FFmpeg详解

关键概念

数字格式、格式转换。

容器(文件)：flv、mkv

流媒体：就是指采用流式传输技术在网络上连续实时播放的媒体格式，如音频、视频或多媒体文件。主要有 HTTP 渐进下载和基于 RTSP/RTP 的实时流媒体协议；CDN 直播中常用的流媒体协议包括 RTMP，HLS，HTTP FLV

硬编解码：通过硬件实现编解码，减轻 CPU 计算的负担，如 GPU 等

软编解码：通过硬件实现编解码，减轻 CPU 计算的负担，如 GPU 等

音频

编码格式

MP3、AAC、PCM、ogg(ogg vorbis 音频)、AMR、AC3(DVD 专用音频编码)、DTS(DVD 专用音频编码)、APE(monkey’s 音频)、WMA

音质效果对比：AAC+>MP3PRO>AAC>WMA>MP3

目前最常见的是：MP3、AAC、AC-3; MP3最广泛支持最多；AAC是MEPEG-4中的音频标准，技术比较先进。

采样率

也称为采样速度或者**采样频率**，定义了**每秒从连续信号中提取并组成离散信号的采样个数**，它用赫兹（Hz）来表示。采样率是指将**模拟信号转换成数字信号时的采样频率**，也就是单位时间内采样多少点。一个采样点数据有多少个比特。比特率是指每秒传送的比特(bit)数。单位为 bps(Bit Per Second)，比特率越高，传送的数据越大，音质越好.

采样率的选择应该遵循奈奎斯特(Harry Nyquist)采样理论(如果对某一模拟信号进行采样，则采样后可还原的最高信号频率只有采样频率的一半，或者说只要采样频率高于输入信号最高频率的两倍，就能从采样信号系列重构原始信号)。根据该采样理论， CD 激光唱盘采样频率为 44kHz，可记录的最高音频为 22kHz，这样的音质与原始声音相差无几，也就是我们常说的超级高保真音质。通信系统中数字电话的采用频率通常为 8kHz，与原 4k 带宽声音一致的。

位数

量化位是对模拟音频信号的幅度轴进行数字化，它**决定了模拟信号数字化以后的动态范围**。由于计算机按字节运算，**一般的量化位数为 8 位和 16 位**。量化位**越高，信号的动态范围越大，数字化后的音**

频信号就越可能接近原始信号，但所需要的存储空间也越大。

通道数

声道数是音频传输的重要指标，现在主要有单声道和双声道之分。**双声道又称为立体声**，在硬件中要占两条线路，音质、音色好， 但立体声数字化后所占空间比单声道多一倍。

码率

单位时间内传输的数据流量。**比特率(音频) = 采样率 x 采用位数 x 声道数.**

视频

容器格式

mp4、mov、flv、avi、rmvb等

- AVI : 可用格式 MPEG-2, DIVX, XVID, WMV3, WMV4, AC-1, H.264
- WMV : 可用格式 WMV3, WMV4, AC-1
- RM/RMVB : 可用格式 RV40, RV50, RV60, RM8, RM9, RM10
- MOV : 可用格式 MPEG-2, MPEG4-ASP(XVID), H.264
- MKV : 所有格式

编码格式

H263、H264（目前最常用编码格式）、H265、MPEG1，MPEG2、Xvid(MPEG4)、RM、RMVB

码率

码率是指视频文件在**单位时间内使用的数据流量**，也叫码流(Data Rate)、码流率、比特率，通俗一点的理解就是**取样率**,是视频编码中**画面质量控制**中最重要的部分，一般我们用的**单位是 kb/s 或者 Mb/s**。一般来说同样分辨率下，视频文件的码流越大，压缩比就越小，画面质量就越高。**码流越大，说明单位时间内采样率越大，数据流，精度就越高**，处理出来的文件就越接近原始文件，图像质量越好，画质越清晰，要求播放**设备的解码能力也越高**。

当然，码率越大，文件体积也越大，其计算公式是文件体积=时间 X 码率/8。例如，网络上常见的一部 90 分钟 1Mbps 码率的 720P RMVB 文件，其体积就=5400 秒×1Mbps/8=675MB。

