

Spring 注解详解

Spring常用注解详解-@Controller

□ @Controller

- 在SpringMVC 中，控制器Controller 负责处理由 DispatcherServlet 分发的请求，它把用户请求的数据经过业务处理层处理之后封装成一个Model，然后再把该Model 返回给对应的View 进行展示。

Spring常用注解详解-@Controller

- 在SpringMVC 中提供了一个非常简便的定义Controller 的方法，你无需继承特定的类或实现特定的接口，只需使用@Controller 标记一个类是Controller，然后使用@RequestMapping 和@RequestParam 等一些注解用以定义URL 请求和Controller 方法之间的映射，这样的Controller 就能被外界访问到。

Spring常用注解详解-@Controller

- 此外Controller 不会直接依赖于
HttpServletRequest 和HttpServletResponse 等
HttpServletRequest 对象，它们可以通过Controller 的方法参数灵活的获取到。

Spring常用注解详解-@Controller

□@Controller 用于标记在一个类上，使用它标记的类就是一个SpringMVC Controller 对象。分发处理器将会扫描使用了该注解的类的方法，并检测该方法是否使用了@RequestMapping 注解。@Controller 只是定义了一个控制器类，而使用@RequestMapping 注解的方法才是真正处理请求的处理器。

Spring常用注解详解-@Controller

□ 单单使用@Controller 标记在一个类上还不能真正意义上的说它就是SpringMVC 的一个控制器类，因为这个时候Spring 还不认识它。那么要如何做Spring 才能认识它呢？这个时候就需要我们把这个控制器类交给Spring 来管理。

Spring常用注解详解-@Controller

□有两种方式:

- (1) 在SpringMVC 的配置文件中定义 **HelloWorldController** 的bean 对象。
- (2) 在SpringMVC 的配置文件中告诉Spring 该到哪里去找标记为@Controller 的Controller 控制器。
(方便, 常用)

<!--方式一:使用类的全名进行配置-->

```
<bean class="com.springmvc.control.HelloWorldController"/>
```

<!--方式二:配置springMV自动扫描, 路径写到controller的上一层-->

```
<context:component-scan base-package = "com.springmvc.control" />
```

@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

□ 下文的访问路径为

localhost:8080/springMVC/test1/helloWorld

□ 在本Controller中定义的@RequestMapping方法, 都必须有 “/test1” 这个父路径

```
@Controller
@RequestMapping(value="/test1")
public class HelloWorldController{

    @RequestMapping(value="/helloWorld")
    public String HelloWorldGet(){
        System.out.println("get方法成功进入");
        return "success";
    }
}
```

@RequestMapping

- RequestMapping注解有六个属性，下面我们把她分成三类进行说明（下面有相应示例）。

@RequestMapping

□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

□url都为“helloWorld”,springMVC会根据前端提交方法的不同,分发至不同的方法中,类似于servlet的doGet方法和doPost方法的作用

```
@Controller
public class HelloWorldController{

    //当发起 项目名/helloWorld的访问时,进入此方法处理
    @RequestMapping(value="helloWorld",method=RequestMethod.POST)
    public String HelloWorldPost(){
        System.out.println("post方法成功进入");
        return "success";
    }
    @RequestMapping(value="helloWorld",method=RequestMethod.GET)
    public String HelloWorldGet(){
        System.out.println("get方法成功进入");
        return "success";
    }
}
```

@RequestMapping

```
<body>
  <form action="helloWorld" method="post">
  <!--
    <label for="username">请输入姓名</label>
    <input id="username" name="username">-->
    <input type="submit" value="登陆">
  </form>
</body>
```

