**SpringMVC基础**

# SpringMVC架构

## Spring web mvc介绍

Spring web mvc和Struts2都属于表现层的框架,它是Spring框架的一部分,我们可以从Spring的整体结构中看得出来：



## Spring web mvc 架构

### 架构图



### 架构流程

1. 用户发送请求至前端控制器DispatcherServlet
2. DispatcherServlet收到请求调用HandlerMapping处理器映射器。
3. 处理器映射器根据请求url找到具体的处理器，生成处理器对象及处理器拦截器(如果有则生成)一并返回给DispatcherServlet。
4. DispatcherServlet通过HandlerAdapter处理器适配器调用处理器
5. 执行处理器(Controller，也叫后端控制器)。
6. Controller执行完成返回ModelAndView
7. HandlerAdapter将controller执行结果ModelAndView返回给DispatcherServlet
8. DispatcherServlet将ModelAndView传给ViewReslover视图解析器
9. ViewReslover解析后返回具体View
10. DispatcherServlet对View进行渲染视图（即将模型数据填充至视图中）。
11. DispatcherServlet响应用户

### 组件说明

以下组件通常使用框架提供实现：

* DispatcherServlet：前端控制器

用户请求到达前端控制器，它就相当于mvc模式中的c，dispatcherServlet是整个流程控制的中心，由它调用其它组件处理用户的请求，dispatcherServlet的存在降低了组件之间的耦合性。

* HandlerMapping：处理器映射器

HandlerMapping负责根据用户请求找到Handler即处理器，springmvc提供了不同的映射器实现不同的映射方式，例如：配置文件方式，实现接口方式，注解方式等。

* Handler：处理器

Handler 是继DispatcherServlet前端控制器的后端控制器，在DispatcherServlet的控制下Handler对具体的用户请求进行处理。

由于Handler涉及到具体的用户业务请求，所以一般情况需要程序员根据业务需求开发Handler。

* HandlAdapter：处理器适配器

通过HandlerAdapter对处理器进行执行，这是适配器模式的应用，通过扩展适配器可以对更多类型的处理器进行执行。

* View Resolver：视图解析器

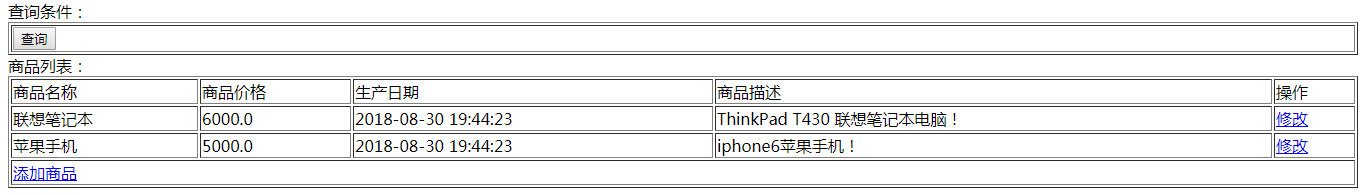
View Resolver负责将处理结果生成View视图，View Resolver首先根据逻辑视图名解析成物理视图名即具体的页面地址，再生成View视图对象，最后对View进行渲染将处理结果通过页面展示给用户。 springmvc框架提供了很多的View视图类型，包括：jstlView、freemarkerView、pdfView等。

一般情况下需要通过页面标签或页面模版技术将模型数据通过页面展示给用户，需要由程序员根据业务需求开发具体的页面。

# SpringMVC入门

## 需求

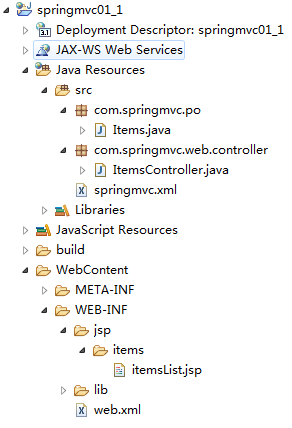
实现商品模拟数据查询列表功能。



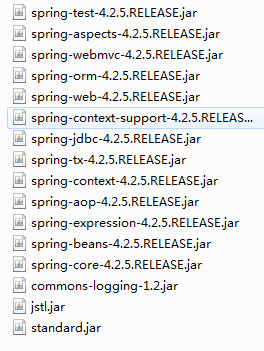
## 创建Web项目

在eclipse下创建动态web工程springmvc01\_1。

## 目录结构



## 导入spring4.2.5的jar包



## 创建Items实体类及jsp页面

包名：com.springmvc.po

**实体类**

public class Items {

private Integer id;

private String name;

private Float price;