通常来说，一个视频文件包括了画面（视频）及声音（音频），例如一个 RMVB 的视频文件，里面包含了视频信息和音频信息，**音频及视频都有各自不同的采样方式和比特率**，也就是说，同一个视频文件音频和视频的码率并不是一样的。而我们所说的一个**视频文件码率大小，一般是指视频文件中音频及视频信息码流率的总和**。

音频码率：sample_rate（采样率） * channles（通道数） * bits （位数）/ 8

帧率

帧率也称为 **FPS**(Frames Per Second)- - - 帧/秒。是指**每秒钟刷新的图片的帧数**，也可以理解为图形处理器每秒钟能够刷新几次。**越高的帧速率可以得到更流畅**、更逼真的动画。每秒钟帧数(FPS)越多，所显示的动作就会越流畅。

关于帧率有如下几个基础数据：

- 帧率越高，cpu 消耗就高
- 秀场视频直播，一般帧率 20fps
- 普通视频直播，一般帧率 15fps

分辨率

视频分辨率是指视频成像产品所成图像的大小或尺寸。常见的视像分辨率有 352×288，176×144，640×480，1024×768。在成像的两组数字中，前者为图片长度，后者为图片的宽度，两者相乘得出的是图片的像素，长宽比一般为 4：3.

480P : 640 x 480 个像素点

720P : 1280 x 720 个像素点

1080P : 1920 x 1080 个像素点

然后还需要关注每个像素点的**存储格式**，每个像素点占用的字节大小。

图像存储格式 yuv

一幅彩色图像的基本要素是什么?

1、宽：一行有多少个像素点。

2、高：一列有多少个像素点，一帧有多少行

3、YUV 格式还是 RGB 格式?

4、一行多少个字节？？

5、图像大小是多少?

6、图像的分辨率多少?

说白了，一幅图像包括的基本东西就是二进制数据，其容量大小实质即为二进制数据的多少。一幅 1920x1080 像素的 YUV422 的图像，大小是 1920X1080X2=4147200（十进制），也就是 3.95M 大小。这个大小跟多少个像素点和数据的存储格式有关。

YUV 与像素的关系：

YUV 格式，与我们熟知的 RGB 类似，YUV 也是一种颜色编码方法，主要用于电视系统以及模拟视频领域，它将亮度信息（Y）与色彩信息（UV）分离，没有 UV 信息一样可以显示完整的图像，只不过是黑白的，这样的设计很好地解决了彩色电视机与黑白电视的兼容问题。并且，YUV 不像 RGB 那样要求三个独立的视频信号同时传输，所以用 YUV 方式传送占用极少的频宽。

YUV 格式有两大类：planar 和 packed。对于 planar 的 YUV 格式，先连续存储所有像素点的 Y，紧接着存储所有像素点的 U，随后是所有像素点的 V。对于 packed 的 YUV 格式，每个像素点的 Y,U,V 是连续交替存储的。

YUV，分为三个分量，“Y”表示明亮度（Luminance 或 Luma），也就是灰度值；而“U”和“V”表示的则是色度（Chrominance 或 Chroma），作用是描述影像色彩及饱和度，用于指定像素的颜色。

YUV 是利用一个亮度（Y）、两个色差(U,V)来代替传统的 RGB 三原色来压缩图像。传统的 RGB 三原色使用红绿蓝三原色表示一个像素，每种原色占用一个字节（8bit），因此一个像素用 RGB 表示则需要 8 * 3=24bit。

如果使用 YUV 表示这个像素，假设 YUV 的采样率为：4：2：0，即每一个像素对于亮度 Y 的采样频率为 1，对于色差 U 和 V，则是每相邻的两个像素各取一个 U 和 V。对于单个的像素来说，色差 U 和 V 的采样频率为亮度的一半。如有三个相邻的像素，如果用 RGB 三原色表示，则共需要占用：8 * 3 * 3 = 72bits；如果采用 YUV（4：2：0）表示，则只需要占用：8 * 3（Y）+ 8* 3 * 0.5（U）+ 8 * 3 * 0.5（V）= 36bits。只需原来一半的空间，就可以表示原来的图像，数据率压缩了一倍，而图像的效果基本没发生变化。

