学成在线-第13天-讲义-在线学习 HLS

1 在线学习需求分析

1.1 需求描述

学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在线学习，本章节将开发录播课程的在线学习功能，需求如下：

1、学生可以在windows浏览器上在线观看视频。

2、播放器具有快进、快退、暂停等基本功能。

3、学生可以方便切换章节进行学习。

什么是录播课程？

录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。

课程视频如何管理？

媒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。

1.2 视频点播解决方案

1.2.1 流媒体

- 流媒体：

流媒体

锁定

本词条由"科普中国"百科科学词条编写与应用工作项目 审核 。

所谓流媒体是指采用流式传输的方式在Internet播放的媒体格式。流媒体又叫流式媒体，它是指商家用一个视频传送服务器把节目当成数据包发出，传送到网络上。用户通过解压设备对这些数据进行解压后，节目就会像发送前那样显示出来。

流媒体（Streaming Media）的出现极大地方便了人们的工作和生活。在地球的另一端，某大学的课堂上，某个教授正在兴致盎然地传授一门你喜欢的课程，想听？太远！放弃？可惜！没关系，网络时代能满足你的愿望。在网络上找到该在线课程，课程很长，但没关系，只管点击播放，教授的身影很快出现在屏幕上，课程一边播放一边下载，虽然远在天涯，却如亲临现场！除了远程教育，流媒体在视频点播、网络电台、网络视频等方面也有着广泛的应用。

详细参考：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，实现一边传输视频 数据 包一边观看视频。

- 流式传输

  在网络上传输音、视频信息有两个方式：下载和流式传输。

  下载：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，播放等待时间较长，无法去播放还未下载的部分视频。

  流式传输：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"边下载边播放"。

  流式传输包括如下两种方式：

  1）顺序流式传输

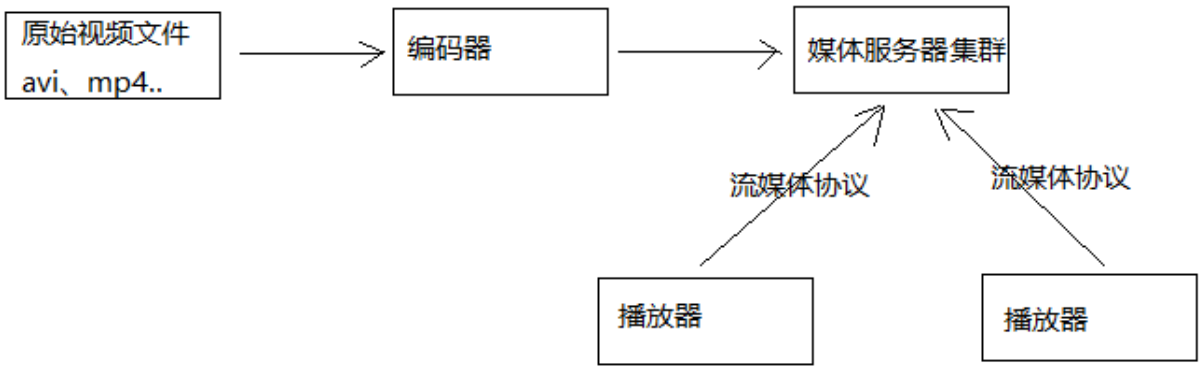
  即顺序下载音、视频文件，可以实现边下载边播放，不过，用户只能观看已下载的视频内容，无法快进到未下载的视频部分，顺序流式传输可以使用Http服务器来实现，比如Nginx、Apache等。

  2）实时流式传输

  实时流式传输可以解决顺序流式传输无法快进的问题，它与Http流式传输不同，它必须使用流媒体服务器并且使用流媒体协议来传输视频，它比Http流式传输复杂。常见的实时流式传输协议有RTSP、RTMP、RSVP等。

- 流媒体系统的概要结构

  通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。



1、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。

原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件，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 。

2、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 ：HTTP、RTSP、RTMP等 。

3、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。

1.2.2 点播方案

本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：

1、 播放器通过 http协议从http服务器上下载视频文件进行播放

问题：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

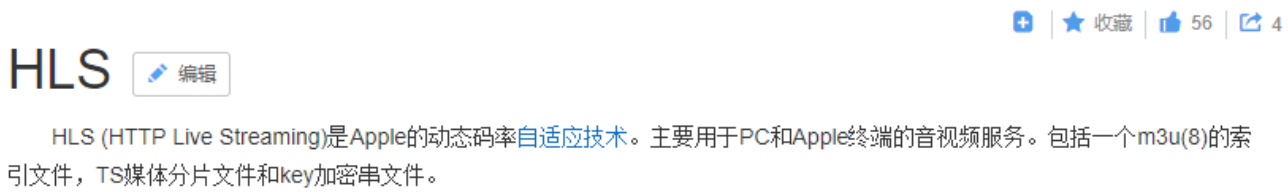
2、 播放器通过rtmp协议连接媒体服务器以实时流方式播放视频

使用rtmp协议需要架设媒体服务器，造价高，对于直播多采用此方案。

3、 播放器使用HLS协议连接http服务器（Nginx、Apache等）实现近实时流方式播放视频

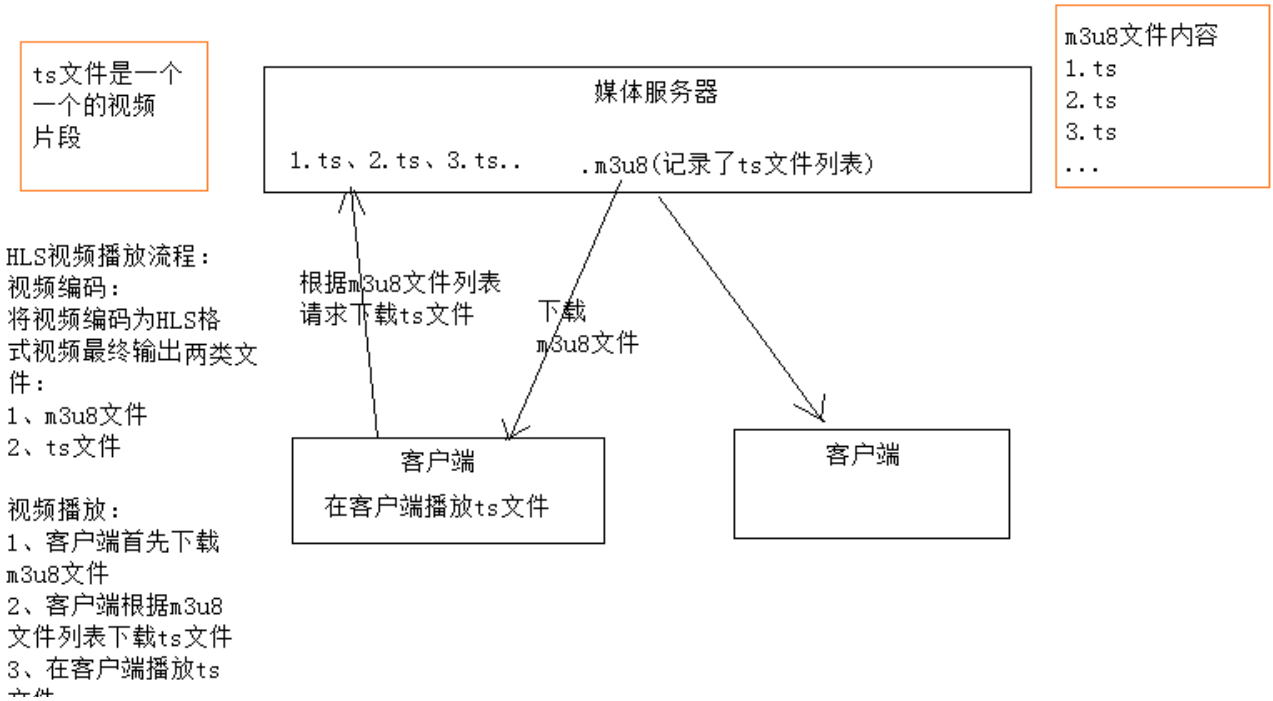
HLS协议规定：基于Http协议，视频封装格式为ts，视频的编码格式为H264,音频编码格式为MP3、AAC或者AC-3。

HLS是什么？



HLS的工作方式是：将视频拆分成若干ts格式的小文件，通过m3u8格式的索引文件对这些ts小文件建立索引。一般10秒一个ts文件，播放器连接m3u8文件播放，当快进时通过m3u8即可找到对应的索引文件，并去下载对应的ts文件，从而实现快进、快退以近实时 的方式播放视频。

IOS、Android设备、及各大浏览器都支持HLS协议。



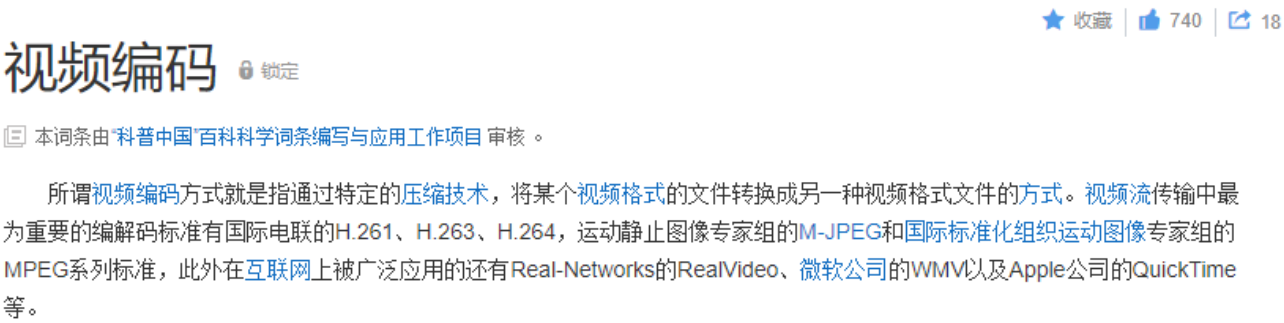
详细参考：https://baike.baidu.com/item/HLS/8328931?fr=aladdin

