* **@param** message the message to set  \*/  **public** **void** setMessage(String message) {  **this**.message = message;  }      } |

### 自定义异常处理器

说明：自定义异常处理器，需要实现接口（HandlerExceptionResolver）。

|  |
| --- |
| **package** cn.itheima.ssm.exception;  **import** javax.servlet.http.HttpServletRequest;  **import** javax.servlet.http.HttpServletResponse;  **import** org.springframework.web.servlet.HandlerExceptionResolver;  **import** org.springframework.web.servlet.ModelAndView;  /\*\*  \* **@ClassName**: SsmExceptionResolver  \* **@Description**: 自定义异常处理器  \* **@author** 传智 小杨老师  \* **@date** 2018-8-16 下午4:48:01  \*  \*/  **public** **class** SsmExceptionResolver **implements** HandlerExceptionResolver {  /\*\*  \* 实现异常处理逻辑  \* 参数：  \* request：请求对象  \* response：响应对象  \* handler：当前抛出异常的controller方法对象  \* ex：异常对象  \* 返回值：  \* ModelAndView：包装异常页面  \*/  **public** ModelAndView resolveException(HttpServletRequest request,  HttpServletResponse response, Object handler, Exception ex) {    // 1.判断是否是系统异常  SsmException ssme = **null**;  **if**(ex **instanceof** SsmException){  ssme = (SsmException)ex;  }**else**{  // 未知异常  ssme = **new** SsmException("未知异常！");  }    // 2.包装异常页面  ModelAndView mav = **new** ModelAndView();  mav.addObject("message", ssme.getMessage());  mav.setViewName("error/error");    **return** mav;  }  } |

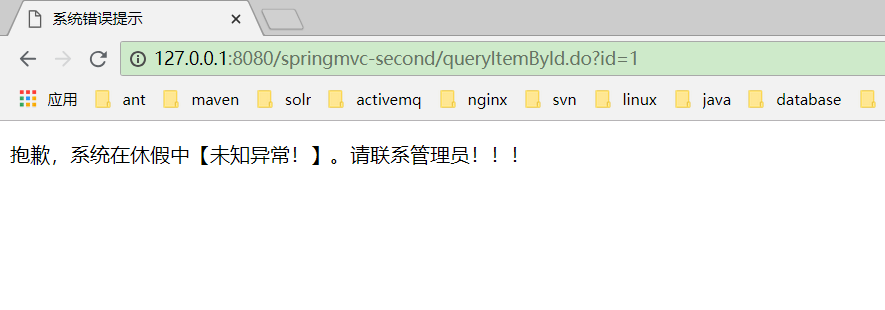
### 配置异常处理器

说明：在springmvc.xml。

|  |
| --- |
| <?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:p=*"http://www.springframework.org/schema/p"*  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-4.0.xsd*  *http://www.springframework.org/schema/mvc*  *http://www.springframework.org/schema/mvc/spring-mvc-4.0.xsd*  *http://www.springframework.org/schema/context*  *http://www.springframework.org/schema/context/spring-context-4.0.xsd"*>    <!-- 配置包扫描controller -->  <context:component-scan base-package=*"cn.itheima.ssm.controller"*/>    <!-- 注解驱动的方式配置处理器映射器和处理器适配器 -->  <mvc:annotation-driven conversion-service=*"conversionService"*/>    <!-- 配置转换器conversionService -->  <bean id=*"conversionService"*  class=*"org.springframework.format.support.FormattingConversionServiceFactoryBean"*>  <property name=*"converters"*>  <set>  <bean class=*"cn.itheima.ssm.converter.DateConterver"*/>  </set>  </property>  </bean>    <!-- 配置视图解析器 -->  <bean class=*"org.springframework.web.servlet.view.InternalResourceViewResolver"*>  <!-- 配置视图的公共目录路径（前缀） -->  <property name=*"prefix"* value=*"/WEB-INF/jsp/"*></property>  <!--配置视图的扩展名称（后缀） -->  <property name=*"suffix"* value=*".jsp"*></property>  </bean>    <!-- 配置异常处理器 -->  <bean class=*"cn.itheima.ssm.exception.SsmExceptionResolver"*/>  </beans> |

### 测试异常

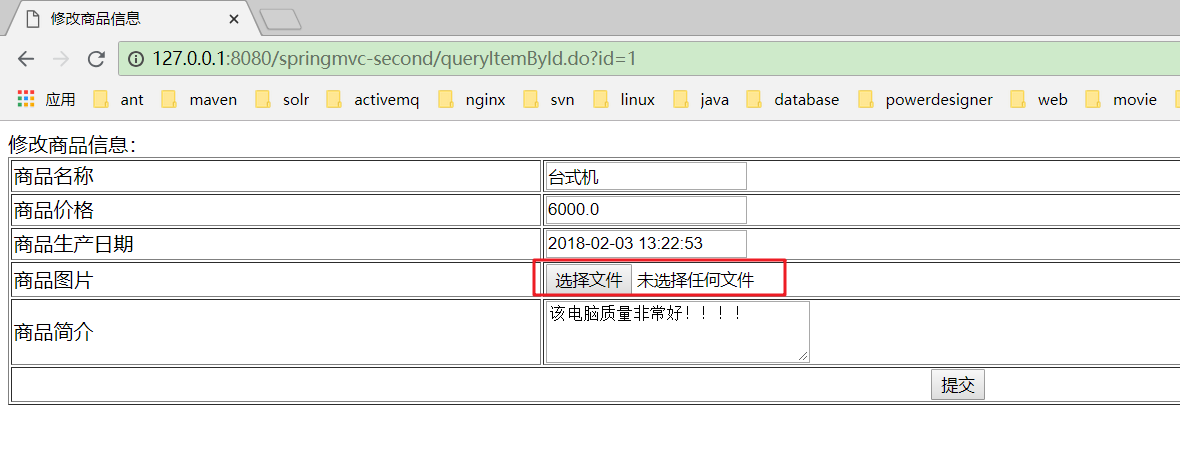
|  |
| --- |
| /\*\*  \* 使用简单类型Integer接收请求的商品Id  \* http://127.0.0.1:8080/ssm/queryItemById.do?id=1  \* **@throws** SsmException  \*/  @RequestMapping("/queryItemById.do")  **public** String queryItemById(Model model,Integer id) **throws** SsmException{    // 1.查询商品数据  Item item = itemService.findItemById(id);    // 增加异常测试=====================start  **if**(item == **null**){  **throw** **new** SsmException("商品不存在！");  }    // 运行时异常  //int i=2/0;    // 增加异常测试=====================end    // 2.响应数据  model.addAttribute("item", item);    **return** "item/itemEdit";  } |



