

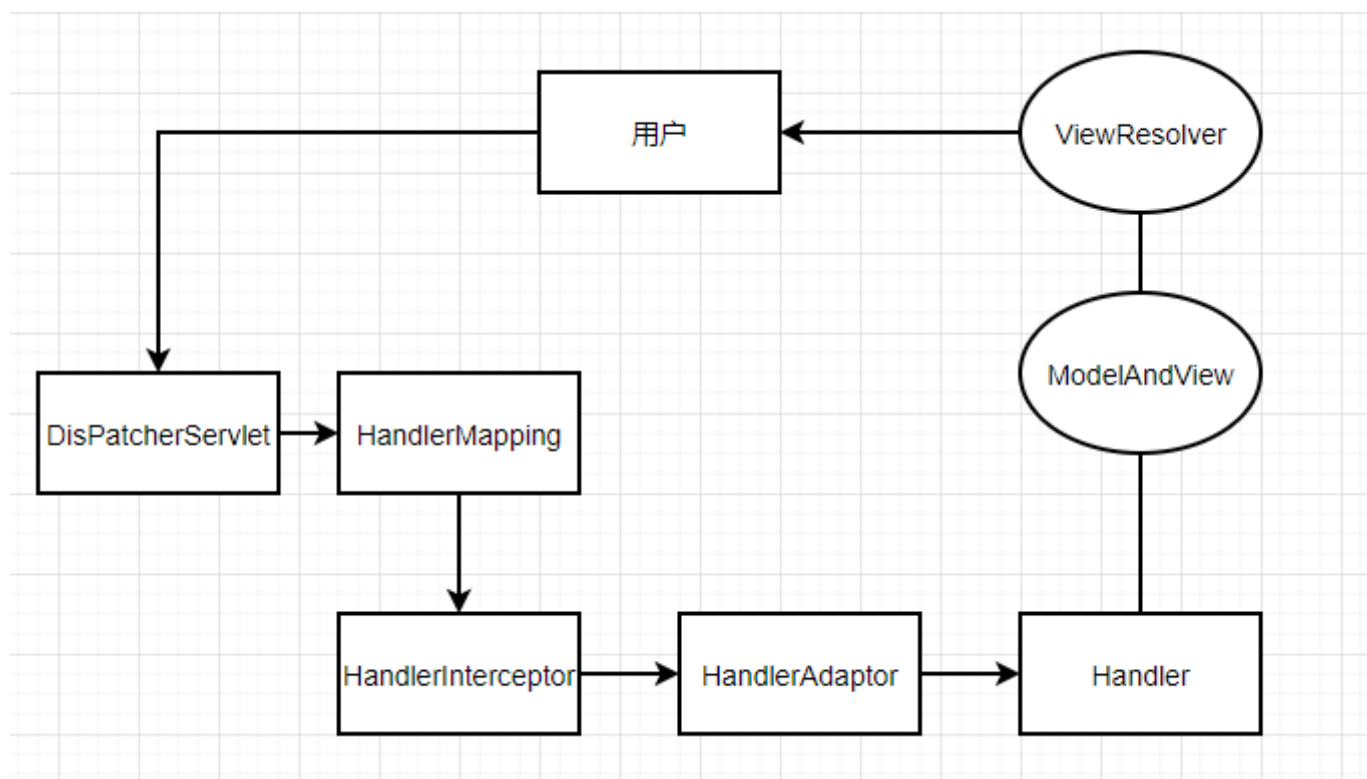
SpringMVC

Springmvc是一个Spring的子模块，可以很便捷的跟Spring进行整合，不用很复杂的配置

MVC设计模式

mvc模式的应用程序分为Controller, view, model三层，controller负责接受客户的请求，调用model生成业务数据，并传递给view视图

Springmvc核心组件如下



DispatcherServlet 相当于一个路由器

HandlerMapping相当于路由规则

HandlerAdaptor在Handler之前执行，主要用于表单验证，数据类型转换等操作

Handler是业务逻辑的处理器，也就是Controller

Springmvc使用流程

- 配置tomcat
- 引入Springmvc依赖

```

-->
2 <dependency>
3   <groupId>org.springframework</groupId>
4   <artifactId>spring-webmvc</artifactId>
5   <version>5.3.22</version>
6 </dependency>

