

## 原 FFMpeg源代码简单分析：常见结构体的初始化和销毁（AVFormatContext，AVFrame等）

2015年03月03日 16:25:13 阅读数：41683

=====

FFmpeg的库函数源代码分析文章列表：

### 【架构图】

[FFmpeg 源代码结构图 - 解码](#)

[FFmpeg 源代码结构图 - 编码](#)

### 【通用】

[FFmpeg 源代码简单分析：av\\_register\\_all\(\)](#)

[FFmpeg 源代码简单分析：avcodec\\_register\\_all\(\)](#)

[FFmpeg 源代码简单分析：内存的分配和释放（av\\_malloc\(\)、av\\_free\(\)等）](#)

[FFmpeg 源代码简单分析：常见结构体的初始化和销毁（AVFormatContext，AVFrame等）](#)

[FFmpeg 源代码简单分析：avio\\_open2\(\)](#)

[FFmpeg 源代码简单分析：av\\_find\\_decoder\(\)和av\\_find\\_encoder\(\)](#)

[FFmpeg 源代码简单分析：avcodec\\_open2\(\)](#)

[FFmpeg 源代码简单分析：avcodec\\_close\(\)](#)

### 【解码】

[图解 FFMPEG 打开媒体的函数 avformat\\_open\\_input](#)

[FFmpeg 源代码简单分析：avformat\\_open\\_input\(\)](#)

[FFmpeg 源代码简单分析：avformat\\_find\\_stream\\_info\(\)](#)

[FFmpeg 源代码简单分析：av\\_read\\_frame\(\)](#)

[FFmpeg 源代码简单分析：avcodec\\_decode\\_video2\(\)](#)

[FFmpeg 源代码简单分析：avformat\\_close\\_input\(\)](#)

### 【编码】

[FFmpeg 源代码简单分析：avformat\\_alloc\\_output\\_context2\(\)](#)

[FFmpeg 源代码简单分析：avformat\\_write\\_header\(\)](#)

[FFmpeg 源代码简单分析：avcodec\\_encode\\_video\(\)](#)

[FFmpeg 源代码简单分析：av\\_write\\_frame\(\)](#)

[FFmpeg 源代码简单分析：av\\_write\\_trailer\(\)](#)

### 【其它】

[FFmpeg 源代码简单分析：日志输出系统（av\\_log\(\)等）](#)

[FFmpeg 源代码简单分析：结构体成员管理系统 -AVClass](#)

[FFmpeg 源代码简单分析：结构体成员管理系统 -AVOption](#)

[FFmpeg 源代码简单分析：libswscale 的 sws\\_getContext\(\)](#)

[FFmpeg 源代码简单分析：libswscale 的 sws\\_scale\(\)](#)

[FFmpeg 源代码简单分析：libavdevice 的 avdevice\\_register\\_all\(\)](#)

[FFmpeg 源代码简单分析：libavdevice 的 gdigrab](#)

### 【脚本】

FFmpeg 源代码简单分析：makefile

FFmpeg 源代码简单分析：configure

【H.264】

FFmpeg 的 H.264 解码器源代码简单分析：概述

本文简单分析FFmpeg常见结构体的初始化和销毁函数的源代码。常见的结构体在文章：

《FFMPEG中最关键的结构体之间的关系》中已经有过叙述，包括：

AVFormatContext：统领全局的基本结构体。主要用于处理封装格式（FLV/MKV/RMVB等）。

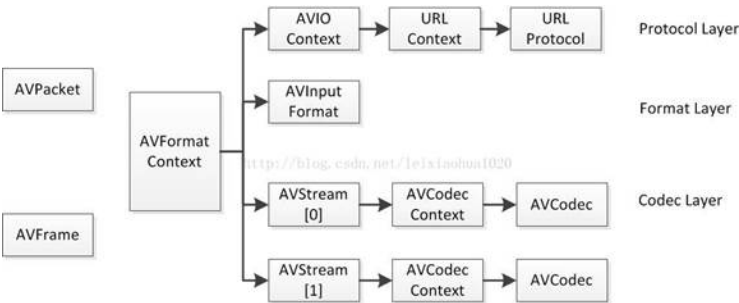
AVIOContext：输入输出对应的结构体，用于输入输出（读写文件，RTMP协议等）。

AVStream，AVCodecContext：视音频流对应的结构体，用于视音频编解码。

AVFrame：存储非压缩的数据（视频对应RGB/YUV像素数据，音频对应PCM采样数据）

AVPacket：存储压缩数据（视频对应H.264等码流数据，音频对应AAC/MP3等码流数据）

他们之间的关系如下图所示（详细信息可以参考上文提到的文章）。



下文简单分析一下上述几个结构体的初始化和销毁函数。这些函数列表如下。

结构体	初始化	销毁
AVFormatContext	avformat_alloc_context()	avformat_free_context()
AVIOContext	avio_alloc_context()	
AVStream	avformat_new_stream()	
AVCodecContext	avcodec_alloc_context3()	
AVFrame	av_frame_alloc(); av_image_fill_arrays()	av_frame_free()
AVPacket	av_init_packet(); av_new_packet()	av_free_packet()

下面进入正文。

## AVFormatContext

AVFormatContext的初始化函数是avformat\_alloc\_context()，销毁函数是avformat\_free\_context()。

### avformat\_alloc\_context()

avformat\_alloc\_context()的声明位于libavformat\avformat.h，如下所示。

```

1.  /**
2.   * Allocate an AVFormatContext.
3.   * avformat_free_context() can be used to free the context and everything
4.   * allocated by the framework within it.
5.   */
6.  AVFormatContext* avformat_alloc_context(void);

