学成在线 **3** 义 线学习 **HLS**

# 线学习需求分析

## 求描述

成在线作为在线教育网站，提供多种学习形式，包括：录播、直播、图文、社群等，学生登录进入学习中心即可 线学习，本章节将开发录播课程的在线学习功能，需求如下：

学生可以在 indows 览器上在线观看视频。播放器具有快进、快退、暂停等基本功能。学生可以方便切换章节进行学习。

么是录播课程？

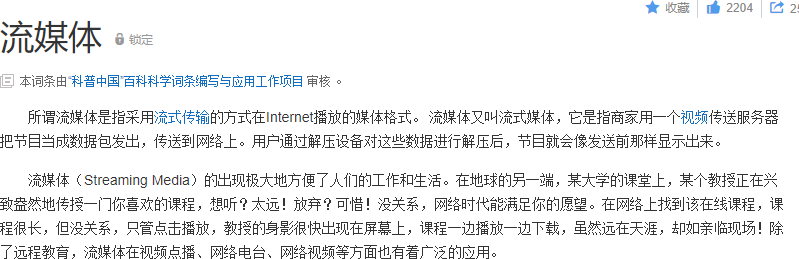
播课程就是提供录制好课程视频，供用户在线点播，反复学习。程视频如何管理？

资管理系统专门来管理课程视频，用户视频文件上传到媒资系统，并对视频进行编码处理。

## 频点播解决方案

### 媒体

流媒体：



细参考： [ttps://baike.baidu.com/item/%E6%B5%81%E5%AA%92%E4%BD%93/98740?fr=aladdin](https://baike.baidu.com/item/%E6%B5%81%E5%AA%92%E4%BD%93/98740?fr=aladdin)

概括理解：流媒体就是将视频文件分成许多小块儿，将这些小块儿作为数据包通过网络发送出去，实现一边传输视 据 一边观看视频。

流式传输

网络上传输音、视频信息有两个方式：下载和流式传输。

载：就是把音、视频文件完全下载到本机后开始播放，它的特点是必须等到视频文件下载完成方可播放， 放等待时间较长，无法去播放还未下载的部分视频。

式传输：就是客户端通过链接视频服务器实时传输音、视频信息，实现 下载边播放式传输包括如下两种方式：

序流式传输

顺序下载音、视频文件，可以实现边下载边播放，不过，用户只能观看已下载的视频内容，无法快进到未 载的视频部分，顺序流式传输可以使用 ttp 务器来实现，比如 ginx pache 。

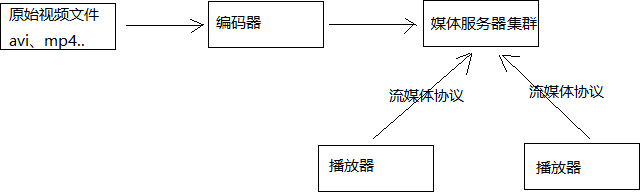
实时流式传输

时流式传输可以解决顺序流式传输无法快进的问题，它与 ttp 式传输不同，它必须使用流媒体服务器并使用流媒体协议来传输视频，它比 ttp 式传输复杂。常见的实时流式传输协议有 TSP TMP SVP

。

流媒体系统的概要结构

过流媒体系统的概要结构学习流媒体系统的基本业务流程。



将原始的视频文件通过编码器转换为适合网络传输的流格式，编码后的视频直接输送给媒体服务器。 始的视频文件通常是事先录制好的视频，比如通过摄像机、摄像头等录像、录音设备采集到的音视频文

，体积较大，要想在网络上传输需要经过压缩处理，即通过编码器进行编码

媒体服务获取到编码好的视频文件，对外提供流媒体数据传输接口，接口协议包括 TTP TSP TMP

播放器通过流媒体协议与媒体服务器通信，获取视频数据，播放视频。

### 播方案

项目包括点播和直播两种方式，我们先调研点播的方案，如下： 放器通过 http 议从 ttp 务器上下载视频文件进行播放

题：必须等到视频下载完才可以播放，不支持快进到某个时间点进行播放

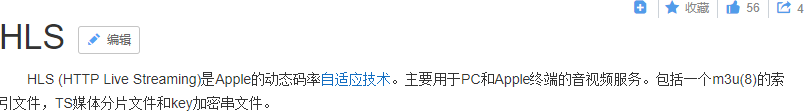
放器通过 tmp 议连接媒体服务器以实时流方式播放视频

用 tmp 议需要架设媒体服务器，造价高，对于直播多采用此方案。

放器使用 LS 议连接 ttp 务器（ ginx pache ）实现近实时流方式播放视频

LS 议规定：基于 ttp 议，视频封装格式为 s 视频的编码格式为 264, 频编码格式为 P3 AC 者 C-

LS 什 么 ？

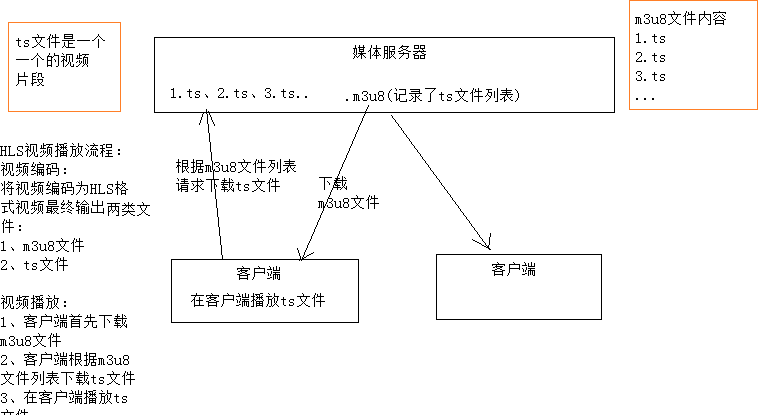


LS 工作方式是：将视频拆分成若干 s 式的小文件，通过 3u8 式的索引文件对这些 s 文件建立索引。一般

0 一个 s 件，播放器连接 3u8 件播放，当快进时通过 3u8 可找到对应的索引文件，并去下载对应的 s

，从而实现快进、快退以近实时 方式播放视频。

OS ndroid 备、及各大浏览器都支持 LS 议。

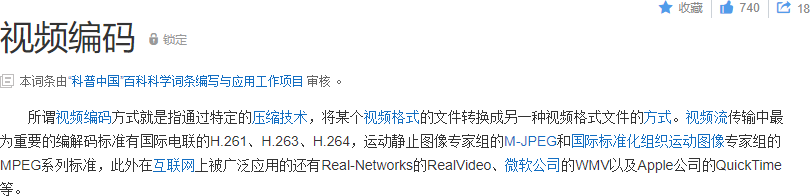


细参考： [ttps://baike.baidu.com/item/HLS/8328931?fr=aladdin](https://baike.baidu.com/item/HLS/8328931?fr=aladdin)

采用 LS 案即可实现边下载边播放，并可不用使用 tmp 流媒体协议，不用构建专用的媒体服务器，节省成本。项目点播方案确定为方案

# 频编码

## 频编码格式



情参考  [ttps://baike.baidu.com/item/%E8%A7%86%E9%A2%91%E7%BC%96%E7%A0%81/839038](https://baike.baidu.com/item/%E8%A7%86%E9%A2%91%E7%BC%96%E7%A0%81/839038)

首先我们要分清文件格式和编码格式：

件格式：是指 mp4 avi rmvb 些不同扩展名的视频文件的文件格式 视频文件的内容主要包括视频和音

，其文件格式是按照一 的编码格式去编码，并且按照该文件所规定的封装格式将视频、音频、字幕等信息封装 一起，播放器会根据它们的封装格式去提取出编码，然后由播放器解码，最终播放音视频。

音视频编码格式：通过音视频的压缩技术，将视频格式转换成另一种视频格式，通过视频编码实现流媒体的传输。 如：一个 avi 视频文件原来的编码是 通过编码后编码格式变为 音频原来为 通过编码后变为