改成get试试

注意：这里将所有的参数提交全部注释掉，后面会详细的讲

@RequestMapping

□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

- ❑ **@Controller**
- ❑ **@RequestMapping(value = "/pets", method = RequestMethod.POST, consumes="application/json")**
- ❑ **public void addPet(@RequestBody Pet pet, Model model) { // implementation omitted }**

produces

```
❑ @Controller @RequestMapping(value =  
    "/pets/{petId}", method = RequestMethod.GET,  
    produces="application/json") @ResponseBody  
    public Pet getPet(@PathVariable String petId,  
        Model model) {  
  
    // implementation omitted  
  
}
```


@RequestMapping

□3、 params, headers

- params: 指定request中必须包含某些参数值时, 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

//当发起 项目名/helloWorld的访问时, 进入此方法处理

```
@RequestMapping(value="helloWorld", params="myParam=myValuehaha", method=RequestMethod.POST)
```

```
public String HelloWorldPost() {
```

```
    System.out.println("post方法成功进入, myParam值为myValuehaha");
```

```
    return "success";
```

```
}
```

```
@RequestMapping(value="helloWorld", params="myParam=myValue", method=RequestMethod.POST)
```

```
public String HelloWorldPostHaveMsg() {
```

```
    System.out.println("post方法成功进入, myParam值为myParam");
```

```
    return "success";
```

```
}
```

params

□Jsp内容

```
<body>  
  <form action="helloWorld?myParam=myValuehaha" method="post">  
    <input type="submit" value="登陆">  
  </form>  
  
</body>
```

headers

- ❑ 1. 添加log.jsp, 内容与login.jsp完全相同
- ❑ 2. 修改HelloWorldController.java

```
@RequestMapping(value="helloWorld",  
    headers="referer=http://localhost:8080/springMVC/login.jsp",  
    method=RequestMethod.POST)  
public String HelloWorldHeader(){  
    System.out.println("post方法成功进入, 含有role");  
    return "success";  
}  
  
@RequestMapping(value="helloWorld", method=RequestMethod.POST)  
public String HelloWorld(HttpServletRequest request){  
    System.out.println("post方法成功进入");  
  
    return "success";  
}
```

由此路径发起的请求将由本方法执行

@RequestParam

- ❑ @requestParam主要用于在SpringMVC后台控制层获取参数，类似request.getParameter("name")
- ❑ 它有三个常用参数： defaultValue, required, value ;
 - **value**: 参数名字，即入参的请求参数名字，如username表示请求的参数区中的名字为username的参数的值将传入；
 - **required**: 是否必须，默认是true，表示请求中一定要有相应的参数，否则将抛出异常；
 - **defaultValue**: 默认值，表示如果请求中没有同名参数或同名参数值为空时的默认值，设置该参数时，自动将required设为false。

value

```
<form action="test1/helloWorld" method="post">
  <label id="namelabel" for="username">请输入姓名</label>
  <input name="username" id="username">
  <input type="submit" value="登陆">
</form>
```

```
@RequestMapping(value="test1")
public class HelloWorldController{

    @RequestMapping(value="helloWorld",
        method=RequestMethod.POST)
    public String HelloWorldHeader(
        @RequestParam(value="username") String username ){
        //注意，此时必须传入名为username的参数，否则浏览器会报400错误
        System.out.println("姓名为: "+username);
        return "success";
    }
}
```

required

□修改HelloWorldHeader方法，添加属性 **required=false**

```
<body>
  <form action="test1/helloWorld" method="post">
    <!-- <label id="namelabel" for="username">请输入姓名</label>
    <input name="username" id="username">-->
    <input type="submit" value="登陆">
  </form>
```

```
@RequestMapping(value="helloWorld",
    method=RequestMethod.POST)
public String HelloWorldHeader(
    @RequestParam(value="username", required=false) String username) {
    //注意，此时必须传入名为username的参数，否则浏览器会报400错误
    System.out.println("姓名为: "+username);
    return "success";
}
```

defaultValue

```
110 class HelloWorldController {
```

```
    @RequestMapping(value="helloWorld",  
        method=RequestMethod.POST)
```

```
    public String HelloWorldHeader(  
        @RequestParam(value="username", required=false, defaultValue="默认") String user
```

```
        //注意，此时必须传入名为username的参数，否则浏览器会报400错误
```

```
        System.out.println("姓名为: "+username);
```

```
        return "success";
```

```
    }
```

