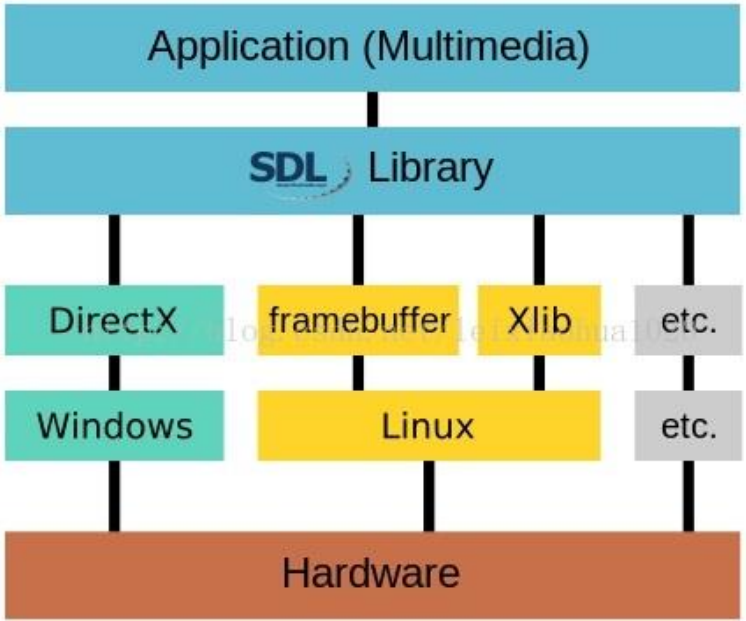
01-SDL简介

官网：https://www.libsdl.org/　文档：http://wiki.libsdl.org/Introduction



SDL（Simple DirectMedia Layer）是一套开放源代码的跨平台多媒体开发库，使用C语言写成。SDL提供了数种控制图像、声音、输出入的函数，让开发者只要用相同或是相似的代码就可以开发出跨多个平台（Linux、Windows、Mac OS X等）的应用软件。目前SDL多用于开发游戏、模拟器、媒体播放器等多媒体应用领域。

对于我们课程而言：SDL主要用来辅助学习FFmpeg，所以我们只会关注我们用到的知识点。

01-Windows环境搭建

下载地址：https://www.libsdl.org/download-2.0.php
先直接下载dll和lib使用

MinGW：Minimalist GNU for Windows

01-Linux环境搭建

下载地址：https://www.libsdl.org/download-2.0.php

1. 下载SDL源码库，SDL2-2.0.10.tar.gz
2. 解压，然后依次执行命令
   ./configure
   make
   sudo make install.
3. 如果出现Could not initialize SDL - No available video device (Did you set the DISPLAY variable?)错误
   说明系统中没有安装x11的库文件，因此编译出来的SDL库实际上不能用。
   下载安装
   sudo apt-get install libx11-dev
   sudo apt-get install xorg-dev

01-SDL子系统

SDL将功能分成下列数个子系统（subsystem）：

- SDL_INIT_TIMER：定时器
- SDL_INIT_AUDIO：音频
- SDL_INIT_VIDEO：视频
- SDL_INIT_JOYSTICK：摇杆
- SDL_INIT_HAPTIC：触摸屏
- SDL_INIT_GAMECONTROLLER：游戏控制器
- SDL_INIT_EVENTS：事件
- SDL_INIT_EVERYTHING：包含上述所有选项

02-SDL Window显示：SDL视频显示函数简介

- SDL_Init()：初始化SDL系统
- SDL_CreateWindow()：创建窗口SDL_Window
- SDL_CreateRenderer()：创建渲染器SDL_Renderer
- SDL_CreateTexture()：创建纹理SDL_Texture
- SDL_UpdateTexture()：设置纹理的数据
- SDL_RenderCopy()：将纹理的数据拷贝给渲染器
- SDL_RenderPresent()：显示
- SDL_Delay()：工具函数，用于延时
- SDL_Quit()：退出SDL系统

02-SDL Windows显示：SDL数据结构简介

- SDL_Window 代表了一个“窗口”
- SDL_Renderer 代表了一个“渲染器”
- SDL_Texture 代表了一个“纹理”
- SDL_Rect 一个简单的矩形结构

存储RGB和存储纹理的区别：
比如一个从左到右由红色渐变到蓝色的矩形，用存储RGB的话就需要把矩形中每个点的具体颜色值存储下来；而纹理只是一些描述信息，比如记录了矩形的大小、起始颜色、终止颜色等信息，显卡可以通过这些信息推算出矩形块的详细信息。所以相对于存储RGB而已，存储纹理占用的内存要少的多。

