# [ffmpeg函数介绍](http://blog.csdn.net/leixiaohua1020/article/details/11979565)

本文对在使用ffmpeg进行音视频编解码时使用到的一些函数做一个简单介绍,我当前使用的ffmpeg版本为:0.8.5,因为本人发现在不同的版本中,有些函数名称会有点小改动,所以在此有必要说明下ffmpeg的版本号。

ffmpeg本人也是刚接触，本文将采用累加的方法逐个介绍我使用到的函数，如有不妥之处，还望谅解！

头文件引入方法:

extern "C" {

#include "libavcodec/avcodec.h"

#include "libavformat/avformat.h"

#include "libavutil/avutil.h"

#include "libavutil/mem.h"

#include "libavutil/fifo.h"

#include "libswscale/swscale.h"

};

**1 avcodec\_init()**

/\*\*

 \* Initialize libavcodec.

 \* If called more than once, does nothing.

 \*

 \* @warning This function must be called before any other libavcodec

 \* function.

 \*

 \* @warning This function is not thread-safe.

 \*/

void avcodec\_init(void);

// 初始化libavcodec,一般最先调用该函数

// 引入头文件： #include "libavcodec/avcodec.h"

// 实现在: \ffmpeg\libavcodec\utils.c

// 该函数必须在调用libavcodec里的其它函数前调用,一般在程序启动或模块初始化时调用,如果你调用了多次也无所谓,因为后面的调用不会做任何事情.从函数的实现里你可以发现,代码中对多次调用进行了控制.

// 该函数是非线程安全的

**2 av\_register\_all()**

/\*\*

 \* Initialize libavformat and register all the muxers, demuxers and

 \* protocols. If you do not call this function, then you can select

 \* exactly which formats you want to support.

 \*

 \* @see av\_register\_input\_format()

 \* @see av\_register\_output\_format()

 \* @see av\_register\_protocol()

 \*/

void av\_register\_all(void);

// 初始化 libavformat和注册所有的muxers、demuxers和protocols，

// 一般在调用avcodec\_init后调用该方法

// 引入头文件：#include "libavformat/avformat.h"

// 实现在:\ffmpeg\libavformat\allformats.c

// 其中会调用avcodec\_register\_all()注册多种音视频格式的编解码器,并注册各种文件的编解复用器

// 当然，你也可以不调用该函数，而通过选择调用特定的方法来提供支持

**3 avformat\_alloc\_context()**

/\*\*

 \* Allocate an AVFormatContext.

 \* avformat\_free\_context() can be used to free the context and everything

 \* allocated by the framework within it.

 \*/

AVFormatContext \*avformat\_alloc\_context(void);

// 分配一个AVFormatContext结构

// 引入头文件：#include "libavformat/avformat.h"

// 实现在:\ffmpeg\libavformat\options.c

// 其中负责申请一个AVFormatContext结构的内存,并进行简单初始化

// avformat\_free\_context()可以用来释放该结构里的所有东西以及该结构本身

// 也是就说使用 avformat\_alloc\_context()分配的结构,需要使用avformat\_free\_context()来释放

// 有些版本中函数名可能为: av\_alloc\_format\_context();

**4 avformat\_free\_context()**

/\*\*

 \* Free an AVFormatContext and all its streams.

 \* @param s context to free

 \*/

void avformat\_free\_context(AVFormatContext \*s);

// 释放一个AVFormatContext结构

// 引入头文件：#include "libavformat/avformat.h"

// 实现在:\ffmpeg\libavformat\utils.c

// 使用 avformat\_alloc\_context()分配的结构,采用该函数进行释放,除释放AVFormatContext结构本身内存之外,AVFormatContext中指针所指向的内存也会一并释放

// 有些版本中函数名猜测可能为: av\_free\_format\_context();

**5 AVFormatContext 结构**

/\*\*

 \* Format I/O context.

 \* New fields can be added to the end with minor version bumps.

 \* Removal, reordering and changes to existing fields require a major

 \* version bump.

 \* sizeof(AVFormatContext) must not be used outside libav\*.

 \*/

typedef struct AVFormatContext {

    struct AVInputFormat \*iformat;

    struct AVOutputFormat \*oformat;

    AVIOContext \*pb;

    unsigned int nb\_streams;

    AVStream \*\*streams;

    char filename[1024]; /\*\*< input or output filename \*/

    ....

} AVFormatContext;

// AVFormatContext在FFMpeg里是一个非常重要的的结构，是其它输入、输出相关信息的一个容器

// 引入头文件：#include "libavformat/avformat.h"

// 以上只列出了其中的部分成员

// 作为输入容器时 struct AVInputFormat \*iformat; 不能为空, 其中包含了输入文件的音视频流信息,程序从输入容器从读出音视频包进行解码处理

// 作为输出容器时 struct AVOutputFormat \*oformat; 不能为空, 程序把编码好的音视频包写入到输出容器中

// AVIOContext \*pb: I/O上下文,通过对该变量赋值可以改变输入源或输出目的

// unsigned int nb\_streams; 音视频流数量

// AVStream \*\*streams; 音视频流

**6 AVIOContext 结构**

/\*\*

 \* Bytestream IO Context.

 \* New fields can be added to the end with minor version bumps.

 \* Removal, reordering and changes to existing fields require a major

 \* version bump.

 \* sizeof(AVIOContext) must not be used outside libav\*.

 \*

 \* @note None of the function pointers in AVIOContext should be called

 \*       directly, they should only be set by the client application

 \*       when implementing custom I/O. Normally these are set to the

 \*       function pointers specified in avio\_alloc\_context()

 \*/

typedef struct {

    unsigned char \*buffer;  /\*\*< Start of the buffer. \*/

    int buffer\_size;        /\*\*< Maximum buffer size \*/

    unsigned char \*buf\_ptr; /\*\*< Current position in the buffer \*/

    unsigned char \*buf\_end; /\*\*< End of the data, may be less than

                                 buffer+buffer\_size if the read function returned

                                 less data than requested, e.g. for streams where

                                 no more data has been received yet. \*/

    void \*opaque;           /\*\*< A private pointer, passed to the read/write/seek/...

                                 functions. \*/

    int (\*read\_packet)(void \*opaque, uint8\_t \*buf,int buf\_size);

    int (\*write\_packet)(void \*opaque, uint8\_t \*buf,int buf\_size);

    int64\_t (\*seek)(void \*opaque, int64\_t offset,int whence);

    int64\_t pos;            /\*\*< position in the file of the current buffer \*/

    int must\_flush;         /\*\*< true if the next seek should flush \*/

    int eof\_reached;        /\*\*< true if eof reached \*/

    int write\_flag;         /\*\*< true if open for writing \*/

#if FF\_API\_OLD\_AVIO

    attribute\_deprecated int is\_streamed;

#endif

    int max\_packet\_size;

    unsigned long checksum;

    unsigned char \*checksum\_ptr;

    unsigned long (\*update\_checksum)(unsigned long checksum, const uint8\_t \*buf, unsigned int size);

    int error;              /\*\*< contains the error code or 0 if no error happened \*/

    /\*\*

     \* Pause or resume playback for network streaming protocols - e.g. MMS.

     \*/

    int (\*read\_pause)(void \*opaque,int pause);

    /\*\*

     \* Seek to a given timestamp in stream with the specified stream\_index.

     \* Needed for some network streaming protocols which don't support seeking

     \* to byte position.

     \*/

    int64\_t (\*read\_seek)(void \*opaque,int stream\_index,

                         int64\_t timestamp, int flags);

    /\*\*

     \* A combination of AVIO\_SEEKABLE\_ flags or 0 when the stream is not seekable.

     \*/

    int seekable;

} AVIOContext;

// 字节流 I/O 上下文

// 在结构的尾部增加变量可以减少版本冲突

// 移除、排序和修改已经存在的变量将会导致较大的版本冲突

// sizeof(AVIOContext)在libav\*.外部不可使用

// AVIOContext里的函数指针不能直接调用,通常使用avio\_alloc\_context()函数来设置其中的函数指针

// unsigned char \*buffer: 缓存的起始指针

// int buffer\_size: 缓存的最大值

// void \*opaque: 在回调函数中使用的指针

// int (\*read\_packet)(void \*opaque, uint8\_t \*buf,int buf\_size): 读文件回调方法

// int (\*write\_packet)(void \*opaque, uint8\_t \*buf,int buf\_size): 写文件回调方法

// int64\_t (\*seek)(void \*opaque, int64\_t offset,int whence): seek文件回调方法

**7 avio\_alloc\_context()**

/\*\*

 \* Allocate and initialize an AVIOContext for buffered I/O. It must be later

 \* freed with av\_free().

 \*

 \* @param buffer Memory block for input/output operations via AVIOContext.

 \*        The buffer must be allocated with av\_malloc() and friends.

 \* @param buffer\_size The buffer size is very important for performance.

 \*        For protocols with fixed blocksize it should be set to this blocksize.

 \*        For others a typical size is a cache page, e.g. 4kb.

 \* @param write\_flag Set to 1 if the buffer should be writable, 0 otherwise.

 \* @param opaque An opaque pointer to user-specific data.

 \* @param read\_packet  A function for refilling the buffer, may be NULL.

 \* @param write\_packet A function for writing the buffer contents, may be NULL.

 \* @param seek A function for seeking to specified byte position, may be NULL.

 \*

 \* @return Allocated AVIOContext or NULL on failure.

 \*/

AVIOContext \*avio\_alloc\_context(

                  unsigned char \*buffer,

                  int buffer\_size,

                  int write\_flag,

                  void \*opaque,

                  int (\*read\_packet)(void \*opaque, uint8\_t \*buf,int buf\_size),

                  int (\*write\_packet)(void \*opaque, uint8\_t \*buf,int buf\_size),

                  int64\_t (\*seek)(void \*opaque, int64\_t offset,int whence));

// 为I/0缓存申请并初始化一个AVIOContext结构,结束使用时必须使用av\_free()进行释放

// unsigned char \*buffer: 输入/输出缓存内存块,必须是使用av\_malloc()分配的

// int buffer\_size: 缓存大小是非常重要的

// int write\_flag: 如果缓存为可写则设置为1,否则设置为0

// void \*opaque: 指针,用于回调时使用

// int (\*read\_packet): 读包函数指针

// int (\*write\_packet): 写包函数指针

// int64\_t (\*seek): seek文件函数指针

**8 av\_open\_input\_file()**

/\*\*

 \* Open a media file as input. The codecs are not opened. Only the file

 \* header (if present) is read.

 \*

 \* @param ic\_ptr The opened media file handle is put here.

 \* @param filename filename to open

 \* @param fmt If non-NULL, force the file format to use.

 \* @param buf\_size optional buffer size (zero if default is OK)

 \* @param ap Additional parameters needed when opening the file

 \*           (NULL if default).

 \* @return 0 if OK, AVERROR\_xxx otherwise

 \*

 \* @deprecated use avformat\_open\_input instead.

 \*/

attribute\_deprecated int av\_open\_input\_file(AVFormatContext \*\*ic\_ptr,constchar \*filename,

                       AVInputFormat \*fmt,

                       int buf\_size,

                       AVFormatParameters \*ap);

// 以输入方式打开一个媒体文件,也即源文件,codecs并没有打开,只读取了文件的头信息.

// 引入头文件：#include "libavformat/avformat.h"

// AVFormatContext \*\*ic\_ptr 输入文件容器

//constchar \*filename 输入文件名,全路径,并且保证文件存在

// AVInputFormat \*fmt 输入文件格式,填NULL即可

//int buf\_size,缓冲区大小,直接填0即可

// AVFormatParameters \*ap, 格式参数,添NULL即可

// 成功返回0,其它失败

// 不赞成使用 avformat\_open\_input 代替

**9 av\_close\_input\_file()**

/\*\*

 \* @deprecated use avformat\_close\_input()

 \* Close a media file (but not its codecs).

 \* @param s media file handle

 \*/

void av\_close\_input\_file(AVFormatContext \*s);

// 关闭使用avformat\_close\_input()打开的输入文件容器,但并不关系它的codecs

// 引入头文件：#include "libavformat/avformat.h"

// 使用av\_open\_input\_file 打开的文件容器,可以使用该函数关闭

// 使用 av\_close\_input\_file 关闭后,就不再需要使用avformat\_free\_context 进行释放了

**10 av\_find\_stream\_info()**

/\*\*

 \* Read packets of a media file to get stream information. This

 \* is useful for file formats with no headers such as MPEG. This

 \* function also computes the real framerate in case of MPEG-2 repeat

 \* frame mode.

 \* The logical file position is not changed by this function;

 \* examined packets may be buffered for later processing.

 \*

 \* @param ic media file handle

 \* @return >=0 if OK, AVERROR\_xxx on error

 \* @todo Let the user decide somehow what information is needed so that

 \*       we do not waste time getting stuff the user does not need.

 \*/

int av\_find\_stream\_info(AVFormatContext \*ic);

// 通过读取媒体文件的中的包来获取媒体文件中的流信息,对于没有头信息的文件如(mpeg)是非常有用的,

// 该函数通常重算类似mpeg-2帧模式的真实帧率,该函数并未改变逻辑文件的position.

// 引入头文件：#include "libavformat/avformat.h"

// 也就是把媒体文件中的音视频流等信息读出来,保存在容器中,以便解码时使用

// 返回>=0时成功,否则失败

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**1 avcodec\_find\_decoder()**

/\*\*

 \* Find a registered decoder with a matching codec ID.

 \*

 \* @param id CodecID of the requested decoder

 \* @return A decoder if one was found, NULL otherwise.

 \*/

AVCodec \*avcodec\_find\_decoder(enum CodecID id);

// 通过code ID查找一个已经注册的音视频解码器

// 引入 #include "libavcodec/avcodec.h"

// 实现在: \ffmpeg\libavcodec\utils.c

// 查找解码器之前,必须先调用av\_register\_all注册所有支持的解码器

// 查找成功返回解码器指针,否则返回NULL

// 音视频解码器保存在一个链表中,查找过程中,函数从头到尾遍历链表,通过比较解码器的ID来查找

**2 avcodec\_find\_decoder\_by\_name()**

/\*\*

 \* Find a registered decoder with the specified name.

 \*

 \* @param name name of the requested decoder

 \* @return A decoder if one was found, NULL otherwise.

 \*/

AVCodec \*avcodec\_find\_decoder\_by\_name(constchar \*name);

// 通过一个指定的名称查找一个已经注册的音视频解码器

// 引入 #include "libavcodec/avcodec.h"

// 实现在: \ffmpeg\libavcodec\utils.c

// 查找解码器之前,必须先调用av\_register\_all注册所有支持的解码器

// 查找成功返回解码器指针,否则返回NULL

// 音视频解码器保存在一个链表中,查找过程中,函数从头到尾遍历链表,通过比较解码器的name来查找

**3 avcodec\_find\_encoder()**

/\*\*

 \* Find a registered encoder with a matching codec ID.

 \*

 \* @param id CodecID of the requested encoder

 \* @return An encoder if one was found, NULL otherwise.

 \*/

AVCodec \*avcodec\_find\_encoder(enum CodecID id);

// 通过code ID查找一个已经注册的音视频编码器

// 引入 #include "libavcodec/avcodec.h"

// 实现在: \ffmpeg\libavcodec\utils.c

// 查找编码器之前,必须先调用av\_register\_all注册所有支持的编码器

// 查找成功返回编码器指针,否则返回NULL

// 音视频编码器保存在一个链表中,查找过程中,函数从头到尾遍历链表,通过比较编码器的ID来查找

**4 avcodec\_find\_encoder\_by\_name()**

/\*\*

 \* Find a registered encoder with the specified name.

 \*

 \* @param name name of the requested encoder

 \* @return An encoder if one was found, NULL otherwise.

 \*/

AVCodec \*avcodec\_find\_encoder\_by\_name(constchar \*name);

// 通过一个指定的名称查找一个已经注册的音视频编码器

// 引入 #include "libavcodec/avcodec.h"

// 实现在: \ffmpeg\libavcodec\utils.c

// 查找编码器之前,必须先调用av\_register\_all注册所有支持的编码器

// 查找成功返回编码器指针,否则返回NULL

// 音视频编码器保存在一个链表中,查找过程中,函数从头到尾遍历链表,通过比较编码器的名称来查找

**5 avcodec\_open()**

/\*\*

 \* Initialize the AVCodecContext to use the given AVCodec. Prior to using this

 \* function the context has to be allocated.

 \*

 \* The functions avcodec\_find\_decoder\_by\_name(), avcodec\_find\_encoder\_by\_name(),

 \* avcodec\_find\_decoder() and avcodec\_find\_encoder() provide an easy way for

 \* retrieving a codec.

 \*

 \* @warning This function is not thread safe!

 \*

 \* @code

 \* avcodec\_register\_all();

 \* codec = avcodec\_find\_decoder(CODEC\_ID\_H264);

 \* if (!codec)

 \*     exit(1);

 \*

 \* context = avcodec\_alloc\_context();

 \*

 \* if (avcodec\_open(context, codec) < 0)

 \*     exit(1);

 \* @endcode

 \*

 \* @param avctx The context which will be set up to use the given codec.

 \* @param codec The codec to use within the context.

 \* @return zero on success, a negative value on error

 \* @see avcodec\_alloc\_context, avcodec\_find\_decoder, avcodec\_find\_encoder, avcodec\_close

 \*/

int avcodec\_open(AVCodecContext \*avctx, AVCodec \*codec);

// 使用给定的AVCodec初始化AVCodecContext

// 引入#include "libavcodec/avcodec.h"

// 方法: avcodec\_find\_decoder\_by\_name(), avcodec\_find\_encoder\_by\_name(), avcodec\_find\_decoder() and avcodec\_find\_encoder() 提供了快速获取一个codec的途径

// 该方法在编码和解码时都会用到

// 返回0时成功,打开作为输出时,参数设置不对的话,调用会失败

**6 av\_guess\_format()**

/\*\*

 \* Return the output format in the list of registered output formats

 \* which best matches the provided parameters, or return NULL if

 \* there is no match.

 \*

 \* @param short\_name if non-NULL checks if short\_name matches with the

 \* names of the registered formats

 \* @param filename if non-NULL checks if filename terminates with the

 \* extensions of the registered formats

 \* @param mime\_type if non-NULL checks if mime\_type matches with the

 \* MIME type of the registered formats

 \*/

AVOutputFormat \*av\_guess\_format(constchar \*short\_name,

                                constchar \*filename,

                                constchar \*mime\_type);

// 返回一个已经注册的最合适的输出格式

// 引入#include "libavformat/avformat.h"

// 可以通过 const char \*short\_name 获取,如"mpeg"

// 也可以通过 const char \*filename 获取,如"E:\a.mp4"

**7 av\_new\_stream()**

/\*\*

 \* Add a new stream to a media file.

 \*

 \* Can only be called in the read\_header() function. If the flag

 \* AVFMTCTX\_NOHEADER is in the format context, then new streams

 \* can be added in read\_packet too.

 \*

 \* @param s media file handle

 \* @param id file-format-dependent stream ID

 \*/

AVStream \*av\_new\_stream(AVFormatContext \*s, int id);

// 为媒体文件添加一个流,一般为作为输出的媒体文件容器添加音视频流

// 引入 #include "libavformat/avformat.h"

// 再打开源文件时用户一般不需要直接调用该方法

**8 dump\_format()**

#if FF\_API\_DUMP\_FORMAT

/\*\*

 \* @deprecated Deprecated in favor of av\_dump\_format().

 \*/

attribute\_deprecated void dump\_format(AVFormatContext \*ic,

                                      int index,

                                      constchar \*url,

                                      int is\_output);

#endif

// 该函数的作用就是检查下初始化过程中设置的参数是否符合规范

// 有些版本中为 av\_dump\_format

**9 av\_set\_parameters()**

#if FF\_API\_FORMAT\_PARAMETERS

/\*\*

 \* @deprecated pass the options to avformat\_write\_header directly.

 \*/

attribute\_deprecated int av\_set\_parameters(AVFormatContext \*s, AVFormatParameters \*ap);

#endif

// 设置初始化参数

// 不赞成跳过该方法,直接调用 avformat\_write\_header/av\_write\_header

**10 av\_write\_header()**

#if FF\_API\_FORMAT\_PARAMETERS

/\*\*

 \* Allocate the stream private data and write the stream header to an

 \* output media file.

 \* @note: this sets stream time-bases, if possible to stream->codec->time\_base

 \* but for some formats it might also be some other time base

 \*

 \* @param s media file handle

 \* @return 0 if OK, AVERROR\_xxx on error

 \*

 \* @deprecated use avformat\_write\_header.

 \*/

attribute\_deprecated int av\_write\_header(AVFormatContext \*s);

#endif

// 把流头信息写入到媒体文件中

// 返回0成功

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**1 AVPacket**

typedef struct AVPacket {

    /\*\*

     \* Presentation timestamp in AVStream->time\_base units; the time at which

     \* the decompressed packet will be presented to the user.

     \* Can be AV\_NOPTS\_VALUE if it is not stored in the file.

     \* pts MUST be larger or equal to dts as presentation cannot happen before

     \* decompression, unless one wants to view hex dumps. Some formats misuse

     \* the terms dts and pts/cts to mean something different. Such timestamps

     \* must be converted to true pts/dts before they are stored in AVPacket.

     \*/

    int64\_t pts;

    /\*\*

     \* Decompression timestamp in AVStream->time\_base units; the time at which

     \* the packet is decompressed.

     \* Can be AV\_NOPTS\_VALUE if it is not stored in the file.

     \*/

    int64\_t dts;

    uint8\_t \*data;

    int   size;

    int   stream\_index;

    int   flags;

int   duration;

.

.

.

} AVPacket

// AVPacket是个很重要的结构,该结构在读媒体源文件和写输出文件时都需要用到

// int64\_t pts; 显示时间戳

// int64\_t dts; 解码时间戳

// uint8\_t \*data; 包数据

// int   size; 包数据长度

// int   stream\_index; 包所属流序号

// int   duration; 时长

// 以上信息,如果是在读媒体源文件那么avcodec会初始化,如果是输出文件,用户需要对以上信息赋值

**2 av\_init\_packet()**

/\*\*

 \* Initialize optional fields of a packet with default values.

 \*

 \* @param pkt packet

 \*/

void av\_init\_packet(AVPacket \*pkt);

// 使用默认值初始化AVPacket

// 定义AVPacket对象后,请使用av\_init\_packet进行初始化

**3 av\_free\_packet()**

/\*\*

 \* Free a packet.

 \*

 \* @param pkt packet to free

 \*/

void av\_free\_packet(AVPacket \*pkt);

// 释放AVPacket对象

**4 av\_read\_frame()**

/\*\*

 \* Return the next frame of a stream.

 \* This function returns what is stored in the file, and does not validate

 \* that what is there are valid frames for the decoder. It will split what is

 \* stored in the file into frames and return one for each call. It will not

 \* omit invalid data between valid frames so as to give the decoder the maximum

 \* information possible for decoding.

 \*

 \* The returned packet is valid

 \* until the next av\_read\_frame() or until av\_close\_input\_file() and

 \* must be freed with av\_free\_packet. For video, the packet contains

 \* exactly one frame. For audio, it contains an integer number of

 \* frames if each frame has a known fixed size (e.g. PCM or ADPCM

 \* data). If the audio frames have a variable size (e.g. MPEG audio),

 \* then it contains one frame.

 \*

 \* pkt->pts, pkt->dts and pkt->duration are always set to correct

 \* values in AVStream.time\_base units (and guessed if the format cannot

 \* provide them). pkt->pts can be AV\_NOPTS\_VALUE if the video format

 \* has B-frames, so it is better to rely on pkt->dts if you do not

 \* decompress the payload.

 \*

 \* @return 0 if OK, < 0 on error or end of file

 \*/

int av\_read\_frame(AVFormatContext \*s, AVPacket \*pkt);

// 从输入源文件容器中读取一个AVPacket数据包

// 该函数读出的包并不每次都是有效的,对于读出的包我们都应该进行相应的解码(视频解码/音频解码),

// 在返回值>=0时,循环调用该函数进行读取,循环调用之前请调用av\_free\_packet函数清理AVPacket

**5 avcodec\_decode\_video2()**

/\*\*

 \* Decode the video frame of size avpkt->size from avpkt->data into picture.

 \* Some decoders may support multiple frames in a single AVPacket, such

 \* decoders would then just decode the first frame.

 \*

 \* @warning The input buffer must be FF\_INPUT\_BUFFER\_PADDING\_SIZE larger than

 \* the actual read bytes because some optimized bitstream readers read 32 or 64

 \* bits at once and could read over the end.

 \*

 \* @warning The end of the input buffer buf should be set to 0 to ensure that

 \* no overreading happens for damaged MPEG streams.

 \*

 \* @note You might have to align the input buffer avpkt->data.

 \* The alignment requirements depend on the CPU: on some CPUs it isn't

 \* necessary at all, on others it won't work at all if not aligned and on others

 \* it will work but it will have an impact on performance.

 \*

 \* In practice, avpkt->data should have 4 byte alignment at minimum.

 \*

 \* @note Some codecs have a delay between input and output, these need to be

 \* fed with avpkt->data=NULL, avpkt->size=0 at the end to return the remaining frames.

 \*

 \* @param avctx the codec context

 \* @param[out] picture The AVFrame in which the decoded video frame will be stored.

 \*             Use avcodec\_alloc\_frame to get an AVFrame, the codec will

 \*             allocate memory for the actual bitmap.

 \*             with default get/release\_buffer(), the decoder frees/reuses the bitmap as it sees fit.

 \*             with overridden get/release\_buffer() (needs CODEC\_CAP\_DR1) the user decides into what buffer the decoder

 \*                   decodes and the decoder tells the user once it does not need the data anymore,

 \*                   the user app can at this point free/reuse/keep the memory as it sees fit.

 \*

 \* @param[in] avpkt The input AVpacket containing the input buffer.

 \*            You can create such packet with av\_init\_packet() and by then setting

 \*            data and size, some decoders might in addition need other fields like

 \*            flags&AV\_PKT\_FLAG\_KEY. All decoders are designed to use the least

 \*            fields possible.

 \* @param[in,out] got\_picture\_ptr Zero if no frame could be decompressed, otherwise, it is nonzero.

 \* @return On error a negative value is returned, otherwise the number of bytes

 \* used or zero if no frame could be decompressed.

 \*/

int avcodec\_decode\_video2(AVCodecContext \*avctx, AVFrame \*picture,

                         int \*got\_picture\_ptr,

                         AVPacket \*avpkt);

// 解码视频流AVPacket

// 使用av\_read\_frame读取媒体流后需要进行判断,如果为视频流则调用该函数解码

// 返回结果<0时失败,此时程序应该退出检查原因

// 返回>=0时正常,假设 读取包为:AVPacket vPacket 返回值为 int vLen; 每次解码正常时,对vPacket做

// 如下处理:

//   vPacket.size -= vLen;  
//   vPacket.data += vLen;

// 如果 vPacket.size==0,则继续读下一流包,否则继续调度该方法进行解码,直到vPacket.size==0

// 返回 got\_picture\_ptr > 0 时,表示解码到了AVFrame \*picture,其后可以对picture进程处理

6 **avcodec\_decode\_audio3()**

/\*\*

 \* Decode the audio frame of size avpkt->size from avpkt->data into samples.

 \* Some decoders may support multiple frames in a single AVPacket, such

 \* decoders would then just decode the first frame. In this case,

 \* avcodec\_decode\_audio3 has to be called again with an AVPacket that contains

 \* the remaining data in order to decode the second frame etc.

 \* If no frame

 \* could be outputted, frame\_size\_ptr is zero. Otherwise, it is the

 \* decompressed frame size in bytes.

 \*

 \* @warning You must set frame\_size\_ptr to the allocated size of the

 \* output buffer before calling avcodec\_decode\_audio3().

 \*

 \* @warning The input buffer must be FF\_INPUT\_BUFFER\_PADDING\_SIZE larger than

 \* the actual read bytes because some optimized bitstream readers read 32 or 64

 \* bits at once and could read over the end.

 \*

 \* @warning The end of the input buffer avpkt->data should be set to 0 to ensure that

 \* no overreading happens for damaged MPEG streams.

 \*

 \* @note You might have to align the input buffer avpkt->data and output buffer

 \* samples. The alignment requirements depend on the CPU: On some CPUs it isn't

 \* necessary at all, on others it won't work at all if not aligned and on others

 \* it will work but it will have an impact on performance.

