SpringMVC-01：

Springmvc采用了松散耦合可插拔组件结构，比其他MVC框架更具有扩展性和灵活性。

Springmvc通过一套MVC注解让pojo称为处理请求的控制器，而无需实现任何接口。

**RequestMapping修饰类：**

/\*\*

\* **@RequestMapping除了修饰方法还可以修饰类**。

\*

\* 类定义出：提供初步的请求映射信息，相对于WEB应用的根目录

\* 方法处：提供进一步的戏份映射信息。

\*相对于类定义出的URL，若类定义出未标注@RequestMapping，则方法处标记的URL相对于WEB应用的根目录

\* \*/

@RequestMapping("/testRequestMapping")

**public** String testRequestMapping() {

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

**return** ***SUCCESS***;

}

**RequestMapping请求方式：**

/\*\*

\* 常用：使用method属性来指定请求方式,只有请求方式为post的请求才能请求到该方法

\* \*/

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

**public** String testMethod() {

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

**return** ***SUCCESS***;

}

**RequestMapping请求参数，请求头**

/\*\*

\* 可以使用params和headers来更加精确的映射请求，params和header支持简单的表达式

\* \*/

@RequestMapping(value="/testParamsAndHeaders",

params= {"username","age!=10"},

headers={"Accept-Language=zh-CN,zh;q=0.8"})

**public** String testParamsAndHeaders() {

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

**return** ***SUCCESS***;

}

**RequestMapping\_Ant路径：**

**RequestMapping\_PathVariable注解：**

/\*\*

\* **@PathVariable** 可以用来映射URL中的占位符到目标方法的参数中

\* \*/

@RequestMapping("/testPathVaribale/{id}")

**public** String testPathVarable(@PathVariable(value="id") Integer id) {

System.***out***.println("testPathVaribale:" + id);

**return** ***SUCCESS***;

}

REST:

REST即Representational State Transfer。(资源)表现层状态转化。是目前最流行的一种互联网软件架构。

**资源：**网络上的一个实体，或则说是网络上的一个具体信息，他可以使一段文本，一张图片，歌曲，总之就是一个具体的存在。可以用UI个URI(统一资源定位符)指向它，每种资源对应一个特定的URI，要获取这个资源访问他的URI就可以。因此URI即为每一个资源的独一无二的识别符。

**表现层：**把资源具体呈现出来的形式，叫做他的表现层。

**状态转化：**每发出一个请求，就代表了客户端和服务器的一次交互过程，HTTP协议是一个无状态的协议，即所有的状态都保存在服务器端。因此，如果客户想要操作服务器，必须通过某种手段，让服务器发生“状态转化”。而这种转化是建立在表现层之上的，所以就是“表现层状态转化”。具体说，就是HTTP协议李曼，四个表示操作方式的动词：GET,POST,PUT,DELETE。他们分别对应四种基本操作：GET用来获取资源，POST用来新建资源，PUT用来更新资源，DELETE用来删除资源。

**RequestMapping\_HiddenHttpMethodFilter过滤器：**

**RequestParam注解**：

/\*\*

\* **@RequestParam来映射请求参数**，

\* value值即请求参数的参数名

\* requireed该参数是否必须

\* defaultValue： 请求参数的默认值

\* \*/

@RequestMapping(value="/testRequestParam")

**public** String testRequestParam(@RequestParam(value="username") String username,

@RequestParam(value="age",required=**false**,defaultValue="0") Integer age) { //接受浏览器递交的请求参数

System.***out***.println("testRequestParam--->username= " + username + " age= " + age);

**return** ***SUCCESS***;

}

**RequestHeader注解**：

/\*\*使用机会较少，了解

\* 映射请求头信息

\* 用法同@RequestParam

\* \*/

@RequestMapping("/testRequestHeader")

**public** String testRequestHeader(@RequestHeader(value="Accept-Language") String al) {

System.***out***.println("testRequestHeader()..... Accept-Language" + al);

