目录

1、@RequestMapping

1.1、 value， method；

value/method 示例

1.2、 consumes，produces；

 consumes/produces 示例

1.3、 params，headers；

 params/headers 示例

2、@PathVariable

2.1、定义单个URL变量

2.2、定义多个URL变量

3、@RequestHeader、@CookieValue

4、@RequestParam

5、@RequestBody

6、@SessionAttributes, @ModelAttribute

6.1、@SessionAttributes:

6.2、@ModelAttribute

7、@RestController

8、其他RabbitMQ

1、@RequestMapping

@RequestMapping是一个用来处理请求地址映射的注解，可用于类或方法上。用于类上，表示类中的所有响应请求的方法都是以该地址作为父路径。@RequestMapping用在类上可以没有，但是用在方法上必须有。

窄化请求映射：在class上添加@RequestMapping(url)指定通用请求前缀， 限制此类下的所有方法请求url必须以请求前缀开头，通过此方法对url进行分类管理。

如下：

@RequestMapping放在类名上边，设置请求前缀

@Controller

@RequestMapping("/item")

方法名上边设置请求映射url：

@RequestMapping放在方法名上边，如下：

@RequestMapping("/queryItem ")

访问地址为：/item/queryItem

@RequestMapping是handler method 参数绑定常用的注解,我们根据他们处理的Request的不同内容部分分为四类：（主要讲解常用类型）

A、处理request url 部分（这里指uri template中variable，不含queryString部分）的注解：   @PathVariable;

B、处理request header部分的注解：   @RequestHeader, @CookieValue;

C、处理request body部分的注解：@RequestParam,  @RequestBody;

D、处理attribute类型是注解： @SessionAttributes, @ModelAttribute;

@RequestMapping注解有六个属性，下面我们把她分成三类进行说明

1.1、 value， method；

value：     指定请求的实际地址，指定的地址可以是URI Template 模式（后面将会说明）；

method：  指定请求的method类型， GET、POST、PUT、DELETE等；

value/method 示例

默认RequestMapping("....str...")即为value的值；

@Controller

@RequestMapping("/appointments")

public class AppointmentsController {

private final AppointmentBook appointmentBook;

@Autowired

public AppointmentsController(AppointmentBook appointmentBook) {

this.appointmentBook = appointmentBook;

}

@RequestMapping(method = RequestMethod.GET)

public Map<String, Appointment> get() {

return appointmentBook.getAppointmentsForToday();

}

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

public Map<String, Appointment> getForDay(@PathVariable @DateTimeFormat(iso=ISO.DATE) Date day, Model model) {

return appointmentBook.getAppointmentsForDay(day);

}

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

public AppointmentForm getNewForm() {

return new AppointmentForm();

}

@RequestMapping(method = RequestMethod.POST)

public String add(@Valid AppointmentForm appointment, BindingResult result) {

if (result.hasErrors()) {

return "appointments/new";

}

appointmentBook.addAppointment(appointment);

return "redirect:/appointments";

}

}

value的url值为以下三类：

A） 可以指定为普通的具体值；

B)  可以指定为含有某变量的一类值(URI Template Patterns with Path Variables)；

C) 可以指定为含正则表达式的一类值( URI Template Patterns with Regular Expressions);

例如-用户名只可能包含小写字母，数字，下划线，我们希望：

/user/fpc是一个合法的URL

/user/#$$$则不是一个合法的URL

 example B

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

public String findOwner(@PathVariable String ownerId, Model model) {

Owner owner = ownerService.findOwner(ownerId);

model.addAttribute("owner", owner);

return "displayOwner";

}

example C

@RequestMapping("/spring-web/{symbolicName:[a-z-]+}-{version:\d\.\d\.\d}.{extension:\.[a-z]}")

public void handle(@PathVariable String version, @PathVariable String extension) {

// ...