那么，具体 yuv 格式所占用的字节数要怎么算呢？

YUV 图像格式的内存大小

- 4:4:4 表示色度值(UV)没有减少采样。即 Y,U,V 各占一个字节，加上 Alpha 通道一个字节，总共占 4 字节.这个格式其实就是 24bpp 的 RGB 格式了。
- 4:2:2 表示 UV 分量采样减半,比如第一个像素采样 Y,U,第二个像素采样 Y,V,依次类推,这样每个点占用 2 个字节.二个像素组成一个宏像素.
- 需要占用的内存：w * h * 2
- 4:2:0 这种采样并不意味着只有 Y，Cb 而没有 Cr 分量，这里的 0 说的 U，V 分量隔行才采样一次。比如第一行采样 4:2:0 ,第二行采样 4:0:2 ,依次类推...在这种采样方式下，每一个像素占用 16bits 或 10bits 空间.
- 内存则是：yyyyyyyyuuvv
- 需要占用的内存：w * h * 3 / 2
- 4:1:1 可以参考 4:2:2 分量，是进一步压缩，每隔四个点才采一次 U 和 V 分量。一般是第 1 点采 Y,U,第 2 点采 Y,第 3 点采 YV,第 4 点采 Y,依次类推。

一帧图像大小

一帧图像原始大小 = 宽像素 * 长像素 ，当然要考虑数据格式，因为数据格式不一样，大小写也不相同，一般数据采用 rgb、yuv 格式，如 rgb32、yuv420、yuv422 等。最常用的应当属于 yuv420。 因此，计算公式为：

