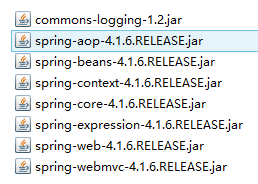
**一、SpringMVC 基础入门**

1.首先，导入SpringMVC需要的jar包。



2.添加Web.xml配置文件中关于SpringMVC的配置

<!--configure the setting of springmvcDispatcherServlet and configure the mapping-->

<servlet>

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

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

<init-param>

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

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

</init-param>

<!-- <load-on-startup>1</load-on-startup> -->

</servlet>

<servlet-mapping>

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

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

</servlet-mapping>

3.在src下添加springmvc-servlet.xml配置文件

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

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

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

*http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-4.1.xsd*

*http://www.springframework.org/schema/mvc http://www.springframework.org/schema/mvc/spring-mvc-4.1.xsd"*>

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

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

<!-- don't handle the static resource -->

<mvc:default-servlet-handler />

<!-- if you use annotation you must configure following setting -->

<mvc:annotation-driven />

<!-- configure the InternalResourceViewResolver -->

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

id=*"internalResourceViewResolver"*>

<!-- 前缀 -->

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

<!-- 后缀 -->

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

</bean>

</beans>

4.在WEB-INF文件夹下创建名为jsp的文件夹，用来存放jsp视图。创建一个hello.jsp，在body中添加“Hello World”。

5.建立包及Controller，如下所示

http://www.admin10000.com/UploadFiles/Document/201506/22/20150622201744832496.PNG

6.编写Controller代码

@Controller

@RequestMapping("/mvc")

**public** **class** mvcController {

@RequestMapping("/hello")

**public** String hello(){

**return** "hello";

}}

7.启动服务器，键入 http://localhost:8080/项目名/mvc/hello

**二、配置解析**

1.Dispatcherservlet

　　DispatcherServlet是**前置控制器**，配置在web.xml文件中的。拦截匹配的请求，Servlet拦截匹配规则要自已定义，把拦截下来的请求，依据相应的规则分发到目标Controller来处理，是配置spring MVC的第一步。

2.InternalResourceViewResolver

视图名称解析器

3.以上出现的注解

　　@Controller 负责注册一个bean 到spring 上下文中

　　@RequestMapping 注解为控制器指定可以处理哪些 URL 请求

**三、SpringMVC常用注解**

@Controller

　　负责注册一个bean 到spring 上下文中

@RequestMapping

注解为控制器指定可以处理哪些 URL 请求

@RequestBody

　　该注解用于读取Request请求的body部分数据，使用系统默认配置的HttpMessageConverter进行解析，然后把相应的数据绑定到要返回的对象上 ,再把HttpMessageConverter返回的对象数据绑定到 controller中方法的参数上

@ResponseBody

　　该注解用于将Controller的方法返回的对象，通过适当的HttpMessageConverter转换为指定格式后，写入到Response对象的body数据区

@ModelAttribute

　　在方法定义上使用 @ModelAttribute 注解：Spring MVC 在调用目标处理方法前，会先逐个调用在方法级上标注了@ModelAttribute 的方法

　　在方法的入参前使用 @ModelAttribute 注解：可以从隐含对象中获取隐含的模型数据中获取对象，再将请求参数 –绑定到对象中，再传入入参将方法入参对象添加到模型中

@RequestParam

　　在处理方法入参处使用 @RequestParam 可以把请求参 数传递给请求方法

@PathVariable

　　绑定 URL 占位符到入参

@ExceptionHandler

　　注解到方法上，出现异常时会执行该方法

@ControllerAdvice

　　使一个Contoller成为全局的异常处理类，类中用@ExceptionHandler方法注解的方法可以处理所有Controller发生的异常

**四、自动匹配参数**

//match automatically

@RequestMapping("/person")

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

System.out.println(name+" "+age);

**return** "hello";

}

**五、自动装箱**

1.编写一个Person实体类

**package** test.SpringMVC.model;

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

**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;

}

**private** String name;

**private** **int** age;

}

2.在Controller里编写方法

//boxing automatically

@RequestMapping("/person1")

**public** String toPerson(Person p){

System.out.println(p.getName()+" "+p.getAge());

**return** "hello";

}

**六、使用InitBinder来处理Date类型的参数**

//the parameter was converted in initBinder

@RequestMapping("/date")

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

System.*out*.println(date);

**return** "hello";

}

//At the time of initialization,convert the type "String" to type "date"

