springmvc 第一、二天 springmvc的基础知识

课程安排：

第一、二天：springmvc的基础知识

什么是springmvc？

springmvc框架原理（掌握）

前端控制器、处理器映射器、处理器适配器、视图解析器

springmvc入门程序

目的：对前端控制器、**处理器映射器、处理器适配器**、视图解析器学习

非注解的处理器映射器、处理器适配器

注解的处理器映射器、处理器适配器（掌握）

springmvc和mybatis整合（掌握）

springmvc注解开发：（掌握）

常用的注解学习

参数绑定（简单类型、pojo、集合类型）

自定义参数绑定（掌握）

springmvc和struts2区别

第三、四天：springmvc的高级应用

参数绑定（集合类型）

数据回显

上传图片

json数据交互

RESTful支持

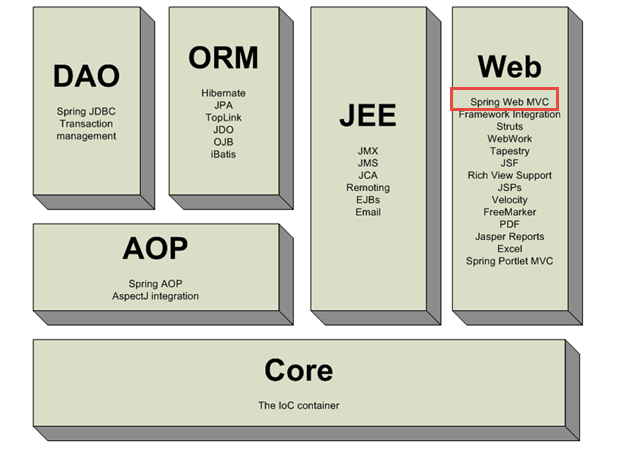
拦截器

# springmvc框架

## 什么是springmvc

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。

springmvc是一个基于mvc的web框架。



## mvc在b/s系统 下的应用

mvc是一个设计模式，mvc在b/s系统 下的应用：

C 控制器

比如struts2中的filter

controller

接收用户请求，

响应

request请求

M 模型(model)

pojo

action

service

dao

请求模型进行处理

处理结果返回

V 视图

view

视图渲染

将模型数据填充到request域

response响应

b/s系统下模型无法将数据填充到视图

## springmvc框架

1request请求 url

前端控制器

DispatcherServlet

接收用户请求，响应

Handler处理器

平常叫做controller

处理器映射器

HandlerMapping

2请求查找Handler

3返回一个执行链

HandlerExecutionChain{

HandlerInterceptor1 HandlerInterceptor2

Handler

}

处理器适配器

HandlerAdapter

去执行Handler

4请求适配器执行Handler

5执行

6返回ModelAndView

7返回ModelAndView

view视图

jsp

freemarker

excel

pdf

视图解析器

View resolver

8请求进行视图解析

9返回view

10视图渲染

将模型数据填充到request域

11response响应

第一步：发起请求到前端控制器(DispatcherServlet)

第二步：前端控制器请求HandlerMapping查找 Handler

可以根据xml配置、注解进行查找

第三步：处理器映射器HandlerMapping向前端控制器返回执行器链（包含Handler）

第四步：前端控制器调用处理器适配器去执行Handler

第五步：处理器适配器去执行Handler

第六步：Handler执行完成给适配器返回ModelAndView

第七步：处理器适配器向前端控制器返回ModelAndView

ModelAndView是springmvc框架的一个底层对象，包括 Model和view

第八步：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)

第九步：视图解析器向前端控制器返回View

第十步：前端控制器进行视图渲染

视图渲染将模型数据(在ModelAndView对象中)填充到request域

第十一步：前端控制器向用户响应结果

组件：

1、前端控制器DispatcherServlet（不需要程序员开发）

作用接收请求，响应结果，相当于转发器，中央处理器。

有了DispatcherServlet减少了其它组件之间的耦合度。

2、处理器映射器HandlerMapping(不需要程序员开发)

作用：根据请求的url查找Handler

3、处理器适配器HandlerAdapter

作用：按照特定规则（HandlerAdapter要求的规则）去执行Handler

4、处理器Handler(需要程序员开发)

注意：编写Handler时按照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View resolver(不需要程序员开发)

作用：进行视图解析，根据逻辑视图名解析成真正的视图（view）

6、视图View(需要程序员开发jsp)

View是一个接口，实现类支持不同的View类型（jsp、freemarker、pdf...）

# 入门程序

## 需求

以案例作为驱动。

springmvc和mybaits使用一个案例（商品订单管理）。

功能需求：商品列表查询

## 环境准备

数据库环境：mysql5.1



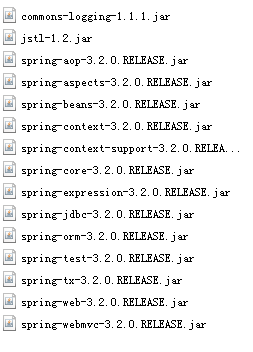
java环境：

jdk1.7.0\_72

myeclipse

springmvc版本：spring3.2

需要spring3.2所有jar（一定包括spring-webmvc-3.2.0.RELEASE.jar）



## 配置前端控制器

在web.xml中配置前端控制器。



## 配置处理器适配器

在springmvc.xml文件配置如下：

<beans xmlns=*"http://www.springframework.org/schema/beans"*

xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"* xmlns:mvc=*"http://www.springframework.org/schema/mvc"*

