第11章 Spring MVC的核心类和注解

湖南科技大学
计算机科学与工程学院

学习目标/Target

- 了解Spring MVC核心类的作用
- 掌握@Controller注解的使用
- 掌握@RequestMapping注解的使用
- 掌握请求的映射方式

章节概述/ Summary

自JDK 5推出以来，注解已成为Java知识体系不可缺少的一部分。Spring MVC在Spring 2.5之后也新增了基于注解的Controller形式。基于注解的Controller简化了XML文件配置，极大地提高了开发效率。本章将对Spring MVC的核心类和注解进行详细地讲解。

目录/Contents

11.1 DispatcherServlet

11.1 DispatcherServlet

- 了解DispatcherServlet类，能够说出DispatcherServlet核心类的作用有哪些

11.1 DispatcherServlet

DispatcherServlet作用

DispatcherServlet是Spring MVC的核心类，也是Spring MVC的流程控制中心，也称为Spring MVC的前端控制器，它可以拦截客户端的请求。拦截客户端请求之后，DispatcherServlet会根据具体规则将请求交给其他组件处理。所有请求都要经过DispatcherServlet进行转发处理，这样就降低了Spring MVC组件之间的耦合性。

11.1 DispatcherServlet

DispatcherServlet案例编写说明

DispatcherServlet的本质是一个Servlet，可以在web.xml文件中完成它的配置和映射。参考第10章Spring MVC入门程序，在IDEA中创建一个名称为chapter11的Maven Web项目。需要注意的是，如无特殊说明，本章的所有案例都将在chapter11项目中开发和运行。项目创建完成之后，在项目web.xml文件中配置DispatcherServlet。

11.1 DispatcherServlet

web.xml中对DispatcherServlet的配置分为两个方面。一是配置Spring MVC的前端控制器，二是配置映射的URL路径。

1. 配置Spring MVC的前端控制器：

```
<servlet>
    <servlet-name>DispatcherServlet</servlet-name>
    <servlet-class>
        org.springframework.web.servlet.DispatcherServlet
    </servlet-class>
    <!-- 配置初始化参数，用于读取Spring MVC的配置文件 -->
    <init-param>
      <param-name>contextConfigLocation</param-name>
      <param-value>classpath:spring-mvc.xml</param-value>
    </init-param>
    <!-- 应用加载时创建--> <load-on-startup>1</load-on-startup>
</servlet>
```

2. 配置映射的URL路径：

```
<servlet-mapping>
    <servlet-name>DispatcherServlet</servlet-name>
    <url-pattern>/</url-pattern>
  </servlet-mapping>
```

WEB-INF文件夹下默认配置文件命名规则

如果web.xml没有通过<init-param>元素指定DispatcherServlet初始化时要加载的文件，则应用程序会去WEB-INF文件夹下寻找并加载默认配置文件，默认配置文件的名称规则如下所示。

```
[servlet-name]-servlet.xml
```

其中，[servlet-name]指的是web.xml文件中<servlet-name>元素的值；“-servlet.xml”是配置文件名的固定拼接。

<load-on-startup>元素取值

<load-on-startup>元素取值分为三种情况：

（1）如果<load-on-startup>元素的值为正整数或者0，表示在项目启动时就加载并初始化这个Servlet，值越小，Servlet的优先级越高，就越先被加载；

（2）如果<load-on-startup>元素的值为负数或者没有设置，则Servlet会在被请求时加载和初始化；

（3）如果<load-on-startup>元素的值为1，表明DispatcherServlet会在项目启动时加载并初始化。

11.2 @Controller注解

11.2 @Controller注解

- 掌握@Controller注解，能够在程序中熟练运用@Controller注解

11.2 @Controller注解

@Controller注解作用

在Spring MVC框架中，传统的处理器类需要直接或间接地实现Controller接口，这种方式需要在Spring MVC配置文件中定义请求和Controller 的映射关系。当后台需要处理的请求较多时，使用传统的处理器类会比较繁琐，且灵活性低，对此，Spring MVC框架提供了@Controller注解。使用@Controller注解，只需要将@Controller注解标注在普通Java类上，然后通过Spring的扫描机制找到标注了该注解的Java类，该Java类就成为了Spring MVC的处理器类。

11.2 @Controller注解

基于@Controller注解的处理器类示例代码

```
import
org.springframework.stereotype.Controller;
…
@Controller        //标注@Controller注解
public class FirstController{
    …
}
```

