# SpringMVC第一天课堂笔记

**学习过的mvc框架：Struts2**

springMVC就是类似于Struts2的mvc框架，属于SpringFrameWork的后续产品。

# 为什么学SpringMVC?

|  |  |  |  |
| --- | --- | --- | --- |
| **SpringMVC与Struts2区别** | | | |
| **对比项目** | **SrpingMVC** | **Struts2** | **优势** |
| 国内市场情况 | 有大量用户，一般新项目启动都会选用springmvc | 有部分老用户，老项目组，由于习惯了，一直在使用。 | 国内情况，springmvc的使用率已经超过Struts2 |
| 框架入口 | 基于servlet | 基于filter | 本质上没太大优势之分，只是配置方式不一样 |
| 框架设计思想 | 控制器基于方法级别的拦截，处理器设计为单实例 | 控制器基于类级别的拦截， 处理器设计为多实例 | 由于设计本身原因，造成了Struts2，通常来讲只能设计为多实例模式，相比于springmvc设计为单实例模式，Struts2会消耗更多的服务器内存。 |
| 参数传递 | 参数通过方法入参传递 | 参数通过类的成员变量传递 | Struts2通过成员变量传递参数，导致了参数线程不安全，有可能引发并发的问题。 |
| 与spring整合 | 与spring同一家公司，可以与spring无缝整合 | 需要整合包 | Springmvc可以更轻松与spring整合 |

# SpringMVC入门

## 开发环境

**Jdk：**jdk1.7

**Eclipse：**mars

**Tomcat：**apache-tomcat-7

**spring：**4.2.4

## 开发步骤

### 创建Dynamic web项目

课堂上起的项目名字：01-springmvc

### 导入springmvc的jar包

位置：\01课前资料\jar包\springmvc独立运行

### 编写TestController类

@Controller

**public** **class** HelloController {

@RequestMapping("hello")

**public** ModelAndView hello(){

System.***out***.println("hello springmvc....");

//创建ModelAndView对象

ModelAndView mav = **new** ModelAndView();

//设置模型数据

mav.addObject("msg", "hello springmvc...");

//设置视图名字

mav.setViewName("/WEB-INF/jsp/hello.jsp");

**return** mav;

}

}

### 创建hello.jsp页面



### 创建与配置springmvc.xml核心配置文件

<?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: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@Controller处理器，包扫描器 -->