采用HLS方案即可实现边下载边播放，并可不用使用rtmp等流媒体协议，不用构建专用的媒体服务器，节省成本。

本项目点播方案确定为方案3。

2 视频编码

2.1 视频编码格式



详情参考 ：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：

文件格式：是指.mp4、.avi、.rmvb等 这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频，其文件格式是按照一 定的编码格式去编码，并且按照该文件所规定的封装格式将视频、音频、字幕等信息封装在一起，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。比如：一个.avi的视频文件原来的编码是a，通过编码后编码格式变为b，音频原来为c，通过编码后变为d。

音视频编码格式各类繁多，主要有几下几类：

MPEG系列（由ISO[国际标准组织机构]下属的MPEG[运动图象专家组]开发）视频编码方面主要是Mpeg1（vcd用的就是它）、Mpeg2（DVD使用）、Mpeg4（的DVDRIP使用的都是它的变种，如：divx，xvid等）、Mpeg4 AVC（正热门）；音频编码方面主要是MPEG Audio Layer 1/2、MPEG Audio Layer 3（大名鼎鼎的mp3）、MPEG-2 AAC 、MPEG-4 AAC等等。注意：DVD音频没有采用Mpeg的。

H.26X系列（由ITU[国际电传视讯联盟]主导，侧重网络传输，注意：只是视频编码）包括H.261、H.262、H.263、H.263+、H.263++、H.264（就是MPEG4 AVC-合作的结晶）

目前最常用的编码标准是视频H.264，音频AAC。

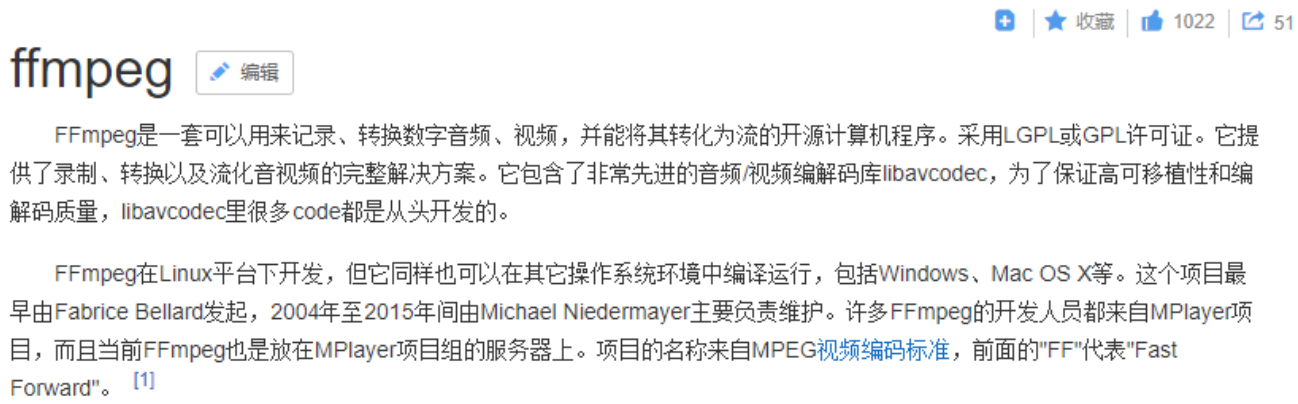
提问：

H.264是编码格式还是文件格式？

mp4是编码格式还是文件格式？

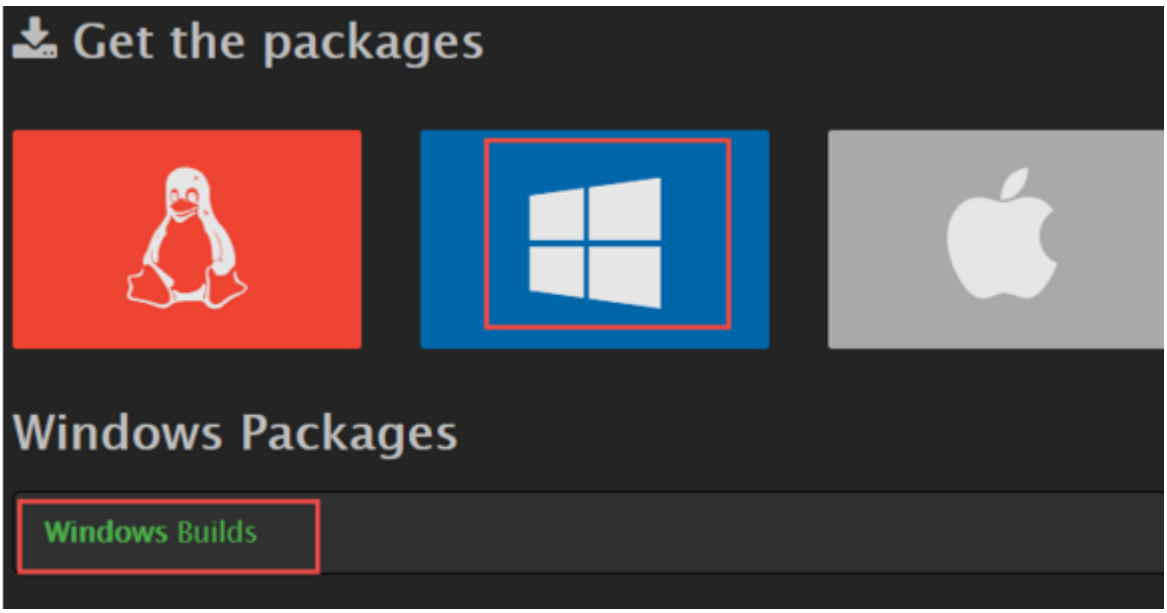
2.2 FFmpeg 的基本使用

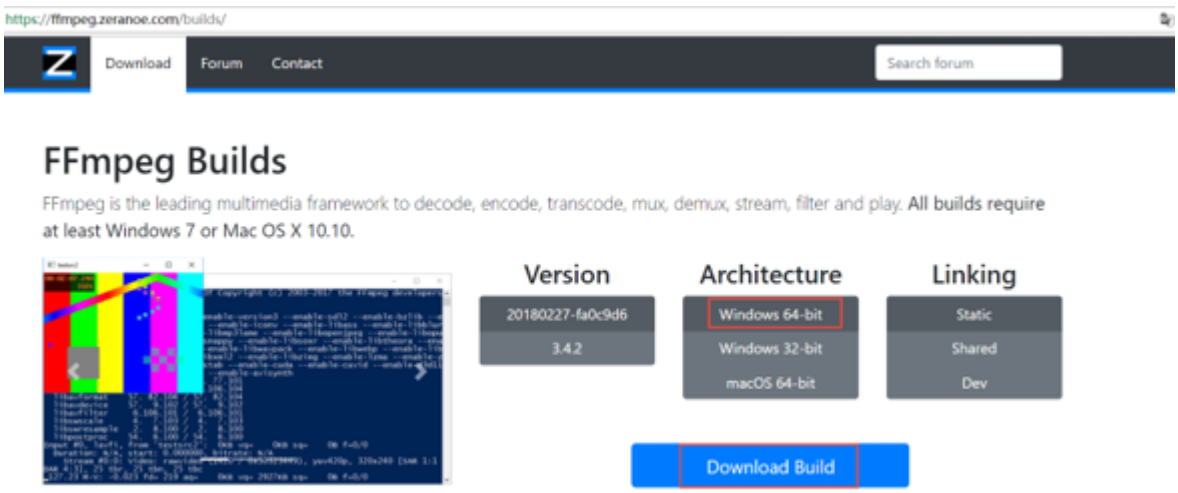
我们将视频录制完成后，使用视频编码软件对视频进行编码，本项目 使用FFmpeg对视频进行编码。



FFmpeg被许多开源项目采用，QQ影音、暴风影音、VLC等。

下载：FFmpeg https://www.ffmpeg.org/download.html#build-windows





下载 ：ffmpeg-20180227-fa0c9d6-win64-static.zip，并解压，本教程将ffmpeg解压到了F:\devenv\edusoft\ffmpeg-20180227-fa0c9d6-win64-static\ffmpeg-20180227-fa0c9d6-win64-static下。

将F:\devenv\edusoft\ffmpeg-20180227-fa0c9d6-win64-static\ffmpeg-20180227-fa0c9d6-win64-static\bin目录配置在path环境变量中。

检测是否安装成功：

```
C:\Users\admin>ffmpeg -version
ffmpeg version N-90173-gfa0c9d69d3 Copyright (c) 2000-2018 the FFmpeg developers
built with gcc 7.3.0 (GCC)
configuration: --enable-gpl --enable-version3 --enable-sdl2 --enable-bzlib --enable-fontconfig --enable-gnu
iconv --enable-libass --enable-libbluray --enable-libfreetype --enable-libmp3lame --enable-libopencore-amrn
bopencore-amrwb --enable-libopenjpeg --enable-libopus --enable-libshine --enable-libsnappy --enable-libsoxr
theora --enable-libtwolame --enable-libvpx --enable-libwavpack --enable-libwebp --enable-libx264 --enable-l
le-libxml2 --enable-libzimg --enable-lzma --enable-zlib --enable-gmp --enable-libvidstab --enable-libvorbis
vo-amrwbenc --enable-libmysofa --enable-libspeex --enable-libxvid --enable-libmfx --enable-amf --enable-cud
vid --enable-d3d11va --enable-nvenc --enable-dxva2 --enable-avisynth
libavutil      56.  7.101 / 56.  7.101
libavcodec     58. 13.100 / 58. 13.100
libavformat    58. 10.100 / 58. 10.100
libavdevice    58.  2.100 / 58.  2.100
libavfilter     7. 12.100 /  7. 12.100
libswscale      5.  0.101 /  5.  0.101
libswresample   3.  0.101 /  3.  0.101
libpostproc    55.  0.100 / 55.  0.100
```