@InitBinder

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

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

}

**七、向前台传递参数**

//pass the parameters to front-end

@RequestMapping("/show")

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

Person p =**new** Person();

map.put("p", p);

p.setAge(20);

p.setName("jayjay");

**return** "show";

}

前台可在Request域中取到"p"

**八、使用Ajax调用**

//pass the parameters to front-end using ajax

@RequestMapping("/getPerson")

**public** **void** getPerson(String name,PrintWriter pw){

pw.write("hello,"+name);

}

@RequestMapping("/name")

**public** String sayHello(){

**return** "name";

}

前台用下面的Jquery代码调用

$(function(){

$("#btn").click(function(){

$.post("mvc/getPerson",{name:$("#name").val()},function(data){

alert(data);

});

});

});

**九、在Controller中使用redirect方式处理请求**

//redirect

@RequestMapping("/redirect")

**public** String redirect(){

**return** "redirect:hello";

}

**十、文件上传**

1.需要导入两个jar包

http://www.admin10000.com/UploadFiles/Document/201506/22/20150622201744368320.PNG

2.在SpringMVC配置文件中加入

<!-- upload settings -->

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

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

</bean>

3.方法代码

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

**public** String upload(HttpServletRequest req) **throws** Exception{

MultipartHttpServletRequest mreq = (MultipartHttpServletRequest)req;

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

String fileName = file.getOriginalFilename();

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

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

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

fos.write(file.getBytes());

fos.flush();

fos.close();

**return** "hello";

}

4.前台form表单

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

<input type="file" name="file"><br>

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

</form>

**十一、使用@RequestParam注解指定参数的name**

@Controller

@RequestMapping("/test")

**public** **class** mvcController1 {

@RequestMapping(value="/param")

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

@RequestParam(value="name")String name){

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

**return** "/hello";

}

}

**十二、RESTFul风格的SringMVC**

1.RestController

@Controller

@RequestMapping("/rest")

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

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

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

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

**return** "/hello";

}

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

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

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

**return** "/hello";

}

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

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

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

**return** "/hello";

}

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

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

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

**return** "/hello";

}

}

2.form表单发送put和delete请求

在web.xml中配置

<!-- 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>

在前台可以用以下代码产生请求

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

</form>

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

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

</form>

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

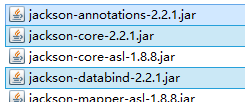
<input type=*"hidden"* name=*"\_method"* value=*"DELETE"*>

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

</form>

**十三、返回json格式的字符串**

1.导入以下jar包



2.方法代码

@Controller

@RequestMapping("/json")

**public** **class** jsonController {

@ResponseBody

@RequestMapping("/user")

**public** User get(){

User u = **new** User();

u.setId(1);

u.setName("jayjay");

u.setBirth(**new** Date());

**return** u;

}

}

**十四、异常的处理**

1.处理局部异常（Controller内）

@ExceptionHandler

**public** ModelAndView exceptionHandler(Exception ex){

ModelAndView mv = **new** ModelAndView("error");

mv.addObject("exception", ex);

System.*out*.println("in testExceptionHandler");

**return** mv;

}

@RequestMapping("/error")

**public** String error(){

**int** i = 5/0;

**return** "hello";

}

2.处理全局异常（所有Controller）

@ControllerAdvice

**public** **class** testControllerAdvice {

@ExceptionHandler

**public** ModelAndView exceptionHandler(Exception ex){

ModelAndView mv = **new** ModelAndView("error");

mv.addObject("exception", ex);

System.*out*.println("in testControllerAdvice");

**return** mv;

}

}

3.另一种处理全局异常的方法

　　在SpringMVC配置文件中配置

<!-- configure SimpleMappingExceptionResolver -->

<bean class=*"org.springframework.web.servlet.handler.SimpleMappingExceptionResolver"*>

<property name=*"exceptionMappings"*>

<props>

<prop key=*"java.lang.ArithmeticException"*>error</prop>

</props>

</property>

</bean>

**十五、设置一个自定义拦截器**

1.创建一个MyInterceptor类，并实现HandlerInterceptor接口

**public** **class** MyInterceptor **implements** HandlerInterceptor {

@Override

**public** **void** afterCompletion(HttpServletRequest arg0,

HttpServletResponse arg1, Object arg2, Exception arg3)

