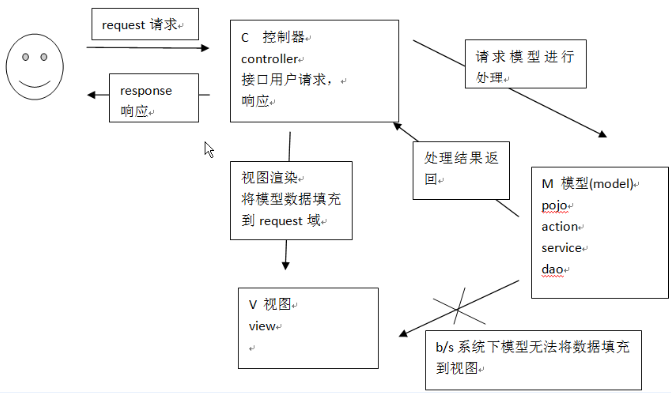
SpringMVC

1. SpringMVC\_day01
   1. springMVC 框架
      1. 什么是springMvc
         1. springmvc是spring框架的一个模块，springmvc和spring无需通过中间整合层进行整合
         2. springMvc是一个基于mvc的web框架
      2. 什么是mvc
         1. mvc是一个设计模式，mvc在b/s系统下的应用
            1. 
      3. springmvc框架

2.请求查找handler

11.response

前端控制器

DispatcherServlet

接受用户请求，响应

7.返回modelandview

8.请求进行试图解析

* + - 1. 发起请求到前端控制器（DispatcherServlet）

10.视图view，进行视图渲染

4.请求执行handler

5.执行handler

9.返回视图view

view

jsp

pdf

视图解析器

view resolver

6.返回ModelAndView

处理器适配器

handlerAdapter

去执行Handler

handler处理器

平常叫做controller

3.返回一个执行链

handlerExecutionChain{

interceptor

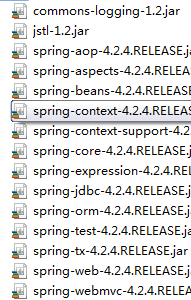
handler

}

处理器映射器

handlerMapping

1.request请求，url

* + - 1. 前端控制器请求handlermapping查找handler（根据xml配置或者注解）
      2. 处理器映射器handlermapping向前端控制器返回handler
      3. 前端控制器调用处理器适配器执行handler
      4. 处理器适配器执行handler
      5. handler向处理器适配器返回modelandview
      6. 处理器适配器向前端控制器返回modelandview
      7. 前端控制器请求视图解析器去进行视图解析（根据逻辑视图名，解析成真正的视图jsp）
      8. 视图解析器向前端控制器返回view
      9. 前端控制器进行试图渲染，将模型数据（modelandview）填充到request域
      10. 前端控制器向用户响应结果
      11. 组件：
          1. 前端控制器DispatcherServlet：接受请求，响应结果，相当于转发器，中央处理器，有了DispatcherServlet减少了其他组件之间的耦合度
          2. 处理器映射器HandlerMapping（不需要程序猿开发）：根据请求的url查找handler
          3. 处理器适配器HandlerAdapter：按照特定的规则（handlerAdapter要求的规则）执行handler。
          4. 处理器（handler）（需要程序猿开发）：编写handler时要按照HandlerAdapter的要求去做，这样适配器才可以去正确执行handler
          5. 视图解析器ViewResolver（不需要程序猿开发）：进行视图解析，根据逻辑视图，解析真正的视图view
          6. 视图view（需要程序猿开发jsp）：view是一个接口，实现类支持不同的view类型（jsp，freemarket）
  1. 入门程序
     1. jar包
        1. 
     2. 需求
        1. springmvc和mybatis使用第一个用例（商品订单管理）

订单表orders

number：订单号

user\_id：外键，用户id

订单明细表orderDetail

orders\_id：外键，订单id

items\_id:外键，商品id

商品表items

用户表user

id：自增主键

user\_id外键

orders\_id

items\_id

一对一

一对多

一对多

一对一

一对一

一对多

orders---》items

一对多+一对一=一对多

items--》orders

一对多+一对一=一对多

总之：items和orders是多对多关系

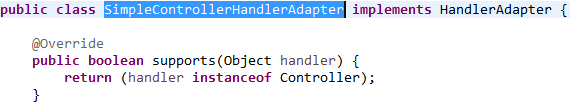
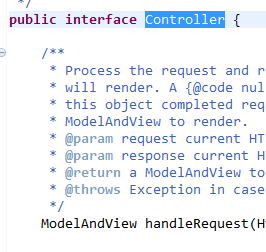
user---》item

一对多+一对多+一对一=一对多

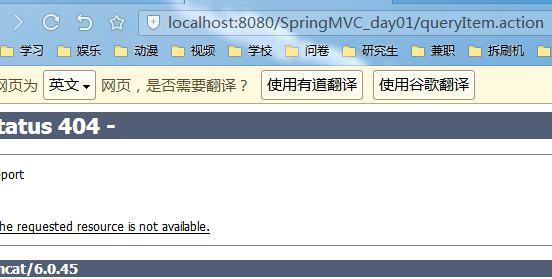
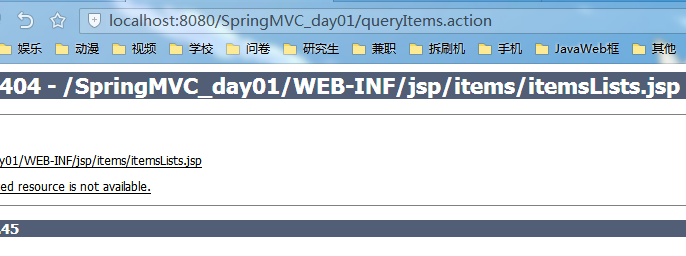
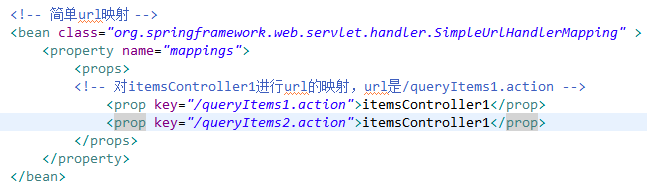
items—》user

一对多+一对一+一对一=一对多

总之：items和user是多对多关系

* + 1. 配置前端控制器
       1. 在web.xml中配置前端控制器dispatcherServlet
          1. 
    2. 配置处理器适配器（springmvc.xml）
       1. 在classpath下的springmvc.xml中配置处理器适配器
       2. 通过查看源代码，分辨它支持哪些handler
          1. 表示此适配器能够执行实现controller接口的handler
          2. 通过实现这个接口能够编写自己的handler
          3. 
    3. 编写handler
       1. 需要实现controller接口，才能由org.springframework.web.servlet.mvc.SimpleControllerHandlerAdapter这个接口来执行

|  |
| --- |
| **package** cn.huangwei.ssm.controller;  **import** java.util.ArrayList;  **import** java.util.List;  **import** javax.servlet.http.HttpServletRequest;  **import** javax.servlet.http.HttpServletResponse;  **import** org.springframework.web.servlet.ModelAndView;  **import** org.springframework.web.servlet.mvc.Controller;  **import** cn.huangwei.ssm.po.Items;  /\*\*  \* 实现controller接口的处理器  \* **@author** sshss  \*  \*/  **public** **class** ItemsController1 **implements** Controller{  @Override  **public** ModelAndView handleRequest(HttpServletRequest request,  HttpServletResponse response) **throws** Exception {  //调用service查找数据库，查询商品列表，这里使用静态数据模拟  List<Items> itemsList = **new** ArrayList<Items>();    //向list中添加数据  Items items\_1 = **new** Items();  items\_1.setName("联想笔记本");  items\_1.setPrice(5000f);  items\_1.setDetail("iphone苹果手机");    Items items\_2 = **new** Items();  items\_2.setName("联想笔记本");  items\_2.setPrice(5000f);  items\_2.setDetail("iphone苹果手机");    itemsList.add(items\_1);  itemsList.add(items\_2);    //返回ModelAndView  ModelAndView modelAndView = **new** ModelAndView();  //相当于request的setAttribute，在jsp页面中通过itemsList取数据  modelAndView.addObject("itemsList", itemsList);    //指定视图  modelAndView.setViewName("/WEB-INF/jsp/items/itemsList.jsp");    **return** modelAndView;  }  } |