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

</beans>

### 在web.xml中配置前端控制器

<!-- 配置前端控制器 -->

<servlet>

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

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

<!-- 加载springmvc核心配置文件 -->

<init-param>

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

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

</init-param>

</servlet>

<!-- 配置拦截路径 -->

<servlet-mapping>

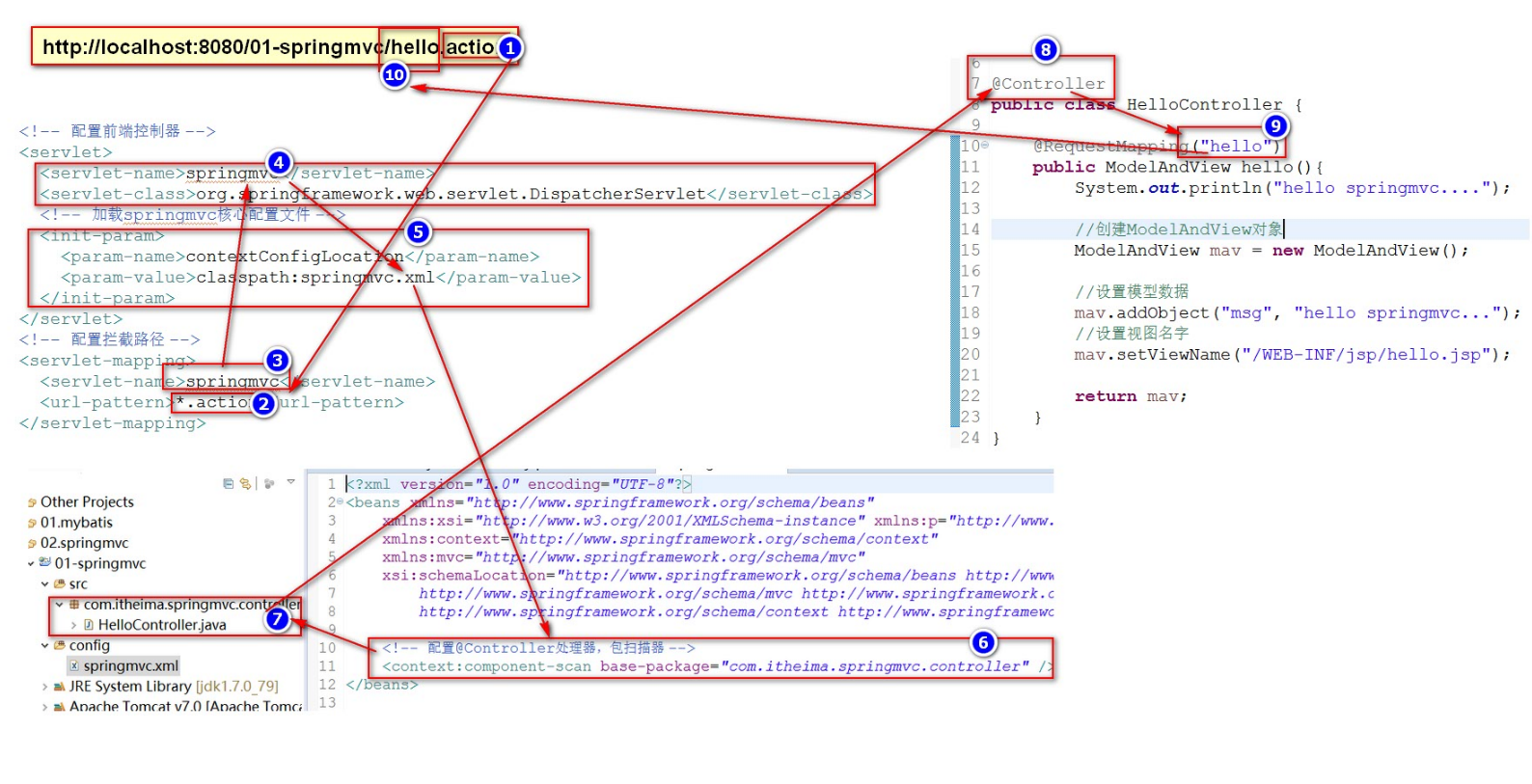
<servlet-name>springmvc</servlet-name>

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

</servlet-mapping>

### 启动项目通过浏览器测试

## 小结-springmvc代码执行流程



# 完成商品列表加载

复制第一个项目01-springmvc，粘贴在本项目目录下改名:02-springmvc，删掉web.xml核心控制器的配置，重新配置一次（多练习一次）。

## 复制itemList.jsp到新工程中

目录：\01课前资料\案例\jsp

## 创建Item的pojo

**public** **class** Item {

// 商品id

**private** **int** id;

// 商品名称

**private** String name;

// 商品价格

**private** **double** price;

// 商品创建时间

**private** Date createtime;

// 商品描述

**private** String detail;

创建带参数的构造器

set/get。。。

}

## 编写ItemController

@Controller

**public** **class** ItemController {

@RequestMapping("itemList")

**public** ModelAndView itemList(){

ModelAndView mav = **new** ModelAndView();

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

itemList.add(**new** Items(1, "冰箱", 1999.0, **new** Date(), "质量不错"));

itemList.add(**new** Items(2, "冰箱2", 1999.0, **new** Date(), "质量不错2"));

itemList.add(**new** Items(3, "冰箱3", 1999.0, **new** Date(), "质量不错3"));

itemList.add(**new** Items(4, "冰箱4", 1999.0, **new** Date(), "质量不错4"));

itemList.add(**new** Items(5, "冰箱5", 1999.0, **new** Date(), "质量不错5"));

//设置商品数据

mav.addObject("itemList", itemList);

//设置视图名字

mav.setViewName("/WEB-INF/jsp/itemList.jsp");

**return** mav;

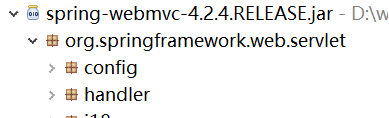
}

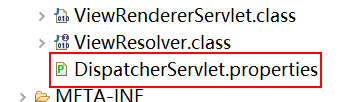
}

## 启动项目通过浏览器测试

# SpringMVC架构

## 框架默认加载组件





### 处理器映射器与处理器适配器

#### 处理器映射器

从spring3.1版本开始，废除了DefaultAnnotationHandlerMapping的使用，推荐使用RequestMappingHandlerMapping完成注解式处理器映射。