**throws** Exception {

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

}

@Override

**public** **void** postHandle(HttpServletRequest arg0, HttpServletResponse arg1,

Object arg2, ModelAndView arg3) **throws** Exception {

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

}

@Override

**public** **boolean** preHandle(HttpServletRequest arg0, HttpServletResponse arg1,

Object arg2) **throws** Exception {

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

**return** **true**;

}

}

2.在SpringMVC的配置文件中配置

<!-- interceptor setting -->

<mvc:interceptors>

<mvc:interceptor>

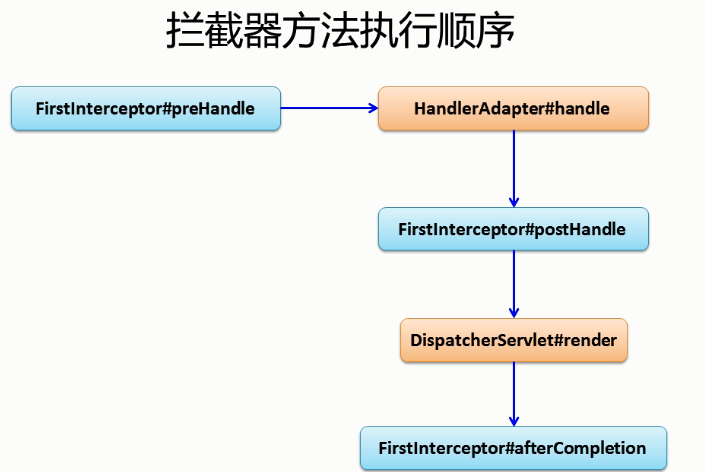
<mvc:mapping path=*"/mvc/\*\*"*/>

<bean class=*"test.SpringMVC.Interceptor.MyInterceptor"*></bean>

</mvc:interceptor>

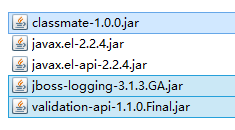
</mvc:interceptors>

3.拦截器执行顺序



**十六、表单的验证（使用Hibernate-validate）及国际化**

1.导入Hibernate-validate需要的jar包

 http://www.admin10000.com/UploadFiles/Document/201506/22/20150622201744680139.PNG

2.编写实体类User并加上验证注解

**public** **class** User {

**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;

}

@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;

}

ps:@Past表示时间必须是一个过去值

3.在jsp中使用SpringMVC的form表单

<form:form action=*"form/add"* method=*"post"* modelAttribute=*"user"*>

id:<form:input path=*"id"*/><form:errors path=*"id"*/><br>

name:<form:input path=*"name"*/><form:errors path=*"name"*/><br>

birth:<form:input path=*"birth"*/><form:errors path=*"birth"*/>

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

</form:form>

ps:path对应name

4.Controller中代码

@Controller

@RequestMapping("/form")

**public** **class** formController {

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

**public** String add(@Valid User u,BindingResult br){

**if**(br.getErrorCount()>0){

**return** "addUser";

}

**return** "showUser";

}

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

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

map.put("user",**new** User());

**return** "addUser";

}

}

ps:

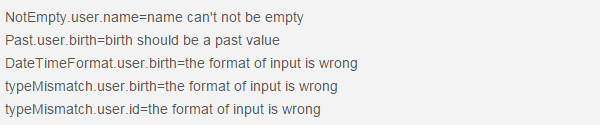
　　1.因为jsp中使用了modelAttribute属性，所以必须在request域中有一个"user".

　　2.@Valid 表示按照在实体上标记的注解验证参数

　　3.返回到原页面错误信息回回显，表单也会回显

　　5.错误信息自定义

在src目录下添加locale.properties



在SpringMVC配置文件中配置

<!-- configure the locale resource -->

<bean id=*"messageSource"* class=*"org.springframework.context.support.ResourceBundleMessageSource"*>

<property name=*"basename"* value=*"locale"*></property>

</bean>

6.国际化显示

在src下添加locale\_zh\_CN.properties

username=账号

password=密码

locale.properties中添加

username=user name

password=password

创建一个locale.jsp

<body>

<fmt:message key=*"username"*></fmt:message>

<fmt:message key=*"password"*></fmt:message>

</body>

在SpringMVC中配置

<!-- make the jsp page can be visited -->

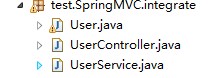
<mvc:view-controller path=*"/locale"* view-name=*"locale"*/>