视频编码格式各类繁多，主要有几下几类：

PEG 列 由 SO[ 际标准组织机构 属的 PEG[ 动图象专家组 发 视频编码方面主要是 peg1 cd

就是它）、 peg2 VD 用）、 peg4 的 VDRIP 用的都是它的变种，如： ivx vid ） 、 peg4

VC 正热门）；音频编码方面主要是 PEG Audio Layer 1/2 PEG Audio Layer 3 大名鼎鼎的 p3 、

PEG-2 AAC PEG-4 AAC 等。注意： VD 频没有采用 peg 。

|  |  |  |  |
| --- | --- | --- | --- |
| .26X | 列 由 TU[ 际电传视讯联盟 导，侧重网络传输，注意：只是视频编码） | 括 .261 | .262 |
| .263 | .263+ .263++ .264 就是 PEG4 AVC- 作的结晶） |  |  |

前最常用的编码标准是视频 .264 音频 AC

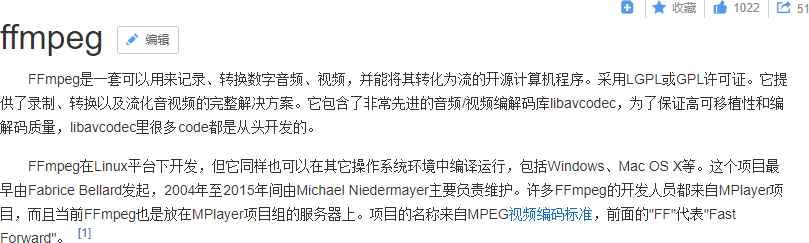
问：

.264 编码格式还是文件格式？

p4 编码格式还是文件格式？

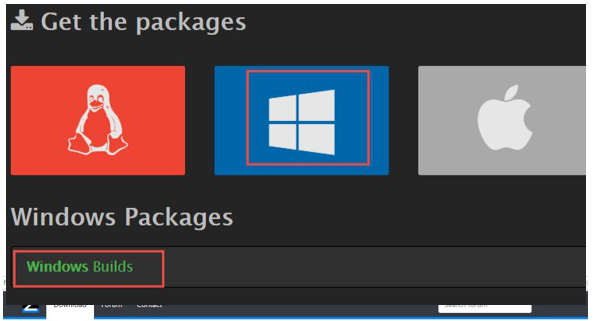
* 1. **FFmpeg** 基本使用

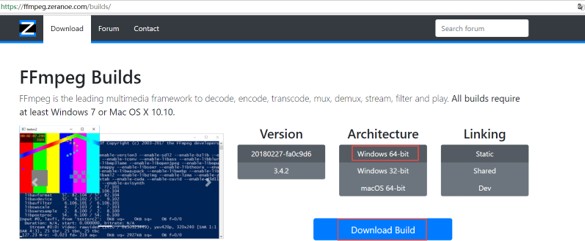
们将视频录制完成后，使用视频编码软件对视频进行编码，本项目 用 Fmpeg 视频进行编码



Fmpeg 许多开源项目采用， Q 音、暴风影音、 LC 。

载 ： Fmpeg [ttps://www.ﬀmpeg.org/download.html#build-windows](https://www.ffmpeg.org/download.html#build-windows)





载 mpeg-20180227-fa0c9d6-win64-static.zip 并解压，本教程将 mpeg 压到了

:\devenv\edusoft\ﬀmpeg-20180227-fa0c9d6-win64-static\ﬀmpeg-20180227-fa0c9d6-win64-static 。

:\devenv\edusoft\ﬀmpeg-20180227-fa0c9d6-win64-static\ﬀmpeg-20180227-fa0c9d6-win64-static\bin

配置在 ath 境变量中。

测是否安装成功：



单的测试：

一个 avi 件转成 p4 p3 if 。

如我们将 ucene.avi 件转成 p4 运行如下命令：

mpeg -i lucene.avi lucene.mp4

成 p3 mpeg -i lucene.avi lucene.mp3

成 if mpeg -i lucene.avi lucene.gif

方文档（英文）： [ttp://ﬀmpeg.org/ﬀmpeg.html](http://ffmpeg.org/ffmpeg.html)

**2.2** 成 **3u8/ts** 件

用 mpeg 成 m3u8 步骤如下： 一步：先将 vi 频转成 p4

fmpeg.exe ‐i lucene.avi ‐c:v libx264 ‐s 1280x720 ‐pix\_fmt yuv420p ‐b:a 63k ‐b:v 753k ‐r 18

\lucene.mp4

面把各参数意思大概讲讲，大概了解意思即可，不再此展开流媒体专业知识的讲解。

c:v 频编码为 264 264 码是 264 一种开源编码格式。

s 置分辨率

pix\_fmt yuv420p 设置像素采样方式，主流的采样方式有三种， UV4:4:4 UV4:2:2 UV4:2:0 它的作用是据采样方式来从码流中还原每个像素点的 UV 亮度信息与色彩信息）值。

b 置码率， b:a b:v 别表示音频的码率和视频的码率， b 示音频加视频的总码率。码率对一个视频质量有大的作用，后边会介绍。

r 帧率，表示每秒更新图像画面的次数，通常大于 4 眼就没有连贯与停顿的感觉了。二 步 ： 将 p4 成 3u8

fmpeg ‐i lucene.mp4 ‐hls\_time 10 ‐hls\_list\_size 0 ‐hls\_segment\_filename

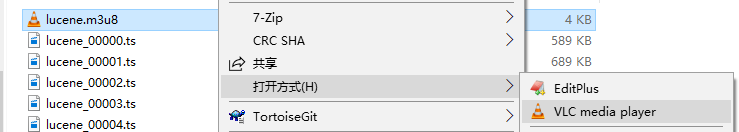
/hls/lucene\_%05d.ts ./hls/lucene.m3u8

hls\_time 置每片的长度，单位为秒

hls\_list\_size n: 存的分片的数量，设置为 示保存所有分片

hls\_segment\_ﬁlename 段文件的名称， 05d 示 数字

成的效果是：将 ucene.mp4 频文件每 0 生成一个 s 件，最后生成一个 3u8 件， 3u8 件是 s 索引件。

用 LC 开 3u8 件，测试播放效果， LC 一款自由、开源的跨平台多媒体播放器及框架，可播放大多数多体文件，以及 DVD 音频 CD CD 各类流媒体协议。（ [ttp://www.videolan.org/](http://www.videolan.org/)）

**.2.1** 率的设置

率又叫比特率即每秒传输的 it ，单位为 ps(Bit Per Second) 码率越大传送数据的速度越快。率的计算公式是：文件大小（转成 it 长（秒） 1024 = kbps 每秒传输千位数

如一个 M 视频，它的时长是 0s 它的码率等于

\*1024\*1024\*8/10/1024 = 819Kbps

率设置到多少才能达到最好，通过根据个人的经验或参考一些视频网台给出的参考，下图是优酷对码率的要求：



果要将视频上传到优酷则必须按照上面的要求，如果是自己搭建视频服务器，码率设置不易过大，最终达到的视 清晰度满足业务需求即可。

# 放器

## 术选型

频编码后要使用播放器对其进行解码、播放视频内容。在 eb 用中常用的播放器有 ash 放器、 5 放器或览器插件播放器，其中以 ash 5 放器最常见。

ash 放器：缺点是需要在客户机安装 dobe Flash Player 放器，优点是 ash 放器已经很成熟了，并且浏览对 ash 持也很好。

5 放器：基于 5 带 ideo 签进行构建，优点是大部分浏览器支持 5 不用再安装第三方的 ash 放器，并随着前端技术的发展， 5 术会越来越成熟。

项目采用 5 放器，使用 ideo.js 源播放器。

Video.js 一款基于 TML5 界的网络视频播放器。它支持 TML5 lash 频，它支持在台式机和移动设备上播视频。这个项目于 010 中开始，目前已在 0 网站使用。

方地址： [ttp://videojs.com/](http://videojs.com/)

