# 音视频预研

# 一、音视频处理基本梳理

## 1.多媒体文件的理解

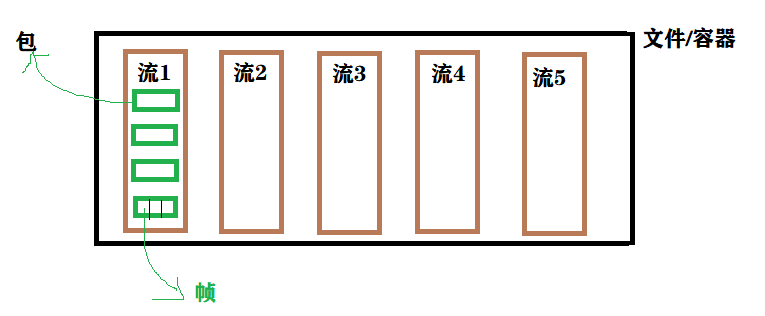


图1 多媒体文件结构图

### 1.1结构分析

* 多媒体文件本质上可以理解为一个容器
* 容器里有很多流
* 每种流是由不同的编码器编码的
* 在众多包中包含多个帧(帧在音视频的处理中是最小单位)

### 1.2 封装格式

封装格式(也叫容器)就是将已经编码压缩好的视频流、音频流及字幕安照一定的方案放到一个文件中，便于播放软件播放。一般来说，视频文件的后缀名就是它的封装格式。封装格式不一样，后缀名也就不一样(xxx.mp4 xxx.flv)。

### 1.3 音视频同步

Audio Master：同步视频到音频

Video Master：同步音频到视频

External Clock Master: 同步音频和视频到外部时钟

### 1.4 音视频录制原理

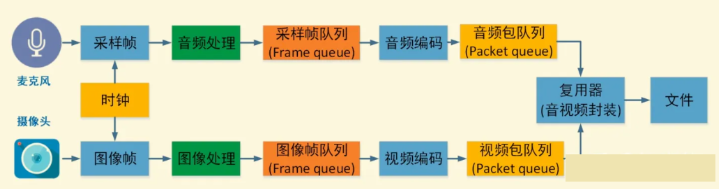


图2 音视频录制原理

### 1.5 音视频播放原理

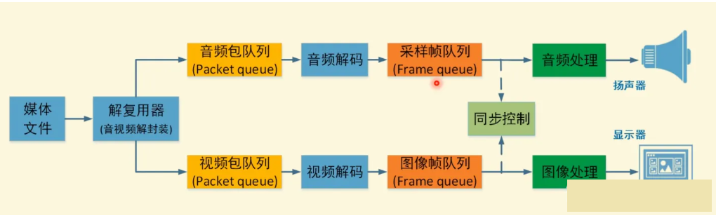


图3 音视频播放原理

## 2.音视频的基础概念

### 2.1声音

对自然界的声音进行采样,采样就是在时间轴上对信号进行数字化信号,即按照一定时间间隔t在模拟信号x(t)上逐点采取其瞬时值。采样率越高，声音的还原程度越高，质量就越好，同时占用空间会变大。

**量化:**用有限个幅度值近似原来连续的幅度值,把模拟信号的连续幅度变为有限数量的有一定间隔的离散值。【采样值的精确度取决于它用多少位来表示,这就是量化。例如8位量化可以表示256个不同值,而CD质量的16位量化可以表示65536值,范围 -32769-32767】

**编码:**按照一定的规律,把量化后的值用二进制数字表示,然后转化成二值或多值的数字信号流。这样得到的数字信号可以通过电缆，卫星通信等数字线路传输。接收端与上述过程相反。

**脉冲编码调制(PCM):**上面数字化的过程又称为脉冲编码调制,通常我们说音频的裸数据格式就是脉冲编码调制数据。描述一段PCM数据需要4量化指标:采样率、位深度、字节序、声道数。

**采样率:**每秒钟采样多少次,以Hz为单位。

|  |  |
| --- | --- |
| 无线广播 | 22000(22kHz) |
| CD音质 | 44100(44.1kHz) |
| 数字电视(DVD) | 48000(48Hz) |
| 蓝光(高清DVD) | 96000(96kHz) |
| 蓝光(高清DVD) | 192000(192kHz) |

表1 常用的采样频率表

**位深度(Bit-depth):**表示用多小个二进制位来描述采样数据,一般位16bit。

**字节序:**表示音频PCM数据存储的字节序是大端存储还是小端存储,为了数据处理效率高效,通常采用小端存储。

**声道数(channel number):**当前PCM文件中包含的声道数是单声道还是双声道

**比特率:**每秒传输的bit数,单位为bps(Bit Per Second)。间接衡量声音质量的一个标准。没有压缩的音频数据的比特率=采样频率\*采样精度\*通道数。

**码率：**压缩后的音频数据的比特率。码率越大，压缩效率越低，音质越好，压缩后数据越大。码率=音频文件大小/时长。

|  |  |
| --- | --- |
| FM质量 | 96kps |
| 一般质量音频 | 128-160kbps |
| CD质量 | 192kbps |
| 高质量音频 | 256-320kbps |

表2 常见码率

**帧:**每次编码的采样单元数。比如MP3通常是1152个采样点作为一个编码单元，AAC通常是1024个采样点作为一个编码单元。

**帧长:**可以指每帧播放的持续时间。每帧持续时间(秒)=每帧采样点数/采样频率(HZ)。也可以指压缩后每帧的数据长度。

**音频编码:**主要作用是将音频采样数据（PCM等）压缩成为音频码流，从而降低音频的数据量,偏于存储和传输。

|  |  |
| --- | --- |
| MP3 | 一种数字音频编码和有损压缩格式，用于大幅度降低音频数量。 |
| AAC | AAC比MP3有更高的压缩比,同样大小的音频文件,AAC音质更高。 |
| WMA | 本身包含有损和无损压缩格式 |

表3 常见音频编码

### 2.2图像

图像是客观对象的一种相似性的、生动性的描述或写真，是人类社会活动中最常用的信息载体。或者说图像是客观对象的一种表示，它包含了被描述对象的有关信息。它是人们最主要的信息源。

