

-

整理自互联网，仅供学习交流。



2018-10-30

[公司名称]

[公司地址]

目录

[一、 SpringMVC 入门 1](#_Toc7204035)

[1.1 简介： 1](#_Toc7204036)

[1.1 1.2 SpringMVC的原理 1](#_Toc7204037)

[1.2 1.3 Spring入门案例 1](#_Toc7204038)

[1.2.1 第一步：导入springmvc jar包 1](#_Toc7204039)

[1.2.2 第二步：在web.xml配置DispatcherServlet 1](#_Toc7204040)

[1.2.3 第三步：在WEB-INF目录下创建DispatcherServlet-servlet.xml 2](#_Toc7204041)

[1.2.4 第四步：配置DispatcherServlet-servlet.xml 2](#_Toc7204042)

[1.2.5 第五步：创建个Controller控制器 2](#_Toc7204043)

[1.2.6 第六步：创建一个userList.jsp页面 2](#_Toc7204044)

[1.2.7 第七步：运行项目，访问路径 3](#_Toc7204045)

[1.3 1.4 Spring的执行流程 3](#_Toc7204046)

[1.3.1 流程图 3](#_Toc7204047)

[1.3.2 控制器处理适配器源码 3](#_Toc7204048)

[1.4 URL处理器映射【了解】 4](#_Toc7204049)

[1.4.1 BeanNameUrlHandlerMapping 4](#_Toc7204050)

[1.4.2 SimpleUrlHandlerMapping 4](#_Toc7204051)

[1.4.3 ControllerClassNameHandlerMapping 5](#_Toc7204052)

[1.5 1.6 处理器适配器 5](#_Toc7204053)

[1.5.1 SimpleControllerHandlerAdapter 5](#_Toc7204054)

[1.5.2 HttpRequestHandlerAdapter 5](#_Toc7204055)

[1.6 命令控制器【了解】 6](#_Toc7204056)

[1.6.1 添加页面 6](#_Toc7204057)

[1.6.2 Command控制器 7](#_Toc7204058)

[1.6.3 Info.jsp 7](#_Toc7204059)

[1.6.4 views结构 7](#_Toc7204060)

[1.6.5 DispatchServelt-sevlet配置文件 8](#_Toc7204061)

[1.6.6 测试 8](#_Toc7204062)

[1.6.7 乱码问题 8](#_Toc7204063)

[1.6.8 日期类型转换 9](#_Toc7204064)

[二、 SpringMVC的注解开发 9](#_Toc7204065)

[2.1 2.1 注解开发入门案例 9](#_Toc7204066)

[2.1.1 修改springmvc的 配置文件 9](#_Toc7204067)

[2.1.2 把UserController修改下 10](#_Toc7204068)

[2.1.3 修改Tomcat运行的jdk版本 10](#_Toc7204069)

[2.1.4 测试注解配置的访问 11](#_Toc7204070)

[2.1.5 MVC:annotation-driven 11](#_Toc7204071)

[2.2 2.2 RequestMapping讲解 11](#_Toc7204072)

[2.2.1 配置请求路径映射路径 12](#_Toc7204073)

[2.2.2 自定义根路径 12](#_Toc7204074)

[2.3 2.3接收请求参数 13](#_Toc7204075)

[2.3.1 封装参数分析 13](#_Toc7204076)

[2.3.2 接收int,String,Date，数组类型 13](#_Toc7204077)

[2.3.3 接收pojo类型 13](#_Toc7204078)

[2.3.4 接收包装类型参数 14](#_Toc7204079)

[2.3.5 接收集合List类型参数 15](#_Toc7204080)

[2.3.6 接收集合Map类型参数 15](#_Toc7204081)

[2.4 Spring与Struts的区别【面试题】 16](#_Toc7204082)

[2.4.1 实现机制： 16](#_Toc7204083)

[2.4.2 运行速度： 16](#_Toc7204084)

[2.4.3 参数封装来分析： 16](#_Toc7204085)

[三、 页面回显 17](#_Toc7204086)

[3.1 配置一个列表方法 17](#_Toc7204087)

[3.1 显示列表数据 17](#_Toc7204088)

[3.2 edit方法 18](#_Toc7204089)

[3.3 编辑页面 18](#_Toc7204090)

[3.4 效果 18](#_Toc7204091)

[四、 URL模版映射 19](#_Toc7204092)

[4.1 体验 19](#_Toc7204093)

[4.2 在web.xml中配置rest路径 19](#_Toc7204094)

[五、 转发和重定 20](#_Toc7204095)

[5.1 转发 20](#_Toc7204096)

[5.1.1 转发到同一个控制器的方法 21](#_Toc7204097)

[5.1.2 转发到不同一个控制器的方法 21](#_Toc7204098)

[5.2 重定向 22](#_Toc7204099)

[5.2.1 重定向，只需要把forward改成redirect 22](#_Toc7204100)

[5.2.2 关于重定向RedirectAttributes的用法 22](#_Toc7204101)

[六、 RequestParam 23](#_Toc7204102)

[七、 ResponseBody和RequestBody 23](#_Toc7204103)

[7.1 简介 23](#_Toc7204104)

[7.2 1.2 使用 24](#_Toc7204105)

[7.2.1 案例:请求和响应都是json数据 24](#_Toc7204106)

[7.2.2 案例:只有响应json数据 25](#_Toc7204107)

[八、 Springmvc多视图 26](#_Toc7204108)

[8.1 第一步：导入xml格式支持的jar包 26](#_Toc7204109)

[8.2 第二步：配置支持多视图 26](#_Toc7204110)

[8.3 第三步：控制器提供一个接口 27](#_Toc7204111)

[8.4 第四步：测试 27](#_Toc7204112)

[九、 eclipse安装spring提示插件 27](#_Toc7204113)

[9.1 3.1 在应用市场搜索spring tools 27](#_Toc7204114)

[9.2 3.2 依次点开菜单 ‘XML -> XMLFiles -> Editor -> Content Assist’配置 28](#_Toc7204115)

[9.3 3.3 使用下面两种文件打开配置文件 29](#_Toc7204116)

[十、 SSM整合 30](#_Toc7204117)

[10.1 导入jar包 30](#_Toc7204118)

[10.2 创建一个springmvc文件 30](#_Toc7204119)

[10.3 c：在web.xml添加springmvc配置 31](#_Toc7204120)

[10.4 d：先配置一个Controller跑出一个页面 31](#_Toc7204121)

[10.5 e：通过MyBatis的逆向工程生成JavaBean/Mapper 32](#_Toc7204122)

[10.6 f：修改ItemsMapper.java和ItemsMapper.xml 32](#_Toc7204123)

[10.7 g：定义Service层接口并实现 32](#_Toc7204124)

[10.8 h：配置SqlMappingConfig.xml、mybatis 33](#_Toc7204125)

[10.9 i：创建spring的applicaiontContext.xml 33](#_Toc7204126)

[10.10 j：配置c3p0数据源和mybatis的会话工厂 34](#_Toc7204127)

[10.11 k：Web.xml配置spring容器 35](#_Toc7204128)

[10.12 l：添加bean的注解装配 35](#_Toc7204129)

[10.13 m：ItemsService 35](#_Toc7204130)

[10.14 n：ItemsController 35](#_Toc7204131)

[10.15 o：事务配置 36](#_Toc7204132)

[10.16 o：添加一个保存方法测试事务 37](#_Toc7204133)

[十一、 增删改查任务 37](#_Toc7204134)

[11.1 显示商品数据 38](#_Toc7204135)

[11.2 删除商品 38](#_Toc7204136)

[11.3 显示编辑商品页面 39](#_Toc7204137)

[11.4 更新商品 40](#_Toc7204138)

[十二、 文件上传 41](#_Toc7204139)

[12.1 导包 41](#_Toc7204140)

[12.2 Springmvc.xml配置支持文件上传 41](#_Toc7204141)

[12.3 Html/JS 42](#_Toc7204142)

[12.4 后台 42](#_Toc7204143)