# 文件上传

## 需求

实现商品图片的上传。

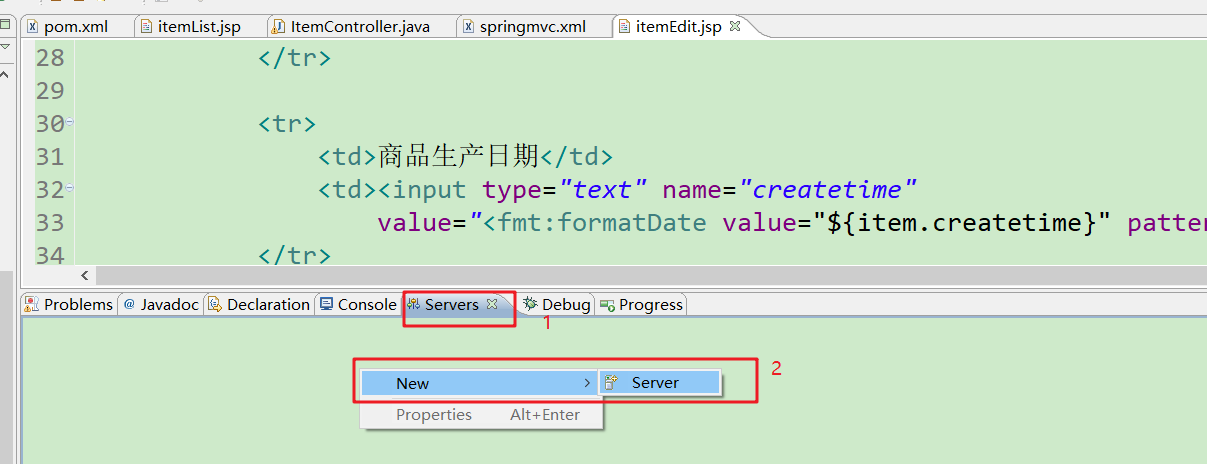


## 需求实现

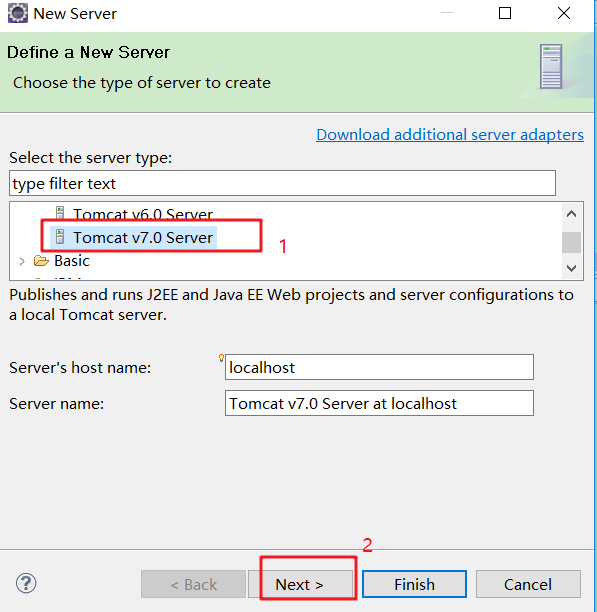
### 配置虚拟目录（搭建一台文件服务器）

#### eclipse集成tomcat

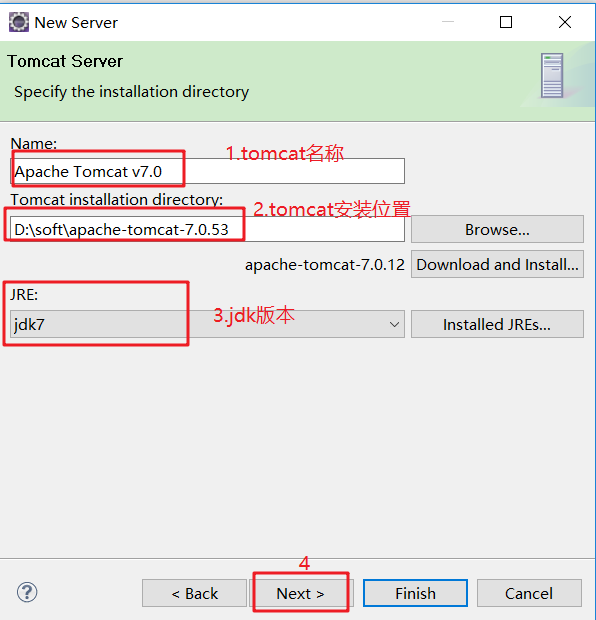
步骤一：



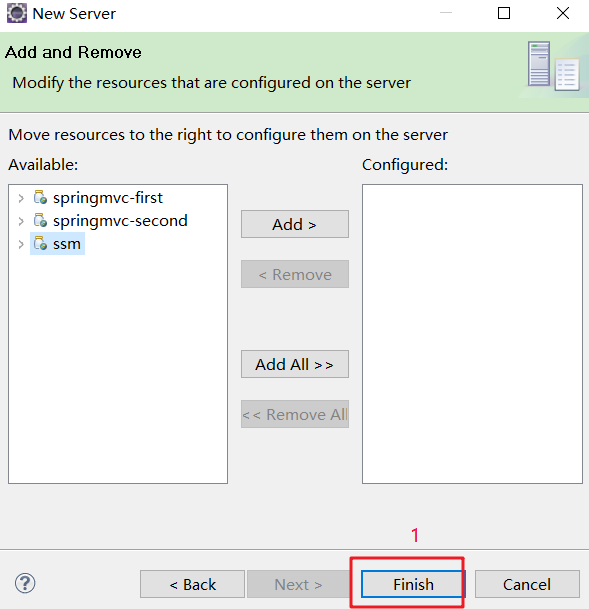
步骤二：



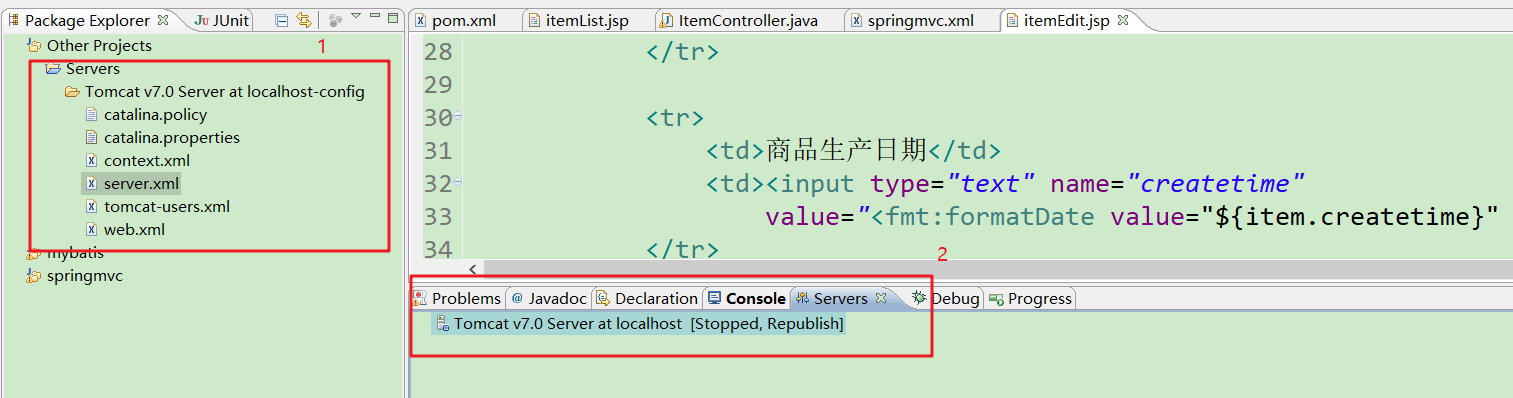
步骤三：



步骤四：

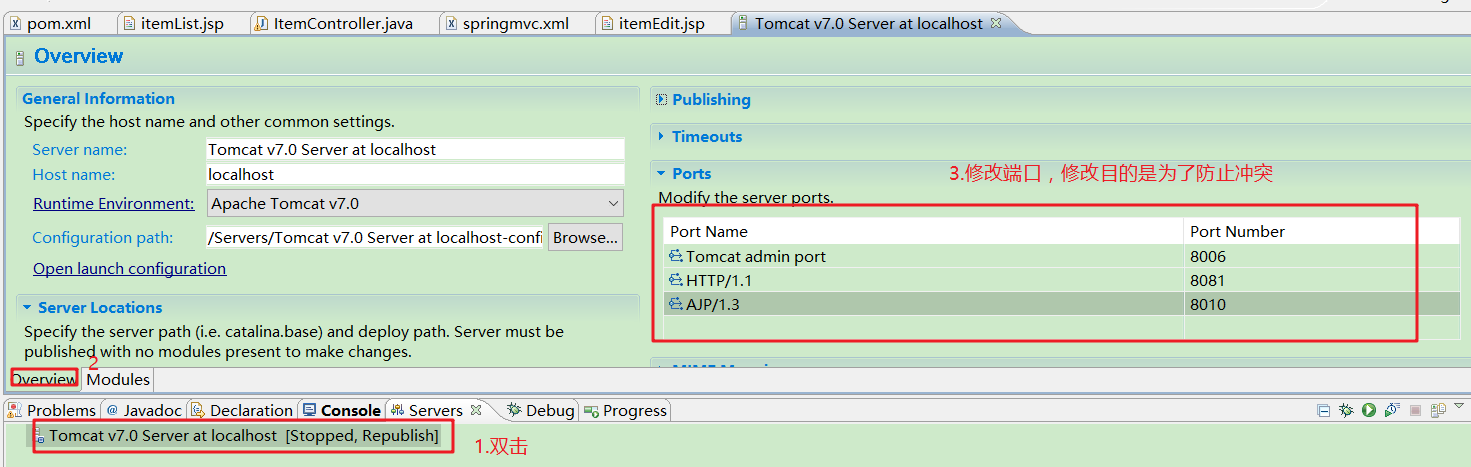


步骤五：

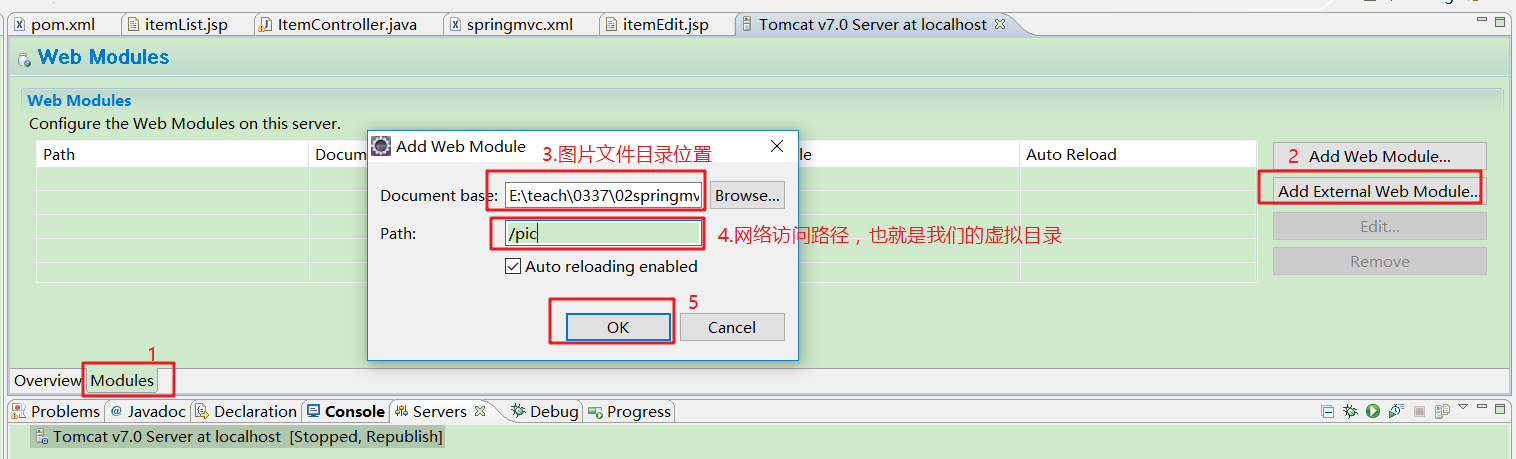


#### 虚拟目录配置

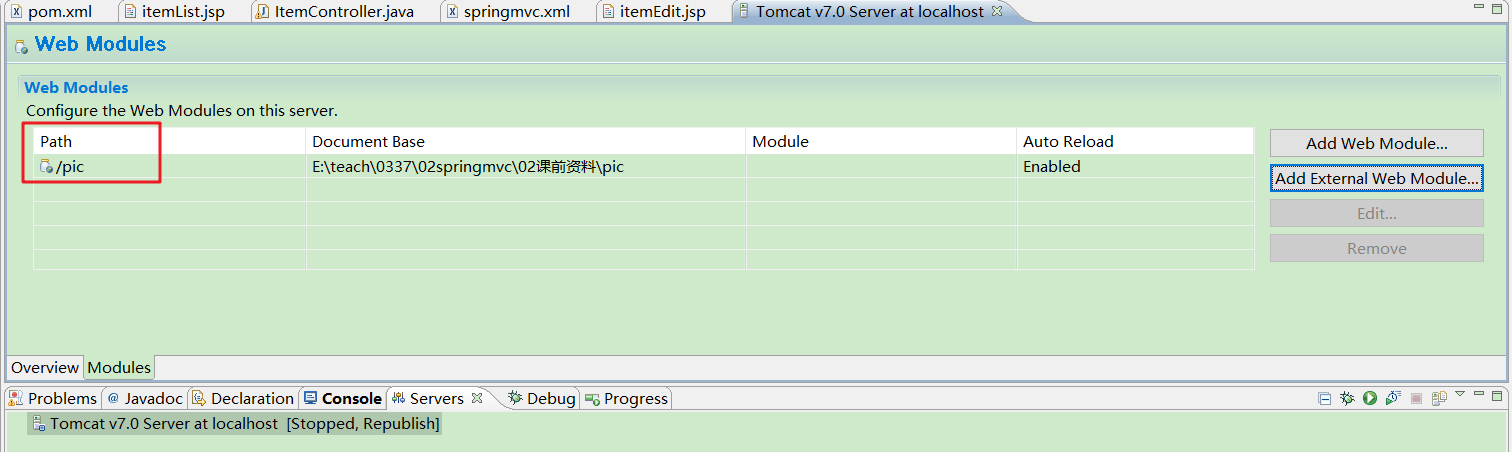
步骤一：



步骤二：

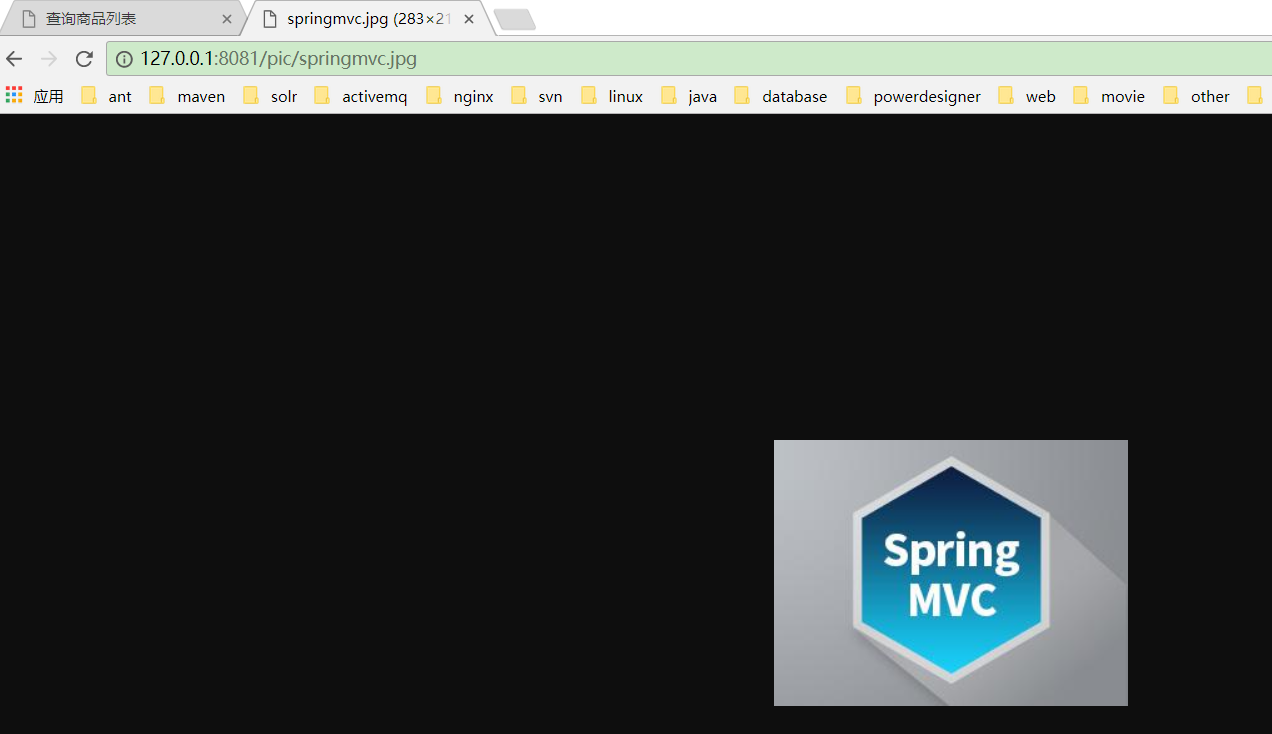


步骤三：



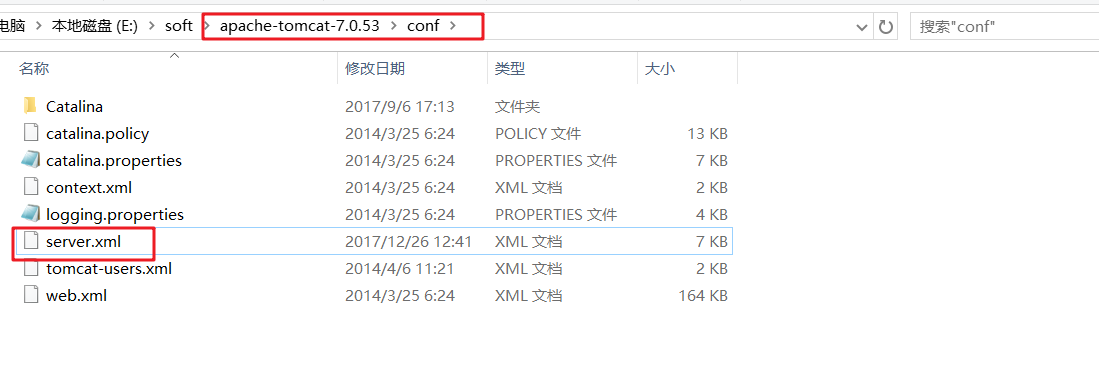
#### 测试虚拟目录

访问地址：http://127.0.0.1:8081/pic/springmvc.jpg

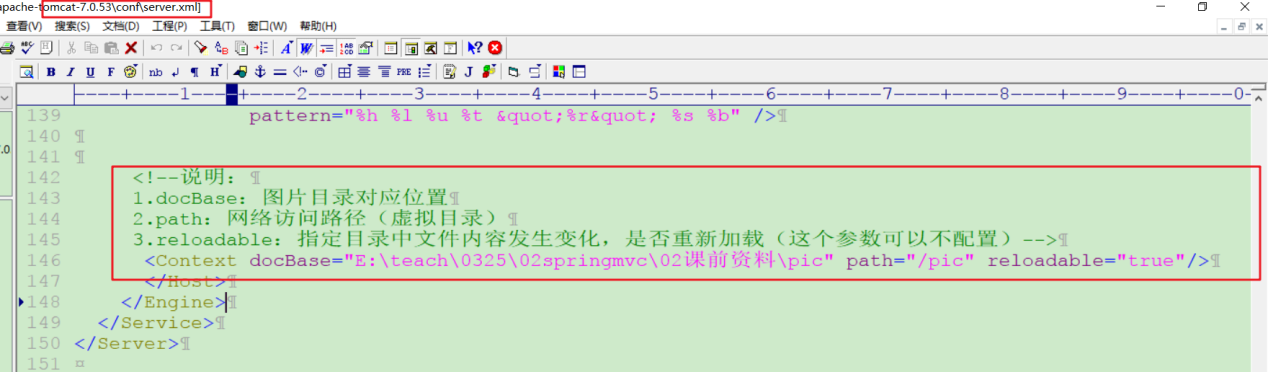


#### 项目部署阶段的虚拟目录配置

步骤一：



步骤二：



### 商品图片上传实现

#### 配置pom.xml，加入依赖包

说明：commons-fileupload.jar，commons-io.jar，commons-codec.jar。

|  |
| --- |
| <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>cn.itheima</groupId>  <artifactId>springmvc-second</artifactId>  <version>0.0.1-SNAPSHOT</version>  <packaging>war</packaging>  <name>springmvc-second Maven Webapp</name>  <url>http://maven.apache.org</url>  <properties>  <!-- spring版本号 -->  <spring.version>4.3.8.RELEASE</spring.version>  <!-- aspectj版本号 -->  <aspectj.version>1.6.12</aspectj.version>  <!-- jstl标签版本 -->  <jstl.version>1.2</jstl.version>  <!-- mybatis版本号 -->  <mybatis.version>3.4.5</mybatis.version>  <!-- mybatis-spring整合包版本 -->  <mybatis.spring.version>1.3.1</mybatis.spring.version>  <!