}

}

/\*\*除了简单地定义{username}变量，还可以定义正则表达式进行更精确的控制，定义语法是{变量名：正则表达式}[a-zA-Z0-9\_]+是一个正则表达式，表示只能包含小写字母，大写字母，数字，下划线。如此设置URL变量规则后，不合法的URL则不会被处理，直接由SpringMVC框架返回404Not Found。\*/

@RequestMapping("/user/{username:[a-zA-Z0-9\_]+}/blog/{blogId}")

1.2、 consumes，produces；

consumes： 指定处理请求的提交内容类型（Content-Type），例如application/json, text/html;

produces:    指定返回的内容类型，仅当request请求头中的(Accept)类型中包含该指定类型才返回；

 consumes/produces 示例

cousumes的样例

@Controller

@RequestMapping(value = "/pets", method = RequestMethod.POST, consumes="application/json")

public void addPet(@RequestBody Pet pet, Model model) {

// implementation omitted

}

该方法仅处理request Content-Type为“application/json”类型的请求。

produces的样例

@Controller

@RequestMapping(value = "/pets/{petId}", method = RequestMethod.GET, produces="application/json")

@ResponseBody

public Pet getPet(@PathVariable String petId, Model model) {

// implementation omitted

}

该方法仅处理request请求中Accept头中包含了"application/json"的请求，同时暗示了返回的内容类型为application/json;

1.3、 params，headers；

params： 指定request中必须包含某些参数值是，才让该方法处理。

headers： 指定request中必须包含某些指定的header值，才能让该方法处理请求。

 params/headers 示例

@Controller

@RequestMapping("/owners/{ownerId}")

public class RelativePathUriTemplateController {

@RequestMapping(value = "/pets/{petId}", method = RequestMethod.GET, params="myParam=myValue")

public void findPet(@PathVariable String ownerId, @PathVariable String petId, Model model) {

// implementation omitted

}

}

仅处理请求中包含了名为“myParam”，值为“myValue”的请求；

仅处理request的header中包含了指定“Refer”请求头和对应值为“http://www.ifeng.com/”的请求

headers的样例：

@Controller

@RequestMapping("/owners/{ownerId}")

public class RelativePathUriTemplateController {

@RequestMapping(value = "/pets", method = RequestMethod.GET, headers="Referer=http://www.ifeng.com/")

public void findPet(@PathVariable String ownerId, @PathVariable String petId, Model model) {

// implementation omitted

}

}

小结：当请求url不包含变量参数的时候 ，且请求方法包含参数的时候，不能限制为GET请求

2、@PathVariable

绑定路径中的占位符参数到方法参数变量中；只能绑定路径中的占位符参数，且路径中必须有参数。无论是 GET 或者POST 只要 URL中有参数即可！

实例如下：

GET

Request URL:http://localhost:8080/SpringMVC-1/springmvc/testPathVariable/1

POST

    <form action="springmvc/testPathVariable/1" method="POST">

        <input type="text" name="username" value=""/>

        <input type="text" name="age" value=""/>

        <input type="text" name="sex" value=""/>

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

    </form>

【注意】如果URL中无参数，将会出错；如果URL有参数，但是没有使用@PathVariabl该注解，那么URL的参数不会默认与方法参数绑定！方法里的参数会默认绑定表单里面对应的参数！

后台code

如果参数名与占位符一致，则可直接使用@PathVariable；如果不一致，则在@PathVariable( )括号内绑定占位符。

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

    public String testPathVariable(@PathVariable("id") Integer id2) {

        System.out.println("testPathVariable: " + id2);

        return SUCCESS;

    }

2.1、定义单个URL变量

在路由中定义变量规则后，通常我们需要在处理方法（也就是@RequestMapping注解的方法）中获取这个URL的具体值，并根据这个值（例如用户名）做相应的操作，SpringMVC提供的@PathVariable可以帮助我们：

@RequestMapping("/users/{username}")

@ResponseBody

public String userProfile(@PathVariable String username){

// return String.format("user %s", username);

return "user" + username;

}

在上述例子中，当@Controller处理HTTP请求时，userProfile方法的参数username会自动设置为URL中对应变量username（同名赋值）的值，例如当HTTP请求为/users/fpc，URL变量username的值fpc会被赋给函数参数username，函数的返回值自然是userfpc。