03-SDL事件

SDL事件

- 函数
  - SDL_WaitEvent()：等待一个事件
  - SDL_PushEvent()：发送一个事件
  - SDL_PumpEvents()：将硬件设备产生的事件放入事件队列，用于读取事件，在调用该函数之前，必须调用SDL_PumpEvents搜集键盘等事件
  - SDL_PeepEvents()：从事件队列提取一个事件
- 数据结构
  - SDL_Event：代表一个事件

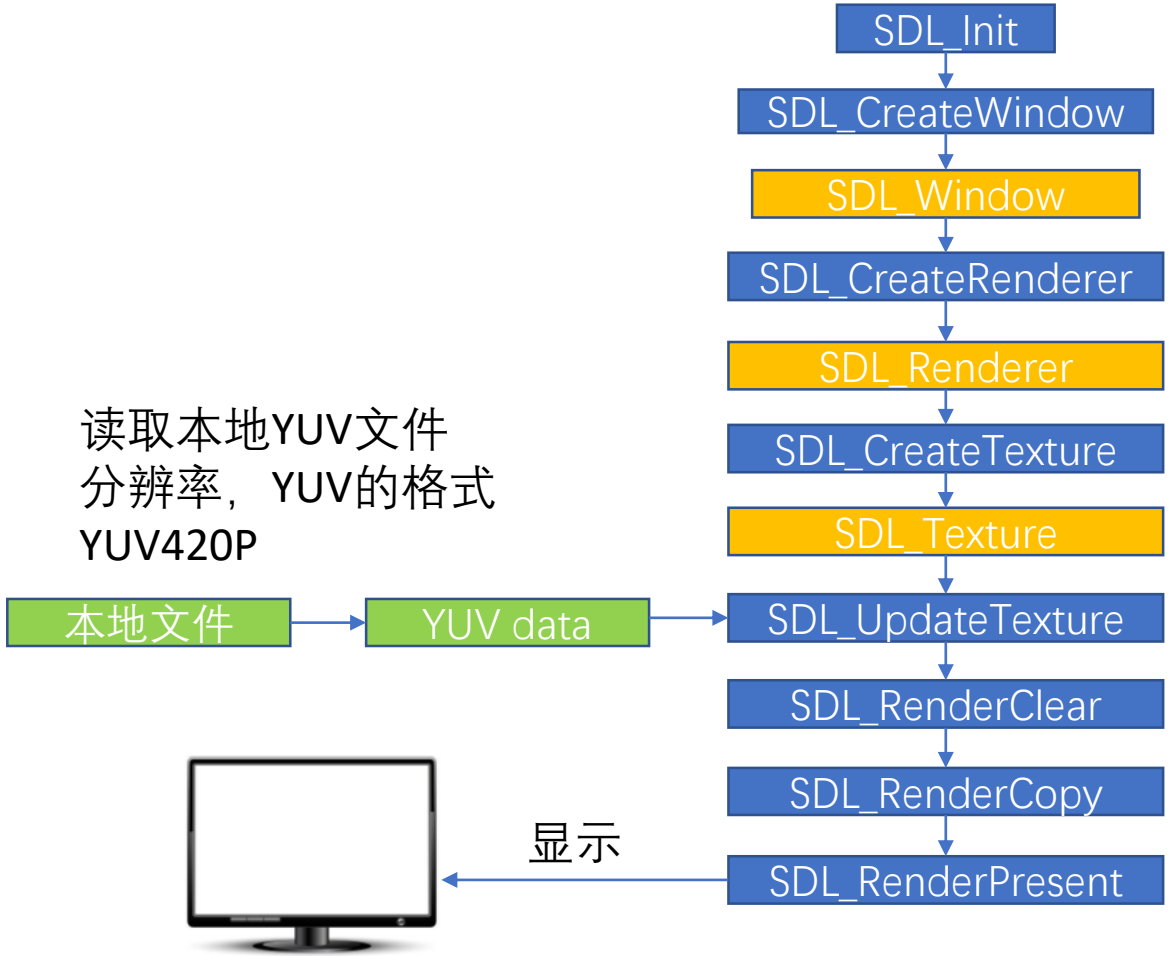
04-SDL线程

SDL多线程

- SDL线程创建：SDL_CreateThread
- SDL线程等待：SDL_WaitThead
- SDL互斥锁：SDL_CreateMutex/SDL_DestroyMutex
- SDL锁定互斥：SDL_LockMutex/SDL_UnlockMutex
- SDL条件变量(信号量)：SDL_CreateCond/SDL_DestoryCond
- SDL条件变量(信号量)等待/通知：SDL_CondWait/SDL_CondSingal

