

前置知识（SpringBoot文件上传与下载）

时间：2023年02月11日 19:40:03

一、Multipart

HTML中有没有一种统一的文件上传方式，例如通过表单的encrypt类型指定。

- enctype 属性规定在将表单数据发送到服务器之前如何对其进行编码。

值	描述
application/x-www-form-urlencoded	默认。在发送前对所有字符进行编码（将空格转换为 "+" 符号，特殊字符转换为 ASCII HEX 值）。
multipart/form-data	不对字符编码。当使用有文件上传控件的表单时，该值是必需的。
text/plain	将空格转换为 "+" 符号，但不编码特殊字符。

1. 什么是MultipartFile

- html表单中的一种数据encrypt格式，用于上传文件
- spring中提供的一个接口+多个实现，用于接收并处理前端web表单请求中的文件数据

ByteResource.java

MultipartFile.java

package org.springframework.web.multipart;

import java.io.InputStream;

A representation of an uploaded file received in a multipart request.
The file contents are either stored in memory or temporarily on disk. In either case, the user is responsible for copying file contents to a session-level or persistent store as and if desired. The temporary storage will be cleared at the end of request processing.
自: 25.09.2003
请参阅: MultipartHttpServletRequest, MultipartFileResolver
作者: Juergen Hoeller, Trevor D. Cook

public interface MultipartFile extends InputStreamSource {

 Return the name of the parameter in the multipart form.
 返回: the name of the parameter (never null or empty)

 String getName();

 Return the original filename in the client's filesystem.
 This may contain path information depending on the browser used, but it typically will not with any other than Opera.
 Note: Please keep in mind this filename is supplied by the client and should not be used blindly. In addition to not using the directory portion, the file name could also contain characters such as ":" and others that can be used maliciously. It is recommended to not use this filename directly. Preferably generate a unique one and save this one somewhere for reference, if necessary.
 返回: the original filename, or the empty String if no file has been chosen in the multipart form, or null if not defined or not available
 请参阅: org.apache.commons.fileupload.FileItem.getName(), org.springframework.web.multipart.commons.CommonsMultipartFile.setResourceAsStream, RFC 7578, Section 4.2 > Unrestricted File Upload <

 @Nullable
 String getOriginalFilename();

 Return the content type of the file.
 返回: the content type, or null if not defined (or no file has been chosen in the multipart form)

 @Nullable
 String getContentType();

 Return whether the uploaded file is empty, that is, either no file has been chosen in the multipart form or the chosen file has no content.

 boolean isEmpty();

 Return the size of the file in bytes.
 返回: the size of the file, or 0 if empty

 long getSize();

 Return the contents of the file as an array of bytes.
 返回: the contents of the file as bytes, or an empty byte array if empty
 抛出: IOException - in case of access errors (if the temporary store fails)

返回MultipartFile的实现方法 (3 found)

ByteResource (feign-form-spring-converter) Maven: io.github.openfeign.feign-form-spring:feign-form-spring-3.8.0 (feign-form-spring-3.8.0.jar) 8a
CommonsMultipartFile (org.springframework.web.multipart.commons) Maven: org.springframework:spring-web:5.3.22 (spring-web-5.3.22.jar) 8a
StandardMultipartFile (StandardMultipartHttpServletRequest) (org.springframework.web.multipart.support) Maven: org.springframework:spring-web:5.3.22 (spring-web-5.3.22.jar) 8a

package org.springframework.core.io;

import java.io.InputStream;

Simple interface for objects that are sources for an InputStream.
This is the base interface for Spring's more extensive Resource interface.
For single-use streams, InputStreamSource can be used for any given InputStream. Spring's ByteArrayResource or any file-based Resource implementation can be used as a concrete instance, allowing one to read the underlying content stream multiple times. This makes this interface useful as an abstract content source for mail attachments, for example.
自: 30.01.2004
请参阅: InputStream, Resource, InputStreamResource, ByteArrayResource
作者: Juergen Hoeller

public interface InputStreamSource {

 Return an InputStream for the content of an underlying resource.
 It is expected that each call creates a fresh stream.
 This requirement is particularly important when you consider an API such as javax.mail, which needs to be able to read the stream multiple times when creating mail attachments. For such a use case, it is required that each getInputStream() call returns a fresh stream.
 返回: the input stream for the underlying resource (must not be null)
 抛出: FileNotFoundException - if the underlying resource does not exist
 IOException - if the content stream could not be opened
 请参阅: Resource.isReadable()

