

韩顺平 SpringMVC(基于 Spring 的 Web 层 MVC 框架) - 2022 版

1 SpringMVC 介绍

1.1 官方文档

1.1.1 离线文档：解压

spring-5.3.8-dist.zip

spring-framework-5.3.8/docs/reference/html/web.html#spring-web

1.2 SpringMVC-基本介绍

1.2.1 SpringMVC 特点&概述

• SpringMVC 特点&概述



1. SpringMVC 从易用性，效率上 比曾经流行的 Struts2 更好

2. SpringMVC 是 WEB 层框架【老韩解读: SpringMVC 接管了 Web 层组件, 比如控制器, 视图, 视图解析, 返回给用户的数据格式, 同时支持 MVC 的开发模式/开发架构】
3. SpringMVC 通过注解, 让 POJO 成为控制器, 不需要继承类或者实现接口
4. SpringMVC 采用低耦合的组件设计方式, 具有更好扩展和灵活性.
5. 支持 REST 格式的 URL 请求.
6. SpringMVC 是基于 Spring 的, 也就是 SpringMVC 是在 Spring 基础上的。SpringMVC 的核心包 spring-webmvc-xx.jar 和 spring-web-xx.jar

1.2.2 梳理 Spring SpringMVC SpringBoot 的关系

1. Spring MVC 只是 Spring 处理 WEB 层请求的一个模块/组件, Spring MVC 的基石是 Servlet[Java WEB]
2. Spring Boot 是为了简化开发者的使用, 推出的封神框架(约定优于配置, 简化了 Spring 的配置流程), SpringBoot 包含很多组件/框架, Spring 就是最核心的内容之一, 也包含 Spring MVC
3. 他们的关系大概是: Spring Boot > Spring > Spring MVC

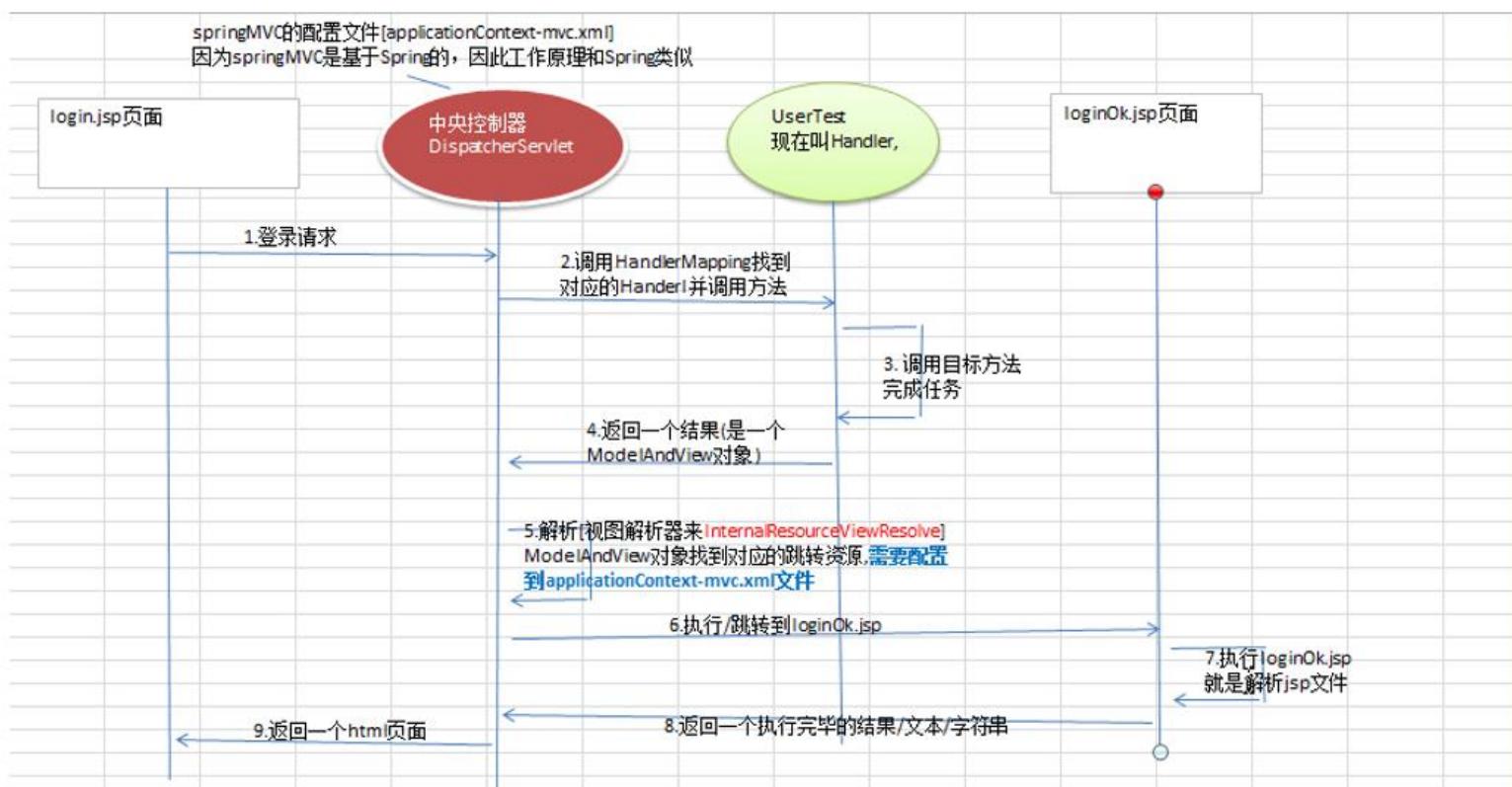
1.3 SpringMVC-快速入门

1.3.1 需求说明/图解

- 需求说明：完成一个最基本的测试案例，登录案例，使用 SpringMVC 完成



1.3.2 SpringMVC 登录流程分析

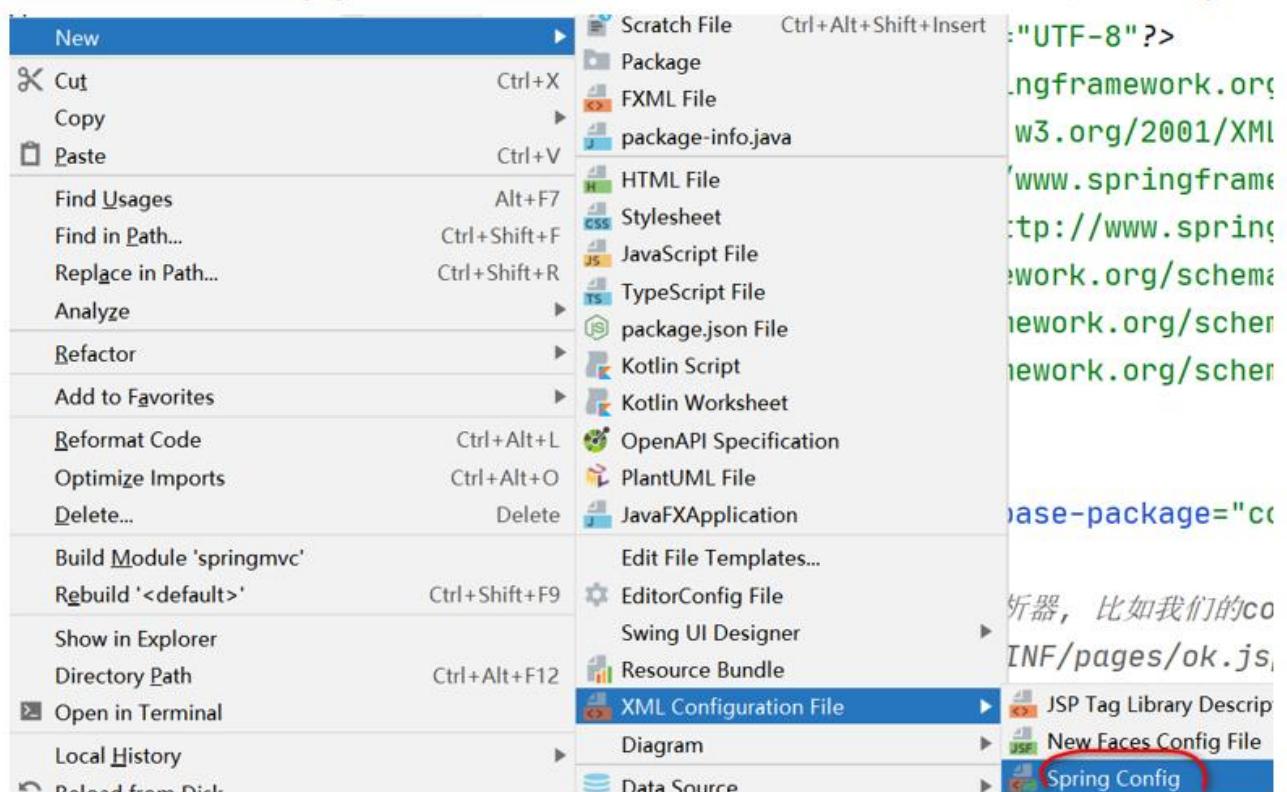


1.3.3 SpringMVC 登录-代码实现

1. 创建 springmvc web 工程并配置 tomcat, 前面我们学习过
2. 导入 SpringMVC 开发需要的 jar 包



3. 创建 src/applicationContext-mvc.xml 文件(就是 spring 的容器文件), 文件名自己定



4. 配置 WEB-INF/web.xml

<!-- The front controller of this Spring Web application, responsible for handling all application requests -->

1. 中央控制器
2. 负责处理所有的应用请求

-->

<servlet>

 <servlet-name>springDispatcherServlet</servlet-name>

 <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>

 <init-param>

 <param-name>contextConfigLocation</param-name>

```
<!-- springmvc 的配置文件和以前 spring 的配置文件类似 -->  
<param-value>classpath:applicationContext-mvc.xml</param-value>  
</init-param>  
  
<!-- 在 web 项目启动时，就加载这个 servlet 实例 1 表示加载的顺序号-->  
<load-on-startup>1</load-on-startup>  
</servlet>  
  
<!-- Map all requests to the DispatcherServlet for handling -->  
<servlet-mapping>  
    <servlet-name>springDispatcherServlet</servlet-name>  
    <!-- 为了支持 rest 风格的 url 这里的 url-pattern 需要写成 / -->  
    <url-pattern>/</url-pattern>  
</servlet-mapping>
```

5. 创建 D:\idea_java_projects\springmvc\web\login.jsp

```
<%--  
  
Created by IntelliJ IDEA.  
  
User: 韩顺平  
  
Version: 1.0  
  
Filename: login  
  
--%>  
  
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
```

```
<html>  
<head>  
    <title>登录</title>  
</head>  
<body>  
    <h3>登录页面</h3>  
    <form action=?>  
        u:<input name=username type=text> <br/>  
        p:<input name=password type=password><br/>  
        <input type=submit value=登录>  
    </form>  
</body>  
</html>
```

6. 创建 D:\idea_java_projects\springmvc\src\com\hspedu\web\UserServlet.java

```
package com.hspedu.web;  
  
import org.springframework.stereotype.Controller;  
import org.springframework.web.bind.annotation.RequestMapping;
```

```
/**  
 * @author 韩顺平  
 * @version 1.0  
 */  
  
//@Controller：被当做一个控制器，作用类似前面讲过的 Servlet  
//这里配置了一个 @Controller 就已经被 springMvc 框架接管  
  
@Controller  
  
public class UserServlet {  
  
    /**  
     * 老韩解读  
     * 1. @RequestMapping("/login")： /login 等价于 web.xml servlet 的 url-pattern  
     * 2. return "login_ok"  
     * 2.1 将结果 "login_ok" 返回给 InternalResourceViewResolver  
     * 2.2 然后跳转到哪个页面这里就是 /WEB-INF/pages/login_ok.jsp  
     * @return  
     */  
  
    @RequestMapping(value = "/login")  
    public String login() {  
        System.out.println("login ok....");  
        return "login_ok";  
    }  
}
```

7. 创建 D:\idea_java_projects\springmvc\web\WEB-INF\pages\login_ok.jsp

```
<%--  
Created by IntelliJ IDEA.  
User: 韩顺平  
Version: 1.0  
Filename: login_ok  
--%>  
<%@ page contentType="text/html;charset=UTF-8" language="java" %>  
<html>  
<head>  
    <title>登录成功</title>  
</head>  
<body>  
    <h1>恭喜 登录成功!</h1>  
</body>  
</html>
```

8. 配置 applicationContext-mvc.xml

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:context="http://www.springframework.org/schema/context"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context.xsd">

<!-- 配置自动扫描包 -->
<context:component-scan base-package="com.hspedu.web"/>

<!-- 配置 SpringMVC 的视图解析器, 比如我们的 controller return 的是 ok
那么这个页面就是 /WEB-INF/pages/ok.jsp
-->

<bean
      class="org.springframework.web.servlet.view.InternalResourceViewResolver">
    <property name="prefix" value="/WEB-INF/pages/"></property>
    <property name="suffix" value=".jsp"></property>
</bean>

</beans>
```

9. 完成登录测试

登录页面

u: hsp
p: ***

火狐官方站点 新手上路 常用网

恭喜 登录成功!

1.3.4 注意事项和细节说明

1. 重点学习如何搭建一个 springmvc 项目，初步理解 springmvc 工作流程
2. 这里的 UserServlet 需要注解成@Controller，我们称为一个 Handler 处理器
3. UserServlet 指定 url 时，还可以这样

```
@RequestMapping(value = "/login")
public String login() {
    System.out.println("login....");
    return "login_ok";
}
```

value 可以省略

4. 关于 SpringMVC 的 DispatcherServlet 的配置文件，如果不在 web.xml 指定 applicationContext-mvc.xml， 默认在 /WEB-INF/springDispatcherServlet-servlet.xml 找这个配置文件【简单看下 DispatcherServlet 的源码】。(推荐使用，我们做下修改，并完成测试)

```

/*
@SuppressWarnings("serial")
public abstract class FrameworkServlet extends HttpServletBean implements Application
{
    /**
     * Suffix for WebApplicationContext namespaces. If a servlet of this class is
     * given the name "test" in a context, the namespace used by the servlet will
     * resolve to "test-servlet".
    */
    public static final String DEFAULT_NAMESPACE_SUFFIX = "-servlet";
}

```

1) 修改 web.xml, 注销 init-param 配置节点

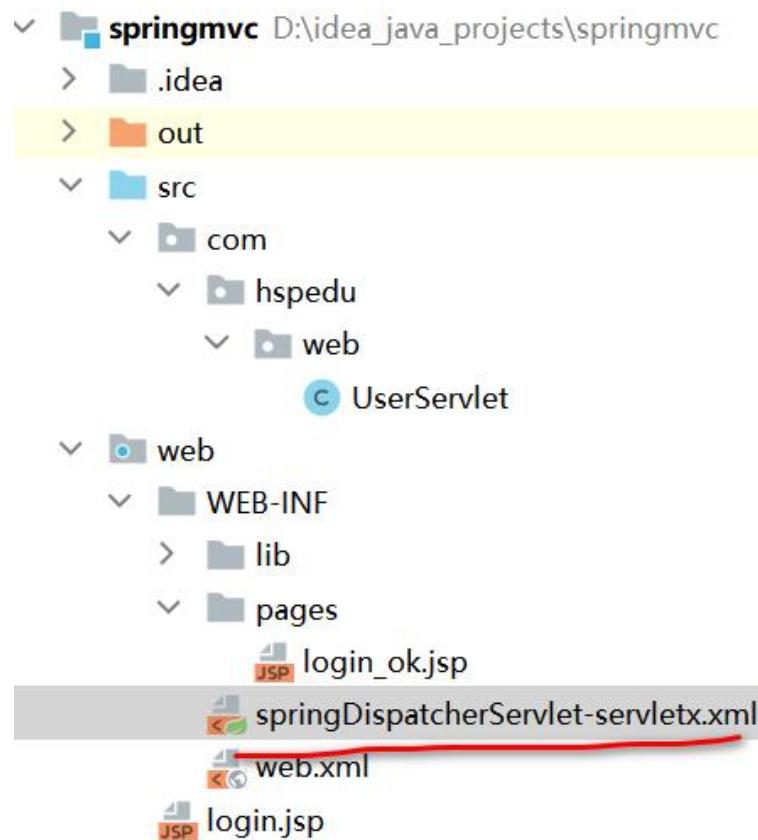
```

<servlet>
    <servlet-name>springDispatcherServlet</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
    <!--      <init-param>-->
    <!--          <param-name>contextConfigLocation</param-name>-->
    <!--          &lt;!&ndash; springmvc 的配置文件和以前 spring 的配置文件类似
&ndash;&gt;-->
    <!--          <param-value>classpath:applicationContext-mvc.xml</param-value>-->
    <!--          </init-param>-->
    <!-- 在 web 项目启动时, 就加载这个 servlet 实例 1 表示加载的顺序号-->
    <load-on-startup>1</load-on-startup>
</servlet>

```

2) 剪切原 applicationContext-mvc.xml 到 /WEB-INF 目录下, 文件名为: 你配置的 DispatcherServlet 名字 -servlet.xml 老师 这就是

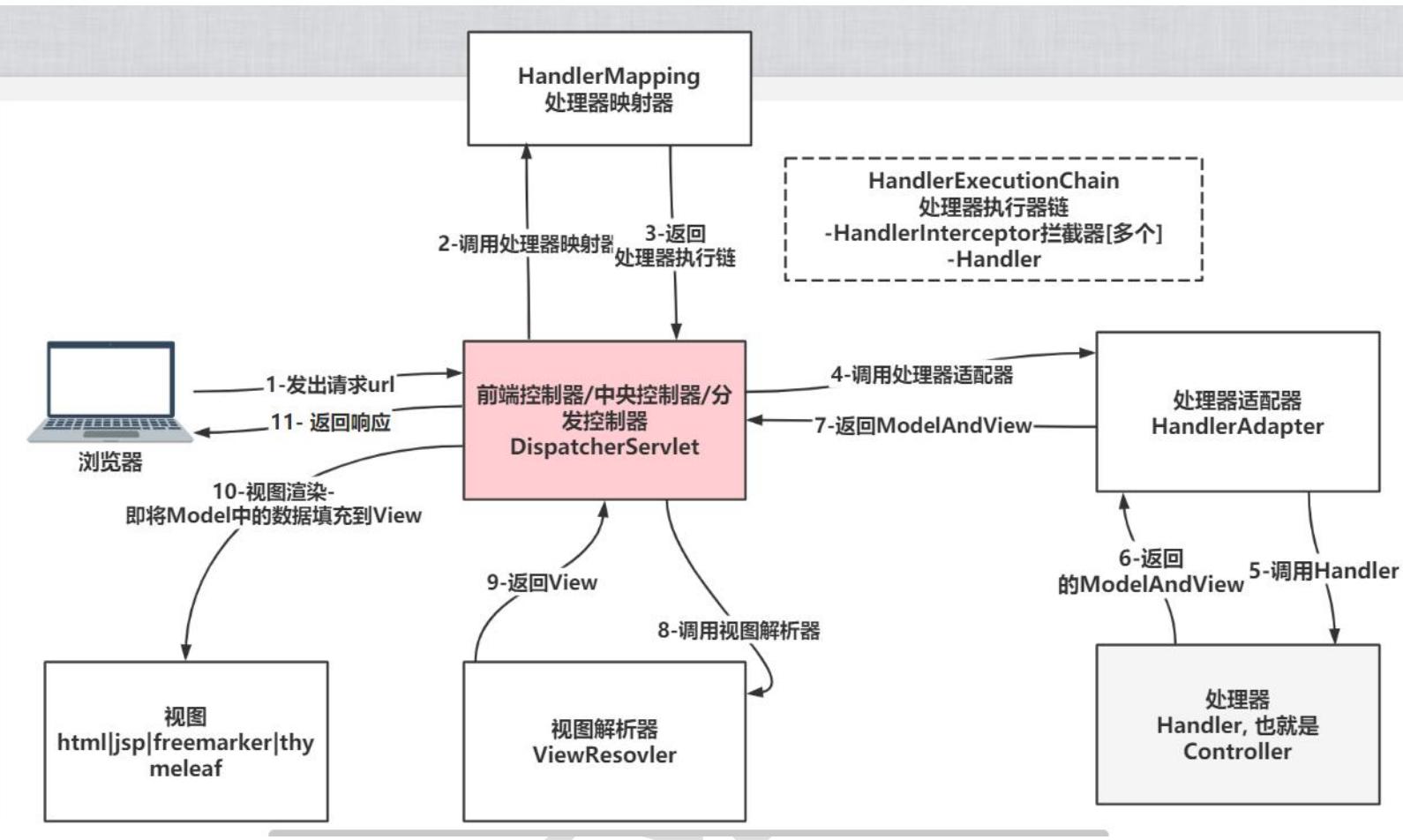
/WEB-INF/springDispatcherServlet-servlet.xml



2 SpringMVC 执行流程

2.1 一图胜千言

- 一图胜千言【自己画 2 遍】



2.2 执行流程-源码剖析 【等小伙伴对 SpringMVC 有一定基础，在 Debug 源码】

3 @RequestMapping

3.1 基本使用

- **@RequestMapping** 注解可以指定控制器/处理器的某个方法的请求的 url, 基本用法前面我们讲过了, **RequestMapping** : 请求映射

```
-@RequestMapping("/login")
public String login(){
    return "ok";
}
```

3.2 @RequestMapping 注解其它使用方式

3.2.1 @RequestMapping 可以修饰方法和类

- 说明: @RequestMapping 注解可以修饰方法, 还可以修饰类 当同时修饰类和方法时, 请求的 url 就是组合 /类请求值/方法请求值 .

2. 应用实例

- 创建 D:\idea_java_projects\springmvc\src\com\hspedu\web\UserHandler.java

```
@Controller
```

```
public class UserHandler {
```

```
/**
```

```
* 老韩解读
```

```
* 1. method=RequestMethod.POST: 表示请求buy 目标方法必须是 post
```

```
* 2. RequestMethod 四个选项 POST, GET, PUT, DELETE
```

```
* 3. SpringMVC 控制器默认支持 GET 和 POST 两种方式
```

```
* @return
```

```
*/  
  
public String buy() {  
    System.out.println("购买商品");  
    return "success";  
}  
}
```

2) 创建 D:\idea_java_projects\springmvc\web\request.jsp

```
<%--  
  
Created by IntelliJ IDEA.  
  
User: 韩顺平  
  
Version: 1.0  
  
Filename: req_mapping  
  
--%>  
  
<%@ page contentType="text/html;charset=UTF-8" language="java" %>  
  
<html>  
  
<head>  
  
    <title>购买商品</title>  
  
</head>
```

```
<body>  
<h1>购买商品</h1>  
<form action "?" method "?">  
    购买人:<input type="text" name="username"><br>  
    够买量:<input type="text" name="nums"><br>  
    <input type="submit" value="购买">  
</form>  
</body>  
</html>
```

3) 创建 D:\java_projects\springmvc\web\WEB-INF\pages\success.jsp

```
<%--  
Created by IntelliJ IDEA.  
User: 韩顺平  
Version: 1.0  
Filename: success  
--%>  
<%@ page contentType="text/html;charset=UTF-8" language="java" %>  
<html>
```

```
<head>
    <title>操作成功</title>
</head>
<body>
    <h1>恭喜，操作成功~</h1>
</body>
</html>
```

4) 完成测试(页面方式)

购买商品

购买人:
够买量:

恭喜，操作成功~

5) 完成测试(Postman 方式)

POST http://localhost:8080/user/buy

Params Auth Headers (9) Body Pre-req. Tests Settings

Body Cookies (1) Headers (5) Test Results

Pretty Raw Preview Visualize

恭喜，操作成功~

3.2.2 @RequestMapping 可以指定请求方式

1. 说明: @RequestMapping 还可以指定请求的方式(post/get/put/delete..), 请求的方式需要和指定的一样, 否则报错
2. SpringMVC 控制器默认支持 GET 和 POST 两种方式, 也就是你不指定 method , 可以接收 GET 和 POST 请求
3. 应用实例

```
@RequestMapping(value="/buy",method=RequestMethod.POST)
public String buy(){
    System.out.println("buy,post方法");
    return "loginOk";
}
```

4. 当明确了 method , 则需要按指定方式请求, 否则会报错, 比如

HTTP Status 405 - Request method 'GET' not supported

type Status report

message Request method 'GET' not supported

description The specified HTTP method is not allowed for the requested resource.

Apache Tomcat/8.0.50

3.2.3 @RequestMapping 可指定 params 和 headers 支持简单表达式

1. **param1:** 表示请求必须包含名为 param1 的请求参数
2. **!=param1:** 表示请求不能包含名为 param1 的请求参数
3. **param1 != value1:** 表示请求包含名为 param1 的请求参数, 但其值不能为 value1
4. **{"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

5. 应用实例

1) 修改 UserHandler.java , 增加方法

```
/**  
 * params="bookId" 表示请求该目标方法时, 必须给一个 bookId 参数  
 */
```

```
* @return  
*/  
@RequestMapping(value = "/find", params = "bookId", method = RequestMethod.GET)  
public String search(String bookId) {  
    System.out.println("查询书籍 bookId= " + bookId);  
    return "success";  
}
```

2) 修改 request.jsp , 增加代码

```
<hr><h1>演示 params 的使用</h1>  
<a href="#">查询书籍</a>
```

3) 页面方式完成测试

4) Postman 方式完成测试

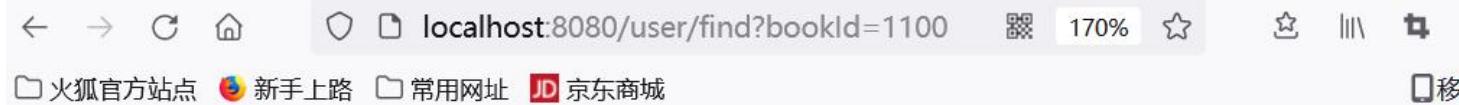
The screenshot shows the Postman interface with a GET request to `http://localhost:8080/user/find?bookId=100`. The 'Body' tab is selected, showing the response body: "恭喜, 操作成功~". Other tabs like 'Headers' and 'Tests' are also visible.

恭喜, 操作成功~

6. 小的细节说明和测试

1) 需要有 bookId 参数，并且值为 100，否则报错

```
@RequestMapping(value = "/find", params = "bookId=100", method = RequestMethod.GET)
```



```
HTTP Status 400 - Parameter conditions  
"bookId=100" not met for actual request  
parameters: bookId={1100}
```

Type Status report

2) 需要有 bookId 参数，并且值不为 100，否则报错

```
@RequestMapping(value = "/find", params = "bookId!=100", method = RequestMethod.GET)
```



HTTP Status 400 - Parameter conditions "bookId!=99" not met for actual request parameters: bookId={99}

type Status report

3.2.4 @RequestMapping 支持 Ant 风格资源地址

1. ?: 匹配文件名中的一个字符

2. *: 匹配文件名中的任意字符

3. **: 匹配多层路径

4. Ant 风格的 url 地址举例

/user/*/createUser: 匹配 /user/aaa/createUser、/user/bbb/createUser 等 URL

/user/**/createUser: 匹配 /user/createUser、/user/aaa/bbb/createUser 等 URL

/user/createUser???: 匹配 /user/createUseraa、/user/createUserbb 等 URL

5. 应用实例

1) 修改 UserHandler.java, 增加方法

```
/**  
 *  
 * 要求可以配置 /user/message/aa, /user/message/aa/bb/cc  
 */  
  
@RequestMapping  
public String im() {  
    System.out.println("发送消息");  
    return "success";  
}
```

2) request.jsp, 增加代码

```
<hr><h1>演示 Ant 风格的请求资源方式 </h1>  
<a href=?>发送消息 1</a><br>  
<a href=?>发送消息 2</a><br>
```

3) 页面完成测试

4) Postman 完成测试

The screenshot shows the Postman interface with a GET request to `http://localhost:8080/user/message/aa`. The 'Body' tab is selected, showing the response body: "恭喜，操作成功~". Other tabs like 'Params', 'Auth', 'Headers (8)', 'Cookies (1)', 'Headers (5)', 'Test Results', 'Pretty', 'Raw', and 'Visualize' are also visible.

恭喜，操作成功~

3.2.5 `@RequestMapping` 可配合 `@PathVariable` 映射 URL 绑定的占位符

1. `@RequestMapping` 还可以配合 `@PathVariable` 映射 URL 绑定的占位符。
2. 这样就不需要在 url 地址上带参数名了，更加的简洁明了

比如：我们的前端页面是 这样的，`kristina` 和 `300` 是参数值

`<hr><h1>占位符的演示</h1>`

`占位符的演示`

3. 应用实例

- 1) 修改 `UserHandler.java`, 增加方法, 注意 `@PathVariable("username")` 不能少.

```
// 我们希望目标方法获取到 username 和 userid, value="/xx/{username}" -
```

```
@PathVariable("username")..  
  
public String register(){  
    System.out.println("接收到参数--" + "username= " + name + "--" + "usreid= " + id);  
    return "success";  
}
```

2) 修改 request.jsp, 增加代码

```
<hr><h1>占位符的演示</h1>  
<a href="user/reg/kristina/300">占位符的演示</a>
```

3) 页面完成测试

4) Postman 完成测试

The screenshot shows a REST client interface with the following details:

- Method: GET
- URL: http://localhost:8080/user/reg/kristina/900
- Headers (8): This tab is selected.
- Body: This tab is also present but not selected.
- Params, Auth, Pre-req., Tests, Settings: These tabs are visible at the top but not selected.
- Body Options: Pretty, Raw, Preview, Visualize. The "Pretty" option is selected.

恭喜，操作成功~

3.2.6 注意事项和使用细节

1. 映射的 URL, 不能重复

修改: D:\hsedu_ssm_temp\springmvc\src\com\hsedu\web\UserHandler.java, 增加方法

```
@RequestMapping(value = "/hi")
public String hi() {
    System.out.println("hi");
    return "success";
}
```

```
@RequestMapping(value = "/hi")
public String hi2() {
    System.out.println("hi");
    return "success";
}
```

```
}
```

服务端报错信息：to { [/user/hi]}: There is already 'userHandler' bean method

2. 各种请求的简写形式

1) 说明

@RequestMapping(value = "/buy",method = RequestMethod.POST) 等价

@PostMapping(value = "/buy")

简写方式一览： @GetMapping @PostMapping @PutMapping @DeleteMapping

2) 应用实例

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

```
@PostMapping(value = "/buy")
```

```
public String buy() {
```

```
    System.out.println("postmapping.. 购买商品");
```

```
    return "success";
```

```
}
```

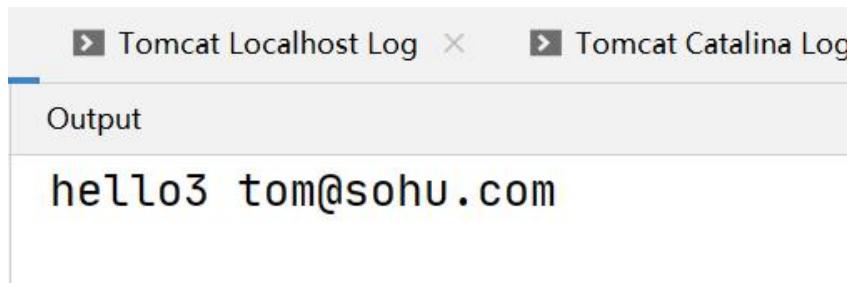
3. 如果我们确定表单或者超链接会提交某个字段数据比如(email), 要求提交的参数名和目标方法的参数名保持一致.

1) 应用实例 , 修 改

D:\hspepu_ssm_temp\springmvc\src\com\hspepu\web\UserHandler.java , 增加方法