```

- 在web.xml配置DispatcherServlet (还是Servlet)

```

1 <!DOCTYPE web-app PUBLIC
2   "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
3   "
4   http://java.sun.com/dtd/web-app\_2\_3.dtd
5   ">
6
7
8 <web-app>
9   <display-name>Archetype Created Web Application</display-name>
10
11   <servlet>
12     <servlet-name>DispatcherServlet</servlet-name>
13     <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
14     <init-param>
15       <param-name>contextConfigLocation</param-name>
16       <param-value>classpath:Spring-mvc.xml</param-value>
17     </init-param>
18   </servlet>
19
20   <servlet-mapping>
21     <servlet-name>DispatcherServlet</servlet-name>
22     <url-pattern>/</url-pattern>
23   </servlet-mapping>
24 </web-app>

```

- 编写Spring容器配置

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <beans xmlns="
3   http://www.springframework.org/schema/beans

```

```

3      xmlns:xsi="
http://www.w3.org/2001/XMLSchema-instance
"

4      xmlns:context="
http://www.springframework.org/schema/context
"

5      xsi:schemaLocation="
http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans.xsd

6
http://www.springframework.org/schema/context

http://www.springframework.org/schema/context/spring-context.xsd
">
7 <!--    扫描控制器（Handler）注册为bean-->
8     <context:component-scan base-package="com.hth"/>
9 <!--    配置视图解析器    /index.jsp    只需要访问index即可，因为配置了前缀和后缀-->
10    <bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
11        <property name="suffix" value=".jsp"/>
12        <property name="prefix" value="/" />
13    </bean>
14 </beans>

```

• 编写Handler

```

1  import org.springframework.stereotype.Controller;
2  import org.springframework.web.bind.annotation.RequestMapping;
3
4  @Controller
5  public class HelloController {
6
7      @RequestMapping("/")
8      public String index(){
9          return "index";
10     }
11     @RequestMapping("/hello")
12     public String hello(){

```

```
13         return "Hello World";
14     }
15 }
```

- 启动Tomcat



← ↻ ⓘ localhost:8080/Springmvc_war/

Hello World!

```
1 <html>
2 <body>
3 <h2>Hello World!</h2>
4 </body>
5 </html>
6 <!--index.jsp-->
```

Springmvc注解

springmvc的控制器类主要有以下两种注解

@RequestMapping

@Controller

- @Controller

表明该类是一个控制器类，由Spring IOC容器管理，并且可以接受DispatcherServlet发送过来的请求

- @RequestMapping

RequestMapping注解表明了URL与控制器方法的映射关系。若在类上面标注，则类中的所有方法的url都会带上类注解的url作为前缀

RequestMapping注解有三种属性

- value, 默认属性，表明请求的url
- method, 表明请求的方法
- params, 表明请求的参数

主要讲一下params属性

```
1 @RequestMapping(value = "/hello", params = {"name", "id"}, method = RequestMethod.GET)
2 public String hello(){
```

