**每日作业卷**

**SpringMVC第1天**

极客营

# 关卡1

## 训练案例1

### 训练描述

使用课堂的资料实现SpringMVC的入门案例

要求

1. 创建工程导入jar包，加入jsp

2. 编写springmvc.xml和web.xml配置文件

3. 编写pojo和Controller实现商品列表展示

### 操作步骤答案

1. 创建工程导入jar包，加入jsp

2. 编写springmvc.xml和web.xml配置文件

编写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: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扫描包 -->

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

</beans>

在web.xml加入以下配置

<!-- 配置SpringMVC前端控制器 -->

<servlet>

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

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

</servlet-mapping>

3. 编写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;

set/get。。。

}

4. 编写Controller实现商品列表展示"

@Controller

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

// @RequestMapping：里面放的是请求的url，和用户请求的url进行匹配

// action可以写也可以不写

@RequestMapping("/itemList.action")

**public** ModelAndView queryItemList() {

// 创建页面需要显示的商品数据

List<Item> list = **new** ArrayList<>();

list.add(**new** Item(1, "1华为 荣耀8", 2399, **new** Date(), "质量好！1"));

list.add(**new** Item(2, "2华为 荣耀8", 2399, **new** Date(), "质量好！2"));

list.add(**new** Item(3, "3华为 荣耀8", 2399, **new** Date(), "质量好！3"));

list.add(**new** Item(4, "4华为 荣耀8", 2399, **new** Date(), "质量好！4"));

list.add(**new** Item(5, "5华为 荣耀8", 2399, **new** Date(), "质量好！5"));

list.add(**new** Item(6, "6华为 荣耀8", 2399, **new** Date(), "质量好！6"));

// 创建ModelAndView，用来存放数据和视图

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

// 设置数据到模型中

modelAndView.addObject("itemList", list);

// 设置视图jsp，需要设置视图的物理地址

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

**return** modelAndView;

}

}

## 训练案例2

### 训练描述

使用课堂的资料实现Spring+SpringMVC+MyBatis的整合，

要求：

1. 创建工程，并加入jar包和jsp

2. 编写dao层的SqlMapConfig.xml和applicationContext-dao.xml

3. 编写service层的applicationContext-service.xml和applicationContext-trans.xml

4. 编写Controller的Springmvc.xml

5. 编写web.xml

6. 使用逆向工程生成pojo和Mapper代码

7. 编写service和Controller实现数据库中的商品列表显示

### 操作步骤答案

1. 创建工程，并加入jar包和jsp
2. 编写dao层的SqlMapConfig.xml、applicationContext-dao.xml和db.properties

<?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>

<?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>

<!-- 配置SqlSessionFactory -->

<bean class=*"org.mybatis.spring.SqlSessionFactoryBean"*>

<!-- 配置数据源 -->

<property name=*"dataSource"* ref=*"dataSource"* />

<!-- 配置MyBatis的核心配置文件 -->

<property name=*"configLocation"* value=*"classpath:mybatis/SqlMapConfig.xml"* />

</bean>

<!-- 配置Mapper扫描 -->

<bean class=*"org.mybatis.spring.mapper.MapperScannerConfigurer"*>

<!-- 配置扫描包 -->

<property name=*"basePackage"* value=*"com.igeek.ssm.mapper"* />

</bean>

</beans>

jdbc.driver=com.mysql.jdbc.Driver

jdbc.url=jdbc:mysql://localhost:3306/springmvc?characterEncoding=utf-8

jdbc.username=root

jdbc.password=root

1. 编写service层的applicationContext-service.xml和applicationContext-trans.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"*>

<!-- 配置service扫描 -->

<context:component-scan base-package=*" com.igeek.ssm.service"* />

</beans>

<?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:method name=*"query\*"* propagation=*"SUPPORTS"* read-only=*"true"* />

</tx:attributes>

</tx:advice>

<!-- 切面 -->

<aop:config>

<aop:advisor advice-ref=*"txAdvice"*

pointcut=*"execution(\*com.igeek.ssm.service.\*.\*(..))"* />

</aop:config>

</beans>

1. 编写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: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扫描 -->

