

SpringMvc

1. SpringMVC的简介

1.1 SpringMVC概述

1.2 SpringMvc处理流程

1.3 SpringMVC入门程序

 1.3.1 新建工程

 1.3.2 导包

 1.3.3 拷贝jsp和静态资源

 1.3.4 拷贝pojo和导入sql

 1.3.5 编写controller

 1.3.6 springmvc.xml

 1.3.7 配置web.xml

 1.3.8 springmvc基本流程

2.SpringMvc架构

2.1 SpringMVC访问流程

2.2 SpringMVC组件解析

2.3 默认加载组件

2.4 配置扫描controller

2.5 配置处理器映射器和处理器适配器

2.6 配置视图解析器

2.7 处理不拦截静态资源

3.SSM整合

3.1 整合思路

3.2 配置dao

3.3 配置service

3.4 配置监听器

3.5 配置controller

3.6 mybatis注解

3.7 整合测试

3.7.1 编写service

3.7.2 编写controller

3.7.3 拷贝资源

4.参数绑定

4.1 默认的参数绑定

4.2 绑定简单类型

4.3 RequestParam注解(重点)

4.4 Model/ModelMap

4.5 绑定pojo类型

4.6 编写日期转换器

4.7 解决乱码问题

4.8 绑定包装的javabean类型 (了解)

4.9 绑定集合或者数组

5.RequestMapping

5.1 URL映射路径

5.2 作用在类上面

5.3 限制请求方式

6.Controller方法的返回

6.1 返回 ModelAndView

6.2 返回 void

6.3 返回 String 类型

6.3.1 逻辑视图的名字

6.3.2 转发到其他处理器

6.3.3 重定向

7.异常处理器

7.1 springmvc 异常处理

7.2 预期异常(自定义异常)

7.3 编写自定义异常类

7.4 编写异常处理器

7.5 测试异常处理

8.图片上传

8.1 nginx

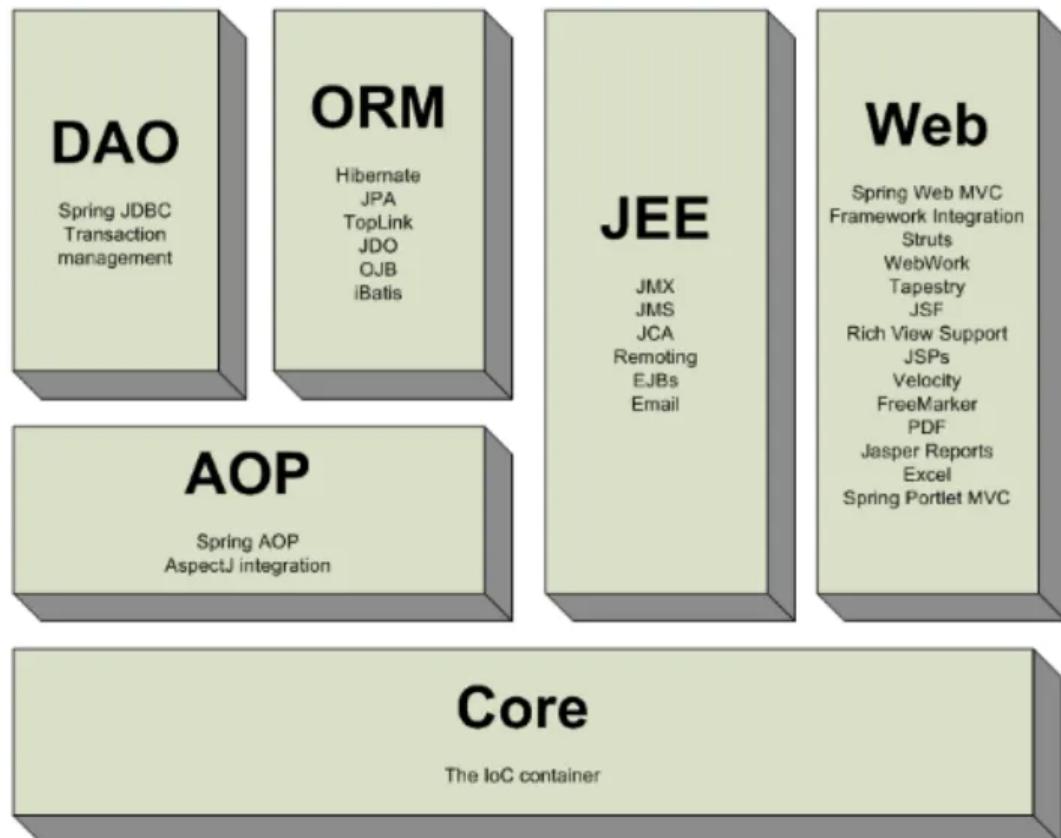
- [8.1.1 nginx是什么](#)
 - [8.1.2 nginx做HTTP服务器](#)
 - [8.1.3 通过端口来区分虚拟主机](#)
 - [8.1.4 nginx可以通过域名来区分](#)
 - [8.1.5 dns解析](#)
 - [8.1.6 搭建图片服务器](#)
 - [8.2 图片上传](#)
- [9.Json数据的交互](#)
- [9.1 ResponseBody注解](#)
 - [9.2 ResponseEntity](#)
 - [9.3 postman使用](#)
 - [9.4 RequestBody注解](#)

1. SpringMVC的简介

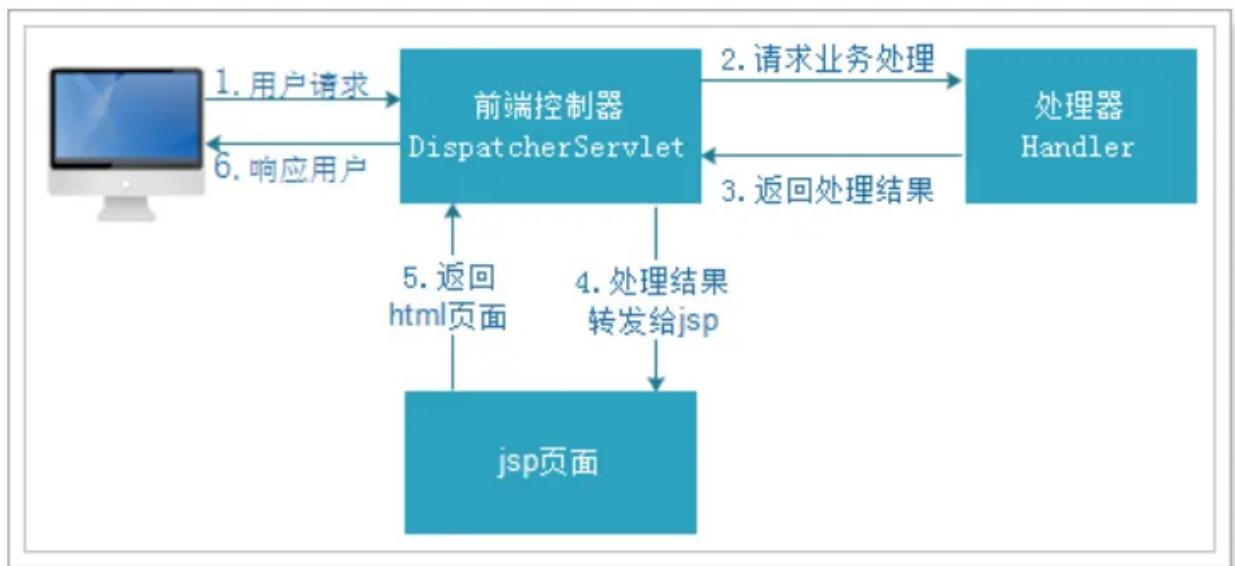
1.1 SpringMVC概述

SpringMVC 是一种基于 Java 的实现 MVC 设计模型的请求驱动类型的轻量级 Web 框架，属于 SpringFrameWork 的后续产品，已经融合在 Spring Web Flow 中。

SpringMVC 已经成为目前最主流的MVC框架之一，并且随着Spring3.0 的发布，全面超越 Struts2，成为最优秀的 MVC 框架。它通过一套注解，让一个简单的 Java 类成为处理请求的控制器，而无须实现任何接口。同时它还支持 RESTful 编程风格的请求。



1.2 SpringMvc处理流程



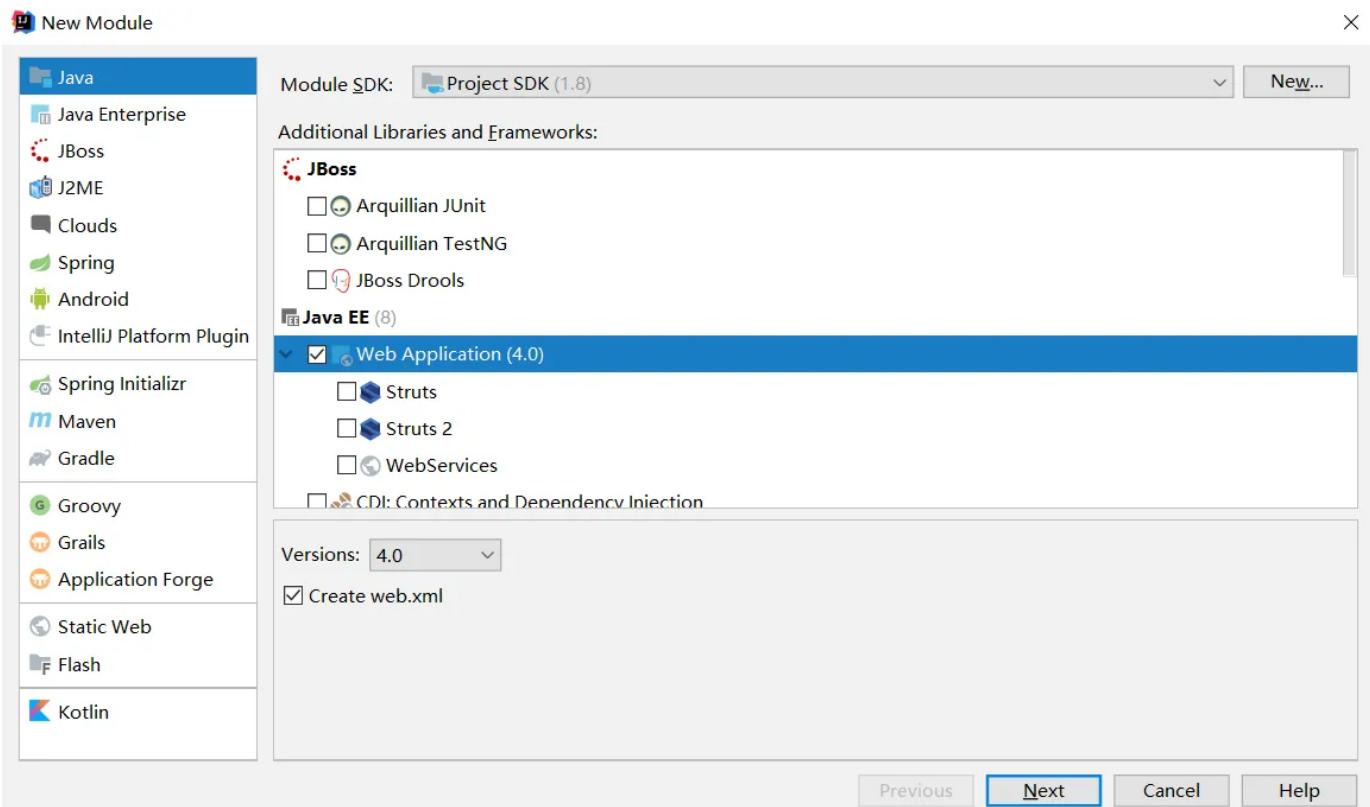
1.3 SpringMVC入门程序

需求：客户端发起请求，服务器端接收请求，执行逻辑并进行视图跳转。

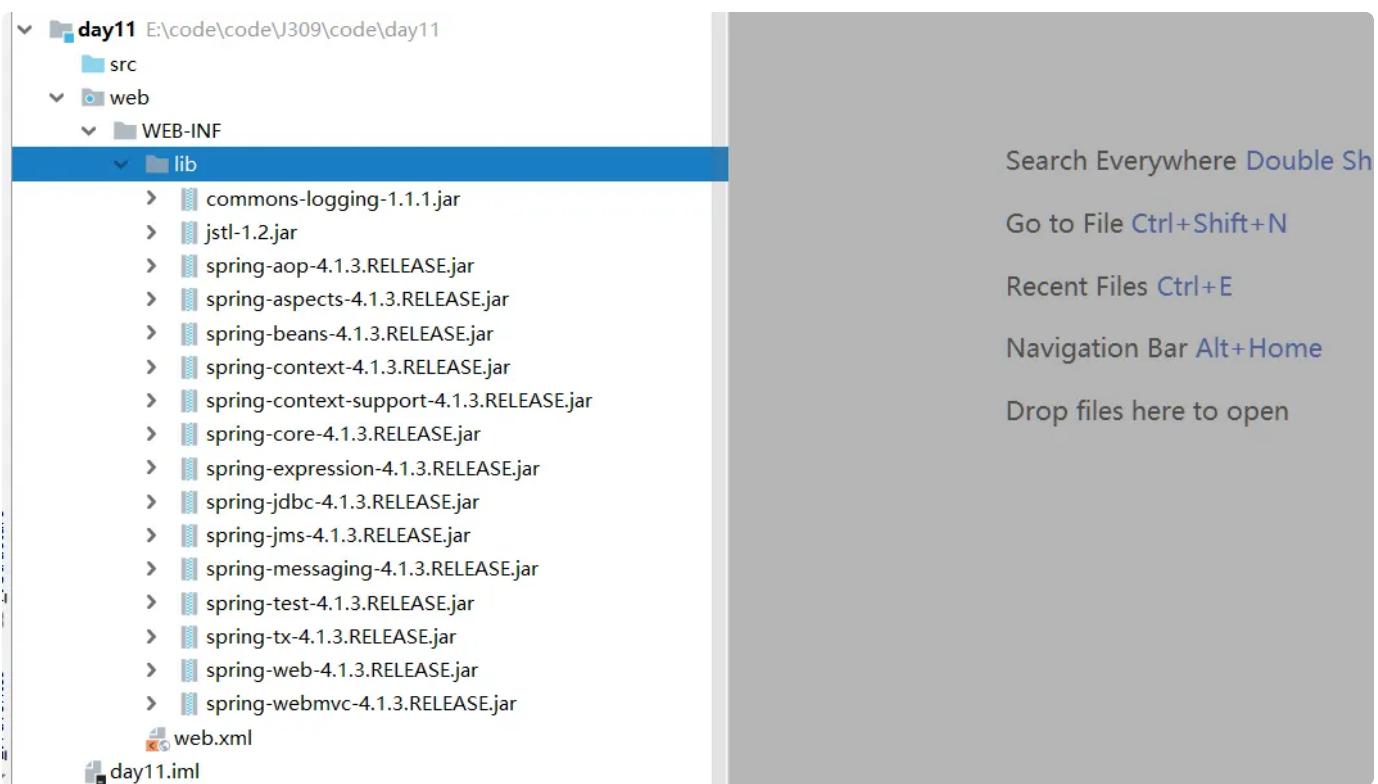
开发步骤

- ①导入SpringMVC相关坐标
- ②配置SpringMVC核心控制器DispatcherServlet
- ③创建Controller类和视图页面
- ④使用注解配置Controller类中业务方法的映射地址
- ⑤配置SpringMVC核心文件 spring-mvc.xml
- ⑥客户端发起请求测试