[十三、 Oscache页面缓存 43](#_Toc7204144)

[13.1 7.1页面缓存测试 43](#_Toc7204145)

[13.1.1 导包 43](#_Toc7204146)

[13.1.2 测试 43](#_Toc7204147)

[13.1.3 知识点： 44](#_Toc7204148)

[十四、 Oscache整合ssm项目 45](#_Toc7204149)

[在web.xml中配置页面缓存 45](#_Toc7204150)

[十五、 Springmvc整合freemarker 46](#_Toc7204151)

[15.1 简介：freemark 46](#_Toc7204152)

[15.2 导包 46](#_Toc7204153)

[15.3 Springmvc配置freemark 46](#_Toc7204154)

[15.4 写一个hello.ftl模板 46](#_Toc7204155)

[15.5 ItemsController 47](#_Toc7204156)

[15.6 测试结果 47](#_Toc7204157)

[15.7 list.jsp改成list.ftl 47](#_Toc7204158)

[十六、 SpringMVC的拦截器 48](#_Toc7204159)

[16.1 登录拦截器 48](#_Toc7204160)

[16.2 登录控制器 49](#_Toc7204161)

[16.3 登录界面 49](#_Toc7204162)

[16.4 登录拦截器配置 49](#_Toc7204163)

[十七、 常见问题 50](#_Toc7204164)

[17.1 SpringMVC访问静态资源的三种方式 50](#_Toc7204165)

[17.1.1 方案一：激活Tomcat的defaultServlet来处理静态文件 50](#_Toc7204166)

[17.1.2 方案二： 在spring3.0.4以后版本提供了mvc:resources 50](#_Toc7204167)

[17.1.3 方案三 ，使用<mvc:default-servlet-handler/> 51](#_Toc7204168)

123

# 我的

常见问题

## 常见问题

### SpringMVC访问静态资源的三种方式

如何你的DispatcherServlet拦截 \*.do这样的URL，就不存在访问不到静态资源的问题。如果你的DispatcherServlet拦截“/”，拦截了所有的请求，同时对\*.js,\*.jpg的访问也就被拦截了。

 目的：可以正常访问静态文件，不要找不到静态文件报404。

#### 方案一：激活Tomcat的defaultServlet来处理静态文件

<servlet-mapping>

    <servlet-name>default</servlet-name>

    <url-pattern>\*.jpg</url-pattern>

</servlet-mapping>

<servlet-mapping>

    <servlet-name>default</servlet-name>

    <url-pattern>\*.js</url-pattern>

</servlet-mapping>

<servlet-mapping>

    <servlet-name>default</servlet-name>

    <url-pattern>\*.css</url-pattern>

</servlet-mapping>

要配置多个，每种文件配置一个

要写在DispatcherServlet的前面， 让defaultServlet先拦截，这个就不会进入Spring了，我想性能是最好的吧。

Tomcat, Jetty, JBoss, and GlassFish  默认 Servlet的名字 -- "default"

Google App Engine 默认 Servlet的名字 -- "\_ah\_default"

Resin 默认 Servlet的名字 -- "resin-file"

WebLogic 默认 Servlet的名字  -- "FileServlet"

WebSphere  默认 Servlet的名字 -- "SimpleFileServlet"

#### 方案二： 在spring3.0.4以后版本提供了mvc:resources

<mvc:resources 的使用方法：

<!--对静态资源文件的访问-->

<mvc:resources mapping="/images/\*\*" location="/images/" />

  /images /\*\*映射到 ResourceHttpRequestHandler 进行处理，location指定静态资源的位置.可以是web application根目录下、jar包里面，这样可以把静态资源压缩到jar包中。cache-period可以使得静态资源进行web cache

如果出现下面的错误，可能是没有配置 <mvc:annotation-driven /> 的原因。

报错WARNING: No mapping found for HTTP request with URI [/mvc/user/findUser/lisi/770] in DispatcherServlet with name 'springMVC'

使用 <mvc:resources/> 元素,把 mapping 的 URI 注册到 SimpleUrlHandlerMapping的urlMap 中,

key 为 mapping 的 URI pattern值,而 value为 ResourceHttpRequestHandler,

这样就巧妙的把对静态资源的访问由 HandlerMapping 转到 ResourceHttpRequestHandler 处理并返回,所以就支持 classpath 目录, jar 包内静态资源的访问.

另外需要注意的一点是,不要对 SimpleUrlHandlerMapping 设置 defaultHandler. 因为对 static uri 的 defaultHandler 就是ResourceHttpRequestHandler,

否则无法处理static resources request.

#### 方案三 ，使用<mvc:default-servlet-handler/>

<mvc:default-servlet-handler/>

会把 "/\*\*" url,注册到 SimpleUrlHandlerMapping 的 urlMap 中,把对静态资源的访问由 HandlerMapping 转到 org.springframework.web.servlet.resource.DefaultServletHttpRequestHandler 处理并返回.

DefaultServletHttpRequestHandler 使用就是各个 Servlet 容器自己的默认 Servlet.

补充说明：多个HandlerMapping的执行顺序问题：

DefaultAnnotationHandlerMapping 的 order 属性值是：0

<mvc:resources/ >自动注册的 SimpleUrlHandlerMapping 的 order 属性值是： 2147483646

<mvc:default-servlet-handler/>自动注册的 SimpleUrlHandlerMapping 的 order 属性值是：2147483647

spring 会先执行 order 值比较小的。当访问一个 a.jpg 图片文件时，先通过 DefaultAnnotationHandlerMapping 来找处理器，一定是找不到的，我们没有叫 a.jpg 的 Action。再按 order 值升序找，由于最后一个 SimpleUrlHandlerMapping 是匹配 "/\*\*" 的，所以一定会匹配上，再响应图片。

访问一个图片，还要走层层匹配。真不知性能如何？改天做一下压力测试，与Apache比一比。

最后再说明一下，如何你的 DispatcherServlet 拦截 \*.do 这样的 URL，就不存上述问题了。

### [<mvc:default-servlet-handler/> 导致 Controller失效](https://ynp.iteye.com/blog/1791944)

<!-- 自动扫描该包，使SpringMVC认为包下用了@controller注解的类是控制器 -->

<context:component-scan base-package="com.yaosiyuan.controller" />

<!-- 扩充了注解驱动，可以将请求参数绑定到控制器参数 -->

<mvc:annotation-driven/>

<!--SpringMVC访问静态资源-->

<mvc:default-servlet-handler />

# 18黑马

## SpringMVC 入门

### 简介：

MVC

M:Model

V:View

C:Controller - servlet/action/controller

Spring MVC是Spring提供的一个强大而灵活的web框架。借助于注解，Spring MVC提供了几乎是POJO的开发模式，使得控制器的开发和测试更加简单。这些控制器一般不直接处理请求，而是将其委托给Spring上下文中的其他bean，通过Spring的依赖注入功能，这些bean被注入到控制器中。

Spring MVC主要由DispatcherServlet、处理器映射【找控制器】、适配器【调用控制器的方法】、控制器【业务】、视图解析器、视图组成。

### SpringMVC的原理

### Spring入门案例

#### 第一步：导入springmvc jar包

暂时先导入下面的jar包

|  |
| --- |
|  |

#### 第二步：在web.xml配置DispatcherServlet

|  |
| --- |
| <servlet>  <servlet-name>DispatcherServlet</servlet-name>  <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>  <load-on-startup>1</load-on-startup>  </servlet>  <servlet-mapping>  <servlet-name>DispatcherServlet</servlet-name>  <url-pattern>\*.do</url-pattern>  </servlet-mapping> |