* 1. 载 **ideo.js**

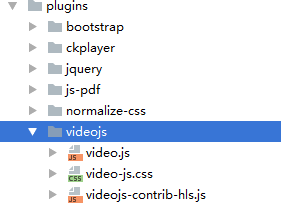
Video.js  [ttps://github.com/videojs/video.js](https://github.com/videojs/video.js)

videojs-contrib-hls  [ttps://github.com/videojs/videojs-contrib-hls#installation](https://github.com/videojs/videojs-contrib-hls#installation)

（ ideojs-contrib-hls 播放 ls 一个插件）

用文档： [ttp://docs.videojs.com/tutorial-videojs\_.html](http://docs.videojs.com/tutorial-videojs_.html)

本教程使用 video.js 6.7.3 本 ， ideojs-contrib-hls .14.1 本 。载上边两个文件，为了测试需求将其放在门户的 lugins 录中。



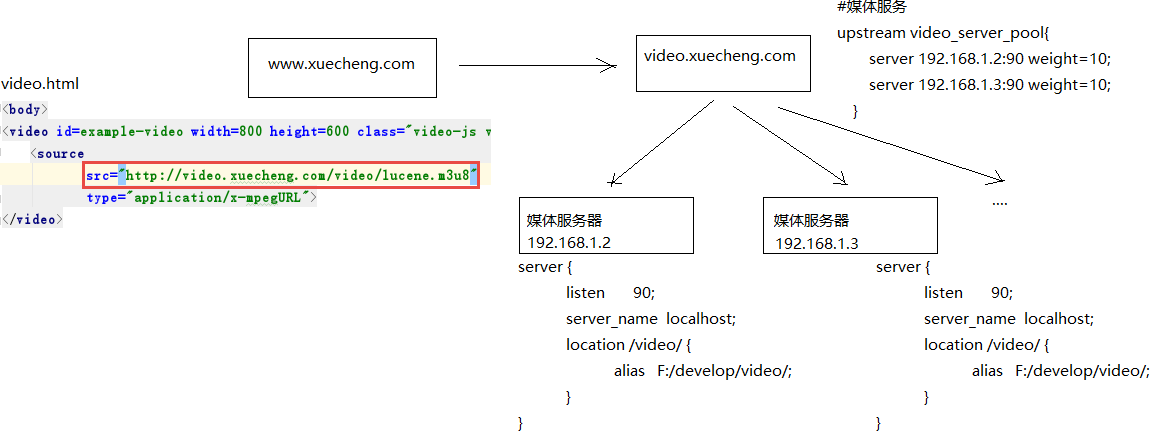
## 建媒体服务器

常使用 ideo.js 放视频是通过一个网页，用户通过浏览器打开网页去播放视频，网页和视频都从 eb 务器请

，通常视频的 rl 址使用单独的域名。

* 1. **Nginx** 体服务器

LS 议基于 ttp 议，本项目使用 ginx 为视频服务器。下图是 ginx 体服务器的配置流程图：



用户打开 [ww.xuecheng.com](NULL)上边的 ideo.html 页

在此网页中引入视频链接，视频地址指向 ideo.xuecheng.com

ideo.xuecheng.com 行负载均衡处理，将视频请求转发到媒体服务器据上边的流程，我们在媒体服务器上安装 ginx 并配置如下：

成网媒体服务

erver {

isten 90; erver\_name localhost;

频目录

ocation /video/ {

lias F:/develop/video/;

### 体服务器代理

体服务器不止一台，通过代理实现负载均衡功能，使用 ginx 为媒体服务器的代理，此代理服务器作为

ideo.xuecheng.com 名服务器。

置 ideo.xuecheng.com 拟主机：

意：开发中代理服务器和媒体服务器在同一台服务器，使用同一个 ginx

成网媒体服务代理

ap $http\_origin $origin\_list{ default http://www.xuecheng.com;

"~[http://www.xuecheng.com](http://www.xuecheng.com/)" http://www.xuecheng.com; "~[http://ucenter.xuecheng.com](http://ucenter.xuecheng.com/)" http://ucenter.xuecheng.com;

成网媒体服务代理

erver {

isten 80;

erver\_name video.xuecheng.com;

ocation /video {

roxy\_pass http://video\_server\_pool;

dd\_header Access‐Control‐Allow‐Origin $origin\_list; add\_header Access‐Control‐Allow‐Origin \*;

dd\_header Access‐Control‐Allow‐Credentials true; dd\_header Access‐Control‐Allow‐Methods GET;

ors 域参数：

ccess-Control-Allow-Origin 允许跨域访问的外域地址

常允许跨域访问的站点不是一个，所以这里用 ap 义了多个站点。果允许任何站点跨域访问则设置为 通常这是不建议的。

ccess-Control-Allow-Credentials 许客户端携带证书访问

ccess-Control-Allow-Methods 允许客户端跨域访问的方法

ideo\_server\_pool 配置如下：

# 体 服 务

upstream video\_server\_pool{

erver 127.0.0.1:90 weight=10;

}

* 1. 试 **ideo.js**

考 [ttps://github.com/videojs/videojs-contrib-hls#installation](https://github.com/videojs/videojs-contrib-hls#installation)

[ttp://jsbin.com/vokipos/8/edit?html](http://jsbin.com/vokipos/8/edit?html),output

编写测试页面 ideo.html

!DOCTYPE html> html lang="en"> head>

<meta http‐equiv="content‐type" content="text/html; charset=utf‐8" />

<title> 频 播 放 /title>

<link href="/plugins/videojs/video‐js.css" rel="stylesheet">

/head> body>

video id=example‐video width=800 height=600 class="video‐js vjs‐default‐skin vjs‐big‐play‐ entered" controls poster="http://127.0.0.1:90/video/add.jpg">

<source

src="<http://video.xuecheng.com/video/hls/lucene.m3u8>" type="application/x‐mpegURL">

/video>

input type="button" onClick="switchvideo()" value="switch"/>

script src="/plugins/videojs/video.js"></script>

script src="/plugins/videojs/videojs‐contrib‐hls.js"></script> script>

var player = videojs('example‐video');

//player.play();

/ 换 视 频

function switchvideo(){ player.src({

src: '[http://video.xuecheng.com/video/hls/lucene.m3u8',](http://video.xuecheng.com/video/hls/lucene.m3u8%27) type: 'application/x‐mpegURL',

withCredentials: true



});

player.play();