简单的测试：

将一个.avi文件转成mp4、mp3、gif等。

比如我们将lucene.avi文件转成mp4，运行如下命令：

ffmpeg -i lucene.avi lucene.mp4

转成mp3：ffmpeg -i lucene.avi lucene.mp3

转成gif：ffmpeg -i lucene.avi lucene.gif

官方文档（英文）：http://ffmpeg.org/ffmpeg.html

2.2 生成m3u8/ts文件

使用ffmpeg生成 m3u8的步骤如下：

第一步：先将avi视频转成mp4

```
ffmpeg.exe -i  lucene.avi -c:v libx264 -s 1280x720 -pix_fmt yuv420p -b:a 63k -b:v 753k -r 18
.\lucene.mp4
```

下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 视频编码为x264 ，x264编码是H264的一种开源编码格式。

-s 设置分辨率

-pix_fmt yuv420p：设置像素采样方式，主流的采样方式有三种，YUV4:4:4，YUV4:2:2，YUV4:2:0，它的作用是根据采样方式来从码流中还原每个像素点的YUV（亮度信息与色彩信息）值。

-b 设置码率，-b:a和-b:v分别表示音频的码率和视频的码率，-b表示音频加视频的总码率。码率对一个视频质量有很大的作用，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24肉眼就没有连贯与停顿的感觉了。

第二步：将mp4生成m3u8

```
ffmpeg -i  lucene.mp4   -hls_time 10 -hls_list_size 0  -hls_segment_filename
./hls/lucene_%05d.ts ./hls/lucene.m3u8
```

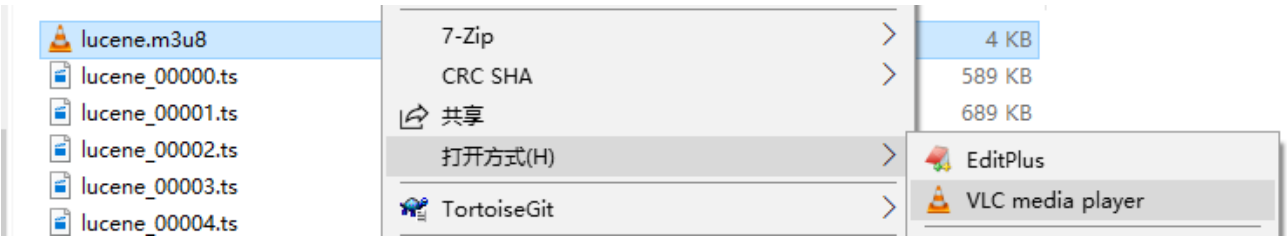
-hls_time 设置每片的长度，单位为秒

-hls_list_size n: 保存的分片的数量，设置为0表示保存所有分片

-hls_segment_filename ：段文件的名称，%05d表示5位数字

生成的效果是：将lucene.mp4视频文件每10秒生成一个ts文件，最后生成一个m3u8文件，m3u8文件是ts的索引文件。

使用VLC打开m3u8文件，测试播放效果，VLC 是一款自由、开源的跨平台多媒体播放器及框架，可播放大多数多媒体文件，以及 DVD、音频 CD、VCD 及各类流媒体协议。（http://www.videolan.org/）



2.2.1 码率的设置

码率又叫比特率即每秒传输的bit数，单位为bps(Bit Per Second)，码率越大传送数据的速度越快。

码率的计算公式是：文件大小（转成bit）/ 时长（秒）/1024 = kbps 即每秒传输千位数

例如一个1M的视频，它的时长是10s，它的码率等于

```
1*1024*1024*8/10/1024 = 819Kbps
```

码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：

转码高清、超清、1080p视频格式要求

一、普通视频要求

分辨率要求：

高清分辨率>=960x400；超清分辨率>=1280x544；1080p分辨率>=1920x800；

时长要求：

高清、超清、1080p时长>=15秒；

视频平均码率要求：

1.以下这些视频平均码率>0.56Mbps时为高清，>0.9Mbps时为超清，>2.1Mbps时为1080p：

—H.265/HEVC 通常使用MP4文件格式

2.以下这些视频平均码率>0.8Mbps时为高清，>1.5Mbps时为超清，>3.5Mbps时为1080p：

—H.264/AVC(Advance Video Coding)/AVCHD/X264 通常使用MP4,MKV文件格式，不建议使用flv格式上传

—RV40/RealVideo 9, 通常使用 RMVB文件格式

—WMV3/WVC1/WMVA/VC-1/Windows Media Video 9, 通常使用WMV文件格式

3.以下这些视频平均码率>1.6Mbps时为高清，>3Mbps时为超清，>5Mbps时为1080p：

—MPEG-4 Visual/Xvid/Divx, 通常使用AVI,MP4文件格式

4.以下这些视频平均码率>4Mbps时为高清，>7.5Mbps时为超清，>8Mbps时为1080p：

—MPEG-2, 通常使用MPEG/MPG/VOB文件格式

—MPEG-1, 通常使用MPEG/MPG文件格式

5.以下这些视频平均码率>9.6Mbps时为高清，>19Mbps时为超清，>45Mbps时为1080p：

—Apple ProRes 422 Proxy(apco), 通常使用MOV文件格式

6.以下这些视频平均码率>23.2Mbps时为高清，>44Mbps时为超清，>102Mbps时为1080p：

—Apple ProRes 422 LT(apcs), 通常使用MOV文件格式

7.以下这些视频平均码率>33.6Mbps时为高清，>63Mbps时为超清，>147Mbps时为1080p：

—Apple ProRes 422 Standard Definition(apcn), 通常使用MOV文件格式

8.以下这些视频平均码率>50.4Mbps时为高清，>94Mbps时为超清，>220Mbps时为1080p：

—Apple ProRes 422 High Quality(apch), 通常使用MOV文件格式

9.以下这些视频平均码率>75.2Mbps时为高清，>141Mbps时为超清，>330Mbps时为1080p：

如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，最终达到的视频清晰度满足业务需求即可。

3 播放器

3.1 技术选型

视频编码后要使用播放器对其进行解码、播放视频内容。在web应用中常用的播放器有flash播放器、H5播放器或浏览器插件播放器，其中以flash和H5播放器最常见。

flash播放器：缺点是需要在客户机安装Adobe Flash Player播放器，优点是flash播放器已经很成熟了，并且浏览器对flash支持也很好。

H5播放器：基于h5自带video标签进行构建，优点是大部分浏览器支持H5，不用再安装第三方的flash播放器，并且随着前端技术的发展，h5技术会越来越成熟。

本项目采用H5播放器，使用Video.js开源播放器。

Video.js是一款基于HTML5世界的网络视频播放器。它支持HTML5和Flash视频，它支持在台式机和移动设备上播放视频。这个项目于2010年中开始，目前已在40万网站使用。

官方地址：http://videojs.com/

3.2 下载video.js

Video.js： https://github.com/videojs/video.js

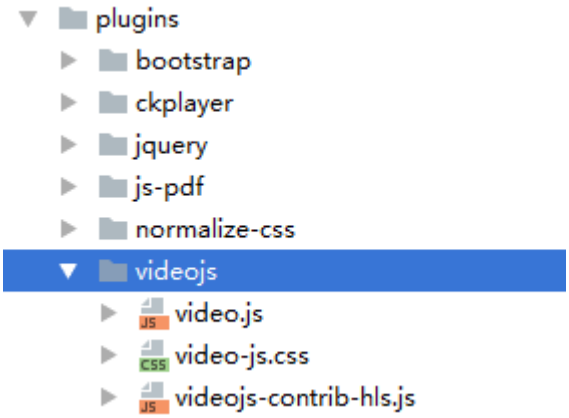
videojs-contrib-hls： https://github.com/videojs/videojs-contrib-hls#installation

（videojs-contrib-hls是播放hls的一个插件）

使用文档：http://docs.videojs.com/tutorial-videojs_.html

本教程使用 video.js 6.7.3 版本，videojs-contrib-hls 5.14.1版本。

下载上边两个文件，为了测试需求将其放在门户的plugins目录中。

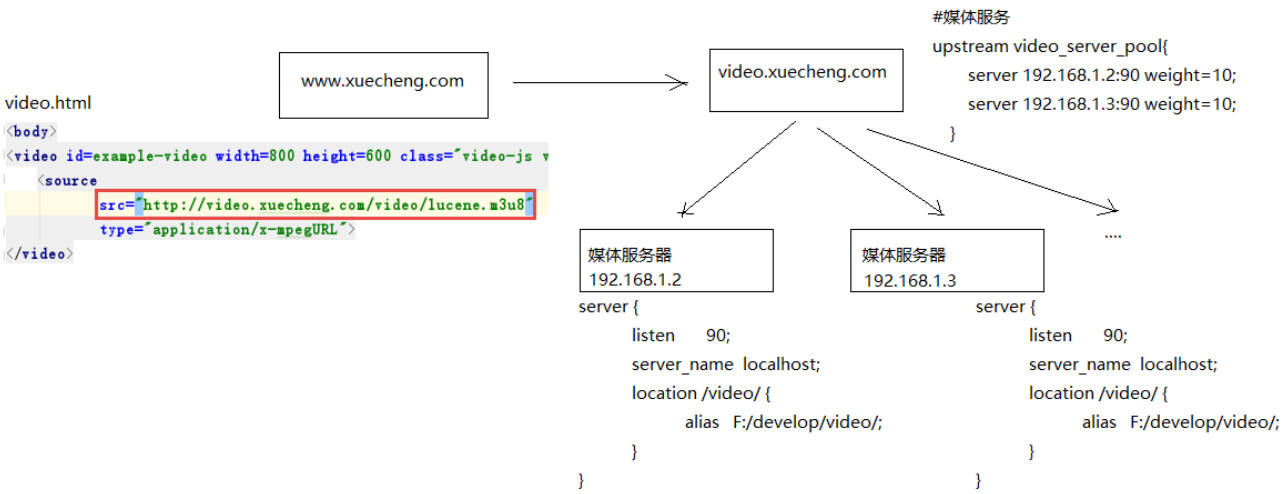


3.3 搭建媒体服务器

正常使用video.js播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web服务器请求，通常视频的url地址使用单独的域名。