#### 第三步：在WEB-INF目录下创建DispatcherServlet-servlet.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:mvc=*"http://www.springframework.org/schema/mvc"*  xmlns:context=*"http://www.springframework.org/schema/context"*  xmlns:aop=*"http://www.springframework.org/schema/aop"* xmlns:tx=*"http://www.springframework.org/schema/tx"*  xsi:schemaLocation=*"http://www.springframework.org/schema/beans*  *http://www.springframework.org/schema/beans/spring-beans-3.2.xsd*  *http://www.springframework.org/schema/mvc*  *http://www.springframework.org/schema/mvc/spring-mvc-3.2.xsd*  *http://www.springframework.org/schema/context*  *http://www.springframework.org/schema/context/spring-context-3.2.xsd*  *http://www.springframework.org/schema/aop*  *http://www.springframework.org/schema/aop/spring-aop-3.2.xsd*  *http://www.springframework.org/schema/tx*  *http://www.springframework.org/schema/tx/spring-tx-3.2.xsd"*>    </beans> |

#### 第四步：配置DispatcherServlet-servlet.xml

|  |
| --- |
| <!-- 1. 配置处理器映射，springmvc默认的处理器映射器  BeanNameUrlHandlerMapping:根据bean的name属性的url去找handlerController -->  <bean class=*"org.springframework.web.servlet.handler.BeanNameUrlHandlerMapping"*/>  <!-- 2. 配置处理器适配置器执行Controller -->  <bean class=*"org.springframework.web.servlet.mvc.SimpleControllerHandlerAdapter"*/>  <!-- 3.配置一个控制器 -->  <bean name=*"/user.do"* class=*"com.gyf.backoffice.web.controller.UserController"*/>  <!-- 4.配置springmvc视图解析器  视图解析器解析的视频路径为：前缀 + 后缀 -->  <bean class=*"org.springframework.web.servlet.view.InternalResourceViewResolver"*>  <property name=*"prefix"* value=*"/WEB-INF/views"*/>  <property name=*"suffix"* value=*".jsp"*/>  </bean> |

#### 第五步：创建个Controller控制器

|  |
| --- |
|  |

#### 第六步：创建一个userList.jsp页面

|  |
| --- |
|  |

#### 第七步：运行项目，访问路径

|  |
| --- |
|  |

### 1.4 Spring的执行流程

#### 流程图

|  |
| --- |
|  |

#### 控制器处理适配器源码

**SimpleControllerHandlerAdapter**

|  |
| --- |
|  |

### URL处理器映射【了解】

#### BeanNameUrlHandlerMapping

功能：寻找Controller

根据url请求去匹配bean的name属性，从而获取Controller

|  |
| --- |
|  |

#### SimpleUrlHandlerMapping

功能：寻找Controller

根据浏览器url匹配简单url的key，key就是Controller的id找到Controller

|  |
| --- |
|  |

测试路径：

[http://localhost:8080/springmvc-demo01/user1.do](http://localhost:8080/springmvc-demo01/user3.do)

[http://localhost:8080/springmvc-demo01/user2.do](http://localhost:8080/springmvc-demo01/user3.do)

http://localhost:8080/springmvc-demo01/user3.do

#### ControllerClassNameHandlerMapping

功能：寻找Controller

根据类名（MyController）类名.do来访问,类名首字母小写

|  |
| --- |
|  |

测试

|  |
| --- |
|  |

### 1.6 处理器适配器

#### SimpleControllerHandlerAdapter

功能：执行controller

调用controller里面方法，返回modelAndView

|  |
| --- |
|  |

#### HttpRequestHandlerAdapter

执行控制器

负责调用实现HttpRequestHandler接口的控制器

|  |
| --- |
|  |
|  |
| http://localhost:8080/springmvc-demo01/user2.do |

注意：两个处理器适配器可以共存

### 命令控制器【了解】

Springmvc通过命令设计模式接受页面参数

#### 添加页面

|  |
| --- |
| <form action="${pageContext.request.contextPath}/command.do" method="post">  用户名:<input type="text" name="username"><br>  密码:<input type="text" name="password"><br>  性别:<input type="text" name="gender"><br>  生日:<input type="text" name="birthday"><br>  <input type="submit">  </form> |
|  |

#### Command控制器

|  |
| --- |
|  |

#### Info.jsp

|  |
| --- |
|  |

#### views结构

|  |
| --- |
|  |

#### DispatchServelt-sevlet配置文件

|  |
| --- |
|  |
| 如果bean中配置name,都要加上/斜杠 |

#### 测试

|  |
| --- |
|  |
| **默认日期格式只能2017/10/11格式，其它格式接收参数会为空** |
|  |
|  |

#### 乱码问题

POST请求乱码

|  |
| --- |
| <!-- 配置编码过滤器 -->  <filter>  <filter-name>EncodingFilter</filter-name>  <filter-class>org.springframework.web.filter.CharacterEncodingFilter</filter-class>  <init-param>  <param-name>encoding</param-name>  <param-value>UTF-8</param-value>  </init-param>  </filter>    <filter-mapping>  <filter-name>EncodingFilter</filter-name>  <url-pattern>/\*</url-pattern>  </filter-mapping> |

GET请求乱码

Tomcat8 默认进行了url编码，get请求不乱码，Tomcat7会乱码

|  |
| --- |
|  |

#### 日期类型转换

springmvc默认接收的日期格式为yyyy/MM/dd,如果添加其它格式，只要重写一个方法，如图

|  |
| --- |
|  |

## SpringMVC的注解开发

### 2.1 注解开发入门案例

#### 修改springmvc的 配置文件

把前面的项目复制一份，修改一个DispatchServlet-Servlet.xml配置文件

|  |
| --- |
| <!-- 1.配置注解扫描位置 -->  <context:component-scan base-package=*"com.gyf.backoffice.web.controller"*/>    <!-- 2. 配置处理器映射，通过注解来查找 -->  <bean class=*"org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerMapping"*/>  <!-- 3.配置注解处理适配器来执行控制器的方法 -->  <bean class=*"org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerAdapter"*/>    <!-- 配置springmvc视图解析器  视图解析器解析的视频路径为：前缀 + 后缀 -->  <bean class=*"org.springframework.web.servlet.view.InternalResourceViewResolver"*>  <property name=*"prefix"* value=*"/WEB-INF/views"*/>  <property name=*"suffix"* value=*".jsp"*/>  </bean> |

#### 把UserController修改下

|  |
| --- |
|  |

#### 修改Tomcat运行的jdk版本

**因为3.2的springmvc注解与jdk1.8不兼容，跑tomat时会报错，如图**

|  |
| --- |
|  |

解决办法1：升级spring到4.0的版本

解决办法2：修改Tomcat运行的jdk版本

|  |
| --- |
|  |

#### 测试注解配置的访问

|  |
| --- |
|  |

#### MVC:annotation-driven

|  |
| --- |
| **代化mvc配置文件** |
|  |

### 2.2 RequestMapping讲解

#### 配置请求路径映射路径

写法：

|  |
| --- |
| **@RequestMapping(“list”)**  **@RequestMapping(“/list.do”)**  **@RequestMapping(value=”/list.do”)**  **@RequestMapping(value = "/list3",method=RequestMethod.POST) 只能使用POST方法**  **@RequestMapping(value = "/list3",method=RequestMethod.Get) 只能使用GET方法** |
|  |

#### 自定义根路径

可以控制器上添加一个RequestMapping,如图

|  |
| --- |
|  |
|  |

### 2.3接收请求参数

#### 封装参数分析

参数类型有：基本类型，int，String等等基本类型, Pojo类

**Struts2参数：基于属性封装,在action会添加属性，提供set方法。**

**Springmvc参数封装：基于方法进行封装。**

#### 接收int,String,Date，数组类型

|  |
| --- |
| 在views/user中提供一个form表单 |
|  |
| 在UserController提供一个用户注册的方法 |
|  |

#### 接收pojo类型

|  |
| --- |
| 创建一个pojo User类型 |
|  |
| 配置一个注册方法 |
|  |
| springmvc接收参数时不需要像struts那样，在表单标签添加**user.**username这第一个东西 |
|  |

