



黑马程序员

www.itheima.com

| 传智播客旗下
高端IT教育品牌

改变中国IT教育，我们正在行动

Spring web mvc

框架课程



1 课程计划

第一天

- 1、Springmvc 介绍
- 2、入门程序
- 3、Springmvc 架构讲解
 - a) 框架结构
 - b) 组件说明
- 4、Springmvc 整合 mybatis
- 5、参数绑定
 - a) Springmvc 默认支持的类型
 - b) 简单数据类型
 - c) Pojo 类型
 - d) Pojo 包装类型
 - e) 自定义参数绑定

6、Springmvc 和 struts2 的区别

第二天

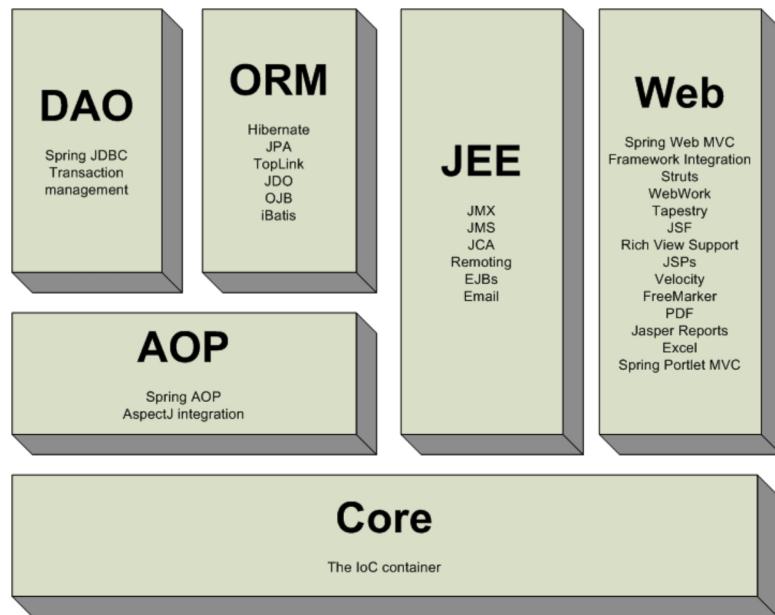
- 1、高级参数绑定
 - a) 数组类型的参数绑定
 - b) List 类型的绑定
- 2、@RequestMapping 注解的使用
- 3、Controller 方法返回值
- 4、Springmvc 中异常处理
- 5、图片上传处理
- 6、Json 数据交互
- 7、Springmvc 实现 Restful
- 8、拦截器



2 Spring web mvc 介绍

2.1 Springmvc 是什么？

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:



2.2 SpringMVC 处理流程

3 入门程序

3.1 开发环境

本教程使用环境:

Jdk: jdk1.7.0_72



Eclipse: mars

Tomcat: apache-tomcat-7.0.53

Springmvc: 4.1.3

3.2 需求

使用 springmvc 实现商品列表的展示。

3.3 需求分析

请求的 url: /itemList.action

参数: 无

数据: 静态数据

3.4 开发步骤

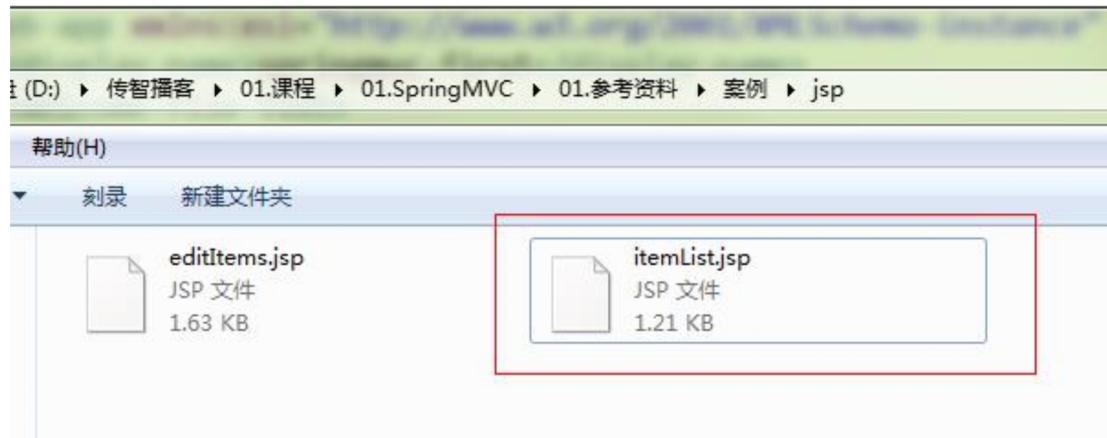
3.4.1 第一步：创建一个 javaweb 工程

3.4.2 第二步：导入 jar 包

(D:) 传智播客 01.课程 01.SpringMVC 01.参考资料 jar包 springmvc独立运行				
帮助(H)				
刻录 新建文件夹				
名称	修改日期	类型	大小	
commons-logging-1.1.1.jar	2014/6/27 18:51	Executable Jar File	60 KB	
jstl-1.2.jar	2014/6/27 18:51	Executable Jar File	405 KB	
spring-aop-4.1.3.RELEASE.jar	2015/7/19 14:05	Executable Jar File	351 KB	
spring-aspects-4.1.3.RELEASE.jar	2015/7/19 14:08	Executable Jar File	56 KB	
spring-beans-4.1.3.RELEASE.jar	2015/7/19 14:06	Executable Jar File	692 KB	
spring-context-4.1.3.RELEASE.jar	2016/1/1 16:18	Executable Jar File	1,003 KB	
spring-context-support-4.1.3.RELEASE.jar	2015/10/29 18:40	Executable Jar File	174 KB	
spring-core-4.1.3.RELEASE.jar	2015/7/19 14:06	Executable Jar File	984 KB	
spring-expression-4.1.3.RELEASE.jar	2015/7/19 14:06	Executable Jar File	254 KB	
spring-jdbc-4.1.3.RELEASE.jar	2015/7/19 14:08	Executable Jar File	417 KB	
spring-jms-4.1.3.RELEASE.jar	2015/8/20 11:40	Executable Jar File	263 KB	
spring-messaging-4.1.3.RELEASE.jar	2015/8/20 11:40	Executable Jar File	282 KB	
spring-tx-4.1.3.RELEASE.jar	2015/7/19 14:08	Executable Jar File	247 KB	
spring-web-4.1.3.RELEASE.jar	2015/7/19 14:08	Executable Jar File	697 KB	
spring-webmvc-4.1.3.RELEASE.jar	2015/7/19 14:07	Executable Jar File	764 KB	



3.4.3 第三步：创建 itemList.jsp



把参考资料中的 itemList.jsp 复制到工程的/WEB-INF/jsp 目录下。

3.4.4 第四步：创建 ItemsController

ItemController 是一个普通的 java 类，不需要实现任何接口，只需要在类上添加@Controller 注解即可。@RequestMapping 注解指定请求的 url，其中 “.action” 可以加也可以不加。在 ModelAndView 对象中，将视图设置为 “/WEB-INF/jsp/itemList.jsp”

```
@Controller
public class ItemController {

    @RequestMapping("/itemList")
    public ModelAndView itemList() throws Exception {

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

        //商品列表
        Items items_1 = new Items();
        items_1.setName("联想笔记本_3");
        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！");

        itemList.add(items_1);
        itemList.add(items_2);
        //创建 ModelAndView 对象
        ModelAndView modelAndView = new ModelAndView();
        //添加 model
        modelAndView.addObject("itemList", itemList);
```



```
//添加视图
modelAndView.setViewName("/WEB-INF/jsp/itemList.jsp");
//modelAndView.setViewName("itemsList");
return modelAndView;
}

}
```

商品数据使用 `Items` 类描述，可以使用参考资料中提供的 `pojo` 类，

3.4.5 第五步：创建 `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:dubbo="http://code.alibabatech.com/schema/dubbo" 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://code.alibabatech.com/schema/dubbo http://code.alibabatech.com/schema/dubbo/dubbo.xsd
http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context-4.0.xsd">