**像素:**屏幕显示是把有效面积化为很多个小格子,每个格子只显示一种颜色,是成像的最小元素,因此就叫做"像素"。

**分辨率:**屏幕在长度和宽度这两个方向上各有多少个像素,就叫做分辨率,一般用A x B来表示。分辨率越高,每个像素的面积越小,显示效果就越平滑细腻。

**RGB表示图像:**8bit表示一个子像素: 取值范围[0~255] 或者 [00~FF]。例如图像格式RGBA\_8888,表示4个8bit表示一个像素,而RGB\_565用5+6+5bit表示一个像素。一张1280\*720的RGBA\_8888格式的图片大小=1280 x 720 x 32bit。所以每一张图像的裸数据都是很大的。一部90分钟的电影,没秒25帧: 90 \* 60 \* 25 \* 1280 \* 720 \* 32 bit = 463.48G。

**YUV表示图像:**YUV，是另外一种颜色编码方法，视频的裸数据一般使用 YUV 数据格式表示。Y 表示明亮度，也称灰度 值（灰阶值）。UY 表示色度，均表示影响的色彩和饱和度，用于指定像素的颜色。

**亮度:**需要透过 RGB 输入信号建立，方式为将 RGB 信号的特定部分（g 分量信号）叠加到一起。

**色度：**定义了颜色的色调和饱和度，分别用 Cr、Cb 表示，(C 代表分量(是 component 的缩写))。Cr 反映 RGB 输入信号红色部分与 RGB 信号亮度值之间的差异。Cb 反映 RGB 输入信号蓝色部分与 RGB 信号亮度值之间的差异。

### 2.3视频

由于人类眼睛的特殊结构，画面快速切换时，画面会有残留（ 视觉暂留），感觉起来就是连贯的动作。所以 ，视频就是由一系列图片构成的。

**视频码率:**指视频文件在单位时间内使用的数据流量，也叫码流率。码率越大，说明单位时间内取样率越大，数据流精度就越高。

**视频帧率:**通常说一个视频的25帧，指的就是这个视频帧率，即1秒中会显示25帧。帧率越高，给人的视觉就越流畅。

**视频分辨率：**分辨率就是我们常说的640x480分辨率、1920x1080分辨率，分辨率影响视频图像的大小。

**帧:**帧不需要参考其他画面而生成，解码时仅靠自己就重构完整图像。

**视频的编码：**编码的目的就是为了压缩， 让各种视频的体积变得更小，有利于存储和传输。国际上主流制定视频编解码技术的组织有两个，一个是“国际电联（ITU-T）”，它制定的标准 有 H.261、H.263、H.263+、H.264 等，另一个是“国际标准化组织（ISO）”它制定的标准有 MPEG-1、MPEG-2、 MPEG-4 等。

|  |  |
| --- | --- |
| WMV | 微软推出的一种流媒体格式，它是在“同门”的 ASF 格式升级延伸来得。在同等视频质量下，WMV 格式的文件可以边下载边播放，因此很适合在网上播放和传输。 |
| VP8 | 来自 On2 的 WebM, VPX(VP6,VP7,VP8,VP9)，这个编码设计用于 web 视频。 |
| WebRTC | 在 2010 年 5 月，Google 以大约 6820 万美元收购了 VoIP 软件开发商 Global IP Solutions 公司，并因此获得了该公司拥有的 WebRTC 技术。WebRTC 集成 VP8, VP9。 |
| AV1 | 是一个开放，免专利的视频编码格式，针对互联网传输视频而设计。 |
| AVS | 是中国具备自主知识产权的第二代信源编码标准，是《信息技术先进音视频编码》系列标准的简称， 其包括系统、视频、音频、数字版权管理等四个主要技术标准和符合性测试等支撑标准。 |
| H265 | 与 H.264 编解码器相比，HEVC 在压缩方面提供了重大的改进。HEVC 压缩视频的效率比 H.264 要高出两倍。使用 HEVC，相同视觉质量的视频只占用一半的空间。 |
| VP9 | 是由 Google 开发的开放式、无版权费的视频编码标准，VP9 也被视为是 VP8 的下一代视频编码标准。 |

表4 常见的视频编码

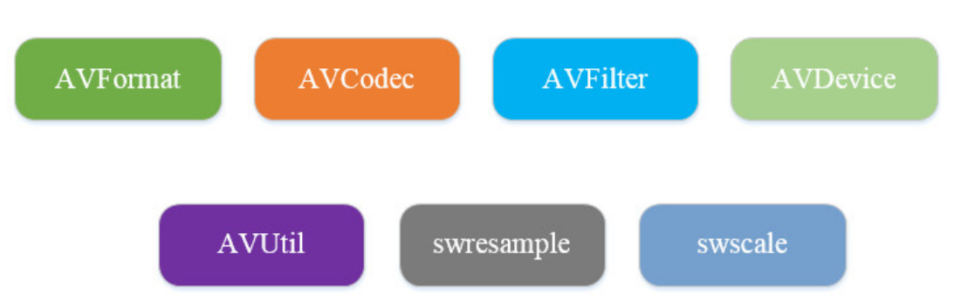
## 3.常用的音视频处理第三方库

### 3.1 FFmpeg

#### 3.2.1基本概念

FFmpeg(Fast Forward MPEG)是全球领先的多媒体框架，能够解码(decode)、编码(encode)、转码(transcode)、复用(mux)、解复用(demux)、流化(stream)、滤波(filter)和播放几乎人类和机器创造的所有多媒体文件。

#### 3.2.2 FFmpeg 的主要基本组成



* **FFmpeg的封装模块AVFormat:**AVFormat实现了多媒体领域绝大数媒体封装格式，包括封装和解封装，如MP4、FLV、KV、 TS等文件封装格式，RTMP、RTSP、MMS、HLS等网络协议封装格式。FFmepg是否支持某种媒体封装格式取决于编译时是否包含了该格式的封装库。
* **FFmpeg的编解码模块AVCodec:**AVCodec包括大多数常用的编解码格式，既支持编码也支持解码。除了支持MPEG4、AAC、MJPEG等自带的媒体格式也支持H.264（x264编码器）、H.265（X265编码器）、MP3（libMP3lame编码器）。
* **FFmepg的滤镜模块AVFilter:**AVFilter库提供了一个通用的音频、视频、字幕等滤镜处理框架。在AVFilter中，滤镜框架可以有多个输入和多个输出。
* **FFmpeg的视频图像转换计算模块swscale:**swscale模块提供了高级别的图像转换API，它能够对图像进行缩放和像素格式转换。
* **FFmpeg的音频转换计算模块swresample:**swresample提供了音频重采样API，支持音频采样、音频通道布局、布局调整。

