ffmpeg中tbc tbr tbn代表的含义：

tbn is the time base in AVStream that has come from the container, I

think. It is used for all AVStream time stamps.

tbc is the time base in AVCodecContext for the codec used for a

particular stream. It is used for all AVCodecContext and related time

stamps.

tbr is guessed from the video stream and is the value users want to see

when they look for the video frame rate, except sometimes it is twice

what one would expect because of field rate versus frame rate.

FFMPEG中结构体很多。最关键的结构体可以分成以下几类：

a) 解协议（http,rtsp,rtmp,mms）

AVIOContext，URLProtocol，URLContext主要存储视音频使用的协议的类型以及状态。URLProtocol存储输入视音频使用的封装格式。每种协议都对应一个URLProtocol结构。（注意：FFMPEG中文件也被当做一种协议“file”）

b) 解封装（flv,avi,rmvb,mp4）

AVFormatContext主要存储视音频封装格式中包含的信息；AVInputFormat存储输入视音频使用的封装格式。每种视音频封装格式都对应一个AVInputFormat 结构。

c) 解码（h264,mpeg2,aac,mp3）

每个AVStream存储一个视频/音频流的相关数据；每个AVStream对应一个AVCodecContext，存储该视频/音频流使用解码方式的相关数据；每个AVCodecContext中对应一个AVCodec，包含该视频/音频对应的解码器。每种解码器都对应一个AVCodec结构。

d) 存数据

视频的话，每个结构一般是存一帧；音频可能有好几帧

解码前数据：AVPacket

解码后数据：AVFrame

FFmpeg数据结构分析：

AVFormatContext：

AVIOContext \*pb：输入数据的缓存

unsigned int nb\_streams：视音频流的个数

AVStream \*\*streams：视音频流

char filename[1024]：文件名

int64\_t duration：时长（单位：微秒us，转换为秒需要除以1000000）

int bit\_rate：比特率（单位bps，转换为kbps需要除以1000）

AVDictionary \*metadata：元数据

视频的原数据（metadata）信息可以通过AVDictionary获取。元数据存储在AVDictionaryEntry结构体中，如下所示

typedef struct AVDictionaryEntry {

char \*key;

char \*value;

} AVDictionaryEntry;

每一条元数据分为key和value两个属性。

在ffmpeg中通过av\_dict\_get()函数获得视频的原数据。

AVInputFormat：

name：封装格式名称

long\_name：封装格式的长名称

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

AVStream：

int index：标识该视频/音频流

AVCodecContext \*codec：指向该视频/音频流的AVCodecContext（它们是一一对应的关系）

AVRational time\_base：时基。通过该值可以把PTS，DTS转化为真正的时间。FFMPEG其他结构体中也有这个字段，但是根据我的经验，只有AVStream中的time\_base是可用的。PTS\*time\_base=真正的时间

int64\_t duration：该视频/音频流长度

AVDictionary \*metadata：元数据信息

AVRational avg\_frame\_rate：帧率（注：对视频来说，这个挺重要的）

AVPacket attached\_pic：附带的图片。比如说一些MP3，AAC音频文件附带的专辑封面。

AVCodecContext：

enum AVMediaType codec\_type：编解码器的类型（视频，音频...）

struct AVCodec \*codec：采用的解码器AVCodec（H.264,MPEG2...）

int bit\_rate：平均比特率

uint8\_t \*extradata; int extradata\_size：针对特定编码器包含的附加信息（例如对于H.264解码器来说，存储SPS，PPS等）