-- mysql版本号 -->  <mysql.version>5.1.30</mysql.version>  <!-- dbcp数据源连接池jar包 -->  <dbcp.version>1.2.2</dbcp.version>  <!-- log4j日志包版本 -->  <slf4j.version>1.7.7</slf4j.version>  <log4j.version>1.2.17</log4j.version>  <!-- commons-lang版本 -->  <commons.lang.version>2.6</commons.lang.version>  <!-- 文件上传包版本 -->  <fileupload.version>1.3.1</fileupload.version>  <commonsio.version>2.4</commonsio.version>  <commonscodec.version>1.9</commonscodec.version>  </properties>  <dependencies>  <!-- springmvc依赖包 -->  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-core</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-tx</artifactId>  <version>${spring.version}</version>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-jdbc</artifactId>  <version>${spring.version}</version>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-webmvc</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-context-support</artifactId>  <version>${spring.version}</version>  </dependency>  <dependency>  <groupId>org.aspectj</groupId>  <artifactId>aspectjweaver</artifactId>  <version>${aspectj.version}</version>  </dependency>  <!-- JSTL标签类 -->  <dependency>  <groupId>jstl</groupId>  <artifactId>jstl</artifactId>  <version>${jstl.version}</version>  </dependency>  <!-- mybatis核心包 -->  <dependency>  <groupId>org.mybatis</groupId>  <artifactId>mybatis</artifactId>  <version>${mybatis.version}</version>  </dependency>  <!