#### 3.2.3 FFmpeg 的优劣

* 高可移植性:可以在Linux、Mac、Windows等系统上编译、运行以及通过FATE(FFMPEG自动化测试环境)测试。
* 高性能:专门针对X86、arm、MIPS、ppc等大多数主流的处理器提供了对应的汇编级的优化实现。
* 高度安全: FFMPEG官方对代码审查总是考虑安全性，而且一旦发布的版本中有安全性的Bug都会尽快的修复并更新发布版本。
* 高度易用性:FFMPEG提供的API都有相关的注释，且官方也有对应的说明文档
* 支持的格式多样性:FFMPEG支持很多媒体格式的解码、编码、复用、解复用等功能，不管是很老的格式，还是比较新的格式均有不错的支持
* 无法识别有空格的文件名
* FFMPEG编码时,时间戳只需要指定AVFrame的pts字段

#### 3.2.4 FFmpeg的安装配置

|  |
| --- |
| **Windows** |
| **Linux**  **方法1(yum安装) **不过这个版本有点旧****   * **更新新系统**   **Sudo yum install epel-release -y**  **Sudo yum update -y**   * 导入密钥并设置源   **sudo rpm --import http://li.nux.ro/download/nux/RPM-GPG-KEY-nux.ro**  **sudo rpm -Uvh http://li.nux.ro/download/nux/dextop/el7/x86\_64/nux-dextop-release-0-5.el7.nux.noarch.rpm**   1. 安装ffmpeg   yum -y install ffmpeg ffmpeg-devel   1. 检查版本   ffmpeg -version   **方法 2（编译安装）** 先下载源码包：  git clone https://git.ffmpeg.org/ffmpeg.git ffmpeg  进入ffmpeg文件夹，依次执行下列语句:  cd ffmpeg  ./configure  make  make install  将编译好的ffmpeg复制到bin目录  cp ffmpeg /usr/bin/ffmpeg  检查版本  ffmpeg -version |

#### 3.2.5 FFmpeg的命令行使用

|  |
| --- |
| * 获取媒体文件信息   ffmpeg -i 文件全路径 -hide\_banner  示例: ffmpeg -i video\_file.mp4 -hide\_banner  注: -hide\_banner 隐藏ffmpeg本身的信息，只显示文件相关信息(编码器、数据流等)。   * 转换媒体文件(可以实现在不同媒体格式之间进行自由转换)   注:ffmpeg会从后缀名猜测格式  ffmpeg -i 待转换的文件路径 转换后的文件路径  示例: ffmpeg -i video\_input.mp4 video\_output.avi  ffmpeg -i video\_input.webm video\_output.flv  ffmpeg -i audio\_input.mp3 video\_output.ogg  ffmpeg -i audio\_input.wav audio\_output.flac video\_output.flv  参数: -qscale 0 保留原始的视频质量   * 从视频中提取音频   ffmpeg -i 视频全路径 -vn 需要保存的音频文件全路径 -hide\_banner  参数说明:  -vn 从视频中提取音频  -ab 指定编码比特率(一些常见的比特率 96k、128k、192k、256k、320k)  -ar 采样率(22050、441000、48000)  -ac 声道数  -f 音频格式(通常会自动识别)   * 让视频静音-(纯视频)   ffmpeg -i video\_input.mp4 -an -video\_output.mp4  注: -an 标记会让所有视频的音频参数无效,因为最后没有音频产生   * 视频文件中提取截图   ffmpeg -i 视频文件名 -r 帧率 -f 输出格式 输出文件名  示例: ffmpeg -i video.mp4 -r 1 -f image2 image-%3d.png  参数说明:  -r 帧率(一秒内导出多少张图像，默认25)  -f 代表输出格式(image2实际上是image2序列的意思)   * 更改视频分辨率或长宽比   ffmpeg -i 视频文件名 -s 分辩率 -c:a -aspect 长:宽 输出文件名  示例:ffmpeg -i video\_input.mov -s 1024x576 -c:a video\_output.mp4  参数说明:  -s 缩放视频  -c:a 保证音频编码正确  -aspect 更改长宽比   * 为音频增加封面(音频转视频)   当你想往某一个网站上传音频，但那个网站只接受视频的情况下非常适用  示例：ffmpeg -loop 1 -i image.jpg -i audio.wav -c:v libx264 -c:a acc -strict experimental -b:a 192k -shortest output.mp4  -c:v 视频编码  -c:a 音频编码  注：如果是4.x版本以上，不需要加 -strict experimental   * 为视频增加字幕   ffmpeg -i video.mp4 -i subtitles.srt -c:v copy -c:a copy -preset veryfast -c:s mov\_text -map 0 -map 1 output.mp4 |

# 二、视频中提取音频

### 1.FFmpeg

|  |
| --- |
| **通过命令行**  ffmpeg -i 视频文件路径 -vn 音频文件全路径 -hide\_banner  参数说明:  -vn 从视频中提取音频  -ab 指定编码比特率(一些常见的比特率 96k、128k、192k、256k、320k)  -ar 采样率(22050、441000、48000)  -ac 声道数  -f 音频格式(通常会自动识别)  示例:  视频中提取音频演示 |