#### 接收包装类型参数

|  |
| --- |
| 把User写成一个类的属性 |
|  |
| 添加一个注册方法 |
|  |
| 修改form表单 |
|  |

#### 接收集合List类型参数

|  |
| --- |
| 在User扩展中添加一个集合 |
|  |
| 添加一个注册 的方法，参数写UserExt |
|  |
| 表单 |
|  |

#### 接收集合Map类型参数

|  |
| --- |
| UserExt上添加一个Map |
|  |
| 添加一个注册方法 |
|  |
| 表单 |
|  |

### Spring与Struts的区别【面试题】

#### 实现机制：

Struts2是基于过滤器实现的。

Springmvc基于servlet实现。

#### 运行速度：

Servlet比过滤器快。

Struts2是多例

每一次请求，都会创建一个Action对象

请求来了以后，struts2创建多少个对象：ActionContext，valuestack，UAction，ActionSuport，ModelDriven

Springmvc是单例。

同一个Controller请求，只会创建一个Controller

#### 参数封装来分析：

Struts基于属性进行封装,Action有参数属性。

Springmvc基于方法封装,参数是写在Controller的方法。

## 页面回显

### 3.1 配置一个列表方法

|  |
| --- |
|  |

### 显示列表数据

|  |
| --- |
|  |
|  |

### edit方法

|  |
| --- |
|  |

### 编辑页面

|  |
| --- |
|  |

### 效果

|  |  |
| --- | --- |
|  |  |

## URL模版映射

**url模版映射可以restfull软件架构**。

### 体验

|  |
| --- |
| 修改url格式，如图 |
|  |
| 配置接收url模版映射  **{}:匹配接受页面Url路径参数**  **@Pathariable：{}里面参数注入后面参数里面** |
|  |

### 在web.xml中配置rest路径

|  |
| --- |
| Web.xml |
|  |
| 测试 |
|  |

## 转发和重定

return "forward:index.jsp"; //转发

return "forward:user.do?method=reg5"; //转发

return new ModelAndView("/toList");//转发

return "redirect:user.do?method=reg5"; //重定向

return "redirect:http://www.baidu.com"; //重定向

return new ModelAndView("redirect:/toList"); //重定向

### 转发

Spring MVC中forward请求转发2种方式（带参数）

Spring MVC中forward请求转发只需要请求一次服务器，可以提高访问速度，同时参数传递可以借助HttpServletRequest来处理，由于是处理同一个请求，所以request对象还是同一个，中文也不会出现乱码现象。

ViewResolver请求转发

@RequestMapping(value="/testredirect",method = { RequestMethod.POST, RequestMethod.GET })

public String testredirect(HttpServletResponse response){

return "forward:/index";

}

带参数

@RequestMapping(value="/testredirect",method = { RequestMethod.POST, RequestMethod.GET })

public String testredirect(HttpServletRequest request){

request.setAttribute("username", "51gjie"); //把username参数传递到request中

return "forward:/user/index";

}

ModelAndView请求转发

@RequestMapping(value="/restredirect",method = { RequestMethod.POST, RequestMethod.GET })

public ModelAndView restredirect(String userName){

ModelAndView model = new ModelAndView("forward:/main/index");//默认forward，可以不用写

return model;

}

带参数

@RequestMapping(value="/toredirect",method = { RequestMethod.POST, RequestMethod.GET })

public ModelAndView toredirect(String userName){

ModelAndView model = new ModelAndView("/user/userinfo");

model.addObject("userName", userName); //把userName参数带入到controller的RedirectAttributes

return model;

}

forward请求转发流程：客户首先发送一个请求到服务器端，服务器端发现匹配的servlet，并指定它去执行，当这个servlet执行完之后，它要调用getRequestDispacther()方法，把请求转发给指定的test.jsp,整个流程都是在服务器端完成的，而且是在同一个请求里面完的，因此servlet和jsp共享的是同一个request，在servlet里面放的所有东西，在jsp中都能取出来，因此，jsp能把结果getAttribute()出来。整个过程是一个请求，一个响应。

#### 转发到同一个控制器的方法

|  |
| --- |
|  |

#### 转发到不同一个控制器的方法

|  |
| --- |
|  |

### 重定向

#### 重定向，只需要把forward改成redirect

|  |
| --- |
|  |

#### 关于重定向RedirectAttributes的用法

RedirectAttributes 是Spring mvc 3.1版本之后出来的一个功能，专门用于重定向之后还能带参数跳转的的工具类

它有两种带参的方式：

##### 第一种：

redirectAttributes.addAttributie("prama",value); 这种方法相当于在重定向链接地址追加传递的参数，例如:

redirectAttributes.addAttributie("prama1",value1);

redirectAttributes.addAttributie("prama2",value2);

return:"redirect：/path/list"

以上重定向的方法等同于 return:"redirect：/path/list？prama1=value1&prama2=value2 " ，注意这种方法直接将传递的参数暴露在链接地址上，非常的不安全，慎用。

第二种：

redirectAttributes.addFlashAttributie("prama",value); 这种方法是隐藏了参数，链接地址上不直接暴露，但是能且只能在重定向的 “页面” 获取prama参数值。

其原理就是放到session中，session在跳到页面后马上移除对象。如果是重定向一个controller中是获取不到该prama属性值的。除非在controller中用(@RequestPrama(value = "prama")String prama)注解，采用传参的方式。

页面获值例如：

redirectAttributes.addFlashAttributie("prama1",value1);

redirectAttributes.addFlashAttributie("prama2",value2);

return:"redirect：/path/list.jsp"

在以上参数均可在list.jsp页面使用EL表达式获取到参数值${prama\*}

controller获得redirectAttributes重定向的值例如：

redirectAttributes.addFlashAttributie("prama1",value1);

redirectAttributes.addFlashAttributie("prama2",value2);

return:"redirect：/path/list/"

@RequestMapping("list")

public List<Student> list(@RequestPrama(value = "prama1")String prama1,

@RequestPrama(value = "prama2")String prama2,...

){

//TODO

//your code

}

## RequestParam

/\*\*

\* RequestParam参数描述

\* value:参数名称

\* defaultValue：默认值

\* required:参数是否必须有值，如果为true,参数又为空，会报错

\* \*/

|  |
| --- |
|  |

## ResponseBody和RequestBody

### 简介

@ResponseBody把**后台pojo转换json对象**，返回到页面。

@RequestBody接受**前台json数据，把json数据自动封装javaBean**

1.1 @responseBody注解的使用

1、

@responseBody注解的作用是将controller的方法返回的对象通过适当的转换器转换为指定的格式之后，写入到response对象的body区，通常用来返回JSON数据或者是XML

数据，需要注意的呢，在使用此注解之后不会再走试图处理器，而是直接将数据写入到输入流中，他的效果等同于通过response对象输出指定格式的数据。

2、

@RequestMapping("/login")

@ResponseBody

public User login(User user){

return user;

}

User字段：userName pwd

那么在前台接收到的数据为：'{"userName":"xxx","pwd":"xxx"}'

效果等同于如下代码：

@RequestMapping("/login")

public void login(User user, HttpServletResponse response){

response.getWriter.write(JSONObject.fromObject(user).toString());

}

### 1.2 使用

#### 案例:请求和响应都是json数据

##### 第一步：导入json的jar

|  |
| --- |
|  |

##### 第二步：

添加个json转换器

|  |
| --- |
| 配置文件 |

##### 第三步：写个jsp页面，提交json请求参数

|  |
| --- |
|  |
|  |

##### 第四步：后台处理

**注意学生类一定要提供无参的构造方法**

|  |
| --- |
|  |
|  |

#### 案例:只有响应json数据

##### 第一步：表单

|  |
| --- |
|  |

##### 第二步：后台处理

|  |
| --- |
|  |

## Springmvc多视图

多视图是一个方法可以返回json/xml等格式的数据

### 第一步：导入xml格式支持的jar包



### 第二步：配置支持多视图

|  |
| --- |
| <bean  class="org.springframework.web.servlet.view.ContentNegotiatingViewResolver">  <!-- 配置支持媒体类型 -->  <property name="contentNegotiationManager">  <bean  class="org.springframework.web.accept.ContentNegotiationManagerFactoryBean">  <property name="mediaTypes">  <map>  <entry key="json" value="application/json"></entry>  <entry key="xml" value="application/xml"></entry>  </map>  </property>  </bean>  </property>  <!-- 指定默认视图 -->  <property name="defaultViews">  <!-- 支持多个视图 -->  <list>  <!-- 对josn格式视图支持 -->  <bean class="org.springframework.web.servlet.view.json.MappingJacksonJsonView"/>  <!-- xml格式视图支持 -->  <bean class="org.springframework.web.servlet.view.xml.MarshallingView">  <constructor-arg>  <bean class="org.springframework.oxm.jaxb.Jaxb2Marshaller">  <property name="classesToBeBound">  <list>  <value>com.gyf.backoffice.domain.Student</value>  </list>  </property>  </bean>  </constructor-arg>  </bean>  </list>  </property>  </bean> |