-- mybatis-spring整合包 -->  <dependency>  <groupId>org.mybatis</groupId>  <artifactId>mybatis-spring</artifactId>  <version>${mybatis.spring.version}</version>  </dependency>  <!-- 导入Mysql数据库链接jar包 -->  <dependency>  <groupId>mysql</groupId>  <artifactId>mysql-connector-java</artifactId>  <version>${mysql.version}</version>  </dependency>  <!-- 导入dbcp数据源连接池jar包 -->  <dependency>  <groupId>commons-dbcp</groupId>  <artifactId>commons-dbcp</artifactId>  <version>${dbcp.version}</version>  </dependency>  <!-- 日志文件管理包 -->  <dependency>  <groupId>log4j</groupId>  <artifactId>log4j</artifactId>  <version>${log4j.version}</version>  </dependency>  <dependency>  <groupId>org.slf4j</groupId>  <artifactId>slf4j-api</artifactId>  <version>${slf4j.version}</version>  </dependency>  <dependency>  <groupId>org.slf4j</groupId>  <artifactId>slf4j-log4j12</artifactId>  <version>${slf4j.version}</version>  </dependency>  <dependency>  <groupId>commons-lang</groupId>  <artifactId>commons-lang</artifactId>  <version>${commons.lang.version}</version>  </dependency>  <!-- 文件上传组件包 -->  <dependency>  <groupId>commons-fileupload</groupId>  <artifactId>commons-fileupload</artifactId>  <version>${fileupload.version}</version>  </dependency>  <dependency>  <groupId>commons-io</groupId>  <artifactId>commons-io</artifactId>  <version>${commonsio.version}</version>  </dependency>  <dependency>  <groupId>commons-codec</groupId>  <artifactId>commons-codec</artifactId>  <version>${commonscodec.version}</version>  </dependency>  <!