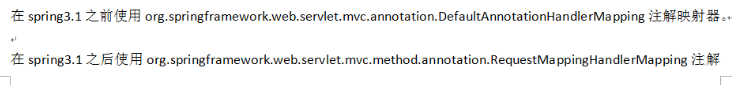
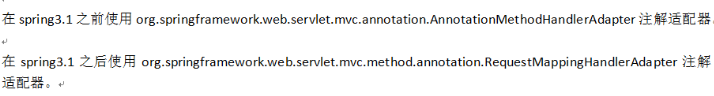
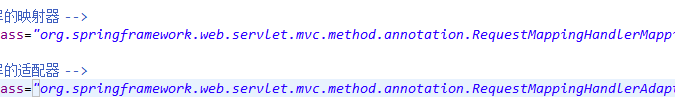
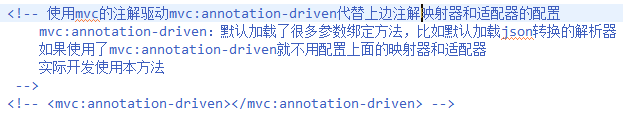
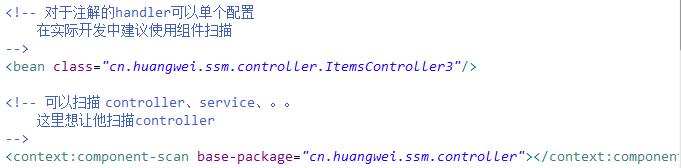
* + 1. 配置handler
       1. 将编写的handler在spring容器中加载
    2. 视图的编写
       1. 
    3. 配置处理器映射器（springmvc.xml）
       1. 
    4. 配置视图解析器（springmvc.xml）
       1. 需要配置解析jsp的视图
       2. 
    5. 部署调试
       1. 访问地址就是<http://localhost:8080/SpringMVC_day01/queryItems.action>
       2. 错误
          1. ，处理器映射器根据url找不到handler，报下边的错误，说明url错误
          2. 处理器映射器根据url找到了handler，转发的jsp的页面找不到了，报地址错误了
  1. 非注解的处理器映射器和适配器
     1. 处理器映射器
        1. org.springframework.web.servlet.handler.BeanNameUrlHandlerMapping
        2. 另外一个映射器org.springframework.web.servlet.handler.SimpleUrlHandlerMapping
        3. 多个映射器可以并存，前端控制器判断url能让哪个映射器映射，就让哪个映射器映射
     2. 适配器
        1. org.springframework.web.servlet.mvc.SimpleControllerHandlerAdapter要求编写的handler要实现controller接口
        2. org.springframework.web.servlet.mvc.HttpRequestHandlerAdapter要求编写的handler实现HttpRequestHandler

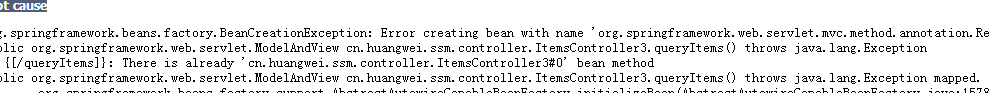
|  |
| --- |
| @Override  **public** **void** handleRequest(HttpServletRequest request,  HttpServletResponse response) **throws** ServletException, IOException {  // **TODO** Auto-generated method stub  //调用service查找数据库，查询商品列表，这里使用静态数据模拟  List<Items> itemsList = **new** ArrayList<Items>();    //向list中添加数据  Items items\_1 = **new** Items();  items\_1.setName("联想笔记本");  items\_1.setPrice(5000f);  items\_1.setDetail("iphone苹果手机");    Items items\_2 = **new** Items();  items\_2.setName("联想笔记本");  items\_2.setPrice(5000f);  items\_2.setDetail("iphone苹果手机");    itemsList.add(items\_1);  itemsList.add(items\_2);    //设置模型数据  request.setAttribute("itemsList", itemsList);  //设置转发的视图  request.getRequestDispatcher("/WEB-INF/jsp/items/itemsList.jsp").forward(request, response);  } |

* + - 1. 使用此方法可以通过修改response，设置响应的数据格式，比如json数据 response.setCharacterEncoding(“utf-8”);

response.setContentType(“application/json;charset=utf-8”);

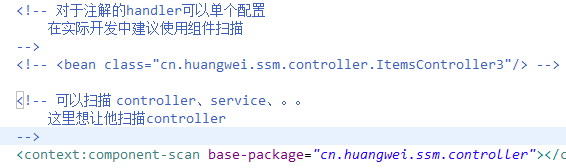
response.getWriter().write(“json串”)；

* 1. dispatcherServlet.properties
     1. 前端控制器会从上边的文件中加载处理器映射器。适配器、视图解析器等组件，如果不在springmvc.xml中配置，就是用默认加载的处理器映射器。适配器、视图解析器等组件
  2. 注解的处理器映射器和适配器
     1. 映射器
        1. 
     2. 适配器
        1. 
     3. 配置注解的映射器和适配器
        1. 
        2. 
     4. 开发注解的handler
        1. 使用注解的映射器和注解的适配器（必须配对使用）
           1. 
        2. 在spring容器中加载你的handler
           1. 
           2. 问题

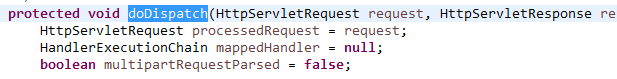


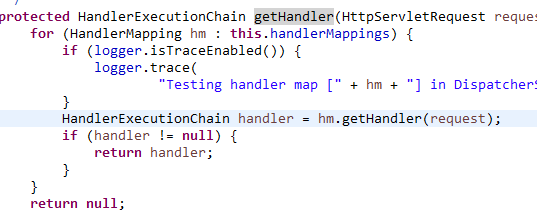
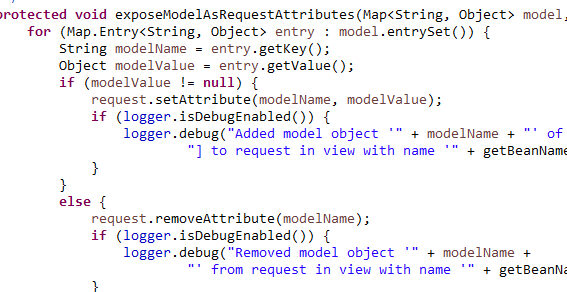


因为这里注解handler的单个配置，又有组件扫描，导致映射不成功



* 1. 源码分析
     1. 通过前端控制器源码分析springmvc的执行过程
        1. 前端控制器接受请求
           1. 调用dodispatcher



