1. RequestMapping 注解

1.1 使用说明

源码：

```
@Target({ElementType.METHOD, ElementType.TYPE})
 @Retention(RetentionPolicy.RUNTIME)
 @Documented
 @Mapping
 public @interface RequestMapping {

}
```

作用： 用于建立请求 URL 和处理请求方法之间的对应关系。出现位置：

1. 类上：请求 URL 的第一级访问目录。此处不写的话，就相当于应用的根目录。写的话
   需要以/开头。 它出现的目的是为了使我们的 URL 可以按照模块化管理: 例如：
   账户模块： /account/add /account/update /account/delete ... 订单模块：
   /order/add /order/update /order/delete 红色的部分就是把 RequsetMapping 写在
   类上，使我们的 URL 更加精细。
2. 方法上：请求URL的二级访问目录

属性：

1. value：用于指定请求的 URL。它和 path 属性的作用是一样的。
2. method：用于 指定请求的方式。
3. params：用于指定限制请求参数的条件。它支持简单的表达 式。要求请求参数的 key 和 value 必须和 配置的一模一样。 例如： params = {"accountName"}，表示请求参数必须有 accountName；params = {"moeny!100"}，表示请求参数中 money 不能是 100。
4. headers：用于指定限定请求消息头的条件。

注意： 以上四个属性只要出现 2 个或以上时，他们的关系是与的关系。

属性名	类型	描述
name	String	可选属性，用于为映射地址指定别名。
value	String[]	可选属性，同时也是默认属性，用于映射一个请求和一种方法，可以标注在一个方法或一个类上。
method	RequestMethod[]	可选属性，用于指定该方法用于处理哪种类型的请求方式，其请求方式包括 GET、POST、HEAD、OPTIONS、PUT、PATCH、DELETE 和 TRACE 例如 method=RequestMethod.GET 表示只支持 GET 请求，如果需要支持多个请求方式则需要通过{}写成数组的形式，并且多个请求方式之间是有英文逗号分隔。
params	String[]	可选属性，用于指定 Request 中必须包含某些参数的值，才可以通过其标注的方法处理。
headers	String[]	可选属性，用于指定 Request 中必须包含某些指定的 header 的值，才可以通过其标注的方法处理。
consumes	String[]	可选属性，用于指定处理请求的提交内容类型（Content-type），比如 application/json，text/html 等。
produces	String[]	可选属性，用于指定返回的内容类型，返回的内容类型必须是 request 请求头（Accept）中所包含的类型。

1.2 使用示例

1.2.1 出现位置的示例：

控制器代码：

```
1  /**
2  * RequestMapping 注解出现的位置
3  * @Version 1.0
4  */
5  @Controller("accountController")
6  @RequestMapping("/account")
7  public class AccountController {
8      @RequestMapping("/findAccount")
9      public String findAccount() {
10 System.out.println("查询了账户。。。。");
11          return "success";
12      }
13  }
```

jsp 中的代码：

```
1  <%@ page language="java" contentType="text/html;
charset=UTF-8"
2  pageEncoding="UTF-8"%>
3  <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01
Transitional//EN"
4  "http://www.w3.org/TR/html4/loose.dtd">
5  <html>
6  <head>
```

```
7       <meta http-equiv="Content-Type"
content="text/html;charset=UTF-8"> <title>requestmapping
的使用</title>
8   </head>
9   <body>
10      <!-- 第一种访问方式     -->
11
 <ahref="${pageContext.request.contextPath}/account/findAc
count">查询账户</a>
12      <br/>
13      <!--第二种访问方式     -->
14      <a href="account/findAccount">查询账户</a>
15  </body>
16  </html>
```

注意：

1.2.2 method 属性的示例：

控制器代码：

```
1   /**
2   * 保存账户
3   * @return
4   */
5
@RequestMapping(value="/saveAccount",method=RequestMethod.
POST)
6   public String saveAccount() {
7       System.out.println("保存了账户");
8       return "success";
9}
```

jsp 代码：

```
1   <!-- 请求方式的示例     -->
2 <a href="account/saveAccount">保存账户，get 请求</a>
3   <br/>
4   <form action="account/saveAccount" method="post">
5 <input type="submit" value="保存账户，post 请求">
6   </form>
```

