1. 返回值分类

> *在SpringMVC 中常见的返回类型如下：*
>
> - *ModelAndView*
> - *Model*
> - *View*
> - *String*
> - *void*
> - *HttpEntity或ResponseEntity*
> - *Callable<?>*
> - *DeferredResult<?>*
>
> *在上述所列举的返回类型中，常见的返回类型是ModelAndView、String、void。其中ModelAndView类型中可以添加Model数据，并指定视图，String类型的返回值可以跳转视图，但是不能携带数据，而void类型的返回值主要在异步晴子u中使用，它只能返回数据，不能跳转视图。由于ModelAndView类型未能实现数据与视图之间的解耦，所以在企业开发的时候方法的返回类型通常都会使用String。而String的返回类型不能携带参数，那么在方法中是如何携带参数的呢？这就用到了上面介绍的Model参数类型，通过该参数类型即可以添加需要在视图中显示的属性。*

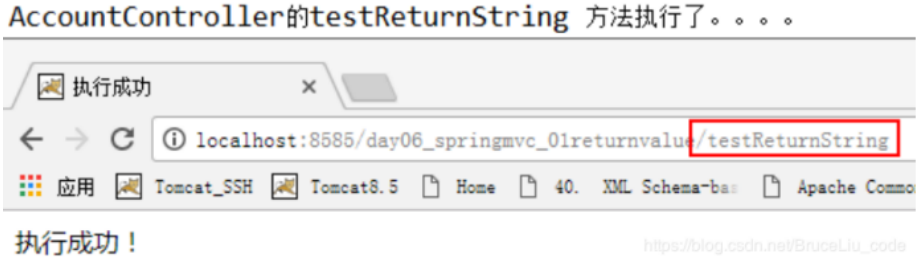
1.1 字符串

controller 方法返回字符串可以指定逻辑视图名，通过视图解析器解析为物理视图地址。

Croller代码：

```
1  /**指定逻辑视图名，经过视图解析器解析为     jsp 物理路径：/WEB-
INF/pages/success.jsp,该文件中的内容是：执行成功!
*/
2  @RequestMapping("/testReturnString")
3  public String testReturnString() {
4      System.out.println("AccountController的
testReturnString 方法执行了。。。。");
5      return "success";
6}
```

运行结果：



1.2 void

我们知道 Servlet 原始 API 可以作为控制器中方法的参数：

```
1  @RequestMapping("/testReturnVoid")
2  public void testReturnVoid(HttpServletRequest
request,HttpServletResponse response)
3  throws Exception {
4}
```

在 controller 方法形参上可以定义 request 和 response，使用 request 或 response 指定响应结果：
1、使用 request 转向页面，如下：

```
request.getRequestDispatcher("/WEB-
INF/pages/success.jsp").forward(request, response)
```

2、也可以通过 response 页面重定向：

（不能重定向到WEB-INF目录下的静态资源，但是可以通过重定向到另一个后端的控制器）

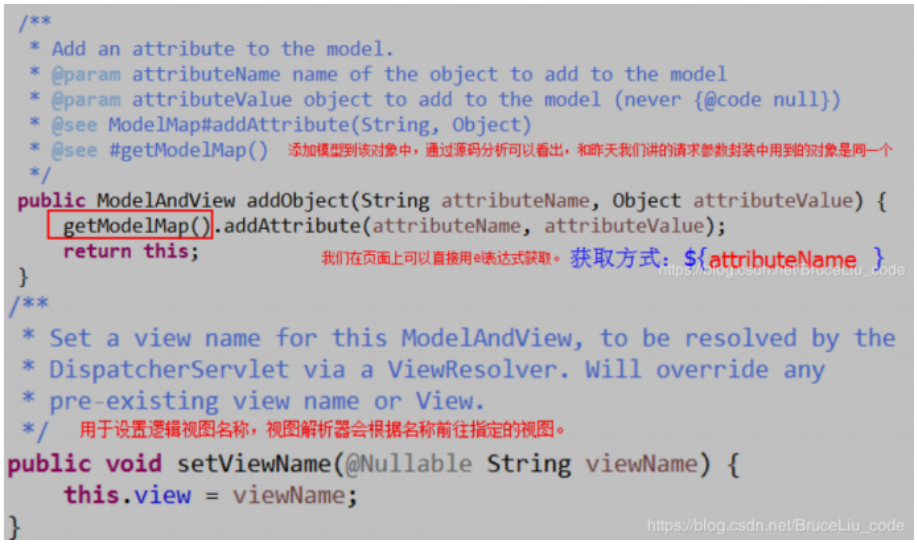
```
response.sendRedirect("testRetrunString");
```

3、也可以通过 response 指定响应结果，例如响应 json 数据：

```
response.setCharacterEncoding("utf-8");
response.setContentType("application/json;charset=utf-8");
response.getWriter().write("json 串");
```