3.3.1 Nginx媒体服务器

HLS协议基于Http协议，本项目使用Nginx作为视频服务器。下图是Nginx媒体服务器的配置流程图：



1、用户打开www.xuecheng.com上边的video.html网页

在此网页中引入视频链接，视频地址指向video.xuecheng.com

2、video.xuecheng.com进行负载均衡处理，将视频请求转发到媒体服务器

根据上边的流程，我们在媒体服务器上安装Nginx，并配置如下：

```
#学成网媒体服务
server {
    listen       90;
    server_name  localhost;

    #视频目录
    location /video/ {
        alias   F:/develop/video/;
    }
}
```

3.3.2 媒体服务器代理

媒体服务器不止一台，通过代理实现负载均衡功能，使用Nginx作为媒体服务器的代理，此代理服务器作为video.xuecheng.com域名服务器。

配置video.xuecheng.com虚拟主机：

注意：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Nginx。

```
#学成网媒体服务代理
map $http_origin $origin_list{
    default http://www.xuecheng.com;
    "~http://www.xuecheng.com" http://www.xuecheng.com;
    "~http://ucenter.xuecheng.com" http://ucenter.xuecheng.com;
}
#学成网媒体服务代理
server {
    listen       80;
    server_name video.xuecheng.com;

    location /video {
        proxy_pass http://video_server_pool;
        add_header Access-Control-Allow-Origin $origin_list;
        #add_header Access-Control-Allow-Origin *;
        add_header Access-Control-Allow-Credentials true;
        add_header Access-Control-Allow-Methods GET;
    }

}
```

cors跨域参数：

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

通常允许跨域访问的站点不是一个，所以这里用map定义了多个站点。

如果允许任何站点跨域访问则设置为*，通常这是不建议的。

Access-Control-Allow-Credentials： 允许客户端携带证书访问

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

video_server_pool的配置如下：

```
    #媒体服务
    upstream video_server_pool{
        server 127.0.0.1:90 weight=10;
    }
```

3.4 测试video.js

参考https://github.com/videojs/videojs-contrib-hls#installation

http://jsbin.com/vokipos/8/edit?html,output

1、编写测试页面video.html。

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta http-equiv="content-type" content="text/html; charset=utf-8" />
    <title>视频播放</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-
centered" 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();
    //切换视频
    function switchvideo(){
        player.src({
            src: 'http://video.xuecheng.com/video/hls/lucene.m3u8',
            type: 'application/x-mpegURL',

            withCredentials: true
```

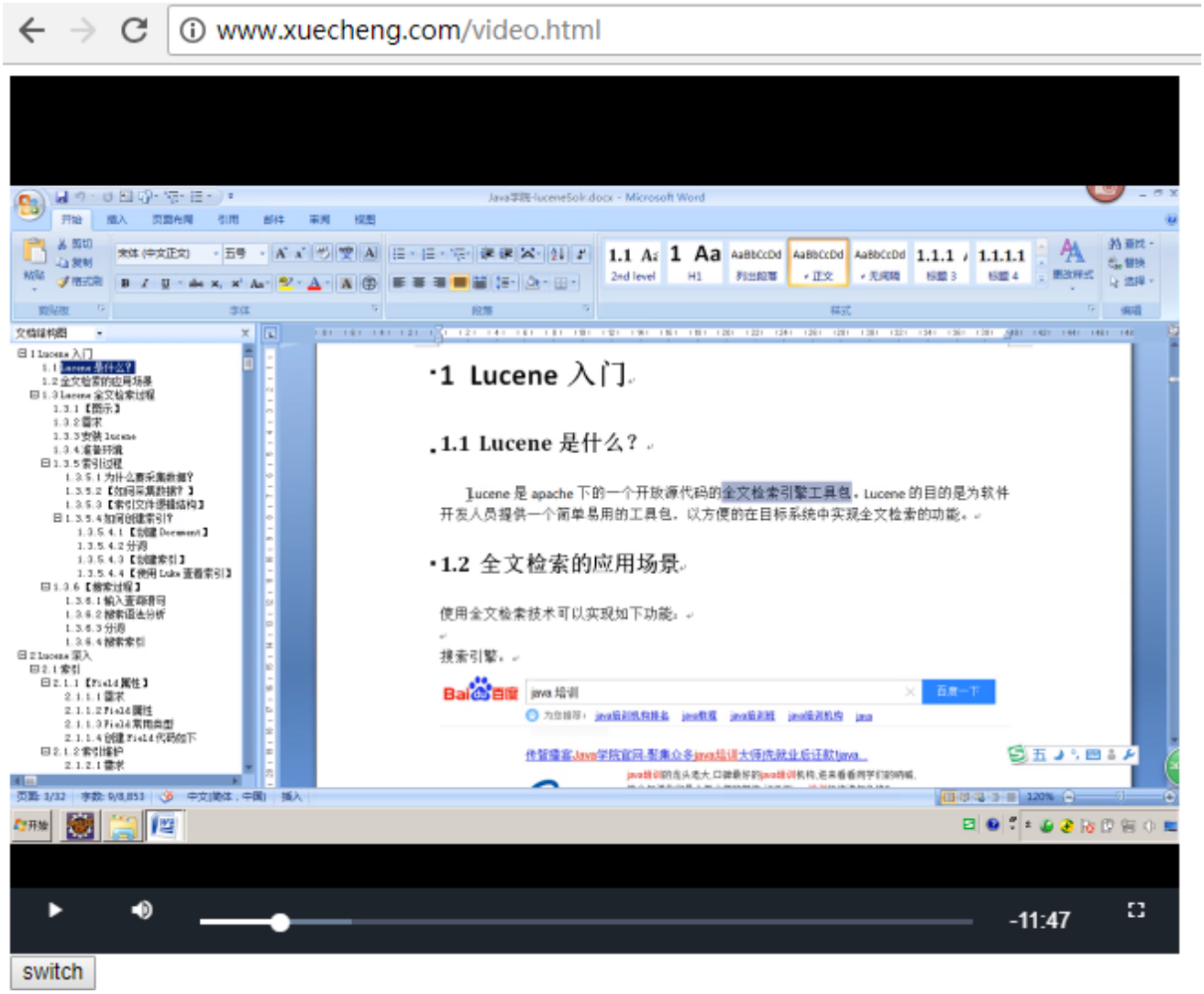
```
        });
        player.play();
    }
</script>

</body>
</html>
```

2、测试

配置hosts文件，本教程开发环境使用Window10，修改C:\Windows\System32\drivers\etc\hosts文件

