

转 FFMPEG源码分析：avformat_open_input()（媒体打开函数）

2013年09月22日 02:11:02 阅读数：18035

本文分析了FFMPEG中的媒体打开函数avformat_open_input()

```
[cpp]
1. //参数ps包含一切媒体相关的上下文结构，有它就有了一切，本函数如果打开媒体成功，
2. //会返回一个AVFormatContext的实例。
3. //参数filename是媒体文件名或URL。
4. //参数fmt是要打开的媒体格式的操作结构，因为是读，所以是inputFormat。此处可以
5. //传入一个调用者定义的inputFormat，对应命令行中的 -f xxx段，如果指定了它，
6. //在打开文件中就不会探测文件的实际格式了，以它为准了。
7. //参数options是对某种格式的一些操作，是为了在命令行中可以对不同的格式传入
8. //特殊的操作参数而建的， 为了了解流程，完全可以无视它。
9. int avformat_open_input(AVFormatContext **ps,
10.     const char *filename,
11.     AVInputFormat *fmt,
12.     AVDictionary **options)
13. {
14.     AVFormatContext *s = *ps;
15.     int ret = 0;
16.     AVFormatParameters ap = { { 0 } };
17.     AVDictionary *tmp = NULL;
18.
19.     //创建上下文结构
20.     if (!s && !(s = avformat_alloc_context()))
21.         return AVERROR(ENOMEM);
22.     //如果用户指定了输入格式，直接使用它
23.     if (fmt)
24.         s->iformat = fmt;
25.
26.     //忽略
27.     if (options)
28.         av_dict_copy(&tmp, *options, 0);
29.
30.     if ((ret = av_opt_set_dict(s, &tmp)) < 0)
31.         goto fail;
32.
33.     //打开输入媒体（如果需要的话），初始化所有与媒体读写有关的结构们，比如
34.     //AVIOContext, AVInputFormat等等
35.     if ((ret = init_input(s, filename)) < 0)
36.         goto fail;
37.     //执行完此函数后，s->pb和s->iformat都已经指向了有效实例。pb是用于读写数据的，它
38.     //把媒体数据当做流来读写，不管是什么媒体格式，而iformat把pb读出来的流按某种媒体格
39.     //式进行分析，也就是说pb在底层，iformat在上层。
40.
41.     //很多静态图像文件格式，都被当作一个格式处理，比如要打开.jpeg文件，需要的格式
42.     //名为image2。此处还不是很了解具体细节，作不得准哦。
43.     /* check filename in case an image number is expected */
44.     if (s->iformat->flags & AVFMT_NEEDNUMBER) {
45.         if (!av_filename_number_test(filename)) {
46.             ret = AVERROR(EINVAL);
47.             goto fail;
48.         }
49.     }
50.
51.     s->duration = s->start_time = AV_NOPTS_VALUE;
52.     //上下文中保存下文件名
53.     av_strlcpy(s->filename, filename, sizeof(s->filename));
54.
55.     /* allocate private data */
56.     //为当前格式分配私有数据，主要用于某格式的读写操作时所用的私有结构。
57.     //此结构的大小在定义AVInputFormat时已指定了。
58.     if (s->iformat->priv_data_size > 0) {
59.         if (!(s->priv_data = av_mallocz(s->iformat->priv_data_size))) {
60.             ret = AVERROR(ENOMEM);
61.             goto fail;
62.         }
63.         //这个可以先不管它
64.         if (s->iformat->priv_class) {
65.             *(const AVClass**) s->priv_data = s->iformat->priv_class;
66.             av_opt_set_defaults(s->priv_data);
67.             if ((ret = av_opt_set_dict(s->priv_data, &tmp)) < 0)
68.                 goto fail;
69.         }
70.     }
71.
72.     /* e.g. AVFMT_NOFILE formats will not have a AVIOContext */
73.     //从mp3文件中读ID3数据并保存之。
74.     if (s->pb)
75.         ff_id3v2_read(s, ID3v2_DEFAULT_MAGIC);
76.
77.     //读一下媒体的头部，在read_header()中主要是做某种格式的初始化工作，比如填充自己的
78.     //私有结构，根据流的数量分配流结构并初始化，把文件指针指向数据区开始处等。
79.     if (!(s->flags & AVFMT_FLAG_PRIV_OPT) && s->iformat->read_header)
```

```

80.         if ((ret = s->iformat->read_header(s, &ap)) < 0)
81.             goto fail;
82.
83.         //保存数据区开始的位置
84.         if (!(s->flags & AVFMT_FLAG_PRIV_OPT) && s->pb && !s->data_offset)
85.             s->data_offset = avio_tell(s->pb);
86.
87.         s->raw_packet_buffer_remaining_size = RAW_PACKET_BUFFER_SIZE;
88.
89.         if (options) {
90.             av_dict_free(options);
91.             *options = tmp;
92.         }
93.         *ps = s;
94.         //执行成功
95.         return 0;
96.
97.         //执行失败
98.         fail: av_dict_free(&tmp);
99.         if (s->pb && !(s->flags & AVFMT_FLAG_CUSTOM_IO))
100.             avio_close(s->pb);
101.         avformat_free_context(s);
102.         *ps = NULL;
103.         return ret;
104.     }

