雷晓华 音视频https://blog.csdn.net/leixiaohua1020/article/details/15811977

重要的数据结构

1.AVFrame

帧结构体存储着原始的数据，比如视频帧RGB

包含着数据帧的重要信息

typedef struct AVFrame

{

uint8\_t \*data[AV\_NUM\_DATA\_POINTERS];//原始的数据(如视频帧RGB)

int width,height;//视频帧的宽和高

int format; //帧的格式，视频格式在AVPixelFormat枚举，音频帧格式在 AVSampleFormat中枚举

int key\_frame; //指明是否关键帧(1是0否)

enum AVPictureType pict\_type; //帧的类型(i,p,b帧等)

AVRational sample\_aspect\_ratio; //帧的宽高比

int64\_t pts;//时间戳，即该帧显示出来的时间

int64\_t pkt\_dts;//包解码的时间

int coded\_picture\_number;编码的帧序号

int display\_picture\_number;//显示的帧序号

//还有一些运动矢量表，参考帧表

}

2.AVPacket

数据流分出的包，从数据流独处数据后放入的结构对象，存在于解复用之后，解码之前

对于视频来讲，一个AVPacket包含刚好一个AVFrame

typedef struct AVPacket｛

AVBufferRef \*buf;//

int64\_t pts; // 该包展示出来的时间

int64\_t dts; //该包解码的时间

unint8\_t \*data;//保存压缩编码的数据，引用的是数据缓冲区

int size;//data的大小

int flags; //关键帧的标志,1是关键帧

AVPacketSideData \*size\_data;(AVStream中的解释)

｝

3.AVStream

存储音视频流信息的结构体

存储了流的重要信息

typedef struct AVStream

{

int index;//AVFormatContext中的对应的index

int id; //指定该格式特定的唯一ID

AVRational time\_base;//时间技术， 1/帧率 方便计算实际时间

int64\_t start\_time//该流中第一帧的时间戳

int64\_t duration;//该留的持续时间

int64\_t nb\_frames;//该流中包含的帧数

AVPacketSideData \*side\_data //这是side data的数组，一个side data是由复用器输出的(解封装时)或者调用设定的(封装时),这样就不会存在于包packet中

AVRational avg\_frame\_rate;//平均帧率

}

4.AVFormatContext

贯穿始终的一个重要结构体，有很多承上启下的重要参数。它是负责封装与解封装的重要载体

typedef struct AVFormatContext{

const AVClass \*av\_class;//如果一个结构体想要支持AVOption(拥有添加其余参数的功能)

AVInputFormat\*iformat余AVOutputFormat\* oformat//输入(解封装用)和输出(封装用)的参数，两者只能有一个不为空

AVIOContext\*pb;//输入和输出的缓存

int flags;//标志位，标明一些特殊要求的属性

unsigned int nb\_streams; //

AVStream\*\* streams; //文件中所有流的列表

Char filename[1024]; //文件名

int64\_t start\_time; //首帧的位置

int64\_t duration; //流的时长

int64\_t bit\_rate; //比特率

int max\_delay; //最大延迟

int64\_t probesize; //探针大小

int64\_t max\_analyze\_duration; //最大分析时长

unsigned int max\_index\_size; //每个流的最大索引

unsigned int max\_picture\_buffer; //最大的图片缓存（从硬件获取的）

//（以上五个可能影响到程序的效率）

enum AVCodecID video\_codec\_id;//视频编解码器类型ID

enum AVCodecID audio\_codec\_id;//音频编解码器类型ID

enum AVCodecID subtitle\_codec\_id;//字幕的编解码器类型ID

AVCodec\* video\_codec;//视频编解码器

AVCodec\* audio\_codec;//音频编解码器

AVCodec\* subtitle\_codec;//字幕的编解码器

char\* codec\_whitelist; //解码器允许白名单

char\* format\_whitelist; //解复用器允许白名单

}

5.AVIOContext

管理二进制流数据输入输出的结构体

该结构体与缓冲相关

typedef struct AVIOContext {

const AVClass\* av\_class; //如果一个结构体想要支持AVOption（拥有添加其余参数的功能）的话，第一个成员变量必须是指向AVClass的指针

unsigned char\* buffer; //数据缓冲区的起始

int buffer\_size; //数据缓冲区的尺寸

unsigned char\* buf\_ptr; //缓冲区当前指向的位置

unsigned char\* buf\_end; //数据的结束位置

int error; //错误码信息

//......

}

6.AVCodec

存储和管理编解码器信息所用的结构体

typedef struct AVCodec {

const char\* name; //编解码器的name（短）

const char\* long\_name; //编解码器的long\_name（长，且更有可读性）

enum AVMediaType type; //类型（VIDEO、AUDIO、DATA等等）

enum AVCodecID id; //每个编解码器专有的唯一ID

const AVRational\* supported\_framerates; //存有支持的码率的数组

enum AVPixelFormat\* pix\_fmts; //存有支持的像素格式的数组（RGB24、YUV420P等等）

const int\* supported\_samplerates; //存有支持的（音频）采样率的数组

enum AVSampleFormat\* sample\_fmts; //存有支持的（音频）采样格式的数组

const uint64\_t\* channel\_layouts; //存有支持的（音频）通道数的数组

//还有一个这样的参数struct AVCodec\* next

//另外，是decode，close，send\_frame等接口，由具体的编解码器提供

//......