-- jsp依赖包，只在编译时需要 -->  <dependency>  <groupId>javax.servlet</groupId>  <artifactId>jsp-api</artifactId>  <version>2.0</version>  <scope>provided</scope>  </dependency>  </dependencies>  <build>  <finalName>ssm</finalName>  <plugins>  <plugin>  <groupId>org.apache.tomcat.maven</groupId>  <artifactId>tomcat7-maven-plugin</artifactId>  <version>2.1</version>  <configuration>  <!-- tomcat 的端口号 -->  <port>8080</port>  <!-- 访问应用的路径 -->  <path>/springmvc-second</path>  <!-- URL按UTF-8进行编码，解决中文参数乱码 -->  <uriEncoding>UTF-8</uriEncoding>  <!-- tomcat名称 -->  <server>tomcat7</server>  </configuration>  </plugin>  </plugins>  </build>  </project> |

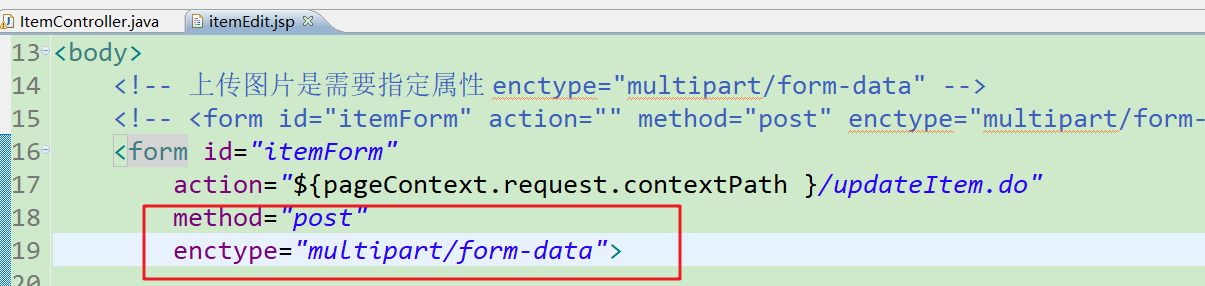
#### 配置文件上传解析器组件

说明：spring框架提供了文件上传解析器接口（MultipartResolver）。

|  |
| --- |
| <!-- 配置文件上传解析器,说明：  1.文件上传解析器的bean的Id属性值，必须是文件上传解析器接口的名称的首字母小写（multipartResolver）  -->  <bean id=*"multipartResolver"* class=*"org.springframework.web.multipart.commons.CommonsMultipartResolver"*>  <!--Provides "maxUploadSize", "maxInMemorySize" and "defaultEncoding" settings as  \* bean properties -->  <!-- maxUploadSize:上传文件的大小限制，以字节为单位。这里配置  10m=10\*1024\*1024  -->  <property name=*"maxUploadSize"* value=*"10485760"*/>    <!--maxInMemorySize:配置内存缓冲区的大小，以字节为单位：  4k：4\*1024  -->  <property name=*"maxInMemorySize"* value=*"4096"*/>    <!-- defaultEncoding：配置字符集编码 -->  <property name=*"defaultEncoding"* value=*"UTF-8"*/>    </bean> |

