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

2015年03月07日 10:58:52　阅读数：15639

==========

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
/**
 * Close an opened input AVFormatContext. Free it and all its contents
 * and set *s to NULL.
 */
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。

avformat_close_input()

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

```cpp
void avformat_close_input(AVFormatContext **ps)
{
    AVFormatContext *s;
    AVIOContext *pb;

    if (!ps || !*ps)
        return;

    s  = *ps;
    pb = s->pb;

    if ((s->iformat && strcmp(s->iformat->name, "image2") && s->iformat->flags & AVFMT_NOFILE) ||
        (s->flags & AVFMT_FLAG_CUSTOM_IO))
        pb = NULL;

    flush_packet_queue(s);

    if (s->iformat)
        if (s->iformat->read_close)
            s->iformat->read_close(s);

    avformat_free_context(s);

    *ps = NULL;

    avio_close(pb);
}
```

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。

```cpp
AVInputFormat ff_flv_demuxer = {
    .name           = "flv",
    .long_name      = NULL_IF_CONFIG_SMALL("FLV (Flash Video)"),
    .priv_data_size = sizeof(FLVContext),
    .read_probe     = flv_probe,
    .read_header    = flv_read_header,
    .read_packet    = flv_read_packet,
    .read_seek      = flv_read_seek,
    .read_close     = flv_read_close,
    .extensions     = "flv",
    .priv_class     = &flv_class,
};
```

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。

```cpp
static int flv_read_close(AVFormatContext *s)
{
    int i;
    FLVContext *flv = s->priv_data;
    for (i=0; i<FLV_STREAM_TYPE_NB; i++)
        av_freep(&flv->new_extradata[i]);
    return 0;
}
```

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
/**
 * Close the resource accessed by the AVIOContext s and free it.
 * This function can only be used if s was opened by avio_open().
 *
 * The internal buffer is automatically flushed before closing the
 * resource.
 *
 * @return 0 on success, an AVERROR < 0 on error.
 * @see avio_closep
 */
int avio_close(AVIOContext *s);
```

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

```cpp
int avio_close(AVIOContext *s)
{
    URLContext *h;

    if (!s)
        return 0;

    avio_flush(s);
    h = s->opaque;
    av_freep(&s->buffer);
    if (s->write_flag)
        av_log(s, AV_LOG_DEBUG, "Statistics: %d seeks, %d writeouts\n", s->seek_count, s->writeout_count);
    else
        av_log(s, AV_LOG_DEBUG, "Statistics: %"PRId64" bytes read, %d seeks\n", s->bytes_read, s->seek_count);
    av_free(s);
    return ffurl_close(h);
}
```

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
/**
 * Force flushing of buffered data.
 *
 * For write streams, force the buffered data to be immediately written to the output,
 * without to wait to fill the internal buffer.
 *
 * For read streams, discard all currently buffered data, and advance the
 * reported file position to that of the underlying stream. This does not
 * read new data, and does not perform any seeks.
 */
void avio_flush(AVIOContext *s);
```

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

```cpp
void avio_flush(AVIOContext *s)
{
    flush_buffer(s);
    s->must_flush = 0;
}
```

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
static void flush_buffer(AVIOContext *s)
{
    if (s->write_flag && s->buf_ptr > s->buffer) {
        writeout(s, s->buffer, s->buf_ptr - s->buffer);
        if (s->update_checksum) {
            s->checksum     = s->update_checksum(s->checksum, s->checksum_ptr,
                                                 s->buf_ptr - s->checksum_ptr);
            s->checksum_ptr = s->buffer;
        }
    }
    s->buf_ptr = s->buffer;
    if (!s->write_flag)
        s->buf_end = s->buffer;
}
```

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
static void writeout(AVIOContext *s, const uint8_t *data, int len)
{
    if (s->write_packet && !s->error) {
        int ret = s->write_packet(s->opaque, (uint8_t *)data, len);
        if (ret < 0) {
            s->error = ret;
        }
    }
    s->writeout_count ++;
    s->pos += len;
}
```

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libavformat\file.c，如下所示。

```cpp
URLProtocol ff_file_protocol = {
    .name                = "file",
    .url_open            = file_open,
    .url_read            = file_read,
    .url_write           = file_write,
    .url_seek            = file_seek,
    .url_close           = file_close,
    .url_get_file_handle = file_get_handle,
    .url_check           = file_check,
    .priv_data_size      = sizeof(FileContext),
    .priv_data_class     = &file_class,
};
```

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
static int file_write(URLContext *h, const unsigned char *buf, int size)
{
    FileContext *c = h->priv_data;
    int r;
    size = FFMIN(size, c->blocksize);
    r = write(c->fd, buf, size);
    return (-1 == r)?AVERROR(errno):r;
}
```

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libavformat\url.h，如下所示。

```cpp
/**
 * Close the resource accessed by the URLContext h, and free the
 * memory used by it. Also set the URLContext pointer to NULL.
 *
 * @return a negative value if an error condition occurred, 0
 * otherwise
 */
int ffurl_closep(URLContext **h);
int ffurl_close(URLContext *h);
```

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

```cpp
int ffurl_close(URLContext *h)
{
    return ffurl_closep(&h);
}
```

可见ffurl_close()调用了ffurl_closep()。
ffurl_closep()的定义如下所示。

```cpp
int ffurl_closep(URLContext **hh)
{
    URLContext *h= *hh;
    int ret = 0;
    if (!h)
        return 0;     /* can happen when ffurl_open fails */

    if (h->is_connected && h->prot->url_close)
        ret = h->prot->url_close(h);
#if CONFIG_NETWORK
    if (h->prot->flags & URL_PROTOCOL_FLAG_NETWORK)
        ff_network_close();
#endif
    if (h->prot->priv_data_size) {
        if (h->prot->priv_data_class)
            av_opt_free(h->priv_data);
        av_freep(&h->priv_data);
    }
    av_freep(hh);
    return ret;
}
```

从ffurl_closep()的定义可以看出，它主要做了两步工作：

（1）调用URLProtocol的url_close()

（2）调用av_freep()释放URLContext结构体

其中URLProtocol的url_close()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
URLProtocol ff_file_protocol = {
    .name                = "file",
    .url_open            = file_open,
    .url_read            = file_read,
    .url_write           = file_write,
    .url_seek            = file_seek,
    .url_close           = file_close,
    .url_get_file_handle = file_get_handle,
    .url_check           = file_check,
    .priv_data_size      = sizeof(FileContext),
    .priv_data_class     = &file_class,
};
```

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

```cpp
static int file_close(URLContext *h)
{
    FileContext *c = h->priv_data;
    return close(c->fd);
}
```

可见file_close()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_close()函数分析完毕。

雷霄骅

leixiaohua1020@126.com

http://blog.csdn.net/leixiaohua1020

文章标签： FFmpeg 源代码 关闭 avformatcontext 函数

个人分类： FFMPEG

所属专栏： FFmpeg