 \*

 \* In practice, avpkt->data should have 4 byte alignment at minimum and

 \* samples should be 16 byte aligned unless the CPU doesn't need it

 \* (AltiVec and SSE do).

 \*

 \* @param avctx the codec context

 \* @param[out] samples the output buffer, sample type in avctx->sample\_fmt

 \* @param[in,out] frame\_size\_ptr the output buffer size in bytes

 \* @param[in] avpkt The input AVPacket containing the input buffer.

 \*            You can create such packet with av\_init\_packet() and by then setting

 \*            data and size, some decoders might in addition need other fields.

 \*            All decoders are designed to use the least fields possible though.

 \* @return On error a negative value is returned, otherwise the number of bytes

 \* used or zero if no frame data was decompressed (used) from the input AVPacket.

 \*/

int avcodec\_decode\_audio3(AVCodecContext \*avctx, int16\_t \*samples,

                         int \*frame\_size\_ptr,

                         AVPacket \*avpkt);

// 解码音频流AVPacket

// 使用av\_read\_frame读取媒体流后需要进行判断,如果为音频流则调用该函数解码

// 返回结果<0时失败,此时程序应该退出检查原因

// 返回>=0时正常,假设 读取包为:AVPacket vPacket 返回值为 int vLen; 每次解码正常时,对vPacket做

// 如下处理:

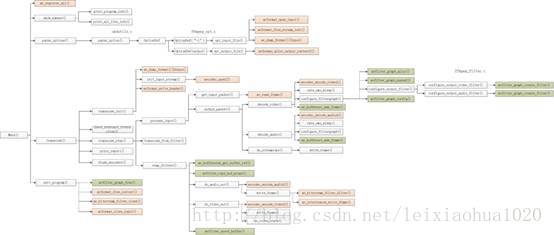
//   vPacket.size -= vLen;  
//   vPacket.data += vLen;

// 如果 vPacket.size==0,则继续读下一流包,否则继续调度该方法进行解码,直到vPacket.size==0

# [ffmpeg.c函数结构简单分析（画图）](http://blog.csdn.net/leixiaohua1020/article/details/39760711)

## 函数调用结构图

FFmpeg的总体函数调用结构图如下图所示

[](http://img.my.csdn.net/uploads/201410/03/1412348668_3334.jpg)

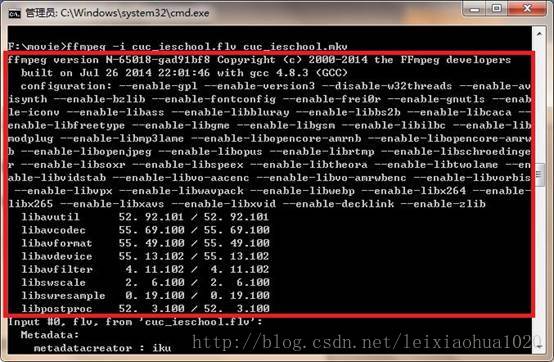
上图所示本是一张高清大图。但是页面显示不下。因此上传了一份：

<http://my.csdn.net/leixiaohua1020/album/detail/1788075>

上面地址的那张图保存下来的话就是一张清晰的图片了。

下文将会对主要函数分别解析。

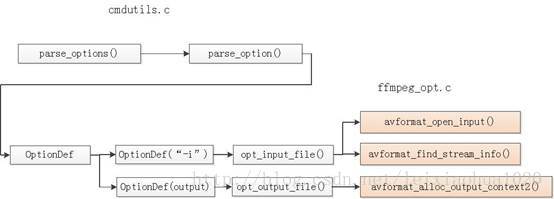
## main()

main()是FFmpeg的主函数。  
调用了如下函数  
av\_register\_all()：注册所有编码器和解码器。  
show\_banner()：打印输出FFmpeg版本信息（编译时间，编译选项，类库信息等）。  
parse\_options()：解析输入的命令。  
transcode()：转码。  
exit\_progam()：退出和清理。  
   
下图红框中的内容即为show\_banner()的输出结果。  
  
  
 

## parse\_options()

parse\_options()解析全部输入选项。即将输入命令“ffmpeg -i xxx.mpg -vcodec libx264 yyy.mkv”中的“-i”，“-vcodec”这样的命令解析出来。其函数调用结构如下图所示。

注：定义位于cmdutils.c中。



调用了如下函数：  
parse\_option()：解析一个输入选项。具体的解析步骤不再赘述。parse\_options()会循环调用parse\_option()直到所有选项解析完毕。FFmpeg的每一个选项信息存储在一个OptionDef结构体中。定义如下：

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/39760711) [copy](http://blog.csdn.net/leixiaohua1020/article/details/39760711)

[在CODE上查看代码片](https://code.csdn.net/snippets/479130)

1. **typedef** **struct** OptionDef {
2. **const** **char** \*name;
3. **int** flags;
4. #define HAS\_ARG    0x0001
5. #define OPT\_BOOL   0x0002
6. #define OPT\_EXPERT 0x0004
7. #define OPT\_STRING 0x0008
8. #define OPT\_VIDEO  0x0010
9. #define OPT\_AUDIO  0x0020
10. #define OPT\_INT    0x0080
11. #define OPT\_FLOAT  0x0100
12. #define OPT\_SUBTITLE 0x0200
13. #define OPT\_INT64  0x0400
14. #define OPT\_EXIT   0x0800
15. #define OPT\_DATA   0x1000
16. #define OPT\_PERFILE  0x2000     /\* the option is per-file (currently ffmpeg-only).
17. implied by OPT\_OFFSET or OPT\_SPEC \*/
18. #define OPT\_OFFSET 0x4000       /\* option is specified as an offset in a passed optctx \*/
19. #define OPT\_SPEC   0x8000       /\* option is to be stored in an array of SpecifierOpt.
20. Implies OPT\_OFFSET. Next element after the offset is
21. an **int** containing element count in the array. \*/
22. #define OPT\_TIME  0x10000
23. #define OPT\_DOUBLE 0x20000
24. **union** {
25. **void** \*dst\_ptr;
26. **int** (\*func\_arg)(**void** \*, **const** **char** \*, **const** **char** \*);
27. **size\_t** off;
28. } u;
29. **const** **char** \*help;
30. **const** **char** \*argname;
31. } OptionDef;

其中的重要字段：  
name：用于存储选项的名称。例如“i”，“f”，“codec”等等。  
flags：存储选项值的类型。例如：HAS\_ARG（包含选项值），OPT\_STRING（选项值为字符串类型），OPT\_TIME（选项值为时间类型。  
u：存储该选项的处理函数。  
help：选项的说明信息。  
FFmpeg使用一个名称为options，类型为OptionDef的数组存储所有的选项。有一部分通用选项存储在cmdutils\_common\_opts.h中。cmdutils\_common\_opts.h内容如下：

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/39760711) [copy](http://blog.csdn.net/leixiaohua1020/article/details/39760711)

[在CODE上查看代码片](https://code.csdn.net/snippets/479130)

1. { "L"          , OPT\_EXIT, {(**void**\*)show\_license},      "show license" },
2. { "h"          , OPT\_EXIT, {(**void**\*) show\_help},         "show help", "topic" },
3. { "?"          , OPT\_EXIT, {(**void**\*)show\_help},         "show help", "topic" },
4. { "help"       , OPT\_EXIT, {(**void**\*)show\_help},         "show help", "topic" },
5. { "-help"      , OPT\_EXIT, {(**void**\*)show\_help},         "show help", "topic" },
6. { "version"    , OPT\_EXIT, {(**void**\*)show\_version},      "show version" },
7. { "formats"    , OPT\_EXIT, {(**void**\*)show\_formats  },    "show available formats" },
8. { "codecs"     , OPT\_EXIT, {(**void**\*)show\_codecs   },    "show available codecs" },
9. { "decoders"   , OPT\_EXIT, {(**void**\*)show\_decoders },    "show available decoders" },
10. { "encoders"   , OPT\_EXIT, {(**void**\*)show\_encoders },    "show available encoders" },
11. { "bsfs"       , OPT\_EXIT, {(**void**\*)show\_bsfs     },    "show available bit stream filters" },
12. { "protocols"  , OPT\_EXIT, {(**void**\*)show\_protocols},    "show available protocols" },
13. { "filters"    , OPT\_EXIT, {(**void**\*)show\_filters  },    "show available filters" },
14. { "pix\_fmts"   , OPT\_EXIT, {(**void**\*)show\_pix\_fmts },    "show available pixel formats" },
15. { "layouts"    , OPT\_EXIT, {(**void**\*)show\_layouts  },    "show standard channel layouts" },
16. { "sample\_fmts", OPT\_EXIT, {(**void**\*)show\_sample\_fmts }, "show available audio sample formats" },
17. { "loglevel"   , HAS\_ARG,  {(**void**\*)opt\_loglevel},      "set libav\* logging level", "loglevel" },
18. { "v",           HAS\_ARG,  {(**void**\*)opt\_loglevel},      "set libav\* logging level", "loglevel" },
19. { "debug"      , HAS\_ARG,  {(**void**\*)opt\_codec\_debug},   "set debug flags", "flags" },
20. { "fdebug"     , HAS\_ARG,  {(**void**\*)opt\_codec\_debug},   "set debug flags", "flags" },
21. { "report"     , 0,        {(**void**\*)opt\_report}, "generate a report" },
22. { "max\_alloc"  , HAS\_ARG,  {(**void**\*) opt\_max\_alloc},     "set maximum size of a single allocated block", "bytes" },
23. { "cpuflags"   , HAS\_ARG | OPT\_EXPERT, {(**void**\*) opt\_cpuflags}, "force specific cpu flags", "flags" },

options数组的定义位于ffmpeg\_opt.c中：

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/39760711) [copy](http://blog.csdn.net/leixiaohua1020/article/details/39760711)

[在CODE上查看代码片](https://code.csdn.net/snippets/479130)

1. **const** OptionDef options[] = {
2. /\* main options \*/
3. #include "cmdutils\_common\_opts.h"//包含了cmdutils\_common\_opts.h中的选项
4. { "f", HAS\_ARG | OPT\_STRING | OPT\_OFFSET,           { (**void**\*)OFFSET(format) },
5. "force format", "fmt" },
6. { "i", HAS\_ARG | OPT\_PERFILE,          { (**void**\*) opt\_input\_file },
7. "input file name", "filename" },
8. { "y", OPT\_BOOL,          { &file\_overwrite },
9. "overwrite output files" },
10. { "n", OPT\_BOOL,          { &no\_file\_overwrite },
11. "do not overwrite output files" },
12. { "c", HAS\_ARG | OPT\_STRING | OPT\_SPEC,{ (**void**\*) OFFSET(codec\_names) },
13. "codec name", "codec" },
14. { "codec",          HAS\_ARG | OPT\_STRING | OPT\_SPEC,{(**void**\*) OFFSET(codec\_names) },
15. "codec name", "codec" },
16. { "pre",            HAS\_ARG | OPT\_STRING | OPT\_SPEC,{ (**void**\*) OFFSET(presets) },
17. "preset name", "preset" },
18. { "map",            HAS\_ARG | OPT\_EXPERT | OPT\_PERFILE,          { (**void**\*) opt\_map },
19. "set input stream mapping",
20. "[-]input\_file\_id[:stream\_specifier][,sync\_file\_id[:stream\_specifier]]" },
21. { "map\_channel",    HAS\_ARG | OPT\_EXPERT | OPT\_PERFILE,          {(**void**\*)opt\_map\_channel },
22. "map an audio channel from one stream to another", "file.stream.channel[:syncfile.syncstream]" },
23. { "map\_metadata",   HAS\_ARG | OPT\_STRING | OPT\_SPEC,{ (**void**\*)OFFSET(metadata\_map) },
24. "set metadata information of outfile from infile",
25. "outfile[,metadata]:infile[,metadata]" },
26. { "map\_chapters",   HAS\_ARG | OPT\_INT | OPT\_EXPERT | OPT\_OFFSET, { (**void**\*) OFFSET(chapters\_input\_file) },
27. "set chapters mapping", "input\_file\_index" },
28. { "t", HAS\_ARG | OPT\_TIME | OPT\_OFFSET,{(**void**\*) OFFSET(recording\_time) },
29. "record or transcode \"duration\" seconds of audio/video",
30. "duration" },
31. { "fs",HAS\_ARG | OPT\_INT64 | OPT\_OFFSET,            { (**void**\*) OFFSET(limit\_filesize) },
32. "set the limit file size in bytes", "limit\_size" },
33. { "ss",HAS\_ARG | OPT\_TIME | OPT\_OFFSET,{ (**void**\*) OFFSET(start\_time) },
34. "set the start time offset", "time\_off" },
35. …//选项太多，不一一列出
36. };

在这里，例举一个选项的OptionDef结构体：输入

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/39760711) [copy](http://blog.csdn.net/leixiaohua1020/article/details/39760711)

[在CODE上查看代码片](https://code.csdn.net/snippets/479130)

1. { "i",HAS\_ARG | OPT\_PERFILE, { (**void**\*) opt\_input\_file }, "input file name", "filename" }

在这个结构体中，可以看出选项的名称为“i”，选项包含选项值（HAS\_ARG），选项的处理函数是opt\_input\_file()，选项的说明是“input file name”。下面可以详细看一下选项的处理函数opt\_input\_file()。该函数的定义位于ffmpeg\_opt.c文件中。可以看出，调用了avformat\_alloc\_context()初始化了AVFormatContext结构体，调用了avformat\_open\_input()函数打开了“-i”选项指定的文件。此外，调用了avformat\_find\_stream\_info()等完成了一些初始化操作。此外，调用了av\_dump\_format()打印输出输入文件信息。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/39760711) [copy](http://blog.csdn.net/leixiaohua1020/article/details/39760711)

[在CODE上查看代码片](https://code.csdn.net/snippets/479130)

1. **static** **int** opt\_input\_file(**void** \*optctx, **const** **char** \*opt, **const** **char** \*filename)
2. {
3. //略…
4. /\* open the input file with generic avformat function \*/
5. err = avformat\_open\_input(&ic, filename, file\_iformat, &format\_opts);
6. **if** (err < 0) {
7. print\_error(filename, err);
8. exit(1);
9. }
11. //略…
12. /\* Set AVCodecContext options for avformat\_find\_stream\_info \*/
13. opts = setup\_find\_stream\_info\_opts(ic, codec\_opts);
14. orig\_nb\_streams = ic->nb\_streams;
16. /\* If not enough info to get the stream parameters, we decode the
17. first frames to get it. (used in mpeg case for example) \*/
18. ret = avformat\_find\_stream\_info(ic, opts);
19. **if** (ret < 0) {
20. av\_log(NULL, AV\_LOG\_FATAL, "%s: could not find codec parameters\n", filename);
21. avformat\_close\_input(&ic);
22. exit(1);
23. }
25. //略…
26. /\* dump the file content \*/
27. av\_dump\_format(ic, nb\_input\_files, filename, 0);
29. //略…
30. **return** 0;
31. }

再例举一个输出文件处理函数opt\_output\_file()。这里需要注意，输出文件的处理并不包含在OptionDef类型的数组options中。因为FFmpeg中指定输出文件时并不包含选项名称，这是一个比较特殊的地方。一般的选项格式是“-名称 值”，例如指定输入文件的时候，选项格式是“-i xxx.flv”。而指定输出文件的时候，直接指定“值”即可，这是新手可能容易搞混的地方。  
例如，最简单的转码命令如下（输出文件前面不包含选项）：

**[plain]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/39760711) [copy](http://blog.csdn.net/leixiaohua1020/article/details/39760711)

[在CODE上查看代码片](https://code.csdn.net/snippets/479130)

1. ffmpeg -i xxx.mpg xxx.mkv

而不是

**[plain]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/39760711) [copy](http://blog.csdn.net/leixiaohua1020/article/details/39760711)

[在CODE上查看代码片](https://code.csdn.net/snippets/479130)

1. ffmpeg -i xxx.mpeg -o xxx.mkv

下面简单看一下opt\_output\_file()函数的定义。该函数的定义同样位于ffmpeg\_opt.c文件中。这个函数的定义特别长，完成了输出视频的初始化工作。在这里就不列出代码了。该函数首先调用avformat\_alloc\_output\_context2()初始化AVFormatContext结构体。而后根据媒体类型的不同，分别调用new\_video\_stream()，new\_audio\_stream()，new\_subtitle\_stream()等创建不同的AVStream。实际上上述的几个创建AVStream的函数调用了new\_output\_stream()。而new\_output\_stream()又调用了FFmpeg类库的API函数avformat\_new\_stream()。

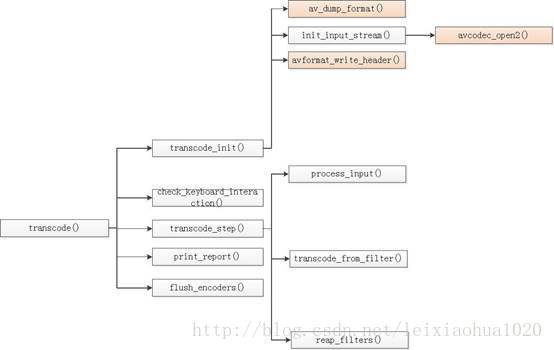
**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/39760711) [copy](http://blog.csdn.net/leixiaohua1020/article/details/39760711)

[在CODE上查看代码片](https://code.csdn.net/snippets/479130)

1. **void** opt\_output\_file(**void** \*optctx, **const** **char** \*filename)
2. {
3. //略…
4. err = avformat\_alloc\_output\_context2(&oc, NULL, o->format, filename);
6. **if** (!oc) {
7. print\_error(filename, err);
8. exit(1);
9. }
10. //略…
11. new\_video\_stream();
12. …
13. new\_audio\_stream();
14. …
15. new\_subtitle\_stream ();
16. //略…
18. }

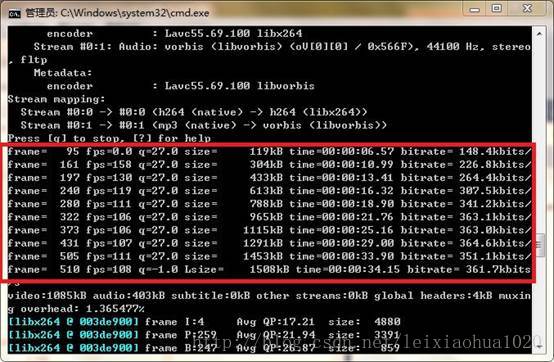
## transcode()

transcode()的功能是转码。其函数调用结构如下图所示。



调用了如下函数  
transcode\_init()：转码的初始化工作。  
check\_keyboard\_interaction()：检测键盘操作。例如转码的过程中按下“Q”键之后，会退出转码。  
transcode\_step()：进行转码。  
print\_report()：打印转码信息，输出到屏幕上。  
flush\_encoder()：输出编码器中剩余的帧。  
其中check\_keyboard\_interaction()，transcode\_step()，print\_report()三个函数位于一个循环之中会不断地执行。

下图红框所示即为print\_report()打印输出到屏幕上的信息。



下面简单介绍两个重要的函数transcode\_init()和transcode\_step()。

### transcode\_init()

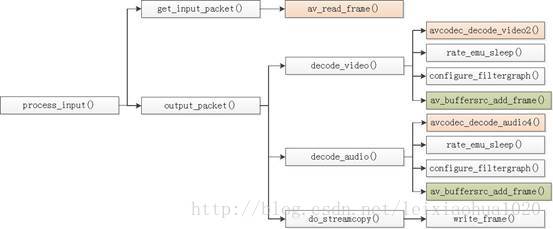
transcode\_init()调用了以下几个重要的函数：  
av\_dump\_format()：在屏幕上打印输出格式信息。注意是输出格式的信息，输入格式的信息的打印是在parse\_options()函数执行过程中调用opt\_input\_file()的时候打印到屏幕上的。  
init\_input\_stream()：其中调用了avcodec\_open2()打开编码器。  
avformat\_write\_header()：写输出文件的文件头。  
 

### transcode\_step()

transcode\_step()调用了如下函数：  
process\_input()：完成解码工作。  
transcode\_from\_filter()：未分析。  
reap\_filters()：完成编码工作。  
 

#### process\_input()

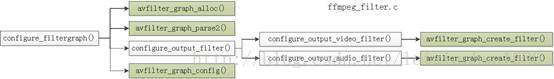
process\_input()主要完成了解码的工作。其函数调用结构如下图所示。



process\_input()调用了如下函数：  
get\_input\_packet()：获取一帧压缩编码数据，即一个AVPacket。其中调用了av\_read\_frame()。  
output\_packet()：解码压缩编码的数据并将之送至AVFilterContext。  
   
output\_packet()调用了如下函数：  
decode\_video()：解码一帧视频（一个AVPacket）。  
decode\_audio()：解码音频（并不一定是一帧，是一个AVPacket）。  
do\_streamcopy()：如果不需要重新编码的话，则调用此函数，一般用于封装格式之间的转换。速度比转码快很多。  
   
decode\_video()调用了如下函数：  
avcodec\_decode\_video2()：解码一帧视频。  
rate\_emu\_sleep()：要求按照帧率处理数据的时候调用，可以避免FFmpeg处理速度过快。常用于网络实时流的处理（RTP/RTMP流的推送）。  
configure\_filtergraph()：设置AVFilterGraph。  
av\_buffersrc\_add\_frame()：将解码后的数据（一个AVFrame）送至AVFilterContext。  
   
decode\_audio()调用的函数和decode\_video()基本一样。唯一的不同在于其解码音频的函数是avcodec\_decode\_audio4()  
 

configure\_filtergraph()

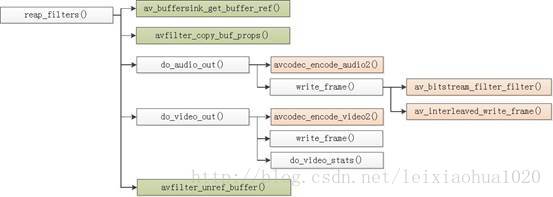
未分析。



#### reap\_filters()

reap\_filters()主要完成了编码的工作。其函数调用结构如下图所示。

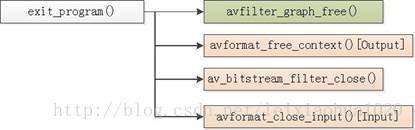


reap\_filters()调用了如下函数  
av\_buffersink\_get\_buffer\_ref()：从AVFilterContext中取出一帧解码后的数据（结构为AVFilterBufferRef，可以转换为AVFrame）。  
avfilter\_copy\_buf\_props()：AVFilterBufferRef转换为AVFrame。  
do\_audio\_out()：编码音频。  
do\_video\_out()：编码视频。  
avfilter\_unref\_buffer()：释放资源。  
   
do\_video\_out()调用了如下函数  
avcodec\_encode\_video2()：编码一帧视频。  
write\_frame()：写入编码后的视频压缩数据。  
   
write\_frame()调用了如下函数：  
av\_bitstream\_filter\_filter()：使用AVBitStreamFilter的时候，会调用此函数进行处理。  
av\_interleaved\_write\_frame()：写入压缩编码数据。  
   
do\_audio\_out()调用的函数与do\_video\_out()基本上一样。唯一的不同在于视频编码函数avcodec\_encode\_video2()变成了音频编码函数avcodec\_encode\_audio2()。

### 

## exit\_program()

exit\_program()主要完成了清理工作。调用关系如下图所示。



   
调用了如下函数：  
avfilter\_graph\_free()：释放AVFilterGraph。  
avformat\_free\_context()：释放输出文件的AVFormatContext。  
av\_bitstream\_filter\_close()：关闭AVBitStreamFilter。  
avformat\_close\_input()：关闭输入文件。  
   
 

## 附录

FFmpeg转码时在屏幕上的输出。

（转码命令为ffmpeg -i cuc\_ieschool.flv cuc\_ieschool.mkv）  
   
F:\movie>ffmpeg -i cuc\_ieschool.flv cuc\_ieschool.mkv  
//版本信息（main()->show\_banner()）

|  |
| --- |
| ffmpeg version N-65018-gad91bf8 Copyright (c) 2000-2014 the FFmpeg developers   built on Jul 26 2014 22:01:46 with gcc 4.8.3 (GCC)   configuration: --enable-gpl --enable-version3 --disable-w32threads --enable-av isynth --enable-bzlib --enable-fontconfig --enable-frei0r --enable-gnutls --enab le-iconv --enable-libass --enable-libbluray --enable-libbs2b --enable-libcaca -- enable-libfreetype --enable-libgme --enable-libgsm --enable-libilbc --enable-lib modplug --enable-libmp3lame --enable-libopencore-amrnb --enable-libopencore-amrw b --enable-libopenjpeg --enable-libopus --enable-librtmp --enable-libschroedinge r --enable-libsoxr --enable-libspeex --enable-libtheora --enable-libtwolame --en able-libvidstab --enable-libvo-aacenc --enable-libvo-amrwbenc --enable-libvorbis  --enable-libvpx --enable-libwavpack --enable-libwebp --enable-libx264 --enable- libx265 --enable-libxavs --enable-libxvid --enable-decklink --enable-zlib   libavutil      52. 92.101 / 52. 92.101   libavcodec     55. 69.100 / 55. 69.100   libavformat    55. 49.100 / 55. 49.100   libavdevice    55. 13.102 / 55. 13.102   libavfilter     4. 11.102 /  4. 11.102   libswscale      2.  6.100 /  2.  6.100   libswresample   0. 19.100 /  0. 19.100   libpostproc    52.  3.100 / 52.  3.100 |

//输入信息（main()->parse\_options()->opt\_input\_file()->av\_dump\_format()）

|  |
| --- |
| Input #0, flv, from 'cuc\_ieschool.flv':   Metadata:     metadatacreator : iku     hasKeyframes    : true     hasVideo        : true     hasAudio        : true     hasMetadata     : true     canSeekToEnd    : false     datasize        : 932906     videosize       : 787866     audiosize       : 140052     lasttimestamp   : 34     lastkeyframetimestamp: 30     lastkeyframelocation: 886498     encoder         : Lavf55.19.104   Duration: 00:00:34.16, start: 0.000000, bitrate: 318 kb/s     Stream #0:0: Video: h264 (Main), yuv420p, 512x288 [SAR 1:1 DAR 16:9], 183 kb /s, 15.17 fps, 15 tbr, 1k tbn, 30 tbc     Stream #0:1: Audio: mp3, 44100 Hz, stereo, s16p, 128 kb/s [libx264 @ 003de900] using SAR=1/1 [libx264 @ 003de900] using cpu capabilities: MMX2 SSE2Fast SSSE3 SSE4.2 AVX [libx264 @ 003de900] profile High, level 2.1 [libx264 @ 003de900] 264 - core 142 r2431 ac76440 - H.264/MPEG-4 AVC codec - Cop yleft 2003-2014 - http://www.videolan.org/x264.html - options: cabac=1 ref=3 deb lock=1:0:0 analyse=0x3:0x113 me=hex subme=7 psy=1 psy\_rd=1.00:0.00 mixed\_ref=1 m e\_range=16 chroma\_me=1 trellis=1 8x8dct=1 cqm=0 deadzone=21,11 fast\_pskip=1 chro ma\_qp\_offset=-2 threads=6 lookahead\_threads=1 sliced\_threads=0 nr=0 decimate=1 i nterlaced=0 bluray\_compat=0 constrained\_intra=0 bframes=3 b\_pyramid=2 b\_adapt=1 b\_bias=0 direct=1 weightb=1 open\_gop=0 weightp=2 keyint=250 keyint\_min=15 scenec ut=40 intra\_refresh=0 rc\_lookahead=40 rc=crf mbtree=1 crf=23.0 qcomp=0.60 qpmin= 0 qpmax=69 qpstep=4 ip\_ratio=1.40 aq=1:1.00 |

//输出信息（main()->transcode()->transcode\_init()->av\_dump\_format()）

|  |
| --- |
| Output #0, matroska, to 'cuc\_ieschool.mkv':   Metadata:     metadatacreator : iku     hasKeyframes    : true     hasVideo        : true     hasAudio        : true     hasMetadata     : true     canSeekToEnd    : false     datasize        : 932906     videosize       : 787866     audiosize       : 140052     lasttimestamp   : 34     lastkeyframetimestamp: 30     lastkeyframelocation: 886498     encoder         : Lavf55.49.100     Stream #0:0: Video: h264 (libx264) (H264 / 0x34363248), yuv420p, 512x288 [SA R 1:1 DAR 16:9], q=-1--1, 15 fps, 1k tbn, 15 tbc     Metadata:       encoder         : Lavc55.69.100 libx264     Stream #0:1: Audio: vorbis (libvorbis) (oV[0][0] / 0x566F), 44100 Hz, stereo , fltp     Metadata:       encoder         : Lavc55.69.100 libvorbis |