    <context:component-scan base-package="cn.itcast.springmvc.controller"/>

</beans>
```

3.4.6 第六步：配置前端控制器

在 `web.xml` 中添加 `DispatcherServlet` 的配置。

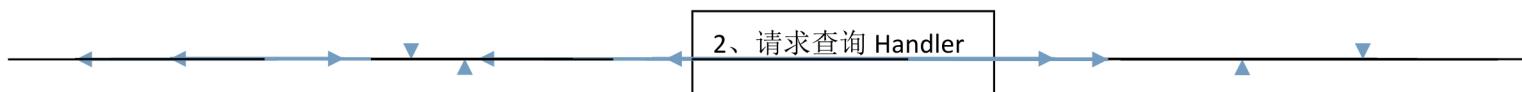
```
<!-- 前端控制器 -->
<servlet>
    <servlet-name>springmvc</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
    <init-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>classpath:springmvc.xml</param-value>
    </init-param>
</servlet>
<servlet-mapping>
```



```
<servlet-name>springmvc</servlet-name>
<url-pattern>*.action</url-pattern>
</servlet-mapping>
```

4 Springmvc 架构

4.1 框架结构



4.2 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ViewResolver视图解析器
- 9、ViewResolver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

4.3 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。

◆ HandlerAdapter：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。

◆ View：视图

springmvc框架提供了很多的View视图类型的支持，包括：jstlView、freemarkerView、pdfView等。我们最常用的视图就是jsp。

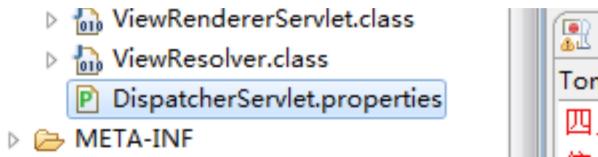
一般情况下需要通过页面标签或页面模版技术将模型数据通过页面展示给用户，需要由程序员根据业务需求开发具体的页面。



说明：在 springmvc 的各个组件中，处理器映射器、处理器适配器、视图解析器称为 springmvc 的三大组件。

需要用户开放的组件有 `handler`、`view`

4.4 框架默认加载组件



4.5 注解映射器和适配器

4.5.1 组件扫描器

使用组件扫描器省去在 spring 容器配置每个 controller 类的繁琐。使用 `<context:component-scan>` 自动扫描标记 @controller 的控制器类，配置如下：

```
<!-- 扫描controller注解,多个包中间使用半角逗号分隔 -->
<context:component-scan base-package="cn.itcast.springmvc.controller.
first"/>
```

4.5.2 RequestMappingHandlerMapping

注解式处理器映射器，对类中标记 `@ResquestMapping` 的方法进行映射，根据 `ResquestMapping` 定义的 url 匹配 `ResquestMapping` 标记的方法，匹配成功返回 `HandlerMethod` 对象给前端控制器，`HandlerMethod` 对象中封装 url 对应的方法 `Method`。

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。

配置如下：

```
<!--注解映射器 -->
<bean class="org.springframework.web.servlet.mvc.method.annotation.Re
questMappingHandlerMapping"/>
```

注解描述：

@RequestMapping: 定义请求 url 到处理器功能方法的映射

4.5.3 RequestMappingHandlerAdapter

注解式处理器适配器，对标记@ResquestMapping 的方法进行适配。

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。

配置如下：

```
<!--注解适配器 -->
<bean class="org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerAdapter"/>
```

4.5.4 <mvc:annotation-driven>

springmvc 使用 <mvc:annotation-driven> 自动加载 RequestMappingHandlerMapping 和 RequestMappingHandlerAdapter ， 可用在 springmvc.xml 配置文件中使用 <mvc:annotation-driven> 替代注解处理器和适配器的配置。

4.6 视图解析器

在 springmvc.xml 文件配置如下：