 InputStream getInputStream() throws IOException;
}

- getName
 - 获取请求文件的属性名

POST localhost:9091/images/upload

参数 授权 Header (9) **Body** 预请求脚本 测试 设置

☐ none
 ☒ form-data
 ☐ x-www-form-urlencoded
 ☐ raw
 ☐ binary
 ☐ GraphQL

键	值	描述
<input checked="" type="checkbox"/> file	yjtp.png ×	
getName()	getOriginalName()	
键	值	描述

- *getOriginalName*
 - 获取文件名
- *getContentType*
 - image/png
 - ...(MIME类型)
- *getSize*
 - 获取文件大小 (字节)
- *isEmpty*
 - 上传文件是否为空 -> getSize==0 (例如一个空的.txt文件)

2. 多文件使用Multipart接收

- 不使用注解指定文件所在的key
- 使用注解指定文件所在的key, 此时可以使用@RequestPart或@RequestParam。

```
/**
 * 根据表单属性解析出Multipart, 可以使用@RequestPart、也可以使用@RequestParam
 *
 * @param fileArray 多文件
 */
0 个用法 新 *
@PostMapping("/upload")
public void multiImageImport(@RequestPart("files") MultipartFile[] fileArray) {

    System.out.println(Arrays.toString(fileArray));
}
```

```
0 个用法 新 *
@PostMapping("/upload")
public void multiImageImport(MultipartFile[] file) { file: MultipartFile[2]@6558
    System.out.println("file = " + file); file: MultipartFile[2]@6558
}

估计表达式(Enter)或添加监视(Ctrl+Shift+Enter)
this = {ImageController@6556}
file = {MultipartFile[2]@6558}
0 = {StandardMultipartHttpServletRequest$StandardMultipartFile@6585}
    > part = {ApplicationPart@6587}
    > filename = "新文件 1.json"
1 = {StandardMultipartHttpServletRequest$StandardMultipartFile@6586}
    > part = {ApplicationPart@6589}
    > filename = "新文件 2.json"
```

3. Multipart数据写入文件

- 使用MultipartFile提供的transferTo (File) , 但是这里只支持绝对路径, 因为相对路径默认不在项目目录下!

```
/**
 * 根据表单属性解析出Multipart, 可以使用@RequestPart、也可以使用@RequestParam
 *
 * @param fileArray 多文件
 */
0 个用法 新 *
@PostMapping("/upload")
public void multiImageImport(@RequestPart("files") MultipartFile[] fileArray) {

    for (MultipartFile multipartFile : fileArray) {
        String filename = Objects.requireNonNull(multipartFile.getOriginalFilename());
        try {
            OutputStream os = new FileOutputStream(filename);
            InputStream is = multipartFile.getInputStream();
        } {
            multipartFile.transferTo(new File(multipartFile.getOriginalFilename()));
        } catch (IOException e) {
            throw new RuntimeException(e);
        }
    }
}
```

```
@Override
public void write(String fileName) throws IOException {
    File file = new File(fileName);
    if (!file.isAbsolute()) {
        file = new File(location, fileName);
    }
    try {
        fileItem.write(file);
    } catch (Exception e) {
        throw new IOException(e);
    }
}

public String getString(String encoding) throws UnsupportedEncodingException {
    return fileItem.getString(encoding);
}

/*
 * Adapted from FileUploadBase.getFileName()
 */
@Override
public String getSubmittedFileName() {
    String fileName = null;
}

@Override
public void transferTo(File dest) throws IOException, IllegalStateException {
    this.port.write(dest.getPath());
    if (dest.isAbsolute() && !dest.exists()) {
        // Servlet 3.0 Part.write is not guaranteed to support absolute file paths:
        // may translate the given path to a relative location within a temp dir
        // (e.g. on Jetty whereas Tomcat and Undertow detect absolute paths).
        // At least we offloaded the file from memory storage; it'll get deleted
        // from the temp dir eventually in any case. And for our user's purposes,
        // we can manually copy it to the requested location as a fallback.
        FileCopyUtils.copy(this.port.getInputStream(), Files.newOutputStream(dest.toPath()));
    }
}
```