让locale.jsp在WEB-INF下也能直接访问

　　最后，访问locale.jsp，切换浏览器语言，能看到账号和密码的语言也切换了

**十七、压轴大戏--整合SpringIOC和SpringMVC**

1.创建一个test.SpringMVC.integrate的包用来演示整合，并创建各类



2.User实体类

**public** **class** User {

**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;

}

@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;

}

3.UserService类

@Component

**public** **class** UserService {

**public** UserService(){

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

}

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

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

}

}

4.UserController

@Controller

@RequestMapping("/integrate")

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

@Autowired

**private** UserService userService;

@RequestMapping("/user")

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

System.*out*.println(u);

userService.save();

**return** "hello";

}

}

5.Spring配置文件

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

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

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

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

*http://www.springframework.org/schema/util/spring-util-4.0.xsd*

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

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

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

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

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

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

</beans>

在Web.xml中添加配置

<!-- 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>

6.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"*

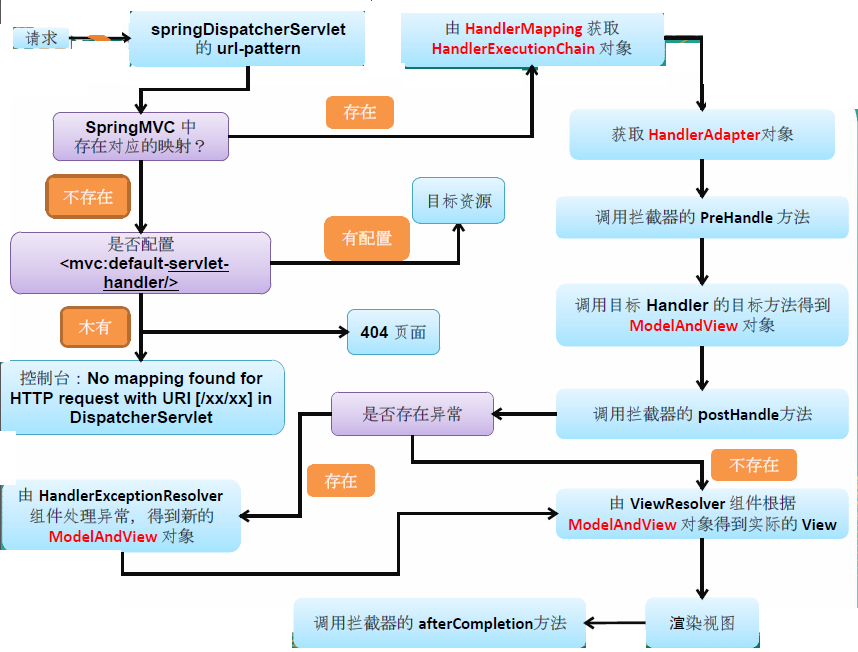
expression=*"org.springframework.stereotype.Controller"*/>

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

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

</context:component-scan>

**十八、SpringMVC详细运行流程图**



**十九、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。

**附录：**

使用Spring MVC的注解及其用法和其它相关知识来实现控制器功能。

之前在使用Struts2实现MVC的注解时，是借助struts2-convention这个插件，如今我们使用Spring自带的spring-webmvc组件来实现同样的功能，而且比之以前更简单。另外，还省掉了整合两个框架带来的不稳定因素。

  对于Spring MVC框架，我主要讲一下它的常用注解，再结合一些示例进行说明，方便大家能够快速理解。

**一、Spring MVC常用注解说明**

**@Controller**

  在类上面定义，表明该类为控制器，返回字符串与redirect:xxx

**@RequestMapping**

  在类或方法上面使用此注解，设置URL访问地址。它有两个属性，value指定访问路径，method指定指定请求方式，请求方式在RequestMethod这个类中，全部以常量形式定义，它默认使用GET请求。

**@RequestParam**

  指定Request请求参数，在方法参数中定义，相当于传统的request.getParameter()。

**@PathVariable**

  获取URL访问路径变量，这是Spring MVC 3.0框架才加入的特性，基于RESTful风格的URL访问路径。

**@ModelAttribute**

全局式的方法，在一组URL访问路径中，每次都会执行，方法返回结果保存在module会话中。

**@Service**

  在类上面定义，指定被注解的类是业务逻辑组件，如果不指定具体的Bean ID，则采用默认命名方式，即类名的首字母小写。之前在SSH2中，大象曾对Dao组件使用@Repository，本例只有业务层，所以就只用@Service注解。

