**Spring MVC 文件上传流程**

Spring MVC 框架提供的文件上传工具类进行文件上传，也就是 **MultipartResolver** ，利用 SpringMVC 实现文件上传功能，离不开对 MultipartResolver 的设置。

MultipartResolver 这个类，你可以将其视为 SpringMVC 实现文件上传功能时的工具类，这个类也只会在文件上传中发挥作用，在配置了具体实现类之后，SpringMVC 中的 DispatcherServlet 在处理请求时会调用 MultipartResolver 中的方法判断此请求是不是文件上传请求，如果是的话 DispatcherServlet 将调用 MultipartResolver 的 resolveMultipart(request) 方法对该请求对象进行装饰，并返回一个新的 MultipartHttpServletRequest 供后继处理流程使用。 注意！此时的请求对象会由 HttpServletRequest 类型转换成 MultipartHttpServletRequest 类型，这个类中会包含所上传的文件对象可供后续流程直接使用，而无需自行在代码中实现对文件内容的读取和对象封装的逻辑。

在 Spring Boot 中也是通过该工具类进行文件上传，与普通的 Spring web 项目不同的是，Spring Boot 在自动配置 DispatcherServlet 时也配置好了 MultipartResolver ，而无需再像原来那样在 Spring MVC 配置文件中增加文件上传配置的 bean。

**Spring Boot 文件上传功能实现**

完整项目结构如下：

lou-springboot

├── pom.xml

├── README.md

└── src

├── main

│ ├── java

│ │ └── com

│ │ └── lou

│ │ └── springboot

│ │ ├── Application.java

│ │ ├── config

│ │ │ └── SpringBootWebMvcConfigurer.java

│ │ └── controller

│ │ └── UploadController.java

│ └── resources

│ ├── application.properties

│ ├── static

│ │ └── upload-test.html

│ └── templates

└── test

└── java

└── com

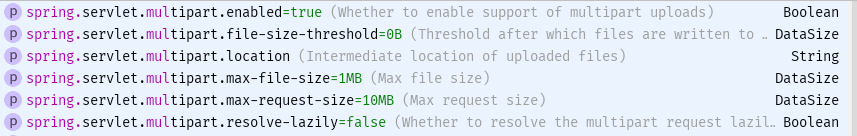
└── lou

└── springboot

└── ApplicationTests.java

**常用配置**

由于 Spring Boot 自动配置机制的存在，我们并不需要进行多余的设置，只要已经在 pom 文件中引入了 web starter 模块即可直接进行文件上传功能，虽然不用配置也可以使用文件上传，但是有些开发者可能会在文件上传时有一些特殊的需求，因此也需要对 Spring Boot 中 MultipartFile 的常用设置进行介绍，配置和默认值如下：



配置含义注释：

* **spring.servlet.multipart.enabled**

是否支持 multipart 上传文件，默认支持

* **spring.servlet.multipart.file-size-threshold**

文件大小阈值，当大于这个阈值时将写入到磁盘，否则存在内存中，（默认值 0 ，一般情况下不用特意修改）

* **spring.servlet.multipart.location**

上传文件的临时目录

* **spring.servlet.multipart.max-file-size**

最大支持文件大小，默认 1 M ，该值可适当的调整

* **spring.servlet.multipart.max-request-size**

最大支持请求大小，默认 10 M

* **spring.servlet.multipart.resolve-lazily**

判断是否要延迟解析文件（相当于懒加载，一般情况下不用特意修改），默认 false

（懒加载：就是对象用到的时候再去加载。一个对象被创建出来需要一定的内存空间，如果接下来这个对象没有被用到就说明这个对象创建过早了。懒加载就是重写对象的getter方法，当系统或者开发者调用对象的getter的时候再去加载对象。）

**上传功能实现**

**新建文件上传页面**

在 static 目录中新建 upload-test.html，上传页面代码如下：

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8" />

<title>Spring Boot 文件上传测试</title>

</head>

<body>

<form action="/upload" method="post" enctype="multipart/form-data">

<input type="file" name="file" />

<input type="submit" value="文件上传" />

</form>

</body>

</html>

这应该是大家都很熟悉的一个文件上传页面 demo ，文件上传的请求地址为 /upload，请求方法为 post，需要注意的是在文件上传时要设置 enctype="multipart/form-data"，页面中包含一个文件选择框和一个提交框，如下所示：

