

原 图解FFMPEG打开媒体的函数avformat_open_input

2013年03月11日 23:44:42 阅读数：57526

=====

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

FFMPEG打开媒体的的过程开始于avformat_open_input，因此该函数的重要性不可忽视。

在该函数中，FFMPEG完成了：

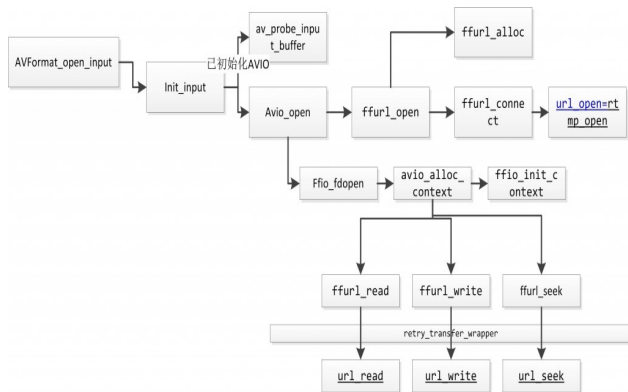
输入输出结构体AVIOContext的初始化；

输入数据的协议（例如RTMP，或者file）的识别（通过一套评分机制）：1判断文件名的后缀 2读取文件头的数据进行比对；

使用获得最高分的文件协议对应的URLProtocol，通过函数指针的方式，与FFMPEG连接（非专业用词）；

剩下的就是调用该URLProtocol的函数进行open,read等操作了

以下是通过eclipse+MinGW调试FFMPEG源代码获得的函数调用关系图



可见最终都调用了URLProtocol结构体中的函数指针。

URLProtocol结构如下，是一大堆函数指针的集合（avio.h文件）

```
[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;
```

URLProtocol功能就是完成各种输入协议的读写等操作

但输入协议种类繁多，它是怎样做到“大一统”的呢？

原来，每个具体的输入协议都有自己对应的URLProtocol。

比如file协议（FFMPEG把文件也当做一种特殊的协议）（file.c文件）

```
[cpp]
1.  URLProtocol ff_pipe_protocol = {
2.      .name          = "pipe",
3.      .url_open       = pipe_open,
4.      .url_read       = file_read,
5.      .url_write      = file_write,
6.      .url_get_file_handle = file_get_handle,
7.      .url_check      = file_check,
8.  };
```

或者rtmp协议（此处使用了librtmp）（librtmp.c文件）

```
[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. };
```

可见它们把各自的函数指针都赋值给了URLProtocol结构体的函数指针

因此avformat_open_input只需调用url_open,url_read这些函数就可以完成各种具体输入协议的open,read等操作了

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/8661601>

文章标签： [FFMPEG](#) [avformat_open_input](#) [解码](#) [函数分析](#)

个人分类： [FFMPEG](#)

所属专栏： [开源多媒体项目源代码分析](#) [FFmpeg](#)

此PDF由spygg生成,请尊重原作者版权!!!

我的邮箱:liushidc@163.com