第3章 Spring Boot的Web应用支持

《Spring Boot企业级开发教程（第2版）》

黑马程序员
www.itheima.com

传智教育旗下
高端IT教育品牌

学习目标/Target

了解使用Spring Bean注册Java Web三大组件，能够简述使用Spring Bean注册Java Web三大组件的步骤

了解使用RegistrationBean注册Java Web三大组件，能够简述使用RegistrationBean注册Java Web三大组件的步骤

了解使用注解扫描注册Java Web三大组件，能够简述使用注解扫描注册Java Web三大组件的步骤

了解Spring MVC自动配置的特性，能够说出Spring MVC自动配置的特性

学习目标/Target

掌握自定义Spring MVC配置，能够自定义配置Spring MVC中的静态资源映射、视图控制器、拦截器

掌握文件上传，能够在Spring Boot项目中实现文件上传

熟悉Spring Boot异常处理自动配置原理，能够说出Spring Boot异常处理自动配置原理

掌握Spring Boot自定义异常处理，能够在Spring Boot项目中自定义异常处理

章节概述/ Summary

通常在Web开发中，会涉及到静态资源的访问支持、视图解析器的配置、转换器和格式化器的定制、文件上传等功能，甚至还需要考虑与Web服务器关联的Java Web三大组件的定制。Spring Boot框架支持整合一些常用Web框架来实现Web开发，并默认支持Web开发中的一些通用功能。下面将对Spring Boot的Web应用支持进行讲解。

目录/Contents

3.1 注册Java Web三大组件

传统Web应用项目

最常用的三大组件有Servlet、Filter和Listener。使用这些组件时需要在项目的web.xml文件中进行配置，或者使用相应的注解进行标注。

Spring Boot项目

默认没有web.xml文件，同时默认情况下Spring Boot项目不能自动识别到这三个组件的相关注解。

可以使用Spring Bean、RegistrationBean、注解扫描的方式注册Java Web三大组件。

3.1.1 使用Spring Bean注册Java Web三大组件

了解使用Spring Bean注册Java Web三大组件，能够简述使用Spring Bean注册Java Web三大组件的步骤

3.1.1 使用Spring Bean注册Java Web三大组件

在Spring Boot项目中，会自动将Spring容器中的Servlet、Filter、Listener实例注册为Web服务器中对应的组件。因此，可以将自定义的Java Web三大组件作为Bean添加到Spring容器中，以实现组件的注册。

使用Spring Bean注册Servlet时，需要自定义两个及以上的Servlet，Servlet对应的映射地址为“Bean名称+/”。Filter的映射地址默认为“/*”。

3.1.1 使用Spring Bean注册Java Web三大组件

下面通过案例演示使用Spring Bean注册Java Web三大组件。

（1）创建自定义原生组件。创建Spring Boot项目chapter03，在项目的java文件夹下创建类包com.itheima.chapter03.web，并在类包下创建自定义的Servlet、Filter、Listener，具体如文件3-1~文件3-4所示。

源 代 码

文件3-1 FirstServlet.java

文件3-2 SecondServlet.java

文件3-3 MyFilter.java

文件3-4 MyListener.java

（2）创建组件配置类。在项目的java文件夹下创建类包com.itheima.chapter03.config，并在类包下创建组件配置类WebConfigure，在该类中创建4个方法，分别返回文件3-1~文件3-4中类的实例交由Spring管理，具体如文件3-5所示。

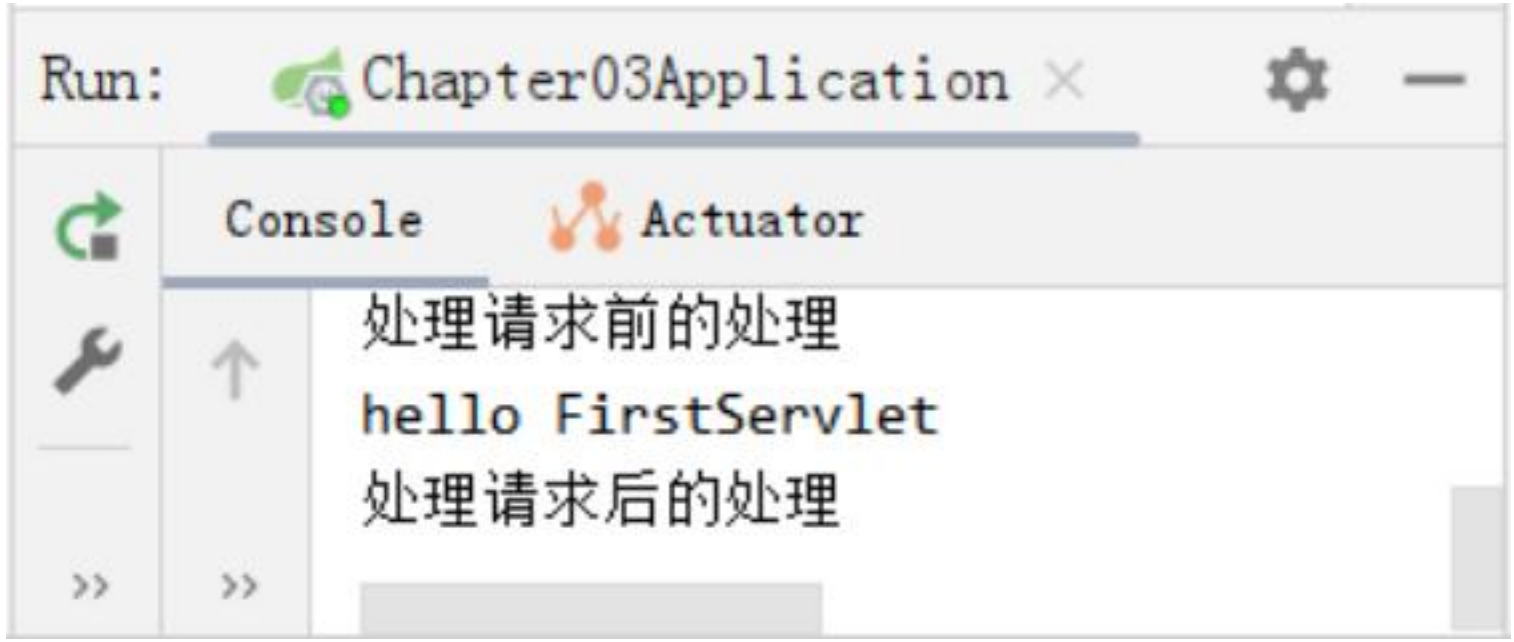
源 代 码

文件3-5

WebConfigure.java

（3）测试程序效果。运行项目的启动类，控制台输出程序启动信息。

在浏览器中访问http://localhost:8080/firstServlet/，控制台输出访问的结果信息。



3.1.2 使用RegistrationBean注册Java Web三大组件

了解使用RegistrationBean注册Java Web三大组件，能够简述使用RegistrationBean注册Java Web三大组件的步骤

3.1.2 使用RegistrationBean注册Java Web三大组件

使用Spring Bean注册Java Web三大组件时，如果容器中只有一个自定义Servlet，则无法使用Bean的名称作为映射路径，而Filter默认只使用“*”的映射地址。为解决此问题Spring Boot提供了更为灵活的注册方法，可以在配置类中使用 RegistrationBean来注册原生Web组件。