11.2 @Controller注解

Spring MVC配置文件的类包扫描配置信息

```
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:context="http://www.springframework.org/schema/context"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   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">
    <!-- 配置要扫描的类包 -->
     <context:component-scan base-package="com.itheima.controller"/>
          ......
</beans>
```

11.2 @Controller注解

Spring扫描配置文件范围

Spring MVC的配置文件被加载时，Spring会自动扫描com.itheima.controller类包及其子包下的Java类。如果被扫描的Java类中带有@Controller、@Service等注解，则把这些类注册为Bean并存放在Spring中。

与传统的处理器类实现方式相比，使用@Controller注解的方式显然更加简单和灵活。因此，在实际开发中通常使用@Controller注解来定义处理器类。

11.3 @RequestMapping注解

11.3.1 @RequestMapping注解的使用

- 掌握@RequestMapping注解的使用，能够在程序中熟练运用@RequestMapping注解

11.3.1 @RequestMapping注解的使用

@RequestMapping注解作用

@RequestMapping注解用于建立请求URL和Handler（处理器）之间的映射关系，该注解可以标注在方法上和类上。下面分别对@RequestMapping注解的这两种使用方式进行介绍。

11.3.1 @RequestMapping注解的使用

方式一——标注在方法上

当@RequestMapping注解标注在方法上时，该方法就成了一个可以处理客户端请求的Handler（处理器），它会在Spring MVC接收到对应的URL请求时被执行。Handler在浏览器中对应的访问地址，由项目访问路径+处理方法的映射路径共同组成。

11.3.1 @RequestMapping注解的使用

接下来通过一个案例演示@RequestMapping注解标注在方法上的使用。

STEP 01

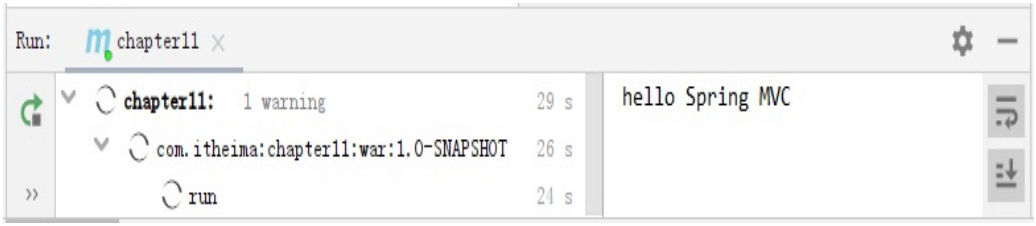
创建FirstController类，在类中创建sayHello()方法，用来处理客户端请求。

```
@Controller
public class FirstController {
        @RequestMapping(value="/firstController")
        public void sayHello(){
        System.out.println("hello Spring MVC");
    }
}
```

11.3.1 @RequestMapping注解的使用

STEP 02

启动项目，在浏览器中访问http://localhost:8080/chapter11/firstController，控制台打印输出信息。



由运行结果可知，sayHello()方法被成功执行，说明@RequestMapping注解标注在方法上时，成功建立了请求URL和处理请求方法之间的对应关系。

11.3.1 @RequestMapping注解的使用

方式二—标注在类上

当@RequestMapping注解标注在类上时，@RequestMapping的value属性值相当于本处理器类的命名空间，即访问该处理器类下的任意处理器都需要带上这个命名空间。@RequestMapping标注在类上时，其value属性值作为请求URL的第一级访问目录。当处理器类和处理器都使用@RequestMapping注解指定了对应的映射路径，处理器在浏览器中的访问地址，由项目访问路径+处理器类的映射路径+处理器的映射路径共同组成。

11.3.1 @RequestMapping注解的使用

STEP 01

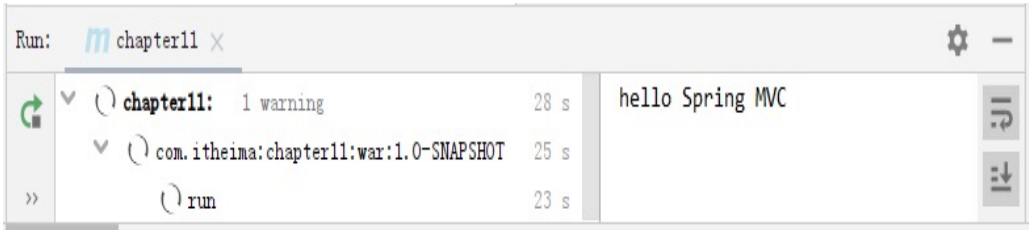
接下来通过一个案例演示@RequestMapping注解标注在类上的使用。

```
@Controller
@RequestMapping(value="/springMVC")
public class FirstController {
        @RequestMapping(value="/firstController")
        public void sayHello(){
        System.out.println("hello Spring MVC");
         }
}
```