}

/script>

/body>

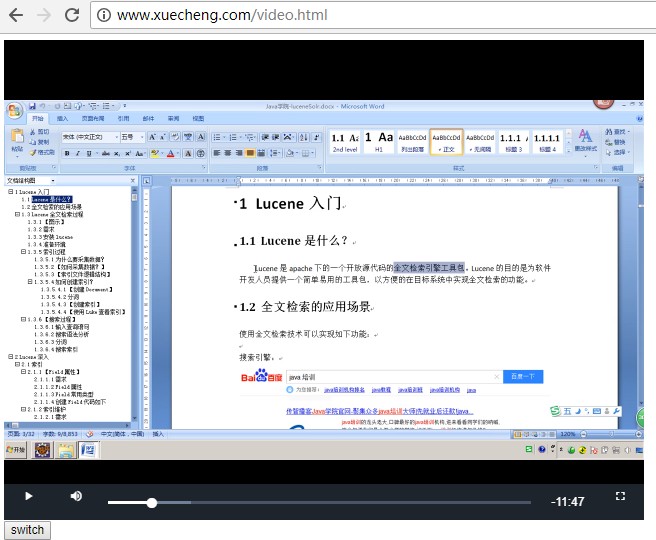
/html>

测试

置 osts 件，本教程开发环境使用 indow10 修改 :\Windows\System32\drivers\etc\hosts 件

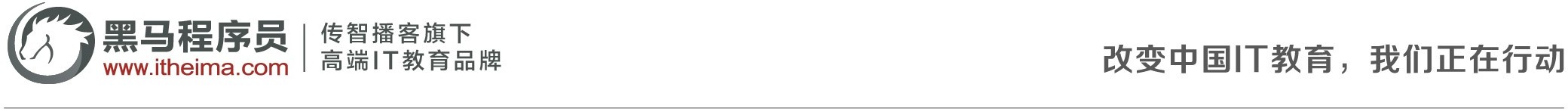
27.0.0.1 video.xuecheng.com

果：



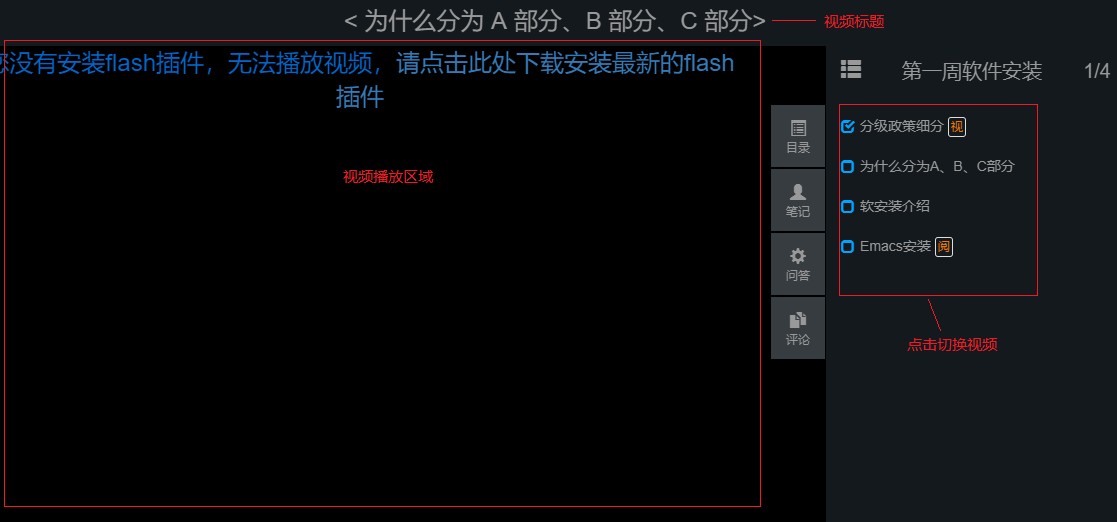
击 switch" 试切换视频功能。

# 建学习中心前端



成网学习中心提供学生在线学习的各各模块，上一章节测试的点播学习功能也属于学习中心的一部分，本章节将 现学习中心点播学习的前端部分。之所以先实现前端部分，主要是因为要将 ideo.js+vue.js 成，一部分精力还要放在技术研究。

## 面原型

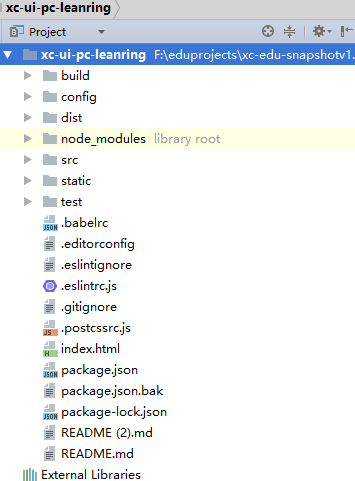
看一下界面原型，如下图，最终的目标是在此页面使用 ideo.js 放视频。

## 建学习中心工程

习中心的用户是学生，为了便于系统维护和扩展，单独创建学习中心工程： 从资料目录拷贝 c-ui-pc-leanring.zip 解 压 到 c-ui-pc-leanring 录 。使 用 ebstorm 建 打 开 c-ui-pc-leanring 录

进入 c-ui-pc-leanring 录，执行 npm install 将根据 ackage.json 依赖配置远程下载依赖的 s 。

建完成， c-ui-pc-leanring 程如下：



### 置域名

习中心的二级域名为 center.xuecheng.com 我们在 ginx 配置 center 拟主机。

成网用户中心

erver {

isten 80;

erver\_name ucenter.xuecheng.com;

人中心

ocation / {

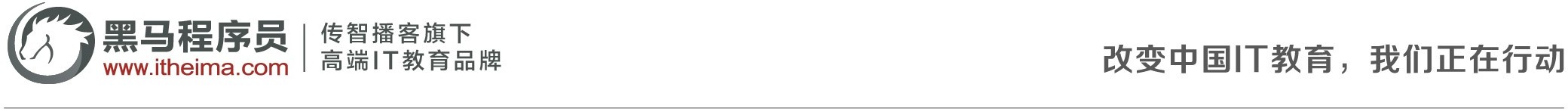
roxy\_pass http://ucenter\_server\_pool;

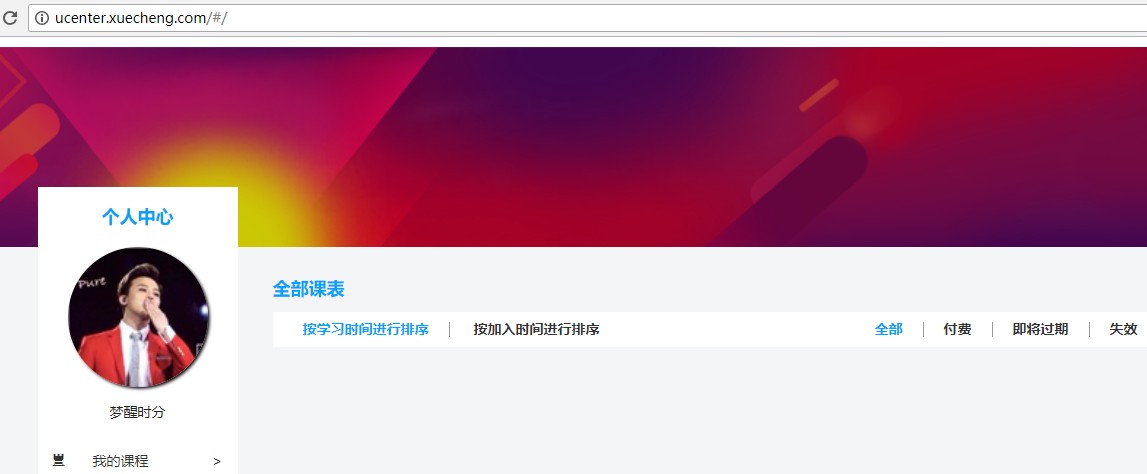
端 center

pstream ucenter\_server\_pool{ #server 127.0.0.1:7081 weight=10;

server 127.0.0.1:13000 weight=10;

### 问



启动工程，看到下边的界面说明本工程创建完成：

**4.3** 试视频播放页面

用 ue-video-player 件将 ideo.js 成到 ue.js ，本项目使用 ue-video-player 现 ideo.js 放。件地址： [ttps://github.com/surmon-china/vue-video-player](https://github.com/surmon-china/vue-video-player)

上面的 xc-ui-pc-learning 程已经添加 ue-video-player 件，我们在 ue 面直接使用即可。

边我们已经测试通过 video.js 下面我们直接在 ue 面中使用 ue-video-player 成视频播放。入 earning\_video.vue 面 到 ourse 块下。

置路由：

mport learning\_video from '@/module/course/page/learning\_video.vue';

{

path: '/learning/:courseId/:chapter', component: learning\_video,

name: ' 播视频学习 ,

hidden: false,

iconCls: 'el‐icon‐document'