**新建文件上传处理 Controller**

在 controller 包下新建 UploadController 并编写实际的文件上传逻辑代码，代码如下：

@Controller

public class UploadController {

// 文件保存路径为 /home/project/upload/ 即当前 project 目录下的 upload 文件夹

private final static String FILE\_UPLOAD\_PATH = "/home/project/upload/";

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

@ResponseBody

public String upload(@RequestParam("file") MultipartFile file) {

if (file.isEmpty()) {

return "上传失败";

}

String fileName = file.getOriginalFilename();

String suffixName = fileName.substring(fileName.lastIndexOf("."));

//生成文件名称通用方法

SimpleDateFormat sdf = new SimpleDateFormat("yyyyMMdd\_HHmmss");

Random r = new Random();

StringBuilder tempName = new StringBuilder();

tempName.append(sdf.format(new Date())).append(r.nextInt(100)).append(suffixName);

String newFileName = tempName.toString();

try {

// 保存文件

byte[] bytes = file.getBytes();

Path path = Paths.get(FILE\_UPLOAD\_PATH + newFileName);

Files.write(path, bytes);

} catch (IOException e) {

e.printStackTrace();

}

return "上传成功";

}

}

由于已经自动配置了 MultipartFile ，因此能够直接在控制器方法中使用 MultipartFile 读取文件信息， **@RequestParam** 中的文件名称需要与文件上传前端页面设置的 name 属性一致，如果文件为空则返回上传失败，如果不为空则生成一个新的文件名，之后读取文件流并写入到指定的上传路径中，最后返回上传成功。

**上传路径**

需要注意的是文件上传路径的设置，我们在代码中设置的文件保存路径为 /home/project/upload/ 即当前 project 目录下的 upload 文件夹，/home/project/upload/ 这种写法是 Linux 系统下的路径写法。如果是 Windows 系统的话，写法与此不同，比如我们想把文件上传到 D 盘下的 upload 文件夹下，就需要把路径设置代码改为

private final static String FILE\_UPLOAD\_PATH = "D:\\upload\\"

如果 project 下没有 upload 目录，你需要在 project 目录下新建 upload 目录，用于存放上传文件，也可以执行 mkdir /home/project/upload/ 命令来创建 upload 目录。

cd ..

mkdir upload

之后启动项目进行文件上传测试。

测试成功。

**Spring Boot 文件上传路径回显**

文件上传后，需要知道它的路径，最好能够在页面中直接看到它的回显效果，像前一个步骤中，只是成功的完成了文件上传，但是如何去访问这个文件还不得而知。Spring Boot 不像普通的 web 项目可以上传到 webapp 指定目录中。Springboot的通常的做法是**使用自定义静态资源映射目录，以此来实现文件上传整个流程的闭环**，比如前一小节中的实际案例，在文件上传到 upload 目录后，增加一个自定义静态资源映射，使得 upload 下的静态资源可以通过该映射地址被访问到，新建 config 包，并在包中新增 SpringBootWebMvcConfigurer 类，实现方法如下：

package com.lou.springboot.config;

import org.springframework.context.annotation.Configuration;

import org.springframework.web.servlet.config.annotation.ResourceHandlerRegistry;

import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;

@Configuration

public class SpringBootWebMvcConfigurer implements WebMvcConfigurer {

public void addResourceHandlers(ResourceHandlerRegistry registry) {

registry.addResourceHandler("/files/\*\*").addResourceLocations("file:/home/project/upload/");

}

}

通过该设置，所有以 /files/ 开头的静态资源请求都会映射到 /home/project/upload 目录下，与前面上传文件时设置目录类似，不同的系统比如 Linux 和 Windows，文件路径的写法不同。

之后修改uploadcontroller类的返回信息，把路径拼装并返回到页面上，以便于我们进行测试，代码修改如下：

return "上传成功，图片地址为：/files/" + newFileName;