11.3.1 @RequestMapping注解的使用

STEP 02

启动项目，在浏览器中访问http://localhost:8080/chapter11/springMVC/firstController，控制台打印输出信息。



由运行结果可知， sayHello()方法被成功执行，说明@RequestMapping注解标注在类上时，成功建立了请求URL和处理请求类之间的对应关系。

11.3.2 @RequestMapping注解的属性

- 掌握@RequestMapping注解的属性，能够在程序中正确使用@RequestMapping注解中的属性

11.3.2 @RequestMapping注解的属性

@RequestMapping注解的属性

属性名	类型	描述
name	String	可选属性，用于为映射地址指定别名。
value	String[]	可选属性，也是默认属性，用于指定请求的URL。
method	RequestMethod[]	可选属性，用于指定该方法可以处理哪种类型的请求方式。
params	String[]	可选属性，用于指定客户端请求中参数的值，必须包含哪些参数的值，才可以通过其标注的方法处理。
headers	String[]	可选属性，用于指定客户端请求中，必须包含哪些header的值，才可以通过其标注的方法处理。
consumes	String[]	可选属性，用于指定处理请求的提交内容类型（Content-type）。
produces	String[]	可选属性，用于指定返回的内容类型，仅当request请求头中的（Accept）类型中包含该指定类型才返回。

11.3.2 @RequestMapping注解的属性

value属性的两种映射路径标注

value属性是@RequestMapping注解的默认属性。当value属性是@RequestMapping注解显式使用的唯一属性时，可以省略value的属性名。例如，下面两种映射路径标注的含义相同。

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

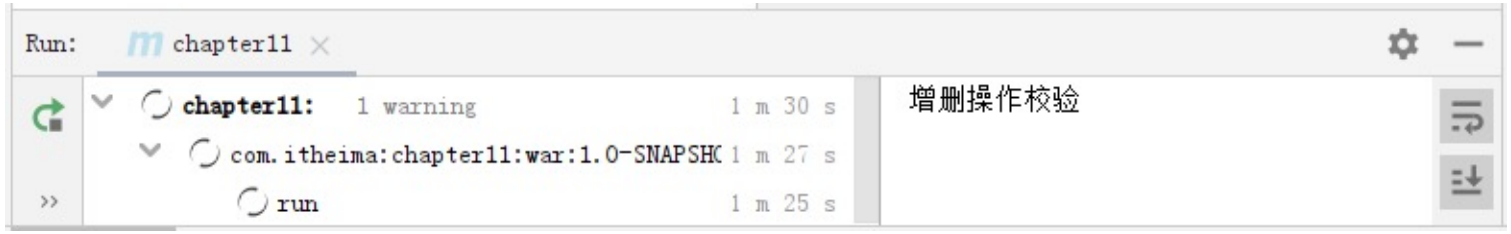
11.3.2 @RequestMapping注解的属性

使用value属性时，可以指定映射单个的请求URL，也可以将多个请求映射到一个方法上。在value属性中添加一个带有请求路径的列表，就可以将这个请求列表中的路径都映射到对应的方法上。

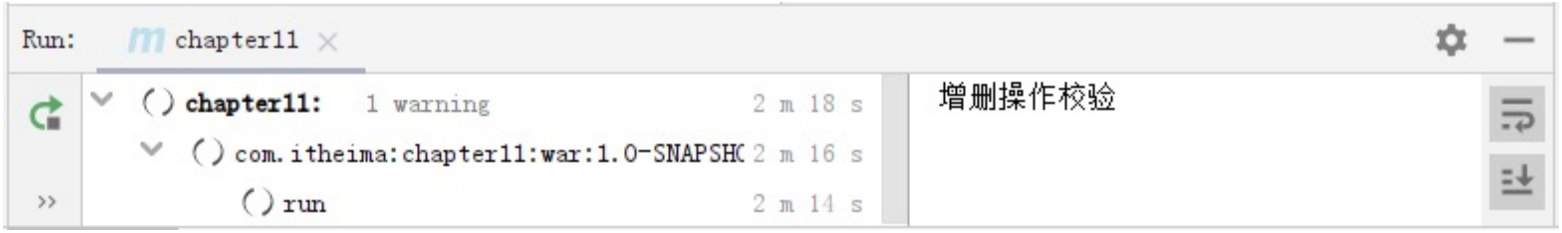
```
@Controller
public class AuthController {
   //设定当前方法的访问映射地址列表
   @RequestMapping(value = {"/addUser","/deleteUser"})
   public void checkAuth(){
      System.out.println("增删操作校验");
   }
}
```