//输出Stream Maping 信息（main()->transcode()->transcode\_init()）

|  |
| --- |
| Stream mapping:   Stream #0:0 -> #0:0 (h264 (native) -> h264 (libx264))   Stream #0:1 -> #0:1 (mp3 (native) -> vorbis (libvorbis)) |

//一行字（main()->transcode()）

|  |
| --- |
| Press [q] to stop, [?] for help |

//输出信息（main()->transcode()->print\_report()）

|  |
| --- |
| frame=   95 fps=0.0 q=27.0 size=     119kB time=00:00:06.57 bitrate= 148.4kbits/ frame=  161 fps=158 q=27.0 size=     304kB time=00:00:10.99 bitrate= 226.8kbits/ frame=  197 fps=130 q=27.0 size=     433kB time=00:00:13.41 bitrate= 264.4kbits/ frame=  240 fps=119 q=27.0 size=     613kB time=00:00:16.32 bitrate= 307.5kbits/ frame=  280 fps=111 q=27.0 size=     788kB time=00:00:18.90 bitrate= 341.2kbits/ frame=  322 fps=106 q=27.0 size=     965kB time=00:00:21.76 bitrate= 363.1kbits/ frame=  373 fps=106 q=27.0 size=    1115kB time=00:00:25.16 bitrate= 363.0kbits/ frame=  431 fps=107 q=27.0 size=    1291kB time=00:00:29.00 bitrate= 364.6kbits/ frame=  505 fps=111 q=27.0 size=    1453kB time=00:00:33.90 bitrate= 351.1kbits/ frame=  510 fps=108 q=-1.0 Lsize=    1508kB time=00:00:34.15 bitrate= 361.7kbits/s |

//最后一次输出

|  |
| --- |
| video:1085kB audio:403kB subtitle:0kB other streams:0kB global headers:4kB muxin g overhead: 1.365477% |

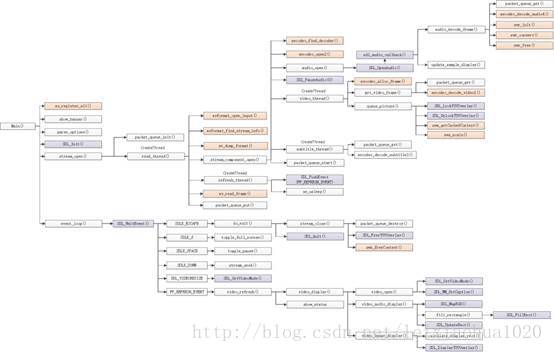
//avcodec\_close()的时候输出（libx264专有的输出信息）

|  |
| --- |
| [libx264 @ 003de900] frame I:4     Avg QP:17.21  size:  4880 [libx264 @ 003de900] frame P:259   Avg QP:21.94  size:  3391 [libx264 @ 003de900] frame B:247   Avg QP:26.87  size:   859 [libx264 @ 003de900] consecutive B-frames: 18.6% 46.3% 12.4% 22.7% [libx264 @ 003de900] mb I  I16..4: 23.0% 57.5% 19.4% [libx264 @ 003de900] mb P  I16..4:  4.2%  7.5%  4.2%  P16..4: 30.1% 12.2%  5.7%  0.0%  0.0%    skip:36.2% [libx264 @ 003de900] mb B  I16..4:  0.3%  0.5%  0.4%  B16..8: 28.8%  4.4%  1.0%  direct: 1.4%  skip:63.2%  L0:40.4% L1:49.9% BI: 9.7% [libx264 @ 003de900] 8x8 transform intra:47.7% inter:41.7% [libx264 @ 003de900] coded y,uvDC,uvAC intra: 47.4% 27.8% 5.0% inter: 13.2% 4.4%  0.3% [libx264 @ 003de900] i16 v,h,dc,p: 22% 36%  9% 33% [libx264 @ 003de900] i8 v,h,dc,ddl,ddr,vr,hd,vl,hu: 23% 26% 28%  3%  3%  3%  4%  3%  4% [libx264 @ 003de900] i4 v,h,dc,ddl,ddr,vr,hd,vl,hu: 22% 25% 20%  5%  6%  5%  6%  5%  6% [libx264 @ 003de900] i8c dc,h,v,p: 71% 18% 10%  1% [libx264 @ 003de900] Weighted P-Frames: Y:0.8% UV:0.8% [libx264 @ 003de900] ref P L0: 69.3% 12.3% 14.1%  4.3%  0.0% [libx264 @ 003de900] ref B L0: 83.9% 15.3%  0.7% [libx264 @ 003de900] ref B L1: 96.0%  4.0% [libx264 @ 003de900] kb/s:261.17 |

# [ffplay.c函数结构简单分析（画图）](http://blog.csdn.net/leixiaohua1020/article/details/39762143)

## 总体结构图

FFplay的总体函数调用结构图如下图所示。

[](http://img.my.csdn.net/uploads/201410/04/1412352241_8878.jpg)

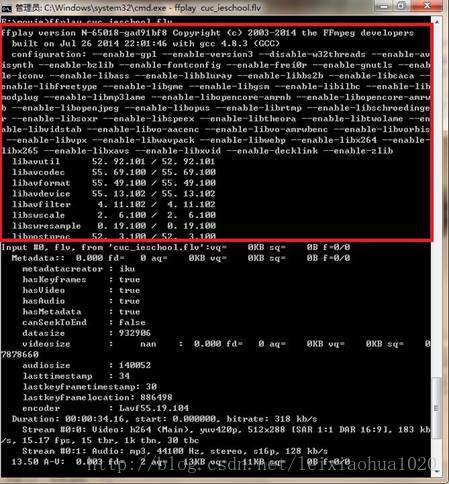
上图所示本是一张高清大图。但是页面显示不下。因此上传了一份：  
<http://my.csdn.net/leixiaohua1020/album/detail/1788077>

上面地址的那张图保存下来的话就是一张清晰的图片了。

下文对主要函数分别解析。

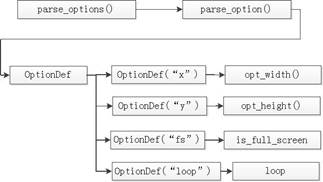
## main()

main()是FFplay的主函数。  
调用了如下函数  
av\_register\_all()：注册所有编码器和解码器。  
show\_banner()：打印输出FFmpeg版本信息（编译时间，编译选项，类库信息等）。  
parse\_options()：解析输入的命令。  
SDL\_Init()：SDL初始化。  
stream\_open ()：打开输入媒体。  
event\_loop()：处理各种消息，不停地循环下去。  
  
  
下图红框中的内容即为show\_banner()的输出结果。



## parse\_options()

parse\_options()解析全部输入选项。即将输入命令“ffplay -f h264 test.264”中的“-f”这样的命令解析出来。其函数调用结构如下图所示。需要注意的是，FFplay（ffplay.c）的parse\_options()和FFmpeg（ffmpeg.c）中的parse\_options()实际上是一样的。因此本部分的内容和《[ffmpeg.c函数结构简单分析（画图）](http://blog.csdn.net/leixiaohua1020/article/details/39760711)》中的parse\_options()有很多重复的地方。



parse\_options()调用了如下函数：  
parse\_option()：解析一个输入选项。具体的解析步骤不再赘述。parse\_options()会循环调用parse\_option()直到所有选项解析完毕。FFmpeg的每一个选项信息存储在一个OptionDef结构体中。定义如下：

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/39762143) [copy](http://blog.csdn.net/leixiaohua1020/article/details/39762143)

1. **typedef** **struct** OptionDef {
2. **const** **char** \*name;
3. **int** flags;
4. #define HAS\_ARG    0x0001
5. #define OPT\_BOOL   0x0002
6. #define OPT\_EXPERT 0x0004
7. #define OPT\_STRING 0x0008
8. #define OPT\_VIDEO  0x0010
9. #define OPT\_AUDIO  0x0020
10. #define OPT\_INT    0x0080
11. #define OPT\_FLOAT  0x0100
12. #define OPT\_SUBTITLE 0x0200
13. #define OPT\_INT64  0x0400
14. #define OPT\_EXIT   0x0800
15. #define OPT\_DATA   0x1000
16. #define OPT\_PERFILE  0x2000     /\* the option is per-file (currently ffmpeg-only).
17. implied by OPT\_OFFSET or OPT\_SPEC \*/
18. #define OPT\_OFFSET 0x4000       /\* option is specified as an offset in a passed optctx \*/
19. #define OPT\_SPEC   0x8000       /\* option is to be stored in an array of SpecifierOpt.
20. Implies OPT\_OFFSET. Next element after the offset is
21. an **int** containing element count in the array. \*/
22. #define OPT\_TIME  0x10000
23. #define OPT\_DOUBLE 0x20000
24. **union** {
25. **void** \*dst\_ptr;
26. **int** (\*func\_arg)(**void** \*, **const** **char** \*, **const** **char** \*);
27. **size\_t** off;
28. } u;
29. **const** **char** \*help;
30. **const** **char** \*argname;
31. } OptionDef;

其中的重要字段：  
name：用于存储选项的名称。例如“i”，“f”，“codec”等等。  
flags：存储选项值的类型。例如：HAS\_ARG（包含选项值），OPT\_STRING（选项值为字符串类型），OPT\_TIME（选项值为时间类型。  
u：存储该选项的处理函数。  
help：选项的说明信息。  
FFmpeg使用一个名称为options，类型为OptionDef的数组存储所有的选项。有一部分通用选项存储在cmdutils\_common\_opts.h中。这些选项对于FFmpeg，FFplay以及FFprobe都试用。  
cmdutils\_common\_opts.h内容如下：

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/39762143) [copy](http://blog.csdn.net/leixiaohua1020/article/details/39762143)

1. { "L"          , OPT\_EXIT, {(**void**\*)show\_license},      "show license" },
2. { "h"          , OPT\_EXIT, {(**void**\*) show\_help},         "show help", "topic" },
3. { "?"          , OPT\_EXIT, {(**void**\*)show\_help},         "show help", "topic" },
4. { "help"       , OPT\_EXIT, {(**void**\*)show\_help},         "show help", "topic" },
5. { "-help"      , OPT\_EXIT, {(**void**\*)show\_help},         "show help", "topic" },
6. { "version"    , OPT\_EXIT, {(**void**\*)show\_version},      "show version" },
7. { "formats"    , OPT\_EXIT, {(**void**\*)show\_formats  },    "show available formats" },
8. { "codecs"     , OPT\_EXIT, {(**void**\*)show\_codecs   },    "show available codecs" },
9. { "decoders"   , OPT\_EXIT, {(**void**\*)show\_decoders },    "show available decoders" },
10. { "encoders"   , OPT\_EXIT, {(**void**\*)show\_encoders },    "show available encoders" },
11. { "bsfs"       , OPT\_EXIT, {(**void**\*)show\_bsfs     },    "show available bit stream filters" },
12. { "protocols"  , OPT\_EXIT, {(**void**\*)show\_protocols},    "show available protocols" },
13. { "filters"    , OPT\_EXIT, {(**void**\*)show\_filters  },    "show available filters" },
14. { "pix\_fmts"   , OPT\_EXIT, {(**void**\*)show\_pix\_fmts },    "show available pixel formats" },
15. { "layouts"    , OPT\_EXIT, {(**void**\*)show\_layouts  },    "show standard channel layouts" },
16. { "sample\_fmts", OPT\_EXIT, {(**void**\*)show\_sample\_fmts }, "show available audio sample formats" },
17. { "loglevel"   , HAS\_ARG,  {(**void**\*)opt\_loglevel},      "set libav\* logging level", "loglevel" },
18. { "v",           HAS\_ARG,  {(**void**\*)opt\_loglevel},      "set libav\* logging level", "loglevel" },
19. { "debug"      , HAS\_ARG,  {(**void**\*)opt\_codec\_debug},   "set debug flags", "flags" },
20. { "fdebug"     , HAS\_ARG,  {(**void**\*)opt\_codec\_debug},   "set debug flags", "flags" },
21. { "report"     , 0,        {(**void**\*)opt\_report}, "generate a report" },
22. { "max\_alloc"  , HAS\_ARG,  {(**void**\*) opt\_max\_alloc},     "set maximum size of a single allocated block", "bytes" },
23. { "cpuflags"   , HAS\_ARG | OPT\_EXPERT, {(**void**\*) opt\_cpuflags}, "force specific cpu flags", "flags" },

options数组的定义位于ffplay.c中，如下所示：

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/39762143) [copy](http://blog.csdn.net/leixiaohua1020/article/details/39762143)

1. **static** **const** OptionDef options[] = {
2. #include "cmdutils\_common\_opts.h"//包含进来
3. { "x", HAS\_ARG, { (**void**\*) opt\_width }, "force displayed width", "width" },
4. { "y", HAS\_ARG, { (**void**\*) opt\_height }, "force displayed height", "height" },
5. { "s", HAS\_ARG | OPT\_VIDEO, { (**void**\*) opt\_frame\_size }, "set frame size (WxH or abbreviation)", "size" },
6. { "fs", OPT\_BOOL, { &is\_full\_screen }, "force full screen" },
7. { "an", OPT\_BOOL, { &audio\_disable }, "disable audio" },
8. { "vn", OPT\_BOOL, { &video\_disable }, "disable video" },
9. { "ast", OPT\_INT | HAS\_ARG | OPT\_EXPERT, { &wanted\_stream[AVMEDIA\_TYPE\_AUDIO] }, "select desired audio stream", "stream\_number" },
10. { "vst", OPT\_INT | HAS\_ARG | OPT\_EXPERT, { &wanted\_stream[AVMEDIA\_TYPE\_VIDEO] }, "select desired video stream", "stream\_number" },
11. { "sst", OPT\_INT | HAS\_ARG | OPT\_EXPERT, { &wanted\_stream[AVMEDIA\_TYPE\_SUBTITLE] }, "select desired subtitle stream", "stream\_number" },
12. { "ss", HAS\_ARG, { (**void**\*) opt\_seek }, "seek to a given position in seconds", "pos" },
13. { "t", HAS\_ARG, { (**void**\*) opt\_duration }, "play  \"duration\" seconds of audio/video", "duration" },
14. //选项众多，不再一一列出…
15. };

选项众多，简单举几个例子：  
强行设置设置屏幕的宽度选项（“-x”选项）：

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/39762143) [copy](http://blog.csdn.net/leixiaohua1020/article/details/39762143)

1. { "x", HAS\_ARG, { (**void**\*) opt\_width }, "force displayed width", "width" }

从代码中可以看出，“-x”选项包含选项值（HAS\_ARG），选项处理函数是opt\_width()。选项说明是"force displayed width"。opt\_width()的内容如下：

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/39762143) [copy](http://blog.csdn.net/leixiaohua1020/article/details/39762143)

1. **static** **int** opt\_width(**void** \*optctx, **const** **char** \*opt, **const** **char** \*arg)
2. {
3. screen\_width = parse\_number\_or\_die(opt, arg, OPT\_INT64, 1, INT\_MAX);
4. **return** 0;
5. }

可以看出其作用是解析输入的字符串为整数并赋值给全局变量screen\_width。  
  
  
全屏（“-fs”选项）

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/39762143) [copy](http://blog.csdn.net/leixiaohua1020/article/details/39762143)

1. { "fs", OPT\_BOOL, { &is\_full\_screen }, "force full screen" }

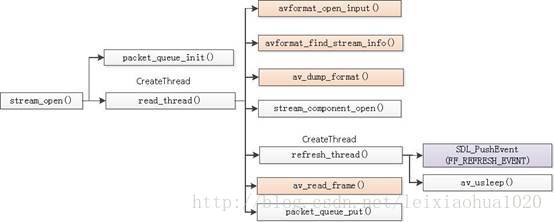
从代码中可以看出，“-fs”选项包含布尔型选项值（OPT\_BOOL），并绑定了全局变量is\_full\_screen。选项说明是"force full screen"。

## SDL\_Init()

SDL\_Init()用于初始化SDL。FFplay中视频的显示和声音的播放都用到了SDL。

## stream\_open()

stream\_open()的作用是打开输入的媒体。这个函数还是比较复杂的，包含了FFplay中各种线程的创建。它的函数调用结构如下图所示。



stream\_open()调用了如下函数：  
packet\_queue\_init()：初始化各个PacketQueue（视频/音频/字幕）  
read\_thread()：读取媒体信息线程。

### read\_thread()

read\_thread()调用了如下函数：

avformat\_open\_input()：打开媒体。  
avformat\_find\_stream\_info()：获得媒体信息。  
av\_dump\_format()：输出媒体信息到控制台。  
stream\_component\_open()：分别打开视频/音频/字幕解码线程。  
refresh\_thread()：视频刷新线程。  
av\_read\_frame()：获取一帧压缩编码数据（即一个AVPacket）。  
packet\_queue\_put()：根据压缩编码数据类型的不同（视频/音频/字幕），放到不同的PacketQueue中。

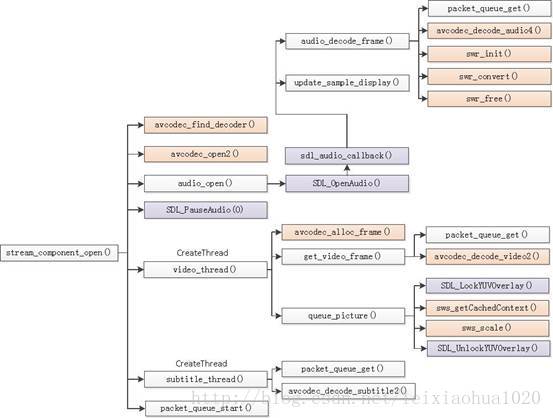
#### refresh\_thread()

refresh\_thread()调用了如下函数：

SDL\_PushEvent(FF\_REFRESH\_EVENT)：发送FF\_REFRESH\_EVENT的SDL\_Event  
av\_usleep()：每两次发送之间，间隔一段时间。

#### stream\_component\_open()

stream\_component\_open()用于打开视频/音频/字幕解码的线程。其函数调用关系如下图所示。



stream\_component\_open()调用了如下函数：  
avcodec\_find\_decoder()：获得解码器。  
avcodec\_open2()：打开解码器。  
audio\_open()：打开音频解码。  
SDL\_PauseAudio(0)：SDL中播放音频的函数。  
video\_thread()：创建视频解码线程。  
subtitle\_thread()：创建字幕解码线程。  
packet\_queue\_start()：初始化PacketQueue。  
  
  
audio\_open()调用了如下函数  
SDL\_OpenAudio()：SDL中打开音频设备的函数。注意它是根据SDL\_AudioSpec参数打开音频设备。SDL\_AudioSpec中的callback字段指定了音频播放的回调函数sdl\_audio\_callback()。当音频设备需要更多数据的时候，会调用该回调函数。因此该函数是会被反复调用的。  
  
  
下面来看一下SDL\_AudioSpec中指定的回调函数sdl\_audio\_callback()。  
sdl\_audio\_callback()调用了如下函数  
audio\_decode\_frame()：解码音频数据。  
update\_sample\_display()：当不显示视频图像，而是显示音频波形的时候，调用此函数。  
  
  
audio\_decode\_frame()调用了如下函数  
packet\_queue\_get()：获取音频压缩编码数据（一个AVPacket）。  
avcodec\_decode\_audio4()：解码音频压缩编码数据（得到一个AVFrame）。  
swr\_init()：初始化libswresample中的SwrContext。libswresample用于音频采样采样数据（PCM）的转换。  
swr\_convert()：转换音频采样率到适合系统播放的格式。  
swr\_free()：释放SwrContext。  
  
  
video\_thread()调用了如下函数  
avcodec\_alloc\_frame()：初始化一个AVFrame。  
get\_video\_frame()：获取一个存储解码后数据的AVFrame。  
queue\_picture()：  
  
  
get\_video\_frame()调用了如下函数  
packet\_queue\_get()：获取视频压缩编码数据（一个AVPacket）。  
avcodec\_decode\_video2()：解码视频压缩编码数据（得到一个AVFrame）。  
  
  
queue\_picture()调用了如下函数  
SDL\_LockYUVOverlay()：锁定一个SDL\_Overlay。  
sws\_getCachedContext()：初始化libswscale中的SwsContext。Libswscale用于图像的Raw格式数据（YUV，RGB）之间的转换。注意sws\_getCachedContext()和sws\_getContext()功能是一致的。  
sws\_scale()：转换图像数据到适合系统播放的格式。  
SDL\_UnlockYUVOverlay()：解锁一个SDL\_Overlay。  
  
  
  
  
subtitle\_thread()调用了如下函数  
packet\_queue\_get()：获取字幕压缩编码数据（一个AVPacket）。

avcodec\_decode\_subtitle2()：解码字幕压缩编码数据。

### 

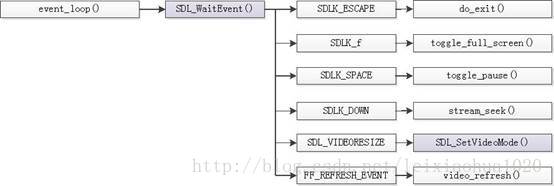
## event\_loop()

FFplay再打开媒体之后，便会进入event\_loop()函数，永远不停的循环下去。该函数用于接收并处理各种各样的消息。有点像Windows的消息循环机制。  
PS：该循环确实是无止尽的，其形式为如下

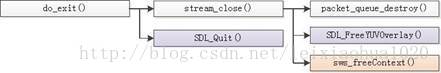
**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/39762143) [copy](http://blog.csdn.net/leixiaohua1020/article/details/39762143)

1. SDL\_Event event;
2. **for** (;;) {
3. SDL\_WaitEvent(&event);
4. **switch** (event.type) {
5. **case** SDLK\_ESCAPE:
6. **case** SDLK\_q:
7. do\_exit(cur\_stream);
8. **break**;
9. **case** SDLK\_f:
10. …
11. …
12. }
13. }

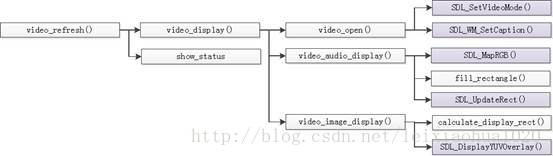
event\_loop()函数调用关系如下所示。



根据event\_loop()中SDL\_WaitEvent()接收到的SDL\_Event类型的不同，会调用不同的函数进行处理（从编程的角度来说就是一个switch()语法）。图中仅仅列举了几个例子：  
SDLK\_ESCAPE（按下“ESC”键）：do\_exit()。退出程序。  
SDLK\_f（按下“f”键）：toggle\_full\_screen()。切换全屏显示。  
SDLK\_SPACE（按下“空格”键）：toggle\_pause()。切换“暂停”。  
SDLK\_DOWN（按下鼠标键）：stream\_seek()。跳转到指定的时间点播放。  
SDL\_VIDEORESIZE（窗口大小发生变化）：SDL\_SetVideoMode()。重新设置宽高。  
FF\_REFRESH\_EVENT（视频刷新事件（自定义事件））：video\_refresh()。刷新视频。  
  
  
下面分析一下do\_exit()函数。该函数用于退出程序。函数的调用关系如下图所示。



do\_exit()函数调用了以下函数  
stream\_close()：关闭打开的媒体。  
SDL\_Quit()：关闭SDL。  
  
  
stream\_close()函数调用了以下函数  
packet\_queue\_destroy()：释放PacketQueue。  
SDL\_FreeYUVOverlay()：释放SDL\_Overlay。  
sws\_freeContext()：释放SwsContext。  
  
  
下面重点分析video\_refresh()函数。该函数用于将图像显示到显示器上。函数的调用关系如下图所示。



video\_refresh()函数调用了以下函数  
video\_display()：显示像素数据到屏幕上。  
show\_status：这算不上是一个函数，但是是一个独立的功能模块，因此列了出来。该部分打印输出播放的状态至屏幕上。如下图所示。

http://img.blog.csdn.net/20141003235043264?watermark/2/text/aHR0cDovL2Jsb2cuY3Nkbi5uZXQvbGVpeGlhb2h1YTEwMjA=/font/5a6L5L2T/fontsize/400/fill/I0JBQkFCMA==/dissolve/70/gravity/SouthEast

video\_display()函数调用了以下函数  
video\_open()：初始化的时候调用，打开播放窗口。  
video\_audio\_display()：显示音频波形图（或者频谱图）的时候调用。里面包含了不少画图操作。  
video\_image\_display()：显示视频画面的时候调用。  
  
  
video\_open()函数调用了以下函数  
SDL\_SetVideoMode()：设置SDL\_Surface（即SDL最基础的黑色的框）的大小等信息。  
SDL\_WM\_SetCaption()：设置SDL\_Surface对应窗口的标题文字。  
  
  
video\_audio\_display()函数调用了以下函数  
SDL\_MapRGB()：获得指定（R，G，B）以及SDL\_PixelFormat的颜色数值。例如获得黑色的值，作为背景。（R，G，B）为（0x00，0x00，0x00）。  
fill\_rectangle()：将指定颜色显示到屏幕上。  
SDL\_UpdateRect()：更新屏幕。  
  
  
video\_image\_display()函数调用了以下函数  
calculate\_display\_rect()：计算显示画面的位置。当拉伸了SDL的窗口的时候，可以让其中的视频保持纵横比。  
SDL\_DisplayYUVOverlay()：显示画面至屏幕。

# [FFMPEG中最关键的结构体之间的关系](http://blog.csdn.net/leixiaohua1020/article/details/11693997)

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。

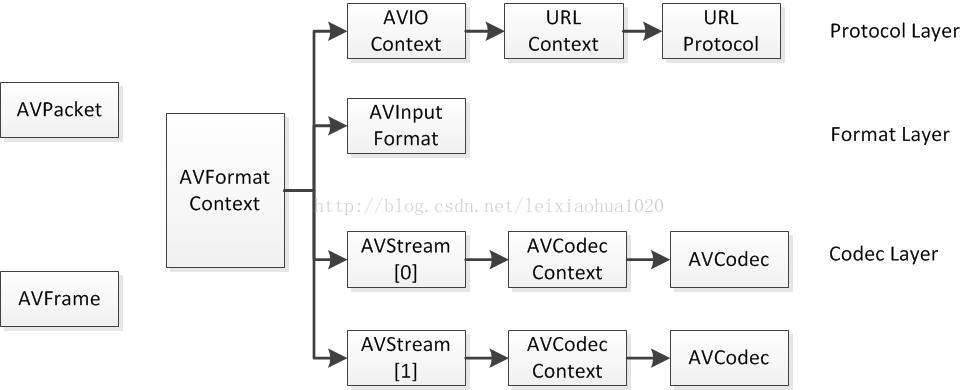
d) 存数据

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

他们之间的对应关系如下所示：



# [FFMPEG结构体分析：AVPacket](http://blog.csdn.net/leixiaohua1020/article/details/14215755)

FFMPEG有几个最重要的结构体，包含了解协议，解封装，解码操作，此前已经进行过分析：

[FFMPEG中最关键的结构体之间的关系](http://blog.csdn.net/leixiaohua1020/article/details/11693997)

在此不再详述，其中AVPacket是存储压缩编码数据相关信息的结构体。本文将会详细分析一下该结构体里重要变量的含义和作用。

首先看一下结构体的定义（位于avcodec.h文件中）：

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/14215755) [copy](http://blog.csdn.net/leixiaohua1020/article/details/14215755)