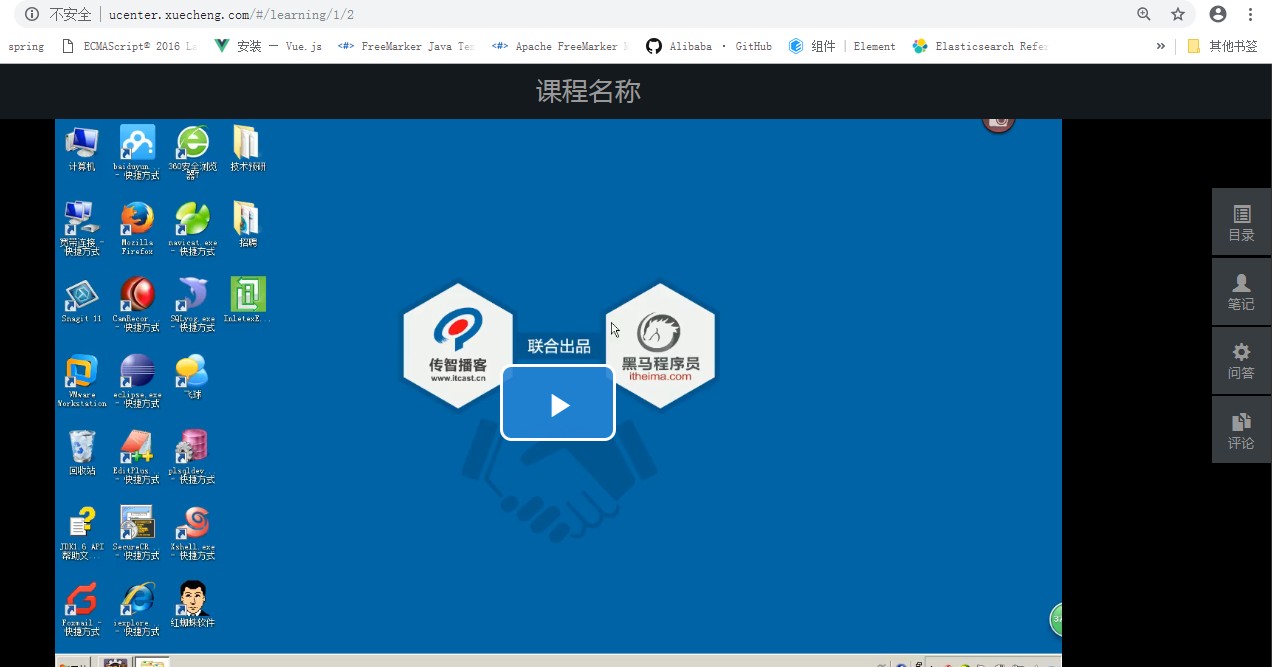
}

览效果：

求 ： [ttp://ucenter.xuecheng.com/#/learning/1/2](http://ucenter.xuecheng.com/%23/learning/1/2)

第一个参数： ourseId 课程 d 这里是测试页面效果随便输入一个 D 可，这里输入

二个参数： hapter 课程计划 d 这里是测试页面效果随便输入一个 D 可，这里输入

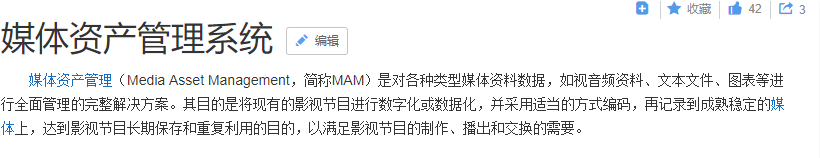


# 资管理

边章节完成在线视频播放，如何实现点击课程计划播放视频呢，课程视频如何管理呢？ 节开始将对课程视频进行管理。

## 求分析

资管理系统是每个在线教育平台所必须具备的，百度百科对它的定义如下：

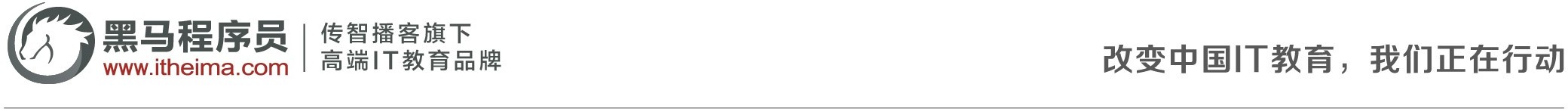


个教学机构都可以在媒资系统管理自己的教学资源，包括：视频、教案等文件。

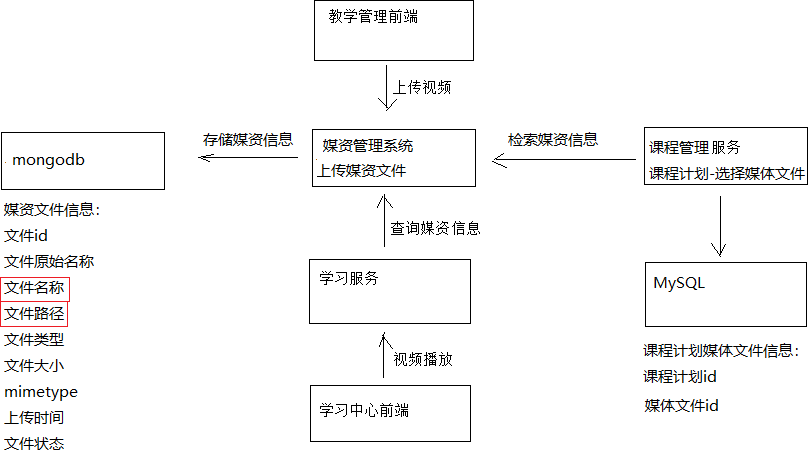
前媒资管理的主要管理对象是课程录播视频，包括：媒资文件的查询、视频上传、视频删除、视频处理等。 资查询：教学机构查询自己所拥有的媒体文件。

频上传：将用户线下录制的教学视频上传到媒资系统。

频处理：视频上传成功，系统自动对视频进行编码处理。频删除 如果该视频已不再使用，可以从媒资系统删除。



边是媒资系统与其它系统的交互情况：



上传媒资文件

端 户端请求媒资系统上传文件。

件上传成功将文件存储到媒资服务器，将文件信息存储到数据库。使用媒资

程管理请求媒资系统查询媒资信息，将课程计划与媒资信息对应、存储。

视频播放

户进入学习中心请求学习服务学习在线播放视频。

习服务校验用户资格通过后请求媒资系统获取视频地址。

## 发环境

### 建媒资数据库

媒资文件信息

Data ToString

Document(collection = "media\_file") ublic class MediaFile {

/\*

件 d 名称、大小、文件类型、文件状态（未上传、上传完成、上传失败）、上传时间、视频处理方式、视频处状态、 ls\_m3u8,hls\_ts\_list 课程视频信息（课程 d 章节 d

\*/



@Id

// 件 d

private String fileId;

// 件 名 称

private String fileName;

// 件原始名称

private String fileOriginalName;

// 件 路 径

private String filePath;

// 件 rl

private String fileUrl;

// 件 类 型

private String fileType;

//mimetype

private String mimeType;

// 件 大 小

private Long fileSize;

// 件 状 态

private String fileStatus;

// 传 时 间

private Date uploadTime;

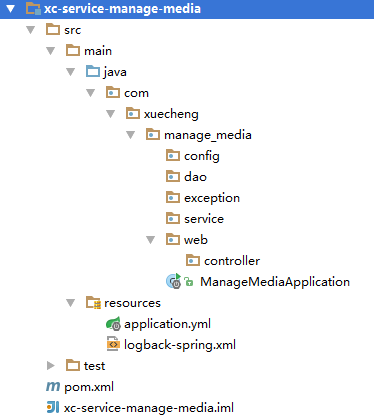
创 建 c\_media 据 库

资系统使用 ongodb 据库存储媒资信息。



### 建媒资服务工程

资管理的相关功能单独在媒资服务中开发，下边创建媒资服务工程（ c-service-manage-media 。资服务的配置与 ms 似，导入 “ 料 -- c-service-manage-media 程，工程结构如下：



**6.3** 传文件

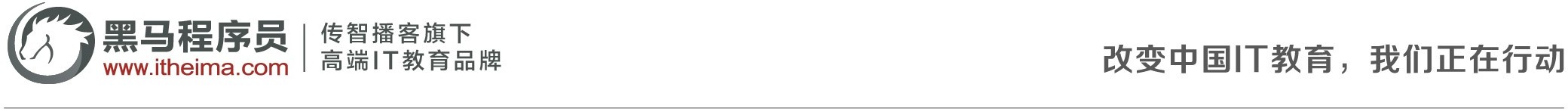
### 点续传解决方案

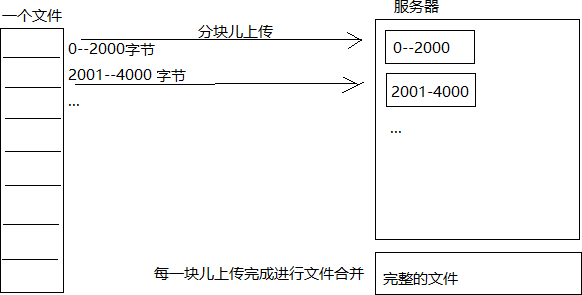
常视频文件都比较大，所以对于媒资系统上传文件的需求要满足大文件的上传要求。 ttp 议本身对上传文件大没有限制，但是客户的网络环境质量、电脑硬件环境等参差不齐，如果一个大文件快上传完了网断了，电断了没 上传完成，需要客户重新上传，这是致命的，所以对于大文件上传的要求最基本的是断点续传。