```
127.0.0.1 video.xuecheng.com
```

效果：



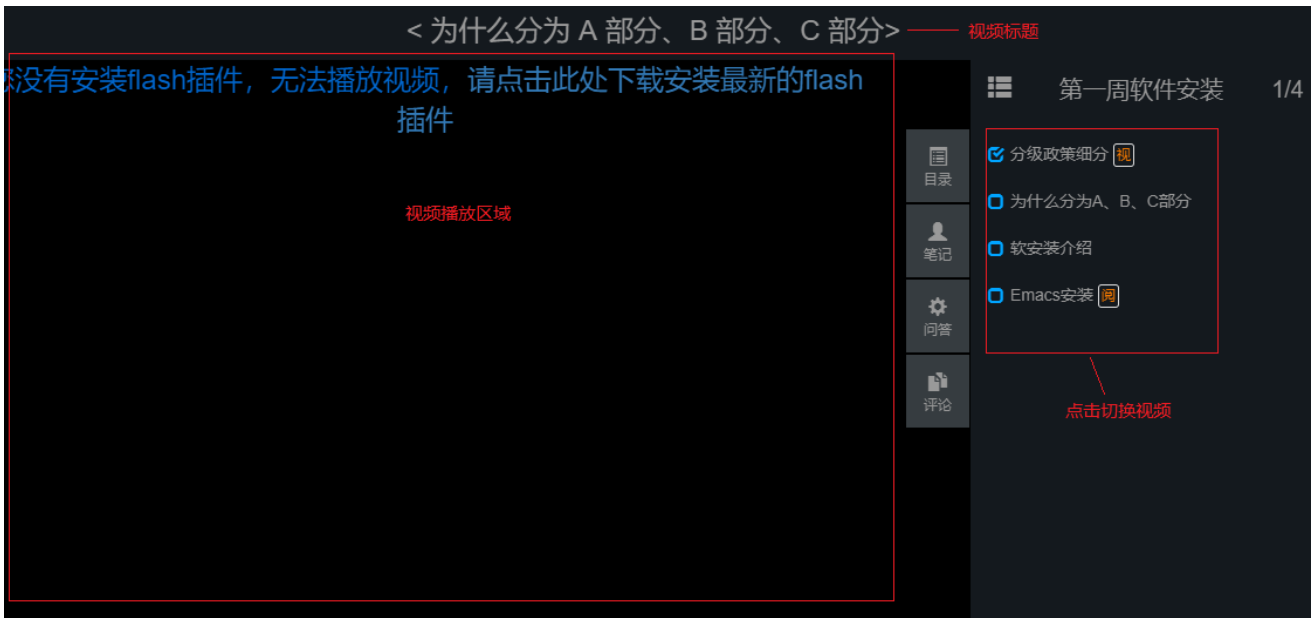
点击"switch"测试切换视频功能。

4 搭建学习中心前端

学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，本章节将实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video.js+vue.js集成，一部分精力还是要放在技术研究。

4.1 界面原型

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



4.2 创建学习中心工程

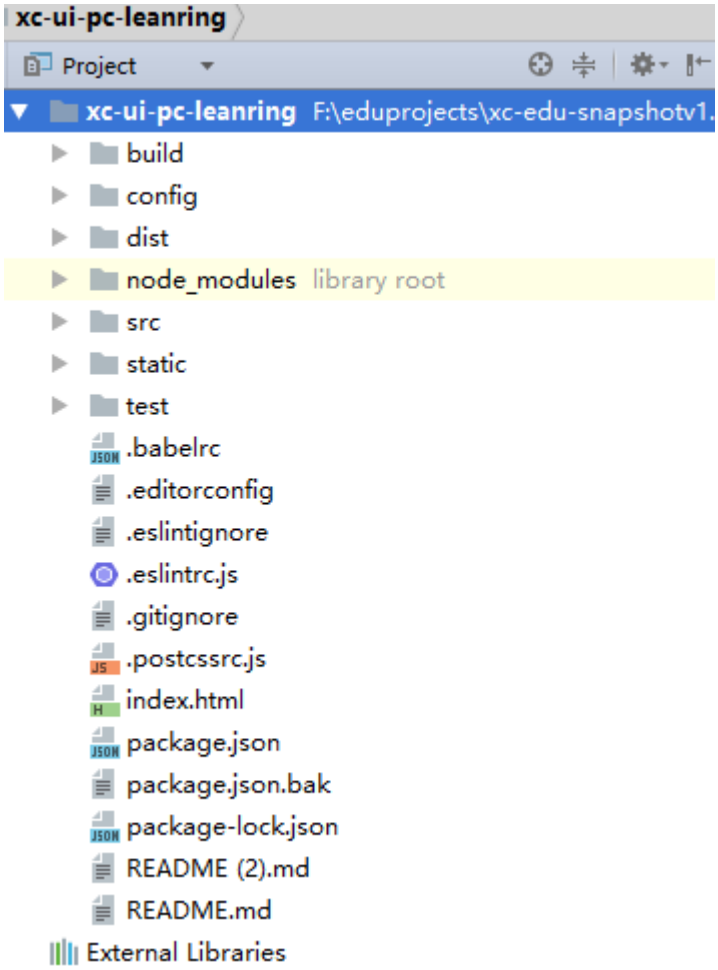
学习中心的用户是学生，为了便于系统维护和扩展，单独创建学习中心工程：

1、从资料目录拷贝xc-ui-pc-leanring.zip 并解压到xc-ui-pc-leanring目录。

2、使用webstorm创建打开xc-ui-pc-leanring目录

3、进入xc-ui-pc-leanring目录，执行cnpm install，将根据package.json的依赖配置远程下载依赖的js包。

创建完成，xc-ui-pc-leanring工程如下：



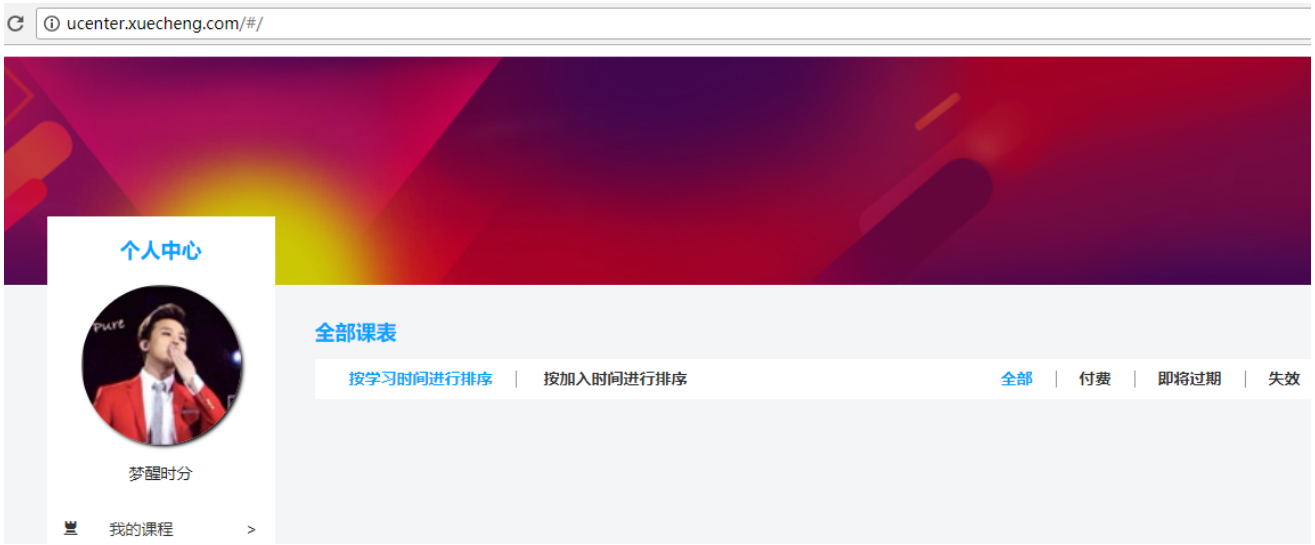
4.2.1 配置域名

学习中心的二级域名为ucenter.xuecheng.com，我们在nginx中配置ucenter虚拟主机。

```
#学成网用户中心
server {
    listen       80;
    server_name ucenter.xuecheng.com;

    #个人中心
    location / {
        proxy_pass http://ucenter_server_pool;
    }
}
#前端ucenter
upstream ucenter_server_pool{
  #server 127.0.0.1:7081 weight=10;
  server 127.0.0.1:13000 weight=10;
}
```

4.2.2 访问

启动工程，看到下边的界面说明本工程创建完成：



4.3 调试视频播放页面

使用vue-video-player组件将video.js集成到vue.js中，本项目使用vue-video-player实现video.js播放。

组件地址：https://github.com/surmon-china/vue-video-player

上面的 xc-ui-pc-learning工程已经添加vue-video-player组件，我们在vue页面直接使用即可。

前边我们已经测试通过 video.js，下面我们直接在vue页面中使用vue-video-player完成视频播放。

导入learning_video.vue页面到course 模块下。

配置路由：

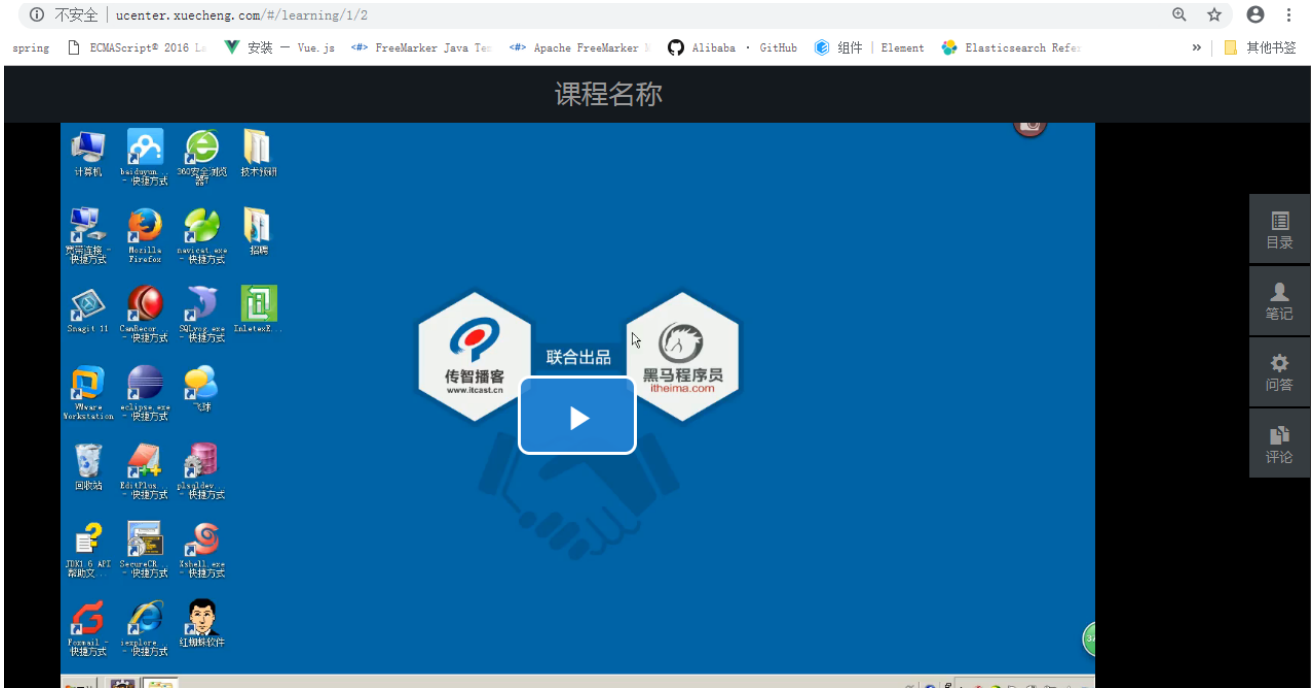
```
import learning_video from '@/module/course/page/learning_video.vue';
  {
    path: '/learning/:courseId/:chapter',
    component: learning_video,
    name: '录播视频学习',
    hidden: false,
    iconCls: 'el-icon-document'
  }
```

预览效果：

请求：http://ucenter.xuecheng.com/#/learning/1/2

第一个参数：courseId，课程id，这里是测试页面效果随便输入一个ID即可，这里输入1

第二个参数：chapter，课程计划id，这里是测试页面效果随便输入一个ID即可，这里输入2



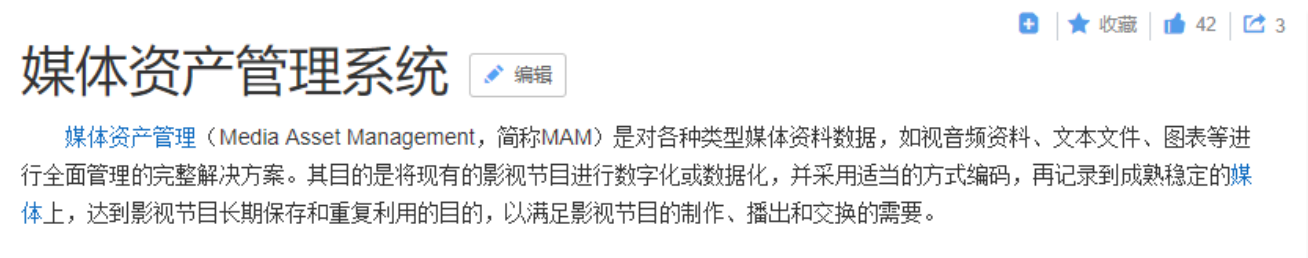
6 媒资管理

前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？

本节开始将对课程视频进行管理。

6.1需求分析

媒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：



每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。

目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。

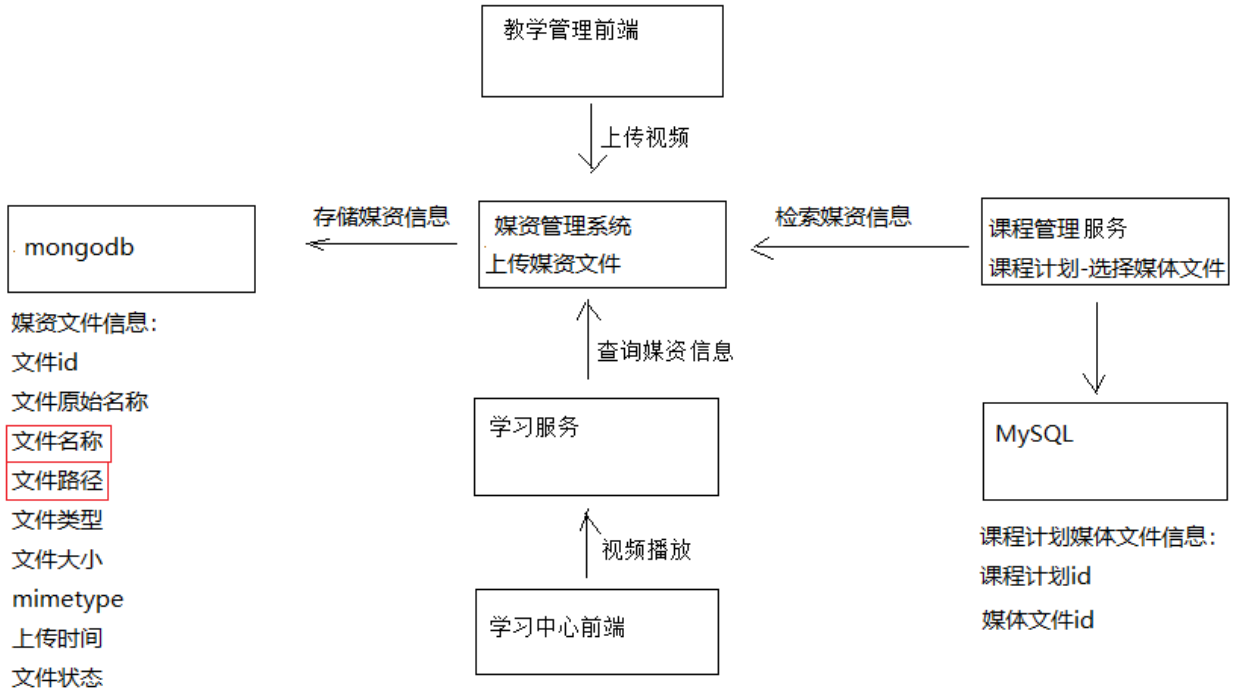
媒资查询：教学机构查询自己所拥有的媒体文件。

视频上传：将用户线下录制的教学视频上传到媒资系统。

视频处理：视频上传成功，系统自动对视频进行编码处理。

视频删除 ：如果该视频已不再使用，可以从媒资系统删除。

下边是媒资系统与其它系统的交互情况：



1、上传媒资文件

前端/客户端请求媒资系统上传文件。

文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。

2、使用媒资

课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。

3、视频播放

用户进入学习中心请求学习服务学习在线播放视频。

学习服务校验用户资格通过后请求媒资系统获取视频地址。

6.2 开发环境

6.2.1 创建媒资数据库

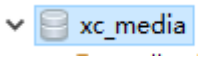
1、媒资文件信息