xmlns:context=*"http://www.springframework.org/schema/context"*

xmlns:aop=*"http://www.springframework.org/schema/aop"* xmlns:tx=*"http://www.springframework.org/schema/tx"*

xsi:schemaLocation=*"http://www.springframework.org/schema/beans*

*http://www.springframework.org/schema/beans/spring-beans-3.2.xsd*

*http://www.springframework.org/schema/mvc*

*http://www.springframework.org/schema/mvc/spring-mvc-3.2.xsd*

*http://www.springframework.org/schema/context*

*http://www.springframework.org/schema/context/spring-context-3.2.xsd*

*http://www.springframework.org/schema/aop*

*http://www.springframework.org/schema/aop/spring-aop-3.2.xsd*

*http://www.springframework.org/schema/tx*

*http://www.springframework.org/schema/tx/spring-tx-3.2.xsd "*>

<!-- 处理器映射器 -->

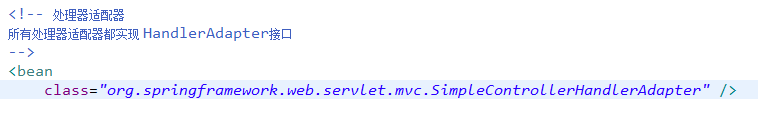
<!-- 根据bean的name进行查找Handler 将action的url配置在bean的name中 -->

<bean

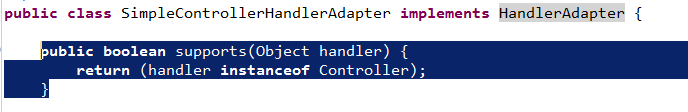
class=*"**org.springframework.web.servlet.handler.BeanNameUrlHandlerMapping"* />

BeanNameUrlHandlerMapping：表示将定义的Bean名字作为请求的url，需要将编写的controller在spring容器中进行配置，且指定bean的name为请求的url，且必须以.action结尾。

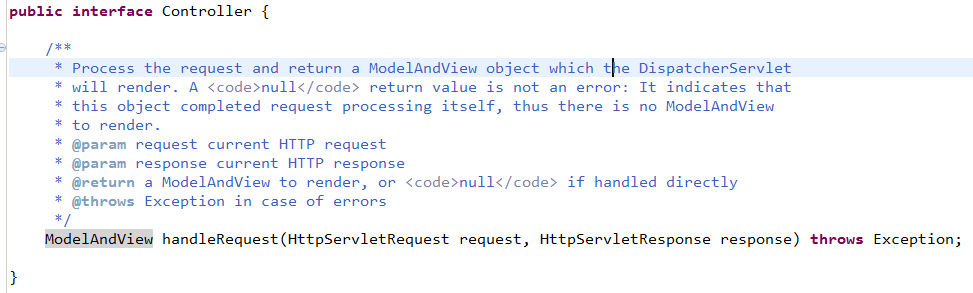
在classpath下的springmvc.xml中配置处理器适配器



通过查看原代码：



此适配器能执行实现 Controller接口的Handler。



## 开发Handler

需要实现 controller接口，才能由org.springframework.web.servlet.mvc.SimpleControllerHandlerAdapter适配器执行。

**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中填充静态数据

Items items\_1 = **new** Items();

items\_1.setName("联想笔记本");

items\_1.setPrice(6000f);

items\_1.setDetail("ThinkPad T430 联想笔记本电脑！");

Items items\_2 = **new** Items();

items\_2.setName("苹果手机");

items\_2.setPrice(5000f);

items\_2.setDetail("iphone6苹果手机！");

itemsList.add(items\_1);

itemsList.add(items\_2);

//返回ModelAndView

ModelAndView modelAndView = **new** ModelAndView();

//相当 于request的setAttribut，在jsp页面中通过itemsList取数据

modelAndView.addObject("itemsList", itemsList);

//指定视图

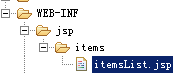
modelAndView.setViewName("/WEB-INF/jsp/items/itemsList.jsp");