**return** ***SUCCESS***;

}

**CookieValue注解：**

/\*\*

\* 了解：

\* **@CookieValue映射一个cookie值**，属性同@RequestParam

\* 即得到请求的浏览器中的sessionid值

\* \*/

@RequestMapping("/testCookieValue")

**public** String testCookieValue(@CookieValue("JSESSIONID") String sessionId) {

System.***out***.println("testCookieValue().... " + sessionId);

**return** ***SUCCESS***;

}

**使用POJO(即一个普通的java类)作为参数**：

**Spring MVC会按请求参数名和POJO属性名进行自动的匹配，自动为该对象填充属性值，支持级联属性**。如：dept.deptId

@RequestMapping("/testPojo")

**public** String testPojo(User user) {

System.***out***.println("testPojo().... " + user);

**return** ***SUCCESS***;

}

**使用servlet API作为入参：**

@RequestMapping("/testServletAPI")

**public** **void** testServletAPI(HttpServletRequest request,

HttpServletResponse response,Writer out) **throws** IOException {

System.***out***.println("testServletAPI," + request + "," +response);

out.write("hello wenpan ");

//return SUCCESS;

}

**处理模型数据之ModelAndView:**

/\*\*

\* 目标方法的返回值可以使ModelAndView类型

\* 其中可以包含视图和model信息

\* **SpringMVC会把ModelAndView的model中数据放入到request域对象中**

\* \*/

@RequestMapping("/testModelAndView")

**public** ModelAndView testModelAndView() {

String viewName = ***SUCCESS***;

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

//添加模型数据到ModelAndView 中，即放到请求域中，可以在另外一个页面获取

**modelAndView.addObject("time",new Date());**

**return** modelAndView;

}

**处理模型数据之Map:**

给请求的方法带一个Map类型的入参 ：

/\*\*

\* 目标方法可以添加Map类型的参数（实际上也可以是Model类型或ModelMap类型的参数）

\*

\* \*/

@RequestMapping("/testMap")

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

System.***out***.println(map.getClass().getName());

//将names保存到request作用域

map.put("names", Arrays.*asList*("tom","jerry","mike"));

**return** ***SUCCESS***;

}

**处理模型数据之SessionAttributes注解:**

即将某些要进行传递的数据放在session域对象中

**@SessionAttributes(value= {"user"}, types= {String.class})**

/\*\*

以上的value中的值可以放多个，且如以上所示，将“user”与map集合中指定put的键值对中key相同的属性放入session属性范围。入参中带有map参数，即可将数据放入request属性范围。

\* **@SessionAttributes除了可以通过属性名指定需要放到会话中的属性外**（实际上使用的是value属性），

\* 还可以通过模型属性的对象类型指定哪些模型属性需要放到会话中（实际上使用的是types属性）

\*

\* 注意：**该注解只能放在类上面，而不能修饰方法**

\* \*/

@RequestMapping("testSessionAttributes")

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

User user = **new** User("WenPan", "1234566","wenpan@qq.com", 16);

map.put("user", user);

map.put("school", "西华大学");

**return** ***SUCCESS***;

}

**ModelAttribute注解之使用场景及其使用：**

ModelAttribute:是用来修改数据时，保持某一些数据不进行修改任然保持原来的值的一个操作。比如从数据库中查询出来的个人信息数据。密码不做修改。

/\*\*

\* **由@ModelAttribute标记的方法，会在每个目标方法执行前被SpringMVC调用**

\* \*/

@ModelAttribute

**public** **void** getUser(@RequestParam(value="id",required=**false**) Integer id,

Map<String,Object> map) {

**if**(id != **null**) {

//模拟从数据库中取出对象

User user = **new** User(1,"wenpan","123456","wenpan@qq.com",16);

System.***out***.println("从数据库中获取的一个对象： " + user);

map.put("user", user); //将user保存到属性范围？？？

}

}

@RequestMapping("/testModelAttribute")

