# <http://www.cnblogs.com/sunniest/p/4555801.html>

# 1简单的spring mvc项目

## 1.1设置web.xml中的内容为下面的

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<web-app xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"* xmlns=*"http://java.sun.com/xml/ns/javaee"* xmlns:web=*"http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd"* xsi:schemaLocation=*"http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd"* id=*"WebApp\_ID"* version=*"2.5"*>

<display-name>StudySpringMvcOne</display-name>

<!-- springmvc前端控制器 -->

<servlet>

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

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

<!-- contextConfigLocation配置springmvc加载的配置文件（配置处理器映射器、适配器等等）

如果不配置contextConfigLocation，默认加载的是/WEB-INF/servlet名称-serlvet.xml（springmvc-servlet.xml）

-->

<init-param>

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

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

</init-param>

</servlet>

<servlet-mapping>

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

<!--

第一种：\*.action，访问以.action结尾 由DispatcherServlet进行解析,决定了请求URL必须是一个带后缀的URL

第二种：/，所有访问的地址都由DispatcherServlet进行解析，对于静态文件的解析需要配置不让DispatcherServlet进行解析，例如<mvc:default-servlet-handler />

使用此种方式可以实现 RESTful风格的url

第三种：/\*，这样配置不对，使用这种配置，最终要转发到一个jsp页面时，

仍然会由DispatcherServlet解析jsp地址，不能根据jsp页面找到handler，会报错。

-->

<url-pattern>/</url-pattern>

</servlet-mapping>

<welcome-file-list>

<welcome-file>index.html</welcome-file>

<welcome-file>index.htm</welcome-file>

<welcome-file>index.jsp</welcome-file>

<welcome-file>default.html</welcome-file>

<welcome-file>default.htm</welcome-file>

<welcome-file>default.jsp</welcome-file>

</welcome-file-list>

</web-app>

## 1.2.设置springmvc-servlet.xml（src目录下创建）

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<beans xmlns=*"http://www.springframework.org/schema/beans"*

xmlns:context=*"http://www.springframework.org/schema/context"*

xmlns:p=*"http://www.springframework.org/schema/p"*

xmlns:mvc=*"http://www.springframework.org/schema/mvc"*

xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xsi:schemaLocation=*"http://www.springframework.org/schema/beans*

*http://www.springframework.org/schema/beans/spring-beans-3.0.xsd*

*http://www.springframework.org/schema/context*

*http://www.springframework.org/schema/context/spring-context.xsd*

*http://www.springframework.org/schema/mvc*

*http://www.springframework.org/schema/mvc/spring-mvc-3.0.xsd"*>

<!--

如果不想在xml文件中配置bean，我们可以给我们的类加上spring组件注解，

只需再配置下spring的扫描器就可以实现bean的自动载入。

-->

<context:component-scan base-package=*"com.hlj.control"*/>

<!-- 针对于web中的 /

在springMVC-servlet.xml中配置<mvc:default-servlet-handler />后，

会在Spring MVC上下文中定义一个org.springframework.web.servlet.resource.DefaultServletHttpRequestHandler，

它会像一个检查员，对进入DispatcherServlet的URL进行筛查，如果发现是静态资源的请求，

就将该请求转由Web应用服务器默认的Servlet处理，如果不是静态资源的请求，才由DispatcherServlet继续处理。

一般Web应用服务器默认的Servlet名称是"default"，因此DefaultServletHttpRequestHandler可以找到它。

如果你所有的Web应用服务器的默认Servlet名称不是"default"，则需要通过default-servlet-name属性显示指定：

<mvc:default-servlet-handler default-servlet-name="所使用的Web服务器默认使用的Servlet名称" />

-->

<mvc:default-servlet-handler/>

<!--

使用了<mvc:annotation-driven />， 它会自动注册DefaultAnnotationHandlerMapping

与AnnotationMethodHandlerAdapter 这两个bean,所以就没有机会再给它注入interceptors属性，就无法指定拦截器。

当然我们可以通过人工配置上面的两个Bean，不使用 <mvc:annotation-driven />，就可以 给interceptors属性 注入拦截器了。

-->

<mvc:annotation-driven/>

<!-- InternalResourceViewResolver视图解析器 -->

<bean id=*"viewResolver"* class=*"org.springframework.web.servlet.view.InternalResourceViewResolver"*>