|  |
| --- |
| 通过提供的API  bool AVInterface::extractAudio(const char\* src, const char\* dstDir)  {      if (NULL == src || NULL == dstDir)      {          printf("Ffmpeg::extractAudio[ERROR]::无效参数,请检查文件路径是否正确\n");          return false;      }      int ret = 0;      // 预存原文件路径      const char\* src\_fileName = src;      // 1.获取媒体文件的全局上下文信息      // 1.1 定义 AVFormatContext 容器      AVFormatContext\* pFormatCtx = NULL;      // AVFormatContext描述了一个媒体文件或者媒体流构成的基本信息      pFormatCtx = avformat\_alloc\_context();   // 为 pFormatCtx 申请内存      // 1.2 打开媒体文件,并且读取媒体文件的头信息放入pFormatCtx中      ret = avformat\_open\_input(&pFormatCtx, src\_fileName, NULL, NULL);      if (ret < 0)      {          printf("Ffmpeg::extractAudio[ERROR]::打开媒体流文件失败\n");          return false;      }      // 2.探测流出信息      // 2.1 探寻文件中是否存在信息流,如果存在则将多媒体文件信息流放到pFormatCtx      ret = avformat\_find\_stream\_info(pFormatCtx, NULL);      if (ret < 0)      {          printf("Ffmpeg::extractAudio[ERROR]::文件中不存在信息流\n");          return false;      }      av\_dump\_format(pFormatCtx, 0, src\_fileName, 0);    // 打印封装格式和流信息      // 2.2 查找文件信息流中是否存在音频流(我们只需要提取音频),并获取到音频流在信息流中的索引      int audio\_stream\_index = -1;      audio\_stream\_index = av\_find\_best\_stream(pFormatCtx, AVMEDIA\_TYPE\_AUDIO, -1, -1, NULL, 0);      if (-1 == audio\_stream\_index)      {          printf("Ffmpeg::extractAudio[ERROR]::文件中不存在音频流\n");          return false;      }      // 3.输出容器的定义      AVFormatContext\* pFormatCtx\_out = NULL;    // 输出格式的上下文信息      const AVOutputFormat\*  pFormatOut = NULL;        // 输出的封装格式      AVPacket packet;      // 输出文件路径      char szFilename[256] = { 0 };      snprintf(szFilename, sizeof(szFilename), "%s/ffmpeg-music.aac", dstDir);      // 3.1 初始化容器      // 初始化一些基础的信息      av\_init\_packet(&packet);      // 给 pFormatCtx\_out 动态分配内存,并且会根据文件名初始化一些基础信息      avformat\_alloc\_output\_context2(&pFormatCtx\_out, NULL, NULL, szFilename);      // 得到封装格式 AAC      pFormatOut = pFormatCtx\_out->oformat;      // 4.读取音频流,并且将输入流的格式拷贝到输出流的格式中        for (int i = 0; i < pFormatCtx->nb\_streams; ++i)   // nb\_streams 流的个数      {            // 流的结构体,封存了一些流相关的信息          AVStream\* out\_stream = NULL;               // 输出流          AVStream\* in\_stream  = pFormatCtx->streams[i];             // 输入流          AVCodecParameters\* in\_codeper = in\_stream->codecpar;   // 编解码器            // 只取音频流          if (in\_codeper->codec\_type == AVMEDIA\_TYPE\_AUDIO)          {              // 建立输出流              out\_stream = avformat\_new\_stream(pFormatCtx\_out, NULL);              if (NULL == out\_stream)              {                  printf("Ffmpeg::extractAudio::[ERROR]建立输出流失败\n");                  return false;              }              // 拷贝编码参数,如果需要转码请不要直接拷贝              // 这里只需要做音频的提取,对转码要求不高              ret = avcodec\_parameters\_copy(out\_stream->codecpar, in\_codeper); // 将输入流的编码拷贝到输出流              if (ret < 0)              {                  printf("Ffmpeg::extractAudio::[ERROR]拷贝编码失败\n");                  return false;              }              out\_stream->codecpar->codec\_tag = 0;              break;  // 拿到音频流就可以直接退出循环,这里我们只需要音频流          }      }      av\_dump\_format(pFormatCtx\_out, 0, szFilename, 1);      // 解复用器,如果没有指定就使用pb      if (!(pFormatCtx->flags & AVFMT\_NOFILE))      {          ret = avio\_open(&pFormatCtx\_out->pb, szFilename, AVIO\_FLAG\_WRITE); // 读写          if (ret < 0)          {              printf("Ffmpeg::extractAudio::[ERROR]创建AVIOContext对象：打开文件失败\n");              return false;          }      }      // 写入媒体文件头部      ret = avformat\_write\_header(pFormatCtx\_out, NULL);      if (ret < 0)      {          printf("Ffmpeg::extractAudio::[ERROR]写入媒体头部失败\n");          return false;      }      // 逐帧提取音频      AVPacket\* pkt = av\_packet\_alloc();      while (av\_read\_frame(pFormatCtx, &packet) >=0 )      {          AVStream\* in\_stream  = NULL;          AVStream\* out\_stream = NULL;          in\_stream = pFormatCtx->streams[pkt->stream\_index];          out\_stream = pFormatCtx\_out->streams[pkt->stream\_index];          if (packet.stream\_index == audio\_stream\_index)          {              packet.pts = av\_rescale\_q\_rnd(packet.pts, in\_stream->time\_base, out\_stream->time\_base, (AVRounding)(AV\_ROUND\_INF|AV\_ROUND\_PASS\_MINMAX));              packet.dts = packet.pts;              packet.duration = av\_rescale\_q(packet.duration, in\_stream->time\_base, out\_stream->time\_base);              packet.pos = -1;              packet.stream\_index = 0;              // 将包写到输出媒体文件              av\_interleaved\_write\_frame(pFormatCtx\_out, &packet);              // 减少引用计数,防止造成内存泄漏              av\_packet\_unref(&packet);          }      }      // 写入尾部信息      av\_write\_trailer(pFormatCtx\_out);      // 释放      av\_packet\_free(&pkt);      avio\_close(pFormatCtx\_out->pb);      avformat\_close\_input(&pFormatCtx);        return true;  } |

### 性能对比

|  |  |  |
| --- | --- | --- |
| 5s | 5min | 30min |
| 0.087017s | 0.138014s | 0.875926s |