**@Autowired**

  IoC自动注入功能，替换以前的set写法，在SSH2中就已经开始使用了。

**@Qualifier**

  对同一接口类有不同实现指定具体的实现类。

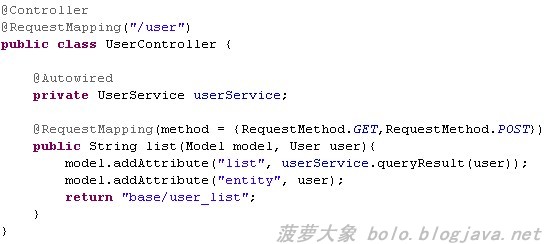
**@ResponseBody**

  同样定义在方法上，Ajax调用声明，指定方法返回结果为Ajax回调函数结果。这是Spring MVC 3.0框架中增加的一个新特性。

**@InitBinder**

  初始化数据绑定与类型转换，将传入的参数转换为自定义类型，或者对参数进行自定义处理。

**二、示例**



@RequestMapping在类名上面定义，相当于指定的URL是此控制器内的所有其它访问路径的父路径。如果在某个方法上面定义@RequestMapping注解，则相对于父路径来说，是其子路径。如果不定义value值，那么按父路径访问就会被默认执行。但请注意，默认的访问方式只能有一个。

  对于UserController的list方法REST访问URL为http://localhost:8080/ssm3/user，而且它同时接收GET和POST两种请求。另外，Spring MVC 3.0有一个很灵活的特性，可以自定义方法参数。看看list方法，我设定了两个参数，一个Model，一个User对象。Model是用来渲染数据，生成页面用的。相当于request.setAttribute，你可以这样理解，但不能就这样认为，Model以及另一个ModelMap，都是作为视图模型传递参数的，它们的作用域为request。除此之外，你还可以定义HttpServletRequest、HttpServletResponse等等各种各样的参数。

如果一个类还要定义其它资源访问怎么办呢？请看下面的RoleController：

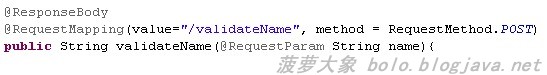


  在RoleController上定义了全局路径/role，这样一来，对于和角色相关的资源都会以/role开头，比如创建角色/role/new；编辑角色/role/edit/{id}等等。

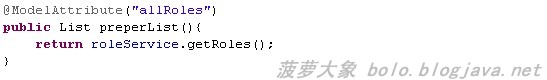
  上图edit方法中的{id}写法，就是RESTful URL风格，与@PathVariable搭配来一起实现该功能。它表示所请求的URL中，可以将变量值作为参数进行动态的传递。例如：http://localhost:8080/ssm3/role/edit/1，另外，除了可以用数字，还可以用字符串，还可以多定义几个变量：/role/edit/{id}/{type}等等。

**每个方法的返回值，其实都对应着一个结果页面**，这一点和struts2-convention这个插件很相像。本例使用FreeMarker模板引擎作为展示层，页面的后缀为.html，页面中除了标准的HTML之外，其余的数据填充，条件判断之类，都要用到FreeMarker指令。

  对于save方法返回值写法表示的是重定向，相当于执行http://localhost:8080/ssm3/role，而这个URL对应的其实就是RoleController这个类里面list方法。如果要带上参数之类的，一定要符合所定义的REST资源路径才可以。



  @ResponseBody用来标识Ajax方法调用，在上面这个方法中，用到了@RequestParam注解，它的作用就和request.getParameter("name")一样。Spring MVC框架支持好几种返回格式，例如：String/JSON/XML等等。不过以这种格式的字符串值形式返回是最简便的一种方式，而且利用JavaScript解析也十分方便。页面调用的时候请用jQuery的$.ajax()这种原生方式来定义，这种写法不会出问题，也很灵活，而且其它几种方式最终也是调用它来完成请求。



  对于拥有相同的一组访问规则的URL，如果**都需要获得相同的数据**，则**使用@ModelAttribute注解**。以RoleController为例，上面这个注解与方法的含义，相当于是在它里面所有的访问路径方法中都调用这个写法：module.addAttribute(“allRoles”,roleService.getRoles())。也即，不管是访问create还是edit，都会执行preperList，都会获得allRoles这个List。



