springmvc（表现层框架）

**第一天**

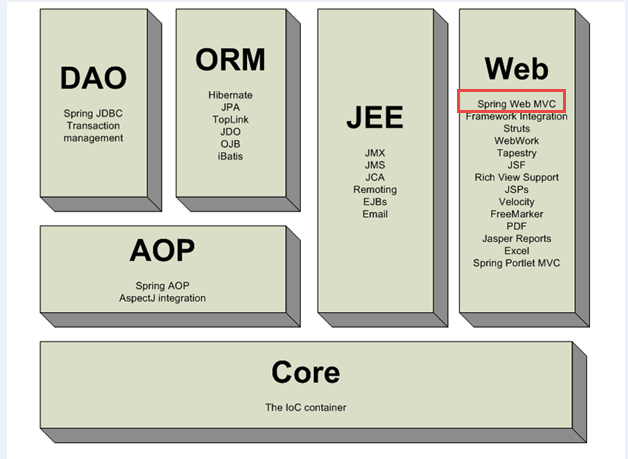
# 学习目标

* 介绍springmvc框架
* 了解springmvc环境搭建
* 掌握springmvc入门程序
* 掌握springmvc框架原理
* 掌握ssm整合（springmvc+spring+mybatis）
* 掌握springmvc参数绑定
* 了解springmvc与struts2区别

# springmvc介绍

## springmvc是什么

说明：springmvc是spring框架的一个模块，是基于mvc的表现层框架。用于web项目的开发。



## mvc是什么（b/s）

说明：mvc是表现层的一个设计模型。Model（模型）-view（视图）-Controller（控制器）。是三层架构的设计模型，主要用于实现前端页面的展现与后端业务数据处理逻辑的分离。

1.request请求

C：控制器

接收请求

响应用户

M：模型

pojo

action

service

dao

2.请求模型处理

3.返回模型数据

V：视图

jsp

4.视图渲染

5.响应用户

mvc设计模型的优点：

1. 分层的设计，实现业务系统各个组件之间的解耦
2. 有利于实现系统的可扩展性，可维护性
3. 有利于实现系统的并行开发，提升开发效率

# springmvc入门程序

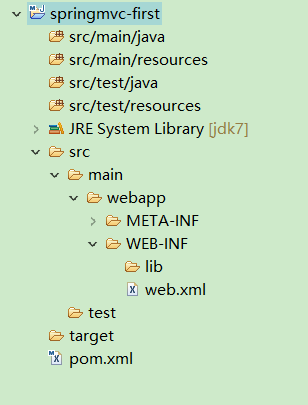
## 准备环境

jdk：1.7

ide：eclipse

web容器：tomcat

## 创建项目



## 配置pom.xml，加入依赖

* spring框架包（springmvc）

|  |
| --- |
| <project xmlns=*"http://maven.apache.org/POM/4.0.0"* xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"* xsi:schemaLocation=*"http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"*>  <modelVersion>4.0.0</modelVersion>    <groupId>cn.itheima</groupId>  <artifactId>springmvc-first</artifactId>  <version>0.0.1-SNAPSHOT</version>    <packaging>war</packaging>  <name>springmvc-first Maven Webapp</name>  <url>http://maven.apache.org</url>  <properties>  <!-- spring版本号 -->  <spring.version>4.3.8.RELEASE</spring.version>  <!-- jstl标签版本 -->  <jstl.version>1.2</jstl.version>  </properties>  <dependencies>  <!-- springmvc依赖包 -->  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-core</artifactId>  <version>${spring.version}</version>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-web</artifactId>  <version>${spring.version}</version>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-tx</artifactId>  <version>${spring.version}</version>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-jdbc</artifactId>  <version>${spring.version}</version>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-webmvc</artifactId>  <version>${spring.version}</version>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-aop</artifactId>  <version>${spring.version}</version>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-context-support</artifactId>  <version>${spring.version}</version>  </dependency>  <!-- JSTL标签类 -->  <dependency>  <groupId>jstl</groupId>  <artifactId>jstl</artifactId>  <version>${jstl.version}</version>  </dependency>  <dependency>  <groupId>junit</groupId>  <artifactId>junit</artifactId>  <version>4.12</version>  <scope>test</scope>  </dependency>  </dependencies>  <build>  <plugins>  <plugin>  <groupId>org.apache.tomcat.maven</groupId>  <artifactId>tomcat7-maven-plugin</artifactId>  <version>2.1</version>  <configuration>  <!-- tomcat 的端口号 -->  <port>8080</port>  <!-- 访问应用的路径 -->  <path>/springmvc-first</path>  <!-- URL按UTF-8进行编码，解决中文参数乱码 -->  <uriEncoding>UTF-8</uriEncoding>  <!-- tomcat名称 -->  <server>tomcat7</server>  </configuration>  </plugin>  </plugins>  </build>  </project> |

## 编写配置文件

### springmvc.xml

说明：它是springmvc框架主配置文件。文件名称可以修改。

|  |
| --- |
| <?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"*>  </beans> |

### 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"* xmlns:web=*"http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd"* 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-first</display-name>    <!-- 配置中央处理器：DispatcherServlet -->  <servlet>  <servlet-name>springmvc-first</servlet-name>  <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>    <!-- 加载springmvc主配置文件 -->  <init-param>  <param-name>contextConfigLocation</param-name>  <param-value>classpath:springmvc.xml</param-value>  </init-param>    <!-- 配置什么时候加载前端控制器，说明：  1.配置大于等于0整数，表示tomcat启动就加载前端控制器  2.配置小于0的整数，表示在第一次请求到达的时候加载  -->  <load-on-startup>1</load-on-startup>  </servlet>    <servlet-mapping>  <servlet-name>springmvc-first</servlet-name>  <!-- 配置请求的url规则，说明：  1.\*.do，表示以.do结尾的请求进入前端控制器  2./,表示所有请求都进入前端控制器  -->  <url-pattern>\*.do</url-pattern>  </servlet-mapping>    <!-- 默认的欢迎页面 -->  <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>  </web-app> |

## 编写jsp页面

|  |
| --- |
| <%@ page language=*"java"* contentType=*"text/html; charset=UTF-8"*  pageEncoding=*"UTF-8"*%>  <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">  <html>  <head>  <meta http-equiv=*"Content-Type"* content=*"text/html; charset=UTF-8"*>  <title>springmvc入门程序页面</title>  </head>  <body>  你好，世界！${hello}  </body>  </html> |

## 编写一个controller

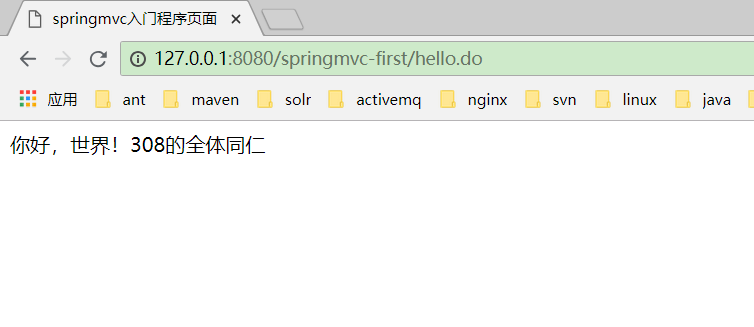
说明：它就相当于struts2中action。

|  |
| --- |
| **package** cn.itheima.controller;  **import** org.springframework.stereotype.Controller;  **import** org.springframework.web.bind.annotation.RequestMapping;  **import** org.springframework.web.servlet.ModelAndView;  /\*\*  \* **@ClassName**: HelloController  \* **@Description**: hello 控制器  \* **@author** 传智 小杨老师  \* **@date** 2018-8-14 下午3:14:25  \*  \*/  @Controller  **public** **class** HelloController {    /\*\*  \* 问好  \* ModelAndView:模型和视图。用于封装响应的数据；用于封装响应的页面  \*  \* **@RequestMapping注解**：配置请求的url  \*/  @RequestMapping("/hello.do")  **public** ModelAndView hello(){    // 1.创建ModelAndView  ModelAndView mav =**new** ModelAndView();    // 2.响应模型数据  /\*\*  \* addObject：封装模型数据  \* 参数：  \* attributeName：属性名称（模型名称）  \* attributeValue：模型数据  \*/  mav.addObject("hello", "308的全体同仁");    // 3.封装响应的视图  /\*\*  \* setViewName：设置视图的名称  \* 参数：  \* viewName：视图的名称  \*/  mav.setViewName("/WEB-INF/jsp/hello.jsp");      **return** mav;    }  } |

## 在springmvc.xml中配置包扫描controller

|  |
| --- |
| <?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"*>    <!-- 配置包扫描controller -->  <context:component-scan base-package=*"cn.itheima.controller"*/>  </beans> |