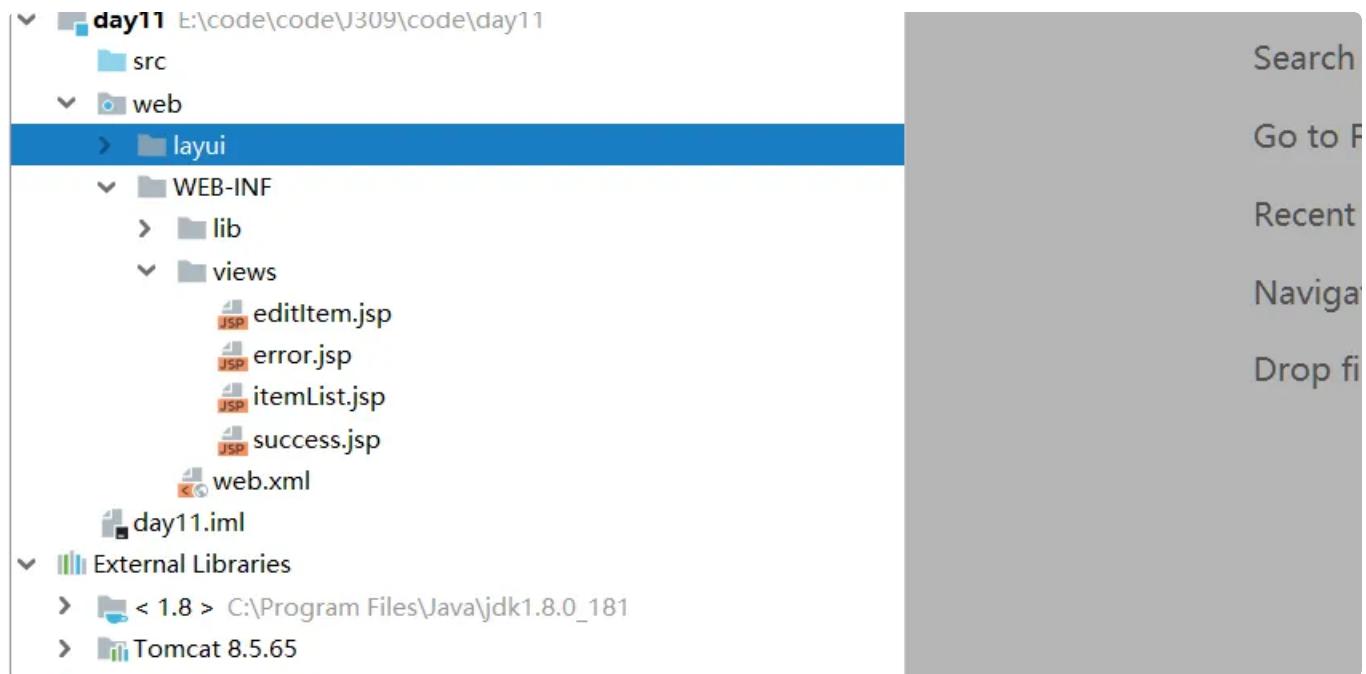
1.3.1 新建工程



1.3.2 导包



1.3.3 拷贝jsp和静态资源



1.3.4 拷贝pojo和导入sql

```
1  public class Item implements Serializable {
2
3      private Integer id;
4
5      private String name;
6
7      private Float price;
8
9      private String pic;
10
11     private Date createtime;
12
13     private String detail;
14
15    public Item(){
16
17    }
18
19    public Item(Integer id, String name, Float price, String pic, Date cre
20        atetime, String detail) {
21        this.id = id;
22        this.name = name;
23        this.price = price;
24        this.pic = pic;
25        this.createtime = createtime;
26        this.detail = detail;
27    }
28
29    public Integer getId() {
30        return id;
31    }
32
33    public void setId(Integer id) {
34        this.id = id;
35    }
36
37    public String getName() {
38        return name;
39    }
40
41    public void setName(String name) {
42        this.name = name == null ? null : name.trim();
43    }
44
45    public Float getPrice() {
```

```
45         return price;
46     }
47
48     public void setPrice(Float price) {
49         this.price = price;
50     }
51
52     public String getPic() {
53         return pic;
54     }
55
56     public void setPic(String pic) {
57         this.pic = pic == null ? null : pic.trim();
58     }
59
60     public Date getCreatetime() {
61         return createtime;
62     }
63
64     public void setCreatetime(Date createtime) {
65         this.createtime = createtime;
66     }
67
68     public String getDetail() {
69         return detail;
70     }
71
72     public void setDetail(String detail) {
73         this.detail = detail == null ? null : detail.trim();
74     }
75 }
76 }
```



1.3.5 编写controller

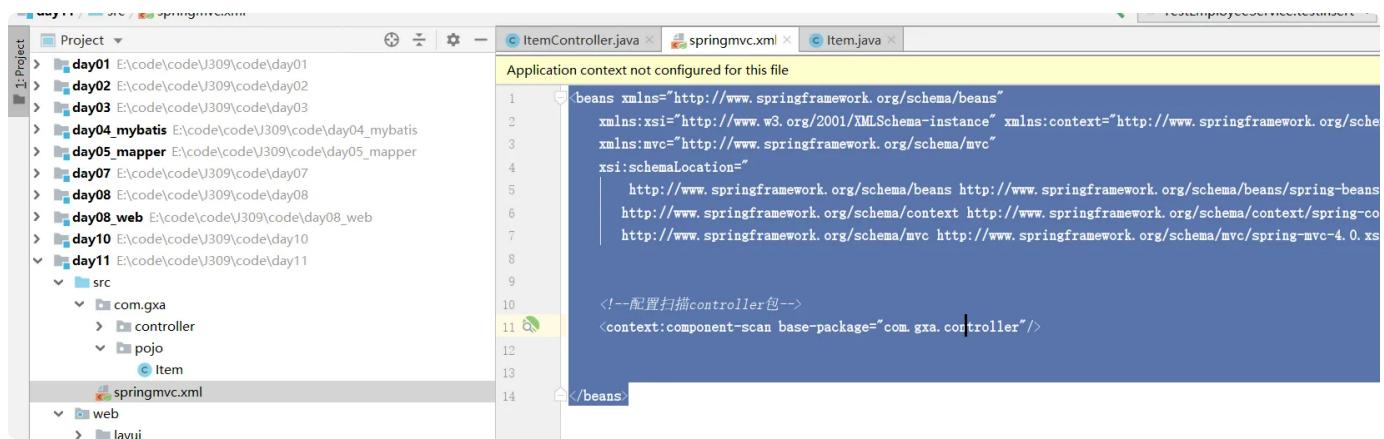
```
1  /**
2   * Created by zxd on 2022/10/27 9:47
3   */
4  @Controller
5  public class ItemController {
6
7
8
9      //配置方法的映射路径 访问路径
10     @RequestMapping("/queryItemList")
11     public ModelAndView queryItemList(){
12
13         //1.查询商品列表
14         List list=new ArrayList();
15         list.add(new Item(1,"华为mate50",8000F,"xxx",new Date(),"xxx"));
16         list.add(new Item(2,"华为mate60",8000F,"xxx",new Date(),"xxx"));
17         list.add(new Item(3,"华为mate80",8000F,"xxx",new Date(),"xxx"));
18
19
20         //2.放入数据到request域
21         ModelAndView modelAndView=new ModelAndView();
22         modelAndView.addObject("list",list);
23
24         //3.转发到jsp
25         modelAndView.setViewName("/WEB-INF/views/itemList.jsp");
26
27         return modelAndView;
28     }
29
30
31 }
```

1.3.6 springmvc.xml

```

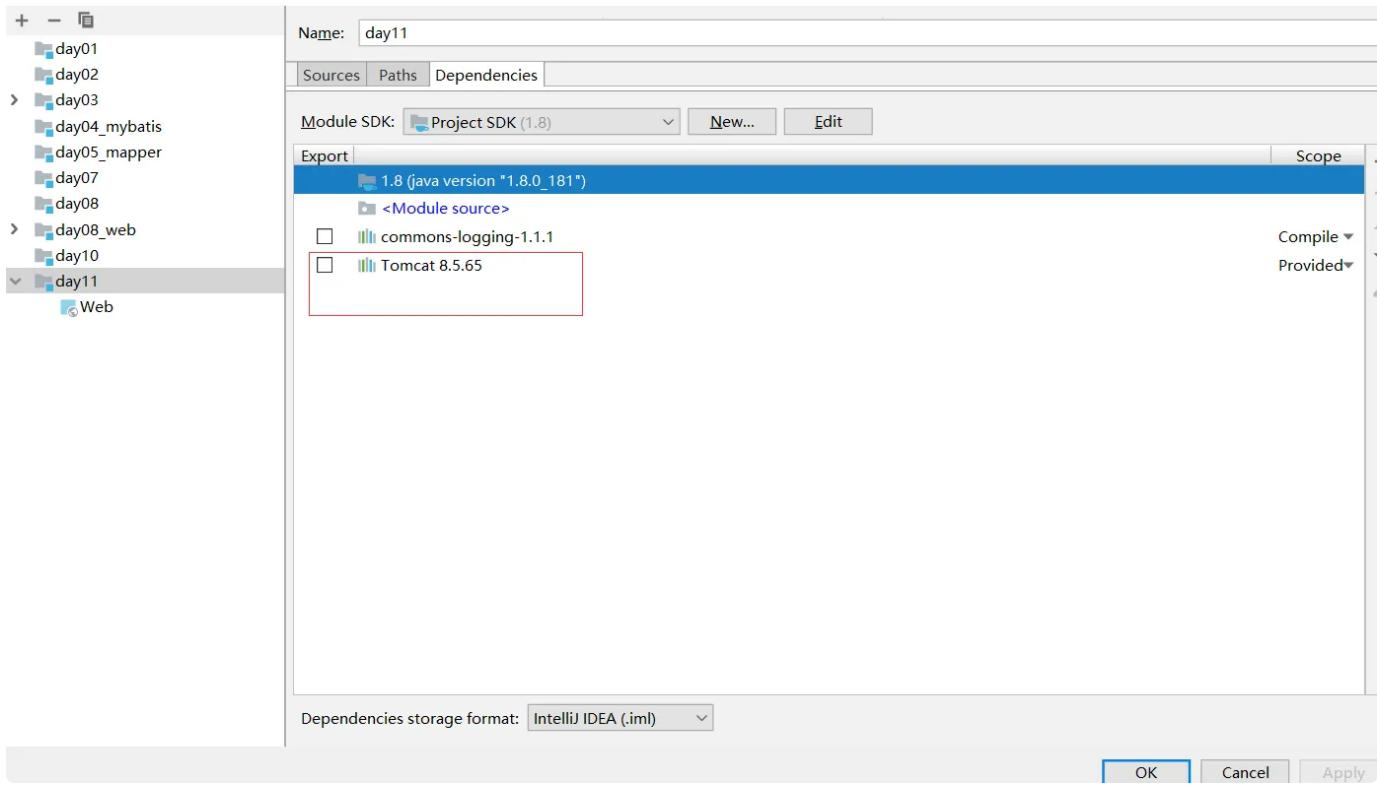
1 <beans xmlns="http://www.springframework.org/schema/beans"
2   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:context="http://www.springframework.org/schema/context"
3   xmlns:mvc="http://www.springframework.org/schema/mvc"
4   xsi:schemaLocation="
5     http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-4.0.xsd
6     http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-4.0.xsd
7     http://www.springframework.org/schema/mvc http://www.springframework.org/schema/mvc/spring-mvc-4.0.xsd">
8
9
10    <!--配置扫描controller包-->
11    <context:component-scan base-package="com.gxa.controller"/>
12
13
14  </beans>

```



1.3.7 配置web.xml

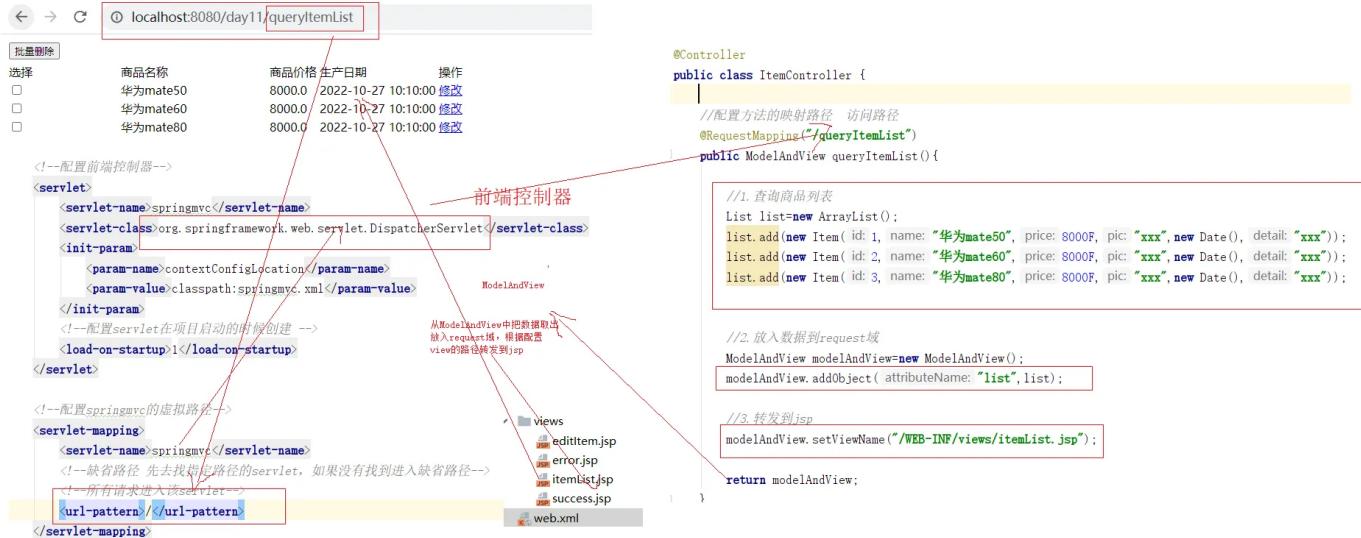
```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
3      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4      xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/web-app_4_0.xsd"
5          version="4.0">
6
7
8      <!--配置前端控制器-->
9      <servlet>
10         <servlet-name>springmvc</servlet-name>
11         <servlet-class>org.springframework.web.servlet.DispatcherServlet</
12             servlet-class>
13             <init-param>
14                 <param-name>contextConfigLocation</param-name>
15                 <param-value>classpath:springmvc.xml</param-value>
16             </init-param>
17             <!--配置servlet在项目启动的时候创建 -->
18             <load-on-startup>1</load-on-startup>
19         </servlet>
20
21         <!--配置springmvc的虚拟路径-->
22         <servlet-mapping>
23             <servlet-name>springmvc</servlet-name>
24             <!--缺省路径 先去找指定路径的servlet, 如果没有找到进入缺省路径-->
25             <!--所有请求进入该servlet-->
26             <url-pattern>/</url-pattern>
27         </servlet-mapping>
28
29
30     </web-app>
```



← → ⌂ ⓘ localhost:8080/day11/queryItemList

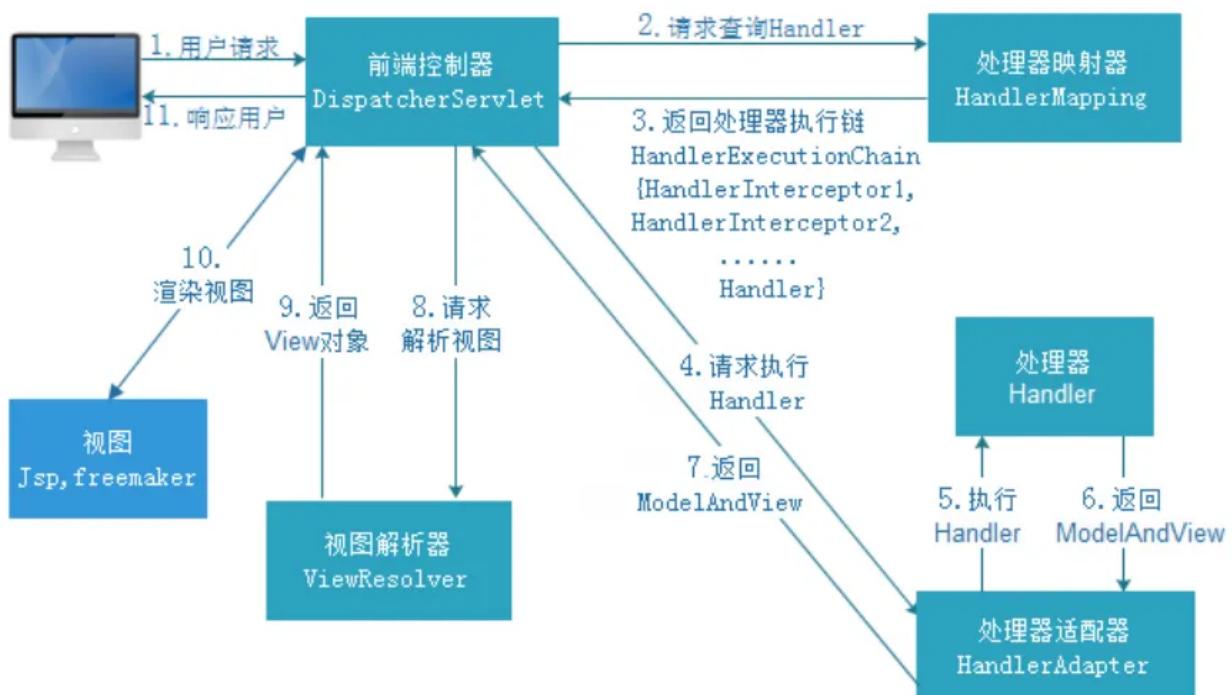
批量删除				
选择	商品名称	商品价格	生产日期	操作
<input type="checkbox"/>	华为mate50	8000.0	2022-10-27 10:10:00	修改
<input type="checkbox"/>	华为mate60	8000.0	2022-10-27 10:10:00	修改
<input type="checkbox"/>	华为mate80	8000.0	2022-10-27 10:10:00	修改