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

<!-- 配置注解驱动 处理器映射器和处理器适配器 -->

<mvc:annotation-driven />

<bean

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

<!-- 配置前缀 -->

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

<!-- 配置后缀 -->

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

</bean>

</beans>

1. 编写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.jsp</welcome-file>

</welcome-file-list>

<!-- 配置spring -->

<context-param>

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

<param-value>classpath:spring/applicationContext-\*.xml</param-value>

</context-param>

<!-- 配置spring监听器 -->

<listener>

<listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>

</listener>

<!-- 配置springMVC前端控制器 -->

<servlet>

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

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

<!-- 配置springMVC的核心配置文件 -->

<init-param>

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

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

</init-param>

</servlet>

<servlet-mapping>

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

<!-- 所有以action结尾的请求进入springMVC -->

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

</servlet-mapping>

</web-app>

1. 使用逆向工程生成pojo和Mapper代码

7. 编写service和Controller实现数据库中的商品列表显示

**public** **interface** ItemService {

/\*\*

\* 查询所有的商品数据

\*

\* **@return**

\*/

**public** List<Item> queryItemList();

}

@Service

**public** **class** ItemServiceImpl **implements** ItemService {

@Autowired

**private** ItemMapper itemMapper;

@Override

**public** List<Item> queryItemList() {

List<Item> list = **this**.itemMapper.selectByExample(**null**);

**return** list;

}

}

@Controller

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

@Autowired

**private** ItemService itemService;

@RequestMapping("itemList")

**public** ModelAndView queryItemList() {

// 使用service查询商品数据

List<Item> itemList = **this**.itemService.queryItemList();

// 创建ModelAndView

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

// 设置数据到ModelAndView中

modelAndView.addObject("itemList", itemList);

// 设置逻辑视图

modelAndView.setViewName("itemList");

**return** modelAndView;

}

}

# 关卡2

## 训练案例1

### 训练描述

在关卡1的基础上实现SpringMVC的参数的接受

要求

1. 获取Request，从Request中获取请求参数，实现商品id的接收

2. 使用Model和返回String类型的方式向页面传递数据，实现商品编辑页面的数据回显

3. 使用直接声明形参的方式接收参数，并实现请求参数名是id，使用方法的形参名itemId接收

4. 使用pojo接受请求的商品数据，实现商品的修改

5. 解决post请求的乱码问题

6. 创建包装类，包含了商品属性，用包装pojo接受查询条件

7. 实现接收时间类型的数据，完成商品的生产日期修改

### 操作步骤答案

1. 获取Request，从Request中获取请求参数，实现商品id的接收

2. 使用Model和返回String类型的方式向页面传递数据，实现商品编辑页面的数据回显

编写Controller

**public** String queryItemById(Model model, HttpServletRequest request) {

// 从request中获取id

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

**int** id = Integer.*parseInt*(sid);

Item item = **this**.itemService.queryItemById(id);

model.addAttribute("item", item);

**return** "itemEdit";

}

编写Service

**public** **interface** ItemService {

/\*\*

\* 查询所有的商品数据

\* **@return**

\*/

**public** List<Item> queryItemList();

/\*\*

\* 根据商品id查询商品

\* **@param** id

\* **@return**

\*/

**public** Item queryItemById(**int** id);

}

@Service

**public** **class** ItemServiceImpl **implements** ItemService {

@Autowired

**private** ItemMapper itemMapper;

@Override

**public** List<Item> queryItemList() {

List<Item> list = **this**.itemMapper.selectByExample(**null**);

**return** list;

}

@Override

**public** Item queryItemById(**int** id) {

Item item = **this**.itemMapper.selectByPrimaryKey(id);

**return** item;

}

}

3. 使用直接声明形参的方式接收参数，并实现请求参数名是id，使用方法的形参名itemId接收

**public** String queryItemById(Model model, HttpServletRequest request, @RequestParam("id") Integer itemId) {

// 从request中获取id

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

**int** id = Integer.*parseInt*(sid);

Item item = **this**.itemService.queryItemById(itemId);

model.addAttribute("item", item);

