转 ffmpeg函数介绍

2013年09月24日 16:01:44 阅读数：31506

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, const char *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

// const char *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( const char *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( const char *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( const char *short_name,
const char *filename,
const char *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,
const char *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
转自： http://blog.chinaunix.net/uid/20718335/frmd/153034.html

文章标签： ffmpeg 函数 介绍
个人分类： FFMPEG
所属专栏： FFmpeg