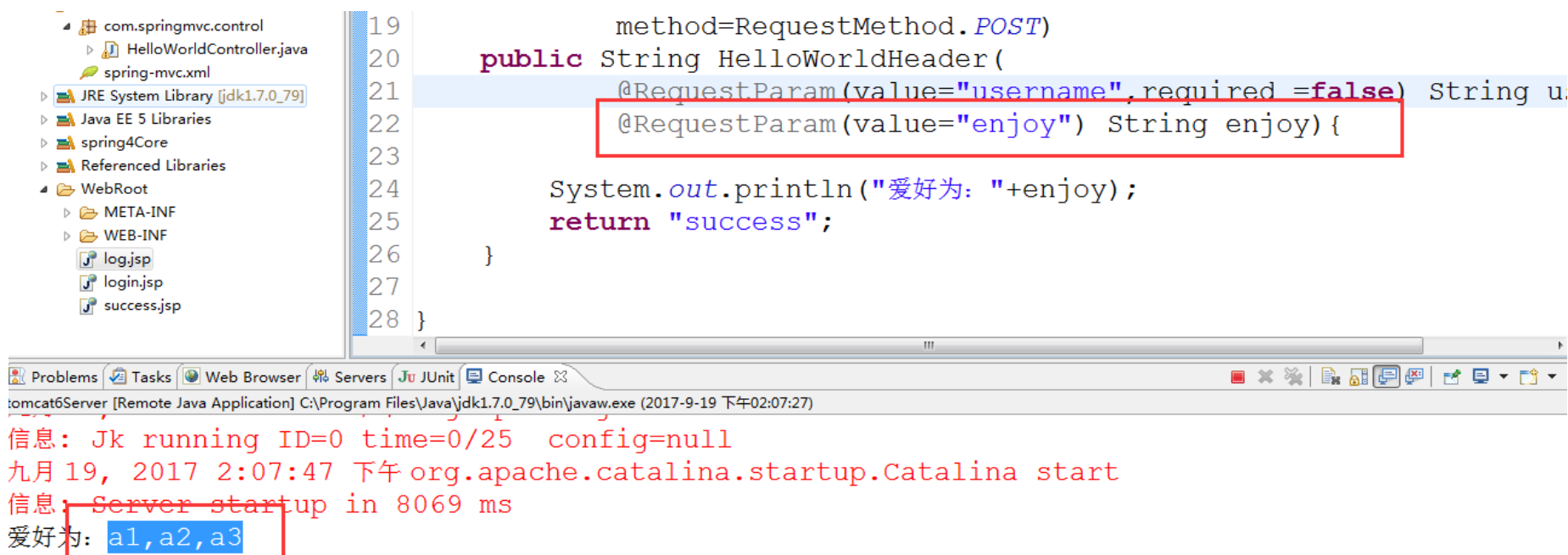

同名参数的处理

- 如果请求中有多个同名的应该如何接收呢？
如添加爱好，可能有多个爱好

```
<form action="test1/helloWorld" method="post">
  <label id="namelabel" for="username">请输入姓名</label>
  <input name="username" id="username">
  <label id="enjoylabel" for="enjoy">请输入爱好</label>
  <input name="enjoy" id="enjoy">
  <input name="enjoy" id="enjoy">
  <input name="enjoy" id="enjoy">
  <input type="submit" value="登陆">
```

同名参数的处理

□如果请求参数类似于url?enjoy=a1&enjoy=a2，则实际roleList参数入参的数据为“a1,a2”，即多个数据之间使用“，”分割；



The screenshot shows an IDE with a project named 'com.springmvc.control'. The 'HelloWorldController.java' file is open, showing the following code:

```
19         method=RequestMethod.POST)
20     public String HelloWorldHeader(
21         @RequestParam(value="username", required=false) String u
22         @RequestParam(value="enjoy") String enjoy){
23
24         System.out.println("爱好为: "+enjoy);
25         return "success";
26     }
27
28 }
```

