



链滴

# 大文件切片上传、视频切片上传转 m3u8 播放

作者: [sirwsl](#)

原文链接: <https://ld246.com/article/1643253584778>

来源网站: [链滴](#)

许可协议: [署名-相同方式共享 4.0 国际 \(CC BY-SA 4.0\)](#)



## 一、故事

前不久干项目，涉及到在线学习，简单来说就是对文章、视频进行在线学习，这个时候问题出现了，是在上传视频的时候，速度很是慢，除此之外，视频播放也是卡的鸭皮，然后就开始疯狂网上搜刮知识，最终解决方案如下。

## 二、解决方案

- 1、视频采用切片上传，通过调用后端切片上传接口进行上传
- 2、切片上传结束后通过合并切片接口进行合并成为完整的视频
- 3、调用ffmpeg工具进行视频转m3u8格式形成ts切片
- 4、ts切片多线程上传至MinIO or OSS
- 5、返回m3u8格式文件地址，前端集成播放器进行播放。

## 三、项目地址

码云: <https://gitee.com/sirwsl/uploadWheel>

GitHub: <https://github.com/sirwsl/uploadWheel>

## 四、demo效果展示



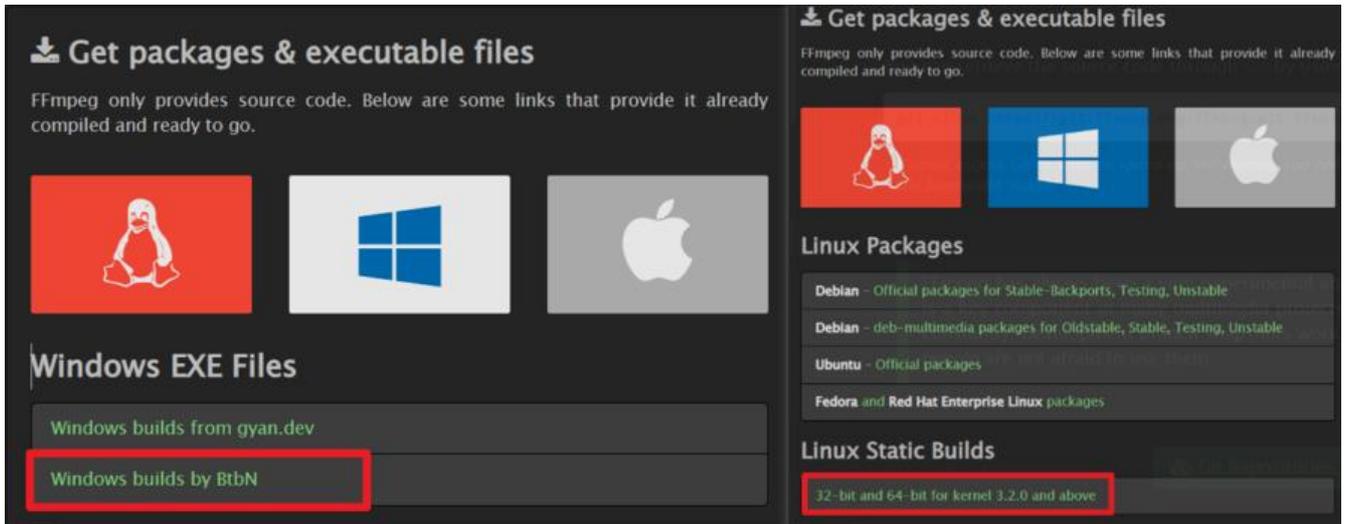
## 五、实现过程

### 1、下载ffmpeg

为了开发方便，建议下载windows和linux两个版本

地址：[Download FFmpeg](#)

hankey 比较懒的也可以跳过，毕竟代码里面我已经弄好了  
ankey



PS: 此处需要注意, 由于网上说的基本都需要配置环境变量, 所以此处需要注意, 下载的文件, 按上方式选择自己适合的版本下载解压。不需要配置环境变量

## 2、开始构建springBoot程序

1) 导入依赖:

关键依赖项

```
...
<ffmpeg.version>0.6.2</ffmpeg.version>
...
<dependency>
  <groupId>net.bramp.ffmpeg</groupId>
  <artifactId>ffmpeg</artifactId>
  <version>${ffmpeg.version}</version>
</dependency>
```

完整依赖:

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/
MLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xs
/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.wsl</groupId>
  <artifactId>uploadWheel</artifactId>
  <version>1</version>
  <name>uploadWheel</name>
  <description>minIO文件上传及切片demo</description>

  <properties>
    <java.version>1.8</java.version>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>
    <spring-boot.version>2.3.7.RELEASE</spring-boot.version>
    <ffmpeg.version>0.6.2</ffmpeg.version>
```

```
<hutool.version>5.7.15</hutool.version>
<aliyun-sdk-oss.version>3.13.2</aliyun-sdk-oss.version>
<fastjson.version>1.2.72</fastjson.version>
<minio.version>6.0.8</minio.version>
</properties>

<dependencies>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-thymeleaf</artifactId>
  </dependency>
  <dependency>
    <groupId>com.alibaba</groupId>
    <artifactId>fastjson</artifactId>
    <version>${fastjson.version}</version>
  </dependency>

  <dependency>
    <groupId>com.aliyun.oss</groupId>
    <artifactId>aliyun-sdk-oss</artifactId>
    <version>${aliyun-sdk-oss.version}</version>
  </dependency>
  <dependency>
    <groupId>org.projectlombok</groupId>
    <artifactId>lombok</artifactId>
    <optional>>true</optional>
  </dependency>
  <dependency>
    <groupId>io.minio</groupId>
    <artifactId>minio</artifactId>
    <version>${minio.version}</version>
  </dependency>
  <dependency>
    <groupId>net.bramp.ffmpeg</groupId>
    <artifactId>ffmpeg</artifactId>
    <version>${ffmpeg.version}</version>
  </dependency>

  <!--hutool-->
  <dependency>
    <groupId>cn.hutool</groupId>
    <artifactId>hutool-all</artifactId>
    <version>${hutool.version}</version>
  </dependency>
  <!--配置文件处理器-->
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-configuration-processor</artifactId>
    <version>${spring-boot.version}</version>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
```

```

    <exclusions>
      <exclusion>
        <groupId>org.junit.vintage</groupId>
        <artifactId>junit-vintage-engine</artifactId>
      </exclusion>
    </exclusions>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
  </dependency>
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-test</artifactId>
  </dependency>
</dependencies>

<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-dependencies</artifactId>
      <version>${spring-boot.version}</version>
      <type>pom</type>
      <scope>import</scope>
    </dependency>
  </dependencies>
</dependencyManagement>

<build>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-compiler-plugin</artifactId>
      <version>3.8.1</version>
      <configuration>
        <source>1.8</source>
        <target>1.8</target>
        <encoding>UTF-8</encoding>
      </configuration>
    </plugin>
    <plugin>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-maven-plugin</artifactId>
      <version>${spring-boot.version}</version>
      <configuration>
        <mainClass>com.wslhome.demo.Application</mainClass>
      </configuration>
    </plugin>
  </plugins>
  <executions>
    <execution>
      <id>repackage</id>
      <goals>
        <goal>repackage</goal>
      </goals>
    </execution>
  </executions>
</build>

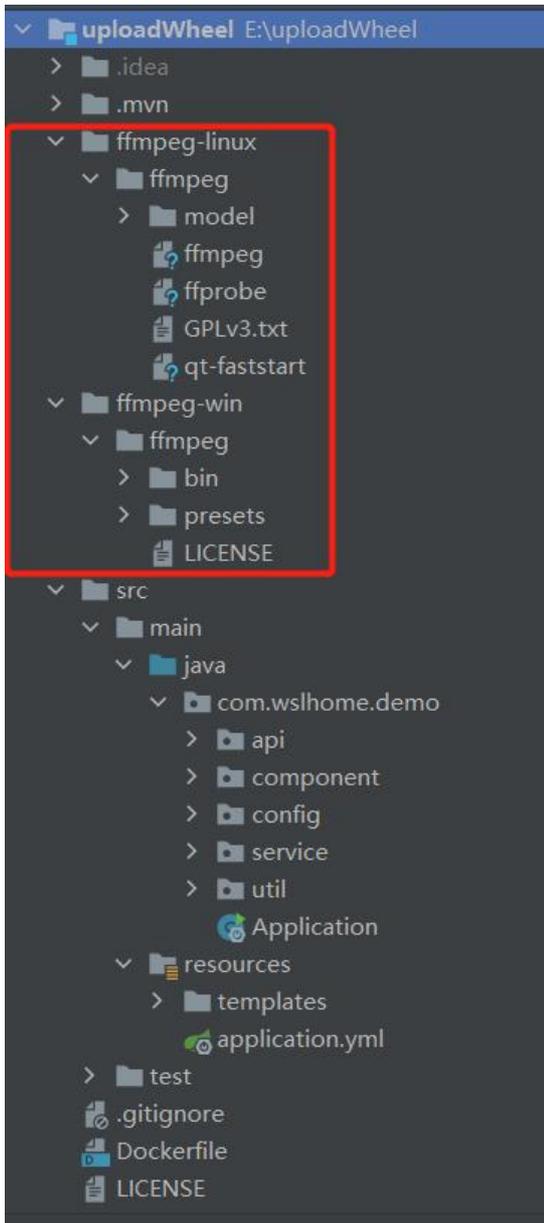
```

```
        </goals>
    </execution>
</executions>
</plugin>
</plugins>
<resources>
    <resource>
        <directory>src/main/java</directory>
        <includes>
            <include>**/*.properties</include>
            <include>**/*.xml</include>
            <include>**/*.yml</include>
        </includes>
        <!--是否替换资源中的属性-->
        <filtering>>false</filtering>
    </resource>
    <resource>
        <directory>src/main/resources</directory>
        <includes>
            <include>**/*.*</include>
        </includes>
        <!--是否替换资源中的属性-->
        <filtering>>false</filtering>
    </resource>
</resources>
</build>