RegistrationBean是个抽象类，SpringBoot提供了三个RegistrationBean的实现类：ServletRegistrationBean、FilterRegistrationBean、ServletListenerRegistrationBean，这三个类分别用来注册Servlet、Filter和Listener，通过这三个类开发者可以获得自定义映射路径等更多的控制权。

下面通过案例演示使用RegistrationBean注册Java Web三大组件。

（1）修改配置类。将文件3-5中WebConfigure类原有的方法进行注释，将自定义的 Servlet、Filter和Listener包装到对应的RegistrationBean中，并使用@Bean 注解将ServletRegistrationBean、FilterRegistrationBean和ServletListenerRegistrationBean注册到 Spring容器中。修改后代码如文件3-6所示。

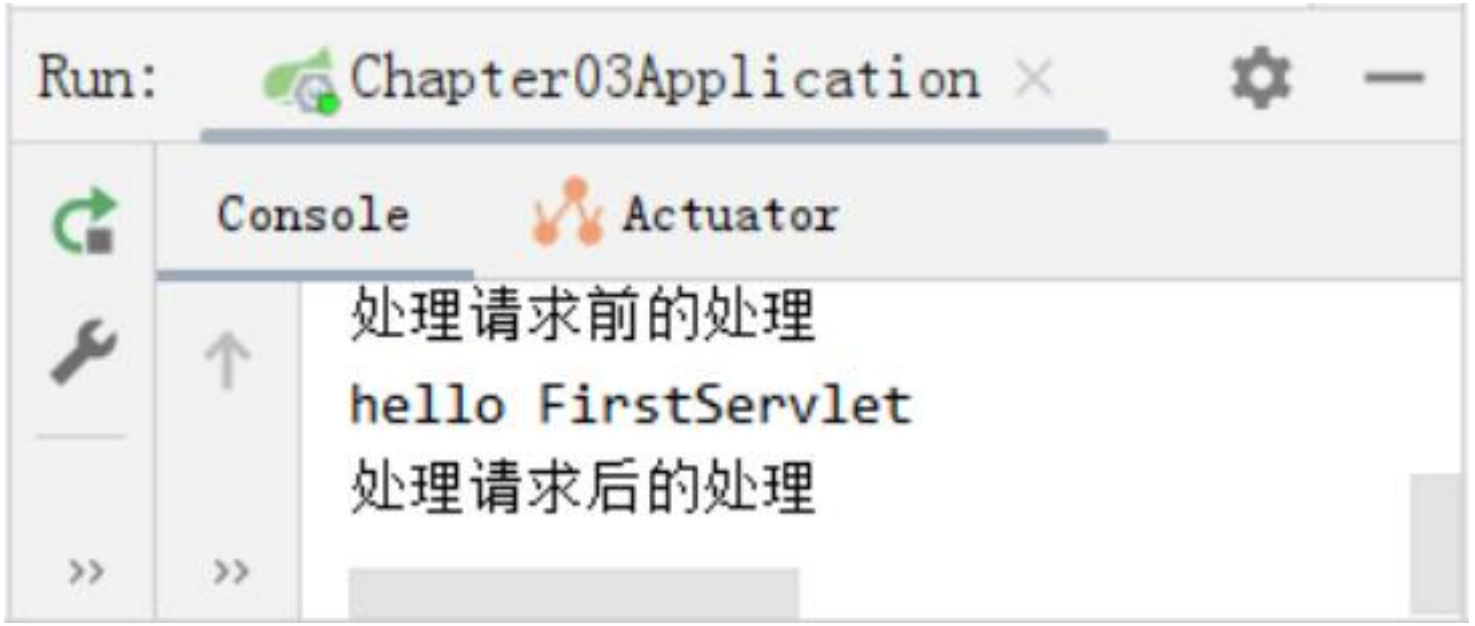
源 代 码

文件3-6

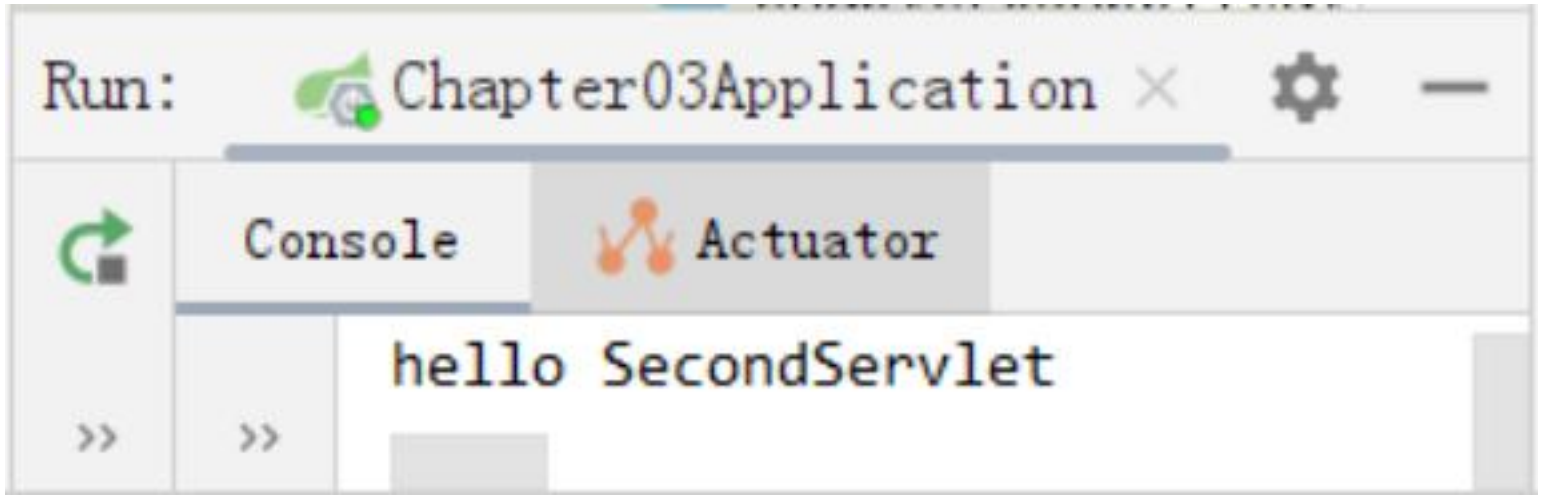
WebConfigure.java

（2）测试程序效果。运行项目的启动类，控制台输出项目启动信息。

在浏览器中访问http://localhost:8080/first，此时控制台输出访问结果。



在浏览器中访问http://localhost:8080/second，此时控制台输出访问结果。



3.1.3使用注解扫描注册Java Web三大组件

了解使用注解扫描注册Java Web三大组件，能够简述使用注解扫描注册Java Web三大组件的步骤

Spring Boot无法自动识别到@WebServlet、@WebFilter、@WebListener标注的类，但其内部可使用嵌入式容器，可以使用@ServletComponentScan扫描标注@WebServlet、@WebFilter和@WebListener的类，并将扫描到的类自动注册到Spring容器。