<!-- 前缀 -->

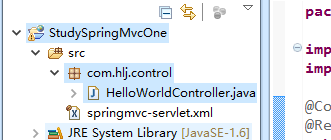
<property name=*"prefix"* value=*"/WEB-INF/jsp/"*></property>

<!-- 后缀 -->

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

</bean>

## 1.3、创建一个Control名字以它结尾



**package** com.hlj.control;

**import** org.springframework.stereotype.Controller;

**import** org.springframework.web.bind.annotation.RequestMapping;

@Controller

@RequestMapping("/mvc")

**public** **class** HelloWorldController {

/\*\*

\* 1

\* **@return**

\*/

@RequestMapping("/hello")

**public** String hello(){

String testword = "测试第一个spring mvc 项目//n"+"hello world ,spring mvc 我来了";

System.*out*.println(testword);

**return** "hellojsp"; //这个就是我们设置的jsp页面的名字

}

## 1.4、第一个测试



http://zhangyj-a-3:8080/StudySpringMvcOne/mvc/hello



# 2.自动匹配参数和一方法多名字路径

## 1.编写自动匹配参数的方法

/\*\*

\* 2

在springmvc中 配置 **@RequestMapping**(value={"test", "test1"})

如果你项目端口是8080；然后你要访问这个方法,

你可以通过 http://localhost:8080/test或者http://localhost:8080/test1来访问

\*/

@RequestMapping(value={"matchAutomatic1", "matchAutomatic2"})

//@RequestMapping(value="matchAutomatic1")

**public** String matchAutomatic(String name ,**double** age){

System.*out*.println("测试自动匹配参数/n"+"名字："+name+"年龄："+age);

**return** "hellojsp";

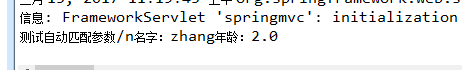
}

## 2.进行第二个测试

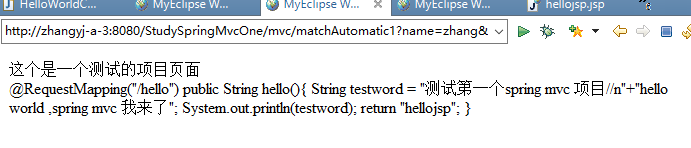
### 1.路径输入

http://zhangyj-a-3:8080/StudySpringMvcOne/mvc/matchAutomatic1?name=zhang&age=2

### 2.观察eclipse后台，发现有数据得到，说明是这个是先进入方法，后进入jsp页面



### 3.观察浏览器



# 3.自动装箱

## 1.创建一个control方法

/\*\*

\* 3 自动装箱，需要一个person实体类

\*

\*/

@RequestMapping(value="autoboxing")

**public** String autoboxing(Person person){

System.*out*.println("测试自动装箱/n"+"名字："+person.getName()+"年龄："+person.getAge());

**return** "hellojsp";

}

## 2.创建person类

**public** **class** Person {

String name;

**int** age;

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

**public** **int** getAge() {

**return** age;

}

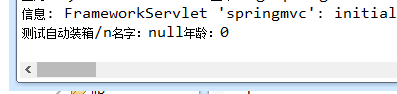
**public** **void** setAge(**int** age) {

**this**.age = age;

}

}

## 3.第三个测试



# 4. 使用InitBinder 处理Date日期

## 1.创建control方法

/\*\*

\* 4 使用InitBinder 处理Date日期

\*/

@RequestMapping(value="date")

**public** String date(Date date){

System.*out*.println(date);

**return** "hellojsp";

}

@InitBinder

**public** **void** initBinder(ServletRequestDataBinder binder){

binder.registerCustomEditor(Date.**class**, **new** CustomDateEditor(**new** SimpleDateFormat("yyyy-MM-dd"), **true**));

}

## 2.第四个测试

http://zhangyj-a-3:8080/StudySpringMvcOne/mvc/date?date=**1884-02-12**



# 5.向前台传递参数

## 1.编写control方法，使用的是map进行放入的

/\*\*

\* 5.向前台传递参数

\*/

@RequestMapping(value="passparameter")

**public** String date(Map<String, Object> map){

Person person = **new** Person();

map.put("person", person);

person.setAge(20);

person.setName("张宇晋");