注意： 当使用 get 请求时，提示错误信息是 405，信息是方法不支持 get 方式请求

1.2.3 params 属性的示例：

控制器的代码：

```
/**
 * 删除账户
 * @return
 */
@RequestMapping(value="/removeAccount",params=
{"accountName","money=100"})
public String removeAccount() {
    System.out.println("删除了账户");
    return "success";
}
```

jsp 中的代码：

```
<!-- 请求参数的示例     -->
<a href="account/removeAccount?
accountName=aaa&money=100">删除账户，
金额     100</a>
<br/>
<a href="account/removeAccount?
accountName=aaa&money=150">删除账户，金额150</a>
```

注意： 当我们点击第一个超链接时,可以访问成功。 当我们点击第二个超链接时，无法访问。如下图：

2. RequestParam注解

对照《SpringMVC专题(三)-SpringMVC的常用注解.md》中1.4节。

2.1 使用说明

源码（部分）：

```
@Target(value=PARAMETER)
@Retention(value=RUNTIME)
@Documented
public @interface RequestParam
```

作用： 把请求中指定名称的参数给控制器中的形参赋值，在旅游项目的分页查询中也使用defaultValue给后端传递默认分页大小。
属性：
value：请求参数中的名称。

required：请求参数中是否必须提供此参数。默认值：true。表示必须提供，如果不提供将报错。

2.2 使用示例

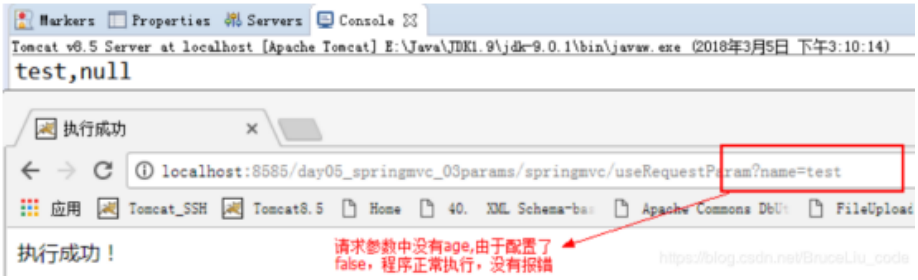
jsp

```
<!--RequestParam注解的使用-->
<a href="springmvc/useRequestParam?name=test">requestParam
注解</a>
```

控制器中的代码：

```
1  /**
2  * requestParams 注解的使用
3  * @param username
4  * @return
5  */
6  @RequestMapping("/useRequestParam")
7  public String
useRequestParam(@RequestParam("name")String
username,@RequestParam(value="age",required=false)Integer
age){
9       System.out.println(username+","+age);
10       return "success";
11  }
```

运行结果：



3. RequestBody注解

3.1 使用说明

源码（部分）：

```
@Target(value=PARAMETER)
@Retention(value=RUNTIME)
@Documented
public @interface RequestBody
```

- 作用： 用于获取请求体内容。直接使用得到是 key=value&key=value...结构的数据。 get 请求方式不适用。
- 属性：
    - required：是否必须有请求体。默认值是:true。当取值为 true 时,get 请求方式会报错。如果取值为 false，get 请求得到是 null。

3.2 使用示例

post 请求 jsp 代码：

```
<!-- request body 注解     -->
<form action="springmvc/useRequestBody" method="post">
    用户名称：<input type="text" name="username"><br/>
用户密码：<input type="password" name="password"><br/>
    用户年龄：<input type="text" name="age" ><br/>
    <input type="submit" value="保存">
</form>
```

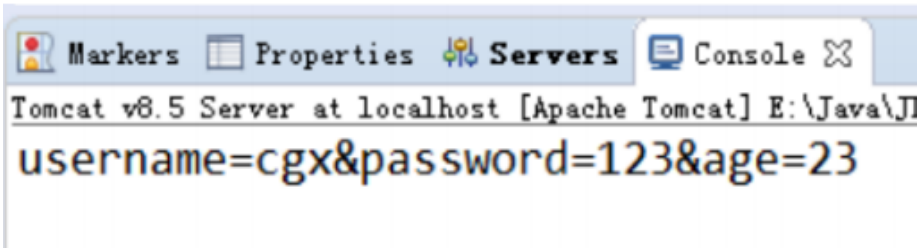
get 请求 jsp 代码：

```
<a href="springmvc/useRequestBody?body=test">requestBody注解get请求</a>
```