3.1.3使用注解扫描注册Java Web三大组件

下面通过案例演示使用注解扫描注册Java Web三大组件。

（1）使用注解声明组件。分别使用@WebServlet、@WebFilter、@WebListener标注FirstServlet类、MyFilter类和MyListener类，具体如文件3-7~文件3-9所示。

源 代 码

文件3-7 FirstServlet.java

文件3-8 MyFilter.java

文件3-9 MyListener.java

（2）添加@ServletComponentScan注解。在项目的启动类上添加@ServletComponentScan注解。具体代码如文件3-10所示。

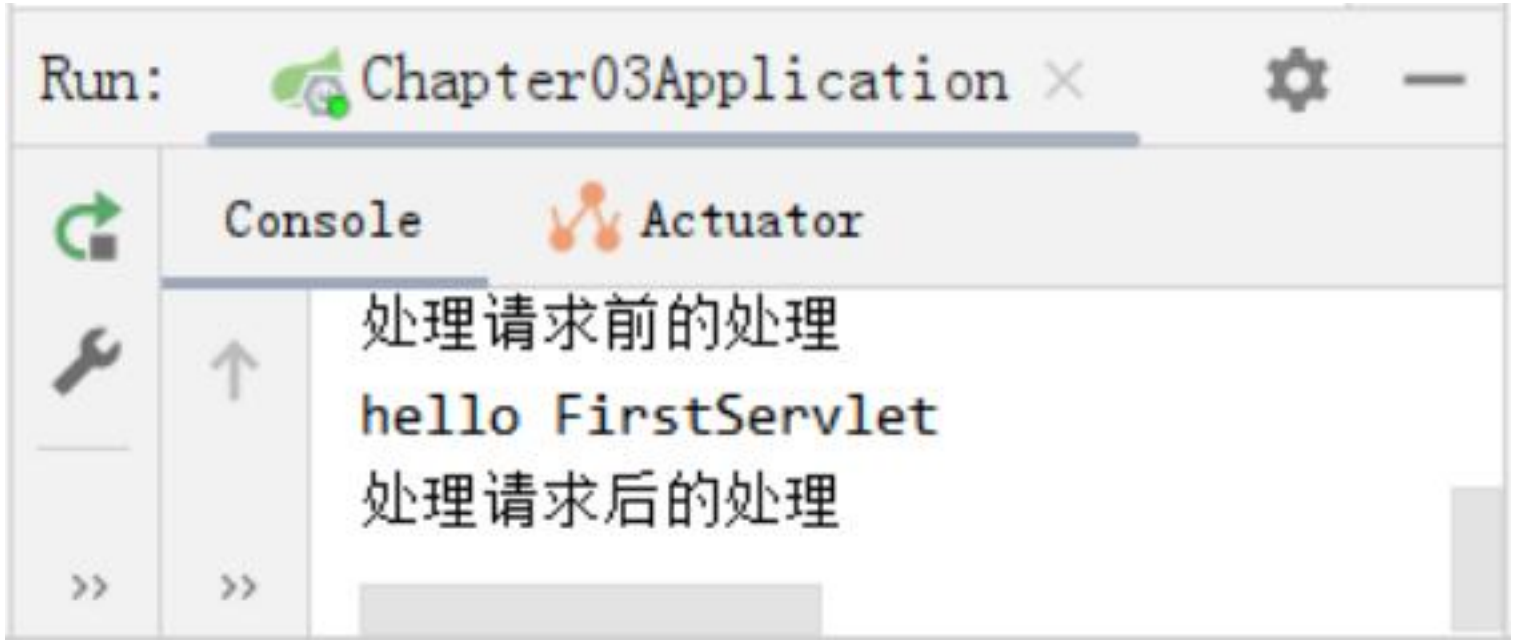
源 代 码

文件3-10

Chapter03Application.java

（3）测试程序效果。使用扫描组件的方式注释原生组件不需要其他配置类，因此，先注释掉文件3-6中的配置类WebConfigure，然后运行项目的启动类，控制台输出项目启动的信息。

在浏览器中访问http://localhost:8080/first，此时控制台输出访问结果信息。



3.2 Spring Boot管理Spring MVC

Spring Boot真正的核心功能是自动配置和快速整合，通常Spring Boot应用的前端MVC框架依然使用Spring MVC。Spring Boot提供的spring-boot-starter-web启动器嵌入了Spring MVC的依赖，并为Spring MVC提供了大量自动配置，可以适用于大多数Web开发场景。

除了使用自动配置所提供的功能，开发者也可以通过自定义配置类定制Spring MVC的配置。下面分别对Spring MVC自动配置的特性和自定义Spring MVC配置进行讲解。

3.2.1 Spring MVC自动配置的特性

了解Spring MVC自动配置的特性，能够说出Spring MVC自动配置的特性

3.2.1 Spring MVC自动配置的特性

Spring Boot为Spring MVC提供了自动配置，并在Spring MVC默认功能的基础上添加了以下特性。

(1)引入了视图解析器 ContentNegotiatingViewResolver和BeanNameViewResolver。

(2)为包括WebJars在内的静态资源提供支持。

(3)自动注册 Converter、GenericConverter和Formatter。

(4)支持使用HttpMessageConverters消息转换器。

(5)自动注册 MessageCodesResolver。

(6)支持静态项目首页index.html。

(7)支持定制应用图标favicon.ico。

(8)自动初始化Web数据绑定器ConfigurableWebBindingInitializer。

3.2.2 自定义Spring MVC配置

掌握自定义Spring MVC配置，能够自定义配置Spring MVC中的静态资源映射、视图控制器、拦截器

在Spring Boot应用中使用Spring MVC时，如果希望在为Spring MVC自动配置提供相关特性的同时，再增加一些自定义的Spring MVC配置，例如添加拦截器、视图控制器等，可以通过自定义WebMvcConfigurer类型的配置类来实现。

下面分别对自定义Spring MVC配置中的配置静态资源映射、配置视图控制器、配置拦截器进行讲解。

1.配置静态资源映射

通常Web应用中会需要使用静态资源，例如，JavaScript文件、CSS文件和HTML文件等。单独使用Spring MVC时，导入静态资源文件后，需要配置静态资源的映射，否则无法正常访问。Spring Boot中提供了默认的静态资源映射，当访问项目中任意的静态资源时，Spring Boot会默认从以下路径中进行查找。