```

init_input

```

[cpp]
1. //打开输入媒体并填充其AVInputFormat结构
2. static int init_input(AVFormatContext *s, const char *filename)
3. {
4.     int ret;
5.     AVProbeData pd = { filename, NULL, 0 };
6.
7.     //当调用者已指定了pb（数据取得的方式）——一般不会这样。
8.     if (s->pb) {
9.         s->flags |= AVFMT_FLAG_CUSTOM_IO;
10.        if (!s->iformat)
11.            //如果已指定了pb但没指定iformat，以pb读取媒体数据进行探测，取得。取得iformat。
12.            return av_probe_input_buffer(s->pb, &s->iformat, filename, s, 0, 0);
13.        else if (s->iformat->flags & AVFMT_NOFILE)
14.            //如果已指定pb也指定了iformat，但是又指定了不需要文件（也包括URL指定的地址），这就矛盾了，
15.            //此时应是不需要pb的，因为不需操作文件，提示一下吧，也不算错。
16.            av_log(s, AV_LOG_WARNING, "Custom AVIOContext makes no sense and "
17.                "will be ignored with AVFMT_NOFILE format.\n");
18.        return 0;
19.    }
20.
21.    //一般会执行到这里
22.    if ((s->iformat && s->iformat->flags & AVFMT_NOFILE)
23.        || (!s->iformat && (s->iformat = av_probe_input_format(&pd, 0))))
24.        //如果已指定了iformat并且不需要文件，也就不需要pb了，可以直接返回
25.        //如果没指定iformat，但是可以从文件名中猜出iformat，也成功。
26.        return 0;
27.
28.    //如果从文件名中也猜不出媒体格式，则只能打开这个文件进行探测了，先打开文件
29.    if ((ret = avio_open(&s->pb, filename, AVIO_FLAG_READ)) < 0)
30.        return ret;
31.    if (s->iformat)
32.        return 0;
33.    //再探测之
34.    return av_probe_input_buffer(s->pb, &s->iformat, filename, s, 0, 0);
35. }

```

avio_open

[cpp]  

```
1. //打开一个地址指向的媒体
2. int avio_open(AVIOContext **s, const char *filename, int flags)
3. {
4.     //URLContext代表一个URL地址指向的媒体文件，本地路径也算一种。它封装了
5.     //操作一个媒体文件的相关数据，最重要的是prot变量，是URLProtocol型的。
6.     //prot代表一个特定的协议和协议操作函数们，URLContext包含不同的prot，
7.     //就可以通过URLContext使用不同的协议读写媒体数据，比如tcp,http，本地
8.     //文件用file协议。
9.     URLContext *h;
10.    int err;
11.
12.    //创建并初始化URLContext，其prot通过文件名确定。然后打开这个媒体文件
13.    err = ffurl_open(&h, filename, flags);
14.    if (err < 0)
15.        return err;
16.    //其实文件已经在上边真正打开了。这里只是填充AVIOContext。使它记录下
17.    //URLContext，以及填充读写数据的函数指针。
18.    err = ffio_fdopen(s, h);
19.    if (err < 0) {
20.        ffurl_close(h);
21.        return err;
22.    }
23.    return 0;
24. }
```