```
@GetMapping(value = "/hello3")  
public String hello3(String email) {  
    System.out.println("hello3 " + email);  
    return "success";  
}
```

2) 测试 输入 localhost:9998/user/hello3?email=tom@sohu.com, 一定要注入提交参数名和后台方法的形参名保持一致, 否则后端接收不到参数



3.2.7 课后作业

1. 将老师画的 SpringMVC 执行流程图画 1 遍[不看老韩画的, 凭自己印象]
2. 把老师讲过的@RequestMapping 注解的使用方式, 自己代码走一遍
3. 编写一个表单, 以 Post 提交 Computer 信息, 后端编写 ComputerHandler, 可以接收到信息

电脑信息

品牌:

价格:

数量:

提交信息--brand=huawei-price=100-nums=200

警告 [http-apr-8080-]

3.3 韩顺平 Postman(接口测试工具)

3.3.1 Postman 介绍

3.3.1.1 Postman 是什么

1. Postman 是一款功能超级强大的用于发送 HTTP 请求的 测试工具
2. 做 WEB 页面开发和测试的人员常用工具
3. 创建和发送任何的 HTTP 请求(Get/Post/Put/Delete...)



3.3.1.2 Postman 相关资源

3.3.1.2.1 官方网站 <https://www.postman.com/>



The screenshot shows the official Postman website homepage. At the top, there is a navigation bar with links for Product, Pricing, Enterprise, Explore, and Learning Center. Below the navigation bar, the main title "The Collaboration Platform for API Development" is displayed in large, bold, white font. A subtitle below it reads: "Simplify each step of building an API and streamline collaboration so you can create better APIs—faster." At the bottom of the main content area, there is a "Learn More" button in red text. The overall design is clean and modern, with a dark background for the main content area.

3.3.1.2.2 文档 <https://learning.postman.com/docs/getting-started/introduction/>

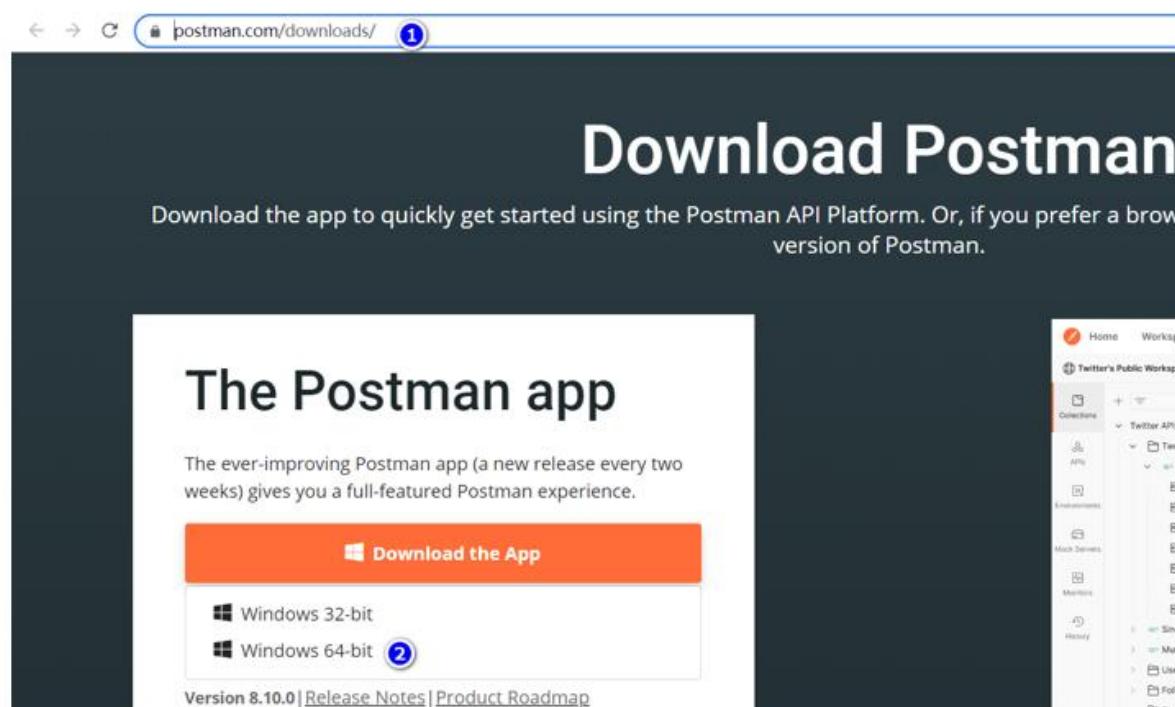
3.3.1.3 Postman 安装

3.3.1.3.1 下载地址 <https://www.postman.com/downloads/>

3.3.1.3.2 具体安装步骤

- 下载 Postman 软件

地址: <https://www.postman.com/downloads/>



● 安装

1. 双击即可安装(非常简单), Postman 不会让你选择安装路径, 会直接安装, 一般安装在系统盘.

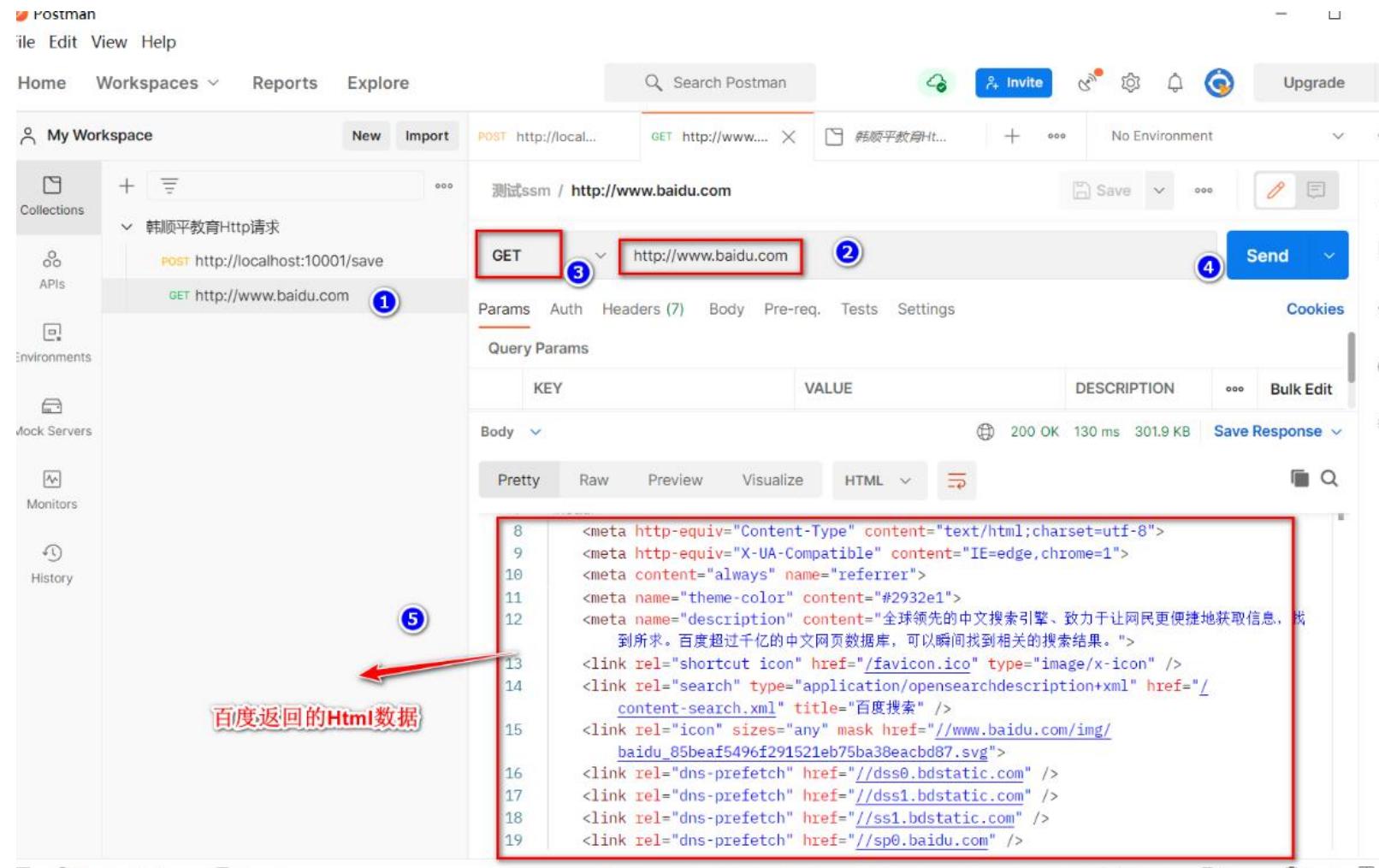
2. 安装成功，在桌面上有快捷图标.



3.3.2 Postman 快速入门

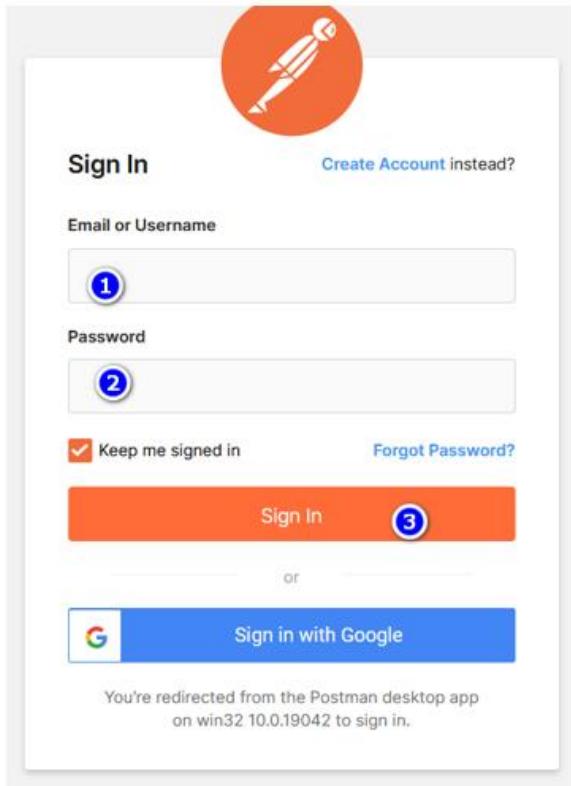
3.3.2.1 快速入门需求说明

- 要求：使用 Postman 向 <http://www.baidu.com> 发出 get 请求，得到返回的 html 格式数据
- 要求：使用 Postman 向 <http://www.sohu.com> 发出 get 请求，得到返回的 html 格式数据



3.3.2.2 快速入门-实现步骤

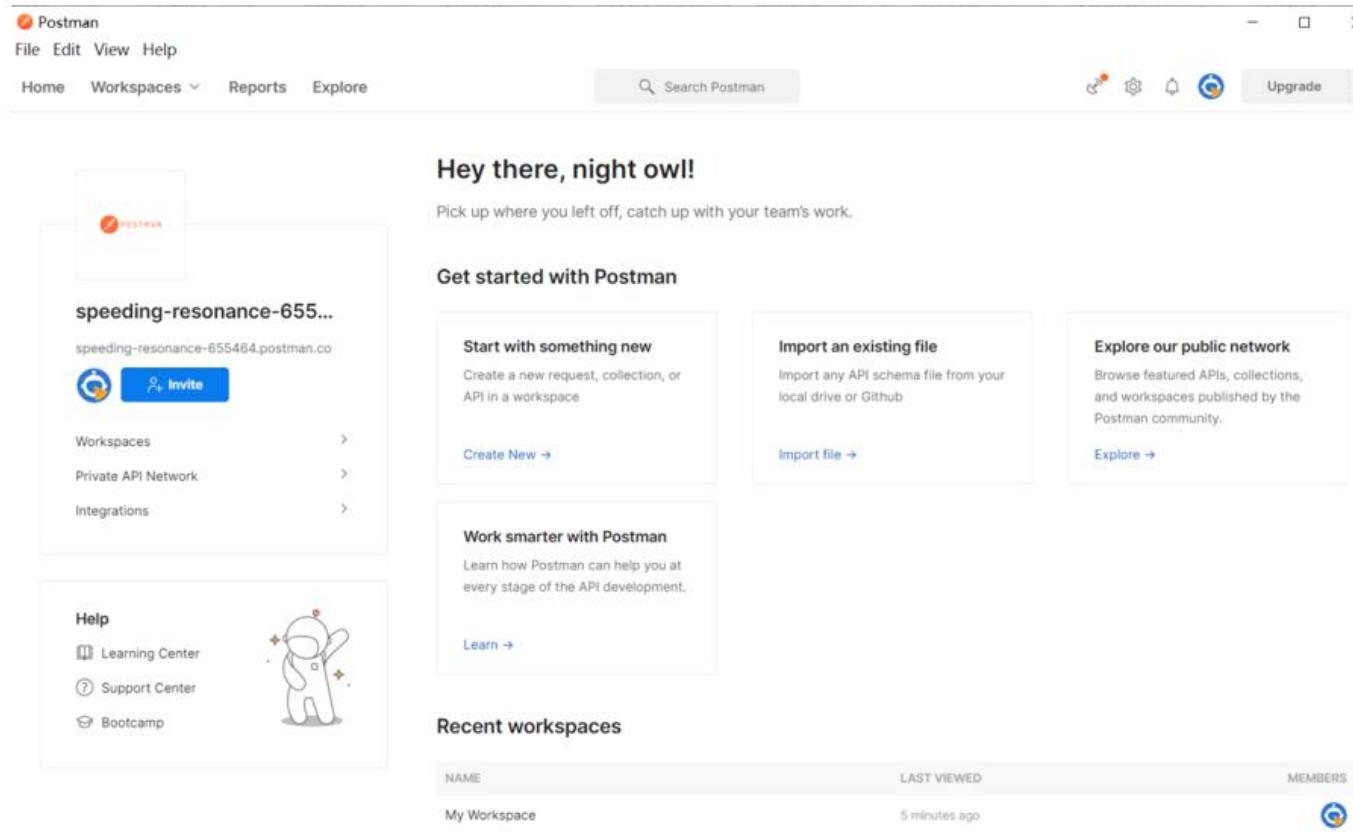
1. 先注册 Postman 一个账号：这个比较简单，输入邮箱，添加账号名和密码即可
2. 登录 Postman



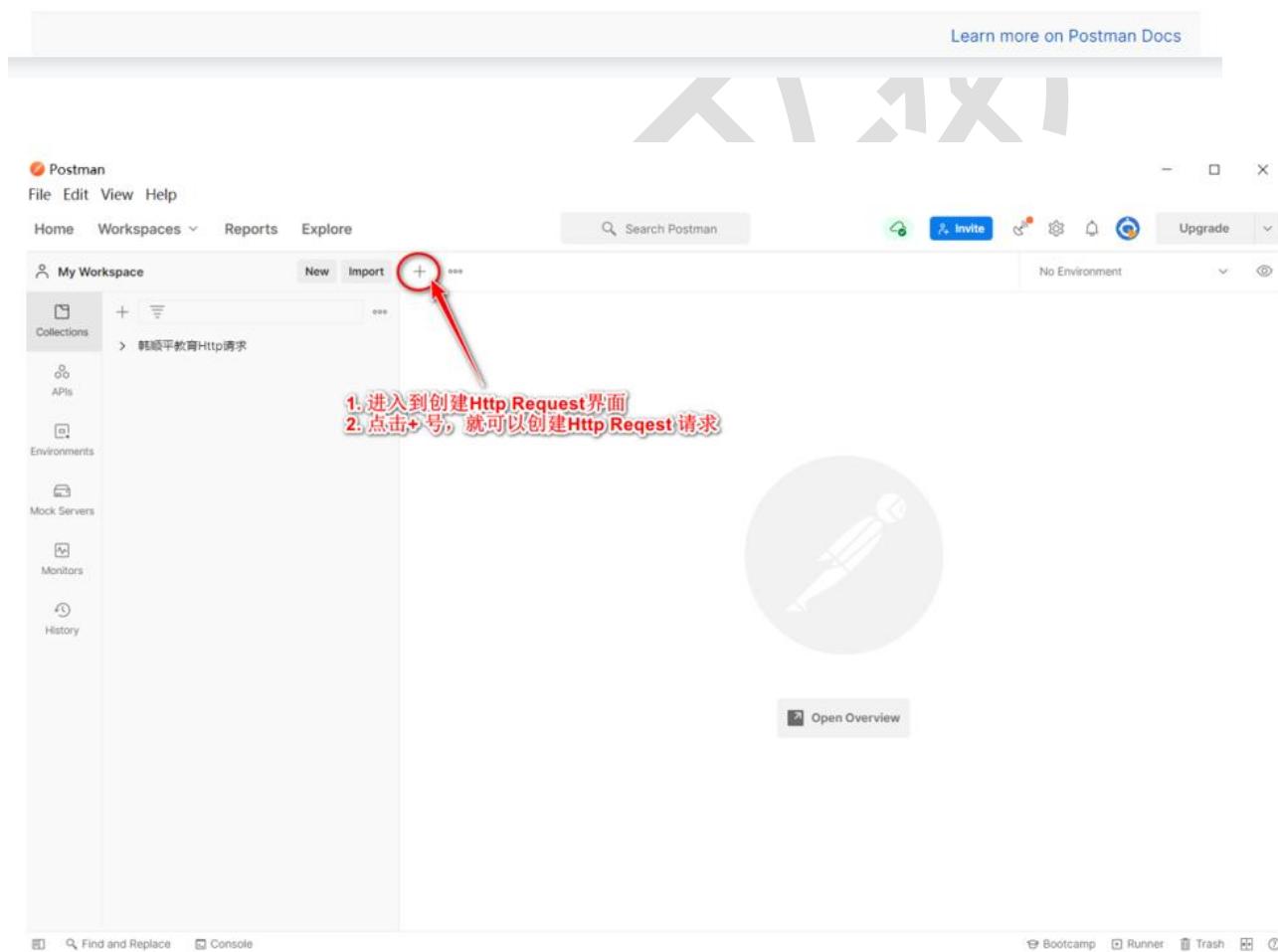
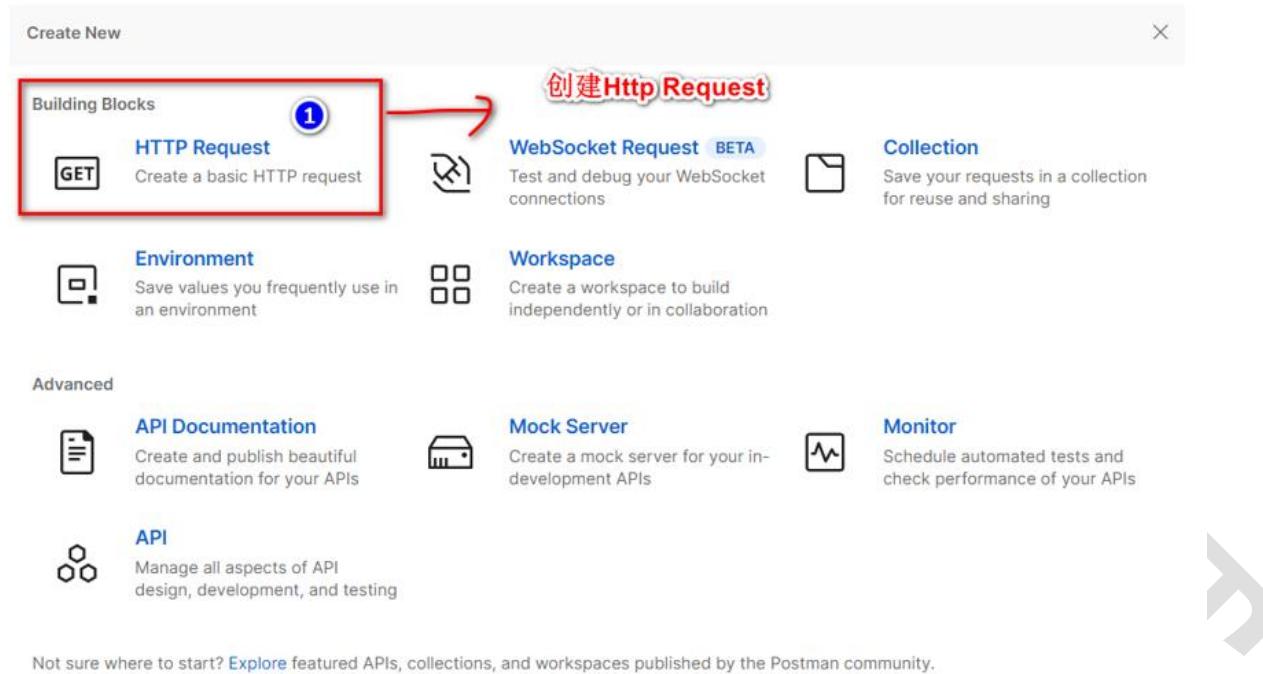
3. 用户名密码没有问题，网站提示如下信息

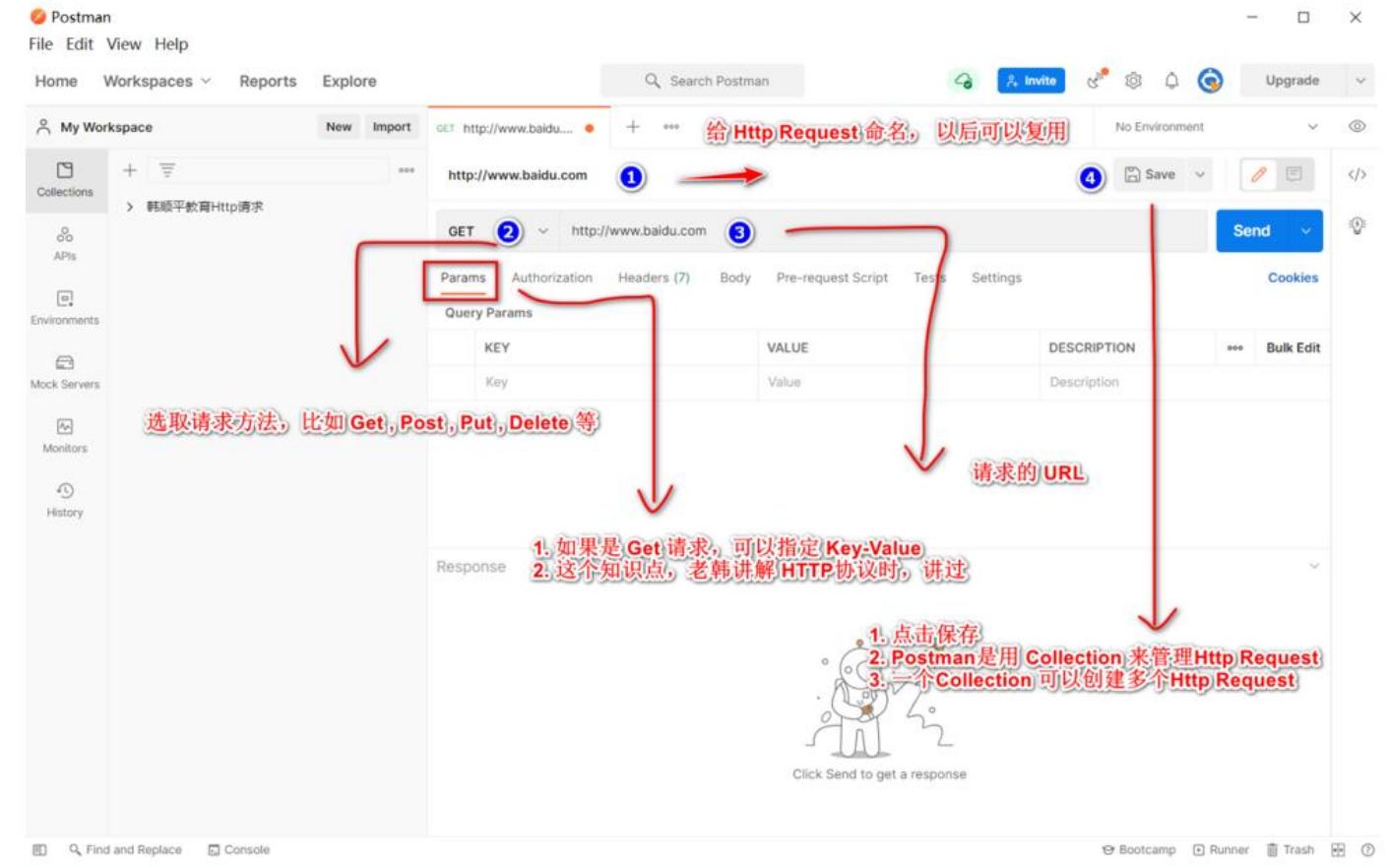


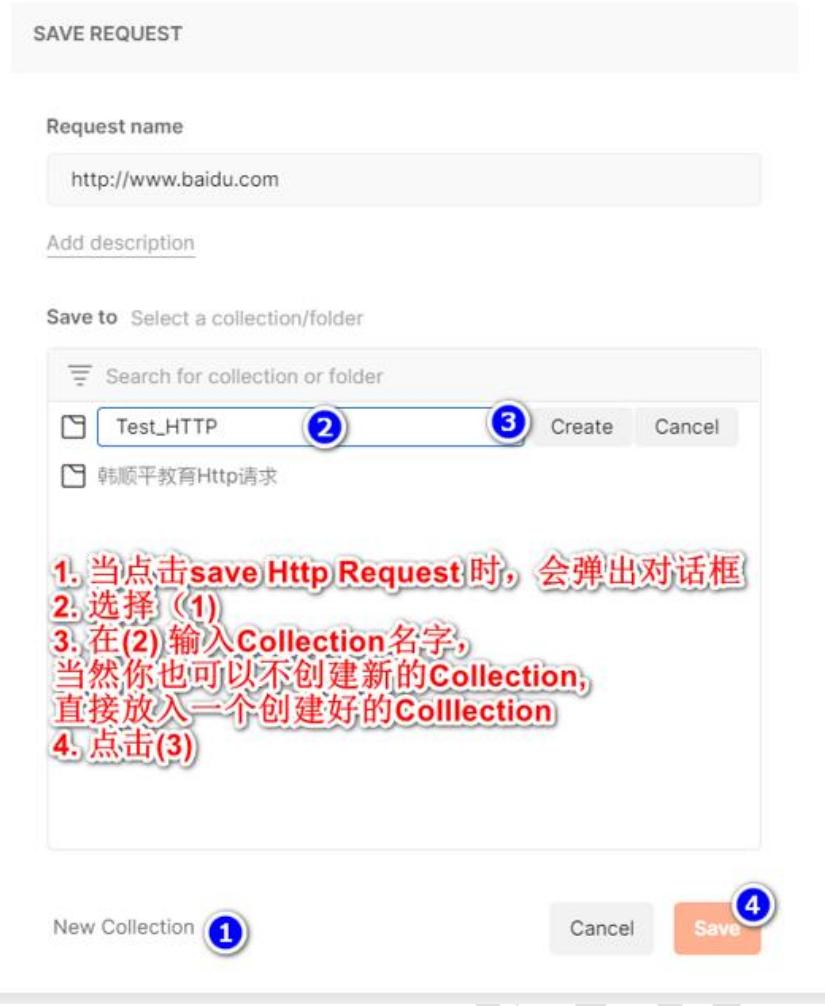
4. 进入 Postman



5. 创建 Http Request , 如果你已经创建过，会直接进入 Workspace, 这里灵活处理即可







The screenshot shows the Postman application interface. On the left, the sidebar includes 'My Workspace' (Collections, APIs, Environments, Mock Servers, Monitors, History), 'File Edit View Help', and tabs for 'Home', 'Workspaces', 'Reports', and 'Explore'. The main workspace shows a collection named 'Test_HTTP / http://www.baidu.com' containing a single GET request to 'http://www.baidu.com'. The request has been saved (1). The 'Params' tab is selected (3). A 'Send' button is highlighted with a blue circle (4). The response body (5) displays the HTML code of a Baidu search result page.

1. 当将一个Http Request 保存到Collection 就可以看到(1)
2. 这时，就可以点击send，如果配置没有问题，就会看到(5)返回结果

6. 到此，我们就完成快速入门案例，后面可以创建新的 Collection 或者 新的 Http Request



3.3.3 Postman 完成 Controller 层测试

3.3.3.1 测试使用实例

说明：使用 Postman ,完成对前面编写的 UserHandler 方法的请求

1. 完成请求

`@RequestMapping(value = "/user")`

`@Controller //UserHandler 就是一个处理器/控制器,注入到容器`

`public class UserHandler {`

```
@PostMapping(value = "/buy")  
public String buy() {  
    System.out.println("购买商品~");  
    return "success";  
}  
}
```

--使用 Postman 测试 Controller 的方法的步骤

- 1) 确定请求的地址 url `http://localhost:8080/springmvc/user/buy`
- 2) 请求的方式 -Post
- 3) 确定请求的参数/数据 - 无
- 4) 确定 Headers 有没有特殊的指定 - 对 http 协议有了解

2. 完成请求

```
@RequestMapping(value = "/user")
```

`@Controller //UserHandler` 就是一个处理器/控制器,注入到容器

```
@RequestMapping(value     = "/find",    params     = "bookId=100",    method     =  
RequestMethod.GET)  
public String search(String bookId) {  
    System.out.println("查询书籍 bookId= " + bookId);
```

```
        return "success";  
    }  
  
}
```

1) 确定请求的地址 url `http://localhost:8080/springmvc/user/find`

2) 请求的方式 -Get

3) 确定请求的参数/数据 - `bookId=100`

4) 确定 Headers 有没有特殊的指定 - 无

3. 完成请求

```
@RequestMapping(value = "/user")
```

`@Controller //UserHandler` 就是一个处理器/控制器,注入到容器

```
@RequestMapping(value = "/message/**")
```

```
public String im() {
```

```
    System.out.println("发送消息");
```

```
    return "success";
```

```
}
```

```
}
```

- 1) 确定请求的地址 url `http://localhost:8080/springmvc/user/message/aa/bb/cc`
- 2) 请求的方式 -Get/Post
- 3) 确定请求的参数/数据 - 无
- 4) 确定 Headers 有没有特殊的指定 - 无

4. 完成请求

```
@RequestMapping(value = "/user")  
@Controller //UserHandler 就是一个处理器/控制器,注入到容器  
  
@RequestMapping(value = "/reg/{username}/{userid}")  
public String register(@PathVariable("username") String name,  
                      @PathVariable("userid") String id) {  
    System.out.println("接收到参数--" + "username= " + name + "--" + "userid= " + id);  
    return "success";  
}
```

- 1) 确定请求的地址 url `http://localhost:8080/springmvc/user/reg/hsp/1000`
- 2) 请求的方式 -Get/Post
- 3) 确定请求的参数/数据 - 无

4) 确定 Headers 有没有特殊的指定 - 无

5. 完成请求

```
@RequestMapping(value = "/user")
```

```
@Controller //UserHandler 就是一个处理器/控制器,注入到容器
```

```
@GetMapping(value = "/hello3")
```

```
public String hello3(String email) {
```

```
    System.out.println("hello3 " + email);
```

```
    return "success";
```

```
}
```

1) 确定请求的地址 url `http://localhost:8080/springmvc/user/hello3`

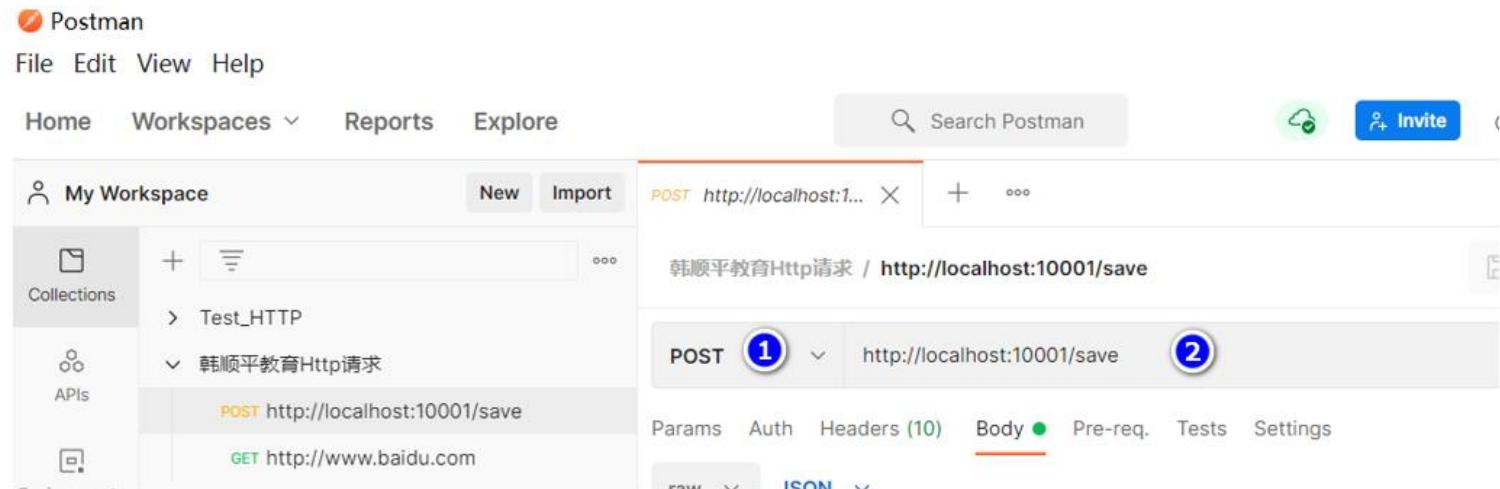
2) 请求的方式 -Get

3) 确定请求的参数/数据 - `email=xx@sohu.com`

4) 确定 Headers 有没有特殊的指定 - 无

3.3.3.2 其它说明

1. 创建 对应的 `Http Request` , 放到新的 `Collection`



2. 在 Headers 选项页，增加 Content-Type application/json

The screenshot shows the Postman Headers tab with the following details:

- Header Card:** POST http://localhost:10001/save
- Header Options:** Params, Authorization, Headers (9) ①, Body (selected), Pre-request Script, Tests, Settings, Cookies.
- Table:** Headers (9)

KEY	VALUE	DESCRIPTION	Bulk Edit	Presets
<input checked="" type="checkbox"/> Postman-Token	<calculated when request is sent>			
<input type="checkbox"/> Content-Type	application/json			
<input checked="" type="checkbox"/> Content-Length	<calculated when request is sent>			
<input checked="" type="checkbox"/> Host	<calculated when request is sent>			
<input checked="" type="checkbox"/> User-Agent	PostmanRuntime/2.2.10			
<input checked="" type="checkbox"/> Accept	/*			
<input checked="" type="checkbox"/> Accept-Encoding	gzip, deflate, br			
<input checked="" type="checkbox"/> Connection	keep-alive			
<input checked="" type="checkbox"/> Content-Type	application/json			

Annotations:

- ①: Points to the 'Headers (9)' tab.
- ②: Points to the 'Content-Type' row in the table.
- Red text annotation: 1. Content-Type 设置为 application/json
2. 如果没有，自己添加即可

3. 因为是 Post 请求，在 Body 选项页填写 Json 数据/Furn 数据



4. 点击 Send ,如果成功，会看到返回 success 的信息，查看 Mysql ,会看到增加新的记录

The screenshot shows a POST request to `http://localhost:10001/save`. The request body is a JSON object:

```
1 {  
2   "name": "简洁沙发",  
3   "maker": "猫猫家居",  
4   "price": 1100,  
5   "sales": 110,  
6   "stock": 210  
7 }
```

The response status is 200 OK, with a response time of 588 ms and a response size of 197 B. The response body is:

```
1 {  
2   "code": 200,  
3   "msg": "success",  
4   "extend": {}  
5 }
```

A red arrow points from the number 2 in the response body to the text "以json, 返回表示成功信息".

Below the Postman interface, there is a screenshot of a MySQL database query results window. The query is:

```
27 SELECT * FROM furn;
```

The results show a table of furniture items:

	id	name	maker	price	sales	st
1	1	北欧风格小桌子	熊猫家居	180.00	666	
2	2	简约风格小椅子	熊猫家居	180.00	666	
3	3	典雅风格小台灯	蚂蚁家居	180.00	666	
4	4	北欧盆景	顺平家居	90.00	666	
5	5	北欧风格沙发	顺平家居	180.00	666	
6	6	北欧风格沙发~	顺平家居~	180.00	666	
7	7	jack	jack	100.00	10	
8	8	简洁沙发	① 猫猫家居	1100.00	110	

3.3.3.3 课后作业

1. 创建新的 Collection, 比如 `jackCollection`, `jack` 用你自己的名字 `zsCollection`

2. 创建多个 http request, 完成对 UserHandler 的各个方法的请求

```
package com.hspedu.web;
```

```
import org.springframework.stereotype.Controller;
```

```
import org.springframework.web.bind.annotation.*;
```

```
/**
```

```
* @author 韩顺平
```

```
* @version 1.0
```

```
*/
```

```
@RequestMapping(value = "/user")
```

```
@Controller //UserHandler 就是一个处理器/控制器,注入到容器
```

```
public class UserHandler {
```

```
    @PostMapping(value = "/buy")
```

```
    public String buy() {
```

```
        System.out.println("购买商品~");
```

```
        return "success";
```

```
}
```

```
/**
```

```
* 老韩解读
```

* 1. `params="bookId"` 表示请求该目标方法时，必须给一个 `bookId` 参数，值没有限
定

* 2. `search(String bookId)`: 表示请求目标方法时，携带的 `bookId=100`, 就会将请
求携带的 `bookId` 对应的

* 值 `100`, 赋给 `String bookId`

* 3. `params = "bookId=100"` 表示必须给一个 `bookId` 参数，而且值必须是 `100`

*

* `@return`

*/

```
@RequestMapping(value = "/find", params = "bookId=100", method =  
RequestMethod.GET)
```

```
public String search(String bookId) {
```

```
    System.out.println("查询书籍 bookId=" + bookId);
```

```
    return "success";
```

```
}
```

```
/**
```

* 要求：可以配置 `/user/message/aa, /user/message/aa/bb/cc`

* 1. `@RequestMapping(value = "/message/**")` /* 可以匹配多层路径

*/

```
@RequestMapping(value = "/message/**")
```

```
public String im() {  
    System.out.println("发送消息");  
    return "success";  
}
```

//要求： 我们希望目标方法获取到 `username` 和 `userid, value="/xx/{username}" - @PathVariable("username")..`

//前端页面: 占位符的演示

//(`value = "/reg/{username}/{userid}"`): 表示 kristina->{username} 300=>{userid}

`@RequestMapping(value = "/reg/{username}/{userid}")`

`public String register(@PathVariable("username") String name,`

`@PathVariable("userid") String id) {`

`System.out.println("接收到参数--" + "username= " + name + "--" + "userid= " + id);`

`return "success";`

`}`

`/**`

* `hello3(String email)`: 如果我们的请求参数有 `email=xx`, 就会将传递的值, 赋给 `String email`

* , 要求名称保持一致, 如果不一致, 那么接收不到数据, 而是 `null`

* `@param email`

```
* @return  
*/  
  
@GetMapping(value = "/hello3")  
public String hello3(String email) {  
    System.out.println("hello3 " + email);  
    return "success";  
}  
}
```

3. 请小伙伴们一定要练习，这样才能有实际的提高

3.3.4 Postman 的其它使用，在讲解框架和项目时，再具体演示

4 Rest-优雅的 url 请求风格

4.1 Rest-基本介绍

- 说明

1. REST：即 Representational State Transfer。**(资源)表现层状态转化。是目前流行的请求方式。**它结构清晰，很多网站采用

2. 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：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. 实例

传统的请求方法：

getBook?id=1 GET

delete?id=1 GET

update POST

add POST

4. 说明：传统的 url 是通过参数来说明 crud 的类型，rest 是通过 get/post/put/delete 来说明 crud 的类型

• REST 的核心过滤器

1. 当前的浏览器只支持 post/get 请求，因此为了得到 put/delete 的请求方式需要使用 Spring 提供的 HiddenHttpMethodFilter 过滤器进行转换。
2. HiddenHttpMethodFilter：浏览器 form 表单只支持 GET 与 POST 请求，而 DELETE、PUT 等 method 并不支持，Spring 添加了一个过滤器，可以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
3. HiddenHttpMethodFilter 能对 post 请求方式进行转换，因此我们需要特别的注意这一点
4. 这个过滤器需要在 web.xml 中配置

4.2 Rest 风格的 url-完成增删改查

4.2.1 需求说明



4.2.2 Rest 应用案例-代码实现

1. 修改 web.xml 添加 HiddenHttpMethodFilter

<!--

老韩解读

1. 配置 *HiddenMethodFilter* 过滤器,

2. 可以将以 post 方式提交的 delete 和 put 请求进行转换

-->

<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>

2. 修改 springDispatcherServlet-servlet.xml

<!-- 加入两个常规配置 -->

<!-- 能支持 SpringMVC 高级功能，比如 JSR303 校验，映射动态请求 -->

<mvc:annotation-driven></mvc:annotation-driven>

<!-- 将 SpringMVC 不能处理的请求交给 Tomcat，比如请求 css,js 等-->

<mvc:default-servlet-handler/>

3. 创建 D:\idea_java_projects\springmvc\web\rest.jsp, 注意需要引入 jquery, 测试的时候查询/添加/删除/修改一个一个的来

```
<%--
```

Created by IntelliJ IDEA.

User: 韩顺平

Version: 1.0

Filename: rest

```
--%>
```

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>rest </title>
    <script type="text/javascript" src="script/jquery-3.6.0.min.js"></script>
    <script type="text/javascript">
        $(function () {
            $("#deleteBook").click(function () {
                alert("ok");
                var href = this.href;
                $("#hiddenForm").attr("action", href);
                $(":hidden").val("DELETE");
                $("#hiddenForm").submit();//这里就是提交删除请求了
                //这里必须返回false,否则会提交两次
                return false;
            });
        });
    </script>
</head>
<body>
    <div>
        <button id="deleteBook">Delete</button>
        <form id="hiddenForm" style="display: none;">
            <input type="hidden" name="method" value="DELETE"/>
        </form>
    </div>
</body>
</html>
```

```
        })  
  
    </script>  
  
</head>  
  
<body>  
  
<h3>Rest 风格的 crud 操作案例</h3>  
  
<br><hr>  
  
<h3>rest 风格的 url 查询书籍[get]</h3>  
  
<a href="user/book/100">点击查询书籍</a>  
  
<br><hr>  
  
<h3>rest 风格的 url 添加书籍[post]</h3>  
  
<form action="user/book" method="post">  
  
    name:<input name="bookName" type="text"><br>  
  
    <input type="submit" value="添加书籍">  
  
</form>  
  
<br><hr>  
  
<h3>rest 风格的 url, 删除一本书</h3>  
  
<!--  
  
    1. 这里我们需要将删除方式 (get) 转成 delete 的方式, 需要使用过滤器和jquery 来完成  
    2. name="_method" 名字需要写成 _method 因为后台的 HiddenHttpMethodFilter  
        就是按这个名字来获取 hidden 域的值, 从而进行请求转换的.  
  
-->  
  
<a href="user/book/100" id="deleteBook">删除指定 id 的书</a>
```

```
<form action="" method="POST" id="hiddenForm">  
    <input type="hidden" name="_method"/>  
</form>  
  
<br><hr>  
  
<h3>rest 风格的 url 修改书籍[put]~</h3>  
  
<form action="user/book/100" method="post">  
    <input type="hidden" name="_method" value="PUT">  
    <input type="submit" value="修改书籍~">  
</form>  
  
</body>  
</html>
```

4. 创建 D:\idea_java_projects\springmvc\src\com\hspedu\web\rest\BookHandler.java , 处理 rest 风格的请求

```
@RequestMapping(value = "/user")  
@Controller  
public class BookHandler {  
    //查询[GET]  
    public String getBook(@PathVariable("id") String id) {  
        System.out.println("查询书籍 id=" + id );  
    }  
}
```

```
        return "success";  
    }  
  
    //添加[POST]  
    public String addBook(String bookName) {  
        System.out.println("添加书籍 bookName== " + bookName);  
        return "success";  
    }  
  
    //删除[DELETE]  
    public String delBook(@PathVariable("id") String id) {  
        System.out.println("删除书籍 id= " + id);  
        //return "success"; [如果这样返回会报错 JSPs only permit GET POST or  
        HEAD]  
        return "redirect:/user/success"; //重定向到一个没有指定 method 的 Handler 方  
法  
    }  
  
    //修改[PUT]  
    public String updateBook(@PathVariable("id") String id) {  
        System.out.println("修改书籍 id=" + id);  
        return "redirect:/user/success"; //重定向到一个没有指定 method 的 Handler 方
```

法

}

```
@RequestMapping(value = "/success")
public String successGenecal() {
    return "success"; //由该方法 转发到success.jsp 页面
}
```

5. 页面完成测试

4.2.3 注意事项和细节说明

1、**HiddenHttpMethodFilter**，在将 post 转成 delete / put 请求时，是按`_method`参数名 来读取的

2、如果 web 项目是运行在 Tomcat 8 及以上，会发现被过滤成 DELETE 和 PUT 请求，到达控制器时能顺利执行，但是返回时（forward）会报 HTTP 405 的错误提示：消息 JSP 只允

许 GET、POST 或 HEAD。

1) 解决方式 1：使用 Tomcat7

2) 解决方式 2：将请求转发（forward）改为请求重定向（redirect）：重定向到一个 Handler，由 Handler 转发到页面

```
//修改[PUT]
@RequestMapping(value = "/book/{id}", method = RequestMethod.PUT)
public String updateBook(@PathVariable("id") String id) {
    System.out.println("修改书籍 id=" + id);
    return "redirect:/user/success"; ①重定向到一个没有指定 method 的 Handler 方法
}

@RequestMapping(value = "/success")
public String successGenecal() { ②
    return "success"; //由该方法 转发到success.jsp 页面
}
```

3、页面测试时，如果出现点击修改书籍，仍然走的是删除 url，是因为浏览器原因（缓存等原因），换成 chrome 即可

4.3 课后作业

1. 把老师讲的 rest 风格的 书籍案例自己玩一把
2. 还是那句话，看起来简单，做起来并不轻松，一定要练习

5 SpringMVC 映射请求数据

5.1 获得参数值

5.1.1 说明

1. 开发中，如何获取到 `http://xxx/url?参数名=参数值&参数名=参数值`

2. 这个使用的非常广泛，我们看一个案例

采集信息

点击获取[href传递的参数和值](#)

vote1 --- name ---tom

```
@RequestMapping(value="/vote1")
public String vote01(@RequestParam(value="name", required=false) String name){
    System.out.println("vote01 --- name ---" + name);
    return "loginOk";
}
```

5.1.2 应用实例

1.

创

建

D:\idea_java_projects\springmvc\src\com\hspedu\web\requestparam\VoteHandler.java

```
package com.hspedu.web.requestparam;
```

```
import com.hspedu.web.requestparam.entity.Master;
```

```
import org.springframework.stereotype.Controller;  
import org.springframework.web.bind.annotation.*;  
import org.springframework.web.servlet.ModelAndView;  
  
import javax.servlet.http.HttpServletRequest;  
import javax.servlet.http.HttpServletResponse;  
import javax.servlet.http.HttpSession;  
import java.util.Map;  
  
/**  
 * @author 韩顺平  
 * @version 1.0  
 */  
  
@RequestMapping("/vote")  
@Controller  
public class VoteHandler {  
    //获取到超链接传递的数据  
  
    /**  
     * 老韩解读 @RequestParam(value="name", required=false)  
     * 1.@RequestParam : 表示说明一个接受到的参数
```

- * 2.value="name": 接收的参数名是 name
- * 3.required=false : 表示该参数可以有, 也可以没有 ,如果 required=true, 表示必须传递该参数.

- * 默认是 required=true

- */

```
@RequestMapping(value = "/vote01")  
public String test01(@RequestParam(value = "name", required = false)  
                      String username) {  
  
    System.out.println("得到的 username= " + username);  
  
    //返回到一个结果  
    return "success";  
}  
}
```

2. 创建 D:\idea_java_projects\springmvc\web\request_parameter.jsp

```
<%--
```

Created by IntelliJ IDEA.

User: 韩顺平

Version: 1.0

Filename: request_parameter

```
--%>

<%@ page contentType="text/html;charset=UTF-8" language="java" %>

<html>
<head>
    <title>测试 request parameter</title>
</head>
<body>
<h2>获取到超链接参数值</h2>
<hr/>
<a href="vote/vote01?name=hspedu">获取超链接的参数</a>
</body>
</html>
```

3. 页面完成测试

4. Postman 完成测试



POST http://localhost:8080/vote/vote01?name=hspedu

Params ● Auth Headers (11) Body ● Pre-req. Tests Settings

Query Params

	KEY	VALUE
<input checked="" type="checkbox"/>	name	hspedu
	Key	Value

Body Cookies (1) Headers (5) Test Results

Pretty Raw Preview Visualize

恭喜，操作成功~

5.2 获取 http 请求消息头

- 说明：
 1. 开发中，如何获取到 http 请求的消息头信息
 2. 使用较少

- 应用实例
 1. 修改 VoteHandler.java，增加方法

```
/**  
 * 获取 http 请求头信息  
 * @param ae  
 * @param host  
 * @return  
 */  
  
@RequestMapping(value = "/vote02")  
public String test02(@RequestHeader("Accept-Encoding") String ae,  
                      @RequestHeader("Host") String host) {  
  
    System.out.println("Accept-Encoding =" + ae);  
    System.out.println("Host =" + host);  
  
    //返回到一个结果  
    return "success";  
}
```

2. 修改 request_parameter.jsp, 增加代码

```
<h2>获取到超链接参数值</h2>  
<hr/>  
<a href="vote/vote01?name=hspedu">获取超链接的参数</a>
```

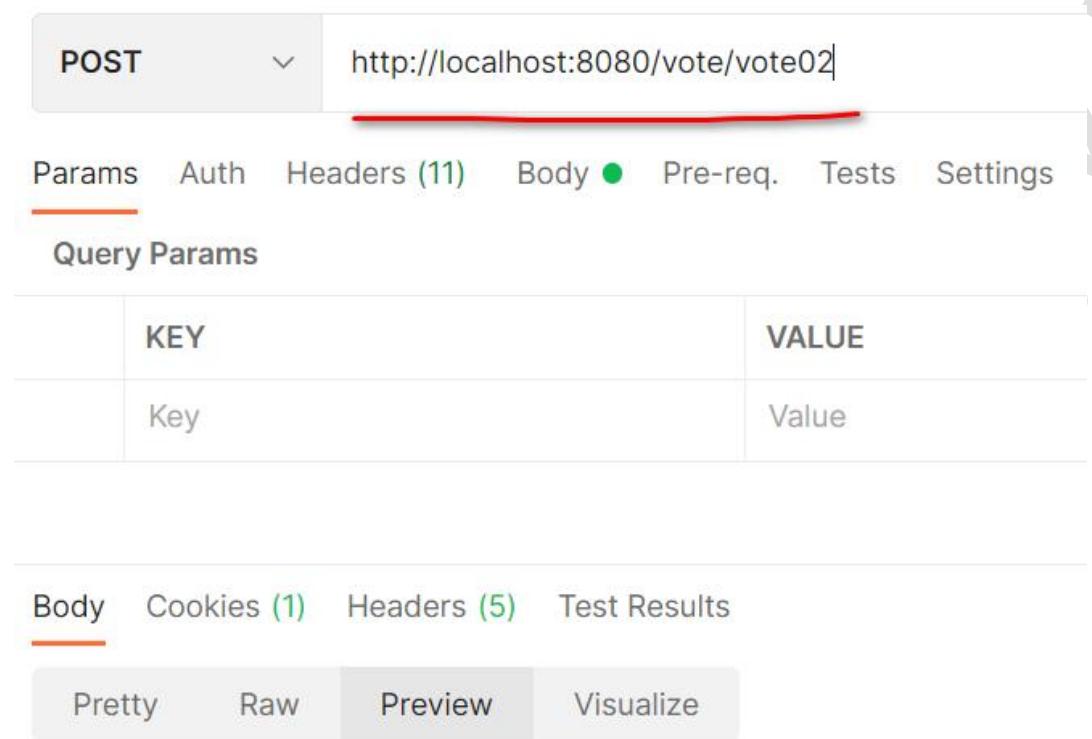
<h1>获取到消息头</h1>

<hr>

获取 http 消息头信息

3. 页面测试

4. Postman 测试



POST http://localhost:8080/vote/vote02

Headers (11)

KEY	VALUE
Key	Value

Body Cookies (1) Headers (5) Test Results

Pretty Raw Preview Visualize

恭喜，操作成功~

Accept-Encoding =gzip, deflate, br

Host =localhost:8080

5.3 获取 javabean 形式的数据

5.3.1 使用场景说明

- 开发中，如何获取到 **javaben** 的数据，就是以前的 **entity/pojo** 对象数据

添加一个主人以bean的数据形式添加....

编号: 300

名字: jerry

宠物: dog

提交

5.3.2 应用实例

1.

创

建

D:\idea_java_projects\springmvc\src\com\hspedu\web\requestparam\entity\Pet.java

```
public class Pet {  
    private Integer id;  
    private String name;  
}
```

2.

创

建

D:\idea_java_projects\springmvc\src\com\hspedu\web\requestparam\entity\Master.java

```
package com.hspedu.web.requestparam.entity;
```

```
/**
```

```
* @author 韩顺平
```

```
* @version 1.0
```

```
*/
```

```
public class Master {
```

```
    private Integer id;
```

```
    private String name;
```

```
    private Pet pet;
```

```
    public Pet getPet() {
```

```
        return pet;
```

```
}
```

```
    public void setPet(Pet pet) {
```

```
        this.pet = pet;
```

```
}
```

```
public Master() {  
}  
  
public Integer getId() {  
    return id;  
}  
  
public void setId(Integer id) {  
    this.id = id;  
}  
  
public String getName() {  
    return name;  
}  
  
public void setName(String name) {  
    this.name = name;  
}  
  
@Override  
public String toString() {  
    return "Master{" +
```

```
"id=" + id +  
", name=\"" + name + "\"" +  
", pet=" + pet +  
'};  
}  
}
```

3. 修改 VoteHandler.java, 增加方法

```
//获取到添加的主人信息  
@RequestMapping(value = "/vote03")  
public String test03(Master master) {  
  
    System.out.println("主人信息= " + master);  
    //返回结果  
    return "success";  
}
```

4. 修改 request_parameter.jsp, 增加演示代码

```
<hr>  
<h1>添加主人信息</h1>
```

```
<!--
```

老韩解读

1. 这里的字段名称和对象的属性名保持一致, 级联添加属性也是一样保持名字对应关系
2. 如果只是添加主人信息, 则去掉宠物号和宠物名输入框即可 ,pet 为 null-->

```
<form action="vote/vote03" method="post">
```

```
    主人号:<input type="text" name="id"><br>
```

```
    主人名:<input type="text" name="name"><br>
```

```
    宠物号:<input type="text" name="pet.id"><br>
```

```
    宠物名:<input type="text" name="pet.name"><br>
```

```
    <input type="submit" value="添加主人和宠物">
```

```
</form>
```

5. 完成测试(页面方式)

6. 完成测试(Postman 方式)

POST ▼ http://localhost:8080/vote/vote03

Params Auth Headers (11) **Body** Pre-req. Tests Settings

x-www-form-urlencoded

KEY	VALUE
<input checked="" type="checkbox"/> id	1000
<input checked="" type="checkbox"/> name	大勇
<input checked="" type="checkbox"/> pet.id	8
<input checked="" type="checkbox"/> pet.name	红黑

Key Value

5.3.3 使用注意事项

- 支持级联数据获取
- 表单的控件名称 name 需要和 javabean 对象字段对应，否则就是 null



5.4 获得 servlet api

5.4.1 应用实例

- 说明

1. 开发中，我们可能需要使用到原生的 servlet api，看看如何获取
2. 使用 servlet api，需要引入 tomcat/lib 下的 `servlet-api.jar`

- 应用实例

1. 修改 `VoteHandler.java`, 增加方法

```
/*
 * 获取servlet api
 * @param request
 * @param response
 * @return
 */
@RequestMapping(value = "/vote04")
public String test04(HttpServletRequest request,
                     HttpServletResponse response) {
```

```
System.out.println("name= " + request.getParameter("username"));

System.out.println("pwd= " + request.getParameter("pwd"));

//返回结果

return "success";

}
```

2. 修改 request_parameter.jsp

```
<hr>

<h1>获取 servlet api </h1>

<form action="vote/vote04" method="post">

用户名:<input type="text" name="username"><br>

密 码:<input type="text" name="pwd"><br>

<input type="submit" value="添加主人和宠物">

</form>
```

3. 完成测试(页面方式)

4. 完成测试(Postman 方式)

The screenshot shows a POST request to `http://localhost:8080/vote/vote04`. The `Body` tab is selected, indicating the use of `x-www-form-urlencoded` encoding. The data is represented as a table:

	Key	Value
<input type="checkbox"/>	bookName	abc
<input checked="" type="checkbox"/>	username	大红
<input checked="" type="checkbox"/>	pwd	11111

Below the table, there are tabs for `Body`, `Cookies (1)`, `Headers (6)`, and `Test Results`. The `Body` tab is underlined, and the `Pretty` button is highlighted.

5.4.2 使用注意事项

1. 除了 `HttpServletRequest`, `HttpServletResponse` 还可以其它对象也可以这样的形式获取
2. `HttpSession`、`java.security.Principal`,`InputStream`,`OutputStream`,`Reader`,`Writer`
3. 其中一些对象也可以通过 `HttpServletRequest / HttpServletResponse` 对象获取，比如 `Session` 对象，既可以 `通过参数传入`，也以 `通过 request.getSession() 获取`，效果一样，推荐使用参数形式传入，更加简单明了
4. 举例说明

```
@RequestMapping(value = "/vote04")  
public String test04(HttpServletRequest request,
```

```
HttpServletResponse response, HttpSession hs) {
```

```
System.out.println("name= " + request.getParameter("username"));
```

```
System.out.println("pwd= " + request.getParameter("pwd"));
```

//可以看到 hs 和 request.getSession() 是同一个对象

```
System.out.println("httpSession=" + httpSession);
```

```
System.out.println("httpSession2=" + request.getSession());
```

//返回结果

```
return "success";
```

```
}
```

6 模型数据

6.1 模型数据处理-数据放入 request

6.1.1 需求分析/图解

- 说明

开发中，控制器/处理器中获取的数据如何放入 request 域，然后在前端(VUE/JSP/...)取出显示

- 应用实例需求

添加主人信息

主人号:

主人名:

宠物号:

宠物名:

获取的数据显示页面

取出 request域的数据

address: 北京

主人名字= jack 主人信息= Master{id=100, name='jack', pet=Pet{id=1, name='happy'}} 宠物名字= happy

6.1.2 方式 1：通过 HttpServletRequest 放入 request 域

1. 创建 D:\idea_java_projects\springmvc\web\model_data.jsp

```
<%--
```

Created by IntelliJ IDEA.