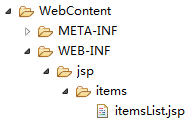
private String pic;

private Date createtime;

private String detail;

}

**jsp页面**



## springmvc配置文件

在src下创建springmvc.xml文件

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

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

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

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

*http://www.springframework.org/schema/beans/spring-beans-4.2.xsd*

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

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

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

*http://www.springframework.org/schema/aop/spring-aop-4.2.xsd*

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

*http://www.springframework.org/schema/mvc/spring-mvc-4.0.xsd"*>

</beans>

## 前端控制器配置

在WEB-INF\web.xml中配置前端控制器，

<servlet>

<servlet-name>springmvc</servlet-name>

<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>

<init-param>

<param-name>contextConfigLocation</param-name>

<param-value>classpath:springmvc.xml</param-value>

</init-param>

<load-on-startup>1</load-on-startup>

</servlet>

<servlet-mapping>

<servlet-name>springmvc</servlet-name>

<url-pattern>\*.action</url-pattern>

</servlet-mapping>

## 配置处理器适配器

在springmvc.xml文件配置如下：

<!-- 配置处理器适配器 -->

<bean

class=*"org.springframework.web.servlet.mvc.SimpleControllerHandlerAdapter"*/>

SimpleControllerHandlerAdapter：即简单控制器处理适配器，所有实现了org.springframework.web.servlet.mvc.Controller 接口的Bean作为

Springmvc的后端控制器。

## 配置处理器映射器

在springmvc.xml文件配置如下：

<!-- 处理器映射器 -->

<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。

## 创建处理器开发类

包名：com.springmvc.web.controller

**public** **class** ItemsController **implements** Controller{

@Override

**public** ModelAndView handleRequest(HttpServletRequest request, HttpServletResponse response) **throws** Exception {

//商品列表

List<Items> itemsList = **new** ArrayList<Items>();

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

items\_1.setName("联想笔记本");

items\_1.setPrice(6000f);

items\_1.setCreatetime(**new** Date());

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.setCreatetime(**new** Date());

items\_2.setDetail("iphone6苹果手机！");

itemsList.add(items\_1);

itemsList.add(items\_2);

//创建modelAndView准备填充数据、设置视图

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

//填充数据

modelAndView.addObject("itemsList", itemsList);

//视图

modelAndView.setViewName("items/itemsList");

**return** modelAndView;

}

}

org.springframework.web.servlet.mvc.Controller：处理器必须实现Controller 接口。

ModelAndView：包含了模型数据及逻辑视图名

## 处理器配置

在springmvc.xml文件配置如下：

<!-- 配置处理器类 -->

<bean name=*"/itemsList.action"* id=*"itemList"*

class=*"com.springmvc.web.controller.ItemsController"*/>

如果请求的URL 为“itemsList.action”将会成功映射到ItemsController控制器。

## 配置视图解析器

在springmvc.xml文件配置如下：

<!-- 配置视图解析器 -->

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

<property name=*"viewClass"* value=*"org.springframework.web.servlet.view.JstlView"*/>

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

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

</bean>

prefix 和suffix：查找视图页面的前缀和后缀

最终视图的址为：前缀+**逻辑视图名**+后缀

比如逻辑视图名为hello，则最终返回的jsp视图地址 “WEB-INF/jsp/hello.jsp”