代码： 05-sdl\04-sdl-thread

05-SDL YUV显示：SDL视频显示的流程



06-SDL播放音频PCM-打开音频设备

打开音频设备

```
int SDLCALL SDL_OpenAudio(SDL_AudioSpec * desired,
                          SDL_AudioSpec * obtained);
// desired：期望的参数。
// obtained：实际音频设备的参数，一般情况下设置为NULL即可。
```

SDL_AudioSpec

```
typedef struct SDL_AudioSpec {
    int freq; // 音频采样率
    SDL_AudioFormat format; // 音频数据格式
    Uint8 channels; // 声道数: 1 单声道, 2 立体声
    Uint8 silence; // 设置静音的值，因为声音采样是有符号的，所以0当然就是这个值
    Uint16 samples; // 音频缓冲区中的采样个数，要求必须是2的n次
    Uint16 padding; // 考虑到兼容性的一个参数
    Uint32 size; // 音频缓冲区的大小，以字节为单位
    SDL_AudioCallback callback; // 填充音频缓冲区的回调函数
    void *userdata; // 用户自定义的数据
} SDL_AudioSpec;
```

06-SDL播放音频PCM-SDL_AudioCallback

SDL_AudioCallback

```
// userdata：SDL_AudioSpec结构中的用户自定义数据，一般情况下可以不用。
// stream：该指针指向需要填充的音频缓冲区。
// len：音频缓冲区的大小（以字节为单位）1024*2*2。
void (SDLCALL * SDL_AudioCallback) (void *userdata, Uint8 *stream, int len);
```

播放音频数据

```
// 当pause_on设置为0的时候即可开始播放音频数据。设置为1的时候，将会播放静音的值。
 void SDLCALL SDL_PauseAudio(int pause_on)
```

