**1)  AVFormatContext**

在使用FFMPEG进行开发的时候，AVFormatContext是一个贯穿始终的数据结构，很多函数都要用到它作为参数。它是FFMPEG解封装（flv，mp4，rmvb，avi）功能的结构体。下面看几个主要变量的作用（在这里考虑解码的情况）：

struct AVInputFormat \*iformat：输入数据的封装格式

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

**2）AVIOContext结构体，URLContext结构体，URLProtocol结构体**

AVIOContext结构体有以下几个变量比较重要：

unsigned char \*buffer：缓存开始位置

int buffer\_size：缓存大小（默认32768）

unsigned char \*buf\_ptr：当前指针读取到的位置

unsigned char \*buf\_end：缓存结束的位置

void \*opaque：URLContext结构体

在解码的情况下，buffer用于存储ffmpeg读入的数据。例如打开一个视频文件的时候，先把数据从硬盘读入buffer，然后在送给解码器用于解码。

其中opaque指向了URLContext。注意，这个结构体并不在FFMPEG提供的头文件中，而是在FFMPEG的源代码中。从FFMPEG源代码中翻出的定义如下所示：

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/14215369) [copy](http://blog.csdn.net/leixiaohua1020/article/details/14215369)

1. **typedef** **struct** URLContext {
2. **const** AVClass \*av\_class; ///< information for av\_log(). Set by url\_open().
3. **struct** URLProtocol \*prot;
4. **int** flags;
5. **int** is\_streamed;  /\*\*< true if streamed (no seek possible), default = false \*/
6. **int** max\_packet\_size;  /\*\*< if non zero, the stream is packetized with this max packet size \*/
7. **void** \*priv\_data;
8. **char** \*filename; /\*\*< specified URL \*/
9. **int** is\_connected;
10. AVIOInterruptCB interrupt\_callback;
11. } URLContext;

URLContext结构体中还有一个结构体URLProtocol。注：每种协议（rtp，rtmp，file等）对应一个URLProtocol。这个结构体也不在FFMPEG提供的头文件中。从FFMPEG源代码中翻出其的定义：

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/14215369) [copy](http://blog.csdn.net/leixiaohua1020/article/details/14215369)

1. **typedef** **struct** URLProtocol {
2. **const** **char** \*name;
3. **int** (\*url\_open)(URLContext \*h, **const** **char** \*url, **int** flags);
4. **int** (\*url\_read)(URLContext \*h, unsigned **char** \*buf, **int** size);
5. **int** (\*url\_write)(URLContext \*h, **const** unsigned **char** \*buf, **int** size);
6. int64\_t (\*url\_seek)(URLContext \*h, int64\_t pos, **int** whence);
7. **int** (\*url\_close)(URLContext \*h);
8. **struct** URLProtocol \*next;
9. **int** (\*url\_read\_pause)(URLContext \*h, **int** pause);
10. int64\_t (\*url\_read\_seek)(URLContext \*h, **int** stream\_index,
11. int64\_t timestamp, **int** flags);
12. **int** (\*url\_get\_file\_handle)(URLContext \*h);
13. **int** priv\_data\_size;
14. **const** AVClass \*priv\_data\_class;
15. **int** flags;
16. **int** (\*url\_check)(URLContext \*h, **int** mask);
17. } URLProtocol;

在这个结构体中，除了一些回调函数接口之外，有一个变量const char \*name，该变量存储了协议的名称。每一种输入协议都对应这样一个结构体。

比如说，文件协议中代码如下（file.c）：

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/14215369) [copy](http://blog.csdn.net/leixiaohua1020/article/details/14215369)

1. URLProtocol ff\_file\_protocol = {
2. .name                = "file",
3. .url\_open            = file\_open,
4. .url\_read            = file\_read,
5. .url\_write           = file\_write,
6. .url\_seek            = file\_seek,
7. .url\_close           = file\_close,
8. .url\_get\_file\_handle = file\_get\_handle,
9. .url\_check           = file\_check,
10. };