表5 ffmpeg提取音频性能情况

# 三、视频文件中提取图片

### 1.FFmpeg

|  |
| --- |
| 通过命令行  ffmpeg -i 视频文件名 -r 帧率 -f 输出格式 输出文件名  示例: ffmpeg -i video.mp4 -r 1 -f image2 image-%3d.png  参数说明:  -r 帧率(一秒内导出多少张图像，默认25)  -f 代表输出格式(image2实际上是image2序列的意思)  示例:  视频中提取图片演示 |
| 通过提供的API  bool AVInterface::extracPictrue(const char\* src, const char\* dstDir)  {      if(NULL == src || NULL == dstDir)      {          printf("Ffmpeg::extracPictrue[ERROR]::无效参数,请检查文件路径是否正确\n");          return false;      }      int ret = 0;        // 预存原文件路径      const char\* src\_fileName = src;        // 1.获取媒体文件的全局上下文信息      // 1.1 定义 AVFormatContext 容器      AVFormatContext\* pFormatCtx = NULL;       // AVFormatContext描述了一个媒体文件或者媒体流构成的基本信息      pFormatCtx = avformat\_alloc\_context();    // 为pFormatCtx申请内存      // 1.2 打开媒体文件,并且读取媒体文件的头信息放入pFormatCtx中      ret = avformat\_open\_input(&pFormatCtx, src\_fileName, NULL, NULL);      if(ret < 0)      {          printf("Ffmpeg::extracPictrue[ERROR]::打开媒体流文件失败\n");          return false;      }      // 2.探测流信息      // 2.1 探寻文件中是否存在信息流,如果存在则将多媒体文件信息流放到pFormatCtx中      ret = avformat\_find\_stream\_info(pFormatCtx, NULL);      if(ret < 0)      {          printf("Ffmpeg::extracPictrue[ERROR]::文件中不存在信息流\n");          return false;      }      av\_dump\_format(pFormatCtx, 0, src\_fileName, 0);      // 可以打印查看      // 2.2 查找文件信息流中是否存在视频流(这里我们需要提取图片),并获取到视频流在信息流中的索引      int vecdio\_stream\_index = -1;      vecdio\_stream\_index = av\_find\_best\_stream(pFormatCtx, AVMEDIA\_TYPE\_VIDEO, -1, -1, NULL, 0);      if(-1 == vecdio\_stream\_index)      {          printf("Ffmpeg::extracPictrue[ERROR]::文件中不存在视频流\n");          return false;      }   // ----------> 丛林方法1      // 3.找到对应的解码器:音视频文件是压缩之后的,我们要对文件内容进行处理,就必须先解码        // 3.1 定义解码器的容器      AVCodecContext\* pCodeCtx = NULL;          // AVCodecContext描述编解码器的结构,包含了众多解码器的基本信息      const AVCodec\* pCodec = NULL;                   // AVCodec 存储解码器的信息        pCodeCtx = avcodec\_alloc\_context3(NULL);  // 初始化解码器上下文      // 3.2 查找解码器      AVStream\* pStream = pFormatCtx->streams[vecdio\_stream\_index]; // 在众多解码器找到视频处理的上下文信息      pCodec = avcodec\_find\_decoder(pStream->codecpar->codec\_id);          // 根据视频流获取视频解码器的基本信息      if(NULL == pCodec)      {          printf("未发现视频编码器\n");          return false;      }      // 初始化解码器上下文      ret = avcodec\_parameters\_to\_context(pCodeCtx, pStream->codecpar);      if (ret < 0)      {          printf("初始化解码器上下文失败\n");          return false;      }      // 3.3 打开解码器      ret = avcodec\_open2(pCodeCtx, pCodec, NULL);      if(ret < 0)      {          printf("无法打开编解码\n");          return false;      }      AVFrame\* pFrame = NULL;      pFrame = av\_frame\_alloc();      if (NULL == pFrame)      {          printf("av\_frame\_alloc is error\n");          return false;      }      int index = 0;      AVPacket avpkt;      while (av\_read\_frame(pFormatCtx, &avpkt) >= 0)      {          if (avpkt.stream\_index == vecdio\_stream\_index)          {              ret = avcodec\_send\_packet(pCodeCtx, &avpkt);              if (ret < 0)              {                  continue;              }              while (avcodec\_receive\_frame(pCodeCtx, pFrame) == 0)              {                  SaveFramePicture(pFrame, dstDir, index);              }              index++;              if (index == 100)              {                  break;              }          }          av\_packet\_unref(&avpkt);      }      avcodec\_close(pCodeCtx);      avformat\_close\_input(&pFormatCtx);      return true;  }  bool AVInterface::SaveFramePicture(AVFrame\* pFrame, const char\* dstDir, int index)  {      char szFilename[256] = {0};      snprintf(szFilename, sizeof(szFilename), "%s/ffmpeg-%d.png", dstDir, index);      int ret = 0;      int width  = pFrame->width;      int height = pFrame->height;      // 1.初始化图片封装格式的结构体      AVCodecContext\*  pCodeCtx = NULL;      AVFormatContext\* pFormatCtx = NULL;      pFormatCtx = avformat\_alloc\_context();        // 2.设置封装格式      // MJPEG格式:按照25帧/秒速度使用JPEG算法压缩视频信号,完成动态视频的压缩 --> 视频文件使用MJPEG进行解压      pFormatCtx->oformat = av\_guess\_format("mjpeg", NULL, NULL);  // 用于从已经注册的输出格式中寻找最匹配的输出格式      // 3.创建AVIOContext对象：打开文件      ret = avio\_open(&pFormatCtx->pb, szFilename, AVIO\_FLAG\_READ\_WRITE); // 读写方式      if(ret < 0)      {          printf("avio\_open is error");          return false;      }      // 构建一个新的stream      AVStream\* pAVStream = NULL;      pAVStream = avformat\_new\_stream(pFormatCtx, 0);      if(pAVStream == NULL)      {          printf("avformat\_new\_stream\n");          return false;      }        AVCodecParameters\* parameters = NULL;                    // 编码器参数的结构体      parameters = pAVStream->codecpar;                        // 设置编码器 mjpeg      parameters->codec\_id = pFormatCtx->oformat->video\_codec; // 视频流      parameters->codec\_type = AVMEDIA\_TYPE\_VIDEO;             // 编码类型      //parameters->format = AV\_PIX\_FMT\_BGR24;                 // 指定图片的显示样式      parameters->format = AV\_PIX\_FMT\_YUVJ420P;                // YUV 解压缩显示样式都是YUV      parameters->width  = pFrame->width;                      // 指定图片的宽度      parameters->height = pFrame->height;                     // 显示图片的高度        // 找到相应的解码器      const AVCodec\* pCodec = avcodec\_find\_encoder(pAVStream->codecpar->codec\_id);      if(NULL == pCodec)      {          printf("avcodec\_find\_encoder is error\n");          return false;      }      // 初始化解码器上下文      pCodeCtx = avcodec\_alloc\_context3(pCodec);      if(NULL == pCodeCtx)      {          printf("avcodec\_alloc\_context3 is error\n");          return false;      }      // 设置解码器的参数      //ret = avcodec\_parameters\_to\_context(pCodeCtx, pAVStream->codecpar);      ret = avcodec\_parameters\_to\_context(pCodeCtx, parameters);      if(ret < 0)      {          printf("avcodec\_parameters\_to\_context is error\n");          return false;      }      AVRational avrational = {1, 25};      pCodeCtx->time\_base = avrational;      // 打开编解码器      ret = avcodec\_open2(pCodeCtx, pCodec, NULL);      if(ret < 0)      {          printf("avcodec\_open2 is error\n");          return false;      }          // 封装格式的头部信息写入      ret = avformat\_write\_header(pFormatCtx, NULL);      if(ret < 0)      {          printf("avformat\_write\_header is error\n");          return false;      }        // 给AVPacket分配足够大的空间      int y\_size = width \* height;    // 分辨率      AVPacket pkt;      av\_new\_packet(&pkt, y\_size \* 3);      // 编码数据      ret = avcodec\_send\_frame(pCodeCtx, pFrame);      if(ret < 0)      {          printf("avcodec\_send\_frame is error\n");          return false;      }      // 得到解码之后的数据      ret = avcodec\_receive\_packet(pCodeCtx, &pkt);      if(ret < 0)      {          printf("avcodec\_receive\_packet is error\n");          return false;      }      ret = av\_write\_frame(pFormatCtx, &pkt);      if(ret < 0)      {          printf("av\_write\_frame is error\n");          return false;      }      av\_packet\_unref(&pkt);      av\_write\_trailer(pFormatCtx);      avcodec\_close(pCodeCtx);      avio\_close(pFormatCtx->pb);      avformat\_free\_context(pFormatCtx);      return true;  } |