(1)classpath:/META-INF/resources/

(2)classpath:/resources/

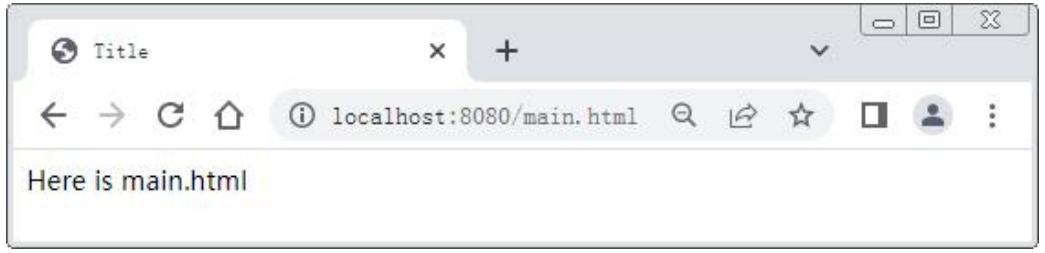
(3)classpath:/static/

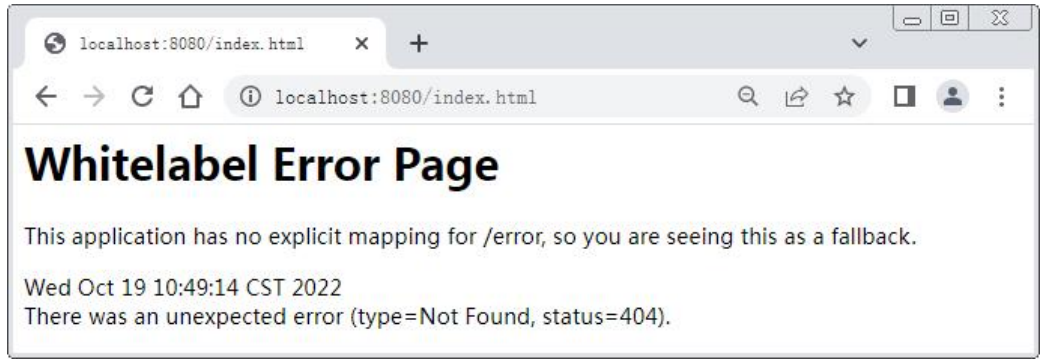
(4)classpath:/public/

优先级从（1）至（4）依次递减，Spring Boot会先查找优先级高的文件夹，再查找优先级低的文件夹。

1.配置静态资源映射

在chapter03项目中的 src/main/resources/static和 src/main/resources目录下分别创建main.html文件和index.html文件，并在项目启动后分别在浏览器中对这两个静态资源进行访问。





1.配置静态资源映射

如果想在项目中访问非默认静态资源文件夹下的资源，可以自定义静态资源的映射。自定义静态资源的映射可以通过配置类和配置文件2种方式实现。

（1）通过配置类实现静态资源映射

通过配置类实现静态资源映射时，配置类需要实现WebMvcConfigurer接口，在重写WebMvcConfigurer接口的addResourceHandlers()方法中指定资源访问路径和资源之间的映射关系。

1.配置静态资源映射

（2）通过配置文件实现静态资源映射

Spring Boot在Spring MVC的自动配置中提供了对应的属性可以配置静态资源访问路径和资源的映射。

```
spring:
  mvc:
    static-path-pattern: /backend/**
  web:
    resources:
      static-locations:
          file:E:\idea\SpringBoot\chapter03\src\main\resources\backend
```

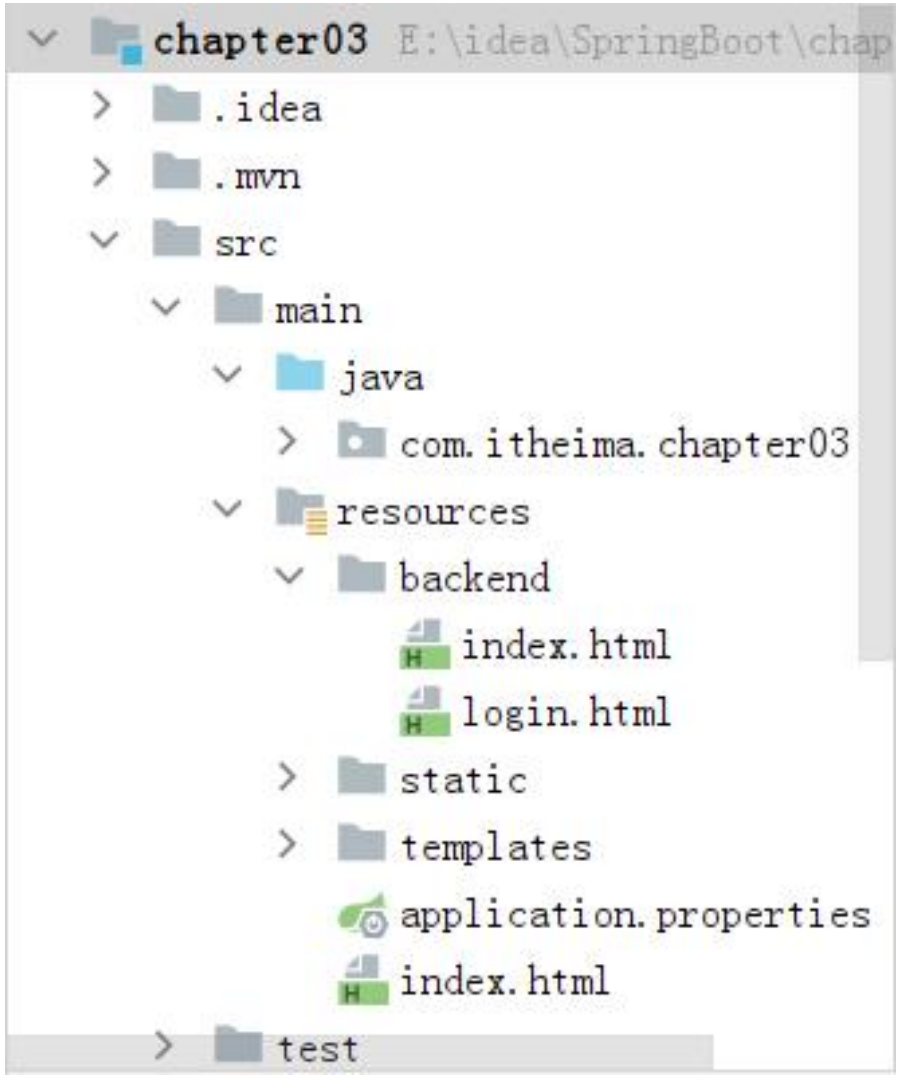
用于指定静态资源的访问路径。

用于指定静态资源存放的目录。

1.配置静态资源映射

下面以通过配置类实现静态资源映射为例，演示配置静态资源映射。

（1）创建静态资源。在chapter03项目中的 src/main/resources目录下，创建文件夹backend，并在文件夹中创建HTML文件index.html和login.htm。