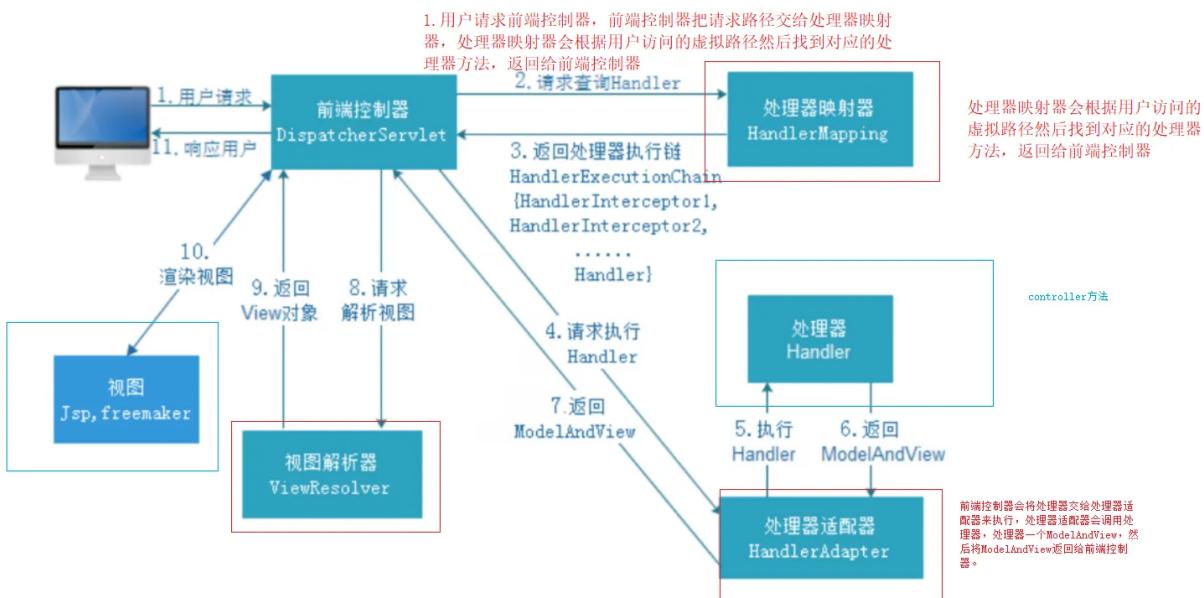
1.3.8 springmvc基本流程



2. SpringMvc架构

2.1 SpringMVC访问流程





①用户发送请求至前端控制器DispatcherServlet。

②DispatcherServlet收到请求调用HandlerMapping处理器映射器。

③处理器映射器找到具体的处理器(可以根据xml配置、注解进行查找)，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。

④DispatcherServlet调用HandlerAdapter处理器适配器。

⑤HandlerAdapter经过适配调用具体的处理器(Controller，也叫后端控制器)。

⑥Controller执行完成返回ModelAndView。

⑦HandlerAdapter将controller执行结果 ModelAndView 返回给DispatcherServlet。

- ⑧DispatcherServlet将ModelAndView传给ViewReslover视图解析器。
- ⑨ViewReslover解析后返回具体View。
- ⑩DispatcherServlet根据View进行渲染视图（即将模型数据填充至视图中）。DispatcherServlet响应用户。

2.2 SpringMVC组件解析

1. 前端控制器：DispatcherServlet

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。

2. 处理器映射器：HandlerMapping

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。

3. 处理器适配器：HandlerAdapter

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。

4. 处理器：Handler

它就是我们开发中要编写的具体业务控制器。由 DispatcherServlet 把用户请求转发到 Handler。由 Handler 对具体的用户请求进行处理。

5. 视图解析器：View Resolver

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。

6. 视图：View

SpringMVC 框架提供了很多的 View 视图类型的支持，包括：jstlView、freemarkerView、pdfView 等。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

2.3 默认加载组件

我们刚才没有做任何的配置，就可以使用三大组件，springmvc做了默认的配置

```

1 # Default implementation classes for DispatcherServlet's strategy interfaces.
2 # Used as fallback when no matching beans are found in the DispatcherServlet context.
3 # Not meant to be customized by application developers.
4
5 org.springframework.web.servlet.LocaleResolver=org.springframework.web.servlet.i18n.AcceptHeaderLocaleResolver
6
7 org.springframework.web.servlet.ThemeResolver=org.springframework.web.servlet.theme.FixedThemeResolver
8 处理器适配器
9 org.springframework.web.servlet.HandlerMapping=org.springframework.web.handler.BeanNameUrlHandlerMapping,\ 
10 org.springframework.web.servlet.mvc.annotation.DefaultAnnotationHandlerMapping
11
12 org.springframework.web.servlet.HandlerAdapter=org.springframework.web.mvc.HttpRequestHandlerAdapter,\ 
13 org.springframework.web.servlet.mvc.SimpleControllerHandlerAdapter,\ 
14 org.springframework.web.mvc.annotation.AnnotationMethodHandlerAdapter
15
16 org.springframework.web.servlet.HandlerExceptionResolver=org.springframework.web.mvc.annotation.AnnotationMethodHandlerExceptionResolver
17 org.springframework.web.servlet.mvc.annotation.ResponseStatusExceptionResolver,\ 
18 org.springframework.web.servlet.mvc.support.DefaultHandlerExceptionResolver
19
20 视图解析器
21 org.springframework.web.servlet.RequestToViewNameTranslator=org.springframework.web.view.DefaultRequestToViewNameTranslator
22
23 org.springframework.web.servlet.ViewResolver=org.springframework.web.view.InternalResourceViewResolver
24
25 org.springframework.web.servlet.FlashMapManager=org.springframework.web.support.SessionFlashMapManager
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124

```

2.4 配置扫描controller

```

1 <beans xmlns="http://www.springframework.org/schema/beans"
2   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:context="http://www.springframework.org/schema/context"
3   xmlns:mvc="http://www.springframework.org/schema/mvc"
4   xsi:schemaLocation="
5       http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-4.0.xsd
6       http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-4.0.xsd
7       http://www.springframework.org/schema/mvc http://www.springframework.org/schema/mvc/spring-mvc-4.0.xsd">
8
9
10    <!--配置扫描controller包-->
11    <context:component-scan base-package="com.gxa.controller"/>
12
13
14  </beans>

```

2.5 配置处理器映射器和处理器适配器

Java

```
1 <beans xmlns="http://www.springframework.org/schema/beans"
2   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:context="h
tp://www.springframework.org/schema/context"
3   xmlns:mvc="http://www.springframework.org/schema/mvc"
4   xsi:schemaLocation="
5       http://www.springframework.org/schema/beans http://www.springfram
ework.org/schema/beans/spring-beans-4.0.xsd
6       http://www.springframework.org/schema/context http://www.springfram
ework.org/schema/context/spring-context-4.0.xsd
7       http://www.springframework.org/schema/mvc http://www.springframewor
k.org/schema/mvc/spring-mvc-4.0.xsd">
8
9
10  <!--配置扫描controller包-->
11  <context:component-scan base-package="com.gxa.controller"/>
12
13  <!--配置处理器映射器-->
14  <bean class="org.springframework.web.servlet.mvc.method.annotation.Req
uestMappingHandlerMapping"/>
15
16  <!--配置处理适配器-->
17  <bean class="org.springframework.web.servlet.mvc.method.annotation.Req
uestMappingHandlerAdapter"/>
18
19
20  </beans>
```

注解驱动

```

1 <beans xmlns="http://www.springframework.org/schema/beans"
2   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:context="h
3     ttp://www.springframework.org/schema/context"
4   xmlns:mvc="http://www.springframework.org/schema/mvc"
5   xsi:schemaLocation="
6     http://www.springframework.org/schema/beans http://www.springfram
7   work.org/schema/beans/spring-beans-4.0.xsd
8     http://www.springframework.org/schema/context http://www.springfram
9   work.org/schema/context/spring-context-4.0.xsd
10    http://www.springframework.org/schema/mvc http://www.springframewor
11   k.org/schema/mvc/spring-mvc-4.0.xsd">
12
13
14  <!--配置扫描controller包-->
15  <context:component-scan base-package="com.gxa.controller"/>
16
17  <!--注解驱动 自动配置注解版本的处理器映射器+适配器-->
18  <mvc:annotation-driven/>
19
20
21  </beans>

```

2.6 配置视图解析器

```

1
2
3  <!--配置视图解析器-->
4  <bean class="org.springframework.web.servlet.view.InternalResourceView
5    Resolver">
6
7    <!--定义逻辑视图的前缀-->
8    <property name="prefix" value="/WEB-INF/views/" />
9
10   <!--定义逻辑视图的后缀-->
11   <property name="suffix" value=".jsp" />
12
13  </bean>

```

```

@RequestMapping("/queryItemList")
public ModelAndView queryItemList() {
    //1. 查询商品列表
    List list=new ArrayList();
    list.add(new Item( id: 1, name: "华为mate50", price: 8000F, pic: "xxx", new Date(), detail: "xxx"));
    list.add(new Item( id: 2, name: "华为mate60", price: 8000F, pic: "xxx", new Date(), detail: "xxx"));
    list.add(new Item( id: 3, name: "华为mate80", price: 8000F, pic: "xxx", new Date(), detail: "xxx"));

    //2. 放入数据到request域
    ModelAndView modelAndView=new ModelAndView();
    modelAndView.addObject( attributeName: "list", list);

    //3. 转发到jsp
    modelAndView.setViewName("itemList");

    return modelAndView;
}

```

!--配置视图解析器-->

```

<bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
    <!--定义逻辑视图的前缀-->
    <property name="prefix" value="/WEB-INF/views/" />
    <!--定义逻辑视图的后缀-->
    <property name="suffix" value=".jsp" />
</bean>

```

2.7 处理不拦截静态资源

思考为什么没有样式？



```
▼ web.xml Java |  
1 <!--配置springmvc的虚拟路径-->  
2 <servlet-mapping>  
3   <servlet-name>springmvc</servlet-name>  
4   <!--缺省路径 先去找指定路径的servlet，如果没有找到进入缺省路径-->  
5   <!--所有请求进入该servlet-->  
6   <url-pattern>/</url-pattern>  
7 </servlet-mapping>
```

思路:

- 1.静态资源的请求不进入springmvc
- 2.进入了springmvc，但是我告诉springmvc他是一个静态资源， springmvc不要处理。

```

1 <beans xmlns="http://www.springframework.org/schema/beans"
2   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:context="h
3     ttp://www.springframework.org/schema/context"
4   xmlns:mvc="http://www.springframework.org/schema/mvc"
5   xsi:schemaLocation="
6       http://www.springframework.org/schema/beans http://www.springfram
7   work.org/schema/beans/spring-beans-4.0.xsd
8       http://www.springframework.org/schema/context http://www.springfram
9   ework.org/schema/context/spring-context-4.0.xsd
10      http://www.springframework.org/schema/mvc http://www.springframewor
11   k.org/schema/mvc/spring-mvc-4.0.xsd">
12
13    <!--配置不拦截静态资源-->
14    <mvc:resources mapping="/layui/**" location="/layui/" />
15
16    <!--配置扫描controller包-->
17    <context:component-scan base-package="com.gxa.controller"/>
18
19    <!--注解驱动 自动配置注解版本的处理器映射器+适配器-->
20    <mvc:annotation-driven/>
21
22    <!--配置视图解析器-->
23    <bean class="org.springframework.web.servlet.view.InternalResourceView
24   Resolver">
25
26      <!--定义逻辑视图的前缀-->
27      <property name="prefix" value="/WEB-INF/views/" />
28
29      <!--定义逻辑视图的后缀-->
30      <property name="suffix" value=".jsp" />
31
32    </bean>
33
34  </beans>

```

批量删除				
选择	商品名称	商品价格	生产日期	操作
<input type="checkbox"/>	华为mate50	8000.0	2022-10-27 11:43:14	修改
<input type="checkbox"/>	华为mate60	8000.0	2022-10-27 11:43:14	修改
<input type="checkbox"/>	华为mate80	8000.0	2022-10-27 11:43:14	修改

3.SSM整合

spring+springmvc+mybatis

整合包



spring+mybatis

hibernate jpa

3.1 整合思路

dao层

1.SqlMapConfig.xml mybatis配置文件 废弃

applicationContext-dao.xml

1.配置数据库连接池

2.SqlSessionFactory 交给spring管理 FactoryBean mybatis-spring整合包

3.配置扫描mapper接口

service

applicationContext-service.xml 扫描service包

applicationContext-tx.xml 配置事务管理

applicationContext.xml

controller层

springmvc.xml

1.配置扫描controller包

2.配置注解驱动

3.配置视图解析器

web.xml

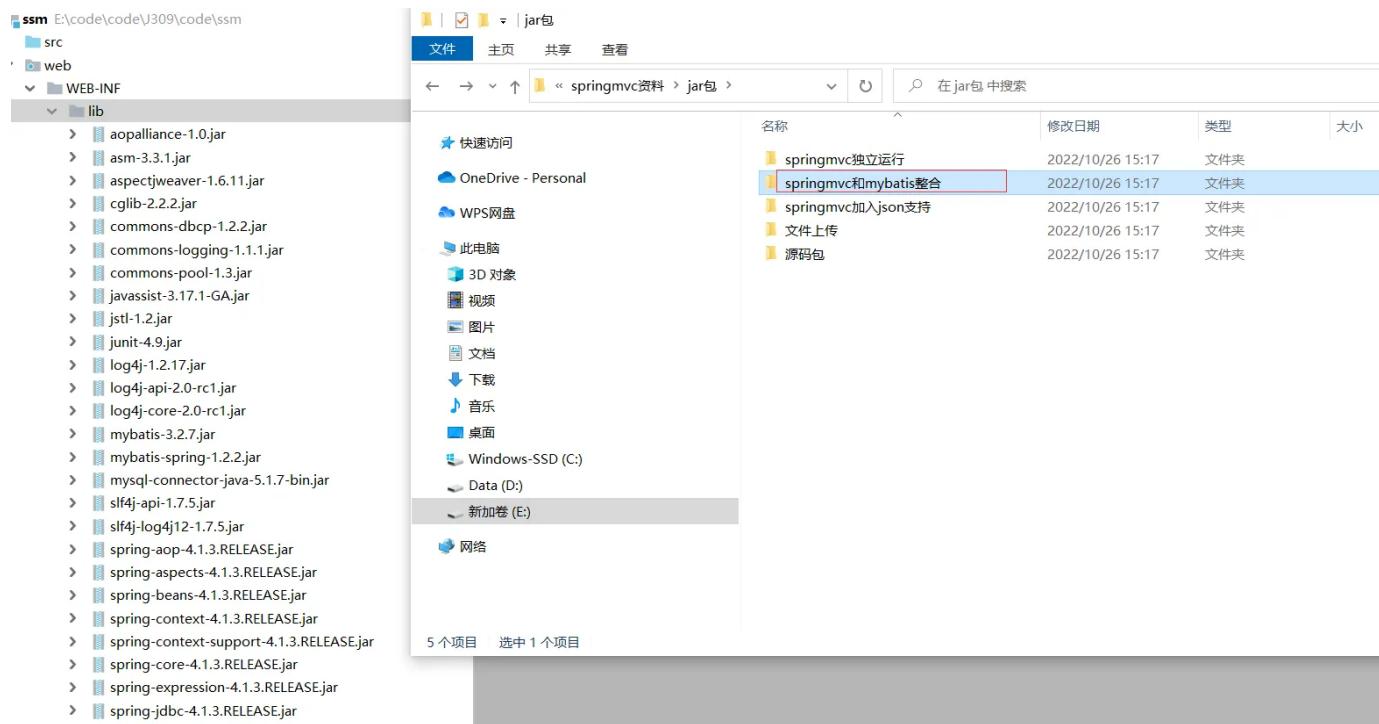
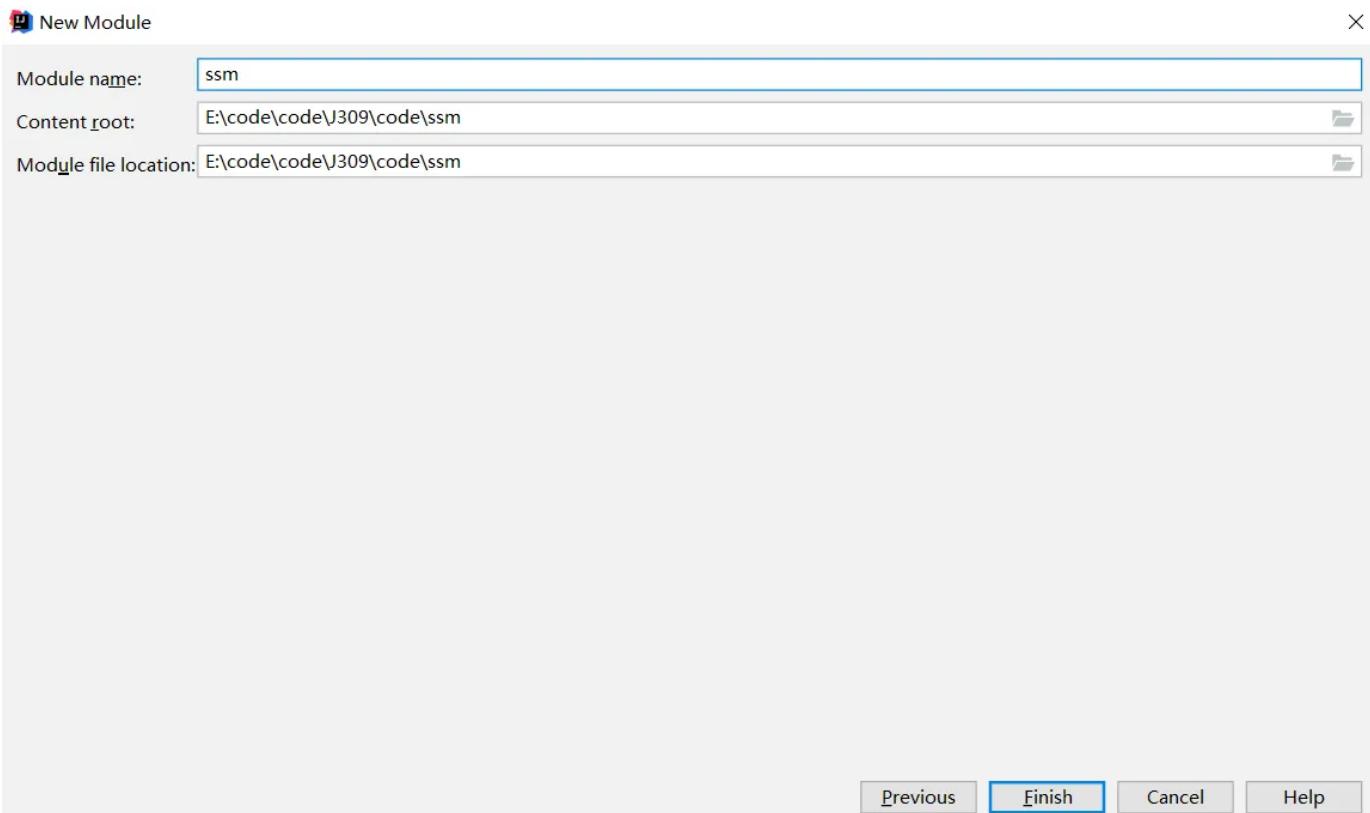
配置前端控制器