```
3     return "hello";
4 }
```

这是一个接受get请求，当指定了params参数时，则请求中必须带有name和id两个query字段才能匹配到这个方法

← → ↻ ⓘ localhost:8080/Springmvc_war/hello

HTTP状态 400 - 错误的请求

类型 状态报告

消息 Parameter conditions "name, id" not met for actual request parameters:

描述 由于被认为是客户端对错误（例如：畸形的请求语法、无效的请求信息帧或者虚拟的请求路由），服务器无法或不会处理当前请求。

Apache Tomcat/9.0.65

当带上了参数才能匹配成功

ⓘ localhost:8080/Springmvc_war/hello?name=zha&id=1

hello
world
!!!

可以在方法的参数中声明与params里相同名字的参数，那么springmvc会将请求路径中相应的值赋予参数

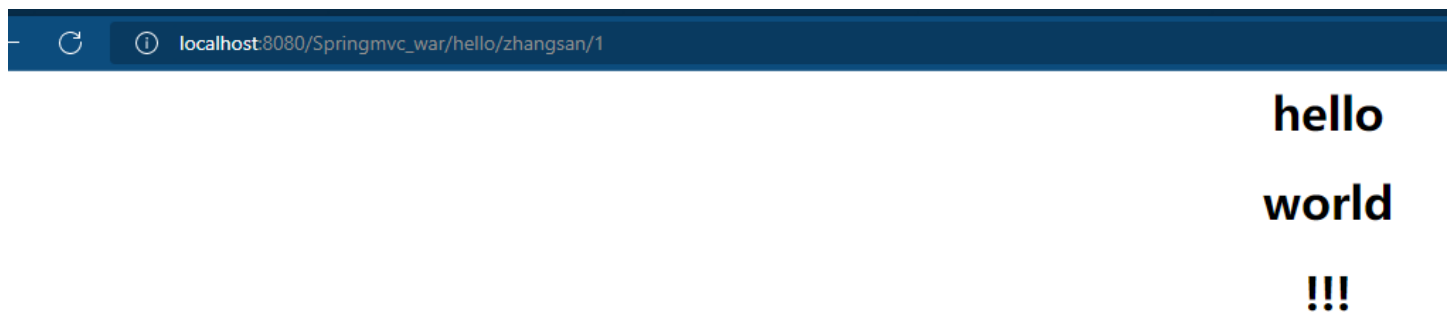
```
1 @RequestMapping(value = "/hello",params = {"name","id"},method = RequestMethod.GET)
2 public String hello(String name,int id){
3     System.out.println(name+id);
4     return "hello";
5 }
```

也可以通过@RequestParam注解将请求路径中的值赋予相应的参数

```
1 @RequestMapping(value = "/hello",params = {"name","id"},method = RequestMethod.GET)
2 public String hello(@RequestParam("name") String haha, int id){
3     System.out.println(haha+id);
4     return "hello";
5 }
```

在Rest风格，请求中的变量参数会在路径里，而不是在query字段中，如下所示，可以用@PathVariable注解进行参数提取

```
1 @RequestMapping("/hello/{name}/{id}")
2 public String world(@PathVariable("name") String name,@PathVariable("id") int id){
3     System.out.println(name+id);
4     return "hello";
5 }
```



结果返回

Spring MVC一个Handler方法的返回结果一般是一个视图的名称，其将会由ViewResolver解析之后返回一个视图给客户端。另一种则会前后端分离用到的，只传回模型数据，而不是给客户端传回一个html视图

SpringMVC这两种不同的返回可由两个注解来控制 @Controller 和 @RestController

@Controller

此注解标注在一个Handler类上，表明该类的方法返回的是一个视图

如下，这个方法将会寻找服务器根目录下的index.jsp文件并返回

```
1 @RequestMapping("/")
2 public String index(){
3     return "index";
4 }
```

返回视图分为转发和重定向

默认的我们访问http:127.0.0.1/ 这个URL时，会给我们返回index.jsp文件，而浏览器的地址栏却没有变成http:127.0.0.1/index.jsp

如何写转发和重定向视图？如下所示，只需要在返回视图的字符串前加上"redirect:"即可实现重定向，而转发则安正常的写法即可

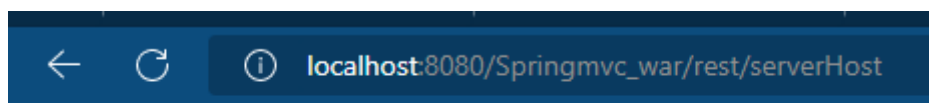
```
1 @RequestMapping("/redirect")
2 public String redirect(){
3     return "redirect:/haha.jsp";
4 }
```

@RestController

此注解表示该类的所有方法将结果直接返回给客户端，而不是经过ViewResolver解析为视图之后向客户端发送一个视图文件

如下这个类，访问serviceHost方法，将会返回一个windows字符串