```
@Data
@ToString
@Document(collection = "media_file")
public class MediaFile {
    /*
    文件id、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、视频处理状态、hls_m3u8,hls_ts_list、课程视频信息（课程id、章节id）
     */
```

```
    @Id
    //文件id
    private String fileId;
    //文件名称
    private String fileName;
    //文件原始名称
    private String fileOriginalName;
    //文件路径
    private String filePath;
     //文件url
    private String fileUrl;
    //文件类型
    private String fileType;
    //mimetype
    private String mimeType;
    //文件大小
    private Long fileSize;
    //文件状态
    private String fileStatus;
    //上传时间
    private Date uploadTime;

}
```

2、创建xc_media数据库

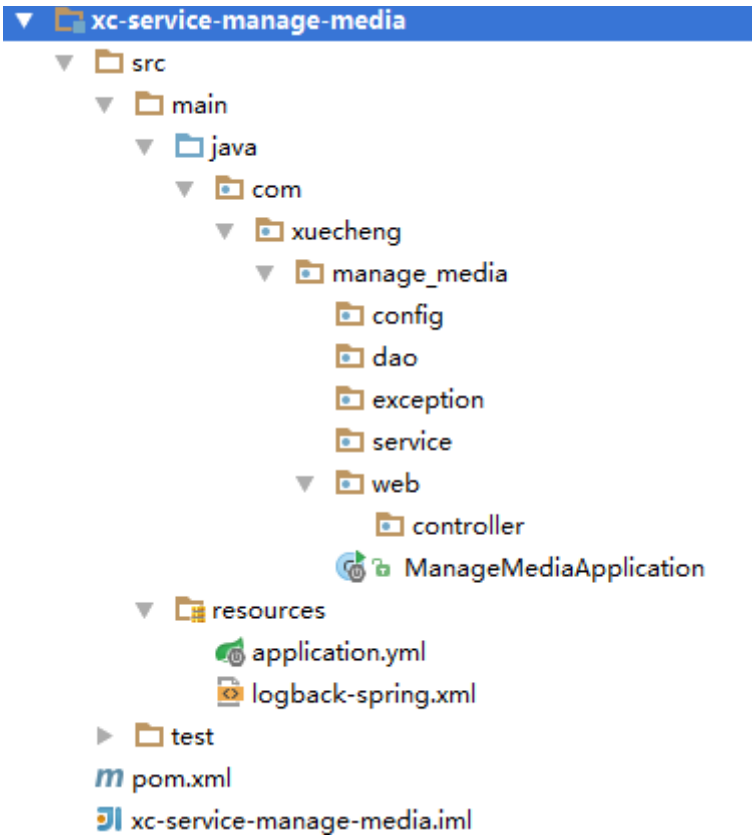
媒资系统使用mongodb数据库存储媒资信息。



6.2.2 创建媒资服务工程

媒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（xc-service-manage-media）。

媒资服务的配置与cms类似，导入“资料”--》xc-service-manage-media工程，工程结构如下：



6.3上传文件

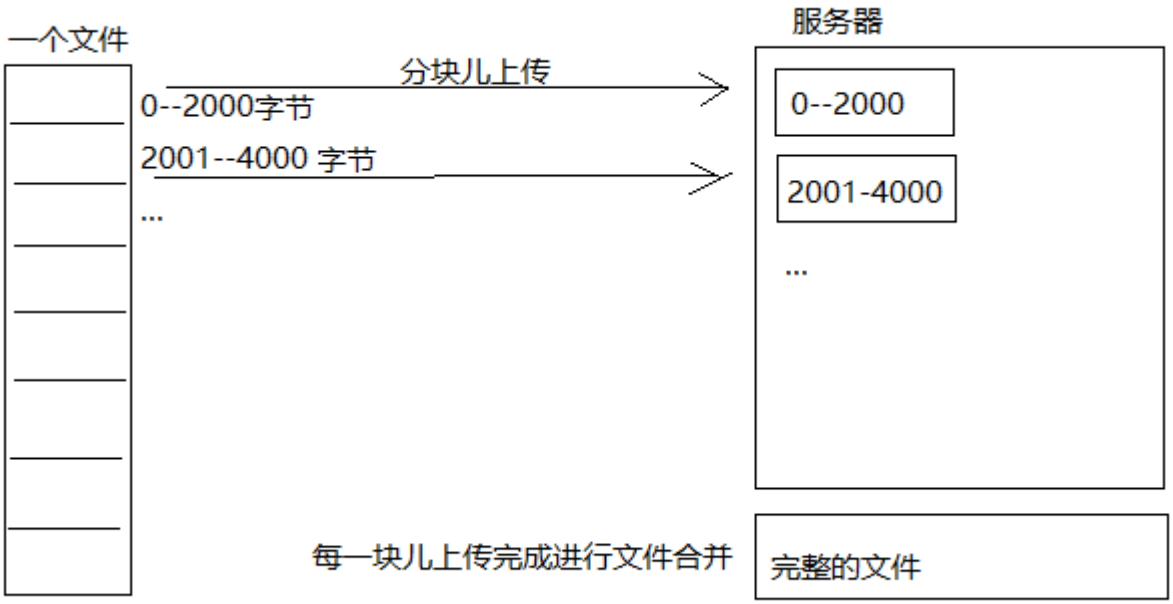
6.3.1 断点续传解决方案

通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。http协议本身对上传文件大小没有限制，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，电断了没有上传完成，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。

什么是断点续传：

引用百度百科：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）人为的划分为几个部分，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下载未完成的部分，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。

如下图：



上传流程如下：

1、上传前先把文件分成块

2、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

3、各分块上传完成最后合并文件

文件下载则同理。

6.3.2 文件分块与合并

为了更好的理解文件分块上传的原理，下边用java代码测试文件的分块与合并。

6.3.2.1文件分块

文件分块的流程如下：

1、获取源文件长度

2、根据设定的分块文件的大小计算出块数

3、从源文件读数据依次向每一个块文件写数据。

```
//测试文件分块方法
    @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();
```

```
        }
        //分块大小
        long chunkSize = 1024*1024*1;
        //分块数量
        long chunkNum = (long) Math.ceil(sourceFile.length() * 1.0 / chunkSize );
        if(chunkNum<=0){
            chunkNum = 1;
        }
        //缓冲区大小
        byte[] b = new byte[1024];
        //使用RandomAccessFile访问文件
        RandomAccessFile raf_read = new RandomAccessFile(sourceFile, "r");
        //分块
        for(int i=0;i<chunkNum;i++){
            //创建分块文件
            File file = new File(chunkPath+i);
            boolean newFile = file.createNewFile();
            if(newFile){
                //向分块文件中写数据
                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();

    }
```

6.3.2.2文件合并

文件合并流程：

1、找到要合并的文件并按文件合并的先后进行排序。

2、创建合并文件

3、依次从合并的文件中读取数据向合并文件写入数