The console output shows the following messages:

```
tomcat6Server [Remote Java Application] C:\Program Files\Java\jdk1.7.0_79\bin\javaw.exe (2017-9-19 下午02:07:27)
信息: Jk running ID=0 time=0/25 config=null
九月 19, 2017 2:07:47 下午 org.apache.catalina.startup.Catalina start
信息: Server startup in 8069 ms
爱好为: a1,a2,a3
```

同名参数的处理

□更好的处理方式：使用数组或集合进行接收

```
@RequestMapping(value="helloWorld",
    method=RequestMethod.POST)
public String HelloWorldHeader(
    @RequestParam(value="username",required =false) String
    @RequestParam(value="enjoy") String[] enjoy){
    for(String s:enjoy){
        System.out.println("爱好为: "+s);}
    return "success";
}
```

```
@RequestMapping(value="helloWorld",
    method=RequestMethod.POST)
public String HelloWorldHeader(
    @RequestParam(value="username",required =false) String
    @RequestParam(value="enjoy") List<String> enjoy){
    for(String s:enjoy){
        System.out.println("爱好为: "+s);}
    return "success";
}
```

@PathVariable

□ @PathVariable 用于将请求URL中的模板变量 (**URI Template Patterns**) 映射到功能处理方法的参数上。

➤ URI 模板可以提供给 @RequestMapping 注解访问特定的url 一个很方便的方式。URI模板是一个类似于URI的String，包含一个或者多个参数名字。

➤ 例如，URI模板是以下的情况：

http://www.example.com/users/{userId} 包含了变量 userId， 如果通过这样的一个URI去访问

http://www.example.com/users/fred， 那么userId的值就是fred。

@PathVariable

```
<form action="test1/{testParam}/helloWorld/{param1}" method="post">
  <!--
  <label id="namelabel" for="username">请输入姓名</label>
  <input name="username" id="username">
  -->
  <input type="submit" value="登陆">
</form>
```

模板参数1

模板参数2

```
@Controller
@RequestMapping(value="test1/{testParam}")
public class HelloWorldController{

    @RequestMapping(value="helloWorld/{param1}",
        method=RequestMethod.POST)
    public String HelloWorldHeader(@PathVariable String testParam,
        @PathVariable String param1){
        System.out.println(testParam);
        System.out.println(param1);
        return "success";
    }
}
```

@PathVariable

□上例中：请求的URL为“控制器

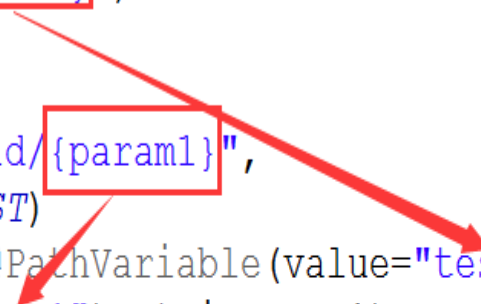
URL/test1/**testParam**/helloWorld/**param1**”，则
自动将URL中模板变量{testParam}和{param1}绑定到
通过@PathVariable注解的同名参数上，即入参后
testParam=testParam、 **param1=param1**。

@PathVariable

□如模板变量{testParam}和{param1}和欲绑定的参数名不同，则@PathVariable必须指定value属性的值

```
@Controller
@RequestMapping(value="test1/{testParam}")
public class HelloWorldController{

    @RequestMapping(value="helloWorld/{param1}",
        method=RequestMethod.POST)
    public String HelloWorldHeader(@PathVariable(value="testParam") String arg0,
        @PathVariable(value="param1") String arg1){
        System.out.println(arg0);
        System.out.println(arg1);
        return "success";
    }
}
```