**public** String testModelAttribute(User user) { //提交过来的数据自动为对象赋值

System.***out***.println("user: " + user);

**return** ***SUCCESS***;

}

**SpringMVC确定目标方法POJO类型入参的过程：**

1. **确定一个key：**
2. **若目标方法的POJO类型的参数没有使用@ModelAttribute 参数则key为POJO类名的第一个字母的小写**
3. **若使用了@ModelAttribute 来修饰，则key为@ModelAttribute 注解的value属性值。**
4. **在implicitModel（可理解为request）中查找key对应的对象，若存在，则作为入参传入。**
5. **若在@ModelAttribute 标记的方法中在map中保存过，且key和1（上一步）确定的key值一致，则会获取到。**
6. **若implicitModel中不存在key对应的对象，则检查当前的Handler是否使用@SessionAttributes 注解修饰，若使用了该注解，且@SessionAttributes 注解的value属性值中包含了key，则会从HttpSession中来获取key所对应的value值，若存在则直接传入到目标方法的入参中，若session属性范围内不存在则将抛出异常（即在@SessionAttributes 的value中存在于key相同的value值，但是该值在session属性范围中不存在对象）。**
7. **若Handler没有标识@SessionAttributes 注解或@SessionAttributes 注解的value值中不包含key，则会通过反射来创建POJO类型的参数，传入为目标方法的参数**
8. **SpringMVC会把key和POJO类型的对象保存到implicitModel中，进而会保存到request中。**

@ModelAttribute :则是将对象保存在request属性范围中

@SessionAttributes :则是将对象保存在session属性范围中

**ModelAttribute注解修饰POJO类型的入参：**

1. **有@ModelAttribute 注解标记的方法，会在每一个目标方法执行之前被springMVC调用！**
2. **@ModelAttribute 注解也可以来修饰目标方法POJO类型的入参，其value属性值有如下的作用：**
   1. **Springmvc会使用value属性值在implicitModel（可理解为request属性范围）中查找对应的对象，若存在则会直接传入到目标方法的入参中，例如：**

**public String testModelAttribute(@ModelAttribute("user")User user) { //提交过来的数据自动为对象赋值**

**System.*out*.println("修改 user: " + user);**

**return *SUCCESS*;**

**}**

**无论我们有没有添加@ModelAttribute("user") springmvc都会把入参放到请求域里面。 若在参数前没有放@ModelAttribute("user") ，则放在属性范围中的key就是类名User的首字母小写，若在参数前放了@ModelAttribute("user") 那么放到属性范围中的key就是对应的value值。**

* 1. **Springmvc会以value为key，POJO类型的对象为value，存入到request中**

范例：

/\*\*

\* 由@ModelAttribute标记的方法，会在每个目标方法执行前被SpringMVC调用

\* \*/

@ModelAttribute

**public** **void** getUser(@RequestParam(value="id",required=**false**) Integer id,

Map<String,Object> map) {

**if**(id != **null**) {

**//模拟从数据库中取出对象**

User user = **new** User(1,"wenpan","123456","wenpan@qq.com",16);

System.***out***.println("从数据库中获取的一个对象： " + user);

map.put("abc", user); //将user保存到request属性范围，方便POJO参数获取

}

}

**/\*\***

**\* 1.先得到一个key，然后检查request属性范围中是否存在key与map集合中保存的key相同的，若有则直接获取作为入参，若没有，则**

**\* 检查是否有@SessionAttributes 注解，若有，则在此注解的value值中查找是否有对应的相同key值，若在session属性范围中有这个**

**\* key存在，则进行获取作为入参，若没有相同的value属性值与key对应，则抛出一个异常**

**\* 若没有@SessionAttributes 注解，则直接利用反射新建一个入参类型的对象**

**\***

**\* 若在入参中添加@ModelAttribute("")注解的话，则入参是以@ModelAttribute（"xxx"）中的value值作为key来进行查找的。**

**\* 若在入参中不加@ModelAttribute（"xxx"）注解的话，则入参是以，入参类名的第一个字母小写作为key进行查找的**

**\***

**\* \*/**

@RequestMapping("/testModelAttribute")

