04_SpringMVC

笔记本:	Spring&SpringMVC		
创建时间:	2019/9/3 17:55	**更新时间:**	2019/9/4 8:56
作者:	韩延兵		

1.HelloWorld

- 1）创建一个Maven版的Web工程，通过依赖导入以下jar包

```
<dependencies>
        <!-- 只需要将context的jar包的依赖配置上，其他依赖的jar包会自动导入 -->
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-context</artifactId>
            <version>4.0.0.RELEASE</version>
        </dependency>
        <!-- 只需要导入spring-webmvc的jar包，Maven会自动将它依赖的spring-web的
jar导入 -->
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-webmvc</artifactId>
            <version>4.0.0.RELEASE</version>
        </dependency>
    </dependencies>
```

- 2）在web.xml中配置DispatcherServlet
  - 快捷键是Alt+/选倒数第二项
  - 配置SpringMVC配置文件的路径
  - 配置url-pattern标签的值为 /

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaeehttp://java.sun.com/xml/ns/javaee/
app_2_5.xsd" id="WebApp_ID" version="2.5">
    <!-- 配置DispatcherServlet，快捷键是：Alt + / 选择倒数第二项 -->
    <servlet>
        <servlet-name>springDispatcherServlet</servlet-name>
        <servlet-
class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
        <init-param>
            <param-name>contextConfigLocation</param-name>
            <!-- 配置SpringMVC的配置文件的路径 -->
            <param-value>classpath:springmvc.xml</param-value>
        </init-param>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>springDispatcherServlet</servlet-name>
        <!-- 注意：url-pattern标签中配置 /，它会处理除jsp页面之外的其他请求 -->
        <url-pattern>/</url-pattern>
    </servlet-mapping>
</web-app>
```

- 3）在类路径下创建SpringMVC的配置文件springmvc.xml

- 配置自动扫描的包
- 配置视图解析器

```
<?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/beanshttp://www.springfra
beans.xsd
http://www.springframework.org/schema/contexthttp://www.springframework.org/schema/cont
context-4.0.xsd">
    <!-- 设置自动扫描的包 -->
    <context:component-scan base-package="com.atguigu.springmvc">
</context:component-scan>

    <!-- 配置视图解析器 -->
    <bean id="viewResolver"
class="org.springframework.web.servlet.view.InternalResourceViewResolver">
        <!-- 设置前缀 -->
        <property name="prefix" value="/WEB-INF/views/"></property>
        <!-- 设置后缀 -->
        <property name="suffix" value=".jsp"></property>
    </bean>
</beans>
```

- 4）创建一个类HelloWorld
  - 在类上添加@Controller注解标识它是一个处理器
  - 在该类中创建一个处理请求的方法并在方法上添加@RequestMapping注解映射请求地址

```
@Controller //标识当前类是一个控制器（处理器）
public class HelloWorld {
    /*
     * 方法的返回值会被SPringMVC配置文件中配置的视图解析器
InternalResourceViewResolver解析为一个真实的物理视图，
     * 会将视图解析器中的前缀+方法的返回值+视图解析器中的后缀，
     * 即：/WEB-INF/views/success.jsp
     *
     *
     */
    @RequestMapping("/hello")
    public String helloSpringMVC() {
        System.out.println("Hello SpringMVC");
        return "success";
    }
}
```

- 5）创建视图
  - 在WEB-INF目录下创建views目录
  - 在views目录下创建success.jsp页面
- 6）测试
  - 在WebContent目录下创建index.jsp页面
  - 在index.jsp页面中创建一个超链接，超链接的地址要与@RequestMapping中的地址保存一致

```
<a href="${pageContext.request.contextPath }/hello">Hello SpringMVC</a>
```

2.@RequestMapping注解

- 用来映射请求地址
  - 通过该注解的value属性指定要映射的请求地址，value属性名可以省略
- 该注解可以添加到类上也可以添加到方法上

```
/*
     * @RequestMapping注解的属性
     *      1.value属性：用来设置要映射的请求地址
     *          该属性的类型是一个String类型的数组，如果只映射一个请求地址，那么数组的大括号可以省略不写，
     *          而且value属性名也可以省略不写
     *      2.method属性：用来设置要映射的请求方式
     *          注意：如果没有指定该属性，那么处理请求的方法（目标方法）只看映射的请求地址，不管请求方式
     */
    @RequestMapping(value= {"/testValue","/testValue2"})
    public String testValue() {
        System.out.println("测试@RequestMapping的value属性");
        return SUCCESS;
    }

    @RequestMapping(value="testMethod",method=RequestMethod.GET,
            params= {"username=admin","age!=18"})
    public String testMethod() {
        System.out.println("测试@RequestMapping的method属性");
        return SUCCESS;
    }
```

3.@RequestParam注解

- 用来映射请求参数
  - 通过该注解的value属性设置要映射的请求参数，value属性名可以省略
- 该注解只能用在入参的前面