配置spring的监听器

spring父子容器

3.2 配置dao

新建工程导入jar包



applicationContext-dao.xml

1.配置数据库连接池

2.SqlSessionFactory 交给spring管理 FactoryBean mybatis-spring整合包

3.配置扫描mapper接口

```
1 <beans xmlns="http://www.springframework.org/schema/beans"
2     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3     xmlns:context="http://www.springframework.org/schema/context"
4     xsi:schemaLocation="
5         http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-4.0.xsd
6         http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-4.0.xsd
7     ">
8
9
10    <!--1.加载外部的properties文件-->
11    <context:property-placeholder location="classpath:jdbc.properties"/>
12
13    <!--2.配置数据库连接池-->
14    <bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource">
15        <property name="driverClassName" value="${jdbc.driverClassName}"><
16 /property>
17        <property name="url" value="${jdbc.url}"></property>
18        <property name="username" value="${jdbc.username}"></property>
19        <property name="password" value="${jdbc.password}"></property>
20        <property name="minIdle" value="${jdbc.minIdle}"></property>
21        <property name="maxActive" value="${jdbc.maxActive}"></property>
22    </bean>
23
24    <!--3.配置SqlSessionFactoryBean-->
25    <bean class="org.mybatis.spring.SqlSessionFactoryBean">
26        <!--注入连接池-->
27        <property name="dataSource" ref="dataSource"/>
28        <!--配置别名-->
29        <property name="typeAliasesPackage" value="com.gxa.pojo"/>
30    </bean>
31
32    <!--配置扫描mapper接口-->
33    <bean class="org.mybatis.spring.mapper.MapperScannerConfigurer">
34        <property name="basePackage" value="com.gxa.mapper"/>
35    </bean>
36
37
38    </beans>
```

3.3 配置service

applicationContext-service.xml 扫描service包

```
▼ applicationContext-service.xml Java |  
1 <beans xmlns="http://www.springframework.org/schema/beans"  
2   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
3   xmlns:context="http://www.springframework.org/schema/context"  
4   xsi:schemaLocation="  
5     http://www.springframework.org/schema/beans http://www.springframe  
work.org/schema/beans/spring-beans-4.0.xsd  
6     http://www.springframework.org/schema/context http://www.springfram  
ework.org/schema/context/spring-context-4.0.xsd  
7   ">  
8  
9   <!--配置扫描service包-->  
10  <context:component-scan base-package="com.gxa.service"/>  
11  
12 </beans>
```

applicationContext-tx.xml 配置事务管理

```

1 <beans xmlns="http://www.springframework.org/schema/beans"
2   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3   xmlns:context="http://www.springframework.org/schema/context"
4   xmlns:tx="http://www.springframework.org/schema/tx"
5   xmlns:aop="http://www.springframework.org/schema/aop"
6   xsi:schemaLocation="
7       http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-4.0.xsd
8       http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-4.0.xsd
9       http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-4.0.xsd
10      http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop-4.0.xsd">
11
12
13    <!--1.配置事务管理器-->
14    <bean id="transactionManager" class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
15      <property name="dataSource" ref="dataSource"/>
16    </bean>
17
18
19    <!--2.配置事务的通知-->
20    <tx:advice id="txAdvice" transaction-manager="transactionManager">
21      <tx:attributes>
22        <tx:method name="insert*" propagation="REQUIRED" isolation="REPEATABLE_READ" read-only="false"/>
23        <tx:method name="update*" propagation="REQUIRED" isolation="REPEATABLE_READ" read-only="false"/>
24        <tx:method name="delete*" propagation="REQUIRED" isolation="REPEATABLE_READ" read-only="false"/>
25        <tx:method name="find*" propagation="REQUIRED" isolation="REPEATABLE_READ" read-only="true"/>
26        <tx:method name="query*" propagation="REQUIRED" isolation="REPEATABLE_READ" read-only="true"/>
27      </tx:attributes>
28    </tx:advice>
29
30
31    <!--3.配置织入-->
32    <aop:config>
33
34      <!--切面-->
35

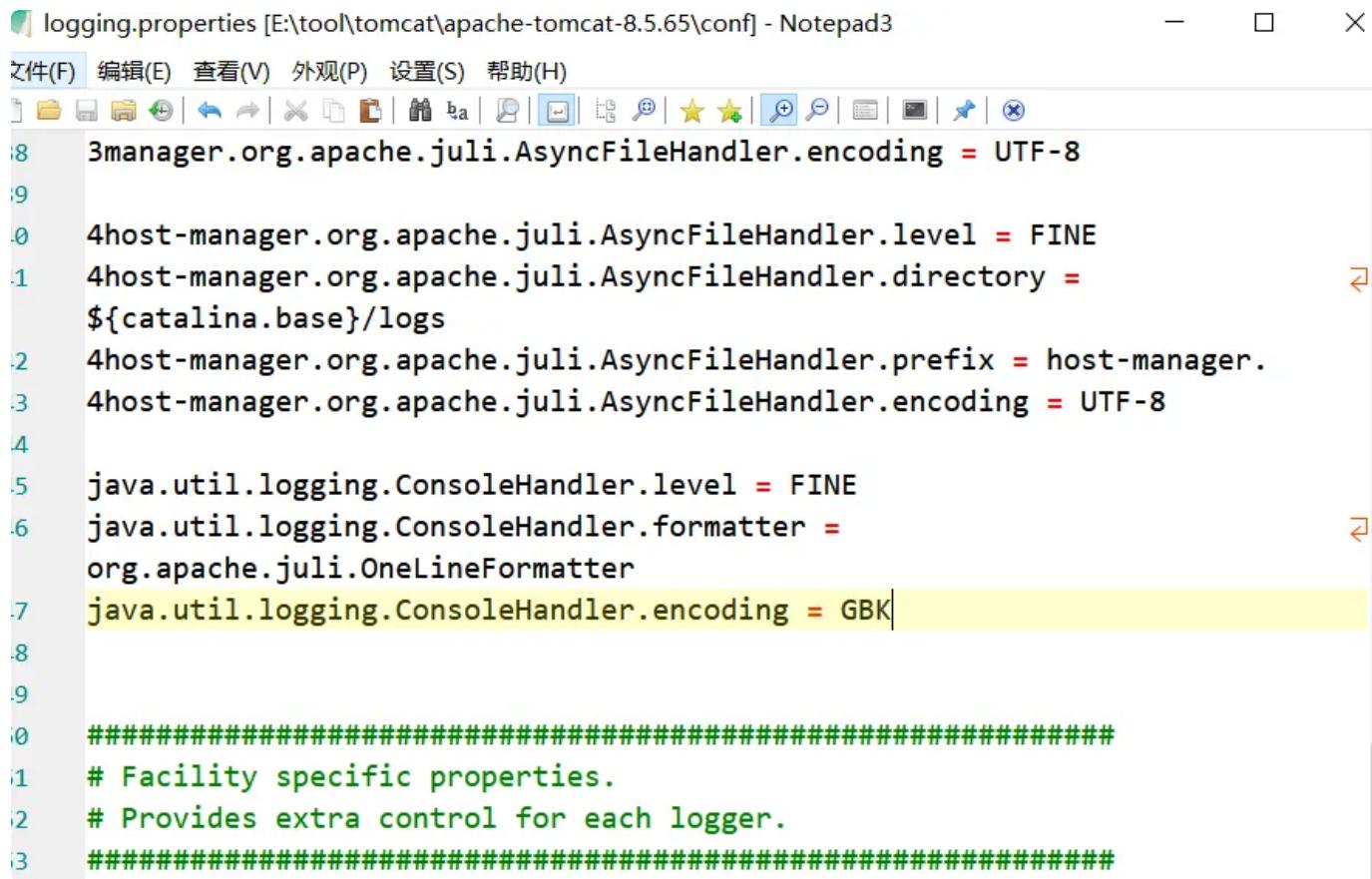
```

```
36         <aop:advisor advice-ref="txAdvice" pointcut="execution(* com.gxa.s  
37 ervice..*ServiceImpl.*(..))"/>  
38     </aop:config>  
39  
40  
41  
    </beans>
```

```
▼ applicationContext.xml Java |  
1  <beans xmlns="http://www.springframework.org/schema/beans"  
2      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
3      xmlns:context="http://www.springframework.org/schema/context"  
4      xsi:schemaLocation="  
5          http://www.springframework.org/schema/beans http://www.springfram  
ework.org/schema/beans/spring-beans-4.0.xsd  
6          http://www.springframework.org/schema/context http://www.springfram  
ework.org/schema/context/spring-context-4.0.xsd  
7      ">  
8  
9  
10     <import resource="applicationContext-dao.xml"/>  
11     <import resource="applicationContext-service.xml"/>  
12     <import resource="applicationContext-tx.xml"/>  
13  
14  
15   </beans>
```

3.4 配置监听器

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
3         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4         xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xml
ns.jcp.org/xml/ns/javaee/web-app_4_0.xsd"
5             version="4.0">
6
7
8     <!--配置监听器-->
9     <listener>
10        <listener-class>org.springframework.web.context.ContextLoaderListe
ner</listener-class>
11    </listener>
12
13    <!--配置spring初始化文件的位置-->
14    <context-param>
15        <param-name>contextConfigLocation</param-name>
16        <param-value>classpath:spring/applicationContext.xml</param-value>
17    </context-param>
18
19
20 </web-app>
```



The screenshot shows the Notepad3 application window with the file "logging.properties" open. The file contains Java properties configurations for logging levels, file handlers, and console handlers. A specific line, "java.util.logging.ConsoleHandler.encoding = GBK", is highlighted in yellow.

```
logging.properties [E:\tool\tomcat\apache-tomcat-8.5.65\conf] - Notepad3
文件(F) 编辑(E) 查看(V) 外观(P) 设置(S) 帮助(H)
18 manager.org.apache.juli.AsyncFileHandler.encoding = UTF-8
19
20 host-manager.org.apache.juli.AsyncFileHandler.level = FINE
21 host-manager.org.apache.juli.AsyncFileHandler.directory =
22 ${catalina.base}/logs
23 host-manager.org.apache.juli.AsyncFileHandler.prefix = host-manager.
24 host-manager.org.apache.juli.AsyncFileHandler.encoding = UTF-8
25
26 java.util.logging.ConsoleHandler.level = FINE
27 java.util.logging.ConsoleHandler.formatter =
28 org.apache.juli.OneLineFormatter
29 java.util.logging.ConsoleHandler.encoding = GBK
30
31
32 #####
33 # Facility specific properties.
34 # Provides extra control for each logger.
35 #####
36
```

3.5 配置controller

controller层

springmvc.xml

1.配置扫描controller包

2.配置注解驱动

3.配置视图解析器

springmvc.xml

Java

```
1 <beans xmlns="http://www.springframework.org/schema/beans"
2   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:context="h
3     ttp://www.springframework.org/schema/context"
4   xmlns:mvc="http://www.springframework.org/schema/mvc"
5   xsi:schemaLocation="
6       http://www.springframework.org/schema/beans http://www.springframe
7   work.org/schema/beans/spring-beans-4.0.xsd
8       http://www.springframework.org/schema/context http://www.springfram
9   ework.org/schema/context/spring-context-4.0.xsd
10      http://www.springframework.org/schema/mvc http://www.springframewor
11   k.org/schema/mvc/spring-mvc-4.0.xsd">
12
13    <!--配置扫描controller包-->
14    <context:component-scan base-package="com.gxa.controller"/>
15
16    <!--配置注解驱动-->
17    <mvc:annotation-driven/>
18
19    <!--配置视图解析器-->
20    <bean class="org.springframework.web.servlet.view.InternalResource
21   ViewResolver">
22      <property name="prefix" value="/WEB-INF/views/" />
23      <property name="suffix" value=".jsp" />
24    </bean>
25
26  </beans>
```

▼ web.xml

Java |

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
3      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4      xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/web-app_4_0.xsd"
5          version="4.0">
6
7
8      <!--配置监听器-->
9      <listener>
10         <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
11     </listener>
12
13     <!--配置spring初始化文件的位置-->
14     <context-param>
15         <param-name>contextConfigLocation</param-name>
16         <param-value>classpath:spring/applicationContext.xml</param-value>
17     </context-param>
18
19
20     <!--配置前端控制器-->
21     <servlet>
22         <servlet-name>springmvc</servlet-name>
23         <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
24         <init-param>
25             <param-name>contextConfigLocation</param-name>
26             <param-value>classpath:spring/springmvc.xml</param-value>
27         </init-param>
28         <load-on-startup>1</load-on-startup>
29     </servlet>
30
31     <servlet-mapping>
32         <servlet-name>springmvc</servlet-name>
33         <url-pattern>/</url-pattern>
34     </servlet-mapping>
35
36
37
38
39 </web-app>
```

3.6 mybatis注解开

```
1  /**
2   * Created by zxd on 2022/10/27 15:47
3   *
4  public class Item implements Serializable {
5      private Integer id;
6
7      private String name;
8
9      private Float price;
10
11     private String pic;
12
13     private Date createtime;
14
15     private String detail;
16
17     public Item(){
18
19     }
20
21     public Item(Integer id, String name, Float price, String pic, Date cre
22     atetime, String detail) {
23         this.id = id;
24         this.name = name;
25         this.price = price;
26         this.pic = pic;
27         this.createtime = createtime;
28         this.detail = detail;
29     }
30
31     public Integer getId() {
32         return id;
33     }
34
35     public void setId(Integer id) {
36         this.id = id;
37     }
38
39     public String getName() {
40         return name;
41     }
42
43     public void setName(String name) {
44         this.name = name == null ? null : name.trim();
45     }
46 }
```

```
45
46     public Float getPrice() {
47         return price;
48     }
49
50     public void setPrice(Float price) {
51         this.price = price;
52     }
53
54     public String getPic() {
55         return pic;
56     }
57
58     public void setPic(String pic) {
59         this.pic = pic == null ? null : pic.trim();
60     }
61
62     public Date getCreatetime() {
63         return createtime;
64     }
65
66     public void setCreatetime(Date createtime) {
67         this.createtime = createtime;
68     }
69
70     public String getDetail() {
71         return detail;
72     }
73
74     public void setDetail(String detail) {
75         this.detail = detail == null ? null : detail.trim();
76     }
77 }
78 }
```