User: 韩顺平

Version: 1.0

Filename: request_parameter

```
--%>

<%@ page contentType="text/html;charset=UTF-8" language="java" %>

<html>
<head>
    <title>测试 模型数据</title>
</head>
<body>
<h1>添加主人信息</h1>
<!--
```

老韩解读

1. 这里的字段名称和对象的属性名保持一致,级联添加属性也是一样保持名字对应关系
2. 如果只是添加主人信息, 则去掉宠物号和宠物名输入框即可 ,pet 为 null-->

```
<form action="vote/vote05" method="post">

    主人号:<input type="text" name="id"><br>
    主人名:<input type="text" name="name"><br>
    宠物号:<input type="text" name="pet.id"><br>
    宠物名:<input type="text" name="pet.name"><br>
    <input type="submit" value="添加主人和宠物">

</form>
</body>
</html>
```

2. 修改 VoteHandler.java, 增加方法

```
/**  
 * 获取模型数据 master, 并放入到request  
 * @param master  
 * @param request  
 * @param response  
 * @return  
 */  
  
@RequestMapping(value = "/vote05")  
public String test05(Master master,  
                     HttpServletRequest request, HttpServletResponse response) {  
  
    //老韩解读  
    //1. springmvc 会自动把获取的 model 模型, 放入到request 域中, 名字就是 master  
    //2. 也可以手动将 master 放入到request  
    //request.setAttribute("master", master);  
    request.setAttribute("address", "北京");  
  
    //返回到一个结果  
    return "vote_ok";  
}
```

3. 创建 D:\idea_java_projects\springmvc\web\WEB-INF\pages\vote_ok.jsp, 显示数据

```
<%--  
Created by IntelliJ IDEA.  
User: 韩顺平  
Version: 1.0  
Filename: vote05  
--%>  
<%@ page contentType="text/html;charset=UTF-8" language="java" %>  
<html>  
<head>  
    <title>vote_ok </title>  
</head>  
<body>  
    <h1>获取的的数据显示页面</h1>  
    <hr>  
    取出 request 域的数据  
    <br>  
    address: ${address }<br>  
    主人名字= ${requestScope.master.name }  
    主人信息= ${requestScope.master }  
    宠物名字= ${requestScope.master.pet.name }
```

```
</body>
```

```
</html>
```

4. 完成测试(页面测试)

5. 完成测试(Postman 测试)

The screenshot shows the Postman interface with the following details:

- Method: POST
- URL: <http://localhost:9998/vote/vote05>
- Body tab is selected, showing "x-www-form-urlencoded" selected.
- Body table:

KEY	VALUE	DES
id	1000	
- Test Results section shows a green icon and "200 OK".
- Bottom navigation: Pretty, Raw, Preview, Visualize.

获取的数据显示页面

取出 request域的数据

address: 北京

主人名字= 大勇 主人信息= Master{id=1000, name='大勇', pet=Pet{id=8, 黑}

6.1.3 方式 2: 通过请求的方法参数 Map<String, Object> 放入 request 域

1. 修改 model_data.jsp, 增加代码

```
<br/><hr/>

<h1>添加主人信息[测试 Map ]</h1>

<form action="vote/vote06" method="post">

    主人号:<input type="text" name="id"><br>
    主人名:<input type="text" name="name"><br>
    宠物号:<input type="text" name="pet.id"><br>
    宠物名:<input type="text" name="pet.name"><br>
    <input type="submit" value="添加主人和宠物">

</form>
```

2. 修改 VoteHandler.java , 增加方法

```
/**

 * 获取到添加的主人信息，并加入到Map<String, Object> 中
 * @param map
 * @param master
 * @return
 */

@RequestMapping(value = "/vote06")
public String test06(Map<String, Object> map, Master master) {
```

```
System.out.println("=====test06()=====~");
```

//老韩解读

//1. springMVC 会将 map 的 k-v 放入到 request 域对象

```
map.put("master", master);
```

```
map.put("address", "beijing");
```

//返回到一个结果

```
return "vote_ok";
```

```
}
```

3. 完成测试(页面测试)

4. 完成测试(Postman 测试)



The screenshot shows a Postman test configuration for a POST request to `http://localhost:9998/vote/vote06`. The 'Body' tab is selected, showing an 'x-www-form-urlencoded' payload with three parameters:

	KEY	VALUE
<input checked="" type="checkbox"/>	id	888
<input checked="" type="checkbox"/>	name	大勇
<input checked="" type="checkbox"/>	pet.id	8

Below the table, the 'Body' tab is highlighted, along with 'Cookies (1)', 'Headers (5)', and 'Test Results'. At the bottom, there are buttons for 'Pretty', 'Raw', 'Preview', and 'Visualize'.

6.1.4 方式 3：通过返回 ModelAndView 对象 实现 request 域数据

1. 修改 model_data.jsp，增加代码

```
<br/><hr/>

<h1>添加主人信息[测试 ModelAndView]</h1>

<form action="vote/vote07" method="post">

    主人号:<input type="text" name="id"><br>
    主人名:<input type="text" name="name"><br>
    宠物号:<input type="text" name="pet.id"><br>
    宠物名:<input type="text" name="pet.name"><br>
    <input type="submit" value="添加主人和宠物">

</form>
```

2. 修改 VoteHandler.java，增加方法

```
/**

 * 老韩解读
 *
 * 1. 将 model 放入到 ModelAndView 对象中，实现将数据放入到 request 域中
 *
 * @param master
 *
 * @return
```

*/

```
@RequestMapping(value = "/vote07")  
public ModelAndView test07(Master master) {  
    System.out.println("=====test07()=====");  
    //创建一个 ModelAndView 对象  
    ModelAndView modelAndView = new ModelAndView();  
    //下面这句话就等价于将 master 对象放入到 request 域中，属性名“master”  
    modelAndView.addObject("master", master);  
    modelAndView.addObject("address", "shanghai");  
    modelAndView.setViewName("vote_ok");  
    //返回结果  
    return modelAndView;  
}
```

3. 完成测试(页面测试)

添加主人信息[测试 ModelAndView]

主人号:	800
主人名:	abc
宠物号:	120
宠物名:	dog

获取的数据显示页面

取出 request域的数据

address: shanghai

主人名字= abc 主人信息= Master{id=800, name='abc', pet=Pet{id=120, name='dog'}} 宠物名字= dog

4. 完成测试(Postman 测试)

The screenshot shows the Postman interface with a successful test result. The URL is `http://localhost:9998/vote/vote07`. The 'Body' tab is selected, showing the following parameters:

	KEY	VALUE
<input checked="" type="checkbox"/>	id	888
<input checked="" type="checkbox"/>	name	大勇
<input checked="" type="checkbox"/>	pet.id	8

Below the table, there are tabs for Body, Cookies (1), Headers (5), and Test Results. The 'Body' tab is highlighted with an orange underline. The status bar at the bottom right shows a globe icon and '20'.

取出 request域的数据

address: shanghai

主人名字= 大勇 主人信息= Master{id=888, name='大勇', pet=Pet{id=8, name='black'}} 宠物名字= black

5. 使用注意事项

- 1) 从本质看，请求响应的方法 `return "xx"`，是返回了一个字符串，其实本质是返回了一个 `ModelAndView` 对象，只是默认被封装起来的。
- 2) `ModelAndView` 即可以包含 `model` 数据，也可以包含视图信息
- 3) `ModelAndView` 对象的 `addObject` 方法可以添加 `key-val` 数据，默认在 `request` 域中
- 4) `ModelAndView` 对象 `setView` 方法可以指定视图名称



6.2 模型数据处理-数据放入 session

6.2.1 需求分析/图解

- 说明

开发中，控制器/处理器中获取的数据如何放入 session 域，然后在前端(VUE/JSP/...)取出显示

- 应用实例需求

添加主人信息

主人号:	100
主人名:	jack
宠物号:	1
宠物名:	happy
<input type="button" value="添加主人和宠物"/>	

获取的数据显示页面

取出 request域的数据
address: guangzhou
主人名字= abc 主人信息= Master{id=333, name='abc', pet=Pet{id=100, name='del'}} 宠物名字= del

取出 session域的数据
主人名字= abc 主人信息= Master{id=333, name='abc', pet=Pet{id=100, name='del'}}

6.2.2 应用实例

1. 修改 model_data.jsp, 增加代码

```
<br/><hr/>  
<h1>添加主人信息[测试 session]</h1>  
<form action="vote/vote08" method="post">
```

```
主人号:<input type="text" name="id"><br>
主人名:<input type="text" name="name"><br>
宠物号:<input type="text" name="pet.id"><br>
宠物名:<input type="text" name="pet.name"><br>
<input type="submit" value="添加主人和宠物">
</form>
```

2. 修改 VoteHandler.java, 增加方法

```
/*
 * 将model(master 对象) 放入到session 域中
 * @param map
 * @param master
 * @param session
 * @return
 */
@RequestMapping(value = "/vote08")
public String test08(Map<String, Object> map, Master master,
HttpSession session) {
    System.out.println("=====test08()=====");
    map.put("address", "guangzhou");
    //在 session 域也放入 master 对象
}
```

```
        session.setAttribute("master2", master);

        return "vote_ok";

    }
```

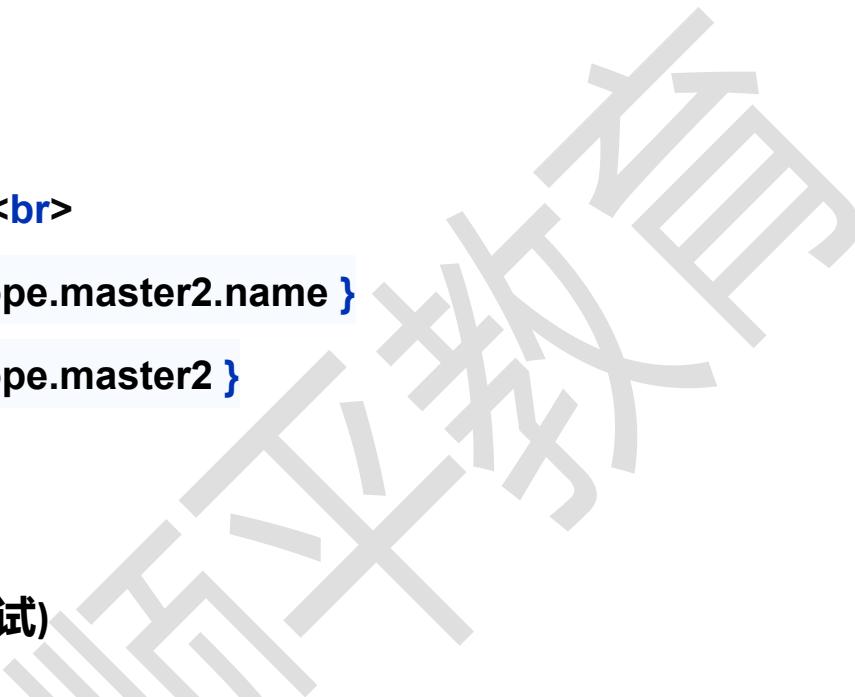
3. 修改 vote_ok.jsp , 增加代码

<hr>
取出 session 域的数据

主人名字= \${sessionScope.master2.name }
主人信息= \${sessionScope.master2 }

4. 完成测试(页面测试)

5. 完成测试(Postman 测试)



POST http://localhost:9998/vote/vote08

Params Auth Headers (11) Body Pre-req. Tests Settings

x-www-form-urlencoded

KEY	VALUE
<input checked="" type="checkbox"/> id	888
<input checked="" type="checkbox"/> name	大勇
<input checked="" type="checkbox"/> pet.id	8

Body Cookies (1) Headers (6) Test Results

Pretty Raw Preview Visualize

address: guangzhou
主人名字= 大勇 主人信息= Master{id=888, name='大勇', pet=Pet{id=8, name='旺财', status='在', type='dog'}}
主人信息= Pet{id=8, name='旺财', status='在', type='dog'}

6.3 @ModelAttribute 实现 prepare 方法

6.3.1 应用实例

- 基本说明

开发中，有时需要使用某个前置方法(比如 `prepareXXX()`, 方法名由程序员定)给目标方法准备一个模型对象

1. `@ModelAttribute` 注解可以实现 这样的需求
2. 在某个方法上，增加了`@ModelAttribute` 注解后
3. 那么在调用该 `Handler` 的任何一个方法时，都会先调用这个方法

- 应用实例

修改 `VoteHandler.java`, 增加方法 并测试

```
/**  
 * 老韩解读  
 * 1. 当在某个方法上，增加了@ModelAttribute 注解  
 * 2. 那么在调用该 Handler 的任何一个方法时，都会先调用这个方法  
 */
```

`@ModelAttribute`

```
public void prepareModel(){
```

```
    System.out.println("prepareModel()----完成准备工作----");
```

}

6.3.2 @ModelAttribute 最佳实践

- 修改用户信息（就是经典的使用这种机制的应用），流程如下：

1. 在修改前，在前置方法中从数据库查出这个用户
2. 在修改方法(目标方法)中，可以使用前置方法从数据库查询的用户
3. 如果表单中对用户的某个属性修改了，则以新的数据为准，如果没有修改，则以数据库的信息为准，比如，用户的某个属性不能修改，就保持原来的值



7 视图和视图解析器

7.1 基本介绍

1. 在 springMVC 中的目标方法最终返回都是一个视图(有各种视图).

2. 返回的视图都会由一个视图解析器来处理 (视图解析器有很多种)

7.2 自定义视图

7.2.1 为什么需要自定义视图

- 在默认情况下，我们都是返回默认的视图，然后这个返回的视图交由 SpringMVC 的 **InternalResourceViewResolver** 视图处理器来处理的

```
<bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
<property name="prefix" value="/WEB-INF/pages/"></property>
<property name="suffix" value=".jsp"></property>
</bean>
```

- 在实际开发中，我们有时需要自定义视图，这样可以满足更多更复杂的需求。

7.2.2 自定义视图实例-代码实现

- 配置 **springDispatcherServlet-servlet.xml**，增加自定义视图解析器

```
<!--
```

老韩解读

- 配置可以解析自定义的视图的解析器

2. **BeanNameViewResolver** 这个就是可以解析自定义视图的解析器

3. **name="order"** : 表示给这个解析器设置优先级，默认优先级很低 值 **Integer.MAX_VALUE**

4. 一般来说说明，我们自己的视图解析优先级高，**Order** 值越小，优先级越高

```
-->
```

```
<bean class="org.springframework.web.servlet.view.BeanNameViewResolver">  
    <property name="order" value="99"></property>  
</bean>
```

2. 创建 D:\idea_java_projects\springmvc\src\com\hspedu\web\viewresolver\MyView.java - 自定义视图类

```
package com.hspedu.web.viewresolver;  
  
import org.springframework.stereotype.Component;  
import org.springframework.web.servlet.view.AbstractView;  
  
import javax.servlet.http.HttpServletRequest;  
import javax.servlet.http.HttpServletResponse;  
import java.util.Map;  
  
/**  
 * 老韩解读  
 * 1. @Component 表示 该视图会被加载到容器,id 为 myView  
 * @author 韩顺平  
 * @version 1.0  
 */  
  
@Component(value="myView")
```

```
public class MyView extends AbstractView {  
  
    @Override  
  
    protected void renderMergedOutputModel(Map<String, Object> arg0,  
                                         HttpServletRequest arg1,  
                                         HttpServletResponse arg2) throws Exception  
{  
  
    System.out.println("进入到自己的视图");  
  
    // 这里我们自己来确定到哪个页面去,默认的视图解析机制就无效  
    arg1.getRequestDispatcher("/WEB-INF/pages/my_view.jsp").forward(arg1, arg2);  
  
}  
}
```

3. 创建视图解析器

D:\idea_java_projects\springmvc\src\com\hspedu\web\viewresolver\GoodsHandler.java

```
package com.hspedu.web.viewresolver;  
  
import org.springframework.stereotype.Controller;  
import org.springframework.web.bind.annotation.RequestMapping;  
  
/**
```

```
* @author 韩顺平  
* @version 1.0  
*/  
  
@RequestMapping("/goods")  
@Controller  
public class GoodsHandler {  
  
    @RequestMapping(value = "/buy")  
    public String buy() {  
        System.out.println("=====buy()=====");  
        return "myView";  
    }  
}
```

4. 创建 D:\idea_java_projects\springmvc\web\view.jsp 和 /WEB-INF/pages/my_view.jsp

<%--

Created by IntelliJ IDEA.

User: 韩顺平

Version: 1.0

Filename: view

--%>

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
```

```
<html>
<head>
    <title>view</title>
</head>
<body>
    <h2>测试自定义视图</h2>
    <a href="goods/buy">测试自定义视图</a><br/>
</body>
</html>

<%--
    Created by IntelliJ IDEA.
    User: 韩顺平
    Version: 1.0
    Filename: my_view
--%>
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>Title</title>
</head>
<body>
```

```
<h1>从自己的视图过来的</h1>  
<hr/>  
welcome! hspedu~  
</body>  
</html>
```

5. 完成测试(页面测试)

6. 完成测试(Postman 测试)

POST http://localhost:9998/goods/buy

Params Auth Headers (11) **Body** Pre-req. Tests Settings

x-www-form-urlencoded

KEY	VALUE
<input checked="" type="checkbox"/> id	888
<input checked="" type="checkbox"/> name	大勇
<input checked="" type="checkbox"/> pet.id	8

Body Cookies (1) Headers (6) Test Results

Pretty Raw Preview Visualize

从自己的视图过来的

7.2.3 自定义视图工作流程小结

- 自定义视图-小结

1. **自定义视图：**创建一个 View 的 bean，该 bean 需要继承自 AbstractView，并实现 renderMergedOutputModel 方法。
2. **并把自定义 View 加入到 IOC 容器中**
3. **自定义视图的视图处理器，使用 BeanNameViewResolver，这个视图处理器也需要配置到 ioc 容器**
4. BeanNameViewResolver 的调用优先级需要设置一下，设置 order 比 Integer.MAX_VAL 小的值。以确保其在 InternalResourceViewResolver 之前被调用

- **自定义视图-工作流程**

1. SpringMVC 调用目标方法，返回自定义 View 在 IOC 容器中的 id
2. SpringMVC 调用 BeanNameViewResolver 解析视图：从 IOC 容器中获取 返回 id 值对应的 bean，即自定义的 View 的对象
3. SpringMVC 调用自定义视图的 renderMergedOutputModel 方法渲染视图
4. **老韩说明：**如果在 SpringMVC 调用目标方法，返回自定义 View 在 IOC 容器中的 id，不存在，则仍然按照默认的视图处理器机制处理。

7.3 目标方法直接指定转发或重定向

7.3.1 使用实例

- 目标方法中指定转发或者重定向

1. 默认返回的方式是请求转发，然后用视图处理器进行处理，比如在目标方法中这样写：

```
@RequestMapping(value="/test1")
public String test1(){
    System.out.println("--test1--");
    return "ok";
}
```

2. 也可以在目标方法直接指定重定向或转发的 url 地址

3. 如果指定重定向，不能定向到 /WEB-INF 目录中

- 应用实例-代码实现

1. 修改 GoodsHandler.java，增加方法 order()

```
/*
 * 在目标方法直接指定 重定向&请求转发
 * @return
 */
@RequestMapping(value = "/order")
```

```
public String order() {  
    System.out.println("=====order()=====");  
  
    //这里直接指定 转发到哪个页面  
  
    //return "forward:/WEB-INF/pages/my_view.jsp";  
  
    //重定向，如果是重定向，就不能重定向到 /WEB-INF 目录中  
    return "redirect:/login.jsp";  
}
```

2. 修改 view.jsp,

```
<h2>测试自定义视图</h2>  
<a href="goods/buy">测试自定义视图</a><br/>  
<a href="goods/order">测试目标方法直接指定 重定向&请求转发</a><br/>
```

3. 完成测试(页面测试)



测试自定义视图

[测试自定义视图](#)

[测试目标方法直接指定 重定向&请求转发](#)



4. 完成测试(Postman 测试)

KEY	VALUE

7.3.2 指定请求转发流程-Debug 源码

7.3.3 指定重定向流程-Debug 源码

7.4 作业布置

- 把前面老师讲过的 SpringMVC 映射数据请求，模型数据，视图和视图解析的案例，自己写一遍

-一定要自己写一遍,否则没有印象, 理解不会深入

-把老师 Dubug 过的源码, 自己也走一下, 加深理解(不用每一条语句, 都 debug, 找流程。。)

2. 完成一个简单的用户登录案例

1) 编写登录页面 login.jsp

2) LoginHandler [doLogin 方法], 如果用户名输入用户名是 hsp , 密码 123 就可以登录成功.

3) 创建 JavaBean : User.java

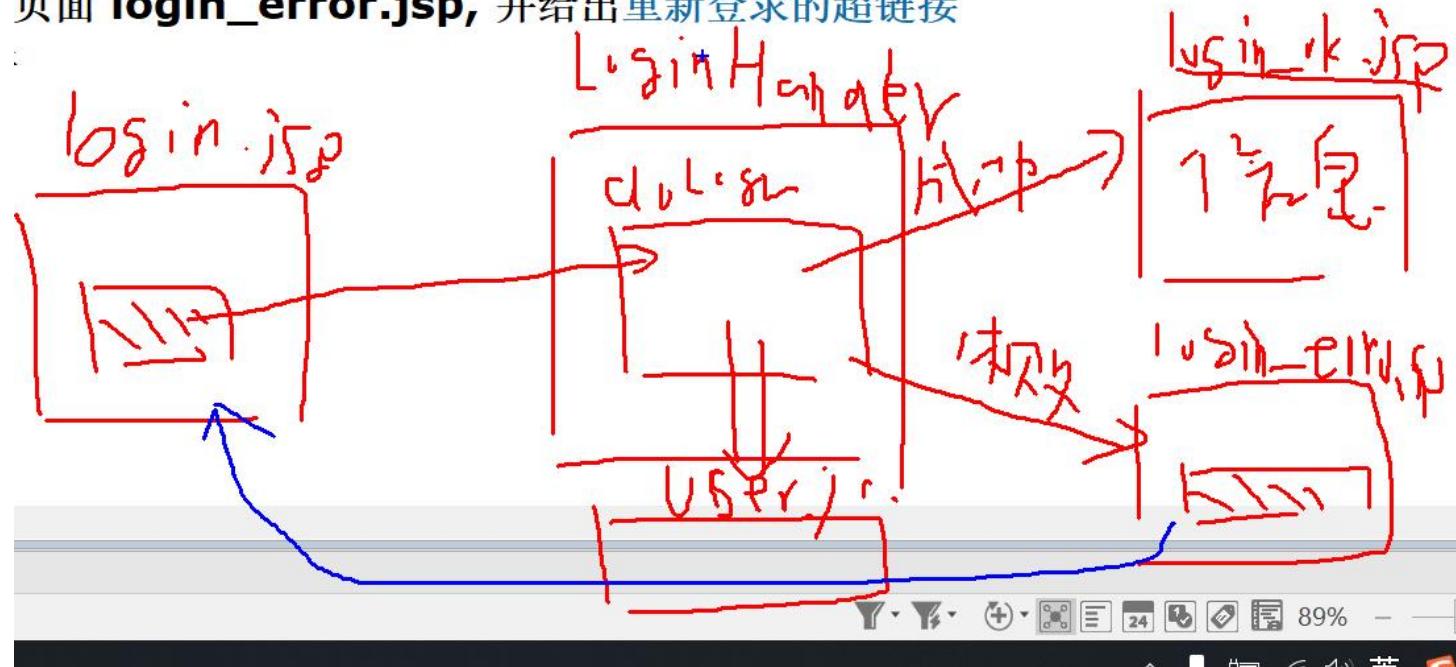
4) 表单提交数据到 doLogin 方法, 以 User 对象形式接收.

5) 登录成功到, 页面 login_ok.jsp 并显示登录欢迎信息

6) 登录失败, 到页面 login_error.jsp, 并给出[重新登录的超链接](#)

-老韩的思路分析

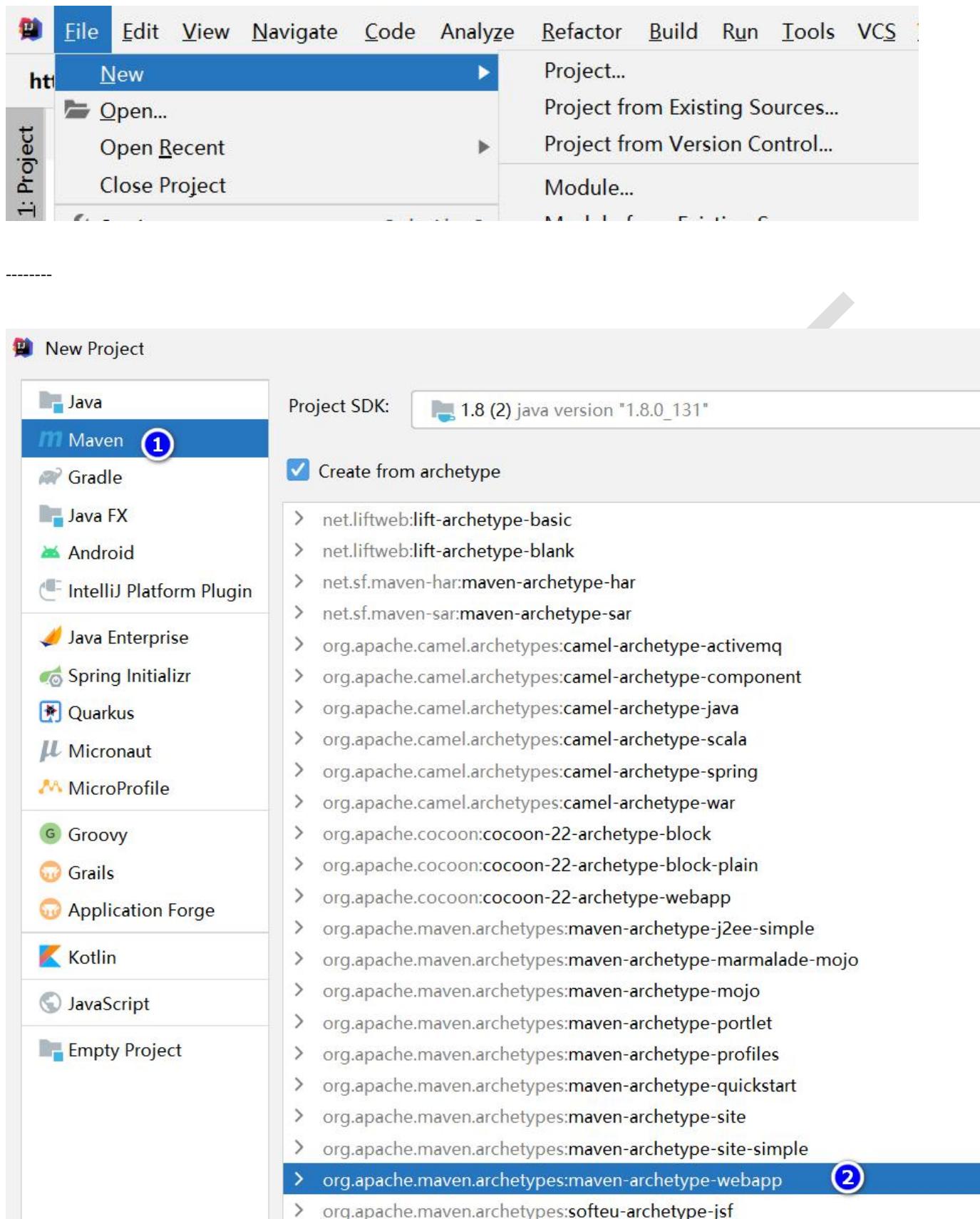
页面 **login_error.jsp**, 并给出重新登录的超链接



8 韩顺平 自己实现 SpringMVC 底层机制 【老韩新增】【核心分发控制器+ Controller 和 Service 注入容器 + 对象自动装配 + 控制器方法获取参数 + 视图解析 + 返回 JSON 格式数据】

8.1 搭建 SpringMVC 底层机制开发环境

1、创建 Maven 项目 **hsp-springmvc** [提示：我们在讲解 **hsptomcat** 时，我们已经使用过]



New Project

Name: hsp-springmvc ①

Location: D:\idea_java_projects\hsp-springmvc ②

▼ Artifact Coordinates

GroupId: com.hspedu ③

The name of the artifact group, usually a company domain

ArtifactId: hsp-springmvc

The name of the artifact within the group, usually a project name

Version: 1.0-SNAPSHOT

New Project

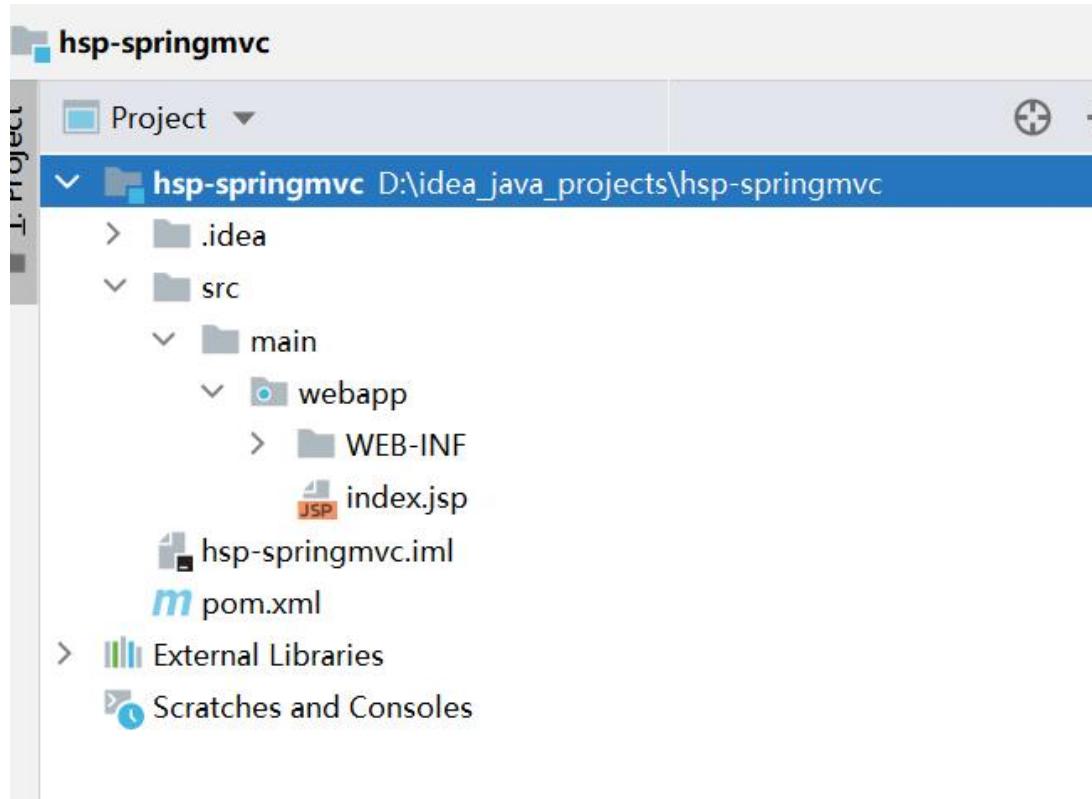
Maven home directory: D:/program/JavaIDEA 2020.2/plugins/maven/lib/maven3
(Version: 3.6.3)

User settings file: C:\Users\Administrator\.m2\settings.xml

Local repository: C:\Users\Administrator\.m2\repository

Properties

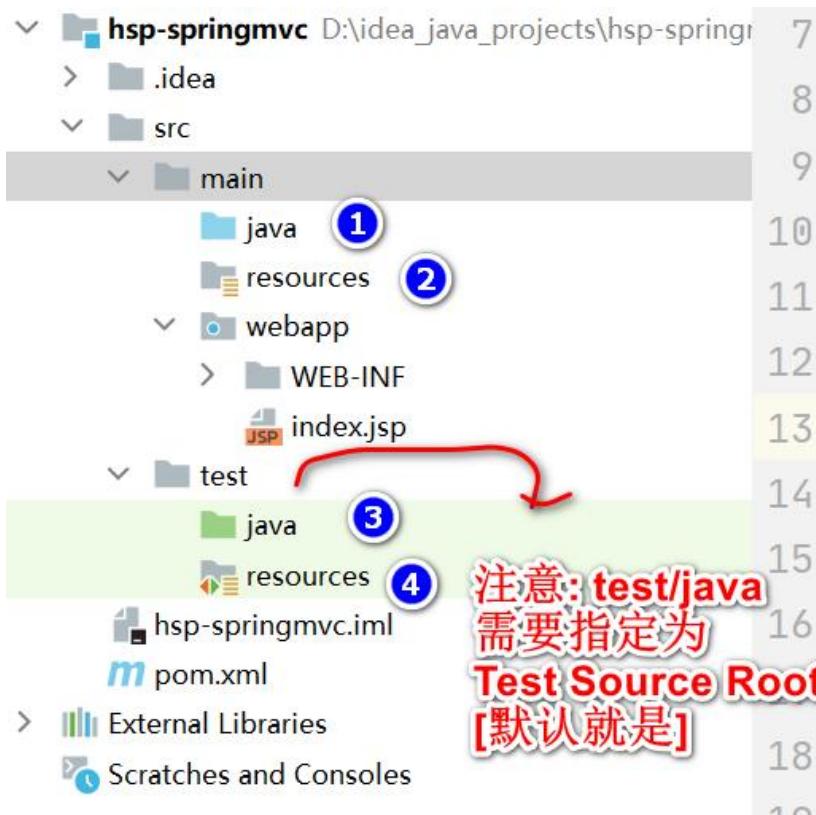
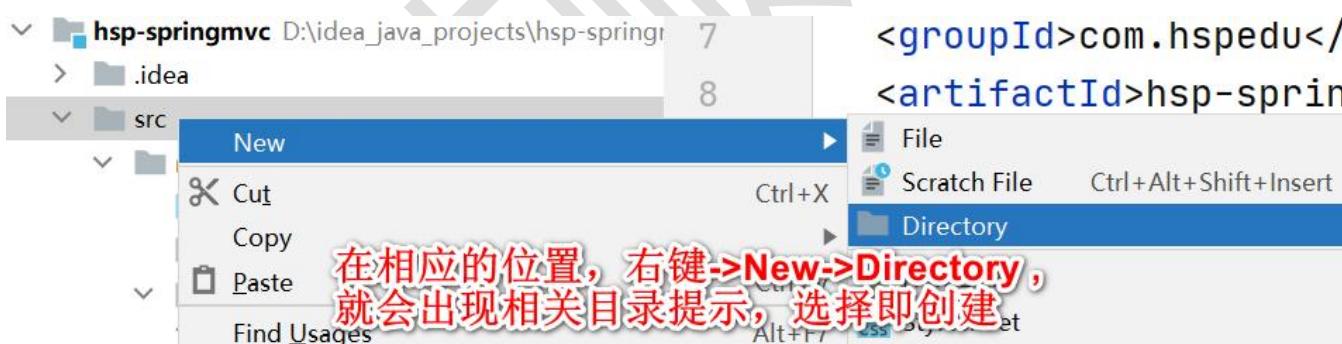
groupId	com.hspedu
artifactId	hsp-springmvc
version	1.0-SNAPSHOT
archetypeGroupId	org.apache.maven.archetypes
archetypeArtifactId	maven-archetype-webapp
archetypeVersion	RELEASE



2、对 hsp-springmvc 进行配置：修改 D:\idea_java_projects\hsp-springmvc\pom.xml，将 1.7 修改成 1.8

```
<properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <maven.compiler.source>1.8</maven.compiler.source>
    <maven.compiler.target>1.8</maven.compiler.target>
</properties>
```

3、对 hsp-springmvc 进行配置：创建 main 和 test 相关的源码目录和资源目录和测试目录（这是 Maven 工程的开发规范，前面讲过..），如图

**提示:****4、引入需要的基本的 jar 包, 修改 D:\idea_java_projects\hsp-springmvc\pom.xml**

```
<dependencies>
```

```
  <dependency>
```

```
<groupId>junit</groupId>
<artifactId>junit</artifactId>
<version>4.11</version>
<scope>test</scope>
</dependency>

<!-- 引入 servlet 原生依赖 -->
<dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>javax.servlet-api</artifactId>
    <version>3.1.0</version>
    <scope>provided</scope>
</dependency>
<!-- 引入 dom4j, 解析 xml-->
<dependency>
    <groupId>dom4j</groupId>
    <artifactId>dom4j</artifactId>
    <version>1.6.1</version>
</dependency>
<!--commons-lang3 有很多常用工具类-->
<dependency>
    <groupId>org.apache.commons</groupId>
```

```
<artifactId>commons-lang3</artifactId>  
<version>3.5</version>  
</dependency>  
</dependencies>
```

5、到此：项目开发环境搭建 OK

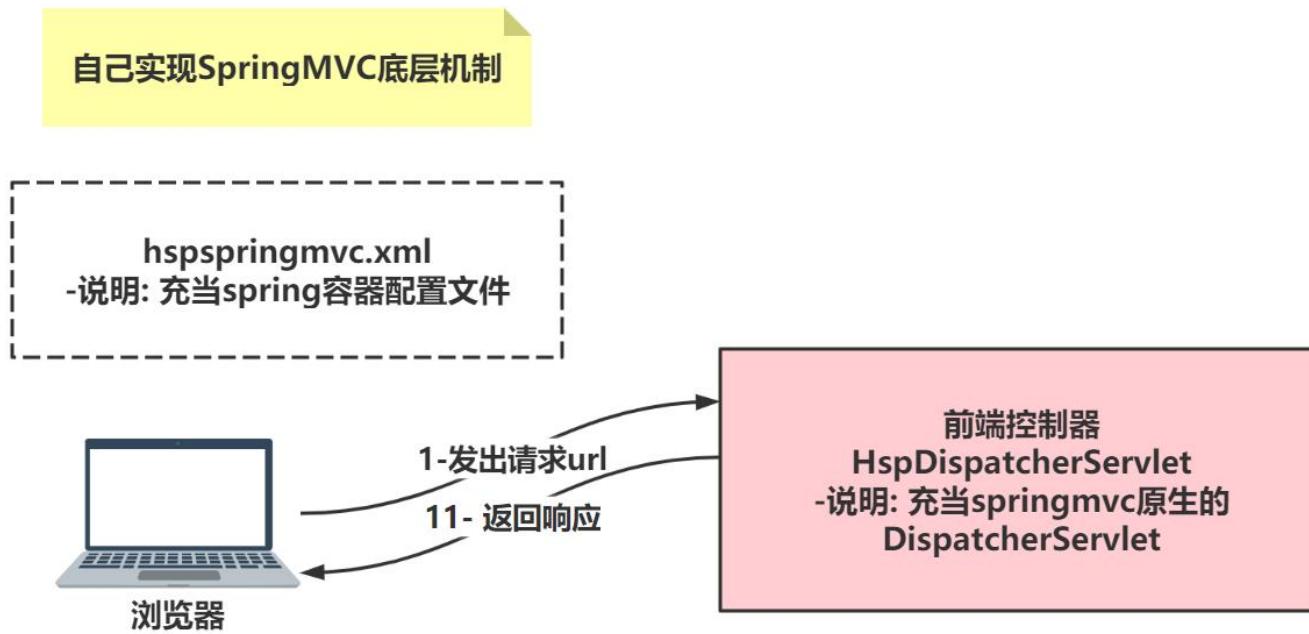
8.2 自己实现 SpringMVC 底层机制 【核心分发控制器+ Controller 和 Service 注入容器 + 对象自动装配 + 控制器方法获取参数 + 视图解析 + 返回 JSON 格式数据】

8.2.1 实现任务阶段 1- 开发 HspDispatcherServlet

8.2.1.1 说明：编写 HspDispatcherServlet 充当原生的 DispatcherServlet(即核心控制器)

8.2.1.2 分析+代码实现

- 分析示意图



- 代码实现

1

D:\idea_java_projects\hsp-springmvc\src\main\java\com\hspedu\hspspringmvc\servlet\Hs
pDispatcherServlet.java

```
package com.hspedu.hspspringmvc.servlet;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;
```

```
/*
 * @author 韩顺平
 * @version 1.0
 */

public class HspDispatcherServlet extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws
    ServletException, IOException {
        super.doGet(req, resp);
    }

    @Override
    protected void doPost(HttpServletRequest req, HttpServletResponse resp) throws
    ServletException, IOException {
        super.doPost(req, resp);
    }
}
```

2、创建 D:\idea_java_projects\hsp-springmvc\src\main\resources\hspspringmvc.xml，充当原生的 applicationContext-mvc.xml 文件(就是 spring 的容器配置文件，比如指定要扫描哪些包下的类)，先创建给空的文件

3、修改 D:\idea_java_projects\hsp-springmvc\src\main\webapp\WEB-INF\web.xml, 完成 HspDispatcherServlet 的配置

```
<!DOCTYPE web-app PUBLIC  
        "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"  
        "http://java.sun.com/dtd/web-app_2_3.dtd" >  
  
<web-app>  
    <display-name>Archetype Created Web Application</display-name>  
  
    <!-- 配置 HspDispatcherServlet -->  
    <servlet>  
        <servlet-name>HspDispatcherServlet</servlet-name>  
  
        <servlet-class>com.hspedu.hspspringmvc.servlet.HspDispatcherServlet</servlet-class>  
    >  
        <!-- 指定要读取的 hspSpringmvc.xml 该文件充当 applicationContext-mvc.xml  
        文件(就是 spring 的容器文件) -->  
        <init-param>  
            <param-name>contextConfigLocation</param-name>  
            <param-value>classpath:hspSpringmvc.xml</param-value>  
        </init-param>
```

```
<!-- 这个HspDispatcherServlet 实例完成分发处理任务，随tomcat 启动而创建-->

<load-on-startup>1</load-on-startup>

</servlet>

<servlet-mapping>

    <servlet-name>HspDispatcherServlet</servlet-name>
    <!--拦截所有请求,因此配成 / 完成分发处理-->
    <url-pattern>/</url-pattern>
</servlet-mapping>

</web-app>
```

8.2.1.3 配置 Tomcat, 完成测试

1、配置 Tomcat，具体步骤我们在讲解 JavaWeb 时，已经操作过多次

听我说: HspTomcat+自己 Servlet

2

、

修

改

D:\idea_java_projects\hsp-springmvc\src\main\java\com\hspedu\servlet\HspDispatcherServlet.java

```
package com.hspedu.hspspringmvc.servlet;
```

```
import javax.servlet.ServletException;
```

```
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;

/**
 * @author 韩顺平
 * @version 1.0
 */

public class HspDispatcherServlet extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws
    ServletException, IOException {
        System.out.println("HspDispatcherServlet doGet()被调用");
    }

    @Override
    protected void doPost(HttpServletRequest req, HttpServletResponse resp) throws
    ServletException, IOException {
        System.out.println("HspDispatcherServlet doPost()被调用");
    }
}
```

3、启动 Tomcat

4、完成测试 http://localhost:8080/hsp_springmvc/abc

19-Feb-2022 11:19:23.434 信息 [localhost-startStop-1]：正在启动名为“abc”的应用

19-Feb-2022 11:19:23.496 信息 [localhost-startStop-1]：正在停止名为“abc”的应用

HspDispatcherServlet doGet()被调用

8.2.2 实现任务阶段 2- 完成客户端/浏览器可以请求控制层

8.2.2.1 创建自己的 Controller 和自定义注解

- 分析示意图

- 代码实现

1.

创

建

D:\idea_java_projects\hsp-springmvc\src\main\java\com\hspedu\controller\MonsterController.java

```
package com.hspedu.controller;
```

```
import com.hspedu.hspsspringmvc.annotation.Controller;
```

```
import com.hspedu.hspsspringmvc.annotation.RequestMapping;
```

```
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;
import java.io.PrintWriter;

/**
 * @author 韩顺平
 * @version 1.0
 * 控制器
 */

public class MonsterController {
    public void listMonsters(HttpServletRequest request, HttpServletResponse response) {
        response.setContentType("text/html;charset=utf-8");
        try {
            PrintWriter printWriter = response.getWriter();
            printWriter.write("<h1>妖怪列表</h1>");
        }
    }
}
```

```
        } catch (IOException e) {  
            e.printStackTrace();  
        }  
    }  
}
```

2. 创建自定义注解 ,

D:\idea_java_projects\hsp-springmvc\src\main\java\com\hspedu\hspspringmvc\annotation\Controller.java

```
package com.hspedu.hspspringmvc.annotation;  
  
import java.lang.annotation.*;  
  
/**  
 * @author 韩顺平  
 * @version 1.0  
 */  
  
@Target(ElementType.TYPE)  
@Retention(RetentionPolicy.RUNTIME)  
@Documented  
public @interface Controller {
```

```
        String value() default "";  
    }
```

3. 创建自定义注解 ,

D:\idea_java_projects\hsp-springmvc\src\main\java\com\hspedu\hspspringmvc\annotation\RequestMapping.java

```
package com.hspedu.hspspringmvc.annotation;
```

```
import java.lang.annotation.*;
```

```
/**
```

```
 * @author 韩顺平
```

```
 * @version 1.0
```

```
 */
```

```
@Target(ElementType.METHOD)
```

```
@Retention(RetentionPolicy.RUNTIME)
```

```
@Documented
```

```
public @interface RequestMapping {
```

```
    String value() default "";
```

```
}
```

8.2.2.2 配置 hpspringmvc.xml

1 在该文件指定，我们的 **springmvc** 要扫描的包

```
<?xml version="1.0" encoding="UTF-8" ?>  
<beans>  
    <component-scan base-package="com.hspedu.controller"></component-scan>  
</beans>
```

8.2.2.3 编写 XMLParser 工具类，可以解析 hpspringmvc.xml

0. 完成功能说明：

-编写 XMLParser 工具类，可以解析 hpspringmvc.xml，得到要扫描的包

-如图