</project>
```

### 3、确定项目结构

为了开发方便，同时以后谁都能启动这项目，所以把ffmpeg的两个版本都丢进来。需要注意版本问。



小事故：windows的ffmpeg倒是容易搞到，毕竟就只需要两个exe，问题是linux版本的时候出现问题，因为是采用docker打包，所以我不想配置什么环境变量、下载ffmpeg之类的。所以ffmpeg的linux版本选择很重要。

开始时候我直接打包后，发现这个demo是运行不了的，因为读取不到exe文件，之后尝试了去linux装好后copy下来，发现也不行（因为姿势不对）所以下载时候版本很重要。

## 六、开始编码

### 1、编写配置文件

PS: 1) minio or OSS 配置文件，任选其一进行配置

2) thymeleaf 部分可以不要，此处为了展示demo

```
server:  
  port: 8080
```

```

minio:
  url: #http://
  access: #admin
  secret: #admin

#m3u8视频转换配置
m3u8:
  convertor:
    base-path: /file/m3u8/
    temp-path: /file/temp/
    big-path: /file/big/
    proxy: m3u8/

ali:
  oss:
    end-point:
    access-key-id:
    access-key-secret:
    bucket-name: websources
    url:
    ali-url:
    get-file-url: ${aliyun.oss.url}${aliyun.oss.fileDir}
    my-host-url:

# 应用名称
spring:
  application:
    name: minioDemo
  servlet:
    multipart:
      max-file-size: 2048MB
      max-request-size: 2048MB
  resources:
    static-locations: classpath:static/
  thymeleaf:
    cache: false
    check-template: true # 检查模板是否存在, 然后再呈现
    check-template-location: true # 检查模板位置是否正确 (默认值 :true )
    enabled: true # 开启 MVC Thymeleaf 视图解析 (默认值: true )
    encoding: UTF-8 # 模板编码
    excluded-view-names: # 要被排除在解析之外的视图名称列表, 用逗号分隔
    mode: HTML5 # 要运用于模板之上的模板模式。另见 StandardTemplate-ModeHandlers( 默认值: HTML5)
    prefix: classpath:/templates/ # 在构建 URL 时添加到视图名称前的前缀 (默认值: classpath:/templates/)
    suffix: .html # 在构建 URL 时添加到视图名称后的后缀 (默认值: .html )
  servlet:
    content-type: text/html #Content-Type 的值 (默认值: text/html )

```

## 2、读取ffmpeg

先写好config来读取ffmpeg，因为我们没配置环境变量，所以这个很关键

```
package com.wslhome.demo.config;

import lombok.SneakyThrows;
import lombok.extern.slf4j.Slf4j;
import net.bramp.ffmpeg.FFmpeg;
import net.bramp.ffmpeg.FFprobe;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

/**
 * ffmpeg 工具路径设置
 * @Author sirwsl
 * @Version 1.0
 */
@Configuration
@Slf4j
public class FFmpegConfig {

    @SneakyThrows
    @Bean
    public FFmpeg fFmpeg() {
        String path = System.getProperty("user.dir");
        if (isLinux()){
            path += "ffmpeg/ffmpeg";
        }else if (isWindows()){
            path += "/ffmpeg-win/ffmpeg/bin/ffmpeg.exe";
        }
        log.info("ffmpeg.exe 路径为{}",path);
        return new FFmpeg(path);
    }

    @SneakyThrows
    @Bean
    public FFprobe fFprobe() {
        String path = System.getProperty("user.dir");
        if (isLinux()){
            path += "ffmpeg/ffprobe";
        }else if (isWindows()){
            path += "/ffmpeg-win/ffmpeg/bin/ffprobe.exe";
        }
        log.info("ffprobe.exe 路径为{}",path);
        return new FFprobe(path);
    }

    public static boolean isLinux() {
        return System.getProperty("os.name").toLowerCase().contains("linux");
    }

    public static boolean isWindows() {
        return System.getProperty("os.name").toLowerCase().contains("windows");
    }
}
```

```
}
```

### 3、编写存储路径配置文件

因为上传切片、合并文件、m3u8转码等会涉及到较多的临时文件，所以得对这些乱七八糟的文件夹进行管理

```
package com.wslhome.demo.config;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
import org.springframework.boot.context.properties.ConfigurationProperties;
import org.springframework.stereotype.Component;

/**
 * @Author wsl
 * @Version 1.0
 */
@Data
@Component
@ConfigurationProperties(prefix = "m3u8.convertor")
public class FilePath {
    /**
     * 文件上传临时路径 (本地文件转换不需要)
     */
    private String tempPath = "/file/tmp/";

    /**
     * m3u8文件转换后，储存的根路径
     */
    private String basePath = "/file/m3u8/";

    /**
     * m3u8文件转换后，储存的根路径
     */
    private String bigPath = "/file/big/";

    private String proxy = "m3u8/";
}
```

### 3、来个线程池，管理多线程上传

```
package com.wslhome.demo.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.scheduling.annotation.EnableAsync;
import org.springframework.scheduling.concurrent.ThreadPoolTaskExecutor;

import java.util.concurrent.Executor;

/**
```

```

* 线程池配置
*
* @author sirwsl
* @date 2022/01/11-11:14
**/
@Configuration
@EnableAsync
public class SpringAsyncConfig {
    /**
     * 线程池参数根据minIO设置，如果开启线程太多会被MinIO拒绝
     * @return :
     */
    @Bean("minIOUploadTreadPool")
    public ThreadPoolTaskExecutor asyncServiceExecutorForMinIo() {
        ThreadPoolTaskExecutor executor = new ThreadPoolTaskExecutor();
        // 设置核心线程数，采用IO密集 h/(1-拥塞)
        executor.setCorePoolSize(6);
        // 设置最大线程数,由于minIO连接数量有限，此处尽力设计大点
        executor.setMaxPoolSize(500);
        // 设置线程活跃时间（秒）
        executor.setKeepAliveSeconds(30);
        // 设置默认线程名称
        executor.setThreadNamePrefix("minio-upload-task-");
        // 等待所有任务结束后再关闭线程池
        executor.setWaitForTasksToCompleteOnShutdown(true);
        //执行初始化
        executor.initialize();
        return executor;
    }

    /**
     * oss async
     * @return
     */
    @Bean("ossUploadTreadPool")
    public ThreadPoolTaskExecutor asyncServiceExecutorForOss() {
        ThreadPoolTaskExecutor executor = new ThreadPoolTaskExecutor();
        // 设置核心线程数，采用IO密集 h/(1-拥塞)
        executor.setCorePoolSize(8);
        // 设置最大线程数,由于minIO连接数量有限，此处尽力设计大点
        executor.setMaxPoolSize(120);
        // 设置线程活跃时间（秒）
        executor.setKeepAliveSeconds(30);
        // 设置默认线程名称
        executor.setThreadNamePrefix("ossUploadTask-");
        // 等待所有任务结束后再关闭线程池
        executor.setWaitForTasksToCompleteOnShutdown(true);
        //执行初始化
        executor.initialize();
        return executor;
    }
}

```

## 4、编写minIO or OSS 的上传组件

这玩意以OSS为例，Google一下，到处都是

### 1) 阿里

```
package com.wslhome.demo.config;

import lombok.Data;
import org.springframework.boot.context.properties.ConfigurationProperties;
import org.springframework.stereotype.Component;

@Component
@ConfigurationProperties(prefix = "ali.oss")
@Data
public class AliOssProperties {

    /**
     * OSS配置信息
     */

    private String endpoint;

    private String accessKeyId;

    private String accessKeySecret;

    private String bucketName;

    private String myHostUrl;

    private String url;

    private String aliUrl;
}
```

### 2) OSS组件

```
package com.wslhome.demo.component;

import com.aliyun.oss.OSS;
import com.aliyun.oss.OSSClientBuilder;
import com.aliyun.oss.model.*;
import com.wslhome.demo.config.AliOssProperties;;
import lombok.Getter;
import lombok.extern.slf4j.Slf4j;
import org.apache.commons.lang3.StringUtils;
import org.springframework.stereotype.Component;
import org.springframework.web.multipart.MultipartFile;

import javax.annotation.Resource;
```

```

import java.io.*;
import java.net.URL;
import java.util.*;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.LinkedBlockingQueue;
import java.util.concurrent.ThreadPoolExecutor;
import java.util.concurrent.TimeUnit;