```
//测试文件合并方法
@Test
public void testMerge() throws IOException {
    //块文件目录
    File chunkFolder = new File("F:/develop/ffmpeg/chunk/");
    //合并文件

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

```
    if(mergeFile.exists()){
        mergeFile.delete();
    }
    //创建新的合并文件
    mergeFile.createNewFile();
    //用于写文件
    RandomAccessFile raf_write = new RandomAccessFile(mergeFile, "rw");
    //指针指向文件顶端
    raf_write.seek(0);
    //缓冲区
    byte[] b = new byte[1024];
    //分块列表
    File[] fileArray = chunkFolder.listFiles();
    // 转成集合，便于排序
    List<File> fileList = new ArrayList<File>(Arrays.asList(fileArray));
    // 从小到大排序
    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;
        }
    });
    //合并文件
    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();

}
```

6.3.3 前端页面

上传文件的页面内容参考：“资料”--》upload.vue文件

6.3.3.1 WebUploader介绍

如何在web页面实现断点续传？

常见的方案有：

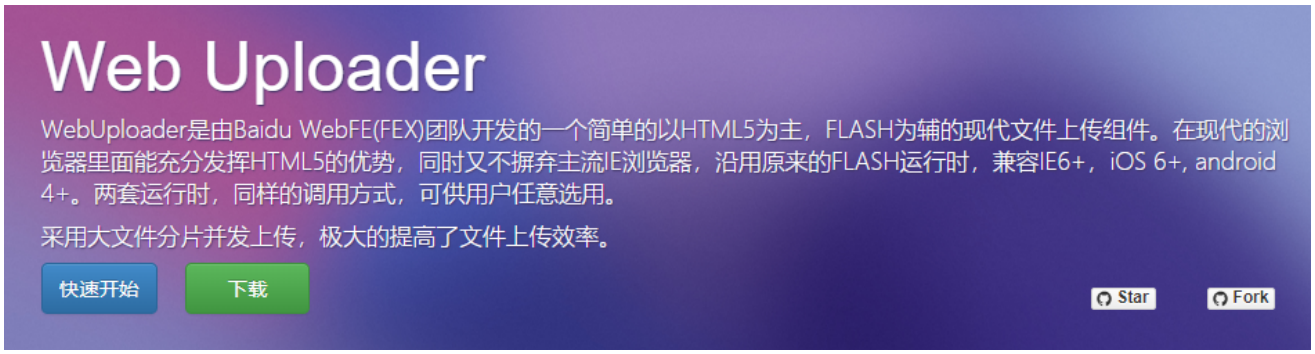
1、通过Flash上传，比如SWFupload、Uploadify。

2、安装浏览器插件，变相的pc客户端，用的比较少。

3、Html5

随着html5的流行，本项目采用Html5完成文件分块上传。

本项目使用WebUploader完成大文件上传功能的开发，WebUploader官网地址：
http://fexteam.gz01.bdysite.com/webuploader/



特性如下：

分片、并发

分片与并发结合，将一个大文件分割成多块，并发上传，极大地提高大文件的上传速度。

当网络问题导致传输错误时，只需要重传出错分片，而不是整个文件。另外分片传输能够更加实时的跟踪上传进度。

预览、压缩

支持常用图片格式jpg,jpeg,gif,bmp,png预览与压缩，节省网络数据传输。

解析jpeg中的meta信息，对于各种orientation做了正确的处理，同时压缩后上传保留图片的所有原始meta数据。

多途径添加文件

支持文件多选，类型过滤，拖拽(文件&文件夹)，图片粘贴功能。

粘贴功能主要体现在当有图片数据在剪切板中时（截屏工具如QQ(Ctrl + ALT + A), 网页中右击图片点击复制），Ctrl + V便可添加此图片文件。

HTML5 & FLASH

兼容主流浏览器，接口一致，实现了两套运行时支持，用户无需关心内部用了什么内核。

同时Flash部分没有做任何UI相关的工作，方便不关心flash的用户扩展和自定义业务需求。

MD5秒传

当文件体积大、量比较多时，支持上传前做文件md5值验证，一致则可直接跳过。

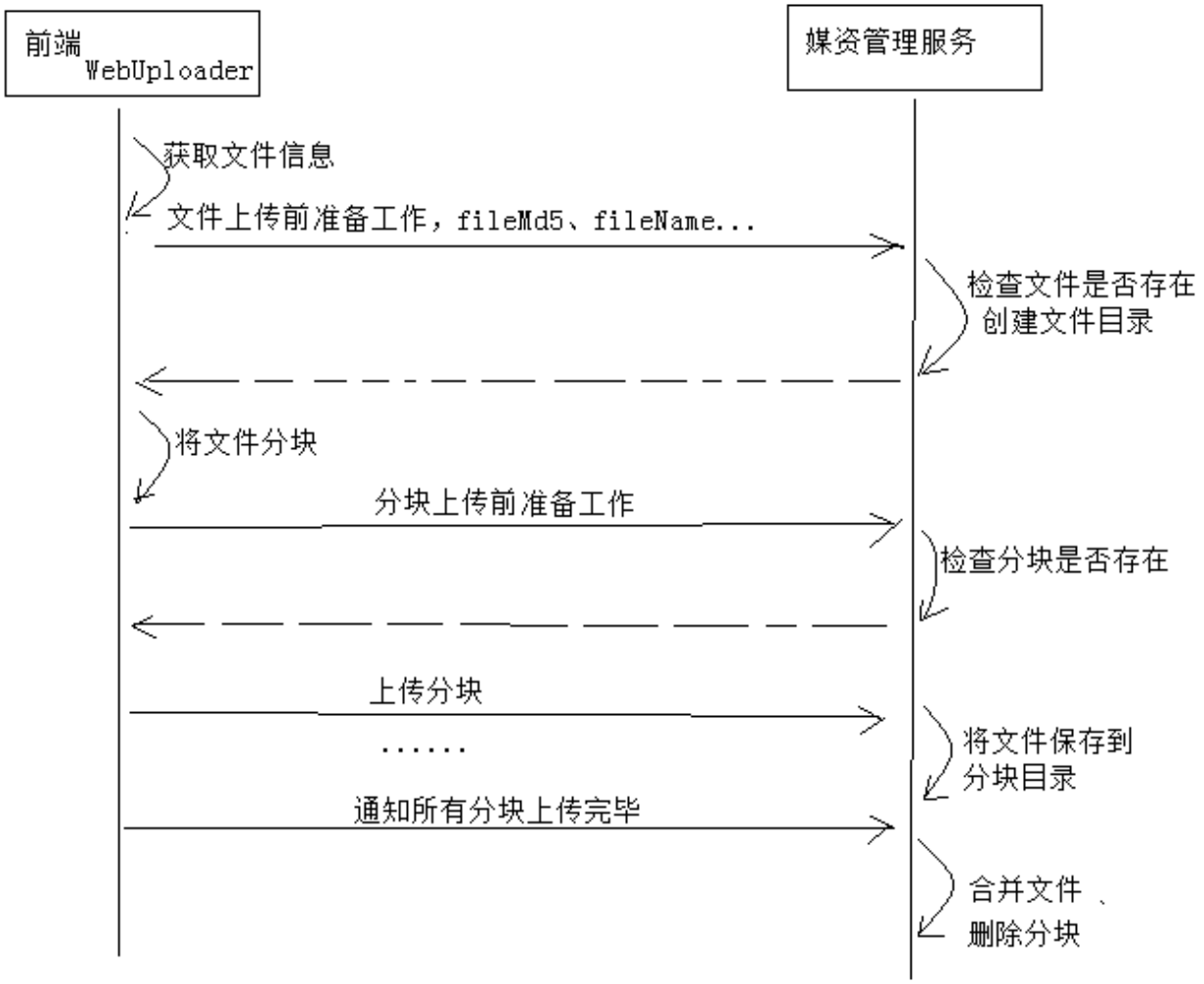
如果服务端与前端统一修改算法，取段md5，可大大提升验证性能，耗时在20ms左右。

易扩展、可拆分

采用可拆分机制，将各个功能独立成了小组件，可自由搭配。

采用AMD规范组织代码，清晰明了，方便高级玩家扩展。

使用WebUploader上传流程如下：



6.3.3.1 钩子方法

在webuploader中提供很多钩子方法，下边列出一些重要的：

名称	参数	说明
add-file	files: File对象或者File数组	用来向队列中添加文件。
before-send-file	file: File对象	在文件发送之前request，此时还没有分片（如果配置了分片的话），可以用来做文件整体md5验证。
before-send	block: 分片对象	在分片发送之前request，可以用来做分片验证，如果此分片已经上传成功了，可返回一个rejected promise来跳过此分片上传
after-send-file	file: File对象	在所有分片都上传完毕后，且没有错误后request，用来做分片验证，此时如果promise被reject，当前文件上传会触发错误。

本项目使用如下钩子方法：

1）before-send-file

在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) before-send

在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。

3) after-send-file

在所有分块上传完成后触发，可以请求服务端合并分块文件。

注册钩子方法源代码：

```
WebUploader.Uploader.register({
    "before-send-file":"beforeSendFile",
    "before-send":"beforeSend",
    "after-send-file":"afterSendFile"
  }
```

6.3.3.2 构建WebUploader

使用webUploader前需要创建webUploader对象。

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

```
// 创建uploader对象，配置参数
this.uploader = WebUploader.create(
  {
    swf:"/static/plugins/webuploader/dist/Uploader.swf",//上传文件的flash文件，浏览器不支持h5时启动
flash
    server:"/api/media/upload/uploadchunk",//上传分块的服务端地址，注意跨域问题
    fileVal:"file",//文件上传域的name
    pick:"#picker",//指定选择文件的按钮容器
    auto:false,//手动触发上传
    disableGlobalDnd:true,//禁掉整个页面的拖拽功能
    chunked:true,// 是否分块上传
    chunkSize:1*1024*1024, // 分块大小（默认5M）
    threads:3, // 开启多个线程（默认3个）
    prepareNextFile:true// 允许在文件传输时提前把下一个文件准备好
  }
)
```