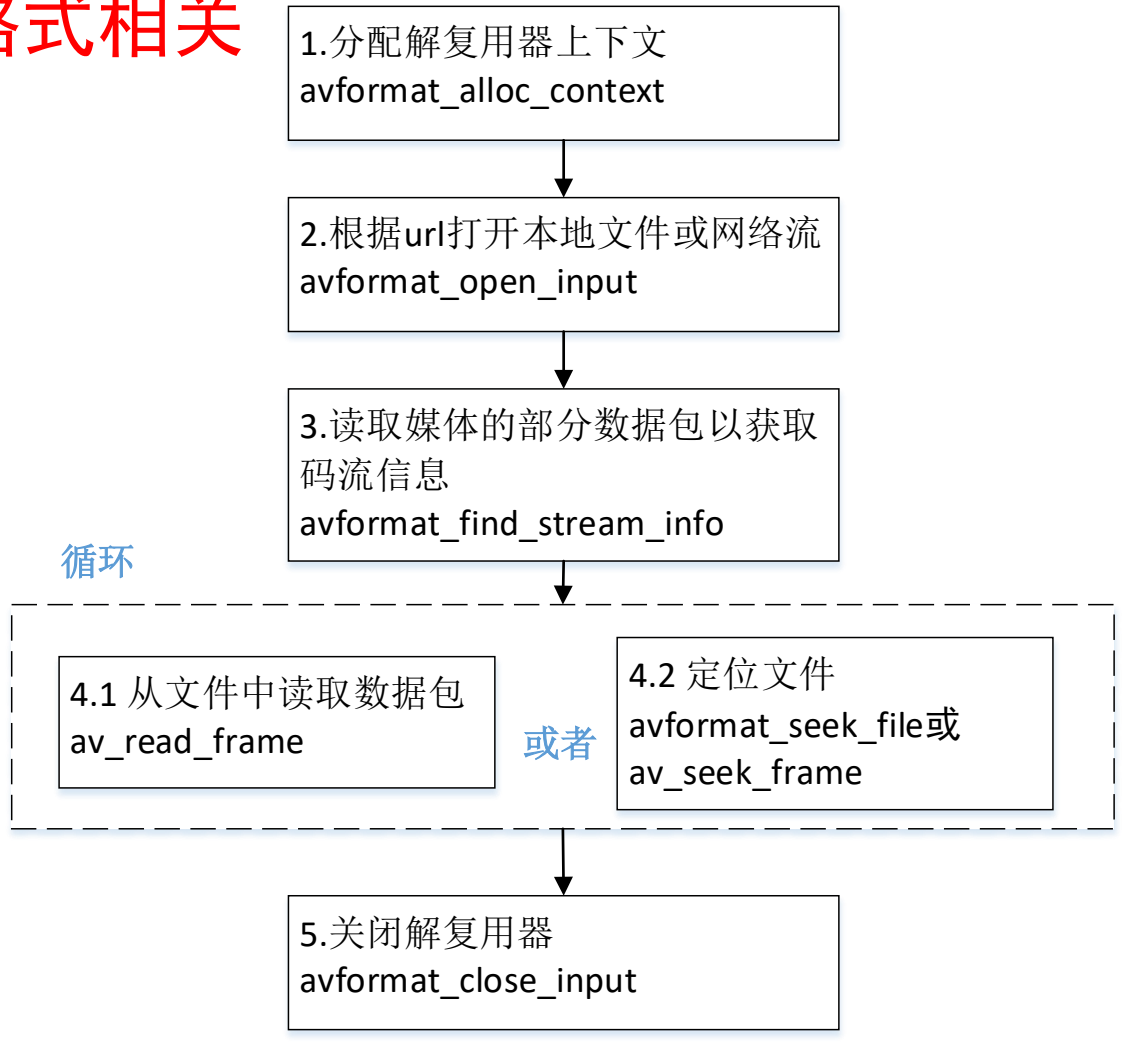
06-SDL播放音频PCM-代码

06-sdl2-pcm范例

FFmpeg函数简介-封装格式相关

- 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

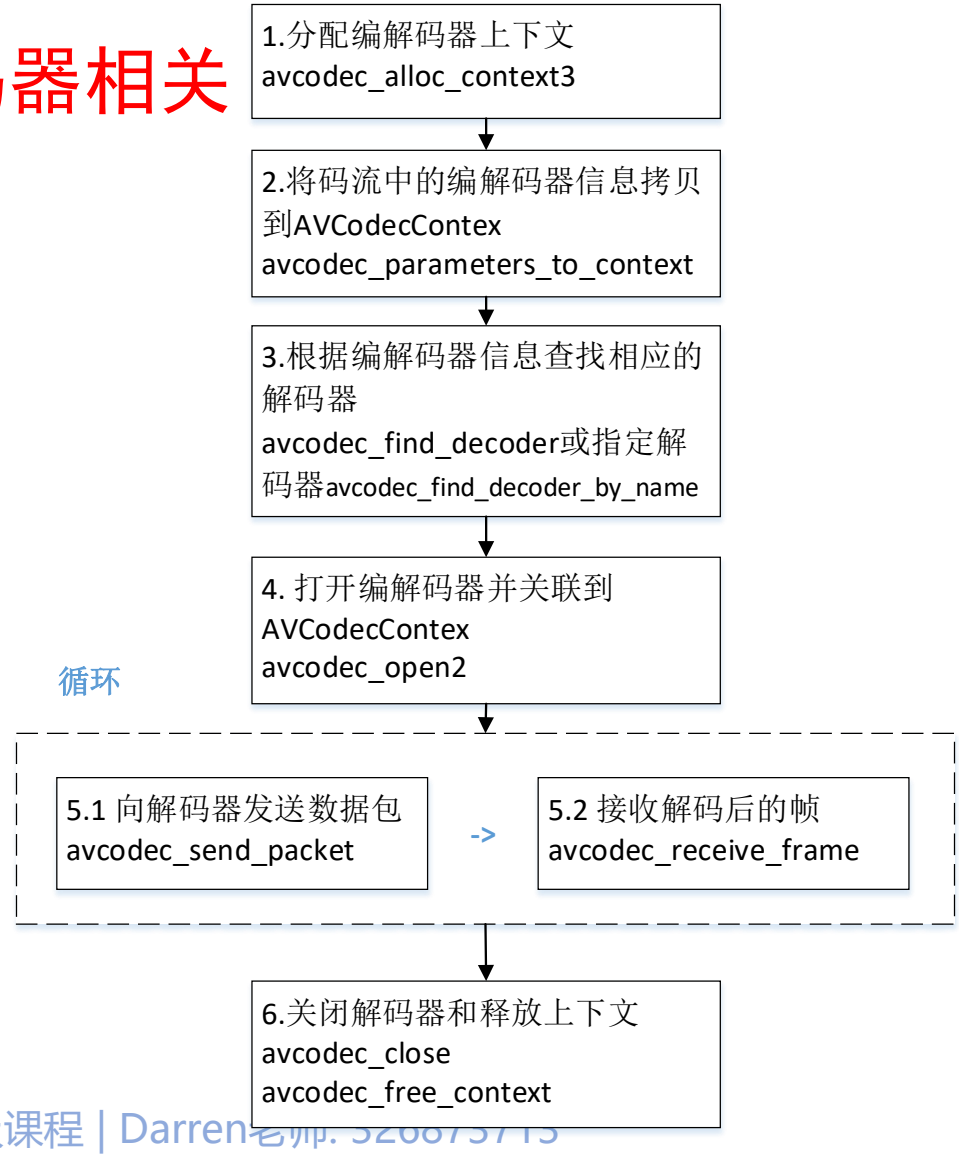
FFmpeg函数简介-封装格式相关



FFmpeg解码函数简介-解码器相关

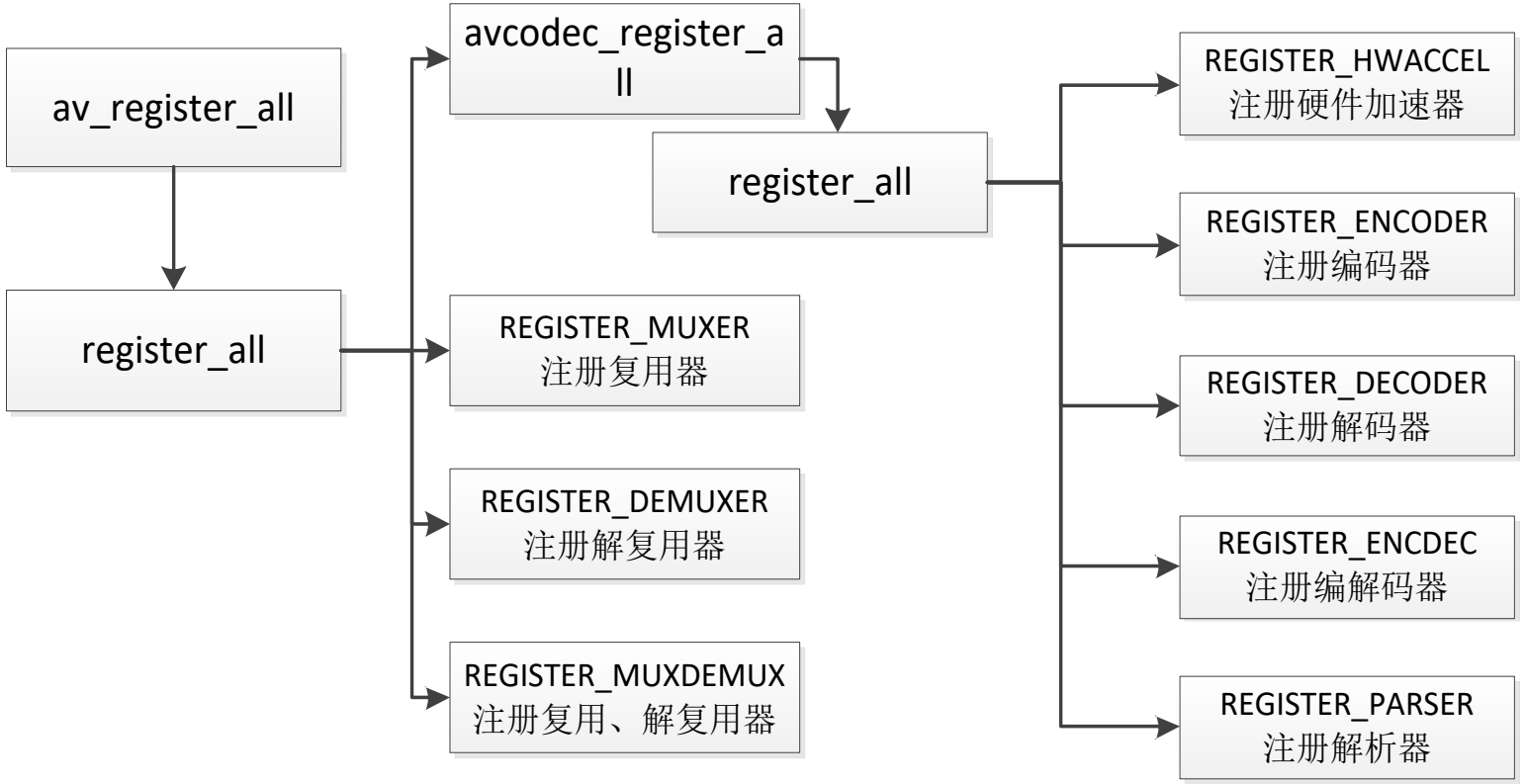
- avcodec_alloc_context3(): 分配解码器上下文
- avcodec_find_decoder(): 根据ID查找解码器
- avcodec_find_decoder_by_name():根据解码器名字
- avcodec_open2(): 打开编解码器
- ~~avcodec_decode_video2(): 解码一帧视频数据~~
- ~~avcodec_decode_audio4(): 解码一帧音频数据~~
- avcodec_send_packet(): 发送编码数据包
- avcodec_receive_frame(): 接收解码后数据
- avcodec_free_context():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