文件的字节数 = 图像分辨率 * 图像量化位数/8

图像分辨率 = X 方向的像素数 * Y 方向的像素数

图像量化数 = 二进制颜色位数

- RGB24 每帧的大小是

size＝width×heigth×3 Bit

- RGB32 每帧的大小是

size＝width×heigth×4

- YUV420 每帧的大小是

size＝width×heigth×**1.5 Bit**

举例说明，对于一个 1024*768 的图像实际的 **YUV422** 数据流大小就为：1024 *768 * 2 = 1572864bit

I 帧、P 帧、B 帧、IDR 帧

参考资料：https://xie.infoq.cn/article/d1167249b8a50cf028d2ba1f9

帧率、码率与分辨率之间关系

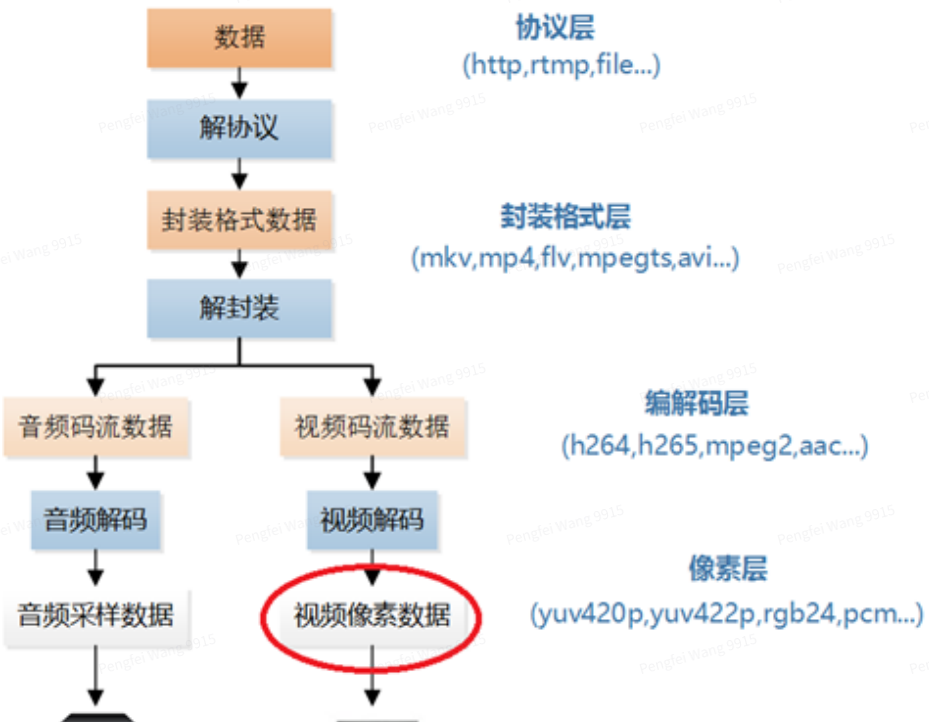