```
D:\program\hspjdk8\bin\java.exe ...  
basePackage= com.hspedu.controller,com.hspedu.service
```

1

创

建

D:\idea_java_projects\hsp-springmvc\src\main\java\com\hspedu\hpspringmvc\xml\XMLP

aser.java

```
package com.hspedu.hspspringmvc.xml;
```

```
import org.dom4j.Attribute;
import org.dom4j.Document;
import org.dom4j.DocumentException;
import org.dom4j.Element;
import org.dom4j.io.SAXReader;
```

```
import java.io.InputStream;
import java.util.List;
```

```
/*
 * @author 韩顺平
 * @version 1.0
 * 开发一个XML工具，可以得到要扫描的包
 */
```

```
public class XMLPaser {
```

```
    public static String getbasePackage(String xmlFile) {
```

```
        try {
```

```
            SAXReader saxReader = new SAXReader();
```

```
            InputStream
```

```
            resourceAsStream
```

```
=
```

```
XMLParser.class.getClassLoader().getResourceAsStream(xmlFile);

        Document document = saxReader.read(resourceAsStream);

        Element rootElement = document.getRootElement();

        Element componentScanElement = rootElement.element("component-scan");

        Attribute attribute = componentScanElement.attribute("base-package");

        String basePackage = attribute.getText();

        return basePackage;

    } catch (DocumentException e) {

        e.printStackTrace();

    }

    return "";
}

}
```

2 创建
D:\idea_java_projects\hsp-springmvc\src\test\java\com\hspedu\test\HspSpringMvcTest.java

a 完成测试

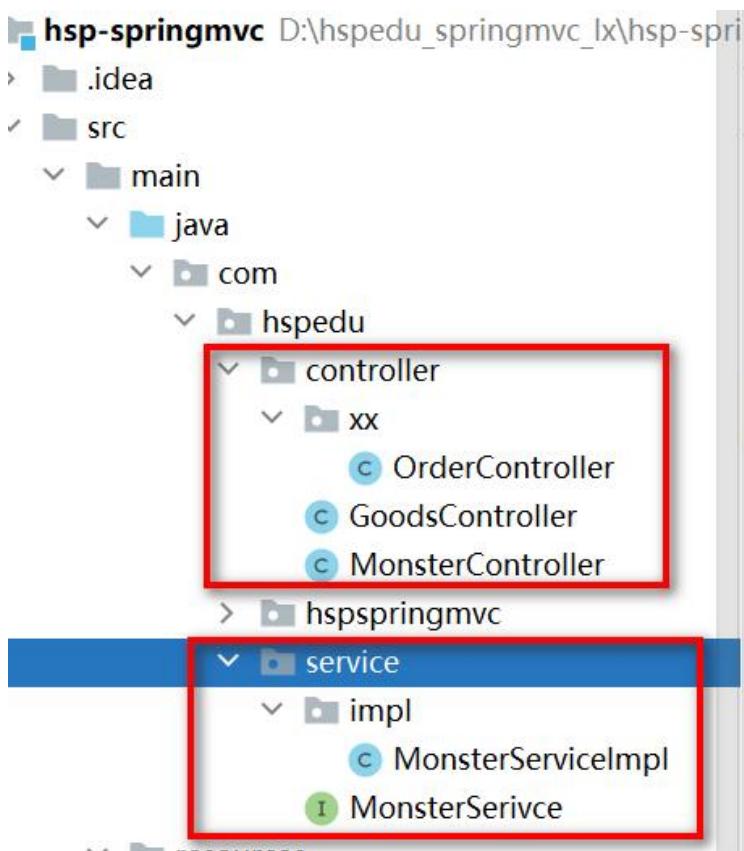
```
package com.hspedu.test;
```

```
import com.hspedu.hspspringmvc.xml.XMLParser;
import org.junit.Test;
```

```
/**  
 * @author 韩顺平  
 * @version 1.0  
 */  
  
public class HspSpringMvcTest {  
  
    @Test  
    public void readXML() {  
        String basePackage = XMLParser.getbasePackage("hpspringmvc.xml");  
        System.out.println("basePackage= " + basePackage);  
    }  
}
```

8.2.2.4 开发 HspWebApplicationContext，充当 Spring 容器-得到扫描类的全路径列表

0. 完成的功能说明



- 把指定的目录包括子目录下的 java 类的全路径扫描到集合中,比如 ArrayList

-如图[对 java 基础知识的使用.]

扫 描 后 的 classFullPathList= [com.hspedu.controller.MonsterController,
com.hspedu.service.impl.MonsterServiceImpl, com.hspedu.service.MonsterService]

1

创

建

D:\idea_java_projects\hsp-springmvc\src\main\java\com\hspedu\hspspringmvc\context\H
spWebApplicationContext.java

```
package com.hspedu.hspspringmvc.context;

import com.hspedu.hspspringmvc.xml.XMLPaser;

import java.io.File;
import java.net.URL;
import java.util.ArrayList;

/**
 * @author 韩顺平
 * @version 1.0
 */
public class HspWebApplicationContext {
    private ArrayList<String> classFullPathList = new ArrayList<>();

    /**
     * 初始化自定义的spring 容器，目标就是把要@Controller 等等初始化到容器中
     */
    public void init() {
        //后面可以通过 web.xml 灵活的获取
        String basePackage = XMLPaser.getbasePackage("hspSpringmvc.xml");
        String[] packages = basePackage.split(",");
        if (packages.length > 0) {
```

```
for (String pack : packages) {  
    //调用方法对各个包进行扫描,获取到 classFullPathList  
    // 比 如 com.hspedu.service.MonsterService  
    com.hspedu.controller.MonsterController  
    //System.out.println("pack= " + pack);  
    scanPackage(pack);  
}  
}  
//测试  
System.out.println("扫描后的 classFullPathList= " + classFullPathList);  
}  
  
public void scanPackage(String pack) {  
    //老韩解读  
    //1. pack 形式如: com.hspedu.controller  
    //2. 需要得到真正执行的类形式为  
    /D:\idea_java_projects\hsp-springmvc\target\hsp-springmvc  
    \WEB-INF\classes\com\hspedu\controller\MonsterController.class  
    URL url = this.getClass()  
        .getClassLoader()  
        .getResource("/") + pack.replaceAll("\\.", "/");
```

```
//3. 可以测试输出 url，注意这个测试，需要走tomcat，不能走Test，否则url为null

//System.out.println("url=" + url);

//4. 取出路径，可以输出看看

String path = url.getFile();

File dir = new File(path);

for (File f : dir.listFiles()) {//扫描这个目录及其子目录

    if(f.isDirectory()) {

        //如果是目录，递归扫描

        scanPackage(pack + "." + f.getName());

    } else {

        //如果是文件，也是就class类

        //获取到类的全路径，并放入到classFullPathList集合

        String classFullPath = pack + "." + f.getName().replaceAll(".class","");
        classFullPathList.add(classFullPath);

    }

}

}
```

pDispatcherServlet.java , 增加

@Override

```
public void init(ServletConfig config) throws ServletException {  
    HspWebApplicationContext hspWebApplicationContext = new  
    HspWebApplicationContext();  
    hspWebApplicationContext.init();  
}
```

3、启动 Tomcat 完成测试，看看扫描是否成功。需要使用 Tomcat 启动方式完成测试，直接用 Junit 测试 URL 是 null

15:03:30.297 信 息 [RMI TCP Connection(3)-127.0.0.1]
org.apache.jasper.servlet.TldScanner.scanJars At least one JAR was scanned for TLDs yet
contained no TLDs. Enable debug logging for this logger for a complete list of JARs that were
scanned but no TLDs were found in them. Skipping unneeded JARs during scanning can
improve startup time and JSP compilation time.

扫 描 后 的 classFullPathList= [com.hspedu.controller.MonsterController,
com.hspedu.service.impl.MonsterServiceImpl, com.hspedu.service.MonsterService]

8.2.2.5 完善 HspWebApplicationContext，充当 Spring 容器-实例化对象到容器中

0 完成功能说明

- 将扫描到的类，在满足条件的情况下(即有相应的注解@Controller @Service...), 反射注入到 ioc 容器
- 如图

扫 描 后 的 ioc=

{monsterController=com.hspedu.controller.MonsterController@690aac1e}

1 修 改

D:\idea_java_projects\hsp-springmvc\src\main\java\com\hspedu\hspspringmvc\context\H
spWebApplicationContext.java

```
package com.hspedu.hspspringmvc.context;

import com.hspedu.hspspringmvc.annotation.Controller;
import com.hspedu.hspspringmvc.xml.XMLParser;

import java.io.File;
import java.net.URL;
import java.util.ArrayList;
import java.util.concurrent.ConcurrentHashMap;
```

```
/**  
 * @author 韩顺平  
 * @version 1.0  
 */  
  
public class HspWebApplicationContext {  
    private ArrayList<String> classFullPathList = new ArrayList<>();  
    //ioc 用于存放反射后的 bean 对象。  
    private ConcurrentHashMap<String, Object> ioc =  
        new ConcurrentHashMap<>();  
  
    /**  
     * 初始化自定义的 spring 容器，目标就是把要@Controller 等等初始化到容器中  
     */  
  
    public void init() {  
        //后面可以通过 web.xml 灵活的获取  
        String basePackage = XMLParser.getbasePackage("hpspringmvc.xml");  
        String[] packages = basePackage.split(",");  
        if (packages.length > 0) {  
            for (String pack : packages) {  
                ClassPathXmlApplicationContext context =  
                    new ClassPathXmlApplicationContext(pack);  
                context.registerShutdownHook();  
                ioc.putAll(context.getBeansOfType(Map.class));  
            }  
        }  
    }  
}
```

```
//调用方法对各个包进行扫描,获取到 classFullPathList  
//      比      如          com.hspedu.service.MonsterService  
com.hspedu.controller.MonsterController  
    //System.out.println("pack= " + pack);  
    scanPackage(pack);  
}  
}  
//测试  
System.out.println("扫描后的 classFullPathList= " + classFullPathList);  
executeInstance();  
System.out.println("扫描后的 ioc= " + ioc);  
}  
  
//实例化扫描到的类->创建对象->放入到IOC 容器[ConcurrentHashMap]  
public void executeInstance() {  
    if (classFullPathList.size() == 0) {  
        //抛出一个异常, 或者抛出一个自定义异常  
        //throw new RuntimeException("没有需要实例化的对象");  
        //我这里直接 return;  
        return;  
    }  
    try {
```

```
for (String classFullPath : classFullPathList) {  
    Class<?> clazz = Class.forName(classFullPath);  
    if (clazz.isAnnotationPresent(Controller.class)) {  
        //得到该类的类名(首字母小写), 作为 key  
        String beanName = clazz.getSimpleName().substring(0,  
1).toLowerCase() +  
            clazz.getSimpleName().substring(1);  
        ioc.put(beanName, clazz.newInstance());  
    } //如果有其它注解, 可以在这里扩展  
}  
} catch (Exception e) {  
    e.printStackTrace();  
} finally {  
}  
}  
}
```

2 完成测试 (提示: 启动 tomcat 来加载 HspDispatcherServlet 的方式测试)

19-Feb-2022 15:53:52.080 信 息 [RMI TCP Connection(3)-127.0.0.1]
org.apache.jasper.servlet.TldScanner.scanJars At least one JAR was scanned for
TLDs yet contained no TLDs. Enable debug logging for this logger for a complete

list of JARs that were scanned but no TLDs were found in them. Skipping unneeded JARs during scanning can improve startup time and JSP compilation time.

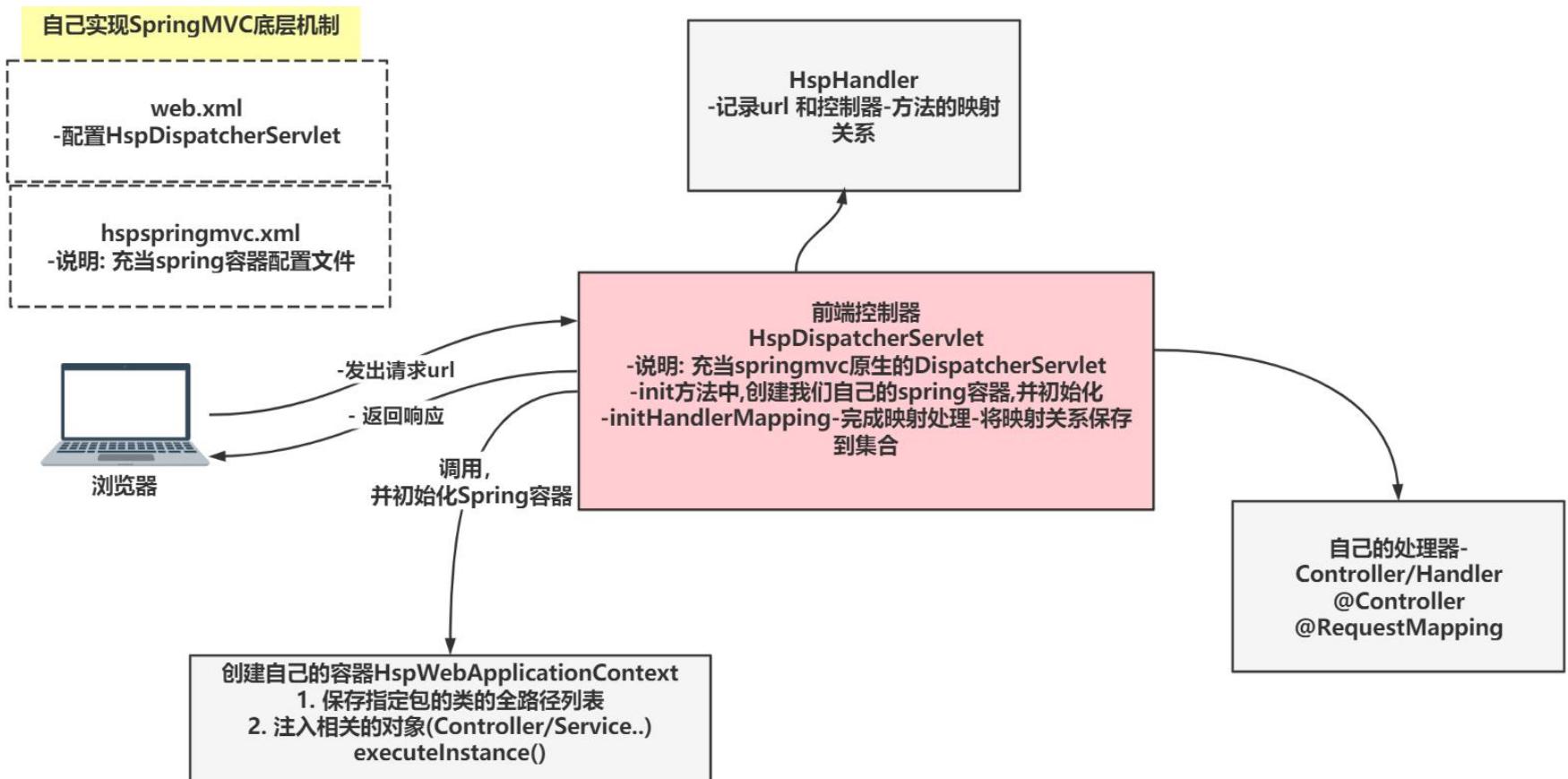
扫 描 后 的 classFullPathList= [com.hspedu.controller.MonsterController,
com.hspedu.service.impl.MonsterServiceImpl,
com.hspedu.service.MonsterService]

扫 描 后 的 ioc=
{monsterController=com.hspedu.controller.MonsterController@690aac1e}

8.2.2.6 完成请求 URL 和控制器方法的映射关系

0. 完成功能说明

-示意图分析



-将配置的@RequestMapping 的 url 和 对应的 控制器-方法 映射关系保存到集合中

-如图

```

handlerList= [HspHandler{url='/monster/list',
controller=com.hspedu.controller.MonsterController@714dab1,      method=public      void
com.hspedu.controller.MonsterController.listMonsters(javax.servlet.http.HttpServletRequest,javax.servlet.http.HttpServletResponse)}]
  
```

1

创

建

D:\idea_java_projects\hsp-springmvc\src\main\java\com\hspedu\hpspringmvc\handler\H
spHandler.java

```
package com.hspedu.hsspringmvc.handler;

import java.lang.reflect.Method;

/**
 * @author 韩顺平
 * @version 1.0
 */

public class HspHandler {

    private String url;
    private Object controller;
    private Method method;

    public HspHandler(String url, Object controller, Method method) {
        this.url = url;
        this.controller = controller;
        this.method = method;
    }

    public HspHandler() {
    }
}
```

```
public String getUrl() {  
    return url;  
}  
  
public void setUrl(String url) {  
    this.url = url;  
}  
  
public Object getController() {  
    return controller;  
}  
  
public void setController(Object controller) {  
    this.controller = controller;  
}  
  
public Method getMethod() {  
    return method;  
}  
  
public void setMethod(Method method) {  
    this.method = method;  
}
```

```
}
```

```
@Override  
public String toString() {  
    return "HspHandler{" +  
        "url='" + url + '\'' +  
        ", controller=" + controller +  
        ", method=" + method +  
        '}';  
}
```

2

修

改

D:\idea_java_projects\hsp-springmvc\src\main\java\com\hspedu\hspspringmvc\servlet\Hs
pDispatcherServlet.java

```
package com.hspedu.hspspringmvc.servlet;  
  
import com.hspedu.hspspringmvc.annotation.Controller;  
import com.hspedu.hspspringmvc.annotation.RequestMapping;  
import com.hspedu.hspspringmvc.context.HspWebApplicationContext;  
import com.hspedu.hspspringmvc.handler.HspHandler;  
  
import javax.servlet.ServletConfig;
```

```
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;
import java.lang.reflect.Method;
import java.util.ArrayList;
import java.util.Map;

/**
 * @author 韩顺平
 * @version 1.0
 */
public class HspDispatcherServlet extends HttpServlet {
    //存放 url 和 控制器方法的映射关系
    private ArrayList<HspHandler> handlerList =
        new ArrayList<>();
    private HspWebApplicationContext hspWebApplicationContext;

    @Override
    public void init(ServletConfig config) throws ServletException {
```

```
hspWebApplicationContext = new HspWebApplicationContext();
hspWebApplicationContext.init();
```

```
//调用initHandlerMapping()
```

搞定 完成控制器层 url---> Controller ---> 方法的映射关系

```
initHandlerMapping();
```

//测试(启动Tomcat 装载 HspDispatcherServlet 对象方式来测试)

```
System.out.println("handlerList= " + handlerList);
```

```
}
```

```
@Override
```

```
protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws
ServletException, IOException {
```

```
    System.out.println("HspDispatcherServlet doGet()被调用");
```

```
}
```

```
@Override
```

```
protected void doPost(HttpServletRequest req, HttpServletResponse resp) throws
ServletException, IOException {
```

```
    System.out.println("HspDispatcherServlet doPost()被调用");
```

```
}
```

```
/**  
 * 1. 完成控制器层 url---> Controller ---> 方法的映射关系(该关系封装到  
HspHandler 对象)  
 * 2. 并放入到 handlerList 集合中  
 * 3. 后面我们可以通过 handlerList 结合 找到某个 url 请求对应的控制器的方法  
(!!!)  
*/  
  
private void initHandlerMapping() {  
    if(hspWebApplicationContext.ioc.isEmpty()) {  
        throw new RuntimeException("spring ioc 容器为空");  
    }  
    for(Map.Entry<String, Object> entry:  
        hspWebApplicationContext.ioc.entrySet() {  
            Class<?> clazz = entry.getValue().getClass();  
            if(clazz.isAnnotationPresent(Controller.class)) {  
                Method[] declaredMethods = clazz.getDeclaredMethods();  
                for (Method declaredMethod : declaredMethods) {  
                    if(declaredMethod.isAnnotationPresent(RequestMapping.class)) {  
                        RequestMapping requestMappingAnnotation =  
                            declaredMethod.getAnnotation(RequestMapping.class);  
                        String url = requestMappingAnnotation.value();
```

```
        handlerList.add(new  
        HspHandler(url,entry.getValue(),declaredMethod));  
    }  
}  
}  
}  
}  
}
```

3 完成测试(启动 Tomcat , 加载 HspDispatcherServlet 方式), 注意看后台输出

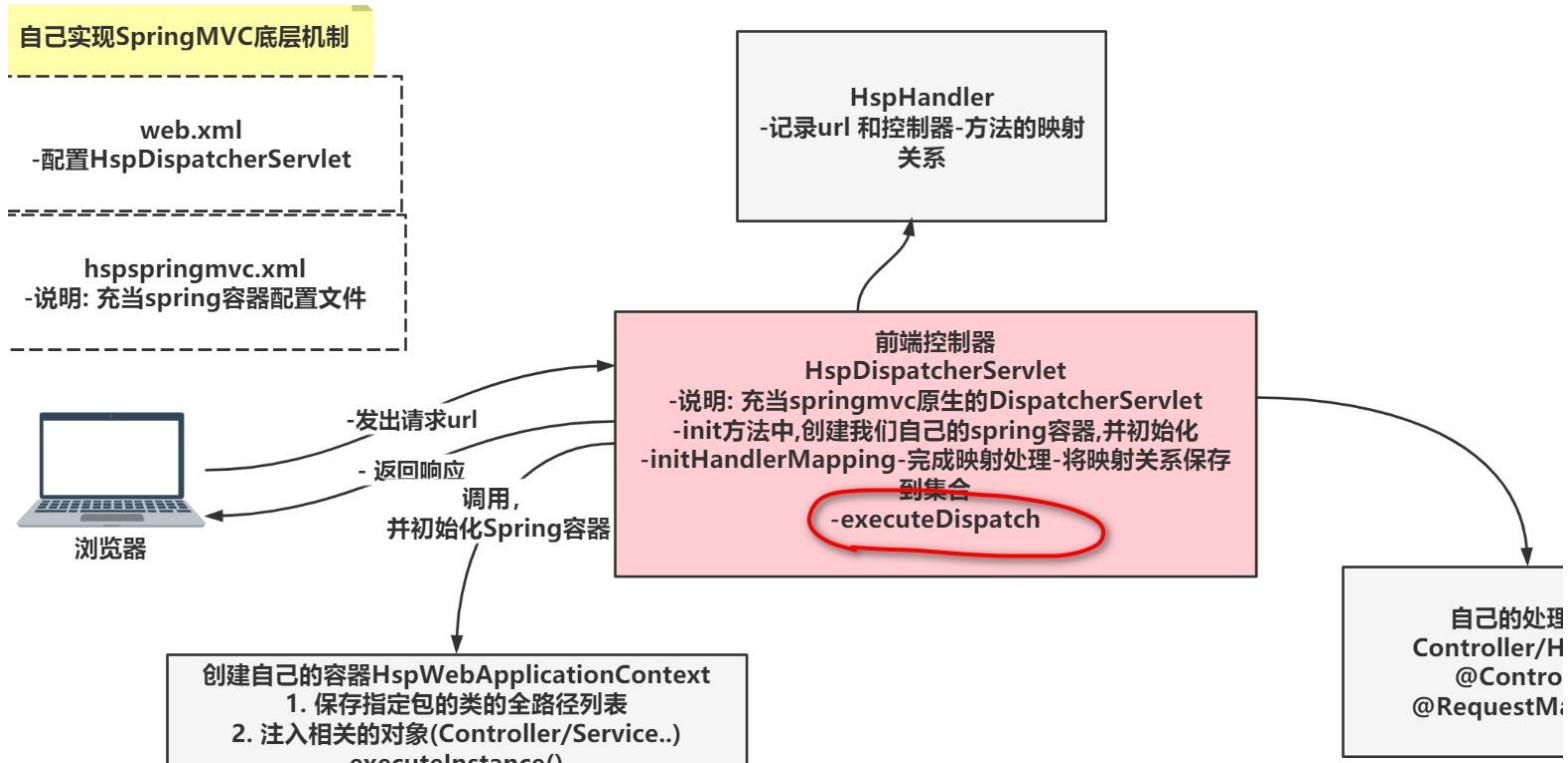
扫描后的 ioc= {monsterController=com.hspedu.controller.MonsterController@714dab1}

```
handlerList=  
[HspHandler{url='/monster/list',  
controller=com.hspedu.controller.MonsterController@714dab1, method=public void  
com.hspedu.controller.MonsterController.listMonsters(javax.servlet.http.HttpServletRequest,javax.servlet.http.HttpServletResponse)}]
```

8.2.2.7 ┆ 完成 HspDispatcherServlet 分发请求到对应控制器方法

0. 完成功能说明

- 示意图[分析说明]



- 当用户发出请求，根据用户请求 url 找到对应的控制器-方法，并反射调用



妖怪列表信息

- 如果用户请求的路径不存在，返回 404



404 NOT FOUND

1

修

改

D:\idea_java_projects\hsp-springmvc\src\main\java\com\hspedu\hspspringmvc\servlet\Hs
pDispatcherServlet.java

```
package com.hspedu.hspspringmvc.servlet;
```

```
import com.hspedu.hspspringmvc.annotation.Controller;  
import com.hspedu.hspspringmvc.annotation.RequestMapping;  
import com.hspedu.hspspringmvc.context.HspWebApplicationContext;  
import com.hspedu.hspspringmvc.handler.HspHandler;
```

```
import javax.servlet.ServletConfig;  
import javax.servlet.ServletException;  
import javax.servlet.http.HttpServlet;  
import javax.servlet.http.HttpServletRequest;  
import javax.servlet.http.HttpServletResponse;  
import java.io.IOException;  
import java.lang.reflect.Method;  
import java.util.ArrayList;  
import java.util.Map;
```

```
/**
```

```
* @author 韩顺平  
* @version 1.0  
*/
```

```
public class HspDispatcherServlet extends HttpServlet {
```

```
//存放url 和 控制器方法的映射关系
```

```
private ArrayList<HspHandler> handlerList = new ArrayList<>();  
private HspWebApplicationContext hspWebApplicationContext;
```

```
@Override
```

```
public void init(ServletConfig config) throws ServletException {  
    hspWebApplicationContext = new HspWebApplicationContext();  
    hspWebApplicationContext.init();
```

```
//调用initHandlerMapping() 搞定 完成控制器层 url---> Controller ---> 方法的  
映射关系
```

```
    initHandlerMapping();
```

```
//测试(启动Tomcat 装载 HspDispatcherServlet 对象方式来测试)
```

```
    System.out.println("handlerList= " + handlerList);
```

```
}
```

```
@Override
```

```
protected void doGet(HttpServletRequest req, HttpServletResponse resp)
throws ServletException, IOException {
    doPost(req, resp);
}

@Override
protected void doPost(HttpServletRequest req, HttpServletResponse resp)
throws ServletException, IOException {
    executeDispatch(req, resp);
}

/**
 * 根据 request 请求，返回对应的 HspHandler 对象
 * @param request
 * @return
 */
private HspHandler getHspHandler(HttpServletRequest request) {
    //获取的 URI 也就是 http 请求的 URI
    //比如 http://localhost:8080/monster/list 的 /monster/list 部分
    String requestURI = request.getRequestURI();
    for (HspHandler hspHandler : handlerList) {
        if (requestURI.equals(hspHandler.getUrl())) { //匹配
            return hspHandler;
        }
    }
    return null;
}
```

```
        return hspHandler;
    }

}

//没有匹配，就返回 null
return null;
}

/**
 * executeDispatch 完成请求的分发处理，在 doPost 调用(所有的请求都会走
doPost)
 *
 * @param req
 * @param response
 */
private void executeDispatch(HttpServletRequest req,
                             HttpServletResponse response) {
    HspHandler hspHandler = getHspHandler(req);
    try {
        if (null == hspHandler) {//没有匹配的 Handler
            response.getWriter().print("<h1>404 NOT FOUND</h1>");
        } else {//有匹配的 Handler，就调用
            hspHandler.getMethod().invoke(hspHandler.getController(), req,
                                         response);
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

```
        }

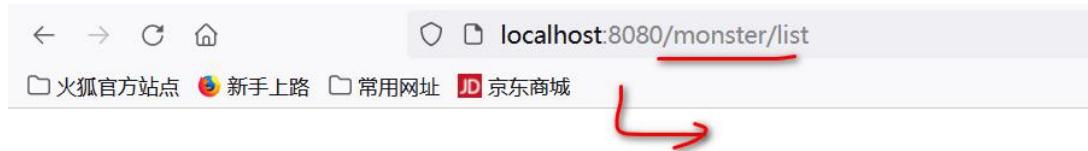
    } catch (Exception e) {
        e.printStackTrace();
    }
}

/**
 * 1. 完成控制器层 url---> Controller ---> 方法的映射关系(该关系封装到 HspHandler 对象)
 * 2. 并放入到 handlerList 集合中
 * 3. 后面我们可以通过 handlerList 结合 找到某个 url 请求对应的方法 (!!!)
 */
private void initHandlerMapping() {
    if (hspWebApplicationContext.ioc.isEmpty()) {
        throw new RuntimeException("spring ioc 容器为空");
    }

    for (Map.Entry<String, Object> entry : hspWebApplicationContext.ioc.entrySet()) {
        Class<?> clazz = entry.getValue().getClass();
```

```
if (clazz.isAnnotationPresent(Controller.class)) {  
    Method[] declaredMethods = clazz.getDeclaredMethods();  
    for (Method declaredMethod : declaredMethods) {  
        if (declaredMethod.isAnnotationPresent(RequestMapping.class)) {  
            RequestMapping requestMappingAnnotation =  
                declaredMethod.getAnnotation(RequestMapping.class);  
            String url = requestMappingAnnotation.value();  
            handlerList.add(new HspHandler(url, entry.getValue(),  
                declaredMethod));  
        }  
    }  
}
```

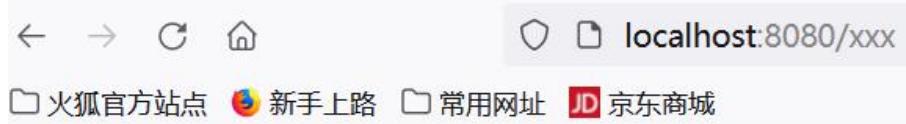
2 完成测试(启动 Tomcat)，如图



妖怪列表

注意：

1. 因为 `getRequestURI()` 返回的是主机后的所有内容
2. 为了和 `handlermapping` 存放的 url 一致，我把 `tomcat` 配置的 `application context`，修改成 /
3. 否则匹配不上，因为按照原来的，得到是 `/hsp_springmvc/monster/list`



404 NOT FOUND

3 增 加 方 法 和 控 制 器 再 玩 一 把 ， 创 建
D:\idea_java_projects\hsp-springmvc\src\main\java\com\hspedu\controller\OrderControll
er.java

```
package com.hspedu.controller;

import com.hspedu.hspspringmvc.annotation.Controller;
import com.hspedu.hspspringmvc.annotation.RequestMapping;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
```

```
import java.io.IOException;
import java.io.PrintWriter;

/**
 * @author 韩顺平
 * @version 1.0
 * 控制器
 */
@Controller
public class OrderController {

    @RequestMapping(value = "/order/list")
    public void listOrders(HttpServletRequest request, HttpServletResponse response) {
        response.setContentType("text/html;charset=utf-8");
        try {
            PrintWriter printWriter = response.getWriter();
            printWriter.write("<h1>订单列表~</h1>");
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

```
}
```

```
@RequestMapping(value = "/order/add")  
public void addOrder(HttpServletRequest request, HttpServletResponse response)
```

```
{
```

```
    response.setContentType("text/html;charset=utf-8");
```

```
    try {
```

```
        PrintWriter printWriter = response.getWriter();
```

```
        printWriter.write("<h1>添加订单～</h1>");
```

```
    } catch (IOException e) {
```

```
        e.printStackTrace();
```

```
}
```

```
}
```

```
}
```



localhost:8080/order/list

火狐官方站点 新手上路 常用网址 JD 京东商城

订单列表～



添加订单~

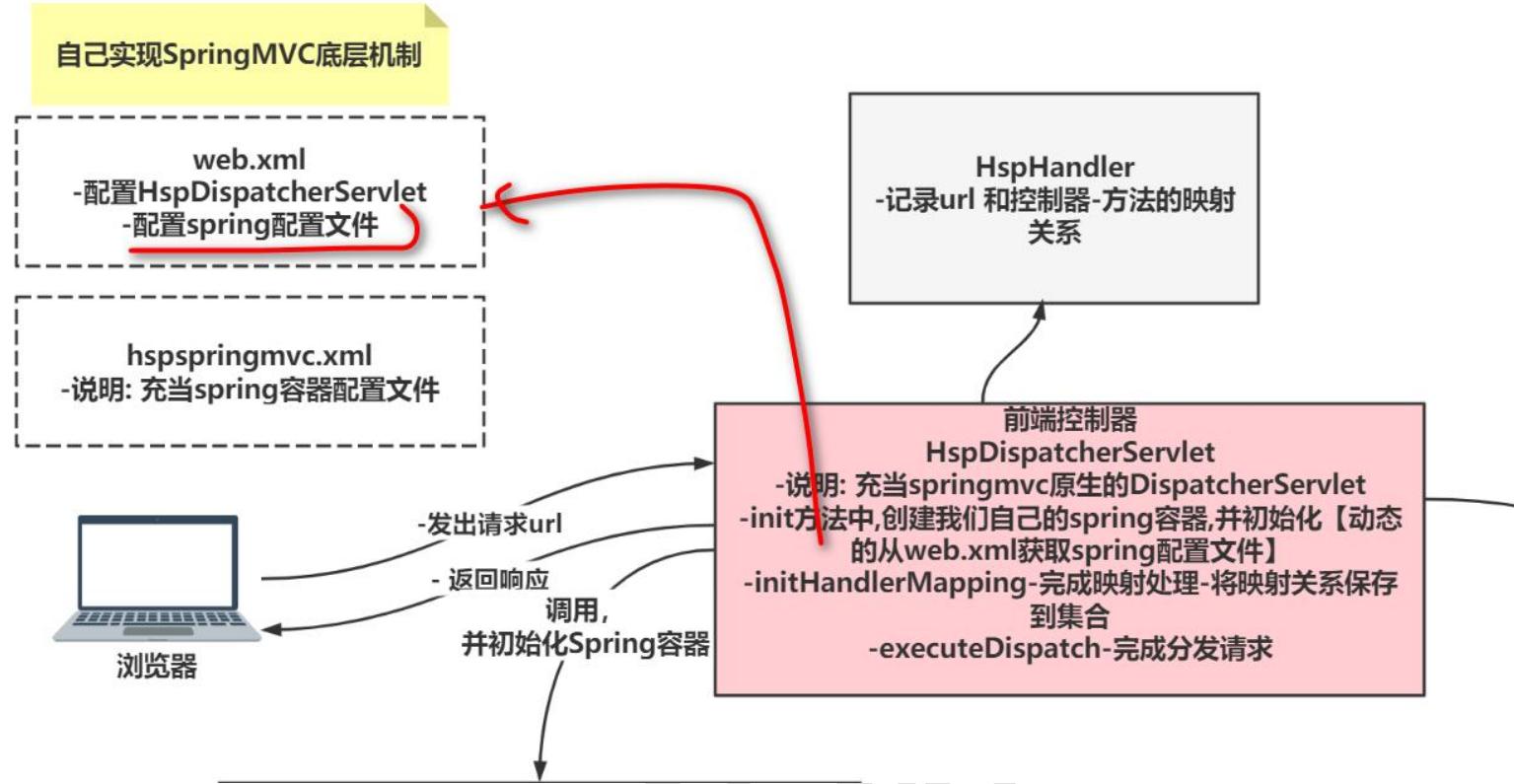
8.2.3 实现任务阶段 3- 从 web.xml 动态获取 hpspringmvc.xml

8.2.3.1 说明：前面我们加载 hpspringmvc.xml 是硬编码，现在做活，从 web.xml 动态获取

8.2.3.2 分析+代码实现+测试

- 完成功能说明

- 分图示意图



- HspDispatcherServlet 在创建并初始化 HspWebApplicationContext, 动态的从 web.xml 中获取到配置文件.

● 代码实现

1.

修

改

D:\idea_java_projects\hsp-springmvc\src\main\java\com\hspedu\hpspringmvc\context\HspWebApplicationContext.java

```
public class HspWebApplicationContext {
    private ArrayList<String> classFullPathList = new ArrayList<>();
```

//ioc 用于存放反射后的bean 对象

```
public ConcurrentHashMap<String, Object> ioc = new ConcurrentHashMap<>();
```

//这个就是我们的spring 容器配置文件,

//这个contextConfigLocation 形式为 classpath:xx.xml

```
private String contextConfigLocation = "";
```

```
public HspWebApplicationContext(String contextConfigLocation) {
```

```
    this.contextConfigLocation = contextConfigLocation;
```

```
}
```

```
/**
```

* 初始化自定义的spring 容器, 目标就是把要@Controller 等等初始化到容器中

```
*/
```

```
public void init() {
```

//后面可以通过web.xml 灵活的获取, 从classpath:xxx.xml 中取出 xxx.xml

String basePackage =

```
XMLParser.getbasePackage(contextConfigLocation.split(":")[1]);
```

```
String[] packages = basePackage.split(",");
```

```
if (packages.length > 0) {
```

```
    for (String pack : packages) {
```

```
//调用方法对各个包进行扫描,获取到 classFullPathList  
// 比 如 com.hspedu.service.MonsterService  
com.hspedu.controller.MonsterController  
    //System.out.println("pack= " + pack);  
    scanPackage(pack);  
}  
}  
//测试  
System.out.println("扫描后的 classFullPathList= " + classFullPathList);  
executeInstance();  
System.out.println("扫描后的 ioc= " + ioc);  
}
```

2. 改修

D:\idea_java_projects\hsp-springmvc\src\main\java\com\hspedu\hspspringmvc\servlet\Hs
pDispatcherServlet.java

```
@Override  
public void init(ServletConfig config) throws ServletException {  
    //初始化时, 读取到 HspDispatcherServlet 的参数 contextConfigLocation  
    String contextConfigLocation =  
        config.getInitParameter("contextConfigLocation");  
    hspWebApplicationContext = new
```

```
HspWebApplicationContext(contextConfigLocation);
```

```
    hspWebApplicationContext.init();
```

//调用 initHandlerMapping() 搞定 完成控制器层 url---> Controller ---> 方法的映射
关系

```
    initHandlerMapping();  
  
    //测试(启动 Tomcat 装载 HspDispatcherServlet 对象方式来测试)  
    System.out.println("handlerList= " + handlerList);  
}
```

3. 完成测试(启动 tomcat 方式, 修改后, redeploy 即可生效)



添加订单~

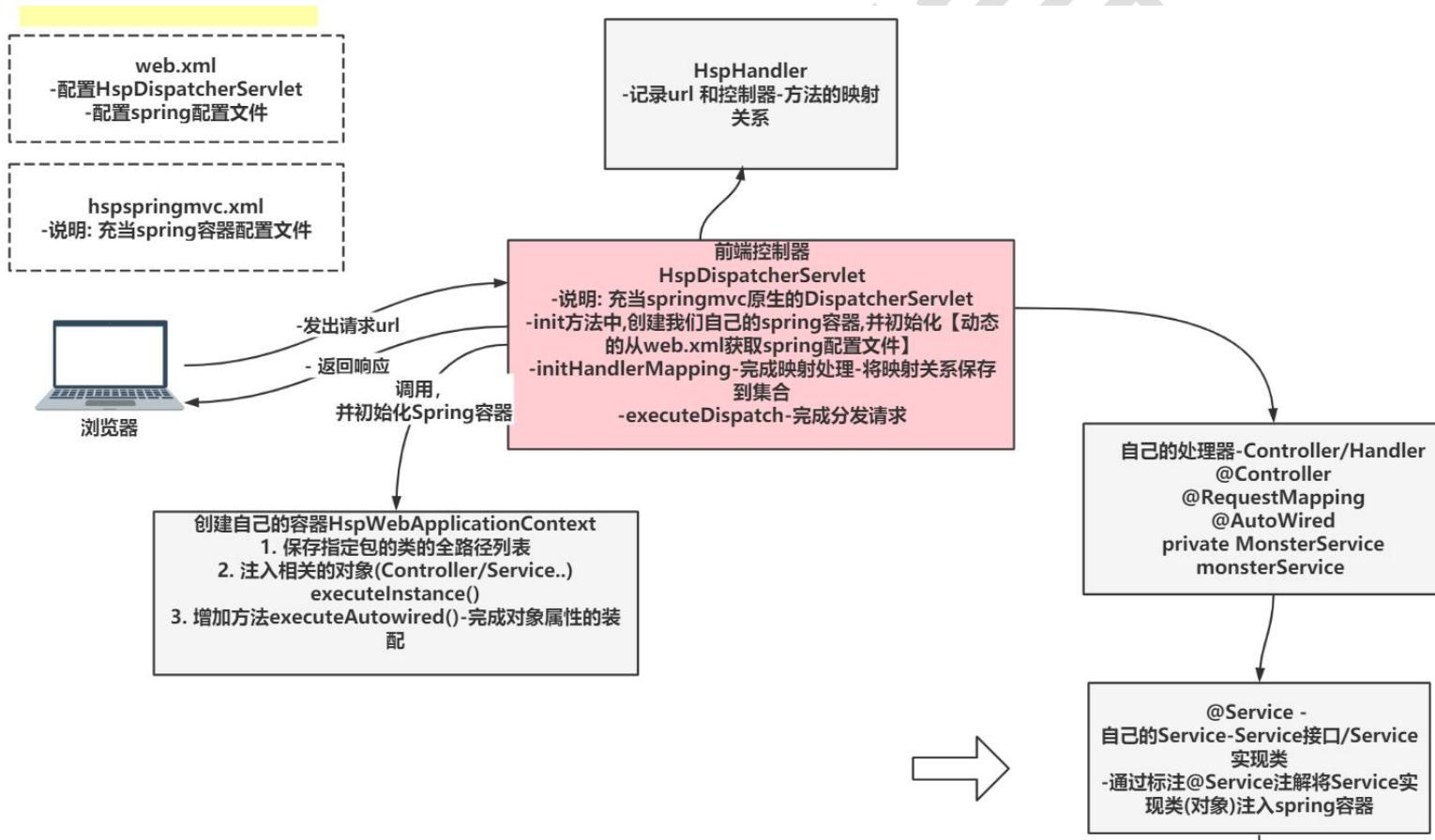
8.2.4 实现任务阶段 4- 完成自定义@Service 注解功能

8.2.4.1 功能说明：如果给某个类加上@Service，则可以将其注入到我们的 Spring 容器

8.2.4.2 分析+代码实现+测试

- 完成功能说明

- 分析示意图



- 给 Service 类标注@Service，可以将对象注入到 Spring 容器中

- 并可以通过接口名支持多级，类名来获取到 Service Bean

- 如图

扫描后的 ioc= {monsterService=com.hspedu.service.impl.MonsterServiceImpl@6323a1c1,
orderController=com.hspedu.controller.OrderController@4c96ec4f,
monsterController=com.hspedu.controller.MonsterController@81664b3}

- 代码实现

1.

D:\idea_java_projects\hsp-springmvc\src\main\java\com\hspdeu\entity\Monster.java

```
package com.hspdeu.entity;

/**
 * @author 韩顺平
 * @version 1.0
 * 也有专门的Lombok 通过注解生成构造器和setter getter 方法，后面用
 */
public class Monster {
    private Integer id;
    private String name;
    private String skill;
```

```
private Integer age;
```

```
public Monster(Integer id, String name, String skill, Integer age) {
```

```
    this.id = id;
```

```
    this.name = name;
```

```
    this.skill = skill;
```

```
    this.age = age;
```

```
}
```

```
public Integer getId() {
```

```
    return id;
```

```
}
```

```
public void setId(Integer id) {
```

```
    this.id = id;
```

```
}
```

```
public String getName() {
```

```
    return name;
```

```
}
```

```
public void setName(String name) {
```

```
this.name = name;  
}  
  
public String getSkill() {  
    return skill;  
}  
  
public void setSkill(String skill) {  
    this.skill = skill;  
}  
  
public Integer getAge() {  
    return age;  
}  
  
public void setAge(Integer age) {  
    this.age = age;  
}  
}
```

2.

创

建

D:\idea_java_projects\hsp-springmvc\src\main\java\com\hspedu\hspspringmvc\annotation\Service.java

```
package com.hspedu.hspspringmvc.annotation;
```

```
import java.lang.annotation.*;
```

```
/**
```

```
* @author 韩顺平
```

```
* @version 1.0
```

```
*/
```

```
@Target(ElementType.TYPE)
```

```
@Retention(RetentionPolicy.RUNTIME)
```

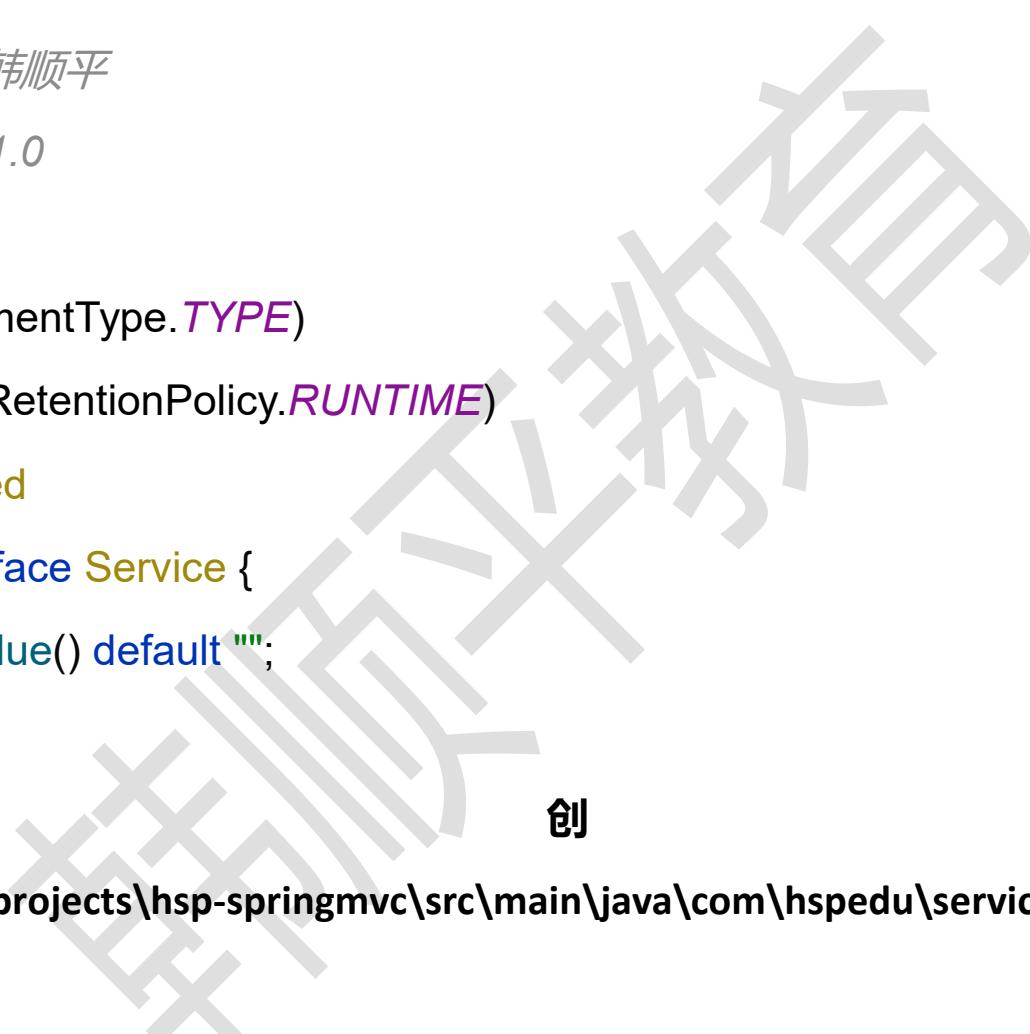
```
@Documented
```

```
public @interface Service {
```

```
    String value() default "";
```

```
}
```

```
3.
```



创

建

```
D:\idea_java_projects\hsp-springmvc\src\main\java\com\hspedu\service\MonsterService.j  
ava
```

```
package com.hspedu.service;
```

```
import com.hspedu.entity.Monster;
```

```
import java.util.List;
```

```
/**  
 * @author 韩顺平  
 * @version 1.0  
 */  
  
public interface MonsterService {  
    public List<Monster> listMonsters();  
}
```

4.

D:\idea_java_projects\hsp-springmvc\src\main\java\com\hspedu\service\impl\MonsterServiceImpl.java

```
package com.hspedu.service.impl;  
  
import com.hspedu.entity.Monster;  
import com.hspedu.hspspringmvc.annotation.Service;  
import com.hspedu.service.MonsterService;
```

```
import java.util.ArrayList;  
import java.util.List;
```

```
/**
```

```
* @author 韩顺平
* @version 1.0
*/
@Service
public class MonsterServiceImpl implements MonsterService {
    @Override
    public List<Monster> listMonsters() {
        ArrayList<Monster> monsters = new ArrayList<>();
        monsters.add(new Monster(100, "牛魔王", "芭蕉扇", 500));
        monsters.add(new Monster(200, "蜘蛛精", "吐口水", 200));
        return monsters;
    }
}
```

5. 修改 D:\idea_java_projects\hsp-springmvc\src\main\resources\hspspringmvc.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<beans>
    <component-scan
        base-package="com.hspedu.controller,com.hspedu.service"></component-scan>
</beans>
```

6.