/**
 * 阿里云 OSS 工具类
 *
 * @author caibenhao
 */
@Component
@Slf4j
@Getter
public class OssComponent{

    @Resource
    private AliOssProperties aliOssProperties;

    /** -----对外功能----- */

    /**
     * 本地文件切片上传
     *
     * @param objectName: 文件名
     * @param path      : 本地完整路径, xxx/xxx.txt
     * @return : 异常
     */
    public String uploadSlice(String objectName, String localPath,String path) throws IOExcepti
n {
        OSS ossClient = new OSSClientBuilder().build(aliOssProperties.getEndpoint(), aliOssProp
erties.getAccessKeyId(), aliOssProperties.getAccessKeySecret());
        String keyPath = path+objectName;
        // 创建InitiateMultipartUploadRequest对象。
        InitiateMultipartUploadRequest request = new InitiateMultipartUploadRequest(aliOssPr
operties.getBucketName(), keyPath);
        // 如果需要在初始化分片时设置请求头, 请参考以下示例代码。
        ObjectMetadata metadata = new ObjectMetadata();
        // 指定该Object的网页缓存行为。
        metadata.setCacheControl("no-cache");
        // 指定该Object被下载时的名称。
        metadata.setContentDisposition("attachment;filename=" + objectName);
        // 指定初始化分片上传时是否覆盖同名Object。此处设置为true, 表示禁止覆盖同名Object。
        metadata.setHeader("x-oss-forbid-overwrite", "true");

        // 初始化分片。
        InitiateMultipartUploadResult result = ossClient.initiateMultipartUpload(request);
        // 返回uploadId, 它是分片上传事件的唯一标识。您可以根据该uploadId发起相关的操作, 例
取消分片上传、查询分片上传等。
        String uploadId = result.getUploadId();
    }
}

```

```
List<PartETag> partETags = new ArrayList<>();
// 每个分片的大小，用于计算文件有多少个分片。单位为字节。
final long partSize = 5 * 1024 * 1024L; //1 MB。
```

// 填写本地文件的完整路径。如果未指定本地路径，则默认从示例程序所属项目对应本地路径上传文件。

```
final File sampleFile = new File(localPath);
long fileLength = sampleFile.length();
int partCount = (int) (fileLength / partSize);
if (fileLength % partSize != 0) {
    partCount++;
}
```

// 遍历分片上传。

```
for (int i = 0; i < partCount; i++) {
    long startPos = i * partSize;
    long curPartSize = (i + 1 == partCount) ? (fileLength - startPos) : partSize;
```

```
try (InputStream inStream = new FileInputStream(sampleFile)) {
```

// 跳过已经上传的分片。

```
long skip = inStream.skip(startPos);
```

```
UploadPartRequest uploadPartRequest = new UploadPartRequest();
```

```
uploadPartRequest.setBucketName(aliOssProperties.getBucketName());
```

```
uploadPartRequest.setKey(keyPath);
```

```
uploadPartRequest.setUploadId(uploadId);
```

```
uploadPartRequest.setInputStream(inStream);
```

// 设置分片大小。除了最后一个分片没有大小限制，其他的分片最小为100 KB。

```
uploadPartRequest.setPartSize(curPartSize);
```

// 设置分片号。每一个上传的分片都有一个分片号，取值范围是1~10000，如果超出此范围，OSS将返回InvalidArgument错误码。

```
uploadPartRequest.setPartNumber(i + 1);
```

// 每个分片不需要按顺序上传，甚至可以在不同客户端上传，OSS会按照分片号排序组成完整的文件。

```
UploadPartResult uploadPartResult = ossClient.uploadPart(uploadPartRequest);
```

// 每次上传分片之后，OSS的返回结果包含PartETag。PartETag将被保存在partETags中

```
partETags.add(uploadPartResult.getPartETag());
```

```
}catch (Exception e){
```

```
log.error("OSS切片上传异常,e:{}",e.getMessage());
```

```
}
```

```
}
```

// 创建CompleteMultipartUploadRequest对象。

// 在执行完成分片上传操作时，需要提供所有有效的partETags。OSS收到提交的partETags后会逐一验证每个分片的有效性。当所有的数据分片验证通过后，OSS将把这些分片组合成一个完整的文件。

```
CompleteMultipartUploadRequest completeMultipartUploadRequest =
```

```
new CompleteMultipartUploadRequest(aliOssProperties.getBucketName(),keyPath ,
uploadId, partETags);
```

// 完成分片上传。

```
CompleteMultipartUploadResult completeMultipartUploadResult = ossClient.complete
multipartUpload(completeMultipartUploadRequest);
```

```

        log.info(completeMultipartUploadResult.getETag());
        // 关闭OSSClient。
        ossClient.shutdown();
        return path+objectName;
    }

    /**
     * 单个文件上传
     *
     * @param file 文件
     * @return 返回完整URL地址
     */
    public String uploadFile(String fileDir, MultipartFile file) {
        String fileUrl = upload2Oss(fileDir, file);
        String str = getFileUrl(fileDir, fileUrl);
        return str.trim();
    }

    /**
     * 单个文件上传(指定文件名 (带后缀) )
     *
     * @param inputStream 文件
     * @param fileName 文件名(带后缀)
     * @return 返回完整URL地址
     */
    public String uploadFile(String fileDir, InputStream inputStream, String fileName) {
        try {
            this.uploadFile2Oss(fileDir, inputStream, fileName);
            String url = getFileUrl(fileDir, fileName);
            if (url != null && url.length() > 0) {
                return url;
            }
        } catch (Exception e) {
            throw new RuntimeException("获取路径失败");
        }
        return "";
    }

    /**
     * 多文件上传
     *
     * @param fileList 文件列表
     * @return 返回完整URL, 逗号分隔
     */
    public String uploadFile(String fileDir, List<MultipartFile> fileList) {
        String fileUrl;
        String str;
        StringBuilder photoUrl = new StringBuilder();
        for (int i = 0; i < fileList.size(); i++) {
            fileUrl = upload2Oss(fileDir, fileList.get(i));
            str = getFileUrl(fileDir, fileUrl);
            if (i == 0) {
                photoUrl = new StringBuilder(str);
            } else {

```

```

        photoUrl.append(",").append(str);
    }
}
return photoUrl.toString().trim();
}

public boolean deleteFile(String fileDir, String fileName) {
    OSS ossClient = new OSSClientBuilder().build(aliOssProperties.getEndpoint(), aliOssProperties.getAccessKeyId(), aliOssProperties.getAccessKeySecret());
    // 删除文件
    ossClient.deleteObject(aliOssProperties.getBucketName(), fileDir + fileName);
    // 判断文件是否存在
    boolean found = ossClient.doesObjectExist(aliOssProperties.getBucketName(), fileDir + fileName);
    // 如果文件存在则删除失败

    return !found;
}

/**
 * 通过文件名获取完整路径
 *
 * @param fileUrl 文件名
 * @return 完整URL路径
 */
public String getFileUrl(String fileDir, String fileUrl) {
    if (fileUrl != null && fileUrl.length() > 0) {
        String[] split = fileUrl.replaceAll("\\\\", "/").split("/");
        String url = aliOssProperties.getMyHostUrl() + fileDir + split[split.length - 1];
        return Objects.requireNonNull(url);
    }
    return null;
}

public File getFile(String url) {
    //对本地文件命名
    String fileName = url.substring(url.lastIndexOf("."));
    File file = null;
    try {
        file = File.createTempFile("net_url", fileName);
    } catch (Exception e) {
        log.error("创建默认文件夹net_url失败! 原因e:{}", e.getMessage());
    }
    if (file != null) {
        try (InputStream inStream = new URL(url).openStream();
            OutputStream os = new FileOutputStream(file)) {
            int bytesRead;
            byte[] buffer = new byte[8192];
            while ((bytesRead = inStream.read(buffer, 0, 8192)) != -1) {
                os.write(buffer, 0, bytesRead);
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}

```

```

    }
    return file;
}

/* -----内部辅助功能----- */

/**
 * 获取去掉参数的完整路径
 *
 * @param url URL
 * @return 去掉参数的URL
 */
private String getShortUrl(String url) {
    String[] imgUrls = url.split("\\?");
    return imgUrls[0].trim();
}

/**
 * 获得url真实外网链接
 * 不提供使用，因为会产生公网OOS流量下行费用
 *
 * @param key 文件名
 * @return URL
 */
@Deprecated
private String getUrl(String key) {
    OSS ossClient = new OSSClientBuilder().build(aliOssProperties.getEndpoint(), aliOssProp
erties.getAccessKeyId(), aliOssProperties.getAccessKeySecret());
    // 设置URL过期时间为20年 3600* 1000*24*365*20
    Date expiration = new Date(System.currentTimeMillis() + 3600L * 1000 * 24 * 365 * 20);
    URL url = ossClient.generatePresignedUrl(aliOssProperties.getBucketName(), key, expirat
on);
    if (url != null) {
        String replaceUrl = url.toString()
            .replace(aliOssProperties.getAliUrl(), aliOssProperties.getUrl());
        return getShortUrl(replaceUrl);
    }
    ossClient.shutdown();
    return null;
}

