

原 FFMpeg源代码简单分析：avformat_close_input()

2015年03月07日 10:58:52 阅读数：15598

=====

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avformat_close_input()函数。该函数用于关闭一个AVFormatContext，一般情况下是和avformat_open_input()成对使用的。

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

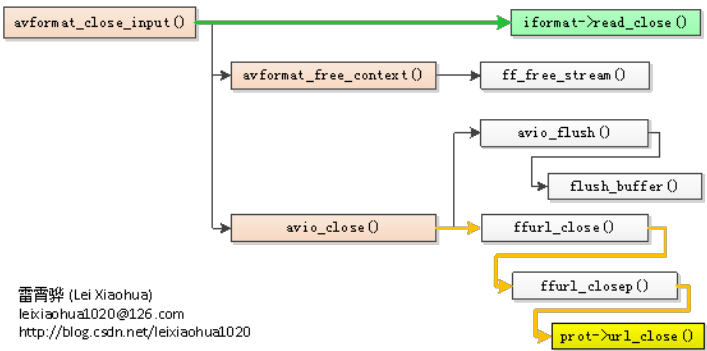
```
[cpp]
1.  /**
2.   * Close an opened input AVFormatContext. Free it and all its contents
3.   * and set *s to NULL.
4.   */
5.  void avformat_close_input(AVFormatContext **s);
```

该函数最典型的例子可以参考：

[最简单的基于FFMPEG+SDL的视频播放器 ver2 （采用SDL2.0）](#)

函数调用关系图

函数的调用关系如下图所示。



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

avformat_close_input()

下面看一下avformat_close_input()的源代码，位于libavformat\utils.c文件中。

```

1. void avformat_close_input(AVFormatContext **ps)
2. {
3.     AVFormatContext *s;
4.     AVIOContext *pb;
5.
6.     if (!ps || !*ps)
7.         return;
8.
9.     s = *ps;
10.    pb = s->pb;
11.
12.    if ((s->iformat && strcmp(s->iformat->name, "image2") && s->iformat->flags & AVFMT_NOFILE) ||
13.        (s->flags & AVFMT_FLAG_CUSTOM_IO))
14.        pb = NULL;
15.
16.    flush_packet_queue(s);
17.
18.    if (s->iformat)
19.        if (s->iformat->read_close)
20.            s->iformat->read_close(s);
21.
22.    avformat_free_context(s);
23.
24.    *ps = NULL;
25.
26.    avio_close(pb);
27. }

```

从源代码中可以看出，avformat_close_input()主要做了以下几步工作：

- (1) 调用AVInputFormat的read_close()方法关闭输入流
- (2) 调用avformat_free_context()释放AVFormatContext
- (3) 调用avio_close()关闭并且释放AVIOContext

下面我们分别来看上述几个步骤。

AVInputFormat-> read_close()

AVInputFormat的read_close()是一个函数指针，指向关闭输入流的函数。不同的AVInputFormat包含有不同的read_close()方法。例如，FLV格式对应的AVInputFormat的定义如下。

```

1. AVInputFormat ff_flv_demuxer = {
2.     .name = "flv",
3.     .long_name = NULL_IF_CONFIG_SMALL("FLV (Flash Video)"),
4.     .priv_data_size = sizeof(FLVContext),
5.     .read_probe = flv_probe,
6.     .read_header = flv_read_header,
7.     .read_packet = flv_read_packet,
8.     .read_seek = flv_read_seek,
9.     .read_close = flv_read_close,
10.    .extensions = "flv",
11.    .priv_class = &flv_class,
12. };

```

从ff_flv_demuxer的定义中可以看出，read_close()指向的函数是flv_read_close()。我们可以看一下flv_read_close()的定义，如下所示。

```

1. static int flv_read_close(AVFormatContext *s)
2. {
3.     int i;
4.     FLVContext *flv = s->priv_data;
5.     for (i=0; i<FLV_STREAM_TYPE_NB; i++)
6.         av_freep(&flv->new_extradata[i]);
7.     return 0;
8. }

```

从flv_read_close()的定义可以看出，该函数释放了FLVContext中的new_extradata数组中每个元素指向的内存。

avformat_free_context()

avformat_free_context()是一个FFmpeg的API函数，用于释放一个AVFormatContext。在这里要注意搞清楚avformat_free_context()和avformat_close_input()之间的区别与联系。