**return** modelAndView;

}

}

## 视图编写



## 配置Handler

将编写Handler在spring容器加载。



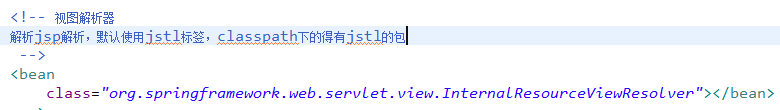
## 配置处理器映射器

在classpath下的springmvc.xml中配置处理器映射器



## 配置视图解析器

需要配置解析jsp的视图解析器。



## 部署调试

访问地址：http://localhost:8080/springmvcfirst/queryItems.action

处理器映射器根据url找不到Handler，报下边的错误。说明url错误。

处理器映射器根据url找到了Handler，转发的jsp页面找到，报下边的错误，说明jsp页面地址错误了。

# 非注解的处理器映射器和适配器

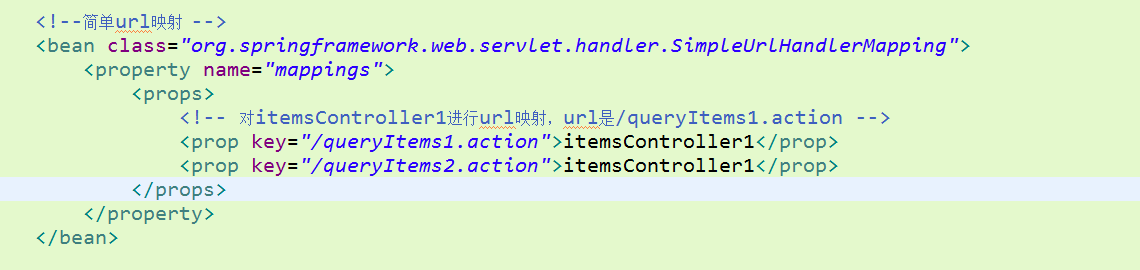
## 非注解的处理器映射器

处理器映射器：

org.springframework.web.servlet.handler.BeanNameUrlHandlerMapping

另一个映射器：

org.springframework.web.servlet.handler.SimpleUrlHandlerMapping



多个映射器可以并存，前端控制器判断url能让哪些映射器映射，就让正确的映射器处理。

## 非注解的处理器适配器

org.springframework.web.servlet.mvc.SimpleControllerHandlerAdapter

要求编写的Handler实现 Controller接口。

org.springframework.web.servlet.mvc.HttpRequestHandlerAdapter

要求编写的Handler实现 HttpRequestHandler接口。



//使用此方法可以通过修改response，设置响应的数据格式，比如响应json数据

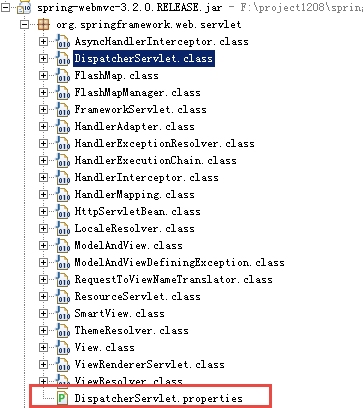
/\*

response.setCharacterEncoding("utf-8");

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

response.getWriter().write("json串");\*/

# DispatcherSerlvet.properties

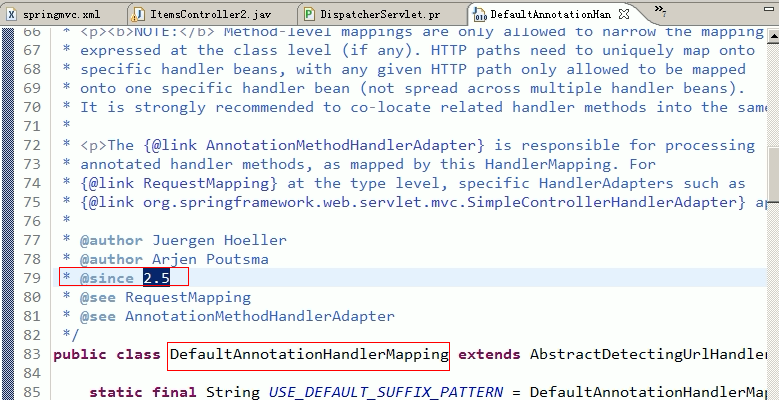


前端控制器从上边的文件中加载处理映射器、适配器、视图解析器等组件，如果不在springmvc.xml中配置，使用默认加载的。

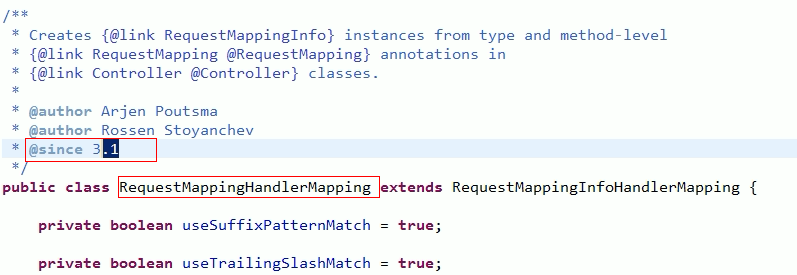
# 注解的处理器映射器和适配器

进源码看一下，里面注释的有每个映射器是在哪个版本中应用的。

在spring3.1之前使用org.springframework.web.servlet.mvc.annotation.DefaultAnnotationHandlerMapping注解映射器。

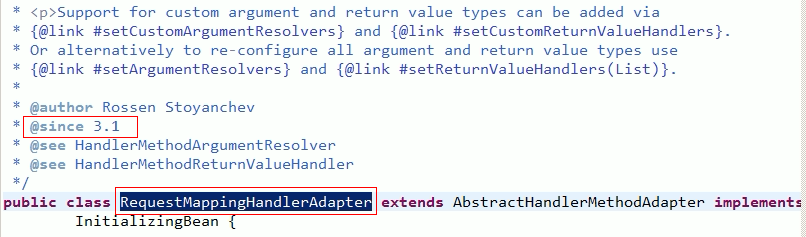


在spring3.1之后使用org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerMapping注解映射器。

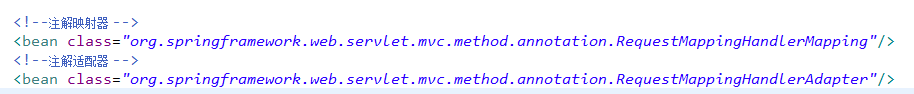


在spring3.1之前使用org.springframework.web.servlet.mvc.annotation.AnnotationMethodHandlerAdapter注解适配器。

在spring3.1之后使用org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerAdapter注解适配器。



## 配置注解映射器和适配器。



<!-- 使用 mvc:annotation-driven代替上边注解映射器和注解适配器配置

mvc:annotation-driven默认加载很多的参数绑定方法，

比如json转换解析器就默认加载了，如果使用mvc:annotation-driven不用配置上边的RequestMappingHandlerMapping和RequestMappingHandlerAdapter