Multipart对于transferTo相对路径并不是项目的目录

- 如果想要使用相对路径，请手动获取InputStream，创建OutputStream，手动将数据写入到输出流。
 - JDK9中对InputStream提供了transferTo接口，可以直接以8KB的缓冲区大小将数据写入到输出流；

```
/**
 * 根据表单属性解析出Multipart，可以使用@RequestPart、也可以使用@RequestParam
 *
 * @param fileArray 多文件
 */

0 个用法 新 *
@PostMapping("/upload")
public void multiImageImport(@RequestPart("files") MultipartFile[] fileArray) {
    for (MultipartFile multipartFile : fileArray) {
        String filename = Objects.requireNonNull(multipartFile.getOriginalFilename());
        try {
            OutputStream os = new FileOutputStream(filename);
            InputStream is = multipartFile.getInputStream();

            // JDK9+
            // is.transferTo(os);

            final int BUFFER_SIZE = 8 * 1024;
            byte[] bytes = new byte[BUFFER_SIZE];
            int read;
            while ((read = is.read(bytes, off: 0, BUFFER_SIZE)) >= 0) {
                os.write(bytes, off: 0, read);
            }
        } catch (IOException e) {
            throw new RuntimeException(e);
        }
    }
}
```



Convert the request into a multipart request, and make multipart resolver available.

If no multipart resolver is set, simply use the existing request.

形参: request – current HTTP request

返回值: the processed request (multipart wrapper if necessary)

请参阅: `MultipartResolver.resolveMultipart`

```
protected HttpServletRequest checkMultipart(HttpServletRequest request) throws MultipartException {
    if (this.multipartResolver != null && this.multipartResolver.isMultipart(request)) {
        if (WebUtils.getNativeRequest(request, MultipartHttpServletRequest.class) != null) {
            if (DispatcherType.REQUEST.equals(request.getDispatcherType())) {
                logger.trace("Request already resolved to MultipartHttpServletRequest, e.g. by MultipartFilter");
            }
        }
    }
    else if (hasMultipartException(request)) {
        logger.debug("Multipart resolution previously failed for current request - " +
            "skipping re-resolution for undisturbed error rendering");
    }
    else {
        try {
            return this.multipartResolver.resolveMultipart(request);
        }
        catch (MultipartException ex) {
            if (request.getAttribute(WebUtils.ERROR_EXCEPTION_ATTRIBUTE) != null) {
                logger.debug("message: \"Multipart resolution failed for error dispatch\", ex);
                // Keep processing error dispatch with regular request handle below
            }
            else {
                throw ex;
            }
        }
    }
    // If not returned before: return original request.
    return request;
}
```