#### 编写代码

##### 修改商品编辑页面的表单，支持文件上传

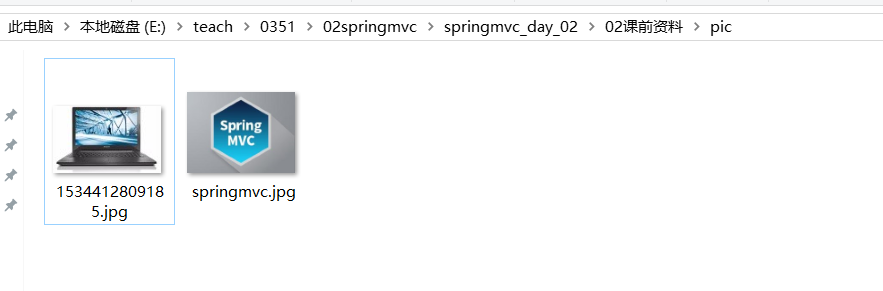


##### 编写java代码

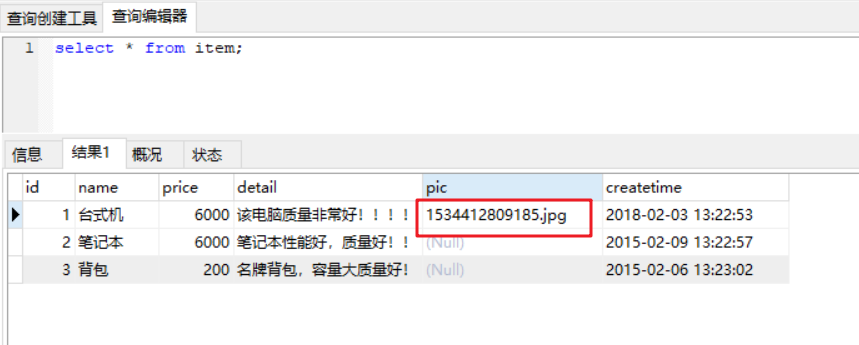
|  |
| --- |
| /\*\*  \* 修改商品数据，保存数据库  \* http://127.0.0.1:8080/ssm/updateItem.do  \*  \* <input type="file" name="pictureFile"/>  \* 增加形参MultipartFile，接收上传的商品图片文件  \*/  @RequestMapping("/updateItem.do")  **public** String updateItm(Item item,MultipartFile pictureFile){    // 增加文件上传逻辑===================start  **if**(pictureFile != **null** &&  pictureFile.getOriginalFilename() != **null**){    // 1.获取上传的文件名称  // http://127.0.0.1:8081/pic/springmvc.jpg  String oriName = pictureFile.getOriginalFilename();  String extName = oriName.substring(oriName.lastIndexOf("."));    // 2.重新命名文件  String newName = System.*currentTimeMillis*()+"";  newName = newName+extName;    // 3.上传文件  File dest = **new** File("E:\\teach\\0351\\02springmvc\\springmvc\_day\_02\\02课前资料\\pic\\"+newName);  **try** {  pictureFile.transferTo(dest);  } **catch** (Exception e) {  e.printStackTrace();  }    // 4.保存文件名称  item.setPic(newName);  }      // 增加文件上传逻辑===================end    **try**{  // 保存商品数据  itemService.updateItem(item);  }**catch**(Exception e){  e.printStackTrace();    // 发生异常，提示保存失败  **return** "common/failure";  }    // 提示保存成功  **return** "common/success";    } |

#### 测试

图一：



图二：



图三：



# json数据交互

## json例子

|  |
| --- |
| {  "id":1,  "name":"凡客衫",  "price":150,  "detail":"一件凡客衫走遍天下",  "pic":"666666.jpg"  } |