/**
 * 上传文件
 *
 * @param file 文件
 * @return 文件名
 */
private String upload2Oss(String fileDir, MultipartFile file) {
    // 2、重命名文件
    String fileName = Objects.requireNonNull(file.getOriginalFilename(), "文件名不能为空");
    // 文件后缀
    String suffix = fileName.substring(fileName.lastIndexOf(".")).toLowerCase(Locale.ENGLIS
);
    String uuid = UUID.randomUUID().toString();

```

```

String name = uuid + suffix;
try {
    InputStream inputStream = file.getInputStream();
    this.uploadFile2Oss(fileDir, inputStream, name);
    return name;
} catch (Exception e) {
    throw new RuntimeException("上传失败");
}
}

/**
 * 上传文件（指定文件名）
 *
 * @param inputStream 输入流
 * @param fileName 文件名
 */
private void uploadFile2Oss(String fileDir, InputStream inputStream, String fileName) {
    OSS ossClient = new OSSClientBuilder().build(aliOssProperties.getEndpoint(), aliOssProp
rties.getAccessKeyId(), aliOssProperties.getAccessKeySecret());
    String ret;
    try {
        //创建上传Object的Metadata
        ObjectMetadata objectMetadata = new ObjectMetadata();
        objectMetadata.setContentLength(inputStream.available());
        objectMetadata.setCacheControl("no-cache");
        objectMetadata.setHeader("Pragma", "no-cache");
        objectMetadata.setContentType(getContentType(fileName.substring(fileName.lastInd
xOf(".")));
        objectMetadata.setContentDisposition("inline;filename=" + fileName);
        //上传文件
        PutObjectResult putResult = ossClient.putObject(aliOssProperties.getBucketName(), fil
Dir + fileName, inputStream, objectMetadata);
        ret = putResult.getETag();
        if (StringUtil.isEmpty(ret)) {
            log.error("上传失败，文件ETag为空");
        }
        ossClient.shutdown();
    } catch (IOException e) {
        log.error(e.getMessage(), e);
    } finally {
        try {
            inputStream.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}

/**
 * 请求类型
 *
 * @param filenameExtension :
 * @return :

```

```

*/
private static String getContentType(String filenameExtension) {
    if (FileNameSuffixEnum.BMP.getSuffix().equalsIgnoreCase(filenameExtension)) {
        return "image/bmp";
    }
    if (FileNameSuffixEnum.GIF.getSuffix().equalsIgnoreCase(filenameExtension)) {
        return "image/gif";
    }
    if (FileNameSuffixEnum.JPEG.getSuffix().equalsIgnoreCase(filenameExtension) ||
        FileNameSuffixEnum.JPG.getSuffix().equalsIgnoreCase(filenameExtension) ||
        FileNameSuffixEnum.PNG.getSuffix().equalsIgnoreCase(filenameExtension)) {
        return "image/jpeg";
    }
    if (FileNameSuffixEnum.HTML.getSuffix().equalsIgnoreCase(filenameExtension)) {
        return "text/html";
    }
    if (FileNameSuffixEnum.TXT.getSuffix().equalsIgnoreCase(filenameExtension)) {
        return "text/plain";
    }
    if (FileNameSuffixEnum.VSD.getSuffix().equalsIgnoreCase(filenameExtension)) {
        return "application/vnd.visio";
    }
    if (FileNameSuffixEnum.PPTX.getSuffix().equalsIgnoreCase(filenameExtension) ||
        FileNameSuffixEnum.PPT.getSuffix().equalsIgnoreCase(filenameExtension)) {
        return "application/vnd.ms-powerpoint";
    }
    if (FileNameSuffixEnum.DOCX.getSuffix().equalsIgnoreCase(filenameExtension) ||
        FileNameSuffixEnum.DOC.getSuffix().equalsIgnoreCase(filenameExtension)) {
        return "application/msword";
    }
    if (FileNameSuffixEnum.XML.getSuffix().equalsIgnoreCase(filenameExtension)) {
        return "text/xml";
    }
    if (FileNameSuffixEnum.PDF.getSuffix().equalsIgnoreCase(filenameExtension)) {
        return "application/pdf";
    }
    return "image/jpeg";
}
}

```

@Getter

```
enum FileNameSuffixEnum {
```

```

/**
 * 文件后缀名
 */
BMP(".bmp", "bmp文件"),
GIF(".gif", "gif文件"),
JPEG(".jpeg", "jpeg文件"),
JPG(".jpg", "jpg文件"),
PNG(".png", "png文件"),

```

```

HTML(".html", "HTML文件"),
TXT(".txt", "txt文件"),
VSD(".vsd", "vsd文件"),
PPTX(".pptx", "PPTX文件"),
DOCX(".docx", "DOCX文件"),
PPT(".ppt", "PPT文件"),
DOC(".doc", "DOC文件"),
XML(".xml", "XML文件"),
PDF(".pdf", "PDF文件");

/**
 * 后缀名
 */
private final String suffix;

/**
 * 描述
 */
private final String description;

FileNameSuffixEnum(String suffix, String description) {
    this.suffix = suffix;
    this.description = description;
}
}

```

## 5、重点：编写m3u8转码组件

```

package com.wslhome.demo.component;

import cn.hutool.core.io.FileUtil;

import com.wslhome.demo.config.FilePath;
import com.wslhome.demo.util.m3u8Util;
import lombok.SneakyThrows;
import lombok.extern.slf4j.Slf4j;
import net.bramp.ffmpeg.FFmpeg;
import net.bramp.ffmpeg.FFmpegExecutor;
import net.bramp.ffmpeg.FFprobe;
import net.bramp.ffmpeg.builder.FFmpegBuilder;
import net.bramp.ffmpeg.probe.FFmpegProbeResult;
import net.bramp.ffmpeg.probe.FFmpegStream;
import org.springframework.stereotype.Component;
import org.springframework.web.multipart.MultipartFile;

import javax.annotation.Resource;
import java.io.File;
import java.util.List;

```

```

import java.util.Optional;
import java.util.stream.Collectors;

/**
 * @Author sirwsl
 * @Version 1.0
 */
@Slf4j
@Component
public class M3u8Component {

    @Resource
    private FFmpeg ffmpeg;

    @Resource
    private FFprobe ffprobe;

    @Resource
    private FilePath filePath;

    /**
     * 视频文件转 m3u8
     * 支持: .mp4 | .flv | .avi | .mov | .wmv | .wav
     * @param file 视频文件
     * @return 路径
     */
    public String mediaFileToM3u8(MultipartFile file){
        if (file.isEmpty()) {
            throw new RuntimeException("未发现文件");
        }
        log.info("开始解析视频");
        long start = System.currentTimeMillis();
        //临时目录创建
        String path = new File(System.getProperty("user.dir")).getAbsolutePath();
        String tempFilePath = path+ filePath.getTempPath();
        if (!FileUtil.exist(tempFilePath)) {
            FileUtil.mkdir(tempFilePath);
        }
        String filePathName = tempFilePath + file.getOriginalFilename();
        File dest = new File(filePathName);
        try {
            file.transferTo(dest);
        }catch (Exception e){
            log.error("视频转m3u8格式存在异常, 异常原因e:{},e.getMessage());
        }
        long end = System.currentTimeMillis();
        log.info("临时文件上传成功.....耗时: {} ms", end - start);
        String m3u8FilePath = localFileToM3u8(filePathName);
        log.info("视频转换已完成！");
        return m3u8FilePath;
    }
}

```

```

/**
 * 本地媒体资源转换
 * @param filePathName : 文件路径
 * @return :
 */
@sneakyThrows
public String localFileToM3u8(String filePathName) {
    long startTime = System.currentTimeMillis();
    final FFmpegProbeResult probe = ffprobe.probe(filePathName);
    final List<FFmpegStream> streams = probe.getStreams().stream().filter(fFmpegStream ->
fFmpegStream.codec_type != null).collect(Collectors.toList());
    final Optional<FFmpegStream> audioStream = streams.stream().filter(fFmpegStream ->
FmpegStream.CodecType.AUDIO.equals(fFmpegStream.codec_type)).findFirst();
    final Optional<FFmpegStream> videoStream = streams.stream().filter(fFmpegStream ->
FmpegStream.CodecType.VIDEO.equals(fFmpegStream.codec_type)).findFirst();

    if (!audioStream.isPresent()) {
        log.error("未发现音频流");
    }
    if (!videoStream.isPresent()) {
        log.error("未发现视频流");
    }
    //m3u8文件 存储路径
    String filePath = m3u8Util.generateFilePath(this.filePath.getBasePath());
    if (!FileUtil.exist(filePath)) {
        FileUtil.mkdir(filePath);
    }
    String mainName = m3u8Util.getFileMainName(filePathName);
    String m3u8FileName = filePath + mainName + ".m3u8";