```
1 @RequestMapping("/rest")
2 @org.springframework.web.bind.annotation.RestController
3 public class RestController {
4
5     @RequestMapping("/serverHost")
6     public String serviceHost(){
7         return "windows";
8     }
9 }
```



windows

也可以在标注了@Controller的Handler类下的方法上使用@ResponseBody注解，此时同样可以达到上面的效果

数据绑定

数据绑定即将客户端传来的参数解析成为处理方法的参数，在业务逻辑中使用。

如前面所说的@RequestParam 和 @PathVariable都可以将客户端参数与处理方法参数建立映射关系。

不过大多数情况下，只要参数名字对应上，就不用额外使用注解建立联系。

映射Cookie

可以使用@CookieValue在方法参数前面，从而获取cookie参数值

使用Java类绑定参数

这种情况一般用在表单上，将表单中的各个输入项的 `name` 属性设置为与Java类中的属性相同的名字，则HandlerAdaptor会将表单数据解析为Java类

示例如下，（Java类各属性要设置getter和setter方法，用lombok库解决）

如果要级联，则在name属性设置 `xx.xx` 以小数点表示属性的子属性

```
1  @RequestMapping(value="/bean",method = RequestMethod.POST)
2  public String bindBean(User user){
3      System.out.println(user);
4      return "index";
5  }
6
7  <form action="/rest/bean" method="post" enctype="application/x-www-form-urlencoded">
8
9      <input type="text" name="name">
10     <input type="text" name="age">
11     <input type="submit" value="提交">
12 </form>
13
14 name 和 age相互对应
15
16 public class User {
17     private String name;
18     private int age;
19
20     public String getName() {
21         return name;
22     }
23
24     public int getAge() {
25         return age;
```



```

26     }
27
28     public void setName(String name) {
29         this.name = name;
30     }
31
32     public void setAge(int age) {
33         this.age = age;
34     }
35
36     @Override
37     public String toString() {
38         return "User{" +
39             "name='" + name + '\'' +
40             ", age=" + age +
41             '}';
42     }
43 }

```

基本数据类型及其包装类

如上所诉，Springmvc基本采用名称来进行参数匹配，否则就用注解来映射。基本数据类型直接声明即可

有时候可能某个基本数据类型输入为空，这种情况下可以用包装类来进行声明。如下的User类。age由int型改成包装类Integer，仍然可以正常使用

```

1  public class User {
2      private String name;
3      private Integer age;
4

```

数组

```

1  @RequestMapping("/array")
2  public String array(String[] names){
3      for (int i = 0; i < names.length; i++) {

```

```

4         System.out.println(names[i]);
5     }
6     return "666";
7
8     //
    http://localhost:8080/Springmvc_war/bind/array?names=jiejie&names=kunkun
9
10    //输出jiejie
11    //kunkun
12 }

```

List和Map

这种不能直接解析，需要一个包装类进行包装，如下所示list的使用

Map与list类似，不过name 需要改成 xx[aa]的形式，而aa即为map中的一个键

```

1 <body>
2     <form action="/bind/list" method="post">
3         <input type="text" name="strs[0]">
4         <input type="text" name="strs[1]">
5         <input type="text" name="strs[2]">
6         <input type="submit" value="submit">
7     </form>
8 </body>
9
10 public class StringList {
11     public List<String> strs;
12
13     public List<String> getStrs() {
14         return strs;
15     }
16
17     public void setStrs(List<String> strs) {
18         this.strs = strs;
19     }
20 }
21 @RequestMapping(value = "/list",method = RequestMethod.POST)
22 public String list(StringList list){
23     for(String s : list.getStrs()){

```

```
24         System.out.println(s);
25     }
26     return "666";
27
28 }
```

JSON

springmvc中的json转换需要用到 fastjson这个包

```
1 <dependency>
2   <groupId>com.alibaba</groupId>
3   <artifactId>fastjson</artifactId>
4   <version>1.2.32</version>
5 </dependency>
```

模型数据视图解析

首先需要了解jsp 的四大作用域

pageContext、request、session、application

而在Spring MVC的视图模型解析中，一般不需要用到pageContext这个作用域，所以只需要注意后三个即可

五种向视图添加模型数据的方法

- Map
- Model
- ModelAndView
- @SessionAttribute
- @ModelAttribute

Map

Springmvc 的HandlerAdaptor很智能，参数所给的map的作用域是requestScope

```
1 @RequestMapping("/map")
```

```
2 public String map(Map<String,User> map){
3     User user = new User();
4     user.setId(1L);
5     user.setName("张三");
6     map.put("user",user);
7     return "view";
8 }
9
10 ${requestScope.user}
```

Model