* + - 1. 前端控制器调用处理器映射器
         1. 
         2. 
      2. 调用处理器适配器执行handler，返回modelAndView
         1. 
         2. 
      3. 试图渲染，将model数据填充到request域
         1. 试图解析，得到view
         2. 把model映射到request中
  1. 入门程序小结
     1. 通过入门程序理解springmvc前端控制器、**处理器映射器、处理器适配器**、视图解析器
     2. 前端控制器的配置
        1. 
     3. 处理器映射器和适配器
        1. 处理器映射器
           1. 非注解处理器映射器（了解）
           2. 注解的处理器映射器）（掌握）

对标记有controller类中标记有requestMapping进行映射，在requestMapping里面定义映射的url

* + - 1. 处理器适配器
         1. 非注解处理器适配器（了解）
         2. 注解的处理器适配器（掌握）

注解的处理器适配器和注解的处理器映射器是配对使用的，不能使用非注解的映射器进行映射，不能一个注解一个非注解，一定要配对使用

* + - 1. 实际开发使用<mvc:annotation-driven></mvc:annotation-driven>注解驱动，就不用配置处理器映射器和处理器适配器了
  1. springmvc和mybatis整合
     1. 需求
        1. 使用springmvc和mybatis完成商品列表的查询
     2. 整合思路

springmvc和mybatis的整合系统架构

表现层

springmvc

业务层

service接口

持久层

mybatis

mysql

spring将各层进行整合，

通过spring管理持久层的mapper（相当于dao接口），

通过spring管理业务层的handler

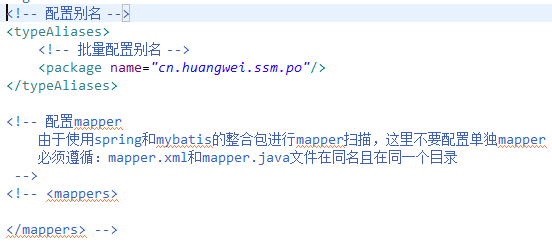
handler、mapper、handler都是Javabean

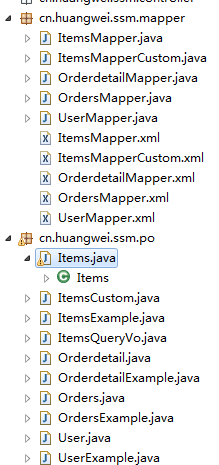
spring进行事务控制

通过spring管理表现层的handler，handler中可以调用service接口

* + - 1. 整合持久层（dao）
         1. mybatis和spring的整合，通过spring管理mapper接口
         2. 使用mapper扫描器自动扫描mapper接口在spring进行注册
      2. 整合service层
         1. 通过spring管理service接口
         2. 使用配置方式将service接口配置在spring配置文件中
         3. 实现事务控制
      3. 整合springmvc
         1. 由于springmvc是spring的模块，不需要整合
    1. 环境
       1. jar
          1. 数据库驱动包、mybatis的jar、mybatis和spring的整合包、log4j、数据库连接池（dbcp，c3p0），spring所有jar
    2. 整合dao
       1. mybatis和spring进行整合
          1. sqlMapConfig.xml

mybatis自己的配置文件



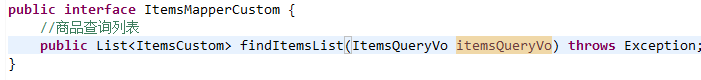
* + - 1. applicationContext-dao.xml
         1. 配置：数据源----SqlSessionFactory------mapper扫描器
         2. 
      2. 逆向工程生成po类及mapper（单表的增删改查）
         1. 将逆向工程生成的类文件拷贝到项目里面
      3. 手动定义商品查询的mapper
         1. 针对综合查询mapper，一般情况会有关联查询，建议自定义mapper

ItemsMapperCustom.xml

sql语句：SELECT \* FROM items WHERE NAME LIKE '%笔记本%'

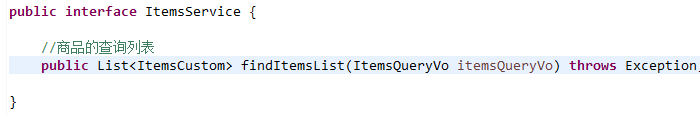


ItemsMapperCustom.java



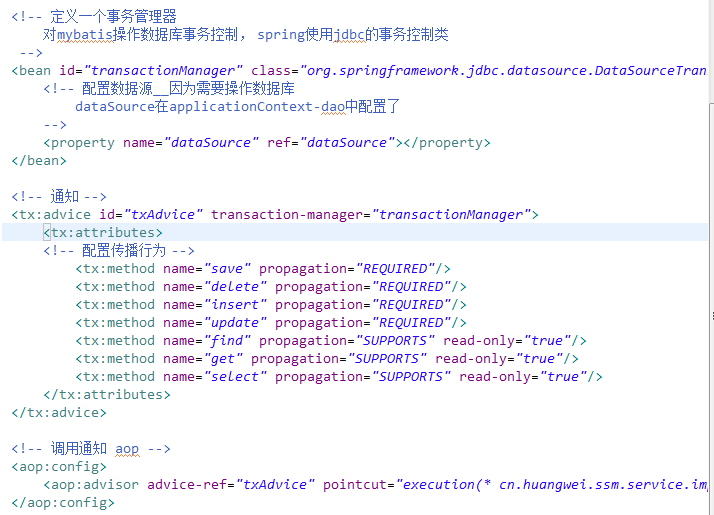
* + 1. 整合service

让spring管理service接口

* + - 1. 先定义service接口
         1. 
      2. service的实现类
         1. 
      3. 在spring容器中配置service
         1. 创建applicationContext-service.xml，文件中配置service

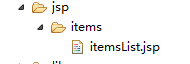


* + - 1. 创建事务控制（applicationContext-transaction.xml）
         1. 使用spring的声明式事务控制方法



* + 1. 整合springMVC

创建springmvc.xml文件，配置处理器映射器、适配器、视图解析器

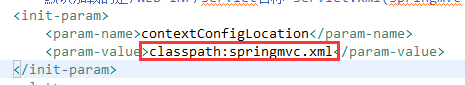
* + - 1. 
      2. 配置前端的控制器
         1. 参考入门程序
      3. 开始编写handler（controller）
         1. 
      4. 编写jsp
         1. 
    1. 加载spring的容器
       1. 将mapper、service、controller加载到spring的容器中



* + - 1. 建议使用通配符的方法加载上面的配置文件
      2. 在web.xml中，添加spring容器的监听器

|  |
| --- |
| <!-- 加载spring的容器 -->  <context-param>  <param-name>contextConfigLocation</param-name>  <param-value>/WEB-INF/classes/spring/applicationContext-\*.xml</param-value>  </context-param>  <listener>  <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>  </listener> |

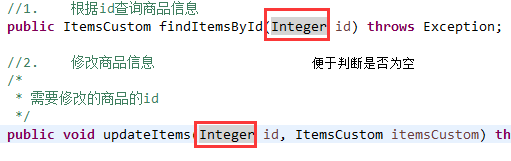
* + 1. 调试问题
       1. 