multipart处理器

StandardMultipartHttpServletRequest.java

```
71 public StandardMultipartHttpServletRequest(HttpServletRequest request) throws MultipartException {
72     this(request, false);
73 }
74
75 // Create a new StandardMultipartHttpServletRequest wrapper for the given request.
76 // @param request - the servlet request to wrap
77 // @param lazyParsing - whether multipart parsing should be triggered lazily on first access of
78 // multipart files or parameters
79 // @throws MultipartException - if an immediate parsing attempt failed
80 // @since 3.2.9
81 public StandardMultipartHttpServletRequest(HttpServletRequest request, boolean lazyParsing)
82     throws MultipartException {
83     super(request);
84     if (!lazyParsing) {
85         parseRequest(request);
86     }
87 }
88
89 private void parseRequest(HttpServletRequest request) {
90     try {
91         Collection<Part> parts = request.getParts();
92         this.multipartParameterNames = new LinkedHashSet<>(parts.size());
93         MultiValueMap<String, MultipartFile> files = new LinkedMultiValueMap<>(parts.size());
94         for (Part part : parts) {
95             String headerValue = part.getHeader(Headers.CONTENT_DISPOSITION);
96             ContentDisposition disposition = ContentDisposition.parse(headerValue);
97             String filename = disposition.getFilename();
98             if (filename != null) {
99                 this.multipartParameterNames.add(filename);
100                 files.add(part.getHeader(Headers.CONTENT_TYPE), part.getFile());
101             }
102         }
103     }
104 }
```

StandardServletMultipartResolver.java

```
111
112
113
114 @Override
115 public boolean isMultipart(HttpServletRequest request) {
116     return StringUtils.startsWithIgnoreCase(request.getContentType(),
117         (this.strictServletCompliance ? MediaType.MULTIPART_FORM_DATA_VALUE : "multipart/"));
118 }
119
120 @Override
121 public MultipartHttpServletRequest resolveMultipart(HttpServletRequest request) throws MultipartException {
122     return new StandardMultipartHttpServletRequest(request, true);
123 }
124
```

构造函数中传入标记tag，在构造函数初始化时后根据tag执行解析逻辑

二、MIME解析？

https://developer.mozilla.org/zh-CN/docs/Web/HTTP/Basics_of_HTTP/MIME_types

- spring-web也提供了MIME解析类

```
@SpringBootApplication
public class App {

    0 个用法  trivis *
    public static void main(String[] args) {
        System.out.println(MediaTypeFactory.getMediaType(filename: "aaa.tzt").orElse(MediaType.TEXT_XML)); // X
        System.out.println(MediaTypeFactory.getMediaType(filename: "aaa.txt").orElse(MediaType.TEXT_XML));
        System.out.println(MediaTypeFactory.getMediaType(filename: "aaa.png").orElse(MediaType.TEXT_XML));
        System.out.println(MediaTypeFactory.getMediaType(filename: "aaa.gif").orElse(MediaType.TEXT_XML));
        System.out.println(MediaTypeFactory.getMediaType(filename: "aaa.xlsx").orElse(MediaType.TEXT_XML));
        System.out.println(MediaTypeFactory.getMediaType(filename: "aaa.pdf").orElse(MediaType.TEXT_XML));
        System.out.println(MediaTypeFactory.getMediaType(filename: "aaa.ts").orElse(MediaType.TEXT_XML)); // X
        System.out.println(MediaTypeFactory.getMediaType(filename: "aaa.flv").orElse(MediaType.TEXT_XML));
        System.out.println(MediaTypeFactory.getMediaType(filename: "aaa.mp4").orElse(MediaType.TEXT_XML));
        SpringApplication.run(App.class, args);
    }
}
```

三、application/octet-stream?

默认blob类型为text/xml，如果手动设置为application/octet-stream，浏览器则不会自动预览（图片、PDF）

- 而是统一下载为一个URI名称的.file格式文件

```
@GetMapping("/download_s1")
public void singleDownload1(HttpServletResponse response) {
    // final String filename = "abc.png";
    // final String filename = "Java面试必知必会.pdf";
    final String filename = "abc.xlsx";
    ClassPathResource classPathResource = new ClassPathResource(filename);
    try (InputStream is = classPathResource.getInputStream(); OutputStream os = response.getOutputStream()) {
        // 需要主动暴露Content-Disposition，否则Axios获取不到响应头的这个header属性
        response.setHeader("Access-Control-Expose-Headers", "Content-Disposition");
        response.setHeader("Content-Disposition", "attachment;filename=" + URLEncoder.encode(filename, StandardCharsets.UTF_8));
        response.addHeader("Content-Length", "" + is.available());

        is.transferTo(response.getOutputStream());
        //final int BUFFER_SIZE = 10 * 1024 * 1024;
        //byte[] buf = new byte[BUFFER_SIZE];
        //int read;
        //while ((read = is.read(buf, 0, BUFFER_SIZE)) >= 0) {
        //    os.write(buf, 0, read);
        //}
    } catch (IOException e) {
        throw new RuntimeException(e);
    }
}
```