### 3.性能对比

|  |  |  |  |
| --- | --- | --- | --- |
|  | 5s | 5min | 30min |
| 10张 | 0.295322s | 0.146283s | 0.151467s |
| 100张 | 1.263546s | 1.226884s | 1.190490s |
| 全部 | 2.670444s(170) | 96.951886s(7514) | 119.161211s(10000) |

# 四、音频文件中提取文字

## 1.百度智能云语音识别

|  |
| --- |
| 百度语音目前只支持语音识别，语音合成和语音唤醒，支持pcm wav amr三种格式，时长为60秒以内，价格为完全免费，调用量限制为无限制。  1、离线语音识别  百度离线语音识别目前只支持Android和IOS，Android 平台的一体化离在线语音识别解决方案，以JAR包 + SO库的形式发布。IOS移动设备的离在线语音识别解决方案，以静态库方式提供。  2、在线语音识别  通过API格式调用，Android,iOS,C#,Java,Node,PHP,Python,C++语言，其实是API模式，所有开发语言都支持。 |

### 1.1百度智能云的优劣

* 支持普通话，英语，粤语，四川话，普通话远场
* 只支持60秒以内识别
* 所有开发语言都支持
* 百度的linux版离线SDK支持centos 和 ubantu14 16
* 需要注册百度云控制台账号

### 1.2百度智能云安装配置

|  |
| --- |
| **安装必要的依赖，**[curl](https://so.csdn.net/so/search?q=curl&spm=1001.2101.3001.7020" \t "https://blog.csdn.net/ping_devil/article/details/_blank)**（必须带ssl） jsoncpp openssl** #安装libcurl  sudo apt-get install libcurl4-openssl-dev  #安装jsoncpp  sudo apt-get install libjsoncpp-dev   * ****直接使用开发包步骤如下：****  1. 在[官方网站](http://ai.baidu.com/sdk" \t "https://ai.baidu.com/ai-doc/SPEECH/_blank)下载C++ SDK压缩包。[SDK下载\_文字识别SDK\_语音识别SDK-百度AI开放平台 (baidu.com)](https://ai.baidu.com/sdk" \l "asr) 2. 将下载的aip-cpp-sdk-version.zip解压, 其中文件为包含实现代码的头文件。 3. 安装依赖库libcurl（需要支持https） openssl jsoncpp(>1.6.2版本，0.x版本将不被支持) 4. 编译工程时添加 C++11 支持 (gcc/clang 添加编译参数 -std=c++11), 添加第三方库链接参数 lcurl, lcrypto, ljsoncpp。 5. 在源码中include speech.h ，引入压缩包中的头文件以使用aip命名空间下的类和方法。 |

### 1.4百度智能云使用示例

|  |
| --- |
| * 用户可以参考如下代码新建一个client：   #include "speech.h"  // 设置APPID/AK/SK  std::string app\_id = "你的 App ID";  std::string api\_key = "你的 Api key";  std::string secret\_key = "你的 Secret Key";  aip::Speech client(app\_id, api\_key, secret\_key);  在上面代码中，常量APP\_ID在百度云控制台中创建，常量API\_KEY与SECRET\_KEY是在创建完毕应用后，系统分配给用户的，均为字符串，用于标识用户，为访问做签名验证，可在AI服务控制台中的****应用列表****中查看。   * 向远程服务上传整段语音进行识别   void asr(aip::Speech client)  {  // 无可选参数调用接口  std::string file\_content;  aip::get\_file\_content("./assets/voice/16k\_test.pcm", &file\_content);  Json::Value result = client.recognize(file\_content, "pcm", 16000, aip::null);  // 极速版调用函数  // Json::Value result = client.recognize\_pro(file\_content, "pcm", 16000, aip::null);  // 如果需要覆盖或者加入参数  std::map<std::string, std::string> options;  options["dev\_pid"] = "1537";  Json::Value result = client.recognize(file\_content, "pcm", 16000, options);  }  ****返回样例：****  // 成功返回  {  "err\_no": 0,  "err\_msg": "success.",  "corpus\_no": "15984125203285346378",  "sn": "481D633F-73BA-726F-49EF-8659ACCC2F3D",  "result": ["北京天气"]  }  // 失败返回  {  "err\_no": 2000,  "err\_msg": "data empty.",  "sn": null  } |

## SpeechRecognition开源离线语音识别

SpeechRecognition，是google出的，专注于语音向文本的转换。wit 和 apiai 提供了一些超出基本语音识别的内置功能，如识别讲话者意图的自然语言处理功能。