### 第三步：控制器提供一个接口

注意学生对象需要声明一个注解

|  |  |
| --- | --- |
|  |  |

### 第四步：测试

|  |
| --- |
|  |
|  |

## eclipse安装spring提示插件

### 3.1 在应用市场搜索spring tools

点击Eclipse的Help 下的 Eclipse Marketplace搜索spring tools,然后红色不用勾选，点下一步安装插件

|  |  |
| --- | --- |
|  |  |

### 3.2 依次点开菜单 ‘XML -> XMLFiles -> Editor -> Content Assist’配置

如下图：

|  |
| --- |
|  |

### 3.3 使用下面两种文件打开配置文件

在xml中使用. + /就会有类提示

|  |
| --- |
|  |

## SSM整合

Spring + SpringMVC + Mybatis整合

### 导入jar包

|  |
| --- |
| * **Spring+ SpringMVC + MyBatis + Mybatis-spring整合包** * **AOP联盟+织入 + c3p0 数据库连接池 + MySQL连接驱动 + jstl** |
|  |

### 创建一个springmvc文件

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <beans xmlns="http://www.springframework.org/schema/beans"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:mvc="http://www.springframework.org/schema/mvc"  xmlns:context="http://www.springframework.org/schema/context"  xmlns:aop="http://www.springframework.org/schema/aop" xmlns:tx="http://www.springframework.org/schema/tx"  xsi:schemaLocation="http://www.springframework.org/schema/beans  http://www.springframework.org/schema/beans/spring-beans-3.2.xsd  http://www.springframework.org/schema/mvc  http://www.springframework.org/schema/mvc/spring-mvc-3.2.xsd  http://www.springframework.org/schema/context  http://www.springframework.org/schema/context/spring-context-3.2.xsd  http://www.springframework.org/schema/aop  http://www.springframework.org/schema/aop/spring-aop-3.2.xsd  http://www.springframework.org/schema/tx  http://www.springframework.org/schema/tx/spring-tx-3.2.xsd">  <!-- 1.配置注解扫描位置 -->  <context:component-scan base-package="com.gyf.backoffice.web.controller" />  <!-- 2.配置注解处理映射-->  <bean class="org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerMapping"/>  <!--3.配置适配器-->  <bean class="org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerAdapter">  </bean>  <!-- 4.配置springmvc视图解析器 视图解析器解析的视频路径为：前缀 + 后缀 -->  <bean  class="org.springframework.web.servlet.view.InternalResourceViewResolver">  <property name="prefix" value="/WEB-INF/views" />  <property name="suffix" value=".jsp" />  </bean>  </beans> |

### c：在web.xml添加springmvc配置

这次我们更改默认的springmvc配置文件路径

|  |
| --- |
|  |
| <servlet>  <servlet-name>DispatcherServlet</servlet-name>  <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>  <!-- 3.0的springmvc 默认加载WEB-INF下的dispatcher-servlet.xml文件 3.2的springmvc  加载DispatcherServlet-servlet.xml文件 -->  <init-param>  <!-- 修改黑底springmvc加载的配置文件路径 -->  <param-name>contextConfigLocation</param-name>  <param-value>classpath:springmvc.xml</param-value>  </init-param>  <load-on-startup>1</load-on-startup>  </servlet>  <servlet-mapping>  <servlet-name>DispatcherServlet</servlet-name>  <url-pattern>\*.do</url-pattern>  </servlet-mapping> |

### d：先配置一个Controller跑出一个页面

|  |  |
| --- | --- |
|  |  |

### e：通过MyBatis的逆向工程生成JavaBean/Mapper

把生成的文件导入当到项目，**步骤不记得可以参数MyBatis的教案**

|  |
| --- |
|  |

### f：修改ItemsMapper.java和ItemsMapper.xml

|  |
| --- |
|  |
|  |

### g：定义Service层接口并实现

|  |
| --- |
|  |
|  |

### h：配置SqlMappingConfig.xml、mybatis

|  |
| --- |
| <?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>  <!-- 别名配置 -->  <typeAliases>  <!-- 批量配置别名：指定批量定义别名的类包，别名为类名（首字母大小写都可） -->  <package name=*"com.gyf.backoffice.domain"*/>  </typeAliases>  <mappers>  <!-- 批量加载映射文件 -->  <package name=*"com.gyf.backoffice.mapper"*/>  </mappers>  </configuration> |

### i：创建spring的applicaiontContext.xml

|  |
| --- |
| 配置数据源和mybaties的session工厂  <?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:context=*"http://www.springframework.org/schema/context"*  xmlns:aop=*"http://www.springframework.org/schema/aop"*  xmlns:tx=*"http://www.springframework.org/schema/tx"*  xsi:schemaLocation=*"http://www.springframework.org/schema/beans*  *http://www.springframework.org/schema/beans/spring-beans-3.2.xsd*  *http://www.springframework.org/schema/context*  *http://www.springframework.org/schema/context/spring-context-3.2.xsd*  *http://www.springframework.org/schema/aop*  *http://www.springframework.org/schema/aop/spring-aop-3.2.xsd*  *http://www.springframework.org/schema/tx*  *http://www.springframework.org/schema/tx/spring-tx-3.2.xsd"*>  </beans> |

### j：配置c3p0数据源和mybatis的会话工厂

|  |
| --- |
| <!-- 1.加载db配置文件 -->  <context:property-placeholder location=*"classpath:db.properties"*/>  <!-- 2.配置c3p0数据源 -->  <bean id=*"dataSource"* class=*"com.mchange.v2.c3p0.ComboPooledDataSource"*>  <property name=*"driverClass"* value=*"${jdbc.driver}"*/>  <property name=*"jdbcUrl"* value=*"${jdbc.url}"*/>  <property name=*"user"* value=*"${jdbc.username}"*/>  <property name=*"password"* value=*"${jdbc.password}"*/>  <property name=*"maxPoolSize"* value=*"30"*/>  <property name=*"minPoolSize"* value=*"2"*/>  </bean>  <!-- 3.让spring管理sqlsessionFactory -->  <bean class=*"org.mybatis.spring.SqlSessionFactoryBean"* id=*"sqlSessionFactoryBean"*>  <property name=*"dataSource"* ref=*"dataSource"*/>  <!-- 指定配置文件位置 -->  <property name=*"configLocation"* value=*"classpath:SqlMapConfig.xml"*/>  </bean>    <!-- 4.配置mapper扫描器.批量扫描创建代理对象 -->  <bean class=*"org.mybatis.spring.mapper.MapperScannerConfigurer"*>  <property name=*"basePackage"* value=*"com.gyf.backoffice.mapper"*/>  <property name=*"sqlSessionFactoryBeanName"* value=*"sqlSessionFactoryBean"*/>  </bean> |
| **db文件内容**  jdbc.driver=com.mysql.jdbc.Driver  jdbc.url=jdbc:mysql://localhost:3306/mybatis  jdbc.username=root  jdbc.password=123456 |