  注册自定义类型编辑器，在Spring MVC中，对于时间类型，框架不会自动帮你转换绑定，需要你自己来定义属性编辑器。除此之外，还可以对某些特殊字符进行转义符处理，都可以放在@InitBinder注解的方法中进行。如果所有的Controller都需要注册相同的属性编辑器，则可以实现WebBindingInitializer接口，定义一个全局的属性编辑器。

**三、在web容器中部署**

想要让Spring MVC框架帮助我们完成工作，就需要在Web容器中配置好它。



**DispatcherServlet**是Spring MVC的核心，是**处理一切请求转发的核心控制器**。大象曾在本系列的第二篇文章中就详细描述了Spring MVC的流程结构，如果没什么印象的话请再去看看。

  Spring MVC有一个默认规则，Web容器启动之后，会自动查找/WEB-INF/<servlet-name>.xml这个Spring类型的配置文件。如果想自定义配置文件路径，就按上面的写法，指定contextConfiglocation这个属性，大象采用maven构建项目，所以servlet-context.xml这个配置文件放在resource目录下。

**四、MVC配置**

  Spring MVC 3.0对使用和配置作了较大的改进，除了提供注解来简化控制器的开发之外，在配置文件上面也进行了简化。



  基于Spring MVC注解的配置就是上面这两行，还有一种更简化的配置写法是只写这一句：<mvc:annotation-driven />就可以了，Spring启动的时候会自动注册上面这两个bean。为什么大象要在这里显示的注册两个bean呢？因为，我们在真正使用的时候，一般来说，使用默认的方式满足不了我们的系统或业务要求。比如**拦截器**，比如**数据验证**，比如**返回消息格式转换**等等一些自定义设置。他们都需要配置在这两个bean里面。因为本例是用来作为入门教程，所以这些东西都没有加进来。

  DefaultAnnotationHandlerMapping这个类是将所有标注了@RequestMapping注解的Controller类，都放到了一个HandlerMapping对象中，当有请求时，就在这个对象中进行查找是否有与之匹配的路径，AnnotationMethodHandlerAdapter是管理所有@RequestMapping注解的方法。

**五、类包详解**

org.springframework.aop- 3.0.0.RELEASE--------------------Spring的面向切面编程,提供AOP(面向切面编程)实现

org.springframework.asm- 3.0.0.RELEASE--------------------Spring独立的asm程序,相遇Spring2.5.6的时候需要asmJar 包.3.0开始提供他自己独立的asmJar

org.springframework.aspects- 3.0.0.RELEASE----------------Spring提供对AspectJ框架的整合\

org.springframework.beans- 3.0.0.RELEASE------------------SpringIoC(依赖注入)的基础实现

org.springframework.context.support- 3.0.0.RELEASE--------Spring-context的扩展支持,用于MVC方面

org.springframework.context- 3.0.0.RELEASE----------------Spring提供在基础IoC功能上的扩展服务，此外还提供许多企业级服务的支持，如邮件服务、任务调度、JNDI定位、EJB集成、远程访问、缓存以及各种视图层框架的封装等

org.springframework.core- 3.0.0.RELEASE-------------------Spring3.0的核心工具包

org.springframework.expression- 3.0.0.RELEASE-------------Spring表达式语言

org.springframework.instrument.tomcat- 3.0.0.RELEASE------Spring3.0对Tomcat的连接池的集成

org.springframework.instrument- 3.0.0.RELEASE-------------Spring3.0对服务器的代理接口

org.springframework.jdbc- 3.0.0.RELEASE-------------------对JDBC的简单封装

org.springframework.jms- 3.0.0.RELEASE--------------------为简化JMS API的使用而作的简单封装

org.springframework.orm- 3.0.0.RELEASE--------------------整合第三方的ORM框架，如hibernate,ibatis,jdo，以及spring的JPA实现

org.springframework.oxm-3.0.0.RELEASE--------------------Spring 对Object/XMl的映射支持,可以让Java与XML之间来回切换

org.springframework.test- 3.0.0.RELEASE--------------------对Junit等测试框架的简单封装

org.springframework.transaction- 3.0.0.RELEASE-------------为JDBC、Hibernate、JDO、JPA等提供的一致的声明式和编程式事务管理

org.springframework.web.portlet- 3.0.0.RELEASE-------------SpringMVC的增强

org.springframework.web.servlet- 3.0.0.RELEASE-------------对JEE6.0 Servlet3.0的支持

org.springframework.web.struts- 3.0.0.RELEASE--------------整合Struts的时候的支持

org.springframework.web- 3.0.0.RELEASE--------------------SpringWeb下的工具包