注解	说明
@Results	代替的是标签<resultMap>该注解中可以使用单个@Result注解，也可以使用@Result集合。使用格式： @Results ({@Result () , @Result () }) 或@Results (@Result ())
@Resut	代替了<id>标签和<result>标签 @Result中属性介绍： column: 数据库的列名 property: 需要装配的属性名 one: 需要使用的@One 注解 (@Result (one=@One) ()) many: 需要使用的@Many 注解 (@Result (many=@many) ())
@One (一对)	代替了<assocation> 标签，是多表查询的关键，在注解中用来指定子查询返回单一对象 @One注解属性介绍： select: 指定用来多表查询的 sqlmapper 使用格式： @Result(column=" ",property="",one=@One(select=""))
@Many (多对一)	代替了<collection>标签, 是多表查询的关键, 在注解中用来指定子查询返回对象集合。 使用格式： @Result(property="",column="",many=@Many(select=""))

```
1  /**
2   * Created by zxd on 2022/10/27 15:26
3   */
4  public interface ItemMapper {
5
6
7      //resultMap
8  @Results({@Result(id = true,column = "id",property = "id"),
9          @Result(column = "name",property = "name")})
10     @Select("select id,name,price,detail,pic,createtime from item where id
11 =#{id}")
12     Item findItemById(Integer id);
13
14     @Select("select id,name,price,detail,pic,createtime from item")
15     List<Item> findItemList();
16
17     @Insert("insert into item(name,price,detail,pic,createtime) values(#{n
ame},#{price},#{detail},#{pic},#{createtime})")
18     void insertItem(Item item);
19
20     @Update("update item set name=#{name},price=#{price},detail=#{detail},
21     pic=#{pic},createtime=#{createtime} where id=#{id}")
22     void updateItem(Item item);
23
24     @Delete("delete from item where id=#{id}")
25     void deleteItemById(Integer id);
26
27     @Select("<script> select * from item      <where>" +
28             "          <if test=\"name !=null and name!=''\">" +
29             "              and `name` like '%${name}%'" +
30             "          </if>" +
31             "      </where> </script>")
32     List<Item> findItemByQueryDto(QueryItemDto queryItemDto);
33 }
```

```
1  /**
2   * Created by zxd on 2022/10/27 15:44
3   */
4  @RunWith(SpringJUnit4ClassRunner.class)
5  @ContextConfiguration(locations = "classpath:spring/applicationContext.xml")
6  public class TestItemMapper {
7
8
9      @Autowired
10     private ItemMapper itemMapper;
11
12     @Test
13     public void testInsert(){
14         Item item=new Item(null,"华为mate100",18000F,"xx",new Date(),"xxx"
15         );
16         itemMapper.insertItem(item);
17     }
18
19     @Test
20     public void testUpdate(){
21         Item item=new Item(3,"华为mate60",18000F,"xx",new Date(),"xxx");
22
23         itemMapper.updateItem(item);
24     }
25
26
27
28     @Test
29     public void testFind(){
30
31         System.out.println(itemMapper.findItemById(3));
32
33         System.out.println(itemMapper.findItemList());
34     }
35
36
37     @Test
38     public void testQuery(){
39
40         QueryItemDto queryItemDto=new QueryItemDto();
41         queryItemDto.setName("华");
42
43         List<Item> itemList = itemMapper.findItemByQueryDto(queryItemDto);
```

```
44         System.out.println(itemList);
45     }
46
47
48
49
50 }
51 }
```

3.7 整合测试

3.7.1 编写service

编写接口



The screenshot shows a code editor window with a Java file named 'ItemService.java'. The code defines a public interface with a single method 'findItemList()' that returns a list of items. The code is annotated with Javadoc comments and includes a timestamp indicating it was created on 2022/10/27 at 15:59.

```
1   */
2  * Created by zxd on 2022/10/27 15:59
3  */
4  public interface ItemService {
5
6       */
7      * 查找所有商品
8      * @return
9      */
10     List<Item> findItemList();
11
12 }
```

编写实现类

```
1  /**
2   * Created by zxd on 2022/10/27 16:00
3   */
4  @Service
5  public class ItemServiceImpl implements ItemService {
6
7      @Autowired
8      private ItemMapper itemMapper;
9
10     @Override
11     public List<Item> findItemList() {
12         return itemMapper.findItemList();
13     }
14 }
15
```

3.7.2 编写controller

```
1  /**
2   * Created by zxd on 2022/10/27 16:01
3   */
4  @Controller
5  public class ItemController {
6
7
8      @Autowired
9      private ItemService itemService;
10
11     @RequestMapping("/queryItemList")
12     public ModelAndView queryItemList(){
13
14         List<Item> list = itemService.findItemList();
15
16         ModelAndView mv=new ModelAndView();
17         mv.addObject("list",list);
18         mv.setViewName("itemList");
19         return mv;
20     }
21
22 }
23
```

3.7.3 拷贝资源

The left side shows a file tree:

- test
- spring
 - applicationContext.xml
 - applicationContext-dao.xml
 - applicationContext-service.xml
 - applicationContext-tx.xml
 - springmvc.xml
 - jdbc.properties
 - log4j.properties
- web
 - layui
 - WEB-INF
 - lib
 - views

The right side shows the content of `springmvc.xml`:

```

4
5
6
7
8
9
10
11 <!--配置不拦截静态资源-->
12 <mvc:resources mapping="/layui/**" location="/layui/" />
13
14
15
16

```

配置不拦截静态资源

The code editor shows the `springmvc.xml` file with the following content:

```

1 <!--配置不拦截静态资源-->
2 <mvc:resources mapping="/layui/**" location="/layui/" />
3

```

4.参数绑定

4.1 默认的参数绑定

需求:

修改商品，查询商品信息 展示商品信息

根据id查询一个商品的信息，展示在修改页面

编写mapper

```
1 @Select("select id,name,price,detail,pic,createtime from item where id=#{id}")
2 Item findItemById(Integer id);
```

编写service

编写接口

```
1  /**
2   * Created by zxd on 2022/10/27 15:59
3   */
4  public interface ItemService {
5
6      /**
7       * 查找所有商品
8       * @return
9       */
10     List<Item> findItemList();
11
12
13     /**
14      * 根据id查询商品
15      * @param id
16      * @return
17      */
18     Item findItemById(Integer id);
19
20 }
```

编写实现类

```
Java |  
1  /**  
2   * Created by zxd on 2022/10/27 16:00  
3   */  
4  @Service  
5  public class ItemServiceImpl implements ItemService {  
6  
7      @Autowired  
8      private ItemMapper itemMapper;  
9  
10     @Override  
11     public List<Item> findItemList() {  
12         return itemMapper.findItemList();  
13     }  
14  
15     @Override  
16     public Item findItemById(Integer id) {  
17         return itemMapper.findItemById(id);  
18     }  
19 }  
20
```

编写controller

选择	商品名称	商品价格	生产日期	操作
<input type="checkbox"/>	华为mate60	18000.0	2022-10-27 15:50:16	<input type="button" value="修改"/>
<input type="checkbox"/>	华为mate60	8000.0	2022-06-27 14:46:26	<input type="button" value="修改"/>
<input type="checkbox"/>	华为mate100	18000.0	2022-10-27 15:48:50	<input type="button" value="修改"/>

```
</thead>
<tbody>
    <c:forEach items="${list}" var="item">
        <tr>
            <td><input type="checkbox" name="ids" value="${item.id}"></td>
            <td>${item.name}</td>
            <td>${item.price}</td>
            <td><fmt:formatDate value="${item.createtime}" pattern="yyyy-MM-dd HH:mm:ss" /></td>
            <td><a href="#">/updateItemUI?id=${item.id}">修改</a></td>
        </tr>
    </c:forEach>
</tbody>

body > form > tablelayui-table > tbody > tr > td > a
```

```
Java |
```

```
1 @RequestMapping("/updateItemUI")
2 public ModelAndView updateItemUI(HttpServletRequest request){
3
4     //1.获取请求参数id
5     String id = request.getParameter("id");
6
7     //2.调用service进行查询
8     Item item = itemService.findItemById(Integer.valueOf(id));
9
10    //3.封装ModelAndView对象
11    ModelAndView mv=new ModelAndView();
12    mv.addObject("item",item);
13    mv.setViewName("editItem");
14
15    return mv;
16 }
17
```

默认的参数绑定

我们只需要在方法上声明HttpServletRequest，处理器适配器自动绑定request对象

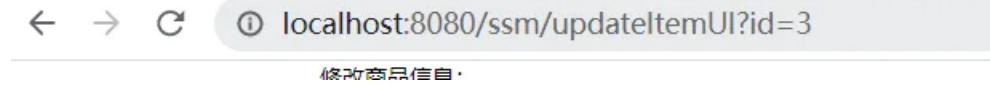
```
HttpServletRequest response
```

```
HttpSession session
```

```
Java |  
1  
2 @RequestMapping("/updateItemUI")  
3 public void updateItemUI(HttpServletRequest request, HttpServletResponse response, HttpSession session) throws ServletException, IOException {  
4     //1.获取请求参数id  
5     String id = request.getParameter("id");  
6  
7     //2.调用service进行查询  
8     Item item = itemService.findItemById(Integer.valueOf(id));  
9  
10    request.setAttribute("item", item);  
11    request.getRequestDispatcher("/WEB-INF/views/editItem.jsp").forward(request, response);  
12  
13    }  
14 }
```

不推荐直接使用原生对象，因为这样的就没有使用springmvc封装的相关工程

4.2 绑定简单类型



localhost:8080/ssm/updateItemUI?id=3

修改商品信息

我们可以直接在处理器方法上声明我们需要接收的参数，当请求参数的名字和处理器形参名一致的时候，springmvc会进行自动的参数绑定。

```
1  @RequestMapping("/updateItemUI")
2  public ModelAndView updateItemUI(Integer id){
3
4
5      //2.调用service进行查询
6      Item item = itemService.findItemById(id);
7
8      //3.封装ModelAndView对象
9      ModelAndView mv=new ModelAndView();
10     mv.addObject("item",item);
11     mv.setViewName("editItem");
12
13     return mv;
14 }
15
```

整型 Integer int

Long long

字符串 String

单精度 Float float

双精度 Double double

布尔类型 Boolean boolean 请求参数传递 true或者false 1或0

4.3 RequestParam注解(重点)

```

1  @Target({ElementType.PARAMETER})
2  @Retention(RetentionPolicy.RUNTIME)
3  @Documented
4  public @interface RequestParam {
5
6      //处理器参数名称不一致的情况
7      String value() default "";
8
9      //参数是否必填
10     boolean required() default true;
11
12     //默认值
13     String defaultValue() default "\n\t\t\n\t\t\t\n\t\t\t\t\n\t\t\t\t\t";
14 }

```

@RequestParam 注解有三个属性 三个作用

required 参数是否必须 默认为true，没有传递参数就会报错

The screenshot shows a browser window with the URL `localhost:8080/ssm/updateItemUI`. The page title is "HTTP状态 400 - 错误的请求". Below it, there's a "类型 状态报告" section with a "消息" entry: "Required Integer parameter 'id' is not present". A descriptive note below states: "由于被认为是客户端对错误 (例如: 畸形的请求语法、无效的请求信息帧或者虚拟的请求路由) , 服务器无法或不会处理当前请求。"

value 前端传递的参数名称 可以处理器前后端参数不一致的情况 如果一致可以不配置

```
@RequestMapping("/updateItemUI")
public ModelAndView updateItemUI(@RequestParam(value = "itemId") Integer id) {
```

修改商品信息:

商品名称:	华为mate60
商品价格:	8000.0
商品生产日期:	2022-06-27 14:46:26
商品简介:	XXX

defaultValue 默认值,如果required 设置为true, 代表参数必传, 如果配置了默认值, 不传递参数使用默认值

```
@RequestMapping("/updateItemUI")
public ModelAndView updateItemUI(@RequestParam(value = "itemId", defaultValue = "3") Integer id) {
```

修改商品信息:

商品名称:	华为mate60
商品价格:	18000.0
商品生产日期:	2022-10-27 15:50:16

4.4 Model/ModelMap

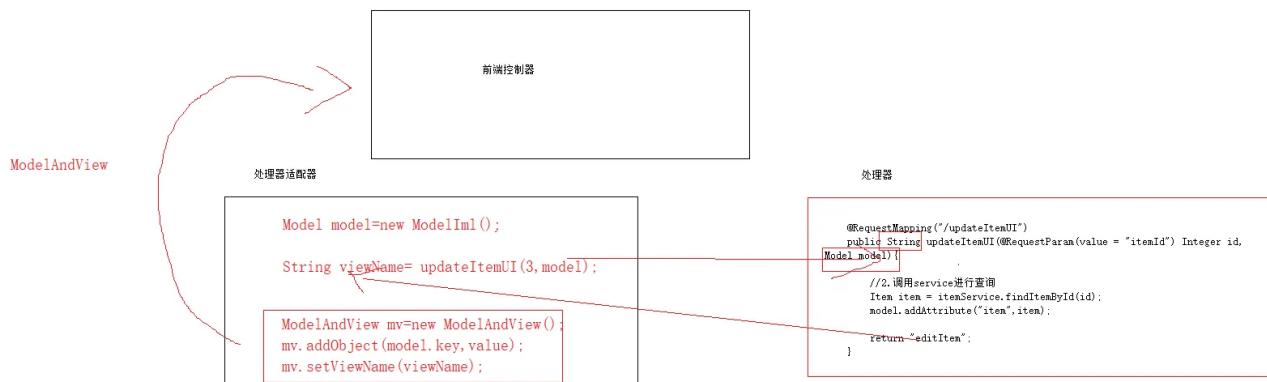
使用Model向request域放置数据, 直接在方法的参数上面进行声明即可

Model就是map集合，处理器适配器会传给我一个map集合，我们就可以往map集合设置数据，处理器方法的返回

是String类型，就是视图的名字。

处理器适配器就会自动创建一个ModelAndView对象给前端控制器

```
Java |  
1 @RequestMapping("/updateItemUI")  
2 public String updateItemUI(@RequestParam(value = "itemId") Integer id,  
3 Model model){  
4     //2.调用service进行查询  
5     Item item = itemService.findById(id);  
6     model.addAttribute("item",item);  
7  
8     return "editItem";  
9 }
```



ModelMap是一个实现类

```
1  @RequestMapping("/updateItemUI")
2  public String updateItemUI(@RequestParam(value = "itemId") Integer id,
3                               ModelMap model){
4      //2.调用service进行查询
5      Item item = itemService.findItemById(id);
6      model.addAttribute("item",item);
7
8      return "editItem";
9  }
```

4.5 绑定pojo类型

修改商品信息：

商品名称:	华为mate60
商品价格:	8000.0
商品生产日期:	2022-06-27 14:46:26
商品简介	XXX

立即提交

需求： 提交修改请求， 修改商品信息

编写mapper

Java |

```
1 @Update("update item set name=#{name},price=#{price},detail=#{detail},pic  
=#{pic},createtime=#{createtime} where id=#{id}")  
2 void updateItem(Item item);
```

编写service

接口

Java |

```
1  /**  
2   * 修改商品  
3   * @param item  
4   */  
5 void updateItem(Item item);
```

编写实现

Java |

```
1 @Override  
2 public void updateItem(Item item) {  
3     itemMapper.updateItem(item);  
4 }
```

编写controller

修改商品信息：

商品名称:	华为mate60
商品价格:	8000.0
商品生产日期:	2022-06-27 14:46:26
商品简介:	XXX

```

<body>
    <!-- 上传图片是需要指定属性 enctype="multipart/form-data" -->
    <!-- <form id="itemForm" action="" method="post" enctype="multipart/form-data"> -->
    <form id="itemForm" class="layui-form"
        action="${pageContext.request.contextPath }/updateItem"
        method="post" style="..." align="center">
        <input type="hidden" name="id" value="${item.id }" /> 修改商品信息:

        <div class="layui-form-item">
            <label class="layui-form-label">商品名称:</label>
            <div class="layui-input-block">
                <input type="text" name="name" value="${item.name }"
                    lay-verify="required" placeholder="请输入商品名称" autocomplete="off"
                    class="layui-input" >
            </div>
        </div>
        <div class="layui-form-item">
            <label class="layui-form-label">商品价格:</label>
            <div class="layui-input-block">
                <input type="text" name="price" value="${item.price }" 