```

avformat\_alloc\_context()的定义位于libavformat/options.c。代码如下所示。

```

1.  AVFormatContext *avformat_alloc_context(void)
2.  {
3.      AVFormatContext *ic;
4.      ic = av_malloc(sizeof(AVFormatContext));
5.      if (!ic) return ic;
6.      avformat_get_context_defaults(ic);
7.
8.
9.      ic->internal = av_mallocz(sizeof(*ic->internal));
10.     if (!ic->internal) {
11.         avformat_free_context(ic);
12.         return NULL;
13.     }
14.
15.
16.     return ic;
17. }

```

从代码中可以看出，avformat\_alloc\_context()调用av\_malloc()为AVFormatContext结构体分配了内存，而且同时也给AVFormatContext中的internal字段分配内存（这个字段是FFmpeg内部使用的，先不分析）。此外调用了avformat\_get\_context\_defaults()函数。该函数用于设置AVFormatContext的字段默认值。它的定义也位于libavformat/options.c，确切的说就位于avformat\_alloc\_context()上面。我们看一下该函数的定义。

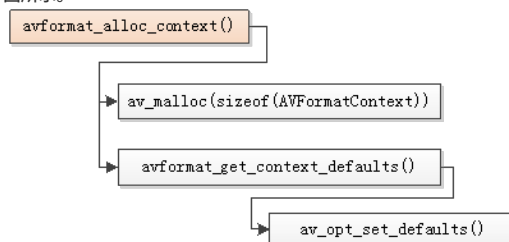
```

1.  static void avformat_get_context_defaults(AVFormatContext *s)
2.  {
3.      memset(s, 0, sizeof(AVFormatContext));
4.
5.
6.      s->av_class = &av_format_context_class;
7.
8.
9.      av_opt_set_defaults(s);
10. }

```

从代码可以看出，avformat\_get\_context\_defaults()首先调用memset()将AVFormatContext的所有字段置0。而后调用了函数av\_opt\_set\_defaults()。av\_opt\_set\_defaults()用于给字段设置默认值。

avformat\_alloc\_context()代码的函数调用关系如下图所示。



雷霄骅 (Lei Xiaohua)  
 Email: leixiaohua1020@126.com  
 Website: <http://blog.csdn.net/leixiaohua1020>

## avformat\_free\_context()

avformat\_free\_context()的声明位于libavformat/avformat.h，如下所示。

```

1.  /**
2.   * Free an AVFormatContext and all its streams.
3.   * @param s context to free
4.   */
5.  void avformat_free_context(AVFormatContext *s);

```

avformat\_free\_context()的定义位于libavformat/options.c。代码如下所示。

```
[cpp]
1. void avformat_free_context(AVFormatContext *s)
2. {
3.     int i;
4.
5.
6.     if (!s)
7.         return;
8.
9.
10.    av_opt_free(s);
11.    if (s->iformat && s->iformat->priv_class && s->priv_data)
12.        av_opt_free(s->priv_data);
13.    if (s->oformat && s->oformat->priv_class && s->priv_data)
14.        av_opt_free(s->priv_data);
15.
16.
17.    for (i = s->nb_streams - 1; i >= 0; i--) {
18.        ff_free_stream(s, s->streams[i]);
19.    }
20.    for (i = s->nb_programs - 1; i >= 0; i--) {
21.        av_dict_free(&s->programs[i]->metadata);
22.        av_freep(&s->programs[i]->stream_index);
23.        av_freep(&s->programs[i]);
24.    }
25.    av_freep(&s->programs);
26.    av_freep(&s->priv_data);
27.    while (s->nb_chapters--) {
28.        av_dict_free(&s->chapters[s->nb_chapters]->metadata);
29.        av_freep(&s->chapters[s->nb_chapters]);
30.    }
31.    av_freep(&s->chapters);
32.    av_dict_free(&s->metadata);
33.    av_freep(&s->streams);
34.    av_freep(&s->internal);
35.    flush_packet_queue(s);
36.    av_free(s);
37. }
```

从代码中可以看出，avformat\_free\_context()调用了各式各样的销毁函数：av\_opt\_free(), av\_freep(), av\_dict\_free()。这些函数分别用于释放不同种类的变量，在这里不再详细讨论。在这里看一个释放AVStream的函数ff\_free\_stream()。该函数的定义位于libavformat/options.c（其实就在avformat\_free\_context()上方）。

```
[cpp]
1. void ff_free_stream(AVFormatContext *s, AVStream *st) {
2.     int j;
3.     av_assert0(s->nb_streams>0);
4.     av_assert0(s->streams[ s->nb_streams - 1 ] == st);
5.
6.
7.     for (j = 0; j < st->nb_side_data; j++)
8.         av_freep(&st->side_data[j].data);
9.     av_freep(&st->side_data);
10.    st->nb_side_data = 0;
11.
12.
13.    if (st->parser) {
14.        av_parser_close(st->parser);
15.    }
16.    if (st->attached_pic.data)
17.        av_free_packet(&st->attached_pic);
18.    av_dict_free(&st->metadata);
19.    av_freep(&st->probe_data.buf);
20.    av_freep(&st->index_entries);
21.    av_freep(&st->codec->extradata);
22.    av_freep(&st->codec->subtitle_header);
23.    av_freep(&st->codec);
24.    av_freep(&st->priv_data);
25.    if (st->info)
26.        av_freep(&st->info->duration_error);
27.    av_freep(&st->info);
28.    av_freep(&s->streams[ --s->nb_streams ]);
29. }
```

从代码中可以看出，与释放AVFormatContext类似，释放AVStream的时候，也是调用了av\_freep(), av\_dict\_free()这些函数释放有关的字段。如果使用了parser的话，会调用av\_parser\_close()关闭该parser。