在默认的情况下，Spring会对@PathVariable注解的变量进行自动赋值，当然你也可以指定@PathVariable使用哪一个URL中的变量：

@RequestMapping("/users/{username}")

@ResponseBody

public String userProfile(@PathVariable("username") String username){

// return String.format("user %s", username);

return "user" + username;

}

当使用@RequestMapping URI template 样式映射时， 即 someUrl/{paramId}, 这时的paramId可通过 @Pathvariable注解绑定它传过来的值到方法的参数上。

示例代码：

@Controller

@RequestMapping("/owners/{ownerId}")

public class RelativePathUriTemplateController {

  @RequestMapping("/pets/{petId}")

  public void findPet(@PathVariable String ownerId, @PathVariable String petId, Model model) {

    // implementation omitted

  }

}

上面代码把URI template 中变量 ownerId的值和petId的值，绑定到方法的参数上。若方法参数名称和需要绑定的uri template中变量名称不一致，需要在@PathVariable("name")指定uri template中的名称。如果方法参数名称和绑定参数名称一致，默认自动对应赋值。

2.2、定义多个URL变量

可以定义URL路由，其中包含多个URL变量：

@RequestMapping("/user/{username}/blog/{blogId}")

@ResponseBody

public String getUerBlog(@PathVariable String username , @PathVariable int blogId) {

return "user: " + username + "blog->" + blogId;

}

这种情况下，Spring能够根据名字自动赋值对应的函数参数值，当然也可以在@PathVariable中显示地表明具体的URL变量值。

在默认情况下，@PathVariable注解的参数可以是一些基本的简单类型：int，long，Date，String等，Spring能根据URL变量的具体值以及函数参数的类型来进行转换，例如/user/fpc/blog/1，会将“fpc”的值赋给username，而1赋值给int变量blogId。

关于@PathVariable 有如下说明：

① 如果路径中的变量与方法中的变量名一致，可直接使用@PathVariable；

② 如果二者不一致，则使用@PathVariable(Variable)显示指定要绑定的路径中的变量 。

@PathVariable只能绑定路径中的占位符参数，且路径中必须有参数。

@PathVariable用法参考路径参数绑定参考

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

    public String testPathVariable(@PathVariable("id") Integer id2) {

        System.out.println("testPathVariable: " + id2);

        return SUCCESS;

    }

    //路径中的 id 与 方法中的 id2 绑定

//可以指定为含正则表达式的一类值( URI Template Patterns with Regular Expressions);

  @RequestMapping("/spring-web/{symbolicName:[a-z-]+}-{version:\d\.\d\.\d}.{extension:\.[a-z]}")

  public void handle(@PathVariable String version, @PathVariable String extension) {

    // ...

  }

}

@PathVariable与@RequestParam的区别

1)    相同点

A.       作用位置相同：都是直接修饰方法参数变量；

B.       功能相似：都是将URL中的变量值映射到方法参数上；

C.        都具有value属性：将URL变量名与方法参数名映射起来；

2)    不同点

A.       对应的URL不同

@PathVariable对应的URL是REST风格，具有占位符{XXX}，即URL模板；如/{id}/{name}

@RequestParam对应的URL是传统URL，key=value形式，如?id=1&name=zhaohong

B.       设置默认值

@RequestParam可以通过defaultValue属性设置默认值，而@PathVariable不可以。

C.        是否必需

@RequestParam可以通过required属性设置是否必需，默认为true；而@PathVariable一定是必需的

3、@RequestHeader、@CookieValue

@RequestHeader 注解，可以把Request请求header部分的值绑定到方法的参数上。

示例代码：

这是一个Request 的header部分：

Host                    localhost:8080

Accept                  text/html,application/xhtml+xml,application/xml;q=0.9

Accept-Language         fr,en-gb;q=0.7,en;q=0.3

Accept-Encoding         gzip,deflate

Accept-Charset          ISO-8859-1,utf-8;q=0.7,\*;q=0.7

Keep-Alive              300

@RequestMapping("/displayHeaderInfo.do")

public void displayHeaderInfo(@RequestHeader("Accept-Encoding") String encoding,

                              @RequestHeader("Keep-Alive") long keepAlive)  {

  //...