有关avformat_free_context()可以参考文章：

[FFmpeg源代码简单分析：内存的分配和释放](#)

avio_close()

avio_close()是一个FFmpeg的API函数，用于关闭和释放AVIOContext。它的声明位于libavformat\avio.h，如下所示。

```
[cpp]
1.  /**
2.   * Close the resource accessed by the AVIOContext s and free it.
3.   * This function can only be used if s was opened by avio_open().
4.   *
5.   * The internal buffer is automatically flushed before closing the
6.   * resource.
7.   *
8.   * @return 0 on success, an AVERROR < 0 on error.
9.   * @see avio_closep
10.  */
11. int avio_close(AVIOContext *s);
```

avio_close()的定义位于libavformat\aviobuf.c，如下所示。

```
[cpp]
1.  int avio_close(AVIOContext *s)
2.  {
3.      URLContext *h;
4.
5.      if (!s)
6.          return 0;
7.
8.      avio_flush(s);
9.      h = s->opaque;
10.     av_freep(&s->buffer);
11.     if (s->write_flag)
12.         av_log(s, AV_LOG_DEBUG, "Statistics: %d seeks, %d writeouts\n", s->seek_count, s->writeout_count);
13.     else
14.         av_log(s, AV_LOG_DEBUG, "Statistics: %"PRIu64" bytes read, %d seeks\n", s->bytes_read, s->seek_count);
15.     av_free(s);
16.     return ffurl_close(h);
17. }
```

从源代码可以看出，avio_close()按照顺序做了以下几个步骤：

- (1) 调用avio_flush()强制清除缓存中的数据
- (2) 调用av_freep()释放掉AVIOContext种的buffer
- (3) 调用av_free()释放掉AVIOContext结构体
- (4) 调用ffurl_close()关闭并且释放掉URLContext

下面按照顺序分别看看avio_flush()和ffurl_close()这两个函数。

avio_flush()

avio_flush()是一个FFmpeg的API函数，声明位于libavformat\avio.h，如下所示。

```
[cpp]
1.  /**
2.   * Force flushing of buffered data.
3.   *
4.   * For write streams, force the buffered data to be immediately written to the output,
5.   * without to wait to fill the internal buffer.
6.   *
7.   * For read streams, discard all currently buffered data, and advance the
8.   * reported file position to that of the underlying stream. This does not
9.   * read new data, and does not perform any seeks.
10.  */
11. void avio_flush(AVIOContext *s);
```

avio_flush()的定义位于libavformat\aviobuf.c，如下所示。

```
[cpp]
1.  void avio_flush(AVIOContext *s)
2.  {
3.      flush_buffer(s);
4.      s->must_flush = 0;
5.  }
```

可以看出avio_flush()简单调用了flush_buffer()函数。我们看一下flush_buffer()的定义。

```
[cpp]
1. static void flush_buffer(AVIOContext *s)
2. {
3.     if (s->write_flag && s->buf_ptr > s->buffer) {
4.         writeout(s, s->buffer, s->buf_ptr - s->buffer);
5.         if (s->update_checksum) {
6.             s->checksum = s->update_checksum(s->checksum, s->checksum_ptr,
7.                                               s->buf_ptr - s->checksum_ptr);
8.             s->checksum_ptr = s->buffer;
9.         }
10.    }
11.    s->buf_ptr = s->buffer;
12.    if (!s->write_flag)
13.        s->buf_end = s->buffer;
14. }
```

从flush_buffer()定义我们可以看出，该函数将当前缓存指针buf_ptr的位置重新设置到缓存buffer的首部，然后根据AVIOContext对应的流是否可写分别做不同的处理。如果AVIOContext对应的流是只读的（write_flag取值为0），就将缓存的尾部buf_end设定到缓存首部位置；如果AVIOContext对应的流如果是可写的（write_flag取值非0），则会调用writeout()函数输出缓存中剩余的数据。

在这里我们看一下writeout()函数的定义，如下所示。

```
[cpp]
1. static void writeout(AVIOContext *s, const uint8_t *data, int len)
2. {
3.     if (s->write_packet && !s->error) {
4.         int ret = s->write_packet(s->opaque, (uint8_t *)data, len);
5.         if (ret < 0) {
6.             s->error = ret;
7.         }
8.     }
9.     s->writeout_count ++;
10.    s->pos += len;
11. }
```