1. /\* 雷霄骅
2. \* 中国传媒大学/数字电视技术
3. \* leixiaohua1020@126.com
4. \*
5. \*/
6. **typedef** **struct** AVPacket {
7. /\*\*
8. \* Presentation timestamp in AVStream->time\_base units; the time at which
9. \* the decompressed packet will be presented to the user.
10. \* Can be AV\_NOPTS\_VALUE if it is not stored in the file.
11. \* pts MUST be larger or equal to dts as presentation cannot happen before
12. \* decompression, unless one wants to view hex dumps. Some formats misuse
13. \* the terms dts and pts/cts to mean something different. Such timestamps
14. \* must be converted to true pts/dts before they are stored in AVPacket.
15. \*/
16. int64\_t pts;
17. /\*\*
18. \* Decompression timestamp in AVStream->time\_base units; the time at which
19. \* the packet is decompressed.
20. \* Can be AV\_NOPTS\_VALUE if it is not stored in the file.
21. \*/
22. int64\_t dts;
23. uint8\_t \*data;
24. **int**   size;
25. **int**   stream\_index;
26. /\*\*
27. \* A combination of AV\_PKT\_FLAG values
28. \*/
29. **int**   flags;
30. /\*\*
31. \* Additional packet data that can be provided by the container.
32. \* Packet can contain several types of side information.
33. \*/
34. **struct** {
35. uint8\_t \*data;
36. **int**      size;
37. **enum** AVPacketSideDataType type;
38. } \*side\_data;
39. **int** side\_data\_elems;
41. /\*\*
42. \* Duration of this packet in AVStream->time\_base units, 0 if unknown.
43. \* Equals next\_pts - this\_pts in presentation order.
44. \*/
45. **int**   duration;
46. **void**  (\*destruct)(**struct** AVPacket \*);
47. **void**  \*priv;
48. int64\_t pos;                            ///< byte position in stream, -1 if unknown
50. /\*\*
51. \* Time difference in AVStream->time\_base units from the pts of this
52. \* packet to the point at which the output from the decoder has converged
53. \* independent from the availability of previous frames. That is, the
54. \* frames are virtually identical no matter if decoding started from
55. \* the very first frame or from this keyframe.
56. \* Is AV\_NOPTS\_VALUE if unknown.
57. \* This field is not the display duration of the current packet.
58. \* This field has no meaning if the packet does not have AV\_PKT\_FLAG\_KEY
59. \* set.
60. \*
61. \* The purpose of this field is to allow seeking in streams that have no
62. \* keyframes in the conventional sense. It corresponds to the
63. \* recovery point SEI in H.264 and match\_time\_delta in NUT. It is also
64. \* essential for some types of subtitle streams to ensure that all
65. \* subtitles are correctly displayed after seeking.
66. \*/
67. int64\_t convergence\_duration;
68. } AVPacket;

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。

这个结构体虽然比较简单，但是非常的常用。

# [FFMPEG结构体分析：AVStream](http://blog.csdn.net/leixiaohua1020/article/details/14215821)

FFMPEG有几个最重要的结构体，包含了解协议，解封装，解码操作，此前已经进行过分析：

[FFMPEG中最关键的结构体之间的关系](http://blog.csdn.net/leixiaohua1020/article/details/11693997)

在此不再详述，其中AVStream是存储每一个视频/音频流信息的结构体。本文将会分析一下该结构体里重要变量的含义和作用。

首先看一下结构体的定义（位于avformat.h文件中）：

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/14215821) [copy](http://blog.csdn.net/leixiaohua1020/article/details/14215821)

1. /\* 雷霄骅
2. \* 中国传媒大学/数字电视技术
3. \* leixiaohua1020@126.com
4. \*
5. \*/
6. /\*\*
7. \* Stream structure.
8. \* New fields can be added to the end with minor version bumps.
9. \* Removal, reordering and changes to existing fields require a major
10. \* version bump.
11. \* sizeof(AVStream) must not be used outside libav\*.
12. \*/
13. **typedef** **struct** AVStream {
14. **int** index;    /\*\*< stream index in AVFormatContext \*/
15. /\*\*
16. \* Format-specific stream ID.
17. \* decoding: set by libavformat
18. \* encoding: set by the user
19. \*/
20. **int** id;
21. AVCodecContext \*codec; /\*\*< codec context \*/
22. /\*\*
23. \* Real base framerate of the stream.
24. \* This is the lowest framerate with which all timestamps can be
25. \* represented accurately (it is the least common multiple of all
26. \* framerates in the stream). Note, this value is just a guess!
27. \* For example, if the time base is 1/90000 and all frames have either
28. \* approximately 3600 or 1800 timer ticks, then r\_frame\_rate will be 50/1.
29. \*/
30. AVRational r\_frame\_rate;
31. **void** \*priv\_data;
33. /\*\*
34. \* encoding: pts generation when outputting stream
35. \*/
36. **struct** AVFrac pts;
38. /\*\*
39. \* This is the fundamental unit of time (in seconds) in terms
40. \* of which frame timestamps are represented. For fixed-fps content,
41. \* time base should be 1/framerate and timestamp increments should be 1.
42. \* decoding: set by libavformat
43. \* encoding: set by libavformat in av\_write\_header
44. \*/
45. AVRational time\_base;
47. /\*\*
48. \* Decoding: pts of the first frame of the stream in presentation order, in stream time base.
49. \* Only set this if you are absolutely 100% sure that the value you set
50. \* it to really is the pts of the first frame.
51. \* This may be undefined (AV\_NOPTS\_VALUE).
52. \* @note The ASF header does NOT contain a correct start\_time the ASF
53. \* demuxer must NOT set this.
54. \*/
55. int64\_t start\_time;
57. /\*\*
58. \* Decoding: duration of the stream, in stream time base.
59. \* If a source file does not specify a duration, but does specify
60. \* a bitrate, this value will be estimated from bitrate and file size.
61. \*/
62. int64\_t duration;
64. int64\_t nb\_frames;                 ///< number of frames in this stream if known or 0
66. **int** disposition; /\*\*< AV\_DISPOSITION\_\* bit field \*/
68. **enum** AVDiscard discard; ///< Selects which packets can be discarded at will and do not need to be demuxed.
70. /\*\*
71. \* sample aspect ratio (0 if unknown)
72. \* - encoding: Set by user.
73. \* - decoding: Set by libavformat.
74. \*/
75. AVRational sample\_aspect\_ratio;
77. AVDictionary \*metadata;
79. /\*\*
80. \* Average framerate
81. \*/
82. AVRational avg\_frame\_rate;
84. /\*\*
85. \* For streams with AV\_DISPOSITION\_ATTACHED\_PIC disposition, this packet
86. \* will contain the attached picture.
87. \*
88. \* decoding: set by libavformat, must not be modified by the caller.
89. \* encoding: unused
90. \*/
91. AVPacket attached\_pic;
93. /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
94. \* All fields below this line are not part of the public API. They
95. \* may not be used outside of libavformat and can be changed and
96. \* removed at will.
97. \* New public fields should be added right above.
98. \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
99. \*/
101. /\*\*
102. \* Stream information used internally by av\_find\_stream\_info()
103. \*/
104. #define MAX\_STD\_TIMEBASES (60\*12+5)
105. **struct** {
106. int64\_t last\_dts;
107. int64\_t duration\_gcd;
108. **int** duration\_count;
109. **double** duration\_error[2][2][MAX\_STD\_TIMEBASES];
110. int64\_t codec\_info\_duration;
111. **int** nb\_decoded\_frames;
112. **int** found\_decoder;
113. } \*info;
115. **int** pts\_wrap\_bits; /\*\*< number of bits in pts (used for wrapping control) \*/
117. // Timestamp generation support:
118. /\*\*
119. \* Timestamp corresponding to the last dts sync point.
120. \*
121. \* Initialized when AVCodecParserContext.dts\_sync\_point >= 0 and
122. \* a DTS is received from the underlying container. Otherwise set to
123. \* AV\_NOPTS\_VALUE by default.
124. \*/
125. int64\_t reference\_dts;
126. int64\_t first\_dts;
127. int64\_t cur\_dts;
128. int64\_t last\_IP\_pts;
129. **int** last\_IP\_duration;
131. /\*\*
132. \* Number of packets to buffer for codec probing
133. \*/
134. #define MAX\_PROBE\_PACKETS 2500
135. **int** probe\_packets;
137. /\*\*
138. \* Number of frames that have been demuxed during av\_find\_stream\_info()
139. \*/
140. **int** codec\_info\_nb\_frames;
142. /\*\*
143. \* Stream Identifier
144. \* This is the MPEG-TS stream identifier +1
145. \* 0 means unknown
146. \*/
147. **int** stream\_identifier;
149. int64\_t interleaver\_chunk\_size;
150. int64\_t interleaver\_chunk\_duration;
152. /\* av\_read\_frame() support \*/
153. **enum** AVStreamParseType need\_parsing;
154. **struct** AVCodecParserContext \*parser;
156. /\*\*
157. \* last packet in packet\_buffer for this stream when muxing.
158. \*/
159. **struct** AVPacketList \*last\_in\_packet\_buffer;
160. AVProbeData probe\_data;
161. #define MAX\_REORDER\_DELAY 16
162. int64\_t pts\_buffer[MAX\_REORDER\_DELAY+1];
164. AVIndexEntry \*index\_entries; /\*\*< Only used if the format does not
165. support seeking natively. \*/
166. **int** nb\_index\_entries;
167. unsigned **int** index\_entries\_allocated\_size;
169. /\*\*
170. \* flag to indicate that probing is requested
171. \* NOT PART OF PUBLIC API
172. \*/
173. **int** request\_probe;
174. } AVStream;

AVStream重要的变量如下所示：

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。

该结构体其他字段的作用目前还有待于探索。

# [FFMPEG结构体分析：AVCodec](http://blog.csdn.net/leixiaohua1020/article/details/14215833)

FFMPEG有几个最重要的结构体，包含了解协议，解封装，解码操作，此前已经进行过分析：

[FFMPEG中最关键的结构体之间的关系](http://blog.csdn.net/leixiaohua1020/article/details/11693997)

在此不再详述，其中AVCodec是存储编解码器信息的结构体。本文将会详细分析一下该结构体里每个变量的含义和作用。

首先看一下结构体的定义（位于avcodec.h文件中）：

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/14215833) [copy](http://blog.csdn.net/leixiaohua1020/article/details/14215833)

1. /\* 雷霄骅
2. \* 中国传媒大学/数字电视技术
3. \* leixiaohua1020@126.com
4. \*
5. \*/
6. /\*\*
7. \* AVCodec.
8. \*/
9. **typedef** **struct** AVCodec {
10. /\*\*
11. \* Name of the codec implementation.
12. \* The name is globally unique among encoders and among decoders (but an
13. \* encoder and a decoder can share the same name).
14. \* This is the primary way to find a codec from the user perspective.
15. \*/
16. **const** **char** \*name;
17. /\*\*
18. \* Descriptive name for the codec, meant to be more human readable than name.
19. \* You should use the NULL\_IF\_CONFIG\_SMALL() macro to define it.
20. \*/
21. **const** **char** \*long\_name;
22. **enum** AVMediaType type;
23. **enum** CodecID id;
24. /\*\*
25. \* Codec capabilities.
26. \* see CODEC\_CAP\_\*
27. \*/
28. **int** capabilities;
29. **const** AVRational \*supported\_framerates; ///< array of supported framerates, or NULL if any, array is terminated by {0,0}
30. **const** **enum** PixelFormat \*pix\_fmts;       ///< array of supported pixel formats, or NULL if unknown, array is terminated by -1
31. **const** **int** \*supported\_samplerates;       ///< array of supported audio samplerates, or NULL if unknown, array is terminated by 0
32. **const** **enum** AVSampleFormat \*sample\_fmts; ///< array of supported sample formats, or NULL if unknown, array is terminated by -1
33. **const** uint64\_t \*channel\_layouts;         ///< array of support channel layouts, or NULL if unknown. array is terminated by 0
34. uint8\_t max\_lowres;                     ///< maximum value for lowres supported by the decoder
35. **const** AVClass \*priv\_class;              ///< AVClass for the private context
36. **const** AVProfile \*profiles;              ///< array of recognized profiles, or NULL if unknown, array is terminated by {FF\_PROFILE\_UNKNOWN}
38. /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
39. \* No fields below this line are part of the public API. They
40. \* may not be used outside of libavcodec and can be changed and
41. \* removed at will.
42. \* New public fields should be added right above.
43. \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
44. \*/
45. **int** priv\_data\_size;
46. **struct** AVCodec \*next;
47. /\*\*
48. \* @name Frame-level threading support functions
49. \* @{
50. \*/
51. /\*\*
52. \* If defined, called on thread contexts when they are created.
53. \* If the codec allocates writable tables in init(), re-allocate them here.
54. \* priv\_data will be set to a copy of the original.
55. \*/
56. **int** (\*init\_thread\_copy)(AVCodecContext \*);
57. /\*\*
58. \* Copy necessary context variables from a previous thread context to the current one.
59. \* If not defined, the next thread will start automatically; otherwise, the codec
60. \* must call ff\_thread\_finish\_setup().
61. \*
62. \* dst and src will (rarely) point to the same context, in which case memcpy should be skipped.
63. \*/
64. **int** (\*update\_thread\_context)(AVCodecContext \*dst, **const** AVCodecContext \*src);
65. /\*\* @} \*/
67. /\*\*
68. \* Private codec-specific defaults.
69. \*/
70. **const** AVCodecDefault \*defaults;
72. /\*\*
73. \* Initialize codec static data, called from avcodec\_register().
74. \*/
75. **void** (\*init\_static\_data)(**struct** AVCodec \*codec);
77. **int** (\*init)(AVCodecContext \*);
78. **int** (\*encode)(AVCodecContext \*, uint8\_t \*buf, **int** buf\_size, **void** \*data);
79. /\*\*
80. \* Encode data to an AVPacket.
81. \*
82. \* @param      avctx          codec context
83. \* @param      avpkt          output AVPacket (may contain a user-provided buffer)
84. \* @param[in]  frame          AVFrame containing the raw data to be encoded
85. \* @param[out] got\_packet\_ptr encoder sets to 0 or 1 to indicate that a
86. \*                            non-empty packet was returned in avpkt.
87. \* @return 0 on success, negative error code on failure
88. \*/
89. **int** (\*encode2)(AVCodecContext \*avctx, AVPacket \*avpkt, **const** AVFrame \*frame,
90. **int** \*got\_packet\_ptr);
91. **int** (\*decode)(AVCodecContext \*, **void** \*outdata, **int** \*outdata\_size, AVPacket \*avpkt);
92. **int** (\*close)(AVCodecContext \*);
93. /\*\*
94. \* Flush buffers.
95. \* Will be called when seeking
96. \*/
97. **void** (\*flush)(AVCodecContext \*);
98. } AVCodec;

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

详细介绍几个变量：

1.enum AVMediaType type

AVMediaType定义如下：

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/14215833) [copy](http://blog.csdn.net/leixiaohua1020/article/details/14215833)

1. **enum** AVMediaType {
2. AVMEDIA\_TYPE\_UNKNOWN = -1,  ///< Usually treated as AVMEDIA\_TYPE\_DATA
3. AVMEDIA\_TYPE\_VIDEO,
4. AVMEDIA\_TYPE\_AUDIO,
5. AVMEDIA\_TYPE\_DATA,          ///< Opaque data information usually continuous
6. AVMEDIA\_TYPE\_SUBTITLE,
7. AVMEDIA\_TYPE\_ATTACHMENT,    ///< Opaque data information usually sparse
8. AVMEDIA\_TYPE\_NB
9. };

2.enum AVCodecID id

AVCodecID定义如下：

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/14215833) [copy](http://blog.csdn.net/leixiaohua1020/article/details/14215833)

1. **enum** AVCodecID {
2. AV\_CODEC\_ID\_NONE,
4. /\* video codecs \*/
5. AV\_CODEC\_ID\_MPEG1VIDEO,
6. AV\_CODEC\_ID\_MPEG2VIDEO, ///< preferred ID for MPEG-1/2 video decoding
7. AV\_CODEC\_ID\_MPEG2VIDEO\_XVMC,
8. AV\_CODEC\_ID\_H261,
9. AV\_CODEC\_ID\_H263,
10. AV\_CODEC\_ID\_RV10,
11. AV\_CODEC\_ID\_RV20,
12. AV\_CODEC\_ID\_MJPEG,
13. AV\_CODEC\_ID\_MJPEGB,
14. AV\_CODEC\_ID\_LJPEG,
15. AV\_CODEC\_ID\_SP5X,
16. AV\_CODEC\_ID\_JPEGLS,
17. AV\_CODEC\_ID\_MPEG4,
18. AV\_CODEC\_ID\_RAWVIDEO,
19. AV\_CODEC\_ID\_MSMPEG4V1,
20. AV\_CODEC\_ID\_MSMPEG4V2,
21. AV\_CODEC\_ID\_MSMPEG4V3,
22. AV\_CODEC\_ID\_WMV1,
23. AV\_CODEC\_ID\_WMV2,
24. AV\_CODEC\_ID\_H263P,
25. AV\_CODEC\_ID\_H263I,
26. AV\_CODEC\_ID\_FLV1,
27. AV\_CODEC\_ID\_SVQ1,
28. AV\_CODEC\_ID\_SVQ3,
29. AV\_CODEC\_ID\_DVVIDEO,
30. AV\_CODEC\_ID\_HUFFYUV,
31. AV\_CODEC\_ID\_CYUV,
32. AV\_CODEC\_ID\_H264,
33. ...（代码太长，略）
34. }

3.const enum AVPixelFormat \*pix\_fmts

AVPixelFormat定义如下：

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/14215833) [copy](http://blog.csdn.net/leixiaohua1020/article/details/14215833)

1. **enum** AVPixelFormat {
2. AV\_PIX\_FMT\_NONE = -1,
3. AV\_PIX\_FMT\_YUV420P,   ///< planar YUV 4:2:0, 12bpp, (1 Cr & Cb sample per 2x2 Y samples)
4. AV\_PIX\_FMT\_YUYV422,   ///< packed YUV 4:2:2, 16bpp, Y0 Cb Y1 Cr
5. AV\_PIX\_FMT\_RGB24,     ///< packed RGB 8:8:8, 24bpp, RGBRGB...
6. AV\_PIX\_FMT\_BGR24,     ///< packed RGB 8:8:8, 24bpp, BGRBGR...
7. AV\_PIX\_FMT\_YUV422P,   ///< planar YUV 4:2:2, 16bpp, (1 Cr & Cb sample per 2x1 Y samples)
8. AV\_PIX\_FMT\_YUV444P,   ///< planar YUV 4:4:4, 24bpp, (1 Cr & Cb sample per 1x1 Y samples)
9. AV\_PIX\_FMT\_YUV410P,   ///< planar YUV 4:1:0,  9bpp, (1 Cr & Cb sample per 4x4 Y samples)
10. AV\_PIX\_FMT\_YUV411P,   ///< planar YUV 4:1:1, 12bpp, (1 Cr & Cb sample per 4x1 Y samples)
11. AV\_PIX\_FMT\_GRAY8,     ///<        Y        ,  8bpp
12. AV\_PIX\_FMT\_MONOWHITE, ///<        Y        ,  1bpp, 0 is white, 1 is black, in each byte pixels are ordered from the msb to the lsb
13. AV\_PIX\_FMT\_MONOBLACK, ///<        Y        ,  1bpp, 0 is black, 1 is white, in each byte pixels are ordered from the msb to the lsb
14. AV\_PIX\_FMT\_PAL8,      ///< 8 bit with PIX\_FMT\_RGB32 palette
15. AV\_PIX\_FMT\_YUVJ420P,  ///< planar YUV 4:2:0, 12bpp, full scale (JPEG), deprecated in favor of PIX\_FMT\_YUV420P and setting color\_range
16. AV\_PIX\_FMT\_YUVJ422P,  ///< planar YUV 4:2:2, 16bpp, full scale (JPEG), deprecated in favor of PIX\_FMT\_YUV422P and setting color\_range
17. AV\_PIX\_FMT\_YUVJ444P,  ///< planar YUV 4:4:4, 24bpp, full scale (JPEG), deprecated in favor of PIX\_FMT\_YUV444P and setting color\_range
18. AV\_PIX\_FMT\_XVMC\_MPEG2\_MC,///< XVideo Motion Acceleration via common packet passing
19. AV\_PIX\_FMT\_XVMC\_MPEG2\_IDCT,
20. ...（代码太长，略）
21. }

4.const enum AVSampleFormat \*sample\_fmts

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/14215833) [copy](http://blog.csdn.net/leixiaohua1020/article/details/14215833)

1. **enum** AVSampleFormat {
2. AV\_SAMPLE\_FMT\_NONE = -1,
3. AV\_SAMPLE\_FMT\_U8,          ///< unsigned 8 bits
4. AV\_SAMPLE\_FMT\_S16,         ///< signed 16 bits
5. AV\_SAMPLE\_FMT\_S32,         ///< signed 32 bits
6. AV\_SAMPLE\_FMT\_FLT,         ///< float
7. AV\_SAMPLE\_FMT\_DBL,         ///< double
9. AV\_SAMPLE\_FMT\_U8P,         ///< unsigned 8 bits, planar
10. AV\_SAMPLE\_FMT\_S16P,        ///< signed 16 bits, planar
11. AV\_SAMPLE\_FMT\_S32P,        ///< signed 32 bits, planar
12. AV\_SAMPLE\_FMT\_FLTP,        ///< float, planar
13. AV\_SAMPLE\_FMT\_DBLP,        ///< double, planar
15. AV\_SAMPLE\_FMT\_NB           ///< Number of sample formats. DO NOT USE if linking dynamically
16. };

每一个编解码器对应一个该结构体，查看一下ffmpeg的源代码，我们可以看一下H.264解码器的结构体如下所示（h264.c）：

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/14215833) [copy](http://blog.csdn.net/leixiaohua1020/article/details/14215833)

1. AVCodec ff\_h264\_decoder = {
2. .name           = "h264",
3. .type           = AVMEDIA\_TYPE\_VIDEO,
4. .id             = CODEC\_ID\_H264,
5. .priv\_data\_size = **sizeof**(H264Context),
6. .init           = ff\_h264\_decode\_init,
7. .close          = ff\_h264\_decode\_end,
8. .decode         = decode\_frame,
9. .capabilities   = /\*CODEC\_CAP\_DRAW\_HORIZ\_BAND |\*/ CODEC\_CAP\_DR1 | CODEC\_CAP\_DELAY |
10. CODEC\_CAP\_SLICE\_THREADS | CODEC\_CAP\_FRAME\_THREADS,
11. .flush= flush\_dpb,
12. .long\_name = NULL\_IF\_CONFIG\_SMALL("H.264 / AVC / MPEG-4 AVC / MPEG-4 part 10"),
13. .init\_thread\_copy      = ONLY\_IF\_THREADS\_ENABLED(decode\_init\_thread\_copy),
14. .update\_thread\_context = ONLY\_IF\_THREADS\_ENABLED(decode\_update\_thread\_context),
15. .profiles = NULL\_IF\_CONFIG\_SMALL(profiles),
16. .priv\_class     = &h264\_class,
17. };

JPEG2000解码器结构体（j2kdec.c）

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/14215833) [copy](http://blog.csdn.net/leixiaohua1020/article/details/14215833)

1. AVCodec ff\_jpeg2000\_decoder = {
2. .name           = "j2k",
3. .type           = AVMEDIA\_TYPE\_VIDEO,
4. .id             = CODEC\_ID\_JPEG2000,
5. .priv\_data\_size = **sizeof**(J2kDecoderContext),
6. .init           = j2kdec\_init,
7. .close          = decode\_end,
8. .decode         = decode\_frame,
9. .capabilities = CODEC\_CAP\_EXPERIMENTAL,
10. .long\_name = NULL\_IF\_CONFIG\_SMALL("JPEG 2000"),
11. .pix\_fmts =
12. (**const** **enum** PixelFormat[]) {PIX\_FMT\_GRAY8, PIX\_FMT\_RGB24, PIX\_FMT\_NONE}
13. };

下面简单介绍一下遍历ffmpeg中的解码器信息的方法（这些解码器以一个链表的形式存储）：

1.注册所有编解码器：av\_register\_all();

2.声明一个AVCodec类型的指针，比如说AVCodec\* first\_c;

3.调用av\_codec\_next()函数，即可获得指向链表下一个解码器的指针，循环往复可以获得所有解码器的信息。注意，如果想要获得指向第一个解码器的指针，则需要将该函数的参数设置为NULL。

# [FFMPEG结构体分析：AVIOContext](http://blog.csdn.net/leixiaohua1020/article/details/14215369)

FFMPEG有几个最重要的结构体，包含了解协议，解封装，解码操作，此前已经进行过分析：

[FFMPEG中最关键的结构体之间的关系](http://blog.csdn.net/leixiaohua1020/article/details/11693997)

在此不再详述，其中AVIOContext是FFMPEG管理输入输出数据的结构体。本文将会详细分析一下该结构体里每个变量的含义和作用。

首先看一下结构体的定义（位于avio.h）：

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

1. /\* 雷霄骅
2. \* 中国传媒大学/数字电视技术
3. \* leixiaohua1020@126.com
4. \*
5. \*/
6. /\*\*
7. \* Bytestream IO Context.
8. \* New fields can be added to the end with minor version bumps.
9. \* Removal, reordering and changes to existing fields require a major
10. \* version bump.
11. \* sizeof(AVIOContext) must not be used outside libav\*.
12. \*
13. \* @note None of the function pointers in AVIOContext should be called
14. \*       directly, they should only be set by the client application
15. \*       when implementing custom I/O. Normally these are set to the
16. \*       function pointers specified in avio\_alloc\_context()
17. \*/
18. **typedef** **struct** {
19. /\*\*
20. \* A class for private options.
21. \*
22. \* If this AVIOContext is created by avio\_open2(), av\_class is set and
23. \* passes the options down to protocols.
24. \*
25. \* If this AVIOContext is manually allocated, then av\_class may be set by
26. \* the caller.
27. \*
28. \* warning -- this field can be NULL, be sure to not pass this AVIOContext
29. \* to any av\_opt\_\* functions in that case.
30. \*/
31. AVClass \*av\_class;
32. unsigned **char** \*buffer;  /\*\*< Start of the buffer. \*/
33. **int** buffer\_size;        /\*\*< Maximum buffer size \*/
34. unsigned **char** \*buf\_ptr; /\*\*< Current position in the buffer \*/
35. unsigned **char** \*buf\_end; /\*\*< End of the data, may be less than
36. buffer+buffer\_size if the read function returned
37. less data than requested, e.g. for streams where
38. no more data has been received yet. \*/
39. **void** \*opaque;           /\*\*< A private pointer, passed to the read/write/seek/...
40. functions. \*/
41. **int** (\*read\_packet)(**void** \*opaque, uint8\_t \*buf, **int** buf\_size);
42. **int** (\*write\_packet)(**void** \*opaque, uint8\_t \*buf, **int** buf\_size);
43. int64\_t (\*seek)(**void** \*opaque, int64\_t offset, **int** whence);
44. int64\_t pos;            /\*\*< position in the file of the current buffer \*/
45. **int** must\_flush;         /\*\*< true if the next seek should flush \*/
46. **int** eof\_reached;        /\*\*< true if eof reached \*/
47. **int** write\_flag;         /\*\*< true if open for writing \*/
48. **int** max\_packet\_size;
49. unsigned **long** checksum;
50. unsigned **char** \*checksum\_ptr;
51. unsigned **long** (\*update\_checksum)(unsigned **long** checksum, **const** uint8\_t \*buf, unsigned **int** size);
52. **int** error;              /\*\*< contains the error code or 0 if no error happened \*/
53. /\*\*
54. \* Pause or resume playback for network streaming protocols - e.g. MMS.
55. \*/
56. **int** (\*read\_pause)(**void** \*opaque, **int** pause);
57. /\*\*
58. \* Seek to a given timestamp in stream with the specified stream\_index.
59. \* Needed for some network streaming protocols which don't support seeking
60. \* to byte position.
61. \*/
62. int64\_t (\*read\_seek)(**void** \*opaque, **int** stream\_index,
63. int64\_t timestamp, **int** flags);
64. /\*\*
65. \* A combination of AVIO\_SEEKABLE\_ flags or 0 when the stream is not seekable.
66. \*/
67. **int** seekable;
69. /\*\*
70. \* max filesize, used to limit allocations
71. \* This field is internal to libavformat and access from outside is not allowed.
72. \*/
73. int64\_t maxsize;
74. } AVIOContext;