}

7.AVCodecContext

这是存有对编解码器详细描述的结构体，这个结构体非常庞大，占据了源码一千多行，对部分关键参数进行了解

typedef struct AVCodecContext {

const AVClass\* av\_class; //如果一个结构体想要支持AVOption（拥有添加其余参数的功能）的话，第一个成员变量必须是指向AVClass的指针

const struct AVCodec\* codec; //所属的Codec

enum AVMediaType codec\_type;

enum AVCodecID codec\_id; //与（6、AVCodec相同）

uint8\_t\* extradata; //有些编解码器需要的可以使用的额外数据

int extradata\_size; //额外数据的大小

int64\_t bit\_rate; //平均比特率

int bit\_rate\_tolerance; //比特率可以容忍的差别

AVRational time\_base; //时间基数，1/帧率

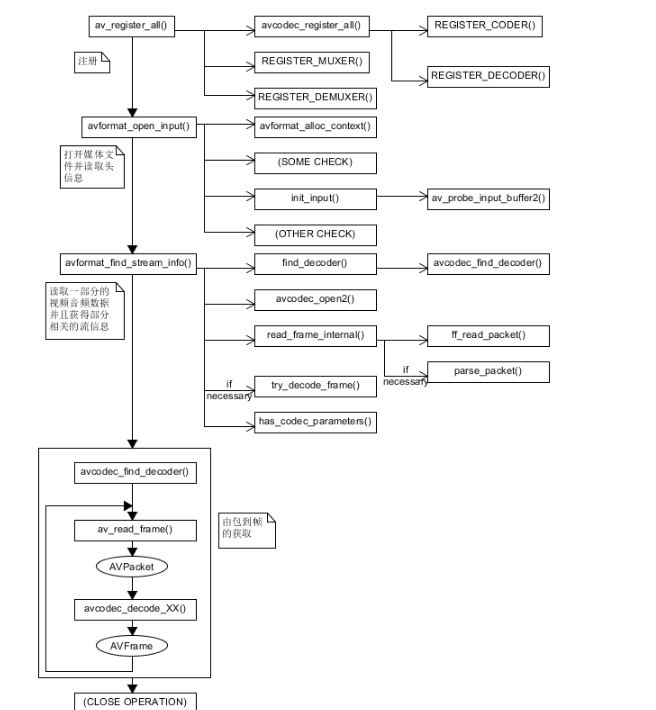
int width, height; //（视频）宽高

//还有其他很多详尽的参数，需要用到时再进行查询

//......

}

**二、解码的主要流程**



2．1 av\_register\_all()

该函数在allformats.c中定义，这是几乎每一个程序开始时候必须要调用的。

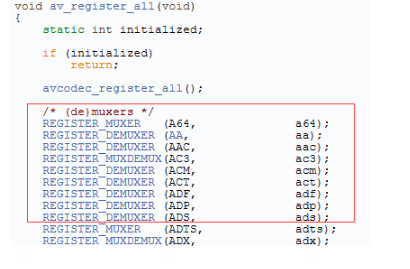
不用传入参数直接调用

作用：注册复用器、解复用器、编码器、解码器等等必须的元件

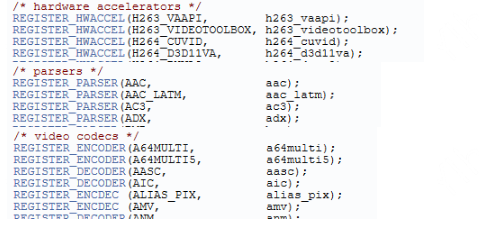
注册方便后续查找解复用器和编码器时，进行遍历

函数的结构：

进入函数可以看到巨大的篇幅将每一个支持的（解）复用器以及其对应的格式列举出来，就像这样：



我们注意到在它们前面还有一个avcodec\_register\_all()，这是在allcodecs.c中定义的，包含着的是如下的有编解码器（视频、音频等）、语法分析器、硬件加速器等等。