### k：Web.xml配置spring容器

|  |
| --- |
| <!-- spring的配置 -->  <context-param>  <param-name>contextConfigLocation</param-name>  <param-value>classpath:applicationContext.xml</param-value>  </context-param>  <listener>  <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>  </listener> |

### l：添加bean的注解装配

|  |
| --- |
| <!-- 配置扫描注解 -->  <context:component-scan base-package=*"com.gyf.backoffice"*/> |

### m：ItemsService

|  |
| --- |
|  |

### n：ItemsController

|  |
| --- |
|  |

**这里可以方法items/list.do，是否可以获取数据**

### o：事务配置

|  |
| --- |
| Spring中Propagation类的事务属性详解：  REQUIRED：支持当前事务，如果当前没有事务，就新建一个事务。这是最常见的选择。  SUPPORTS：支持当前事务，如果当前没有事务，就以非事务方式执行。  MANDATORY：支持当前事务，如果当前没有事务，就抛出异常。  REQUIRES\_NEW：新建事务，如果当前存在事务，把当前事务挂起。  NOT\_SUPPORTED：以非事务方式执行操作，如果当前存在事务，就把当前事务挂起。  NEVER：以非事务方式执行，如果当前存在事务，则抛出异常。  NESTED：支持当前事务，如果当前事务存在，则执行一个嵌套事务，如果当前没有事务，就新建一个事务 |

|  |
| --- |
| <!-- 6.事务管理器 -->  <**bean id="transactionManager" class="org.springframework.jdbc.datasource.DataSourceTransactionManager"**>  <**property name="dataSource" ref="dataSource"**/>  </**bean**>  <!-- 6.通知 -->  <tx:advice id=*"txAdvice"* transaction-manager=*"transactionManager"*>  <tx:attributes>  <!-- 传播行为 -->  <tx:method name=*"save\*"* propagation=*"REQUIRED"*/>  <tx:method name=*"insert\*"* propagation=*"REQUIRED"*/>  <tx:method name=*"delete\*"* propagation=*"REQUIRED"*/>  <tx:method name=*"update\*"* propagation=*"REQUIRED"*/>  <tx:method name=*"find\*"* propagation=*"SUPPORTS"* read-only=*"true"*/>  <tx:method name=*"get\*"* propagation=*"SUPPORTS"* read-only=*"true"*/>  </tx:attributes>  </tx:advice>    <!-- 7. 切面 -->  <aop:config>  <aop:advisor advice-ref=*"txAdvice"*  pointcut=*"execution(\* com.gyf.backoffice.service.\*.\*(..))"*/>  </aop:config> |
| *<!-- 5.配置事务管理器 -->*  <**bean id="transactionManager" class="org.springframework.jdbc.datasource.DataSourceTransactionManager"**>  <**property name="dataSource" ref="dataSource"**/>  </**bean**>  *<!-- 6.开启事务注解-->*  <**tx:annotation-driven**></**tx:annotation-driven**> |

### o：添加一个保存方法测试事务

|  |
| --- |
| service |
|  |
| controller |
|  |

## 增删改查任务

### 显示商品数据

|  |
| --- |
|  |
| Controller.java |
|  |
| Views/items/list.jsp |
|  |

### 删除商品

|  |
| --- |
| service |
|  |
| controller |
|  |

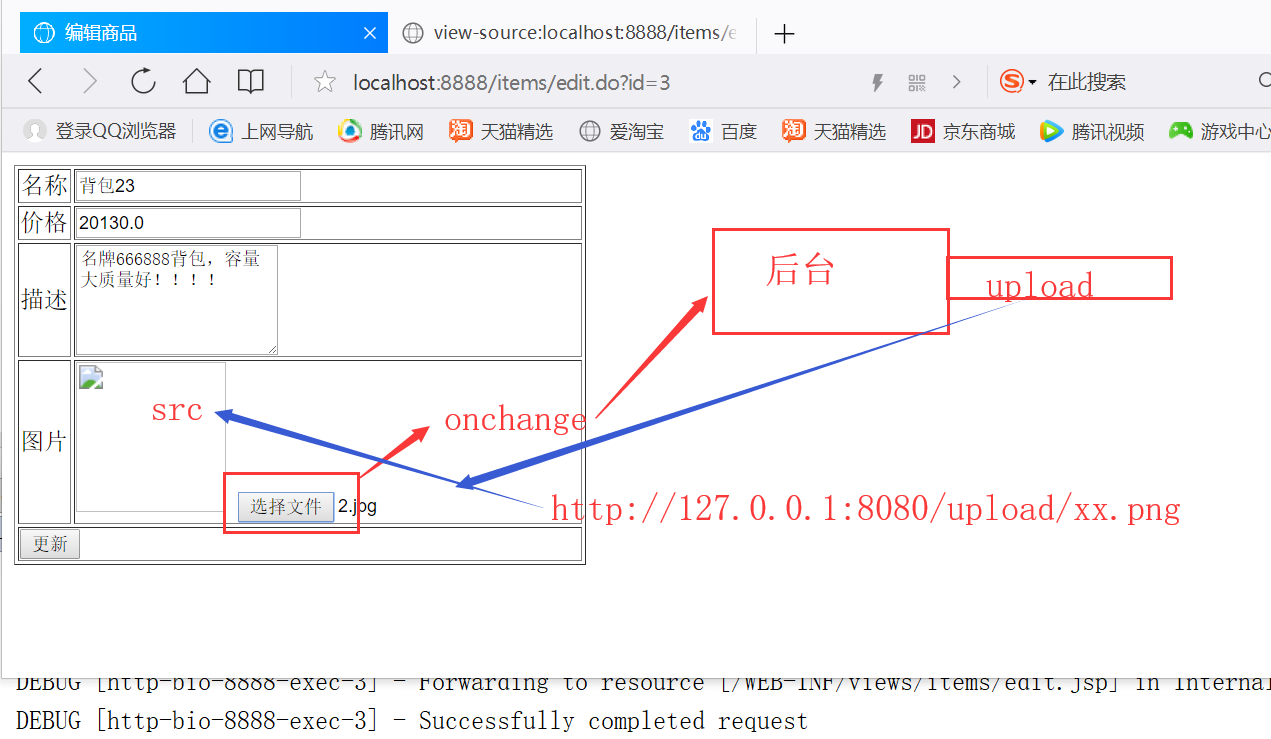
### 显示编辑商品页面

|  |
| --- |
|  |
| controller |
|  |
| Edit.jsp |
|  |

### 更新商品

|  |
| --- |
| 需要在页面中隐藏一个input标签存id |
|  |
| 乱码，在web.xml配置 |
| <!-- 配置编码过滤器 -->  <filter>  <filter-name>EncodingFilter</filter-name>  <filter-class>org.springframework.web.filter.CharacterEncodingFilter</filter-class>  <init-param>  <param-name>encoding</param-name>  <param-value>UTF-8</param-value>  </init-param>  </filter>    <filter-mapping>  <filter-name>EncodingFilter</filter-name>  <url-pattern>/\*</url-pattern>  </filter-mapping> |
| controller |
|  |
| Service:需要改下逻辑 |
|  |