```
<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>
```

InternalResourceViewResolver：支持JSP视图解析

viewClass: JstlView表示JSP模板页面需要使用JSTL标签库，所以classpath中必须包含 jstl 的相关 jar 包。此属性可以不设置，默认为JstlView。

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，逻辑视图名需要在 controller 中返回 ModelAndView 指定，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”

5 整合 mybatis

为了更好的学习 springmvc 和 mybatis 整合开发的方法，需要将 springmvc 和 mybatis 进行整合。

整合目标：控制层采用 springmvc、持久层使用 mybatis 实现。



5.1 需求

实现商品查询列表，从 mysql 数据库查询商品信息。

5.2 jar 包

包括：spring（包括 springmvc）、mybatis、mybatis-spring 整合包、数据库驱动、第三方连接池。

参考：“mybatis 与 springmvc 整合全部 jar 包”目录

5.3 工程搭建

5.3.1 整合思路

Dao 层：

- 1、SqlMapConfig.xml，空文件即可。需要文件头。
- 2、applicationContext-dao.xml。
 - a) 数据库连接池
 - b) SqlSessionFactory 对象，需要 spring 和 mybatis 整合包下的。
 - c) 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。

表现层：

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

配置前端控制器。

5.3.2 sqlMapConfig.xml

在 classpath 下创建 *mybatis/sqlMapConfig.xml*

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE configuration
PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
"http://mybatis.org/dtd/mybatis-3-config.dtd">
<configuration>
</configuration>
```

5.3.3 applicationContext-dao.xml

配置数据源、配置 SqlSessionFactory、mapper 扫描器。

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:context="http://www.springframework.org/schema/context" xmlns:p="http://www.springframework.org/schema/p">
```



```
k.org/schema/p"
    xmlns:aop="http://www.springframework.org/schema/aop" xmlns:tx="http://www.springframework.org/schema/tx"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-4.0.xsd
http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context-4.0.xsd
http://www.springframework.org/schema/aop
http://www.springframework.org/schema/aop/spring-aop-4.0.xsd http://www.springframework.org/schema/tx
http://www.springframework.org/schema/tx/spring-tx-4.0.xsd
http://www.springframework.org/schema/util
http://www.springframework.org/schema/util/spring-util-4.0.xsd">

<!-- 加载配置文件 -->
<context:property-placeholder location="classpath:db.properties"/>
<!-- 数据库连接池 -->
<bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource"
    destroy-method="close">
    <property name="driverClassName" value="${jdbc.driver}"/>
    <property name="url" value="${jdbc.url}"/>
    <property name="username" value="${jdbc.username}"/>
    <property name="password" value="${jdbc.password}"/>
    <property name="maxActive" value="10"/>
    <property name="maxIdle" value="5"/>
</bean>
<!-- mapper 配置 -->
<!-- 让 spring 管理 sqlSessionfactory 使用 mybatis 和 spring 整合包中的 -->
<bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean">
    <!-- 数据库连接池 -->
    <property name="dataSource" ref="dataSource"/>
    <!-- 加载 mybatis 的全局配置文件 -->
    <property name="configLocation" value="classpath:mybatis/SqlMapConfig.xml"/>
</bean>
<!-- 配置 Mapper 扫描器 -->
<bean class="org.mybatis.spring.mapper.MapperScannerConfigurer">
    <property name="basePackage" value="cn.itcast.springmvc.mapper"/>
</bean>

</beans>
```

Db.properties

```
jdbc.driver=com.mysql.jdbc.Driver
jdbc.url=jdbc:mysql://localhost:3306/springmvc?characterEncoding=utf-8
```



```
jdbc.username=root
jdbc.password=root
```

5.3.4 applicationContext-service.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:context="http://www.springframework.org/schema/context" xmlns:p="http://www.springframework.org/schema/p"
       xmlns:aop="http://www.springframework.org/schema/aop" xmlns:tx="http://www.springframework.org/schema/tx"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-4.0.xsd
http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context-4.0.xsd
http://www.springframework.org/schema/aop
http://www.springframework.org/schema/aop/spring-aop-4.0.xsd http://www.springframework.org/schema/tx
http://www.springframework.org/schema/tx/spring-tx-4.0.xsd
http://www.springframework.org/schema/util
http://www.springframework.org/schema/util/spring-util-4.0.xsd">

    <context:component-scan base-package="cn.itcast.springmvc.service"/>

</beans>
```

