02 SpringMVC-RequestMapping注解讲解

`SpringMVC`

一：@RequestMapping描述

> 1、Spring MVC 使用 @RequestMapping 注解为控制器指定可以处理哪些 URL 请求

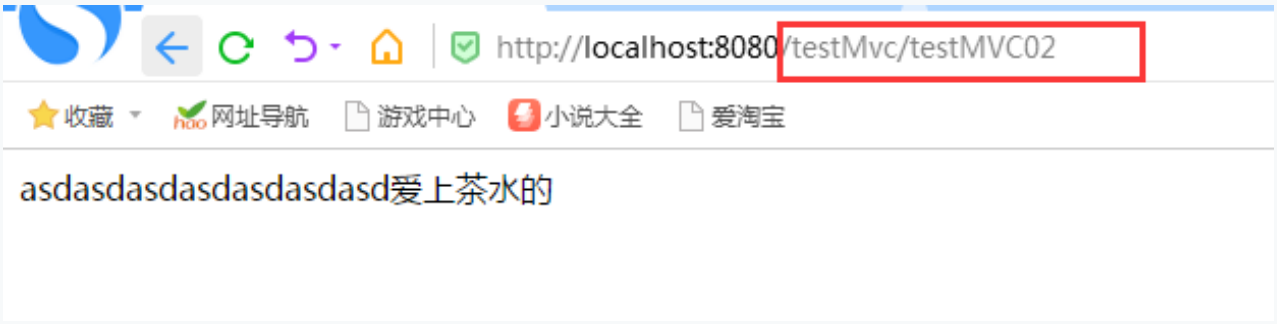
二：修饰类

> 1、在控制器的类定义及方法定义处都可进行使用
> -类定义处：
> 提供初步的请求映射信息
> 相对于 WEB 应用的根目录
> – 方法处：
> 提供进一步的细分映射信息
> 相对于类定义处的 URL
> 若类定义处未标注 @RequestMapping，则方法处标记的 URL 相对于 WEB 应用的根目录

```
@Controller
@RequestMapping("/testMvc")
public class TestMvc02 {
    @RequestMapping("/testMVC02")
    public String testMVC02(){
        System.out.println("==========");
       return "success";
    }
}
```

> 2、请求地址

http://localhost:8080/testMvc/testMVC02



三：@RequestMapping参数

1、请求 URL & 请求方法

1、value:指定的路径地址

2、method:指定请求的方式(get还是post)

```
@RequestMapping(value = "/springmvc01",method = RequestMethod.POST)
    public String Test02(){
        System.out.println("sssssssssssss");
        return "success";
    }
```

3、页面请求编写

```
  <form  action="springmvc01" method="post">
    <input type="submit" value="提交">
  </form>
```

2、请求参数 & 请求头

1、请求参数及请求头的映射条件，他们之间是与的关系，联合使用多个条件可让请求映射更加精确化

2、params 和 headers支持简单的表达式(如下)

– param1:
表示请求必须包含名为 param1 的请求参数
– !param1:
表示请求不能包含名为 param1 的请求参数
– param1 != value1:
表示请求包含名为 param1 的请求参数，但其值
不能为 value1
– { “param1=value1”, “param2” }:
请求必须包含名为 param1 和param2
的两个请求参数，且 param1 参数的值必须为 value1

```
@RequestMapping(value = "/test03",params = {"name","age!=10"},
headers = {"Accept-Language=en-US,zh;q=0.9"})
public String Test03(){
    System.out.println("Test03");
    return "success";
}
```

3、页面请求

```
<!--这里是满足params的要求-->
<a href="test03?name=小张&age=11" >test03</a>
```

4、请求头的信息如下，不满足,请求的时候会进行报404找不页面

四：@RequestMapping支持Ant路径(匹配符)

1、Ant 风格资源地址支持 3 种匹配符：
– ?：
匹配文件名中的一个字符
– *：
匹配文件名中的任意字符
– ：
匹配多层路径

```
/*
     *地址：都可访问。只要满足这条件
     * http://localhost:8080/test04/q/abcd/a/s
     *http://localhost:8080/test04/qq/abcs/qqq/aaa
     * */
     @RequestMapping(value = "/test04/*/abc?/**")
     public String Test04(){
         System.out.println("Test04");
         return "success";
     }
```