AVIOContext中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. };

libRTMP中代码如下（libRTMP.c）：

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

1. URLProtocol ff\_rtmp\_protocol = {
2. .name                = "rtmp",
3. .url\_open            = rtmp\_open,
4. .url\_read            = rtmp\_read,
5. .url\_write           = rtmp\_write,
6. .url\_close           = rtmp\_close,
7. .url\_read\_pause      = rtmp\_read\_pause,
8. .url\_read\_seek       = rtmp\_read\_seek,
9. .url\_get\_file\_handle = rtmp\_get\_file\_handle,
10. .priv\_data\_size      = **sizeof**(RTMP),
11. .flags               = URL\_PROTOCOL\_FLAG\_NETWORK,
12. };

udp协议代码如下（udp.c）：

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

1. URLProtocol ff\_udp\_protocol = {
2. .name                = "udp",
3. .url\_open            = udp\_open,
4. .url\_read            = udp\_read,
5. .url\_write           = udp\_write,
6. .url\_close           = udp\_close,
7. .url\_get\_file\_handle = udp\_get\_file\_handle,
8. .priv\_data\_size      = **sizeof**(UDPContext),
9. .flags               = URL\_PROTOCOL\_FLAG\_NETWORK,
10. };

等号右边的函数是完成具体读写功能的函数。可以看一下file协议的几个函数（其实就是读文件，写文件这样的操作）（file.c）：

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

1. /\*
2. \*雷霄骅
3. \*leixiaohua1020@126.com
4. \*中国传媒大学/数字电视技术
5. \*/
6. /\* standard file protocol \*/
8. **static** **int** file\_read(URLContext \*h, unsigned **char** \*buf, **int** size)
9. {
10. **int** fd = (**intptr\_t**) h->priv\_data;
11. **int** r = read(fd, buf, size);
12. **return** (-1 == r)?AVERROR(errno):r;
13. }
15. **static** **int** file\_write(URLContext \*h, **const** unsigned **char** \*buf, **int** size)
16. {
17. **int** fd = (**intptr\_t**) h->priv\_data;
18. **int** r = write(fd, buf, size);
19. **return** (-1 == r)?AVERROR(errno):r;
20. }
22. **static** **int** file\_get\_handle(URLContext \*h)
23. {
24. **return** (**intptr\_t**) h->priv\_data;
25. }
27. **static** **int** file\_check(URLContext \*h, **int** mask)
28. {
29. **struct** stat st;
30. **int** ret = stat(h->filename, &st);
31. **if** (ret < 0)
32. **return** AVERROR(errno);
34. ret |= st.st\_mode&S\_IRUSR ? mask&AVIO\_FLAG\_READ  : 0;
35. ret |= st.st\_mode&S\_IWUSR ? mask&AVIO\_FLAG\_WRITE : 0;
37. **return** ret;
38. }
40. #if CONFIG\_FILE\_PROTOCOL
42. **static** **int** file\_open(URLContext \*h, **const** **char** \*filename, **int** flags)
43. {
44. **int** access;
45. **int** fd;
47. av\_strstart(filename, "file:", &filename);
49. **if** (flags & AVIO\_FLAG\_WRITE && flags & AVIO\_FLAG\_READ) {
50. access = O\_CREAT | O\_TRUNC | O\_RDWR;
51. } **else** **if** (flags & AVIO\_FLAG\_WRITE) {
52. access = O\_CREAT | O\_TRUNC | O\_WRONLY;
53. } **else** {
54. access = O\_RDONLY;
55. }
56. #ifdef O\_BINARY
57. access |= O\_BINARY;
58. #endif
59. fd = open(filename, access, 0666);
60. **if** (fd == -1)
61. **return** AVERROR(errno);
62. h->priv\_data = (**void** \*) (**intptr\_t**) fd;
63. **return** 0;
64. }
66. /\* XXX: use llseek \*/
67. **static** int64\_t file\_seek(URLContext \*h, int64\_t pos, **int** whence)
68. {
69. **int** fd = (**intptr\_t**) h->priv\_data;
70. **if** (whence == AVSEEK\_SIZE) {
71. **struct** stat st;
72. **int** ret = fstat(fd, &st);
73. **return** ret < 0 ? AVERROR(errno) : st.st\_size;
74. }
75. **return** lseek(fd, pos, whence);
76. }
78. **static** **int** file\_close(URLContext \*h)
79. {
80. **int** fd = (**intptr\_t**) h->priv\_data;
81. **return** close(fd);
82. }

# [FFMPEG结构体分析：AVCodecContext](http://blog.csdn.net/leixiaohua1020/article/details/14214859)

FFMPEG有几个最重要的结构体，包含了解协议，解封装，解码操作，此前已经进行过分析：

[FFMPEG中最关键的结构体之间的关系](http://blog.csdn.net/leixiaohua1020/article/details/11693997)

在此不再详述，其中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）。本文将会大概分析一下该结构体里每个变量的含义和作用。因为如果每个变量都分析的话，工作量太大，实在来不及。

首先看一下结构体的定义（位于avcodec.h）：

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/14214859) [copy](http://blog.csdn.net/leixiaohua1020/article/details/14214859)

1. /\*
2. \*雷霄骅
3. \*leixiaohua1020@126.com
4. \*中国传媒大学/数字电视技术
5. \*/
6. /\*\*
7. \* main external API structure.
8. \* New fields can be added to the end with minor version bumps.
9. \* Removal, reordering and changes to existing fields require a major
10. \* version bump.
11. \* Please use AVOptions (av\_opt\* / av\_set/get\*()) to access these fields from user
12. \* applications.
13. \* sizeof(AVCodecContext) must not be used outside libav\*.
14. \*/
15. **typedef** **struct** AVCodecContext {
16. /\*\*
17. \* information on struct for av\_log
18. \* - set by avcodec\_alloc\_context3
19. \*/
20. **const** AVClass \*av\_class;
21. **int** log\_level\_offset;
23. **enum** AVMediaType codec\_type; /\* see AVMEDIA\_TYPE\_xxx \*/
24. **const** **struct** AVCodec  \*codec;
25. **char**             codec\_name[32];
26. **enum** AVCodecID     codec\_id; /\* see AV\_CODEC\_ID\_xxx \*/
28. /\*\*
29. \* fourcc (LSB first, so "ABCD" -> ('D'<<24) + ('C'<<16) + ('B'<<8) + 'A').
30. \* This is used to work around some encoder bugs.
31. \* A demuxer should set this to what is stored in the field used to identify the codec.
32. \* If there are multiple such fields in a container then the demuxer should choose the one
33. \* which maximizes the information about the used codec.
34. \* If the codec tag field in a container is larger than 32 bits then the demuxer should
35. \* remap the longer ID to 32 bits with a table or other structure. Alternatively a new
36. \* extra\_codec\_tag + size could be added but for this a clear advantage must be demonstrated
37. \* first.
38. \* - encoding: Set by user, if not then the default based on codec\_id will be used.
39. \* - decoding: Set by user, will be converted to uppercase by libavcodec during init.
40. \*/
41. unsigned **int** codec\_tag;
43. /\*\*
44. \* fourcc from the AVI stream header (LSB first, so "ABCD" -> ('D'<<24) + ('C'<<16) + ('B'<<8) + 'A').
45. \* This is used to work around some encoder bugs.
46. \* - encoding: unused
47. \* - decoding: Set by user, will be converted to uppercase by libavcodec during init.
48. \*/
49. unsigned **int** stream\_codec\_tag;
51. #if FF\_API\_SUB\_ID
52. /\*\*
53. \* @deprecated this field is unused
54. \*/
55. attribute\_deprecated **int** sub\_id;
56. #endif
58. **void** \*priv\_data;
60. /\*\*
61. \* Private context used for internal data.
62. \*
63. \* Unlike priv\_data, this is not codec-specific. It is used in general
64. \* libavcodec functions.
65. \*/
66. **struct** AVCodecInternal \*internal;
68. /\*\*
69. \* Private data of the user, can be used to carry app specific stuff.
70. \* - encoding: Set by user.
71. \* - decoding: Set by user.
72. \*/
73. **void** \*opaque;
75. /\*\*
76. \* the average bitrate
77. \* - encoding: Set by user; unused for constant quantizer encoding.
78. \* - decoding: Set by libavcodec. 0 or some bitrate if this info is available in the stream.
79. \*/
80. **int** bit\_rate;
82. /\*\*
83. \* number of bits the bitstream is allowed to diverge from the reference.
84. \*           the reference can be CBR (for CBR pass1) or VBR (for pass2)
85. \* - encoding: Set by user; unused for constant quantizer encoding.
86. \* - decoding: unused
87. \*/
88. **int** bit\_rate\_tolerance;
90. /\*\*
91. \* Global quality for codecs which cannot change it per frame.
92. \* This should be proportional to MPEG-1/2/4 qscale.
93. \* - encoding: Set by user.
94. \* - decoding: unused
95. \*/
96. **int** global\_quality;
98. /\*\*
99. \* - encoding: Set by user.
100. \* - decoding: unused
101. \*/
102. **int** compression\_level;
103. #define FF\_COMPRESSION\_DEFAULT -1
105. /\*\*
106. \* CODEC\_FLAG\_\*.
107. \* - encoding: Set by user.
108. \* - decoding: Set by user.
109. \*/
110. **int** flags;
112. /\*\*
113. \* CODEC\_FLAG2\_\*
114. \* - encoding: Set by user.
115. \* - decoding: Set by user.
116. \*/
117. **int** flags2;
119. /\*\*
120. \* some codecs need / can use extradata like Huffman tables.
121. \* mjpeg: Huffman tables
122. \* rv10: additional flags
123. \* mpeg4: global headers (they can be in the bitstream or here)
124. \* The allocated memory should be FF\_INPUT\_BUFFER\_PADDING\_SIZE bytes larger
125. \* than extradata\_size to avoid prolems if it is read with the bitstream reader.
126. \* The bytewise contents of extradata must not depend on the architecture or CPU endianness.
127. \* - encoding: Set/allocated/freed by libavcodec.
128. \* - decoding: Set/allocated/freed by user.
129. \*/
130. uint8\_t \*extradata;
131. **int** extradata\_size;
133. /\*\*
134. \* This is the fundamental unit of time (in seconds) in terms
135. \* of which frame timestamps are represented. For fixed-fps content,
136. \* timebase should be 1/framerate and timestamp increments should be
137. \* identically 1.
138. \* - encoding: MUST be set by user.
139. \* - decoding: Set by libavcodec.
140. \*/
141. AVRational time\_base;
143. /\*\*
144. \* For some codecs, the time base is closer to the field rate than the frame rate.
145. \* Most notably, H.264 and MPEG-2 specify time\_base as half of frame duration
146. \* if no telecine is used ...
147. \*
148. \* Set to time\_base ticks per frame. Default 1, e.g., H.264/MPEG-2 set it to 2.
149. \*/
150. **int** ticks\_per\_frame;
152. /\*\*
153. \* Encoding: Number of frames delay there will be from the encoder input to
154. \*           the decoder output. (we assume the decoder matches the spec)
155. \* Decoding: Number of frames delay in addition to what a standard decoder
156. \*           as specified in the spec would produce.
157. \*
158. \* Video:
159. \*   Number of frames the decoded output will be delayed relative to the
160. \*   encoded input.
161. \*
162. \* Audio:
163. \*   For encoding, this is the number of "priming" samples added to the
164. \*   beginning of the stream. The decoded output will be delayed by this
165. \*   many samples relative to the input to the encoder. Note that this
166. \*   field is purely informational and does not directly affect the pts
167. \*   output by the encoder, which should always be based on the actual
168. \*   presentation time, including any delay.
169. \*   For decoding, this is the number of samples the decoder needs to
170. \*   output before the decoder's output is valid. When seeking, you should
171. \*   start decoding this many samples prior to your desired seek point.
172. \*
173. \* - encoding: Set by libavcodec.
174. \* - decoding: Set by libavcodec.
175. \*/
176. **int** delay;

179. /\* video only \*/
180. /\*\*
181. \* picture width / height.
182. \* - encoding: MUST be set by user.
183. \* - decoding: Set by libavcodec.
184. \* Note: For compatibility it is possible to set this instead of
185. \* coded\_width/height before decoding.
186. \*/
187. **int** width, height;
189. /\*\*
190. \* Bitstream width / height, may be different from width/height if lowres enabled.
191. \* - encoding: unused
192. \* - decoding: Set by user before init if known. Codec should override / dynamically change if needed.
193. \*/
194. **int** coded\_width, coded\_height;
196. #define FF\_ASPECT\_EXTENDED 15
198. /\*\*
199. \* the number of pictures in a group of pictures, or 0 for intra\_only
200. \* - encoding: Set by user.
201. \* - decoding: unused
202. \*/
203. **int** gop\_size;
205. /\*\*
206. \* Pixel format, see AV\_PIX\_FMT\_xxx.
207. \* May be set by the demuxer if known from headers.
208. \* May be overridden by the decoder if it knows better.
209. \* - encoding: Set by user.
210. \* - decoding: Set by user if known, overridden by libavcodec if known
211. \*/
212. **enum** AVPixelFormat pix\_fmt;
214. /\*\*
215. \* Motion estimation algorithm used for video coding.
216. \* 1 (zero), 2 (full), 3 (log), 4 (phods), 5 (epzs), 6 (x1), 7 (hex),
217. \* 8 (umh), 9 (iter), 10 (tesa) [7, 8, 10 are x264 specific, 9 is snow specific]
218. \* - encoding: MUST be set by user.
219. \* - decoding: unused
220. \*/
221. **int** me\_method;
223. /\*\*
224. \* If non NULL, 'draw\_horiz\_band' is called by the libavcodec
225. \* decoder to draw a horizontal band. It improves cache usage. Not
226. \* all codecs can do that. You must check the codec capabilities
227. \* beforehand.
228. \* When multithreading is used, it may be called from multiple threads
229. \* at the same time; threads might draw different parts of the same AVFrame,
230. \* or multiple AVFrames, and there is no guarantee that slices will be drawn
231. \* in order.
232. \* The function is also used by hardware acceleration APIs.
233. \* It is called at least once during frame decoding to pass
234. \* the data needed for hardware render.
235. \* In that mode instead of pixel data, AVFrame points to
236. \* a structure specific to the acceleration API. The application
237. \* reads the structure and can change some fields to indicate progress
238. \* or mark state.
239. \* - encoding: unused
240. \* - decoding: Set by user.
241. \* @param height the height of the slice
242. \* @param y the y position of the slice
243. \* @param type 1->top field, 2->bottom field, 3->frame
244. \* @param offset offset into the AVFrame.data from which the slice should be read
245. \*/
246. **void** (\*draw\_horiz\_band)(**struct** AVCodecContext \*s,
247. **const** AVFrame \*src, **int** offset[AV\_NUM\_DATA\_POINTERS],
248. **int** y, **int** type, **int** height);
250. /\*\*
251. \* callback to negotiate the pixelFormat
252. \* @param fmt is the list of formats which are supported by the codec,
253. \* it is terminated by -1 as 0 is a valid format, the formats are ordered by quality.
254. \* The first is always the native one.
255. \* @return the chosen format
256. \* - encoding: unused
257. \* - decoding: Set by user, if not set the native format will be chosen.
258. \*/
259. **enum** AVPixelFormat (\*get\_format)(**struct** AVCodecContext \*s, **const** **enum** AVPixelFormat \* fmt);
261. /\*\*
262. \* maximum number of B-frames between non-B-frames
263. \* Note: The output will be delayed by max\_b\_frames+1 relative to the input.
264. \* - encoding: Set by user.
265. \* - decoding: unused
266. \*/
267. **int** max\_b\_frames;
269. /\*\*
270. \* qscale factor between IP and B-frames
271. \* If > 0 then the last P-frame quantizer will be used (q= lastp\_q\*factor+offset).
272. \* If < 0 then normal ratecontrol will be done (q= -normal\_q\*factor+offset).
273. \* - encoding: Set by user.
274. \* - decoding: unused
275. \*/
276. **float** b\_quant\_factor;
278. /\*\* obsolete FIXME remove \*/
279. **int** rc\_strategy;
280. #define FF\_RC\_STRATEGY\_XVID 1
282. **int** b\_frame\_strategy;
284. #if FF\_API\_MPV\_GLOBAL\_OPTS
285. /\*\*
286. \* luma single coefficient elimination threshold
287. \* - encoding: Set by user.
288. \* - decoding: unused
289. \*/
290. attribute\_deprecated **int** luma\_elim\_threshold;
292. /\*\*
293. \* chroma single coeff elimination threshold
294. \* - encoding: Set by user.
295. \* - decoding: unused
296. \*/
297. attribute\_deprecated **int** chroma\_elim\_threshold;
298. #endif
300. /\*\*
301. \* qscale offset between IP and B-frames
302. \* - encoding: Set by user.
303. \* - decoding: unused
304. \*/
305. **float** b\_quant\_offset;
307. /\*\*
308. \* Size of the frame reordering buffer in the decoder.
309. \* For MPEG-2 it is 1 IPB or 0 low delay IP.
310. \* - encoding: Set by libavcodec.
311. \* - decoding: Set by libavcodec.
312. \*/
313. **int** has\_b\_frames;
315. /\*\*
316. \* 0-> h263 quant 1-> mpeg quant
317. \* - encoding: Set by user.
318. \* - decoding: unused
319. \*/
320. **int** mpeg\_quant;
322. /\*\*
323. \* qscale factor between P and I-frames
324. \* If > 0 then the last p frame quantizer will be used (q= lastp\_q\*factor+offset).
325. \* If < 0 then normal ratecontrol will be done (q= -normal\_q\*factor+offset).
326. \* - encoding: Set by user.
327. \* - decoding: unused
328. \*/
329. **float** i\_quant\_factor;
331. /\*\*
332. \* qscale offset between P and I-frames
333. \* - encoding: Set by user.
334. \* - decoding: unused
335. \*/
336. **float** i\_quant\_offset;
338. /\*\*
339. \* luminance masking (0-> disabled)
340. \* - encoding: Set by user.
341. \* - decoding: unused
342. \*/
343. **float** lumi\_masking;
345. /\*\*
346. \* temporary complexity masking (0-> disabled)
347. \* - encoding: Set by user.
348. \* - decoding: unused
349. \*/
350. **float** temporal\_cplx\_masking;
352. /\*\*
353. \* spatial complexity masking (0-> disabled)
354. \* - encoding: Set by user.
355. \* - decoding: unused
356. \*/
357. **float** spatial\_cplx\_masking;
359. /\*\*
360. \* p block masking (0-> disabled)
361. \* - encoding: Set by user.
362. \* - decoding: unused
363. \*/
364. **float** p\_masking;
366. /\*\*
367. \* darkness masking (0-> disabled)
368. \* - encoding: Set by user.
369. \* - decoding: unused
370. \*/
371. **float** dark\_masking;
373. /\*\*
374. \* slice count
375. \* - encoding: Set by libavcodec.
376. \* - decoding: Set by user (or 0).
377. \*/
378. **int** slice\_count;
379. /\*\*
380. \* prediction method (needed for huffyuv)
381. \* - encoding: Set by user.
382. \* - decoding: unused
383. \*/
384. **int** prediction\_method;
385. #define FF\_PRED\_LEFT   0
386. #define FF\_PRED\_PLANE  1
387. #define FF\_PRED\_MEDIAN 2
389. /\*\*
390. \* slice offsets in the frame in bytes
391. \* - encoding: Set/allocated by libavcodec.
392. \* - decoding: Set/allocated by user (or NULL).
393. \*/
394. **int** \*slice\_offset;
396. /\*\*
397. \* sample aspect ratio (0 if unknown)
398. \* That is the width of a pixel divided by the height of the pixel.
399. \* Numerator and denominator must be relatively prime and smaller than 256 for some video standards.
400. \* - encoding: Set by user.
401. \* - decoding: Set by libavcodec.
402. \*/
403. AVRational sample\_aspect\_ratio;
405. /\*\*
406. \* motion estimation comparison function
407. \* - encoding: Set by user.
408. \* - decoding: unused
409. \*/
410. **int** me\_cmp;
411. /\*\*
412. \* subpixel motion estimation comparison function
413. \* - encoding: Set by user.
414. \* - decoding: unused
415. \*/
416. **int** me\_sub\_cmp;
417. /\*\*
418. \* macroblock comparison function (not supported yet)
419. \* - encoding: Set by user.
420. \* - decoding: unused
421. \*/
422. **int** mb\_cmp;
423. /\*\*
424. \* interlaced DCT comparison function
425. \* - encoding: Set by user.
426. \* - decoding: unused
427. \*/
428. **int** ildct\_cmp;
429. #define FF\_CMP\_SAD    0
430. #define FF\_CMP\_SSE    1
431. #define FF\_CMP\_SATD   2
432. #define FF\_CMP\_DCT    3
433. #define FF\_CMP\_PSNR   4
434. #define FF\_CMP\_BIT    5
435. #define FF\_CMP\_RD     6
436. #define FF\_CMP\_ZERO   7
437. #define FF\_CMP\_VSAD   8
438. #define FF\_CMP\_VSSE   9
439. #define FF\_CMP\_NSSE   10
440. #define FF\_CMP\_W53    11
441. #define FF\_CMP\_W97    12
442. #define FF\_CMP\_DCTMAX 13
443. #define FF\_CMP\_DCT264 14
444. #define FF\_CMP\_CHROMA 256
446. /\*\*
447. \* ME diamond size & shape
448. \* - encoding: Set by user.
449. \* - decoding: unused
450. \*/
451. **int** dia\_size;
453. /\*\*
454. \* amount of previous MV predictors (2a+1 x 2a+1 square)
455. \* - encoding: Set by user.
456. \* - decoding: unused
457. \*/
458. **int** last\_predictor\_count;
460. /\*\*
461. \* prepass for motion estimation
462. \* - encoding: Set by user.
463. \* - decoding: unused
464. \*/
465. **int** pre\_me;
467. /\*\*
468. \* motion estimation prepass comparison function
469. \* - encoding: Set by user.
470. \* - decoding: unused
471. \*/
472. **int** me\_pre\_cmp;
474. /\*\*
475. \* ME prepass diamond size & shape
476. \* - encoding: Set by user.
477. \* - decoding: unused
478. \*/
479. **int** pre\_dia\_size;
481. /\*\*
482. \* subpel ME quality
483. \* - encoding: Set by user.
484. \* - decoding: unused
485. \*/
486. **int** me\_subpel\_quality;
488. /\*\*
489. \* DTG active format information (additional aspect ratio
490. \* information only used in DVB MPEG-2 transport streams)
491. \* 0 if not set.
492. \*
493. \* - encoding: unused
494. \* - decoding: Set by decoder.
495. \*/
496. **int** dtg\_active\_format;
497. #define FF\_DTG\_AFD\_SAME         8
498. #define FF\_DTG\_AFD\_4\_3          9
499. #define FF\_DTG\_AFD\_16\_9         10
500. #define FF\_DTG\_AFD\_14\_9         11
501. #define FF\_DTG\_AFD\_4\_3\_SP\_14\_9  13
502. #define FF\_DTG\_AFD\_16\_9\_SP\_14\_9 14
503. #define FF\_DTG\_AFD\_SP\_4\_3       15
505. /\*\*
506. \* maximum motion estimation search range in subpel units
507. \* If 0 then no limit.
508. \*
509. \* - encoding: Set by user.
510. \* - decoding: unused
511. \*/
512. **int** me\_range;
514. /\*\*
515. \* intra quantizer bias
516. \* - encoding: Set by user.
517. \* - decoding: unused
518. \*/
519. **int** intra\_quant\_bias;
520. #define FF\_DEFAULT\_QUANT\_BIAS 999999
522. /\*\*
523. \* inter quantizer bias
524. \* - encoding: Set by user.
525. \* - decoding: unused
526. \*/
527. **int** inter\_quant\_bias;
529. #if FF\_API\_COLOR\_TABLE\_ID
530. /\*\*
531. \* color table ID
532. \* - encoding: unused
533. \* - decoding: Which clrtable should be used for 8bit RGB images.
534. \*             Tables have to be stored somewhere. FIXME
535. \*/
536. attribute\_deprecated **int** color\_table\_id;
537. #endif
539. /\*\*
540. \* slice flags
541. \* - encoding: unused
542. \* - decoding: Set by user.
543. \*/
544. **int** slice\_flags;
545. #define SLICE\_FLAG\_CODED\_ORDER    0x0001 ///< draw\_horiz\_band() is called in coded order instead of display
546. #define SLICE\_FLAG\_ALLOW\_FIELD    0x0002 ///< allow draw\_horiz\_band() with field slices (MPEG2 field pics)
547. #define SLICE\_FLAG\_ALLOW\_PLANE    0x0004 ///< allow draw\_horiz\_band() with 1 component at a time (SVQ1)
549. /\*\*
550. \* XVideo Motion Acceleration
551. \* - encoding: forbidden
552. \* - decoding: set by decoder
553. \*/
554. **int** xvmc\_acceleration;
556. /\*\*
557. \* macroblock decision mode
558. \* - encoding: Set by user.
559. \* - decoding: unused
560. \*/
561. **int** mb\_decision;
562. #define FF\_MB\_DECISION\_SIMPLE 0        ///< uses mb\_cmp
563. #define FF\_MB\_DECISION\_BITS   1        ///< chooses the one which needs the fewest bits
564. #define FF\_MB\_DECISION\_RD     2        ///< rate distortion
566. /\*\*
567. \* custom intra quantization matrix
568. \* - encoding: Set by user, can be NULL.
569. \* - decoding: Set by libavcodec.
570. \*/
571. uint16\_t \*intra\_matrix;
573. /\*\*
574. \* custom inter quantization matrix
575. \* - encoding: Set by user, can be NULL.
576. \* - decoding: Set by libavcodec.
577. \*/
578. uint16\_t \*inter\_matrix;
580. /\*\*
581. \* scene change detection threshold
582. \* 0 is default, larger means fewer detected scene changes.
583. \* - encoding: Set by user.
584. \* - decoding: unused
585. \*/
586. **int** scenechange\_threshold;
588. /\*\*
589. \* noise reduction strength
590. \* - encoding: Set by user.
591. \* - decoding: unused
592. \*/
593. **int** noise\_reduction;
595. #if FF\_API\_INTER\_THRESHOLD
596. /\*\*
597. \* @deprecated this field is unused
598. \*/
599. attribute\_deprecated **int** inter\_threshold;
600. #endif
602. #if FF\_API\_MPV\_GLOBAL\_OPTS
603. /\*\*
604. \* @deprecated use mpegvideo private options instead
605. \*/
606. attribute\_deprecated **int** quantizer\_noise\_shaping;
607. #endif
609. /\*\*
610. \* Motion estimation threshold below which no motion estimation is
611. \* performed, but instead the user specified motion vectors are used.
612. \*
613. \* - encoding: Set by user.
614. \* - decoding: unused
615. \*/
616. **int** me\_threshold;
618. /\*\*
619. \* Macroblock threshold below which the user specified macroblock types will be used.
620. \* - encoding: Set by user.
621. \* - decoding: unused
622. \*/
623. **int** mb\_threshold;
625. /\*\*
626. \* precision of the intra DC coefficient - 8
627. \* - encoding: Set by user.
628. \* - decoding: unused
629. \*/
630. **int** intra\_dc\_precision;
632. /\*\*
633. \* Number of macroblock rows at the top which are skipped.
634. \* - encoding: unused
635. \* - decoding: Set by user.
636. \*/
637. **int** skip\_top;
639. /\*\*
640. \* Number of macroblock rows at the bottom which are skipped.
641. \* - encoding: unused
642. \* - decoding: Set by user.
643. \*/
644. **int** skip\_bottom;
646. /\*\*
647. \* Border processing masking, raises the quantizer for mbs on the borders
648. \* of the picture.
649. \* - encoding: Set by user.
650. \* - decoding: unused
651. \*/
652. **float** border\_masking;
654. /\*\*
655. \* minimum MB lagrange multipler
656. \* - encoding: Set by user.
657. \* - decoding: unused
658. \*/
659. **int** mb\_lmin;
661. /\*\*
662. \* maximum MB lagrange multipler
663. \* - encoding: Set by user.
664. \* - decoding: unused
665. \*/
666. **int** mb\_lmax;
668. /\*\*
669. \*
670. \* - encoding: Set by user.
671. \* - decoding: unused
672. \*/
673. **int** me\_penalty\_compensation;
675. /\*\*
676. \*
677. \* - encoding: Set by user.
678. \* - decoding: unused
679. \*/
680. **int** bidir\_refine;
682. /\*\*
683. \*
684. \* - encoding: Set by user.
685. \* - decoding: unused
686. \*/
687. **int** brd\_scale;
689. /\*\*
690. \* minimum GOP size
691. \* - encoding: Set by user.
692. \* - decoding: unused
693. \*/
694. **int** keyint\_min;
696. /\*\*
697. \* number of reference frames
698. \* - encoding: Set by user.
699. \* - decoding: Set by lavc.
700. \*/
701. **int** refs;
703. /\*\*
704. \* chroma qp offset from luma
705. \* - encoding: Set by user.
706. \* - decoding: unused
707. \*/
708. **int** chromaoffset;
710. /\*\*
711. \* Multiplied by qscale for each frame and added to scene\_change\_score.
712. \* - encoding: Set by user.
713. \* - decoding: unused
714. \*/
715. **int** scenechange\_factor;
717. /\*\*
718. \*
719. \* Note: Value depends upon the compare function used for fullpel ME.
720. \* - encoding: Set by user.
721. \* - decoding: unused
722. \*/
723. **int** mv0\_threshold;
725. /\*\*
726. \* Adjust sensitivity of b\_frame\_strategy 1.
727. \* - encoding: Set by user.
728. \* - decoding: unused
729. \*/
730. **int** b\_sensitivity;
732. /\*\*
733. \* Chromaticity coordinates of the source primaries.
734. \* - encoding: Set by user
735. \* - decoding: Set by libavcodec
736. \*/
737. **enum** AVColorPrimaries color\_primaries;
739. /\*\*
740. \* Color Transfer Characteristic.
741. \* - encoding: Set by user
742. \* - decoding: Set by libavcodec
743. \*/
744. **enum** AVColorTransferCharacteristic color\_trc;
746. /\*\*
747. \* YUV colorspace type.
748. \* - encoding: Set by user
749. \* - decoding: Set by libavcodec
750. \*/
751. **enum** AVColorSpace colorspace;
753. /\*\*
754. \* MPEG vs JPEG YUV range.
755. \* - encoding: Set by user
756. \* - decoding: Set by libavcodec
757. \*/
758. **enum** AVColorRange color\_range;
760. /\*\*
761. \* This defines the location of chroma samples.
762. \* - encoding: Set by user
763. \* - decoding: Set by libavcodec
764. \*/
765. **enum** AVChromaLocation chroma\_sample\_location;
767. /\*\*
768. \* Number of slices.
769. \* Indicates number of picture subdivisions. Used for parallelized
770. \* decoding.
771. \* - encoding: Set by user
772. \* - decoding: unused
773. \*/
774. **int** slices;
776. /\*\* Field order
777. \* - encoding: set by libavcodec
778. \* - decoding: Set by user.
779. \*/
780. **enum** AVFieldOrder field\_order;
782. /\* audio only \*/
783. **int** sample\_rate; ///< samples per second
784. **int** channels;    ///< number of audio channels
786. /\*\*
787. \* audio sample format
788. \* - encoding: Set by user.
789. \* - decoding: Set by libavcodec.
790. \*/
791. **enum** AVSampleFormat sample\_fmt;  ///< sample format
793. /\* The following data should not be initialized. \*/
794. /\*\*
795. \* Samples per packet, initialized when calling 'init'.
796. \*/
797. **int** frame\_size;
799. /\*\*
800. \* Frame counter, set by libavcodec.
801. \*
802. \* - decoding: total number of frames returned from the decoder so far.
803. \* - encoding: total number of frames passed to the encoder so far.
804. \*
805. \*   @note the counter is not incremented if encoding/decoding resulted in
806. \*   an error.
807. \*/
808. **int** frame\_number;
810. /\*\*
811. \* number of bytes per packet if constant and known or 0
812. \* Used by some WAV based audio codecs.
813. \*/
814. **int** block\_align;
816. /\*\*
817. \* Audio cutoff bandwidth (0 means "automatic")
818. \* - encoding: Set by user.
819. \* - decoding: unused
820. \*/
821. **int** cutoff;
823. #if FF\_API\_REQUEST\_CHANNELS
824. /\*\*
825. \* Decoder should decode to this many channels if it can (0 for default)
826. \* - encoding: unused
827. \* - decoding: Set by user.
828. \* @deprecated Deprecated in favor of request\_channel\_layout.
829. \*/
830. **int** request\_channels;
831. #endif
833. /\*\*
834. \* Audio channel layout.
835. \* - encoding: set by user.
836. \* - decoding: set by user, may be overwritten by libavcodec.
837. \*/
838. uint64\_t channel\_layout;
840. /\*\*
841. \* Request decoder to use this channel layout if it can (0 for default)
842. \* - encoding: unused
843. \* - decoding: Set by user.
844. \*/
845. uint64\_t request\_channel\_layout;
847. /\*\*
848. \* Type of service that the audio stream conveys.
849. \* - encoding: Set by user.
850. \* - decoding: Set by libavcodec.
851. \*/
852. **enum** AVAudioServiceType audio\_service\_type;
854. /\*\*
855. \* desired sample format
856. \* - encoding: Not used.
857. \* - decoding: Set by user.
858. \* Decoder will decode to this format if it can.
859. \*/
860. **enum** AVSampleFormat request\_sample\_fmt;
862. /\*\*
863. \* Called at the beginning of each frame to get a buffer for it.
864. \*
865. \* The function will set AVFrame.data[], AVFrame.linesize[].
866. \* AVFrame.extended\_data[] must also be set, but it should be the same as
867. \* AVFrame.data[] except for planar audio with more channels than can fit
868. \* in AVFrame.data[]. In that case, AVFrame.data[] shall still contain as
869. \* many data pointers as it can hold.
870. \*
871. \* if CODEC\_CAP\_DR1 is not set then get\_buffer() must call
872. \* avcodec\_default\_get\_buffer() instead of providing buffers allocated by
873. \* some other means.
874. \*
875. \* AVFrame.data[] should be 32- or 16-byte-aligned unless the CPU doesn't
876. \* need it. avcodec\_default\_get\_buffer() aligns the output buffer properly,
877. \* but if get\_buffer() is overridden then alignment considerations should
878. \* be taken into account.
879. \*
880. \* @see avcodec\_default\_get\_buffer()
881. \*
882. \* Video:
883. \*
884. \* If pic.reference is set then the frame will be read later by libavcodec.
885. \* avcodec\_align\_dimensions2() should be used to find the required width and
886. \* height, as they normally need to be rounded up to the next multiple of 16.
887. \*
888. \* If frame multithreading is used and thread\_safe\_callbacks is set,
889. \* it may be called from a different thread, but not from more than one at
890. \* once. Does not need to be reentrant.
891. \*
892. \* @see release\_buffer(), reget\_buffer()
893. \* @see avcodec\_align\_dimensions2()
894. \*
895. \* Audio:
896. \*
897. \* Decoders request a buffer of a particular size by setting
898. \* AVFrame.nb\_samples prior to calling get\_buffer(). The decoder may,
899. \* however, utilize only part of the buffer by setting AVFrame.nb\_samples
900. \* to a smaller value in the output frame.
901. \*
902. \* Decoders cannot use the buffer after returning from
903. \* avcodec\_decode\_audio4(), so they will not call release\_buffer(), as it
904. \* is assumed to be released immediately upon return.
905. \*
906. \* As a convenience, av\_samples\_get\_buffer\_size() and
907. \* av\_samples\_fill\_arrays() in libavutil may be used by custom get\_buffer()
908. \* functions to find the required data size and to fill data pointers and
909. \* linesize. In AVFrame.linesize, only linesize[0] may be set for audio
910. \* since all planes must be the same size.
911. \*
912. \* @see av\_samples\_get\_buffer\_size(), av\_samples\_fill\_arrays()
913. \*
914. \* - encoding: unused
915. \* - decoding: Set by libavcodec, user can override.
916. \*/
917. **int** (\*get\_buffer)(**struct** AVCodecContext \*c, AVFrame \*pic);
919. /\*\*
920. \* Called to release buffers which were allocated with get\_buffer.
921. \* A released buffer can be reused in get\_buffer().
922. \* pic.data[\*] must be set to NULL.
923. \* May be called from a different thread if frame multithreading is used,
924. \* but not by more than one thread at once, so does not need to be reentrant.
925. \* - encoding: unused
926. \* - decoding: Set by libavcodec, user can override.
927. \*/
928. **void** (\*release\_buffer)(**struct** AVCodecContext \*c, AVFrame \*pic);
930. /\*\*
931. \* Called at the beginning of a frame to get cr buffer for it.
932. \* Buffer type (size, hints) must be the same. libavcodec won't check it.
933. \* libavcodec will pass previous buffer in pic, function should return
934. \* same buffer or new buffer with old frame "painted" into it.
935. \* If pic.data[0] == NULL must behave like get\_buffer().
936. \* if CODEC\_CAP\_DR1 is not set then reget\_buffer() must call
937. \* avcodec\_default\_reget\_buffer() instead of providing buffers allocated by
938. \* some other means.
939. \* - encoding: unused
940. \* - decoding: Set by libavcodec, user can override.
941. \*/
942. **int** (\*reget\_buffer)(**struct** AVCodecContext \*c, AVFrame \*pic);