}

上面的代码，把request header部分的 Accept-Encoding的值，绑定到参数encoding上了， Keep-Alive header的值绑定到参数keepAlive上。

@CookieValue 可以把Request header中关于cookie的值绑定到方法的参数上。

例如有如下Cookie值：

JSESSIONID=415A4AC178C59DACE0B2C9CA727CDD84

参数绑定的代码：

@RequestMapping("/displayHeaderInfo.do")

public void displayHeaderInfo(@CookieValue("JSESSIONID") String cookie)  {

  //...

}

即把JSESSIONID的值绑定到参数cookie上。

4、@RequestParam

A） 常用来处理简单类型的绑定，通过Request.getParameter() 获取的String可直接转换为简单类型的情况（ String--> 简单类型的转换操作由ConversionService配置的转换器来完成）；因为使用request.getParameter()方式获取参数，所以可以处理get 方式中queryString的值，也可以处理post方式中 body data的值；

B）用来处理Content-Type: 为 application/x-www-form-urlencoded编码的内容，提交方式GET、POST；

C) 该注解有两个属性： value、required； value用来指定要传入值的id名称，required用来指示参数是否必须绑定；

示例代码：

@Controller

@RequestMapping("/pets")

@SessionAttributes("pet")

public class EditPetForm {

    // ...

    @RequestMapping(method = RequestMethod.GET)

    public String setupForm(@RequestParam("petId") int petId, ModelMap model) {

        Pet pet = this.clinic.loadPet(petId);

        model.addAttribute("pet", pet);

        return "petForm";

    }

    // ...

该注解相关属性如下：

value：参数key，可以不写，默认为""；

name：和value作用一样；

required：默认值为true，可以不写；

该注解常用来处理Content-Type: 不是application/x-www-form-urlencoded编码的内容，例如application/json, application/xml等；它是通过使用HandlerAdapter 配置的HttpMessageConverters来解析post data body，然后绑定到相应的bean上的。

因为配置有FormHttpMessageConverter，所以也可以用来处理 application/x-www-form-urlencoded的内容，处理完的结果放在一个MultiValueMap<String, String>里，这种情况在某些特殊需求下使用，详情查看FormHttpMessageConverter api;

示例代码：

@RequestMapping(value = "/something", method = RequestMethod.PUT)

public void handle(@RequestBody String body, Writer writer) throws IOException {

  writer.write(body);

}

5、@RequestBody

@RequestBody主要用来接收前端传递给后端的json字符串中的数据的(请求体中的数据的);GET方式无请求体，所以使用@RequestBody接收数据时，前端不能使用GET方式提交数据，而是用POST方式进行提交.

在后端的同一个接收方法里，@RequestBody 与@RequestParam()可以同时使用，@RequestBody最多只能有一个，而@RequestParam()可以有多个。

注:一个请求，只有一个RequestBody;一个请求，可以有多个RequestParam

注:当同时使用@RequestParam（）和@RequestBody时，@RequestParam（）指定的参数可以是普通元素、数组、集合、对象等等(即:当，@RequestBody 与@RequestParam()可以同时使用时，原SpringMVC接收参数的机制不变，只不过RequestBody 接收的是请求体里面的数据；而RequestParam接收的是key-value里面的参数，所以它会被切面进行处理从而可以用普通元素、数组、集合、对象等接收)。

即:如果参数时放在请求体中，传入后台的话，那么后台要用@RequestBody才能接收到;如果不是放在请求体中的话，那么后台接收前台传过来的参数时，要用@RequestParam（）来接收，或则形参前什么也不写也能接收。

注:如果参数前写了@RequestParam(xxx)，那么前端必须有对应的xxx名字才行(不管其是否有值)，如果没有xxx名的话，那么请求会出错，报400

注:如果参数前不写@RequestParam(xxx)的话，那么就前端可以有可以没有对应的xxx名字才行，如果有xxx名的话，那么就会自动匹配；没有的话，请求也能正确发送。