1.3 ModelAndView

ModelAndView 是 SpringMVC 为我们提供的一个对象，该对象也可以用作控制器方法的返回值。 该对象中有两个方法：



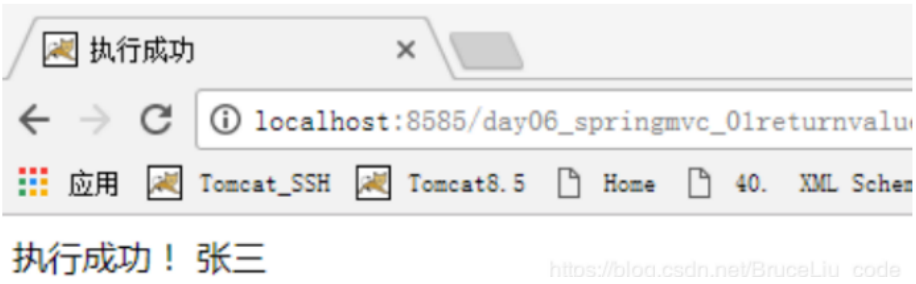
示例代码：

```
/**
 * 返回ModeAndView
 * @return
 */
@RequestMapping("/testReturnModelAndView")
public ModelAndView testReturnModelAndView() {
    ModelAndView mv = new ModelAndView();
    mv.addObject("username", "张三");
    mv.setViewName("success");
    return mv;
}
```

响应的 jsp 代码：

```
<%@ page language="java" contentType="text/html;
charset=UTF-8"
pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01
Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd"> <html> <head>
<meta http-equiv="Content-Type"
content="text/html;charset=UTF-8"> <title>执行成功</title>
</head>
<body>
    执行成功!
    ${requestScope.username}
</body>
</html>
```

输出结果：



注意：我们在页面上上获取使用的是 requestScope.username 取的，所以返回 ModelAndView 类型时，浏览器跳转只能是请求转发。

1.4 转发和重定向

1.4.1 forward 转发

controller 方法在提供了 String 类型的返回值之后，默认就是请求转发。我们也可以写成：

```
/**
 * 转发
 * @return
 */
@RequestMapping("/testForward")
public String testForward() {
    System.out.println("AccountController的testForward方法执行了。。。。");
    return "forward:/WEB-INF/pages/success.jsp";
}
```

需要注意的是，如果用了——> forward：则路径必须写成实际视图 url。 它相当于`request.getRequestDispatcher("url").forward(request,response)` 。使用请求 转发，既可以转发到 jsp，也可以转发到其他的控制器方法。

1.4.2 Redirect 重定向

contrller 方法提供了一个 String 类型返回值之后，它需要在返回值里使用——>redirect:

```
/**
 * 重定向
 * @return
 */
@RequestMapping("/testRedirect")
public String testRedirect() {
    System.out.println("AccountController的testRedirect方法执行了。。。。");
    return "redirect:testReturnModelAndView";
}
```

它相当于`response.sendRedirect(url)`。需要注意的是，如果是重定向到 jsp 页面，则 jsp 页面不能写在 WEB-INF 目录中，否则无法找到；如果非要访问 WEB-INF 目录中的静态资源，可以通过重定向到另一个控制器中，例如示例代码中的testReturnModelAndView，在这个testReturnModelAndView控制器中返回了WEB-INF 目录下的静态资源。

1.5 @ResponseBody响应json数据

1.5.1 使用说明

源码（部分）：

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

作用： 该注解用于将 Controller 的方法返回的对象，通过 HttpMessageConverter接口转换为指定格式的数据如：json,xml 等，通过 Response 响应给客户端.

1.5.2 使用示例

需求： 使用@ResponseBody 注解实现将 controller 方法返回对象转换为 json 响应给客户端。

前置知识点： Springmvc 默认用 MappingJacksonHttpMessageConverter 对 json数据进行转换，需要加入 jackson 的包。

```
<dependency>
  <groupId>com.fasterxml.jackson.core</groupId>
    <artifactId>jackson-databind</artifactId>
    <version>2.9.0</version>
</dependency>
```

> *注意：2.7.0 以下的版本用不了*

jsp 中的代码：

```
<script type="text/javascript"
 src="${pageContext.request.contextPath}/js/jquery.min.js"
>
</script> <script type="text/javascript">
    $(function(){
        $("#testJson").click(function(){
            $.ajax({
                type:"post",
url:"${pageContext.request.contextPath}/testResponseJson",
contentType:"application/json;charset=utf-8",
 data:'{"id":1,"name":"test","money":999.0}',
                dataType:"json",
                success:function(data){
                 alert(data);
                }
            });
        });
```

```
16      })
17  </script>
18  <!-- 测试异步请求     -->
19  <input type="button" value="测试ajax 请求json和响应json"
id="testJson"/>
```

控制器中的代码：

```
1  /**
2  * 响应json数据的控制器
3  * @Version 1.0
4  */
5  @Controller("jsonController")
6  public class JsonController {
7
8  /**
9  * 测试响应     json 数据
10  */
11  @RequestMapping("/testResponseJson")
    //@ResponseBody  //@ResponseBody注解也可以写在这里
12  public @ResponseBody Account
testResponseJson(@RequestBody Account account) {
13      System.out.println("异步请求："+account);
14      return account;
15     }
16  }
```

运行结果：