修

改

D:\idea_java_projects\hsp-springmvc\src\main\java\com\hspedu\hspspringmvc\context\H
spWebApplicationContext.java

//实例化扫描到的类->创建对象->放入到IOC 容器[ConcurrentHashMap]

```
public void executeInstance() {  
    if (classFullPathList.size() == 0) {  
        //抛出一个异常，或者抛出一个自定义异常  
        //throw new RuntimeException("没有需要实例化的对象");  
        //我这里直接 return;  
        return;  
    }  
    try {  
        for (String classFullPath : classFullPathList) {  
            Class<?> clazz = Class.forName(classFullPath);  
            if (clazz.isAnnotationPresent(Controller.class)) {  
                //得到该类的类名(首字母小写)，作为 key  
                String beanName = clazz.getSimpleName().substring(0,  
1).toLowerCase() +  
                    clazz.getSimpleName().substring(1);  
                ioc.put(beanName, clazz.newInstance());  
            } //如果有其它注解，可以在这里扩展  
        }  
    } catch (Exception e) {  
        e.printStackTrace();  
    }  
}
```

```
else if (clazz.isAnnotationPresent(Service.class)) { //如果是@Service注解
```

```
    Service serviceAnnotation = clazz.getAnnotation(Service.class);  
    String beanName = serviceAnnotation.value();  
    if ("".equals(beanName)) { //如果@Service 没有指定 value  
        //得到该 Service 的所有接口名 (首字母小写)  
        //相当于可以通过该类的多个接口名来注入该 Service 实例  
        Class<?>[] interfaces = clazz.getInterfaces();  
        for (Class<?> anInterface : interfaces) {  
            String beanName2 =  
                anInterface.getSimpleName().substring(0, 1).toLowerCase() +  
                anInterface.getSimpleName().substring(1);  
            ioc.put(beanName2, clazz.newInstance());  
        }  
    } else { //如果指定了 value  
        ioc.put(beanName, clazz.newInstance());  
    }  
}  
}  
}  
} catch (Exception e) {  
    e.printStackTrace();  
} finally {
```

```
    }  
}
```

7. 完成测试(启动 Tomcat, 自动加载 HspDispatcherServlet, 完成 IOC 容器的注入)

扫描后的 ioc= {monsterService=com.hspedu.service.impl.MonsterServiceImpl@6323a1c1,
orderController=com.hspedu.controller.OrderController@4c96ec4f,
monsterController=com.hspedu.controller.MonsterController@81664b3}

8.2.5 实现任务阶段 5- 完成 Spring 容器对象的自动装配 -@Autowried

8.2.5.1 说明: 完成 Spring 容器中对象的注入/自动装配

8.2.5.2 分析+代码实现

- 完成任务说明
- 分析示意图
- 加入@AutoWired 注解, 进行对象属性的装配-如图

ioc

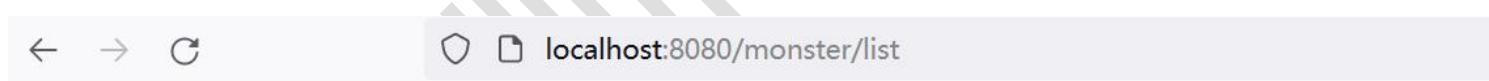
Result:

Not showing null elements

- > 1 = {ConcurrentHashMap\$Node@3000} "goodsController=com.hspedu.controller.xx.GoodsController@1e8
- > 2 = {ConcurrentHashMap\$Node@3001} "monsterService=com.hspedu.service.impl.MonsterServiceImpl@60f
 f hash = 324632098
- > f key = "monsterService"
- > f val = {MonsterServiceImpl@3010}
 - i Class has no fields
 - f next = null
- > 9 = {ConcurrentHashMap\$Node@3002} "orderController=com.hspedu.controller.OrderController@10a287
- > 14 = {ConcurrentHashMap\$Node@3003} "monsterController=com.hspedu.controller.MonsterController@3
 f hash = 1083752334
- > f key = "monsterController"
- > f val = {MonsterController@3009}
 - > f monsterService = {MonsterServiceImpl@3010}
 - i Class has no fields
 - f next = null
- f nextTable = null
- f baseCount = 4



- 浏览器输入 <http://localhost:8080/monster/list>, 返回列表信息



妖怪列表信息

100	牛魔王	芭蕉扇	300
100	老猫精	抓老鼠	500

- 代码实现，老师说明，整个实现思路，就是参考 SpringMVC 规范

1.

创

建

D:\idea_java_projects\hsp-springmvc\src\main\java\com\hspedu\hspspringmvc\annotation\AutoWired.java

```
package com.hspedu.hspspringmvc.annotation;
```

```
import java.lang.annotation.*;
```

```
/**
```

```
* @author 韩顺平
```

```
* @version 1.0
```

```
*/
```

```
@Target(ElementType.FIELD)
```

```
@Retention(RetentionPolicy.RUNTIME)
```

```
@Documented
```

```
public @interface AutoWired {
```

```
    String value() default "";
```

```
}
```

2.

修

改

D:\idea_java_projects\hsp-springmvc\src\main\java\com\hspedu\controller\MonsterController.java

```
@Controller
```

```
public class MonsterController {  
  
    @AutoWired  
    private MonsterService monsterService;  
  
    @RequestMapping(value = "/monster/list")  
    public void listMonsters(HttpServletRequest request, HttpServletResponse response) {  
  
        response.setContentType("text/html;charset=utf-8");  
        try {  
            List<Monster> monsters = monsterService.listMonsters();  
            StringBuilder content = new StringBuilder("<h1>妖怪列表</h1>");  
            content.append("<table width='500px' style='border-collapse:  
collapse' border='1px'>");  
            for (Monster monster : monsters) {  
                content.append("<tr><td>" + monster.getId() + "</td><td>"  
                + monster.getName() + "</td><td>" + monster.getSkill()  
                + "</td>");  
            }  
            content.append("</table>");  
            PrintWriter printWriter = response.getWriter();  
        }  
    }  
}
```

```
        printWriter.write(content.toString());  
    } catch (IOException e) {  
        e.printStackTrace();  
    }  
}  
  
3. 修 改  
D:\idea_java_projects\hsp-springmvc\src\main\java\com\hspsedu\hspspringmvc\context\H  
spWebApplicationContext.java  
, 增加 方法  
  
/**  
 * 初始化自定义的spring 容器, 目标就是把要@Controller 等等初始化到容器中  
 */  
public void init() {  
  
    //后面可以通过 web.xml 灵活的获取, 从 classpath:xxx.xml 中取出 xxx.xml  
    System.out.println("uu= " + contextConfigLocation.split(":")[1]);  
    String basePackage =
```

```
XMLParser.getbasePackage(contextConfigLocation.split(":")[1]);  
  
String[] packages = basePackage.split(",");  
  
if (packages.length > 0) {  
  
    //调用方法对各个包进行扫描,获取到 classFullPathList  
    //      比      如          com.hspedu.service.MonsterService  
    //      com.hspedu.controller.MonsterController  
    //System.out.println("pack= " + pack);  
    scanPackage(pack);  
}  
}  
  
//测试  
System.out.println("扫描后的 classFullPathList= " + classFullPathList);  
executeInstance();  
System.out.println("扫描后的 ioc= " + ioc);  
  
//完成 Spring 容器中对象的自动装配/注入  
executeAutoWired();  
}  
  
//完成 bean 自动注入/装配
```

```
public void executeAutoWired() {  
    if (ioc.isEmpty()) {  
        throw new RuntimeException("容器中，没有可以装配的 bean");  
    }  
    //遍历 ioc 容器中所有，已经实例化的 bean  
    for (Map.Entry<String, Object> entry : ioc.entrySet()) {  
        String key = entry.getKey();  
        Object bean = entry.getValue();  
        Field[] declaredFields = bean.getClass().getDeclaredFields();  
        for (Field declaredField : declaredFields) {  
            if (declaredField.isAnnotationPresent(Autowired.class)) {  
                Autowired autowiredAnnotation =  
                    declaredField.getAnnotation(Autowired.class);  
                String beanName = autowiredAnnotation.value();  
                if ("".equals(beanName)) {  
                    //如果没有设置要注入的 bean 名字，则按  
                    //默认规则  
                    Class<?> type = declaredField.getType();  
                    //获取被标注@AutoWired 字段的类型的名称(首字母小写)  
                    beanName      =      type.getSimpleName().substring(0,  
1).toLowerCase() + type.getSimpleName().substring(1);  
                }  
            }  
        }  
    }  
}
```

```
//放在该@AutoWired 属性是 private  
declaredField.setAccessible(true);  
  
//属性注入  
try {  
    if (ioc.get(beanName) == null) {  
        throw new RuntimeException("ioc 容器中, 没有可以注入  
的 bean");  
    }  
    declaredField.set(bean, ioc.get(beanName));  
} catch (IllegalAccessException e) {  
    e.printStackTrace();  
}  
}  
}  
}
```

8.2.5.3 启动 Tomcat，完成测试

1. 启动 Tomcat
2. 浏览器输入 <http://localhost:8080/monster/list>



妖怪列表

100	牛魔王	芭蕉扇
200	蜘蛛精	吐口水

8.2.6 实现任务阶段 6- 完成控制器方法获取参数-@RequestParam

8.2.6.1 功能说明：自定义@RequestParam 和 方法参数名获取参数

- 完成任务说明

- 前端页面



你找到的妖怪列表

100	牛魔王	芭蕉扇
-----	-----	-----

- 后端 Handler 的目标方法

`@RequestMapping(value = "/monster/find")`

`public void findMonstersByName(HttpServletRequest request,`

HttpServletResponse**response,**

```
@RequestParam(value = "name") String name) {
```

```
//代码....
```

```
}
```

8.2.6.2 完成：将 方法的 HttpServletRequest 和 HttpServletResponse 参数封装到参数数组，进行反射调用

- 代码实现，老师说明，整个实现思路，就是参考 SpringMVC 规范

1.

修

改

```
D:\idea_java_projects\hsp-springmvc\src\main\java\com\hspedu\hspspringmvc\servlet\Hs  
pDispatcherServlet.java
```

```
/**
```

```
* executeDispatch 完成请求的分发处理
```

```
* @param req
```

```
* @param response
```

```
*/
```

```
private void executeDispatch(HttpServletRequest req, HttpServletResponse response)
```

```
{
```

```
HspHandler hspHandler = getHspHandler(req);
```

```
try {
```

```
if (null == hspHandler) {//没有匹配的 Handler
```

```
response.getWriter().print("<h1>404 NOT FOUND</h1>");
```

```
    } else { //有匹配的 Handler, 就调用
        //通过反射得到的参数数组-> 在反射调用方法时会使用到
        //getParameterTypes 或得到当前这个方法的所有参数信息
        Class<?>[] parameterTypes = hspHandler.getMethod().getParameterTypes();
        //定义一个请求的参数集合, 后面在进行反射调用方法时会使用到
        Object[] params = new Object[parameterTypes.length];
        //先搞定 HttpServletRequest 和 HttpServletResponse 这两个参数
        //说明
        //1. 老师这里使用的是名字匹配, 是简单的处理
        //2. 标准的方式可以使用类型匹配, 同学们有兴趣自己试试, 也不难
        for (int i = 0; i < parameterTypes.length; i++) {
            Class<?> parameterType = parameterTypes[i];
            if ("HttpServletRequest".equals(parameterType.getSimpleName())) {
                params[i] = req;
            } else if ("HttpServletResponse".equals(parameterType.getSimpleName())) {
                params[i] = response;
            }
        }
        //hspHandler.getMethod().invoke(hspHandler.getController(), req,
```

```
response);
```

```
    hspHandler.getMethod().invoke(hspHandler.getController(), params);
```

```
}
```

```
} catch (Exception e) {
```

```
    e.printStackTrace();
```

```
}
```

```
}
```

2. 完成测试(启动 tomcat), 浏览器输入 `http://localhost:8080/monster/list`, 仍然可以看到正确的返回



妖怪列表

100	牛魔王	芭蕉扇
200	蜘蛛精	吐口水

8.2.6.3 完成: 在方法参数 指定 @RequestParam 的参数封装到参数数组, 进行反射调用

- 完成任务说明

- 测试页面

A screenshot of a Firefox browser window. The address bar shows 'localhost:8080/monster/find?name=牛魔王'. Below the address bar, there are several bookmarks: '火狐官方站点', '新手上路', '常用网址', and '京东商城'. The main content area has a large title '你找到的妖怪列表'. Below the title is a table with three columns: '100', '牛魔王', and '芭蕉扇'.

100	牛魔王	芭蕉扇
-----	-----	-----

- 后端 Handler 的目标方法

```
@RequestMapping(value = "/monster/find")  
public void findMonstersByName(HttpServletRequest request,  
                                HttpServletResponse response,  
                                @RequestParam(value = "name") String name) {  
    //代码....  
}
```

- 思路分析示意图

- 代码实现，老师说明，整个实现思路，就是参考 SpringMVC 规范

1.

创

建

D:\idea_java_projects\hsp-springmvc\src\main\java\com\hspedu\hspspringmvc\annotation

n\RequestParam.java

```
package com.hspedu.hsspringmvc.annotation;
```

```
import java.lang.annotation.*;
```

```
/**
```

```
* @author 韩顺平
```

```
* @version 1.0
```

```
*/
```

```
@Target(ElementType.PARAMETER)
```

```
@Retention(RetentionPolicy.RUNTIME)
```

```
@Documented
```

```
public @interface RequestParam {
```

```
    String value() default "";
```

```
}
```

2.

修

改

D:\idea_java_projects\hsp-springmvc\src\main\java\com\hspedu\service\MonsterService.j

ava

```
package com.hspedu.service;
```

```
import com.hspedu.entity.Monster;
```

```
import java.util.List;

/**
 * @author 韩顺平
 * @version 1.0
 */

public interface MonsterService {
    public List<Monster> listMonsters();
    public List<Monster> findMonstersByName(String name);
}

3. 修 改
D:\idea_java_projects\hsp-springmvc\src\main\java\com\hspedu\service\impl\MonsterSer
viceImpl.java

package com.hspedu.service.impl;

import com.hspedu.entity.Monster;
import com.hspedu.hspspringmvc.annotation.Service;
import com.hspedu.service.MonsterService;

import java.util.ArrayList;
```

```
import java.util.List;

/**
 * @author 韩顺平
 * @version 1.0
 */

@Service
public class MonsterServiceImpl implements MonsterService {

    @Override
    public List<Monster> listMonsters() {
        ArrayList<Monster> monsters = new ArrayList<>();
        monsters.add(new Monster(100, "牛魔王", "芭蕉扇", 500));
        monsters.add(new Monster(200, "蜘蛛精", "吐口水", 200));
        return monsters;
    }

    @Override
    public List<Monster> findMonstersByName(String name) {
        ArrayList<Monster> monsters = new ArrayList<>();
        monsters.add(new Monster(100, "牛魔王", "芭蕉扇", 500));
        monsters.add(new Monster(200, "蜘蛛精", "吐口水", 200));
        monsters.add(new Monster(300, "白骨精", "吐口水~", 200));
    }
}
```

```
monsters.add(new Monster(400, "青牛怪", "吐口水~~", 200));
```

```
//根据 name 进行一个过滤，老韩在这里简单的模拟
ArrayList<Monster> findmonsters = new ArrayList<>();
for(Monster monster: monsters) {
    if(monster.getName().contains(name)) {
        findmonsters.add(monster);
    }
}
System.out.println(findmonsters.size());
return findmonsters;
}

4. 修 改
D:\idea_java_projects\hsp-springmvc\src\main\java\com\hspedu\controller\MonsterController.java ,增加方法

package com.hspedu.controller;

import com.hspedu.entity.Monster;
import com.hspedu.hspspringmvc.annotation.Autowired;
import com.hspedu.hspspringmvc.annotation.Controller;
import com.hspedu.hspspringmvc.annotation.RequestMapping;
```

```
import com.hspedu.hspsspringmvc.annotation.RequestParam;  
import com.hspedu.service.MonsterService;  
  
import javax.servlet.http.HttpServletRequest;  
import javax.servlet.http.HttpServletResponse;  
import java.io.IOException;  
import java.io.PrintWriter;  
import java.util.List;  
  
/**  
 * @author 韩顺平  
 * @version 1.0  
 * 控制器  
 */  
@Controller  
public class MonsterController {  
  
    @AutoWired  
    private MonsterService monsterService;  
  
    @RequestMapping(value = "/monster/find")  
    public void findMonstersByName(HttpServletRequest request,
```

HttpServletResponse

response,

```
@RequestParam(value = "name") String name) {  
  
    response.setContentType("text/html;charset=utf-8");  
  
    try {  
        System.out.println("接收到 name= " + name);  
        List<Monster> monsters =  
            monsterService.findMonstersByName(name);  
        StringBuilder content = new StringBuilder("<h1>你找到的妖怪列表  
</h1>");  
        content.append("<table width='500px' style='border-collapse:  
collapse' border='1px'>");  
        for (Monster monster : monsters) {  
            content.append("<tr><td>" + monster.getId() + "</td><td>"  
                + monster.getName() + "</td><td>" + monster.getSkill()  
                + "</td>");  
        }  
        content.append("</table>");  
        PrintWriter printWriter = response.getWriter();  
        printWriter.write(content.toString());  
    } catch (Exception e) {  
        e.printStackTrace();  
    }  
}
```

```
        } catch (IOException e) {  
            e.printStackTrace();  
        }  
    }  
}
```

5. **D:\idea_java_projects\hsp-springmvc\src\main\java\com\hspedu\hspspringmvc\servlet\Hs
pDispatcherServlet.java 增加方法**

```
package com.hspedu.hspspringmvc.servlet;  
  
/**  
 * @author 韩顺平  
 * @version 1.0  
 */  
public class HspDispatcherServlet extends HttpServlet {  
  
    /**  
     * executeDispatch 完成请求的分发处理  
     */
```

```
* @param req  
* @param response  
*/  
  
private void executeDispatch(HttpServletRequest req, HttpServletResponse  
response) {  
    HspHandler hspHandler = getHspHandler(req);  
    try {  
        if (null == hspHandler) {//没有匹配的 Handler  
            response.getWriter().print("<h1>404 NOT FOUND</h1>");  
        } else {//有匹配的 Handler, 就调用  
            //通过反射得到的参数数组-> 在反射调用方法时会使用到  
            //getParameterTypes 或得到当前这个方法的所有参数信息  
            Class<?>[] parameterTypes =  
                hspHandler.getMethod().getParameterTypes();  
            //定义一个请求的参数集合, 后面在进行反射调用方法时会使用到  
            Object[] params = new Object[parameterTypes.length];  
            //先搞定 HttpServletRequest 和 HttpServletResponse 这两个参数  
            //说明  
            //1. 老师这里使用的是名字匹配, 是简单的处理  
            //2. 标准的方式可以使用类型匹配, 同学们有兴趣自己试试, 也不难  
            for (int i = 0; i < parameterTypes.length; i++) {  
                Class<?> parameterType = parameterTypes[i];  
            }  
        }  
    } catch (Exception e) {  
        e.printStackTrace();  
    }  
}
```

```
if ("HttpServletRequest".equals(parameterType.get SimpleName())) {  
    params[i] = req;  
}  
else if ("HttpServletResponse".equals(parameterType.get SimpleName())) {  
    params[i] = response;  
}  
}  
  
// 再 搞 定 用户 请求 时 带 的 参 数 , 比 如  
http://localhost:8080/monster/list?name=xx&age=10  
//这里的 name=xx&age=10  
//获取请求中的参数集合  
Map<String, String[]> parameterMap = req.getParameterMap();  
for (Map.Entry<String, String[]> entry : parameterMap.entrySet())  
{  
    //目前我们就考虑取出一个, 不考虑 checkbox 的一组数据情况  
    String name = entry.getKey();  
    String value = entry.getValue()[0];  
    //这里可先测试一下, 给学员看看效果  
    System.out.println("请求参数: " + name + "----" + value);  
    //1. 去得到这个参数在被调用方法的参数的第几个位置(也就是
```

index)

//2. 我们专门编写一个方法 getIndexRequestParamIndex 来玩

int indexRequestParamIndex =

getIndexRequestParamIndex(hspHandler.getMethod(), name);

if(indexRequestParamIndex != -1) { // 找到了，就加入到
params 数组中。

params[indexRequestParamIndex] = value;

} else {

//如果没有找到，我们就按照默认的参数名的匹配规则来做

[一会完成]

}

}

//hspHandler.getMethod().invoke(hspHandler.getController(), req,
response);

hspHandler.getMethod().invoke(hspHandler.getController(), params);

}

} catch (Exception e) {

e.printStackTrace();

}

```
}
```

```
/**
```

```
* 得到@RequestParam 注解的参数, 在方法的 index
```

```
* 如果返回-1, 表示没有
```

```
* @param method
```

```
* @param name
```

```
* @return
```

```
*/
```

```
public int getIndexRequestParamIndex(Method method, String name) {
```

```
    Parameter[] parameters = method.getParameters();
```

```
    for (int i = 0; i < parameters.length; i++) {
```

```
        Parameter parameter = parameters[i];
```

```
        boolean annotationPresent =
```

```
        parameter.isAnnotationPresent(RequestParam.class);
```

```
        if(annotationPresent) {//如果存在
```

```
            RequestParam requestParamAnnotation =
```

```
            parameter.getAnnotation(RequestParam.class);
```

```
            String value = requestParamAnnotation.value();
```

```
            if(name.equals(value)) {
```

```
                return i;//找到返回
```

```
}
```

```
    }  
}  
//没有找到就返回-1  
return -1;  
}  
}
```

6. 完成 测试 (Redeploy Tomcat 即可) , 浏览器输入

<http://localhost:8080/monster/find?name=%E7%89%9B%E9%AD%94%E7%8E%8B>



你找到的妖怪列表

100	牛魔王	芭蕉扇
-----	-----	-----

8.2.6.4 完成：在方法参数 没有指定 @RequestParam ，按照默认参数名获取值，进行反射调用

- 完成任务说明

- 前端页面



- 后端 Handler 的目标方法

```
@RequestMapping(value = "/monster/find")
public void findMonstersByName(HttpServletRequest request,
                                HttpServletResponse response,
/*@RequestParam(value = "name")*/ String name) {
    //代码....
}
```

- 思路分析示意图

- 代码实现，老师说明，整个实现思路，就是参考 SpringMVC 规范

1.

修

改

D:\idea_java_projects\hsp-springmvc\src\main\java\com\hspedu\controller\MonsterController.java

```
@RequestMapping(value = "/monster/find")
public void findMonstersByName(HttpServletRequest request,
                                HttpServletResponse response,
                                /*@RequestParam(value = "name")*/ String name) {
    response.setContentType("text/html;charset=utf-8");
    try {
        System.out.println("接收到 name= " + name);
        if(name == null) { //如果没有匹配到，设置为""
            name = "";
        }
        List<Monster> monsters = monsterService.findMonstersByName(name);
        StringBuilder content = new StringBuilder("<h1>你找到的妖怪列表</h1>");
        content.append("<table width='500px' style='border-collapse: collapse' border='1px'>");
        for (Monster monster : monsters) {
            content.append("<tr><td>" + monster.getId() + "</td><td>" +
                           monster.getName() + "</td><td>" + monster.getSkill() + "
```

```
"</td>");  
  
}  
  
content.append("</table>");  
  
PrintWriter printWriter = response.getWriter();  
  
printWriter.write(content.toString());  
} catch (IOException e) {  
  
e.printStackTrace();  
}  
  
}  
  
2. 修改  
D:\idea_java_projects\hsp-springmvc\src\main\java\com\hspedu\hspspringmvc\servlet\Hs  
pDispatcherServlet.java  
  
package com.hspedu.hspspringmvc.servlet;  
  
/**  
 * @author 韩顺平  
 * @version 1.0  
 */
```

```
public class HspDispatcherServlet extends HttpServlet {  
  
    /**  
     * executeDispatch 完成请求的分发处理  
     * @param req  
     * @param response  
     */  
  
    private void executeDispatch(HttpServletRequest req, HttpServletResponse  
response) {  
        HspHandler hspHandler = getHspHandler(req);  
        try {  
            if (null == hspHandler) {//没有匹配的 Handler  
                response.getWriter().print("<h1>404 NOT FOUND</h1>");  
            } else {//有匹配的 Handler, 就调用  
                //通过反射得到的参数数组-> 在反射调用方法时会使用到  
                //getParameterTypes 或得到当前这个方法的所有参数信息  
                Class<?>[] parameterTypes =  
                    hspHandler.getMethod().getParameterTypes();  
                //定义一个请求的参数集合, 后面在进行反射调用方法时会使用到  
                Object[] params = new Object[parameterTypes.length];  
                //先搞定 HttpServletRequest 和 HttpServletResponse 这两个参数  
                //说明
```

//1. 老师这里使用的是名字匹配，是简单的处理
//2. 标准的方式可以使用类型匹配，同学们有兴趣自己试试，也不难

```
for (int i = 0; i < parameterTypes.length; i++) {  
    Class<?> parameterType = parameterTypes[i];  
  
    if ("HttpServletRequest".equals(parameterType.getSimpleName())) {  
        params[i] = req;  
    }  
    else if ("HttpServletResponse".equals(parameterType.getSimpleName())) {  
        params[i] = response;  
    }  
}
```

// 再搞定用户请求时带的参数，比如
<http://localhost:8080/monster/list?name=xx&age=10>

```
//这里的 name=xx&age=10  
//获取请求中的参数集合  
Map<String, String[]> parameterMap = req.getParameterMap();  
  
for (Map.Entry<String, String[]> entry : parameterMap.entrySet()) {  
    //目前我们就考虑取出一个，不考虑checkbox 的一组数据情况  
    String name = entry.getKey();  
    String value = entry.getValue()[0];
```

```
//这里可先测试一下，给学员看看效果
System.out.println("请求参数: " + name + "----" + value);
//1. 去得到这个参数在被调用方法的参数的第几个位置(也就是
index)

//2. 我们专门编写一个方法 getIndexRequestParamIndex 来玩
int indexRequestParamIndex = 
getIndexRequestParamIndex(hspHandler.getMethod(), name);

if (indexRequestParamIndex != -1) {//找到了，就加入到 params
数组中.

    params[indexRequestParamIndex] = value;
} else {
    //如果没有找到，我们就按照默认的参数名的匹配规则来做
[一会完成]

//编写 getParameterNames() 方法获取到该方法的所有参
数名
List<String> parameterNames =
getParameterNames(hspHandler.getMethod());

for (int i = 0; i < parameterNames.size(); i++) {

    //如果请求参数和方法参数名一致，就匹配到
    if (name.equals(parameterNames.get(i))) {
```

```
        params[i] = value;
    break;
}
}

}

//hspHandler.getMethod().invoke(hspHandler.getController(), req,
response);

hspHandler.getMethod().invoke(hspHandler.getController(), params);

}

} catch (Exception e) {
    e.printStackTrace();
}

}

/**
 * 得到控制器方法的参数名，比如 public void
findMonstersByName(HttpServletRequest request,
    * HttpServletResponse response, @RequestParam(value = "name") String
name)
```

- * *request, response, name*
- * 注意：
 - * 1. 在默认情况下，返回的并不是 *request, response ,name* 而是 *arg0, arg1, arg2*
 - * 2. 需要使用到 jdk8 的新特性，并需要在 *pom.xml* 配置 maven 编译插件(可以在百度搜索到)，才能得到 *request, response, name*
- * <plugin>
 - * <groupId>org.apache.maven.plugins</groupId>
 - * <artifactId>maven-compiler-plugin</artifactId>
 - * <version>3.7.0</version>
 - * <configuration>
 - * <source>1.8</source>
 - * <target>1.8</target>
 - * <compilerArgs>
 - * <arg>-parameters</arg>
 - * </compilerArgs>
 - * <encoding>utf-8</encoding>
 - * </configuration>
- * </plugin>
- * 3.
- * *@param method*
- * *@return*

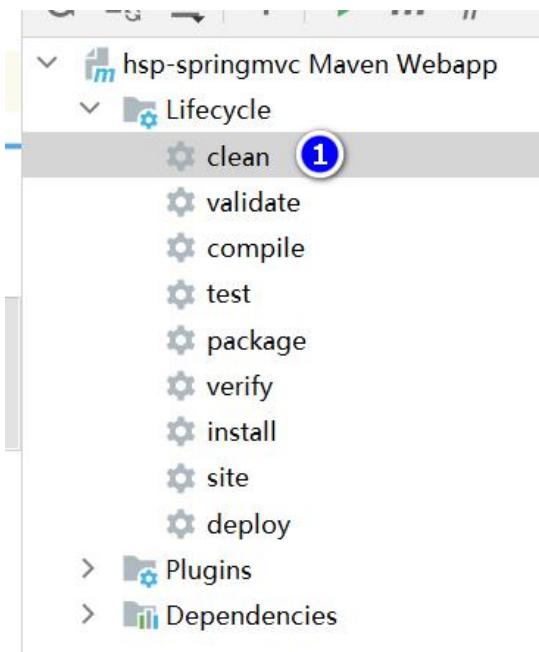
```
*/  
  
public List<String> getParameterNames(Method method) {  
    ArrayList<String> lists = new ArrayList<>();  
    Parameter[] parameters = method.getParameters();  
    for (Parameter parameter : parameters) {  
        lists.add(parameter.getName());  
    }  
    System.out.println("方法参数名列表= " + lists);  
    return lists;  
}  
}
```

3. 修改 D:\idea_java_projects\hsp-springmvc\pom.xml，保证和我给的版本一致

```
<plugin>  
    <groupId>org.apache.maven.plugins</groupId>  
    <artifactId>maven-compiler-plugin</artifactId>  
    <version>3.7.0</version>  
    <configuration>  
        <source>1.8</source>  
        <target>1.8</target>  
        <compilerArgs>
```

```
<arg>-parameters</arg>  
</compilerArgs>  
<encoding>utf-8</encoding>  
</configuration>  
</plugin>
```

4. 点击 maven 管理，clean 项目，在重启一下 tomcat ,完成测试



浏览器 输入: <http://localhost:8080/monster/find?name=%E7%99%BD%E9%AA%A8%E7%B2%BE>



后台：

请求参数： name----牛

方法参数名列表= [request, response, name]

接收到name= 牛

2

如果请求参数和方法参数不一致：



100	牛魔王	芭蕉扇
200	蜘蛛精	吐口水
300	白骨精	吐口水~
400	青牛怪	吐口水~~

后台

请求参数： uname----白骨精

方法参数名列表= [request, response, name]

接收到name= **null**

8.2.7 实现任务阶段 7- 完成简单视图解析

8.2.7.1 功能说明：通过方法返回的 String，转发或者重定向到指定页面

8.2.7.2 分析+代码

- 完成任务说明

- 用户输入**白骨精**,可以登录成功, 否则失败

- 根据登录的结果, 可以重定向或者请求转发到 login_ok.jsp / login_error.jsp, 并显示妖

怪名.

- 测试页面



妖怪登录

妖怪名: 白骨精

登录



登录成功

欢迎你- | 白骨精



登录失败

sorry, 登录失败- 黄袍怪

- 思路分析示意图

- 代码实现，老师说明，整个实现思路，就是参考 SpringMVC 规范

1.

修

改

D:\idea_java_projects\hsp-springmvc\src\main\java\com\hspedu\service\MonsterService.java

```
public interface MonsterService {
```

```
    public List<Monster> listMonsters();
```

```
    public List<Monster> findMonstersByName(String name);
```

```
    public boolean login(String name);
```

```
}
```

2.

修

改

D:\idea_java_projects\hsp-springmvc\src\main\java\com\hspedu\service\impl\MonsterSer

vicImpl.java

```
package com.hspedu.service.impl;

import com.hspdeu.entity.Monster;
import com.hspedu.hspspringmvc.annotation.Service;
import com.hspedu.service.MonsterService;

import java.util.ArrayList;
import java.util.List;

/**
 * @author 韩顺平
 * @version 1.0
 */
@Service
public class MonsterServiceImpl implements MonsterService {

    @Override
    public boolean login(String name) {
        if ("白骨精".equals(name)) {
            return true;
        } else {
            return false;
        }
    }
}
```

```
        return false;  
    }  
}  
}
```

3. 修 改

D:\idea_java_projects\hsp-springmvc\src\main\java\com\hspedu\controller\MonsterController.java ,增加方法

```
@RequestMapping(value = "/monster/login")  
public String login(HttpServletRequest request,  
                     HttpServletResponse response, String mName) {  
    boolean b = monsterService.login(mName);  
    System.out.println("mName= " + mName);  
    //将数据放入 request 域  
    request.setAttribute("mName", mName);  
    if (b) {  
        //重定向  
        //return "redirect:/loginOk.jsp";  
        //默认是转发  
        //return "loginOk.jsp";  
        //转发  
        return "forward:/login_ok.jsp";  
    } else {
```

```
        return "forward:/login_error.jsp";  
    }  
}
```

4. 增加 D:\idea_java_projects\hsp-springmvc\src\main\webapp\login_ok.jsp

```
<%--  
Created by IntelliJ IDEA.  
User: 韩顺平  
Version: 1.0  
Filename: login_ok  
--%>  
<%@ page contentType="text/html;charset=UTF-8" language="java"  
isELIgnored="false" %>  
<html>  
<head>  
    <title>登录成功</title>  
</head>  
<body>  
    <h1>欢迎: ${requestScope.mName}</h1>  
</body>  
</html>
```

5. 创建 D:\idea_java_projects\hsp-springmvc\src\main\webapp\login_error.jsp

```
<%--
```

Created by IntelliJ IDEA.

User: 韩顺平

Version: 1.0

Filename: login_error

```
--%>
```

```
<%@ page contentType="text/html;charset=UTF-8" language="java"
isELIgnored="false" %>

<html>
<head>
    <title>登录失败</title>
</head>
<body>
<h1>sorry 登录失败: ${requestScope.mName}</h1>
</body>
</html>
```

6.

修

改

D:\idea_java_projects\hsp-springmvc\src\main\java\com\hspedu\hspspringmvc\servlet\Hs
pDispatcherServlet.java

```
/**
```

* executeDispatch 完成请求的分发处理

```
* @param req
* @param response
*/
private void executeDispatch(HttpServletRequest req, HttpServletResponse response)
{
    HspHandler hspHandler = getHspHandler(req);
    try {
        if (null == hspHandler) {//没有匹配的 Handler
            response.getWriter().print("<h1>404 NOT FOUND</h1>");
        } else {//有匹配的 Handler, 就调用
            //通过反射得到的参数数组-> 在反射调用方法时会使用到
            //getParameterTypes 或得到当前这个方法的所有参数信息
            Class<?>[] parameterTypes = hspHandler.getMethod().getParameterTypes();
            //定义一个请求的参数集合, 后面在进行反射调用方法时会使用到
            Object[] params = new Object[parameterTypes.length];
            //先搞定 HttpServletRequest 和 HttpServletResponse 这两个参数
            //说明
            //1. 老师这里使用的是名字匹配, 是简单的处理
            //2. 标准的方式可以使用类型匹配, 同学们有兴趣自己试试, 也不难
            for (int i = 0; i < parameterTypes.length; i++) {
                Class<?> parameterType = parameterTypes[i];
```

```
if ("HttpServletRequest".equals(parameterType.get SimpleName())) {  
    params[i] = req;  
}  
else  
if  
("HttpServletResponse".equals(parameterType.get SimpleName())) {  
    params[i] = response;  
}  
}  
}
```

// 再 搞 定 用 户 请 求 时 带 的 参 数 , 比 如
http://localhost:8080/monster/list?name=xx&age=10

```
//这里的 name=xx&age=10  
//获取请求中的参数集合  
Map<String, String[]> parameterMap = req.getParameterMap();  
for (Map.Entry<String, String[]> entry : parameterMap.entrySet()) {  
    //目前我们就考虑取出一个, 不考虑 checkbox 的一组数据情况  
    String name = entry.getKey();  
    String value = entry.getValue()[0];  
    //这里可先测试一下, 给学员看看效果  
    System.out.println("请求参数: " + name + "----" + value);  
    //1. 去得到这个参数在被调用方法的参数的第几个位置(也就是 index)  
    //2. 我们专门编写一个方法 getRequestParamIndex 来玩  
    int indexRequestParamIndex =
```

```
getIndexRequestParamIndex(hspHandler.getMethod(), name);

    if (indexRequestParamIndex != -1) { //找到了，就加入到params数组中。

        params[indexRequestParamIndex] = value;
    } else {

        //如果没有找到，我们就按照默认的参数名的匹配规则来做[一会完成]

        //编写getParameterNames() 方法获取到该方法的所有参数名
        List<String> parameterNames = getParameterNames(hspHandler.getMethod());

        for (int i = 0; i < parameterNames.size(); i++) {

            if (name.equals(parameterNames.get(i))) { //如果请求参数和方法参数名一致，就匹配到

                params[i] = value;
                break;
            }
        }
    }

    //hspHandler.getMethod().invoke(hspHandler.getController(), req,
    response);
```

Object

result

=

```
hspHandler.getMethod().invoke(hspHandler.getController(), params);

//对结果，执行一个视图解析(这里我们简单处理，帮助大家理解机制)

if (result instanceof String) {

    String viewName = (String) result;

    if (viewName.contains(":")) {//说明是 forward:/xx.xx 或者
        redirect:/xx.xx

        String viewType = viewName.split(":")[0];
        String viewPage = viewName.split(":")[1];
        if ("forward".equals(viewType)) {//如果是转发
            req.getRequestDispatcher(viewPage).forward(req,
response);

        } else if ("redirect".equals(viewType)) {//如果是重定向
            response.sendRedirect(viewPage);
        }
    } else {//默认是转发
        req.getRequestDispatcher(viewName).forward(req,
response);
    }
}

} catch (Exception e) {
```

```
        e.printStackTrace();  
    }  
}
```

8.2.7.3 完成测试

1 启动 tomcat



3. 浏览器输入 `http://localhost:8080/monster/login?mName=abc`



8.2.8 实现任务阶段 8- 完成返回 JSON 格式数据-@ResponseBody

8.2.8.1 功能说明：通过自定义@ResponseBody 返回 JSON 格式数据

8.2.8.2 分析+代码实现

- 分析示意图
- 代码实现，老师说明，整个实现思路，就是参考 SpringMVC 规范

1.

创

建

D:\idea_java_projects\hsp-springmvc\src\main\java\com\hspedu\hspspringmvc\annotation\ResponseBody.java

```
package com.hspedu.hspspringmvc.annotation;
```

```
import java.lang.annotation.*;
```

```
/**
```

```
* @author 韩顺平
```

```
* @version 1.0
```

```
*/
```

```
@Target(ElementType.METHOD)
```

```
@Retention(RetentionPolicy.RUNTIME)
```

```
@Documented
```

```
public @interface ResponseBody {
```

```
}
```

2.

创

建

D:\idea_java_projects\hsp-springmvc\src\main\java\com\hspedu\controller\MonsterController.java, 增加方法

```
@RequestMapping(value = "/monster/list/json")
```

```
@ResponseBody
```

```
public List<Monster> listMonstersByJson(HttpServletRequest request,  
HttpServletResponse response) {
```

```
    List<Monster> monsters = monsterService.listMonsters();
```

```
    return monsters;
```

```
}
```

3. 修改 D:\idea_java_projects\hsp-springmvc\pom.xml

```
<!-- 引入 jackson-->
```

```
<dependency>
```

```
    <groupId>com.fasterxml.jackson.core</groupId>
```

```
    <artifactId>jackson-databind</artifactId>
```

```
    <version>2.12.4</version>
```

</dependency>

4.