    //下面这一串参数别乱动，经过调优的，1G视频大概需要10秒左右，如果是大佬随意改
    //"-vsync", "2", "-c:v", "copy", "-c:a", "copy", "-tune", "fastdecode", "-hls_wrap", "0", "-hls_t
me", "10", "-hls_list_size", "0", "-threads", "12"
    FFmpegBuilder builder = new FFmpegBuilder()
        .setInput(filePathName)
        .overrideOutputFiles(true)
        .addOutput(m3u8FileName)//输出文件
        .setFormat(probe.getFormat().format_name) //"mp4"
        .setAudioBitRate(audioStream.map(fFmpegStream -> fFmpegStream.bit_rate).orElse
0L))
        .setAudioChannels(1)
        .setAudioCodec("aac") // using the aac codec
        .setAudioSampleRate(audioStream.get().sample_rate)
        .setAudioBitRate(audioStream.get().bit_rate)
        .setStrict(FFmpegBuilder.Strict.STRICT)
        .setFormat("hls")
        .setPreset("ultrafast")
        .addExtraArgs("-vsync", "2", "-c:v", "copy", "-c:a", "copy", "-tune", "fastdecode", "-hls
wrap", "0", "-hls_time", "10", "-hls_list_size", "0", "-threads", "12")
        .done();

    FFmpegExecutor executor = new FFmpegExecutor(ffmpeg, ffprobe);
    // Run a one-pass encode
    executor.createJob(builder).run();
}

```

```

        File dest = new File(filePathName);
        if (dest.isFile() && dest.exists()) {
            dest.delete();
            System.gc();
            log.warn("临时文件 {}已删除", dest.getName());
        }
        long endTime = System.currentTimeMillis();
        log.info("文件: {} 转换完成! 共耗时{} ms", dest.getName(), (endTime - startTime));
        return m3u8FileName;
    }
}

```

还有两个Util合并在下面了

m3u8Util.java

```

package com.wslhome.demo.util;

import cn.hutool.core.date.DateUtil;
import cn.hutool.core.io.FileUtil;
import cn.hutool.core.util.StrUtil;

import java.io.*;
import java.time.LocalDateTime;

/**
 * @Description 工具类
 * @Author sirwsl
 * @Version 1.0
 */
public class m3u8Util {

    /**
     * @Description 根据基础路径, 生成文件存储路径
     * @param basePath 基础路径 (根路径)
     * @Return
     */
    public static String generateFilePath(String basePath){
        String temp = basePath;
        if(StrUtil.isNotBlank(basePath)){
            if(basePath.endsWith("/")){
                temp = basePath.substring(0,basePath.lastIndexOf("/"));
            }
        }
        return temp+"/"+generateDateDir()+"/";
    }

    /**
     * @Description 根据当前时间, 生成下级存储目录
     * @Return
     */
    public static String generateDateDir(){

```

```

        LocalDateTime now = LocalDateTime.now();
        return DateUtil.format(now, "yyyyMMdd/HH/mm/ss");
    }

    /**
     * @Description 根据文件全路径, 获取文件主名称
     * @param fullPath 文件全路径 (包含文件名)
     * @Return
     */
    public static String getFileMainName(String fullPath){
        String fileName = FileUtil.getName(fullPath);
        return fileName.substring(0,fileName.lastIndexOf("."));
    }

}

```

## FileUtil.java

```

package com.wslhome.demo.util;

import java.io.*;

public class FileUtil {

    public static void deleteFiles(String path) {
        File file = new File(path);
        if (file.exists()) {
            if (file.isDirectory()) {
                File[] temp = file.listFiles(); //获取该文件夹下的所有文件
                for (File value : temp) {
                    deleteFile(value.getAbsolutePath());
                }
            } else {
                file.delete(); //删除子文件
            }
            file.delete(); //删除文件夹
        }
    }

    public static void deleteFile(String path){
        File dest = new File(path);
        if (dest.isFile() && dest.exists()) {
            dest.delete();
        }
    }

    public static void replaceTextContent(String path,String srcStr,String replaceStr) throws IOE
ception {
        // 读
        File file = new File(path);
        FileReader in = new FileReader(file);
        BufferedReader bufIn = new BufferedReader(in);
        // 内存流, 作为临时流

```

```

CharArrayWriter tempStream = new CharArrayWriter();
// 替换
String line = null;
while ( (line = bufIn.readLine()) != null) {
    // 替换每行中, 符合条件的字符串
    line = line.replaceAll(srcStr, replaceStr);
    // 将该行写入内存
    tempStream.write(line);
    // 添加换行符
    tempStream.append(System.getProperty("line.separator"));
}
// 关闭 输入流
bufIn.close();
// 将内存中的流 写入 文件
FileWriter out = new FileWriter(file);
tempStream.writeTo(out);
out.close();
System.out.println("===path:" + path);
}
}

```

## 6、重点Service编写

```

package com.wslhome.demo.service;

import cn.hutool.core.date.DateUtil;
import com.wslhome.demo.api.Result;
import com.wslhome.demo.component.M3u8Component;
import com.wslhome.demo.component.MinioComponent;
import com.wslhome.demo.component.OssComponent;
import com.wslhome.demo.config.AliOssProperties;
import com.wslhome.demo.config.FilePath;
import com.wslhome.demo.util.FileUtil;
import io.minio.ObjectStat;
import lombok.extern.slf4j.Slf4j;
import org.apache.commons.io.FileUtils;
import org.apache.commons.lang3.StringUtils;
import org.apache.tomcat.util.http.fileupload.servlet.ServletFileUpload;
import org.springframework.mock.web.MockMultipartFile;
import org.springframework.scheduling.concurrent.ThreadPoolTaskExecutor;
import org.springframework.stereotype.Component;
import org.springframework.util.CollectionUtils;
import org.springframework.web.multipart.MultipartFile;

import javax.annotation.Resource;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.*;
import java.net.URLEncoder;
import java.text.SimpleDateFormat;
import java.time.LocalDateTime;
import java.util.*;

```

```

import java.util.concurrent.CountDownLatch;

/**
 * 文件上传
 */
@Component
@Slf4j
public class FileService {

    @Resource
    private M3u8Component m3U8ComponentTemplate;

    @Resource
    private FilePath filePath;

    //@Resource(name = "minIOUploadTreadPool")
    @Resource(name = "ossUploadTreadPool")
    private ThreadPoolTaskExecutor poolTaskExecutor;

    @Resource
    private MinioComponent minioComponent;

    @Resource
    private AliOssProperties aliOssProperties;

    @Resource
    private OssComponent ossComponent;

    String projectUrl = System.getProperty("user.dir").replaceAll("\\\\", "/");

    /**
     * 视频上传并转m3u8，转存至oss或minIO
     *
     * @param file : 视频文件
     * @return 保存路径
     */
    public String uploadVideo2M3u8(MultipartFile file) throws Exception {
        String path = m3U8ComponentTemplate.mediaFileToM3u8(file);
        return upload2M3u8(path);
    }

    /**
     * 本地视频转m3u8后上传至OSS或minIO
     *
     * @param path
     * @return
     * @throws Exception
     */
    public String localVideo2M3u8(String path) throws Exception {
        String paths = m3U8ComponentTemplate.localFileToM3u8(path);
        return upload2M3u8(paths);
    }
}

```

```

/**
 * 上传转码后得视频至OSS或minIO
 * @param path
 * @return 路径
 * @throws Exception
 */
public String upload2M3u8(String path) throws Exception {
    //存储转码后文件
    String realPath = path.substring(0, path.lastIndexOf("/"));
    log.info("视频解析后的 realPath {}", realPath);
    String name = path.substring(path.lastIndexOf("/") + 1);
    log.info("解析后视频 name {}", name);
    File allFile = new File(realPath);
    File[] files = allFile.listFiles();
    if (null == files || files.length == 0) {
        return null;
    }
    String patch = DateUtil.format(LocalDateTime.now(), "yyyy/MM/") + name.substring(0, name.lastIndexOf(".")) + "/";
    List<File> errorFile = new ArrayList<>();

    long start = System.currentTimeMillis();
    //替换m3u8文件中的路径
    FileUtil.replaceTextContent(path, name.substring(0, name.lastIndexOf(".")),
        aliOssProperties.getHostUrl() + filePath.getProxy() + patch +
        name.substring(0, name.lastIndexOf(".")));
    //开始上传
    CountdownLatch countDownLatch = new CountdownLatch(files.length);
    Arrays.stream(files).forEach(li -> poolTaskExecutor.execute() -> {
        try (FileInputStream fileInputStream = new FileInputStream(li)) {
            //minioComponent.FileUploaderExist("m3u8", patch + li.getName(), fileInputStream)

            ossComponent.uploadFile(filePath.getProxy() + patch, fileInputStream, li.getName())

            log.info("文件: {} 正在上传", li.getName());
        } catch (Exception e) {
            errorFile.add(li);
            e.printStackTrace();
        } finally {
            countDownLatch.countDown();
        }
    });
    countDownLatch.await();
    long end = System.currentTimeMillis();
    log.info("解析文件上传成功,共计: {} 个文件,失败: {},共耗时: {}ms", files.length, errorFile.size(), end - start);
    // try {
    //     minioComponent.mkBucket("m3u8");
    // } catch (Exception e) {
    //     log.error("创建Bucket失败! ");
    // }