av_probe_input_buffer

[cpp]  

```
1. int av_probe_input_buffer(AVIOContext *pb,
2.    AVInputFormat **fmt,
3.    const char *filename,
4.    void *logctx,
5.    unsigned int offset,
6.    unsigned int max_probe_size)
7. {
8.    AVProbeData pd = { filename ? filename : "", NULL, -offset };
9.    unsigned char *buf = NULL;
10.   int ret = 0, probe_size;
11.
12.   //计算最多探测数据的字节数
13.   if (!max_probe_size) {
14.       max_probe_size = PROBE_BUF_MAX;
15.   } else if (max_probe_size > PROBE_BUF_MAX) {
16.       max_probe_size = PROBE_BUF_MAX;
17.   } else if (max_probe_size < PROBE_BUF_MIN) {
18.       return AVERROR(EINVAL);
19.   }
20.
21.   if (offset >= max_probe_size) {
22.       return AVERROR(EINVAL);
23.   }
24.
25.   //循环直到探测完指定的数据
26.   for (probe_size = PROBE_BUF_MIN;
27.       probe_size <= max_probe_size && !*fmt;
28.       probe_size =
29.           FFMIN(probe_size<<1, FFMAX(max_probe_size, probe_size+1))) {
30.       int score = probe_size < max_probe_size ? AVPROBE_SCORE_MAX / 4 : 0;
31.       int buf_offset = (probe_size == PROBE_BUF_MIN) ? 0 : probe_size >> 1;
32.       void *buftmp;
33.
34.       if (probe_size < offset) {
35.           continue;
36.       }
37.
38.       /* read probe data */
39.       //分配读取数据存放的缓冲
40.       buftmp = av_realloc(buf, probe_size + AVPROBE_PADDING_SIZE);
41.       if (!buftmp) {
42.           av_free(buf);
43.           return AVERROR(ENOMEM);
44.       }
45.       buf = buftmp;
46.       //利用pb读数据到缓冲的剩余空间中
47.       if ((ret = avio_read(pb, buf + buf_offset, probe_size - buf_offset))
48.           < 0) {
49.           /* fail if error was not end of file, otherwise, lower score */
50.           if (ret != AVERROR_EOF) {
51.               av_free(buf);
52.               return ret;
53.           }
54.           score = 0;
55.           ret = 0; /* error was end of file, nothing read */
56.       }
```

```

57.     pd.buf_size += ret;
58.     pd.buf = &buf[offset];
59.
60.     //缓冲中没有数据的部分要清0
61.     memset(pd.buf + pd.buf_size, 0, AVPROBE_PADDING_SIZE);
62.
63.     /* guess file format */
64.     //从一个打开的文件只探测媒体格式
65.     *fmt = av_probe_input_format2(&pd, 1, &score);
66.     if (*fmt) {
67.         if (score <= AVPROBE_SCORE_MAX / 4) { //this can only be true in the last iteration
68.             av_log(
69.                 logctx,
70.                 AV_LOG_WARNING,
71.                 "Format %s detected only with low score of %d, misdetection possible!\n",
72.                 (*fmt)->name, score);
73.         } else
74.             av_log(logctx, AV_LOG_DEBUG,
75.                 "Format %s probed with size=%d and score=%d\n",
76.                 (*fmt)->name, probe_size, score);
77.     }
78.     //不成功, 继续
79. }
80.
81. if (!*fmt) {
82.     av_free(buf);
83.     return AVERROR_INVALIDDATA;
84. }
85.
86. /* rewind. reuse probe buffer to avoid seeking */
87. //把探测时读入的数据保存到pb中, 为的是真正读时直接利用之.
88. if ((ret = ffio_rewind_with_probe_data(pb, buf, pd.buf_size)) < 0)
89.     av_free(buf);
90.
91. return ret;
92. }

```

原文地址：<http://wodamazi.iteye.com/blog/1293994>

文章标签：[ffmpeg](#) [打开](#) [avformat_open_input](#) [源码](#)

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