# ⊕ FFMPEG结构体分析：AVIOContext

2013年11月08日 23:52:17　阅读数：25301

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）：

```cpp
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.      /**
70.       * max filesize, used to limit allocations
71.       * This field is internal to libavformat and access from outside is not allowed.
72.       */
73.      int64_t maxsize;
74.  } AVIOContext;
```

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

URLContext结构体中还有一个结构体URLProtocol。注：每种协议（rtp，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
typedef struct URLProtocol {
    const char *name;
    int (*url_open)(URLContext *h, const char *url, int flags);
    int (*url_read)(URLContext *h, unsigned char *buf, int size);
    int (*url_write)(URLContext *h, const unsigned char *buf, int size);
    int64_t (*url_seek)(URLContext *h, int64_t pos, int whence);
    int (*url_close)(URLContext *h);
    struct URLProtocol *next;
    int (*url_read_pause)(URLContext *h, int pause);
    int64_t (*url_read_seek)(URLContext *h, int stream_index,
        int64_t timestamp, int flags);
    int (*url_get_file_handle)(URLContext *h);
    int priv_data_size;
    const AVClass *priv_data_class;
    int flags;
    int (*url_check)(URLContext *h, int mask);
} 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
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,
};
```

libRTMP中代码如下（libRTMP.c）：

```cpp
URLProtocol ff_rtmp_protocol = {
    .name                = "rtmp",
    .url_open            = rtmp_open,
    .url_read            = rtmp_read,
    .url_write           = rtmp_write,
    .url_close           = rtmp_close,
    .url_read_pause      = rtmp_read_pause,
    .url_read_seek       = rtmp_read_seek,
    .url_get_file_handle = rtmp_get_file_handle,
    .priv_data_size      = sizeof(RTMP),
    .flags               = URL_PROTOCOL_FLAG_NETWORK,
};
```

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
/*
 *雷霄骅
 *leixiaohua1020@126.com
 *中国传媒大学/数字电视技术
 */
/* standard file protocol */

static int file_read(URLContext *h, unsigned char *buf, int size)
{
    int fd = (intptr_t) h->priv_data;
    int r = read(fd, buf, size);
    return (-1 == r)?AVERROR(errno):r;
}

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

static int file_get_handle(URLContext *h)
{
    return (intptr_t) h->priv_data;
}

static int file_check(URLContext *h, int mask)
{
    struct stat st;
    int ret = stat(h->filename, &st);
    if (ret < 0)
        return AVERROR(errno);

    ret |= st.st_mode&S_IRUSR ? mask&AVIO_FLAG_READ  : 0;
    ret |= st.st_mode&S_IWUSR ? mask&AVIO_FLAG_WRITE : 0;

    return ret;
}

#if CONFIG_FILE_PROTOCOL

static int file_open(URLContext *h, const char *filename, int flags)
{
    int access;
    int fd;

    av_strstart(filename, "file:", &filename);

    if (flags & AVIO_FLAG_WRITE && flags & AVIO_FLAG_READ) {
        access = O_CREAT | O_TRUNC | O_RDWR;
    } else if (flags & AVIO_FLAG_WRITE) {
        access = O_CREAT | O_TRUNC | O_WRONLY;
    } else {
        access = O_RDONLY;
    }
#ifdef O_BINARY
    access |= O_BINARY;
#endif
    fd = open(filename, access, 0666);
    if (fd == -1)
        return AVERROR(errno);
    h->priv_data = (void *) (intptr_t) fd;
    return 0;
}

/* XXX: use llseek */
static int64_t file_seek(URLContext *h, int64_t pos, int whence)
{
    int fd = (intptr_t) h->priv_data;
    if (whence == AVSEEK_SIZE) {
        struct stat st;
        int ret = fstat(fd, &st);
        return ret < 0 ? AVERROR(errno) : st.st_size;
    }
    return lseek(fd, pos, whence);
}

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

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

个人分类： FFMPEG

所属专栏： FFmpeg

---