## 启动执行



## 入门程序小结

1. 导入springmvc jar包
2. 编写核心配置文件：springmvc.xml
3. 在web.xml中配置前端控制器：DispatcherServlet
4. 编写一个jsp页面
5. 编写一个controller：HelloController
6. 在springmvc.xml中配置包扫描controller
7. 启动执行

# springmvc三大组件

处理器映射器（HandlerMapping），处理器适配器（HandlerAdapter），视图解析器（ViewResolver）。

## 处理器映射器

作用：根据请求的url找到处理器方法。

### 默认的处理器映射器

DispatcherServlet.properties：

|  |
| --- |
| org.springframework.web.servlet.HandlerMapping=org.springframework.web.servlet.handler.BeanNameUrlHandlerMapping,\  org.springframework.web.servlet.mvc.annotation.DefaultAnnotationHandlerMapping |

注意事项：默认的处理器映射器已经过时，在企业项目中不推荐使用。

### 配置处理器映射器

|  |
| --- |
| <?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"*>    <!-- 配置包扫描controller -->  <context:component-scan base-package=*"cn.itheima.controller"*/>    <!-- 配置处理器映射器 -->  <bean class=*"org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerMapping"*/>  </beans> |

## 处理器适配器

作用：执行处理器方法。

### 默认的处理器适配器

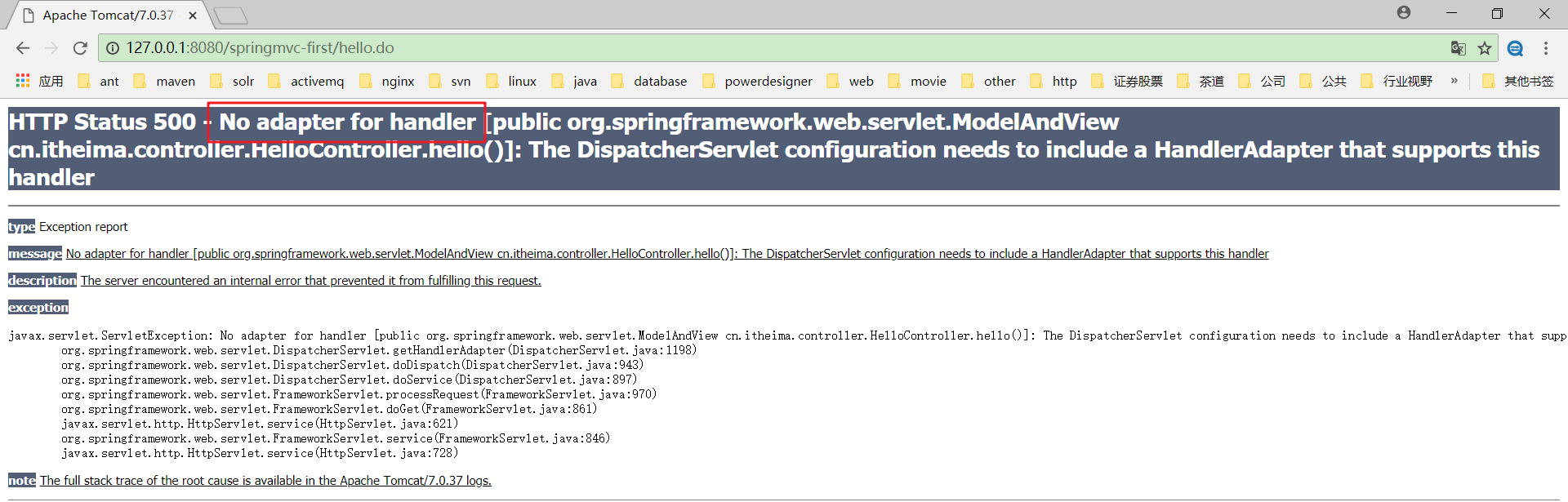
|  |
| --- |
| org.springframework.web.servlet.HandlerAdapter=org.springframework.web.servlet.mvc.HttpRequestHandlerAdapter,\  org.springframework.web.servlet.mvc.SimpleControllerHandlerAdapter,\  org.springframework.web.servlet.mvc.annotation.AnnotationMethodHandlerAdapter |

注意事项：默认的处理器适配器已经过时。在企业项目中不推荐使用。

### 配置处理器适配器

|  |
| --- |
| <?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"*>    <!-- 配置包扫描controller -->  <context:component-scan base-package=*"cn.itheima.controller"*/>    <!-- 配置处理器映射器 -->  <bean class=*"org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerMapping"*/>  <!-- 配置处理器适配器 -->  <bean class=*"org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerAdapter"*/>  </beans> |

注意事项：处理器映射器和适配器需要配对使用。



## 注解驱动方式配置处理器映射器和处理器适配器【掌握】

|  |
| --- |
| <?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"*>    <!-- 配置包扫描controller -->  <context:component-scan base-package=*"cn.itheima.controller"*/>    <!-- 配置处理器映射器  <bean class="org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerMapping"/> -->  <!-- 配置处理器适配器  <bean class="org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerAdapter"/>-->    <!-- 注解驱动方式配置处理器映射器和处理器适配器,说明：  1.等于同时配置了RequestMappingHandlerMapping/RequestMappingHandlerAdapter  2.在企业项目中推荐使用的方式  -->  <mvc:annotation-driven/>                </beans> |

## 视图解析器

作用：把逻辑视图（在处理器方法中设置的视图名称）解析成物理视图（在浏览器实际看到的页面）。

### 默认视图解析器

|  |
| --- |
| org.springframework.web.servlet.ViewResolver=org.springframework.web.servlet.view.InternalResourceViewResolver |

### 配置视图解析器

|  |
| --- |
| <?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"*>    <!-- 配置包扫描controller -->  <context:component-scan base-package=*"cn.itheima.controller"*/>    <!-- 配置处理器映射器  <bean class="org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerMapping"/> -->  <!-- 配置处理器适配器  <bean class="org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerAdapter"/>-->    <!-- 注解驱动方式配置处理器映射器和处理器适配器,说明：  1.等于同时配置了RequestMappingHandlerMapping/RequestMappingHandlerAdapter  2.在企业项目中推荐使用的方式  -->  <mvc:annotation-driven/>      <!-- 配置视图解析器 -->  <bean class=*"org.springframework.web.servlet.view.InternalResourceViewResolver"*>  <!-- 配置视图的公共目录路径（前缀） -->  <property name=*"prefix"* value=*"/WEB-INF/jsp/"*></property>  <!-- 配置视图的扩展名称（后缀） -->  <property name=*"suffix"* value=*".jsp"*></property>  </bean>              </beans> |

# springmvc框架原理

1.用户请求

C：前端控制器

DispatcherServlet

接收用户请求

响应用户

处理器映射器：

HandlerMapping

根据请求的url找到处理器

2.请求处理器映射器找到处理器方法

M：处理器

controller

3.根据请求的url找到处理器方法

4.返回处理器

处理器适配器：

HandlerAdapter

执行处理器方法

5.请求处理器适配器执行处理器方法

6.执行处理器

7.返回模型和视图

8.返回模型和适图

视图解析器：

ViewResolver

解析视图

9.请求视图解析器解析视图

V：视图

jsp

10.解析视图

11.返回物理视图

12.响应用户

# ssm（springmvc+spring+mybatis）整合

## 整合思路

### 项目分层

1. 表现层：springmvc(controller)
2. 业务层：service
3. 持久层：mybatis(mapper)

controller，service，mapper都是javaBean，通过spring框架整合。

表现层：springmvc

业务层：service

持久层：mybatis

数据库：mysql

通过spring框架整合：

1. 整合表层。把表现层对象交给spring管理起来。完成在表现层调用业务层对象
2. 整合业务层。把业务层对象交给spring管理，把事务也管理起来。完成在业务层中调用持久层的对象。
3. 整合持久层。把持久层的对象交给spring管理起来。完成数据库CRUD操作