FFmpeg解码函数简介-解码器相关



FFmpeg 3.3 组件注册方式

我们使用ffmpeg，首先要执行av_register_all，把全局的解码器、编码器等结构体注册到**各自全局的对象链表里**，以便后面查找调用。



FFmpeg 4.0.2 组件注册方式

FFmpeg内部去做，不需要用户调用API去注册。
以codec编解码器为例：

1. 在configure的时候生成要注册的组件
./configure:7203:print_enabled_components libavcodec/codec_list.c AVCodec codec_list $CODEC_LIST
这里会生成一个**codec_list.c** 文件，里面只有static const AVCodec * const **codec_list[]**数组。

2. 在**libavcodec/allcodecs.c**将static const AVCodec * const codec_list[]的编解码器用链表的方式组织起来。

Ffmpeg 4.0.2 组件注册方式

FFmepg内部去做，不需要用户调用API去注册。

对于demuxer/muxer（解复用器，也称容器）则对应
1. libavformat/muxer_list.c
libavformat/demuxer_list.c 这两个文件也是在configure的时候生成，
也就是说直接下载源码是没有这两个文件的。

2. 在libavformat/allformats.c将demuxer_list[]和muexr_list[]以链表的方式组织。

其他组件也是类似的方式。

FFmpeg数据结构简介

AVFormat**Context**

封装格式上下文结构体，也是统领全局的结构体，保存了视频文件封装格式相关信息。

AVInputFormat demuxer

每种封装格式（例如FLV, MKV, MP4, AVI）对应一个该结构体。

AVOutputFormat muxer

AVStream

视频文件中每个视频（音频）流对应一个该结构体。

结构体名带context意味着什么?

AVCodec**Context**

编解码器上下文结构体，保存了视频（音频）编解码相关信息。

AVCodec

每种视频（音频）编解码器(例如H.264解码器)对应一个该结构体。

AVPacket

存储一帧压缩编码数据。

AVFrame

存储一帧解码后像素（采样）数据。

FFmpeg数据结构之间的关系

面向对象的封装？

AVFormatContext和AVInputFormat之间的关系
AVFormatContext API调用
AVInputFormat 主要是FFMPEG内部调用

AVFormatContext 封装格式上下文结构体

struct AVInputFormat *iformat;

数据

所有的方法可重入的

AVInputFormat 每种封装格式（例如FLV, MKV, MP4）

int (*read_header)(struct AVFormatContext *);

int (*read_packet)(struct AVFormatContext *, AVPacket *pkt);

方法

int avformat_open_input(AVFormatContext **ps, const char *filename,
AVInputFormat *fmt, AVDictionary **options)

