

## 原 FFMpeg源代码简单分析：avcodec\_close()

2015年03月12日 00:06:14 阅读数：11602

=====

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的avcodec\_close()函数。该函数用于关闭编码器。avcodec\_close()函数的声明位于libavcodec\avcodec.h，如下所示。

```
[cpp]
1.  /**
2.   * Close a given AVCodecContext and free all the data associated with it
3.   * (but not the AVCodecContext itself).
4.   *
5.   * Calling this function on an AVCodecContext that hasn't been opened will free
6.   * the codec-specific data allocated in avcodec_alloc_context3() /
7.   * avcodec_get_context_defaults3() with a non-NULL codec. Subsequent calls will
8.   * do nothing.
9.   */
10. int avcodec_close(AVCodecContext *avctx);
```

该函数只有一个参数，就是需要关闭的编码器的AVCodecContext。

## 函数调用关系图

函数的调用关系图如下所示。

□

## avcodec\_close()



avcodec\_close()的定义位于libavcodec\utils.c，如下所示。

```
[cpp]
1.  av_cold int avcodec_close(AVCodecContext *avctx)
2.  {
3.      if (!avctx)
4.          return 0;
5.
6.      if (avcodec_is_open(avctx)) {
7.          FramePool *pool = avctx->internal->pool;
8.          int i;
9.          if (CONFIG_FRAME_THREAD_ENCODER &&
10.             avctx->internal->frame_thread_encoder && avctx->thread_count > 1) {
11.              ff_frame_thread_encoder_free(avctx);
12.          }
13.          if (HAVE_THREADS && avctx->internal->thread_ctx)
14.              ff_thread_free(avctx);
15.          //关闭编解码器
16.          if (avctx->codec && avctx->codec->close)
17.              avctx->codec->close(avctx);
18.          avctx->coded_frame = NULL;
19.          avctx->internal->byte_buffer_size = 0;
20.          av_freep(&avctx->internal->byte_buffer);
21.          av_frame_free(&avctx->internal->to_free);
22.          for (i = 0; i < FF_ARRAY_ELEMS(pool->pools); i++)
23.              av_buffer_pool_uninit(&pool->pools[i]);
24.          av_freep(&avctx->internal->pool);
25.
26.          if (avctx->hwaccel && avctx->hwaccel->uninit)
27.              avctx->hwaccel->uninit(avctx);
28.          av_freep(&avctx->internal->hwaccel_priv_data);
29.
30.          av_freep(&avctx->internal);
31.      }
32.
33.      if (avctx->priv_data && avctx->codec && avctx->codec->priv_class)
34.          av_opt_free(avctx->priv_data);
35.      av_opt_free(avctx);
36.      av_freep(&avctx->priv_data);
37.      if (av_codec_is_encoder(avctx->codec))
38.          av_freep(&avctx->extradata);
39.      avctx->codec = NULL;
40.      avctx->active_thread_type = 0;
41.
42.      return 0;
43.  }
```



从avcodec\_close()的定义可以看出，该函数释放AVCodecContext中有关的变量，并且调用了AVCodec的close()关闭了解码器。

## AVCodec->close()

AVCodec的close()是一个函数指针，指向了特定编码器的关闭函数。在这里我们以libx264为例，看一下它对应的AVCodec的结构体的定义，如下所示。

```
[cpp]    
1. AVCodec ff_libx264_encoder = {  
2.     .name           = "libx264",  
3.     .long_name      = NULL_IF_CONFIG_SMALL("libx264 H.264 / AVC / MPEG-4 AVC / MPEG-4 part 10"),  
4.     .type           = AVMEDIA_TYPE_VIDEO,  
5.     .id             = AV_CODEC_ID_H264,  
6.     .priv_data_size = sizeof(X264Context),  
7.     .init           = X264_init,  
8.     .encode2        = X264_frame,  
9.     .close          = X264_close,  
10.    .capabilities    = CODEC_CAP_DELAY | CODEC_CAP_AUTO_THREADS,  
11.    .priv_class      = &x264_class,  
12.    .defaults        = x264_defaults,  
13.    .init_static_data = X264_init_static,  
14. };
```

从ff\_libx264\_encoder的定义可以看出：close()函数对应的是X264\_close()函数。继续看一下X264\_close()函数的定义，如下所示。

```
[cpp]    
1. static av_cold int X264_close(AVCodecContext *avctx)  
2. {  
3.     X264Context *x4 = avctx->priv_data;  
4.  
5.     av_freep(&avctx->extradata);  
6.     av_freep(&x4->sei);  
7.     //关闭编码器  
8.     if (x4->enc)  
9.         x264_encoder_close(x4->enc);  
10.  
11.    av_frame_free(&avctx->coded_frame);  
12.  
13.    return 0;  
14. }
```

从X264\_close()的定义可以看出，该函数调用了libx264的x264\_encoder\_close()关闭了libx264编码器。

**雷霄骅**

**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/44206699>

文章标签：[FFmpeg](#) [AVCodec](#) [编码器](#) [源代码](#)

个人分类：[FFMPEG](#)

所属专栏：[FFmpeg](#)

此PDF由[spyyg](#)生成, 请尊重原作者版权!!!

我的邮箱: [liushidc@163.com](mailto:liushidc@163.com)