## 整合步骤

### 准备环境

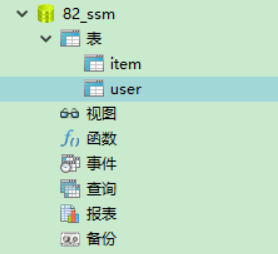
jdk：1.7

ide：eclipse

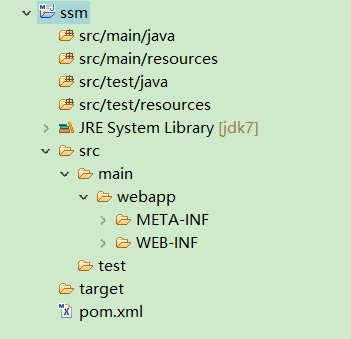
数据库：mysql

web容器：tomcat7

### 准备数据



### 创建整合项目



### 配置pom文件，导入jar包

* mybatis框架包
* spring框架包（包括springmvc）
* mybatis-spring整合包
* 数据库驱动包
* 数据库连接池包
* log4j.日志包
* jstl标签库包

|  |
| --- |
| <project xmlns=*"http://maven.apache.org/POM/4.0.0"* xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xsi:schemaLocation=*"http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"*>  <modelVersion>4.0.0</modelVersion>  <groupId>cn.itheima</groupId>  <artifactId>ssm</artifactId>  <version>0.0.1-SNAPSHOT</version>  <packaging>war</packaging>  <name>ssm Maven Webapp</name>  <url>http://maven.apache.org</url>  <properties>  <!-- spring版本号 -->  <spring.version>4.3.8.RELEASE</spring.version>  <!-- aspectj版本号 -->  <aspectj.version>1.6.12</aspectj.version>  <!-- jstl标签版本 -->  <jstl.version>1.2</jstl.version>  <!-- mybatis版本号 -->  <mybatis.version>3.4.5</mybatis.version>  <!-- mybatis-spring整合包版本 -->  <mybatis.spring.version>1.3.1</mybatis.spring.version>  <!-- mysql版本号 -->  <mysql.version>5.1.30</mysql.version>  <!-- dbcp数据源连接池jar包 -->  <dbcp.version>1.2.2</dbcp.version>  <!-- log4j日志包版本 -->  <slf4j.version>1.7.7</slf4j.version>  <log4j.version>1.2.17</log4j.version>  <!-- commons-lang版本 -->  <commons.lang.version>2.6</commons.lang.version>  </properties>  <dependencies>  <!-- springmvc依赖包 -->  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-core</artifactId>  <version>${spring.version}</version>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-web</artifactId>  <version>${spring.version}</version>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-tx</artifactId>  <version>${spring.version}</version>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-jdbc</artifactId>  <version>${spring.version}</version>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-webmvc</artifactId>  <version>${spring.version}</version>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-aop</artifactId>  <version>${spring.version}</version>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-context-support</artifactId>  <version>${spring.version}</version>  </dependency>  <dependency>  <groupId>org.aspectj</groupId>  <artifactId>aspectjweaver</artifactId>  <version>${aspectj.version}</version>  </dependency>  <!-- JSTL标签类 -->  <dependency>  <groupId>jstl</groupId>  <artifactId>jstl</artifactId>  <version>${jstl.version}</version>  </dependency>  <!-- mybatis核心包 -->  <dependency>  <groupId>org.mybatis</groupId>  <artifactId>mybatis</artifactId>  <version>${mybatis.version}</version>  </dependency>  <!-- mybatis-spring整合包 -->  <dependency>  <groupId>org.mybatis</groupId>  <artifactId>mybatis-spring</artifactId>  <version>${mybatis.spring.version}</version>  </dependency>  <!-- 导入Mysql数据库链接jar包 -->  <dependency>  <groupId>mysql</groupId>  <artifactId>mysql-connector-java</artifactId>  <version>${mysql.version}</version>  </dependency>  <!-- 导入dbcp数据源连接池jar包 -->  <dependency>  <groupId>commons-dbcp</groupId>  <artifactId>commons-dbcp</artifactId>  <version>${dbcp.version}</version>  </dependency>  <!-- 日志文件管理包 -->  <dependency>  <groupId>log4j</groupId>  <artifactId>log4j</artifactId>  <version>${log4j.version}</version>  </dependency>  <dependency>  <groupId>org.slf4j</groupId>  <artifactId>slf4j-api</artifactId>  <version>${slf4j.version}</version>  </dependency>  <dependency>  <groupId>org.slf4j</groupId>  <artifactId>slf4j-log4j12</artifactId>  <version>${slf4j.version}</version>  </dependency>  <dependency>  <groupId>commons-lang</groupId>  <artifactId>commons-lang</artifactId>  <version>${commons.lang.version}</version>  </dependency>  <!-- jsp依赖包，只在编译时需要 -->  <dependency>  <groupId>javax.servlet</groupId>  <artifactId>jsp-api</artifactId>  <version>2.0</version>  <scope>provided</scope>  </dependency>  </dependencies>  <build>  <finalName>ssm</finalName>  <plugins>  <plugin>  <groupId>org.apache.tomcat.maven</groupId>  <artifactId>tomcat7-maven-plugin</artifactId>  <version>2.1</version>  <configuration>  <!-- tomcat 的端口号 -->  <port>8080</port>  <!-- 访问应用的路径 -->  <path>/ssm</path>  <!-- URL按UTF-8进行编码，解决中文参数乱码 -->  <uriEncoding>UTF-8</uriEncoding>  <!-- tomcat名称 -->  <server>tomcat7</server>  </configuration>  </plugin>  </plugins>  </build>  </project> |

### 编写配置文件

#### sqlMapConfig.xml

* 配置别名

|  |
| --- |
| <?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>    <!--配置别名 -->  <typeAliases>  <!-- 包扫描的方式配置别名 -->  <package name=*"cn.itheima.ssm.po"*/>  </typeAliases>  </configuration> |

#### springmvc.xml

* 配置包扫描controller
* 配置处理器映射器
* 配置处理器适配器
* 配置视图解析器

|  |
| --- |
| <?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"*>    <!-- 配置包扫描controller -->  <context:component-scan base-package=*"cn.itheima.ssm.controller"*/>    <!-- 注解驱动的方式配置处理器映射器和处理器适配器 -->  <mvc:annotation-driven/>    <!-- 配置视图解析器 -->  <bean class=*"org.springframework.web.servlet.view.InternalResourceViewResolver"*>  <!-- 配置视图的公共目录路径（前缀） -->  <property name=*"prefix"* value=*"/WEB-INF/jsp/"*></property>  <!--配置视图的扩展名称（后缀） -->  <property name=*"suffix"* value=*".jsp"*></property>  </bean>  </beans> |

#### applicationContext-dao.xml

* 配置数据源对象（DataSource）
* 配置SqlSessionFactory
* 配置mapper扫描器（MapperScannerConfigurer）

|  |
| --- |
| <?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"*>    <!-- 加载db.properties文件 -->  <context:property-placeholder location=*"classpath:db.properties"*/>    <!-- 配置数据源 -->  <bean id=*"dataSource"* class=*"org.apache.commons.dbcp.BasicDataSource"*  destroy-method=*"close"*>    <property name=*"driverClassName"* value=*"${db.driverClassName}"*/>  <property name=*"url"* value=*"${db.url}"* />  <property name=*"username"* value=*"${db.username}"* />  <property name=*"password"* value=*"${db.password}"* />    <!-- 最大连接数量 -->  <property name=*"maxActive"* value=*"${db.maxActive}"*/>  <!-- 最小空闲连接 -->  <property name=*"minIdle"* value=*"${db.minIdle}"*/>  <!-- 最大空闲连接 -->  <property name=*"maxIdle"* value=*"${db.maxIdle}"*/>  <!-- 初始化连接数 -->  <property name=*"initialSize"* value=*"${db.initialSize}"*/>  <!-- 超时等待时间以毫秒为单位 -->  <property name=*"maxWait"* value=*"${db.maxWait}"*/>  </bean>    <!-- 配置sqlSessionFactory -->  <bean id=*"sqlSessionFactory"* class=*"org.mybatis.spring.SqlSessionFactoryBean"*>  <!-- 注入数据源对象 -->  <property name=*"dataSource"* ref=*"dataSource"*></property>  <!-- 加载mybatis主配置文件 -->  <property name=*"configLocation"* value=*"classpath:mybatis/sqlMapConfig.xml"*></property>  </bean>    <!-- 配置mapper扫描器 -->  <bean class=*"org.mybatis.spring.mapper.MapperScannerConfigurer"*>  <!-- 配置要扫描的包,说明：  1.如果有多个包，在同一个父包下，配置父包即可  2.如果不在同一个父包，以半角逗号进行分割  -->  <property name=*"basePackage"* value=*"cn.itheima.ssm.mapper"*/>  </bean>    </beans> |

#### applicationContext-service.xml

* 配置包扫描service

|  |
| --- |
| <?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"*>    <!-- 配置service扫描 -->  <context:component-scan base-package=*"cn.itheima.ssm.service"*/>  </beans> |