路径错误

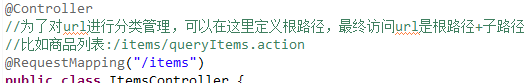
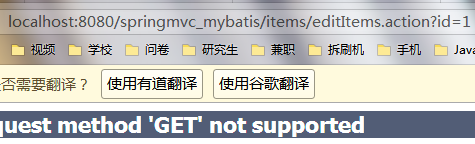
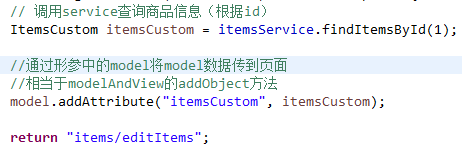
改正：

* 1. 商品修改的功能开发
     1. 需求

操作流程

* + - 1. 进入商品查询列表页面
      2. 点击修改，进入商品修改页面，页面中显示了要修改的商品信息（数据库查询）根据商品的id去查询商品信息
      3. 在商品修改页面，修改商品信息，修改后，点击提交
    1. 开发mapper（数据库操作）
       1. 根据id查询商品信息
       2. 更新items表的数据（根据id）
       3. 不用开发，使用逆向工程的代码
    2. 开发service
       1. 根据id查询商品信息
       2. 修改商品信息
       3. 
       4. 
    3. 开发controller（请求就有他）
       1. 商品信息修改页面显示
       2. 商品信息修改页面的提交方法
  1. @RequestMapping

定义controller方法对应的url，进行处理器映射使用

* + - 1. 窄化请求映射
         1. 
      2. 限制http的请求方法
         1. 限制请求为post，此时请求是get，就会报错
         2. 
  1. controller方法的返回值
     1. 返回modelAndview
        1. 需要在方法结束时，定义modelandView，将model和view分别进行设置
     2. 返回逻辑视图名
        1. 如果controller方法返回string，表示返回逻辑视图名
        2. 真正视图（jsp路径）=前缀+逻辑视图名+后缀
           1. 
        3. redirect重定向
           1. 商品修改提交后，重定向到商品查询列表
           2. 特点：浏览器地址栏中的url会发生变化，修改提交的request数据无法传到重定向的地址，因为重定向后重新进行request（request无法共享）
        4. forward页面转发
           1. 通过forward进行页面转发，浏览器地址栏url不变，request可以共享



* + 1. 返回void
       1. 在controller方法形参上可以定义request和response，使用request或者response指定响应结果：
          1. 使用request转向页面，如下：

request.getRequestDispatcher(“页面路径”).forward(request, response);

* + - * 1. 也可以通过response页面重定向

response.sendRedirect(“url”)

* + - * 1. 也可以通过response指定响应结果，例如响应json数据如下：

response.setCharacterEncoding(“utf-8”);

response.setContentType(“application/json;charset=utf-8”);

response.getWriter().write(‘’json串”);

* 1. 参数绑定
     1. spring参数绑定的过程
        1. 从客户端请求key/value数据，经过参数绑定，将key/value数据绑定到controller方法的形参上，springmvc中，接受 页面提交的数据是通过方法形参来接受，而不是在controller类中定义成员变量接受

客户端请求key/value

处理器适配器调用springmvc提供的参数绑定组件，将key/value数据转成 controller方法的 形参

参数绑定组件：在springmvc早期版本使用的是PropertyEditor（只能将字符串转成java对象）**后期使用converter（进行任意类型的转换），springmvc提供了很多converter转换器**

在特殊情况下需要自定义converter（对日期数据绑定需要自定义converter）

controller方法形参

* + 1. 参数绑定默认支持的类型(直接在controller方法形参上定义下边类型的对象，就可以使用这些对象，在参数绑定过程中如果遇到下边类型，直接进行参数绑定)
       1. HttpServletRequest

通过request对象获取请求信息

* + - 1. HttpServletResponse

通过response进行响应

* + - 1. HttpSession
         1. 通过session对象得到session中存放的对象
      2. Model/ModelMap
         1. model是一个接口，modelMap是一个接口实现
         2. 作用：将model数据填充到request域
    1. 支持一些简单的类型（int ，string ，float ，double ，。。。）

通过@RequestParam对简单类型的参数进行绑定，如果不使用@RequestParam，要求request传入的参数名称和controller方法形参名称一致，就可以绑定成功

如果使用@RequestParam，就不用限制request传入的参数名称和controller方法的形参名称一致

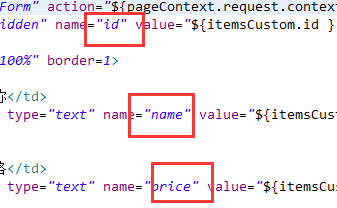
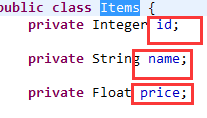


设置必须要传入的参数，通过required属性指定参数是否必须传入



如果不传参数就会报错

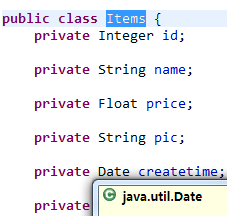
* + 1. 绑定pojo
       1. 页面中input的name和controller的pojo形参的属性名称一致，将页面中的数据绑定到pojo



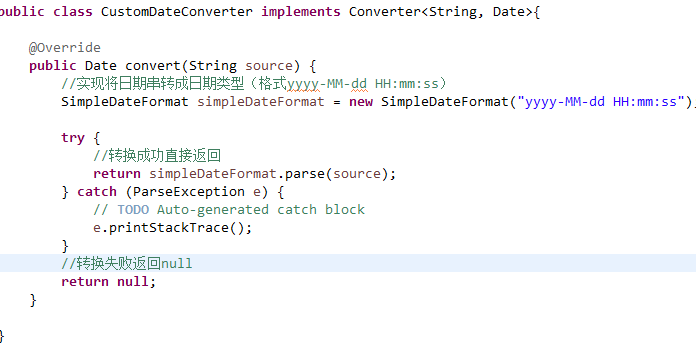
* + 1. 日期问题有变化，需要自定义参数绑定

对于controller形参中pojo的对象，如果属性中有日期类型，需要自定义参数绑定

* + - 1. 将请求日期数据串转成日期类型，要转换的日期类型和pojo中日期属性的类型保持一致

自定义的参数绑定，要将日期串转成java.util.Date的类型

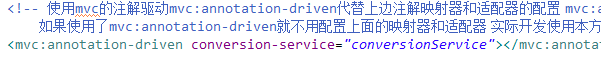
* + - 1. 需要向处理器映射器中注入自定义的参数绑定组件
         1. 
         2. 自定义参数转换器



* 1. 问题
     1. 无法传输值
        1.  enctype="multipart/form-data"是上传二进制数据;  
           form里面的input的值以2进制的方式传过去，所以request就得不到值了
        2. 解决：删除enctype="multipart/form-data"即可
     2. post乱码
        1. 在web.xml添加post乱码filter

|  |
| --- |
| <!-- post乱码的过滤器 -->  <filter>  <filter-name>CharacterEncodingFilter</filter-name>  <filter-class>org.springframework.web.filter.CharacterEncodingFilter</filter-class>  <init-param>  <param-name>encoding</param-name>  <param-value>utf-8</param-value>  </init-param>  </filter>  <filter-mapping>  <filter-name>CharacterEncodingFilter</filter-name>  <url-pattern>/\*</url-pattern>  </filter-mapping> |

* + 1. get请求乱码
       1. 修改tomcat配置文件添加编码与工程编码一致，如下
          1. <Connector URIEncoding=”utf-8” connectionTimeout=”20000” port=”8080” protocol=”HTTP/1.1” redirectProt=”8443” />
       2. 另外一种方法对参数进行重新编码
          1. String username = new String (request.getParameter(“userName”).getBytes(“ISO8859-1”),”utf-8”);
  1. springmvc和struts2的本质区别
     1. springmvc是基于方法开发的，struts2是基于类开发的
        1. springmvc将url和controller方法映射，映射成功后springmvc生成一个handler对象，对象中只包括了一个method，方法执行结束，形参的参数直接销毁，springmvc的controller的开发类似于service开发
        2. springmvc可以进行单例开发，并且建议使用单例开发，struts2通过类的成员变量接受参数，无法使用单例，只能使用多例开发
        3. struts2速度慢，在于它使用了struts标签，如果使用struts2，建议使用jstl的标签