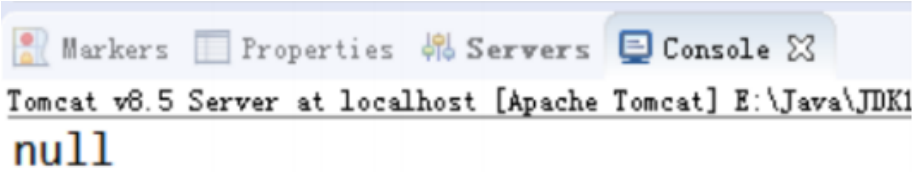
控制器代码：

```
/**
 * RequestBody 注解
 * @param user
 * @return
 */
@RequestMapping("/useRequestBody")
public String
useRequestBody(@RequestBody(required=false) String
body){
    System.out.println(body);
    return "success";
 }
```

post 请求运行结果：



> *get 请求运行结果:*



4. PathVariable注解

对照《SpringMVC专题(三)-SpringMVC的常用注解.md》中1.6节。

4.1 使用说明

- 作用： 用于绑定 url 中的占位符。例如：请求 url 中 /delete/{id}，这个{id}就是 url 占位符。 url 支持占位符是 spring3.0 之后加入的。是 springmvc 支持 rest 风格URL 的一个重要标志.
- 属性：
    - value：用于指定 url 中占位符名称。
    - required：是否必须提供占位符。

4.2 使用示例

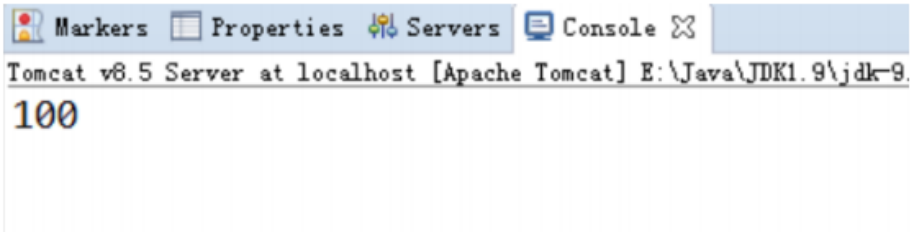
jsp代码:

```
<!--PathVariable注解-->
<a href="springmvc/usePathVariable/100">pathVariable 注解
</a>
```

控制器代码：

```
1  /**
2  * PathVariable 注解
3  * @param user
4  * @return
5  */
6 @RequestMapping("/usePathVariable/{id}")
7 public String usePathVariable(@PathVariable("id")
Integer id){
8     System.out.println(id);
9     return "success";
10  }
```

运行结果：



4.3 REST 风格 URL

4.3.1 什么是 rest

REST（英文：Representational State Transfer，简称 REST）描述了一个架构样式的网络系统，比如 web 应用程序。它首次出现在 2000 年 Roy Fielding 的博士论文中，他是 HTTP 规范的主要编写者之一。在目前主流的三种 Web 服务交互方案中，REST 相比于 SOAP（Simple Object Access protocol，简单对象访问协议）以及 XML-RPC 更加简单明了，无论是对 URL 的处理还是对 Payload 的编码，REST 都倾向于用更加简单轻量的方法设计和实现。值得注意的是 REST 并没有一个明确的标准，而更像是一种设计的风格。它本身并没有什么实用性，其核心价值在于如何设计出符合 REST 风格的网络接口。(这段话的意思就是：rest是URL的一种请求方式、设计风格、设计规范)

