

## JSP 页面的转发和重定向

Spring MVC 默认以转发的形式响应 JSP，可以手动进行修改。

重定向

```
@RequestMapping("/restful/{id}/{name}")
public String restful(@PathVariable("id") Integer id,@PathVariable("name")
String name){
    System.out.println(id+"-"+name);
    return "redirect:/index.jsp";
}
```

设置重定向的时候不能写逻辑视图，必须写明资源的物理路径，如“redirect: /index.jsp”

转发

```
@RequestMapping("/restful/{id}/{name}")
public String restful(@PathVariable("id") Integer id,@PathVariable("name")
String name){
    System.out.println(id+"-"+name);
    return "forward:/index.jsp";
}
```

等同于

```
@RequestMapping("/restful/{id}/{name}")
public String restful(@PathVariable("id") Integer id,@PathVariable("name")
String name){
    System.out.println(id+"-"+name);
    return "index";
}
```

## Spring MVC 数据绑定

数据绑定：在后台业务方法中，直接获取前端 HTTP 请求中的参数。

HTTP 请求传输的参数都是 String 类型的，Handler 业务方法中的参数是开发者指定的数据类型，int、Integer、Object，因此需要进行数据类型的转换。

Spring MVC 的 HandlerAdapter 组件会在执行 Handler 业务方法之前，完成参数的绑定，开发者直接使用即可。

- 基本数据类型

如果没有@ResponseBody那么视图解析器返回的就是/id1.jsp，假设id为1  
如果加了@ResponseBody注解，就会把返回的内容作为文本返回，不再进行视图的解析

```
@RequestMapping("/baseType")
@ResponseBody
public String baseType(int id){
    return "id:"+id;
}
```

客户端 HTTP 请求中必须包含 id 参数，否则抛出 500 异常，因为 id 不能为 null。

← → ↻ ⓘ localhost:8080/hello/baseType

## HTTP Status 500 – Internal Server Error

**Type** Exception Report

**Message** Request processing failed; nested exception is java.lang.IllegalStateException: Optional int parameter 'id' is present as a primitive type. Consider declaring it as object wrapper for the corresponding primitive type.

**Description** The server encountered an unexpected condition that prevented it from fulfilling the request.

**Exception**

```
org.springframework.web.util.NestedServletException: Request processing failed; nested
    org.springframework.web.servlet.FrameworkServlet.processRequest(FrameworkServlet
```

同时 id 的值必须为数值且必须为整数，否则抛出 400 异常。

← → ↻ ⓘ localhost:8080/hello/baseType?id=a

## HTTP Status 400 – Bad Request

**Type** Status Report

**Description** The server cannot or will not process the request due to something that is perceived to be a client error (e.g., malformed request syntax, invalid request message routing, or deceptive request routing).

Apache Tomcat/9.0.8

← → ↻ ⓘ localhost:8080/hello/baseType?id=1.5

## HTTP Status 400 – Bad Request

**Type** Status Report

**Description** The server cannot or will not process the request due to something that is perceived to be a client error (e.g., malformed request syntax, invalid request message routing, or deceptive request routing).

Apache Tomcat/9.0.8

- 包装类

```
@RequestMapping("/packageType")
@ResponseBody
public String packageType(Integer id){
    return "id:"+id;
}
```

如果 HTTP 请求中没有包含 id 参数，不会报错，id 的值就是 null，会直接返回 id:null 给客户端。

← → ↻ ⓘ localhost:8080/hello/packageType

id:null

但是如果 id = a 或者 id = 1.5，同样会抛出 400 异常，因为数据类型无法匹配。

← → ↻ ⓘ localhost:8080/hello/packageType?id=1.5

## HTTP Status 400 – Bad Request

### Type Status Report

**Description** The server cannot or will not process the request due to something that is deceptive request routing).

### Apache Tomcat/9.0.8

可以给参数列表添加 @RequestParam 注解，可以对参数进行相关设置。

```
@RequestMapping("/packageType")
@ResponseBody
public String packageType(@RequestParam(value = "id",required =
true,defaultValue = "0") Integer id){
    return "id:"+id;
}
```

- value = "id": 将 HTTP 请求中名为 id 的参数与 Handler 业务方法中的形参进行映射。
- required: true 表示 id 参数必填，false 表示非必填。
- defaultValue = "0": 表示当 HTTP 请求中没有 id 参数时，形参的默认值为 0。