1. springmvc\_day02
   1. 复习
      1. springmvc框架：
         1. DispatcherServlet前端控制器：接受请求，进行响应
         2. HandlerMapping处理器映射器（根据url查找handler，通过xml或者注解）
         3. HandlerAdapter处理器适配器（根据特定规则执行handler，编写handler时要按照handlerAdapter的要求去编写）
         4. Handler处理器（后端控制器）（需要程序猿编写，常用注解开发方式）
            1. Handler处理器执行后结果是ModelAndView，具体开发时，handler返回的类型包括：ModelAndView、String（逻辑视图名）、void（在handler形参中添加request和response，类似于原始的servlet开发方式，可以通过指定response相应的结果类型实现json数据输出）
         5. 视图解析器：根据逻辑视图名生成真正的视图（在springmvc中使用view对象表示）
            1. view视图：jsp页面，仅是数据的展示，不是业务逻辑。
         6. 注解开发
            1. 使用注解方式的处理器映射器和适配器
            2. 在实际开发中，使用mvc注解驱动标签来代替上边的处理器映射器和适配器
            3. 
            4. @controller注解必须要加，作用表示类是一个handler处理器
            5. @requestMapping:注解必须要加，作用

对url和handler的方法进行映射

可以窄化请求映射，设置handler类的根路径，url就是根路径+子路径

可以限制http请求的方法

* + - * 1. 映射成功后，springmvc框架生成一个handler对象，对象中只包括一个映射成功的method
        2. 注解开发中参数绑定

将request请求过来的key/value的数据(理解成一个串)，通过转换（参数绑定的一部分），=将key/value转成形参，将转换后的结果传给形参

springmvc所支持的参数绑定类型

request，response，session，model、modelMap，将模型数据填充到request域

支持简单数参数类型型，整形，字符串

只要保证request请求的参数名和形参名一致，自动绑定成功

如果request请求的参数名和形参名不一致，可以使用@requestParam（指定request请求的参数名），@requestParam加载形参前面

支持pojo类型

只要保证页面提交的input的名称和pojo中的属性名一致，就会自动将request请求的参数设置到pojo的属性中

形参中既有pojo类型又有简单类型，参数绑定互相不影响

自定义参数绑定：

日期类型绑定自定义：

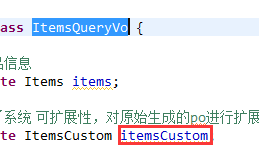
定义的Converter<源类型，目标类型>接口实现类，比如：Converter<String，Date>表示：将请求的日期数据转换成java中的日期类型

注意：要转换的目标类型一定是和我们pojo的属性类型一致

将定义的converter实现类，注入到处理器适配器中。

* 1. 包装类型的参数绑定
     1. 需求

商品查询的controller方法中实现商品查询条件的传入

* + 1. 实现方法
       1. 在形参中添加HttpServletRequest参数，通过request快接受查询条件参数
       2. 在形参中让包装类型的pojo接受查询条件参数
       3. 分析：
          1. 页面参数特点：比较复杂，比较多样性（用户账号 ，商品编号，订单信息）如果将用户账号，商品编号，订单信息等放在简单的pojo（属性是简单类型）中，pojo类属性比较多，比较乱，不方便维护
          2. 建议使用包装类型的pojo，pojo属性中有pojo
    2. 页面的参数和controller方法形参定义
       1. 页面参数
          1. 商品名称：<input name="itemsCustom.name" />
          2. itemsCustom和包装pojo中的属性名一致
       2. controller方法的形参
          1. 
          2. 
          3. 注意：包装类里面要写出get和set的方法
  1. 集合类型的绑定
     1. 数组绑定
        1. 需求

商品批量删除，用户在页面选择多个商品，批量删除

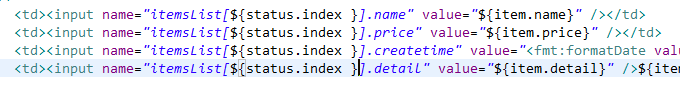
* + - 1. 表现层的实现
         1. 关键：将页面选择的商品的id，传到controller方法的形参中（使用数组接收）
         2. 页面定义

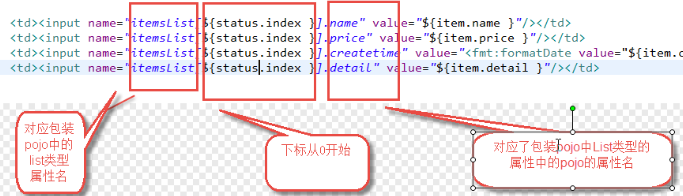


* + - * 1. controller方法的定义



* + 1. list**绑定**
       1. **需求**
          1. 通常需要批量提交数据时，将提交的数据绑定到list<pojo>中，比如：成绩录入（录入多门成绩，批量提交），本离子需求：批量商品修改，在页面输入多个商品的信息，将多个商品信息，将多个信息提交到controller方法中
       2. 表现层实现
          1. 页面定义





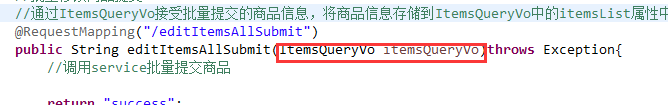
* + - * 1. controller方法定义

进入批量商品修改页面（参考商品列表实现）

批量修改商品提交

使用list接受页面提交的批量数据



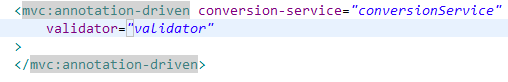
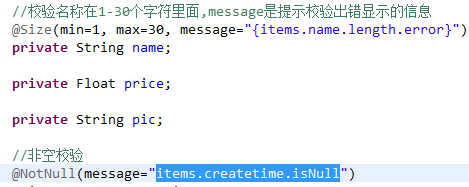
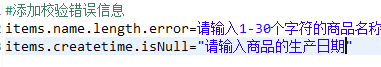
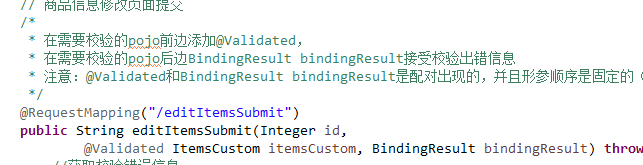


* + 1. map绑定
       1. 也通过在包装pojo中定义map类型
       2. 过程
          1. 在包装类中定义map对象，并添加get/set方法，action使用包装对象接受，包装类中定义map对象如下

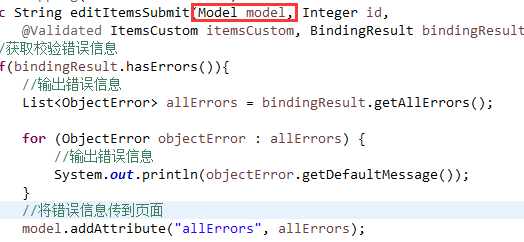
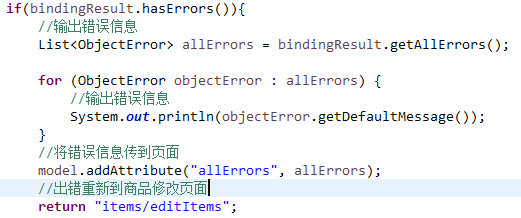
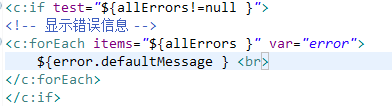
|  |
| --- |
| Public class QueryVo{  Private Map<String, Object> itemInfo = new HashMap< String, Object >;  //get/set方法  } |
| 页面定义如下：  <tr>  <td>学生信息</td>  <td>  姓名：<input type = “text” name = “itemInfo[‘name’]”/>  价格：<input type = “text” name = “itemInfo[‘price’]”/>  </td>  </tr> |
|  |