修

改

D:\idea_java_projects\hsp-springmvc\src\main\java\com\hspedu\hspspringmvc\servlet\Hs
pDispatcherServlet.java

```
/*
 * executeDispatch 完成请求的分发处理
 * @param req
 * @param response
 */
private void executeDispatch(HttpServletRequest req, HttpServletResponse response)
{
    HspHandler hspHandler = getHspHandler(req);
    try {
        if (null == hspHandler) {//没有匹配的 Handler
            response.getWriter().print("<h1>404 NOT FOUND</h1>");
        } else {//有匹配的 Handler, 就调用
            //通过反射得到的参数数组-> 在反射调用方法时会使用到
            //getParameterTypes 或得到当前这个方法的所有参数信息
            Class<?>[] parameterTypes =
                hspHandler.getMethod().getParameterTypes();
            //定义一个请求的参数集合, 后面在进行反射调用方法时会使用到
        }
    }
}
```

```
Object[] params = new Object[parameterTypes.length];  
//先搞定 HttpServletRequest 和 HttpServletResponse 这两个参数  
//说明  
//1. 老师这里使用的是名字匹配，是简单的处理  
//2. 标准的方式可以使用类型匹配，同学们有兴趣自己试试，也不难  
for (int i = 0; i < parameterTypes.length; i++) {  
    Class<?> parameterType = parameterTypes[i];  
    if ("HttpServletRequest".equals(parameterType.getSimpleName())) {  
        params[i] = req;  
    }  
    else if ("HttpServletResponse".equals(parameterType.getSimpleName())) {  
        params[i] = response;  
    }  
}  
}  
  
// 再 搞 定 用 户 请 求 时 带 的 参 数 ， 比 如  
http://localhost:8080/monster/list?name=xx&age=10  
//这里的 name=xx&age=10  
//获取请求中的参数集合  
Map<String, String[]> parameterMap = req.getParameterMap();  
for (Map.Entry<String, String[]> entry : parameterMap.entrySet()) {  
    //目前我们就考虑取出一个，不考虑checkbox的一组数据情况
```

```
String name = entry.getKey();
String value = entry.getValue()[0];
//这里可先测试一下，给学员看看效果
System.out.println("请求参数: " + name + "----" + value);
//1. 去得到这个参数在被调用方法的参数的第几个位置(也就是index)
//2. 我们专门编写一个方法getRequestParamIndex 来玩
int indexRequestParamIndex = getIndexRequestParamIndex(hspHandler.getMethod(), name);
if (indexRequestParamIndex != -1) {//找到了，就加入到params 数组中。
    params[indexRequestParamIndex] = value;
} else {
    //如果没有找到，我们就按照默认的参数名的匹配规则来做[一会完成]
    //编写getParameterNames() 方法获取到该方法的所有参数名
    List<String> parameterNames = getParameterNames(hspHandler.getMethod());
    for (int i = 0; i < parameterNames.size(); i++) {
        if (name.equals(parameterNames.get(i))) {//如果请求参数和方法参数名一致，就匹配到
            params[i] = value;
            break;
        }
    }
}
```

```
        }

    }

}

//hspHandler.getMethod().invoke(hspHandler.getController(),      req,
response);

Object result = hspHandler.getMethod().invoke(hspHandler.getController(), params);

//对结果，执行一个视图解析(这里我们简单处理，帮助大家理解机制)

if (result instanceof String) {

    String viewName = (String) result;

    if (viewName.contains(":")) {// 说明是 forward:/xx.xx 或者
redirect:/xx.xx

        String viewType = viewName.split(":")[0];

        String viewPage = viewName.split(":")[1];

        if ("forward".equals(viewType)) {//如果是转发
            req.getRequestDispatcher(viewPage).forward(req,
response);

        } else if ("redirect".equals(viewType)) {//如果是重定向
            response.sendRedirect(viewPage);
        }
    }
}
```

```
        } else {  
            //默认是转发  
            req.getRequestDispatcher(viewName).forward(req, response);  
        }  
    } else if (result instanceof ArrayList) {  
        //返回 json 数据  
        Method method = hspHandler.getMethod();  
        if (method.isAnnotationPresent(ResponseBody.class)) {  
            //把返回值转成 json 字符串-使用工具类 jackson  
            ObjectMapper objectMapper = new ObjectMapper();  
            String json = objectMapper.writeValueAsString(result);  
            response.setContentType("text/html;charset=utf-8");  
            PrintWriter writer = response.getWriter();  
            writer.print(json);  
            writer.flush();  
            writer.close();  
        }  
    } catch (Exception e) {  
        e.printStackTrace();  
    }
```

{}

8.2.8.3 完成测试

1. 启动 Tomcat

2. 浏览器输入 `http://localhost:8080/monster/list/json`

The screenshot shows a browser window with the URL `localhost:8080/monster/list/json` in the address bar. The page content displays the following JSON array:

```
[{"id":100,"name":"牛魔王","skill":"芭蕉扇","age":500}, {"id":200, "name": "蜘蛛精", "skill": "吐口水", "age": 200}, {"id":300, "name": "蜘蛛精2", "skill": "吐口水2", "age": 1200}]
```

Below the browser is a screenshot of a network traffic analysis tool (like Fiddler or NetworkMiner) showing two requests:

状态	方法	域名	文件	发起者	类型	传输	耗时	消息头	Cookie	请求	响应	耗时
200	GET	localhost...	json	docu...	httr	334 ...	18	过滤属性				
200	GET	localhost...	favicon.ico	img	httr	125 ...	22	JSON				

The "响应" tab is selected, showing the JSON response for the second request (id: 300). The response is displayed as:

```
0: Object { id: 100, name: "牛魔王", skill: "芭蕉扇", ... }  
    id: 100  
    name: "牛魔王"  
    skill: "芭蕉扇"  
    age: 500  
1: Object { id: 200, name: "蜘蛛精", skill: "吐口水", ... }  
    id: 200  
    name: "蜘蛛精"  
    skill: "吐口水"  
    age: 200  
2: Object { id: 300, name: "蜘蛛精2", skill: "吐口水2", ... }  
    id: 300  
    name: "蜘蛛精2"  
    skill: "吐口水2"  
    age: 1200
```

8.3 课后作业题

8.3.3 不看老师代码，完成手动实现 Spring MVC 底层机制

9 数据格式化

9.1 基本介绍

说明：在我们提交数据(比如表单时)SpringMVC 怎样对提交的数据进行转换和处理的

1. 基本数据类型可以和字符串之间自动完成转换，比如：

Spring MVC 上下文中内建了很多转换器，可完成大多数 Java 类型的转换工作。[相互转换，这里只列出部分]

2. ConversionService converters =

java.lang.Boolean -> java.lang.String :

org.springframework.core.convert.support.ObjectToStringConverter@f874ca

java.lang.Character -> java.lang.Number : CharacterToNumberFactory@f004c9

java.lang.Character -> java.lang.String : ObjectToStringConverter@68a961

java.lang.Enum -> java.lang.String : EnumToStringConverter@12f060a

java.lang.Number -> java.lang.Character : NumberToCharacterConverter@1482ac5

java.lang.Number -> java.lang.Number : NumberToNumberConverterFactory@126c6f

java.lang.Number -> java.lang.String : ObjectToStringConverter@14888e8

java.lang.String -> java.lang.Boolean : StringToBooleanConverter@1ca6626

java.lang.String -> java.lang.Character : StringToCharacterConverter@1143800

java.lang.String -> java.lang.Enum : StringToEnumConverterFactory@1bba86e

java.lang.String -> java.lang.Number : StringToNumberConverterFactory@18d2c12

java.lang.String -> java.util.Locale : StringToLocaleConverter@3598e1

java.lang.String -> java.util.Properties : StringToPropertiesConverter@c90828

java.lang.String -> java.util.UUID : StringToUUIDConverter@a42f23

java.util.Locale -> java.lang.String : ObjectToStringConverter@c7e20a

java.util.Properties -> java.lang.String : PropertiesToStringConverter@367a7f

java.util.UUID -> java.lang.String : ObjectToStringConverter@112b07f
.....

9.2 基本数据类型和字符串自动转换

9.2.1 应用实例 - 页面演示方式

- 说明：基本数据类型可以和字符串之间自动完成转换

SpringMVC[数据格式/验证等]

[添加妖怪](#)



添加妖怪~~

妖怪名字:

妖怪年龄~:

妖怪生日: 要求以"9999-11-11"的形式

妖怪工资: 0 要求以"123,890.12"的形式

电子邮件:

• 代码实现

1.

创

建

JavaBean:

D:\idea_java_projects\springmvc\src\com\hspedu\web\datavalid\Monster.java

```
package com.hspedu.web.datavalid;
```

```
import org.hibernate.validator.constraints.NotEmpty;
```

```
import org.hibernate.validator.constraints.Range;
```

```
import org.springframework.format.annotation.DateTimeFormat;  
import org.springframework.format.annotation.NumberFormat;
```

```
import java.util.Date;
```

```
/**
```

```
* @author 韩顺平
```

```
* @version 1.0
```

```
*/
```

```
public class Monster {
```

```
    private Integer id;
```

```
    private String email;
```

```
    private Integer age;
```

```
    private String name;
```

```
    public Integer getAge() {
```

```
        return age;
```

```
}
```

```
    public void setAge(Integer age) {
```

```
        this.age = age;
```

```
}
```

```
public Monster() {
```

```
}
```

```
public Monster(Integer id, String email, Integer age, String name) {
```

```
    this.id = id;
```

```
    this.email = email;
```

```
    this.age = age;
```

```
    this.name = name;
```

```
}
```

```
public Integer getId() {
```

```
    return id;
```

```
}
```

```
public void setId(Integer id) {
```

```
    this.id = id;
```

```
}
```

```
public String getName() {
```

```
        return name;  
    }  
  
    public void setName(String name) {  
        this.name = name;  
    }  
  
    public String getEmail() {  
        return email;  
    }  
  
    public void setEmail(String email) {  
        this.email = email;  
    }  
  
    @Override  
    public String toString() {  
        return "Monster{" +  
               "id=" + id +  
               ", email='" + email + '\'' +  
               ", age=" + age +  
               ", name='" + name + '\'' +
```

```
'};  
}  
}
```

2. 创建 D:\idea_java_projects\springmvc\web\data_valid.jsp

```
<%--  
     Created by IntelliJ IDEA.  
     User: 韩顺平  
     Version: 1.0  
     Filename: monster_manage  
--%>  
<%@ page contentType="text/html;charset=UTF-8" language="java" %>  
<html>  
<head>  
    <title>SpringMVC[数据格式/验证等]</title>  
</head>  
<body>  
    <h1>SpringMVC[数据格式/验证等]</h1>  
    <hr>  
    <a href="addMonsterUI">添加妖怪</a>  
</body>
```

```
</body>
```

```
</html>
```

3.

创

建

D:\idea_java_projects\springmvc\src\com\hspedu\web\datavalid\MonsterHandler.java

```
package com.hspedu.web.datavalid;

import org.springframework.context.annotation.Scope;
import org.springframework.stereotype.Controller;
import org.springframework.validation.Errors;
import org.springframework.validation.ObjectError;
import org.springframework.web.bind.WebDataBinder;
import org.springframework.web.bind.annotation.*;

import javax.validation.Valid;
import java.util.Map;

/**
```

```
* @author 韩顺平
```

```
* @version 1.0
```

```
*/
```

```
@Controller
```

```
@Scope(value = "prototype")
```

```
public class MonsterHandler {
```

```
//显示添加monster 的界面
```

```
@RequestMapping(value = "/addMonsterUI")
```

```
public String addMonsterUI(Map<String, Object> map) {
```

```
/**老韩解读:
```

1. 这里的表单，我们使用 springMVC 的标签来完成

2. SpringMVC 表单标签在显示之前必须在 request 中有一个 bean，该 bean 的属性和表单标签的字段要对应！

request 中的 key 为：form 标签的 modelAttribute 属性值，比如这里的 monsters

3. SpringMVC 的 form:form 标签的 action 属性值中的 / 不代表 WEB 应用的根目录。

4. <form:form action="monster" method="POST" modelAttribute="monster">

// 这里需要给 request 增加一个 monster，因为 jsp 页面的 modelAttribute="monster" 需要

//这时是 springMVC 的内部的检测机制 即使是一个空的也需要，否则报错。

```
*/  
  
    map.put("monster", new Monster());  
  
    return "datavalid/monster_addUI";  
  
}  
  
}
```

4.

创

建

D:\idea_java_projects\springmvc\web\WEB-INF\pages\datavalid\monster_addUI.jsp

```
<%--  
  
Created by IntelliJ IDEA.  
  
User: 韩顺平  
  
Version: 1.0  
  
Filename: monster_addUI  
  
--%>  
  
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form" %>  
  
<%@ page contentType="text/html;charset=UTF-8" language="java" %>  
  
<html>  
  
<head>  
  
    <title>Title</title>  
  
</head>  
  
<body>
```

<h3>添加妖怪~~</h3>

<!-- 这里的表单，我们使用 springMVC 的标签来完成

特别说明几点：

1. SpringMVC 表单标签在显示之前必须在 request 中有一个 bean，该 bean 的属性和表单标签的字段要对应！

request 中的 key 为：form 标签的 modelAttribute 属性值，比如这里的 monsters

2. SpringMVC 的 form:form 标签的 action 属性值中的 / 不代表 WEB 应用的根目录。

-->

```
<form:form action="save" method="POST" modelAttribute="monster">
```

妖怪名字：<form:input path="name"/>

妖怪年龄~：<form:input path="age"/>

电子邮件：<form:input path="email"/>

<input type="submit" value="添加妖怪"/>

```
</form:form>
```

```
</body>
```

```
</html>
```

5. 创建 D:\idea_java_projects\springmvc\web\WEB-INF\pages\datavalid\success.jsp

```
<%--
```

Created by IntelliJ IDEA.

User: 韩顺平

Version: 1.0

Filename: success

--%>

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>操作成功</title>
</head>
<body>
    <h1>恭喜, 操作成功~</h1>
</body>
</body>
</html>
```

6. 测试, 看看是否可以正确显示添加要求界面(页面测试)

浏览器: http://localhost:8080/springmvc/data_valid.jsp



SpringMVC[数据格式/验证等]

[添加妖怪](#)



添加妖怪~~

妖怪名字:

妖怪年龄~:

电子邮件:

[添加妖怪](#)

7. 修改 MonsterHandler.java , 增加处理添加请求

//处理添加，更加请求的方法来确定

```
@RequestMapping(value = "/save", method = RequestMethod.POST)  
public String save(Monster monster) {
```

```
//注意观察输出内容  
System.out.println("monster= " + monster);  
return "datavalid/success";  
}
```

8. 测试， 浏览器: http://localhost:8080/springmvc/data_valid.jsp

- 1) 如果 age 输入的是 数字，则通过，说明 SpringMVC 可以将提交的字符串 数字，比如"28"，转成 Integer/int
- 2) 如果不是数字，则给出 400 的页面

HTTP Status 400 -

type Status report
message
description The request sent by the client was syntactically incorrect.

Apache Tomcat/8.0.50

- 3) 如何给出对应的提示，我们后面马上讲解

9.2.2 应用实例 -Postman 完成测试

- 完成测试，看看是否可以正确提交数据(Postman 测试)

POST http://localhost:9998/save

Params Auth Headers (11) Body Pre-req. Tests Settings

x-www-form-urlencoded

Key	Value
pwd	11111
email	qq@sohu.com
age	99

1) 如果 age 输入的是 数字，则通过，说明 SpringMVC 可以将提交的字符串 数字，比如"28"，转成 Integer/int

2) 如果不是数字，则给出 400 的页面

HTTP Status 400 -

type Status report

message

description The request sent by the client was syntactically incorrect.

Apache Tomcat/8.0.50

3) 如何给出对应的提示，我们后面马上讲解

9.3 特殊数据类型和字符串间的转换

9.3.1 应用实例 - 页面演示方式

1. 特殊数据类型和字符串之间的转换使用注解(比如日期，规定格式的小数比如货币形式等)
2. 对于日期和货币可以使用 `@DateTimeFormat` 和 `@NumberFormat` 注解. 把这两个注解标记在字段上即可.

添加妖怪~~

妖怪名字:

妖怪年龄~:

妖怪生日: 要求以"9999-11-11"的形式 ①

妖怪工资: 要求以"123,890.12"的形式 ②

电子邮件:

3. 修改 Monster.java , 增加 birthday 和 salary 字段

```
@DateTimeFormat(pattern="yyyy-MM-dd")
```

```
private Date birthday;
```

```
@NumberFormat(pattern="###,###.##")
```

```
private float salary;
```

```
public Date getBirthday() {  
    return birthday;  
}
```

```
public void setBirthday(Date birthday) {  
    this.birthday = birthday;  
}
```

```
public float getSalary() {  
    return salary;  
}
```

```
public void setSalary(float salary) {  
    this.salary = salary;  
}
```

```
public Monster(Integer id, String email, Integer age, String name, Date birthday, float  
salary) {  
    this.id = id;  
    this.email = email;  
    this.age = age;
```

```
this.name = name;  
this.birthday = birthday;  
this.salary = salary;  
}
```

```
@Override  
public String toString() {  
    return "Monster{" +  
        "id=" + id +  
        ", email='" + email + '\'' +  
        ", age=" + age +  
        ", name='" + name + '\'' +  
        ", birthday='" + birthday +  
        ", salary=" + salary +  
        '}';  
}
```

4. 修改 monster_addUI, 增加 birthday 和 salary 字段

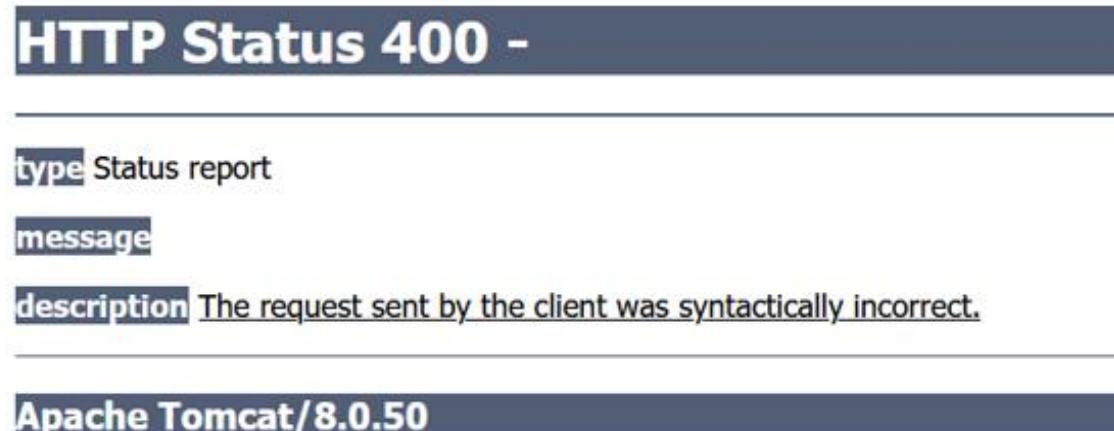
妖怪年龄~: <form:input path="age"/>

妖怪生日: <form:input path="birthday"/> 要求以"9999-11-11"的形式

妖怪工资: <form:input path="salary"/> 要求以"123,890.12"的形式

5. 测试

- 1) 如果 birthday 和 salary 是按照指定格式输入，则通过，说明 SpringMVC 可以按注解指定格式转换
- 2) 如果没有按照注解指定格式，则给出 400 的页面
- 3) 如何给出对应的提示，我们后面马上讲解



9.3.2 应用实例 -Postman 完成测试

- 测试

- 1) 如果 birthday 和 email 是按照指定格式输入，则通过，说明 SpringMVC 可以按注解指定格式转换

POST ▼ http://localhost:9998/save

Params Auth Headers (11) **Body** ● Pre-req. Tests Settings

x-www-form-urlencoded ▼

Key	Value
<input checked="" type="checkbox"/> email	qq@sohu.com
<input checked="" type="checkbox"/> age	99
<input checked="" type="checkbox"/> birthday	2001-11-11
<input checked="" type="checkbox"/> salary	12,3888.5

2) 如果没有按照注解指定格式，则给出 400 的页面

3) 如何给出对应的提示，我们后面马上讲解

HTTP Status 400 -

type Status report

message

description The request sent by the client was syntactically incorrect.

Apache Tomcat/8.0.50

10 验证以及国际化

10.1 概述

- 概述

1. 对输入的数据(比如表单数据), 进行必要的验证, 并给出相应的提示信息。
2. 对于验证表单数据, springMVC 提供了很多实用的注解, 这些注解由 JSR 303 验证框架提供.

- JSR 303 验证框架

1. JSR 303 是 Java 为 Bean 数据合法性校验提供的标准框架, 它已经包含在 JavaEE 中
2. JSR 303 通过在 Bean 属性上标注类似于 @NotNull、@Max 等标准的注解指定校验规则, 并通过标准的验证接口对 Bean 进行验证
3. JSR 303 提供的基本验证注解有:

注解	功能说明
@Null	被注释的元素必须为 null
@NotNull	被注释的元素必须不为 null
@AssertTrue	被注释的元素必须为 true
@AssertFalse	被注释的元素必须为 false
@Min(value)	被注释的元素必须是一个数字, 其值必须大于等于指定的最小值
@Max(value)	被注释的元素必须是一个数字, 其值必须小于等于指定的最大值
@DecimalMin(value)	被注释的元素必须是一个数字, 其值必须大于等于指定的最小值
@DecimalMax(value)	被注释的元素必须是一个数字, 其值必须小于等于指定的最大值
@Size(max, min)	被注释的元素的大小必须在指定的范围内
@Digits (integer, fraction)	被注释的元素必须是一个数字, 其值必须在可接受的范围内
@Past	被注释的元素必须是一个过去的日期
@Future	被注释的元素必须是一个将来的日期
@Pattern(value)	被注释的元素必须符合指定的正则表达式

● Hibernate Validator 扩展注解

1. Hibernate Validator 和 Hibernate 没有关系，只是 JSR 303 实现的一个扩展.
2. Hibernate Validator 是 JSR 303 的一个参考实现，除支持所有标准的校验注解外，它还支持以下的扩展注解：
3. 扩展注解有如下

注解

@Email
@Length
@NotEmpty
@Range

功能说明

被注释的元素必须是电子邮箱地址
被注释的字符串的大小必须在指定的范围内
被注释的字符串的必须非空
被注释的元素必须在合适的范围内

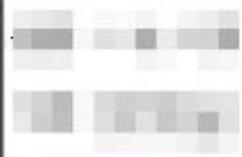
10.2 应用实例

10.2.1 使用实例-代码实现

- 应用实例-需求说明

添加妖怪~~

妖怪名字:	<input type="text"/> 名字不能为空~
妖怪年龄:	<input type="text"/> 输入年龄需要在1-100之间
妖怪生日:	<input type="text"/> 类型不匹配 要求以"1999-11-11"的形式
妖怪工资:	<input type="text"/> 要求以"123,890.12"的形式
电子邮件:	<input type="text"/> 电子邮件不能为空~





• 应用实例-代码实现

1. 引入验证和国际化相关的 jar 包

› 数据验证和国际化需要的jar包

名称
classmate-0.8.0.jar
hibernate-validator-5.0.0.CR2.jar
hibernate-validator-annotation-p...
jboss-logging-3.1.1.GA.jar
validation-api-1.1.0.CR1.jar

2. 修改 Monster.java

```
@NotEmpty  
private String name;  
//增加一个年龄  
@Range(min=1,max=100)  
private Integer age;
```

Monster.java

3. 修改 MonsterHandler.java

```
//处理添加，根据请求的方法来确定  
@RequestMapping(value = "/save", method = RequestMethod.POST)  
public String save(@Valid Monster monster, Errors errors, Map<String, Object> map) {  
    System.out.println("email= " + email);  
    System.out.println("monster= " + monster);  
    if (errors.hasErrors()) {  
        System.out.println("验证出错!");  
        for (ObjectError error : errors.getAllErrors()) {  
            System.out.println(error);  
        }  
        //返回添加界面  
        return "/datavalid/monster_addUI";  
        //return "forward:/addMonsterUI";  
    }  
}
```

```
        return "datavalid/success";  
    }  
}
```

---这里可以测试一下,看看效果---

=====map=====

```
key=monster value=Monster{id=null, email='jack@sohu.com', age=900, name='',  
birthday=Thu Nov 11 00:00:00 CST 1999, salary=11.11}
```

```
key=org.springframework.validation.BindingResult.monster  
value=org.springframework.validation.BeanPropertyBindingResult: 2 errors
```

```
Field error in object 'monster' on field 'age': rejected value [900]; codes  
[Range.monster.age,Range.age,Range.java.lang.Integer,Range]; arguments  
[org.springframework.context.support.DefaultMessageSourceResolvable: codes  
[monster.age,age]; arguments []; default message [age],100,1]; default message  
[需要在 1 和 100 之间]
```

```
Field error in object 'monster' on field 'name': rejected value []; codes  
[NotEmpty.monster.name,NotEmpty.name,NotEmpty.java.lang.String,NotEmpty];  
arguments  
[org.springframework.context.support.DefaultMessageSourceResolvable: codes
```

[monster.name,name]; arguments []; default message [name]]; default message
[不能为空]

=====errors=====

验证出现错误

验证错误=Field error in object 'monster' on field 'age': rejected value [900]; codes
[Range.monster.age,Range.age,Range.java.lang.Integer,Range]; arguments
[org.springframework.context.support.DefaultMessageSourceResolvable: codes
[monster.age,age]; arguments []; default message [age],100,1]; default message
[需要在 1 和 100 之间]]

验证错误=Field error in object 'monster' on field 'name': rejected value []; codes
[NotEmpty.monster.name,NotEmpty.name,NotEmpty.java.lang.String,NotEmpty];
arguments
[org.springframework.context.support.DefaultMessageSourceResolvable: codes
[monster.name,name]; arguments []; default message [name]]; default message
[不能为空]]

添加妖怪

妖怪名字: 不能为空

妖怪年龄: 需要在1和100之间

电子邮件:

妖怪生日: 要求以"9999-11-11"的形式

妖怪工资: Failed to convert property 'java.lang.String' to required type 'java.lang.Float' for property 'salary'. Cause: java.lang.NumberFormatException: For input string: "11xxx.11".
的形式

4. 配置国际化文件 springDispatcherServlet-servlet.xml

```
<bean class="org.springframework.web.servlet.view.BeanNameViewResolver">
    <property name="order" value="99"></property>
</bean>
```

```
<!-- 配置国际化错误信息的资源处理 bean -->
```

```
<bean id="messageSource" class="org.springframework.context.support.ResourceBundleMessageSource">  
    <!-- 配置国际化文件名字  
        如果你这样配的话，表示 messageSource 回到 src/i18nXXX.properties 去读取错误  
        信息-->  
    <property name="basename" value="i18n"></property>  
</bean>
```

5. 创建国际化文件 D:\idea_java_projects\springmvc\src\i18n.properties

NotEmpty.monster.name=\u7528\u6237\u540d\u4e0d\u80fd\u4e3a\u7a7a
typeMismatch.monster.age=\u5e74\u9f84\u8981\u6c42\u5728\u0031\u002d\u0031\u0035\u0030\u4e4b\u95f4
typeMismatch.monster.birthday=\u751f\u65e5\u683c\u5f0f\u4e0d\u6b63\u786e
typeMismatch.monster.salary=\u85aa\u6c34\u683c\u5f0f\u4e0d\u6b63\u786e

6. 修改 monster_addUI.jsp，回显错误信息

```
<form:form action="save" method="POST" modelAttribute="monster">  
    妖怪名字: <form:input path="name"/> <form:errors path="name"/><br><br>  
    妖怪年龄~: <form:input path="age"/> <form:errors path="age"/><br><br>
```

妖怪生日: <form:input path="birthday"/> <form:errors path="birthday"/> 要求以"9999-11-11"的形式

妖怪工资: <form:input path="salary"/> <form:errors path="salary"/> 要求以"123,890.12"的形式

电子邮件: <form:input path="email"/>

<input type="submit" value="添加妖怪"/>
</form:form>

7. 完成测试

添加妖怪~~

妖怪名字: 用户名不能为空

妖怪年龄~: 需要在1和100之间

妖怪生日: 生日格式不正确 要求以"9999-11-11"的形式

妖怪工资: 薪水格式不正确要求以"123,890.12"的形式

电子邮件:

添加妖怪



10.2.2 细节说明和注意事项

1. 在需要验证的 Javabean/POJO 的字段上加上相应的验证注解.
2. 目标方法上,在 JavaBean/POJO 类型的参数前, 添加 @Valid 注解. 告知 SpringMVC 该 bean 是需要验证的
3. 在 @Valid 注解之后, 添加一个 Errors 或 BindingResult 类型的参数, 可以获取到验证的错误信息
4. 需要使用 <form:errors path="email"></form:errors> 标签来显示错误消息, 这个标签, 需要写在<form:form> 标签内生效.
5. 错误消息的国际化文件 i18n.properties , 中文需要是 Unicode 编码, 使用工具转码.

✓ 格式: 验证规则.表单 modelAttribute 值.属性名=消息信息

✓ NotEmpty.monster.name=\u540D\u5B57\u4E0D\u80FD\u4E3A\u7A7A

✓ typeMismatch.monster.age=\u7C7B\u578B\u4E0D\u5339\u914D

5. 注解@NotNull 和 @NotEmpty 的区别说明

- 1) 查看源码可以知道 : @NotEmpty Asserts that the annotated string, collection, map or array is not {@code null} or empty.
- 2) 查看源码可以知道 : @NotNull * The annotated element must not be {@code null}. *

Accepts any type.

3) 老韩解读：如果是字符串验证空，建议使用 @NotEmpty

6. SpringMVC 验证时，会根据不同的验证错误，返回对应的信息

10.3 注解的结合使用

10.3.1 先看一个问题

- 问题提出，age 没有，是空的，提交确是成功了

添加妖怪~~

妖怪名字:

用户名不能为空

妖怪年龄~:

注意看：age我没有填写

妖怪生日:

要求以"9999-11-11"的形式

妖怪工资:

要求以"123,890.12"的形式

电子邮件:

电子邮件不为空~~~~



恭喜，操作成功~

• 解决方案 注解组合使用

10.3.2 解决问题

1. 使用@NotNull + @Range 组合使用解决

2. 具体代码

```
public class Monster {  
  
    private Integer id;  
  
    @NotEmpty(message = "电子邮件不为空~~~~")  
    private String email;  
  
    @NotNull(message = "年龄必须填写 1-100")  
    @Range(min = 1, max = 100)  
    private Integer age;
```

```
@NotNull(message = "生日不能为空")  
@DateTimeFormat(pattern = "yyyy-MM-dd")  
private Date birthday;  
  
@NotNull(message = "薪水不能为空")  
@NumberFormat(pattern = "###,###.##")  
private Float salary;  
}
```

3. 测试(页面方式), 这时 age 不能为空, 同时必须是 1-100, (也不能输入 haha, hello 等不能转成数字的内容)

添加妖怪~~

妖怪名字:

妖怪年龄~: 年龄必须填写1-100

妖怪生日: 要求以"9999-11-11"的形式

妖怪工资: 要求以"123,890.12"的形式

电子邮件:

4. 测试(Postman 方式)

POST ▼ http://localhost:9998/save 1

Params Auth Headers (9) Body Pre-req. Tests Settings

x-www-form-urlencoded 2

Key 3	Value
妖怪名字:	
妖怪年龄~:	
妖怪生日:	

Body Cookies (1) Headers (6) Test Results 🌐

Pretty Raw Preview Visualize

添加妖怪~~

妖怪名字: 用户名不能为空 4

妖怪年龄~: 年龄必须填写1-100

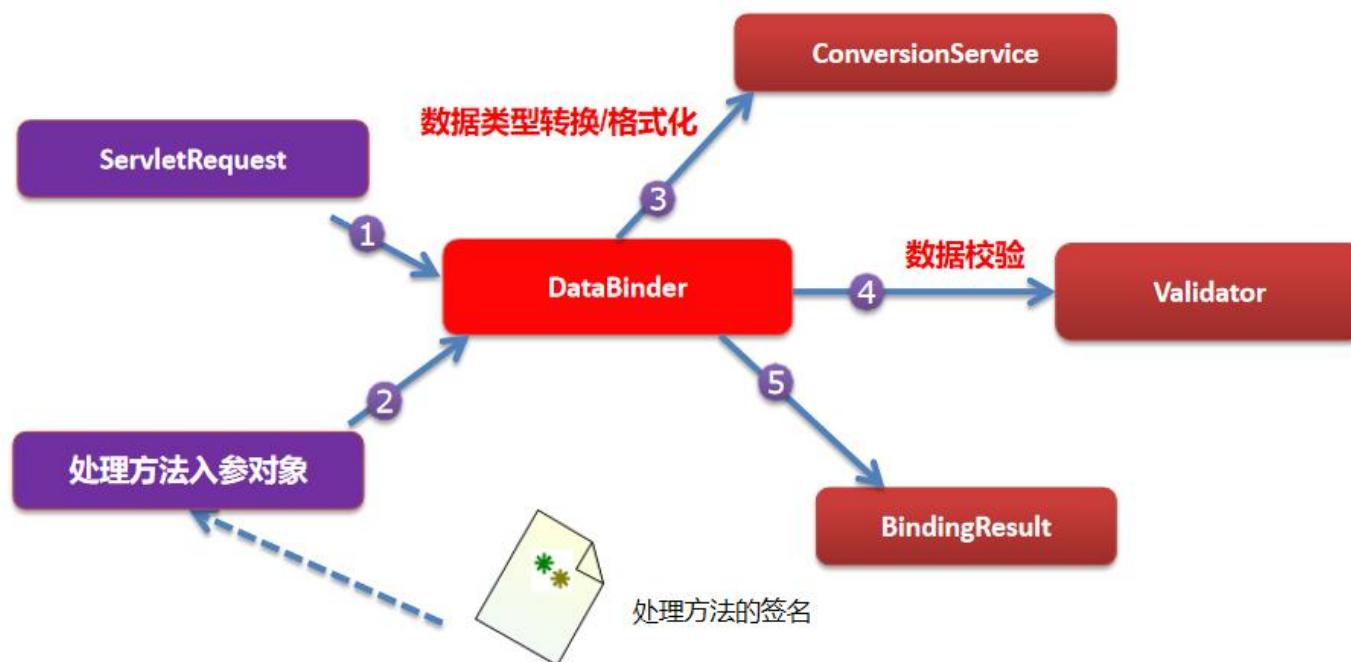
妖怪生日: 要求以"9999-11-11"的形式

10.4 数据类型转换校验核心类-DataBinder

- DataBinder 工作机制-了解

图例 Spring MVC 通过反射机制对目标方法进行解析，将请求消息绑定到处理方法的入参中。

数据绑定的核心部件是 DataBinder，运行机制如下



- Debug 一下 validate 得到验证 errors 信息

```
public void validate(Object... validationHints)
    Object target = getTarget();  target: "Monst
    Assert.state(target != null, "No target to v
    BindingResult bindingResult = getBindingResu
    // Call each validator with the same binding
    for (Validator validator : getValidators())
        if (!ObjectUtils.isEmpty(validationHints))
            if (validationHints.length > 1)
                r) validator.validate(target, bindingR
    else if (validator != null) {
        validator.validate(target, bindingR
```

韩顺平
教育

英 ; 圖 影

m	registerCustomEditor(Class<?
m	findCustomEditor(Class<?>, S
m	convertIfNecessary(Object, Cl
m	bind(PropertyValues): void
m	doBind(MutablePropertyValu
m	checkAllowedFields(MutableF
m	isAllowed(String): boolean
m	checkRequiredFields(Mutable
m	applyPropertyValues(Mutable
m	validate(): void
1	validate(Object...): void
m	close(): Map<?, ?>
f	DEFAULT_OBJECT_NAME: Stri
f	DEFAULT_AUTO_GROW_COLL
f	logger: Log = LogFactory.getl

10.5 取消某个属性的绑定

10.5.1 使用实例

- 说明

在默认情况下，表单提交的数据都会和 pojo 类型的 javabean 属性绑定，如果程序员在开发中，希望取消某个属性的绑定，也就是说，不希望接收到某个表单对应的属性的值，则可以通过 `@InitBinder` 注解取消绑定。

1. 编写一个方法，使用`@InitBinder` 标识的该方法，可以对 `WebDataBinder` 对象进行初始化。`WebDataBinder` 是 `DataBinder` 的子类，用于完成由表单字段到 `JavaBean` 属性的绑定
2. `@InitBinder` 方法不能有返回值，它必须声明为 `void`。
3. `@InitBinder` 方法的参数通常是 `WebDataBinder`

- 案例-不希望接收怪物的名字属性

1. 修改 `MonsterHandler.java`，增加方法

```
//取消绑定 monster 的 name 表单提交的值给 monster.name 属性
```

```
@InitBinder
```

```
public void initBinder(WebDataBinder dataBinder) {
```

//老韩解读

```
//1. setDisallowedFields() 是可变形参，可以指定多个字段  
//2. 当将一个字段属性，设置为 disallowed, 就不在接收表单提交的值  
//3. 那么这个字段属性的值，就是该对象默认的值（具体看程序员定义时指定）  
//4. 一般来说，如果不接收表单字段提交数据，则该对象字段的验证也就没有意义了  
//    ,可以注销掉，比如 注销 //@NotEmpty  
//    //@NotEmpty  
//    private String name;  
//测试完，记得注销。  
dataBinder.setDisallowedFields("name");  
}
```

2. 修改 Monster.java

```
//如果希望取消绑定某个属性，那么验证的注解就没有意义了，注销  
//@NotEmpty  
private String name;
```

Monster.java

3. 完成测试(页面测试)

4. 完成 postman 测试

10.5.2 注意事项和细节说明

1. `setDisallowedFields()` 是可变形参，可以指定多个字段
2. 当将一个字段/属性，设置为 `disallowed`, 就不在接收表单提交的值，那么这个字段/属性的值，就是该对象默认的值（具体看程序员定义时指定）
3. 一般来说，如果不接收表单字段提交数据，则该对象字段的验证也就没有意义了可以注销掉，比如 注销 `//@NotEmpty`

```
// 如果希望取消绑定某个属性，那么验证的注解就没有意义了，注销  
//@NotEmpty  
private String name;
```

Monster.java

11 中文乱码处理

11.1 自定义中文乱码过滤器

- 说明

当表单提交数据为中文时，会出现乱码，我们来解决一下（**老师提示**:先恢复 `name` 属性的绑定）

添加妖怪

妖怪名字:

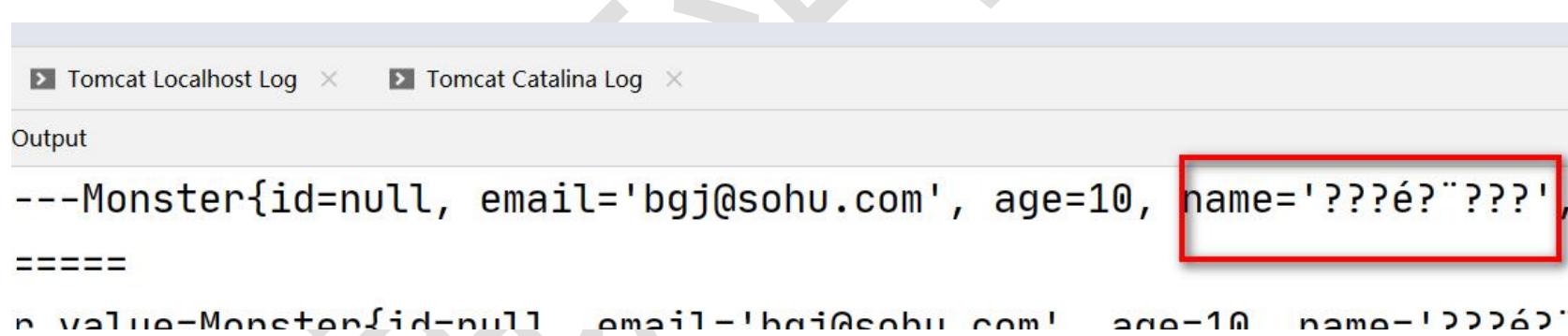
妖怪年龄~:

电子邮件:

妖怪生日: 要求以"9999-11-11"的形式

妖怪薪水: 要求以"123,890.12"的形式

添加妖怪



```
Tomcat Localhost Log  Tomcat Catalina Log
Output
---Monster{id=null, email='bgj@sohu.com', age=10, name='????é?''????'}
=====
n value-Monster{id=null, email='bgj@sohu.com', age=10, name='????é?''????'}
```

● 应用实例

1. 创 建 过 濾 器

D:\idea_java_projects\springmvc\src\com\hspedu\web\filter\MyCharacterFilter.java

```
package com.hspedu.web.filter;
```

```
import javax.servlet.*;  
import java.io.IOException;  
  
/**  
 * @author 韩顺平  
 * @version 1.0  
 */  
  
public class MyCharacterFilter implements Filter {  
    @Override  
    public void init(FilterConfig filterConfig) throws ServletException {  
    }  
    @Override  
    public void doFilter(ServletRequest servletRequest, ServletResponse servletResponse,  
            FilterChain filterChain) throws IOException, ServletException {  
        servletRequest.setCharacterEncoding("utf-8");  
        filterChain.doFilter(servletRequest, servletResponse);  
    }  
    @Override  
    public void destroy() {  
    }  
}
```

```
}
```

```
}
```

2. 配置 web.xml，将该过滤器配置在最前面

```
<!--
```

老韩解读

1. 配置中文乱码处理过滤器
2. 配置到最前面

```
-->
```

```
<filter>
```

```
    <filter-name>myCharacterFilter</filter-name>
```

```
    <filter-class>com.hspedu.web.filter.MyCharacterFilter</filter-class>
```

```
</filter>
```

```
<filter-mapping>
```

```
    <filter-name>myCharacterFilter</filter-name>
```

```
    <url-pattern>/*</url-pattern>
```

```
</filter-mapping>
```

3. 完成测试，浏览器 <http://localhost:8080/springmvc/addMonsterUI>

The screenshot shows a Firefox browser window with the URL `localhost:8080/springmvc/addMonsterUI`. The page title is "添加妖怪~~". The form fields are as follows:

- 妖怪名字:
- 妖怪年龄~:
- 妖怪生日: 要求以"9999-11-11"的形式
- 妖怪工资: 要求以"123,890.12"的形式
- 电子邮件:

```
Monster{id=null, email='kk@sohu.com', age=10, name='小米',
```

11.2 Spring 提供的过滤器处理中文

1. 修改 `web.xml`, 换成 Spring 提供的过滤器, 处理中文乱码

```
<!-- <filter>-->  
<!-- <filter-name>myCharacterFilter</filter-name>-->  
<!-- <filter-class>com.hspedu.web.filter.MyCharacterFilter</filter-class>-->  
<!-- </filter>-->  
<!-- <filter-mapping>-->
```

```
<!--      <filter-name>myCharacterFilter</filter-name>-->
<!--      <url-pattern>/*</url-pattern>-->
<!--      </filter-mapping>-->

<!-- 使用spring 提供的过滤器处理中文, 放在其它Servlet 前 --&gt;

&lt;filter&gt;
    &lt;filter-name&gt;CharacterEncodingFilter&lt;/filter-name&gt;
    &lt;filter-class&gt;org.springframework.web.filter.CharacterEncodingFilter&lt;/filter-class&gt;
    &lt;init-param&gt;
        &lt;param-name&gt;encoding&lt;/param-name&gt;
        &lt;param-value&gt;UTF-8&lt;/param-value&gt;
    &lt;/init-param&gt;
&lt;/filter&gt;
&lt;filter-mapping&gt;
    &lt;filter-name&gt;CharacterEncodingFilter&lt;/filter-name&gt;
    &lt;url-pattern&gt;/*&lt;/url-pattern&gt;
&lt;/filter-mapping&gt;</pre>
```

2. 完成测试，浏览器 <http://localhost:8080/springmvc/addMonsterUI>

The screenshot shows a Firefox browser window with the URL `localhost:8080/springmvc/addMonsterUI`. The page title is "添加妖怪~~". The form fields are:

- 妖怪名字:
- 妖怪年龄~:
- 妖怪生日: 要求以"9999-11-11"的形式
- 妖怪工资: 要求以"123,890.12"的形式
- 电子邮件:

```
Monster{id=null, email='kk@sohu.com', age=10, name='小米',
```

12 处理 json 和 `HttpMessageConverter<T>`

12.1 处理 JSON-`@ResponseBody`

- 说明：

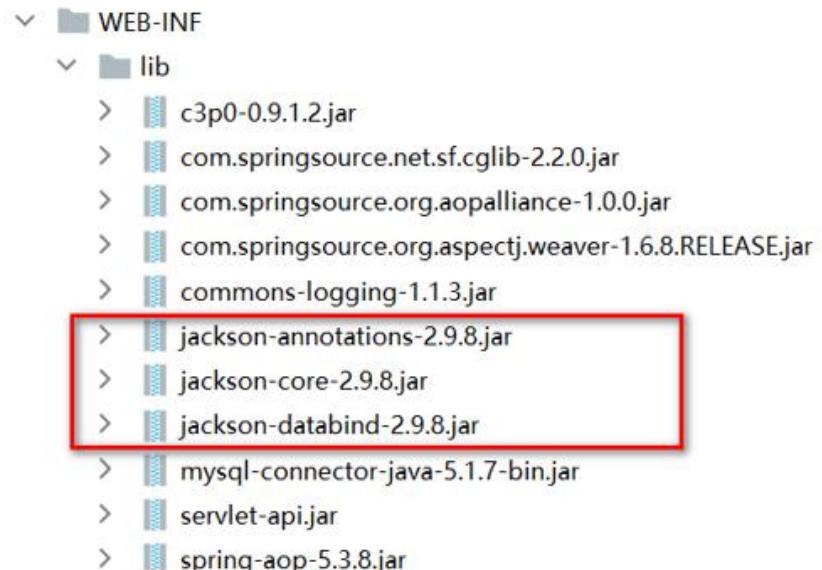
项目开发中，我们往往需要服务器返回的数据格式是按照 json 来返回的，我们看一下 SpringMVC 是如何处理的，功能示意图，下面是老韩要完成的效果

The screenshot shows a browser window with the URL `localhost:8080/springmvc/json.jsp`. Below the address bar is a toolbar with icons for FireFox, Newbie Road, Common Websites, JD.com, and Taobao 11. The main content area displays the title "请求一个json数据" and a blue link "点击获取json数据". Below this is a network monitoring tool interface. The tabs at the top are: 查看器 (Inspector), 控制台 (Console), 调试器 (Debugger), 网络 (Network), 样式编辑器 (Style Editor), and 性能 (Performance). The "Network" tab is selected. Underneath are filters: 垃圾 (Garbage) and 过滤 URL (Filter URL). The main table has columns: 方 (Method), 域名 (Domain), 文件 (File), 发送 (Send), 传输 (Transfer), 消息头 (Headers), Cookie (Cookie), 请求 (Request), 响应 (Response), 耗时 (Time), and 条件 (Condition). A single row is selected, showing a POST request to `getJson.jsp` with status 206. The "Response" column shows the JSON content:

```
{ "name": "大黄狗", "address": "北京八达岭~"}
```

• 应用案例

1. 引入处理 json 需要的 jar 包，注意 spring5.x 需要使用 jackson-2.9.x.jar 的包。



2. 创建 D:\idea_java_projects\springmvc\web\json.jsp