```

Java |

```

1 @RequestMapping("/updateItem")
2     public String updateItem(Item item){
3         itemService.updateItem(item);
4         return "success";
5     }

```

HTTP状态 400 - 错误的请求

状态报告

由于被认为是客户端对错误（例如：畸形的请求语法、无效的请求信息帧或者虚拟的请求路由），服务器无法或不会处理当前请求。

Apache Tomcat/8.5.65

```
ing to 'Editor' suffix convention
ing com.gxa.controller.ItemController.updateItem(com.gxa.pojo.Item)]; org.springframework.validation.BindException: org.springframework.validation.BeanPropertyBindingResult: 1
m.createtime,typeMismatch.createtime,typeMismatch.java.util.Date,typeMismatch]; arguments [org.springframework.context.support.DefaultMessageSourceResolvable: codes [item.creat
g com.gxa.controller.ItemController.updateItem(com.gxa.pojo.Item)]; org.springframework.validation.BindException: org.springframework.validation.BeanPropertyBindingResult: 1 er
m.createtime,typeMismatch.createtime,typeMismatch.java.util.Date,typeMismatch]; arguments [org.springframework.context.support.DefaultMessageSourceResolvable: codes [item.creat
g com.gxa.controller.ItemController.updateItem(com.gxa.pojo.Item)]; org.springframework.validation.BindException: org.springframework.validation.BeanPropertyBindingResult: 1 er
m.createtime,typeMismatch.createtime,typeMismatch.java.util.Date,typeMismatch]; arguments [org.springframework.context.support.DefaultMessageSourceResolvable: codes [item.creat
: assuming HandlerAdapter completed request handling
```

4.6 编写日期转换器

```
1
2  /**
3   * Created by zxd on 2022/10/28 10:37
4   */
5  public class DateConverter implements Converter<String, Date> {
6
7      @Override
8      public Date convert(String source) {
9          try{
10
11              SimpleDateFormat df=new SimpleDateFormat("yyyy-MM-dd HH:mm:ss"
12          );
13              Date date = df.parse(source);
14              return date;
15          }catch (Exception e){
16              e.printStackTrace();
17              throw new RuntimeException("日转换失败!");
18          }
19      }
20  }
```

配置

```
1 <beans xmlns="http://www.springframework.org/schema/beans"
2     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:context="h
3         ttp://www.springframework.org/schema/context"
4     xmlns:mvc="http://www.springframework.org/schema/mvc"
5     xsi:schemaLocation="
6         http://www.springframework.org/schema/beans http://www.springframe
7         work.org/schema/beans/spring-beans-4.0.xsd
8         http://www.springframework.org/schema/context http://www.springfram
9         ework.org/schema/context/spring-context-4.0.xsd
10        http://www.springframework.org/schema/mvc http://www.springframewor
11        k.org/schema/mvc/spring-mvc-4.0.xsd">
12
13
14    <!--配置不拦截静态资源-->
15    <mvc:resources mapping="/layui/**" location="/layui/" />
16
17    <!--配置扫描controller包-->
18    <context:component-scan base-package="com.gxa.controller" />
19
20    <!--配置注解驱动-->
21    <mvc:annotation-driven conversion-service="conversionService" />
22
23    <!--配置日期转换器-->
24    <bean id="conversionService" class="org.springframework.format.sup
25        port.FormattingConversionServiceFactoryBean">
26        <property name="converters">
27            <set>
28                <bean class="com.gxa.converter.DateConverter" />
29            </set>
30        </property>
31    </bean>
32
33    <!--配置视图解析器-->
34    <bean class="org.springframework.web.servlet.view.InternalResource
35        ViewResolver">
36        <property name="prefix" value="/WEB-INF/views/" />
37        <property name="suffix" value=".jsp" />
38    </bean>
39
40</beans>
```

4.7 解决乱码问题

Java

```
1      <!--配置解决乱码的过滤器-->
2      <filter>
3          <filter-name>characterEncodingFilter</filter-name>
4          <filter-class>org.springframework.web.filter.CharacterEncodingFilter</filter-class>
5          <init-param>
6              <param-name>encoding</param-name>
7              <param-value>UTF-8</param-value>
8          </init-param>
9      </filter>
10
11     <filter-mapping>
12         <filter-name>characterEncodingFilter</filter-name>
13         <url-pattern>/*</url-pattern>
14     </filter-mapping>
15
```

4.8 绑定包装的javabean类型 (了解)

需求: 根据id和商品名称进行查询

Java |

```
1  /**
2   * Created by zxd on 2022/10/28 10:52
3   */
4  public class QueryDto {
5
6      private Item item;
7
8      public Item getItem() {
9          return item;
10     }
11
12     public void setItem(Item item) {
13         this.item = item;
14     }
15
16 }
```

Java |

```
1 @RequestMapping("/queryItemListByWhere")
2 public String queryItemListByWhere(QueryDto queryDto){
3     System.out.println(queryDto);
4     return "success";
5 }
```

```
<form action="${pageContext.request.contextPath }/queryItemListByWhere" method="post">






```

```
59
60     @RequestMapping("/queryItemListByWhere")
61     public String queryItemListByWhere(QueryDto queryDto) {    queryDto: (
62         System.out.println(queryDto);    queryDto: QueryDto@5849
63         return "success";
64     }
```

Evaluate

Expression: queryDto

Result:

```
result = {QueryDto@5849}
    f item = {Item@5876} "Item{id=1, name='ddd', price=null, pic='null', createtime=}
```

4.9 绑定集合或者数组

```
1
2  /**
3   * Created by zxd on 2022/10/28 10:52
4   */
5  public class QueryDto {
6
7      private Item item;
8
9      private List<Integer> ids;
10
11     public List<Integer> getIds() {
12         return ids;
13     }
14
15     public void setIds(List<Integer> ids) {
16         this.ids = ids;
17     }
18
19     public Item getItem() {
20         return item;
21     }
22
23     public void setItem(Item item) {
24         this.item = item;
25     }
26
27 }
28
```

修改前端

```

<body>
    <form action="${pageContext.request.contextPath }/queryItemListByWhere"
        method="post">
        <table >
        <%-->
            <h6>查询条件: </h6>
            <div class="layui-form-item">
                <label class="layui-form-label">商品id:</label>
                <div class="layui-input-block">
                    <input type="text" name="item.id" required lay-verify="required"
                        placeholder="请输入标题" autocomplete="off" class="layui-input" style="width:200px">
                </div>
            </div>
            <div class="layui-form-item">
                <label class="layui-form-label">商品名称:</label>
                <div class="layui-input-block">
                    <input type="text" name="item.name" required lay-verify="required"
                        placeholder="请输入标题" autocomplete="off" class="layui-input" style="width:200px">
                </div>
            </div>
        <%-->
        <tr>
            <td><input type="submit" class="layui-btn layui-btn-normal">
                <!--<button type="button" value="提交"/>-->
            </td>
            <td>商品名称</td>
            <td>商品价格</td>
            <td>生产日期</td>
            <td>操作</td>
        </tr>
    </thead>
    <tbody>
        <c:forEach items="${list}" var="item">
            <tr>
                <td><input type="checkbox" name="ids" value="${item.id}"></td>
                <td>${item.name}</td>
                <td>${item.price}</td>
            </tr>
        </c:forEach>
    </tbody>
</table>

```

后端

```

1  @RequestMapping("/queryItemListByWhere")
2  public String queryItemListByWhere(QueryDto queryDto){
3      System.out.println(queryDto);
4      return "success";
5  }

```

<input checked="" type="checkbox"/>	华为mate60	18000.0	2022-10-27 15:50:16
<input checked="" type="checkbox"/>	华为mate300	8000.0	2022-06-27 14:46:26
<input type="checkbox"/>	华为mate100	18000.0	2022-10-27 15:48:50

DevTools is now available in Chinese! [Always match Chrome's language](#) [Switch DevTools to Chinese](#) [Don't show again](#)

Elements Recorder Console Sources Network Performance Memory Application Lighthouse JavaScript Profiler

Preserve log Disable cache No throttling

Filter Invert Hide data URLs All Fetch/XHR JS CSS Img Media Font Doc WS Wasm Manifest Other Has blocked cookies Blocked Requests

5 ms 10 ms 15 ms 20 ms 25 ms 30 ms 35 ms 40 ms 45 ms 50 ms 55 ms 60 ms 65 ms 70 ms 75 ms 80 ms

Name Headers **Payload** Preview Response Initiator Timing Cookies

queryItemListByWhere Form Data [view source](#) [view URL-encoded](#)

ids: 3
ids: 6

```

    @RequestMapping("/queryItemListByWhere")
    public String queryItemListByWhere(QueryDto queryDto) { queryDto: QueryDto@5858
        System.out.println(queryDto); queryDto: QueryDto@5858
        return "success";
    }

    Evaluate
    Expression: queryDto
    Result:
    > <@result = {QueryDto@5858}
      f item = null
    > f ids = {ArrayList@5884} size = 2
  
```

list集合不能直接声明在方法的参数中，需要用javabean类型包装装起来

```

1  @RequestMapping("/queryItemListByWhere")
2  public String queryItemListByWhere(List<Integer> ids){
3      System.out.println(ids);
4      return "success";
5  }
6

```

HTTP状态 500 - 内部服务器错误

类型 异常报告

消息 Request processing failed; nested exception is org.springframework.beans.BeanInstantiationException: Failed to instantiate [java.util.List]: Specified class is an interface

描述 服务器遇到一个意外的情况，阻止它完成请求。

[例外情况](#)

可以直接在参数上直接绑定数组

```

1  @RequestMapping("/queryItemListByWhere")
2  public String queryItemListByWhere(Integer[] ids){
3      System.out.println(ids);
4      return "success";
5  }

```

5.RequestMapping

RequestMapping注解可以配置不同处理器的映射路径

5.1 URL映射路径

一个处理器可以有多个映射路径

```

1
2     @RequestMapping(value = {"#/queryItemList","/list"})
3     public ModelAndView queryItemList(){
4
5         List<Item> list = itemService.findItemList();
6
7         ModelAndView mv=new ModelAndView();
8         mv.addObject("list",list);
9         mv.setViewName("itemList");
10        return mv;
11    }

```

不能出现一个映射路径出现在多个处理器上面

HTTP状态 500 - 内部服务器错误

异常报告

Request processing failed: nested exception is java.lang.IllegalStateException: Ambiguous handler methods mapped for HTTP path 'http://localhost:8080/ssm/list': {public org.springframework.web.servlet.ModelAndView com.gxa.controller.ItemController.queryItemList(), public java.lang.String com.gxa.controller.ItemController.updateItemUI(java.lang.Integer org.springframework.ui.ModelMap)}

服务器遇到一个意外的情况，阻止它完成请求。

错误堆栈

```

org.springframework.web.util.NestedServletException: Request processing failed. nested exception is java.lang.IllegalStateException: Ambiguous handler methods mapped for HTTP path 'http://localhost:8080/ssm/list': {public org.springframework.web.servlet.ModelAndView com.gxa.controller.ItemController.queryItemList(), public java.lang.String com.gxa.controller.ItemController.updateItemUI(java.lang.Integer org.springframework.ui.ModelMap)}
org.springframework.web.util.NestedServletException: Request processing failed. nested exception is java.lang.IllegalStateException: Ambiguous handler methods mapped for HTTP path 'http://localhost:8080/ssm/list': {public org.springframework.web.servlet.ModelAndView com.gxa.controller.ItemController.queryItemList(), public java.lang.String com.gxa.controller.ItemController.updateItemUI(java.lang.Integer org.springframework.ui.ModelMap)}
org.springframework.web.servlet.FrameworkServlet.service(FrameworkServlet.java:978)
org.springframework.web.servlet.HttpServletHandler.handleRequestInternal(AbstractHandlerMethodMapping.java:857)
org.springframework.web.servlet.MockMvc.perform(MockMvc.java:1044)
org.springframework.http.HttpFilter.doFilterInternal(HttpFilter.java:626)
org.springframework.web.servlet.FrameworkServlet.service(FrameworkServlet.java:842)
javax.servlet.http.HttpServlet.service(HttpServlet.java:733)
org.apache.tomcat.websocket.server.WsFilter.doFilter(WsFilter.java:52)
org.springframework.web.filter.CharacterEncodingFilter.doFilterInternal(CharacterEncodingFilter.java:88)
org.springframework.web.filter.CharacterEncodingFilter.doFilterInternal(CharacterEncodingFilter.java:88)
org.springframework.web.filter.OncePerRequestFilter.doFilter(OncePerRequestFilter.java:107)
org.apache.logging.log4j.core.web.Log4jServletFilter.doFilter(Log4jServletFilter.java:66)

```

根本原因

```

java.lang.IllegalStateException: Ambiguous handler methods mapped for HTTP path 'http://localhost:8080/ssm/list': {public org.springframework.web.servlet.ModelAndView com.gxa.controller.ItemController.queryItemList(), public java.lang.String com.gxa.controller.ItemController.updateItemUI(java.lang.Integer org.springframework.ui.ModelMap)}
org.springframework.web.servlet.HandlerMapping.getHandlerInternal(HandlerMapping.java:106)
org.springframework.web.servlet.HandlerMapping.getHandlerInternal(AbstractHandlerMethodMapping.java:344)
org.springframework.web.servlet.HandlerMapping.getHandlerInternal(AbstractHandlerMethodMapping.java:299)
org.springframework.web.servlet.handler.AbstractHandlerMethodMapping.getHandlerInternal(AbstractHandlerMethodMapping.java:57)
org.springframework.web.servlet.handler.AbstractHandlerMapping.getHandler(AbstractHandlerMapping.java:299)
org.springframework.web.servlet.DispatcherServlet.getHandler(DispatcherServlet.java:1104)
org.springframework.web.servlet.DispatcherServlet.doDispatch(DispatcherServlet.java:916)
org.springframework.web.servlet.DispatcherServlet.doService(DispatcherServlet.java:877)
org.springframework.web.servlet.FrameworkServlet.processRequest(FrameworkServlet.java:966)
org.springframework.web.servlet.FrameworkServlet.doGet(FrameworkServlet.java:867)
javax.servlet.http.HttpServlet.service(HttpServlet.java:626)
org.springframework.web.servlet.FrameworkServlet.service(FrameworkServlet.java:842)
javax.servlet.http.HttpServlet.service(HttpServlet.java:733)
org.apache.tomcat.websocket.server.WsFilter.doFilter(WsFilter.java:52)
org.springframework.web.filter.CharacterEncodingFilter.doFilterInternal(CharacterEncodingFilter.java:88)
org.springframework.web.filter.CharacterEncodingFilter.doFilterInternal(CharacterEncodingFilter.java:88)
org.springframework.web.filter.OncePerRequestFilter.doFilter(OncePerRequestFilter.java:107)
org.apache.logging.log4j.core.web.Log4jServletFilter.doFilter(Log4jServletFilter.java:66)

```

主要问题的全部 stack 信息可以在 server logs 里查看

Apache Tomcat/8.5.65

5.2 作用在类上面

如果添加在类上面，以后访问这个类下面所有的处理器加上这个前缀

```
1  /**
2   * Created by zxd on 2022/10/27 16:01
3   */
4   @Controller
5   @RequestMapping("/item")
6   public class ItemController {
7
8
9     @Autowired
10    private ItemService itemService;
11
12    @RequestMapping(value = {"queryItemList","/list"})
13    public ModelAndView queryItemList(){
14
15        List<Item> list = itemService.findItemList();
16
17        ModelAndView mv=new ModelAndView();
18        mv.addObject("list",list);
19        mv.setViewName("itemList");
20
21        return mv;
22    }
23}
```

5.3 限制请求方式

```
12  
13     @RequestMapping(value = "/list", method = RequestMethod.POST)  
14     public ModelAndView queryItemList(){  
15         List<Item> list = itemService.findItemList();  
16  
17         ModelAndView mv=new ModelAndView();  
18         mv.addObject( attributeName: "list", list);  
19         mv.setViewName("itemList");  
20     }  
21 
```

A 课堂派-简单好用的互动课堂管理 × HTTP状态 405 - 方法不允许 × +
← → C ⓘ localhost:8080/ssm/item/list

HTTP状态 405 - 方法不允许

类型 状态报告

消息 Request method 'GET' not supported

描述 请求行中接收的方法由源服务器知道，但目标资源不支持

如果不配置就代表不限制请求方式

6.Controller方法的返回

6.1 返回 ModelAndView

```

1  @RequestMapping(value = "/list",method = RequestMethod.GET)
2  public ModelAndView queryItemList(){
3
4      List<Item> list = itemService.findItemList();
5
6      ModelAndView mv=new ModelAndView();
7      mv.addObject("list",list);
8      mv.setViewName("itemList");
9
10     return mv;
11 }
```