6.3.3.3 before-send-file

文件开始上传前前端请求服务端准备上传工作。

参考源代码如下：

```
type:"POST",
url:"/api/media/upload/register",
data:{
  // 文件唯一表示
  fileMd5:this.fileMd5,
  fileName: file.name,
  fileSize:file.size,
  mimetype:file.type,
  fileExt:file.ext
}
```

6.3.3.4 before-send

上传分块前前端请求服务端校验分块是否存在。

参考源代码如下：

```
type:"POST",
url:"/api/media/upload/checkchunk",
data:{
  // 文件唯一表示
  fileMd5:this.fileMd5,
  // 当前分块下标
  chunk:block.chunk,
  // 当前分块大小
  chunkSize:block.end-block.start
}
```

6.3.3.5 after-send-file

在所有分块上传完成后触发，可以请求服务端合并分块文件

参考代码如下：

```
type:"POST",
url:"/api/media/upload/mergechunks",
data:{
  fileMd5:this.fileMd5,
  fileName: file.name,
  fileSize:file.size,
  mimetype:file.type,
  fileExt:file.ext
}
```

6.3.3.6 页面效果

6.3.4 Api接口

定义文件上传的Api接口，此接收是前端WebUploader调用服务端的接口。

编写此接口需要参数前端WebUploader应用代码。

```
Api(value = "媒资管理接口",description = "媒资管理接口，提供文件上传，文件处理等接口")
public interface MediaUploadControllerApi {

    @ApiOperation("文件上传注册")
    public ResponseResult register(String fileMd5,
                                   String fileName,
                                   Long fileSize,
                                   String mimetype,
                                   String fileExt);

    @ApiOperation("分块检查")
    public CheckChunkResult checkchunk(String fileMd5,
                                       Integer chunk,
                                       Integer chunkSize);
    @ApiOperation("上传分块")
    public ResponseResult uploadchunk(MultipartFile file,
                                      Integer chunk,
                                      String fileMd5);
    @ApiOperation("合并文件")
    public ResponseResult mergechunks(String fileMd5,
                                      String fileName,
                                      Long fileSize,
                                      String mimetype,
                                      String fileExt);
     }
```

6.3.5媒资服务端编写

6.3.5.1 业务流程

服务端需要实现如下功能：

1、上传前检查上传环境

检查文件是否上传，已上传则直接返回。

检查文件上传路径是否存在，不存在则创建。

2、分块检查

检查分块文件是否上传，已上传则返回true。

未上传则检查上传路径是否存在，不存在则创建。

3、分块上传

将分块文件上传到指定的路径。

4、合并分块

将所有分块文件合并为一个文件。

在数据库记录文件信息。

6.3.5.2 上传注册

由于上传过程复杂，开发时按业务流程分别实现。

1、配置

application.yml配置上传文件的路径：

```
xc-service-manage-media:
  upload-location: F:/develop/video/
```

2、定义Dao

媒资文件管理Dao

```
public interface MediaFileRepository extends MongoRepository<MediaFile,String> {

}
```

3、Service

功能：

1）检查上传文件是否存在

2）创建文件目录

```
@Service
public class MediaUploadService {
```

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

    @Autowired
    MediaFileRepository mediaFileRepository;

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

    /**
     * 根据文件md5得到文件路径
     * 规则：
     * 一级目录：md5的第一个字符
     * 二级目录：md5的第二个字符
     * 三级目录：md5
     * 文件名：md5+文件扩展名
     * @param fileMd5 文件md5值
     * @param fileExt 文件扩展名
     * @return 文件路径
     */
    private String getFilePath(String fileMd5,String fileExt){
        String filePath = uploadPath+fileMd5.substring(0, 1) + "/" + fileMd5.substring(1, 2) +
"/" + fileMd5 + "/" + fileMd5 + "." + fileExt;
        return filePath;
    }
    //得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) + "/" +
fileMd5 + "/";
        return filePath;
    }

    //得到文件所在目录
    private String getFileFolderPath(String fileMd5){
        String fileFolderPath = uploadPath+ fileMd5.substring(0, 1) + "/" + fileMd5.substring(1,
2) + "/" + fileMd5 + "/" ;
        return fileFolderPath;
    }

    //创建文件目录
    private boolean createFileFold(String fileMd5){
        //创建上传文件目录
        String fileFolderPath = getFileFolderPath(fileMd5);
        File fileFolder = new File(fileFolderPath);
        if (!fileFolder.exists()) {
            //创建文件夹
            boolean mkdirs = fileFolder.mkdirs();
            return mkdirs;
        }
        return true;
    }
```

```
    //文件上传注册
    public ResponseResult register(String fileMd5, String fileName, String fileSize, String
mimetype, String fileExt) {
        //检查文件是否上传
        //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);
        //文件存在直接返回
        if(file.exists() && optional.isPresent()){
            ExceptionCast.cast(MediaCode.UPLOAD_FILE_REGISTER_EXIST);
        }
        boolean fileFold = createFileFold(fileMd5);
        if(!fileFold){
            //上传文件目录创建失败
            ExceptionCast.cast(MediaCode.UPLOAD_FILE_REGISTER_CREATEFOLDER_FAIL);
        }
        return new ResponseResult(CommonCode.SUCCESS);
    }



}
```

6.3.5.3 分块检查

在Service 中定义分块检查方法：

```
//得到块文件所在目录
private String getChunkFileFolderPath(String fileMd5){
     String fileChunkFolderPath = getFileFolderPath(fileMd5) +"/" + "chunks" + "/";
     return fileChunkFolderPath;
}
//检查块文件
public CheckChunkResult checkchunk(String fileMd5, String chunk, String chunkSize) {
    //得到块文件所在路径
    String chunkfileFolderPath = getChunkFileFolderPath(fileMd5);
    //块文件的文件名称以1,2,3..序号命名，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);
    }
}
```

6.3.5.4 上传分块

在Service 中定义分块上传分块方法：

```
//块文件上传
public ResponseResult uploadchunk(MultipartFile file, String fileMd5, String chunk) {
    if(file == null){
        ExceptionCast.cast(MediaCode.UPLOAD_FILE_REGISTER_ISNULL);
    }
    //创建块文件目录
    boolean fileFold = createChunkFileFolder(fileMd5);
    //块文件
    File chunkfile = new File(getChunkFileFolderPath(fileMd5) + chunk);
    //上传的块文件
    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);
}
    //创建块文件目录
    private boolean createChunkFileFolder(String fileMd5){
        //创建上传文件目录
        String chunkFileFolderPath = getChunkFileFolderPath(fileMd5);
        File chunkFileFolder = new File(chunkFileFolderPath);
        if (!chunkFileFolder.exists()) {
            //创建文件夹
            boolean mkdirs = chunkFileFolder.mkdirs();
            return mkdirs;
        }
        return true;
    }
```

6.3.5.5 合并分块

在Service 中定义分块合并分块方法，功能如下：

1）将块文件合并

2）校验文件md5是否正确

3）向Mongodb写入文件信息

//合并块文件

```
public ResponseResult mergechunks(String fileMd5, String fileName, Long fileSize, String
mimetype, String fileExt) {
    //获取块文件的路径
    String chunkfileFolderPath = getChunkFileFolderPath(fileMd5);
    File chunkfileFolder = new File(chunkfileFolderPath);
    if(!chunkfileFolder.exists()){
        chunkfileFolder.mkdirs();
    }
    //合并文件路径
    File mergeFile = new File(getFilePath(fileMd5,fileExt));
    //创建合并文件
    //合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);
    }
    //获取块文件，此列表是已经排好序的列表
    List<File> chunkFiles = getChunkFiles(chunkfileFolder);
    //合并文件
    mergeFile = mergeFile(mergeFile, chunkFiles);
    if(mergeFile == null){
        ExceptionCast.cast(MediaCode.MERGE_FILE_FAIL);
    }
    //校验文件
    boolean checkResult = this.checkFileMd5(mergeFile, fileMd5);
    if(!checkResult){
        ExceptionCast.cast(MediaCode.MERGE_FILE_CHECKFAIL);
    }
    //将文件信息保存到数据库
    MediaFile mediaFile = new MediaFile();
    mediaFile.setFileId(fileMd5);
    mediaFile.setFileName(fileMd5+"."+fileExt);
    mediaFile.setFileOriginalName(fileName);
    //文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);
```

```
    //状态为上传成功
    mediaFile.setFileStatus("301002");
    MediaFile save = mediaFileDao.save(mediaFile);

    return new ResponseResult(CommonCode.SUCCESS);
}
 //校验文件的md5值
    private boolean checkFileMd5(File mergeFile,String md5){
        if(mergeFile == null || StringUtils.isEmpty(md5)){
            return false;
        }
        //进行md5校验
        FileInputStream mergeFileInputstream = null;
        try {
            mergeFileInputstream = new FileInputStream(mergeFile);
            //得到文件的md5
            String  mergeFileMd5 = DigestUtils.md5Hex(mergeFileInputstream);
            //比较md5
            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;

    }


//获取所有块文件
    private List<File> getChunkFiles(File chunkfileFolder){
        //获取路径下的所有块文件
        File[] chunkFiles = chunkfileFolder.listFiles();
        //将文件数组转成list，并排序
        List<File> chunkFileList = new ArrayList<File>();
        chunkFileList.addAll(Arrays.asList(chunkFiles));
        //排序
        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;
    }
    //合并文件
    private File mergeFile(File mergeFile,List<File> chunkFiles){
        try {
            //创建写文件对象
            RandomAccessFile raf_write = new RandomAccessFile(mergeFile,"rw");
            //遍历分块文件开始合并
            //读取文件缓冲区
            byte[] b = new byte[1024];
            for(File chunkFile:chunkFiles){
                RandomAccessFile raf_read = new RandomAccessFile(chunkFile,"r");
                int len = -1;
                //读取分块文件
                while((len = raf_read.read(b))!=-1){
                    //向合并文件中写数据
                    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;
    }
```

6.3.5.6 Controller

```
@RestController
@RequestMapping("/media/upload")
public 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);
    }
}
```