945. /\* - encoding parameters \*/
946. **float** qcompress;  ///< amount of qscale change between easy & hard scenes (0.0-1.0)
947. **float** qblur;      ///< amount of qscale smoothing over time (0.0-1.0)
949. /\*\*
950. \* minimum quantizer
951. \* - encoding: Set by user.
952. \* - decoding: unused
953. \*/
954. **int** qmin;
956. /\*\*
957. \* maximum quantizer
958. \* - encoding: Set by user.
959. \* - decoding: unused
960. \*/
961. **int** qmax;
963. /\*\*
964. \* maximum quantizer difference between frames
965. \* - encoding: Set by user.
966. \* - decoding: unused
967. \*/
968. **int** max\_qdiff;
970. /\*\*
971. \* ratecontrol qmin qmax limiting method
972. \* 0-> clipping, 1-> use a nice continuous function to limit qscale wthin qmin/qmax.
973. \* - encoding: Set by user.
974. \* - decoding: unused
975. \*/
976. **float** rc\_qsquish;
978. **float** rc\_qmod\_amp;
979. **int** rc\_qmod\_freq;
981. /\*\*
982. \* decoder bitstream buffer size
983. \* - encoding: Set by user.
984. \* - decoding: unused
985. \*/
986. **int** rc\_buffer\_size;
988. /\*\*
989. \* ratecontrol override, see RcOverride
990. \* - encoding: Allocated/set/freed by user.
991. \* - decoding: unused
992. \*/
993. **int** rc\_override\_count;
994. RcOverride \*rc\_override;
996. /\*\*
997. \* rate control equation
998. \* - encoding: Set by user
999. \* - decoding: unused
1000. \*/
1001. **const** **char** \*rc\_eq;
1003. /\*\*
1004. \* maximum bitrate
1005. \* - encoding: Set by user.
1006. \* - decoding: unused
1007. \*/
1008. **int** rc\_max\_rate;
1010. /\*\*
1011. \* minimum bitrate
1012. \* - encoding: Set by user.
1013. \* - decoding: unused
1014. \*/
1015. **int** rc\_min\_rate;
1017. **float** rc\_buffer\_aggressivity;
1019. /\*\*
1020. \* initial complexity for pass1 ratecontrol
1021. \* - encoding: Set by user.
1022. \* - decoding: unused
1023. \*/
1024. **float** rc\_initial\_cplx;
1026. /\*\*
1027. \* Ratecontrol attempt to use, at maximum, <value> of what can be used without an underflow.
1028. \* - encoding: Set by user.
1029. \* - decoding: unused.
1030. \*/
1031. **float** rc\_max\_available\_vbv\_use;
1033. /\*\*
1034. \* Ratecontrol attempt to use, at least, <value> times the amount needed to prevent a vbv overflow.
1035. \* - encoding: Set by user.
1036. \* - decoding: unused.
1037. \*/
1038. **float** rc\_min\_vbv\_overflow\_use;
1040. /\*\*
1041. \* Number of bits which should be loaded into the rc buffer before decoding starts.
1042. \* - encoding: Set by user.
1043. \* - decoding: unused
1044. \*/
1045. **int** rc\_initial\_buffer\_occupancy;
1047. #define FF\_CODER\_TYPE\_VLC       0
1048. #define FF\_CODER\_TYPE\_AC        1
1049. #define FF\_CODER\_TYPE\_RAW       2
1050. #define FF\_CODER\_TYPE\_RLE       3
1051. #define FF\_CODER\_TYPE\_DEFLATE   4
1052. /\*\*
1053. \* coder type
1054. \* - encoding: Set by user.
1055. \* - decoding: unused
1056. \*/
1057. **int** coder\_type;
1059. /\*\*
1060. \* context model
1061. \* - encoding: Set by user.
1062. \* - decoding: unused
1063. \*/
1064. **int** context\_model;
1066. /\*\*
1067. \* minimum Lagrange multipler
1068. \* - encoding: Set by user.
1069. \* - decoding: unused
1070. \*/
1071. **int** lmin;
1073. /\*\*
1074. \* maximum Lagrange multipler
1075. \* - encoding: Set by user.
1076. \* - decoding: unused
1077. \*/
1078. **int** lmax;
1080. /\*\*
1081. \* frame skip threshold
1082. \* - encoding: Set by user.
1083. \* - decoding: unused
1084. \*/
1085. **int** frame\_skip\_threshold;
1087. /\*\*
1088. \* frame skip factor
1089. \* - encoding: Set by user.
1090. \* - decoding: unused
1091. \*/
1092. **int** frame\_skip\_factor;
1094. /\*\*
1095. \* frame skip exponent
1096. \* - encoding: Set by user.
1097. \* - decoding: unused
1098. \*/
1099. **int** frame\_skip\_exp;
1101. /\*\*
1102. \* frame skip comparison function
1103. \* - encoding: Set by user.
1104. \* - decoding: unused
1105. \*/
1106. **int** frame\_skip\_cmp;
1108. /\*\*
1109. \* trellis RD quantization
1110. \* - encoding: Set by user.
1111. \* - decoding: unused
1112. \*/
1113. **int** trellis;
1115. /\*\*
1116. \* - encoding: Set by user.
1117. \* - decoding: unused
1118. \*/
1119. **int** min\_prediction\_order;
1121. /\*\*
1122. \* - encoding: Set by user.
1123. \* - decoding: unused
1124. \*/
1125. **int** max\_prediction\_order;
1127. /\*\*
1128. \* GOP timecode frame start number
1129. \* - encoding: Set by user, in non drop frame format
1130. \* - decoding: Set by libavcodec (timecode in the 25 bits format, -1 if unset)
1131. \*/
1132. int64\_t timecode\_frame\_start;
1134. /\* The RTP callback: This function is called    \*/
1135. /\* every time the encoder has a packet to send. \*/
1136. /\* It depends on the encoder if the data starts \*/
1137. /\* with a Start Code (it should). H.263 does.   \*/
1138. /\* mb\_nb contains the number of macroblocks     \*/
1139. /\* encoded in the RTP payload.                  \*/
1140. **void** (\*rtp\_callback)(**struct** AVCodecContext \*avctx, **void** \*data, **int** size, **int** mb\_nb);
1142. **int** rtp\_payload\_size;   /\* The size of the RTP payload: the coder will  \*/
1143. /\* do its best to deliver a chunk with size     \*/
1144. /\* below rtp\_payload\_size, the chunk will start \*/
1145. /\* with a start code on some codecs like H.263. \*/
1146. /\* This doesn't take account of any particular  \*/
1147. /\* headers inside the transmitted RTP payload.  \*/
1149. /\* statistics, used for 2-pass encoding \*/
1150. **int** mv\_bits;
1151. **int** header\_bits;
1152. **int** i\_tex\_bits;
1153. **int** p\_tex\_bits;
1154. **int** i\_count;
1155. **int** p\_count;
1156. **int** skip\_count;
1157. **int** misc\_bits;
1159. /\*\*
1160. \* number of bits used for the previously encoded frame
1161. \* - encoding: Set by libavcodec.
1162. \* - decoding: unused
1163. \*/
1164. **int** frame\_bits;
1166. /\*\*
1167. \* pass1 encoding statistics output buffer
1168. \* - encoding: Set by libavcodec.
1169. \* - decoding: unused
1170. \*/
1171. **char** \*stats\_out;
1173. /\*\*
1174. \* pass2 encoding statistics input buffer
1175. \* Concatenated stuff from stats\_out of pass1 should be placed here.
1176. \* - encoding: Allocated/set/freed by user.
1177. \* - decoding: unused
1178. \*/
1179. **char** \*stats\_in;
1181. /\*\*
1182. \* Work around bugs in encoders which sometimes cannot be detected automatically.
1183. \* - encoding: Set by user
1184. \* - decoding: Set by user
1185. \*/
1186. **int** workaround\_bugs;
1187. #define FF\_BUG\_AUTODETECT       1  ///< autodetection
1188. #define FF\_BUG\_OLD\_MSMPEG4      2
1189. #define FF\_BUG\_XVID\_ILACE       4
1190. #define FF\_BUG\_UMP4             8
1191. #define FF\_BUG\_NO\_PADDING       16
1192. #define FF\_BUG\_AMV              32
1193. #define FF\_BUG\_AC\_VLC           0  ///< Will be removed, libavcodec can now handle these non-compliant files by default.
1194. #define FF\_BUG\_QPEL\_CHROMA      64
1195. #define FF\_BUG\_STD\_QPEL         128
1196. #define FF\_BUG\_QPEL\_CHROMA2     256
1197. #define FF\_BUG\_DIRECT\_BLOCKSIZE 512
1198. #define FF\_BUG\_EDGE             1024
1199. #define FF\_BUG\_HPEL\_CHROMA      2048
1200. #define FF\_BUG\_DC\_CLIP          4096
1201. #define FF\_BUG\_MS               8192 ///< Work around various bugs in Microsoft's broken decoders.
1202. #define FF\_BUG\_TRUNCATED       16384
1204. /\*\*
1205. \* strictly follow the standard (MPEG4, ...).
1206. \* - encoding: Set by user.
1207. \* - decoding: Set by user.
1208. \* Setting this to STRICT or higher means the encoder and decoder will
1209. \* generally do stupid things, whereas setting it to unofficial or lower
1210. \* will mean the encoder might produce output that is not supported by all
1211. \* spec-compliant decoders. Decoders don't differentiate between normal,
1212. \* unofficial and experimental (that is, they always try to decode things
1213. \* when they can) unless they are explicitly asked to behave stupidly
1214. \* (=strictly conform to the specs)
1215. \*/
1216. **int** strict\_std\_compliance;
1217. #define FF\_COMPLIANCE\_VERY\_STRICT   2 ///< Strictly conform to an older more strict version of the spec or reference software.
1218. #define FF\_COMPLIANCE\_STRICT        1 ///< Strictly conform to all the things in the spec no matter what consequences.
1219. #define FF\_COMPLIANCE\_NORMAL        0
1220. #define FF\_COMPLIANCE\_UNOFFICIAL   -1 ///< Allow unofficial extensions
1221. #define FF\_COMPLIANCE\_EXPERIMENTAL -2 ///< Allow nonstandardized experimental things.
1223. /\*\*
1224. \* error concealment flags
1225. \* - encoding: unused
1226. \* - decoding: Set by user.
1227. \*/
1228. **int** error\_concealment;
1229. #define FF\_EC\_GUESS\_MVS   1
1230. #define FF\_EC\_DEBLOCK     2
1232. /\*\*
1233. \* debug
1234. \* - encoding: Set by user.
1235. \* - decoding: Set by user.
1236. \*/
1237. **int** debug;
1238. #define FF\_DEBUG\_PICT\_INFO   1
1239. #define FF\_DEBUG\_RC          2
1240. #define FF\_DEBUG\_BITSTREAM   4
1241. #define FF\_DEBUG\_MB\_TYPE     8
1242. #define FF\_DEBUG\_QP          16
1243. #define FF\_DEBUG\_MV          32
1244. #define FF\_DEBUG\_DCT\_COEFF   0x00000040
1245. #define FF\_DEBUG\_SKIP        0x00000080
1246. #define FF\_DEBUG\_STARTCODE   0x00000100
1247. #define FF\_DEBUG\_PTS         0x00000200
1248. #define FF\_DEBUG\_ER          0x00000400
1249. #define FF\_DEBUG\_MMCO        0x00000800
1250. #define FF\_DEBUG\_BUGS        0x00001000
1251. #define FF\_DEBUG\_VIS\_QP      0x00002000
1252. #define FF\_DEBUG\_VIS\_MB\_TYPE 0x00004000
1253. #define FF\_DEBUG\_BUFFERS     0x00008000
1254. #define FF\_DEBUG\_THREADS     0x00010000
1256. /\*\*
1257. \* debug
1258. \* - encoding: Set by user.
1259. \* - decoding: Set by user.
1260. \*/
1261. **int** debug\_mv;
1262. #define FF\_DEBUG\_VIS\_MV\_P\_FOR  0x00000001 //visualize forward predicted MVs of P frames
1263. #define FF\_DEBUG\_VIS\_MV\_B\_FOR  0x00000002 //visualize forward predicted MVs of B frames
1264. #define FF\_DEBUG\_VIS\_MV\_B\_BACK 0x00000004 //visualize backward predicted MVs of B frames
1266. /\*\*
1267. \* Error recognition; may misdetect some more or less valid parts as errors.
1268. \* - encoding: unused
1269. \* - decoding: Set by user.
1270. \*/
1271. **int** err\_recognition;
1272. #define AV\_EF\_CRCCHECK  (1<<0)
1273. #define AV\_EF\_BITSTREAM (1<<1)
1274. #define AV\_EF\_BUFFER    (1<<2)
1275. #define AV\_EF\_EXPLODE   (1<<3)
1277. #define AV\_EF\_CAREFUL    (1<<16)
1278. #define AV\_EF\_COMPLIANT  (1<<17)
1279. #define AV\_EF\_AGGRESSIVE (1<<18)