## 文件上传



### 导包

|  |
| --- |
|  |
|  |

### Springmvc.xml配置支持文件上传

|  |
| --- |
|  |

### Html/JS

|  |
| --- |
|  |
|  |
|  |

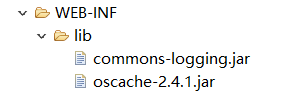
### 后台

|  |
| --- |
| @Controller  @RequestMapping("upload")  **public** **class** UploadController {  @RequestMapping("itemsPic")  **public** **void** itemsPic(HttpServletRequest request,String fileName, PrintWriter writer) **throws** Exception{  MultipartHttpServletRequest mulRequest = (MultipartHttpServletRequest) request;  //通过字段名获取文件数据  MultipartFile file1 = mulRequest.getFile(fileName);    System.***out***.println(fileName + ":上传文件大小 " + file1.getSize());    //格式化文件名  SimpleDateFormat sdf = **new** SimpleDateFormat("yyyyMMddHHmmssSSS");  String saveName = sdf.format(**new** Date());  Random rd = **new** Random();  **for**(**int** i=0;i<3;i++){  saveName += rd.nextInt(10);//添加三个随机数  }    //文件后缀名  String suffix = file1.getOriginalFilename().split("\\.")[1];    String saveFileName = saveName + "." + suffix;    //文件保存目录  String dir = request.getServletContext().getRealPath("/upload");  File dirFile = **new** File(dir);  **if**(dirFile.exists() == **false**)dirFile.mkdirs();    //文件保存路径  String relativePath = "/upload/" + saveFileName;  String totalPath = request.getServletContext().getRealPath(relativePath);  System.***out***.println(totalPath);    File newFile = **new** File(totalPath);    //保存  FileCopyUtils.*copy*(file1.getInputStream(), **new** FileOutputStream(newFile));  Thread.*sleep*(2000);  //file1.transferTo(newFile);    //返回一个相对路径 相对路径和全路径  String serverIp = "http://127.0.0.1:8080" + request.getServletContext().getContextPath();  String respJson = "{\"imgUrl\":\"" + serverIp + relativePath+"\"}";    writer.write(respJson);  }  } |

## Oscache页面缓存

### 7.1页面缓存测试

#### 导包



#### 测试

创建一个index.jsp页面，使用时间来测试

|  |
| --- |
|  |

#### 知识点：

缓存数据结构：map，key存储浏览器访问url，如果访问2个url不一致，缓存肯定变化。

##### 缓存的作用域

缓存默认存储在application域当中，可更改缓存数据。

|  |
| --- |
|  |

**这样，不同浏览器访问时，缓存时间是不一样的，如果是application,不同浏览器访问同一个路径时，时间是一样的**

##### 缓存刷新时间

|  |
| --- |
|  |

##### 固定存储的Key

|  |
| --- |
|  |

##### 缓存持久化

创建oscache.properties这个配置文件必须在classpath下面：

|  |
| --- |
|  |

内容

|  |
| --- |
| cache.memory=false  cache.persistence.class=com.opensymphony.oscache.plugins.diskpersistence.DiskPersistenceListener  cache.path=C:\\test |

访问路径后会在test的application中创建缓存文件

|  |
| --- |
|  |

## Oscache整合ssm项目

约定：商品页面访问量特别大，给商品页面缓存。Items路径下所有请求都缓存。

### 在web.xml中配置页面缓存

|  |
| --- |
| <!-- 配置页面缓存 -->  <filter>  <filter-name>oscache</filter-name> <filter-class>com.opensymphony.oscache.web.filter.CacheFilter</filter-class>  <init-param>  <param-name>time</param-name>  <param-value>3600</param-value>  </init-param>  <init-param>  <param-name>scope</param-name>  <param-value>application</param-value>  </init-param>  </filter>  <filter-mapping>  <filter-name>oscache</filter-name>  <url-pattern>/items/\*</url-pattern>  </filter-mapping> |

第一次访问Items下面的页面，控制器方法会执行，第一次就走缓存页面

**如图，把持久化缓存加到项目中，缓存的数据是一个页面**

|  |
| --- |
|  |
| cache.persistence.class=com.opensymphony.oscache.plugins.diskpersistence.DiskPersistenceListener  cache.path=C:\\test |

## Springmvc整合freemarker

### 简介：freemark

FreeMarker是一款模板引擎： 即一种基于模板和要改变的数据， 并用来生成输出文本（HTML网页、电子邮件、配置文件、源代码等）的通用工具。

<https://freemarker.apache.org/>

<http://freemarker.foofun.cn>

模型引擎框架还有：https://www.thymeleaf.org/

### 导包

|  |
| --- |
|  |

### Springmvc配置freemark

|  |
| --- |
| <!-- 配置freemarker -->  <bean class=*"org.springframework.web.servlet.view.freemarker.FreeMarkerConfigurer"*>  <property name=*"templateLoaderPath"* value=*"/WEB-INF/views"* />  <property name=*"defaultEncoding"* value=*"UTF-8"*></property>  </bean>    <bean class=*"org.springframework.web.servlet.view.freemarker.FreeMarkerViewResolver"*>  <property name=*"contentType"* value=*"text/html;charset=utf-8"*/>  <property name=*"suffix"* value=*".ftl"* />  </bean> |

### 写一个hello.ftl模板

|  |
| --- |
|  |

### ItemsController

|  |
| --- |
|  |

### 测试结果

数据就通过模板引擎生成了

|  |
| --- |
|  |

### list.jsp改成list.ftl

|  |
| --- |
|  |

掌握freemark的两个表达工：

赋值

<#assign/>

遍历

<#list xxList as xx></list>

${item.createtime?string("yyyy-MM-dd HH:mm:ss zzzz")}

## SpringMVC的拦截器

**Spring Web MVC 的处理器拦截器类似于Servlet 开发中的过滤器Filter，用于对处理器进行预处理和后处理**

### 登录拦截器

|  |
| --- |
| **public** **class** LoginInterceptor **implements** HandlerInterceptor{  //controller执行后且视图返回后调用此方法  **public** **void** afterCompletion(HttpServletRequest arg0, HttpServletResponse arg1, Object arg2, Exception arg3)  **throws** Exception {  System.***out***.println("返回视图前 后处理");  }  //controller执行后但未返回视图前调用此方法  **public** **void** postHandle(HttpServletRequest arg0, HttpServletResponse arg1, Object arg2, ModelAndView arg3)  **throws** Exception {  System.***out***.println("未返回视图前 后处理....");  }  // controller执行前调用此方法  **public** **boolean** preHandle(HttpServletRequest request, HttpServletResponse response, Object obj) **throws** Exception {  // **TODO** Auto-generated method stub  System.***out***.println("预处理....");  //如果是登录页面则放行  **if**(request.getRequestURI().indexOf("login.do")>=0){  **return** **true**;  }  HttpSession session = request.getSession();  //如果用户已登录也放行  **if**(session.getAttribute("user")!=**null**){  **return** **true**;  }  //用户没有登录挑战到登录页面  request.getRequestDispatcher("/WEB-INF/views/login.jsp").forward(request, response);    **return** **false**;  }  } |