```
<%--  
     Created by IntelliJ IDEA.  
     User: 韩顺平  
     Version: 1.0  
     Filename: json  
--%>  
<%@ page contentType="text/html;charset=UTF-8" language="java" %>  
<html>  
<head>  
    <title>json 提交</title>  
<!-- 引入 jquery -->
```

```
<script type="text/javascript" src="script/jquery-3.6.0.min.js"></script>

<!-- 编写jquery 代码和请求 --&gt;

&lt;script type="text/javascript"&gt;

$(function () {

    //绑定超链接点击事件

    $("#getJSON").click(function () {

        //href 是一个完整的请求地址

        var href = this.href;

        alert(href);

        var args = {"time": new Date()};//防止缓存

        //发出一个jquery 的 post 请求，请求返回json

        $.post(href, args, function (data) {

            alert(" name= " + data.name + " address= " + data.address);

            //1. data 是json 对象

            //2. JSON.stringify(data) 是将json 对象转成字符串

            alert("返回数据 json=" + data)

            alert("返回数据 json=" + JSON.stringify(data))

        });

    });

    //防止重复提交

    return false;

})</pre>
```

```
})
</script>
</head>
<body>
<h1>请求一个 json 数据</h1>
<a href="getJSON" id="getJSON">点击获取 json 数据</a>
</body>
</html>
```

3. 创建 JavaBean: D:\idea_java_projects\springmvc\src\com\hspedu\web\json\Dog.java , 作为返回的数据

```
package com.hspedu.web.json;

/**
 * @author 韩顺平
 * @version 1.0
 */
public class Dog {
    private String name;
    private String address;
```

```
public Dog() {  
}  
  
public Dog(String name, String address) {  
    this.name = name;  
    this.address = address;  
}  
  
public String getName() {  
    return name;  
}  
  
public void setName(String name) {  
    this.name = name;  
}  
  
public String getAddress() {  
    return address;  
}  
  
public void setAddress(String address) {  
    this.address = address;  
}
```

```
}
```

```
@Override  
public String toString() {  
    return "Dog{" +  
        "name=\"" + name + "\"" +  
        ", address=\"" + address + "\"" +  
        '}';  
}
```

4. 创建 D:\idea_java_projects\springmvc\src\com\hspedu\web\json\JsonHandler.java

```
package com.hspedu.web.json;  
  
import org.springframework.http.HttpHeaders;  
import org.springframework.http.HttpStatus;  
import org.springframework.http.ResponseEntity;  
import org.springframework.stereotype.Controller;  
import org.springframework.web.bind.annotation.*;  
  
import javax.servlet.http.HttpSession;
```

```
import java.io.InputStream;

/**
 * @author 韩顺平
 * @version 1.0
 */

@Controller
public class JsonHandler {

    @RequestMapping(value = "/getJSON")
    //指定返回的数据格式json，靠这个@ResponseBody
    @ResponseBody
    public Dog getJson() {
        //创建一只狗
        Dog dog = new Dog();
        dog.setName("大黄狗");
        dog.setAddress("北京八达岭~");
        return dog;
    }
}
```

5. 完成测试(页面方式), 浏览器 <http://localhost:8080/springmvc/json.jsp>

The screenshot shows the Postman application interface. At the top, there's a toolbar with icons for back, forward, search, and refresh, followed by the URL 'localhost:8080/springmvc/json.jsp'. Below the toolbar are several links: 火狐官方站点, 新手上路, 常用网址, JD 京东商城, and 天猫双11. The main title of the application is '请求一个json数据' (Request a json data). Below the title is a blue link '点击获取json数据' (Click to get json data). The Postman interface has tabs for View, Console, Debugger, Network (which is selected), Style Editor, and Performance. There's also a trash icon and a filter URL input field. A sub-menu bar below the tabs includes All, HTML, CSS, JS, XHR, Font, Image, Media, WS, and Other. The Network tab shows a single request listed: '2 PO 🔒 ... getJson.jsp js 206'. Underneath this, there's a '过滤属性' (Filter Properties) dropdown. The response section is titled 'JSON' and displays the following JSON object:

```
1 {"name": "大黄狗", "address": "北京八达岭~"}
```

6. 完成测试(Postman 方式)

The screenshot shows a POST request to `http://localhost:9998/getJson`. The 'Body' tab is selected, showing the following JSON payload:

```
1 {  
2   "name": "大黄狗",  
3   "address": "北京八达岭~"  
4 }
```

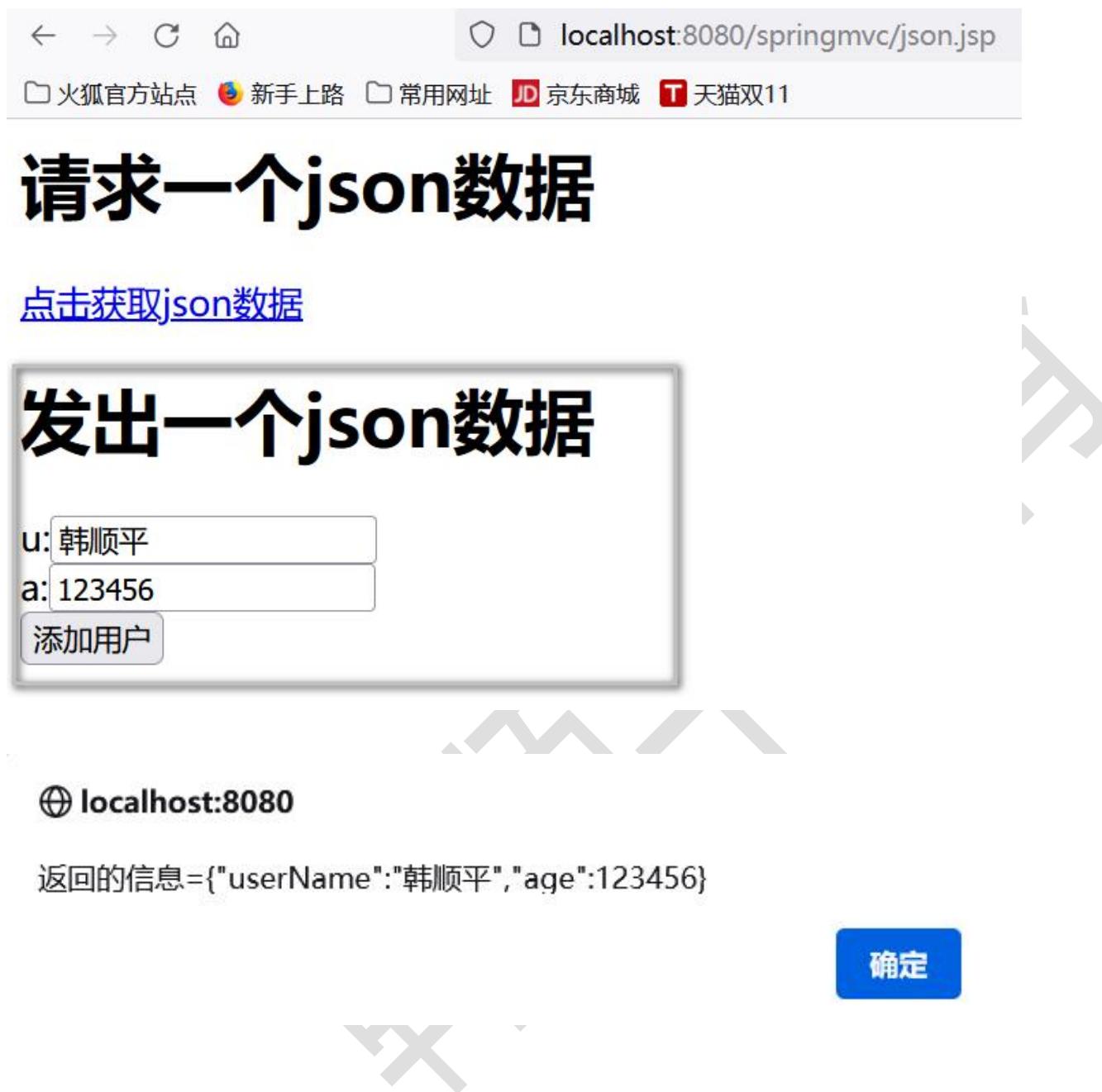
12.2 处理 JSON-@RequestBody

- 应用案例

- 前面老韩给大家讲解的是通过表单，或者 url 请求携带 `参数名=参数值` 把数据提交给目标方法

- 1) 给大家举例客户端发送 json 字符串数据，
- 2) 使用 SpringMVC 的 `@RequestBody` 将客户端提交的 json 数据，封装成 JavaBean 对象
- 3) 再把这个 javabean 以 json 对象形式返回

4) 完成效果示意图



1. 修改 json.jsp, 增加发送 json 数据代码

```
//绑定按钮点击事件，提交json 数据
//springmvc 可以在后台将json 转成对象
$("button[name='butt1']").click(function () {

    var userName = $("#userName").val()

    var age = $("#age").val()

    $.ajax({
        url: "/springmvc/save2",
        data: JSON.stringify({"userName": userName, "age": age}),
        type: "POST",
        success: function (data) {
            alert("返回的信息=" + JSON.stringify(data));
        },
        contentType: "application/json;charset=utf-8"
    });
})
```

<h1>请求一个 json 数据</h1>

点击获取 json 数据

<h1>发出一个 json 数据</h1>

u:<input id="userName" type="text">


```
a:<input id="age" type="text"><br/>
<button name="butt1">添加用户</button>
```

2. 修改 JsonHandler.java , 增加处理 json 代码, 注意: 老韩用的是@PostMapping , 等价:@RequestMapping(method = RequestMethod.POST)

```
@PostMapping("/save2")
@ResponseBody
public User save2(@RequestBody User user){
    //将前台传过来的数据 以json 的格式相应回浏览器
    System.out.println("user~= " + user);
    return user ;
}
```

3. 增加 JavaBean : User.java

```
package com.hspedu.web.json;
```

```
/*
 * @author 韩顺平
 * @version 1.0
```

```
*/  
  
public class User {  
  
    private String userName;  
  
    private Integer age;  
  
  
    public User() {  
    }  
  
  
    public User(String userName, Integer age) {  
        this.userName = userName;  
        this.age = age;  
    }  
  
  
    public String getUserName() {  
        return userName;  
    }  
  
  
    public void setUserName(String userName) {  
        this.userName = userName;  
    }  
  
  
    public Integer getAge() {
```

```
        return age;  
    }  
  
    public void setAge(Integer age) {  
        this.age = age;  
    }  
  
    @Override  
    public String toString() {  
        return "User{" +  
               "userName='" + userName + '\'' +  
               ", age='" + age + '\'' +  
               '}';  
    }  
}
```

4. 并完成测试(页面方式), 浏览器 <http://localhost:8080/springmvc/json.jsp>



请求一个json数据

[点击获取json数据](#)

发出一个json数据

u: 韩顺平

a: 123456

[添加用户](#)

⊕ localhost:8080

返回的信息 = {"userNmae": "韩顺平", "age": 123456}

确定

5. 并完成测试(Postman 方式)

POST ▼ http://localhost:9998/save2

Params Auth Headers (10) Body Pre-req. Tests Settings

<input checked="" type="checkbox"/>	Cache-Control ⓘ	no-cache
<input checked="" type="checkbox"/>	Postman-Token ⓘ	<calculated when request is sent>
<input checked="" type="checkbox"/>	Content-Type ⓘ	application/json 1
<input checked="" type="checkbox"/>	Content-Length ⓘ	<calculated when request is sent>
<input checked="" type="checkbox"/>	Host ⓘ	<calculated when request is sent>

Body Cookies (1) Headers (4) Test Results

POST ▼ http://localhost:9998/save2

Params Auth Headers (10) Body Pre-req. Tests Settings

raw ▼ JSON ▼

```
1 {  
2   "userName": "老韩~",  
3   "age": 18  
4 }
```

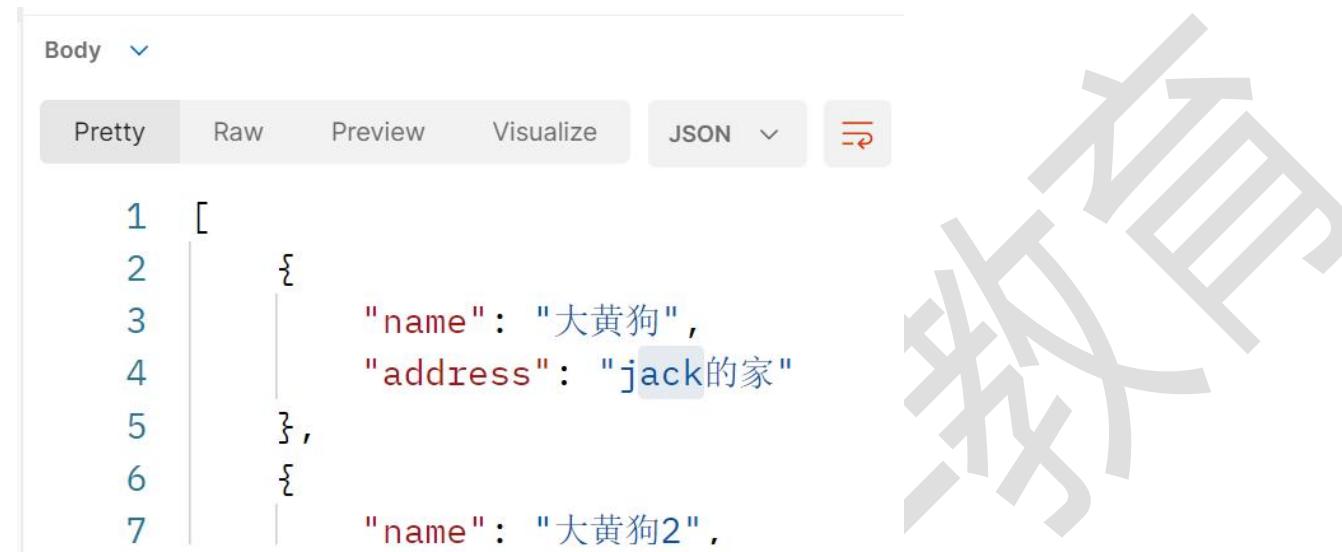
Body Cookies (1) Headers (4) Test Results

Pretty Raw Preview Visualize JSON ▼

```
1 {  
2   "userName": "老韩~",  
3   "age": 18
```

12.3 处理 JSON-注意事项和细节

1. 目标方法正常返回 JSON 需要的数据，可以是一个对象，也可以是一个集合[举例]



The screenshot shows a JSON editor interface with the following code:

```
1 [  
2   {  
3     "name": "大黄狗",  
4     "address": "jack的家"  
5   },  
6   {  
7     "name": "大黄狗2",  
8   }  
9 ]
```

The code represents a list of two dog objects. Each object has a name and an address. The address for the first dog is highlighted in blue.

2. 前面我讲解的是返回一个 Dog 对象->转成 Json 数据格式返回

3. `@ResponseBody` 可以直接写在 controller 上，这样对所有方法生效

1) 应用实例：

```
package com.hspedu.web.json;
```

```
/**  
 * @author 韩顺平  
 * @version 1.0
```

```
*/
```

```
import org.springframework.stereotype.Controller;  
import org.springframework.web.bind.annotation.PostMapping;  
import org.springframework.web.bind.annotation.RequestBody;  
import org.springframework.web.bind.annotation.RequestMapping;  
import org.springframework.web.bind.annotation.ResponseBody;  
  
/**  
 * @author 韩顺平  
 * @version 1.0  
 */  
@Controller  
@ResponseBody  
public class JsonHandler {  
  
    @RequestMapping(value = "/getJSON")  
    //指定放回的数据格式json，靠这个@ResponseBody  
    //@ResponseBody  
    public Dog getJson() {  
        //创建一只狗  
        Dog dog = new Dog();
```

```
    dog.setName("大黄狗~~!");
    dog.setAddress("北京八达岭~!");
    return dog;
}
```

```
@PostMapping("/save2")
//@ResponseBody
public User save2(@RequestBody User user){
    //将前台传过来的数据 以 json 的格式相应回浏览器
    System.out.println("user~= " + user);
    return user ;
}
}
```

2) 完成测试(页面和 Postman 都可以通过)

4. `@ResponseBody + @Controller` 可以直接写成 `@RestController`，我们看一下源码！

```
@Target(ElementType.TYPE)
@Retention(RetentionPolicy.RUNTIME)
@Documented
@Controller
```

@ResponseBody

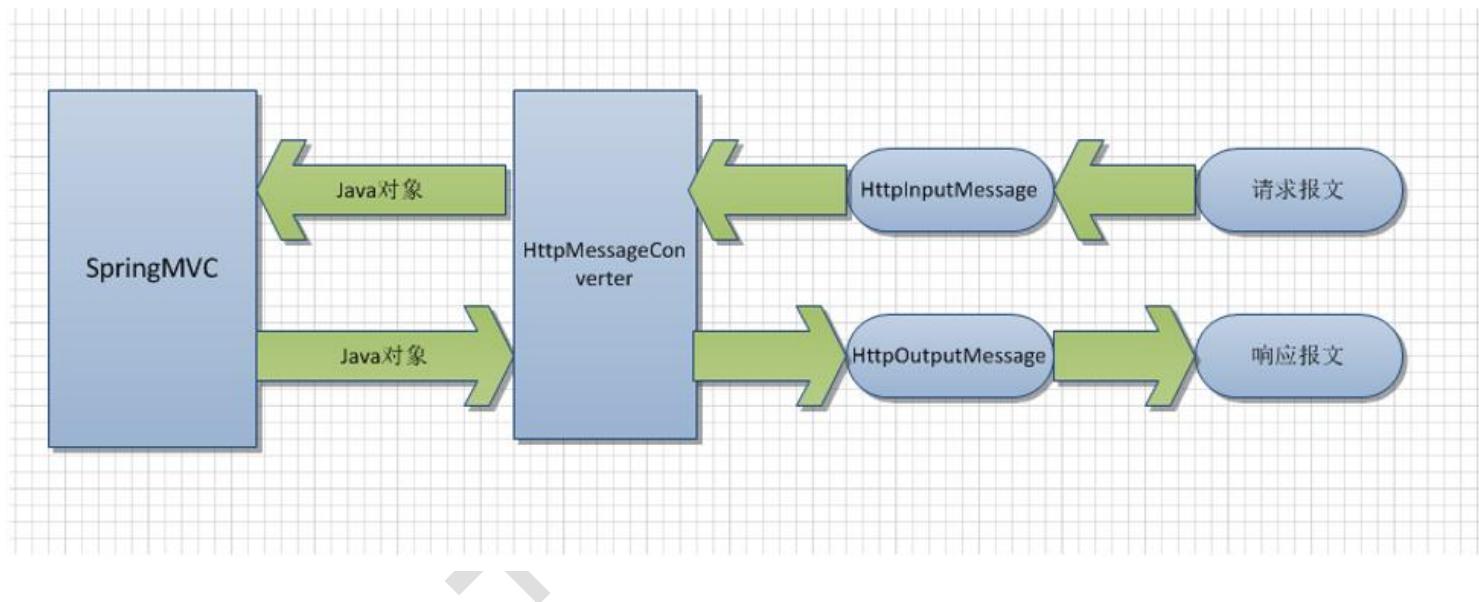
```
public @interface RestController {  
}  
}
```

12.4 HttpMessageConverter<T>

- 基本说明

SpringMVC 处理 JSON-底层实现是依靠 `HttpMessageConverter<T>` 来进行转换的

- 工作机制简图



- 处理 JSON-底层实现(`HttpMessageConverter<T>`)

1. 使用 `HttpMessageConverter<T>` 将请求信息转化并绑定到处理方法的入参中，或将响应

结果转为对应类型的响应信息，Spring 提供了两种途径：

- ✓ 使用 `@RequestBody / @ResponseBody` 对目标方法进行标注
- ✓ 使用 `HttpEntity<T> / ResponseEntity<T>` 作为目标方法的入参或返回值

2. 当控制器处理方法使用到 `@RequestBody/@ResponseBody` 或 `HttpEntity<T>/ResponseEntity<T>` 时，Spring 首先根据请求头或响应头的 `Accept` 属性选择匹配的 `HttpMessageConverter`，进而根据参数类型或泛型类型的过滤得到匹配的 `HttpMessageConverter`，若找不到可用的 `HttpMessageConverter` 将报错

- Dubug 源码-梳理一下



```
AbstractJackson2HttpMessageConverter.java × json.jsp × MonsterHandler.java × JsonHandler.java ×
49             JavaType javaType = getJavaType(clazz, null);
50             return readJavaType(javaType, inputMessage);
51     }
52
53     @
54     private Object readJavaType(JavaType javaType, HttpI
55         MediaType contentType = inputMessage.getHeaders();
56         Charset charset = getCharset(contentType);
57
58         ObjectMapper objectMapper = selectObjectMapper(j
59         Assert.state(objectMapper != null, "No ObjectMapper
60
61         boolean isUnicode = ENCODINGS.containsKey(charse
62             "UTF-16".equals(charset.name()) ||
```

```
    @RequestMapping(value = "/json/save2")
    // @ResponseBody
    public User getJson(@RequestBody User user) {
        System.out.println("---接收到的user---" + user);
        return user;
    }
}
```

```
AbstractJackson2HttpMessageConverter.java  json.jsp  MonsterHandler.java  JsonHandler.java
406
407     }
408 }
409
410 @Override
411 protected void writeInternal(Object object, @Nullable
412             throws IOException, HttpMessageNotWritableException)
413
414 MediaType contentType = outputMessage.getHeaders()
415 JsonEncoding encoding = getJsonEncoding(contentType)
416
417 Class<?> clazz = (object instanceof MappingJacksonValue)
418             ((MappingJacksonValue) object).getValue().getClass();
419 ObjectMapper objectMapper = selectObjectMapper(clazz);
```

12.5 文件下载- `ResponseEntity<T>`

- 说明：

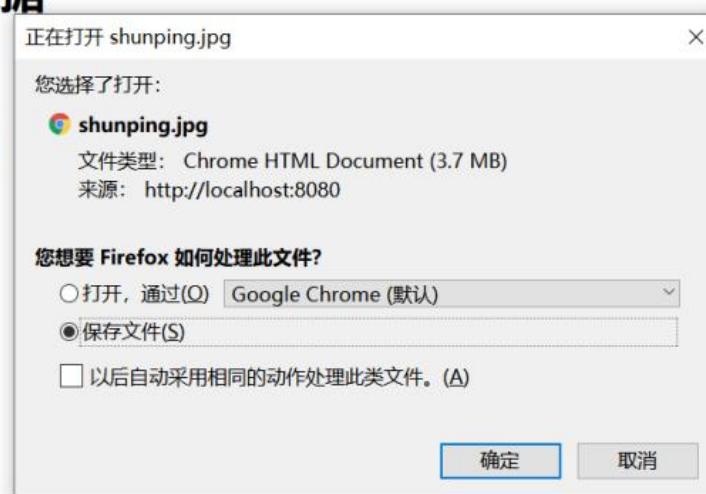
在 SpringMVC 中，通过返回 `ResponseEntity<T>` 的类型，可以实现文件下载的功能

请求一个json数据

[点击获取json数据](#)

下载文件的测试

[点击下载文件](#)



- 案例演示

1. 修改 json.jsp

<h1>发出一个 json 数据</h1>

u:<input id="userName" type="text">

a:<input id="age" type="text">

<button name="butt1">添加用户</button>

<h1>下载文件的测试 </h1>

点击下载文件

2. 修改 JsonHandler.java，增加方法

```
//响应用户下载文件的请求  
 @RequestMapping(value="/downFile")  
 public ResponseEntity<byte[]> downFile(HttpSession session) throws Exception{  
     //先获取到你要下载的文件InputStream  
     InputStream is = session.getServletContext().getResourceAsStream("/img/shunping.jpg");  
     //开辟一个存放文件内容 byte 数组,因为 byte 是字节, 因此可以返回二进制的文件【图片, 视频。。】  
     byte[] bytes= new byte[is.available()];  
     is.read(bytes);  
     //构建返回的 ResponseEntity<byte[]>  
     HttpStatus statuts = HttpStatus.OK;//返回成功  
     HttpHeaders headers = new HttpHeaders();//根据 http 协议这个就是告诉浏览器, 这是返回  
     的一个文件, 浏览器就弹出小窗口  
     headers.add("Content-Disposition", "attachment;filename=shunping.jpg");  
     ResponseEntity<byte[]> responseEntity = new  
     ResponseEntity<byte[]>(bytes,headers,statuts);
```

```
return ResponseEntity;
```

```
}
```

文件下载响应头的设置

content-type 指示响应内容的格式

content-disposition 指示如何处理响应内容。

一般有两种方式：

inline: 直接在页面显示

attachment: 以附件形式下载

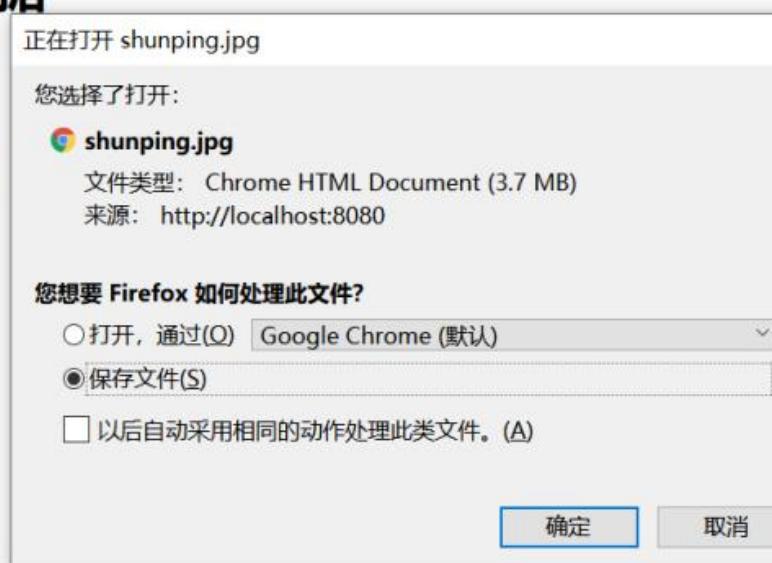
3. 完成测试(页面方式) <http://localhost:8080/springmvc/json.jsp>

请求一个json数据

[点击获取json数据](#)

下载文件的测试

[点击下载文件](#)



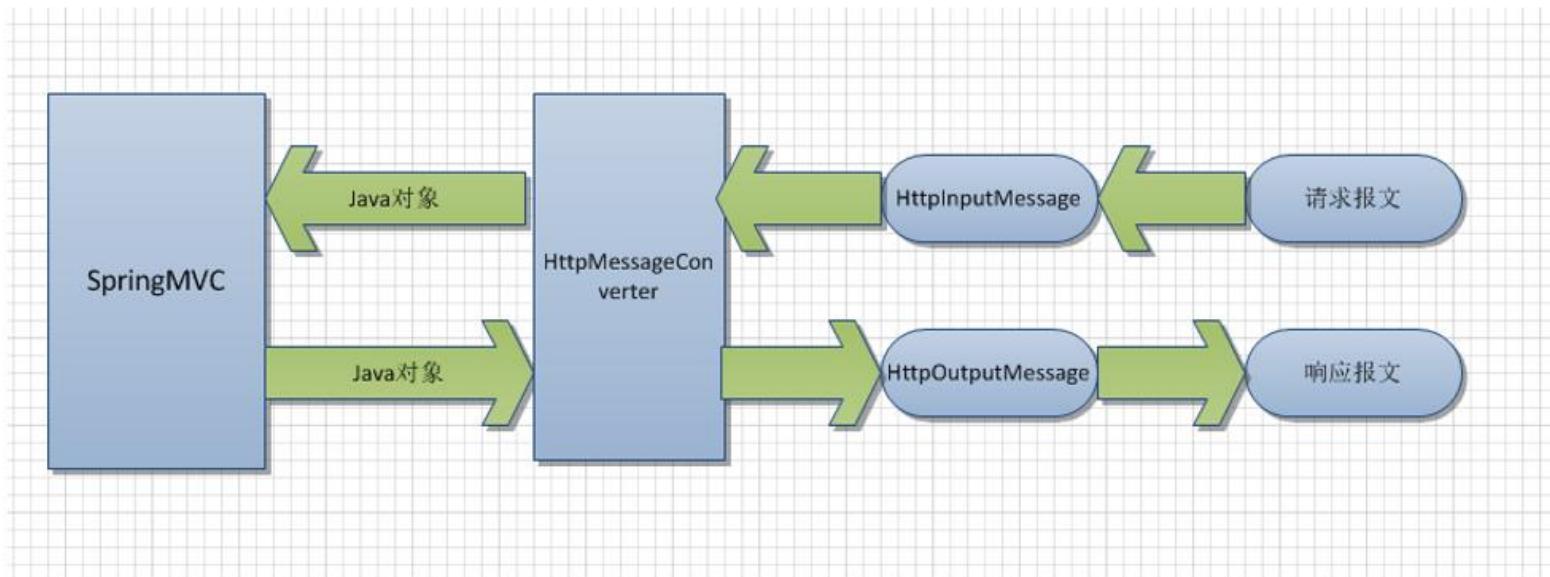
4. 完成测试(Postman 方式)

12.6 作业布置

1. 把前面老师讲过的数据格式化、验证以及国际化、Json 处理，文件下载，相关代码和案例，自己写一遍

-一定要自己写一遍,否则没有印象, 理解不会深入

2. 把老师 Debug 过的 HttpMessageConverter 源码，自己也走一下，加深理解(不用每一条语句，都 debug, 找流程...)



3. DataBinder 工作机制-将示意图画出

4. Debug 一下 validate 得到验证 errors 信息，加深理解(不用每一条语句都 debug, 找流程)

13 SpringMVC 文件上传

13.1 基本介绍

- 基本介绍

1. Spring MVC 为文件上传提供了直接的支持，这种支持是通过即插即用的 MultipartResolver 实现的。Spring 用 Jakarta Commons FileUpload 技术实现了一个 MultipartResolver 实现类：CommonsMultipartResolver
2. Spring MVC 上下文中默认没有装配 MultipartResolver，因此默认情况下不能处理文件的上传工作，如果想使用 Spring 的文件上传功能，需现在上下文中配置 MultipartResolver

```
<!-- 配置文件上传解析器 -->
<bean
    id="multipartResolver"
    class="org.springframework.web.multipart.commons.CommonsMultipartResolver">
</bean>
```

13.2 需求分析/图解

文件上传的演示

文件介绍:

选择文件: 浏览... 未选择文件。
上传文件

13.3 应用实例-代码实现

1. 引入 springmvc 文件上传需要的 jar 包



2. 创建 D:\idea_java_projects\springmvc\web\fileUpload.jsp

```
<%--
```

Created by IntelliJ IDEA.

User: 韩顺平

Version: 1.0

Filename: fileUpload

--%>

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>文件上传</title>
</head>
<body>
<h1>文件上传的演示</h1>
<form action="fileUpload" method="post"
      enctype="multipart/form-data">
    文件介绍:<input type="text" name="introduce"><br>
    选择文件:<input type="file" name="file"><br>
    <input type="submit" value="上传文件">
</form>
</body>
</html>
```

3. 配置中文过滤器，在 web.xml，使用 Spring 提供的，前面我们配置过了

4. 配置文件上传解析器，在 springDispatcherServlet-servlet.xml，简单看一下 CommonsMultipartResolver 源码

```
<!-- 配置一个 springmvc 的文件上传解析器 -->
<bean id="multipartResolver"
      class="org.springframework.web.multipart.commons.CommonsMultipartResolver"></bean>

<!-- 加入两个常规配置 -->
<!-- 能支持 SpringMVC 高级功能，比如 JSR303 校验，映射动态请求 -->
<mvc:annotation-driven></mvc:annotation-driven>
<!-- 将 SpringMVC 不能处理的请求交给 Tomcat，比如请求 css,js 等-->
<mvc:default-servlet-handler/>
```

5.

创

建

D:\idea_java_projects\springmvc\src\com\hspedu\web\fileupload\FileUploadHandler.java

```
package com.hspedu.web.fileupload;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.multipart.MultipartFile;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
```

```
import java.io.File;
import java.io.IOException;
import java.util.UUID;

/**
 * @author 韩顺平
 * @version 1.0
 */

@Controller
public class FileUploadHandler {
    @RequestMapping(value="/fileUpload")
    public String fileUpload(@RequestParam(value="file") MultipartFile file,
                           HttpServletRequest request)
            throws Exception, IOException {
        String originalFilename = file.getOriginalFilename();
        System.out.println("上传文件名= " + originalFilename);
        String filepath = request.getServletContext().getRealPath("/img/" +
originalFilename);
        //转存到 d:/kk.jpg
        File saveToFile = new File(filepath);
        file.transferTo(saveToFile);
        return "success";
    }
}
```

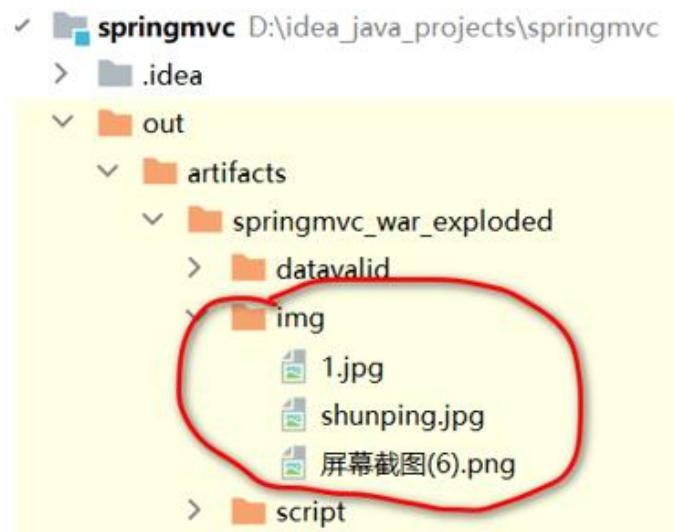
```
}
```

```
}
```

6. 完成测试（页面方式），看文件是否成功上传，浏览器：

<http://localhost:8080/springmvc/fileUpload.jsp>，简单的 debug 一下 transferTo()





7. 完成测试(Postman 方式)

The screenshot shows the Postman interface for a 'New Request'. The method is set to 'POST' and the URL is 'http://localhost:9998/fileUpload'. The 'Body' tab is selected, showing 'form-data' selected. There are two fields: 'introduce' with value '太远1' and 'file' with value '5.jpg'. A red circle highlights the URL field. A red arrow points from the text '选择File' to the 'File' dropdown button next to the 'file' field. A tooltip message below the 'file' field reads: 'This file isn't in your working directory. Teammates you share this request with won't be able to use this file. To make collaboration easier you can setup your'.

恭喜，操作成功~

14 自定义拦截器

14.1 什么是拦截器

• 说明

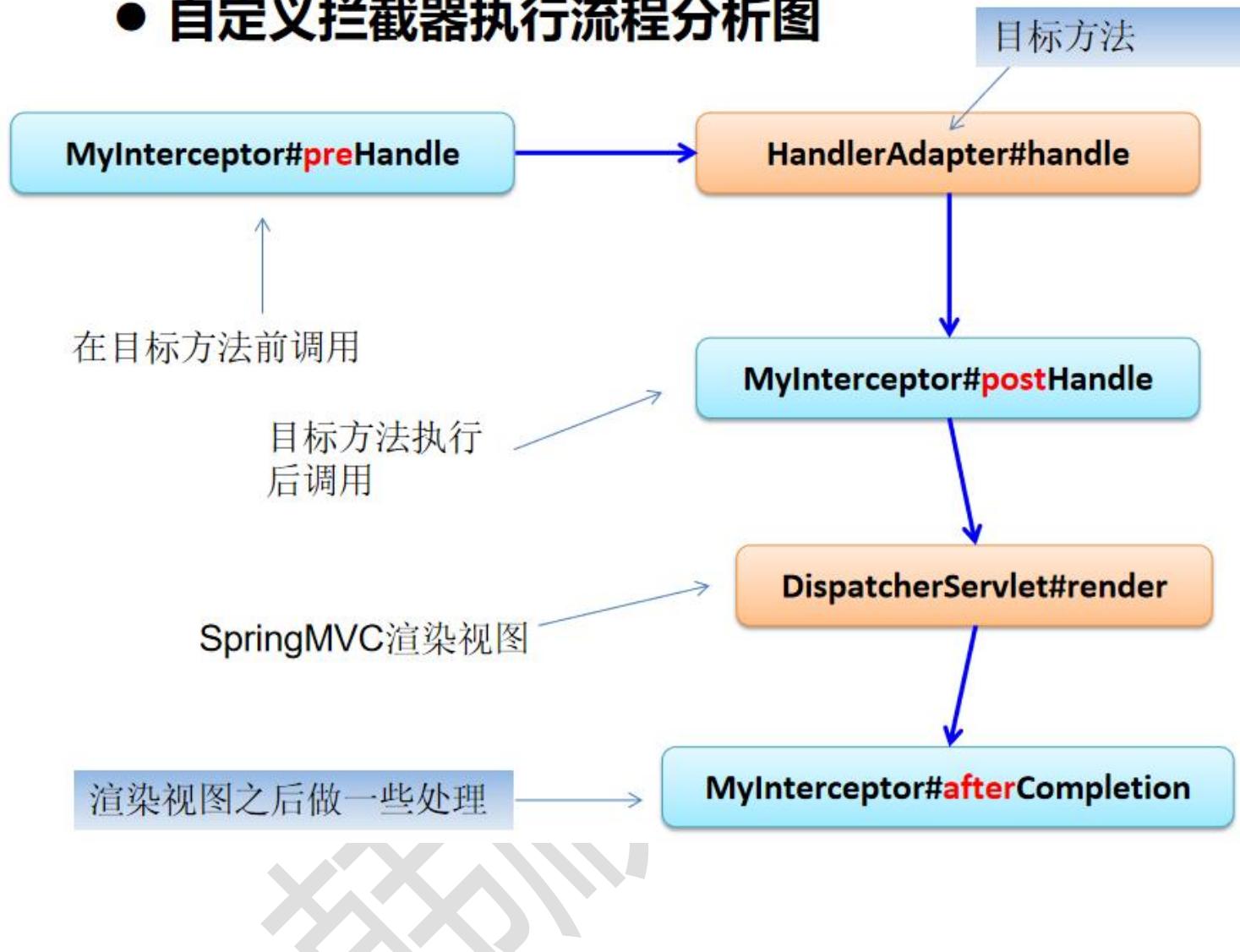
1. Spring MVC 也可以使用拦截器对请求进行拦截处理，用户可以自定义拦截器来实现特定的功能。
2. 自定义的拦截器必须实现 HandlerInterceptor 接口

- 自定义拦截器的三个方法

1. **preHandle()**: 这个方法在业务处理器处理请求之前被调用，在该方法中对用户请求 **request** 进行处理。
2. **postHandle()**: 这个方法在目标方法处理完请求后执行
3. **afterCompletion()**: 这个方法在完全处理完请求后被调用，可以在该方法中进行一些资源清理的操作。

14.2 自定义拦截器执行流程分析图

● 自定义拦截器执行流程分析图



● 自定义拦截器执行流程说明

1. 如果 `preHandle` 方法 返回 `false`, 则不再执行目标方法, 可以在此指定返回页面
2. `postHandle` 在目标方法被执行后执行. 可以在方法中访问到目标方法返回的 `ModelAndView` 对象
3. 若 `preHandle` 返回 `true`, 则 `afterCompletion` 方法 在渲染视图之后被执行.

4. 若 preHandle 返回 false，则 afterCompletion 方法不会被调用
5. 在配置拦截器时，可以指定该拦截器对哪些请求生效，哪些请求不生效

14.3 自定义拦截器应用实例

14.3.1 快速入门

- 应用实例需求

完成一个自定义拦截器，学习一下如何配置拦截器和拦截器的运行流程

- 应用实例-代码实现

1.

创

建

D:\idea_java_projects\springmvc\src\com\hspedu\web\interceptor\MyInterceptor01.java

```
package com.hspedu.web.interceptor;
```

```
import org.springframework.stereotype.Component;
```

```
import org.springframework.web.servlet.HandlerInterceptor;
```

```
import org.springframework.web.servlet.ModelAndView;
```

```
import javax.servlet.http.HttpServletRequest;
```

```
import javax.servlet.http.HttpServletResponse;

/**
 * @author 韩顺平
 * @version 1.0
 */

//配置一个 @Component 交给 springmvc 管理
@Component
public class MyInterceptor01 implements HandlerInterceptor {

    /**
     * 1.pre 方法在目标方法执行之前被调用.
     * 2.返回 false, 则不会再执行目标方法. 可以在此响应请求返回给页面
     * 3.不管返回 true 还是 false
     *      会执行当前拦截器之前的拦截器的 afterCompletion 方法. 注意:不会执行当前拦截器的 afterCompletion 方法.
     */
    @Override
    public boolean preHandle
        (HttpServletRequest arg0, HttpServletResponse arg1,
         Object arg2) throws Exception {
        // System.out.println("preHandle.....");
    }
}
```

```
// String mess = "炸弹";  
// if(mess.equals("炸弹")){  
// //返回到一个警告页面  
// arg0.getRequestDispatcher("/WEB-INF/pages/warning.jsp").forward(arg0, arg1);  
// return false;  
// }else{  
// return true;  
// }  
  
System.out.println("==>MyInterceptor01 preHandle()=<==");  
return true;  
}  
  
/*  
* 说明 在目标方法被执行之后执行. 可以在方法中访问到目标方法返回的  
ModelAndView 对象.  
*/  
@Override  
public void postHandle(HttpServletRequest arg0,  
                         HttpServletResponse arg1,  
                         Object arg2, ModelAndView arg3) throws Exception {  
    System.out.println("==>MyInterceptor01 postHandle()=<==");  
}  
/*
```

- * 若 preHandle 返回 true，则方法在渲染视图之后被执行.
- * 若 preHandle 返回 false，则该方法不会被调用.
- * 若当前拦截器的下一个拦截器的 preHandle 方法返回 false，则在执行下一个拦截器 preHandle 方法后马上被执行.

* 可以访问到目标方法中出现的异常:

*/

@Override

```
public void afterCompletion(HttpServletRequest arg0,
                             HttpServletResponse arg1, Object arg2, Exception
                             arg3)
        throws Exception {
    System.out.println("==>MyInterceptor01 afterCompletion()");
}
```

2.

创

建

D:\idea_java_projects\springmvc\src\com\hspedu\web\interceptor\FurnHandler.java 作为
目标方法

```
package com.hspedu.web.interceptor;
```

```
import org.springframework.stereotype.Controller;
```

```
import org.springframework.web.bind.annotation.RequestMapping;
```

```
/**  
 * @author 韩顺平  
 * @version 1.0  
 */  
  
@Controller  
public class FurnHandler {  
  
    @RequestMapping(value = "/hi")  
    public String hi() {  
        System.out.println("FurnHandler hi()");  
        return "success";  
    }  
  
    @RequestMapping(value = "/hello")  
    public String hello() {  
        System.out.println("FurnHandler hello()");  
        return "success";  
    }  
}
```

3. 在 springDispatcherServlet-servlet.xml 配置拦截器

<!-- 如何配置自定义的拦截器 -->

<!--

了解：

1. 创建实现 HandlerInterceptor 接口的 bean

2. 在 mvc:interceptors 中配置拦截器

-->

<**mvc:interceptors**>

<!-- 没有看出来是不是必须配置这个 bean, 注销也没有发现错误 -->

<**bean**

class="org.springframework.web.servlet.i18n.LocaleChangeInterceptor"></bean>

<!-- 第一种方法：自己在 interceptors 配置一个引用 指向你需要使用的拦截器 -->

这种配置方式，对所有的请求都拦截

-->

<!-- <ref bean="myInterceptor01"/>-->

<!--

老韩解读

1. 第二种配置方法：也可以在 interceptor 配置

2. mvc:mapping 表示自定义拦截器 只对 /hi 请求进行拦截

比如：

<**mvc:interceptor**>

<**mvc:mapping path="/hi"/>**

```
<ref bean="myInterceptor01"/>  
</mvc:interceptor>
```

3. mvc:exclude-mapping 表示自定义拦截器 不对哪些 请求拦截

比如: myInterceptor 拦截器对/h 开头请求拦截,排除 /hello

```
<mvc:interceptor>  
    <mvc:mapping path="/h*"/>  
    <mvc:exclude-mapping path="/hello"/>  
    <ref bean="myInterceptor01"/>  
</mvc:interceptor>
```

-->

```
<mvc:interceptor>  
    <mvc:mapping path="/hi"/>  
    <ref bean="myInterceptor01"/>  
</mvc:interceptor>  
</mvc:interceptors>
```

4. 创建 D:\idea_java_projects\springmvc\web\interceptor.jsp

```
<%--
```

Created by IntelliJ IDEA.