@PathVariable

- 当@PathVariable用在Map<String, String>参数上时, 那么map中的值就会被URI 模板中的参数值填补, key 为参数名, value为参数值。

```
@Controller
@RequestMapping(value="test1/{testParam}")
public class HelloWorldController{

    @RequestMapping(value="helloWorld/{param1}",
                    method=RequestMethod.POST)
    public String HelloWorldHeader(@PathVariable Map<String,String> map){
        for(String s:map.keySet()){
            System.out.println(s+"---->"+map.get(s));
        }
        return "success";
    }
}
```


@PathVariable

- @PathVariable注解的参数可以是简单的数据类型，例如int, String, Date等，String自动的进行转换或者如果转换失败的话，那么就会抛出
TypeMismatchException

@SessionAttributes

- 在默认情况下，ModelMap 的作用域是 request 级别，也就是说，当本次请求结束后，ModelMap 中的属性将销毁。如果希望在多个请求中共享 ModelMap 中的属性，必须将其属性转存到 session 中，这样 ModelMap 的属性才可以被跨请求访问。
- Spring 允许我们有选择地指定 ModelMap 中的哪些属性需要转存到 session 中，以便下一个请求属对应的 ModelMap 的属性列表中还能访问到这些属性。这一功能是通过类定义处标注 @SessionAttributes 注解来实现的。

@SessionAttributes

□ 单个参数

```
@Controller
@RequestMapping(value="/test1")
@SessionAttributes("username")
public class HelloWorldController{

    @RequestMapping(value="/helloWorld",
                    method=RequestMethod.POST)
    public String HelloWorldHeader(@RequestParam String username,ModelMap modelMap){
        System.out.println(username);
        modelMap.put("username", username);
        return "redirect:/success.jsp";
    }
}
```

@SessionAttributes

□多个参数：以String数组的形式定义

```
@Controller
@RequestMapping(value="/test1")
@SessionAttributes({"username", "arg0"})
public class HelloWorldController{

    @RequestMapping(value="/helloWorld",
        method=RequestMethod.POST)
    public String HelloWorldHeader(@RequestParam String username, ModelMap modelMap) {
        System.out.println(username);
        modelMap.put("username", username);
        modelMap.put("arg0", "test");
        return "redirect:/success.jsp";
    }
}
```

@SessionAttributes

□ @SessionAttributes清除:

- @SessionAttributes需要清除时, 使用 `SessionStatus.setComplete();`来清除。
- 注意, 它只清除@SessionAttributes的session, 不会清除HttpSession的数据。故如用户身份验证对象的session一般不同它来实现, 还是用session.setAttribute等传统的方式实现。

@SessionAttributes清除

```
@Controller
@RequestMapping(value="/test2")
@SessionAttributes("username")
public class HelloWorldController{

    @RequestMapping("/testClearSession")
    public String testClearSession(SessionStatus sessionStatus,ModelMap map) {
        sessionStatus.setComplete();
        return "success";
    }
}
```

@ModelAttribute

□ **ModelAttribute**可以应用在方法参数上或方法上。

- 运用在参数上，会将客户端传递过来的参数按名称注入到指定对象中，并且会将这个对象自动加入ModelMap中，便于View层使用；
- 运用在方法上，会在每一个@RequestMapping标注的方法前执行，如果有返回值，则自动将该返回值加入到ModelMap中并将该方法变成一个非请求处理的方法。

注释void返回值的方法

□例一：

```
@Controller
@RequestMapping(value="/test2")
public class HelloWorldController{

    @ModelAttribute
    public void before(@RequestParam String username, ModelMap map){
        map.addAttribute("username",username);
    }

    @RequestMapping(value="/testModelAttribute")
    public String testModelAttribute(){
        return "success";
    }
}
```