```
/*
     * @RequestParam注解
     *  作用：用来设置映射的请求参数
     *  如果处理器方法的入参的参数名与请求参数的参数名一致，那么可以省略该注解，不过不建议这样
     *  该注解中的属性：
     *  -★value属性：用来设置要映射的请求参数的参数名
     *  -（了解）required属性：用来设置该请求参数是否是必须的，默认是true
     *      -（了解）defaultValue属性：给请求参数设置一个默认值
     */
    @RequestMapping("/testRequestParam")
    public String testRequestParam(@RequestParam("username") String user,
            @RequestParam(value="age",required=false,defaultValue="0") int nianling) {
        System.out.println("测试@RequestParam注解");
        System.out.println("获取的用户名是："+user);
        System.out.println("获取的用户的年龄是："+nianling);
        return "success";
    }
```

4.处理器（Controller）方法中支持的原生ServletAPI

- Controller方法中可以直接传入HttpServletRequest、HttpServletResponse和HttpSession类型的参数，在Handler的方法中可以直接使用

```
/*
     *  MVC 的 Handler方法可以接受哪些 ServletAPI 类型的参数
        1)  ★HttpServletRequest
        2)  ★HttpServletResponse
        3)  ★HttpSession
        4)  java.security.Principal
        5)  Locale
        6)  InputStream
        7)  OutputStream
        8)  Reader
        9)  Write
     *
     */
    @RequestMapping("/testServletAPI")
    public String testServletAPI(HttpServletRequest request ,
HttpServletResponse response) {
        //获取用户名和密码
        String username = request.getParameter("username");
        String password = request.getParameter("password");
        System.out.println(username);
        System.out.println(password);
        return "success";
    }
```

5.处理器方法入参为POJO

- SpringMVC可以将表单中的数据自动设置到POJO的属性中，唯一的要求就是表单项的name属性值要与POJO类的属性名保持一致，而且还支持级联属性赋值
  - POJO类

```
public class Employee {
    private Integer id;
    private String lastName;
    private String email;
    private Department dept;


public class Department {
    private Integer id;
    private String deptName;
```

- 表单

```
<h1>处理POJO入参</h1>
    <form action="${pageContext.request.contextPath }/testPOJO"
method="post">
        <!-- 表单中的name属性值一定要与POJO的属性名保持一致 -->
        工号：<input type="text" name="id"><br>
        姓名：<input type="text" name="lastName"><br>
        邮箱：<input type="text" name="email"><br>
        部门编号：<input type="text" name="dept.id"><br>
        部门名称：<input type="text" name="dept.deptName"><br>
        <input type="submit">
    </form>
```

- 处理器方法

```
/*
     * Spring MVC 会按请求参数名和 POJO属性名进行自动匹配，自动为该对象填充属性值。支持级联属性。
     */
    @RequestMapping("/testPOJO")
    public String testPOJO(Employee employee) {
        System.out.println("员工的信息是: "+employee);
        return "success";
    }
```

6.处理响应数据

- 1）方法的返回值设置为ModelAndView对象

```
// （了解）处理响应数据方式一：方法的返回值设置为ModelAndView
    @RequestMapping("/testModelAndView")
    public ModelAndView testModelAndView() {
        //1.创建ModelAndView对象
        ModelAndView mv = new ModelAndView();
        //2.设置模型数据
        mv.addObject("employee", new Employee(1, "韩总", "hybing@atguigu.com", new Department(101, "开发部")));
        //3.设置视图名
        mv.setViewName("success");
        return mv;
    }
```

- 2）方法的返回值仍是String，入参为Map、Model或ModelMap
  - 放到map中的数据会自动放到request域中

```
/*
     * ★处理响应数据方式二：Handler的方法的入参为Map、Model、ModelMap
     *  不管方法的入参是Map、Model还是ModelMap，SpringMVC都会转换为一个ModelAndView对象
     *
     */
    @RequestMapping("/testMap")
    public String testMap(Map<String , Object> map) {
        //向map中添加一个Employee，最终会放到request域中
        map.put("employee", new Employee(1, "韩总", "hybing@atguigu.com", new Department(101, "开发部")));
        return "success";
    }
```

7.重定向

- 如果Handler的方法的返回值前面有forward:或redirect:前缀，那么将直接进行转发或重定向操作

```
@RequestMapping("/testRedirect")
    public String testRedirect() {
        System.out.println("测试重定向");
        return "redirect:/redirect.jsp";
    }
```

8.@ResponseBody注解

- 添加了该注解的方法的返回值将执行响应给浏览器
- 该注解可以添加到类上也可以添加到方法上
  - 添加到类上时SpringMVC的配置文件中需要配置<mvc:annotation-driven></mvc:annotation-driven>标签才生效

```
    @ResponseBody
    @RequestMapping("/testResponseBody")
    public String testResponseBody() {
        System.out.println("测试@ResponseBody注解");
        return "success";
    }
```

9.处理静态资源

- 在SpringMVC的配置文件中配置两个标签即可

```
<!-- 配置处理静态资源 -->
    <mvc:default-servlet-handler />
    <!-- 配置处理静态资源之后，还需要配置以下标签 -->
    <mvc:annotation-driven></mvc:annotation-driven>
```