    //异步移除所有文件
    poolTaskExecutor.execute() -> {

```

```

        FileUtil.deleteFile(projectUrl+filePath.getTempPath());
    });
    if (CollectionUtils.isEmpty(errorFile)) {
        return aliOssProperties.getMyHostUrl() + filePath.getProxy() + patch + name;
    }
    return "";
}
/**
 * 普通上传文件转存至Oss或MinIo
 *
 * @param file : 文件
 * @return : 文件路径
 */
public String uploadFile(MultipartFile file) {
    //文件名字
    String fileName = file.getOriginalFilename();
    if (StringUtils.isBlank(fileName)) {
        return null;
    }
    String patch = "test/" + DateUtil.format(LocalDateTime.now(), "yyyy/MM/dd/");
    try {
        //String contentType = fileName.substring(fileName.lastIndexOf(".") + 1);
        //minioComponent.fileUploader(contentType, patch + fileName, file.getInputStream())

        ossComponent.uploadFile( patch, file.getInputStream(), fileName);
    } catch (Exception e) {
        log.error("文件上传失败");
        return "文件上传失败! ";
    } finally {
        try {
            file.getInputStream().close();
        } catch (Exception e) {
            log.error("关闭文件流异常, 异常原因e:{}", e.getMessage());
        }
    }
    return patch + fileName;
}

/**
 * 大文件上传至本地
 *
 * @param request : 请求
 * @param guid : 编码文件名
 * @param chunk : 切片数
 * @param file : 切片文件
 * @return : 是否成功
 */
public boolean uploadSlice(HttpServletRequest request, String guid, Integer chunk, MultipartFile file) {
    try {

        boolean isMultipart = ServletFileUpload.isMultipartContent(request);

```

```

    if (isMultipart) {
        if (chunk == null) chunk = 0;
        // 临时目录用来存放所有分片文件
        String tempFileDir = projectUrl + filePath.getBigPath() + guid;
        File parentFileDir = new File(tempFileDir);
        if (!parentFileDir.exists()) {
            parentFileDir.mkdirs();
        }
        // 分片处理时, 前台会多次调用上传接口, 每次都会上传文件的一部分到后台
        File tempPartFile = new File(parentFileDir, guid + "_" + chunk + ".part");
        FileUtils.copyInputStreamToFile(file.getInputStream(), tempPartFile);
    }
} catch (Exception e) {
    return false;
}
return true;
}

/**
 * 合并切片并上传至服务器
 * @param guid :
 * @param fileName :
 * @return :
 */
public String uploadMerge(String guid, String fileName){
    String localPath = mergeFile(guid, fileName);
    //此处需要注意, OSS需要再次切片上传, 但minIO是不用得, 它默认5M超过就会自动切片
    String path = "";
    if (StringUtils.isNotBlank(localPath)){
        try {
            path = ossComponent.uploadSlice(fileName, localPath,"file/bigfile/");
        } catch (Exception e){
            log.error("OSS切片上传失败! ");
        }
    }

    //移除文件
    poolTaskExecutor.execute() -> {
        FileUtil.deleteFile(projectUrl+filePath.getBigPath());
    };
    return path;
}

/**
 * 合并切片文件至本地
 *
 * @param guid    : 编码
 * @param fileName : 文件名
 * @return : 是否成功
 */
public String mergeFile(String guid, String fileName) {
    try {
        String sName = fileName.substring(fileName.lastIndexOf("."));
        //时间格式化格式

```

```

        Date currentTime = new Date();
        SimpleDateFormat simpleDateFormat = new SimpleDateFormat("yyyyMMddHHmmss
SS");
        //获取当前时间并作为时间戳
        String timeStamp = simpleDateFormat.format(currentTime);
        //拼接新的文件名
        String newName = timeStamp + sName;
        simpleDateFormat = new SimpleDateFormat("yyyyMM");
        String tempPath = projectUrl + filePath.getBigPath()+guid;
        String margePath = projectUrl + filePath.getBigPath()+simpleDateFormat.format(curr
ntTime);
        File parentFileDir = new File(tempPath);
        if (parentFileDir.isDirectory() {
            File destTempFile = new File(margePath, newName);
            if (!destTempFile.exists() {
                //先得到文件的上级目录, 并创建上级目录, 在创建文件
                destTempFile.getParentFile().mkdir();
                destTempFile.createNewFile();
            }
            for (int i = 0; i < Objects.requireNonNull(parentFileDir.listFiles()).length; i++) {
                File partFile = new File(parentFileDir, guid + "_" + i + ".part");
                FileOutputStream destTempfos = new FileOutputStream(destTempFile, true);
                //遍历"所有分片文件"到"最终文件"中
                FileUtils.copyFile(partFile, destTempfos);
                destTempfos.close();
            }
            // 删除临时目录中的分片文件
            FileUtils.deleteDirectory(parentFileDir);
            return destTempFile.getAbsolutePath();
        }
    } catch (Exception e) {
        log.error("切片文件合并, 失败原因: {}", e.getMessage());
    }
    return null;
}

/**
 * minIO文件下载
 *
 * @param response : 相应
 * @param bucket : bucket名称
 * @param fileName : 文件名
 * @throws Exception : 异常
 */
public void downloadFileMinIO(HttpServletRequest response, String bucket, String fileName)
throws Exception {
    ObjectStat objectStat = minioComponent.statObject(bucket, fileName);
    response.setContentType(objectStat.contentType());
    response.addHeader("Content-Disposition", "attachment;filename=" + URLEncoder.encode(fileName, "UTF-8"));
    response.addHeader("Content-Length", String.valueOf(objectStat.length()));
    byte[] bytes = minioComponent.fileDownloader(bucket, fileName);
    OutputStream outputStream = response.getOutputStream();

```

```

        outputStream.write(bytes);
        outputStream.close();
    }

    /**
     * Oss文件下载
     *
     * @param response: 相应
     * @param url      : 路径
     * @param fileName : 文件名
     * @throws Exception : 异常
     */
    public void downloadFileOss(HttpServletRequest response, String url, String fileName) throws Exception {
        response.setHeader("Content-Disposition", "attachment;filename=" + URLEncoder.encode(fileName, "UTF-8"));
        File file = ossComponent.getFile(url);
        OutputStream outputStream = response.getOutputStream();
        outputStream.write(new BufferedInputStream(new FileInputStream(file)).read());
        outputStream.close();
    }

    public String uploadVideoMerge(String guid, String fileName) {
        String localPath = mergeFile(guid, fileName);
        //此处需要注意, OSS需要再次切片上传, 但minIO是不用得, 它默认5M超过就会自动切片
        String path = "";
        try {
            path = localVideo2M3u8(localPath);
        } catch (Exception e) {
            log.error("OSS切片上传失败! ");
        }
        //移除文件
        poolTaskExecutor.execute() -> {
            String[] split = localPath.replaceAll("\\\\", "/").split("/");
            if (split.length >= 1) {
                FileUtil.deleteFile(split[1]);
            } else {
                FileUtil.deleteFile(split[0]);
            }
        });
        return path;
    }
}

```

## 7、写个Controller

```
package com.wslhome.demo.api;
```

```
import cn.hutool.core.util.StrUtil;
```

```
import com.wslhome.demo.service.FileService;
import lombok.extern.slf4j.Slf4j;
import org.apache.commons.io.FileUtils;
import org.apache.commons.lang3.StringUtils;
import org.springframework.web.bind.annotation.*;
import org.springframework.web.multipart.MultipartFile;
```

```
import javax.annotation.Resource;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
```

```
/**
 * @Author sirwsl
 * @Version 1.0
 */
@RestController
@RequestMapping("/")
@Slf4j
public class FileApi {
```

```
    @Resource
    private FileService fileService;
```

```
    /**
     * 测试文件上传
     * @param file : 上传文件
     * @return : 路径
     */
    @PostMapping("/uploadFile")
    public Result uploadImg(@RequestParam("file") MultipartFile file) {
        String path = fileService.uploadFile(file);
        if (StringUtils.isNotBlank(path)) {
            return Result.success("上传成功",path);
        }
        return Result.error("上传失败");
    }
```

```
    /**
     * 整个文件上传
     * 上传视频文件转m3u8
     * @param file 文件
     * @return String: 路径
     */
    @PostMapping("/uploadVideo")
    public Result uploadVideo(@RequestParam("file") MultipartFile file) { ;
        try {
            String path = fileService.uploadVideo2M3u8(file);
            if (StringUtils.isNotBlank(path)) {
```

```

        return Result.success("上传成功",path);
    }
}catch (Exception e){
    log.error("视频上传转码异常,异常原因e:{}",e.getMessage());
}
return Result.error("上传失败");
}