1.配置静态资源映射

（2）配置静态资源映射。在项目chapter03的com.itheima.chapter03.config包下创建配置类WebMvcConfig，该配置类实现WebMvcConfigurer接口，并重写该接口的方法实现自定义Spring MVC的配置，具体如文件3-11所示。

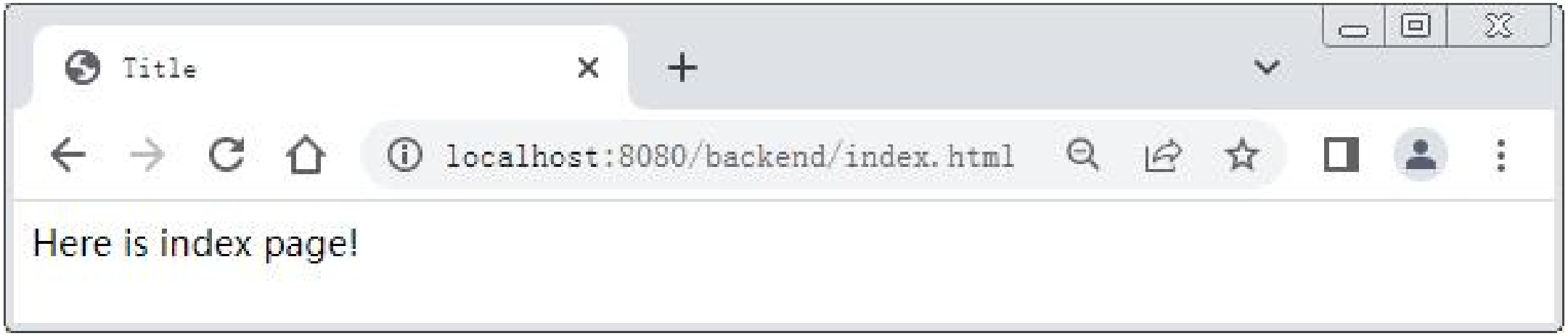
源 代 码

文件3-11

WebMvcConfig.java

1.配置静态资源映射

（3）测试程序效果。启动项目，在浏览器中访问backend文件夹下的index.html。



2.配置视图控制器

使用Spring MVC默认的配置进行开发时，如果仅操作需要实现无业务逻辑的页面跳转，也需要创建Controller类，然后定义方法跳转到页面，操作比较麻烦。对此，可以在视图控制器中添加自定义的映射，直接将请求映射为视图。

2.配置视图控制器

下面通过案例演示在视图控制器中配置请求和视图的映射，具体如下。

（1）配置视图控制器映射信息。在文件3-11中重写WebMvcConfigurer接口的addViewControllers()方法，在该方法中添加访问路径和视图的映射。

```
@Override
public void addViewControllers(ViewControllerRegistry registry) {
                registry.addViewController("/backend/toLoginPage")
.setViewName("/backend/login.html");
                registry.addViewController("/backend")
.setViewName("/backend/index.html");}
```

2.配置视图控制器

（2）测试程序效果。启动项目，在浏览器中访问http://localhost:8080/toLoginPage。

3.配置拦截器

拦截器可以根据请求的URL对请求进行拦截，主要应用于登录校验、权限验证、乱码解决、性能监控和异常处理等方面。在Spring Boot项目中配置拦截器也非常简单，只需要定义拦截器和注册拦截器即可。

3.配置拦截器

下面通过案例演示在Spring Boot项目中配置拦截器。

（1）定义拦截器。在项目chapter03下创建com.itheima.chapter03.interceptor包，在该包下创建拦截器类，该类实现了 HandlerInterceptor接口，并重写了接口的preHandle()方法，具体如文件3-12所示。

源 代 码

文件3-12

LoginInterceptor.java

3.配置拦截器

（2）注册拦截器。在文件3-11中重写重写WebMvcConfigurer接口的addInterceptors()方法，在该方法中添加拦截器。

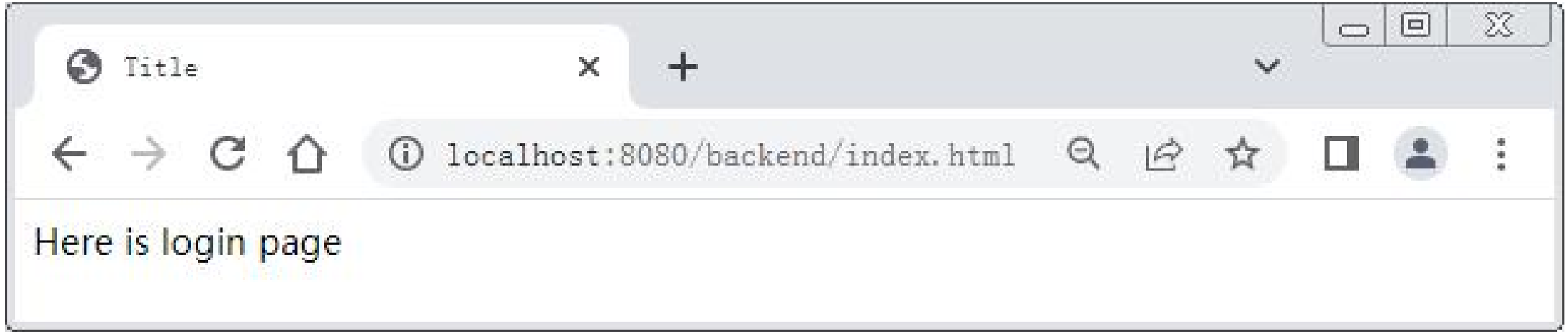
```
@Override
public void addInterceptors(InterceptorRegistry registry) {
//拦截所有请求，包括静态资源文件
        registry.addInterceptor(new oginInterceptor()).addPathPatterns("/**")
                .excludePathPatterns("/backend/login.html");
}
```

3.配置拦截器

（3）测试程序效果。启动项目，在浏览器中访问http://localhost:8080/backend/login.html。

3.配置拦截器

在浏览器中访问http://localhost:8080/backend/index.html。



3.配置拦截器

需要说明的是，Spring Boot在整合Spring MVC过程中提供了许多默认自动化配置和特性，开发者可以通过Spring Boot提供的WebMvcConfigurer接口对MVC功能进行定制和扩展。如果开发者不想使用Spring Boot整合MVC时提供的一些默认配置，而是想要绝对的自定义管理，那么可以编写一个@Configuration注解配置类，同时添加@EnableWebMvc注解关闭Spring Boot提供的所有关于MVC功能的默认配置。

3.3 文件上传

掌握文件上传，能够在Spring Boot项目中实现文件上传

在项目开发过程中，文件的上传和下载是比较常见的开发需求。Spring Boot为Spring MVC的文件上传同样提供了自动配置，Spring Boot推荐使用基于Servlet 3的文件上传机制，这样可直接利用Web服务器内部的文件上传支持，而无须引入第三方JAR包。

Spring Boot的文件上传自动配置主要由MultipartAutoConfiguration类和MultipartProperties类组成，其中MultipartProperties负责加载 以“spring.servlet.multipart”开头的配置属性，而MultipartAutoConfiguration则根据MultipartProperties读取的配置属性来初始化StandardServletMultipartResolver解析器对象。