## AVIOContext

### avio\_alloc\_context()

AVIOContext的初始化函数是avio\_alloc\_context()，销毁的时候使用av\_free()释放掉其中的缓存即可。它的声明位于libavformat/avio.h中，如下所示。

```

1.  /**
2.   * Allocate and initialize an AVIOContext for buffered I/O. It must be later
3.   * freed with av_free().
4.   *
5.   * @param buffer Memory block for input/output operations via AVIOContext.
6.   *       The buffer must be allocated with av_malloc() and friends.
7.   * @param buffer_size The buffer size is very important for performance.
8.   *       For protocols with fixed blocksize it should be set to this blocksize.
9.   *       For others a typical size is a cache page, e.g. 4kb.
10.  * @param write_flag Set to 1 if the buffer should be writable, 0 otherwise.
11.  * @param opaque An opaque pointer to user-specific data.
12.  * @param read_packet A function for refilling the buffer, may be NULL.
13.  * @param write_packet A function for writing the buffer contents, may be NULL.
14.  *       The function may not change the input buffers content.
15.  * @param seek A function for seeking to specified byte position, may be NULL.
16.  *
17.  * @return Allocated AVIOContext or NULL on failure.
18.  */
19. AVIOContext *avio_alloc_context(
20.     unsigned char *buffer,
21.     int buffer_size,
22.     int write_flag,
23.     void *opaque,
24.     int (*read_packet)(void *opaque, uint8_t *buf, int buf_size),
25.     int (*write_packet)(void *opaque, uint8_t *buf, int buf_size),
26.     int64_t (*seek)(void *opaque, int64_t offset, int whence));

```

avio\_alloc\_context()定义位于libavformat\aviobuf.c中，如下所示。

```

1.  AVIOContext *avio_alloc_context(
2.     unsigned char *buffer,
3.     int buffer_size,
4.     int write_flag,
5.     void *opaque,
6.     int (*read_packet)(void *opaque, uint8_t *buf, int buf_size),
7.     int (*write_packet)(void *opaque, uint8_t *buf, int buf_size),
8.     int64_t (*seek)(void *opaque, int64_t offset, int whence))
9.  {
10.     AVIOContext *s = av_mallocz(sizeof(AVIOContext));
11.     if (!s)
12.         return NULL;
13.     ffio_init_context(s, buffer, buffer_size, write_flag, opaque,
14.         read_packet, write_packet, seek);
15.     return s;
16. }

```

从代码中可以看出，avio\_alloc\_context()首先调用av\_mallocz()为AVIOContext分配内存。而后调用了函数ffio\_init\_context()。该函数完成了真正的初始化工作。我们看一下ffio\_init\_context()函数的定义。

```

1.  int ffio_init_context(AVIOContext *s,
2.                      unsigned char *buffer,
3.                      int buffer_size,
4.                      int write_flag,
5.                      void *opaque,
6.                      int (*read_packet)(void *opaque, uint8_t *buf, int buf_size),
7.                      int (*write_packet)(void *opaque, uint8_t *buf, int buf_size),
8.                      int64_t (*seek)(void *opaque, int64_t offset, int whence))
9.  {
10.     s->buffer      = buffer;
11.     s->orig_buffer_size =
12.     s->buffer_size = buffer_size;
13.     s->buf_ptr      = buffer;
14.     s->opaque       = opaque;
15.     s->direct       = 0;
16.
17.
18.     url_resetbuf(s, write_flag ? AVIO_FLAG_WRITE : AVIO_FLAG_READ);
19.
20.
21.     s->write_packet = write_packet;
22.     s->read_packet  = read_packet;
23.     s->seek         = seek;
24.     s->pos          = 0;
25.     s->must_flush   = 0;
26.     s->eof_reached  = 0;
27.     s->error        = 0;
28.     s->seekable     = seek ? AVIO_SEEKABLE_NORMAL : 0;
29.     s->max_packet_size = 0;
30.     s->update_checksum = NULL;
31.
32.
33.     if (!read_packet && !write_flag) {
34.         s->pos      = buffer_size;
35.         s->buf_end = s->buffer + buffer_size;
36.     }
37.     s->read_pause = NULL;
38.     s->read_seek  = NULL;
39.
40.
41.     return 0;
42. }

```

从函数的代码可以看出，ffio\_init\_context()对AVIOContext中的缓存，函数指针等等进行了赋值。

## AVStream, AVCodecContext

AVStream的初始化函数是avformat\_new\_stream()，销毁函数使用销毁AVFormatContext的avformat\_free\_context()就可以了。

### avformat\_new\_stream()

avformat\_new\_stream()的声明位于libavformat\avformat.h中，如下所示。

```

1.  /**
2.   * Add a new stream to a media file.
3.   *
4.   * When demuxing, it is called by the demuxer in read_header(). If the
5.   * flag AVFMTCTX_NOHEADER is set in s.ctx_flags, then it may also
6.   * be called in read_packet().
7.   *
8.   * When muxing, should be called by the user before avformat_write_header().
9.   *
10.  * User is required to call avcodec_close() and avformat_free_context() to
11.  * clean up the allocation by avformat_new_stream().
12.  *
13.  * @param s media file handle
14.  * @param c If non-NULL, the AVCodecContext corresponding to the new stream
15.  * will be initialized to use this codec. This is needed for e.g. codec-specific
16.  * defaults to be set, so codec should be provided if it is known.
17.  *
18.  * @return newly created stream or NULL on error.
19.  */
20. AVStream *avformat_new_stream(AVFormatContext *s, const AVCodec *c);