**return** "getParemeter";

}

## 2.前台进行接收

<%

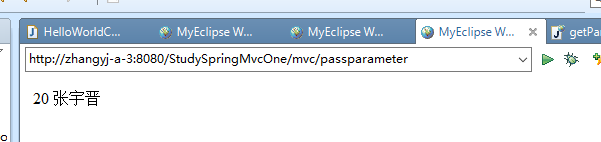
Person person = (Person)request.getAttribute("person");

out.println(person.getAge());

out.println(person.getName());

%>

## 3.第五个测试



# 6. 未实现通过AJAX调用（未实现）

# 7、未实现在control中使用redirect处理请求

# 8.文件上传

## 1.先在springmvc 配置文件中写入下面的，就是文件支持长传

<!-- 8,文件上传-->

<!-- 支持上传文件 ,并且设置最大文件容量-->

<bean id=*"multipartResolver"* class=*"org.springframework.web.multipart.commons.CommonsMultipartResolver"*>

<property name=*"maxUploadSize"* value=*"102400000"*></property>

</bean>

## 2.编写control方法

@RequestMapping(value="uploadfile", method=RequestMethod.*POST*)

**public** String upload(HttpServletRequest request) **throws** IOException{

System.*out*.println("开始");

MultipartHttpServletRequest merp =(MultipartHttpServletRequest)request;

MultipartFile file = merp.getFile("file");

String fileName = file.getOriginalFilename();

SimpleDateFormat sdf = **new** SimpleDateFormat("yyyyMMddHHmmss");

FileOutputStream fos = **new** FileOutputStream(request.getSession().getServletContext().getRealPath("/")+

"upload/"+sdf.format(**new** Date())+fileName.substring(fileName.lastIndexOf(".")));

fos.write(file.getBytes());

fos.flush();

fos.close();

**return** "success";

}

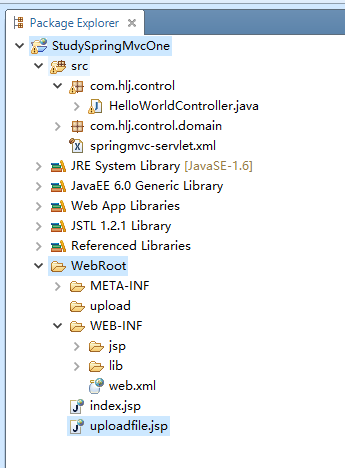
## 3.编写html文件’

<form action=*"mvc/uploadfile"* method=*"post"* enctype=*"multipart/form-data"*>

<input type=*"file"* name=*"file"* />

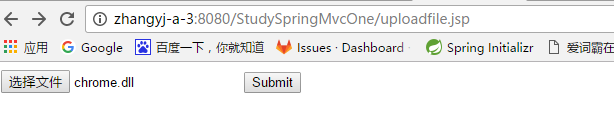
<input type=*"submit"* value=*"Submit"* />

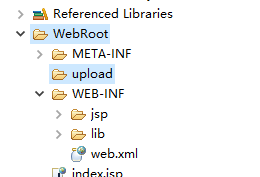
</form>

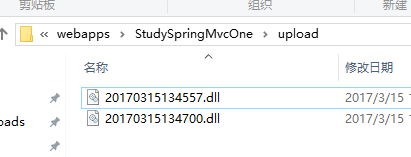


## 4.进行第六次测试

http://zhangyj-a-3:8080/StudySpringMvcOne/uploadfile.jsp







# 9. 使用@RequestParam 注解指定参数的name

# 10、十二、RESTFul风格的SringMVC

## 1.重建一个contoller类

@Controller

@RequestMapping("/rest")

**public** **class** RestController {

@RequestMapping(value="/user/{id}",method=RequestMethod.*PUT*)

**public** String put(@PathVariable("id") Integer id){

System.*out*.println("put"+id);

**return** "/RestController";

}

@RequestMapping(value="/user/{id}",method=RequestMethod.*DELETE*)

**public** String delete(@PathVariable("id") Integer id){

System.*out*.println("delete"+id);

**return** "/RestController";

}

@RequestMapping(value="/user/{id}",method=RequestMethod.*GET*)

**public** String get(@PathVariable("id") Integer id){

System.*out*.println("get"+id);

**return** "/RestController";

}

@RequestMapping(value="/user/{id}",method=RequestMethod.*POST*)