1282. /\*\*
1283. \* opaque 64bit number (generally a PTS) that will be reordered and
1284. \* output in AVFrame.reordered\_opaque
1285. \* @deprecated in favor of pkt\_pts
1286. \* - encoding: unused
1287. \* - decoding: Set by user.
1288. \*/
1289. int64\_t reordered\_opaque;
1291. /\*\*
1292. \* Hardware accelerator in use
1293. \* - encoding: unused.
1294. \* - decoding: Set by libavcodec
1295. \*/
1296. **struct** AVHWAccel \*hwaccel;
1298. /\*\*
1299. \* Hardware accelerator context.
1300. \* For some hardware accelerators, a global context needs to be
1301. \* provided by the user. In that case, this holds display-dependent
1302. \* data FFmpeg cannot instantiate itself. Please refer to the
1303. \* FFmpeg HW accelerator documentation to know how to fill this
1304. \* is. e.g. for VA API, this is a struct vaapi\_context.
1305. \* - encoding: unused
1306. \* - decoding: Set by user
1307. \*/
1308. **void** \*hwaccel\_context;
1310. /\*\*
1311. \* error
1312. \* - encoding: Set by libavcodec if flags&CODEC\_FLAG\_PSNR.
1313. \* - decoding: unused
1314. \*/
1315. uint64\_t error[AV\_NUM\_DATA\_POINTERS];
1317. /\*\*
1318. \* DCT algorithm, see FF\_DCT\_\* below
1319. \* - encoding: Set by user.
1320. \* - decoding: unused
1321. \*/
1322. **int** dct\_algo;
1323. #define FF\_DCT\_AUTO    0
1324. #define FF\_DCT\_FASTINT 1
1325. #define FF\_DCT\_INT     2
1326. #define FF\_DCT\_MMX     3
1327. #define FF\_DCT\_ALTIVEC 5
1328. #define FF\_DCT\_FAAN    6
1330. /\*\*
1331. \* IDCT algorithm, see FF\_IDCT\_\* below.
1332. \* - encoding: Set by user.
1333. \* - decoding: Set by user.
1334. \*/
1335. **int** idct\_algo;
1336. #define FF\_IDCT\_AUTO          0
1337. #define FF\_IDCT\_INT           1
1338. #define FF\_IDCT\_SIMPLE        2
1339. #define FF\_IDCT\_SIMPLEMMX     3
1340. #define FF\_IDCT\_LIBMPEG2MMX   4
1341. #define FF\_IDCT\_MMI           5
1342. #define FF\_IDCT\_ARM           7
1343. #define FF\_IDCT\_ALTIVEC       8
1344. #define FF\_IDCT\_SH4           9
1345. #define FF\_IDCT\_SIMPLEARM     10
1346. #define FF\_IDCT\_H264          11
1347. #define FF\_IDCT\_VP3           12
1348. #define FF\_IDCT\_IPP           13
1349. #define FF\_IDCT\_XVIDMMX       14
1350. #define FF\_IDCT\_CAVS          15
1351. #define FF\_IDCT\_SIMPLEARMV5TE 16
1352. #define FF\_IDCT\_SIMPLEARMV6   17
1353. #define FF\_IDCT\_SIMPLEVIS     18
1354. #define FF\_IDCT\_WMV2          19
1355. #define FF\_IDCT\_FAAN          20
1356. #define FF\_IDCT\_EA            21
1357. #define FF\_IDCT\_SIMPLENEON    22
1358. #define FF\_IDCT\_SIMPLEALPHA   23
1359. #define FF\_IDCT\_BINK          24
1361. #if FF\_API\_DSP\_MASK
1362. /\*\*
1363. \* Unused.
1364. \* @deprecated use av\_set\_cpu\_flags\_mask() instead.
1365. \*/
1366. attribute\_deprecated unsigned dsp\_mask;
1367. #endif
1369. /\*\*
1370. \* bits per sample/pixel from the demuxer (needed for huffyuv).
1371. \* - encoding: Set by libavcodec.
1372. \* - decoding: Set by user.
1373. \*/
1374. **int** bits\_per\_coded\_sample;
1376. /\*\*
1377. \* Bits per sample/pixel of internal libavcodec pixel/sample format.
1378. \* - encoding: set by user.
1379. \* - decoding: set by libavcodec.
1380. \*/
1381. **int** bits\_per\_raw\_sample;
1383. /\*\*
1384. \* low resolution decoding, 1-> 1/2 size, 2->1/4 size
1385. \* - encoding: unused
1386. \* - decoding: Set by user.
1387. \*/
1388. **int** lowres;
1390. /\*\*
1391. \* the picture in the bitstream
1392. \* - encoding: Set by libavcodec.
1393. \* - decoding: Set by libavcodec.
1394. \*/
1395. AVFrame \*coded\_frame;
1397. /\*\*
1398. \* thread count
1399. \* is used to decide how many independent tasks should be passed to execute()
1400. \* - encoding: Set by user.
1401. \* - decoding: Set by user.
1402. \*/
1403. **int** thread\_count;
1405. /\*\*
1406. \* Which multithreading methods to use.
1407. \* Use of FF\_THREAD\_FRAME will increase decoding delay by one frame per thread,
1408. \* so clients which cannot provide future frames should not use it.
1409. \*
1410. \* - encoding: Set by user, otherwise the default is used.
1411. \* - decoding: Set by user, otherwise the default is used.
1412. \*/
1413. **int** thread\_type;
1414. #define FF\_THREAD\_FRAME   1 ///< Decode more than one frame at once
1415. #define FF\_THREAD\_SLICE   2 ///< Decode more than one part of a single frame at once
1417. /\*\*
1418. \* Which multithreading methods are in use by the codec.
1419. \* - encoding: Set by libavcodec.
1420. \* - decoding: Set by libavcodec.
1421. \*/
1422. **int** active\_thread\_type;
1424. /\*\*
1425. \* Set by the client if its custom get\_buffer() callback can be called
1426. \* synchronously from another thread, which allows faster multithreaded decoding.
1427. \* draw\_horiz\_band() will be called from other threads regardless of this setting.
1428. \* Ignored if the default get\_buffer() is used.
1429. \* - encoding: Set by user.
1430. \* - decoding: Set by user.
1431. \*/
1432. **int** thread\_safe\_callbacks;
1434. /\*\*
1435. \* The codec may call this to execute several independent things.
1436. \* It will return only after finishing all tasks.
1437. \* The user may replace this with some multithreaded implementation,
1438. \* the default implementation will execute the parts serially.
1439. \* @param count the number of things to execute
1440. \* - encoding: Set by libavcodec, user can override.
1441. \* - decoding: Set by libavcodec, user can override.
1442. \*/
1443. **int** (\*execute)(**struct** AVCodecContext \*c, **int** (\*func)(**struct** AVCodecContext \*c2, **void** \*arg), **void** \*arg2, **int** \*ret, **int** count, **int** size);
1445. /\*\*
1446. \* The codec may call this to execute several independent things.
1447. \* It will return only after finishing all tasks.
1448. \* The user may replace this with some multithreaded implementation,
1449. \* the default implementation will execute the parts serially.
1450. \* Also see avcodec\_thread\_init and e.g. the --enable-pthread configure option.
1451. \* @param c context passed also to func
1452. \* @param count the number of things to execute
1453. \* @param arg2 argument passed unchanged to func
1454. \* @param ret return values of executed functions, must have space for "count" values. May be NULL.
1455. \* @param func function that will be called count times, with jobnr from 0 to count-1.
1456. \*             threadnr will be in the range 0 to c->thread\_count-1 < MAX\_THREADS and so that no
1457. \*             two instances of func executing at the same time will have the same threadnr.
1458. \* @return always 0 currently, but code should handle a future improvement where when any call to func
1459. \*         returns < 0 no further calls to func may be done and < 0 is returned.
1460. \* - encoding: Set by libavcodec, user can override.
1461. \* - decoding: Set by libavcodec, user can override.
1462. \*/
1463. **int** (\*execute2)(**struct** AVCodecContext \*c, **int** (\*func)(**struct** AVCodecContext \*c2, **void** \*arg, **int** jobnr, **int** threadnr), **void** \*arg2, **int** \*ret, **int** count);
1465. /\*\*
1466. \* thread opaque
1467. \* Can be used by execute() to store some per AVCodecContext stuff.
1468. \* - encoding: set by execute()
1469. \* - decoding: set by execute()
1470. \*/
1471. **void** \*thread\_opaque;
1473. /\*\*
1474. \* noise vs. sse weight for the nsse comparsion function
1475. \* - encoding: Set by user.
1476. \* - decoding: unused
1477. \*/
1478. **int** nsse\_weight;
1480. /\*\*
1481. \* profile
1482. \* - encoding: Set by user.
1483. \* - decoding: Set by libavcodec.
1484. \*/
1485. **int** profile;
1486. #define FF\_PROFILE\_UNKNOWN -99
1487. #define FF\_PROFILE\_RESERVED -100
1489. #define FF\_PROFILE\_AAC\_MAIN 0
1490. #define FF\_PROFILE\_AAC\_LOW  1
1491. #define FF\_PROFILE\_AAC\_SSR  2
1492. #define FF\_PROFILE\_AAC\_LTP  3
1493. #define FF\_PROFILE\_AAC\_HE   4
1494. #define FF\_PROFILE\_AAC\_HE\_V2 28
1495. #define FF\_PROFILE\_AAC\_LD   22
1496. #define FF\_PROFILE\_AAC\_ELD  38
1498. #define FF\_PROFILE\_DTS         20
1499. #define FF\_PROFILE\_DTS\_ES      30
1500. #define FF\_PROFILE\_DTS\_96\_24   40
1501. #define FF\_PROFILE\_DTS\_HD\_HRA  50
1502. #define FF\_PROFILE\_DTS\_HD\_MA   60
1504. #define FF\_PROFILE\_MPEG2\_422    0
1505. #define FF\_PROFILE\_MPEG2\_HIGH   1
1506. #define FF\_PROFILE\_MPEG2\_SS     2
1507. #define FF\_PROFILE\_MPEG2\_SNR\_SCALABLE  3
1508. #define FF\_PROFILE\_MPEG2\_MAIN   4
1509. #define FF\_PROFILE\_MPEG2\_SIMPLE 5
1511. #define FF\_PROFILE\_H264\_CONSTRAINED  (1<<9)  // 8+1; constraint\_set1\_flag
1512. #define FF\_PROFILE\_H264\_INTRA        (1<<11) // 8+3; constraint\_set3\_flag
1514. #define FF\_PROFILE\_H264\_BASELINE             66
1515. #define FF\_PROFILE\_H264\_CONSTRAINED\_BASELINE (66|FF\_PROFILE\_H264\_CONSTRAINED)
1516. #define FF\_PROFILE\_H264\_MAIN                 77
1517. #define FF\_PROFILE\_H264\_EXTENDED             88
1518. #define FF\_PROFILE\_H264\_HIGH                 100
1519. #define FF\_PROFILE\_H264\_HIGH\_10              110
1520. #define FF\_PROFILE\_H264\_HIGH\_10\_INTRA        (110|FF\_PROFILE\_H264\_INTRA)
1521. #define FF\_PROFILE\_H264\_HIGH\_422             122
1522. #define FF\_PROFILE\_H264\_HIGH\_422\_INTRA       (122|FF\_PROFILE\_H264\_INTRA)
1523. #define FF\_PROFILE\_H264\_HIGH\_444             144
1524. #define FF\_PROFILE\_H264\_HIGH\_444\_PREDICTIVE  244
1525. #define FF\_PROFILE\_H264\_HIGH\_444\_INTRA       (244|FF\_PROFILE\_H264\_INTRA)
1526. #define FF\_PROFILE\_H264\_CAVLC\_444            44
1528. #define FF\_PROFILE\_VC1\_SIMPLE   0
1529. #define FF\_PROFILE\_VC1\_MAIN     1
1530. #define FF\_PROFILE\_VC1\_COMPLEX  2
1531. #define FF\_PROFILE\_VC1\_ADVANCED 3
1533. #define FF\_PROFILE\_MPEG4\_SIMPLE                     0
1534. #define FF\_PROFILE\_MPEG4\_SIMPLE\_SCALABLE            1
1535. #define FF\_PROFILE\_MPEG4\_CORE                       2
1536. #define FF\_PROFILE\_MPEG4\_MAIN                       3
1537. #define FF\_PROFILE\_MPEG4\_N\_BIT                      4
1538. #define FF\_PROFILE\_MPEG4\_SCALABLE\_TEXTURE           5
1539. #define FF\_PROFILE\_MPEG4\_SIMPLE\_FACE\_ANIMATION      6
1540. #define FF\_PROFILE\_MPEG4\_BASIC\_ANIMATED\_TEXTURE     7
1541. #define FF\_PROFILE\_MPEG4\_HYBRID                     8
1542. #define FF\_PROFILE\_MPEG4\_ADVANCED\_REAL\_TIME         9
1543. #define FF\_PROFILE\_MPEG4\_CORE\_SCALABLE             10
1544. #define FF\_PROFILE\_MPEG4\_ADVANCED\_CODING           11
1545. #define FF\_PROFILE\_MPEG4\_ADVANCED\_CORE             12
1546. #define FF\_PROFILE\_MPEG4\_ADVANCED\_SCALABLE\_TEXTURE 13
1547. #define FF\_PROFILE\_MPEG4\_SIMPLE\_STUDIO             14
1548. #define FF\_PROFILE\_MPEG4\_ADVANCED\_SIMPLE           15
1550. /\*\*
1551. \* level
1552. \* - encoding: Set by user.
1553. \* - decoding: Set by libavcodec.
1554. \*/
1555. **int** level;
1556. #define FF\_LEVEL\_UNKNOWN -99
1558. /\*\*
1559. \*
1560. \* - encoding: unused
1561. \* - decoding: Set by user.
1562. \*/
1563. **enum** AVDiscard skip\_loop\_filter;
1565. /\*\*
1566. \*
1567. \* - encoding: unused
1568. \* - decoding: Set by user.
1569. \*/
1570. **enum** AVDiscard skip\_idct;
1572. /\*\*
1573. \*
1574. \* - encoding: unused
1575. \* - decoding: Set by user.
1576. \*/
1577. **enum** AVDiscard skip\_frame;
1579. /\*\*
1580. \* Header containing style information for text subtitles.
1581. \* For SUBTITLE\_ASS subtitle type, it should contain the whole ASS
1582. \* [Script Info] and [V4+ Styles] section, plus the [Events] line and
1583. \* the Format line following. It shouldn't include any Dialogue line.
1584. \* - encoding: Set/allocated/freed by user (before avcodec\_open2())
1585. \* - decoding: Set/allocated/freed by libavcodec (by avcodec\_open2())
1586. \*/
1587. uint8\_t \*subtitle\_header;
1588. **int** subtitle\_header\_size;
1590. /\*\*
1591. \* Simulates errors in the bitstream to test error concealment.
1592. \* - encoding: Set by user.
1593. \* - decoding: unused
1594. \*/
1595. **int** error\_rate;
1597. /\*\*
1598. \* Current packet as passed into the decoder, to avoid having
1599. \* to pass the packet into every function. Currently only valid
1600. \* inside lavc and get/release\_buffer callbacks.
1601. \* - decoding: set by avcodec\_decode\_\*, read by get\_buffer() for setting pkt\_pts
1602. \* - encoding: unused
1603. \*/
1604. AVPacket \*pkt;
1606. /\*\*
1607. \* VBV delay coded in the last frame (in periods of a 27 MHz clock).
1608. \* Used for compliant TS muxing.
1609. \* - encoding: Set by libavcodec.
1610. \* - decoding: unused.
1611. \*/
1612. uint64\_t vbv\_delay;
1614. /\*\*
1615. \* Timebase in which pkt\_dts/pts and AVPacket.dts/pts are.
1616. \* Code outside libavcodec should access this field using:
1617. \* avcodec\_set\_pkt\_timebase(avctx)
1618. \* - encoding unused.
1619. \* - decodimg set by user
1620. \*/
1621. AVRational pkt\_timebase;
1623. /\*\*
1624. \* AVCodecDescriptor
1625. \* Code outside libavcodec should access this field using:
1626. \* avcodec\_get\_codec\_descriptior(avctx)
1627. \* - encoding: unused.
1628. \* - decoding: set by libavcodec.
1629. \*/
1630. **const** AVCodecDescriptor \*codec\_descriptor;
1632. /\*\*
1633. \* Current statistics for PTS correction.
1634. \* - decoding: maintained and used by libavcodec, not intended to be used by user apps
1635. \* - encoding: unused
1636. \*/
1637. int64\_t pts\_correction\_num\_faulty\_pts; /// Number of incorrect PTS values so far
1638. int64\_t pts\_correction\_num\_faulty\_dts; /// Number of incorrect DTS values so far
1639. int64\_t pts\_correction\_last\_pts;       /// PTS of the last frame
1640. int64\_t pts\_correction\_last\_dts;       /// DTS of the last frame
1641. } AVCodecContext;

光定义就真是够多的。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 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）

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。

其实这些参数都比较容易理解。就不多费篇幅了。在这里看一下以下几个参数：

**1.codec\_type**

编解码器类型有以下几种：

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/14214859) [copy](http://blog.csdn.net/leixiaohua1020/article/details/14214859)

1. **enum** AVMediaType {
2. AVMEDIA\_TYPE\_UNKNOWN = -1,  ///< Usually treated as AVMEDIA\_TYPE\_DATA
3. AVMEDIA\_TYPE\_VIDEO,
4. AVMEDIA\_TYPE\_AUDIO,
5. AVMEDIA\_TYPE\_DATA,          ///< Opaque data information usually continuous
6. AVMEDIA\_TYPE\_SUBTITLE,
7. AVMEDIA\_TYPE\_ATTACHMENT,    ///< Opaque data information usually sparse
8. AVMEDIA\_TYPE\_NB
9. };

**2.sample\_fmt**

在FFMPEG中音频采样格式有以下几种：

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/14214859) [copy](http://blog.csdn.net/leixiaohua1020/article/details/14214859)

1. **enum** AVSampleFormat {
2. AV\_SAMPLE\_FMT\_NONE = -1,
3. AV\_SAMPLE\_FMT\_U8,          ///< unsigned 8 bits
4. AV\_SAMPLE\_FMT\_S16,         ///< signed 16 bits
5. AV\_SAMPLE\_FMT\_S32,         ///< signed 32 bits
6. AV\_SAMPLE\_FMT\_FLT,         ///< float
7. AV\_SAMPLE\_FMT\_DBL,         ///< double
9. AV\_SAMPLE\_FMT\_U8P,         ///< unsigned 8 bits, planar
10. AV\_SAMPLE\_FMT\_S16P,        ///< signed 16 bits, planar
11. AV\_SAMPLE\_FMT\_S32P,        ///< signed 32 bits, planar
12. AV\_SAMPLE\_FMT\_FLTP,        ///< float, planar
13. AV\_SAMPLE\_FMT\_DBLP,        ///< double, planar
15. AV\_SAMPLE\_FMT\_NB           ///< Number of sample formats. DO NOT USE if linking dynamically
16. };

**3.profile**

在FFMPEG中型有以下几种，可以看出AAC，MPEG2，H.264，VC-1，MPEG4都有型的概念。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/14214859) [copy](http://blog.csdn.net/leixiaohua1020/article/details/14214859)

1. #define FF\_PROFILE\_UNKNOWN -99
2. #define FF\_PROFILE\_RESERVED -100
4. #define FF\_PROFILE\_AAC\_MAIN 0
5. #define FF\_PROFILE\_AAC\_LOW  1
6. #define FF\_PROFILE\_AAC\_SSR  2
7. #define FF\_PROFILE\_AAC\_LTP  3
8. #define FF\_PROFILE\_AAC\_HE   4
9. #define FF\_PROFILE\_AAC\_HE\_V2 28
10. #define FF\_PROFILE\_AAC\_LD   22
11. #define FF\_PROFILE\_AAC\_ELD  38
13. #define FF\_PROFILE\_DTS         20
14. #define FF\_PROFILE\_DTS\_ES      30
15. #define FF\_PROFILE\_DTS\_96\_24   40
16. #define FF\_PROFILE\_DTS\_HD\_HRA  50
17. #define FF\_PROFILE\_DTS\_HD\_MA   60
19. #define FF\_PROFILE\_MPEG2\_422    0
20. #define FF\_PROFILE\_MPEG2\_HIGH   1
21. #define FF\_PROFILE\_MPEG2\_SS     2
22. #define FF\_PROFILE\_MPEG2\_SNR\_SCALABLE  3
23. #define FF\_PROFILE\_MPEG2\_MAIN   4
24. #define FF\_PROFILE\_MPEG2\_SIMPLE 5
26. #define FF\_PROFILE\_H264\_CONSTRAINED  (1<<9)  // 8+1; constraint\_set1\_flag
27. #define FF\_PROFILE\_H264\_INTRA        (1<<11) // 8+3; constraint\_set3\_flag
29. #define FF\_PROFILE\_H264\_BASELINE             66
30. #define FF\_PROFILE\_H264\_CONSTRAINED\_BASELINE (66|FF\_PROFILE\_H264\_CONSTRAINED)
31. #define FF\_PROFILE\_H264\_MAIN                 77
32. #define FF\_PROFILE\_H264\_EXTENDED             88
33. #define FF\_PROFILE\_H264\_HIGH                 100
34. #define FF\_PROFILE\_H264\_HIGH\_10              110
35. #define FF\_PROFILE\_H264\_HIGH\_10\_INTRA        (110|FF\_PROFILE\_H264\_INTRA)
36. #define FF\_PROFILE\_H264\_HIGH\_422             122
37. #define FF\_PROFILE\_H264\_HIGH\_422\_INTRA       (122|FF\_PROFILE\_H264\_INTRA)
38. #define FF\_PROFILE\_H264\_HIGH\_444             144
39. #define FF\_PROFILE\_H264\_HIGH\_444\_PREDICTIVE  244
40. #define FF\_PROFILE\_H264\_HIGH\_444\_INTRA       (244|FF\_PROFILE\_H264\_INTRA)
41. #define FF\_PROFILE\_H264\_CAVLC\_444            44
43. #define FF\_PROFILE\_VC1\_SIMPLE   0
44. #define FF\_PROFILE\_VC1\_MAIN     1
45. #define FF\_PROFILE\_VC1\_COMPLEX  2
46. #define FF\_PROFILE\_VC1\_ADVANCED 3
48. #define FF\_PROFILE\_MPEG4\_SIMPLE                     0
49. #define FF\_PROFILE\_MPEG4\_SIMPLE\_SCALABLE            1
50. #define FF\_PROFILE\_MPEG4\_CORE                       2
51. #define FF\_PROFILE\_MPEG4\_MAIN                       3
52. #define FF\_PROFILE\_MPEG4\_N\_BIT                      4
53. #define FF\_PROFILE\_MPEG4\_SCALABLE\_TEXTURE           5
54. #define FF\_PROFILE\_MPEG4\_SIMPLE\_FACE\_ANIMATION      6
55. #define FF\_PROFILE\_MPEG4\_BASIC\_ANIMATED\_TEXTURE     7
56. #define FF\_PROFILE\_MPEG4\_HYBRID                     8
57. #define FF\_PROFILE\_MPEG4\_ADVANCED\_REAL\_TIME         9
58. #define FF\_PROFILE\_MPEG4\_CORE\_SCALABLE             10
59. #define FF\_PROFILE\_MPEG4\_ADVANCED\_CODING           11
60. #define FF\_PROFILE\_MPEG4\_ADVANCED\_CORE             12
61. #define FF\_PROFILE\_MPEG4\_ADVANCED\_SCALABLE\_TEXTURE 13
62. #define FF\_PROFILE\_MPEG4\_SIMPLE\_STUDIO             14
63. #define FF\_PROFILE\_MPEG4\_ADVANCED\_SIMPLE           15

# [FFMPEG结构体分析：AVFormatContext](http://blog.csdn.net/leixiaohua1020/article/details/14214705)

FFMPEG有几个最重要的结构体，包含了解协议，解封装，解码操作，此前已经进行过分析：

[FFMPEG中最关键的结构体之间的关系](http://blog.csdn.net/leixiaohua1020/article/details/11693997)

在此不再详述，其中AVFormatContext是包含码流参数较多的结构体。本文将会详细分析一下该结构体里每个变量的含义和作用。

首先看一下结构体的定义（位于avformat.h）：

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/14214705) [copy](http://blog.csdn.net/leixiaohua1020/article/details/14214705)

1. /\* 雷霄骅
2. \* 中国传媒大学/数字电视技术
3. \* leixiaohua1020@126.com
4. \*
5. \*/
6. /\*\*
7. \* Format I/O context.
8. \* New fields can be added to the end with minor version bumps.
9. \* Removal, reordering and changes to existing fields require a major
10. \* version bump.
11. \* sizeof(AVFormatContext) must not be used outside libav\*, use
12. \* avformat\_alloc\_context() to create an AVFormatContext.
13. \*/
14. **typedef** **struct** AVFormatContext {
15. /\*\*
16. \* A class for logging and AVOptions. Set by avformat\_alloc\_context().
17. \* Exports (de)muxer private options if they exist.
18. \*/
19. **const** AVClass \*av\_class;
21. /\*\*
22. \* Can only be iformat or oformat, not both at the same time.
23. \*
24. \* decoding: set by avformat\_open\_input().
25. \* encoding: set by the user.
26. \*/
27. **struct** AVInputFormat \*iformat;
28. **struct** AVOutputFormat \*oformat;
30. /\*\*
31. \* Format private data. This is an AVOptions-enabled struct
32. \* if and only if iformat/oformat.priv\_class is not NULL.
33. \*/
34. **void** \*priv\_data;
36. /\*
37. \* I/O context.
38. \*
39. \* decoding: either set by the user before avformat\_open\_input() (then
40. \* the user must close it manually) or set by avformat\_open\_input().
41. \* encoding: set by the user.
42. \*
43. \* Do NOT set this field if AVFMT\_NOFILE flag is set in
44. \* iformat/oformat.flags. In such a case, the (de)muxer will handle
45. \* I/O in some other way and this field will be NULL.
46. \*/
47. AVIOContext \*pb;
49. /\* stream info \*/
50. **int** ctx\_flags; /\*\*< Format-specific flags, see AVFMTCTX\_xx \*/
52. /\*\*
53. \* A list of all streams in the file. New streams are created with
54. \* avformat\_new\_stream().
55. \*
56. \* decoding: streams are created by libavformat in avformat\_open\_input().
57. \* If AVFMTCTX\_NOHEADER is set in ctx\_flags, then new streams may also
58. \* appear in av\_read\_frame().
59. \* encoding: streams are created by the user before avformat\_write\_header().
60. \*/
61. unsigned **int** nb\_streams;
62. AVStream \*\*streams;
64. **char** filename[1024]; /\*\*< input or output filename \*/
66. /\*\*
67. \* Decoding: position of the first frame of the component, in
68. \* AV\_TIME\_BASE fractional seconds. NEVER set this value directly:
69. \* It is deduced from the AVStream values.
70. \*/
71. int64\_t start\_time;
73. /\*\*
74. \* Decoding: duration of the stream, in AV\_TIME\_BASE fractional
75. \* seconds. Only set this value if you know none of the individual stream
76. \* durations and also do not set any of them. This is deduced from the
77. \* AVStream values if not set.
78. \*/
79. int64\_t duration;
81. /\*\*
82. \* Decoding: total stream bitrate in bit/s, 0 if not
83. \* available. Never set it directly if the file\_size and the
84. \* duration are known as FFmpeg can compute it automatically.
85. \*/
86. **int** bit\_rate;
88. unsigned **int** packet\_size;
89. **int** max\_delay;
91. **int** flags;
92. #define AVFMT\_FLAG\_GENPTS       0x0001 ///< Generate missing pts even if it requires parsing future frames.
93. #define AVFMT\_FLAG\_IGNIDX       0x0002 ///< Ignore index.
94. #define AVFMT\_FLAG\_NONBLOCK     0x0004 ///< Do not block when reading packets from input.
95. #define AVFMT\_FLAG\_IGNDTS       0x0008 ///< Ignore DTS on frames that contain both DTS & PTS
96. #define AVFMT\_FLAG\_NOFILLIN     0x0010 ///< Do not infer any values from other values, just return what is stored in the container
97. #define AVFMT\_FLAG\_NOPARSE      0x0020 ///< Do not use AVParsers, you also must set AVFMT\_FLAG\_NOFILLIN as the fillin code works on frames and no parsing -> no frames. Also seeking to frames can not work if parsing to find frame boundaries has been disabled
98. #define AVFMT\_FLAG\_CUSTOM\_IO    0x0080 ///< The caller has supplied a custom AVIOContext, don't avio\_close() it.
99. #define AVFMT\_FLAG\_DISCARD\_CORRUPT  0x0100 ///< Discard frames marked corrupted
100. #define AVFMT\_FLAG\_MP4A\_LATM    0x8000 ///< Enable RTP MP4A-LATM payload
101. #define AVFMT\_FLAG\_SORT\_DTS    0x10000 ///< try to interleave outputted packets by dts (using this flag can slow demuxing down)
102. #define AVFMT\_FLAG\_PRIV\_OPT    0x20000 ///< Enable use of private options by delaying codec open (this could be made default once all code is converted)
103. #define AVFMT\_FLAG\_KEEP\_SIDE\_DATA 0x40000 ///< Dont merge side data but keep it separate.
105. /\*\*
106. \* decoding: size of data to probe; encoding: unused.
107. \*/
108. unsigned **int** probesize;
110. /\*\*
111. \* decoding: maximum time (in AV\_TIME\_BASE units) during which the input should
112. \* be analyzed in avformat\_find\_stream\_info().
113. \*/
114. **int** max\_analyze\_duration;
116. **const** uint8\_t \*key;
117. **int** keylen;
119. unsigned **int** nb\_programs;
120. AVProgram \*\*programs;
122. /\*\*
123. \* Forced video codec\_id.
124. \* Demuxing: Set by user.
125. \*/
126. **enum** CodecID video\_codec\_id;
128. /\*\*
129. \* Forced audio codec\_id.
130. \* Demuxing: Set by user.
131. \*/
132. **enum** CodecID audio\_codec\_id;
134. /\*\*
135. \* Forced subtitle codec\_id.
136. \* Demuxing: Set by user.
137. \*/
138. **enum** CodecID subtitle\_codec\_id;
140. /\*\*
141. \* Maximum amount of memory in bytes to use for the index of each stream.
142. \* If the index exceeds this size, entries will be discarded as
143. \* needed to maintain a smaller size. This can lead to slower or less
144. \* accurate seeking (depends on demuxer).
145. \* Demuxers for which a full in-memory index is mandatory will ignore
146. \* this.
147. \* muxing  : unused
148. \* demuxing: set by user
149. \*/
150. unsigned **int** max\_index\_size;
152. /\*\*
153. \* Maximum amount of memory in bytes to use for buffering frames
154. \* obtained from realtime capture devices.
155. \*/
156. unsigned **int** max\_picture\_buffer;
158. unsigned **int** nb\_chapters;
159. AVChapter \*\*chapters;
161. AVDictionary \*metadata;
163. /\*\*
164. \* Start time of the stream in real world time, in microseconds
165. \* since the unix epoch (00:00 1st January 1970). That is, pts=0
166. \* in the stream was captured at this real world time.
167. \* - encoding: Set by user.
168. \* - decoding: Unused.
169. \*/
170. int64\_t start\_time\_realtime;
172. /\*\*
173. \* decoding: number of frames used to probe fps
174. \*/
175. **int** fps\_probe\_size;
177. /\*\*
178. \* Error recognition; higher values will detect more errors but may
179. \* misdetect some more or less valid parts as errors.
180. \* - encoding: unused
181. \* - decoding: Set by user.
182. \*/
183. **int** error\_recognition;
185. /\*\*
186. \* Custom interrupt callbacks for the I/O layer.
187. \*
188. \* decoding: set by the user before avformat\_open\_input().
189. \* encoding: set by the user before avformat\_write\_header()
190. \* (mainly useful for AVFMT\_NOFILE formats). The callback
191. \* should also be passed to avio\_open2() if it's used to
192. \* open the file.
193. \*/
194. AVIOInterruptCB interrupt\_callback;
196. /\*\*
197. \* Flags to enable debugging.
198. \*/
199. **int** debug;
200. #define FF\_FDEBUG\_TS        0x0001
202. /\*\*
203. \* Transport stream id.
204. \* This will be moved into demuxer private options. Thus no API/ABI compatibility
205. \*/
206. **int** ts\_id;
208. /\*\*
209. \* Audio preload in microseconds.
210. \* Note, not all formats support this and unpredictable things may happen if it is used when not supported.
211. \* - encoding: Set by user via AVOptions (NO direct access)
212. \* - decoding: unused
213. \*/
214. **int** audio\_preload;
216. /\*\*
217. \* Max chunk time in microseconds.
218. \* Note, not all formats support this and unpredictable things may happen if it is used when not supported.
219. \* - encoding: Set by user via AVOptions (NO direct access)
220. \* - decoding: unused
221. \*/
222. **int** max\_chunk\_duration;
224. /\*\*
225. \* Max chunk size in bytes
226. \* Note, not all formats support this and unpredictable things may happen if it is used when not supported.
227. \* - encoding: Set by user via AVOptions (NO direct access)
228. \* - decoding: unused
229. \*/
230. **int** max\_chunk\_size;
232. /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
233. \* All fields below this line are not part of the public API. They
234. \* may not be used outside of libavformat and can be changed and
235. \* removed at will.
236. \* New public fields should be added right above.
237. \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
238. \*/
240. /\*\*
241. \* This buffer is only needed when packets were already buffered but
242. \* not decoded, for example to get the codec parameters in MPEG
243. \* streams.
244. \*/
245. **struct** AVPacketList \*packet\_buffer;
246. **struct** AVPacketList \*packet\_buffer\_end;
248. /\* av\_seek\_frame() support \*/
249. int64\_t data\_offset; /\*\*< offset of the first packet \*/
251. /\*\*
252. \* Raw packets from the demuxer, prior to parsing and decoding.
253. \* This buffer is used for buffering packets until the codec can
254. \* be identified, as parsing cannot be done without knowing the
255. \* codec.
256. \*/
257. **struct** AVPacketList \*raw\_packet\_buffer;
258. **struct** AVPacketList \*raw\_packet\_buffer\_end;
259. /\*\*
260. \* Packets split by the parser get queued here.
261. \*/
262. **struct** AVPacketList \*parse\_queue;
263. **struct** AVPacketList \*parse\_queue\_end;
264. /\*\*
265. \* Remaining size available for raw\_packet\_buffer, in bytes.
266. \*/
267. #define RAW\_PACKET\_BUFFER\_SIZE 2500000
268. **int** raw\_packet\_buffer\_remaining\_size;
269. } 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**](http://lib.csdn.net/base/datastructure)，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

视频的时长可以转换成HH:MM:SS的形式，示例代码如下：

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/14214705) [copy](http://blog.csdn.net/leixiaohua1020/article/details/14214705)