从定义可以看出，writeout()调用了AVIOContext的write_packet()方法。根据此前文章《[FFmpeg源代码简单分析:avio_open2\(\)](#)》中的分析我们可以了解到，AVIOContext的write_packet()实际指向了ffurl_write()函数，而ffurl_write()经过retry_transfer_wrapper()函数最终调用了URLProtocol的url_write()函数。url_write()是一个函数指针，不同的URLProtocol的url_write()指向不同的函数。

例如，file（文件）对应的URLProtocol的定义位于libavformatfile.c，如下所示。

```
[cpp]
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.    .priv_data_size = sizeof(FileContext),
11.    .priv_data_class = &file_class,
12. };
```

可以看出ff_file_protocol中的url_write()指向的是file_write()函数。我们继续看一下file_write()的源代码，如下所示。

```
[cpp]
1. static int file_write(URLContext *h, const unsigned char *buf, int size)
2. {
3.     FileContext *c = h->priv_data;
4.     int r;
5.     size = FFMIN(size, c->blocksize);
6.     r = write(c->fd, buf, size);
7.     return (-1 == r)?AVERROr(errno):r;
8. }
```

从源代码中可以看出file_write()调用了系统的write()方法向文件中写数据（很多人可能对write()函数很陌生，可以简单理解为它等同于fwrite()）。

ffurl_close()和ffurl_closep()

ffurl_close()和ffurl_closep()是FFmpeg内部的两个函数，它们的声明位于libavformaturl.h，如下所示。

```
[cpp]
1.  /**
2.   * Close the resource accessed by the URLContext h, and free the
3.   * memory used by it. Also set the URLContext pointer to NULL.
4.   *
5.   * @return a negative value if an error condition occurred, 0
6.   * otherwise
7.   */
8.  int ffurl_cclose(URLContext **h);
9.  int ffurl_cclose(URLContext *h);
```

其实这两个函数是等同的。ffurl_cclose()的定义位于libavformat\lavio.c，如下所示。

```
[cpp]
1.  int ffurl_cclose(URLContext *h)
2.  {
3.      return ffurl_cclose(&h);
4.  }
```

可见ffurl_cclose()调用了ffurl_cclose()。

ffurl_cclose()的定义如下所示。

```
[cpp]
1.  int ffurl_cclose(URLContext **hh)
2.  {
3.      URLContext *h= *hh;
4.      int ret = 0;
5.      if (!h)
6.          return 0; /* can happen when ffurl_open fails */
7.
8.      if (h->is_connected && h->prot->url_cclose)
9.          ret = h->prot->url_cclose(h);
10. #if CONFIG_NETWORK
11.     if (h->prot->flags & URL_PROTOCOL_FLAG_NETWORK)
12.         ff_network_cclose();
13. #endif
14.     if (h->prot->priv_data_size) {
15.         if (h->prot->priv_data_class)
16.             av_opt_free(h->priv_data);
17.         av_freep(&h->priv_data);
18.     }
19.     av_freep(hh);
20.     return ret;
21. }
```

从ffurl_cclose()的定义可以看出，它主要做了两步工作：

- (1) 调用URLProtocol的url_cclose()
- (2) 调用av_freep()释放URLContext结构体

其中URLProtocol的url_cclose()是一个函数指针，其指向的函数与具体的URLProtocol有关，这里我们还是看一下file（文件）对应的URLProtocol，如下所示。

```
[cpp]
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_cclose     = file_cclose,
8.      .url_get_file_handle = file_get_handle,
9.      .url_check      = file_check,
10.     .priv_data_size  = sizeof(FileContext),
11.     .priv_data_class = &file_class,
12. };
```

从ff_file_protocol中可以看出，url_cclose()指向file_cclose()函数。我们再看一下file_cclose()的定义，如下所示。

```
[cpp]
1.  static int file_cclose(URLContext *h)
2.  {
3.      FileContext *c = h->priv_data;
4.      return close(c->fd);
5.  }
```

可见file_cclose()最终调用了系统函数close()关闭了文件指针（不熟悉close()的可以简单把它理解为fclose()）。

至此avio_cclose()函数分析完毕。

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/44110683>

文章标签：

FFmpeg

源代码

关闭

avformatcontext

函数

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:liushide@163.com