11.3.2 @RequestMapping注解的属性

启动项目，在浏览器中访问地址http://localhost:8080/chapter11/addUser，控制台打印输出信息。



在浏览器中访问地址http://localhost:8080/chapter11/deleteUser，控制台打印输出信息。



11.3.2 @RequestMapping注解的属性

method属性限定处理器映射

method属性可以对处理器映射的URL请求方式进行限定。当请求的URL和处理器映射成功，但请求方式和method属性指定的属性值不匹配，处理器也不能正常处理请求。

11.3.2 @RequestMapping注解的属性

STEP 01

接下来通过一个案例演示method属性中HTTP请求类型的声明。

```
@Controller
@RequestMapping("/method")
public class MethodController {
    //只展示了 处理请求方式为GET的请求，get()方法
    @RequestMapping(method = RequestMethod.GET)
    public void get() {
        System.out.println("RequestMethod.GET");            }
    //处理请求方式为DELETE的请求，delete()方法
    //处理请求方式为POST的请求，post()方法
    //处理请求方式为PUT的请求，put()方法
    ...
}
```

11.3.2 @RequestMapping注解的属性

STEP 02

启动项目后，在客户端依次以GET方式、DELETE方式、POST方式和PUT方式请求访问http://localhost:8080/chapter11/method时，程序会分别执行文件中的get()方法、delete()方法、post()方法和put()方法，控制台打印输出信息。

11.3.2 @RequestMapping注解的属性

method属性中有多个HTTP请求类型

如果需要同时支持多个请求方式，则需要将请求方式列表存放在英文大括号中，以数组的形式给method属性赋值，并且多个请求方式之间用英文逗号分隔，示例代码如下所示。

```
@RequestMapping(value = "/method",
method = {RequestMethod.GET,RequestMethod.POST})
public void getAndPost() {
        System.out.println("RequestMethod.GET+RequestMethod.POST");
}
```

11.3.2 @RequestMapping注解的属性

params属性值的定义方式

params属性中定义的值可以将请求映射的定位范围缩小。当客户端进行请求时，如果请求参数的值等于params属性定义的值，可以正常执行所映射到的方法，否则映射到的方法不执行。

```
@Controller
public class ParamsController {
    @RequestMapping(value = "/params",params = "id=1")
    public void findById(String id) {
        System.out.println("id="+id);      }}
```

11.3.3 请求映射方式

- 掌握请求映射方式，能够在编程时熟练使用三种请求映射方式，包括基于请求方式的URL路径映射、基于Ant风格的URL路径映射和基于REST风格的URL路径映射

11.3.3 请求映射方式

请求映射方式的分类

基于注解风格的Spring MVC，通过@RequestMapping注解指定请求映射的URL路径。URL路径映射常用的方式有基于请求方式的URL路径映射、基于Ant风格的URL路径映射和基于REST风格的URL路径映射。接下来分别对这三种请求映射方式进行详细讲解。

11.3.3 请求映射方式

a. 基于请求方式的URL路径映射

上一节中学习到可以使用@RequestMapping注解的method属性，来限定当前方法匹配哪种类型的请求方式。除了可以使用@RequestMapping注解来限定客户端的请求方式之外，从Spring 4.3版本开始，还可以使用组合注解完成客户端请求方式的限定。组合注解简化了常用的HTTP请求方式的映射，并且更好的表达了被注解方法的语义。

11.3.3 请求映射方式

Spring MVC组合注解

@GetMapping：匹配GET方式的请求。

@PostMapping：匹配POST方式的请求。

@PutMapping：匹配PUT方式的请求。

@DeleteMapping：匹配DELETE方式的请求。

@PatchMapping：匹配PATCH方式的请求。

11.3.3 请求映射方式

@GetMapping用法示例

接下来以@GetMapping为例讲解组合注解的用法，@GetMapping是@RequestMapping(method = RequestMethod.GET)的缩写，使用组合注解替代@RequestMapping注解，可以省略method属性，从而简化代码。@GetMapping用法示例代码如下所示。

```
@GetMapping(value="/firstController")
public void sayHello(){
    …
}
```