```

avformat\_new\_stream()的定义位于libavformat\utils.c中，如下所示。

```

1. AVStream *avformat_new_stream(AVFormatContext *s, const AVCodec *c)
2. {
3.     AVStream *st;
4.     int i;
5.     AVStream **streams;
6.
7.
8.     if (s->nb_streams >= INT_MAX/sizeof(*streams))
9.         return NULL;
10.    streams = av_realloc_array(s->streams, s->nb_streams + 1, sizeof(*streams));
11.    if (!streams)
12.        return NULL;
13.    s->streams = streams;
14.
15.
16.    st = av_mallocz(sizeof(AVStream));
17.    if (!st)
18.        return NULL;
19.    if (!(st->info = av_mallocz(sizeof(*st->info)))) {
20.        av_free(st);
21.        return NULL;
22.    }
23.    st->info->last_dts = AV_NOPTS_VALUE;
24.
25.
26.    st->codec = avcodec_alloc_context3(c);
27.    if (s->iformat) {
28.        /* no default bitrate if decoding */
29.        st->codec->bit_rate = 0;
30.
31.
32.        /* default pts setting is MPEG-like */
33.        avpriv_set_pts_info(st, 33, 1, 90000);
34.    }
35.
36.
37.    st->index = s->nb_streams;
38.    st->start_time = AV_NOPTS_VALUE;
39.    st->duration = AV_NOPTS_VALUE;
40.    /* we set the current DTS to 0 so that formats without any timestamps
41.     * but durations get some timestamps, formats with some unknown
42.     * timestamps have their first few packets buffered and the
43.     * timestamps corrected before they are returned to the user */
44.    st->cur_dts = s->iformat ? RELATIVE_TS_BASE : 0;
45.    st->first_dts = AV_NOPTS_VALUE;
46.    st->probe_packets = MAX_PROBE_PACKETS;
47.    st->pts_wrap_reference = AV_NOPTS_VALUE;
48.    st->pts_wrap_behavior = AV_PTS_WRAP_IGNORE;
49.
50.
51.    st->last_IP_pts = AV_NOPTS_VALUE;
52.    st->last_dts_for_order_check = AV_NOPTS_VALUE;
53.    for (i = 0; i < MAX_REORDER_DELAY + 1; i++)
54.        st->pts_buffer[i] = AV_NOPTS_VALUE;
55.
56.
57.    st->sample_aspect_ratio = (AVRational) { 0, 1 };
58.
59.
60.    #if FF_API_R_FRAME_RATE
61.        st->info->last_dts = AV_NOPTS_VALUE;
62.    #endif
63.    st->info->fps_first_dts = AV_NOPTS_VALUE;
64.    st->info->fps_last_dts = AV_NOPTS_VALUE;
65.
66.
67.    st->inject_global_side_data = s->internal->inject_global_side_data;
68.
69.
70.    s->streams[s->nb_streams++] = st;
71.    return st;
72. }

```

从代码中可以看出，avformat\_new\_stream()首先调用av\_mallocz()为AVStream分配内存。接着给新分配的AVStream的各个字段赋上默认值。然后调用了另一个函数avcodec\_alloc\_context3()初始化AVStream中的AVCodecContext。

## avcodec\_alloc\_context3()

avcodec\_alloc\_context3()的声明位于libavcodec\avcodec.h中，如下所示。

```

1.  /**
2.   * Allocate an AVCodecContext and set its fields to default values. The
3.   * resulting struct should be freed with avcodec_free_context().
4.   *
5.   * @param codec if non-NULL, allocate private data and initialize defaults
6.   *             for the given codec. It is illegal to then call avcodec_open2()
7.   *             with a different codec.
8.   *             If NULL, then the codec-specific defaults won't be initialized,
9.   *             which may result in suboptimal default settings (this is
10.   *             important mainly for encoders, e.g. libx264).
11.   *
12.   * @return An AVCodecContext filled with default values or NULL on failure.
13.   * @see avcodec_get_context_defaults
14.   */
15. AVCodecContext *avcodec_alloc_context3(const AVCodec *codec);

```

下面我们看一下avcodec\_alloc\_context3()的定义。下面我们看一下avcodec\_alloc\_context3()的定义。avcodec\_alloc\_context3()的定义位于libavcodec/options.c中。

```

1.  AVCodecContext *avcodec_alloc_context3(const AVCodec *codec)
2.  {
3.      AVCodecContext *avctx= av_malloc(sizeof(AVCodecContext));
4.
5.
6.      if (!avctx)
7.          return NULL;
8.
9.
10.     if(avcodec_get_context_defaults3(avctx, codec) < 0){
11.         av_free(avctx);
12.         return NULL;
13.     }
14.
15.
16.     return avctx;
17. }

```

从代码中可以看出，avcodec\_alloc\_context3()首先调用av\_malloc()为AVCodecContext分配存储空间，然后调用了函数avcodec\_get\_context\_defaults3()用于设置该AVCodecContext的默认值。avcodec\_get\_context\_defaults3()的定义如下。