#### applicationContext-trans.xml

* 配置事务管理器（transactionManager）
* 配置aop（aop:config）
* 配置通知（txAdivce）

|  |
| --- |
| <?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"*>    <!-- 配置事务管理器 -->  <bean id=*"transactionManager"*  class=*"org.springframework.jdbc.datasource.DataSourceTransactionManager"*>  <!-- 指定数据源 -->  <property name=*"dataSource"* ref=*"dataSource"*/>  </bean>    <!-- 配置事务通知 -->  <tx:advice id=*"txAdvice"* transaction-manager=*"transactionManager"*>  <tx:attributes>  <!-- 配置传播行为 -->  <tx:method name=*"add\*"* propagation=*"REQUIRED"*/>  <tx:method name=*"insert\*"* propagation=*"REQUIRED"*/>  <tx:method name=*"save\*"* propagation=*"REQUIRED"*/>  <tx:method name=*"update\*"* propagation=*"REQUIRED"*/>  <tx:method name=*"delete\*"* propagation=*"REQUIRED"*/>    <tx:method name=*"find\*"* propagation=*"SUPPORTS"* read-only=*"true"*/>  <tx:method name=*"select\*"* propagation=*"SUPPORTS"* read-only=*"true"*/>  <tx:method name=*"get\*"* propagation=*"SUPPORTS"* read-only=*"true"*/>  <tx:method name=*"query\*"* propagation=*"SUPPORTS"* read-only=*"true"*/>  </tx:attributes>  </tx:advice>    <!-- 事务切面配置 -->  <aop:config>  <!-- 指定cn.itcast.ssm.service包下的任意类的任意方法，需要事务 -->  <aop:advisor advice-ref=*"txAdvice"*  pointcut=*"execution(\* cn.itheima.ssm.service..\*.\*(..))"*/>  </aop:config>    </beans> |

#### db.properties

|  |
| --- |
| db.driverClassName=com.mysql.jdbc.Driver  db.url=jdbc:mysql://127.0.0.1:3306/82\_ssm?characterEncoding=utf-8  db.username=root  db.password=admin  db.maxActive=10  db.minIdle=2  db.maxIdle=5  db.initialSize=5  db.maxWait=6000 |

#### log4j.properties

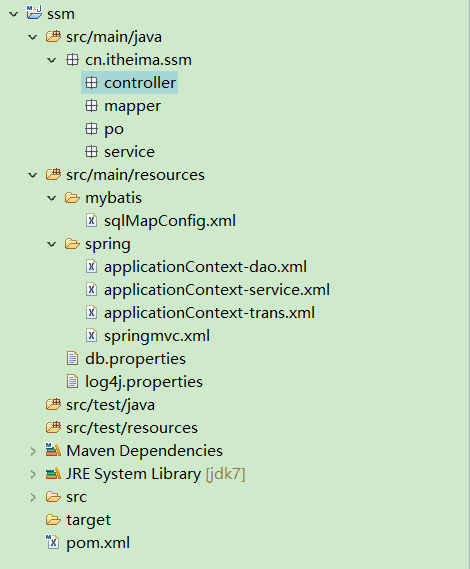
|  |
| --- |
| # Global logging configuration  log4j.rootLogger=INFO, stdout  # Console output...  log4j.appender.stdout=org.apache.log4j.ConsoleAppender  log4j.appender.stdout.layout=org.apache.log4j.PatternLayout  log4j.appender.stdout.layout.ConversionPattern=%5p [%t] - %m%n |

#### web.xml

* 配置加载spring配置文件（context-param）
* 配置spring的监听器（ContextLoaderListener）
* 配置前端控制器（DispatcherServlet）

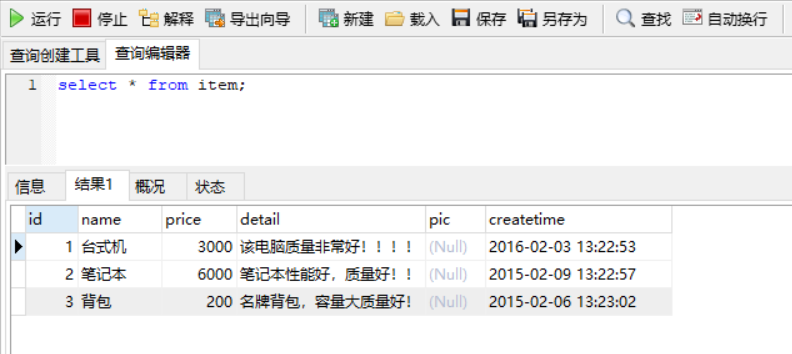
|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <web-app xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"* xmlns=*"http://java.sun.com/xml/ns/javaee"* xmlns:web=*"http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd"* xsi:schemaLocation=*"http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd"* id=*"WebApp\_ID"* version=*"2.5"*>  <display-name>ssm</display-name>    <!-- 加载spring配置文件 -->  <context-param>  <param-name>contextConfigLocation</param-name>  <param-value>classpath:spring/applicationContext-\*.xml</param-value>  </context-param>    <!-- 配置spring的监听器 -->  <listener>  <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>  </listener>    <!-- 配置前端控制器 -->  <servlet>  <servlet-name>ssm</servlet-name>  <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>    <!-- 加载springmvc的主配置文件 -->  <init-param>  <param-name>contextConfigLocation</param-name>  <param-value>classpath:spring/springmvc.xml</param-value>  </init-param>    <!-- 配置tomcat启动就加载前端控制器 -->  <load-on-startup>1</load-on-startup>  </servlet>    <servlet-mapping>  <servlet-name>ssm</servlet-name>  <!-- 配置请求的url规则，说明：  1.\*.do，以.do结尾的请求进入前端控制器  2./，表示所有请求都进入前端控制器  -->  <url-pattern>\*.do</url-pattern>  </servlet-mapping>    <!-- 默认的欢迎页面 -->  <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>  </web-app> |

### 整合好的项目



## 需求

查询全部商品列表数据，显示到商品列表页面。



## 需求实现

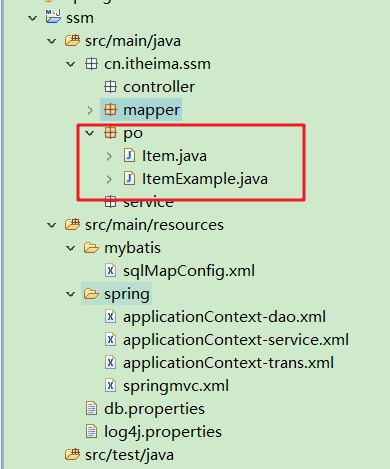
### 表现层开发

#### 编写商品列表jsp页面

|  |
| --- |
| <%@ page language=*"java"* contentType=*"text/html; charset=UTF-8"*  pageEncoding=*"UTF-8"*%>  <%@ taglib uri=*"http://java.sun.com/jsp/jstl/core"* prefix=*"c"*%>  <%@ taglib uri=*"http://java.sun.com/jsp/jstl/fmt"* prefix=*"fmt"*%>  <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"  "http://www.w3.org/TR/html4/loose.dtd">  <html>  <head>  <meta http-equiv=*"Content-Type"* content=*"text/html; charset=UTF-8"*>  <title>查询商品列表</title>  </head>  <body>  <form action=*"*${pageContext.request.contextPath }*/queryItem.do"*  method=*"post"*>  查询条件：  <table width=*"100%"* border=*1*>  <tr>  <td>  <!--商品名称：<input type="text" name="item.name" value=""/>-->  <input type=*"submit"* value=*"查询"* />  </td>  </tr>  </table>  商品列表：  <table width=*"100%"* border=*1*>  <tr>  <td>商品名称</td>  <td>商品价格</td>  <td>生产日期</td>  <td>商品描述</td>  <td>操作</td>  </tr>  <c:forEach items=*"*${itemList}*"* var=*"item"*>  <tr>  <td>${item.name }</td>  <td>${item.price }</td>  <td><fmt:formatDate value=*"*${item.createtime}*"*  pattern=*"yyyy-MM-dd HH:mm:ss"* /></td>  <td>${item.detail }</td>  <td><a  href=*"*${pageContext.request.contextPath }*/queryItemById.do?id=*${item.id}*"*>修改</a></td>  </tr>  </c:forEach>  </table>  </form>  </body>  </html> |