实际开发时使用mvc:annotation-driven

-->

<!-- <mvc:annotation-driven></mvc:annotation-driven> -->

## 开发注解Handler

使用注解的映射器和注解的适配器。（注解的映射器和注解的适配器必须配对使用）

//使用Controller标识 它是一个控制器

@Controller

**public** **class** ItemsController3 {

//商品查询列表

//@RequestMapping实现 对queryItems方法和url进行映射，一个方法对应一个url

//一般建议将url和方法写成一样

@RequestMapping("/queryItems")

**public** ModelAndView queryItems()**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中填充静态数据

Items items\_1 = **new** Items();

items\_1.setName("联想笔记本");

items\_1.setPrice(6000f);

items\_1.setDetail("ThinkPad T430 联想笔记本电脑！");

Items items\_2 = **new** Items();

items\_2.setName("苹果手机");

items\_2.setPrice(5000f);

items\_2.setDetail("iphone6苹果手机！");

itemsList.add(items\_1);

itemsList.add(items\_2);

//返回ModelAndView

ModelAndView modelAndView = **new** ModelAndView();

//相当 于request的setAttribut，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;

}

## 在spring容器中加载Handler

<!-- 对于注解的Handler可以单个配置

实际开发中建议使用组件扫描

-->

<!-- <bean class="cn.hr.ssm.controller.ItemsController3" /> -->

<!-- 可以扫描controller、service、...

这里让扫描controller，指定controller的包

-->

<context:component-scan base-package=*"cn.hr.ssm.controller"*></context:component-scan>

## 部署调试

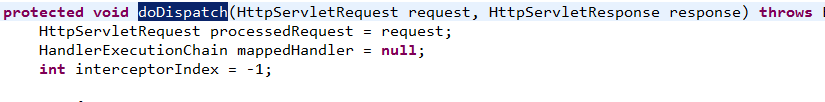
访问：http://localhost:8080/springmvcfirst/queryItems.action

# 源码分析（了解）

通过前端控制器源码分析springmvc的执行过程。

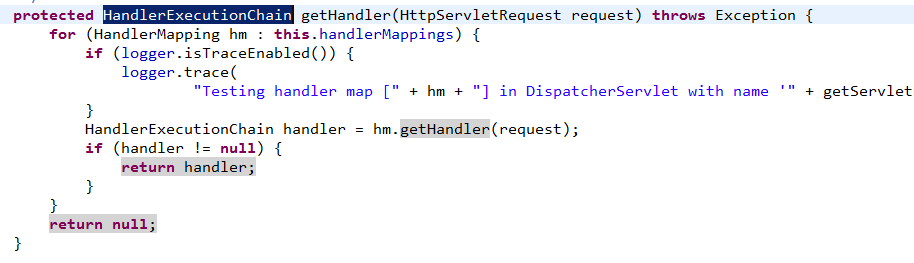
第一步：前端控制器接收请求

调用doDiapatch

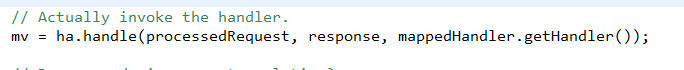


第二步：前端控制器调用处理器映射器查找 Handler





第三步：调用处理器适配器执行Handler，得到执行结果ModelAndView



第四步：视图渲染，将model数据填充到request域。

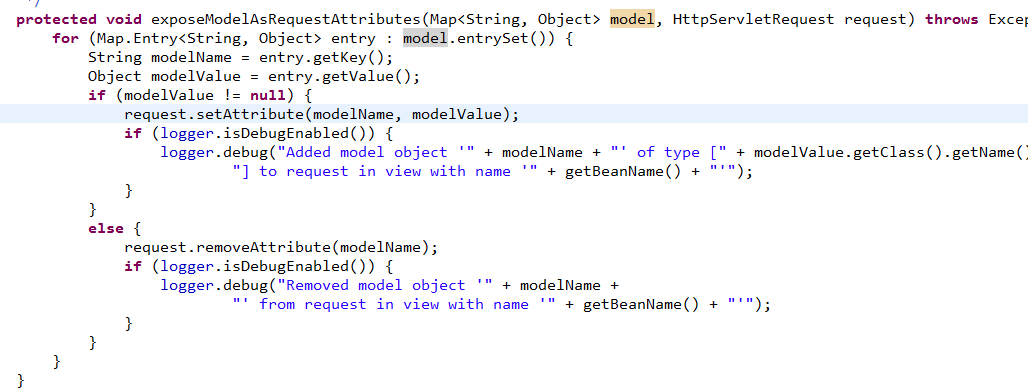
视图解析，得到view:



调用view的渲染方法，将model数据填充到request域

渲染方法：





# 入门程序小结

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**、视图解析器用法。

前端控制器配置：

第一种：\*.action，访问以.action结尾 由DispatcherServlet进行解析

第二种：/，所以访问的地址都由DispatcherServlet进行解析，对于静态文件的解析需要配置不让DispatcherServlet进行解析

使用此种方式可以实现 RESTful风格的url

**处理器映射器：**

**非注解处理器映射器（了解）**

**注解的处理器映射器（掌握）**

**对标记@Controller类中标识有@RequestMapping的方法进行映射。在@RequestMapping里边定义映射的url。使用注解的映射器不用在xml中配置url和Handler的映射关系。**

**处理器适配器：**

**非注解处理器适配器（了解）**

**注解的处理器适配器（掌握）**

**注解处理器适配器和注解的处理器映射器是配对使用。理解为不能使用非注解映射器进行映射。**

<mvc:annotation-driven></mvc:annotation-driven>可以代替下边的配置：

<!--注解映射器 -->

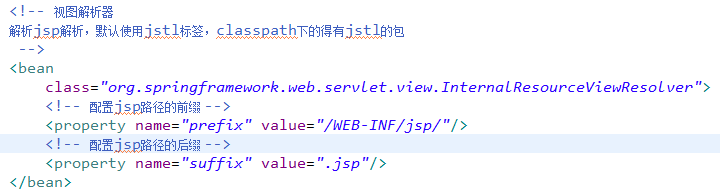
<bean class=*"org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerMapping"*/>

<!--注解适配器 -->

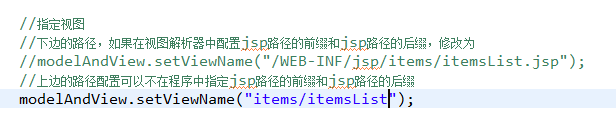
<bean class=*"org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerAdapter"*/>

实际开发使用：mvc:annotation-driven

视图解析器配置前缀和后缀：



程序中不用指定前缀和后缀：



# Springmvc、Spring和mybatis整合

## 需求

使用springmvc、spring和mybatis完成商品列表查询。

## 整合思路

springmvc+mybaits的系统架构：

表现层

springmvc

业务层

service接口

持久层

mybaits

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service，service中可以调用mapper接口。

spring进行事务控制。

通过spring管理表现层Handler，Handler中可以调用service接口。

mapper、service、Handler都是javabean。

第一步：整合dao层

mybatis和spring整合，通过spring管理mapper接口。

使用mapper的扫描器自动扫描mapper接口在spring中进行注册。

第二步：整合service层

通过spring管理 service接口。

使用配置方式将service接口配置在spring配置文件中。

实现事务控制。

第三步：整合springmvc

由于springmvc是spring的模块，不需要整合。

## 准备环境

数据库环境：mysql5.7



java环境：

jdk1.7.0\_72

myeclipse

springmvc版本：spring3.2

所需要的jar包：

数据库驱动包：mysql5.1

mybatis的jar包

mybatis和spring整合包

log4j包

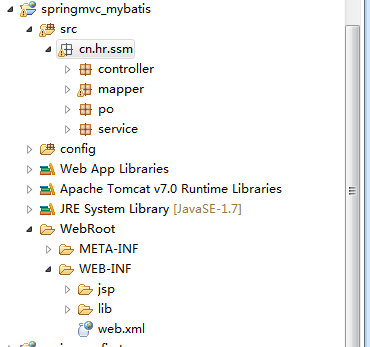
dbcp数据库连接池包

spring3.2所有jar包

jstl包

参考：

工程结构：

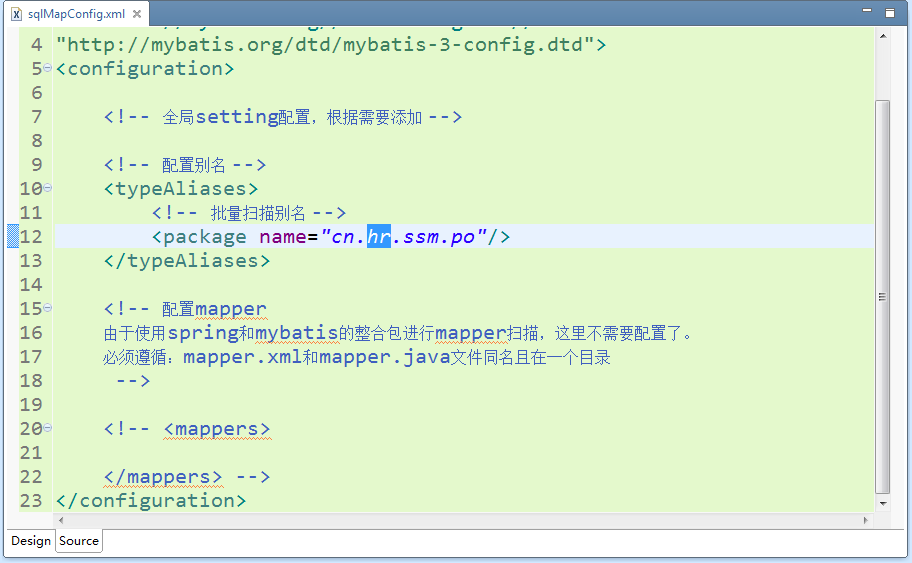


## 整合dao

mybatis和spring进行整合。

### sqlMapConfig.xml

mybatis自己的配置文件。



### applicationContext-dao.xml

配置：

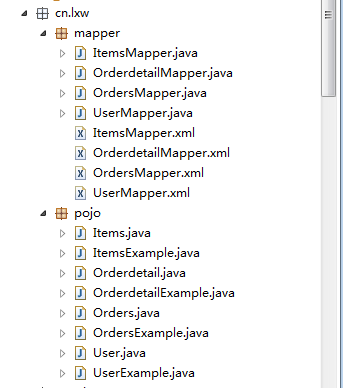
数据源

SqlSessionFactory

mapper扫描器



### 逆向工程生成po类及mapper(单表增删改查)



将生成的文件拷贝至工程 中。

### 手动定义商品查询mapper

针对综合查询mapper，一般情况会有关联查询，建议自定义mapper

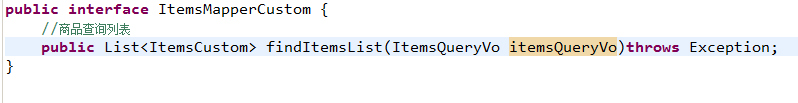
#### ItemsMapperCustom.xml

sql语句：

SELECT \* FROM items WHERE items.name LIKE '%笔记本%'