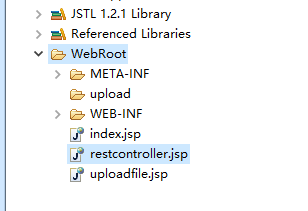
**public** String post(@PathVariable("id") Integer id){

System.*out*.println("post"+id);

**return** "/RestController";

}

## 2.编辑html页面



<form action=*"rest/user/1"* method=*"post"*>

<input type=*"hidden"* name=*"\_method"* value=*"PUT"*>

<input type=*"submit"* value=*"put"*>

</form>

<form action=*"rest/user/1"* method=*"post"*>

<input type=*"hidden"* name=*"\_method"* value=*"DELETE"*>

<input type=*"submit"* value=*"delete"*>

</form>

<form action=*"rest/user/1"* method=*"post"*>

<input type=*"submit"* value=*"post"*>

</form>

<form action=*"rest/user/1"* method=*"get"*>

<input type=*"submit"* value=*"get"*>

</form>

## 3.编辑web.xml进行转化put和delete

配置隐藏的httpmethod过滤器转化post方法到put或者是delete

configure the HiddenHttpMethodFilter,convert the post method to put or delete

-->

<filter>

<filter-name>HiddenHttpMethodFilter</filter-name>

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

</filter>

<filter-mapping>

<filter-name>HiddenHttpMethodFilter</filter-name>

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

</filter-mapping>

# 11、未实现、返回json格式的字符串

}

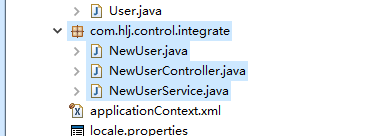
# 12. 未实现 异常的处理

# 13. 未实现设置一个自定义拦截器

# 14未实现表单的验证（使用Hibernate-validate）及国际化

# 15、整合SpringIOC和SpringMVC

## 1.创建一个com.hlj.control.integrate的包用来演示整合，并创建各类



## 2.NewUser实体类

**public** **class** NewUser {

@Override

**public** String toString() {

**return** "User [id=" + id + ", name=" + name + ", birth=" + birth + "]";

}

**private** **int** id;

@NotEmpty

**private** String name;

@Past

@DateTimeFormat(pattern="yyyy-MM-dd")

**private** Date birth;

**public** **int** getId() {

**return** id;

}

**public** **void** setId(**int** id) {

**this**.id = id;

}

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

**public** Date getBirth() {

**return** birth;

}

**public** **void** setBirth(Date birth) {

**this**.birth = birth;

}

## 3.NewUserService类

@component （把普通pojo实例化到spring容器中，相当于配置文件中的<bean id="" class=""/>）

@Component

**public** **class** NewUserService {

**public** NewUserService(){

System.*out*.println("UserService Constructor...\n\n\n\n\n\n");

}

**public** **void** save(){

System.*out*.println("save");

}

}

## 4.NewUserController

@Controller

@RequestMapping("/integrate")

**public** **class** NewUserController {

@Autowired // 省略 get/setter

**private** NewUserService userService;

@RequestMapping("/user")

**public** String saveUser(@RequestBody @ModelAttribute NewUser u){

System.*out*.println(u);

userService.save();

**return** "hello";

}

}

## 5.Spring配置文件 Spring接管

在src目录下创建SpringIOC的配置文件applicationContext.xml

<context:component-scan base-package=*"com.hlj.control.integrate"*>

<context:exclude-filter type=*"annotation"*

expression=*"org.springframework.stereotype.Controller"*/>

<context:exclude-filter type=*"annotation"*

expression=*"org.springframework.web.bind.annotation.ControllerAdvice"*/>

</context:component-scan>

## 6.在Web.xml中添加配置,配置spring

<!-- 15、整合SpringIOC和SpringMVC -->

<!-- configure the springIOC -->

<listener>

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

</listener>

<context-param>

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

<param-value>classpath:applicationContext.xml</param-value>

</context-param>

## 7在SpringMVC中进行一些配置，防止SpringMVC和SpringIOC对同一个对象的管理重合

在SpringMVC中进行一些配置，防止SpringMVC和SpringIOC对同一个对象的管理重合

-->

<!-- scan the package and the sub package -->

<context:component-scan base-package=*"test.SpringMVC.integrate"*>

<context:include-filter type=*"annotation"*

expression=*"org.springframework.stereotype.Controller"*/>

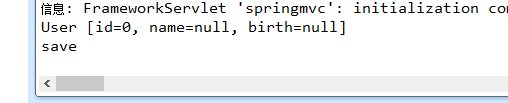
<context:include-filter type=*"annotation"*