* 1. 服务端的校验
     1. 校验理解
        1. 项目中，通常使用较多的是前端的校验，比如页面中js的校验。对于安全要求较高的建议在服务端进行校验
        2. 服务端校验
           1. 控制层controller：校验页面请求的参数的合法性，在服务端控制层controller校验，不区分客户端类型（浏览器。手机客户端、远程调用）
           2. 业务层service（使用较多）：主要校验关键业务参数，仅限于service接口中使用的参数
           3. 持久层dao：一般是不校验的
     2. springmvc校验
        1. springmvc使用hibernate的校验框架validation
        2. 思路：
           1. 页面提交请求参数，请求到controller方法中，使用validation进行校验，如果校验出错，将错误信息展示到页面
        3. 需求：
           1. 商品修改，添加校验（校验商品名称长度，生产日期的非空校验），如果校验出错，在商品修改页面显示错误信息
     3. 环境准备
        1. 
     4. 配置校验器,springmvc.xml

|  |
| --- |
| <!-- 校验器 -->  <bean id=*"validator"* class=*"org.springframework.validation.beanvalidation.LocalValidatorFactoryBean"*>  <!-- hibernate校验器 -->  <property name=*"providerClass"* value=*"org.hibernate.validator.HibernateValidator"* />  <!-- 指定校验使用的资源文件，如果不指定则默认使用classpath下的validationMessages.properties -->  <property name=*"validationMessageSource"* ref=*"messageSource"*></property>  </bean>  <!-- 校验错误信息配置文件 -->  <bean id=*"messageSource"* class=*"org.springframework.context.support.ReloadableResourceBundleMessageSource"*>  <!-- 资源文件名 -->  <property name=*"basenames"*>  <list>  <value>classpath:CustomValidationMessages</value>  </list>  </property>  <!-- 资源文件编码格式 -->  <property name=*"fileEncodings"* value=*"utf-8"*></property>  <!-- 对资源文件内容缓存时间，单位秒 -->  <property name=*"cacheSeconds"* value=*"120"*></property>  </bean> |

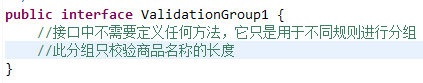
* + 1. 校验器注入到处理器适配器中springmvc.xml
       1. 
    2. 在pojo中添加校验规则
       1. 在itemsCustom.java中添加校验规则
          1. 
       2. CustomValidationMessages.properties
          1. 
    3. 捕获校验错误信息
       1. 

|  |
| --- |
| /\*  \* 在需要校验的pojo前边添加@Validated，  \* 在需要校验的pojo后边BindingResult bindingResult接受校验出错信息  \* 注意：@Validated和BindingResult bindingResult是配对出现的，并且形参顺序是固定的（一前一后）  \*/ |

* + 1. 在页面显示校验错误信息
       1. 
       2. 页面显示错误信息
          1. 
          2. 
    2. 分组校验
       1. 需求

在pojo中定义校验规则，而pojo是被多个controller所共用，当不同的controller方法对同一个pojo进行校验，但是每个controller方法需要不同的校验

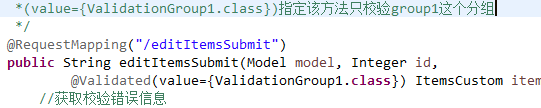
* + - 1. 定义多个校验分组（其实是一个java接口），分组中定义有哪些规则
         1. 每个controller方法使用不同的校验分组
         2. 校验分组



* + - * 1. 在校验规则中添加分组



* + - * 1. 在controller方法中使用指定分组校验



* 1. 数据的回显
     1. 什么是数据回显

提交后如果出现错误，将刚才提交的数据回显到刚才的页面

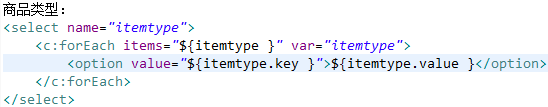
* + 1. pojo数据回显的方法
       1. springmvc默认对pojo的数据进行回显
          1. pojo数据传入controller方法后，springmvc自动将pojo数据放到request域，key等于pojo的类型（首字母小写）
          2. 使用注解标识@ModelAttribute("items") 可以指定pojo回显到页面在request域中的key
       2. @ModelAttribute还可以将方法的返回值也传到页面

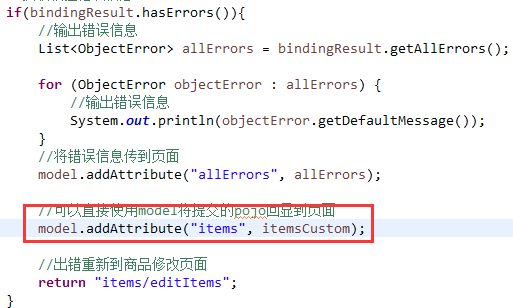
在商品查询列表页面，通过商品类型查询商品信息

* + - * 1. 在controller中定义商品类型查询方法 ，最终将商品类型传到页面



页面上得到itemTypes数据



* + - 1. 最简单的方法就是使用model
         1. 
    1. 简单类型的数据回显
       1. 最简单的方法就是使用model
  1. 异常处理、
     1. 思路

系统中异常包括两类：预期异常和运行时异常runtimeException，前者通过捕获异常从而获取异常信息，后者主要通过规范代码开发、测试通过手段减少运行时异常的发生。

系统的dao、service、controller出现都通过throws Exception向上抛出，最后由springmvc前端控制器交由异常处理器进行异常处理，如下图

客户端

springmvc

dispatcherservlet

controller

service

dao

ExceptionResolver

异常处理器

springmvc提供全局异常处理器（一个系统只有一个异常处理器）进行统一异常处理

* + 1. 自定义异常类

对不同的异常类型定义异常类，继承Exception

* + - 1. 
    1. 全局异常处理器

思路：系统遇到异常，在程序中手动抛出，到抛给service、service给controller、controller抛给前端控制器，前端控制器调用全局异常处理器。

全局异常处理器处理思路：

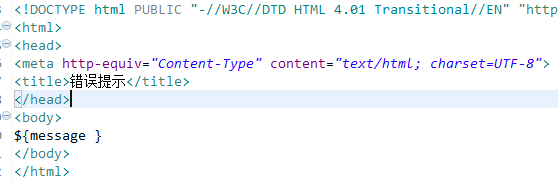
解析出异常类型

如果该 异常类型是系统自定义的异常，直接取出异常信息，在错误页面显示

如果该异常类型不是系统自定义的异常，构造一个自定义的异常类型（信息为“位置错误”）

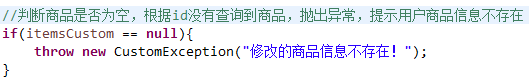
springmvc提供一个HandlerExceptionResolver接口

|  |
| --- |
| **public** **class** CustomExceptionResolver **implements** HandlerExceptionResolver{  /\*  \* ex就是系统抛出的异常  \*/  @Override  **public** ModelAndView resolveException(HttpServletRequest request,  HttpServletResponse response, Object handler, Exception ex) {  //handler就是处理器适配器要执行的handler对象（只有一个method方法）      // 解析出异常类型  // String message = null;  //// \* 如果该 异常类型是系统自定义的异常，直接取出异常信息，在错误页面显示  // if(ex instanceof CustomException){  // message = ((CustomException)ex).getMessage();  // }  //// \* 如果该异常类型不是系统自定义的异常，构造一个自定义的异常类型（信息为“位置错误”）  // else{  // message = "未知错误";  // }    //上边代码变换为  CustomException customException = **null**;  **if**(ex **instanceof** CustomException){  customException = (CustomException)ex;  }**else**{  customException = **new** CustomException("未知错误");  }    //错误信息  String message = customException.getMessage();    ModelAndView modelAndView = **new** ModelAndView();    //将错误信息传导页面  modelAndView.addObject("message", message);    //指向错误页面  modelAndView.setViewName("error");    **return** modelAndView;  } |