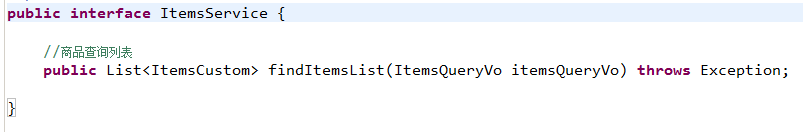
#### ItemsMapperCustom.java

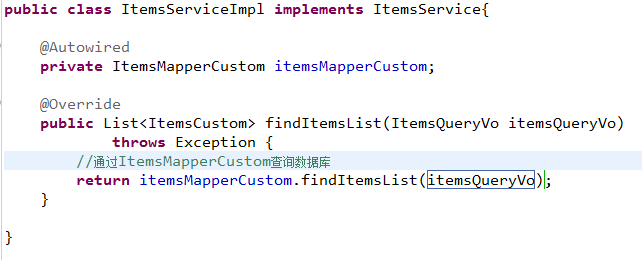


## 整合service

让spring管理service接口。

### 定义service接口





### 在spring容器配置service(applicationContext-service.xml)

创建applicationContext-service.xml，文件中配置service。



### 事务控制(applicationContext-transaction.xml)

在applicationContext-transaction.xml中使用spring声明式事务控制方法。



## 整合springmvc

### springmvc.xml

创建springmvc.xml文件，配置处理器映射器、适配器、视图解析器。

<!-- 可以扫描controller、service、...

这里让扫描controller，指定controller的包

-->

<context:component-scan base-package=*"cn.hr.ssm.controller"*></context:component-scan>

<!--注解映射器 -->

<!-- <bean class="org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerMapping"/> -->

<!--注解适配器 -->

<!-- <bean class="org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerAdapter"/> -->

<!-- 使用 mvc:annotation-driven代替上边注解映射器和注解适配器配置

mvc:annotation-driven默认加载很多的参数绑定方法，

比如json转换解析器就默认加载了，如果使用mvc:annotation-driven不用配置上边的RequestMappingHandlerMapping和RequestMappingHandlerAdapter

实际开发时使用mvc:annotation-driven

-->

<mvc:annotation-driven></mvc:annotation-driven>

<!-- 视图解析器

解析jsp解析，默认使用jstl标签，classpath下的得有jstl的包

-->

<bean

class=*"org.springframework.web.servlet.view.InternalResourceViewResolver"*>

<!-- 配置jsp路径的前缀 -->

<property name=*"prefix"* value=*"/WEB-INF/jsp/"*/>

<!-- 配置jsp路径的后缀 -->

<property name=*"suffix"* value=*".jsp"*/>

</bean>

### 配置前端控制器

参考入门程序。

### 编写Controller(就是Handler)



### 编写jsp



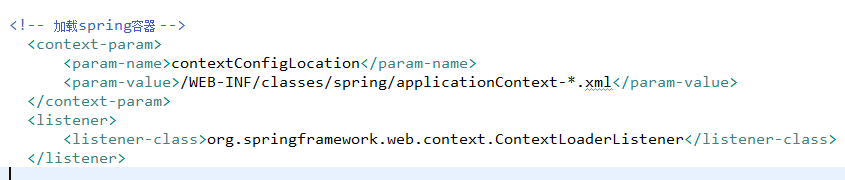
## 加载spring容器

将mapper、service、controller加载到spring容器中。



建议使用通配符加载上边的配置文件。

在web.xml中，添加spring容器监听器，加载spring容器。



# 商品修改功能的开发

需求：

1. 进入商品列表页面
2. 点击修改按钮，进入修改商品页面。修改商品页面的数据是从数据库中获取的，根据商品ID获取商品信息，把商品信息展示到商品修改页面。
3. 在修改商品页面可以修改商品的信息，修改之后点击提交，完成商品提交功能。

## 商品修改功能Mapper层

1. 根据商品id获取商品信息
2. 修改商品信息

单表操作，可以不用开发，用逆向工程生成代码即可。

## 商品修改功能Service层

1. 根据商品ID获取商品信息
2. 接收的参数，int 商品ID
3. 返回的结果为，商品类型 Items
4. 修改商品信息
5. 接收的参数，Items 商品对象
6. 返回的结果为空
7. 代码实现

|  |
| --- |
| @Override  **public** List<ItemsCustom> getItemCustomList() {  // **TODO** Auto-generated method stub  **return** itemCustomMapper.getItemListCustom();  }  @Override  **public** Items findItemById(**int** id) {  **return** itemsMapper.selectByPrimaryKey(id);  } |