#### 编写商品的实体类

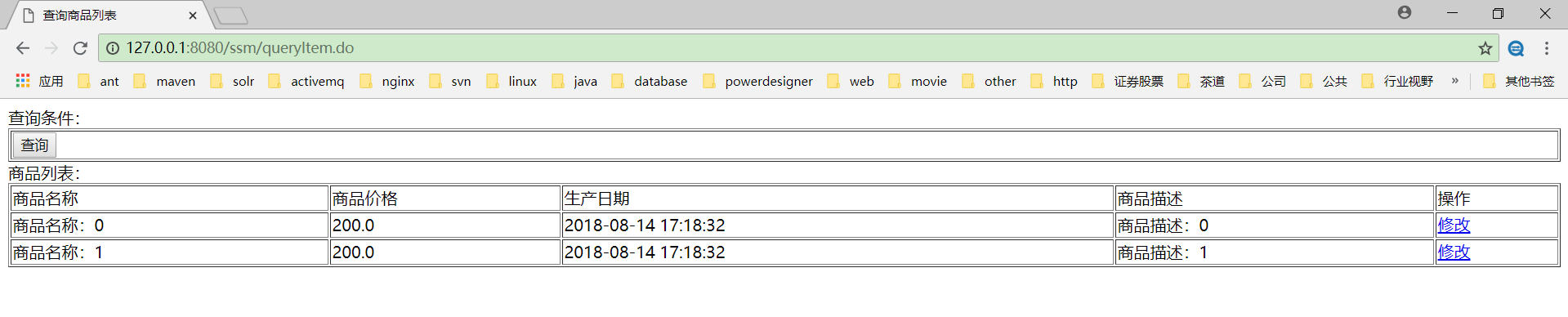
说明：使用逆向工程生成。



#### 编写商品controller

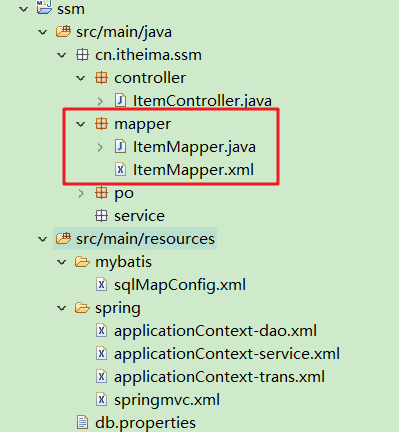
|  |
| --- |
| **package** cn.itheima.ssm.controller;  **import** java.util.ArrayList;  **import** java.util.Date;  **import** java.util.List;  **import** org.springframework.stereotype.Controller;  **import** org.springframework.web.bind.annotation.RequestMapping;  **import** org.springframework.web.servlet.ModelAndView;  **import** cn.itheima.ssm.po.Item;  /\*\*  \* **@ClassName**: ItemController  \* **@Description**: 商品处理器  \* **@author** 传智 小杨老师  \* **@date** 2018-8-14 下午5:11:45  \*  \*/  @Controller  **public** **class** ItemController {    /\*\*  \* 查询全部商品列表数据  \* <form action="${pageContext.request.contextPath }/queryItem.do"  \*/  @RequestMapping("/queryItem.do")  **public** ModelAndView queryItem(){    // 1.创建ModelAndView  ModelAndView mav = **new** ModelAndView();    // 2.响应商品列表数据  // 准备静态的商品数据  List<Item> itemList = **new** ArrayList<Item>();  **for**(**int** i=0;i<2;i++){  // 创建商品对象  Item item = **new** Item();    // 商品Id  item.setId(i+1);  // 商品名称  item.setName("商品名称："+i);  // 商品价格  item.setPrice(200f);  // 商品日期  item.setCreatetime(**new** Date());  // 商品描述  item.setDetail("商品描述："+i);    itemList.add(item);  }    mav.addObject("itemList", itemList);    // 3.响应页面  mav.setViewName("item/itemList");    **return** mav;    }  } |

#### 初步测试



### 持久层开发

说明：使用逆向工程生成。



### 业务层开发

#### 编写商品service接口

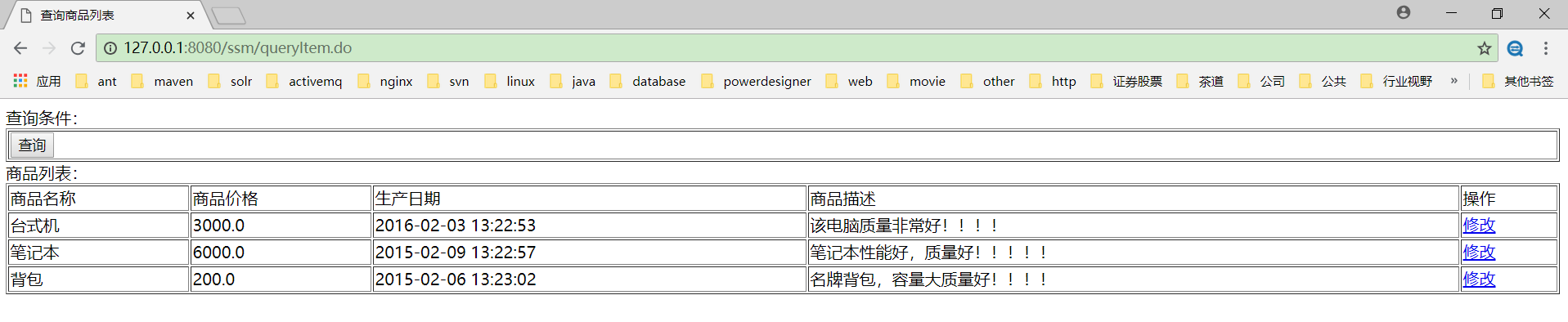
|  |
| --- |
| **package** cn.itheima.ssm.service;  **import** java.util.List;  **import** cn.itheima.ssm.po.Item;  /\*\*  \* **@ClassName**: ItemService  \* **@Description**: 商品service接口  \* **@author** 传智 小杨老师  \* **@date** 2018-8-14 下午5:20:14  \*  \*/  **public** **interface** ItemService {    /\*\*  \* 查询全部商品列表数据  \*/  List<Item> findAllItems();  } |

#### 编写商品service实现类

|  |
| --- |
| **package** cn.itheima.ssm.service.impl;  **import** java.util.List;  **import** org.springframework.beans.factory.annotation.Autowired;  **import** org.springframework.stereotype.Service;  **import** cn.itheima.ssm.mapper.ItemMapper;  **import** cn.itheima.ssm.po.Item;  **import** cn.itheima.ssm.service.ItemService;  /\*\*  \* **@ClassName**: ItemServiceImpl  \* **@Description**: 商品service实现类  \* **@author** 传智 小杨老师  \* **@date** 2018-8-14 下午5:21:08  \*  \*/  @Service  **public** **class** ItemServiceImpl **implements** ItemService {    // 注入商品mapper  @Autowired  **private** ItemMapper itemMapper;    /\*\*  \* 查询全部商品列表数据  \*/  **public** List<Item> findAllItems() {  List<Item> list = itemMapper.selectByExample(**null**);  **return** list;  }  } |

### 在queryItem方法中增加查询数据库商品

|  |
| --- |
| /\*\*  \* 查询全部商品列表数据  \* <form action="${pageContext.request.contextPath }/queryItem.do"  \*/  @RequestMapping("/queryItem.do")  **public** ModelAndView queryItem(){    // 1.创建ModelAndView  ModelAndView mav = **new** ModelAndView();    // 2.响应商品列表数据  // 准备静态的商品数据  /\*List<Item> itemList = new ArrayList<Item>();  for(int i=0;i<2;i++){  // 创建商品对象  Item item = new Item();    // 商品Id  item.setId(i+1);  // 商品名称  item.setName("商品名称："+i);  // 商品价格  item.setPrice(200f);  // 商品日期  item.setCreatetime(new Date());  // 商品描述  item.setDetail("商品描述："+i);    itemList.add(item);  }\*/    // 查询数据库中的商品数据，替换静态的商品数据  List<Item> itemList = itemService.findAllItems();  mav.addObject("itemList", itemList);    // 3.响应页面  mav.setViewName("item/itemList");    **return** mav;    } |