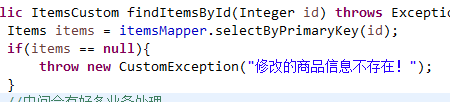
* + 1. 错误页面
       1. 
    2. 在springmvc.xml配置全局异常处理器



* + 1. 异常测试
       1. 在controller，service，dao中任意一处需要手动抛出异常。如果是程序中手动抛出的异常，在错误页面中显示自定义的异常信息，如果不是手动抛出异常，说明是一个运行时异常，在错误页面只显示“未知错误”
       2. 在商品修改的controller方法中抛出异常



* + - 1. 在service方法中抛出异常

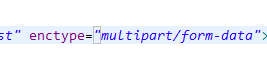


如果与业务功能相关的异常，建议在service中抛出异常

如果与业务功能没有关系的异常，建议在controller中抛出

上边的功能，建议在service中的抛出异常。

* 1. 上传图片
     1. 需求
        1. 在修改商品页面添加上传商品图片的功能
     2. springmvc中对多部件类型的解析
        1. 在jsp的form中提交enctype=”multipart/form-data”的数据时，需要springmvc对multipart类型的数据进行解析。

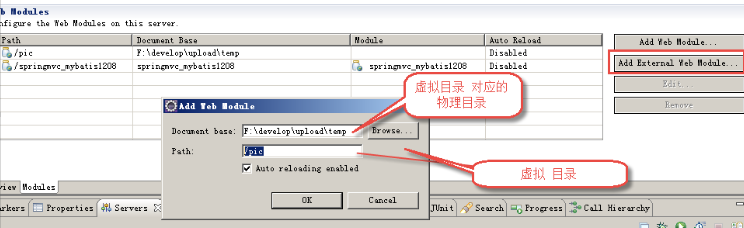


* + - 1. 在springmvc.xml中配置multipart类型解析器

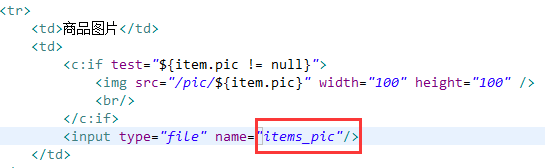
|  |
| --- |
| <!-- 文件上传 -->  <bean id=*"multipartResolver"*  class=*"org.springframework.web.multipart.commons.CommonsMultipartResolver"*>  <!-- 设置上传文件的最大尺寸为5mb -->  <property name=*"maxUploadSize"*>  <value>5242880</value>  </property>  </bean> |

* + - 1. **注意：如果你配置了enctype属性，但是没有在springmvc配置文件中配置multipart解析器，那么springmvc的参数绑定就会失败**
    1. 加入上传图片的jar包

上面的解析器就是用了这里的jar包

* + 1. 创建图片的虚拟目录，存储图片
       1. 图形界面配置
       2. 也可以直接修改tomcat的配置文件
          1. 在conf/server.xml中，添加虚拟目录



* + - * 1. 注意：在图片虚拟目录中，一定将图片目录分级创建（提高i/o性能），一般采用按日期进行分级创建
    1. 上传图片代码
       1. 修改：修改商品的controller
          1. 
       2. jsp页面代码
          1. 
  1. json数据交互
     1. 为什么要进行json数据交互

json数据格式在接口调用中，html页面中较常用，json格式比较简单，解析还比较方便。

比如：webservice接口，传输json数据

* + 1. springmvc进行json交互

客户端请求

请求的是json串

contentType=application/json(需要指定)

@RequestBody将json串转成java对象

@ResponseBody将java对象转成json串输出

请求的是key/value

contentType=application/x-www-form-urlencoded

不需要@RequestBody将json串转成java对象

@ResponseBody将java对象转成json串输出

最终都输出json数据，为了在前端页面方便对请求结果进行解析

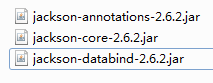
* + - 1. 请求json，输出的也是json，要求请求的是json，所以在前端页面中需要将请求的内容转成json，不太方便
      2. 请求key/value，输出json。此方法比较常用
    1. 环境准备
       1. 加载json的jar包

（spring4.0以下）

springmvc默认用mappingjacksonHttpMessageConverter对json数据进行转换，需要加入jackson的包（requestBody和responseBody），如下



spring4.0以上：



* + - 1. 配置转换器

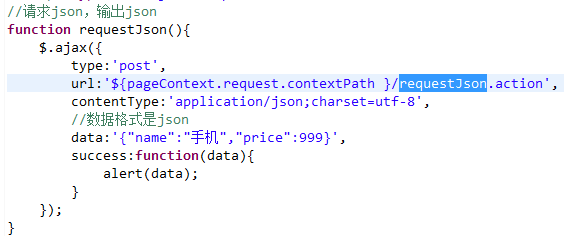
在注解适配器中加入messageConverter

|  |
| --- |
| <!-- 注解适配器 -->  <bean class=*"org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerAdapter"*>  <property name=*"messageConverters"*>  <list>  <bean class=*"org.springframework.http.converter.json.MappingJackson2HttpMessageConverter"*></bean>  </list>  </property>  </bean> |

注意：如果使用<mvc:annotation-driven/>则不用定义上边的内容

* + 1. json交互的测试
       1. 输入json串，输出的json串
          1. jsp页面

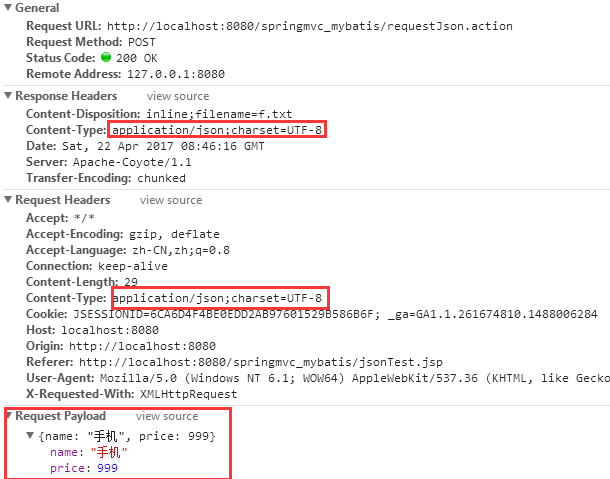
使用jquery的ajax提交json串，对输出的json结果进行解析



* + - * 1. controller



* + - * 1. 测试结果





* + - 1. 输入key/value，输出的是json串
         1. jsp页面



* + - * 1. controller



* + - * 1. 测试结果





* 1. RESTful支持
     1. 什么是RESTful（representation state transfer表现层的状态转换）

RESTful架构，就是目前最流行的一种互联网软件架构，它结构清晰，符合标准，易于理解，扩展方便，所以正得到越来越多的网站的采用

RESTful其实就是一个开发理念，是对http的很好的诠释

* + - 1. 对url进行规范，写RESTful格式的url

非RESTful的url：<http://......../queryItems.action?id=001&type=T01>

RESTful的<url:http://......./items/001>

特点：url简洁，将参数通过url传到服务器

不管是删除、添加、更新。。使用url是一致的，如果进行删除，需要设置http的方法为delete

* + - 1. 后台controller方法：判断http方法，如果是delete执行删除，如果是post执行添加
      2. 对http的contentType规范
         1. 请求是指定contentType，要json数据，设置成json格式的type
    1. RESTful的例子
       1. 需求