### SpeechRecognition的优/劣

* 满足几种主流语音 API ，灵活性高
* Google Web Speech API 支持硬编码到 SpeechRecognition 库中的默认 API 密钥，无需注册就可使用
* 易用性很高
* python的语音识别库
* 中国的识别效果不是特别好

### 2.2 SpeechRecognition安装配置

1.安装语音识别环境

|  |
| --- |
| pip install SpeechRecognition (pip install -i https://pypi.tuna.tsinghua.edu.cn/simple SpeechRecognition)  yum install python3-devel  yum install pulseaudio-libs-devel  yum install alse-lib-devel  pip install packetSphinx |

1. 配置中文语音识别数据

|  |
| --- |
| 下载地址  [https://sourceforge.net/projects/cmusphinx/files/Acoustic%20and%20Language%20Models/](https://sourceforge.net/projects/cmusphinx/files/Acoustic and Language Models/)  选择  Mandarin->cmusphinx-zh-cn-5.2.tar.gz  安装中文语音包  cd /usr/local/python3.6.8/lib/python3.6/site-packages/speech\_recognition/pocketsphinx-data  tar zxvf cmusphinx-zh-cn-5.2.tar.gz  mv cmusphinx-zh-cn-5.2 zh-cn  cd zh-cn  mv zh\_cn.cd\_cont\_5000 acoustic-model  mv zh\_cn.lm.bin language-model.lm.bin  mv zh\_cn.dic pronounciation-dictionary.dict |

配置环境

|  |
| --- |
| cd /usr/local/python3.6.8/lib/python3.6/site-packages/speech\_recognition/pocketsphinx-data  tar zxvf py36asr.tar.gz  source ./py36asr/bin/activate |

### 2.3 SpeechRecognition使用示例

****语音识别示例：****

|  |
| --- |
| ****[root@localhost pocketsphinx-data]# pwd****  ****/usr/local/python3.6.8/lib/python3.6/site-packages/speech\_recognition/pocketsphinx-data****  ****[root@localhost pocketsphinx-data]# ls****  ****cmusphinx-zh-cn-5.2.tar.gz py36asr test1.py test2.wav zh-cn.tar.gz****  ****en-US py36asr.tar.gz test1.wav zh-cn****  ****[root@localhost pocketsphinx-data]# python3 test1.py****  ****自然 语言 李杰 和 申城 是一 个 多方 面 问题 我们 对 他 可能 也 只是 部分 礼节****  ****[root@localhost pocketsphinx-data]#**** |

****程序示例：****

|  |
| --- |
| ****# -\*- coding: utf-8 -\*-****  ****# /usr/bin/python****  ****import speech\_recognition as sr****  ****r = sr.Recognizer()****  ****test = sr.AudioFile("test1.wav")****  ****with test as source:****  ****audio = r.record(source)****  ****type(audio)****  ****c=r.recognize\_sphinx(audio, language='zh-cn')****  ****print(c)**** |

## 3.FastASR语音识别

这是一个用C++实现ASR推理的项目,它的依赖很少，安装也很简单，推理速度很快。支持的模型是由Google的Transformer模型中优化而来,数据集是开源。Wennetspeech(1000+小时)或阿里私有数据集(60000+小时),所以识别效果有很好,可以媲美许多商用的ASR软件。

* **流式模型:模拟的输入是语音流,并实时返回语音识别的结果,但是准确率会降低些。**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **名称** | **来源** | **数据集** | **模型** | **语言** |
| **conformer\_online** | **paddlespeech** | **WenetSpeech(1000h)** | **conformer\_online\_wenetspeech-zh-16k** | **zh** |

* **非流式模型:每次识别是以句子为单位,所以实时性会差一些,但是准确率会高一些。**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **名称** | **来源** | **数据集** | **模型** | **语言** |
| **paraformer** | **阿里达摩院** | **私有数据集(6000h)** | **Paraformer-large** | **En+zh** |
| **k2\_rnnt2** | **kaldi2** | **WenetSpeech(10000h)** | **Prouned\_transducer\_stateless2** | **zh** |
| **Conformer\_online** | **paddlespeech** | **WenetSpeech(10000h)** | **Conformer\_online\_wenetspeech-zh-16k** | **zh** |

上面提到的这些模型都是基于深度学习框架(paddlepaddle和pytorch)实现的,本身的性能很不错,在个人电脑上运行,也能满足实时性要求(时长为10s的语言,推理视觉小于10s,即可满足实时性)。

### FastASR的优/劣

* 语言优势:由于C++和Python不同,是编译型语言，编译器会根据编译选项针对不同的平台的CPU进行优化,更合适在不同CPU平台上面部署,充分利用CPU的计算资源。
* 实现独立:不依赖于现有的深度学习框架如pytorch、paddle、tensorflow等
* 依赖少:项目仅使用了两个第三方libfftw和libopenblas，并无其它依赖，所以在各个平台的可以移植性很好,通用性很强。



* 缺少量化和压缩模型
* 支持C++ 和python

### 3.3 FastASR安装配置

1. 依赖安装库 libfftw3

|  |
| --- |
| sudo apt-get install libfftw3-dev libfftw3-single3 |

1. 安装依赖库 libopenblas

|  |
| --- |
| sudo apt-get install libopenblas-dev |

1. 安装python环境

|  |
| --- |
| sudo apt-get install python3 python3-dev |

1. 下载最新版的源码

|  |
| --- |
| git clone https://github.com/chenkui164/FastASR.git |

1. 编译最新版本的源码

|  |
| --- |
| cd FastASR/  mkdir build  cd build  cmake -DCMAKE\_BUILD\_TYPE=Release ..  make |