5.3.5 applicationContext-transaction.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:context="http://www.springframework.org/schema/context" xmlns:p="http://www.springframework.org/schema/p"
       xmlns:aop="http://www.springframework.org/schema/aop" xmlns:tx="http://www.springframework.org/schema/tx"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-4.0.xsd
http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context-4.0.xsd
http://www.springframework.org/schema/aop
http://www.springframework.org/schema/aop/spring-aop-4.0.xsd http://www.springframework.org/schema/tx
http://www.springframework.org/schema/tx/spring-tx-4.0.xsd
http://www.springframework.org/schema/util
http://www.springframework.org/schema/util/spring-util-4.0.xsd">

    <!-- 事务管理器 -->
    <bean id="transactionManager"
```



```
class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
    <!-- 数据源 -->
    <property name="dataSource" ref="dataSource"/>
</bean>
<!-- 通知 -->
<tx:advice id="txAdvice" transaction-manager="transactionManager">
    <tx:attributes>
        <!-- 传播行为 -->
        <tx:method name="save*" propagation="REQUIRED"/>
        <tx:method name="insert*" propagation="REQUIRED"/>
        <tx:method name="delete*" propagation="REQUIRED"/>
        <tx:method name="update*" propagation="REQUIRED"/>
        <tx:method name="find*" propagation="SUPPORTS" read-only="true"/>
        <tx:method name="get*" propagation="SUPPORTS" read-only="true"/>
    </tx:attributes>
</tx:advice>
<!-- 切面 -->
<aop:config>
    <aop:advisor advice-ref="txAdvice"
        pointcut="execution(* cn.itcast.springmvc.service.*.*(..))"/>
</aop:config>
</beans>
```

5.3.6 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:dubbo="http://code.alibabatech.com/schema/dubbo" 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://code.alibabatech.com/schema/dubbo
    http://code.alibabatech.com/schema/dubbo/dubbo.xsd
    http://www.springframework.org/schema/context
    http://www.springframework.org/schema/context/spring-context-4.0.xsd">

    <!-- 扫描带 Controller 注解的类 -->
    <context:component-scan base-package="cn.itcast.springmvc.controller"/>
    <!-- 加载注解驱动 -->
    <mvc:annotation-driven/>
    <!-- 视图解析器 -->
```



```
<bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
    <property name="viewClass"
        value="org.springframework.web.servlet.view.JstlView"/>
    <!-- jsp 前缀 -->
    <property name="prefix" value="/WEB-INF/jsp/" />
    <!-- jsp 后缀 -->
    <property name="suffix" value=".jsp" />
</bean>
</beans>
```

5.3.7 web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns="http://java.sun.com/xml/ns/javaee"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
    http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
    id="WebApp_ID" version="2.5">
    <display-name>springmvc-web</display-name>
    <welcome-file-list>
        <welcome-file>index.html</welcome-file>
        <welcome-file>index.htm</welcome-file>
        <welcome-file>index.jsp</welcome-file>
        <welcome-file>default.html</welcome-file>
        <welcome-file>default.htm</welcome-file>
        <welcome-file>default.jsp</welcome-file>
    </welcome-file-list>
    <!-- 加载 spring 容器 -->
    <context-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>classpath:spring/applicationContext-*.xml</param-value>
    </context-param>
    <listener>
        <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
    </listener>

    <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:spring/springmvc.xml</param-value>
        </init-param>
    </servlet>

```



```
<servlet-mapping>
    <servlet-name>springmvc</servlet-name>
    <url-pattern>*.action</url-pattern>
</servlet-mapping>
</web-app>
```

5.4 Dao

mybatis 逆向工程。

5.5 Service

- 1、Service 由 spring 管理
- 2、spring 对 Service 进行事务控制。

5.5.1 ItemService 接口

```
public interface ItemService {
    List<Items> getItemsList();
}
```

5.5.2 ItemServiceImpl 实现类