码率和帧率没有半毛钱关系

码率关系着带宽、文件体积

帧率关系着画面流畅度和 cpu 消耗

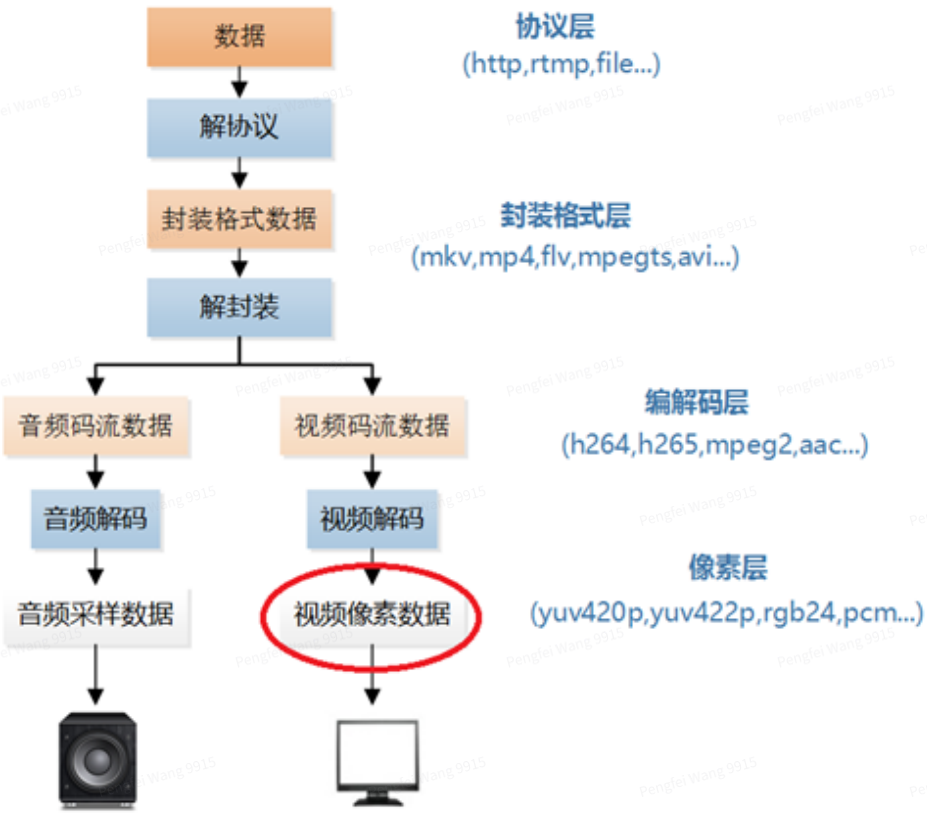
分辨率关系着图像尺寸和清晰度

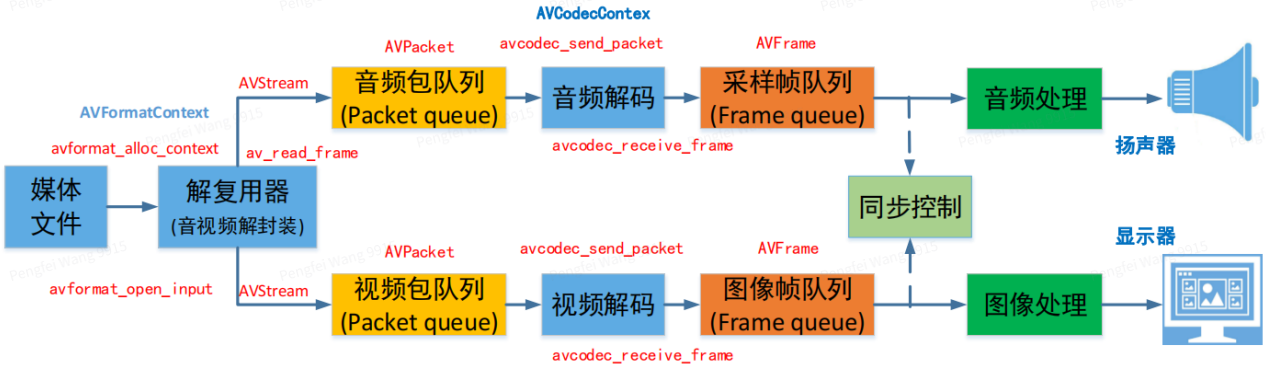
一、整体结构



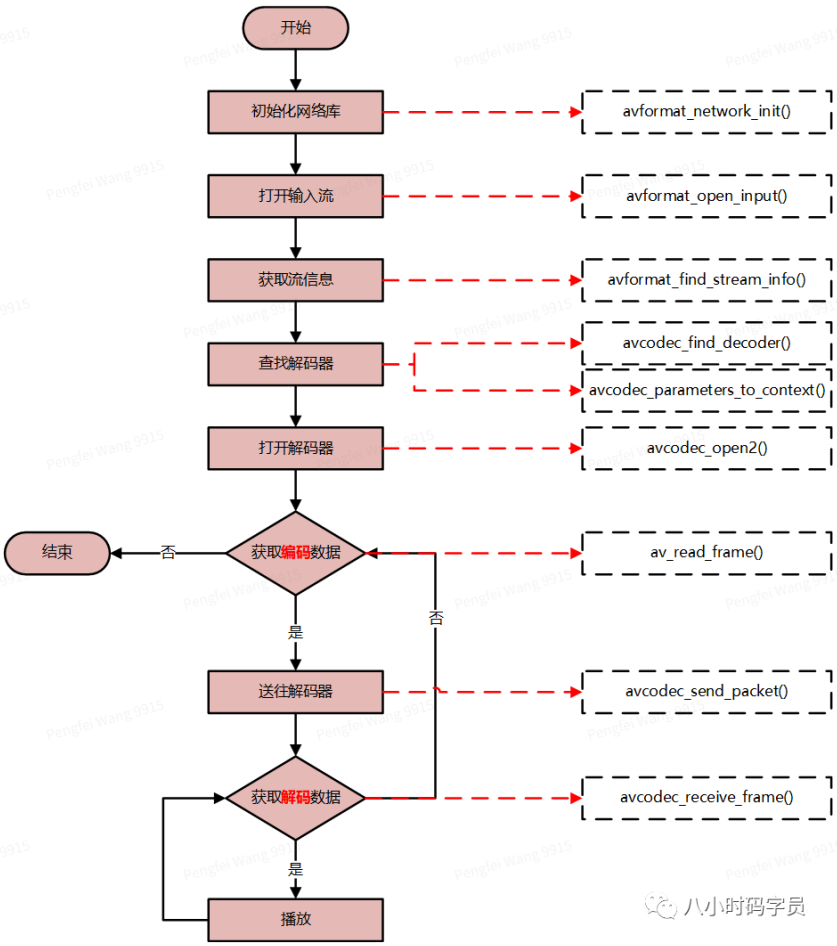
二、流程

播放器


数据
解协议
封装格式数据
解封装
音频码流数据
视频码流数据
音频解码
视频解码
音频采样数据
视频像素数据
协议层
(http,rtmp,file...)
封装格式层
(mkv,mp4,flv,mpegts,avi...)
编解码层
(h264,h265,mpeg2,aac...)
像素层
(yuv420p,yuv422p,rgb24,pcm...)