注释void返回值的方法

□在获得请求/**testModelAttribute**后，**before**方法在**testModelAttribute**方法之前先被调用，它把请求参数（username）加入到一个名为map的model属性中，在它执行后**testModelAttribute**被调用。

本例中**Before**方法会在本**Controller**类中所有的@RequestMapping注解的方法前执行，可用于接受请求参数，减少处理方法的入参。

注释返回具体类型的方法

□例二：前端提交的username不受影响，此方法在@RequestMapping注解的方法前执行，相当于在ModelMap中放入了一个key为“newStr”的属性，返回值即为value,可以在前端使用EL获取。

```
@ModelAttribute("newStr")
public String before(@RequestParam String username, ModelMap map){
    map.addAttribute("username",username);
    return "test";
}
```

注释返回具体类型的方法

- 例三：当model属性的名称没有指定，它由返回类型隐含表示，如这个方法返回Student类型，那么这个model属性的名称是student（首字符小写）。

```
@ModelAttribute()  
public Student before(@RequestParam String username, ModelMap modelMap) {  
    modelMap.addAttribute("username", username);  
    Student s = new Student();  
    s.setName("aa");  
    s.setAge(11);  
    return s;  
}  
  
<head></head>  
<body>  
跳转成功，欢迎${username}  
----- ${student} -----  
</body>  
</html>
```

@ModelAttribute和 @RequestMapping同时注释一个 方法

□例四：

```
@Controller
//@RequestMapping(value="/test2")//已被注释
public class HelloWorldController{
    @RequestMapping(value="/success")
    @ModelAttribute("attributeName")
    public String testModelAttribute(@RequestParam String username){
        return "go --->success.jsp";
    }
}

<html>
    <head></head>
    <body>
        跳转成功, ${attributeName }
    </body>
</html>
```

@ModelAttribute和 @RequestMapping同时注释一个 方法

- 这时这个方法的返回值并不是表示一个视图名称，而是model属性的值，视图名由RequestToViewNameTranslator根据请求“/success”转换为success。
- Model属性名称由@ModelAttribute(value= “key”)指定，相当于在request中封装了key=attributeName, value= “go --->success.jsp”。
- 简单的说：当发起<http://localhost:8080/springMVC/success>请求时，最终会请求转发至success.jsp，并在Model中添加一个key为attributeName, value为 “go --->success.jsp”的属性

注释方法的参数（常用）

□ @ModelAttribute 注释方法的一个参数表示应从模型 model 中取得。

- 若在 model 中未找到，那么这个参数将先被实例化后加入到 model 中。
- 若在 model 中找到，则请求参数名称和 model 属性字段若相匹配就会自动填充。
- 这个机制对于表单提交数据绑定到对象属性上很有效。

注释方法的参数（常用）

□例五：下例中，前端提交一个username参数，@ModelAttribute根据参数名绑定到入参username上，并自动将其放入Model中，key为username，value为提交的值

```
@Controller
@RequestMapping(value="/test2")
public class HelloWorldController{
    @RequestMapping(value="/testModelAttribute")
    public String testModelAttribute(@ModelAttribute("username") String username){
        return "success";
    }
}
```

注释方法的参数（常用）

□例六：下例中，前端提交name和age，
@ModelAttribute根据参数名，绑定到入参student
（先被实例化）上，根据属性名自动将值注入，并将其
放入Model中。

□Student类

```
public class Student {  
    private String name;  
    private Integer age;  
  
    //省略getter和setter方法
```


注释方法的参数（常用）

□Controller类

```
@Controller
@RequestMapping(value="/test2")
public class HelloWorldController{
    @RequestMapping(value="/testModelAttribute")
    public String testModelAttribute(@ModelAttribute Student student){
        System.out.println(student);
        return "success";
    }
}
```

□Login.jsp

```
<form action="test2/testModelAttribute" method="post">
    <label id="namelabel" for="name">请输入姓名</label>
    <input name="name" id="name">
    <label id="psdlabel" for="age">请输入年龄</label>
    <input name="age" id="age">
    <input type="submit" value="登陆">
</form>
```