查询商品的信息，放回json数据

* + - 1. controller方法

定义方法，进行url映射使用REST风格的url，将查询商品的信息的id传入controller

输出json，使用@responseBody将java对线输出json

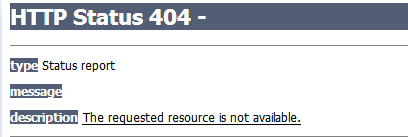


* + - 1. 前端控制器的配置



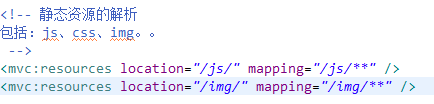
* + 1. 对静态资源的解析

配置前端控制器的url-partten中指定/，对静态资源的解析出现问题



解决：

在springmvc.xml中添加静态资源解析的方法



* 1. 拦截器
     1. 拦截器的定义

定义拦截器，实现handlerInterceptor接口，接口中实现三个方法

|  |
| --- |
| //进入handler方法之前  //用于身份认证，身份授权  //比如身份认证不通过，便是当前用户没有登录，需要此方法拦截不在向下执行  @Override  **public** **boolean** preHandle(HttpServletRequest request,  HttpServletResponse response, Object handler) **throws** Exception {  // **TODO** Auto-generated method stub  //return false表示拦截，return true表示放行  **return** **false**;  }  //进入handler方法之后，返回modelandview之前  //应用场景从modelAndView出发：将公用的模型数据（菜单的导航）在这里传到视图  @Override  **public** **void** postHandle(HttpServletRequest request,  HttpServletResponse response, Object handler,  ModelAndView modelAndView) **throws** Exception {  // **TODO** Auto-generated method stub    }  //执行handler方法之后完成，执行此方法  //应用场景：统一异常处理，统一日志处理，  @Override  **public** **void** afterCompletion(HttpServletRequest request,  HttpServletResponse response, Object handler, Exception ex)  **throws** Exception {  // **TODO** Auto-generated method stub    } |

* + 1. 拦截器的配置
       1. springMvc拦截器是针对handlerMapping进行拦截设置

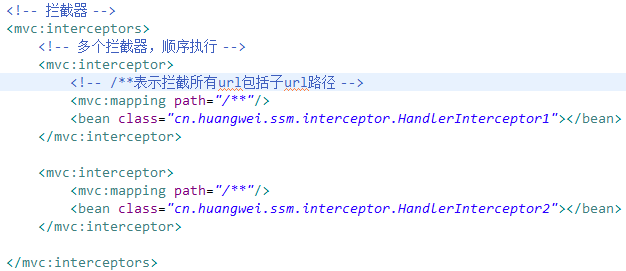
如果在某个handlermapping中配置拦截，经过该hanflerMapping映射成功的handler最终使用该拦截器。

|  |
| --- |
| <bean class=*"org.springframework.web.servlet.handler.BeanNameUrlHandlerMapping"*>  <property name=*"interceptors"*>  <list>  <ref bean=*"handlerInterceptor1"* />  <ref bean=*"handlerInterceptor2"* />  </list>  </property>  </bean>    <bean id=*"handlerInterceptor1"* class=*"cn.huangwei.ssm.interceptor.HandlerInterceptor1"*></bean>  <bean id=*"handlerInterceptor2"* class=*"cn.huangwei.ssm.interceptor.HandlerInterceptor2"*></bean> |

一般不推荐使用

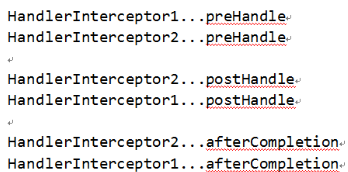
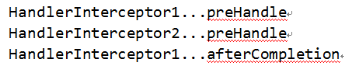
* + - 1. springmvc配置类似全局的拦截器

springmvc框架将配置的类似的全局拦截器注入到每个handlermapping中



* + 1. 拦截器的测试
       1. 测试需求

测试多个拦截器各个方法的执行时机

* + - 1. 编写两个拦截器
      2. 两个拦截器都放行
         1. prehandler方法按顺序执行，postHandle和afterCompletion按拦截器配置逆向顺序执行
      3. 拦截器1放行，拦截器2不放行
         1. 
         2. 总结：

拦截器1放行，拦截器2prehandle才会执行

拦截器2prehandle不放行，拦截器2postHandle和afterCompletion不会执行

只要有一个拦截器不放行，postHandle就不会执行

* + - 1. 拦截器1不放行，拦截器2不放行
         1. 
         2. 拦截器1不放行，postHandle和afterCompletion不会执行，
         3. 拦截器2也不执行
      2. 小结

根据测试结果，对拦截器应用

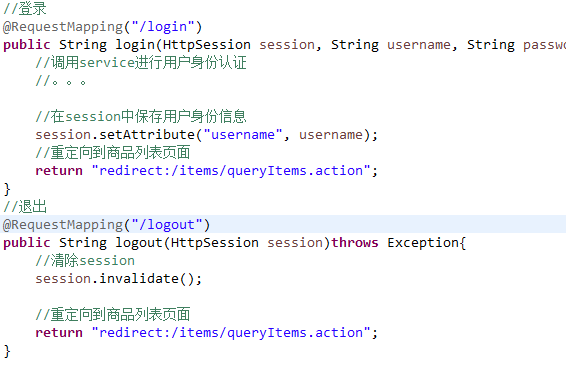
* + - * 1. 比如:统一日志处理拦截器，需要该拦截器prehandle一定要放行，且将它方法拦截器链接中的第一个位置
        2. 比如：登录认证拦截器，放在拦截器链接中的第一个位置，权限校验拦截器，放在登录认证拦截器之后。（因为登录通过后才校验权限）
    1. 拦截器的应用（实现登录认证）
       1. 需求
          1. 用户请求url
          2. 拦截器

如果请求的url是公开地址（无需登录即可访问的url），放行

如果用户session不存在跳转到登录页面

如果用户session存在放行，继续操作

* + - 1. 登录controller方法



* + - 1. 登录的jsp页面



* + - 1. 登录认证拦截实现

代码实现

|  |
| --- |
| **public** **boolean** preHandle(HttpServletRequest request,  HttpServletResponse response, Object handler) **throws** Exception {  //获取请求的url，判断是不是公开地址  String url = request.getRequestURI();  //判断url是否是公开的地址（实际使用时将公开地址配置成配置文件）  //这里公开地址是登录提交的地址  **if**(url.indexOf("login.action")>=0){  **return** **true**;  }    //判断session  HttpSession session = request.getSession();    //从session中获取用户信息  String username = (String) session.getAttribute("username");    **if**(username != **null**){  **return** **true**;  }    //执行到这里表示用户信息认证不通过，需要重新认证，跳转登录页面  request.getRequestDispatcher("/WEB-INF/jsp/login.jsp").forward(request, response);  //return false表示拦截，return true表示放行  **return** **true**;  } |

配置拦截器springmvc.xml