## 商品修改功能的Controller层

1. 根据商品id获取商品信息
2. 映射的URL：/items/editItems



b)接收的参数：商品列表页面传来的商品ID

c)返回的结果：商品对象（Items）

1. 修改商品信息
2. 映射的url：/items/editItemsSubmit



b)请求的参数为：商品表单的内容及商品对象（Items）

c)响应的内容为：String，逻辑视图

1. 代码实现

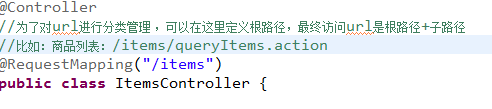
|  |
| --- |
| //根据商品id获取商品信息  @RequestMapping("editItems")  **public** ModelAndView findItemsById(@RequestParam("id") Integer itemId){  ModelAndView mv = **new** ModelAndView();  Items items = itemService.findItemById(itemId);  //添加模型数据  mv.addObject("itemsCustom", items);  //设置视图名称  mv.setViewName("items/editItems");  **return** mv;  }    //修改商品信息  @RequestMapping("editItemsSubmit")  **public** String updateItems(Items items){  itemService.updateItem(items);  **return** "redirect:queryItem.action";  } |

# @RequestMapping

* url映射

定义controller方法对应的url，进行处理器映射使用。

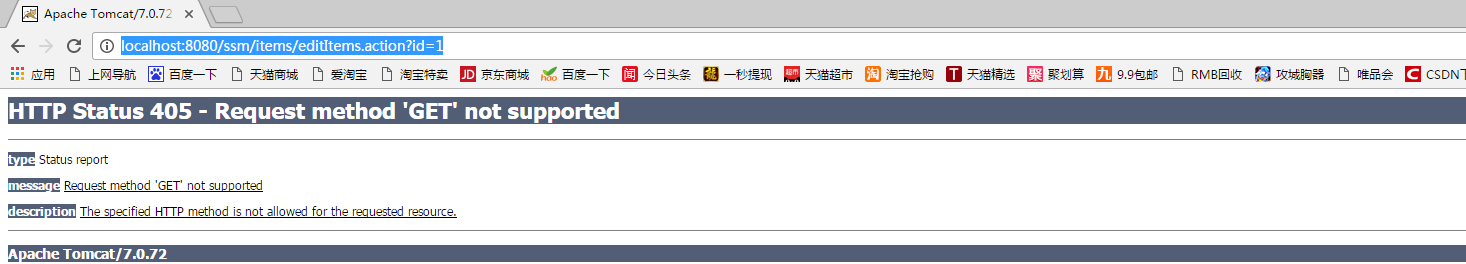
* 窄化请求映射



* 限制http请求方法

出于安全性考虑，对http的链接进行方法限制。

如果限制请求为post方法，进行get请求，报错：





# controller方法的返回值

* 返回ModelAndView

需要方法结束时，定义ModelAndView，将model和view分别进行设置。

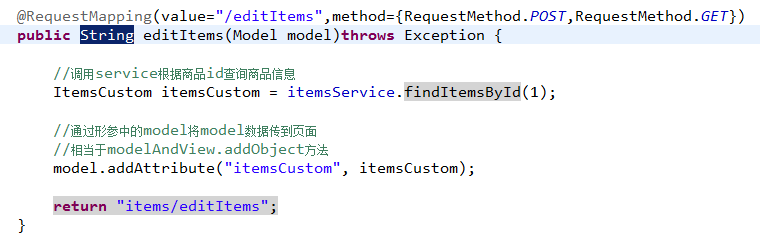
* 返回string

如果controller方法返回string，

1、表示返回逻辑视图名。

真正视图(jsp路径)=前缀+逻辑视图名+后缀=/WEB-INF/jsp/items/editItems.jsp





2、redirect重定向

商品修改提交后，重定向到商品查询列表。

redirect重定向特点：浏览器地址栏中的url会变化。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）



3、forward页面转发

通过forward进行页面转发，浏览器地址栏url不变，request可以共享。



* 返回void(了解)

在controller方法形参上可以定义request和response，使用request或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);

源码：

|  |
| --- |
| //修改商品信息  @RequestMapping("editItemsSubmit")  **public** **void** updateItems(Items items,HttpServletRequest request,HttpServletResponse response){  itemService.updateItem(items);  **try** {  // request.getRequestDispatcher("queryItem.action").forward(request, response);  response.sendRedirect("queryItem.action");  } **catch** (Exception e) {  e.printStackTrace();  }    } |

2、也可以通过response页面重定向：

response.sendRedirect("url")

3、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");

# 参数绑定

## spring参数绑定过程

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，接收页面提交的数据是通过方法形参来接收。而不是在controller类定义成员变更接收！！！！

客户端请求

key/value

处理器适配器调用springmvc提供参数绑定组件将key/value数据转成controller方法的形参

参数绑定组件：在spirngmvc早期版本使用PropertyEditor（只能将字符串传成java对象）

**后期使用converter**（进行任意类型的传换）

spirngmvc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**

**2017-08-09 11:21:55 2017/08/09**

controller方法（形参）

## 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绑定。

#### HttpServletRequest

通过request对象获取请求信息

#### HttpServletResponse

通过response处理响应信息

#### HttpSession

通过session对象得到session中存放的对象

#### Model/ModelMap

model是一个接口，modelMap是一个接口实现 。

作用：将model数据填充到request域。

## 简单类型

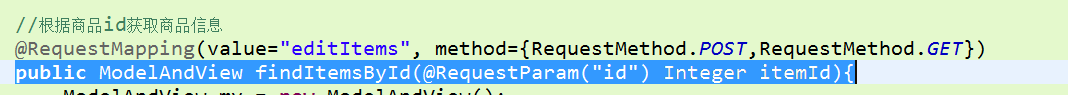
通过@RequestParam对简单类型的参数进行绑定。

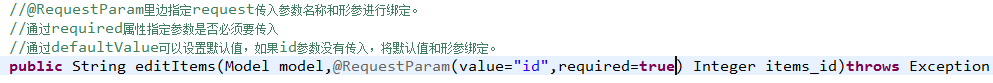
如果不使用@RequestParam，要求request传入参数名称和controller方法的形参名称一致，方可绑定成功。