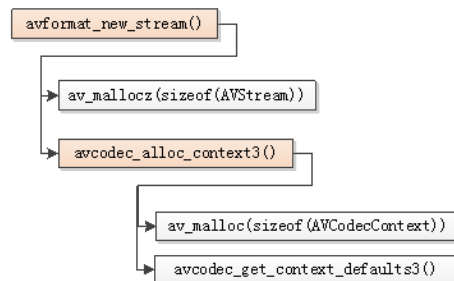


```

1. int avcodec_get_context_defaults3(AVCodecContext *s, const AVCodec *codec)
2. {
3.     int flags=0;
4.     memset(s, 0, sizeof(AVCodecContext));
5.
6.
7.     s->av_class = &av_codec_context_class;
8.
9.
10.    s->codec_type = codec ? codec->type : AVMEDIA_TYPE_UNKNOWN;
11.    if (codec)
12.        s->codec_id = codec->id;
13.
14.
15.    if(s->codec_type == AVMEDIA_TYPE_AUDIO)
16.        flags= AV_OPT_FLAG_AUDIO_PARAM;
17.    else if(s->codec_type == AVMEDIA_TYPE_VIDEO)
18.        flags= AV_OPT_FLAG_VIDEO_PARAM;
19.    else if(s->codec_type == AVMEDIA_TYPE_SUBTITLE)
20.        flags= AV_OPT_FLAG_SUBTITLE_PARAM;
21.    av_opt_set_defaults2(s, flags, flags);
22.
23.
24.    s->time_base      = (AVRational){0,1};
25.    s->get_buffer2     = avcodec_default_get_buffer2;
26.    s->get_format      = avcodec_default_get_format;
27.    s->execute         = avcodec_default_execute;
28.    s->execute2        = avcodec_default_execute2;
29.    s->sample_aspect_ratio = (AVRational){0,1};
30.    s->pix_fmt         = AV_PIX_FMT_NONE;
31.    s->sample_fmt      = AV_SAMPLE_FMT_NONE;
32.    s->timecode_frame_start = -1;
33.
34.
35.    s->reordered_opaque = AV_NOPTS_VALUE;
36.    if(codec && codec->priv_data_size){
37.        if(!s->priv_data){
38.            s->priv_data= av_mallocz(codec->priv_data_size);
39.            if (!s->priv_data) {
40.                return AVERROR(ENOMEM);
41.            }
42.        }
43.        if(codec->priv_class){
44.            *(const AVClass**)s->priv_data = codec->priv_class;
45.            av_opt_set_defaults(s->priv_data);
46.        }
47.    }
48.    if (codec && codec->defaults) {
49.        int ret;
50.        const AVCodecDefault *d = codec->defaults;
51.        while (d->key) {
52.            ret = av_opt_set(s, d->key, d->value, 0);
53.            av_assert0(ret >= 0);
54.            d++;
55.        }
56.    }
57.    return 0;
58. }

```

avformat\_new\_stream()函数的调用结构如下所示。



雷霄骅 (Lei Xiaohua)  
 Email: leixiaohua1020@126.com  
 Website: <http://blog.csdn.net/leixiaohua1020>

## AVFrame

AVFrame的初始化函数是av\_frame\_alloc(), 销毁函数是av\_frame\_free()。在这里有一点需要注意, 旧版的FFmpeg都是使用avcodec\_alloc\_frame()初始化AVFrame的, 但是我在写这篇文章的时候, avcodec\_alloc\_frame()已经被标记为“过时的”了, 为了保证与时俱进, 决定分析新的API——av\_frame\_alloc()。

### av\_frame\_alloc()

av\_frame\_alloc()的声明位于libavutil/frame.h，如下所示。

```
[cpp]
1. /**
2.  * Allocate an AVFrame and set its fields to default values. The resulting
3.  * struct must be freed using av_frame_free().
4.  *
5.  * @return An AVFrame filled with default values or NULL on failure.
6.  *
7.  * @note this only allocates the AVFrame itself, not the data buffers. Those
8.  * must be allocated through other means, e.g. with av_frame_get_buffer() or
9.  * manually.
10. */
11. AVFrame *av_frame_alloc(void);
```

av\_frame\_alloc()的定义位于libavutil/frame.c。代码如下所示。

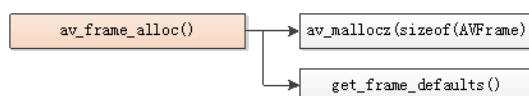
```
[cpp]
1. AVFrame *av_frame_alloc(void)
2. {
3.     AVFrame *frame = av_mallocz(sizeof(*frame));
4.
5.
6.     if (!frame)
7.         return NULL;
8.
9.
10.    frame->extended_data = NULL;
11.    get_frame_defaults(frame);
12.
13.
14.    return frame;
15. }
```

从代码可以看出，av\_frame\_alloc()首先调用av\_mallocz()为AVFrame结构体分配内存。而后调用了函数get\_frame\_defaults()用于设置一些默认参数。get\_frame\_defaults()定义如下。

```
[cpp]
1. static void get_frame_defaults(AVFrame *frame)
2. {
3.     if (frame->extended_data != frame->data)
4.         av_freep(&frame->extended_data);
5.
6.
7.     memset(frame, 0, sizeof(*frame));
8.
9.
10.    frame->pts =
11.    frame->pkt_dts =
12.    frame->pkt_pts = AV_NOPTS_VALUE;
13.    av_frame_set_best_effort_timestamp(frame, AV_NOPTS_VALUE);
14.    av_frame_set_pkt_duration(frame, 0);
15.    av_frame_set_pkt_pos(frame, -1);
16.    av_frame_set_pkt_size(frame, -1);
17.    frame->key_frame = 1;
18.    frame->sample_aspect_ratio = (AVRational){ 0, 1 };
19.    frame->format = -1; /* unknown */
20.    frame->extended_data = frame->data;
21.    frame->color_primaries = AVCOL_PRI_UNSPECIFIED;
22.    frame->color_trc = AVCOL_TRC_UNSPECIFIED;
23.    frame->colourspace = AVCOL_SPC_UNSPECIFIED;
24.    frame->color_range = AVCOL_RANGE_UNSPECIFIED;
25.    frame->chroma_location = AVCHROMA_LOC_UNSPECIFIED;
26. }
```