expression=*"org.springframework.web.bind.annotation.ControllerAdvice"*/>

</context:component-scan>

## 8.测试

http://zhangyj-a-3:8080/StudySpringMvcOne/integrate/user



# 16. SpringMVC运行原理

1. 客户端请求提交到DispatcherServlet  
2. 由DispatcherServlet控制器查询一个或多个HandlerMapping，找到处理请求的Controller  
3. DispatcherServlet将请求提交到Controller  
4. Controller调用业务逻辑处理后，返回ModelAndView  
5. DispatcherServlet查询一个或多个ViewResoler视图解析器，找到ModelAndView指定的视图  
6. 视图负责将结果显示到客户端

# 17. SpringMVC与struts2的区别

1、springmvc基于方法开发的，struts2基于类开发的。springmvc将url和controller里的方法映射。映射成功后springmvc生成一个Handler对象，对象中只包括了一个method。方法执行结束，形参数据销毁。springmvc的controller开发类似web service开发。  
2、springmvc可以进行单例开发，并且建议使用单例开发，struts2通过类的成员变量接收参数，无法使用单例，只能使用多例。  
3、经过实际测试，struts2速度慢，在于使用struts标签，如果使用struts建议使用jstl。

# 18 书本再次学习 P581

@Controller

@RequestMapping("/user")

//将ModelMap中名为user的属性放到Session中，以便这个属性可以跨请求访问

@SessionAttributes("user")

**public** **class** UserController {

//处理登录请求

@RequestMapping(value="/login")

**public** String login(Users user,ModelMap model){

//将user对象以user为键放入到model中

model.put("user", user);

**return** "success";

}

//@RequestMapping(value="/toReg")

//@RequestMapping(value="/reg",method=RequestMethod.GET)

@RequestMapping(params="method=toReg")

//获取"爱好"下拉列表数据，显示注册页面

**public** String toReg(ModelMap model){

model.put("hobbyList", **new** String[]{"Swimming","Running"});

**return** "register";

}

//日期格式转换

@InitBinder

**public** **void** initBinder(WebDataBinder binder){

binder.registerCustomEditor(Date.**class**, **new** CustomDateEditor(**new** SimpleDateFormat("yyyy-MM-dd"), **true**));

}

//处理注册请求

//@RequestMapping(value="/doReg")

//@RequestMapping(value="/reg",method=RequestMethod.POST)

@RequestMapping(params="method=doReg") //使用参数之后就是 user.do?method=doReg

**public** String doReg(Users user, ModelMap model){

//将user对象以user为键放入到model中

model.put("user", user);

//这样重定向是访问不到user 的所以上面我们添加了了 @SessionAttributes("user")

**return** "redirect:/show.jsp";

}

# 19 ModelMap 和ModelAndView, Model

## 1.ModelMap对象的 addAttribute,put两个方法有什么区搜索别就是 addAttribute是不允许添加空值的key，put是允许的

## 2.Model 有addAttribute

## 3. ModelAndView mv.addObject("mnxras",mnxras);

## 4测试

package com.gaussic.controller;

import com.gaussic.model.AccountModel;

import org.springframework.stereotype.Controller;

import org.springframework.ui.Model;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.RequestMethod;

import org.springframework.ui.ModelMap;

import org.springframework.web.servlet.ModelAndView;

import java.util.Arrays;

import java.util.Map;

import java.util.Date;

/\*\*

\* Created by Administrator on 2016/9/9 0009.

\*/

@Controller

@RequestMapping(value = "/test")

public class TestController {

@RequestMapping("/test")

public String test(Map<String,Object> map,Model model,ModelMap modelMap){

map.put("names", Arrays.asList("caoyc","zhh","cjx"));

model.addAttribute("time", new Date());

modelMap.addAttribute("city", "ChengDu");

modelMap.put("gender", "male");

return "/test/hello";

}

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

public ModelAndView index(){

ModelAndView modelAndView = new ModelAndView();

modelAndView.addObject("message", "Hello World!");

modelAndView.setViewName("/test/index");

return modelAndView;

}

}