Content-Type默认text/xml -> 浏览器自动预览
一旦设置Content-Type为application/octet-stream，
浏览器则不会自动预览，而是下载一个uri命名的文件

不配置这些

四、Axios无法获取到Content-Disposition?

默认情况下，header只有六种 simple response headers（简单响应首部）可以暴露给外部：

- Cache-Control
- Content-Language
- Content-Type
- Expires
- Last-Modified
- Pragma

1. 解决方案一（全局控制）

```
import lombok.NonNull;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.web.servlet.config.annotation.CorsRegistry;
import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;

@Configuration
public class CORSConfiguration {

    /**
     * 注意：跨域会导致页面获取不到response-headers!
     */
    * @return WebMvcConfigurer
    */
    @Bean
    public WebMvcConfigurer corsConfigurer() {
        return new WebMvcConfigurer() {
            @Override
            public void addCorsMappings(@NonNull CorsRegistry registry) {
                registry.addMapping("/images/**")
                    .allowedOrigins("**")
                    .allowedMethods("GET", "POST")
                    .exposedHeaders("Content-Disposition",
                        "access-control-allow-headers",
                        "Access-Control-Expose-Headers",
                        "access-control-allow-methods",
                        "access-control-allow-origin",
                        "access-control-max-age",
                        "X-Frame-Options")
                    .allowCredentials(false).maxAge(3600);
            }
        };
    }
}
```



```

> res
< ▼ (data: Blob, status: 200, statusText: '', headers: i, config: {_, _})
  ► config: {transitional: {_, adapter: Array(2), transformRequest: Array(1), transformResponse: Array(1), timeout: 30000, _}}
  ► data: Blob {size: 9794, type: 'text/xml'}
  ▼ headers: i
    access-control-allow-origin: ""
    access-control-expose-headers: "Content-Disposition, access-control-allow-headers, Access-Control-Expose-Headers, access-control-allow-methods, access-control-allow-origin, access-control-max-age, X-Frame-Options"
    content-disposition: "attachment;filename=abc.xlsx"
    content-length: "9794"
    Symbol(Symbol.toStringTag): (...)
  ► [[Prototype]]: Object
  ► request: XMLHttpRequest {onreadystatechange: null, readyState: 4, timeout: 30000, withCredentials: false, upload: XMLHttpRequestUpload, _}
  status: 200
  statusText: ""
  ► [[Prototype]]: Object

```

▼ 响应头

[查看源](#)

Access-Control-Allow-Origin: *

Access-Control-Expose-Headers: Content-Disposition, access-control-allow-headers, Access-Control-Expose-Headers, access-control-allow-methods