```
1 @RequestMapping("/model")
2 public String model(Model model){
3     User user = new User();
4     user.setId(1L);
5     user.setName("张三");
6     model.addAttribute("user",user);
7     return "view";
8 }
```

ModelAndView

modelAndView 不能由HandlerAdaptor给出，需要自己创建

```
1 @RequestMapping("/modelAndView3")
2 public ModelAndView modelAndView3(){
3     User user = new User();
4     user.setId(1L);
5     user.setName("张三");
6     ModelAndView modelAndView = new ModelAndView("view");
7     modelAndView.addObject("user",user);
8     return modelAndView;
9 }
```

HttpServletRequest

除了以上三种，也可以用原生servlet方式进行设置

@ModelAttribute

```
1 @ModelAttribute
2 public User getUser(){
3     User user = new User();
4     user.setId(1L);
5     user.setName("张三");
6     return user;
7 }
```

如上所示，标注了这个注解的方法的作用域范围是整个Controller，即这个类中所有调用的视图都能获取到这个user对象，

可以在getUser方法中加入参数 Map 或者 Model ，像上面一样设置

```
1 @ModelAttribute
2 public void getUser(Model model){
3     User user = new User();
4     user.setId(1L);
5     user.setName("张三");
6     model.addAttribute("user",user);
7 }
```

@SessionAttribute

用法和@ModelAttribute类似，

同样的像HttpServletRequest，可以通过他获取session对象，也可以设置模型数据


```
1 @RequestMapping("/session")
2 public String session(HttpServletRequest request){
3     HttpSession session = request.getSession();
4     User user = new User();
5     user.setId(1L);
6     user.setName("张三");
7     session.setAttribute("user",user);
8     return "view";
9 }
```

文件上传下载

需要用到以下依赖

```
1 <dependency>
2   <groupId>commons-io</groupId>
3   <artifactId>commons-io</artifactId>
4   <version>2.5</version>
5 </dependency>
6 <dependency>
7   <groupId>commons-fileupload</groupId>
8   <artifactId>commons-fileupload</artifactId>
9   <version>1.3.3</version>
10 </dependency>
```

springmvc表单标签库

 `<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form" %>`

Spring MVC 数据校验

Spring MVC 提供了两种数据校验的方式：1、基于 Validator 接口。2、使用 Annotation JSR - 303 标

准进行校验。

一般 采用注解校验



@Null 被注解的元素必须为null

@NotNull 被注解的元素不能为null

@Min(value) 被注解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) 被注解的元素必须是一个数字，其值必须小于等于指定的最大值

@Email 被注解的元素必须是电子邮箱地址

@Pattern 被注解的元素必须符合对应的正则表达式

@Length 被注解的元素的大小必须在指定的范围内

@NotEmpty 被注解的字符串的值必须非空

Null 和 Empty 是不同的结果，String str = null，str 是 null，String str = ""，str 不是 null，其值为

空。

```
<!-- JSR-303 -->

<dependency>

<groupId>org.hibernate</groupId>

<artifactId>hibernate-validator</artifactId>

<version>5.3.6.Final</version>

</dependency>

<dependency>

<groupId>javax.validation</groupId>

<artifactId>validation-api</artifactId>

<version>2.0.1.Final</version>

</dependency>

<dependency>

<groupId>org.jboss.logging</groupId>

<artifactId>jboss-logging</artifactId>

<version>3.3.2.Final</version>

</dependency>
```

杂项

因为默认的DispatcherServlet只会处理JSP文件

处理 Spring MVC 无法加载静态资源，在 web.xml 中添加配置即可。

```
<servlet-mapping>
    <servlet-name>default</servlet-name>
    <url-pattern>*.js</url-pattern>
</servlet-mapping>
```

如果出现中文乱码问题，只需在 web.xml 添加 Spring MVC 自带的过滤器即可。

```
<filter>
    <filter-name>encodingFilter</filter-name>
    <filter-
class>org.springframework.web.filter.CharacterEncodingFilter</filter-class>
    <init-param>
        <param-name>encoding</param-name>
        <param-value>UTF-8</param-value>
    </init-param>
</filter>
```

处理 @ResponseBody 中文乱码，在 springmvc.xml 中配置消息转换器。

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