## 配置pom.xml，加入jackson依赖

|  |
| --- |
| <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>cn.itheima</groupId>  <artifactId>springmvc-second</artifactId>  <version>0.0.1-SNAPSHOT</version>  <packaging>war</packaging>  <name>springmvc-second Maven Webapp</name>  <url>http://maven.apache.org</url>  <properties>  <!-- spring版本号 -->  <spring.version>4.3.8.RELEASE</spring.version>  <!-- aspectj版本号 -->  <aspectj.version>1.6.12</aspectj.version>  <!-- jstl标签版本 -->  <jstl.version>1.2</jstl.version>  <!-- mybatis版本号 -->  <mybatis.version>3.4.5</mybatis.version>  <!-- mybatis-spring整合包版本 -->  <mybatis.spring.version>1.3.1</mybatis.spring.version>  <!-- mysql版本号 -->  <mysql.version>5.1.30</mysql.version>  <!-- dbcp数据源连接池jar包 -->  <dbcp.version>1.2.2</dbcp.version>  <!-- log4j日志包版本 -->  <slf4j.version>1.7.7</slf4j.version>  <log4j.version>1.2.17</log4j.version>  <!-- commons-lang版本 -->  <commons.lang.version>2.6</commons.lang.version>  <!-- 文件上传包版本 -->  <fileupload.version>1.3.1</fileupload.version>  <commonsio.version>2.4</commonsio.version>  <commonscodec.version>1.9</commonscodec.version>  <!-- json支持,jackson包版本 -->  <jackson.version>2.7.4</jackson.version>  </properties>  <dependencies>  <!-- springmvc依赖包 -->  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-core</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-tx</artifactId>  <version>${spring.version}</version>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-jdbc</artifactId>  <version>${spring.version}</version>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-webmvc</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-context-support</artifactId>  <version>${spring.version}</version>  </dependency>  <dependency>  <groupId>org.aspectj</groupId>  <artifactId>aspectjweaver</artifactId>  <version>${aspectj.version}</version>  </dependency>  <!-- JSTL标签类 -->  <dependency>  <groupId>jstl</groupId>  <artifactId>jstl</artifactId>  <version>${jstl.version}</version>  </dependency>  <!-- mybatis核心包 -->  <dependency>  <groupId>org.mybatis</groupId>  <artifactId>mybatis</artifactId>  <version>${mybatis.version}</version>  </dependency>  <!-- mybatis-spring整合包 -->  <dependency>  <groupId>org.mybatis</groupId>  <artifactId>mybatis-spring</artifactId>  <version>${mybatis.spring.version}</version>  </dependency>  <!-- 导入Mysql数据库链接jar包 -->  <dependency>  <groupId>mysql</groupId>  <artifactId>mysql-connector-java</artifactId>  <version>${mysql.version}</version>  </dependency>  <!-- 导入dbcp数据源连接池jar包 -->  <dependency>  <groupId>commons-dbcp</groupId>  <artifactId>commons-dbcp</artifactId>  <version>${dbcp.version}</version>  </dependency>  <!-- 日志文件管理包 -->  <dependency>  <groupId>log4j</groupId>  <artifactId>log4j</artifactId>  <version>${log4j.version}</version>  </dependency>  <dependency>  <groupId>org.slf4j</groupId>  <artifactId>slf4j-api</artifactId>  <version>${slf4j.version}</version>  </dependency>  <dependency>  <groupId>org.slf4j</groupId>  <artifactId>slf4j-log4j12</artifactId>  <version>${slf4j.version}</version>  </dependency>  <dependency>  <groupId>commons-lang</groupId>  <artifactId>commons-lang</artifactId>  <version>${commons.lang.version}</version>  </dependency>  <!-- 文件上传组件包 -->  <dependency>  <groupId>commons-fileupload</groupId>  <artifactId>commons-fileupload</artifactId>  <version>${fileupload.version}</version>  </dependency>  <dependency>  <groupId>commons-io</groupId>  <artifactId>commons-io</artifactId>  <version>${commonsio.version}</version>  </dependency>  <dependency>  <groupId>commons-codec</groupId>  <artifactId>commons-codec</artifactId>  <version>${commonscodec.version}</version>  </dependency>  <!-- jackson依赖包 -->  <dependency>  <groupId>com.fasterxml.jackson.core</groupId>  <artifactId>jackson-databind</artifactId>  <version>${jackson.version}</version>  </dependency>  <dependency>  <groupId>com.fasterxml.jackson.core</groupId>  <artifactId>jackson-core</artifactId>  <version>${jackson.version}</version>  </dependency>  <dependency>  <groupId>com.fasterxml.jackson.core</groupId>  <artifactId>jackson-annotations</artifactId>  <version>${jackson.version}</version>  </dependency>  <!-- jsp依赖包，只在编译时需要 -->  <dependency>  <groupId>javax.servlet</groupId>  <artifactId>jsp-api</artifactId>  <version>2.0</version>  <scope>provided</scope>  </dependency>  </dependencies>  <build>  <finalName>ssm</finalName>  <plugins>  <plugin>  <groupId>org.apache.tomcat.maven</groupId>  <artifactId>tomcat7-maven-plugin</artifactId>  <version>2.1</version>  <configuration>  <!-- tomcat 的端口号 -->  <port>8080</port>  <!-- 访问应用的路径 -->  <path>/springmvc-second</path>  <!-- URL按UTF-8进行编码，解决中文参数乱码 -->  <uriEncoding>UTF-8</uriEncoding>  <!-- tomcat名称 -->  <server>tomcat7</server>  </configuration>  </plugin>  </plugins>  </build>  </project> |