FFmpeg数据结构之间的关系

AVCodecContext和AVCodec之间的关系

AVCodecContext 编码器上下文结构体

```
struct AVCodec  *codec;
```

数据

AVCodec 每种视频（音频）编解码器

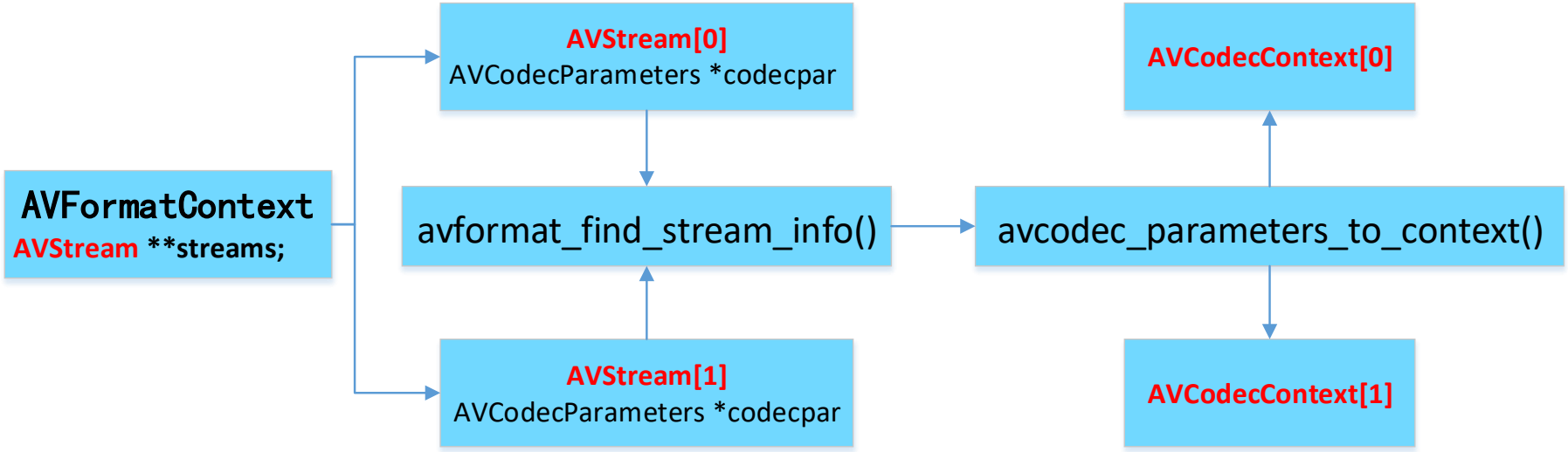
```
int (*decode)(AVCodecContext *, void *outdata, int *outdata_size,
AVPacket *avpkt);

int (*encode2)(AVCodecContext *avctx, AVPacket *avpkt, const AVFrame
*frame,   int *got_packet_ptr);
```

方法

FFmpeg数据结构之间的关系

AVFormatContext, AVStream和AVCodecContext之间的关系



FFmpeg数据结构之间的关系

区分不同的码流

- AVMEDIA_TYPE_VIDEO视频流

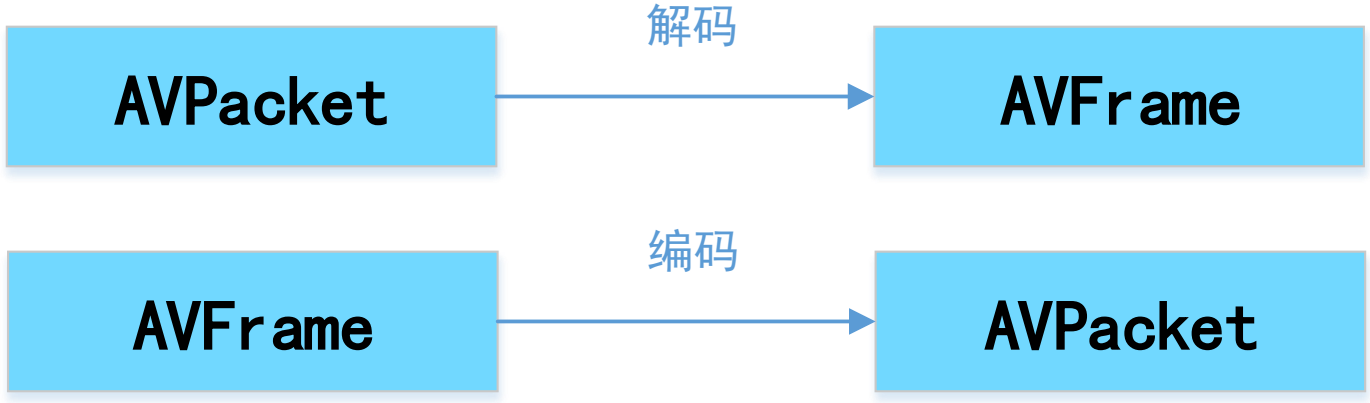
```
video_index = av_find_best_stream(ic, AVMEDIA_TYPE_VIDEO,
                                  -1,-1, NULL, 0)
```