## 数组

```
@RequestMapping("/arrayType")
@ResponseBody
public String arrayType(String[] names){
    StringBuffer stringBuffer = new StringBuffer();
    for (String str:names){
        stringBuffer.append(str).append(" ");
    }
    return "names:"+stringBuffer.toString();
}
```

## POJO

```
package com.southwind.entity;

import lombok.AllArgsConstructor;
import lombok.Data;

@Data
@AllArgsConstructor 加@NoArgsConstructor
public class User {
    private Integer id;
    private String name;
    private Address address;
}
```

```
package com.southwind.entity;

import lombok.Data;

@Data
public class Address {
    private Integer code;
    private String value;
}
```

```
<%--
    Created by IntelliJ IDEA.
    User: southwind
    Date: 2020-02-07
    Time: 16:19
    To change this template use File | Settings | File Templates.
--%>
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
```

```

<head>
    <title>Title</title>
</head>
<body>
    <form action="/hello/add" method="post">
        <table>
            <tr>
                <td>编号: </td>
                <td>
                    <input type="text" name="id" />
                </td>
            </tr>
            <tr>
                <td>姓名: </td>
                <td>
                    <input type="text" name="name" />
                </td>
            </tr>
            <tr>
                <td>地址编号: </td>
                <td>
                    <input type="text" name="address.code" />
                </td>
            </tr>
            <tr>
                <td>地址信息: </td>
                <td>
                    <input type="text" name="address.value" />
                </td>
            </tr>
            <tr>
                <td>
                    <input type="submit" value="提交" />
                </td>
            </tr>
        </table>
    </form>
</body>
</html>

```

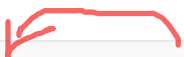
```

@RequestMapping(value = "/add",method = RequestMethod.POST)
@ResponseBody
public String add(User user){
    return user.toString();
}

```

解决响应时乱码问题，springmvc.xml 中配置转换器即可。

我们配置了两个转换，一个（web.xml）是从客户端传到服务器的乱码解决方法，一个（springmvc.xml）是从服务器到客户端的乱码解决方法。

 `<mvc:annotation-driven>` 让注解生效

```
<!-- 消息转换器 -->
<mvc:message-converters>
    <bean
class="org.springframework.http.converter.StringHttpMessageConverter">
        <property name="supportedMediaTypes" value="text/html;charset=UTF-
8"></property>
    </bean>
</mvc:message-converters>
</mvc:annotation-driven>
```

## List

Spring MVC 不支持 List 类型的直接转换，需要包装成 Object。

List 的自定义包装类

```
package com.southwind.entity;

import lombok.Data;

import java.util.List;

@Data
public class UserList {
    private List<User> users;
}
```

业务方法

```
@RequestMapping("/listType")
@ResponseBody
public String listType(UserList users){
    StringBuffer stringBuffer = new StringBuffer();
    for(User user:users.getUsers()){
        stringBuffer.append(user);
    }
    return "用户: "+stringBuffer.toString();
}
```

JSP

```
<%--
Created by IntelliJ IDEA.
User: southwind
Date: 2020-02-08
Time: 15:16
To change this template use File | Settings | File Templates.
```

```

--%>
<%@ page contentType="text/html; charset=UTF-8" language="java" %>
<html>
<head>
    <title>Title</title>
</head>
<body>
    <form action="/hello/listType" method="post">
        用户1ID: <input type="text" name="users[0].id" /><br/>
        用户1姓名: <input type="text" name="users[0].name" /><br/>
        用户2ID: <input type="text" name="users[1].id" /><br/>
        用户2姓名: <input type="text" name="users[1].name" /><br/>
        用户3ID: <input type="text" name="users[2].id" /><br/>
        用户3姓名: <input type="text" name="users[2].name" /><br/>
        <input type="submit" value="提交" />
    </form>
</body>
</html>

```

表单提交把数据传递到后台后springmvc发现你要封装成一个userlist的集合，需要创建3个user对象，需要通过反射传入参数，但是如果user没有无参构造的化数据就传不进来了

需要注意的是 User 类一定要有无参构造函数，否则抛出异常。

## JSON

“java web开发中spring是很常用的,其IOC利用了java的反射,而spring的反射要求这个bean必须要有一个无参构造器。

JSP

```

<%--
    Created by IntelliJ IDEA.
    User: southwind
    Date: 2020-02-08
    Time: 15:25
    To change this template use File | Settings | File Templates.
--%>
<%@ page contentType="text/html; charset=UTF-8" language="java" %>
<html>
<head>
    <title>Title</title>
    <script type="text/javascript" src="js/jquery-1.8.3.min.js"></script>
    <script type="text/javascript">
        $(function(){
            var user = {
                "id":1,
                "name":"张三"
            };
            $.ajax({
                url:"/hello/jsonType",
                data:JSON.stringify(user),
                type:"POST",
                contentType:"application/json; charset=UTF-8",
                dataType:"JSON",