从av\_frame\_alloc()的代码我们可以看出，该函数并没有为AVFrame的像素数据分配空间。因此AVFrame中的像素数据的空间需要自行分配空间，例如使用avpicture\_fill()，av\_image\_fill\_arrays()等函数。

av\_frame\_alloc()函数的调用结构如下所示。



<http://blog.csdn.net/leixiaohua1020>

## avpicture\_fill()

avpicture\_fill()的声明位于libavcodec\avcodec.h，如下所示。

```
[cpp]
1.  /**
2.   * Setup the picture fields based on the specified image parameters
3.   * and the provided image data buffer.
4.   *
5.   * The picture fields are filled in by using the image data buffer
6.   * pointed to by ptr.
7.   *
8.   * If ptr is NULL, the function will fill only the picture linesize
9.   * array and return the required size for the image buffer.
10.  *
11.  * To allocate an image buffer and fill the picture data in one call,
12.  * use avpicture_alloc().
13.  *
14.  * @param picture    the picture to be filled in
15.  * @param ptr        buffer where the image data is stored, or NULL
16.  * @param pix_fmt    the pixel format of the image
17.  * @param width      the width of the image in pixels
18.  * @param height     the height of the image in pixels
19.  * @return the size in bytes required for src, a negative error code
20.  * in case of failure
21.  *
22.  * @see av_image_fill_arrays()
23.  */
24.  int avpicture_fill(AVPicture *picture, const uint8_t *ptr,
25.                    enum AVPixelFormat pix_fmt, int width, int height);
```

avpicture\_fill()的定义位于libavcodec\avpicture.c，如下所示。

PS：目测这个函数未来也有可能成为“过时的”函数，因为通过观察这一年FFmpeg代码的变化，发现FFmpeg组织似乎想把AVFrame相关的函数（原先定义在AVCodec的头文件中）从AVCodec的代码中分离出来，形成一套单独的API。所以很多和AVFrame相关的名称为avcodec\_XXX()的函数都被标记上了“过时的”标记。当然，上述推测也是我自己猜测的。

```
[cpp]
1.  int avpicture_fill(AVPicture *picture, const uint8_t *ptr,
2.                    enum AVPixelFormat pix_fmt, int width, int height)
3.  {
4.      return av_image_fill_arrays(picture->data, picture->linesize,
5.                                  ptr, pix_fmt, width, height, 1);
6.  }
```

从代码中可以看出，avpicture\_fill()仅仅是简单调用了一下av\_image\_fill\_arrays()。也就是说这两个函数实际上是等同的。

## av\_image\_fill\_arrays()

av\_image\_fill\_arrays()的声明位于libavutil\imgutils.h中，如下所示。

```
[cpp]
1.  /**
2.   * Setup the data pointers and linesizes based on the specified image
3.   * parameters and the provided array.
4.   *
5.   * The fields of the given image are filled in by using the src
6.   * address which points to the image data buffer. Depending on the
7.   * specified pixel format, one or multiple image data pointers and
8.   * line sizes will be set. If a planar format is specified, several
9.   * pointers will be set pointing to the different picture planes and
10.  * the line sizes of the different planes will be stored in the
11.  * lines_sizes array. Call with !src to get the required
12.  * size for the src buffer.
13.  *
14.  * To allocate the buffer and fill in the dst_data and dst_linesize in
15.  * one call, use av_image_alloc().
16.  *
17.  * @param dst_data    data pointers to be filled in
18.  * @param dst_linesize linesizes for the image in dst_data to be filled in
19.  * @param src        buffer which will contain or contains the actual image data, can be NULL
20.  * @param pix_fmt    the pixel format of the image
21.  * @param width      the width of the image in pixels
22.  * @param height     the height of the image in pixels
23.  * @param align      the value used in src for linesize alignment
24.  * @return the size in bytes required for src, a negative error code
25.  * in case of failure
26.  */
27.  int av_image_fill_arrays(uint8_t *dst_data[4], int dst_linesize[4],
28.                          const uint8_t *src,
29.                          enum AVPixelFormat pix_fmt, int width, int height, int align);
```

av\_image\_fill\_arrays()的定义位于libavutil\imgutils.c中。

```
[cpp]
1. int av_image_fill_arrays(uint8_t *dst_data[4], int dst_linesize[4],
2.                          const uint8_t *src,
3.                          enum AVPixelFormat pix_fmt, int width, int height, int align)
4. {
5.     int ret, i;
6.
7.
8.     if ((ret = av_image_check_size(width, height, 0, NULL)) < 0)
9.         return ret;
10.
11.
12.     if ((ret = av_image_fill_linesizes(dst_linesize, pix_fmt, width)) < 0)
13.         return ret;
14.
15.
16.     for (i = 0; i < 4; i++)
17.         dst_linesize[i] = FFALIGN(dst_linesize[i], align);
18.
19.
20.     if ((ret = av_image_fill_pointers(dst_data, pix_fmt, width, NULL, dst_linesize)) < 0)
21.         return ret;
22.
23.
24.     return av_image_fill_pointers(dst_data, pix_fmt, height, (uint8_t *)src, dst_linesize);
25. }
```

av\_image\_fill\_arrays()函数中包含3个函数：av\_image\_check\_size(), av\_image\_fill\_linesizes(), av\_image\_fill\_pointers()。av\_image\_check\_size()用于检查输入的宽高参数是否合理，即不能太大或者为负数。av\_image\_fill\_linesizes()用于填充dst\_linesize。av\_image\_fill\_pointers()则用于填充dst\_data。它们的定义相对比较简单，不再详细分析。

av\_image\_check\_size()代码如下所示。

```
[cpp]
1. int av_image_check_size(unsigned int w, unsigned int h, int log_offset, void *log_ctx)
2. {
3.     ImgUtils imgutils = { &imgutils_class, log_offset, log_ctx };
4.
5.
6.     if ((int)w>0 && (int)h>0 && (w+128)*(uint64_t)(h+128) < INT_MAX/8)
7.         return 0;
8.
9.
10.     av_log(&imgutils, AV_LOG_ERROR, "Picture size %ux%u is invalid\n", w, h);
11.     return AVERROR(EINVAL);
12. }
```

av\_image\_fill\_linesizes()代码如下所示。

```
[cpp]
1. int av_image_fill_linesizes(int linesizes[4], enum AVPixelFormat pix_fmt, int width)
2. {
3.     int i, ret;
4.     const AVPixFmtDescriptor *desc = av_pix_fmt_desc_get(pix_fmt);
5.     int max_step [4]; /* max pixel step for each plane */
6.     int max_step_comp[4]; /* the component for each plane which has the max pixel step */
7.
8.
9.     memset(linesizes, 0, 4*sizeof(linesizes[0]));
10.
11.
12.     if (!desc || desc->flags & AV_PIX_FMT_FLAG_HWACCEL)
13.         return AVERROR(EINVAL);
14.
15.
16.     av_image_fill_max_pixsteps(max_step, max_step_comp, desc);
17.     for (i = 0; i < 4; i++) {
18.         if ((ret = image_get_linesize(width, i, max_step[i], max_step_comp[i], desc)) < 0)
19.             return ret;
20.         linesizes[i] = ret;
21.     }
22.
23.
24.     return 0;
25. }
```