**public** String testModelAttribute(**@ModelAttribute("mnxyz")** User user) { //提交过来的数据自动为对象赋值

System.***out***.println("修改 user: " + user);

**return** ***SUCCESS***;

}

**SessionAttributes注解引发的异常：**

在session属性范围中如果不存在与入参key对应的value值，则会抛出一个异常，

可以通过以下两种方式解决：

1. **在入参的前面加上@ModelAttribute("mnxyz") 将此注解中的value值设置得不跟sessionAttribute注解中的value值一样，这样在sessionAttribute中查找不到对应的值以后，springmvc就会通过反射新建一个入参对象。**
2. **加入@ModelAttribute()用于修饰一个方法，在该方法中将入参需要的对象，通过map保存在request属性范围中。**

**SpringMVC\_JstlView：**

**\_mvc\_view-controller标签：**

<!-- 配置直接转发的页面,即不用链接，直接就可以访问 -->

<!-- 可以直接响应转发的页面，而无需再经过handler的方法 -->

**<mvc:view-controller path=*"/success"* view-name=*"success"*/>**

<!-- 在实际开发中通常都需要配置mvc:annotation-driven标签 -->

<mvc:annotation-driven></mvc:annotation-driven>

在如上所示的程序中，可以直接通过浏览器访问/WEB-INF/view下的success页面。

但是在配置了**mvc:view-controller标签后，其他的handler方法跳转的页面就不能跳转了，需要添加<mvc:annotation-driven>标签才能正常访问了**。

**SpringMVC\_自定义视图：**

1. 先在springmvc的配置文件中进行配置自定义视图所需要的语句：

<!-- 配置视图解析器 BeanNameViewResolver:使用视图的名字来解析视图-->

<!-- 通过order属性来定义视图的优先级，order值越小，优先级越高. 自定义视图：配置了一下语句就可以使用自定义视图了 -->

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

<property name=*"order"* value=*"100"*></property>

</bean>

1. 然后编写一个链接的超链接或其他

<a href=*"****springmvc/testView****"*>Test View </a>

<br><br>

1. 在相应的java类中编写请求映射的函数：

@RequestMapping("testView")

**public** String testView() {

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

**return** "helloView";

}

1. 根据函数的返回值，由容器寻找容器中相应的bean。该bean必须要实现view接口

package com.wp.springmvc.views;

import java.util.Date;

import java.util.Map;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import org.springframework.stereotype.Component;

import org.springframework.web.servlet.View;

**@Component //一定要将该bean加入容器**

public class HelloView i**mplements View** {

@Override

public String getContentType() {

**return "text/html";**

}

@Override

public void render(Map<String, ?> arg0,

HttpServletRequest arg1, HttpServletResponse arg2) throws Exception {

arg2.getWriter().print("hello view ,time: " + new Date());

//可以在这里进行页面的跳转

//比如可以在这里执行自定义的要跳转的页面

//arg1.getRequestDispatcher("../success.jsp").forward(arg1, arg2);;

}

}

**SpringMVC\_重定向：**

如果返回字符串中带forword或redirect前缀时springmvc会对他们进行特殊的处理，将forword和redirect作为指示符，其他的当做URL来处理

Eg:redirect:success.jsp会完成一个从定向的的操作。

Forword:success.jsp会完成一个转发到susccess.jsp页面的操作

1. 先在相应的java类中编写一个请求映射的函数，并写好映射名

@RequestMapping("/testRedirect")

**public** String testRedirect() {

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

return "redirect:/success.jsp"; //重定向到根目录下的success.jsp页面

}

1. 在jsp页面编写一个到该函数的连接

<a href=*"springmvc/testRedirect"*>testRedirect </a>

<br><br>

3.当点击后就可以跳转到重定向的页面了。（类似servlet的转发）