下面通过案例演示在Spring Boot项目中进行文件上传，该案例演示的是图片文件的上传，图片上传后将图片显示在上传页面，具体如下。

（1）设置上传配置。在chapter03项目的application.yml文件对静态资源的映射和文件上传属性进行配置。

```
spring:
  mvc:
    static-path-pattern: /backend/**
  web:
    resources:
      static-locations:
file:E:\idea\SpringBoot\chapter03\src\main\resources\backend
  servlet:
    multipart:
      max-file-size: 10MB
      max-request-size: 50MB
```

单个上传文件的大小限制

（2）创建文件上传页面。在项目中的 src/main/resources目录下创建HTML页面用于操作文件上传，具体如文件3-13所示。

（3）创建文件上传控制器类。在项目chapter03的com.itheima.chapter03.controller包下创建控制器类FileController，在该类中处理文件上传的请求，将上传的文件存放在指定路径下，并返回上传结果。具体如文件3-14所示。

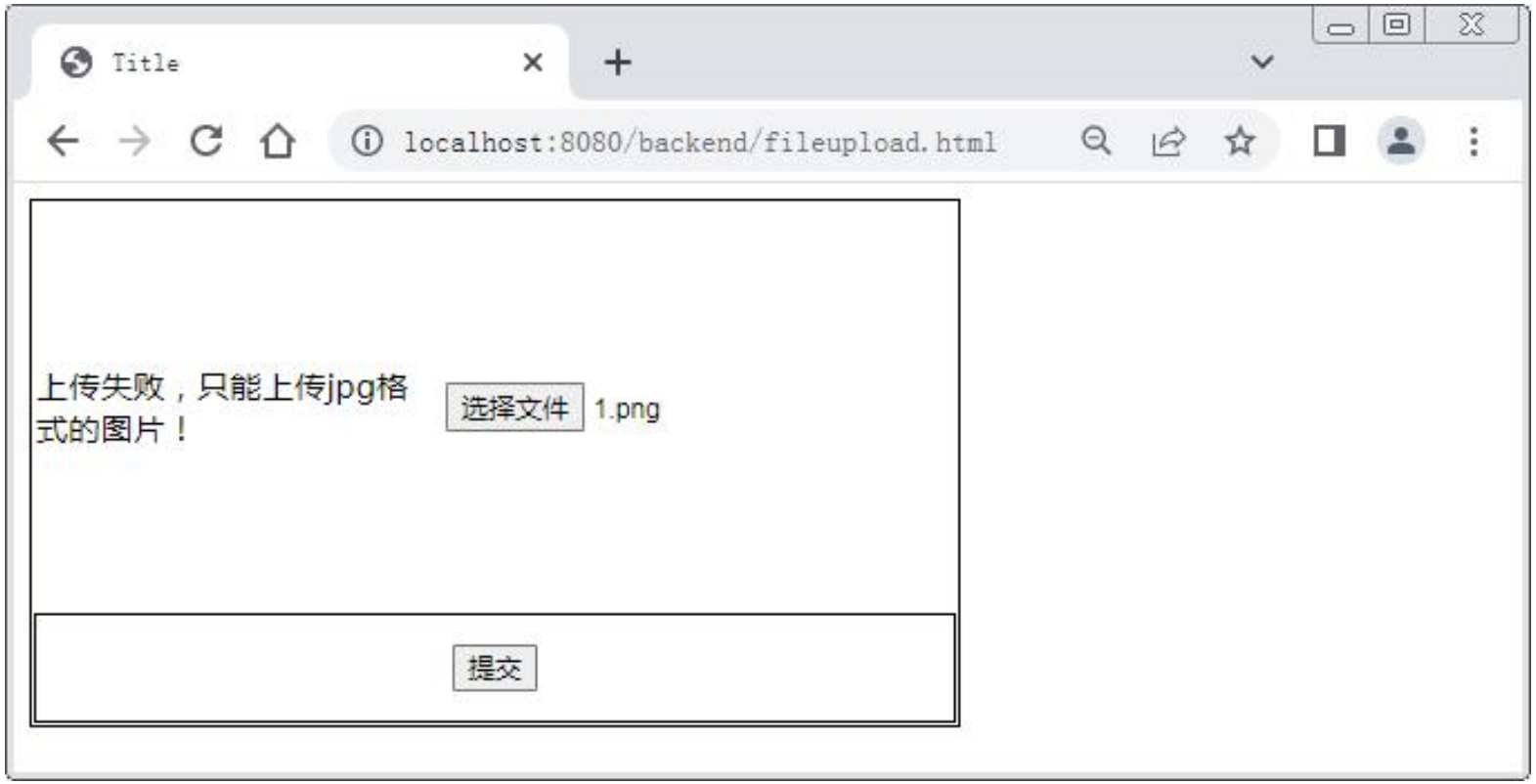
源 代 码

文件3-14

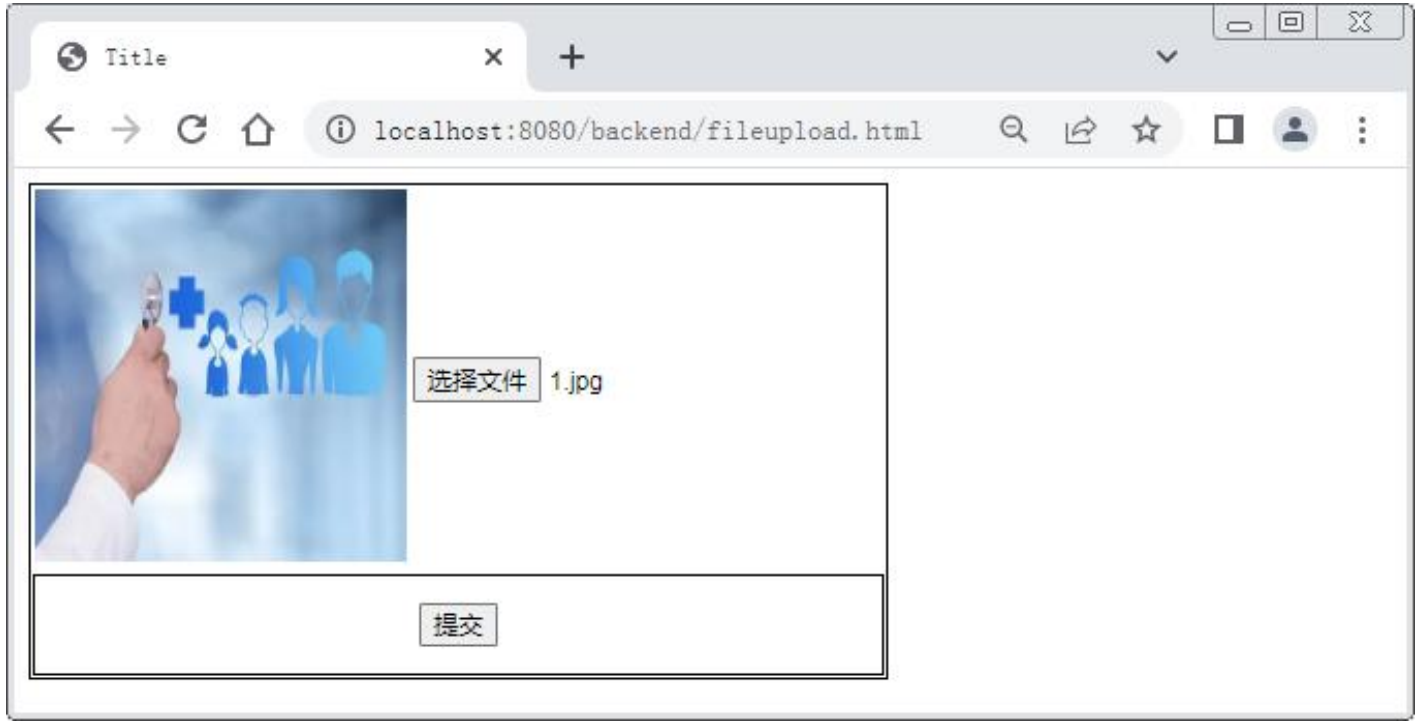
FileController.java

（4）程序效果测试。为了方便测试，先将chapter03项目中拦截器相关配置进行注释，启动项目，在浏览器访问fileupload.html。

在文件上传页面中单击“选择文件”按钮，选择一个后缀名不是jpg的文件，然后单击“提交”按钮。



再次单击“选择文件”按钮，选择后缀名为jpg的文件，然后单击“提交”按钮。



3.4 异常处理

在日常的 Web 开发中，项目中难免会出现各种异常，为了使客户端能接收较为友好的提示，通常开发者会对异常进行统一处理。为了便于开发者处理异常，Spring Boot通过自动装配提供了一套默认的异常处理机制，一旦程序中出现了异常，Spring Boot会根据该机制进行默认的异常处理。除了默认的异常处理，Spring Boot也支持自定义异常处理。下面对Spring Boot中的异常处理进行讲解。

3.4.1 Spring Boot异常处理自动配置原理

熟悉Spring Boot异常处理自动配置原理，能够说出Spring Boot异常处理自动配置原理

3.4.1 Spring Boot异常处理自动配置原理

Spring Boot通过配置类ErrorMvcAutoConfiguration为异常处理提供了自动配置，该配置类向容器中注入了以下4个组件。

- ErrorPageCustomizer：错误页面响应规则类，该组件会在在系统发生异常后，默认将请求转发到“/error”。
- BasicErrorController：错误控制器，处理“/error”请求。
- DefaultErrorViewResolver：默认的错误视图解析器，将异常信息解析到相应的错误视图。
- DefaultErrorAttributes：错误属性处理类，用于页面上共享异常信息。

1.ErrorPageCustomizer

Spring Boot项目中配置了Spring MVC的启动器后，项目启动时会加载异常处理的自动配置类ErrorMvcAutoConfiguration，ErrorMvcAutoConfiguration会向容器中注入了一个名为ErrorPageCustomizer的组件，该组件用于定制错误页面的响应规则。

ErrorPageCustomizer类中提供了registerErrorPages()方法用于注册错误页面的响应规则。当系统发生异常后，ErrorPageCustomizer组件会自动生效，并将请求转发到Spring Boot的异常处理地址，默认为“/error”。

2.BasicErrorController

ErrorMvcAutoConfiguration还向容器中注入了错误控制器组件BasicErrorController，BasicErrorController中会对异常处理路径进行统一映射处理。