4.3.2 restful 的优点

它结构清晰、符合标准、易于理解、扩展方便，所以正得到越来越多网站的采用。

4.3.3 restful 的特性

- 资源（Resources）：网络上的一个实体，或者说是网络上的一个具体信息。 它可以是一段文本、一张图片、一首歌曲、一种服务，总之就是一个具体的存在。可以用一个 URI（统一资源定位符）指向它，每种资源对应一个特定的 URI。要获取这个资源，访问它的 URI 就可以，因此 URI 即为每一个资源的独一无二的识别符。
- 表现层（Representation）：把资源具体呈现出来的形式，叫做它的表现层（Representation）。 比如，文本可以用 txt 格式表现，也可以用 HTML 格式、XML 格式、JSON 格式表现，甚至可以采用二进制格式。
- 状态转化（State Transfer）：每 发出一个请求，就代表了客户端和服务器的一次交互过程。 HTTP 协议，是一个无状态协议，即所有的状态都保存在服务器端。因此，如果客户端想要操作服务器，必须通过某种手段，让服务器端发生"状态转化"（State Transfer）。而这种转化是建立在表现层之上的，所以就是"表现层状态转化"。具体说，就是 HTTP 协议里面，四个表示操作方式的动词：GET 、POST 、PUT、DELETE。它们分别对应四种基本操作：

```
1  GET 用来获取资源
2  POST 用来新建资源
3  PUT 用来更新资源
4  DELETE 用来删除资源
```

4.3.4 restful 的示例：

```
1 /account/1     HTTP GET :         得到 id = 1 的account
2 /account/1     HTTP DELETE:       删除 id = 1 的account
3 /account/1     HTTP PUT:         更新 id = 1 的account
4 /account       HTTP POST:            新增account
```

4.4 基于HiddenHttpMethodFilter的示例

org.springframework.web.filter包中的类

- 作用： 由于浏览器 form 表单只支持 GET 与 POST 请求，而 DELETE、PUT 等method 并不支持，Spring3.0 添加了一个过滤器，可以将浏览器请求改为指定的请求方式，发送给我们的控制器方法，使得支持 GET、POST、PUT 与 DELETE 请求。
- 使用方法：
    - 第一步：在 web.xml 中配置该过滤器。
    - 第二步：请求方式必须使用post 请求。
    - 第三步：在页面的post请求的标签中按照要求提供_method 请求参数，该参数的取值就是我们需要的请求方式。
- 源码分析：略

web.xml添加：

```
<!--配置HiddenHttpMethodFilter-->
<!--
            配置
org.springframework.web.filter.HiddenHttpMethodFilter: 可
以把 POST 请求转为 DELETE 或 PUT 请求
    -->
<filter>
    <filter-name>HiddenHttpMethodFilter</filter-name>
    <filter-
class>org.springframework.web.filter.HiddenHttpMethodFilte
r</filter-class>
</filter>
<filter-mapping>
    <filter-name>HiddenHttpMethodFilter</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
```

jsp中示例代码：（在页面中隐藏_method字段）

```
1  <!-- 保存     -->
2  <form action="springmvc/testRestPOST" method="post">
```

```
    用户名称：<input type="text" name="username"><br/>
<!-- <input type="hidden" name="_method" value="POST"> -->
    <input type="submit" value="保存">
</form>

<hr/>
<!-- 更新    -->
<form action="springmvc/testRestPUT/1" method="post">
    用户名称：<input type="text" name="username"><br/>
    <input type="hidden" name="_method" value="PUT">
    <input type="submit" value="更新">
</form>
<hr/>
<!-- 删除    -->
<form action="springmvc/testRestDELETE/1" method="post">
    <input type="hidden" name="_method" value="DELETE">
    <input type="submit" value="删除">
</form>
<hr/>
<!-- 查询一个    -->
<form action="springmvc/testRestGET/1" method="post">
    <input type="hidden" name="_method" value="GET">
    <input type="submit" value="查询">
</form>
```

