1. SpringMVC的数据响应
2. SpringMVC的数据响应方式

SpringMVC中在调用了Controller中的方法后，可能会跳转页面，也可能返回字符串。

1. 页面跳转

跳转页面的方式又分为直接返回字符串和返回ModelAndView对象。

1. 直接返回字符串

返回的字符串代表的是要跳转的页面资源名称，这种方式会将返回的字符串和视图解析器的前后缀拼接后跳转，如

在Controller的方法中返回 “index”，视图解析器中定义的前缀为 /views，后缀为.jsp，则拼接起来就是跳转的资源名称：/views/index.jsp

Controller方法定义

@RequestMapping(“/quick”)

public String quickMethod(){

return “index”;

}

视图解析器配置：

<property name=”prefix” value=”/views”/>

<property name=”suffix” value=”.jsp”/>

跳转分为 转发forward和 重定向redirect，默认是转发，要使用重定向 ，要在名称前加上redirect:

1. 返回ModelAndView对象

ModelAndView对象分为Model模型对象，View视图对象。模型的作用是封装数据，视图的作用是展示数据。因此，我们可以创建一个ModelAndView对象，然后分别设置模型和视图，如设置视图的名称为success，它会根据视图解析器的前后缀配置找到success文件，同时设置模型中的数据，键值对形式，视图中可以使用模型的数据。

@RequestMapping(value=”/quick”)

public ModelAndView save(){

ModelAndView modelAndView = new ModelAndView();

//设置模型数据

modelAndView.addObject(“username”, “wxx”);

//设置视图名称

modelAndView.setViewName(“success”);

return modelAndView;

}

也可以在方法的参数中传入一个ModelAndView参数，让SpringMVC来注入这个对象，方法中可以直接使用。

public ModelAndView save(ModelAndView modelAndView){}

或者在方法中只传入Model对象，返回String字符串，即视图的名称，这种方式与之前的效果是相同的：

public String save(Model model){

model.addAttribute(“username”, “xxx”);

return “success”;

}

前端控制器收到ModelAndView对象后，会发送给视图解析器解析视图，再渲染视图返回给浏览器。

1. 回写数据
2. 直接返回字符串

客户端发送请求给服务器，要想直接返回字符串给客户端，只需要使用response.getWriter().print(“hello world”)即可，因此，在Controller的方法参数中传入response对象即可，SpringMVC会自动注入response对象。

@RequestMapping(“/quick”)

public void quickMethod(HttpServletResponse response) throws IOException{

response.getWriter().print(“hello world”);

}

但是，这种直接调用response对象，会使得代码的耦合性提升，应尽量避免使用。

我们可以直接返回字符串，但是这样会使得SpringMVC框架认为这是要跳转。因此，我们需要使用@ResponseBody注解告知SpringMVC框架，这个方法返回的字符串不是跳转，而是直接在http响应体中返回。

@RequestMapping(value=”/quick”)

@ResponseBody

public String save() throws IOException{

return “hello world”;

}

1. 返回对象或集合

执行完Controller的方法后，可能会返回对象或集合，SpringMVC会自动将他们转换为json格式的数据。

首先，我们知道Controller中的方法是由处理器映射器来调用的，因此，像视图解析器一样，我们也要配置处理器映射器的消息转换参数来自动转换数据格式，同样是在spring-mvc.xml中进行配置：

如，指定使用jackson进行对象或集合的转换

<bean class=”org.springfreamework.web.servlet.mvc.mthod.annotation.RequestMappingHandlerAdapter”>

<property name=”messageConverters”>

<list>

<bean class=”org.springframework.http.converter.json.MappingJackson2HttpMessageConverter”>

</list>

</property>

</bean>

1. 简单的配置方式

上面我们介绍了视图解析器，处理器适配器的配置方式，看起来都比较繁琐，这里介绍一个简单的方式自动配置。在spring-mvc.xml配置文件中，使用mvc的注解驱动即可替代上述的配置，当然首先要引入mvc的命名空间：

<mvc:annotation-driven/>

这个注解驱动可以自动加载RequestMappingHandlerMapping(处理器映射器)， RequestMappingHandlerAdapter（处理器适配器），因此spring-mvc.xml在创建时，就应该先写上这个注解驱动。

1. SpringMVC获得请求数据
2. 获得请求参数

客户端请求参数的格式是： name=value&name=value…，服务器端要获得请求的参数，有时还要对数据进行封装，SpringMVC可以接受如下类型的参数：

1. 基本类型参数

客户端发送基本类型数据，服务器端直接获取这些数据，Controller中的业务方法想要使用这些参数，那么方法的参数名称要和请求参数的name一致，参数值就会自动映射匹配。