BasicErrorController提供了errorHtml()和error()方法。其中，errorHtml()方法用于处理请求的MediaType为text/html的请求，并使用错误视图解析器生成包含错误页面地址和页面内容的ModelAndView对象；error()方法用于处理其他的请求，并返回JSON格式的数据展示错误信息。

3.DefaultErrorViewResolver

ErrorMvcAutoConfiguration加载的同时也会向容器中注入DefaultErrorViewResolver类。当BasicErrorController使用errorHtml()方法处理异常请求时，Spring Boot会获取容器中所有的ErrorViewResolver（错误视图解析器）对象，并分别调用resolveErrorView()方法对异常信息进行解析，其中默认错误信息解析器为 DefaultErrorViewResolver。

3.DefaultErrorViewResolver

DefaultErrorViewResolver 解析异常信息的步骤如下。

① 根据错误状态码，生成对应的错误视图。错误视图的名称格式为“error/status”，其中，status为错误状态码。

② 尝试使用模板引擎解析错误视图。从classpath类路径的templates/error目录下查找status.html文件，其中status为错误状态码。若模板引擎能够解析到对应的视图，则将视图和数据封装成ModelAndView返回，并结束整个解析流程。

③ 如果模板引擎不能正确解析到对应的视图，则依次从各个静态资源文件夹的error文件夹中查找status.html文件，若在静态文件夹中找到了该错误页面，则将视图和数据封装成ModelAndView返回，并结束整个解析流程。

④ 如果上述流程都没有查找到对应的视图，则使用Spring Boot 默认的错误页面作为响应的视图。

4.DefaultErrorAttributes

DefaultErrorAttributes是Spring Boot的默认错误属性处理类，它可以从请求中获取异常或错误信息，并将其封装为一个Map对象返回。

在BasicErrorController处理异常时，会调用DefaultErrorAttributes的getErrorAttributes()方法获取错误或异常信息，并封装到Model中，返回到页面或 JSON 数据中。该Model中主要包含以下属性。

- timestamp：时间戳。
- status：错误的状态码。
- error：错误的原因描述。
- exception：导致请求处理失败的异常类名。
- message：错误/异常消息。
- trace：错误/异常栈信息。
- path：错误/异常抛出时所请求的URL路径。

3.4.2 Spring Boot自定义异常处理

掌握Spring Boot自定义异常处理，能够在Spring Boot项目中自定义异常处理

3.4.2 Spring Boot自定义异常处理

Spring Boot默认的异常处理其实就是对Spring MVC异常处理的自动配置。如果想要自定义异常处理，可以在Spring Boot提供的自动配置的基础上，通过修改配置信息以修改Spring Boot默认的错误处理行为，或者使用@ResponseStatus、@ExceptionHandler、@ControllerAdvice等注解基于Spring MVC异常处理机制进行异常处理。

下面通过案例演示使用@ExceptionHandler和@ControllerAdvice完成Spring Boot全局异常处理。

（1）创建自定义异常类

在项目的com.itheima.chapter03.exception包下创建自定义异常类，具体如文件3-15所示。

源 代 码

文件3-15

MyException.java

（2）创建异常处理类

在com.itheima.chapter03.handler包下创建异常处理器·，在该类中定义了2个方法，分别处理控制器执行时抛出的Exception异常和自定义异常， 具体如文件13-16所示。