/**
 * 大文件上传至本地
 * @param request : 请求
 * @param guid : 编码
 * @param chunk : 切片数
 * @param file : 文件
 * @return : 返回结果
 */
@PostMapping("/uploadSlice")
public Result uploadSlice(HttpServletRequest request, @RequestParam("guid") String guid,
                        @RequestParam("chunk") Integer chunk,
                        @RequestParam("file") MultipartFile file) {
    if (fileService.uploadSlice(request, guid, chunk, file)){
        return Result.success("上传成功","");
    }else{
        return Result.error();
    }
}

/**
 * 大文件上传后合并
 * @param guid:
 * @param fileName :
 * @return :
 */
@RequestMapping("/uploadMerge")
public Result uploadMerge(@RequestParam("guid") String guid, @RequestParam("fileNam
") String fileName) {
    // 得到 destTempFile 就是最终的文件
    String path = fileService.uploadMerge(guid, fileName);
    if (StringUtils.isNotBlank(path)){
        return Result.success("合并成功",path);
    }else{
        return Result.error("合并文件失败");
    }
}

/**
 * 切片上传后合并转M3U8格式 :
 * @param fileName : 文件名
 * @param guid: 随机id
 * @return :
 */

```

```

    @PostMapping("/uploadVideoMerge")
    public Result uploadVideoMerge(@RequestParam("guid") String guid, @RequestParam("fileName") String fileName) {
        try {
            String path = fileService.uploadVideoMerge(guid, fileName);
            if (StringUtils.isNotBlank(path)) {
                return Result.success("上传成功",path);
            }
        } catch (Exception e){
            log.error("视频上传转码异常,异常原因e:{},e.getMessage());
        }
        return Result.error("上传失败");
    }

    /**
     * 下载文件
     *
     * @param response response
     * @param url bucket名称
     * @param fileName 文件名
     */
    @GetMapping("/downloadFile")
    public void downloadFile(HttpServletRequest response, String url, String fileName) throws Exception {
        fileService.downloadFileOss(response,url,fileName);
    }
}

```

## 八、前端代码

### 1、封装后的JS

前端为了万能兼容，以后管你啥语言都能够移植过去，因此就采用最古老的XMLHttpRequest请求方。这里做了封装，拿走不谢。

```

/*****
 * 普通文件上传
 * *****/
*/

function request(method,path,param,callback) {
    let XHR = null;
    if (window.XMLHttpRequest) {
        XHR = new XMLHttpRequest();
    } else if (window.ActiveXObject) {
        XHR = new ActiveXObject("Microsoft.XMLHTTP");
    } else {
        XHR = null;
    }
    if (XHR) {
        XHR.open(method, path);
    }
}

```

```

        XHR.onreadystatechange = function () {
            if (XHR.readyState == 4 && XHR.status == 200) {
                callback(XHR.responseText);
            }
        }
    }
    XHR.send(param);
}

/**
 * 切片上传
 * @returns {*}
 */
function partUpload(GUID, partFile, name, chunks, chunk, partUrl, partMethod, callback) {
    const form = new FormData();
    form.append("guid", GUID);
    form.append("file", partFile); //slice方法用于切出文件的一部分
    form.append("fileName", name);
    form.append("chunks", chunks); //总片数
    form.append("chunk", chunk); //当前是第几片
    return request(partMethod, partUrl, form, callback);
}

/**
 * 文件合并
 * @returns {*}
 */
function mergeFile(GUID, name, mergeUrl, partMethod, callback) {
    const formMerge = new FormData();
    formMerge.append("guid", GUID);
    formMerge.append("fileName", name);
    return request(partMethod, mergeUrl, formMerge, callback);
}

/**
 * 生成id
 * @returns {string}
 */
function guid(prefix) {
    let counter = 0;
    let guid = (+new Date()).toString(32),
        i = 0;
    for (; i < 5; i++) {
        guid += Math.floor(Math.random() * 65535).toString(32);
    }
    return (prefix || 'sirwsl_') + guid + (counter++).toString(32);
}

```

## 2、懒的人直接copy这html把

```
<!DOCTYPE html>
```

```
<html lang="zh-CN" >
<head>

  <meta charset="UTF-8" >
  <title> </title>
  <link href="https://unpkg.com/video.js/dist/video-js.css" th:href="@{https://unpkg.com/video.js/dist/video-js.css}" rel="stylesheet" >
  <script src="https://unpkg.com/video.js/dist/video.js" th:src="@{https://unpkg.com/video.js/dist/video.js}" ></script>
  <script src="https://unpkg.com/videojs-contrib-hls/dist/videojs-contrib-hls.js" th:src="@{https://unpkg.com/videojs-contrib-hls/dist/videojs-contrib-hls.js}" ></script>
  <meta content="text/html; charset=utf-8" http-equiv="Content-Type"/>
  <style type="text/css" >
    #red {
      width: 350px;
      height: 350px;
      background: #fedcbd;
      float: left;
    }

    #blue {
      width: 350px;
      height: 350px;
      background: #cde6c7;
      float: left;
    }

    #green {
      width: 350px;
      height: 350px;
      background: #d5c59f;
      float: left;
      clear: left;
    }

    #grey {
      width: 350px;
      height: 350px;
      background: #fffef9;
      float: left;
    }

    .video-js .vjs-tech {
      position: relative !important;
    }

    .all {
      width: 350px;
      height: 50px;
    }

  </style>
</head>
<body>
```

```

<div style=" position: absolute;left: 50%;top: 50%;transform: translate(-50%,-50%);" >
  <div id="red">
    <div style="width:350px;height:100px;text-align:center"> <h4>普通文件上传至OSS或Mini
n</h4> </div>
    <input id = "uploadFile" class="all" type="file" name="uploadFile"/> </br>
    <input type="submit" value="上传" onclick="upload()"/>
    <div id="detail1" style="text-align:center"> </div>
  </div>
  <div id="blue">
    <div style="width:350px;height:100px;text-align:center"> <h4>大文件切片上传至OSS或M
NIO</h4> </div>
    <input id = "uploadBigFile" class="all" type="file" name="uploadBigFile"/> </br>
    <input type="submit" value="大文件切片上传" onclick="uploadBigFile()"/>
    <div id="detail2" style="text-align:center"> </div>
  </div>

  <div id="green">
    <div style="width:350px;height:100px;text-align:center"> <h4>视频文件上传转m3u8格式<
br>并保存至OSS或minIO</h4> </div>
    <input id = "uploadVideo" class="all" type="file" name="uploadFile"/> </br>
    <input type="submit" value="普通上传并转码" onclick="uploadVideo()"/>
    <input type="submit" value="切片上传并转码" onclick="uploadVideoSlice()"/>
    <div id="detail3" style=" width:300px;word-break:break-all; text-align:center"> </div>

  </div>

  <div id="grey">
    <div style="width:350px;height:350px;position:absolute;top:50%;left:50%;" >
      <video id="myVideo"
        style="width:350px;height:350px;"
        class="video-js vjs-default-skin vjs-big-play-centered" controls preload="auto" da
a-setup='{}' >
        <source id="source" src="/test.m3u8" type="application/x-mpegURL"> </source>
      </video>
    </div>
  </div>
</div>
</body>

<script src="/index.js" th:src="@{/index.js}"> </script>
<script type="text/javascript">

function upload(){
  let file = document.getElementById("uploadFile").files[0];//IE10以下不支持
  let fd = new FormData();
  fd.append('file', file);
  document.getElementById("detail1").innerHTML = '文件正在上传... ..';
  let call = function (result) {
    let res = JSON.parse(result);
    if (res.code == 200&&res.data !=" ") {
      console.log(res);
      document.getElementById("detail1").innerHTML = '文件上传成功!</br>路径为: '+
es.data;
    }
  }
  call(uploadFile);
}

```

```

    }
}
request("POST", "/uploadFile", fd, call);

}
function uploadBigFile(){
    let file = document.getElementById("uploadBigFile").files[0]; //IE10以下不支持
    let name = file.name, //文件名
        size = file.size; //总大小
    let GUID = guid();
    let shardSize = 2 * 1024 * 1024, //以1MB为一个分片
        shardCount = Math.ceil(size / shardSize); //总片数
    let count = 0;
    for (let i = 0; i < shardCount; ++i) {
        //计算每一片的起始与结束位置
        let start = i * shardSize,
            end = Math.min(size, start + shardSize);
        let partFile = file.slice(start, end);
        let call = function (result) {
            let res = JSON.parse(result);
            if (res.code == 200){
                document.getElementById("detail2").innerHTML = '此次文件共切片'+shardCount+
片</br>切片上传进度: '+ (i+1) + "/" + shardCount;
                count++;
            }

            if (count == shardCount){
                let call2 = function (result) {
                    let res = JSON.parse(result);
                    if (res.code == 200){
                        document.getElementById("detail2").innerHTML = "切片上传成功! 地址: "+r
s.data;
                    }
                }
                document.getElementById("detail2").innerHTML = "正在进行文件合并并上传至文
服务";
                mergeFile(GUID,name,"/uploadMerge","POST",call2);
            }
        }
        partUpload(GUID, partFile, name, shardCount, i, "/uploadSlice", "POST",call);
    }