如，请求参数为 username=zz&age=12

则，方法需要定义为：

@RequestMapping(“/quick”)

@ResponseBody

public void quickMethod(String username, int age){}

1. POJO类型参数

在Controller中的业务方法中可以使用POJO对象，POJO对象的参数也可以通过请求参数获得，只要POJO对象的属性名和请求参数的name都对应一致即可，参数值会自动映射匹配。

如，请求参数为 username=zz&age=12

定义一个POJO对象User类为：

public class User{

private String username;

private int age;

}

Controller中的业务方法定义为：

@RequestMapping(“/quick”)

@ResponseBody

public void quickMethod(User user){}

1. 数组类型参数

定义一个数组，Controller中的业务方法使用该数组，只要该数组的名称和请求参数的name一致即可，参数值后自动映射匹配。

请求参数： strs=222

方法定义：

public void quickMethod(String[] strs){}

1. 集合类型参数

如果要从请求中获取集合参数，则需要将集合的参数包装到一个POJO中，一般称该POJO对象为一个VO对象，该对象存储着集合。

我们要通过请求向服务器传递集合参数，但是整个集合是无法传递的，因此我们将集合封装到一个POJO中，且集合中的参数是一个一个地传入的。且集合的名称要对应起来。

如，定义了一个User类对象，其中有属性username和age，我们要传递的是一个集合对象userList，其中存储的User对象。因此请求中传递参数时要一个个地传递User对象的username参数和age参数，先组合成User对象，再由多个User对象组合成一个userList集合对象，这样就可以传递了。

1. 请求数据乱码问题

当POST请求时，数据会出现乱码，我们可以设置一个过滤器来进行编码的过滤，指定以什么编码格式进行编码。

<filter>

<filter-name>CharacterEncodingFilter</filter-name>

<filter-class>org.springfamework.web.filter.CharacterEncodingFilter</filter-class>

<init-param>

<param-name>encoding</param-name>

<param-value>URF-8</param-value>

</init-param>

</filter>

<filter-mapping>

<filter-name>CharacterEncodingFilter</filter-name>

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

</filter-mapping>

1. 参数绑定注解@RequestParam

当请求的参数名称和Controller中业务方法的参数名称不一致时，就需要通过@RequestParam注解显示地绑定两个不同的参数。

如，请求中传入的参数名称为name，而方法中的参数名称为username，则需要在方法的参数中使用注解@RequestParam(“name”)绑定到username上面：

@RequestMapping(“/quick”)

@ResponseBody

public void quickMethod(@RequestParam(“name”)String username){}

@RequestParam注解包含以下的参数：

1. value

请求参数名称，默认的参数

2, required

在此指定的请求参数是否必须包括，默认是true，提交时如果没有此参数则会报错。

1. defaultValue

当没有指定请求参数时，则使用指定的默认值赋值。

1. 获得Restful风格的参数
2. Restful概述

Restful是一种软件架构风格，设计风格，提供了一组设计原则和约束条件，主要用于客户端和服务器之间交互的软件，基于这个风格设计的软件可以更加简洁，更有层次，更易于实现缓存机制。

1. 请求方式

Restful风格的请求是使用”url+请求方式”表示一次请求的目的，HTTP协议中四个表示操作方式的词：

GET : 用于获取资源

POST: 用于新建资源

PUT: 用于更新资源

DELETE: 用于删除资源

如，

/user/1 GET :表示得到id=1的user

/user/1 DELETE:表示删除id=1的user

/user/1 PUT: 表示更新id=1的user

/user POST: 表示新增user

1. 获取GET参数实例

在url地址 /user/1 中的1就是要获取的请求参数，在SpringMVC中可以使用占位符进行参数绑定。可以写成 /user/{id}，占位符{id}就是1的值，在Controller的业务方法中我们可以使用@PathVariable注解进行占位符的匹配获取工作。

如，请求url为： http://localhost:8080/springmvc/quick/zhangsan

则要获取 zhangsan 这个参数，通过占位符{name}就可以获取，如下所示：

@RequestMapping(“/quick/{name}”)

@ResponseBody

public void quickMethod(@PathVariable(value=”name” required=true) String name){

System.ou.println(name);

}

1. 自定义类型转换器

客户端中用户会输入各种类型的数据，我们要将用户的输入转换为服务器可以接受的数据类型，因此我们需要自定义一些类型转换器。

1. 自定义类型转换器的步骤
2. 定义转换器类实现Converter接口

自定义一个转换器类，将客户端输入的String字符串转换为Date格式的数据，

public class DataConverter implements Converter<String , Date>{

public Date convert(String dateStr){

SimpleDateFormat format = new SimpleDateFormat(“yyyy-MM-dd”);