五：@PathVariable 绑定的占位符进行映射

1、带占位符的 URL 是 Spring3.0 新增的功能，该功能在 SpringMVC 向 REST 目标挺进发展开启新里程碑

2、通过 @PathVariable 可以将 URL 中占位符参数绑定到控制器处理方法的入参中：URL 中的 {xxx} 占位符可以通过 @PathVariable("xxx") 绑定到操作方法的入参中

```
     @RequestMapping(value = "/Test05/{name}/{age}")
     public String Test05(@PathVariable("name") String name,@PathVariable("age") Integer age){
         System.out.println("Test05:"+name+":"+age);
         return "success";
     }

======页面请求======
    <a href="Test05/小张/11" >Test05</a>
```

请求

```
09-Apr-2019 12:05:02.407 INFO [localhost-startStop-1] org.apache.ca
09-Apr-2019 12:05:02.460 INFO [localhost-startStop-1] org.apache.ca
Test05:小张: 11
```

六：REST

1、REST：（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。结构清晰、符合标准、易于理解、扩展方便，越来越多网站进行采用

2、HTTP 协议里面，四个表示操作方式的动词：GET、POST、PUT、DELETE。
-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

3、POST和GET都用使用，那么PUT和DELETE怎么使用呢？

4、这里就需要 HiddenHttpMethodFilter类了将这些请求转换
为标准的 http 方法，使得支持 GET、POST、PUT 与DELETE 请求

1、查询HiddenHttpMethodFilter源码

1、Ctrl+N查询(IDEA)源码,用有个doFilterInternal方法,进行转换

2、注意Tomcat8以上(包含8),是无法进行跳转页面的,只能进行访问请求。

3、建议使用Tomcat7完成

```
1.   private String methodParam = "_method";
2.
3.  protected void doFilterInternal(HttpServletRequest request,
    HttpServletResponse response, FilterChain filterChain) throws
    ServletException, IOException {
4.          String paramValue = request.getParameter(this.methodParam);
5.         //获取对应的请求类型
```

```
            if ("POST".equals(request.getMethod()) && StringUtils.hasLength(paramValue)) {
                String method = paramValue.toUpperCase(Locale.ENGLISH);
                HttpServletRequest wrapper = new HiddenHttpMethodFilter.HttpMethodRequestWrapper(request, method);
                filterChain.doFilter(wrapper, response);
            } else {
                filterChain.doFilter(request, response);
            }
        }
```

2、xml文件配置

```
    <!--配置所有的地址进行过滤对应的请求-->
    <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>
```

3、controller编写

```
    @RequestMapping(value = "/TestGet/{name}",method = RequestMethod.GET)
    public String TestGet(@PathVariable("name") String name){
        System.out.println("TestGet:"+name);
        return "success";
    }

    @RequestMapping(value = "/TestPost",method = RequestMethod.POST)
    public String TestPost(){
        System.out.println("TestPost:");
        return "success";
    }

    @RequestMapping(value = "/TestDelete/{name}",method = RequestMethod
```

```
.DELETE)
    public String TestDelete(@PathVariable("name") String name){
        System.out.println("TestDelete:"+name);
        return "success";
    }


    @RequestMapping(value = "/TestPut/{name}",method = RequestMethod.PUT)
    public String TestPut(@PathVariable("name") String name){
        System.out.println("TestPut:"+name);
        return "/success";
    }
```

4、页面请求

```
  <%--Delete的请求--%>
  <form  ACTION="TestPut/小张" METHOD="post">
    <input type="hidden"  name="_method" value="PUT">
    <input TYPE="submit" VALUE="TestPut">
  </form>

  <%--Delete的请求--%>
  <form  ACTION="TestDelete/小张" METHOD="post">
    <input type="hidden" value="DELETE" name="_method">
    <input TYPE="submit" VALUE="Delete">
  </form>
<%--post请求--%>
  <form  ACTION="TestPost" METHOD="post">
    <input TYPE="submit" VALUE="PSOT">
  </form>
  <%--get请求--%>
  <a href="TestGet/小张" >TestGet</a>
```