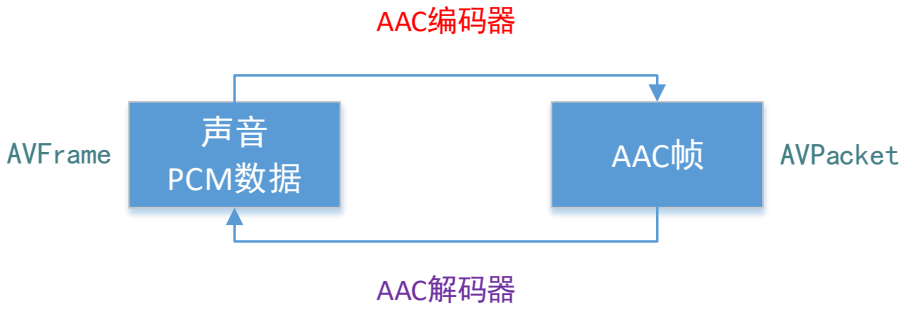
- AVMEDIA_TYPE_AUDIO音频流

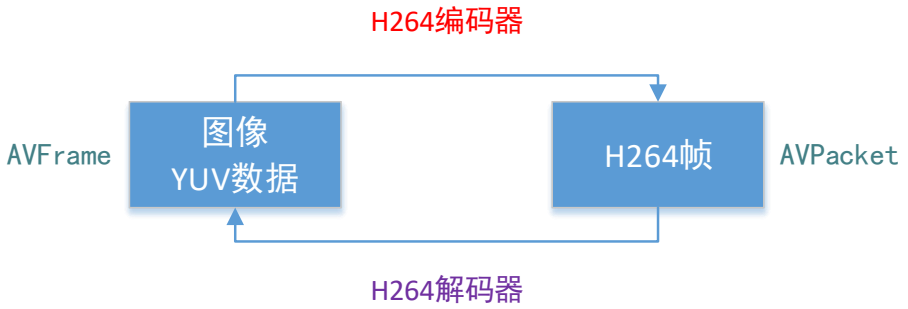
```
audio_index = av_find_best_stream(ic, AVMEDIA_TYPE_AUDIO,
                                  -1,-1, NULL, 0)
```

FFmpeg数据结构之间的关系

AVPacket和AVFrame之间的关系







FFmpeg数据结构分析

- **AVFormatContext**
  - iformat：输入媒体的AVInputFormat，比如指向AVInputFormat ff_flv_demuxer
  - nb_streams：输入媒体的AVStream 个数
  - streams：输入媒体的AVStream []数组
  - duration：输入媒体的时长（以微秒为单位），计算方式可以参考av_dump_format()函数。
  - bit_rate：输入媒体的码率

- **AVInputFormat**
  - name：封装格式名称
  - extensions：封装格式的扩展名
  - id：封装格式ID
  - 一些封装格式处理的接口函数，比如read_packet()

FFmpeg数据结构分析

- **AVStream**
  - index：标识该视频/音频流
  - time_base：该流的时基，PTS*time_base=真正的时间（秒）
  - avg_frame_rate： 该流的帧率
  - duration：该视频/音频流长度
  - codecpar：编解码器参数属性

- **AVCodecParameters**
  - codec_type：媒体类型AVMEDIA_TYPE_VIDEO/ AVMEDIA_TYPE_AUDIO等
  - codec_id：编解码器类型， AV_CODEC_ID_H264/ AV_CODEC_ID_AAC等。

FFmpeg数据结构分析

- **AVCodecContext**
  - codec：编解码器的AVCodec，比如指向AVCodec ff_aac_latm_decoder
  - width, height：图像的宽高（只针对视频）
  - pix_fmt：像素格式（只针对视频）
  - sample_rate：采样率（只针对音频）
  - channels：声道数（只针对音频）
  - sample_fmt：采样格式（只针对音频）

- **AVCodec**
  - name：编解码器名称
  - type：编解码器类型
  - id：编解码器ID
  - 一些编解码的接口函数，比如int (*decode)()

FFmpeg数据结构分析

- **AVCodecContext**
  - codec：编解码器的AVCodec，比如指向AVCodec ff_aac_latm_decoder
  - width, height：图像的宽高（只针对视频）
  - pix_fmt：像素格式（只针对视频）
  - sample_rate：采样率（只针对音频）
  - channels：声道数（只针对音频）
  - sample_fmt：采样格式（只针对音频）

- **AVCodec**
  - name：编解码器名称
  - type：编解码器类型
  - id：编解码器ID
  - 一些编解码的接口函数，比如int (*decode)()

Ffmpeg 4.0.2 组件注册方式