么是断点续传：

用百度百科：断点续传指的是在下载或上传时，将下载或上传任务（一个文件或一个压缩包）人为的划分为几个 分，每一个部分采用一个线程进行上传或下载，如果碰到网络故障，可以从已经上传或下载的部分开始继续上传 载未完成的部分，而没有必要从头开始上传下载，断点续传可以提高节省操作时间，提高用户体验性。

下图：





传流程如下：

上传前先把文件分成块

一块一块的上传，上传中断后重新上传，已上传的分块则不用再上传各分块上传完成最后合并文件

件下载则同理。

### 件分块与合并

了更好的理解文件分块上传的原理，下边用 ava 码测试文件的分块与合并。

* + 1. 件分块件分块的流程如下： 获取源文件长度

根据设定的分块文件的大小计算出块数

从源文件读数据依次向每一个块文件写数据。

/ 试文件分块方法

@Test

public void testChunk() throws IOException {

File sourceFile = new File("F:/develop/ffmpeg/lucene.mp4");

/ File sourceFile = new File("d:/logo.png"); String chunkPath = "F:/develop/ffmpeg/chunk/"; File chunkFolder = new File(chunkPath); if(!chunkFolder.exists()){

chunkFolder.mkdirs();



}

// 块 大 小

long chunkSize = 1024\*1024\*1;

// 块 数 量

long chunkNum = (long) Math.ceil(sourceFile.length() \* 1.0 / chunkSize ); if(chunkNum<=0){

chunkNum = 1;

}

// 冲区大小

byte[] b = new byte[1024];

// 用 andomAccessFile 问文件

RandomAccessFile raf\_read = new RandomAccessFile(sourceFile, "r");

// 块

for(int i=0;i<chunkNum;i++){

// 建分块文件

File file = new File(chunkPath+i); boolean newFile = file.createNewFile(); if(newFile){

// 分块文件中写数据

RandomAccessFile raf\_write = new RandomAccessFile(file, "rw"); int len = ‐1;

while((len = raf\_read.read(b))!=‐1){ raf\_write.write(b,0,len); if(file.length()>chunkSize){

break;

}

}

raf\_write.close();

}

}

raf\_read.close();

}

##### 件合并

件合并流程：

找到要合并的文件并按文件合并的先后进行排序。创建合并文件

依次从合并的文件中读取数据向合并文件写入数

/ 试文件合并方法

Test

ublic void testMerge() throws IOException {

// 文件目录

File chunkFolder = new File("F:/develop/ffmpeg/chunk/");

// 并 文 件

File mergeFile = new File("F:/develop/ffmpeg/lucene1.mp4");

if(mergeFile.exists()){ mergeFile.delete();

}

// 建新的合并文件

mergeFile.createNewFile();

// 于写文件

RandomAccessFile raf\_write = new RandomAccessFile(mergeFile, "rw");

// 针指向文件顶端

raf\_write.seek(0);

// 冲 区

byte[] b = new byte[1024];

// 块 列 表

File[] fileArray = chunkFolder.listFiles();

// 成集合，便于排序

List<File> fileList = new ArrayList<File>(Arrays.asList(fileArray));

// 小到大排序

Collections.sort(fileList, new Comparator<File>() { @Override

public int compare(File o1, File o2) {

if (Integer.parseInt(o1.getName()) < Integer.parseInt(o2.getName())) { return ‐1;

}

});

}

return 1;

// 并 文 件



for(File chunkFile:fileList){

RandomAccessFile raf\_read = new RandomAccessFile(chunkFile,"rw"); int len = ‐1;

while((len=raf\_read.read(b))!=‐1){ raf\_write.write(b,0,len);

}

raf\_read.close();

}

raf\_write.close();

### 端页面

传文件的页面内容参考： 料 -- pload.vue 件

* + 1. **WebUploader** 绍何在 eb 面实现断点续传？ 见的方案有：

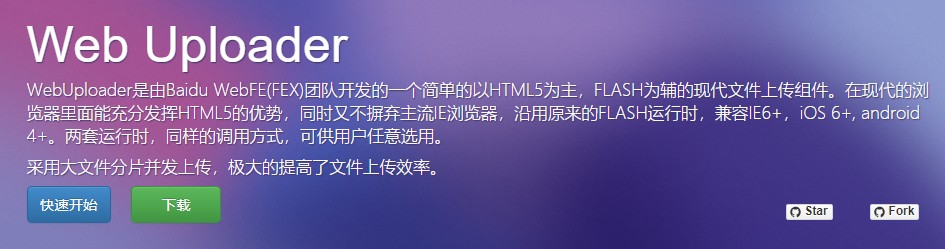
通过 lash 传，比如 WFupload ploadify

2 安装浏览器插件，变相的 c 户端，用的比较少。

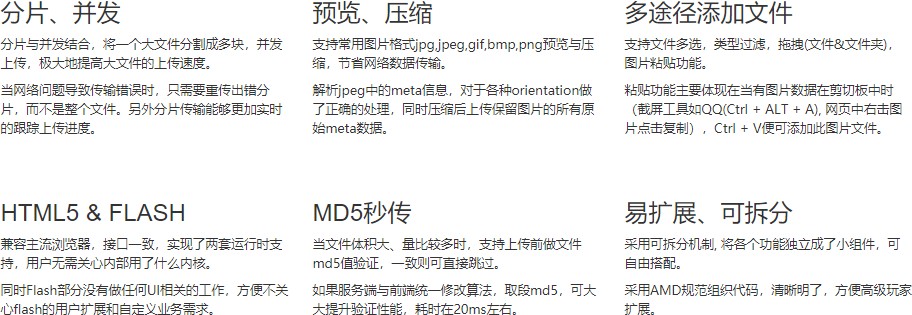
tml5

着 tml5 流行，本项目采用 tml5 成文件分块上传。

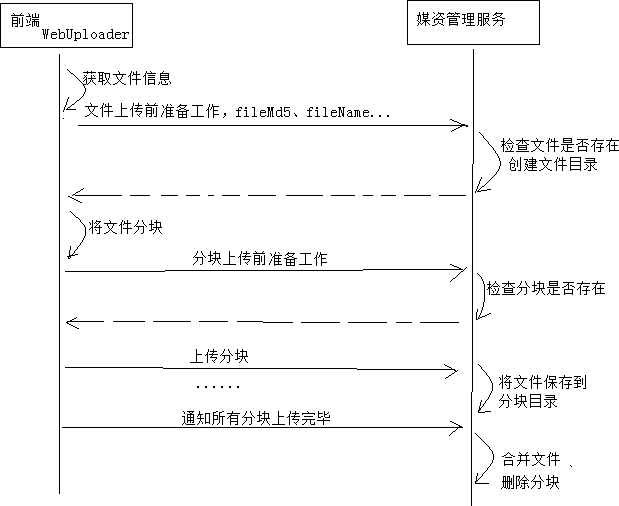
项目使用 ebUploader 成大文件上传功能的开发， ebUploader 网地址：

 [ttp://fexteam.gz01.bdysite.com/webuploader/](http://fexteam.gz01.bdysite.com/webuploader/)