Connection: keep-alive

Content-Disposition: attachment;filename=abc.xlsx

Content-Length: 9794

Date: Sat, 11 Feb 2023 11:14:38 GMT

Keep-Alive: timeout=60

Vary: Origin

Vary: Access-Control-Request-Method

Vary: Access-Control-Request-Headers

2. 方案二（针对指定接口进行控制）

```
response.setHeader("Access-Control-Expose-Headers", "Content-Disposition");
```

```

/**
 * 使用Content-Disposition进行图片下载
 *
 * @param response HttpServletResponse
 */
@GetMapping("/download_s1")
public void singleDownload1(HttpServletResponse response) {
    // final String filename = "abc.png";
    // final String filename = "Java面试必知必会.pdf";
    final String filename = "abc.xlsx";
    ClassPathResource classPathResource = new ClassPathResource(filename);
    try (InputStream is = classPathResource.getInputStream(); OutputStream os = response.getOutputStream()) {
        response.setHeader("Access-Control-Expose-Headers", "Content-Disposition");
        response.setHeader("Content-Disposition", "attachment;filename=" + URLEncoder.encode(filename,
StandardCharsets.UTF_8));
        response.addHeader("Content-Length", "" + is.available());

        is.transferTo(response.getOutputStream());
        //final int BUFFER_SIZE = 10 * 1024 * 1024;
        //byte[] buf = new byte[BUFFER_SIZE];
        //int read;
        //while ((read = is.read(buf, 0, BUFFER_SIZE)) >= 0) {
        //    os.write(buf, 0, read);
        //}
    } catch (IOException e) {
        throw new RuntimeException(e);
    }
}

```



```

        "Access-Control-Expose-Headers",
        "access-control-allow-methods",
        "access-control-allow-origin",
        "access-control-max-age",
        "X-Frame-Options")
        .allowCredentials(false).maxAge(3600);
    }
};
}
}
}

```

六、SpringBoot上传文件太大报错？

1. 上传文件大小超出限制

```

org.apache.tomcat.util.http.fileupload.impl.FileSizeLimitExceededException Create breakpoint : The field file exceeds its maximum permitted size of 10485760 bytes. <3 个内联行>
at java.base/java.io.FilterInputStream.read(FilterInputStream.java:107) ~[na:na] <5 个内联行>
at org.springframework.web.multipart.support.StandardMultipartHttpServletRequest.parseRequest(StandardMultipartHttpServletRequest.java:95) ~[spring-web-5.3.22.jar:5.3.22] 设置10M后，超出会抛出异常（不配置默认1M）
at org.springframework.web.multipart.support.StandardMultipartHttpServletRequest.<init>(StandardMultipartHttpServletRequest.java:88) ~[spring-web-5.3.22.jar:5.3.22]
at org.springframework.web.multipart.support.StandardServletMultipartResolver.resolveMultipart(StandardServletMultipartResolver.java:122) ~[spring-web-5.3.22.jar:5.3.22] <5 个内联行>
at javax.servlet.http.HttpServlet.service(HttpServlet.java:681) ~[tomcat-embed-core-9.0.65.jar:4.0.FR] <1 个内联行>
at javax.servlet.http.HttpServlet.service(HttpServlet.java:764) ~[tomcat-embed-core-9.0.65.jar:4.0.FR] <33 个内联行>

```

```

spring:
  servlet:
    multipart:
      max-file-size: 100MB
      max-request-size: 300MB

```

```

<h3>上传单个文件</h3>
<div style="text-align: center">
  <form action="upload" enctype="multipart/form-data" method="post">
    <input type="file" name="file" required>
    <br>
    <input type="submit" value="上传单个文件">
  </form>
</div>
<br><hr style="width: 400px;">
<h3>上传多个文件</h3>
<form action="multiUpload" enctype="multipart/form-data" method="post">
  <input type="file" name="file" required>
  <br>
  <input type="file" name="file" required>
  <br>
  <input type="file" name="file" required>
  <br>
  <input type="submit" value="上传多个文件">
</form>

```

2. POST请求大小超出限制

配置Tomcat允许的POST请求最大值

```

server:
  port: 8081
  tomcat:
    threads:
      max: 200
    max-connections: 8192
    max-http-form-post-size: 2MB
    max-swallow-size: 2MB

```

调整为-1表示不限制

form表单中post请求最大大小
requestBody最大大小

提示!
前端如果使用Axios统一发送get/post请求,
请在后端调节max-swallow-size至合适大小
↓
特别是某些业务需要上传序列化后的二进制数据,
此时requestBody中数据很容易会>2MB

(下方配置是默认值)

```

server:
  port: 8081
  tomcat:
    threads:
      max: 200
    max-connections: 8192
    max-http-form-post-size: 2MB
    max-swallow-size: 2MB
    max-http-header-size: 8KB

spring:
  servlet:
    multipart:
      max-file-size: 1MB
      max-request-size: 10MB

```