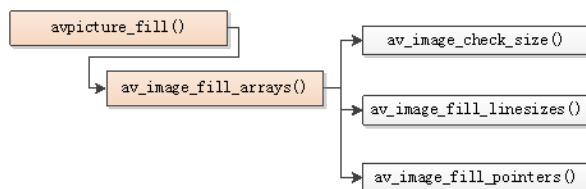
av\_image\_fill\_pointers()代码如下所示。

```

1. int av_image_fill_pointers(uint8_t *data[4], enum AVPixelFormat pix_fmt, int height,
2.                          uint8_t *ptr, const int linesizes[4])
3. {
4.     int i, total_size, size[4] = { 0 }, has_plane[4] = { 0 };
5.
6.
7.     const AVPixFmtDescriptor *desc = av_pix_fmt_desc_get(pix_fmt);
8.     memset(data, 0, sizeof(data[0])*4);
9.
10.
11.     if (!desc || desc->flags & AV_PIX_FMT_FLAG_HWACCEL)
12.         return AVERROR(EINVAL);
13.
14.
15.     data[0] = ptr;
16.     if (linesizes[0] > (INT_MAX - 1024) / height)
17.         return AVERROR(EINVAL);
18.     size[0] = linesizes[0] * height;
19.
20.
21.     if (desc->flags & AV_PIX_FMT_FLAG_PAL ||
22.         desc->flags & AV_PIX_FMT_FLAG_PSEUDOPAL) {
23.         size[0] = (size[0] + 3) & ~3;
24.         data[1] = ptr + size[0]; /* palette is stored here as 256 32 bits words */
25.         return size[0] + 256 * 4;
26.     }
27.
28.
29.     for (i = 0; i < 4; i++)
30.         has_plane[desc->comp[i].plane] = 1;
31.
32.
33.     total_size = size[0];
34.     for (i = 1; i < 4 && has_plane[i]; i++) {
35.         int h, s = (i == 1 || i == 2) ? desc->log2_chroma_h : 0;
36.         data[i] = data[i-1] + size[i-1];
37.         h = (height + (1 << s) - 1) >> s;
38.         if (linesizes[i] > INT_MAX / h)
39.             return AVERROR(EINVAL);
40.         size[i] = h * linesizes[i];
41.         if (total_size > INT_MAX - size[i])
42.             return AVERROR(EINVAL);
43.         total_size += size[i];
44.     }
45.
46.
47.     return total_size;
48. }

```

avpicture\_fill()函数调用关系如下图所示。



雷霄骅 (LeiXiaohua)  
Email: leixiaohua1020@126.com  
Website: <http://blog.csdn.net/leixiaohua1020>

## av\_frame\_free()

av\_frame\_free()的声明位于libavutil/frame.h，如下所示。

```

1. /**
2.  * Free the frame and any dynamically allocated objects in it,
3.  * e.g. extended data. If the frame is reference counted, it will be
4.  * unreferenced first.
5.  *
6.  * @param frame frame to be freed. The pointer will be set to NULL.
7.  */
8. void av_frame_free(AVFrame **frame);

```

av\_frame\_free()的定义位于libavutil/frame.c。代码如下所示。

```
[cpp]
1. void av_frame_free(AVFrame **frame)
2. {
3.     if (!frame || !*frame)
4.         return;
5.
6.
7.     av_frame_unref(*frame);
8.     av_freep(frame);
9. }
```

在释放AVFrame结构体之前，首先调用了函数av\_frame\_unref()。av\_frame\_unref()也是一个FFmpeg的API，它的作用是释放AVFrame中参考的缓存（还没完全弄懂），并且重置AVFrame中的字段。调用这个函数的目的应该是为了确保AVFrame可以被正常释放。代码如下。

```
[cpp]
1. void av_frame_unref(AVFrame *frame)
2. {
3.     int i;
4.
5.
6.     for (i = 0; i < frame->nb_side_data; i++) {
7.         free_side_data(&frame->side_data[i]);
8.     }
9.     av_freep(&frame->side_data);
10.
11.
12.     for (i = 0; i < FF_ARRAY_ELEMS(frame->buf); i++)
13.         av_buffer_unref(&frame->buf[i]);
14.     for (i = 0; i < frame->nb_extended_buf; i++)
15.         av_buffer_unref(&frame->extended_buf[i]);
16.     av_freep(&frame->extended_buf);
17.     av_dict_free(&frame->metadata);
18.     av_buffer_unref(&frame->qp_table_buf);
19.
20.
21.     get_frame_defaults(frame);
22. }
```

## AVPacket

AVPacket的初始化函数有两个：av\_init\_packet()，av\_new\_packet()。销毁函数是av\_free\_packet()。在初始化函数中av\_init\_packet()比较简单，初始化一些字段；而av\_new\_packet()相对“高级”一些，除了包含av\_init\_packet()的功能之外，还包含了AVPacket内部内存的分配。下面分别看看这些函数。