<!-- 配置处理器映射器 -->

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

#### 处理器适配器

从spring3.1版本开始，废除了AnnotationMethodHandlerAdapter的使用，推荐使用RequestMappingHandlerAdapter完成注解式处理器适配。

<!-- 处理器适配器 -->

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

#### 小结

映射器与适配器必需配套使用，如果映射器使用了推荐的RequestMappingHandlerMapping，适配器也必需使用推荐的RequestMappingHandlerAdapter。

#### 注解驱动

<!-- 注解驱动配置，代替映射器与适配器的单独配置，同时支持json响应(推荐使用) -->

<mvc:annotation-driven />

### 视图解析器

<!-- 配置视图解析器 -->

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

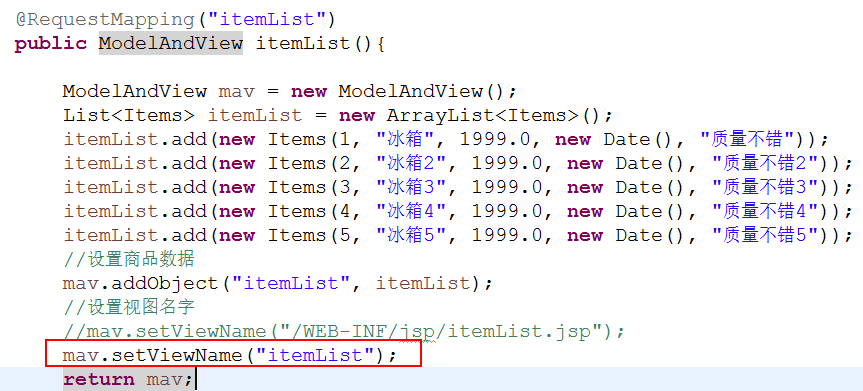
<!-- 配置视图响应的前缀 -->

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

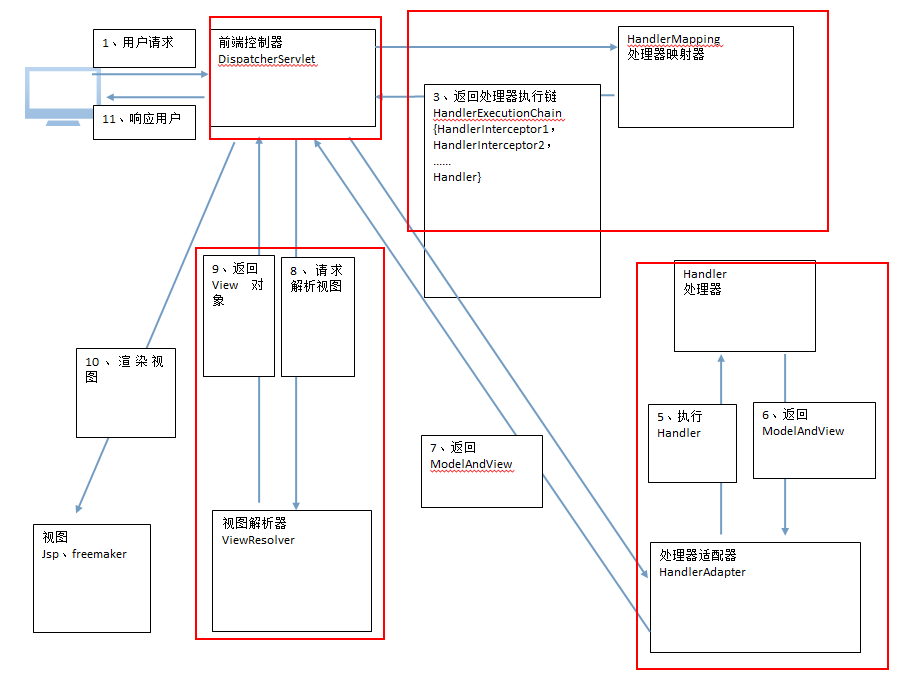
<!-- 配置视图响应的后缀 -->

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

</bean>



## 总结-springMVC架构



# SpringMVC与Mybatis整合

创建Web新工程：**03-springmvc-mybatis**

1. 参考思路与我提供的代码工程，自己动手搭建新工程。
2. 整合功能主要是把商品列表加载，从数据库中查出。
3. Dao开发可以用Mybatis逆向工程。

## 思路

**Dao层：**

1. SqlMapConfig.xml，空文件即可。需要文件头。
2. applicationContext-dao.xml。
   1. 数据库连接池
   2. SqlSessionFactory对象，需要spring和mybatis整合包下的。
   3. 配置mapper文件扫描器。