### 登录控制器

|  |
| --- |
|  |

### 登录界面

|  |
| --- |
|  |
|  |

### 登录拦截器配置

|  |
| --- |
|  |

总结拦截注意点：

拦截所有的请求，使用/\*\*,有两个\*\*

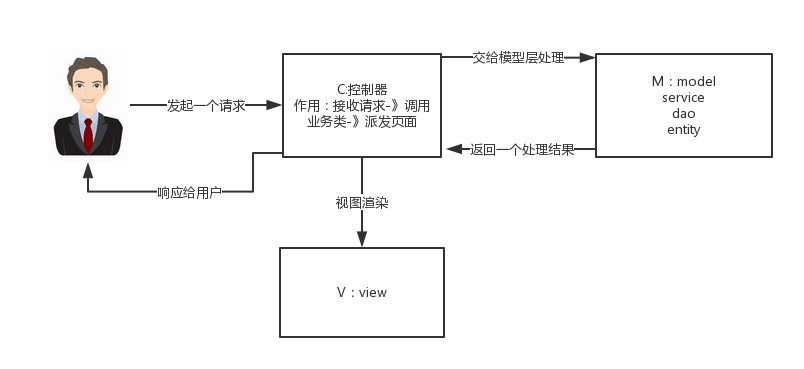
如果拦截的路径在项目不存在，直接报错

# 秒杀

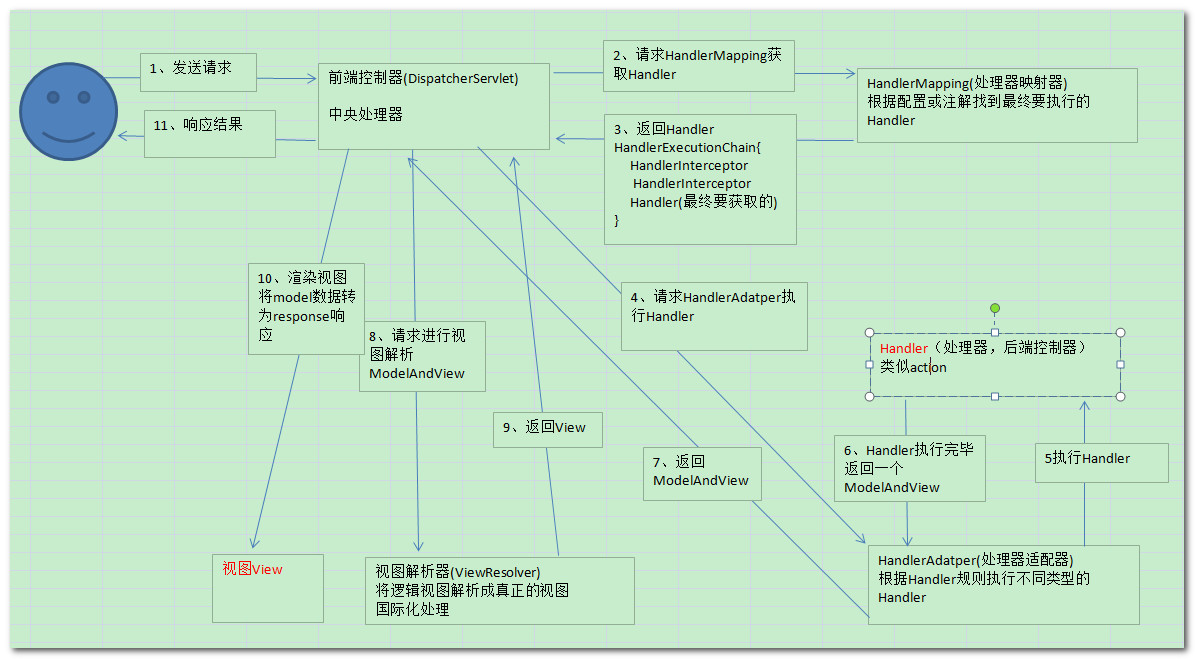
## MVC概念

模型-视图-控制器（MVC）是一个以设计界面应用程序为基础的设计模式。它主要通过分离模型、视图及控制器在应用程序中的角色将业务逻辑从界面中解耦。通常，模型负责封装应用程序数据在视图层展示。视图仅仅只是展示这些数据，不包含任何业务逻辑。控制器负责接收来自用户的请求，并调用后台服务（manager或者dao）来处理业务逻辑。处理后，后台业务层可能会返回了一些数据在视图层展示。控制器收集这些数据及准备模型在视图层展示。MVC模式的核心思想是将业务逻辑从界面中分离出来，允许它们单独改变而不会相互影响。

## mvc原理



## mvc流程



1、 用户发送请求至前端控制器DispatcherServlet。

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

3、 处理器映射器找到具体的处理器(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。

4、 DispatcherServlet调用HandlerAdapter处理器适配器。

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

6、 Controller执行完成返回ModelAndView。

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

8、 DispatcherServlet将ModelAndView传给ViewReslover视图解析器。

9、 ViewReslover解析后返回具体View。

10、DispatcherServlet根据View进行渲染视图（即将模型数据填充至视图中）。

11、 DispatcherServlet响应用户。

## 构建项目

### maven

|  |
| --- |
| <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-webmvc</artifactId>  <version>3.2.4.RELEASE</version>  </dependency>  <dependency>  <groupId>javax.servlet</groupId>  <artifactId>servlet-api</artifactId>  <version>2.4</version>  </dependency>  <dependency>  <groupId>javax.servlet</groupId>  <artifactId>jstl</artifactId>  <version>1.2</version>  </dependency> |

### application-context.xml

配置

|  |
| --- |
| <?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:context="http://www.springframework.org/schema/context"  xsi:schemaLocation="http://www.springframework.org/schema/beans  http://www.springframework.org/schema/beans/spring-beans-3.1.xsd  http://www.springframework.org/schema/context  http://www.springframework.org/schema/context/spring-context-3.1.xsd">  <context:component-scan base-package="test" />  <context:annotation-config />  </beans> |

### web.xml

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <web-app version="3.0" xmlns="http://java.sun.com/xml/ns/javaee"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app\_3\_0.xsd">  <context-param>  <param-name>contextConfigLocation</param-name>  <param-value>  classpath:application-context.xml  </param-value>  </context-param>    <filter>  <filter-name>characterEncodingFilter</filter-name>  <filter-class>org.springframework.web.filter.CharacterEncodingFilter</filter-class>  <init-param>  <param-name>encoding</param-name>  <param-value>UTF-8</param-value>  </init-param>  </filter>  <filter-mapping>  <filter-name>characterEncodingFilter</filter-name>  <url-pattern>/\*</url-pattern>  </filter-mapping>  <listener>  <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>  </listener>  <listener>  <listener-class>org.springframework.web.util.IntrospectorCleanupListener</listener-class>  </listener>  <servlet>  <servlet-name>spring</servlet-name>  <servlet-class>  org.springframework.web.servlet.DispatcherServlet  </servlet-class>  <load-on-startup>1</load-on-startup>  </servlet>  <servlet-mapping>  <servlet-name>spring</servlet-name>  <url-pattern>/\*</url-pattern>  </servlet-mapping>  <display-name>Archetype Created Web Application</display-name>  </web-app> |

### spring-servlet.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:context="http://www.springframework.org/schema/context"  xmlns:tx="http://www.springframework.org/schema/tx" xmlns:mvc="http://www.springframework.org/schema/mvc"  xsi:schemaLocation="http://www.springframework.org/schema/beans  http://www.springframework.org/schema/beans/spring-beans.xsd  http://www.springframework.org/schema/context  http://www.springframework.org/schema/context/spring-context.xsd  http://www.springframework.org/schema/tx  http://www.springframework.org/schema/tx/spring-tx.xsd  http://www.springframework.org/schema/mvc  http://www.springframework.org/schema/mvc/spring-mvc.xsd">    <context:component-scan base-package="test.\*" />  <mvc:annotation-driven />  <!--  <bean class="org.springframework.web.servlet.mvc.annotation.DefaultAnnotationHandlerMapping"/>  <bean class="org.springframework.web.servlet.mvc.annotation.AnnotationMethodHandlerAdapter"/>  -->  <mvc:view-controller path="/" view-name="index"/>  <bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">  <property name="order" value="1" />  <property name="prefix" value="/WEB-INF/view/" />  <property name="suffix" value=".jsp" />  <property name="viewClass" value="org.springframework.web.servlet.view.JstlView" />  </bean>    </beans> |

### controller

|  |
| --- |
| /\*\*  \* @description: 功能描述：()  \* @copyright: Copyright (c) 2019  \* @company: 昭阳科技  \* @author:  \* @version: 2.0  \* @date: 2019 2019年5月2日 下午8:55:51  \*/  package com.yaosiyuan.controler;  import java.io.IOException;  import java.io.PrintWriter;  import javax.servlet.http.HttpServletRequest;  import javax.servlet.http.HttpServletResponse;  import org.springframework.stereotype.Controller;  import org.springframework.web.bind.annotation.RequestMapping;  /\*\*  \* @description: 功能描述 ()  \* @copyright: Copyright (c) 2019  \* @company: 昭阳科技  \* @author:  \* @version: 2.0  \* @date: 2019 2019年5月2日 下午8:55:51  \*/  @Controller  @RequestMapping("hello")  public class HelloController {  @RequestMapping("test")  public void hello(HttpServletRequest request, HttpServletResponse response) {  String name = request.getParameter("name");  response.setCharacterEncoding("utf-8");  response.setContentType("application/json");  try {  PrintWriter writer = response.getWriter();  writer.println("hello==" + name);  } catch (IOException e) {  // TODO Auto-generated catch block  e.printStackTrace();  }  }  } |