注释方法的参数（常用）

- @ModelAttribute标注在参数上：同时具有 取/存的功能.
- @ModelAttribute("name") String name:
 - 1.首先把 “?” 后如果有名为name的参数就把此参数绑定要同名的方法形式参数中.
 - 2.同时将 “name” 的值存在Model中, key=@ModelAttributeValue,value= 参数的值同时不影响同类中的其他方法

@CookieValue

□ @CookieValue 可让处理方法入参绑定某个 Cookie 值, 有三个属性, 分别如下:

- (1) value 请求参数的参数名;
- (2) required 该参数是否必填, 默认为true(必填), 当设置成必填时, 如果没有传入参数, 报错;
- (3) defaultValue 设置请求参数的默认值;

@CookieValue

□Jsp

```
<a href="test2/testCookieValue">cookie测试</a>
```

□Controller类

```
@RequestMapping("/testCookieValue")  
public String testCookieValue(@CookieValue("JSESSIONID") String sessionId ) {  
    System.out.println("testCookieValue,sessionId="+sessionId);  
    return "success";  
}
```

@RequestBody

□ **@RequestBody** 可让处理方法入参绑定某个 web 请求的头信息，有三个属性，分别如下：

- (1) value 请求参数的参数名；
- (2) required 该参数是否必填，默认为true(必填)，当设置成必填时，如果没有传入参数，报错；
- (3) defaultValue 设置请求参数的默认值；

@RequestHeader

□这时这个参数info将获得请求的Accept头信息

```
@RequestMapping("/testHeader")  
public String testHeader(@RequestHeader("Accept") String info, ModelMap map) {  
    System.out.println(info);  
    return "success";  
}
```

@RequestBody

作用:

- 1) 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;
- 2) 再把HttpMessageConverter返回的对象数据绑定到controller中方法的参数上。
- 简单的说: 能够实现json格式的字符串与pojo类的自动绑定; **Content-Type值要为application/json, 且须配合jackson.jar使用。**

@RequestBody

□使用时机：

□A) GET、POST方式提时， 根据request header Content-Type的值来判断：

- application/x-www-form-urlencoded, 可选（即非必须，因为这种情况的数据@RequestParam, @ModelAttribute也可以处理，当然@RequestBody也能处理）；
- multipart/form-data, 不能处理（即使用@RequestBody不能处理这种格式的数据）；
- 其他格式， 必须（其他格式包括application/json, application/xml等。这些格式的数据， 必须使用@RequestBody来处理）；

@RequestBody

□B) PUT方式提交时， 根据request header Content-Type的值来判断：

- application/x-www-form-urlencoded, 必须；
- multipart/form-data, 不能处理；
- 其他格式, 必须；

@ResponseBody

□作用：

- 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。

@ResponseBody

□使用时机：

- 返回的数据不是html标签的页面，而是其他某种格式的数据时（如json、xml等）使用；
- 通常用于AJAX应用

@RequestBody、@ResponseBody实例

□Jsp

```
<script src="jquery-1.8.3.min.js"></script>
<script type="text/javascript">
    function ajaxgo() {
        $.ajax({
            type: "post",
            contentType: "application/json",
            url: "test2/testAjax",
            data: '{"age":19,"name":"zhang"}',
            success: function(data) {
                alert(data);
            }
        })
    }
}
```

@RequestBody、@ResponseBody实例

□Controller:

```
@Controller
@RequestMapping(value="/test2")
public class HelloWorldController{

    @RequestMapping(value="/testAjax")
    @ResponseBody
    public String testModelAttribute(@RequestBody Student s){

        System.out.println(s);
        return "AJAX success";
    }

}
```

@RequestBody、 @ResponseBody实例

- 上例通过ajax将json字符串传递至Controller，通过RequestBody自动绑定为Student对象

谢谢！