```

```

        success: function (data) {
            alert(data.id)
            alert(data.name)
        }
    })
}
</script>
</head>
<body>

</body>
</html>

```

注意

- JSON 数据必须用 `JSON.stringify()` 方法转换成字符串
- `contentType:"application/json;charset=UTF-8"` 不能省略

业务方法

```

@RequestMapping("/jsonType")
@ResponseBody
public User jsonType(@RequestBody User user){
    System.out.println(user);
    user.setId(2);
    return user;
}

```

`ResponseBody`是响应回去的

`RequestBody`是请求来的，json不能直接映射到User中需要加这个注解才能映射到实体

`@RequestBody` 注解

读取 HTTP 请求参数，通过 Spring MVC 提供的 `HttpMessageConverter` 接口将读取的参数转为 JSON、XML 格式的数据，绑定到业务方法的形参。

`@ResponseBody` 注解

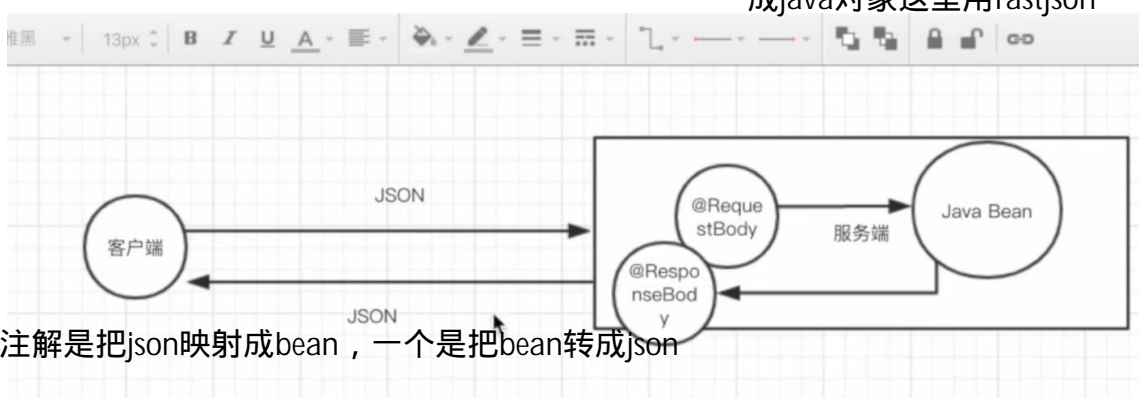
将业务方法返回的对象，通过 `HttpMessageConverter` 接口转为指定格式的数据，JSON、XML 等，响应给客户端。

需要使用组件结合 `@RequestBody` 注解将 JSON 转为 Java Bean，这里使用 `Fastjson`，其优势时候如果属性为空就不会将其转为 JSON。

如何使用 `Fastjson`

1、pom.xml 中添加 `Fastjson` 相关依赖。

前端传过来的是json格式的，后端要把json格式的转换成bean，仅仅加`RequestBody`注解是不够的，要借助第三方的工具，把json格式的进行转换解析成java对象这里用`fastjson`



一个注解是把json映射成bean，一个是把bean转成json



```

<!-- fastjson -->
<dependency>
  <groupId>com.alibaba</groupId>
  <artifactId>fastjson</artifactId>
  <version>1.2.32</version>
</dependency>

```

2、springmvc.xml 中配置 FastJson。

```

<mvc:annotation-driven>
  <!-- 消息转换器 -->
  <mvc:message-converters>
    <bean
      class="org.springframework.http.converter.StringHttpMessageConverter">
      <property name="supportedMediaTypes" value="text/html;charset=UTF-8"/></property>
    </bean>
    <!-- fastjson -->
    <bean
      class="com.alibaba.fastjson.support.spring.FastJsonHttpMessageConverter4">
    </bean>
  </mvc:message-converters>
</mvc:annotation-driven>

```

必须配置fastjson才能生效

## Spring MVC 视图层解析

调用 Web 资源给域对象传值

page

request

四个作用域

session

application

业务数据的绑定是指将业务数据绑定给 JSP 域对象，业务数据的绑定是由 ViewResolver 来完成的，开发时，我们先添加业务数据，再交给 ViewResolver 来绑定，因此学习的重点在于如何添加业务数据，Spring MVC 提供了以下几种方式来添加业务数据：