## 部署在tomcat测试

http://localhost:8080/springmvc\_01/itemsList.action

# 基于注解开发【重点】

## 需求

拷贝上述案例，重命名为springmvc01\_2，使用注解完成

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

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

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

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

*http://www.springframework.org/schema/beans/spring-beans-4.2.xsd*

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

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

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

*http://www.springframework.org/schema/aop/spring-aop-4.2.xsd*

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

*http://www.springframework.org/schema/mvc/spring-mvc-4.0.xsd"*>

<!-- 扫描controller注解,多个包中间使用半角逗号分隔 -->

<context:component-scan base-package=*"com.springmvc.web.controller"*/>

<!-- 注解自动加载器 -->

<mvc:annotation-driven/>

<!-- ViewResolver -->

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

<property name=*"viewClass"* value=*"org.springframework.web.servlet.view.JstlView"*/>

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

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

</bean>

</beans>

## Controller的代码

@Controller

@RequestMapping("/items")

**public** **class** ItemsController{

@RequestMapping(value = "/itemsList")

**public** String itemsList(Model model) **throws** Exception {

//商品列表

List<Items> itemsList = **new** ArrayList<Items>();

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

items\_1.setName("联想笔记本");

items\_1.setPrice(6000f);

items\_1.setCreatetime(**new** Date());

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.setCreatetime(**new** Date());

items\_2.setDetail("iphone6苹果手机！");

itemsList.add(items\_1);

itemsList.add(items\_2);

//填充数据到request

model.addAttribute("itemsList", itemsList);

**return** "items/itemsList";

}

}

## 部署在tomcat测试

<http://localhost:8080/springmvc01_2/items/itemsList.action>

## controller方法返回值

### 返回ModelAndView

controller方法中定义ModelAndView对象并返回，对象中可添加model数据、指定view。

### 返回void

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);

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");

### 返回字符串

#### 逻辑视图名

controller方法返回字符串可以指定逻辑视图名，通过视图解析器解析为物理视图地址。

//指定逻辑视图名，经过视图解析器解析为jsp物理路径：/WEB-INF/jsp/item/editItem.jsp

**return** "item/editItem";

#### Redirect重定向

Contrller方法返回结果重定向到一个url地址，如下商品修改提交后重定向到商品查询方法，参数无法带到商品查询方法中。

//重定向到queryItem.action地址,request无法带过去

**return** "redirect:queryItem.action";

redirect方式相当于“response.sendRedirect()”，转发后浏览器的地址栏变为转发后的地址，因为转发即执行了一个新的request和response。

由于新发起一个request原来的参数在转发时就不能传递到下一个url，如果要传参数可以/item/queryItem.action后边加参数，如下：

/item/queryItem?...&…..

#### forward转发

controller方法执行后继续执行另一个controller方法，如下商品修改提交后转向到商品修改页面，修改商品的id参数可以带到商品修改方法中。

//结果转发到editItem.action，request可以带过去

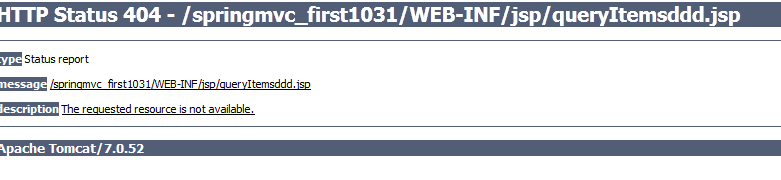
**return** "forward:editItem.action";

forward方式相当于“request.getRequestDispatcher().forward(request,response)”，转发后浏览器地址栏还是原来的地址。转发并没有执行新的request和response，而是和转发前的请求共用一个request和response。所以转发前请求的参数在转发后仍然可以读取到。

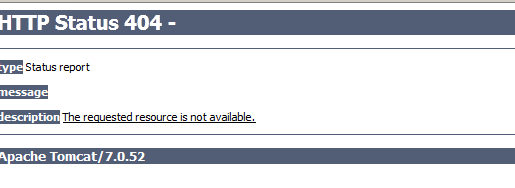
# 问题总结

## 404

页面找不到，视图找不到。



HandlerMapping根据url没有找到Handler。



## 中文乱码处理

### Post乱码处理

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。

### get乱码处理

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

还有一种方法，编写处理post和get过滤器类进行统一处理