User: 韩顺平

Version: 1.0

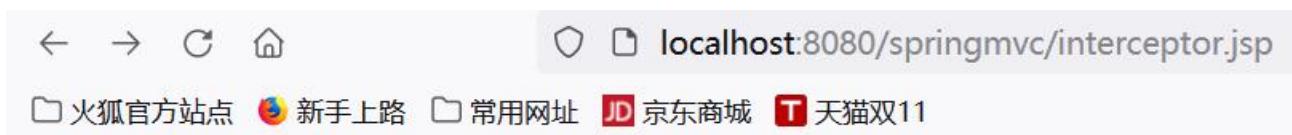
Filename: interceptor

--%>

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>

<html>
<head>
    <title>测试自定义拦截器</title>
</head>
<body>
    <h1>测试自定义拦截器</h1>
    <a href="hi">测试自定义拦截器</a><br><br>
    <a href="hello">登录</a>
</body>
</html>
```

5. 完成测试(页面方式), 浏览器 <http://localhost:8080/springmvc/interceptor.jsp>



测试自定义拦截器

[测试自定义拦截器](#)

[登录](#)

```
Output
====MyInterceptor01 preHandle() ====
FurnHandler hi()
====MyInterceptor01 postHandle() ====
====MyInterceptor01 afterCompletion() ====
```

6. 完成测试(Postman 方式)

The screenshot shows the Postman interface with a POST request to `http://localhost:9998/hi`. The 'Body' tab is selected, showing a `form-data` payload with two fields: `introduce` (value: 太远1) and `Key` (value: Value). The response status is 200 OK.

	KEY	VALUE	DESCI
<input type="checkbox"/>	introduce	太远1	
	Key	Value	Descri

Body Cookies (1) Headers (5) Test Results 200 OK

Pretty Raw Preview Visualize

恭喜，操作成功~

```
==>MyInterceptor01 preHandle(~~)==>
FurnHandler hi()
==>MyInterceptor01 postHandle()==>
==>MyInterceptor01 afterCompletion()==>
```

14.3.2 注意事项和细节

1、默认配置是都所有的目标方法都进行拦截，也可以指定拦截目标方法，比如只是拦截 `hi`

```
<mvc:interceptor>

    <mvc:mapping path="/hi"/>

    <ref bean="myInterceptor01"/>

</mvc:interceptor>
```

2、 mvc:mapping 支持通配符，同时指定不对哪些目标方法进行拦截

```
<mvc:interceptor>

    <mvc:mapping path="/h*"/>

    <mvc:exclude-mapping path="/hello"/>

    <ref bean="myInterceptor01"/>

</mvc:interceptor>
```

3、 拦截器需要配置才生效，不配置是不生效的。

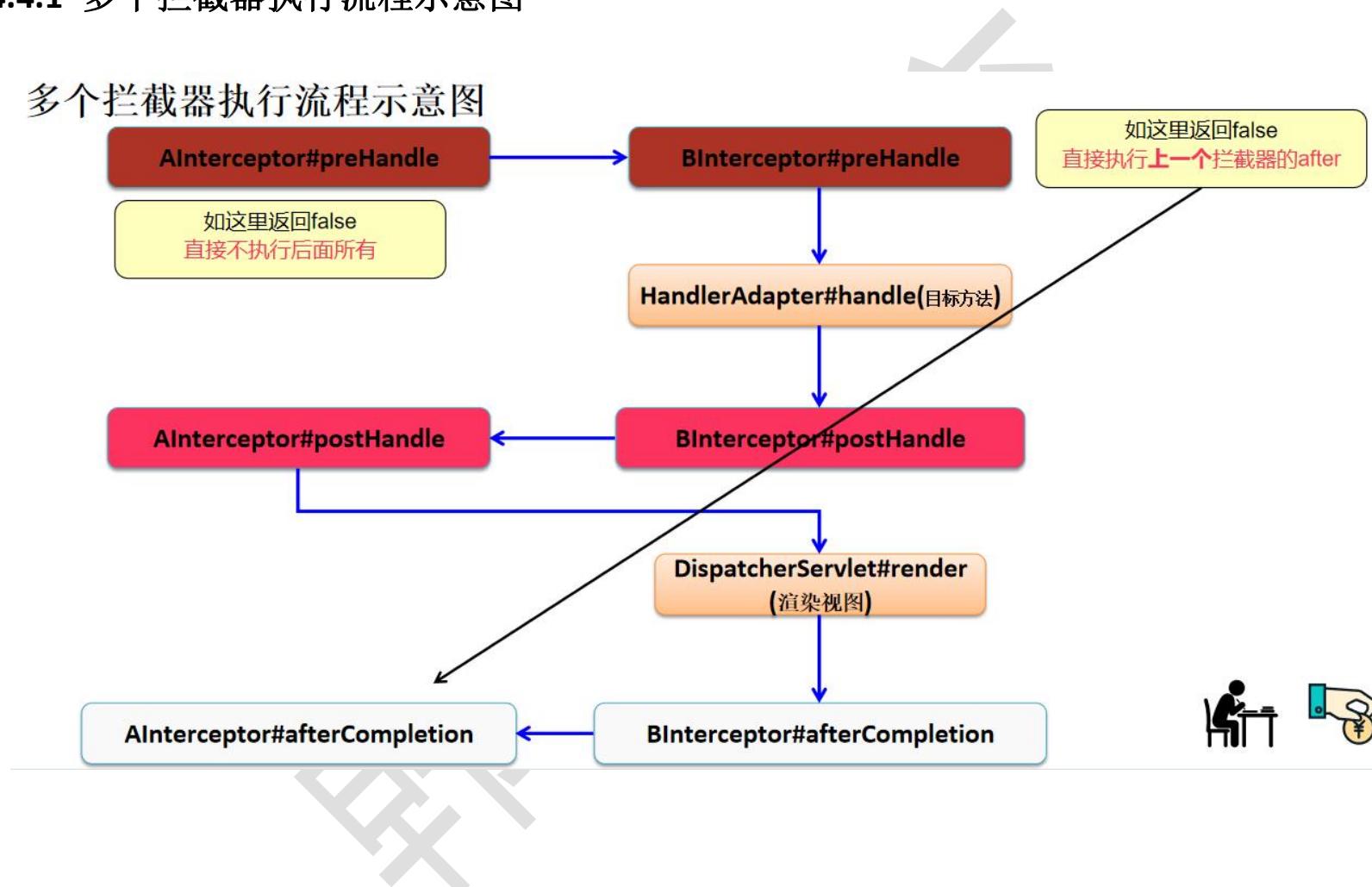
4、 如果 preHandler() 方法返回了 false，就不会执行目标方法(前提是你的目标方法被拦截了)，程序员可以在这里根据业务需要指定跳转页面。

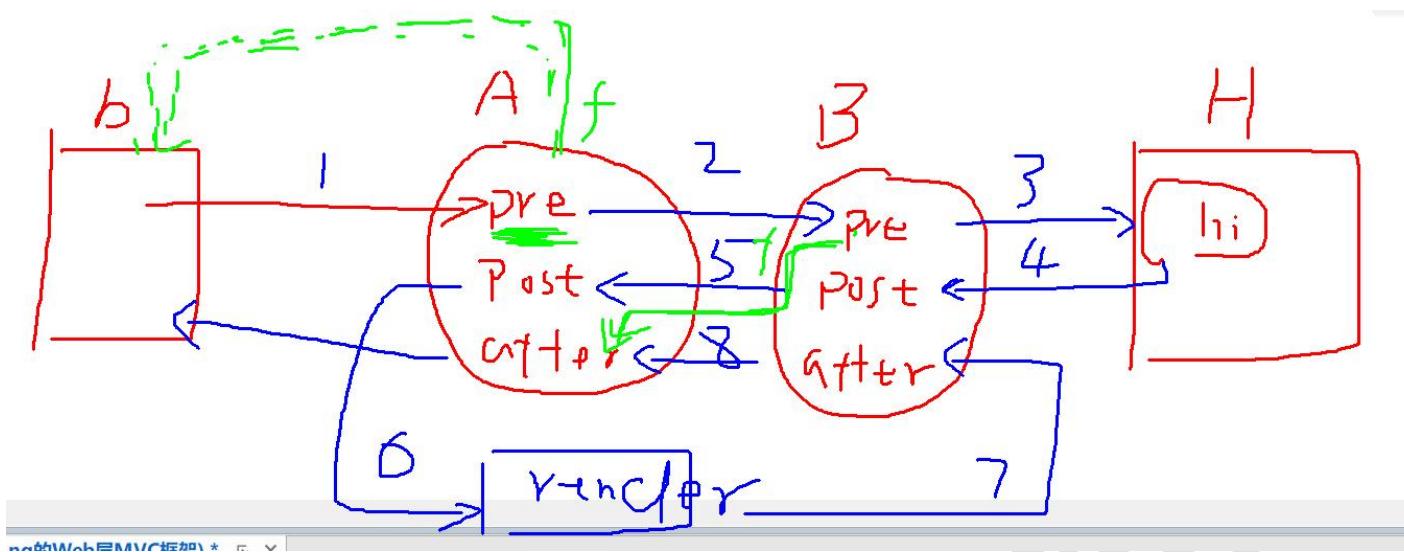
14.3.3 Debug 执行流程

- 看老师的 Debug 演示

14.4 多个拦截器

14.4.1 多个拦截器执行流程示意图





14.4.2 应用实例 1

14.4.2.1 代码实现

1.

创

建

D:\idea_java_projects\springmvc\src\com\hspedu\web\interceptor\MyInterceptor02.java

```
package com.hspedu.web.interceptor;

import org.springframework.stereotype.Component;
import org.springframework.web.servlet.HandlerInterceptor;
import org.springframework.web.servlet.ModelAndView;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
```

```
/*
 * @author 韩顺平
 * @version 1.0
 */

//配置一个 @Component 交给 springmvc 管理

@Component
public class MyInterceptor02 implements HandlerInterceptor {

    /**
     * 1.pre 方法在目标方法执行之前被调用
     * 2.返回 false, 则不会再执行目标方法. 可以在此响应请求返回给页面
     * 3.不管返回 true 还是 false
     *      会执行当前拦截器之前的拦截器的 afterCompletion 方法. 注意:不会执行当前拦截器的 afterCompletion 方法.
     */
    @Override
    public boolean preHandle
        (HttpServletRequest arg0, HttpServletResponse arg1,
         Object arg2) throws Exception {
        // System.out.println("preHandle.....");
        // String mess = "炸弹";
        // if(mess.equals("炸弹")){
        /// 返回到一个警告页面
    }
}
```

```
// arg0.getRequestDispatcher("/WEB-INF/pages/warning.jsp").forward(arg0, arg1);
// return false;
// }else{
// return true;
// }

System.out.println("==>MyInterceptor02 preHandle()=<==");
return true;
}

/**
 * 说明 在目标方法被执行之后执行. 可以在方法中访问到目标方法返回的
ModelAndView 对象.
*/
@Override
public void postHandle(HttpServletRequest arg0,
                        HttpServletResponse arg1,
                        Object arg2, ModelAndView arg3) throws Exception {
    System.out.println("==>MyInterceptor02 postHandle()=<==");
}

/**
 * 若 preHandle 返回 true, 则方法在渲染视图之后被执行.
 * 若 preHandle 返回 false, 则该方法不会被调用.
 * 若当前拦截器的下一个拦截器的 preHandle 方法返回 false, 则在执行下一个拦截器

```

preHandle 方法后马上被执行.

* 可以访问到目标方法中出现的异常.

*/

@Override

public void afterCompletion(HttpServletRequest arg0,

HttpServletResponse arg1, Object arg2, Exception arg3)

throws Exception {

System.out.println("==>MyInterceptor02 afterCompletion()");

}

}

2. 配置 springDispatcherServlet-servlet.xml

```
<mvc:interceptor>
    <mvc:mapping path="/hi"/>
    <ref bean="myInterceptor01"/>
</mvc:interceptor>
<mvc:interceptor>
    <mvc:mapping path="/hi"/>
    <ref bean="myInterceptor02"/>
</mvc:interceptor>
```

```
<mvc:interceptors>
    <bean class="org.springframework.web.servlet.i18n.LocaleChangeInterceptor"></bean>
    <!-- 第一种方法：自己在interceptors配置一个引用 指向你需要使用的拦截器
        这种配置方式，对所有的请求都拦截
    -->
<!--      <ref bean="myInterceptor01"/>--> springDispatcherServlet-servlet.xml

    <!--...-->
    <mvc:interceptor>
        <mvc:mapping path="/hi"/>
        <ref bean="myInterceptor01"/>
    </mvc:interceptor>
    <mvc:interceptor>
        <mvc:mapping path="/hi"/>
        <ref bean="myInterceptor02"/>
    </mvc:interceptor>
</mvc:interceptors>
```

1. 对myInterceptor02拦截器配置
2. 注意拦截器执行顺序和配置顺序一致

3. 完成测试(页面方式)，浏览器 <http://localhost:8080/springmvc/interceptor.jsp>



测试自定义拦截器

[测试自定义拦截器](#)

[登录](#)

```
▼ Output  
→ ===MyInterceptor01 preHandle()===  
← ===MyInterceptor02 preHandle()===  
↻ FurnHandler hi()  
↓ ===MyInterceptor02 postHandle()===  
== MyInterceptor01 postHandle()===  
== MyInterceptor02 afterCompletion()===  
== MyInterceptor01 afterCompletion()===
```

4. 完成测试(Post 方式)



GET ▼ http://localhost:9998/hi

Params Auth Headers (8) **Body** ● Pre-req. Tests Settings

form-data ▼

	KEY	VALUE
<input type="checkbox"/>	introduce	太远1
	Key	Value

Body Cookies (1) Headers (5) Test Results

Pretty Raw Preview Visualize

14.4.2.2 注意事项和细节

1. 如果第 1 个拦截器的 `preHandle()` 返回 `false`，后面都不在执行
2. 如果第 2 个拦截器的 `preHandle()` 返回 `false`，就直接执行第 1 个拦截器的 `afterCompletion()` 方法，如果拦截器更多，规则类似
3. 说明：前面说的规则，都是目标方法被拦截的前提

14.4.3 应用实例 2

1. 需求：如果用户提交的数据有禁用词(比如 病毒)，则，在第 1 个拦截器就返回，不执行目标方法，功能效果示意图

The screenshot shows a POSTMAN interface with the following details:

- Method: GET
- URL: `http://localhost:9998/hi?topic=病毒`
- Params tab is active, showing a table with one row:

	KEY	VALUE	DE
<input checked="" type="checkbox"/>	topic	病毒	
- Body tab is active, showing a table with two rows:

Key	Value	De
- Response status: 200 OK

不要乱讲话~

The screenshot shows the Postman interface with a successful API call. The method is GET, and the URL is `http://localhost:9998/hi?topic=你好`. The 'Params' tab is selected, showing a table with one row: 'topic' (Value: 你好). Other tabs like Headers, Body, and Cookies are visible but not selected.

恭喜，操作成功~

2. 代码实现：创建 D:\hspedu_ssm_temp\springmvc\web\WEB-INF\pages\warning.jsp

```
<%--
```

Created by IntelliJ IDEA.

User: 韩顺平

Version: 1.0

Filename: warning

```
--%>

<%@ page contentType="text/html;charset=UTF-8" language="java" %>

<html>
<head>
    <title>警告</title>
</head>
<body>
<h1>不要乱讲话~</h1>
</body>
</html>
```

3. 代 码 实 现 : 修 改

D:\hsedu_ssm_temp\springmvc\src\com\hsedu\web\interceptor\MyInterceptor01.java

```
@Override
public boolean preHandle
(HttpServletRequest arg0, HttpServletResponse arg1,
Object arg2) throws Exception {
    if ("病毒".equals(arg0.getParameter("topic"))) {
        arg0.getRequestDispatcher("/WEB-INF/pages/warning.jsp").forward(arg0,
arg1);
        return false;
    }
}
```

```
}
```

```
System.out.println("==>MyInterceptor01 preHandle(~~)==>");
```

```
return true;
```

```
}
```

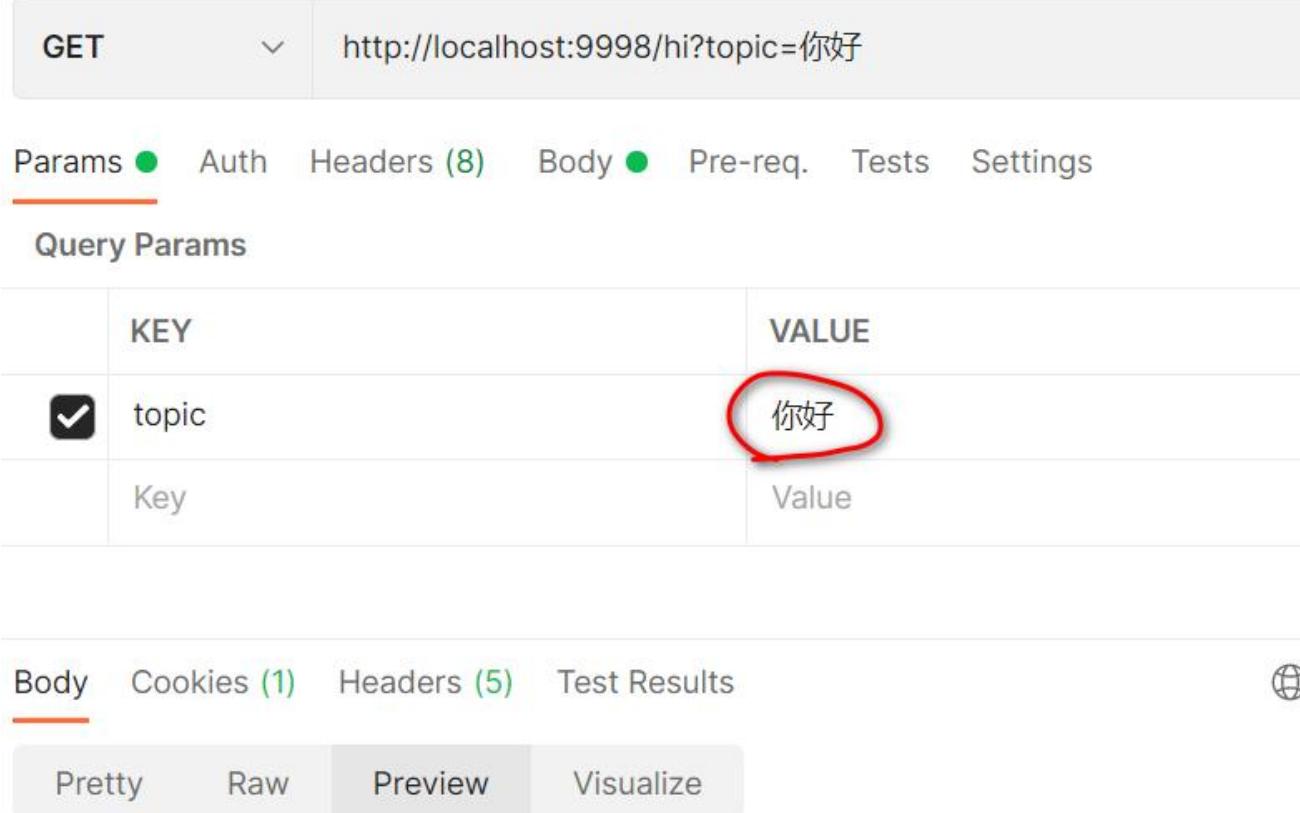
4. 完成测试(使用 Postman)

The screenshot shows the Postman interface with the following details:

- Method: GET
- URL: <http://localhost:9998/hi?topic=%E7%A7%91>
- Params tab is selected.
- Query Params table:

KEY	VALUE	DE
topic	病毒	
Key	Value	De
- Body tab is selected.
- Status: 200 OK

不要乱讲话~



GET ▼ http://localhost:9998/hi?topic=你好

Params ● Auth Headers (8) Body ● Pre-req. Tests Settings

Query Params

	KEY	VALUE
<input checked="" type="checkbox"/>	topic	你好
	Key	Value

Body Cookies (1) Headers (5) Test Results 🌐

Pretty Raw Preview Visualize

恭喜，操作成功~

14.5 作业布置

1. 把前面老师讲过的 **SpringMVC** 文件上传、自定义拦截器相关代码和案例，自己写一遍
-一定要自己写一遍，否则没有印象，理解不会深入
2. 简述 **SpringMVC** 自定义拦截器工作流程，并画出示意图
3. 把老师 **Debug** 过的自定义拦截器源码，自己也走一下，加深理解(不用每一条语句都 **debug**，主要是梳理流程)

15 异常处理

15.1 异常处理-基本介绍

- 基本介绍

1. Spring MVC 通过 HandlerExceptionResolver 处理程序的异常，包括 Handler 映射、数据绑定以及目标方法执行时发生的异常。
2. 主要处理 Handler 中用 `@ExceptionHandler` 注解定义的方法。

3. ExceptionHandlerMethodResolver 内部若找不到 `@ExceptionHandler` 注解的话，会找 `@ControllerAdvice` 类的`@ExceptionHandler` 注解方法，这样就相当于一个全局异常处理器

15.2 局部异常

15.2.1 应用实例

- 应用实例需求

演示局部异常处理机制

- 如果不处理异常，非常的不友好

[←](#) [→](#) [C](#)

localhost:8080/springmvc/testException01?num=0

HTTP Status 500 - Request processing failed; nested exception is java.lang.Arithme

ticException: / by zero

type Exception report**message** Request processing failed; nested exception is java.lang.ArithmeticException: / by zero**description** The server encountered an internal error that prevented it from fulfilling this request.**exception**

```
org.springframework.web.util.NestedServletException: Request processing failed;
    org.springframework.web.servlet.FrameworkServlet.processRequest(FrameworkServlet.java:953)
    org.springframework.web.servlet.FrameworkServlet.doGet(FrameworkServlet.java:857)
    javax.servlet.http.HttpServlet.service(HttpServlet.java:622)
    org.springframework.web.servlet.FrameworkServlet.service(FrameworkServlet.java:834)
```

● 应用实例-代码实现

1.

创

建

D:\idea_java_projects\springmvc\src\com\hspedu\web\exception\MyExceptionHandler.java

```
package com.hspedu.web.exception;
```

```
import org.springframework.stereotype.Controller;
```

```
import org.springframework.web.bind.annotation.ExceptionHandler;
```

```
import org.springframework.web.bind.annotation.RequestMapping;
```

```
import javax.servlet.http.HttpServletRequest;
```

```
/**  
 * @author 韩顺平  
 * @version 1.0  
 */  
  
@Controller  
public class MyExceptionHandler {  
  
    //局部异常就是直接在这个 Handler 中编写即可  
    @ExceptionHandler({ArithmaticException.class,NullPointerException.class})  
    public String localException(Exception ex, HttpServletRequest request){  
        System.out.println("异常信息是~" + ex.getMessage());  
  
        //如何将异常的信息带到下一个页面  
        request.setAttribute("reason", ex.getMessage());  
        return "exception_mes";  
    }  
  
    @RequestMapping(value="/testException01")  
    public String test01(Integer num){  
        int i = 9/num;  
  
        return "success";  
    }  
}
```

}

2. 创建 D:\idea_java_projects\springmvc\web\exception.jsp

```
<%--  
     Created by IntelliJ IDEA.  
     User: 韩顺平  
     Version: 1.0  
     Filename: exception  
--%>  
<%@ page contentType="text/html;charset=UTF-8" language="java" %>  
<html>  
<head>  
    <title>异常信息</title>  
</head>  
<body>  
    <h1>测试异常</h1>  
    <a href="testException01?num=0">点击测试异常</a><br><br>  
</body>  
</html>
```

3. 创建显示异常信息页面 /WEB-INF/pages/exception_mes.jsp

<%--

Created by IntelliJ IDEA.

User: 韩顺平

Version: 1.0

Filename: exception

--%>

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>异常提示信息</title>
</head>
<body>
    <h1>朋友，发生异常,信息如下 :</h1>
    ${reason }
</body>
</html>
```

4. 测试(页面方式), 浏览器 <http://localhost:8080/springmvc/exception.jsp>



测试异常

[点击测试异常](#)



朋友，发生异常，信息如下：

/ by zero

5. 测试(Postman 方式)

GET ▼ http://localhost:9998/testException01?num=0

Params ● Auth Headers (8) Body ● Pre-req. Tests Settings

Query Params

	KEY	VALUE	D
<input checked="" type="checkbox"/>	num	0	D
	Key	Value	D

Body Cookies (1) Headers (6) Test Results 🌐 200 OK

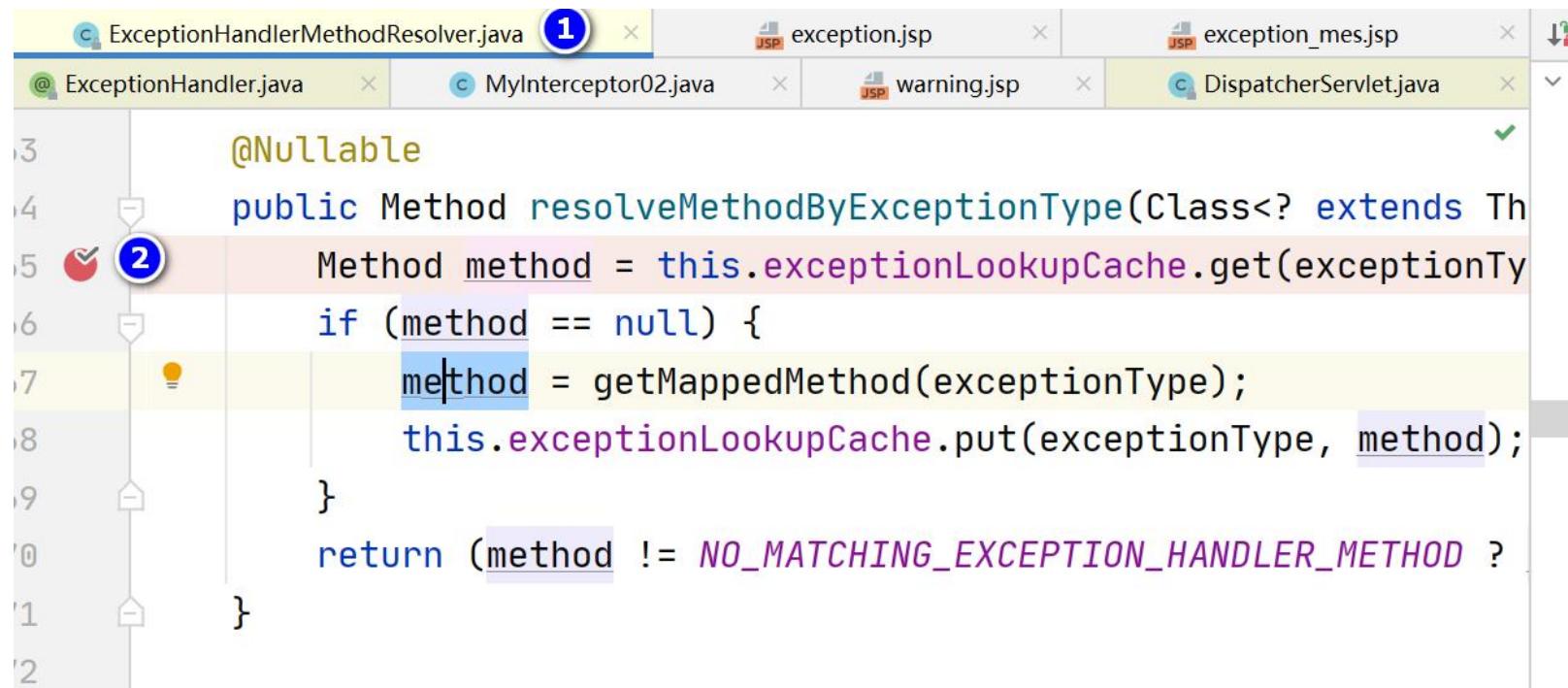
Pretty Raw Preview Visualize

朋友，发生异常，信息如下：

/ by zero

15.2.2 Debug 处理流程

- 看老师演示



```

3     @Nullable
4     public Method resolveMethodByExceptionType(Class<? extends Th
5     Method method = this.exceptionLookupCache.get(exceptionTy
6     if (method == null) {
7         method = getMappedMethod(exceptionType);
8         this.exceptionLookupCache.put(exceptionType, method);
9     }
10    return (method != NO_MATCHING_EXCEPTION_HANDLER_METHOD ?
11        method
12    }

```

15.3 全局异常

15.3.1 应用实例

- 应用实例需求

演示全局异常处理机制，ExceptionHandlerMethodResolver 内部若找不到 @ExceptionHandler 注解的话，会找 @ControllerAdvice 类的 @ExceptionHandler 注解方法，这样就相当于一个全局异常处理器

- 应用实例-代码实现

1. 创建

D:\idea_java_projects\springmvc\src\com\hspedu\web\exception\MyGlobalException.java

```
package com.hspedu.web.exception;
```

```
import org.springframework.web.bind.annotation.ControllerAdvice;  
import org.springframework.web.bind.annotation.ExceptionHandler;  
  
import javax.servlet.http.HttpServletRequest;  
  
/**  
 * @author 韩顺平  
 * @version 1.0  
 */  
//加入这个注解后，就表示是一个全局异常.  
@ControllerAdvice  
public class MyGlobalException {  
  
    //全局异常不管是哪个 Handler 抛出的异常，都可以捕获  
    @ExceptionHandler({ClassCastException.class, NumberFormatException.class})  
    public String localException(Exception ex, HttpServletRequest request){  
        System.out.println("全局异常信息是= " + ex.getMessage());  
        request.setAttribute("reason", ex.getMessage());  
        //如何将异常的信息带到下一个页面.  
        return "exception_mes";  
    }  
}
```

```
}
```

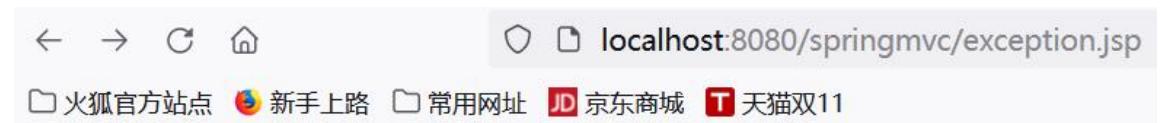
2. 修改 MyExceptionHandler.java，增加方法

```
@RequestMapping(value="/testGlobalException")  
public String global(){  
    //老韩解读  
    //1. 模拟 NumberFormatException  
    //2. 在老韩定义的局部异常中，没有对 NumberFormatException  
    //3. 所有就会去找全局异常处理  
    int num = Integer.parseInt("hello");  
    return "success";  
}
```

3. 修改 exception.jsp

```
<a href="testException01?num=0">点击测试异常</a><br><br>  
<a href="testGlobalException">点击测试全局异常</a><br><br>
```

4. 完成测试(页面方式) 浏览器 <http://localhost:8080/springmvc/exception.jsp>



测试异常

[点击测试异常](#)

[点击测试全局异常](#)



朋友，发生异常，信息如下：

For input string: "hello"

5. 完成测试(Postman 方式)

GET ▼ http://localhost:9998/testGlobalException

Params Auth Headers (8) Body ● Pre-req. Tests Settings

Query Params

	KEY	VALUE
	Key	Value

Body Cookies (1) Headers (6) Test Results 🌐 200

Pretty Raw Preview Visualize

朋友，发生异常，信息如下：

For input string: "hello"

15.3.2 Debug 处理流程

- 看老师演示
- 注意观察

```
ExceptionHandlerMethodResolver.java (1) × exception.jsp × exception_mes.jsp ×
ExceptionHandler.java × MyInterceptor02.java × warning.jsp × DispatcherServlet.java ×

3     @Nullable
4     public Method resolveMethodByExceptionType(Class<? extends Throwable> exceptionType) {
5         Method method = this.exceptionLookupCache.get(exceptionType);
6         if (method == null) {
7             method = getMappedMethod(exceptionType);
8             this.exceptionLookupCache.put(exceptionType, method);
9         }
10        return (method != NO_MATCHING_EXCEPTION_HANDLER_METHOD ? method : null);
11    }
12 }
```

15.3.3 异常处理时：局部异常 优先级高于 全局异常

15.4 自定义异常

15.4.1 应用实例

- 应用实例需求

通过`@ResponseStatus`注解，可以自定义异常的说明

- 应用实例-代码实现

1.

创

建

D:\idea_java_projects\springmvc\src\com\hspedu\web\exception\AgeException.java

```
package com.hspedu.web.exception;

import org.springframework.http.HttpStatus;
import org.springframework.web.bind.annotation.ResponseStatus;

/**
 * @author 韩顺平
 * @version 1.0
 */
@ResponseStatus(reason="年龄需要在 1-120 之间 ",value= HttpStatus.BAD_REQUEST)
public class AgeException extends RuntimeException{}
```

2. 修改 MyExceptionHandler.java, 增加方法

```
//抛出自定义异常
@RequestMapping(value="/testException02")
public String test02(){
    throw new AgeException();
}
```

3. 修改 exception.jsp, 增加超链接

[点击测试全局异常](testGlobalException)

[点击测试自定义异常](testException02)

4. 并完成测试，浏览器 <http://localhost:8080/springmvc/exception.jsp>





Apache Tomcat/8.0.50

5. 完成测试(Postman 方式)

GET ▼ http://localhost:9998/testException02

Params Auth Headers (8) Body ● Pre-req. Tests Settings

Query Params

	KEY	VALUE
	Key	Value

Body Cookies (1) Headers (6) Test Results 🌐 400 |

Pretty Raw Preview Visualize

HTTP Status 400 - 年龄需要在1-120之间~

type Status report

message 年龄需要在1-120之间~

description The request sent by the client was syntactically incorrect.

15.4.2 Debug 处理流程

- 看老师演示

- 注意观察

```
ExceptionHandlerMethodResolver.java ①
ExceptionHandler.java
MyInterceptor02.java
DispatcherServlet.java

3 @Nullable
4     public Method resolveMethodByExceptionType(Class<? extends Throwable> exceptionType) {
5         Method method = this.exceptionLookupCache.get(exceptionType);
6         if (method == null) {
7             method = getMappedMethod(exceptionType);
8             this.exceptionLookupCache.put(exceptionType, method);
9         }
10        return (method != NO_MATCHING_EXCEPTION_HANDLER_METHOD ? method : null);
11    }
12 }
```

15.5 SimpleMappingExceptionResolver

15.5.1 基本说明

- 基本说明
 1. 如果希望对所有异常进行统一处理，可以使用 **SimpleMappingExceptionResolver**
 2. 它将异常类名映射为视图名，即发生异常时使用对应的视图报告异常
 3. 需要在 **ioc** 容器中配置

15.5.2 应用实例

- 应用实例-需求

对数组越界异常进行统一处理，使用 SimpleMappingExceptionResolver

- 应用实例-代码实现

1. 修改 MyExceptionHandler.java，增加方法 test03

```
//统一的异常  
 @RequestMapping(value="/testException03")  
 public String test03(){  
     int[] arr = new int[]{3,9,10,190};  
     System.out.println(arr[90]);  
     return "success";  
 }
```

2. 配置 springDispatcherServlet-servlet.xml

```
<!-- 配置一个统一异常处理 -->  
<bean  
      class="org.springframework.web.servlet.handler.SimpleMappingExceptionResolver">  
      <property name="exceptionMappings">  
          <props>  
              <prop key="java.lang.ArrayIndexOutOfBoundsException">arrEx</prop>
```

```
</props>  
</property>  
</bean>  
<!-- 加入两个常规配置 --&gt;<br/>!-- 能支持 SpringMVC 高级功能，比如 JSR303 校验，映射动态请求 -->  
<mvc:annotation-driven></mvc:annotation-driven>  
!-- 将 SpringMVC 不能处理的请求交给 Tomcat，比如请求 css,js 等-->  
<mvc:default-servlet-handler/>
```

3. 创建/WEB-INF/pages/arrEx.jsp

```
<%--  
Created by IntelliJ IDEA.  
User: 韩顺平  
Version: 1.0  
Filename: arrEx  
--%>  
<%@ page contentType="text/html;charset=UTF-8" language="java" %>  
<html>  
<head>  
    <title>数组越界了</title>  
</head>
```

```
<body>  
<h1>数组越界了~</h1>  
</body>  
</html>
```

4. 修改 exception.jsp

```
<a href="testException03">  
    测试统一的异常[SimpleMappingExceptionResolver]</a>
```

5. 并完成测试(页面测试), 浏览器 <http://localhost:8080/springmvc/exception.jsp>



测试异常

[点击测试异常](#)

[点击测试全局异常](#)

[点击测试自定义异常](#)

[测试统一的异常\[SimpleMappingExceptionResolver\]](#)



数组越界了~

6. 完成测试(Postman)

POST <http://localhost:9998/testException03>

Params Auth Headers (9) Body Pre-req. Tests Settings

x-www-form-urlencoded

Key	Value
abc	123

Body Cookies (1) Headers (6) Test Results

Pretty Raw Preview Visualize

数组越界了~

15.5.3 对未知异常进行统一处理

- 应用实例-需求

对未知异常进行统一处理，使用 SimpleMappingExceptionResolver

- 应用实例-代码实现

1. 修改 MyExceptionHandler.java，增加方法 test04

```
//如果发生了没有归类的异常， 可以给出统一提示页面
@RequestMapping(value="/testException04")
public String test04(){
    String str = "hello";
    char c = str.charAt(10);
    return "success";
}
```

2. 配置 springDispatcherServlet-servlet.xml

```
<!-- 配置一个统一异常处理 -->
<bean
    class="org.springframework.web.servlet.handler.SimpleMappingExceptionResolver">
    <property name="exceptionMappings">
        <props>
            <prop key="java.lang.ArrayIndexOutOfBoundsException">arrEx</prop>
            <prop key="java.lang.Exception">otherEx</prop>
        </props>
    </property>
</bean>
```

```
</property>  
</bean>
```

3. 创建/WEB-INF/pages/otherEx.jsp

```
<%--  
     Created by IntelliJ IDEA.  
     User: 韩顺平  
     Version: 1.0  
     Filename: otherEx  
--%>  
<%@ page contentType="text/html;charset=UTF-8" language="java" %>  
<html>  
<head>  
    <title>未知异常信息</title>  
</head>  
<body>  
    <h1>朋友，系统发生了未知异常~</h1>  
</body>  
</html>
```

4. 修改 exception.jsp，增加超链接

```
<a href="testException03">  
    测试统一的异常[SimpleMappingExceptionResolver]</a>  
<br/><br/><a href="testException04">  
    统一未归类异常</a>
```

5. 并完成测试(页面测试), 浏览器 <http://localhost:8080/springmvc/exception.jsp>



测试异常

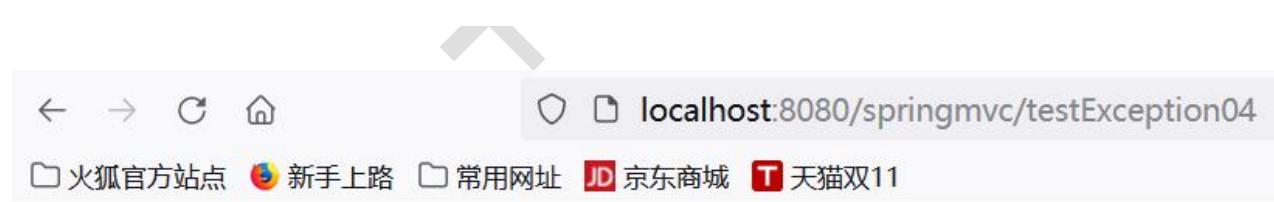
[点击测试异常](#)

[点击测试全局异常](#)

[点击测试自定义异常](#)

[测试统一的异常\[SimpleMappingExceptionResolver\]](#)

[统一未归类异常](#)



朋友，系统发生了未知异常~

6. 并完成测试(Postman 方式)

The screenshot shows the Postman application interface. At the top, it says "POST" and the URL "http://localhost:9998/testException04". Below this, there are tabs for "Params", "Auth", "Headers (9)", "Body", "Pre-req.", "Tests", and "Settings". The "Body" tab is currently selected. Under "Body", there is a dropdown menu set to "x-www-form-urlencoded". A table below has columns "Key" and "Value". There are four rows in the table, each with a "Delete" button. At the bottom of the table, there are buttons for "Pretty", "Raw", "Preview", and "Visualize". Above the table, there are tabs for "Body", "Cookies (1)", "Headers (6)", and "Test Results".

朋友，系统发生了未知异常~

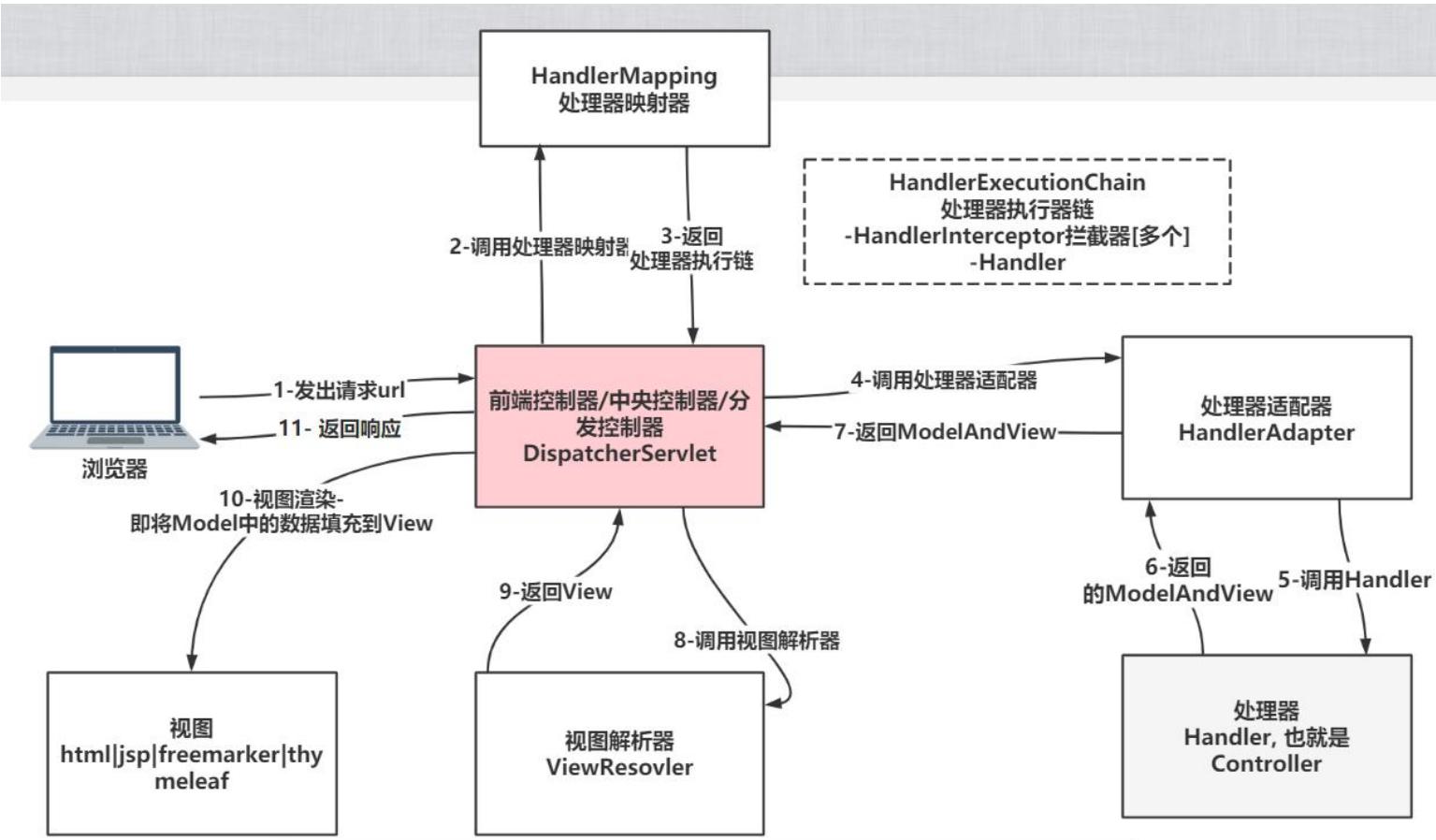
15.5.4 异常处理的优先级梳理

- 异常处理的优先级

局部异常 > 全局异常 > SimpleMappingExceptionResolver > tomcat 默认机制

16 SpringMVC 执行流程-源码剖析

- Debug SpringMVC 执行流程



- 准备的目标 Handler 和方法

```

package com.hspedu.web.debug;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.servlet.ModelAndView;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

```

```
/**  
 * @author 韩顺平  
 * @version 1.0  
 */  
  
@Controller  
  
public class HelloHandler {  
    /**  
     *  
     * @param request  
     * @param response  
     * @return  
     */  
  
    @RequestMapping(value = "/debug/springmvc")  
    public ModelAndView hello(HttpServletRequest request, HttpServletResponse response) {  
  
        ModelAndView modelAndView = new ModelAndView();  
        modelAndView.setViewName("ok");  
        modelAndView.addObject("name", "老韩");  
        return modelAndView;  
    }  
}
```

- 断点开始位置

```
DispatcherServlet.java | AbstractView.java | InternalResourceView.java | Help.java  
InvokerHandlerMethod.java | ServletInvocableHandlerMethod.java | RequestWrapper.java  
921     * for the actual dispatching.  
922     */  
923     @Override  
924     protected void doService(HttpServletRequest request) {  
925         logRequest(request);  
926         // Keep a snapshot of the request and response  
         // to be able to restore the original state  
927     }  
928 }
```

-从这里开始Debug整个执行流程

16.1 作业布置

1. 把前面老师讲过的 SpringMVC 异常处理相关代码和案例，自己写一遍
-一定要自己写一遍,否则没有印象, 理解不会深入
2. 简述 SpringMVC 执行流程，并画出示意图(**一定要自己动手画..**)
3. 把老师 Debug 过的 SpringMVC 执行流程源码，自己也走一下，加深理解(**不用每一条语句都 debug, 主要是梳理流程**)