无论是什么器，它们的注册过程实际上就是将它们放在一个容器中。

    在后面的遍历过程中，就是找到对应的容器，从中去查找。

当然，有时也可以不调用av\_register\_all()这样的函数，如果不需要去支持那么多的格式，那就列举出需要的几个。

但是归根到底，这些做法都是“注册”，它就是整个程序的纽带。

2.2 avformat\_open\_input(0

该函数用于打开媒体数据，读取头获取一些信息

2.2.1涉及的参数

AVFormatContext\*\* ps：AVFormatContext(输入的的指针，可以是指向空的指针（在该函数中AVFormatContext会被分配内存并写入相关信息）

const char\* url：要打开的媒体流的url

AVInputFormat\* fmt：如果fmt非空的话，那就是指定了一种输入格式，否则输入格式将会被自己检测并获取

AVDictionary\*\* options：附加的选项，用于添加字段，一般为NULL,

返回值：0表示成功，失败的话返回一个负数AVERROR

2.2.2 函数的结构

1.调用avformat\_alloc\_context()，为AVFormatContext分配内存，初始化等基本操作

2.判断1是否分配成功，如果失败的话，返回对应的错误码AVERROR

3.检查是否已经指定了一个特定的格式fmt，如果是的话，将它赋予AVFormatContext

4.检查是否给了一个附加的选项，如果有的话，那么就添加字段进AVFormatContext

5.检查是否有缓存，如果有，标志位标好“先别关，这还有数据”（这个要放在所有可能跳到fail处理的判断前面）

6.添加字段，如果添加失败的话，跳到fail处理

7.init\_input()：这是该函数中较为重要的部分，打开输入的文件并在必要时探寻格式就是在这个里面。

它的整体思路是这样的，首先检查是否已经指定了特定的格式，如果是的话，那就直接返回；

否则，就需要打开文件来探测文件的格式了，这里涉及到一个函数，av\_probe\_input\_buffer2()，这个函数的输入包括a.要探测的二进制流 b.输入格式也就是最终确定的格式放置的位置 c.流的url  d.探针的偏移（判断是否超出最大范围） e.最大的探测范围。

该函数所做的工作是：探测一个二进制流来决定输入的格式，对每一次尝试进行匹配评分（格式越适合分数越高），每一次探针返回一个低分数时，探针缓存会增加，并且开始另外一次尝试，直到达到最大的缓存。

返回：最高分数的格式和该分数，失败则返回一个负数

8.检查确认黑名单白名单

9. 读入文件并存入AVFormatContext的流里

2.2.3 avformat\_find\_stream\_info()

1.该函数是用来读取一部分的视频音频数据并且获得部分相关的流信息

2.涉及的参数：AVFormatContext\* ic：要处理的媒体文件 AVDictionary\*\* options：附加的选项

3.函数的结构：

find\_decoder()：输入包括AVFormatContext（要处理的文件），AVStream（指定的数据流），AVCodecID（解码器的ID），返回一个AVCodec（解码器），如果指定的AVStream包含解码器的话，直接返回，否则调用avcodec\_find\_decoder()，这个就是遍历解码器容器，寻找对应的解码器。

调用avcodec\_open2()，作用是用给定的AVCodec去初始化AVCodecContext

read\_frame\_internal()，读取码流中的若干帧音频或者一帧视频，其中先是调用ff\_read\_packet()从相应的AVInputFormat读取数据（调用AVInputFormat中的read\_packet()方法接口），接着对于需要语法分析的调用parse\_packet()，提供解析AVPacket的功能。

（解码视频的时候，每解码一个视频帧，需要先获得一帧视频的压缩数据，然后才能对该数据进行解码）

4. try\_decode\_frame()，如果还没能获得信息，尝试打开解码器来对帧进行解压，但是这个尽量还是要避免，因为它需要耗费更多的时间、使用更多的内存

5. has\_codec\_parameters()判断是参数是否都已经找到，如果没有的话会输出log警告

2.2.4 av\_read\_frame()

 参数包含AVFormatContext\* s（要处理的文件），AVPacket\* pkt（获取的包）

返回的是流的下一帧。

这个函数返回文件中所存储的，但是并不能确定那是解码器所需要的有效帧。它将文件中存储的东西分成帧，并且每次调用返回一个。不阻止无效的数据，以使解码器得到最多的信息去解码。

对于视频来讲，packet包含恰好一个帧。

# 三、延伸扩展

1、在初期的学习过程中，自己提出了一个疑惑，为什么某些格式有复用器没解复用器，有编码器没解码器呢？

通过后续的学习，我才了解到关于编解码格式和封装格式是两回事，封装格式其实只是外部的包装，是外层的“皮”；而编码格式是内容物，是皮里边的“馅”。

2、前面提到了AVFrame的类型中存在三种重要的帧类型，分别是I帧、B帧和P帧。

    I帧是关键帧，I帧是数据完整的保留

    P帧是向前预测编码帧，P帧表示了该帧跟之前一个I帧或者P帧的差别

    B帧是双向差别帧，B帧记录了该帧和前后帧的差别

    解码时，I帧是只需要自己的数据就可以完成，而P帧需要用之前帧叠加上该帧的差别，B帧不仅需要用到之前的帧，还需要用到之后的帧来确定。因此，解码的顺序不一定和帧的顺序一样，这样就有了之前介绍的，帧有“解码时间”和“展示时间”两个不同的参数。

    I、P、B帧中，I帧的压缩率最低，B帧的压缩率最高，但是I帧是最关键的，如果在解码过程中I帧出现失误，将会影响到它前后的若干帧，就容易出现花屏、黑屏，甚至crash的情况，需要看具体的处理方式。