```
@Service
public class ItemServiceImpl implements ItemService {

    @Autowired
    private ItemMapper itemMapper;

    @Override
    public List<Items> getItemsList() {
        List<Items> itemList = itemMapper.getItemList();
        return itemList;
    }
}
```

5.6 Controller

```
@Controller
public class ItemController {

    @Autowired
    private ItemService itemService;
```



```
@RequestMapping("/itemList")
public ModelAndView getItemList() {
    List<Items>list = itemService.getItemsList();
    ModelAndView modelAndView = new ModelAndView();
    modelAndView.addObject("itemList", list);
    modelAndView.setViewName("itemList");
    return modelAndView;
}
```

5.7 测试

访问: <http://localhost:8080/springmvc-web/itemList.action>

6 参数绑定

6.1 绑定简单数据类型

6.1.1 需求

打开商品编辑页面，展示商品信息。

6.1.2 需求分析

编辑商品信息，需要根据商品 id 查询商品信息，然后展示到页面。

请求的 url: /itemEdit.action

参数: id (商品 id)

响应结果: 商品编辑页面，展示商品详细信息。

6.1.3 Service

```
@Override
public Items getItemById(int id) {
    Items items = itemMapper.getItemById(id);
    return items;
}
```

6.1.4 Controller 参数绑定

要根据 id 查询商品数据，需要从请求的参数中把请求的 id 取出来。Id 应该包含在 Request 对象中。可以从 Request 对象中取 id。



```
@RequestMapping("/itemEdit")
public ModelAndView itemEdit(HttpServletRequest request) {
    //从 Request 中取 id
    String strId = request.getParameter("id");
    Integer id = null;
    //如果 id 有值则转换成 int 类型
    if (strId != null && !"".equals(strId)) {
        id = new Integer(strId);
    } else {
        //出错
        return null;
    }
    Items items = itemService.getItemById(id);
    //创建 ModelAndView
    ModelAndView modelAndView = new ModelAndView();
    //向 jsp 传递数据
    modelAndView.addObject("item", items);
    //设置跳转的 jsp 页面
    modelAndView.setViewName("editItem");
    return modelAndView;
}
```

如果想获得 Request 对象只需要在 Controller 方法的形参中添加一个参数即可。Springmvc 框架会自动把 Request 对象传递给方法。

6.1.5 默认支持的参数类型

处理器形参中添加如下类型的参数处理适配器会默认识别并进行赋值。

6.1.5.1 HttpServletRequest

通过 request 对象获取请求信息

6.1.5.2 HttpServletResponse

通过 response 处理响应信息

6.1.5.3 HttpSession

通过 session 对象得到 session 中存放的对象

6.1.5.4 Model/ModelMap

ModelMap 是 Model 接口的实现类，通过 Model 或 ModelMap 向页面传递数据，如下：

```
//调用service查询商品信息
Items item = itemService.findItemById(id);
model.addAttribute("item", item);
```

页面通过\${item.XXXX}获取 item 对象的属性值。



使用 Model 和 ModelMap 的效果一样，如果直接使用 Model，springmvc 会实例化 ModelMap。

如果使用 Model 则可以不使用 ModelAndView 对象，Model 对象可以向页面传递数据，View 对象则可以使用 String 返回值替代。不管是 Model 还是 ModelAndView，其本质都是使用 Request 对象向 jsp 传递数据。

如果使用 Model 则方法可以改造成：

```
@RequestMapping("/itemEdit")
public String itemEdit(HttpServletRequest request, Model model) {
    //从 Request 中取 id
    String strId = request.getParameter("id");
    Integer id = null;
    //如果 id 有值则转换成 int 类型
    if (strId != null && !"".equals(strId)) {
        id = new Integer(strId);
    } else {
        //出错
        return null;
    }
    Items items = itemService.getItemId(id);
    //创建 ModelAndView
    //ModelAndView modelAndView = new ModelAndView();
    //向 jsp 传递数据
    //modelAndView.addObject("item", items);
    model.addAttribute("item", items);
    //设置跳转的 jsp 页面
    //modelAndView.setViewName("editItem");
    //return modelAndView;
    return "editItem";
}
```

6.1.6 绑定简单类型

当请求的参数名称和处理器形参名称一致时会将请求参数与形参进行绑定。从 Request 取参数的方法可以进一步简化。