11.3.3 请求映射方式

b. 基于Ant风格的URL路径映射

Spring MVC支持Ant风格的URL路径映射， 所谓Ant风格其实就是一种通配符风格，可以在处理器映射路径中使用通配符对访问的URL路径进行关联。Ant风格的通配符有以下3种，分别是：?匹配任何单字符；*匹配0或者任意数量的字符；**匹配0或者多级目录。

11.3.3 请求映射方式

Ant风格通配符的路径匹配

通配符	URL路径	通配符匹配说明
?	/ant1?	匹配项目根路径下/ant1[anyone]路径，其中[anyone]可以是任意单字符，即/ant1后有且只有1个字符。如/ant12、/ant1a。
*	/ant2/*.do	匹配项目根路径下/ant2/[any].do路径，其中[any]可以是任意数量的字符。如/ant2/findAll.do、/ant2/.do。
*	/*/ant3	匹配项目根路径下/[onemore]/ant3路径，其中[onemore]可以是数量多于0个的任意字符。如/a/ant3、/findAll/ant3，但是字符数量不能为0个，并且目录层数必须一致，如//ant3、/findAll/a/ant3。
**	/**/ant4	匹配项目根路径下/[anypath]/ant4路径，其中[anypath]可以是0或者多层的目录。如/ant4、/a/ant4、/a/b/ant4。
**	/ant5/**	匹配项目根路径下/ant5/[anypath]路径，其中[anypath]可以是0或者多层的目录。如/ant5、/ant5/a、/ant5/a/b。

11.3.3 请求映射方式

映射路径使用多个通配符情况

当映射路径中同时使用多个通配符时，会有通配符冲突的情况。当多个通配符冲突时，路径会遵守最长匹配原则（has more characters）去匹配通配符，如果一个请求路径同时满足两个或多个Ant风格的映射路径匹配规则，那么请求路径最终会匹配满足规则字符最多的路径。例如，/ant/a/path同时满足 /**/path和/ant/*/path匹配规则，但/ant/path最终会匹配“/ant/*/path”路径。

11.3.3 请求映射方式

c.基于RESTful风格的URL路径映射

RESTful是按照REST风格访问网络资源，简单说RESTful就是把请求参数变成请求路径的一种风格。 而REST（Representational State Transfer）是一种网络资源的访问风格，规范对了网络资源的访问方式。REST所访问的网络资源可以是一段文本、一首歌曲、一种服务，总之是一个具体的存在。每个网络资源都有一个URI指向它， 要获取这个资源，访问它的 URI 就可以，因此URI 即为每一个资源的独一无二的标识符。

11.3.3 请求映射方式

传统风格与RESTful风格访问URL格式的不同

传统风格访问的URL格式如下所示。

```
http://.../findUserById?id=1
```

而采用RESTful风格后，其访问的URL格式如下所示。

```
http://.../user/id/1
```

需要注意的是，RESTful风格中的URL不使用动词形式的路径，例如，findUserById表示查询用户，是一个动词，而user表示用户，为名词。

11.3.3 请求映射方式

RESTful风格的基本请求操作

RESTful风格在HTTP请求中，通过GET 、 POST 、 PUT和DELETE 4个动词对应四种基本请求操作，具体如下所示。

- GET用于获取资源
- POST用于新建资源
- PUT用于更新资源
- DELETE用于删除资源

11.3.3 请求映射方式

RESTful风格四种请求的约定方式

URL路径	请求方式	说明
http://localhost:8080/chapter11/user/1	HTTP GET	获得参数1进行查询user操作
http://localhost:8080/chapter11/user/1	HTTP DELETE	获得参数1进行删除user操作
ttp://localhost:8080/chapter11/user/1	HTTP PUT	获得参数1进行更新user操作
http://localhost:8080/chapter11/user	HTTP POST	新增user操作

11.3.3 请求映射方式

使用RESTful风格的优势

约定不是规范，约定是可以打破，所以称为RESTful风格，而不是RESTful规范。使用RESTful风格的优势在于路径的书写比较简便，并且通过地址无法得知做的是何种操作，可以隐藏资源的访问行为。

本章小结

本章主要对Spring MVC的核心类及相关注解的使用进行了讲解。首先介绍了DispatcherServlet核心类的作用和配置；然后介绍了@Controller注解的使用；最后讲解了@RequestMapping注解的相关知识。通过本章的学习，读者能够了解Spring MVC核心类DispatcherServlet的作用，并掌握@Controller注解和@RequestMapping注解的使用。