1. AVFormatContext \*pFormatCtx;
2. CString timelong;
3. ...
4. //duration是以微秒为单位
5. //转换成hh:mm:ss形式
6. **int** tns, thh, tmm, tss;
7. tns  = (pFormatCtx->duration)/1000000;
8. thh  = tns / 3600;
9. tmm  = (tns % 3600) / 60;
10. tss  = (tns % 60);
11. timelong.Format("%02d:%02d:%02d",thh,tmm,tss);

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

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/14214705) [copy](http://blog.csdn.net/leixiaohua1020/article/details/14214705)

1. **typedef** **struct** AVDictionaryEntry {
2. **char** \*key;
3. **char** \*value;
4. } 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。

下列代码显示了获取元数据并存入meta字符串变量的过程，注意每一条key和value之间有一个"\t:"，value之后有一个"\r\n"

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/14214705) [copy](http://blog.csdn.net/leixiaohua1020/article/details/14214705)

1. //MetaData------------------------------------------------------------
2. //从AVDictionary获得
3. //需要用到AVDictionaryEntry对象
4. //CString author,copyright,description;
5. CString meta=NULL,key,value;
6. AVDictionaryEntry \*m = NULL;
7. //不用一个一个找出来
8. /\*  m=av\_dict\_get(pFormatCtx->metadata,"author",m,0);
9. author.Format("作者：%s",m->value);
10. m=av\_dict\_get(pFormatCtx->metadata,"copyright",m,0);
11. copyright.Format("版权：%s",m->value);
12. m=av\_dict\_get(pFormatCtx->metadata,"description",m,0);
13. description.Format("描述：%s",m->value);
14. \*/
15. //使用循环读出
16. //(需要读取的数据，字段名称，前一条字段（循环时使用），参数)
17. **while**(m=av\_dict\_get(pFormatCtx->metadata,"",m,AV\_DICT\_IGNORE\_SUFFIX)){
18. key.Format(m->key);
19. value.Format(m->value);
20. meta+=key+"\t:"+value+"\r\n" ;
21. }

# [FFMPEG结构体分析：AVFrame](http://blog.csdn.net/leixiaohua1020/article/details/14214577)

FFMPEG有几个最重要的结构体，包含了解协议，解封装，解码操作，此前已经进行过分析：

[FFMPEG中最关键的结构体之间的关系](http://blog.csdn.net/leixiaohua1020/article/details/11693997)

在此不再详述，其中AVFrame是包含码流参数较多的结构体。本文将会详细分析一下该结构体里主要变量的含义和作用。

首先看一下结构体的定义（位于avcodec.h）：

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/14214577) [copy](http://blog.csdn.net/leixiaohua1020/article/details/14214577)

1. /\*
2. \*雷霄骅
3. \*leixiaohua1020@126.com
4. \*中国传媒大学/数字电视技术
5. \*/
6. /\*\*
7. \* Audio Video Frame.
8. \* New fields can be added to the end of AVFRAME with minor version
9. \* bumps. Similarly fields that are marked as to be only accessed by
10. \* av\_opt\_ptr() can be reordered. This allows 2 forks to add fields
11. \* without breaking compatibility with each other.
12. \* Removal, reordering and changes in the remaining cases require
13. \* a major version bump.
14. \* sizeof(AVFrame) must not be used outside libavcodec.
15. \*/
16. **typedef** **struct** AVFrame {
17. #define AV\_NUM\_DATA\_POINTERS 8
18. /\*\*图像数据
19. \* pointer to the picture/channel planes.
20. \* This might be different from the first allocated byte
21. \* - encoding: Set by user
22. \* - decoding: set by AVCodecContext.get\_buffer()
23. \*/
24. uint8\_t \*data[AV\_NUM\_DATA\_POINTERS];
26. /\*\*
27. \* Size, in bytes, of the data for each picture/channel plane.
28. \*
29. \* For audio, only linesize[0] may be set. For planar audio, each channel
30. \* plane must be the same size.
31. \*
32. \* - encoding: Set by user
33. \* - decoding: set by AVCodecContext.get\_buffer()
34. \*/
35. **int** linesize[AV\_NUM\_DATA\_POINTERS];
37. /\*\*
38. \* pointers to the data planes/channels.
39. \*
40. \* For video, this should simply point to data[].
41. \*
42. \* For planar audio, each channel has a separate data pointer, and
43. \* linesize[0] contains the size of each channel buffer.
44. \* For packed audio, there is just one data pointer, and linesize[0]
45. \* contains the total size of the buffer for all channels.
46. \*
47. \* Note: Both data and extended\_data will always be set by get\_buffer(),
48. \* but for planar audio with more channels that can fit in data,
49. \* extended\_data must be used by the decoder in order to access all
50. \* channels.
51. \*
52. \* encoding: unused
53. \* decoding: set by AVCodecContext.get\_buffer()
54. \*/
55. uint8\_t \*\*extended\_data;
57. /\*\*宽高
58. \* width and height of the video frame
59. \* - encoding: unused
60. \* - decoding: Read by user.
61. \*/
62. **int** width, height;
64. /\*\*
65. \* number of audio samples (per channel) described by this frame
66. \* - encoding: Set by user
67. \* - decoding: Set by libavcodec
68. \*/
69. **int** nb\_samples;
71. /\*\*
72. \* format of the frame, -1 if unknown or unset
73. \* Values correspond to enum AVPixelFormat for video frames,
74. \* enum AVSampleFormat for audio)
75. \* - encoding: unused
76. \* - decoding: Read by user.
77. \*/
78. **int** format;
80. /\*\*是否是关键帧
81. \* 1 -> keyframe, 0-> not
82. \* - encoding: Set by libavcodec.
83. \* - decoding: Set by libavcodec.
84. \*/
85. **int** key\_frame;
87. /\*\*帧类型（I,B,P）
88. \* Picture type of the frame, see ?\_TYPE below.
89. \* - encoding: Set by libavcodec. for coded\_picture (and set by user for input).
90. \* - decoding: Set by libavcodec.
91. \*/
92. **enum** AVPictureType pict\_type;
94. /\*\*
95. \* pointer to the first allocated byte of the picture. Can be used in get\_buffer/release\_buffer.
96. \* This isn't used by libavcodec unless the default get/release\_buffer() is used.
97. \* - encoding:
98. \* - decoding:
99. \*/
100. uint8\_t \*base[AV\_NUM\_DATA\_POINTERS];
102. /\*\*
103. \* sample aspect ratio for the video frame, 0/1 if unknown/unspecified
104. \* - encoding: unused
105. \* - decoding: Read by user.
106. \*/
107. AVRational sample\_aspect\_ratio;
109. /\*\*
110. \* presentation timestamp in time\_base units (time when frame should be shown to user)
111. \* If AV\_NOPTS\_VALUE then frame\_rate = 1/time\_base will be assumed.
112. \* - encoding: MUST be set by user.
113. \* - decoding: Set by libavcodec.
114. \*/
115. int64\_t pts;
117. /\*\*
118. \* reordered pts from the last AVPacket that has been input into the decoder
119. \* - encoding: unused
120. \* - decoding: Read by user.
121. \*/
122. int64\_t pkt\_pts;
124. /\*\*
125. \* dts from the last AVPacket that has been input into the decoder
126. \* - encoding: unused
127. \* - decoding: Read by user.
128. \*/
129. int64\_t pkt\_dts;
131. /\*\*
132. \* picture number in bitstream order
133. \* - encoding: set by
134. \* - decoding: Set by libavcodec.
135. \*/
136. **int** coded\_picture\_number;
137. /\*\*
138. \* picture number in display order
139. \* - encoding: set by
140. \* - decoding: Set by libavcodec.
141. \*/
142. **int** display\_picture\_number;
144. /\*\*
145. \* quality (between 1 (good) and FF\_LAMBDA\_MAX (bad))
146. \* - encoding: Set by libavcodec. for coded\_picture (and set by user for input).
147. \* - decoding: Set by libavcodec.
148. \*/
149. **int** quality;
151. /\*\*
152. \* is this picture used as reference
153. \* The values for this are the same as the MpegEncContext.picture\_structure
154. \* variable, that is 1->top field, 2->bottom field, 3->frame/both fields.
155. \* Set to 4 for delayed, non-reference frames.
156. \* - encoding: unused
157. \* - decoding: Set by libavcodec. (before get\_buffer() call)).
158. \*/
159. **int** reference;
161. /\*\*QP表
162. \* QP table
163. \* - encoding: unused
164. \* - decoding: Set by libavcodec.
165. \*/
166. int8\_t \*qscale\_table;
167. /\*\*
168. \* QP store stride
169. \* - encoding: unused
170. \* - decoding: Set by libavcodec.
171. \*/
172. **int** qstride;
174. /\*\*
175. \*
176. \*/
177. **int** qscale\_type;
179. /\*\*跳过宏块表
180. \* mbskip\_table[mb]>=1 if MB didn't change
181. \* stride= mb\_width = (width+15)>>4
182. \* - encoding: unused
183. \* - decoding: Set by libavcodec.
184. \*/
185. uint8\_t \*mbskip\_table;
187. /\*\*运动矢量表
188. \* motion vector table
189. \* @code
190. \* example:
191. \* int mv\_sample\_log2= 4 - motion\_subsample\_log2;
192. \* int mb\_width= (width+15)>>4;
193. \* int mv\_stride= (mb\_width << mv\_sample\_log2) + 1;
194. \* motion\_val[direction][x + y\*mv\_stride][0->mv\_x, 1->mv\_y];
195. \* @endcode
196. \* - encoding: Set by user.
197. \* - decoding: Set by libavcodec.
198. \*/
199. int16\_t (\*motion\_val[2])[2];
201. /\*\*宏块类型表
202. \* macroblock type table
203. \* mb\_type\_base + mb\_width + 2
204. \* - encoding: Set by user.
205. \* - decoding: Set by libavcodec.
206. \*/
207. uint32\_t \*mb\_type;
209. /\*\*DCT系数
210. \* DCT coefficients
211. \* - encoding: unused
212. \* - decoding: Set by libavcodec.
213. \*/
214. **short** \*dct\_coeff;
216. /\*\*参考帧列表
217. \* motion reference frame index
218. \* the order in which these are stored can depend on the codec.
219. \* - encoding: Set by user.
220. \* - decoding: Set by libavcodec.
221. \*/
222. int8\_t \*ref\_index[2];
224. /\*\*
225. \* for some private data of the user
226. \* - encoding: unused
227. \* - decoding: Set by user.
228. \*/
229. **void** \*opaque;
231. /\*\*
232. \* error
233. \* - encoding: Set by libavcodec. if flags&CODEC\_FLAG\_PSNR.
234. \* - decoding: unused
235. \*/
236. uint64\_t error[AV\_NUM\_DATA\_POINTERS];
238. /\*\*
239. \* type of the buffer (to keep track of who has to deallocate data[\*])
240. \* - encoding: Set by the one who allocates it.
241. \* - decoding: Set by the one who allocates it.
242. \* Note: User allocated (direct rendering) & internal buffers cannot coexist currently.
243. \*/
244. **int** type;
246. /\*\*
247. \* When decoding, this signals how much the picture must be delayed.
248. \* extra\_delay = repeat\_pict / (2\*fps)
249. \* - encoding: unused
250. \* - decoding: Set by libavcodec.
251. \*/
252. **int** repeat\_pict;
254. /\*\*
255. \* The content of the picture is interlaced.
256. \* - encoding: Set by user.
257. \* - decoding: Set by libavcodec. (default 0)
258. \*/
259. **int** interlaced\_frame;
261. /\*\*
262. \* If the content is interlaced, is top field displayed first.
263. \* - encoding: Set by user.
264. \* - decoding: Set by libavcodec.
265. \*/
266. **int** top\_field\_first;
268. /\*\*
269. \* Tell user application that palette has changed from previous frame.
270. \* - encoding: ??? (no palette-enabled encoder yet)
271. \* - decoding: Set by libavcodec. (default 0).
272. \*/
273. **int** palette\_has\_changed;
275. /\*\*
276. \* codec suggestion on buffer type if != 0
277. \* - encoding: unused
278. \* - decoding: Set by libavcodec. (before get\_buffer() call)).
279. \*/
280. **int** buffer\_hints;
282. /\*\*
283. \* Pan scan.
284. \* - encoding: Set by user.
285. \* - decoding: Set by libavcodec.
286. \*/
287. AVPanScan \*pan\_scan;
289. /\*\*
290. \* reordered opaque 64bit (generally an integer or a double precision float
291. \* PTS but can be anything).
292. \* The user sets AVCodecContext.reordered\_opaque to represent the input at
293. \* that time,
294. \* the decoder reorders values as needed and sets AVFrame.reordered\_opaque
295. \* to exactly one of the values provided by the user through AVCodecContext.reordered\_opaque
296. \* @deprecated in favor of pkt\_pts
297. \* - encoding: unused
298. \* - decoding: Read by user.
299. \*/
300. int64\_t reordered\_opaque;
302. /\*\*
303. \* hardware accelerator private data (FFmpeg-allocated)
304. \* - encoding: unused
305. \* - decoding: Set by libavcodec
306. \*/
307. **void** \*hwaccel\_picture\_private;
309. /\*\*
310. \* the AVCodecContext which ff\_thread\_get\_buffer() was last called on
311. \* - encoding: Set by libavcodec.
312. \* - decoding: Set by libavcodec.
313. \*/
314. **struct** AVCodecContext \*owner;
316. /\*\*
317. \* used by multithreading to store frame-specific info
318. \* - encoding: Set by libavcodec.
319. \* - decoding: Set by libavcodec.
320. \*/
321. **void** \*thread\_opaque;
323. /\*\*
324. \* log2 of the size of the block which a single vector in motion\_val represents:
325. \* (4->16x16, 3->8x8, 2-> 4x4, 1-> 2x2)
326. \* - encoding: unused
327. \* - decoding: Set by libavcodec.
328. \*/
329. uint8\_t motion\_subsample\_log2;
331. /\*\*（音频）采样率
332. \* Sample rate of the audio data.
333. \*
334. \* - encoding: unused
335. \* - decoding: read by user
336. \*/
337. **int** sample\_rate;
339. /\*\*
340. \* Channel layout of the audio data.
341. \*
342. \* - encoding: unused
343. \* - decoding: read by user.
344. \*/
345. uint64\_t channel\_layout;
347. /\*\*
348. \* frame timestamp estimated using various heuristics, in stream time base
349. \* Code outside libavcodec should access this field using:
350. \* av\_frame\_get\_best\_effort\_timestamp(frame)
351. \* - encoding: unused
352. \* - decoding: set by libavcodec, read by user.
353. \*/
354. int64\_t best\_effort\_timestamp;
356. /\*\*
357. \* reordered pos from the last AVPacket that has been input into the decoder
358. \* Code outside libavcodec should access this field using:
359. \* av\_frame\_get\_pkt\_pos(frame)
360. \* - encoding: unused
361. \* - decoding: Read by user.
362. \*/
363. int64\_t pkt\_pos;
365. /\*\*
366. \* duration of the corresponding packet, expressed in
367. \* AVStream->time\_base units, 0 if unknown.
368. \* Code outside libavcodec should access this field using:
369. \* av\_frame\_get\_pkt\_duration(frame)
370. \* - encoding: unused
371. \* - decoding: Read by user.
372. \*/
373. int64\_t pkt\_duration;
375. /\*\*
376. \* metadata.
377. \* Code outside libavcodec should access this field using:
378. \* av\_frame\_get\_metadata(frame)
379. \* - encoding: Set by user.
380. \* - decoding: Set by libavcodec.
381. \*/
382. AVDictionary \*metadata;
384. /\*\*
385. \* decode error flags of the frame, set to a combination of
386. \* FF\_DECODE\_ERROR\_xxx flags if the decoder produced a frame, but there
387. \* were errors during the decoding.
388. \* Code outside libavcodec should access this field using:
389. \* av\_frame\_get\_decode\_error\_flags(frame)
390. \* - encoding: unused
391. \* - decoding: set by libavcodec, read by user.
392. \*/
393. **int** decode\_error\_flags;
394. #define FF\_DECODE\_ERROR\_INVALID\_BITSTREAM   1
395. #define FF\_DECODE\_ERROR\_MISSING\_REFERENCE   2
397. /\*\*
398. \* number of audio channels, only used for audio.
399. \* Code outside libavcodec should access this field using:
400. \* av\_frame\_get\_channels(frame)
401. \* - encoding: unused
402. \* - decoding: Read by user.
403. \*/
404. int64\_t channels;
405. } 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。

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）

具体参见：[FFMPEG 实现 YUV，RGB各种图像原始数据之间的转换（swscale）](http://blog.csdn.net/leixiaohua1020/article/details/14215391)

**2.pict\_type**

包含以下类型：

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/14214577) [copy](http://blog.csdn.net/leixiaohua1020/article/details/14214577)

1. **enum** AVPictureType {
2. AV\_PICTURE\_TYPE\_NONE = 0, ///< Undefined
3. AV\_PICTURE\_TYPE\_I,     ///< Intra
4. AV\_PICTURE\_TYPE\_P,     ///< Predicted
5. AV\_PICTURE\_TYPE\_B,     ///< Bi-dir predicted
6. AV\_PICTURE\_TYPE\_S,     ///< S(GMC)-VOP MPEG4
7. AV\_PICTURE\_TYPE\_SI,    ///< Switching Intra
8. AV\_PICTURE\_TYPE\_SP,    ///< Switching Predicted
9. AV\_PICTURE\_TYPE\_BI,    ///< BI type
10. };

**3.sample\_aspect\_ratio**

宽高比是一个分数，FFMPEG中用AVRational表达分数：

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/14214577) [copy](http://blog.csdn.net/leixiaohua1020/article/details/14214577)

1. /\*\*
2. \* rational number numerator/denominator
3. \*/
4. **typedef** **struct** AVRational{
5. **int** num; ///< numerator
6. **int** den; ///< denominator
7. } AVRational;

**4.qscale\_table**

QP表指向一块内存，里面存储的是每个宏块的QP值。宏块的标号是从左往右，一行一行的来的。每个宏块对应1个QP。

qscale\_table[0]就是第1行第1列宏块的QP值；qscale\_table[1]就是第1行第2列宏块的QP值；qscale\_table[2]就是第1行第3列宏块的QP值。以此类推...

宏块的个数用下式计算：

注：宏块大小是16x16的。

每行宏块数：

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/14214577) [copy](http://blog.csdn.net/leixiaohua1020/article/details/14214577)

1. **int** mb\_stride = pCodecCtx->width/16+1

宏块的总数：

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/14214577) [copy](http://blog.csdn.net/leixiaohua1020/article/details/14214577)

1. **int** mb\_sum = ((pCodecCtx->height+15)>>4)\*(pCodecCtx->width/16+1)

**5.motion\_subsample\_log2**

1个运动矢量所能代表的画面大小（用宽或者高表示，单位是像素），注意，这里取了log2。

代码注释中给出以下数据：

4->16x16, 3->8x8, 2-> 4x4, 1-> 2x2

即1个运动矢量代表16x16的画面的时候，该值取4；1个运动矢量代表8x8的画面的时候，该值取3...以此类推

**6.motion\_val**

运动矢量表存储了一帧视频中的所有运动矢量。

该值的存储方式比较特别：

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/14214577) [copy](http://blog.csdn.net/leixiaohua1020/article/details/14214577)

1. int16\_t (\*motion\_val[2])[2];

为了弄清楚该值究竟是怎么存的，花了我好一阵子功夫...

注释中给了一段代码：

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/14214577) [copy](http://blog.csdn.net/leixiaohua1020/article/details/14214577)

1. **int** mv\_sample\_log2= 4 - motion\_subsample\_log2;
2. **int** mb\_width= (width+15)>>4;
3. **int** mv\_stride= (mb\_width << mv\_sample\_log2) + 1;
4. motion\_val[direction][x + y\*mv\_stride][0->mv\_x, 1->mv\_y];

大概知道了该数据的结构：

1.首先分为两个列表L0和L1

2.每个列表（L0或L1）存储了一系列的MV（每个MV对应一个画面，大小由**motion\_subsample\_log2**决定）

3.每个MV分为横坐标和纵坐标（x,y）

注意，在FFMPEG中MV和MB在存储的结构上是没有什么关联的，第1个MV是屏幕上左上角画面的MV（画面的大小取决于**motion\_subsample\_log2**），第2个MV是屏幕上第1行第2列的画面的MV，以此类推。因此在一个宏块（16x16）的运动矢量很有可能如下图所示（line代表一行运动矢量的个数）：

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/14214577) [copy](http://blog.csdn.net/leixiaohua1020/article/details/14214577)

1. //例如8x8划分的运动矢量与宏块的关系：
2. //-------------------------
3. //|          |            |
4. //|mv[x]     |mv[x+1]     |
5. //-------------------------
6. //|          |            |
7. //|mv[x+line]|mv[x+line+1]|
8. //-------------------------

**7.mb\_type**

宏块类型表存储了一帧视频中的所有宏块的类型。其存储方式和QP表差不多。只不过其是uint32类型的，而QP表是uint8类型的。每个宏块对应一个宏块类型变量。

宏块类型如下定义所示：

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/14214577) [copy](http://blog.csdn.net/leixiaohua1020/article/details/14214577)

1. //The following defines may change, don't expect compatibility if you use them.
2. #define MB\_TYPE\_INTRA4x4   0x0001
3. #define MB\_TYPE\_INTRA16x16 0x0002 //FIXME H.264-specific
4. #define MB\_TYPE\_INTRA\_PCM  0x0004 //FIXME H.264-specific
5. #define MB\_TYPE\_16x16      0x0008
6. #define MB\_TYPE\_16x8       0x0010
7. #define MB\_TYPE\_8x16       0x0020
8. #define MB\_TYPE\_8x8        0x0040
9. #define MB\_TYPE\_INTERLACED 0x0080
10. #define MB\_TYPE\_DIRECT2    0x0100 //FIXME
11. #define MB\_TYPE\_ACPRED     0x0200
12. #define MB\_TYPE\_GMC        0x0400
13. #define MB\_TYPE\_SKIP       0x0800
14. #define MB\_TYPE\_P0L0       0x1000
15. #define MB\_TYPE\_P1L0       0x2000
16. #define MB\_TYPE\_P0L1       0x4000
17. #define MB\_TYPE\_P1L1       0x8000
18. #define MB\_TYPE\_L0         (MB\_TYPE\_P0L0 | MB\_TYPE\_P1L0)
19. #define MB\_TYPE\_L1         (MB\_TYPE\_P0L1 | MB\_TYPE\_P1L1)
20. #define MB\_TYPE\_L0L1       (MB\_TYPE\_L0   | MB\_TYPE\_L1)
21. #define MB\_TYPE\_QUANT      0x00010000
22. #define MB\_TYPE\_CBP        0x00020000
23. //Note bits 24-31 are reserved for codec specific use (h264 ref0, mpeg1 0mv, ...)

一个宏块如果包含上述定义中的一种或两种类型，则其对应的宏块变量的对应位会被置1。  
注：一个宏块可以包含好几种类型，但是有些类型是不能重复包含的，比如说一个宏块不可能既是16x16又是8x8。

**8.ref\_index**

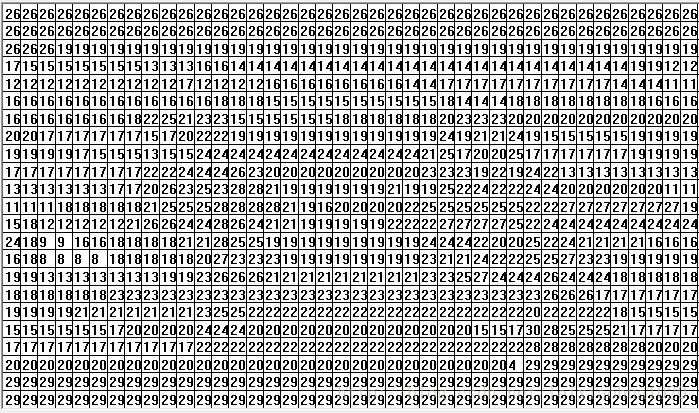
运动估计参考帧列表存储了一帧视频中所有宏块的参考帧索引。这个列表其实在比较早的压缩编码标准中是没有什么用的。只有像H.264这样的编码标准才有多参考帧的概念。但是这个字段目前我还没有研究透。只是知道每个宏块包含有4个该值，该值反映的是参考帧的索引。以后有机会再进行细研究吧。

**在这里展示一下自己做的码流分析软件的运行结果。将上文介绍的几个列表图像化显示了出来（在这里是使用MFC的绘图函数画出来的）**

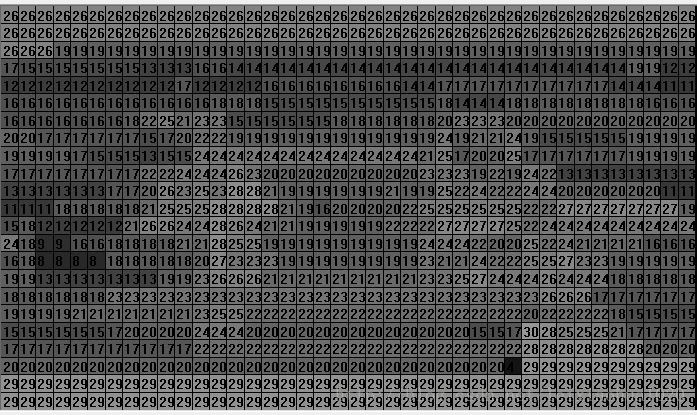
视频帧：



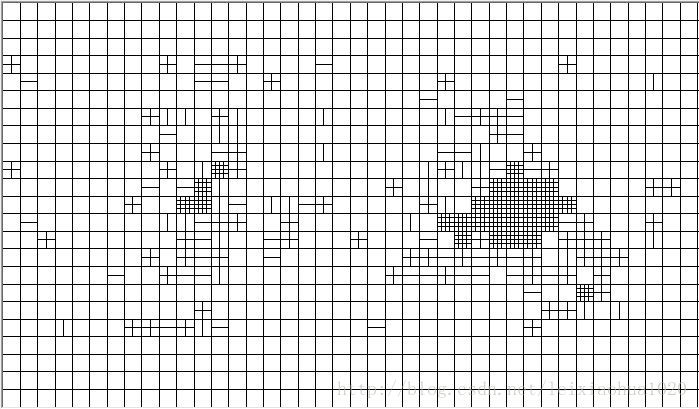
QP参数提取的结果：



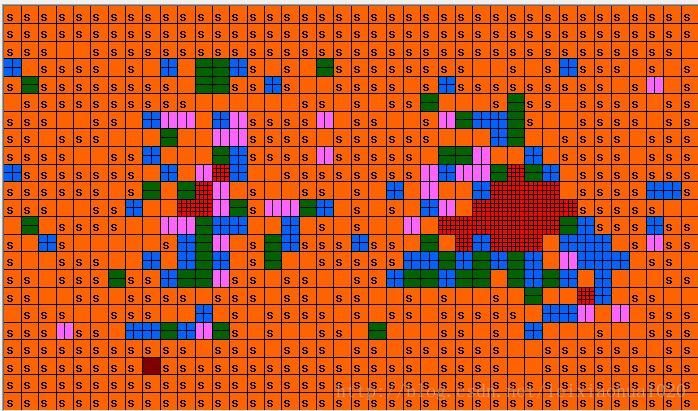
美化过的（加上了颜色）：



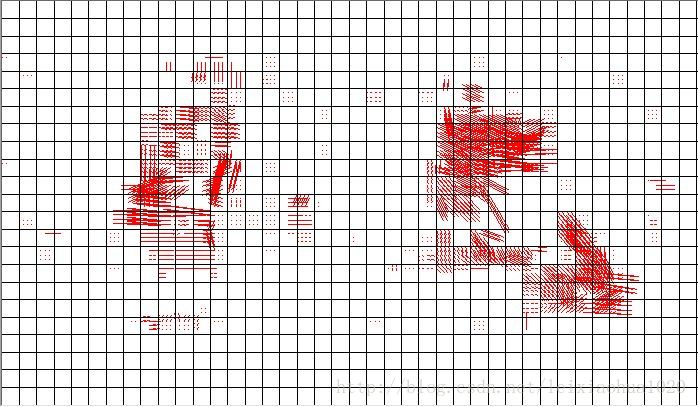
宏块类型参数提取的结果：



美化过的（加上了颜色，更清晰一些，s代表skip宏块）：



运动矢量参数提取的结果（在这里是List0）：



运动估计参考帧参数提取的结果：