**3）AVStream结构体**

AVStream是存储每一个视频/音频流信息的结构体

AVStream重要的变量如下所示：

int index：标识该视频/音频流

AVCodecParameters \*codecpar: 指向该视频/音频流编解码信息，比如视频帧格式，宽高，bit\_rate，音频采样率，采样格式，用来初始化编解码器

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。

4**）AVCodecContext结构体**

AVCodecContext是包含变量较多的结构体（感觉差不多是变量最多的结构体），下面挑一些关键的变量来看看（这里只考虑解码）

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 sample\_rate：采样率（音频）

int channels：声道数（音频）

enum AVSampleFormat sample\_fmt：采样格式

在这里需要注意：AVCodecContext中很多的参数是编码的时候使用的，而不是解码的时候使用的。

5**）AVCodec结构体**

AVCodec是存储编解码器信息的结构体

下面说一下最主要的几个变量：

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

6）AVPacket结构体

AVPacket是存储压缩编码数据相关信息的结构体。

在AVPacket结构体中，重要的变量有以下几个：

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](http://blog.csdn.net/leixiaohua1020/article/details/11800877)

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。

说明：FFMPEG使用AVPacket来暂存解复用之后、解码之前的媒体数据（一个音/视频帧、一个字幕包等）及附加信息（解码时间戳、显示时间戳、时长等）。其中：

* dts表示解码时间戳，pts表示显示时间戳，它们的单位是所属媒体流的时间基准。
* stream\_index给出所属媒体流的索引；
* data为数据缓冲区指针，size为长度；
* duration为数据的时长，也是以所属媒体流的时间基准为单位；
* pos表示该数据在媒体流中的字节偏移量；
* destruct为用于释放数据缓冲区的函数指针；
* flags为标志域，其中，最低为置1表示该数据是一个关键帧。

AVPacket结构本身只是个容器，它使用data成员引用实际的数据缓冲区。这个缓冲区通常是由av\_new\_packet创建的，但也可能由 FFMPEG的API创建（如av\_read\_frame）。当某个AVPacket结构的数据缓冲区不再被使用时，要需要通过调用 av\_free\_packet释放。av\_free\_packet调用的是结构体本身的destruct函数，它的值有两种情 况：1)av\_destruct\_packet\_nofree或0；2)av\_destruct\_packet，其中，情况1)仅仅是将data和 size的值清0而已，情况2)才会真正地释放缓冲区。

FFMPEG内部使用AVPacket结构建立缓冲区装载数据，同时提供destruct函数，如果FFMPEG打算自己维护缓冲区，则将 destruct设为av\_destruct\_packet\_nofree，用户调用av\_free\_packet清理缓冲区时并不能够将其释放；如果 FFMPEG打算将该缓冲区彻底交给调用者，则将destruct设为av\_destruct\_packet，表示它能够被释放。安全起见，如果用户希望 自由地使用一个FFMPEG内部创建的AVPacket结构，最好调用av\_dup\_packet进行缓冲区的克隆，将其转化为缓冲区能够被释放的 AVPacket，以免对缓冲区的不当占用造成异常错误。av\_dup\_packet会为destruct指针为 av\_destruct\_packet\_nofree的AVPacket新建一个缓冲区，然后将原缓冲区的数据拷贝至新缓冲区，置data的值为新缓冲区 的地址，同时设destruct指针为av\_destruct\_packet。

**7）AVFrame结构体**

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。

下面看几个主要变量的作用（在这里考虑解码的情况）：

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

其他的变量不再一一列举，源代码中都有详细的说明。在这里重点分析一下几个需要一定的理解的变量：

**1.data[]**

对于packed格式的数据（例如RGB24），会存到data[0]里面。

对于planar格式的数据（例如YUV420P），则会分开成data[0]，data[1]，data[2]..（YUV420P中data[0]存Y，data[1]存U，data[2]存V）

PTS计算：

avframe→pts\*av\_q2d(avStream.time\_base);

关于AVFrame的使用：

av\_image\_alloc;

av\_sample\_alloc;