# springmvc参数绑定

说明：参数绑定指的是通过处理器方法的形参，接收请求的url或者表单的参数数据。

## 参数绑定的案例

### 需求

根据商品id查询商品数据，显示到商品编辑页面。

http://127.0.0.1:8080/ssm/queryItemById.do?id=1

### 需求实现

#### 声明商品service接口方法

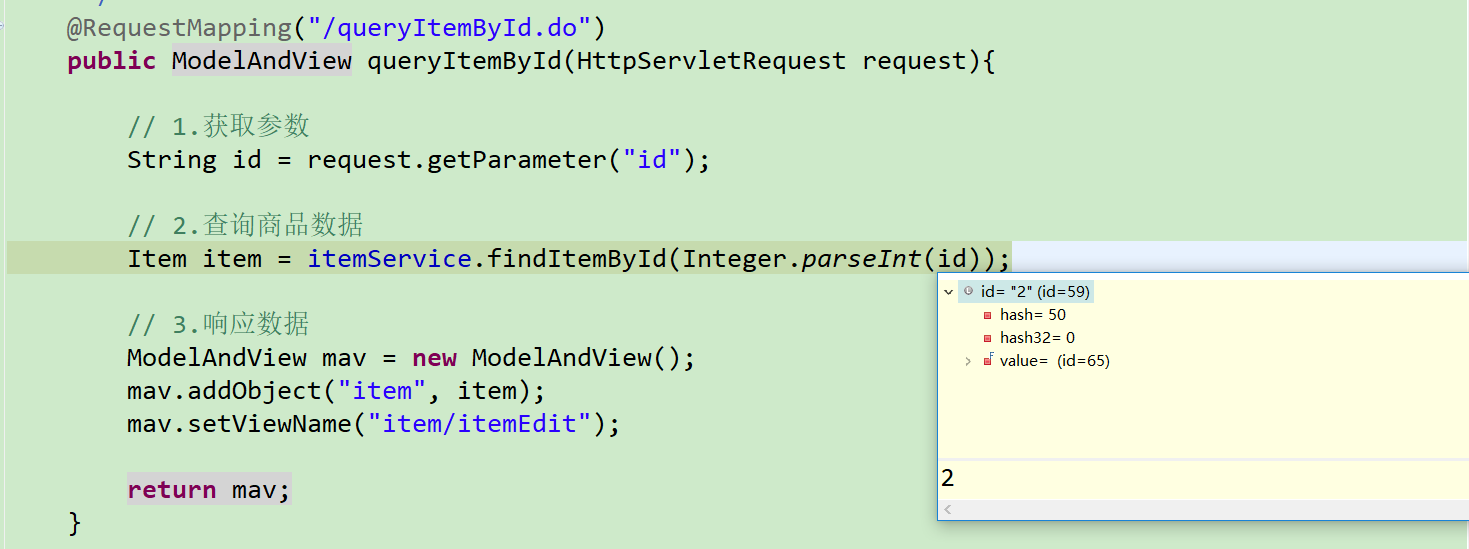
|  |
| --- |
| **package** cn.itheima.ssm.service;  **import** java.util.List;  **import** cn.itheima.ssm.po.Item;  /\*\*  \* **@ClassName**: ItemService  \* **@Description**: 商品service接口  \* **@author** 传智 小杨老师  \* **@date** 2018-8-14 下午5:20:14  \*  \*/  **public** **interface** ItemService {    /\*\*  \* 查询全部商品列表数据  \*/  List<Item> findAllItems();    /\*\*  \* 根据商品Id查询商品  \*/  Item findItemById(Integer id);  } |

#### 实现商品service接口方法

|  |
| --- |
| /\*\*  \* 根据商品Id查询商品数据  \*/  **public** Item findItemById(Integer id) {    **return** itemMapper.selectByPrimaryKey(id);  } |

#### 在商品controller中，增加根据商品Id查询的方法

|  |
| --- |
| /\*\*  \* 根据商品Id查询商品数据  \* http://127.0.0.1:8080/ssm/queryItemById.do?id=1  \*  \*形参request，接收请求商品id参数。在springmvc执行方法的时候，会把request对象传递过来  \*/  @RequestMapping("/queryItemById.do")  **public** ModelAndView queryItemById(HttpServletRequest request){    // 1.获取参数  String id = request.getParameter("id");    // 2.查询商品数据  Item item = itemService.findItemById(Integer.*parseInt*(id));    // 3.响应数据  ModelAndView mav = **new** ModelAndView();  mav.addObject("item", item);  mav.setViewName("item/itemEdit");    **return** mav;  } |



## 默认支持的参数类型

### HttpServletRequest

作用：通过request，获取请求数据。

### HttpServletResponse

作用：通过response响应数据。

### HttpSession

作用：通过session操作会话域的数据。

### Model/ModelMap

说明：

1. Model是模型，是一个接口。用于封装模型数据
2. ModelMap是一个实现类。使用Model和ModelMap都是一样的。如果使用Model，在执行的时候，会实例化成ModelMap
3. Model封装模型数据，就可以不使用ModelAndView，视图可以使用字符串String响应。

|  |
| --- |
| /\*\*  \* 使用Model响应模型数据；使用字符串String响应模型数据。  \*/  @RequestMapping("/queryItemById.do")  **public** String queryItemById(Model model,HttpServletRequest request){    // 1.获取参数  String id = request.getParameter("id");    // 2.查询商品数据  Item item = itemService.findItemById(Integer.*parseInt*(id));    // 3.响应数据  model.addAttribute("item", item);    **return** "item/itemEdit";  } |

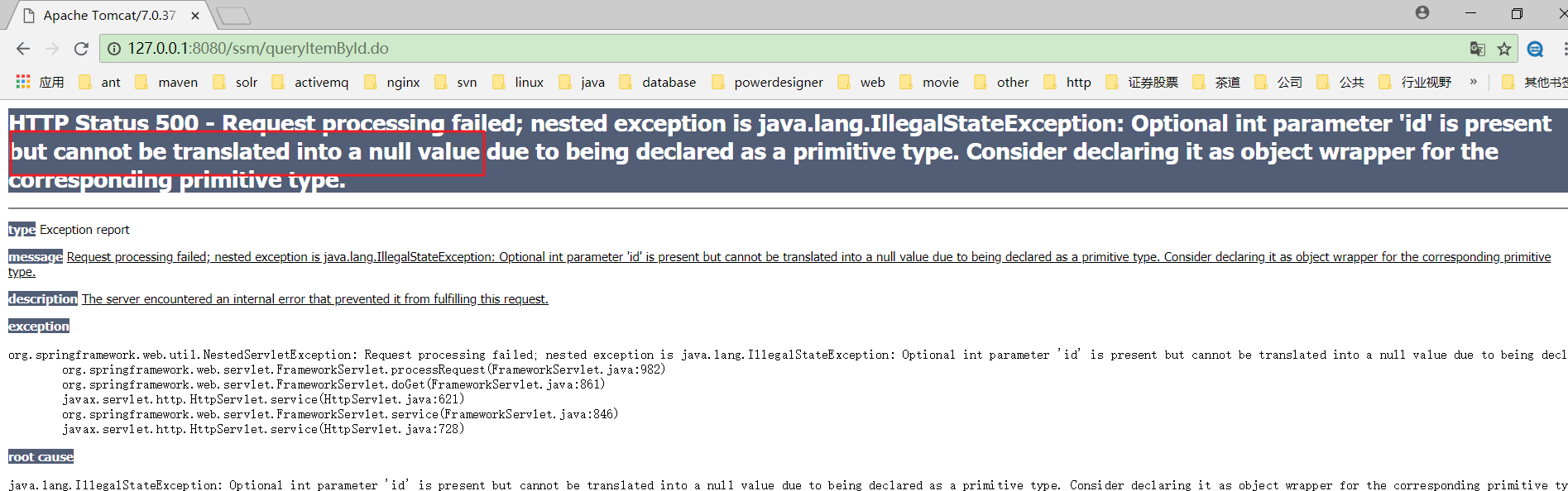
## java简单类型

### 使用简单类型Integer接收请求的商品Id

|  |
| --- |
| /\*\*  \* 使用简单类型Integer接收请求的商品Id  \* http://127.0.0.1:8080/ssm/queryItemById.do?id=1  \*/  @RequestMapping("/queryItemById.do")  **public** String queryItemById(Model model,Integer id){    // 1.查询商品数据  Item item = itemService.findItemById(id);    // 2.响应数据  model.addAttribute("item", item);    **return** "item/itemEdit";  } |



注意事项：使用简单类型接收参数数据，推荐使用简单类型的包装类型（Integer），不建议使用简单类型的基础类型。原因是基础类型不能为null。如果不传递会报异常。



### 常用的简单类型

|  |  |  |
| --- | --- | --- |
| 类型 | 包装类型 | 基础类型 |
| 整型 | Integer | int |
| 长整型 | Long | long |
| 单精度浮点型 | Float | float |
| 双精度浮点型 | Double | double |
| 字符串 | String | String |

### @RequestParam注解【了解】

作用：设置请求的参数名称，与方法的形参名称匹配。

|  |
| --- |
| /\*\*  \* 学习@RequestPara注解:  \* 作用：设置请求的参数名称，与方法的形参名称匹配  \* 属性：  \* value：设置请求的参数名称  \* required：设置参数是否必须要传递。true：要传递；false：可以不传递。默认是true。  \* defaultValue：设置参数的默认值。如果传递，使用实际的参数值；不传递使用默认值  \*  \* http://127.0.0.1:8080/ssm/queryItemById.do?itemId=1  \*/  @RequestMapping("/queryItemById.do")  **public** String queryItemById(Model model,@RequestParam(value="itemId",  required=**true**,  defaultValue="2")Integer id){    // 1.查询商品数据  Item item = itemService.findItemById(id);    // 2.响应数据  model.addAttribute("item", item);    **return** "item/itemEdit";  } |