## 配置相关

说明：处理器映射器和处理器适配器使用注解驱动的方式配置<mvc:annotation-driven/>。默认支持jackson的转换。

## 两个注解

### @RequestBody注解

作用：在处理器方法形参上使用，把请求的json格式的数据，转换成java对象。

### @ResponseBody注解

作用：在处理器方法返回值上，或者处理器方法上使用。把响应的java对象，转换json格式的数据。

## json交互案例演示

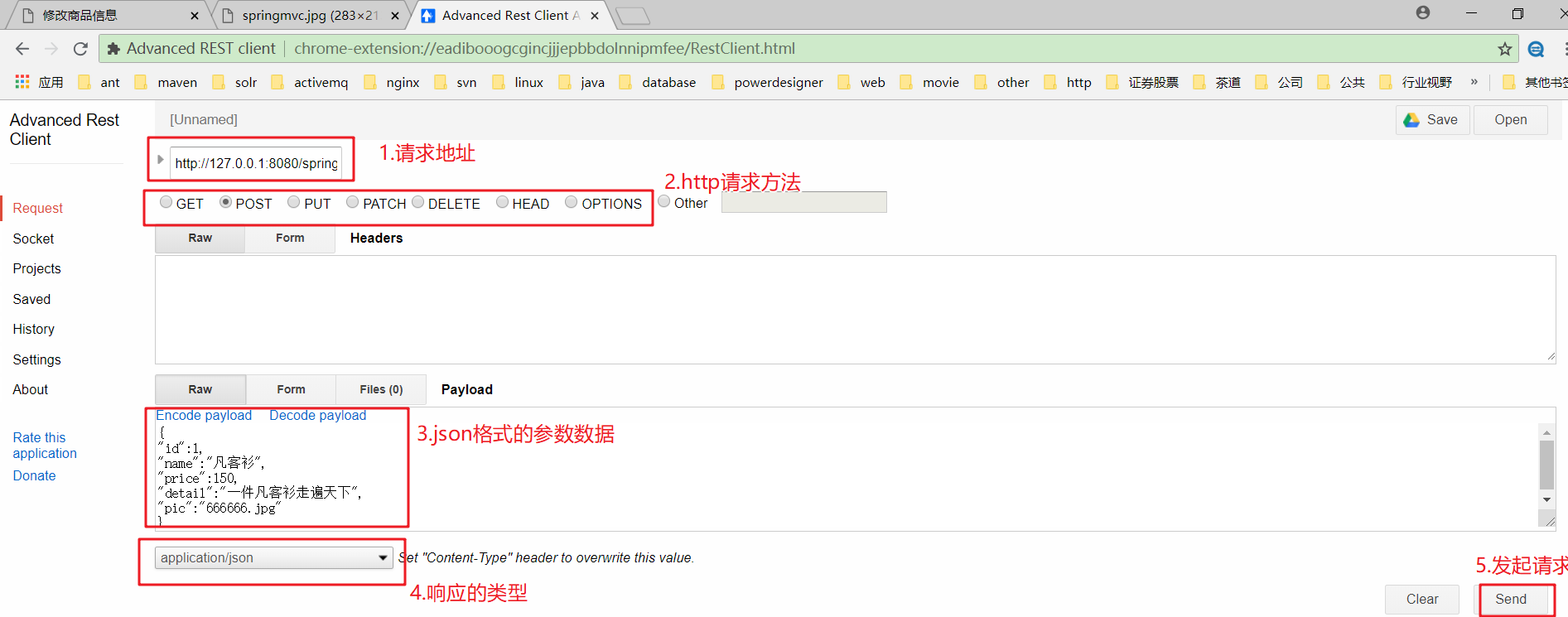
### 需求

请求json格式的商品数据==》在处理器方法中用java对象接收==》响应一个java对象==》在前端获取到json格式的数据。

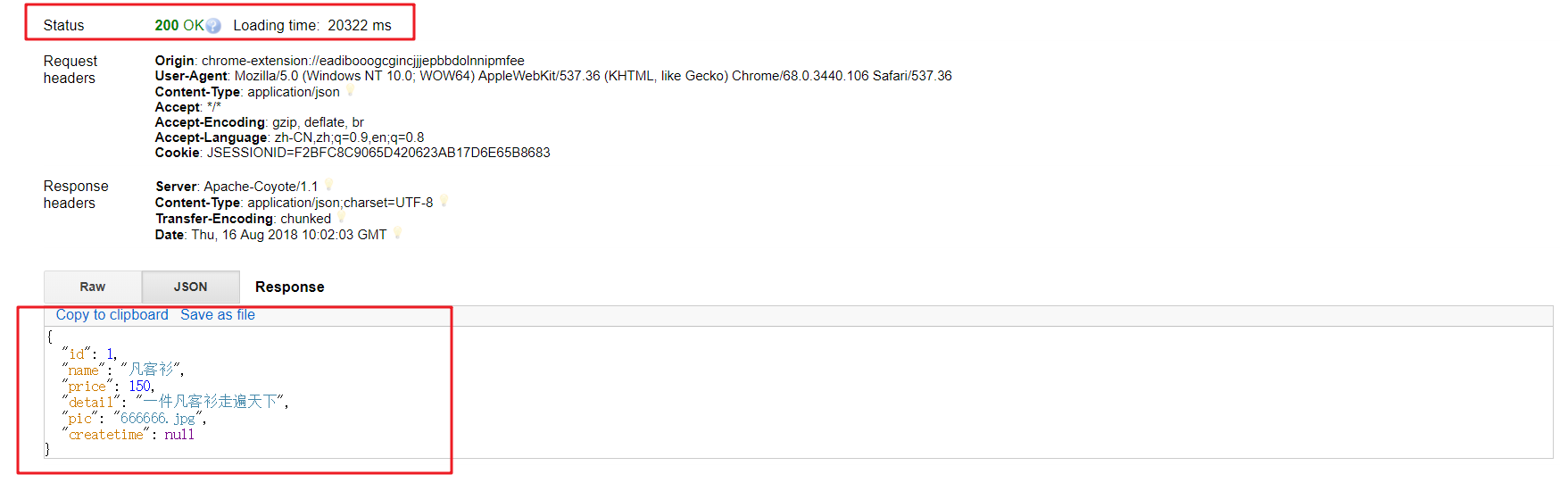
### 需求实现

|  |
| --- |
| /\*\*  \* json交互学习：  \* 请求json格式的商品数据==》在处理器方法中用java对象接收==》  \* 响应一个java对象==》在前端获取到json格式的数据。  \*/  @RequestMapping("/testJson.do")  @ResponseBody  **public** Item testJson(@RequestBody Item item){    **return** item;  } |

图一：



图二：



# restful支持

## restful介绍

restful，它是一种软件设计风格，指的是表现层资源的状态转换（Representational state transfer）。互联网上的一切都可以看成是资源，比如一张图片，一部电影。restful根据HTTP请求方法：POST/GET/PUT/DELETE，定义了资源的操作方法：新增/查询/修改/删除。

使用restful风格的优点：结构清晰、符合标准、易于理解、 扩展方便、请求的url更加简洁，得到越来越多网站的采用。

restful风格url示例：

/item/1 HTTP GET ： 查询 id = 1 的 Item

/item/1 HTTP DELETE： 删除 id = 1 的 Item

/item/1 HTTP PUT： 修改 id = 1 的 Item

/item HTTP POST： 新增 Item

## 传统与REST风格url比较



## restful案例演示

### 需求

使用restful风格，实现商品的增删改查操作。

### 需求实现

#### 编写jsp页面

|  |
| --- |
| <%@ page language=*"java"* contentType=*"text/html; charset=UTF-8"*  pageEncoding=*"UTF-8"*%>  <!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=UTF-8"*>  <title>restful学习</title>  </head>  <body>  <h3>rest请求 - post</h3>  <form action=*"rest"* method=*"post"*>  商品名称： <input type=*"text"* name=*"name"*><br/>  <input type=*"hidden"* name=*"\_method"* value=*"POST"*>  <input type=*"submit"* value=*"新增"*>  </form>  <h3>rest请求 - put</h3>  <form action=*"rest/1"* method=*"post"*>  商品名称： <input type=*"text"* name=*"name"*><br/>  <input type=*"hidden"* name=*"\_method"* value=*"PUT"*>  <input type=*"submit"* value=*"修改"*>  </form>  <h3>rest请求 - get</h3>  <form action=*"rest/1"* method=*"post"*>  <input type=*"hidden"* name=*"\_method"* value=*"GET"*>  <input type=*"submit"* value=*"查询"*>  </form>  <h3>rest请求 - delete</h3>  <form action=*"rest/1"* method=*"post"*>  <input type=*"hidden"* name=*"\_method"* value=*"DELETE"*>  <input type=*"submit"* value=*"删除"*>  </form>  </body>  </html> |

#### 配置web.xml中的前端控制器配置，改成/支持restful

|  |
| --- |
| <!-- 配置前端控制器 -->  <servlet>  <servlet-name>ssm</servlet-name>  <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>    <!-- 加载springmvc的主配置文件 -->  <init-param>  <param-name>contextConfigLocation</param-name>  <param-value>classpath:spring/springmvc.xml</param-value>  </init-param>    <!-- 配置tomcat启动就加载前端控制器 -->  <load-on-startup>1</load-on-startup>  </servlet>    <servlet-mapping>  <servlet-name>ssm</servlet-name>  <!-- 配置请求的url规则，说明：  1.\*.do，以.do结尾的请求进入前端控制器  2./，表示所有请求都进入前端控制器  -->  <!-- <url-pattern>\*.do</url-pattern> -->  <!-- 配置所有请求都进入前端控制器，支持restful -->  <url-pattern>/</url-pattern>  </servlet-mapping> |

#### 在web.xml配置HiddenHttpMethodFilter过滤器

作用：

由于浏览器 form 表单只支持 POST 与 GET 请求，不支持PUT、 DELETE 请求。 Spring3.0 添加了一个过滤器，将浏览器请求改为指定的请求方式，发送给我们的控制器(controller)方法，使得完整支持 POST、GET、 PUT、 DELETE 请求。

使用细节：

第一步：在 web.xml 中配置该过滤器  
 第二步：请求方式必须使用 post 请求  
 第三步：按照要求提供\_method 请求参数，该参数的取值就是我们需要的http请求方法：POST/GET/PUT/DELETE

|  |
| --- |
| <!-- 配置HiddenHttpMethodFilter -->  <filter>  <filter-name>httpMethodFilter</filter-name>  <filter-class>org.springframework.web.filter.HiddenHttpMethodFilter</filter-class>  </filter>    <filter-mapping>  <filter-name>httpMethodFilter</filter-name>  <url-pattern>/\*</url-pattern>  </filter-mapping> |

#### 编写RestFul的controller

# 拦截器