注:如果后端参数是一个对象，且该参数前是以@RequestBody修饰的，那么前端传递json参数时，必须满足以下要求:

a、后端@RequestBody注解对应的类在将HTTP的输入流(含请求体)装配到目标类(即:@RequestBody后面的类)时,

   会根据json字符  串中的key来匹配对应实体类的属性,如果匹配一致且json中的该key对应的值符合(或可转换为)

   实体类的对应属性的类型要求时,会调用实体类的setter方法将值赋给该属性

b、json字符串中，如果value为""的话，后端对应属性如果是String类型的，那么接受到的就是"",如果是后端属性的

  类型是Integer、Double等类型，那么接收到的就是null。

c、json字符串中，如果value为null的话，后端对应收到的就是null。

d、如果某个参数没有value的话，在传json字符串给后端时，要么干脆就不把该字段写到json字符串中;要么写value时，

    必须有值，null  或""都行。千万不能有类似"stature":,这样的写法，如:

6、@SessionAttributes, @ModelAttribute

6.1、@SessionAttributes:

该注解用来绑定HttpSession中的attribute对象的值，便于在方法中的参数里使用。

该注解有value、types两个属性，可以通过名字和类型指定要使用的attribute 对象；

示例代码：

@Controller

@RequestMapping("/editPet.do")

@SessionAttributes("pet")

public class EditPetForm {

    // ...

}

6.2、@ModelAttribute

该注解有两个用法，一个是用于方法上，一个是用于参数上；

用于方法上时：  通常用来在处理@RequestMapping之前，为请求绑定需要从后台查询的model；

用于参数上时： 用来通过名称对应，把相应名称的值绑定到注解的参数bean上；要绑定的值来源于：

A） @SessionAttributes 启用的attribute 对象上；

B） @ModelAttribute 用于方法上时指定的model对象；

C） 上述两种情况都没有时，new一个需要绑定的bean对象，然后把request中按名称对应的方式把值绑定到bean中。

用到方法上@ModelAttribute的示例代码：

// Add one attribute

// The return value of the method is added to the model under the name "account"

// You can customize the name via @ModelAttribute("myAccount")

@ModelAttribute

public Account addAccount(@RequestParam String number) {

    return accountManager.findAccount(number);

}

这种方式实际的效果就是在调用@RequestMapping的方法之前，为request对象的model里put（“account”， Account）；

用在参数上的@ModelAttribute示例代码：

@RequestMapping(value="/owners/{ownerId}/pets/{petId}/edit", method = RequestMethod.POST)

public String processSubmit(@ModelAttribute Pet pet) {

}

首先查询 @SessionAttributes有无绑定的Pet对象，若没有则查询@ModelAttribute方法层面上是否绑定了Pet对象，若没有则将URI template中的值按对应的名称绑定到Pet对象的各属性上。

params： 指定request中必须包含某些参数值是，才让该方法处理。

    @RequestMapping(value = "testParamsAndHeaders", params = { "username","age!=10" })

    public String testParamsAndHeaders() {

        System.out.println("testParamsAndHeaders");

        return SUCCESS;

    }

params 只是判断url 或者 form data 中的参数是否复合params的定义，并不会直接绑定数据到方法的参数中！

7、@RestController

只需要类添加  @RestController  即可，默认类中的方法都会以json的格式返回

8、其他RabbitMQ

消息队列，主要是用来实现应用程序的异步和解耦，同时也能起到消息缓冲，消息分发的作用

消息中间件最主要的作用是解耦，中间件最标准的用法是生产者生产消息传送到队列，消费者从队列中拿取消息并处理，生产者不用关心是谁来消费，消费者不用关心谁在生产消息，从而达到解耦的目的.

RabbitMQ主要是为了实现系统之间的双向解耦而实现的。当生产者大量产生数据时，消费者无法快速消费，那么需要一个中间层。保存这个数据。

https://blog.csdn.net/justry\_deng/article/details/80972817

https://blog.csdn.net/j080624/article/details/56280382

————————————————

版权声明：本文为CSDN博主「月半花开」的原创文章，遵循 CC 4.0 BY-SA 版权协议，转载请附上原文出处链接及本声明。

原文链接：https://blog.csdn.net/qq\_20957669/article/details/87686899