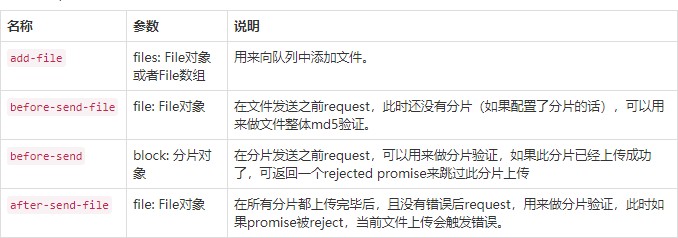
性如下：



用 ebUploader 传流程如下：



##### 子方法

ebuploader 提供很多钩子方法，下边列出一些重要的：

项目使用如下钩子方法：

efore-send-ﬁle

开始对文件分块儿之前调用，可以做一些上传文件前的准备工作，比如检查文件目录是否创建完成等。

2 efore-send

上传文件分块之前调用此方法，可以请求服务端检查分块是否存在，如果已存在则此分块儿不再上传。

fter-send-ﬁle

所有分块上传完成后触发，可以请求服务端合并分块文件。册钩子方法源代码：

ebUploader.Uploader.register({ "before‐send‐file":"beforeSendFile", "before‐send":"beforeSend", "after‐send‐file":"afterSendFile"

}

* + 1. 建 **ebUploader**

用 ebUploader 需 要 创 建 ebUploader 象 。

定上传分块的地址： api/media/upload/uploadchunk

/ 建 ploader 象，配置参数

his.uploader = WebUploader.create(

{

swf:"/static/plugins/webuploader/dist/Uploader.swf",// 传文件的 lash 件，浏览器不支持 5 启动

lash

server:"/api/media/upload/uploadchunk",// 传分块的服务端地址，注意跨域问题

fileVal:"file",// 件上传域的 ame

pick:"#picker",// 定选择文件的按钮容器

auto:false,// 动触发上传

disableGlobalDnd:true,// 掉整个页面的拖拽功能

chunked:true,// 否分块上传chunkSize:1\*1024\*1024, // 块大小（默认 M threads:3, // 启多个线程（默认 ）

prepareNextFile:true// 许在文件传输时提前把下一个文件准备好

}

#### before-send-ﬁle

件开始上传前前端请求服务端准备上传工作。考源代码如下：



ype:"POST",

rl:"/api/media/upload/register", ata:{

// 件唯一表示

fileMd5:this.fileMd5, fileName: file.name, fileSize:file.size, mimetype:file.type, fileExt:file.ext

#### before-send

传分块前前端请求服务端校验分块是否存在。考源代码如下：

ype:"POST",

rl:"/api/media/upload/checkchunk", ata:{

// 件唯一表示

fileMd5:this.fileMd5,

// 前分块下标

chunk:block.chunk,

// 前分块大小

chunkSize:block.end‐block.start

#### after-send-ﬁle

所有分块上传完成后触发，可以请求服务端合并分块文件考代码如下：

ype:"POST",

rl:"/api/media/upload/mergechunks", ata:{

fileMd5:this.fileMd5, fileName: file.name, fileSize:file.size, mimetype:file.type, fileExt:file.ext

##### 面效果



* 1. **Api** 口

义文件上传的 pi 口，此接收是前端 ebUploader 用服务端的接口。写此接口需要参数前端 ebUploader 用代码。

pi(value = " 资管理接口 ,description = " 资管理接口，提供文件上传，文件处理等接口 ) ublic interface MediaUploadControllerApi {

@ApiOperation(" 件上传注册 )

public ResponseResult register(String fileMd5,

String fileName, Long fileSize, String mimetype, String fileExt);

@ApiOperation(" 块检查 )

public CheckChunkResult checkchunk(String fileMd5,

Integer chunk, Integer chunkSize);

@ApiOperation(" 传分块 )

public ResponseResult uploadchunk(MultipartFile file,

Integer chunk, String fileMd5);

@ApiOperation(" 并文件 )

public ResponseResult mergechunks(String fileMd5,

String fileName, Long fileSize, String mimetype, String fileExt);

}

### 资服务端编写



**6.3.5.1** 务流程

务端需要实现如下功能： 上传前检查上传环境

查文件是否上传，已上传则直接返回。

查文件上传路径是否存在，不存在则创建。分块检查

查分块文件是否上传，已上传则返回 rue

上传则检查上传路径是否存在，不存在则创建。分块上传

分块文件上传到指定的路径。

合并分块

所有分块文件合并为一个文件。数据库记录文件信息。

##### 传注册

于上传过程复杂，开发时按业务流程分别实现。配置

pplication.yml 置上传文件的路径：

c‐service‐manage‐media: upload‐location: F:/develop/video/

定 义 ao

资文件管理 ao

ublic interface MediaFileRepository extends MongoRepository<MediaFile,String> {

ervice

能：

检查上传文件是否存在创建文件目录

Service

ublic class MediaUploadService {

private final static Logger LOGGER = LoggerFactory.getLogger(MediaUploadController.class);



@Autowired

MediaFileRepository mediaFileRepository;

// 传文件根目录@Value("${xc‐service‐manage‐media.upload‐location}") String uploadPath;

/\*\*

* 据文件 d5 到文件路径
* 则：
* 级目录： d5 第一个字符
* 级目录： d5 第二个字符
* 级目录： d5
* 件名： d5+ 件扩展名
* @param fileMd5 件 d5
* @param fileExt 件扩展名
* @return 件路径

\*/

private String getFilePath(String fileMd5,String fileExt){

String filePath = uploadPath+fileMd5.substring(0, 1) + "/" + fileMd5.substring(1, 2) +

/" + fileMd5 + "/" + fileMd5 + "." + fileExt; return filePath;

}

// 到文件目录相对路径，路径中去掉根目录

private String getFileFolderRelativePath(String fileMd5,String fileExt){

String filePath = fileMd5.substring(0, 1) + "/" + fileMd5.substring(1, 2) + "/" + ileMd5 + "/";

return filePath;

}

// 到文件所在目录

private String getFileFolderPath(String fileMd5){

String fileFolderPath = uploadPath+ fileMd5.substring(0, 1) + "/" + fileMd5.substring(1,

) + "/" + fileMd5 + "/" ; return fileFolderPath;

}

// 建文件目录

private boolean createFileFold(String fileMd5){

// 建上传文件目录

String fileFolderPath = getFileFolderPath(fileMd5); File fileFolder = new File(fileFolderPath);

if (!fileFolder.exists()) {

// 建文件夹

boolean mkdirs = fileFolder.mkdirs(); return mkdirs;

}

return true;

}



// 件上传注册

public ResponseResult register(String fileMd5, String fileName, String fileSize, String imetype, String fileExt) {

// 查文件是否上传

//1 得到文件的路径

String filePath = getFilePath(fileMd5, fileExt); File file = new File(filePath);

//2 查询数据库文件是否存在

Optional<MediaFile> optional = mediaFileRepository.findById(fileMd5);

// 件存在直接返回

if(file.exists() && optional.isPresent()){ ExceptionCast.cast(MediaCode.UPLOAD\_FILE\_REGISTER\_EXIST);

}

boolean fileFold = createFileFold(fileMd5); if(!fileFold){

// 传文件目录创建失败

ExceptionCast.cast(MediaCode.UPLOAD\_FILE\_REGISTER\_CREATEFOLDER\_FAIL);

}

return new ResponseResult(CommonCode.SUCCESS);

}

##### 块检查

ervice 定义分块检查方法：

/ 到块文件所在目录

rivate String getChunkFileFolderPath(String fileMd5){

tring fileChunkFolderPath = getFileFolderPath(fileMd5) +"/" + "chunks" + "/"; eturn fileChunkFolderPath;

/ 查块文件

ublic CheckChunkResult checkchunk(String fileMd5, String chunk, String chunkSize) {

// 到块文件所在路径

String chunkfileFolderPath = getChunkFileFolderPath(fileMd5);

// 文件的文件名称以 ,2,3.. 号命名，没有扩展名

File chunkFile = new File(chunkfileFolderPath+chunk); if(chunkFile.exists()){

return new CheckChunkResult(MediaCode.CHUNK\_FILE\_EXIST\_CHECK,true);

}else{

return new CheckChunkResult(MediaCode.CHUNK\_FILE\_EXIST\_CHECK,false);

}

##### 传分块

ervice 定义分块上传分块方法：

/ 文件上传



ublic ResponseResult uploadchunk(MultipartFile file, String fileMd5, String chunk) { if(file == null){