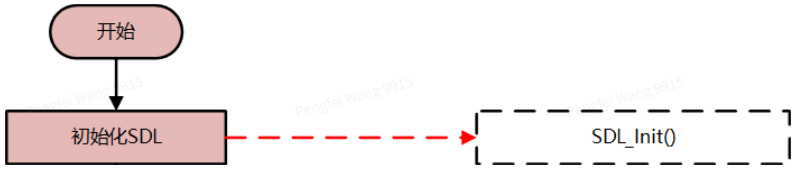


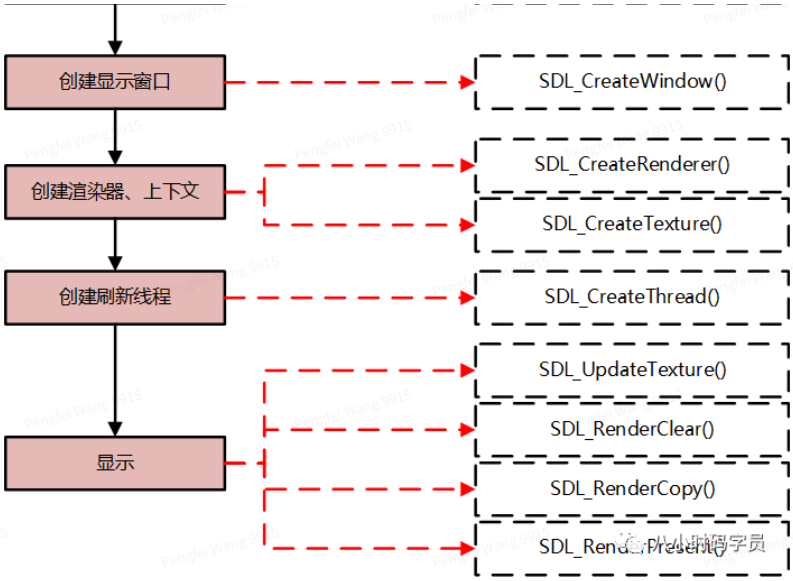
音视频解码



音视频播放

Windows平台使用SDL2来播放音视频，调用流程及相关代码如下

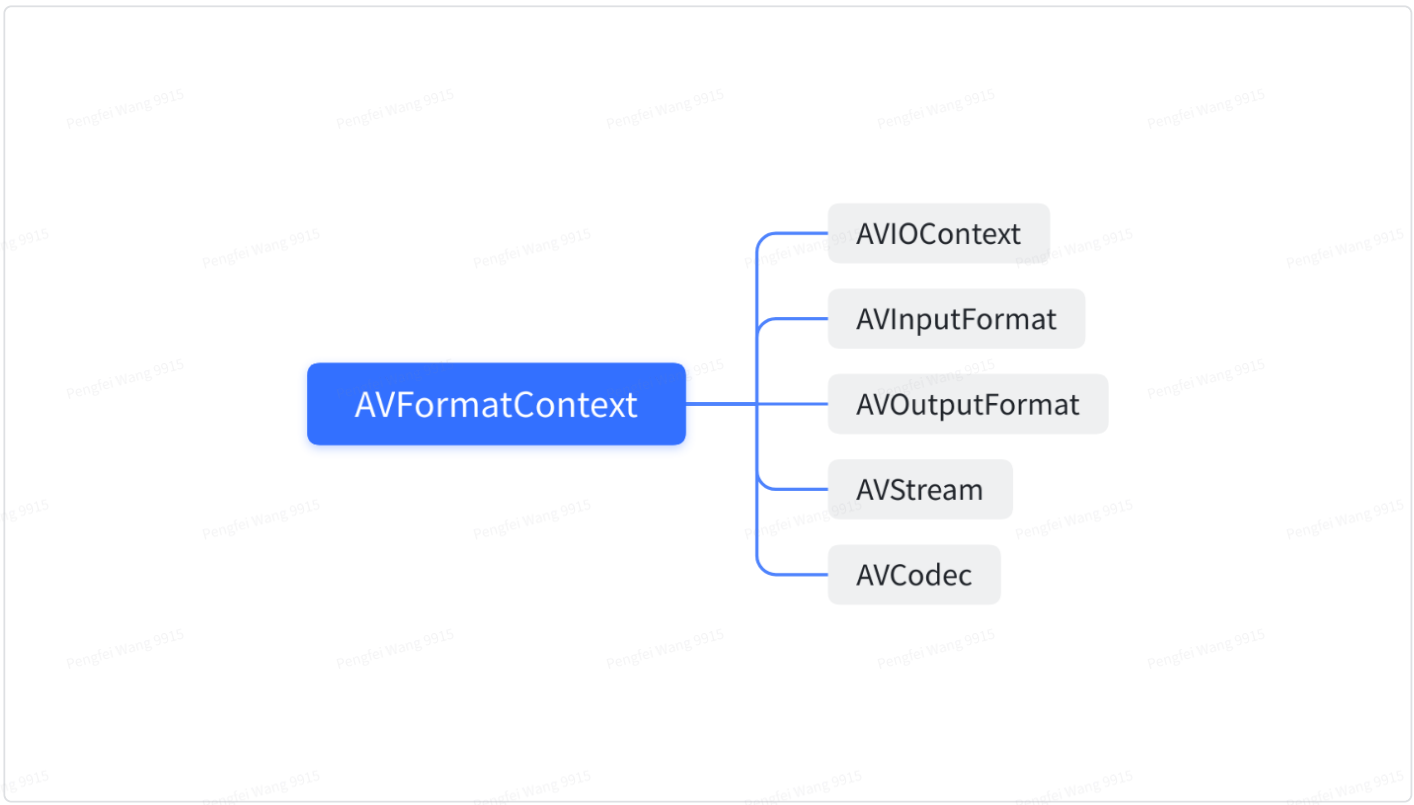




三、关键对象

https://www.cnblogs.com/lidabo/p/15039835.html

https://blog.csdn.net/gushansanren/article/details/123839194



初始化函数

- av_register_all(): 注册所有组件4.0已经弃用
- avdevice_register_all(): 对设备进行注册，比如V4L2等。
- avformat_network_init(): 初始化网络卡以及网络加密协议相关的库（比如openssl）

封装相关函数

- avformat_alloc_context(): 负责申请一个AVFormatContext结构的内存，并进行简单初始化。
- avformat_free_context(): 释放该结构里的所有东西以及该结构本身。
- avformat_close_input(): 关闭解复用器。关闭后就不再需要使用avformat_free_context进行释放。
- avformat_open_input(): 打开输入视频文件。
- avformat_find_stream_info(): 获取音视频文件信息。
- av_read_frame(): 读取音视频包。
- avformat_seek_file(): 定位文件。
- av_seek_frame(): 定位文件。

封装步骤

- 分配解复用器上下文：avformat_alloc_context。
- 根据url打开本地文件或网络流: avformat_open_input。
- 读取媒体的部分数据包以获取码流信息: avformat_find_stream_info。
- 从文件中读取数据包: av_read_frame。或 定位文件: avformat_seek_file、av_seek_frame。
- 关闭解复用器: avformat_close_format。

解码器相关函数

- avcodec_alloc_context3(): 分配解码器上下文
- avcodec_find_decoder()：根据ID查找解码器
- avcodec_find_decoder_by_name():根据解码器名字
- avcodec_open2()： 打开编解码器
- avcodec_decode_video2()：解码一帧视频数据
- avcodec_decode_audio4()：解码一帧音频数据
- avcodec_send_packet(): 发送编码数据包
- avcodec_receive_frame(): 接收解码后数据
- avcodec_free_context():释放解码器上下文，包含了avcodec_close()
- avcodec_close():关闭解码器