**return** "itemEdit";

}

4. 使用pojo接受请求的商品数据，实现商品的修改

编写Controller

// 使用pojo接受请求的商品数据，实现商品的修改

@RequestMapping("updateItem")

**public** String updateItem(Item item) {

**this**.itemService.updateItem(item);

**return** "success";

}

编写Service

**public** **interface** ItemService {

/\*\*

\* 查询所有的商品数据

\*

\* **@return**

\*/

**public** List<Item> queryItemList();

/\*\*

\* 根据商品id查询商品

\*

\* **@param** id

\* **@return**

\*/

**public** Item queryItemById(**int** id);

/\*\*

\* 更新商品

\*

\* **@param** item

\*/

**public** **void** updateItem(Item item);

}

@Service

**public** **class** ItemServiceImpl **implements** ItemService {

@Autowired

**private** ItemMapper itemMapper;

@Override

**public** List<Item> queryItemList() {

List<Item> list = **this**.itemMapper.selectByExample(**null**);

**return** list;

}

@Override

**public** Item queryItemById(**int** id) {

Item item = **this**.itemMapper.selectByPrimaryKey(id);

**return** item;

}

@Override

**public** **void** updateItem(Item item) {

**this**.itemMapper.updateByPrimaryKeySelective(item);

}

}

5. 解决post请求的乱码问题

在web.xml添加如下配置：

<!-- 解决post乱码问题 -->

<filter>

<filter-name>encoding</filter-name>

<filter-class>org.springframework.web.filter.CharacterEncodingFilter</filter-class>

<!-- 设置编码参是UTF8 -->

<init-param>

<param-name>encoding</param-name>

<param-value>UTF-8</param-value>

</init-param>

</filter>

<filter-mapping>

<filter-name>encoding</filter-name>

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

</filter-mapping>

6. 创建包装类，包含了商品属性，用包装pojo接受查询条件，搜索功能在商品列表页

创建包装类

**public** **class** QueryVo {

**private** Item item;

**public** Item getItem() {

**return** item;

}

**public** **void** setItem(Item item) {

**this**.item = item;

}

}

编写Controller

// 创建包装类，包含了商品属性，用包装pojo接受查询条件，搜索功能在商品列表页

@RequestMapping("queryItem")

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

System.***out***.println(queryVo.getItem().getId());

System.***out***.println(queryVo.getItem().getName());

**return** "success";

}

7. 实现接收时间类型的数据，完成商品的生产日期修改

修改itemEdit.jsp，显示商品的时间

<tr>

<td>商品生产日期</td>

<td><input type=*"text"* name=*"createtime"*

value=*"*<fmt:formatDate value="${item.createtime}" pattern=*"yyyy-MM-dd HH:mm:ss"*/>*"* /></td>

</tr>

编写转换器

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

@Override

**public** Date convert(String source) {

SimpleDateFormat df = **new** SimpleDateFormat("yyyy-MM-dd HH:mm:ss");

Date date = **null**;

**try** {

date = df.parse(source);

} **catch** (ParseException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

**return** date;

}

}

在Springmvc.xml配置转换器

<!-- 配置注解驱动 处理器映射器和处理器适配器 -->

<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=*" com.igeek.ssm.converter.DateConverter"* />

</set>

</property>

</bean>

# 关卡3

## 训练案例1

### 训练描述

执行以下sql语句修改课上提供的数据库表，给item表增加userId字段

ALTER TABLE item add userId int(11) DEFAULT NULL COMMENT '维护人id';

给商品设置维护人id

UPDATE item SET userId = 1 WHERE id=1;

UPDATE item SET userId = 10 WHERE id=2;

UPDATE item SET userId = 10 WHERE id=3;

实现维护人基本功能

要求

1. 参考itemList.jsp和itemEdit.jsp编写维护人的jsp

2. 实现显示维护人列表

3. 实现根据维护人姓名和商品名查询维护人数据

4. 实现维护人修改功能，能够修改维护人的生日