ExceptionCast.cast(MediaCode.UPLOAD\_FILE\_REGISTER\_ISNULL);

}

// 建块文件目录

boolean fileFold = createChunkFileFolder(fileMd5);

// 文 件

File chunkfile = new File(getChunkFileFolderPath(fileMd5) + chunk);

// 传的块文件

InputStream inputStream= null; FileOutputStream outputStream = null; try {

inputStream = file.getInputStream(); outputStream = new FileOutputStream(chunkfile); IOUtils.copy(inputStream,outputStream);

} catch (Exception e) { e.printStackTrace();

LOGGER.error("upload chunk file fail:{}",e.getMessage()); ExceptionCast.cast(MediaCode.CHUNK\_FILE\_UPLOAD\_FAIL);

}finally {

try {

inputStream.close();

} catch (IOException e) { e.printStackTrace();

}

try {

outputStream.close();

} catch (IOException e) { e.printStackTrace();

}

}

return new ResponseResult(CommonCode.SUCCESS);

// 建块文件目录

private boolean createChunkFileFolder(String fileMd5){

// 建上传文件目录

String chunkFileFolderPath = getChunkFileFolderPath(fileMd5); File chunkFileFolder = new File(chunkFileFolderPath);

if (!chunkFileFolder.exists()) {

// 建文件夹

boolean mkdirs = chunkFileFolder.mkdirs(); return mkdirs;

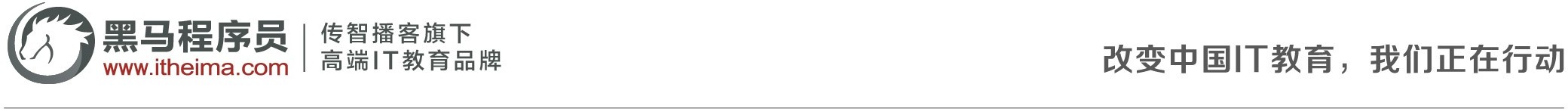
}

return true;

}

##### 并分块

ervice 定义分块合并分块方法，功能如下： 将块文件合并



2 校验文件 d5 否正确

向 ongodb 入文件信息

/ 并块文件

ublic ResponseResult mergechunks(String fileMd5, String fileName, Long fileSize, String imetype, String fileExt) {

// 取块文件的路径

String chunkfileFolderPath = getChunkFileFolderPath(fileMd5); File chunkfileFolder = new File(chunkfileFolderPath); if(!chunkfileFolder.exists()){

chunkfileFolder.mkdirs();

}

// 并文件路径

File mergeFile = new File(getFilePath(fileMd5,fileExt));

// 建合并文件

// 并文件存在先删除再创建

if(mergeFile.exists()){ mergeFile.delete();

}

boolean newFile = false; try {

newFile = mergeFile.createNewFile();

} catch (IOException e) { e.printStackTrace();

LOGGER.error("mergechunks..create mergeFile fail:{}",e.getMessage());

}

if(!newFile){

ExceptionCast.cast(MediaCode.MERGE\_FILE\_CREATEFAIL);

}

// 取块文件，此列表是已经排好序的列表

List<File> chunkFiles = getChunkFiles(chunkfileFolder);

// 并 文 件

mergeFile = mergeFile(mergeFile, chunkFiles); if(mergeFile == null){

ExceptionCast.cast(MediaCode.MERGE\_FILE\_FAIL);

}

// 验 文 件

boolean checkResult = this.checkFileMd5(mergeFile, fileMd5); if(!checkResult){

ExceptionCast.cast(MediaCode.MERGE\_FILE\_CHECKFAIL);

}

// 文件信息保存到数据库

MediaFile mediaFile = new MediaFile(); mediaFile.setFileId(fileMd5); mediaFile.setFileName(fileMd5+"."+fileExt); mediaFile.setFileOriginalName(fileName);

// 件路径保存相对路径

mediaFile.setFilePath(getFileFolderRelativePath(fileMd5,fileExt)); mediaFile.setFileSize(fileSize);

mediaFile.setUploadTime(new Date()); mediaFile.setMimeType(mimetype); mediaFile.setFileType(fileExt);

// 态为上传成功



mediaFile.setFileStatus("301002");

MediaFile save = mediaFileDao.save(mediaFile); return new ResponseResult(CommonCode.SUCCESS);

// 验文件的 d5

private boolean checkFileMd5(File mergeFile,String md5){ if(mergeFile == null || StringUtils.isEmpty(md5)){

return false;

}

// 行 d5 验

FileInputStream mergeFileInputstream = null; try {

mergeFileInputstream = new FileInputStream(mergeFile);

// 到文件的 d5

String mergeFileMd5 = DigestUtils.md5Hex(mergeFileInputstream);

// 较 d5 if(md5.equalsIgnoreCase(mergeFileMd5)){

return true;

}

} catch (Exception e) { e.printStackTrace();

LOGGER.error("checkFileMd5 error,file is:{},md5 is:

}",mergeFile.getAbsoluteFile(),md5);

}finally{

try {

mergeFileInputstream.close();

} catch (IOException e) { e.printStackTrace();

}

}

return false;

}

/ 取所有块文件

private List<File> getChunkFiles(File chunkfileFolder){

// 取路径下的所有块文件

File[] chunkFiles = chunkfileFolder.listFiles();

// 文件数组转成 ist 并排序

List<File> chunkFileList = new ArrayList<File>(); chunkFileList.addAll(Arrays.asList(chunkFiles));

// 序

Collections.sort(chunkFileList, new Comparator<File>() { @Override

public int compare(File o1, File o2) { if(Integer.parseInt(o1.getName())>Integer.parseInt(o2.getName())){

return 1;

}

return ‐1;



}

});

return chunkFileList;

}

// 并 文 件

private File mergeFile(File mergeFile,List<File> chunkFiles){ try {

// 建写文件对象

RandomAccessFile raf\_write = new RandomAccessFile(mergeFile,"rw");

// 历分块文件开始合并

// 取文件缓冲区

byte[] b = new byte[1024]; for(File chunkFile:chunkFiles){

RandomAccessFile raf\_read = new RandomAccessFile(chunkFile,"r"); int len = ‐1;

// 取分块文件

while((len = raf\_read.read(b))!=‐1){

// 合并文件中写数据

raf\_write.write(b,0,len);

}

raf\_read.close();

}

raf\_write.close();

} catch (Exception e) { e.printStackTrace();

LOGGER.error("merge file error:{}",e.getMessage()); return null;

}

return mergeFile;

}

* + 1. **Controller**

RestController RequestMapping("/media/upload")

ublic class MediaUploadController implements MediaUploadControllerApi {

@Autowired

MediaUploadService mediaUploadService;

@Override @PostMapping("/register")

public ResponseResult register(@RequestParam("fileMd5") String fileMd5, RequestParam("fileName") String fileName, @RequestParam("fileSize") Long fileSize, RequestParam("mimetype") String mimetype, @RequestParam("fileExt") String fileExt) {

return mediaUploadService.register(fileMd5,fileName,fileSize,mimetype,fileExt);

}

@Override @PostMapping("/checkchunk")

public CheckChunkResult checkchunk(@RequestParam("fileMd5") String fileMd5,



RequestParam("chunk") Integer chunk, @RequestParam("chunkSize") Integer chunkSize) { return mediaUploadService.checkchunk(fileMd5,chunk,chunkSize);

}

@Override @PostMapping("/uploadchunk")

public ResponseResult uploadchunk(@RequestParam("file") MultipartFile file, RequestParam("fileMd5") String fileMd5, @RequestParam("chunk") Integer chunk) {

return mediaUploadService.uploadchunk(file,fileMd5,chunk);

}

@Override @PostMapping("/mergechunks")

public ResponseResult mergechunks(@RequestParam("fileMd5") String fileMd5, RequestParam("fileName") String fileName, @RequestParam("fileSize") Long fileSize, RequestParam("mimetype") String mimetype, @RequestParam("fileExt") String fileExt) {

return mediaUploadService.mergechunks(fileMd5,fileName,fileSize,mimetype,fileExt);

}