6.2 返回void

```

1  @RequestMapping("/updateItemUI")
2  public void updateItemUI(HttpServletRequest request, HttpServletResponse response, HttpSession session) throws ServletException, IOException {
3
4      //1.获取请求参数id
5      String id = request.getParameter("id");
6
7      //2.调用service进行查询
8      Item item = itemService.findItemById(Integer.valueOf(id));
9
10
11     request.setAttribute("item",item);
12     request.getRequestDispatcher("/WEB-INF/views/editItem.jsp").forward(request,response);
13
14 }
```

6.3 返回String类型

6.3.1 逻辑视图的名字

转发到jsp

```
1      @RequestMapping("/updateItemUI")
2      public String updateItemUI(@RequestParam(value = "itemId") Integer id,
3                                   ModelMap model){
4          //2.调用service进行查询
5          Item item = itemService.findItemById(id);
6          model.addAttribute("item",item);
7
8          return "editItem";
9      }
```

6.3.2 转发到其他处理器

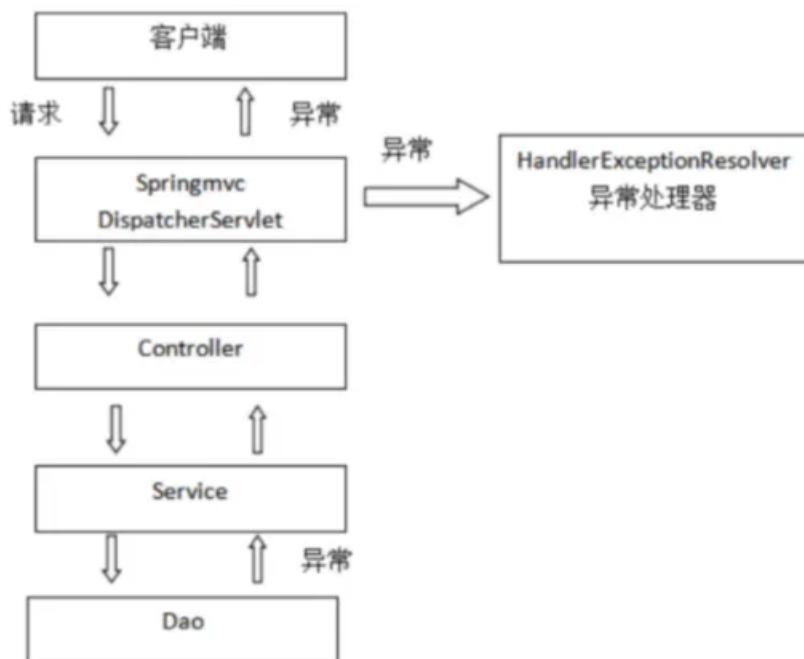
```
1      @RequestMapping("/updateItem")
2      public String updateItem(Item item){
3          itemService.updateItem(item);
4          return "forward:/item/list";
5      }
```

6.3.3 重定向

```
1      @RequestMapping("/updateItem")
2      public String updateItem(Item item){
3          itemService.updateItem(item);
4          return "redirect:/item/list";
5      }
```

7.异常处理器

7.1 springmvc异常处理



7.2 预期异常(自定义异常)

预期异常：自定义异常

不可预期异常 未知异常 系统繁忙，请待会再来！

7.3 编写自定义异常类

```
1
2  /**
3   * Created by zxd on 2022/10/28 14:04
4   */
5  public class CustomerException extends RuntimeException {
6
7      //定义异常信息
8      private String message;
9
10     public CustomerException(String message) {
11         super(message);
12         this.message = message;
13     }
14
15     @Override
16     public String getMessage() {
17         return message;
18     }
19
20     public void setMessage(String message) {
21         this.message = message;
22     }
23 }
24
```

7.4 编写异常处理器

```
1  /**
2   * Created by zxd on 2022/10/28 14:08
3   */
4  public class MyExceptionHandler implements HandlerExceptionResolver {
5      @Override
6      public ModelAndView resolveException(HttpServletRequest request, HttpServletResponse response, Object handler, Exception exception) {
7
8
9         String msg="";
10
11        if(exception instanceof CustomerException){
12            //判断是否自定义异常
13            CustomerException customerException= (CustomerException) exception;
14            msg=customerException.getMessage();
15        }else{
16            exception.printStackTrace();
17            msg="对不起，系统繁忙，请明天再来!";
18        }
19
20        ModelAndView modelAndView=new ModelAndView();
21        modelAndView.addObject("msg",msg);
22        modelAndView.setViewName("error");
23
24        return modelAndView;
25    }
26}
27
```

配置

```
1 <beans xmlns="http://www.springframework.org/schema/beans"
2     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:context="h
3         ttp://www.springframework.org/schema/context"
4     xmlns:mvc="http://www.springframework.org/schema/mvc"
5     xsi:schemaLocation="
6             http://www.springframework.org/schema/beans http://www.springframe
7         work.org/schema/beans/spring-beans-4.0.xsd
8             http://www.springframework.org/schema/context http://www.springfram
9         ework.org/schema/context/spring-context-4.0.xsd
10            http://www.springframework.org/schema/mvc http://www.springframewor
11         k.org/schema/mvc/spring-mvc-4.0.xsd">
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
```

```
40  
41    </beans>
```

7.5 测试异常处理

```
1  @RequestMapping(value = "/list")  
2  public ModelAndView queryItemList(){  
3  
4      if(true){  
5          throw new CustomerException("用户名密码错误!");  
6      }  
7  
8      List<Item> list = itemService.findItemList();  
9  
10     ModelAndView mv=new ModelAndView();  
11     mv.addObject("list",list);  
12     mv.setViewName("itemList");  
13     return mv;  
14  }  
15
```

8.图片上传

8.1 nginx

8.1.1 nginx是什么

HTTP和反向代理web服务器

Nginx (engine x) 是一个高性能的HTTP和反向代理web服务器，同时也提供了IMAP/POP3/SMTP服务。

Nginx是由伊戈尔·赛索耶夫为俄罗斯访问量第二的Rambler.ru站点（俄文：Рамблер）开发的，公开版本1.19.6发布于2020年12月15日。[\[12\]](#)

其将源代码以类BSD许可证的形式发布，因它的稳定性、丰富的功能集、简单的配置文件和低系统资源的消耗而闻名。2022年01月25日，nginx 1.21.6发布。[\[13\]](#)

Nginx是一款轻量级的Web 服务器/反向代理服务器及电子邮件（IMAP/POP3）代理服务器，在BSD-like 协议下发行。其特点是占有内存少，并发能力强，事实上nginx的并发能力在同类型的网页服务器中表现较好。

单机并发5发

nginx可以做http服务器，但是nginx只能作为静态资源服务器的

nginx可以做反向代理服务器

8.1.2 nginx做HTTP服务器

名称	修改日期	类型	大小
conf 配置目录	2022/10/28 14:30	文件夹	
contrib	2022/10/28 14:30	文件夹	
docs	2022/10/28 14:30	文件夹	
html	2022/10/28 14:30	文件夹	
logs	2020/4/21 22:12	文件夹	
temp	2020/4/21 22:12	文件夹	
G nginx.exe 启动程序	2020/4/21 21:26	应用程序	3,630 KB

G nginx.exe (32 位)	0%	2.2 MB	0 MB/秒	0 Mbps
G nginx.exe (32 位)	0%	1.9 MB	0 MB/秒	0 Mbps

The screenshot shows a web browser window with the URL 'localhost' in the address bar. The page content is the standard 'Welcome to nginx!' message. Below the main text, there is a note about further configuration required, a link to the online documentation at nginx.org, and a link to commercial support at nginx.com. A small note at the bottom says 'Thank you for using nginx.'

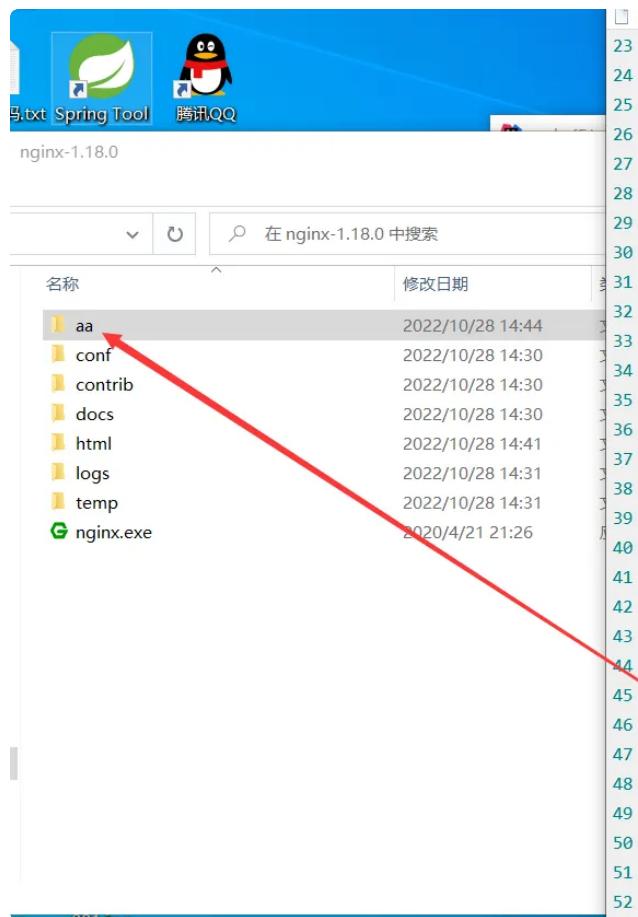
8.1.3 通过端口来区分虚拟主机

nginx中一个server节点就是一个虚拟主机

The screenshot shows a file manager interface on the left and a code editor on the right. The file manager lists several directories: conf, contrib, docs, html, logs, temp, and nginx.exe. The 'html' directory is highlighted with a red arrow. The code editor displays the nginx configuration file, specifically the 'server' block for port 80, which maps the root directory to 'html'. The browser window at the bottom shows the result of the configuration, displaying a small image of a dog.

```
20
21
22 server {
23
24     listen      80;
25     server_name localhost;
26
27
28     location / {
29         root  html;
30         index index.html index.htm;
31     }
32
33
34     error_page   500 502 503 504  /50x.html;
35     location = /50x.html {
36         root  html;
37     }
38
39
40 }
```

nginx可以通过端口来区分同时虚拟出多台服务器



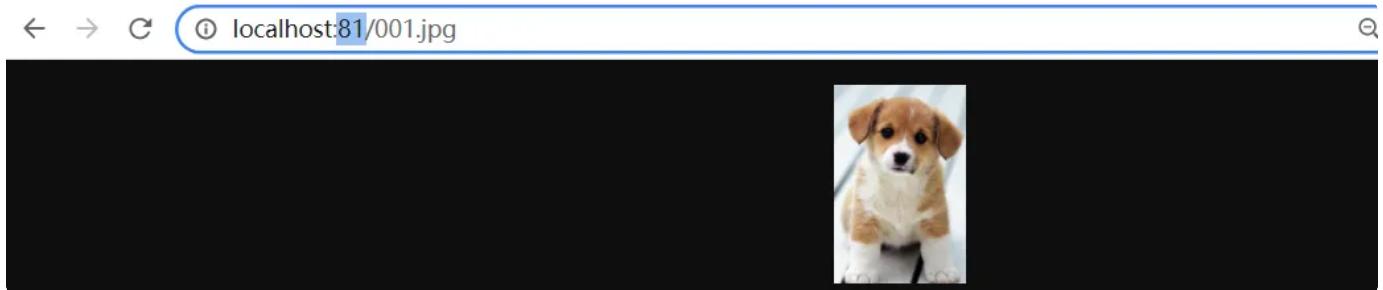
```
23
24     listen      80;
25     server_name  localhost;
26
27
28     location / {
29         root    html;
30         index   index.html index.htm;
31     }
32
33
34     error_page  500 502 503 504  /50x.html;
35     location = /50x.html {
36         root    html;
37     }
38
39
40
41
42
43     server {
44
45         listen      81;
46         server_name localhost;
47
48
49     location / {
50         root    aa;
51     }
52 }
```

文件夹内容

名称	修改日期	类型
aa	2022/10/28 14:44	文件夹
conf	2022/10/28 14:30	文件夹
contrib	2022/10/28 14:30	文件夹
docs	2022/10/28 14:30	文件夹
html	2022/10/28 14:41	文件夹
logs	2022/10/28 14:31	文件夹
temp	2022/10/28 14:31	文件夹
nginx.exe	2020/4/21 21:26	应用程序

```
C:\Windows\system32\cmd.exe
Microsoft Windows [版本 10.0.19043.2130]
(c) Microsoft Corporation。保留所有权利。

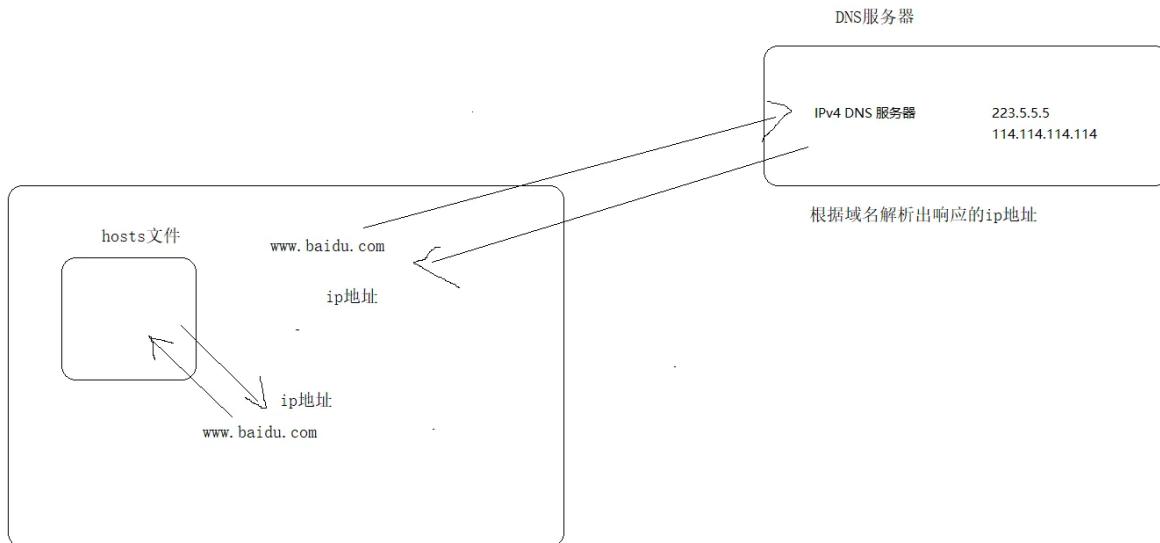
C:\Users\gxa>cd E:\tool\nginx-1.18.0
C:\Users\gxa>e:
E:\tool\nginx-1.18.0>nginx.exe -s reload
```



8.1.4 nginx可以通过域名来区分

```
server {  
  
    listen      80;  
    server_name www.j309.com;  
  
    location / {  
        root  html;  
        index index.html index.htm;  
    }  
  
    error_page   500 502 503 504  /50x.html;  
    location = /50x.html {  
        root  html;  
    }  
  
}  
  
server {  
  
    listen      80;  
    server_name www.banban.com;
```

