源码解析（解码）1.打开媒体函数：avformat_open_input

- FFMPEG打开媒体的过程始于avformat_open_input

```
avformat_open_input(AVFormatContext **ps, const char *filename, AVInputFormat *fmt, AVDi
```

- 主要是对输入输出结构体AVFormatContext的初始化

```

```

- 通过判断文件名以及读取文件数据来确定数据流的类型
- 调用URLProtocol的函数进行open read的操作
- URLProtocol函数如下所示

```
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;
```

- 根据不同类型的数据类型来定义不同的URLProtocol对象

  如果是文件file类型则定义为

```
URLProtocol ff_pipe_protocol = {
    .name                = "pipe",
    .url_open            = pipe_open,
    .url_read            = file_read,
```

```
    .url_write           = file_write,
    .url_get_file_handle = file_get_handle,
    .url_check           = file_check,
};
```

如果是rtmp协议

```
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,
};
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
```

源码详解

函数声明位于libavformat\avformat.h

```
/**
 * Open an input stream and read the header. The codecs are not opened.
 * The stream must be closed with avformat_close_input().
 *
 * @param ps Pointer to user-supplied AVFormatContext (allocated by avformat_alloc_contex
 *           May be a pointer to NULL, in which case an AVFormatContext is allocated by t
 *           function and written into ps.
 *           Note that a user-supplied AVFormatContext will be freed on failure.
 * @param filename Name of the stream to open.
 * @param fmt If non-NULL, this parameter forces a specific input format.
 *            Otherwise the format is autodetected.
 * @param options  A dictionary filled with AVFormatContext and demuxer-private options.
 *                 On return this parameter will be destroyed and replaced with a dict co
 *                 options that were not found. May be NULL.
 *
 * @return 0 on success, a negative AVERROR on failure.
```

```
 *
 * @note If you want to use custom IO, preallocate the format context and set its pb fie
 */
int avformat_open_input(AVFormatContext **ps, const char *filename, AVInputFormat *fmt, A
```

这里拿中文简述把一下

```
ps：函数调用成功之后处理过的AVFormatContext结构体。
file：打开的视音频流的URL。
fmt：强制指定AVFormatContext中AVInputFormat的。这个参数一般情况下可以设置为NULL，这样FFmpeg可
dictionay：附加的一些选项，一般情况下可以设置为NULL。
```

函数调用关系

avformat_open_input()源代码比较长，一部分是一些容错代码。其中最重要的是以下两个函数

```
init_input()：绝大部分初始化工作都是在这里做的。
s->iformat->read_header()：读取多媒体数据文件头，根据视音频流创建相应的AVStream。
```

- init_input()函数:打开输入的视频数据，而且探测视频的格式，函数定义位于libavformat\utils.c

```
static int init_input(AVFormatContext *s, const char *filename,
                      AVDictionary **options)
{
    int ret;
    AVProbeData pd = { filename, NULL, 0 };
    int score = AVPROBE_SCORE_RETRY;

    if (s->pb) {
        s->flags |= AVFMT_FLAG_CUSTOM_IO;
        if (!s->iformat)
            return av_probe_input_buffer2(s->pb, &s->iformat, filename,
                                          s, 0, s->format_probesize);
        else if (s->iformat->flags & AVFMT_NOFILE)
            av_log(s, AV_LOG_WARNING, "Custom AVIOContext makes no sense and "
                                      "will be ignored with AVFMT_NOFILE format.\n");
        return 0;
    }

    if ((s->iformat && s->iformat->flags & AVFMT_NOFILE) ||
```

```
        (!s->iformat && (s->iformat = av_probe_input_format2(&pd, 0, &score))))
        return score;

    if ((ret = s->io_open(s, &s->pb, filename, AVIO_FLAG_READ | s->avio_flags, options)) < 0)
        return ret;

    if (s->iformat)
        return 0;
    return av_probe_input_buffer2(s->pb, &s->iformat, filename,
                                  s, 0, s->format_probesize);
}
```

逻辑分析

score变量是一个判决视频输入格式（AVInputFormat）分数的界限值，根据文件数据的head数据来给分数。如果最后得到的AVInputFormat的分数低于该门限值，就认为没有找到合适的AVInputFormat。FFmpeg内部判断封装格式的原理实际上是对每种AVInputFormat给出一个分数，满分是100分，越有可能正确的AVInputFormat给出的分数就越高。最后选择分数最高的AVInputFormat作为推测结果。score的值是一个宏定义AVPROBE_SCORE_RETRY，我们可以看一下它的定义

```
#define AVPROBE_SCORE_RETRY (AVPROBE_SCORE_MAX/4);  //AVPROBE_SCORE_MAX为100
```

整个函数的逻辑

1）当使用了自定义的AVIOContext的时候（AVFormatContext中的AVIOContext不为空，即s->pb!=NULL），如果指定了AVInputFormat就直接返回，如果没有指定就调av_probe_input_buffer2()推测AVInputFormat。这一情况出现的不算很多，但是当我们从内存中读取数据的时候（需要初始化自定义的AVIOContext），就会执行这一步骤。

2）在更一般的情况下，如果已经指定了AVInputFormat，就直接返回；如果没有指定AVInputFormat，就调用av_probe_input_format(NULL,...)根据文件路径判断文件格式。这里特意把av_probe_input_format()的第1个参数写成“NULL”，是为了强调这个时候实际上并没有给函数提供输入数据，此时仅仅通过文件路径推测AVInputFormat。

3）如果发现通过文件路径判断不出来文件格式，那么就需要打开文件探测文件格式了，这个时候会首先调用avio_open2()打开文件，然后调用av_probe_input_buffer2()推测AVInputFormat。

AVInputFormat-> read_header()

- 读取媒体文件的头文件，不同的封装格式会调用不同的read_header函数
- 如果检测到媒体文件中包含了视频流或者音频流，就会调用create_stream函数