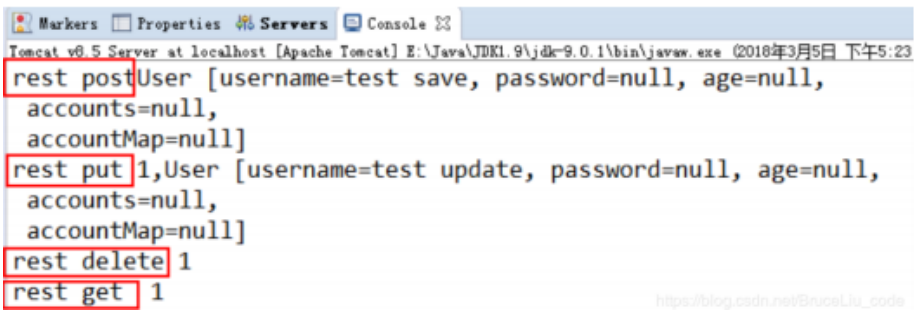
控制器中示例代码

```
/**
 * post 请求：保存
 * @param username
 * @return
 */
@RequestMapping(value="/testRestPOST",method=RequestMethod.POST)
public String testRestfulURLPOST(User user){
    System.out.println("rest post"+user);
    return "success";
}

/**
 * put 请求：更新
 * @param username
 * @return
 */
@RequestMapping(value="/testRestPUT/{id}",method=RequestMethod.PU
```

```
T)
18  public String
testRestfulURLPUT(@PathVariable("id")Integer
id,User user){
19      System.out.println("rest put "+id+","+user);
20      return "success";
21  }
22
23  /**
24  * post 请求：删除
25  * @param username
26  * @return
27  */
28
 @RequestMapping(value="/testRestDELETE/{id}",method=Reque
stMethod
.DELETE)
29  public String
testRestfulURLDELETE(@PathVariable("id")Integer id)
{
30      System.out.println("rest delete "+id);
31      return "success";
32  }
33
34  /**
35  * post 请求：查询
36  * @param username
37  * @return
38  */
39
 @RequestMapping(value="/testRestGET/{id}",method=RequestM
ethod.GE
T)
40  public String
testRestfulURLGET(@PathVariable("id")Integer id){
41      System.out.println("rest get "+id);
42      return "success";
43  }
```

运行结果



5. RequestHeader注解

5.1 使用说明

源码（部分）：

```
@Target(value=PARAMETER)
@Retention(value=RUNTIME)
@Documented
public @interface RequestHeader
```

作用： 用于获取请求消息头。

属性： value：提供消息头名称 required：是否必须有此消息头

5.2 使用示例

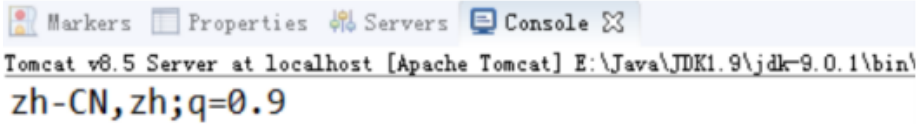
jsp中代码

```
<!-- RequestHeader 注解     -->
<a href="springmvc/useRequestHeader">获取请求消息头</a>
```

控制器中代码:

```
1  /**
2  * RequestHeader 注解
3  * @param user
4  * @return
5  */
6  @RequestMapping("/useRequestHeader")
7  public String
 useRequestHeader(@RequestHeader(value="Accept-
Language",required=false)String requestHeader){
9       System.out.println(requestHeader);
10       return "success";
11  }
```

运行结果：



6. CookieValue注解

6.1 使用说明

源码（部分）：

```
@Target(value=PARAMETER)
@Retention(value=RUNTIME)
@Documented
public @interface CookieValue
```

- 作用： 用于把指定 cookie 名称的值传入控制器方法参数。
- 属性： value：指定cookie 的名称。 required：是否必须有此 cookie。

6.2 使用示例

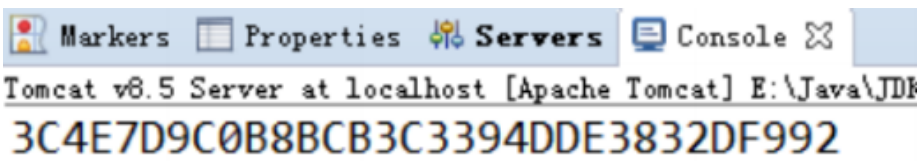
jsp中代码：

```
<!-- CookieValue 注解    -->
<a href="springmvc/useCookieValue">绑定 cookie 的值</a>
```