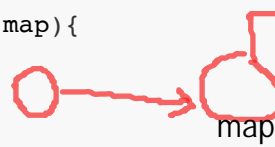
- Map
- Model
- ModelAndView
- @SessionAttribute
- @ModelAttribute
- Servlet API

## 业务数据绑定到 request 域对象

Map

Spring MVC 在调用业务方法之前会创建一个隐含对象作为业务数据的存储容器，设置业务方法的入参为 Map 类型，Spring MVC 会将隐含对象的引用传递给入参。

```
@RequestMapping("/map")
public String map(Map<String,Object> map){
    User user = new User();
    user.setId(1);
    user.setName("张三");
    map.put("user",user);
    return "show";
}
```



我们对map进行操作，springmvc会自动把map的东西取出来

springmvc 绑定是由 viewresolver来完成的  
jsp

把值放在了request域当中

```
<!--
  Created by IntelliJ IDEA.
  User: southwind
  Date: 2020-02-08
  Time: 18:05
  To change this template use File | Settings | File Templates.
-->
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<%@ page isELIgnored="false" %>
<html>
<head>
    <title>Title</title>
</head>
<body>
    ${requestScope.user}
</body>
</html>
```

## Model

Model 与 Map 类似，业务方法通过入参来完成业务数据的绑定。

```
@RequestMapping("/model")
public String model(Model model){
    User user = new User();
    user.setId(1);
    user.setName("张三");
    model.addAttribute("user",user);
    return "show";
}
```

Model是spring提供的

## ModelAndView

与 Map 或者 Model 不同的是，ModelAndView 不但包含业务数据，同时也封装了视图信息，如果使用 ModelAndView 来处理业务数据，业务方法的返回值必须是 ModelAndView 对象。

业务方法中对 ModelAndView 进行两个操作：

- 填充业务数据
- 绑定视图信息

```
@RequestMapping("/mav1")
public ModelAndView modelAndView1(){
    ModelAndView modelAndView = new ModelAndView();
    User user = new User();
    user.setId(1);
    user.setName("张三");
    //填充业务数据
    modelAndView.addObject("user",user);
    //绑定视图信息
    modelAndView.setViewName("show");
    return modelAndView;
}

@RequestMapping("/mav2")
public ModelAndView modelAndView2(){
    ModelAndView modelAndView = new ModelAndView();
    User user = new User();
    user.setId(1);
    user.setName("张三");
    modelAndView.addObject("user",user);
    View view = new InternalResourceView("/show.jsp");
    modelAndView.setView(view);
    return modelAndView;
}

@RequestMapping("/mav3")
public ModelAndView modelAndView3(){
    ModelAndView modelAndView = new ModelAndView("show");
    User user = new User();
    user.setId(1);
    user.setName("张三");
    modelAndView.addObject("user",user);
    return modelAndView;
}

@RequestMapping("/mav4")
public ModelAndView modelAndView4(){
    View view = new InternalResourceView("/show.jsp");
    ModelAndView modelAndView = new ModelAndView(view);
    User user = new User();
    user.setId(1);
    user.setName("张三");
    modelAndView.addObject("user",user);
    return modelAndView;
}
```

```

@RequestMapping( "/mav5" )
public ModelAndView modelAndView5(){
    Map<String,Object> map = new HashMap<>();
    User user = new User();
    user.setId(1);
    user.setName( "张三" );
    map.put( "user",user);
    ModelAndView modelAndView = new ModelAndView( "show",map);
    return modelAndView;
}

@RequestMapping( "/mav6" )
public ModelAndView modelAndView6(){
    Map<String,Object> map = new HashMap<>();
    User user = new User();
    user.setId(1);
    user.setName( "张三" );
    map.put( "user",user);
    View view = new InternalResourceView( "/show.jsp" );
    ModelAndView modelAndView = new ModelAndView( view,map);
    return modelAndView;
}

@RequestMapping( "/mav7" )
public ModelAndView modelAndView7(){
    User user = new User();
    user.setId(1);
    user.setName( "张三" );
    ModelAndView modelAndView = new ModelAndView( "show", "user",user);
    return modelAndView;
}

@RequestMapping( "/mav8" )
public ModelAndView modelAndView8(){
    User user = new User();
    user.setId(1);
    user.setName( "张三" );
    View view = new InternalResourceView( "/show.jsp" );
    ModelAndView modelAndView = new ModelAndView( view, "user",user);
    return modelAndView;
}

```