## pojo参数类型

### 需求

修改商品数据，保存数据库。

### 需求实现

#### 声明商品service接口方法

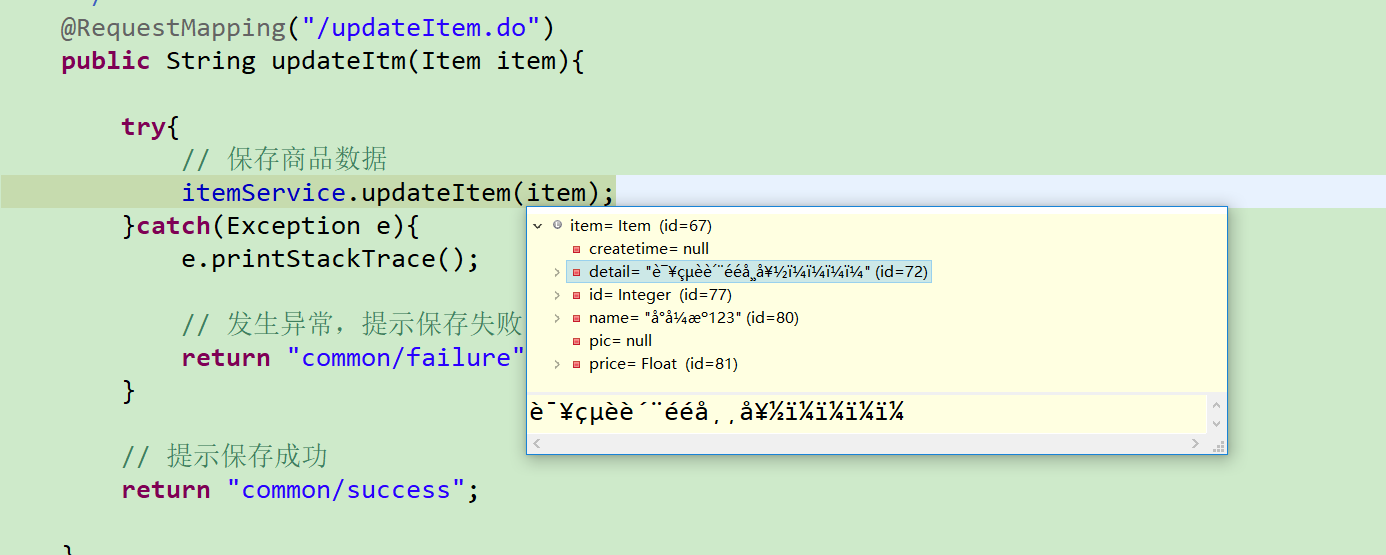
|  |
| --- |
| **package** cn.itheima.ssm.service;  **import** java.util.List;  **import** cn.itheima.ssm.po.Item;  /\*\*  \* **@ClassName**: ItemService  \* **@Description**: 商品service接口  \* **@author** 传智 小杨老师  \* **@date** 2018-8-14 下午5:20:14  \*  \*/  **public** **interface** ItemService {    /\*\*  \* 查询全部商品列表数据  \*/  List<Item> findAllItems();    /\*\*  \* 根据商品Id查询商品  \*/  Item findItemById(Integer id);    /\*\*  \* 修改商品数据，保存数据库  \*/  **void** updateItem(Item item);  } |

#### 实现商品service接口方法

|  |
| --- |
| /\*\*  \* 修改商品数据，保存数据库  \*/  **public** **void** updateItem(Item item){  itemMapper.updateByPrimaryKeySelective(item);  } |

#### 在商品Controller中，增加修改保存商品的方法

|  |
| --- |
| /\*\*  \* 修改商品数据，保存数据库  \* http://127.0.0.1:8080/ssm/updateItem.do  \*/  @RequestMapping("/updateItem.do")  **public** String updateItm(Item item){    **try**{  // 保存商品数据  itemService.updateItem(item);  }**catch**(Exception e){  e.printStackTrace();    // 发生异常，提示保存失败  **return** "common/failure";  }    // 提示保存成功  **return** "common/success";    } |



问题：商品参数数据可以接收到，但是出现了乱码。乱码的原因是tomcat默认的字符集编码是ISO-8859-1不支持中文。

#### 中文乱码的解决

说明：spring框架提供了一个字符集编码的过滤器（CharacterEncodingFilter），解决post请求的中文乱码。

在web.xml配置字符集编码过滤器：

|  |
| --- |
| <!-- 配置字符集编码过滤器 -->  <filter>  <filter-name>encodingFilter</filter-name>  <filter-class>org.springframework.web.filter.CharacterEncodingFilter</filter-class>    <!-- 指定使用的字符集编码 -->  <init-param>  <param-name>encoding</param-name>  <param-value>UTF-8</param-value>  </init-param>  </filter>    <filter-mapping>  <filter-name>encodingFilter</filter-name>  <!-- 配置所有请求都经过字符集编码过滤器处理 -->  <url-pattern>/\*</url-pattern>  </filter-mapping> |

## pojo包装类型

### 需求

使用pojo包装类型，接收综合查询条件

商品名称：<input type="text" name="item.name" value=""/>

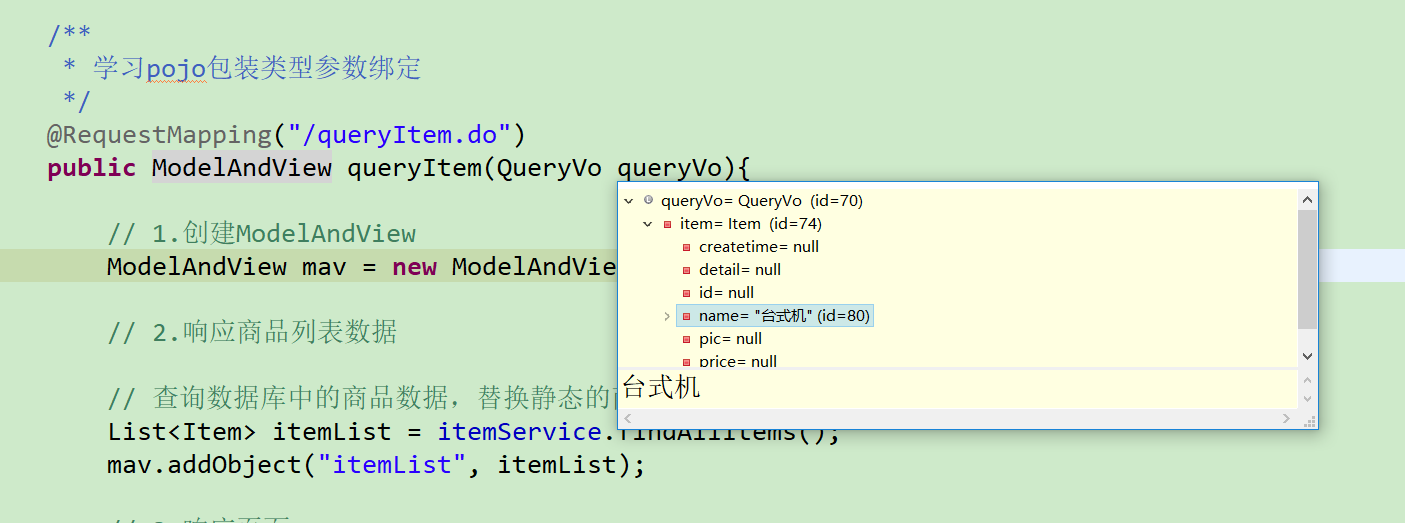
### 需求实现

#### 编写pojo包装类型

|  |
| --- |
| **package** cn.itheima.ssm.po;  /\*\*  \* **@ClassName**: QueryVo  \* **@Description**: pojo包装类型  \* **@author** 传智 小杨老师  \* **@date** 2018-8-14 下午6:45:20  \*  \*/  **public** **class** QueryVo {    // 包装商品对象  /\*\*  \* 商品名称：<input type="text" name="item.name" value=""/>  \*/  **private** Item item;  /\*\*  \* **@return** the item  \*/  **public** Item getItem() {  **return** item;  }  /\*\*  \* **@param** item the item to set  \*/  **public** **void** setItem(Item item) {  **this**.item = item;  }      } |

#### 在querytItem方法中，增加形参QueryVo

|  |
| --- |
| /\*\*  \* 学习pojo包装类型参数绑定  \*/  @RequestMapping("/queryItem.do")  **public** ModelAndView queryItem(QueryVo queryVo){    // 1.创建ModelAndView  ModelAndView mav = **new** ModelAndView();    // 2.响应商品列表数据    // 查询数据库中的商品数据，替换静态的商品数据  List<Item> itemList = itemService.findAllItems();  mav.addObject("itemList", itemList);    // 3.响应页面  mav.setViewName("item/itemList");    **return** mav;    } |