控制器中的代码：

```
1  /**
2  * Cookie 注解注解
3  * @param user
4  * @return
5  */
6  @RequestMapping("/useCookieValue")
7  public String
useCookieValue(@CookieValue(value="JSESSIONID",required=fa
lse)String cookieValue){
9      System.out.println(cookieValue);
10      return "success";
11  }
```

运行结果：



7. ModelAttribute注解

7.1 使用说明

源码（部分）：

```
@Target(value={PARAMETER,METHOD})
@Retention(value=RUNTIME)
@Documented
public @interface ModelAttribute
```

- 作用： 该注解是 SpringMVC4.3 版本以后新加入的。它可以用于修饰方法和参数。 出现在方法上，表示当前方法会在控制器中的其他的方法执行之前，先执行。它可以修饰没有返回值的方法，也可以修饰有具体返回值的方法。出现在参数上， 获取指定的数据给参数赋值。
- 属性： value：用于获取数据的 key。key 可以是 POJO 的属性名称，也可以是map 结构的 key。
- 应用场景： 当表单提交数据不是完整的实体类数据时，保证没有提交数据的字段使用数据库对象原来的数据。
- 例如： 我们在编辑一个用户时，用户有一个创建信息字段，该字段的值是不允许被修改的。在提交表单数据是肯定没有此字段的内容，一旦更新会把该字段内容置为 null，此时就可以使用此注解解决问题。

7.2 使用示例

7.2.1 基于 POJO 属性的基本使用

jps 代码：

```
<!-- ModelAttribute 注解的基本使用     -->
<a href="springmvc/testModelAttribute?username=test">测试
modelattribute</a>
```

控制器代码：

```
1  /**
2  * 被    ModelAttribute 修饰的方法
3  * @param user
4  */
5  @ModelAttribute
6  public void showModel(User user) {
7 System.out.println("执行了 showModel 方
法"+user.getUsername());
8}
9
10  /**
11  * 接收请求的方法
12  * @param user
13  * @return
14  */
```

```
15  @RequestMapping("/testModelAttribute")
16  public String testModelAttribute(User user) {
17      System.out.println("执行了控制器的方
法"+user.getUsername());
18      return "success";
19  }
```

运行结果：

7.2.2 基于pojo属性的应用场景

示例 1：ModelAttribute 修饰方法带返回值

- 需求：修改用户信息，要求用户的密码不能修改
- jsp代码:

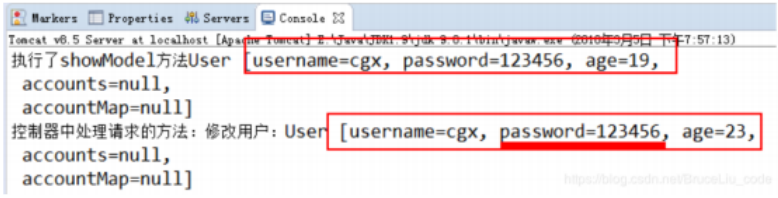
```
1  <!-- 修改用户信息     -->
2  <form action="springmvc/updateUser"
method="post">
3       用户名称：<input type="text"
name="username" ><br/>
4      用户年龄：<input type="text" name="age" >
<br/>
5      <input type="submit" value="保存">
6  </form>
```

- 控制的代码：

```
1  /**
2  * 查询数据库中用户信息
3  * @param user
4  */
5  @ModelAttribute
6  public User showModel(String username) {
7      //模拟去数据库查询
8      User abc = findUserByName(username);
9      System.out.println("执行了     showModel 方
法"+abc);
10      return abc;
11  }
12
13  /**
14  * 模拟修改用户方法
15  * @param user
16  * @return
```

```
17  */
18  @RequestMapping("/updateUser")
19  public String testModelAttribute(User user) {
20 System.out.println("控制器中处理请求的方法：修改用
户："+user);
21      return "success";
22  }
23
24  /**
25  * 模拟去数据库查询
26  * @param username
27  * @return
28  */
29  private User findUserByName(String username)
{
30      User user = new User();
31      user.setUsername(username);
32      user.setAge(19);
33      user.setPassword("123456");
34      return user;
35  }
```