```
@RequestMapping("/itemEdit")
public String itemEdit(Integer id, Model model) {
    Items items = itemService.getItemId(id);
    //向 jsp 传递数据
    model.addAttribute("item", items);
    //设置跳转的 jsp 页面
    return "editItem";
}
```



6.1.6.1 支持的数据类型

参数类型推荐使用包装数据类型，因为基础数据类型不可以为 null

整形：Integer、int

字符串：String

单精度：Float、float

双精度：Double、double

布尔型：Boolean、boolean

说明：对于布尔类型的参数，请求的参数值为 true 或 false。

处理器方法：

```
public String editItem(Model model, Integer id, Boolean status) throws Exception
```

请求 url：

<http://localhost:8080/xxx.action?id=2&status=false>

6.1.6.2 @RequestParam

使用@RequestParam 常用于处理简单类型的绑定。

value: 参数名字，即入参的请求参数名字，如value=“item_id”表示请求的参数区中的名字为item_id的参数的值将传入；

required: 是否必须，默认是true，表示请求中一定要有相应的参数，否则将报；

HTTP Status 400 – Required Integer parameter 'XXXX' is not present

defaultValue: 默认值，表示如果请求中没有同名参数时的默认值

定义如下：

```
public String editItem(@RequestParam(value="item_id",required=true) String id) {
```

```
}
```

形参名称为 id，但是这里使用 value="item_id" 限定请求的参数名为 item_id，所以页面传递参数的名必须为 item_id。

注意：如果请求参数中没有 item_id 将跑出异常：

HTTP Status 500 - Required Integer parameter 'item_id' is not present

这里通过 required=true 限定 item_id 参数为必需传递，如果不传递则报 400 错误，可以使用 defaultValue 设置默认值，即使 required=true 也可以不传 item_id 参数值

6.2 绑定 pojo 类型

6.2.1 需求

将页面修改后的商品信息保存到数据库中。



6.2.2 需求分析

请求的 url: /updateitem.action

参数: 表单中的数据。

响应内容: 更新成功页面

6.2.3 使用 pojo 接收表单数据

如果提交的参数很多, 或者提交的表单中的内容很多的时候可以使用 pojo 接收数据。要求 pojo 对象中的属性名和表单中 input 的 name 属性一致。

页面定义如下;

```
<input type="text" name="name"/>
<input type="text" name="price"/>
```

Pojo 定义:

```
2
3 import java.util.Date;
4
5 public class Items {
6     private Integer id;
7
8     private String name;
9
10    private Float price;
11
12    private String pic;
13
14    private Date createtime;
15
```

请求的参数名称和 pojo 的属性名称一致, 会自动将请求参数赋值给 pojo 的属性。

```
@RequestMapping("/updateitem")
public String updateItem(Items items) {
    itemService.updateItem(items);
    return "success";
}
```

注意: 提交的表单中不要有日期类型的数据, 否则会报 400 错误。如果想提交日期类型的数据需要用到后面的自定义参数绑定的内容。



6.2.4 解决 post 乱码问题

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

6.3 绑定包装 pojo

6.3.1 需求

使用包装的 pojo 接收商品信息的查询条件。

6.3.2 需求分析

包装对象定义如下：

```
PublicclassQueryVo {
privateItemsitems;
}
```

页面定义：

```
<input type="text" name="items.name" />
<input type="text" name="items.price" />
```

Controller 方法定义如下：

```
public String useraddsubmit(Model model, QueryVoqueryVo) throws Exception {
    System.out.println(queryVo.getItems());
```

6.3.3 接收查询条件

