

原 FFMPEG结构体分析：AVIOContext

2013年11月08日 23:52:17 阅读数：25363

注：写了一系列的结构体的分析的文章，在这里列一个列表：

[FFMPEG结构体分析：AVFrame](#)

[FFMPEG结构体分析：AVFormatContext](#)

[FFMPEG结构体分析：AVCodecContext](#)

[FFMPEG结构体分析：AVIOContext](#)

[FFMPEG结构体分析：AVCodec](#)

[FFMPEG结构体分析：AVStream](#)

[FFMPEG结构体分析：AVPacket](#)

FFMPEG有几个最重要的结构体，包含了解协议，解封装，解码操作，此前已经进行过分析：

[FFMPEG中最关键的结构体之间的关系](#)

在此不再详述，其中AVIOContext是FFMPEG管理输入输出数据的结构体。本文将会详细分析一下该结构体里每个变量的含义和作用。

首先看一下结构体的定义（位于avio.h）：

```

1.  /* 雷霄骅
2.   * 中国传媒大学/数字电视技术
3.   * leixiaohua1020@126.com
4.   *
5.   */
6.  /**
7.   * Bytestream IO Context.
8.   * New fields can be added to the end with minor version bumps.
9.   * Removal, reordering and changes to existing fields require a major
10.  * version bump.
11.  * sizeof(AVIOContext) must not be used outside libav*.
12.  *
13.  * @note None of the function pointers in AVIOContext should be called
14.  *        directly, they should only be set by the client application
15.  *        when implementing custom I/O. Normally these are set to the
16.  *        function pointers specified in avio_alloc_context()
17.  */
18.  typedef struct {
19.      /**
20.       * A class for private options.
21.       *
22.       * If this AVIOContext is created by avio_open2(), av_class is set and
23.       * passes the options down to protocols.
24.       *
25.       * If this AVIOContext is manually allocated, then av_class may be set by
26.       * the caller.
27.       *
28.       * warning -- this field can be NULL, be sure to not pass this AVIOContext
29.       * to any av_opt_* functions in that case.
30.       */
31.      AVClass *av_class;
32.      unsigned char *buffer; /**< Start of the buffer. */
33.      int buffer_size; /**< Maximum buffer size */
34.      unsigned char *buf_ptr; /**< Current position in the buffer */
35.      unsigned char *buf_end; /**< End of the data, may be less than
36.                               buffer+buffer_size if the read function returned
37.                               less data than requested, e.g. for streams where
38.                               no more data has been received yet. */
39.      void *opaque; /**< A private pointer, passed to the read/write/seek/...
40.                      functions. */
41.      int (*read_packet)(void *opaque, uint8_t *buf, int buf_size);
42.      int (*write_packet)(void *opaque, uint8_t *buf, int buf_size);
43.      int64_t (*seek)(void *opaque, int64_t offset, int whence);
44.      int64_t pos; /**< position in the file of the current buffer */
45.      int must_flush; /**< true if the next seek should flush */
46.      int eof_reached; /**< true if eof reached */
47.      int write_flag; /**< true if open for writing */
48.      int max_packet_size;
49.      unsigned long checksum;
50.      unsigned char *checksum_ptr;
51.      unsigned long (*update_checksum)(unsigned long checksum, const uint8_t *buf, unsigned int size);
52.      int error; /**< contains the error code or 0 if no error happened */
53.      /**
54.       * Pause or resume playback for network streaming protocols - e.g. MMS.
55.       */
56.      int (*read_pause)(void *opaque, int pause);
57.      /**
58.       * Seek to a given timestamp in stream with the specified stream_index.
59.       * Needed for some network streaming protocols which don't support seeking
60.       * to byte position.
61.       */
62.      int64_t (*read_seek)(void *opaque, int stream_index,
63.                           int64_t timestamp, int flags);
64.      /**
65.       * A combination of AVIO_SEEKABLE_ flags or 0 when the stream is not seekable.
66.       */
67.      int seekable;
68.      /**
69.       * max filesize, used to limit allocations
70.       * This field is internal to libavformat and access from outside is not allowed.
71.       */
72.      int64_t maxsize;
73.  } AVIOContext;
74.

```

AVIOContext中有以下几个变量比较重要：

unsigned char *buffer：缓存开始位置

int buffer_size：缓存大小（默认32768）

unsigned char *buf_ptr：当前指针读取到的位置

unsigned char *buf_end：缓存结束的位置

void *opaque：URLContext结构体

在解码的情况下，buffer用于存储ffmpeg读入的数据。例如打开一个视频文件的时候，先把数据从硬盘读入buffer，然后在送给解码器用于解码。

其中opaque指向了URLContext。注意，这个结构体并不在FFMPEG提供的头文件中，而是在FFMPEG的源代码中。从FFMPEG源代码中翻出的定义如下所示：

```
[cpp]
1.  typedef struct URLContext {
2.      const AVClass *av_class; ///< information for av_log(). Set by url_open().
3.      struct URLProtocol *prot;
4.      int flags;
5.      int is_streamed; /**< true if streamed (no seek possible), default = false */
6.      int max_packet_size; /**< if non zero, the stream is packetized with this max packet size */
7.      void *priv_data;
8.      char *filename; /**< specified URL */
9.      int is_connected;
10.     AVIOInterruptCB interrupt_callback;
11. } URLContext;
```