1. 编译python的whl安装包

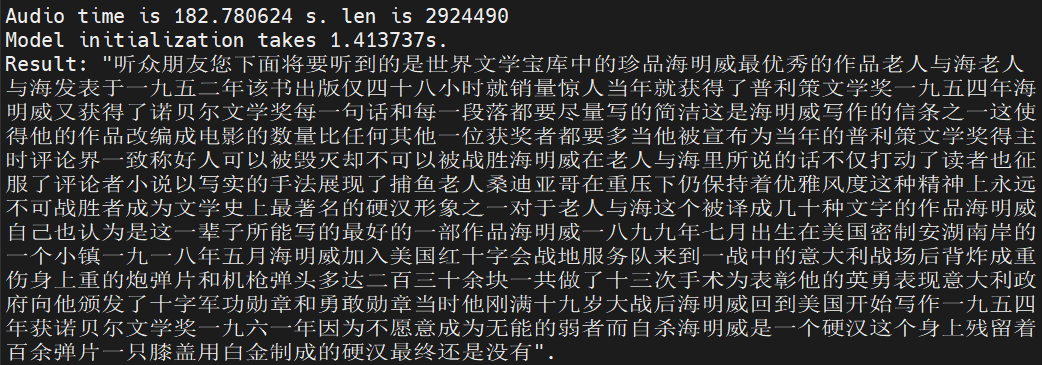
|  |
| --- |
| cd FastASR  python -m build |

1. 下载预训练模型

|  |
| --- |
| **paraformer预训练模型下载**  cd ../models/paraformer\_cli  **1.从modelscope官网下载预训练模型**  wget --user-agent="Mozilla/5.0" -c "https://www.modelscope.cn/api/v1/models/damo/speech\_paraformer-large\_asr\_nat-zh-cn-16k-common-vocab8404-pytorch/repo?Revision=v1.0.4&FilePath=model.pb"   1. **重命名**   mv repo\?Revision\=v1.0.4\&FilePath\=model.pb model.pb   1. **将用于Python的模型转换为C++的**   ../scripts/paraformer\_convert.py model.pb   1. **通过md5检查是否等于 c77bce5758ebdc28a9024460e48602**   md5sum -b wenet\_params.bin  **K2\_rnnt2预训练模型下载**  cd ../models/k2\_rnnt2\_cli  **1.从huggingface官网下载预训练模型**  wget -c https://huggingface.co/luomingshuang/icefall\_asr\_wenetspeech\_pruned\_transducer\_stateless2/resolve/main/exp/pretrained\_epoch\_10\_avg\_2.pt  **2.将用于Python的模型转换为C++的**  ../scripts/k2\_rnnt2\_convert.py pretrained\_epoch\_10\_avg\_2.pt  3.**通过md5检查是否等于 33a941f3c1a20a5adfb6f18006c11513**  md5sum -b wenet\_params.bin  **PaddleSpeech预训练模型下载**  **1.从PaddleSpeech官网下载预训练模型**  wget -c https://paddlespeech.bj.bcebos.com/s2t/wenetspeech/asr1\_conformer\_wenetspeech\_ckpt\_0.1.1.model.tar.gz  **2.将压缩包解压wenetspeech目录下**  mkdir wenetspeech  tar -xzvf asr1\_conformer\_wenetspeech\_ckpt\_0.1.1.model.tar.gz -C wenetspeech  **3.将用于Python的模型转换为C++的**  ../scripts/paddlespeech\_convert.py wenetspeech/exp/conformer/checkpoints/wenetspeech.pdparams  **4.md5检查是否等于 9cfcf11ee70cb9423528b1f66a87eafd**  md5sum -b wenet\_params.bin  **流模式预训练模型下载**  cd ../models/paddlespeech\_stream   1. **从PaddleSpeech官网下载预训练模型**   wget -c https://paddlespeech.bj.bcebos.com/s2t/wenetspeech/asr1/asr1\_chunk\_conformer\_wenetspeech\_ckpt\_1.0.0a.model.tar.gz  **2.将压缩包解压wenetspeech目录下**  mkdir wenetspeech  tar -xzvf asr1\_chunk\_conformer\_wenetspeech\_ckpt\_1.0.0a.model.tar.gz -C wenetspeech  **3.将用于Python的模型转化为C++的**  ../scripts/paddlespeech\_convert.py wenetspeech/exp/chunk\_conformer/checkpoints/avg\_10.pdparams  **4.md5检查是否等于 367a285d43442ecfd9c9e5f5e1145b84**  md5sum -b wenet\_params.bin |

### FastASR使用示例

|  |
| --- |
| #include <iostream>  #ifndef \_WIN32  #include <sys/time.h>  #else  #include <win\_func.h>  #endif  #include <Audio.h>  #include <Model.h>  using namespace std;  int main(int argc, char \*argv[])  {      struct timeval start, end;      Audio audio(0);      audio.loadwav(argv[2]);      audio.disp();      gettimeofday(&start, NULL);      Model \*mm = create\_model(argv[1], 0);      mm->reset();      gettimeofday(&end, NULL);      long seconds = (end.tv\_sec - start.tv\_sec);      long micros = ((seconds \* 1000000) + end.tv\_usec) - (start.tv\_usec);      printf("Model initialization takes %lfs.\n", (double)micros / 1000000);      audio.split();      setbuf(stdout, NULL);      cout << "Result: \"";      gettimeofday(&start, NULL);      float \*buff;      int len;      int flag;      while (audio.fetch(buff, len, flag) > 0) {          mm->reset();          string msg = mm->forward(buff, len, flag);          cout << msg;      }      gettimeofday(&end, NULL);      cout << "\"." << endl;      seconds = (end.tv\_sec - start.tv\_sec);      micros = ((seconds \* 1000000) + end.tv\_usec) - (start.tv\_usec);      printf("Model inference takes %lfs.\n", (double)micros / 1000000);      delete mm;      return 0;  } |



## 4.性能对比

|  |  |  |
| --- | --- | --- |
|  | **短语音测试(5s)** | **长语音测试(30min)** |
| **Paraformer**  **预训练模型** | 0.695871s | 238.797095s |
| **K2\_rnnt2**  **预训练模型** | 0.570641s | 186.848961s |
| **conformer\_wenetspeech-zh-16k**  **预训练模型** | 1.101319s | 351.0674497s |
| **Confermer\_online\_wenetspeech-zh-16k**  **预训练模型** | 1.657996s |  |

表6 FastASR四预训练模型性能测试结果

# 扩展与思考

根据FastASR的思想,根据自己的业务需求构建一个语音识别模块(如果有需要可以将音视频处理集成在一块)。FastASR建立了三个预训练模型,实际的过程中只需要一个预处理模型(如果没有实时性要求,使用paraformer就完全足够使用)，建议使用FastASR的思想进行搭建一个属于自己的语音识别库。