```
@RequestMapping("/queryitem")
public String queryItem(QueryVo queryVo) {
    System.out.println(queryVo.getItems().getName());
    System.out.println(queryVo.getItems().getPrice());
    return null;
}
```

6.4 自定义参数绑定

6.4.1 需求

在商品修改页面可以修改商品的生产日期，并且根据业务需求自定义日期格式。

6.4.2 需求分析

由于日期数据有很多种格式，所以 springmvc 没办法把字符串转换成日期类型。所以需要自定义参数绑定。前端控制器接收到请求后，找到注解形式的处理器适配器，对 `RequestMapping` 标记的方法进行适配，并对方法中的形参进行参数绑定。在 `springmvc` 这可以在处理器适配器上自定义 `Converter` 进行参数绑定。如果使用 `<mvc:annotation-driven>` 可以在此标签上进行扩展。

6.4.3 自定义 Converter

```
public class DateConverter implements Converter<String, Date> {

    @Override
    public Date convert(String source) {
        SimpleDateFormat simpleDateFormat = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
        try {
            return simpleDateFormat.parse(source);
        } catch (ParseException e) {
            e.printStackTrace();
        }
    }
}
```

```
    return null;
}
}
```

6.4.4 配置 Converter



```
<!-- 加载注解驱动 -->
<mvc:annotation-driven conversion-service="conversionService"/>
<!-- 转换器配置 -->
<bean id="conversionService"
      class="org.springframework.format.support.FormattingConversionServiceFactoryBean">
    <property name="converters">
      <set>
        <bean class="cn.itcast.springmvc.convert.DateConverter"/>
      </set>
    </property>
</bean>
```

6.4.5 配置方式 2（了解）

```
<?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:dubbo="http://code.alibabatech.com/schema/dubbo" 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/context
http://www.springframework.org/schema/dubbo
http://www.springframework.org/schema/mvc
http://www.springframework.org/schema/mvc/spring-mvc-4.0.xsd"/>
```



```
http://code.alibabatech.com/schema/dubbo http://code.alibabatech.com/schema/dubbo/dubbo.xsd
http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context-4.0.xsd">

<!-- 扫描带 Controller 注解的类 -->
<context:component-scan base-package="cn.itcast.springmvc.controller"/>

<!-- 转换器配置 -->
<bean id="conversionService"
      class="org.springframework.format.support.FormattingConversionServiceFactoryBean">
    <property name="converters">
      <set>
        <bean class="cn.itcast.springmvc.convert.DateConverter"/>
      </set>
    </property>
  </bean>
<!-- 自定义 webBinder -->
<bean id="customBinder"
      class="org.springframework.web.bind.support.ConfigurableWebBindingInitializer">
    <property name="conversionService" ref="conversionService"/>
  </bean>
<!-- 注解适配器 -->
<bean class="org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerAdapter"
>
  <property name="webBindingInitializer" ref="customBinder"/></property>
</bean>
<!-- 注解处理器映射器 -->
<bean class="org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerMapping"
/>

<!-- 加载注解驱动 -->
<!-- <mvc:annotation-driven/> -->
<!-- 视图解析器 -->
<bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
  <property name="viewClass"
            value="org.springframework.web.servlet.view.JstlView"/>
  <!-- jsp 前缀 -->
  <property name="prefix" value="/WEB-INF/jsp/" />
  <!-- jsp 后缀 -->
  <property name="suffix" value=".jsp" />
</bean>
</beans>
```

注意：此方法需要独立配置处理器映射器、适配器，不再使用<mvc:annotation-driven/>



7 springmvc 与 struts2 不同

- 1、springmvc 的入口是一个 servlet 即前端控制器，而 struts2 入口是一个 filter 过滤器。
- 2、**springmvc** 是基于方法开发(一个 url 对应一个方法)，请求参数传递到方法的形参，可以设计为单例或多例(建议单例)，**struts2** 是基于类开发，传递参数是通过类的属性，只能设计为多例。
- 3、Struts 采用值栈存储请求和响应的数据，通过 OGNL 存取数据，springmvc 通过参数解析器是将 request 请求内容解析，并给方法形参赋值，将数据和视图封装成 ModelAndView 对象，最后又将 ModelAndView 中的模型数据通过 request 域传输到页面。Jsp 视图解析器默认使用 jstl。