解码器步骤

- 分配编解码上下文: avcodec_alloc_context3

- 将码流中的编解码信息拷贝到AVCodecContex: avcode_parameter_to_context
- 根据编解码器信息查找相应的编码器: avcodec_find_decoder或指定解码器 avcodec_find_decoder_by_name
- 打开编解码器并关联到AVCodecContex: avcodec_open2
- 向解码器发送数据包:avcodec_send_packet -> 接受解码后的帧: avcodec_receive_frame
- 关闭解码器和释放上下文avcodec_close: avcodec_free_context

四、常用的操作

命令查询：

- ffmpeg -h
- ffmpeg -h long
- ffmpeg -h full
- ffplay -h
- ffprobe -h

扩展命令：

- ffmpeg -h > ffmpeg_h.log 内容输出到ffmpeg_h.log文件中
- ffmpeg -h | findstr 264 查询ffmpeg -h中 有264的部分

通用参数：

- -i 设置输入流
- -f 设置输出格式
- -ss 设置开发时间

视频参数：

- -b 设置码率 默认200kb/s
- -r 设置帧率 默认20
- -s 设置画面的宽高
- -aspect 设置画面的比例
- -vn 不处理视频
- -vcodec 设置视频编码格式

音频参数：

- -acodec 设置音频编码格式
- -ar 设置采样率

- -ac 设置通道channels数
- -an 不处理音频

常用语句：

Ffmpeg

- 视频容器转换；ffmpeg -i input.mp4 output.ts （MP4转ts）
- 提取音频: ffmpeg -i input.mp4 -vn -acodec copy xxx.aac (mp4的audio codec是aac，根据mp4的音频编码格式决定)
- 提前视频：ffmpeg -i input.mp4 -an -vcodec copy xxx.mp4
- `ffmpeg -ss 00:00:15 -t 00:00:05 -i input.mp4 -vcodec copy -acodec copy output.mp4` -ss表示开始切割的时间，-t表示要切多少

Ffplay

- 播放本地pcm：ffplay -ar 44100 -ac 2 -f s16le -i D:\wpf\vs\c1test\FfmpegFirst\audio.pcm (因pcm数据格式是s16- AC_SAMPLE_FMT_S16,aac是AV_SAMPLE_FMT_FLTP)

五、开发环境搭建

https://www.cnblogs.com/qq21497936/p/13654837.html

https://blog.csdn.net/qq_44623068/article/details/107648236

六、问题记录及解决思路

弱网优化

弱网优化的策略包括如下：

- 播放器 Buffer
- 丢帧策略 (优先丢 P 帧，其次 I 帧，最后音频)
- 自适应码率算法
- 双向链路优化
- 音频 FEC 冗余算法(20%丢包率)

丢帧

在弱网情况下，为了达到更好的体验，可能会采取丢帧的策略，但是丢帧，怎么丢呢？丢音频帧还是视频帧呢？ 因为视频帧比较大，并且视频帧前后是有关联的；音频帧很小，关键是音频帧是连

续采样的，丢了音频帧，那声音就会明显出现瑕疵。为此，一般的丢帧策略是丢视频帧

自适应码率

在弱网情况下，另外一种靠谱的策略是自适应码率算法，通过设置码率降级为多个档次，这样，当网络不好的情况下，通过降低码率进行预测，如果码率降低后，还不够 buffer 缓冲，那么继续降低一个档次，是一个循环探测的过程，如果再次降级一个档次后，发现 buffer 缓冲足够了，那么说明当前网络能够适应这个码率，因此就会采取当前码率。同理，升档也是一样的。但是这个属于厂商的核心算法，

出现花屏、绿屏问题

采集问题、编解码问题、声网传输丢帧问题

声画不同步

采集问题

画面有时候有点糊

弱网，码率的自适应

有声音没有画面

弱网，触发了丢帧策略

画面播放有时候卡顿

CPU 消耗过高导致卡顿，比如 AR 模块

参考资料

https://www.cnblogs.com/lidabo/p/15039835.html

https://blog.csdn.net/gushansanren/article/details/123839194

https://blog.csdn.net/qq21497936/article/details/102478062

https://blog.csdn.net/zhouyongku/article/details/44961447