源 代 码

文件3-16

MyExceptionHandler.java

（3）创建控制器类

在com.itheima.chapter03.controller包下创建控制器类DishController，在该类中定义delDish()方法处理请求， 具体如文件13-17所示。

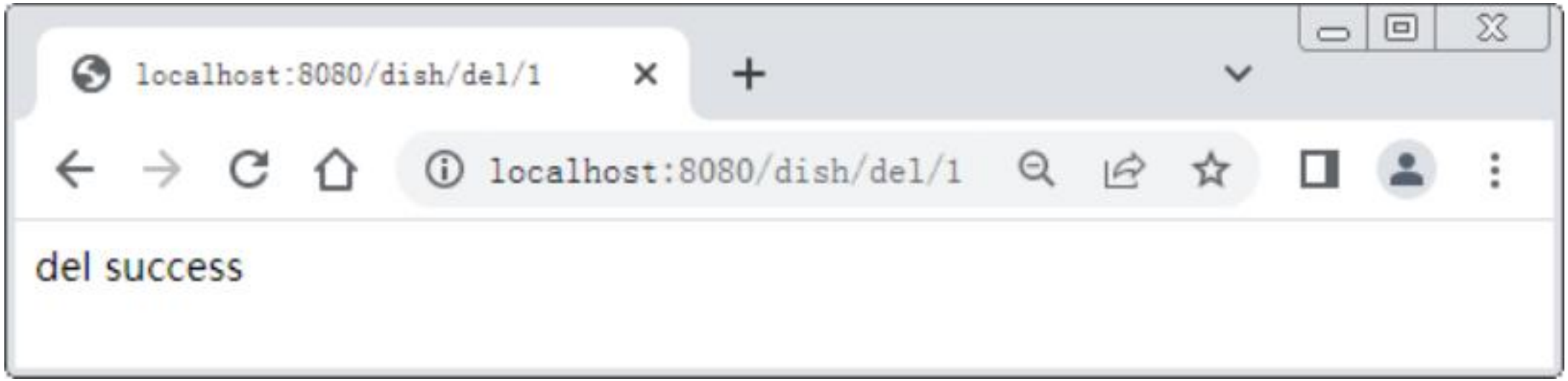
源 代 码

文件3-17

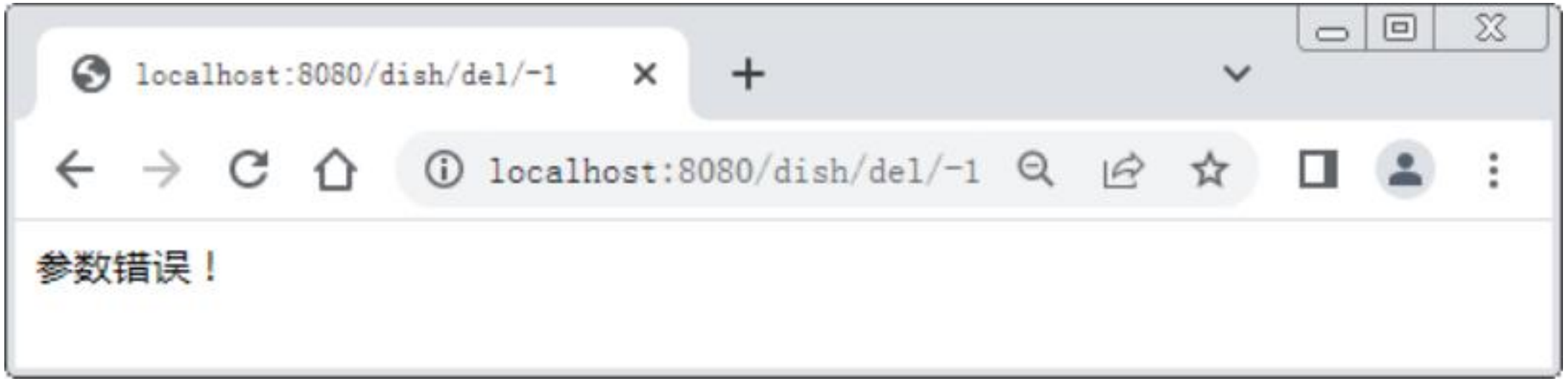
DishController.java

（4）测试全局异常处理

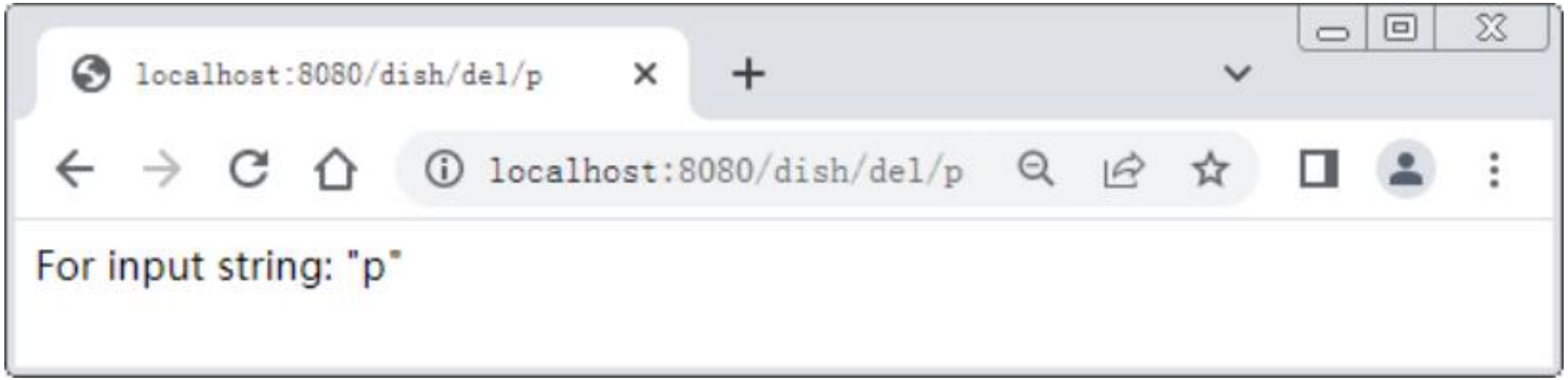
启动程序，在浏览器中访问http://localhost:8080/dish/del/1。



将请求删除的参数修改为-1，在浏览器中访问http://localhost:8080/dish/del/-1。



请求删除的参数修改为字母，在浏览器中访问http://localhost:8080/dish/del/p。



本章小结

本章主要对Spring Boot的Web应用支持进行了讲解。首先讲解了Spring Boot注册Java Web三大组件；然后讲解了Spring Boot管理Spring MVC；接着讲解了文件上传；最后讲解了异常处理。通过本章的学习，希望大家可以对Spring Boot的Web应用支持有所了解，为后续更深入学习Spring Boot做好铺垫。

為千萬學生少走彎路而著書

黑马程序员
www.itheima.com

传智教育旗下
高端IT教育品牌