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

通过required属性指定参数是否必须要传入，如果设置为true，没有传入参数，报下边错误：





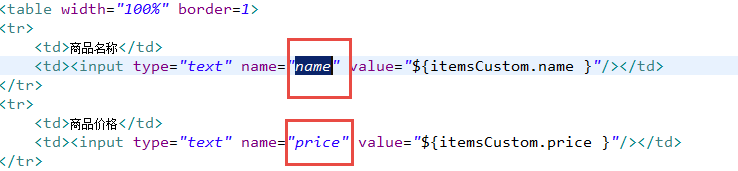


参考教案 对其它简单类型绑定进行测试。

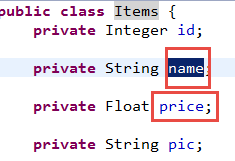
## pojo绑定

页面中input的name和controller的pojo形参中的属性名称一致，将页面中数据绑定到pojo。

页面定义：



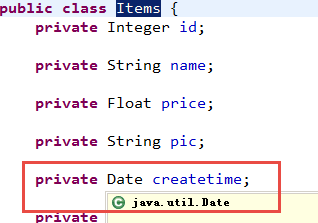
controller的pojo形参的定义：



## 自定义参数绑定实现日期类型绑定

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

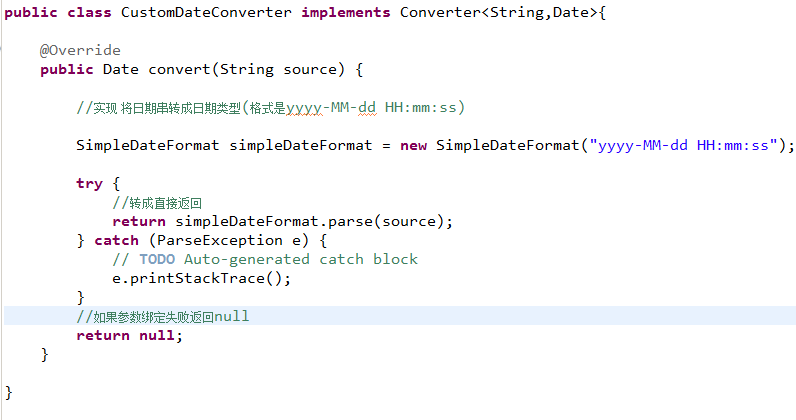
将请求日期数据串传成 日期类型，要转换的日期类型和pojo中日期属性的类型保持一致。



所以自定义参数绑定将日期串转成java.util.Date类型。

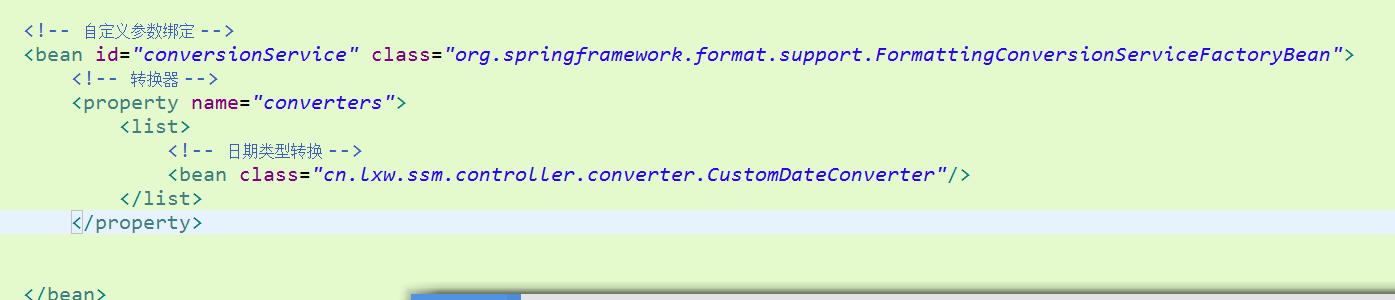
需要向处理器适配器中注入自定义的参数绑定组件。

### 自定义日期类型绑定



### 配置方式





# springmvc和struts2的区别

1、springmvc基于方法开发的，struts2基于类开发的。

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，形参数据销毁。

springmvc的controller开发类似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收参数，无法使用单例，只能使用多例。

3、经过实际测试，struts2速度慢，在于使用struts标签，如果使用struts建议使用jstl。

# 问题

## post乱码

在web.xml添加post乱码filter

在web.xml中加入：

<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>

以上可以解决post请求乱码问题。

对于get请求中文参数出现乱码解决方法有两个：

修改tomcat配置文件添加编码与工程编码一致，如下：

<Connector URIEncoding="utf-8" connectionTimeout="20000" port="8080" protocol="HTTP/1.1" redirectPort="8443"/>

另外一种方法对参数进行重新编码：

String userName new

String(request.getParamter("userName").getBytes("ISO8859-1"),"utf-8")

ISO8859-1是tomcat默认编码，需要将tomcat编码后的内容按utf-8编码