**Service层：**

1. applicationContext-service.xml包扫描器，扫描@service注解的类。
2. applicationContext-trans.xml配置事务。

**Controller层：**

Springmvc.xml

1. 包扫描器，扫描@Controller注解的类。
2. 配置注解驱动。
3. 视图解析器

**Web.xml**

1. 配置spring容量监听器
2. 配置前端控制器

# 参数绑定

## 默认支持的参数类型

基于完成需求：点击修改商品时，打开商品编辑页面，展示商品信息

/\*\*

\* 演示springmvc默认参数的传递

\* 跳转修改商品信息页面

\* **@return**

\*/

@RequestMapping("itemEdit")

**public** ModelAndView itemEdit(HttpServletRequest request,HttpServletResponse response,HttpSession session){

ModelAndView mav = **new** ModelAndView();

//request获取参数

String id = request.getParameter("id");

System.***out***.println("id为：" + id);

//其它对象输出

System.***out***.println("response对象:" + response);

System.***out***.println("session对象:" + session);

//查询商品信息

Item item = itemServices.getItemById(**new** Integer(id));

//设置商品数据返回页面

mav.addObject("item", item);

//设置视图名称

mav.setViewName("itemEdit");

**return** mav;

}

## 绑定简单参数

/\*\*

\* 演示简单参数传递

\* 跳转修改商品信息页面

\* **@RequestParam用法**：入参名字与方法名参数名不一致时使用{

\* value:传入的参数名，required：是否必填,defaultValue:默认值

\* }

\*

\*/

@RequestMapping("itemEdit")

**public** ModelAndView itemEdit(@RequestParam(value="id",required=**true**,defaultValue="1")Integer ids){

ModelAndView mav = **new** ModelAndView();

//查询商品信息

Item item = itemServices.getItemById(ids);

//设置商品数据返回页面

mav.addObject("item", item);

//设置视图名称

mav.setViewName("itemEdit");

**return** mav;

}

## Model/ModelMap

/\*\*

\* 演示返回String，通过Model/ModelMap返回数据模型

\* 跳转修改商品信息页面

\* **@param** id

\* **@return**

\*/

@RequestMapping("itemEdit")

**public** String itemEdit(@RequestParam("id")Integer ids,Model m,ModelMap model){

//查询商品信息

Item item = itemServices.getItemById(ids);

//通过Model把商品数据返回页面

model.addAttribute("item", item);

//返回String视图名字

**return** "itemEdit";

}

## 绑定pojo对象

**要点：表单提交的name属性必需与pojo的属性名称一致。**

/\*\*

\* 演示传递pojo参数

\* 更新商品信息

\* **@return**

\*/

@RequestMapping("updateItem")

**public** String updateItem(Item item,Model model){

//更新商品

itemServices.update(item);

//返回商品模型

model.addAttribute("item", item);

//返回担任提示

model.addAttribute("msg", "修改商品成功");

//返回修改商品页面

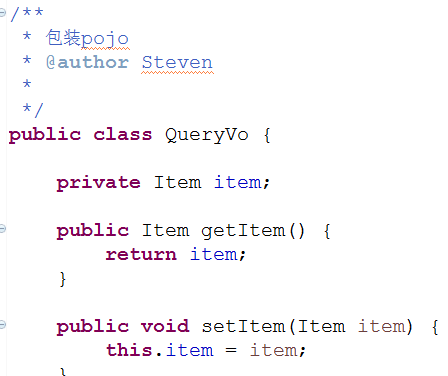
**return** "itemEdit";

}

## 绑定包装的pojo

**要点：通过点(.)传递属性。**

### pojo:



### controller代码：

/\*\*

\* 演示包装pojo传递

\* **@param** vo

\* **@return**

\*/

@RequestMapping("queryItem")

**public** String queryItem(QueryVo vo){

//打印传入参数

System.***out***.println(vo);

//返回视图

**return** "itemList";

}

### jsp修改：