}

function uploadVideoSlice(){
    let file = document.getElementById("uploadVideo").files[0]; //IE10以下不支持
    let name = file.name, //文件名
        size = file.size; //总大小
    let GUID = this.guid();
    let shardSize = 5 * 1024 * 1024, //以1MB为一个分片
        shardCount = Math.ceil(size / shardSize); //总片数
    let count = 0;

```

```

for (let i = 0; i < shardCount; ++i) {
  //计算每一片的起始与结束位置
  let start = i * shardSize,
      end = Math.min(size, start + shardSize);
  let partFile = file.slice(start, end);
  let call1 = function (result){
    let res = JSON.parse(result);
    if (res.code == 200){
      document.getElementById("detail3").innerHTML = '此次文件共切片'+shardCount+
切片上传进度: '+(i+1)+"/"+shardCount;
      count++;
    }
    if (count == shardCount){
      document.getElementById("detail3").innerHTML = "正在进行视频合并、视频转码
转存、请耐心等待...如果觉得看不到进度, 可以自己用Socket实现";
      let call2 = function (result) {
        let res = JSON.parse(result);
        if (res.code == 200){
          document.getElementById("detail3").innerHTML = "视频转码成功, 且已存储
地址: "+res.data;
          changeVideo(res.data);
        }
      }
      mergeFile(GUID,name,"/uploadVideoMerge","POST",call2);
    }
  }
  partUpload(GUID, partFile, name, shardCount, i, "/uploadSlice", "POST",call1);
}
}

function uploadVideo(){
  let file = document.getElementById("uploadVideo").files[0];//IE10以下不支持
  let fd = new FormData();
  fd.append('file', file);
  document.getElementById("detail3").innerHTML = '视频上传中...';
  let call = function (result) {
    let res = JSON.parse(result)
    if (res.code == 200&&res.data != ''){
      document.getElementById("detail3").innerHTML = '文件上传成功正在加载,解析后路
为路径为: '+res.data;
      changeVideo(res.data);
    }
  }
  request("POST","/uploadVideo",fd,call);
}

/**
 * m3u8 播放器
 */
let myVideo = videojs('myVideo', {
  bigPlayButton: true,
  textTrackDisplay: false,
  posterImage: false,

```

```
    errorDisplay: false,
  })
  myVideo.play()

  let changeVideo = function (vdoSrc) {
    if (/\.m3u8$/\.test(vdoSrc)) {
      myVideo.src({
        src: vdoSrc,
        type: 'application/x-mpegURL'
      })
    } else {
      myVideo.src(vdoSrc)
    }
    myVideo.load();
    myVideo.play();
  }

</script>
</html>
```

## 九、收尾

截至目前，项目正常来说应该可以在windows上跑起来了，但是不能够在linux上跑更不能docker ru，因此我们来写个dockerfile

### 1、dockerfile编写

```
#指定基础镜像，在其上进行定制
FROM java:8
```

```
#维护者信息
MAINTAINER wangshilei <sirwsl@163.com>
VOLUME /tmp
```

```
COPY target/uploadWheel-1.jar upload-wheel.jar
COPY ffmpeg-linux/ffmpeg /ffmpeg
RUN bash -c "touch /upload-wheel.jar" &&\
  cp /usr/share/zoneinfo/Asia/Shanghai /etc/localtime &&\
  echo "Asia/Shanghai" >> /etc/timezone
EXPOSE 8080
```

```
ENTRYPOINT [ "java", "-Djava.security.egd=file:/dev/./urandom", "-jar", "/upload-wheel.jar" ]
```

### 2、编写nginx配置文件

```
location /m3u8 {
  types{
    application/vnd.apple.mpegurl m3u8;
    video/mp2t ts;
  }
}
```

```

}
proxy_pass xxx;
access_log off;
add_header Cache-Control no-cache;
}

```

如果涉及到跨域请加入

```

add_header Access-Control-Allow-Origin *;
add_header Access-Control-Allow-Methods GET,POST,PUT,DELETE,OPTIONS always;
add_header Access-Control-Allow-Credentials true always;
add_header Access-Control-Allow-Headers DNT,X-CustomHeader,Keep-Alive,User-Agent
X-Requested-With,If-Modified-Since,Cache-Control,Content-Type,Authorization,x-auth-token
always;

```

## 十、优化

在调用ffmpeg进行转码时候，上面已经优化过，1G视频转码大概需要10秒，在没优化之前大概需要5多分钟，如果觉得慢自己去优化。

优化前：

```

2022-01-09 23:35:42.036 INFO 5916 --- [nio-8080-exec-5] com.wslhome.demo.config.M3u8Converter : 开始解析视频
2022-01-09 23:35:47.947 INFO 5916 --- [nio-8080-exec-5] com.wslhome.demo.config.M3u8Converter : 临时文件上传成功.....耗时: 5911 ms
2022-01-09 23:35:47.947 INFO 5916 --- [nio-8080-exec-5] net.bramp.ffmpeg.RunProcessFunction : E:\minioDemo\target\classes\ffmpeg\bin\ffprobe.exe -version
2022-01-09 23:35:48.427 INFO 5916 --- [nio-8080-exec-5] net.bramp.ffmpeg.RunProcessFunction : E:\minioDemo\target\classes\ffmpeg\bin\ffprobe.exe -v quiet -print_format json -show_error
2022-01-09 23:35:48.673 INFO 5916 --- [nio-8080-exec-5] net.bramp.ffmpeg.RunProcessFunction : E:\minioDemo\target\classes\ffmpeg\bin\ffmpeg.exe -y -v error -i D:/video/tmp/测试视频.mp4
2022-01-10 00:42:13.552 WARN 5916 --- [nio-8080-exec-5] com.wslhome.demo.config.M3u8Converter : 临时文件测试视频.mp4已删除
2022-01-10 00:42:13.554 INFO 5916 --- [nio-8080-exec-5] com.wslhome.demo.config.M3u8Converter : 文件: 测试视频.mp4 转换完成! 共耗时5985687 ms
2022-01-10 00:42:13.555 INFO 5916 --- [nio-8080-exec-5] com.wslhome.demo.config.M3u8Converter : 视频转换已完成!

```

优化后：

```

2022-01-10 14:30:52.292 INFO 10180 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet : Completed initialization in 7 ms
2022-01-10 14:31:23.030 INFO 10180 --- [nio-8080-exec-1] com.wslhome.demo.config.M3u8Converter : 开始解析视频
2022-01-10 14:31:29.666 INFO 10180 --- [nio-8080-exec-1] com.wslhome.demo.config.M3u8Converter : 临时文件上传成功.....耗时: 6636 ms
2022-01-10 14:31:29.666 INFO 10180 --- [nio-8080-exec-1] net.bramp.ffmpeg.RunProcessFunction : E:\minioDemo\target\classes\ffmpeg\bin\ffprobe.exe -version
2022-01-10 14:31:29.781 INFO 10180 --- [nio-8080-exec-1] net.bramp.ffmpeg.RunProcessFunction : E:\minioDemo\target\classes\ffmpeg\bin\ffprobe.exe -v quiet -print_format json -show_error
-show_format -show_streams D:/video/tmp/测试视频.mp4
2022-01-10 14:31:30.537 INFO 10180 --- [nio-8080-exec-1] net.bramp.ffmpeg.RunProcessFunction : E:\minioDemo\target\classes\ffmpeg\bin\ffmpeg.exe -y -v error -i D:/video/tmp/测试视频.mp4
-strict strict -f hls -preset ultrafast -acodec aac -ac 1 -ar 44100 -b:a 128293 -vsync 2 -c:v copy -c:a copy -tune fastdecode -hls_wrap 0 -hls_time 10 -hls_list_size 0 -threads 12
D:/video/tmp/base/20220110/14/31/测试视频.m3u8
2022-01-10 14:31:42.867 WARN 10180 --- [nio-8080-exec-1] com.wslhome.demo.config.M3u8Converter : 临时文件测试视频.mp4已删除
2022-01-10 14:31:42.867 INFO 10180 --- [nio-8080-exec-1] com.wslhome.demo.config.M3u8Converter : 文件: 测试视频.mp4 转换完成! 共耗时13201 ms
2022-01-10 14:31:42.867 INFO 10180 --- [nio-8080-exec-1] com.wslhome.demo.config.M3u8Converter : 视频转换已完成!

```

优化参数：

```

"-vsync", "2",
"-c:v", "copy",
"-c:a", "copy",
"-tune", "fastdecode",
"-hls_wrap", "0",
"-hls_time", "10",
"-hls_list_size", "0",
"-threads", "12"

```

## 十一、项目缺陷

- 1、文件在下载时候，没办法下载视频文件，因为视频都被切片了，因为我们项目不需要，所以就没写
- 2、多线程上传没有优化、需要自己根据系统情况进行优化
- 3、不管是linux还是window打成jar包要运行，需注意jar包位置与ffmpeg文件的位置，不然运行不了
- 4、因为开发和部署方便，所以该项目只是做了在idea、和docker中能够平稳运行

## 十二、项目地址：

码云：<https://gitee.com/sirwsl/uploadWheel>

GitHub：<https://github.com/sirwsl/uploadWheel>