URLContext结构体中还有一个结构体URLProtocol。注：每种协议（rtsp, rtmp, file等）对应一个URLProtocol。这个结构体也不在FFMPEG提供的头文件中。从FFMPEG源代码中翻出其的定义：

```
[cpp]
1.  typedef struct URLProtocol {
2.      const char *name;
3.      int (*url_open)(URLContext *h, const char *url, int flags);
4.      int (*url_read)(URLContext *h, unsigned char *buf, int size);
5.      int (*url_write)(URLContext *h, const unsigned char *buf, int size);
6.      int64_t (*url_seek)(URLContext *h, int64_t pos, int whence);
7.      int (*url_close)(URLContext *h);
8.      struct URLProtocol *next;
9.      int (*url_read_pause)(URLContext *h, int pause);
10.     int64_t (*url_read_seek)(URLContext *h, int stream_index,
11.                             int64_t timestamp, int flags);
12.     int (*url_get_file_handle)(URLContext *h);
13.     int priv_data_size;
14.     const AVClass *priv_data_class;
15.     int flags;
16.     int (*url_check)(URLContext *h, int mask);
17. } URLProtocol;
```



在这个结构体中，除了一些回调函数接口之外，有一个变量const char *name，该变量存储了协议的名称。每一种输入协议都对应这样一个结构体。比如说，文件协议中代码如下（file.c）：

```
[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. };
```

libRTMP中代码如下（libRTMP.c）：

```
[cpp]
1.  URLProtocol ff_rtmp_protocol = {
2.      .name           = "rtmp",
3.      .url_open       = rtmp_open,
4.      .url_read       = rtmp_read,
5.      .url_write      = rtmp_write,
6.      .url_close      = rtmp_close,
7.      .url_read_pause = rtmp_read_pause,
8.      .url_read_seek  = rtmp_read_seek,
9.      .url_get_file_handle = rtmp_get_file_handle,
10.     .priv_data_size  = sizeof(RTMP),
11.     .flags           = URL_PROTOCOL_FLAG_NETWORK,
12. };
```

udp协议代码如下（udp.c）：

```
[cpp]    
1.  URLProtocol ff_udp_protocol = {  
2.      .name          = "udp",  
3.      .url_open       = udp_open,  
4.      .url_read       = udp_read,  
5.      .url_write      = udp_write,  
6.      .url_close      = udp_close,  
7.      .url_get_file_handle = udp_get_file_handle,  
8.      .priv_data_size = sizeof(UDPContext),  
9.      .flags          = URL_PROTOCOL_FLAG_NETWORK,  
10. };
```

等号右边的函数是完成具体读写功能的函数。可以看一下file协议的几个函数（其实就是读文件，写文件这样的操作）（file.c）：

[cpp]  

```
1.  /*
2.  *雷霄骅
3.  *leixiaohua1020@126.com
4.  *中国传媒大学/数字电视技术
5.  */
6.  /* standard file protocol */
7.
8.  static int file_read(URLContext *h, unsigned char *buf, int size)
9.  {
10.     int fd = (intptr_t) h->priv_data;
11.     int r = read(fd, buf, size);
12.     return (-1 == r)?AERROR(errno):r;
13. }
14.
15. static int file_write(URLContext *h, const unsigned char *buf, int size)
16. {
17.     int fd = (intptr_t) h->priv_data;
18.     int r = write(fd, buf, size);
19.     return (-1 == r)?AERROR(errno):r;
20. }
21.
22. static int file_get_handle(URLContext *h)
23. {
24.     return (intptr_t) h->priv_data;
25. }
26.
27. static int file_check(URLContext *h, int mask)
28. {
29.     struct stat st;
30.     int ret = stat(h->filename, &st);
31.     if (ret < 0)
32.         return AERROR(errno);
33.
34.     ret |= st.st_mode&S_IRUSR ? mask&AVIO_FLAG_READ : 0;
35.     ret |= st.st_mode&S_IWUSR ? mask&AVIO_FLAG_WRITE : 0;
36.
37.     return ret;
38. }
39.
40. #if CONFIG_FILE_PROTOCOL
41.
42. static int file_open(URLContext *h, const char *filename, int flags)
43. {
44.     int access;
45.     int fd;
46.
47.     av_strstart(filename, "file:", &filename);
48.
49.     if (flags & AVIO_FLAG_WRITE && flags & AVIO_FLAG_READ) {
50.         access = O_CREAT | O_TRUNC | O_RDWR;
51.     } else if (flags & AVIO_FLAG_WRITE) {
52.         access = O_CREAT | O_TRUNC | O_WRONLY;
53.     } else {
54.         access = O_RDONLY;
55.     }
56.     #ifndef O_BINARY
57.     access |= O_BINARY;
58.     #endif
59.     fd = open(filename, access, 0666);
60.     if (fd == -1)
61.         return AERROR(errno);
62.     h->priv_data = (void *) (intptr_t) fd;
63.     return 0;
64. }
65.
66. /* XXX: use llseek */
67. static int64_t file_seek(URLContext *h, int64_t pos, int whence)
68. {
69.     int fd = (intptr_t) h->priv_data;
70.     if (whence == AVSEEK_SIZE) {
71.         struct stat st;
72.         int ret = fstat(fd, &st);
73.         return ret < 0 ? AERROR(errno) : st.st_size;
74.     }
75.     return lseek(fd, pos, whence);
76. }
77.
78. static int file_close(URLContext *h)
79. {
80.     int fd = (intptr_t) h->priv_data;
81.     return close(fd);
82. }
```

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/14215369>

文章标签：[ffmpeg](#) [aviocontext](#) [源代码](#) [视频](#) [IO](#)

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