AVRational time\_base：根据该参数，可以把PTS转化为实际的时间（单位为秒s）

int width, height：如果是视频的话，代表宽和高

int refs：运动估计参考帧的个数（H.264的话会有多帧，MPEG2这类的一般就没有了）

int sample\_rate：采样率（音频）

int channels：声道数（音频）

enum AVSampleFormat sample\_fmt：采样格式

int profile：型（H.264里面就有，其他编码标准应该也有）

int level：级（和profile差不太多）

AVCodec：

const char \*name：编解码器的名字，比较短

const char \*long\_name：编解码器的名字，全称，比较长

enum AVMediaType type：指明了类型，是视频，音频，还是字幕

enum AVCodecID id：ID，不重复

const AVRational \*supported\_framerates：支持的帧率（仅视频）

const enum AVPixelFormat \*pix\_fmts：支持的像素格式（仅视频）

const int \*supported\_samplerates：支持的采样率（仅音频）

const enum AVSampleFormat \*sample\_fmts：支持的采样格式（仅音频）

const uint64\_t \*channel\_layouts：支持的声道数（仅音频）

int priv\_data\_size：私有数据的大小

AVPacket：

uint8\_t \*data：压缩编码的数据。

例如对于H.264来说。1个AVPacket的data通常对应一个NAL。

注意：在这里只是对应，而不是一模一样。他们之间有微小的差别：使用FFMPEG类库分离出多媒体文件中的H.264码流

因此在使用FFMPEG进行视音频处理的时候，常常可以将得到的AVPacket的data数据直接写成文件，从而得到视音频的码流文件。

int size：data的大小

int64\_t pts：显示时间戳

int64\_t dts：解码时间戳

int stream\_index：标识该AVPacket所属的视频/音频流。

AVFrame：

AVFrame结构体一般用于存储原始数据（即非压缩数据，例如对视频来说是YUV，RGB，对音频来说是PCM），

此外还包含了一些相关的信息。比如说，解码的时候存储了宏块类型表，QP表，运动矢量表等数据。

编码的时候也存储了相关的数据。因此在使用FFMPEG进行码流分析的时候，AVFrame是一个很重要的结构体。

uint8\_t \*data[AV\_NUM\_DATA\_POINTERS]：解码后原始数据（对视频来说是YUV，RGB，对音频来说是PCM）

int linesize[AV\_NUM\_DATA\_POINTERS]：data中“一行”数据的大小。注意：未必等于图像的宽，一般大于图像的宽。

int width, height：视频帧宽和高（1920x1080,1280x720...）

int nb\_samples：音频的一个AVFrame中可能包含多个音频帧，在此标记包含了几个

int format：解码后原始数据类型（YUV420，YUV422，RGB24...）

int key\_frame：是否是关键帧

enum AVPictureType pict\_type：帧类型（I,B,P...）

AVRational sample\_aspect\_ratio：宽高比（16:9，4:3...）

int64\_t pts：显示时间戳

int coded\_picture\_number：编码帧序号

int display\_picture\_number：显示帧序号

int8\_t \*qscale\_table：QP表

uint8\_t \*mbskip\_table：跳过宏块表

int16\_t (\*motion\_val[2])[2]：运动矢量表

uint32\_t \*mb\_type：宏块类型表

short \*dct\_coeff：DCT系数，这个没有提取过

int8\_t \*ref\_index[2]：运动估计参考帧列表（貌似H.264这种比较新的标准才会涉及到多参考帧）

int interlaced\_frame：是否是隔行扫描

uint8\_t motion\_subsample\_log2：一个宏块中的运动矢量采样个数，取log的

ffmpeg 编码器

流程：

av\_register\_all()：注册FFmpeg所有编解码器。

avformat\_alloc\_output\_context2()：初始化输出码流的AVFormatContext。

avio\_open()：打开输出文件。

av\_new\_stream()：创建输出码流的AVStream。

avcodec\_find\_encoder()：查找编码器。

avcodec\_open2()：打开编码器。

avformat\_write\_header()：写文件头（对于某些没有文件头的封装格式，不需要此函数。比如说MPEG2TS）。

avcodec\_encode\_video2()：编码一帧视频。即将AVFrame（存储YUV像素数据）编码为AVPacket（存储H.264等格式的码流数据）。

av\_write\_frame()：将编码后的视频码流写入文件。

flush\_encoder()：输入的像素数据读取完成后调用此函数。用于输出编码器中剩余的AVPacket。

av\_write\_trailer()：写文件尾（对于某些没有文件头的封装格式，不需要此函数。比如说MPEG2TS）。

**FFmpeg内部判断封装格式的原理实际上是对每种AVInputFormat给出一个分数，满分是100分，越有可能正确的AVInputFormat给出的分数就越高。最后选择分数最高的AVInputFormat作为推测结果。**