8.1.5 dns解析



配置hosts文件

	名称	修改日期	类型
司	hosts	2022/6/15 11:47	文件
ve - Personal	hosts.backup_switchhosts	2021/6/25 10:03	BACKUP_SWITC...
盘	hosts.dz	2022/1/18 9:50	DZ 文件
象	hosts.ics	2022/5/6 8:47	Calendar
	lmhosts.sam	2019/12/7 17:12	SAM 文件
	networks	2019/12/7 17:12	文件
	protocol	2019/12/7 17:12	文件
	services	2019/12/7 17:12	文件

```

# For example:
#
#      102.54.94.97      rhino.acme.com      # source
#      38.25.63.10       x.acme.com        # x client

# localhost name resolution is handled within DNS itself:
#      127.0.0.1        localhost
#      ::1               localhost
127.0.0.1      activate.navicat.com
127.0.0.1      www.gxa.com
127.0.0.1      www.sunshine.com
127.0.0.1      image.sunshine.com
127.0.0.1      web.sunshine.com
127.0.0.1      www.zps.com
127.0.0.1      www.j295.com
127.0.0.1      web.happiness.com
127.0.0.1      image.happiness.com
127.0.0.1      www.happiness.com
127.0.0.1      eureka01.com
127.0.0.1      eureka02.com
127.0.0.1      eureka03.com
127.0.0.1      www.htl.com
127.0.0.1      www.j309.com
127.0.0.1      www.banban.com

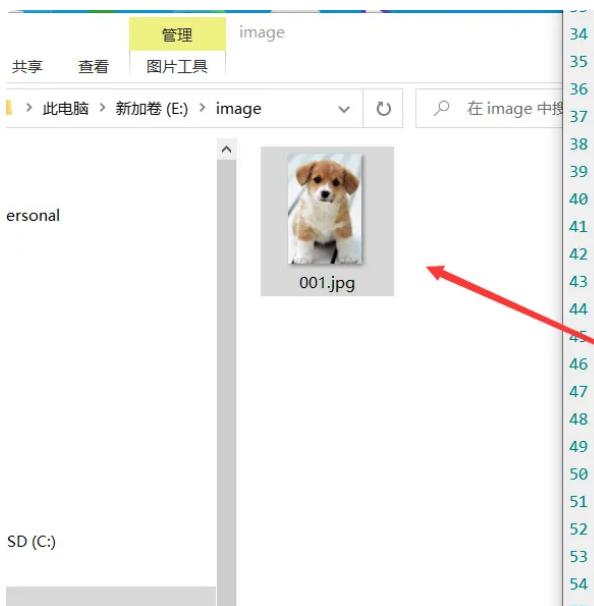
23
24      listen      80;
25      server_name www.j309.com;
26
27      location / {
28          root      html;
29          index     index.html index.htm;
30      }
31
32
33
34      error_page 500 502 503 504 /50x.html;
35      location = /50x.html {
36          root      html;
37      }
38
39
40
41
42
43      server {
44
45          listen      80;
46          server_name www.banban.com;
47
48
49      location / {

```

/38 列 1/41 字符 1/41 求值 -- 选定 -- 选行 -- 匹配 -- 1.32 KB Unicode (U) 选中 1 个项目 1.32 KB

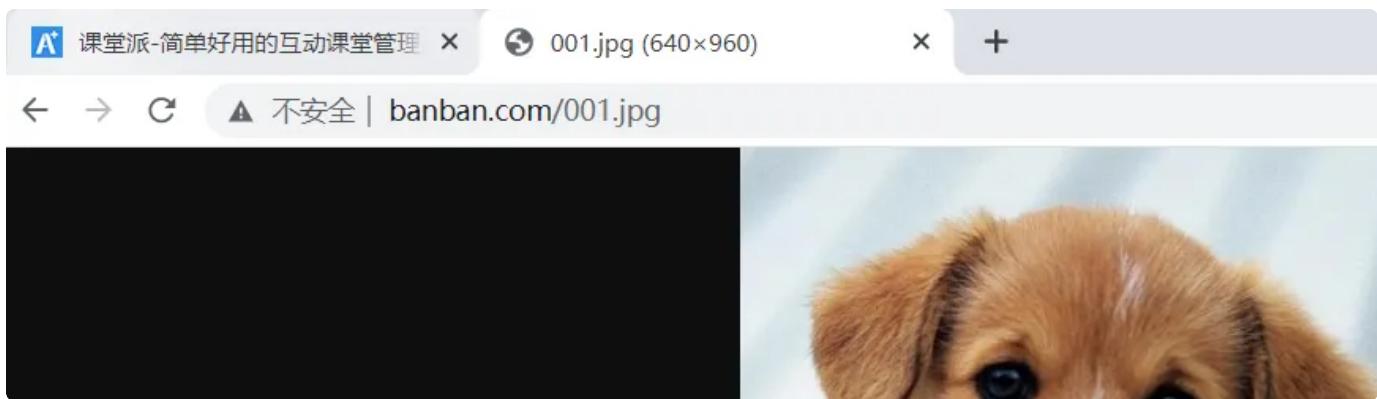


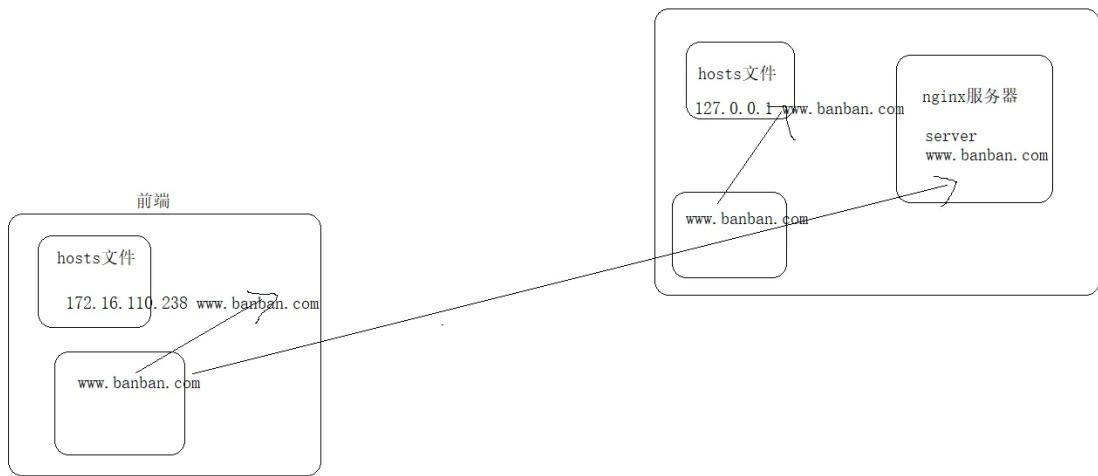
8.1.6 搭建图片服务器



```
34 error_page 500 502 503 504 /50x.html;
35 location = /50x.html {
36     root html;
37 }
38
39
40 }
41
42
43 server {
44
45     listen 80;
46     server_name www.banban.com;
47
48     location / {
49         root E:/image;
50     }
51
52
53 }
54
```

A red arrow points from the image file in the file explorer to the highlighted line of code in the configuration file.





8.2 图片上传

导入jar包

commons-fileupload-1.2.2.jar	2016/5/24 16:57
commons-io-2.4.jar	2016/5/24 16:57

修改jsp

```
<body>
    <!-- 上传图片是需要指定属性 enctype="multipart/form-data" -->
    <!-- <form id="itemForm" action="" method="post" enctype="multipart/form-data"> -->
    <form id="itemForm" class="layui-form" enctype="multipart/form-data"
        action="${pageContext.request.contextPath }/item/updateItem"
        method="post" style="..." align="center">
        <input type="hidden" name="id" value="${item.id }" /> 修改商品信息:

        <div class="layui-form-item">
            <label class="layui-form-label">商品名称:</label>
            <div class="layui-input-block">
                <input type="text" name="name" value="${item.name }"
                    lay-verify="required" placeholder="请输入商品名称" autocomplete="off"
                    class="layui-input" >
            </div>
        </div>
        <div class="layui-form-item">
            <label class="layui-form-label">商品价格:</label>
            <div class="layui-input-block">
                <input type="text" name="price" value="${item.price }" >
            </div>
        </div>
        <div class="layui-form-item">
            <label class="layui-form-label">商品图片:</label>
            <div class="layui-input-block">
                <c:if test="${item.pic !=null}">
                    
                    <br />
                </c:if>
                <input type="file" name="pictureFile" value="${item.pic }"
                    lay-verify="required" placeholder="请输入商品生产时间" autocomplete="off"
                    class="layui-input" >
            </div>
        </div>
        <div class="layui-form-item layui-form-taot" >
```

编写修改的controller方法

Java

```
1 //MultipartFile 封装了上传文件
2 @RequestMapping("/updateItem")
3 public String updateItem(Item item,MultipartFile pictureFile) throws I
OException {
4
5
6     //1.获取原来的名字
7     String originalFilename = pictureFile.getOriginalFilename();
8
9     //2.获取后缀 .jpg
10    String extName=originalFilename.substring(originalFilename.lastIndexOf("."));
11
12    //3.生成新的名字
13    String picName= UUID.randomUUID().toString().replace("-","");
14
15    //4.把图片保存到指定的磁盘中
16    pictureFile.transferTo(new File("E:\\image"+File.separator+picName
));
17
18    //5.保存图片的访问路径
19    item.setPic("http://www.banban.com/"+picName);
20
21    itemService.updateItem(item);
22
23
24    return "redirect:/item/list";
25 }
```

配置

Java

```
1 <!--配置上传的解析器-->
2 <!--id必须设置为multipartResolver-->
3 <bean id="multipartResolver" class="org.springframework.web.multipart.c
ommons.CommonsMultipartResolver">
4     <!--配置上传文件大小-->
5     <property name="maxUploadSize" value="6291456"/>
6 </bean>
```

9.Json数据的交互

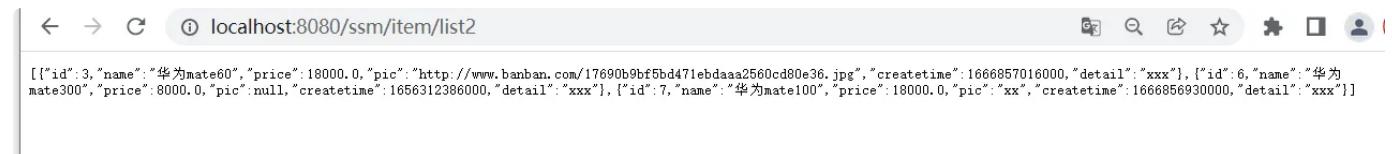
9.1 ResponseBody注解

导包

jackson-annotations-2.4.0.jar	2016/5/24 16:57
jackson-core-2.4.2.jar	2016/5/24 16:57
jackson-databind-2.4.2.jar	2016/5/24 16:58

ResponseBody注解表示这个方法返回json数据

```
1  @RequestMapping(value = "/list2")
2      @ResponseBody //返回json数据
3  -     public List<Item> queryItemList2(){
4          List<Item> list = itemService.findItemList();
5          return list;
6      }
7  }
```



@ResponseBody作用在类上面，代表整个类的所有方法都返回json数据

```
Java |  
1 @Controller  
2 @RequestMapping("/item")  
3 @ResponseBody  
4 public class ItemController {  
5  
6     @RequestMapping(value = "/list2")  
7     public List<Item> queryItemList2(){  
8         List<Item> list = itemService.findItemList();  
9         return list;  
10    }  
11  
12  
13 }
```

9.2 ResponseEntity

```
Java |  
1 @RequestMapping(value = "/list3")  
2 // ResponseEntity 告诉springmvc返回json数据  
3 public ResponseEntity<List<Item>> queryItemList3(){  
4     List<Item> list = itemService.findItemList();  
5     //return ResponseEntity.status(201).body(list);  
6     return ResponseEntity.ok(list);  
7 }
```

9.3 postman使用

The screenshot shows the Postman login page. On the left, there's a "Create an account or sign in" section with a "Create Free Account" button and a "Sign in" link. On the right, there's a section titled "A free Postman account lets" with a bulleted list of benefits:

- ✓ Organize all your API development in workspaces
- ✓ Create public workspaces to collaborate with over 10
- ✓ Back up your work on Postman's cloud
- ✓ Experience the best API development platform for free

The screenshot shows the Postman application interface. At the top, there are navigation tabs: Home, Workspaces (which is the active tab, indicated by a blue background), API Network, and Explore. A search bar on the right contains the placeholder "Search Postman".

The main workspace on the right is titled "afternoon, ddjunshi!". It features a message: "here you left off, catch up with your team's work." Below this is a section titled "a shortcut to sending requests" with a sub-section "n HTTP request to any endpoint." A request configuration panel is visible, showing a URL field with "https://postman-echo.com/get", several tabs (Auth, Headers, Body, Pre-req, Tests, Settings, Cookies) at the top, and a table for "Query Params" below. A prominent orange button labeled "Send" is at the top right of the panel.

On the left side, there is a sidebar titled "bold" which lists "Recently visited" workspaces: "J295", "My Workspace", and "Team Workspace". Below this is a section titled "More workspaces" with the message "No workspaces found". Further down, sections for "Workspaces", "Private", "Integrations", and "Reports" are listed. At the bottom of the sidebar, there is a "Let's get started" section with a "Create workspace" button.

A red box highlights the "Create Workspace" button in the sidebar.

Create workspace

Name

J309

Summary

Add a brief summary about this workspace.

J309

Visibility

Determines who can access this workspace.

Personal

Only you can access

Private

Only invited team members can access

Team

All team members can access

Public

Everyone can view

Create Workspace

Cancel

新建文件夹

The screenshot shows a user interface for managing API collections. On the left, there's a sidebar with icons for APIs, Environments, Mock Servers, Monitors, Flows, and History. The main area displays a collection named "My first collection" which contains two folders: "First folder inside collection" and "Second folder inside collection". Each folder has some request details. Below this, there's a section titled "Create a collection for your requests" with a descriptive text and a "Create collection" button.

Overview

J309

J309

Add a description to explain all about this work...

Activity

Today

ddjunshi created this team workspace just now

新建请求

This screenshot shows the Postman interface. On the left, there's a sidebar with a 'ns' icon, followed by a list of items: 'This collection', 'Add a request', 'Share', 'Move', 'Run collection', 'Edit', 'Add folder', 'Monitor collection', 'Mock collection', 'Create a fork' (with a keyboard shortcut 'Ctrl+Alt+F'), 'Create pull request', 'Merge changes', and 'Pull changes'. The 'Add request' option is highlighted with a red box. On the right, under the heading '测试', there are tabs for 'Authorization', 'Pre-request Script', 'Tests', and 'Variables'. The 'Authorization' tab is selected. It contains a note: 'This authorization method will be used for every request in this collection'. Below this is a 'Type' section with a button labeled 'No Auth'.

测试发送普通表单

This screenshot shows a POST request in Postman. The URL is 'localhost:8080/ssm/item/updateItem2'. The 'Body' tab is selected, showing 'form-data' selected. The request body contains the following parameters:

KEY	VALUE	DESCRIPTION
<input checked="" type="checkbox"/> id	3	
<input checked="" type="checkbox"/> name	华为	
<input checked="" type="checkbox"/> price	9000	
Key	Value	Description

At the bottom, the status bar shows 'Status: 200 OK Time: 16.70 s Size: 459 B Save Response'.

```
1 @RequestMapping(value = "/updateItem2",method = RequestMethod.POST)
2 public String update2(Item item){
3     System.out.println(item);
4     return "success";
5 }
```

9.4 RequestBody注解

接收前端的json数据

```
1 @RequestMapping(value = "/updateItem2",method = RequestMethod.POST)
2 // @RequestBody 前端给后端发送json数据
3 public String update2(@RequestBody Item item){
4     System.out.println(item);
5     return "success";
6 }
```

前端需要在请求头 Content-Type 设置为application/json 告诉后端我现在给你发的是json

POST localhost:8080/ssm/item/updateItem2 Send

Params Authorization Headers (9) Body Pre-request Script Tests Settings Cookie

<input checked="" type="checkbox"/> Content-Length	① 0
<input checked="" type="checkbox"/> Host	① <calculated when request is sent>
<input checked="" type="checkbox"/> User-Agent	① PostmanRuntime/7.29.2
<input checked="" type="checkbox"/> Accept	① */*
<input checked="" type="checkbox"/> Accept-Encoding	① gzip, deflate, br
<input checked="" type="checkbox"/> Connection	① keep-alive
<input checked="" type="checkbox"/> Content-Type	application/json
Key	Value
Description	

Body Cookies (1) Headers (6) Test Results Status: 415 Unsupported Media Type Time: 604 ms Size: 884 B Save Response

Pretty Raw Preview Visualize HTML

```
1 <!doctype html><html lang="zh"><head><title>HTTP状态 415 - 不支持的媒体类型</title><style type="text/css">body {font-family:Tahoma,Arial,
```

测试 / 测试添加商品

POST localhost:8080/ssm/item/updateItem2

Params Authorization Headers (10) Body Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```
1 {"id":3, "name": "华为", "price":9000}
```

Body Cookies (1) Headers (6) Test Results Status: 415 Unsupported Media Type

Pretty Raw Preview Visualize HTML

```
1 <!doctype html><html lang="zh"><head><title>HTTP状态 415 - 不支持的媒体类型</title><style type="text/css">body {font-family:sans-serif; h1, h2, h3, b {color:white;background-color:#525D76;font-size:22px;} h2 {font-size:12px;} a {color:black;} .line {height:1px;background-color:#525D76;border:none;}</style><p>不支持的媒体类型</p><hr class="line" /><p><b>类型</b> 状态报告</p><hr class="line" /><h3>Apache Tomcat/8.5.65</h3></body>
```

Sending request... Cancel