Date date = null;

try{

date = format.parse(dateStr);\_

}catch(ParseException){

e.printStackTrace();

}

return date;

}

}

1. 在配置文件spring-mvc.xml中声明转换器

定义完转换器后，需要在spring-mvc.xml中配置转换器，在最外围创建一个转换器工厂Bean对象，使用这个工厂来定义一些转换器，list子标签下创建转换器Bean对象，class属性为自定义的转换器的全类名。

<bean id=”conversionService” class=”org.springframework.context.support.ConversionServiceFactoryBean”>

<property name=”converters”>

<list>

<bean class=”com.spring.converter.DateConverter”></bean>

</list>

</property>

</bean>

1. 在<annotation-driven>中引用转换器

同样是在spring-mvc.xml中的<annotation-driven>注解驱动洞，设置conversion-service属性的值为上面定义的转换器工厂Bean对象的id属性值。这是用于指定转换器。

<mvc::annotation-driven conversion-service=”conversionService”>

1. 获取请求头的参数

客户端发送请求的请求头中的参数可以在Controller的方法中获取，使用@RequestHeader注解，获取请求头中的参数。注解中的属性有value，请求头的名称，注意，这里的名称必须和请求头中的名称一致；required是否必须携带此请求头。

如，获取User-Agent参数：

@RequestMapping(value=”/quick”)

@ResponseBody

public void save(@RequestHeader(value=”User-Agent”, required=false) String user-agent){}

1. 获取Cookie参数

虽然说Cookie参数也是请求头中的参数，但是它有单独的注解@CookieValue，因为Cookie参数是键值对形式的，且可能有多个Cookie，它的属性也有value，Cookie键值对的键名，reuqired是否必须携带该Cookie。

如，有一个Cookie的键名为JSESSIONID，则在Controller方法中获取该Cookie：

@RequestMapping(“/quick”)

@ResponseBody

public void save(@CookieValue(value=”JSESSIONID”) String jsessionId){}

1. 文件上传
2. 文件上传客户端三要素
3. 表单项 type=”file”
4. 表单的提交方式为“POST”
5. 表单的enctype属性是多部分表单形式，即enctype=”myltipart/form-data”

<form action=”${pageContext.request.contextPath}/quick” method=”post” enctype=”multipart/form-data”>

名称：<input type=”text” name=”name”><br>

文件：<input type=”file” name=”file”><br>

<input type=”submit” value=”提交”><br>

</form>

1. 文件上传原理

（1） 当form表单的enctype=”application/x-www.form-urlencoded”时，form表单的正文内容格式是： key=value&key=value键值对形式

（2）当form表单的enctype=”Multipart/form-data”时，请求正文内容就变为多部分表单形式，此时request.getParameter()将会失效。

1. 单文件上传步骤
2. 导入fileupload和io坐标

在pom.xml中导入 commons-fileupload的坐标和commons-io的坐标。

1. 配置文件上传解析器

在spring-mvc.xml中配置文件上传解析器：

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

<!—上传文件总大小-->

<property name=”maxUploadSize” value=”5242800”/>

<!—上传单个文件的大小-->

<property name=”maxUploadSizePerFile” value=”5242800”/>

<!—上传文件的编码类型 -->

<property name=”defaultEncoding” value=”UTF-8”>

</bean>

1. 编写文件上传代码

在Controller的方法中编写文件上传代码，文件会被封装为一个MultipartFile对象，且该对象的名称要和表单中文件的名称一致，其他的参数同样名称要一致。

@RequestMappint(value=”/quick”)

@ResponseBody

public void save(String username, MultipartFile uploadFile){

//获取上传文件的名称

String originalFilename = uploadFile.getOriginalFilename();

uploadFile.transferTo(new File(“C:\\upload\\” + originalFilename));

}

1. 多文件上传实现

多文件上传，只需要将表单页面修改为多个文件上传项，每个文件名称相同，再将Controller中的文件上传代码的方法参数MultipartFile类型修改为MultipartFile[]数组即可，通过for循环为每个文件设置存储位置。

@RequestMapping(“quick”)

@ResponseBody

public void quickMethod(String name, MultipartFile[] uploadFiles){

for(MultipartFile uploadFile : uploadFiles){

String originalFilename = upliadFile.getOriginalFilename();

uploadFile.transferTo(new File(“C:\\upload\\” + originalFilename));

}

}