#### 解决get请求乱码

##### 方式一：直接使用UTF-8编码，重新编码参数

|  |
| --- |
| /\*\*  \* 学习pojo包装类型参数绑定  \*/  @RequestMapping("/queryItem.do")  **public** ModelAndView queryItem(QueryVo queryVo){    // get请求乱码解决====================start  **try**{  // 1.获取商品名称参数  String name = queryVo.getItem().getName();  // 2.根据ISO-8859-1获取到商品名称字节码  **byte**[] bytes = name.getBytes("ISO-8859-1");  // 3.使用UTF-8重新编码商品名称  name = **new** String(bytes, "UTF-8");  // 4.设置商品名称参数  queryVo.getItem().setName(name);  }**catch**(Exception e){  e.printStackTrace();  }  // get请求乱码解决====================end    // 1.创建ModelAndView  ModelAndView mav = **new** ModelAndView();    // 2.响应商品列表数据    // 查询数据库中的商品数据，替换静态的商品数据  List<Item> itemList = itemService.findAllItems();  mav.addObject("itemList", itemList);    // 3.响应页面  mav.setViewName("item/itemList");    **return** mav;    } |

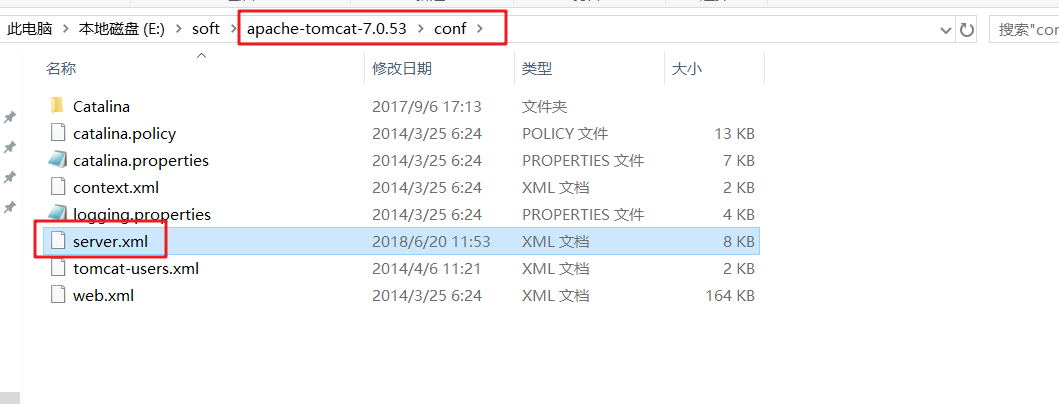
##### 方式二：直接修改tomcat的默认字符集编码

1. 项目开发阶段配置tomcat插件

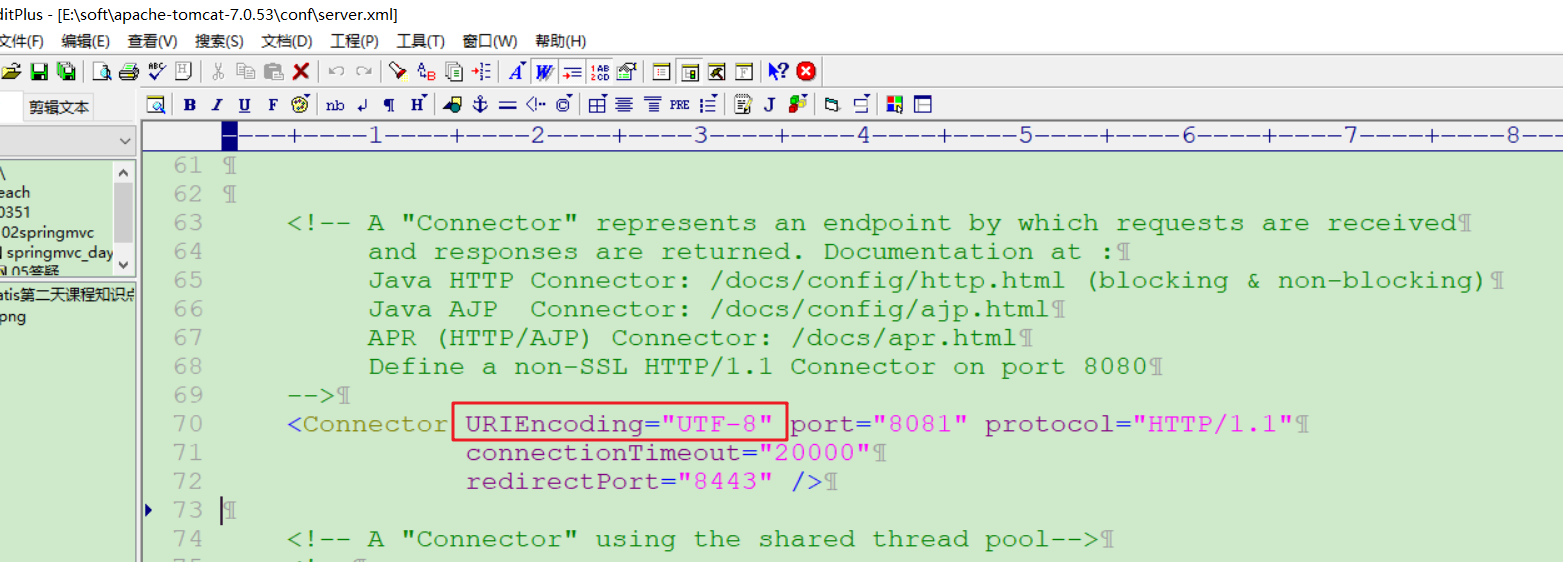
|  |
| --- |
| <build>  <finalName>ssm</finalName>  <plugins>  <plugin>  <groupId>org.apache.tomcat.maven</groupId>  <artifactId>tomcat7-maven-plugin</artifactId>  <version>2.1</version>  <configuration>  <!-- tomcat 的端口号 -->  <port>8080</port>  <!-- 访问应用的路径 -->  <path>/ssm</path>  <!-- URL按UTF-8进行编码，解决中文参数乱码 -->  <uriEncoding>UTF-8</uriEncoding>  <!-- tomcat名称 -->  <server>tomcat7</server>  </configuration>  </plugin>  </plugins>  </build> |

1. 项目部署阶段的配置

图一：



图二：



|  |
| --- |
| <Connector URIEncoding="UTF-8" port="8081" protocol="HTTP/1.1"  connectionTimeout="20000"  redirectPort="8443" /> |

## 自定义参数类型绑定

说明：自定义参数类型，主要解决比如日期类型的参数数据，格式比较多不固定，需要根据业务需求才能确定。

### 自定义日期类型参数的转换器

说明：spring框架提供了转换器的接口（Converter）。

|  |
| --- |
| **package** cn.itheima.ssm.converter;  **import** java.text.ParseException;  **import** java.text.SimpleDateFormat;  **import** java.util.Date;  **import** org.springframework.core.convert.converter.Converter;  /\*\*  \* **@ClassName**: DateConterver  \* **@Description**: 自定义日期类型的转换器  \* **@author** 传智 小杨老师  \* **@date** 2018-8-14 下午7:04:53  \*  \* Converter<S, T>  \* S:source：源，转换之前的数据，这里是字符串String数据  \* T:target：目标，转换之后的数据，这里是Date类型的数据  \*  \*/  **public** **class** DateConterver **implements** Converter<String, Date> {  /\*\*  \* 实现转换逻辑  \*/  **public** Date convert(String source) {  // 日期格式化对象2016-02-03 13:22:53  SimpleDateFormat format = **new** SimpleDateFormat("yyyy-MM-dd HH:mm:ss");    **try** {  // 转换成功，直接返回  **return** format.parse(source);  } **catch** (ParseException e) {  e.printStackTrace();  }  // 转换失败，返回null  **return** **null**;  }  } |

### 配置自定义日期转换器

在springmvc.xml中配置：

|  |
| --- |
| <!-- 注解驱动的方式配置处理器映射器和处理器适配器 -->  <mvc:annotation-driven conversion-service=*"conversionService"*/>    <!-- 配置转换器conversionService -->  <bean id=*"conversionService"*  class=*"org.springframework.format.support.FormattingConversionServiceFactoryBean"*>  <property name=*"converters"*>  <set>  <bean class=*"cn.itheima.ssm.converter.DateConterver"*/>  </set>  </property>  </bean> |

# springmvc与struts2比较

## 相同点

springmvc和struts2都是基于mvc设计模式的表现层框架，都用于web项目的开发。

## 不同点

1. 前端控制器不一样。springmvc的前端控制器是servlet（DispatcherServlet）。struts2的前端控制器是filter（StrutsPrepareAndExecutorFilter）。
2. 请求参数的接收方式不一样。springmvc通过方法的形参接收请求的参数，基于方法的开发，线程安全。可以设计为单例或多例的开发，推荐使用单例开发，执行效率更高。struts2通过类的成员变量接收请求参数，基于类的开发，线程不安全。只能设计为多例的开发。