### av\_init\_packet()

av\_init\_packet()的声明位于libavcodec\avcodec.h，如下所示。

```
[cpp]
1. /**
2.  * Initialize optional fields of a packet with default values.
3.  *
4.  * Note, this does not touch the data and size members, which have to be
5.  * initialized separately.
6.  *
7.  * @param pkt packet
8.  */
9. void av_init_packet(AVPacket *pkt);
```

av\_init\_packet()的定义位于libavcodec\avpacket.c。如下所示。

```
[cpp]
1. void av_init_packet(AVPacket *pkt)
2. {
3.     pkt->pts          = AV_NOPTS_VALUE;
4.     pkt->dts          = AV_NOPTS_VALUE;
5.     pkt->pos          = -1;
6.     pkt->duration      = 0;
7.     pkt->convergence_duration = 0;
8.     pkt->flags        = 0;
9.     pkt->stream_index  = 0;
10. #if FF_API_DESTRUCT_PACKET
11. FF_DISABLE_DEPRECATED_WARNINGS
12.     pkt->destruct      = NULL;
13. FF_ENABLE_DEPRECATED_WARNINGS
14. #endif
15.     pkt->buf          = NULL;
16.     pkt->side_data     = NULL;
17.     pkt->side_data_elems = 0;
18. }
```

## av\_new\_packet()

av\_new\_packet()的声明位于libavcodec\avcodec.h。如下所示。

```
[cpp]
1. /**
2.  * Allocate the payload of a packet and initialize its fields with
3.  * default values.
4.  *
5.  * @param pkt packet
6.  * @param size wanted payload size
7.  * @return 0 if OK, AERROR_xxx otherwise
8.  */
9. int av_new_packet(AVPacket *pkt, int size);
```

av\_new\_packet()的定义位于libavcodec\avpacket.c。如下所示。

```
[cpp]
1. int av_new_packet(AVPacket *pkt, int size)
2. {
3.     AVBufferRef *buf = NULL;
4.     int ret = packet_alloc(&buf, size);
5.     if (ret < 0)
6.         return ret;
7.
8.
9.     av_init_packet(pkt);
10.    pkt->buf      = buf;
11.    pkt->data      = buf->data;
12.    pkt->size      = size;
13. #if FF_API_DESTRUCT_PACKET
14. FF_DISABLE_DEPRECATED_WARNINGS
15.     pkt->destruct = dummy_destruct_packet;
16. FF_ENABLE_DEPRECATED_WARNINGS
17. #endif
18.
19.
20.     return 0;
21. }
```

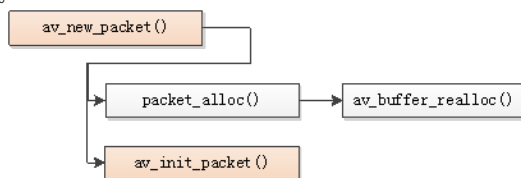
从代码可以看出，av\_new\_packet()调用了av\_init\_packet(pkt)。此外还调用了函数packet\_alloc()。packet\_alloc()函数的定义如下。

```
[cpp]
1. static int packet_alloc(AVBufferRef **buf, int size)
2. {
3.     int ret;
4.     if ((unsigned)size >= (unsigned)size + FF_INPUT_BUFFER_PADDING_SIZE)
5.         return AERROR(EINVAL);
6.
7.
8.     ret = av_buffer_realloc(buf, size + FF_INPUT_BUFFER_PADDING_SIZE);
9.     if (ret < 0)
10.        return ret;
11.
12.
13.     memset((*buf)->data + size, 0, FF_INPUT_BUFFER_PADDING_SIZE);
14.
15.
16.     return 0;
17. }
```

packet\_alloc()中调用av\_buffer\_realloc()为AVPacket分配内存。然后调用memset()将分配的内存置0。

PS：发现AVPacket的结构随着FFmpeg的发展越发复杂了。原先AVPacket中的数据仅仅存在一个uint8\_t类型的数组里，而现在已经使用一个专门的结构体AVBufferRef存储数据。

av\_new\_packet()代码的函数调用关系如下图所示。



雷霄骅 (Lei Xiaohua)  
Email: leixiaohua1020@126.com  
Website: <http://blog.csdn.net/leixiaohua1020>

## av\_free\_packet()

av\_free\_packet()的声明位于libavcodec\avcodec.h，如下所示。

```
[cpp]
1.  /**
2.   * Free a packet.
3.   *
4.   * @param pkt packet to free
5.   */
6.  void av_free_packet(AVPacket *pkt);
```

av\_free\_packet()的定义位于libavcodec\avpacket.c。如下所示。

```
[cpp]
1.  void av_free_packet(AVPacket *pkt)
2.  {
3.      if (pkt) {
4.          FF_DISABLE_DEPRECATION_WARNINGS
5.          if (pkt->buf)
6.              av_buffer_unref(&pkt->buf);
7.          #if FF_API_DESTRUCT_PACKET
8.              else if (pkt->destruct)
9.                  pkt->destruct(pkt);
10.             pkt->destruct = NULL;
11.          #endif
12.          FF_ENABLE_DEPRECATION_WARNINGS
13.             pkt->data = NULL;
14.             pkt->size = 0;
15.
16.
17.             av_packet_free_side_data(pkt);
18.         }
19.     }
```

从代码可以看出，av\_free\_packet()调用av\_buffer\_unref()释放AVPacket中的数据，而后还调用了av\_packet\_free\_side\_data()释放了side\_data（存储封装格式可以提供的额外的数据）。

**雷霄骅 (Lei Xiaohua)**

**leixiaohua1020@126.com**

**<http://blog.csdn.net/leixiaohua1020>**

版权声明：本文为博主原创文章，未经博主允许不得转载。 <https://blog.csdn.net/leixiaohua1020/article/details/41181155>

文章标签： [ffmpeg](#) [avformatcontext](#) [avstream](#) [avcodecontext](#) [avpacket](#)

个人分类： [FFMPEG](#)

所属专栏： [FFmpeg](#)

此PDF由spygg生成,请尊重原作者版权!!!

我的邮箱:liushidc@163.com