- 运行结果：



7.2.3 基于 Map 的应用场景

示例2：ModelAttribute 修饰方法不带返回值

- 需求：修改用户信息，要求用户的密码不能修改
- jsp 中的代码：

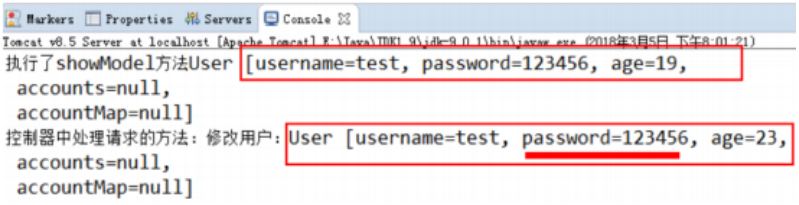
```
1  <!-- 修改用户信息     -->
2  <form action="springmvc/updateUser"
method="post">
3       用户名称：<input type="text"
name="username" ><br/>
4      用户年龄：<input type="text" name="age" >
<br/>
5      <input type="submit" value="保存">
6  </form>
```

- 控制器中的代码：

```java
/**
 * 查询数据库中用户信息
 * @param user
 */
@ModelAttribute
public void showModel(String username,Map<String,User> map) {
    //模拟去数据库查询
    User user = findUserByName(username);
    System.out.println("执行了    showModel 方法"+user);
     map.put("abc",user);
}

/**
 * 模拟修改用户方法
 * @param user
 * @return
 */
@RequestMapping("/updateUser")
public String testModelAttribute(@ModelAttribute("abc")User user)
{
    System.out.println("控制器中处理请求的方法：修改用户："+user);
     return "success";
}
/**
 * 模拟去数据库查询
 * @param username
 * @return
 */
private User findUserByName(String username) {
    User user = new User();
    user.setUsername(username);
    user.setAge(19);
    user.setPassword("123456");
    return user;
}
```

- 运行结果:



8. SessionAttribute注解

8.1 使用说明

作用： 用于多次执行控制器方法间的参数共享。

属性： value：用于指定存入的属性名称 type：用于指定存入的数据类型。

8.2 使用示例

源码（部分）：

```
@Target(value=PARAMETER)
@Retention(value=RUNTIME)
@Documented
public @interface SessionAttribute
```

jsp中的代码:

```
1  <!-- SessionAttribute 注解的使用    -->
2 <a href="springmvc/testPut">存入 SessionAttribute</a>
<hr/>
3 <a href="springmvc/testGet">取出 SessionAttribute</a>
<hr/>
4  <a href="springmvc/testClean">清除
SessionAttribute</a>
```

控制器中的代码：

```
1  /**
2  * SessionAttribute 注解的使用
3  * @Version 1.0
4  */
5 @Controller("sessionAttributeController")
6  @RequestMapping("/springmvc")
运行结果：
7  @SessionAttributes(value =
{"username","password"},types=
{Integer.class})
8  public class SessionAttributeController {
9
10      /**
11      * 把数据存入    SessionAttribute
12      * @param model
13      * @return
```

```
14      * Model 是    spring 提供的一个接口，该接口有一个实现类
ExtendedModelMap
15      * 该类继承了    ModelMap，而    ModelMap 就是
LinkedHashMap 子类
16      */
17      @RequestMapping("/testPut")
18      public String testPut(Model model){
19          model.addAttribute("username", "泰斯特");
20          model.addAttribute("password","123456");
21          model.addAttribute("age", 31);
22          //跳转之前将数据保存到    username、password 和
age中，因为
            //注解@SessionAttribute 中有这几个参数
23          return "success";
24      }
25
26      @RequestMapping("/testGet")
27      public String testGet(ModelMap model)
{System.out.println(model.get("username")+";"+model.get("p
assword
")+";"+model.get("a
28      ge"));
29          return "success";
30      }
31
32      @RequestMapping("/testClean")
33      public String complete(SessionStatus
sessionStatus){
34          sessionStatus.setComplete();
35          return "success";
36      }
37  }
```

运行结果：