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：18008

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

```cpp
//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，
//会返回一个AVFormatContext的实例.
//参数filename是媒体文件名或URL.
//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
//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，
//在打开文件中就不会探测文件的实际格式了，以它为准了.
//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
//特殊的操作参数而建的， 为了了解流程，完全可以无视它.
int avformat_open_input(AVFormatContext **ps,
        const char *filename,
        AVInputFormat *fmt,
        AVDictionary **options)
{
    AVFormatContext *s = *ps;
    int ret = 0;
    AVFormatParameters ap = { { 0 } };
    AVDictionary *tmp = NULL;

    //创建上下文结构
    if (!s && !(s = avformat_alloc_context()))
        return AVERROR(ENOMEM);
    //如果用户指定了输入格式，直接使用它
    if (fmt)
        s->iformat = fmt;

    //忽略
    if (options)
        av_dict_copy(&tmp, *options, 0);

    if ((ret = av_opt_set_dict(s, &tmp)) < 0)
        goto fail;

    //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
    //AVIOContext，AVInputFormat等等
    if ((ret = init_input(s, filename)) < 0)
        goto fail;
    //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
    //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
    //式进行分析，也就是说pb在底层，iformat在上层.

    //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
    //名为image2. 此处还不是很了解具体细节，作不得准哦.
    /* check filename in case an image number is expected */
    if (s->iformat->flags & AVFMT_NEEDNUMBER) {
        if (!av_filename_number_test(filename)) {
            ret = AVERROR(EINVAL);
            goto fail;
        }
    }

    s->duration = s->start_time = AV_NOPTS_VALUE;
    //上下文中保存下文件名
    av_strlcpy(s->filename, filename, sizeof(s->filename));

    /* allocate private data */
    //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.
    //此结构的大小在定义AVInputFormat时已指定了.
    if (s->iformat->priv_data_size > 0) {
        if (!(s->priv_data = av_mallocz(s->iformat->priv_data_size))) {
            ret = AVERROR(ENOMEM);
            goto fail;
        }
        //这个可以先不必管它
        if (s->iformat->priv_class) {
            *(const AVClass**) s->priv_data = s->iformat->priv_class;
            av_opt_set_defaults(s->priv_data);
            if ((ret = av_opt_set_dict(s->priv_data, &tmp)) < 0)
                goto fail;
        }
    }

    /* e.g. AVFMT_NOFILE formats will not have a AVIOContext */
    //从mp3文件中读ID3数据并保存之.
    if (s->pb)
        ff_id3v2_read(s, ID3v2_DEFAULT_MAGIC);

    //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
    //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.
    if (!(s->flags & AVFMT_FLAG_PRIV_OPT) && s->iformat->read_header)
```

```
        if ((ret = s->iformat->read_header(s, &ap)) < 0)
            goto fail;

    //保存数据区开始的位置
    if (!(s->flags & AVFMT_FLAG_PRIV_OPT) && s->pb && !s->data_offset)
        s->data_offset = avio_tell(s->pb);

    s->raw_packet_buffer_remaining_size = RAW_PACKET_BUFFER_SIZE;

    if (options) {
        av_dict_free(options);
        *options = tmp;
    }
    *ps = s;
    //执行成功
    return 0;

    //执行失败
    fail: av_dict_free(&tmp);
    if (s->pb && !(s->flags & AVFMT_FLAG_CUSTOM_IO))
        avio_close(s->pb);
    avformat_free_context(s);
    *ps = NULL;
    return ret;
}
```

init_input

```cpp
//打开输入媒体并填充其AVInputFormat结构
static int init_input(AVFormatContext *s, const char *filename)
{
    int ret;
    AVProbeData pd = { filename, NULL, 0 };

    //当调用者已指定了pb（数据取得的方式）－－一般不会这样.
    if (s->pb) {
        s->flags |= AVFMT_FLAG_CUSTOM_IO;
        if (!s->iformat)
            //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.
            return av_probe_input_buffer(s->pb, &s->iformat, filename, s, 0, 0);
        else if (s->iformat->flags & AVFMT_NOFILE)
            //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，
            //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.
            av_log(s, AV_LOG_WARNING, "Custom AVIOContext makes no sense and "
                   "will be ignored with AVFMT_NOFILE format.\n");
        return 0;
    }

    //一般会执行到这里
    if ((s->iformat && s->iformat->flags & AVFMT_NOFILE)
            || (!s->iformat && (s->iformat = av_probe_input_format(&pd, 0))))
        //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
        //如果没指定iformat，但是可以从文件名中猜出iformat，也成功.
        return 0;

    //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
    if ((ret = avio_open(&s->pb, filename, AVIO_FLAG_READ)) < 0)
        return ret;
    if (s->iformat)
        return 0;
    //再探测之
    return av_probe_input_buffer(s->pb, &s->iformat, filename, s, 0, 0);
}
```

avio_open

```cpp
//打开一个地址指向的媒体
int avio_open(AVIOContext **s, const char *filename, int flags)
{
    //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
    //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．
    //prot代表一个特定的协义和协议操作函数们，URLContext包含不同的prot，
    //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
    //文件用file协议．
    URLContext *h;
    int err;

    //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
    err = ffurl_open(&h, filename, flags);
    if (err < 0)
        return err;
    //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
    //URLContext，以及填充读写数据的函数指针．
    err = ffio_fdopen(s, h);
    if (err < 0) {
        ffurl_close(h);
        return err;
    }
    return 0;
}
```

av_probe_input_buffer

```cpp
int av_probe_input_buffer(AVIOContext *pb,
        AVInputFormat **fmt,
        const char *filename,
        void *logctx,
        unsigned int offset,
        unsigned int max_probe_size)
{
    AVProbeData pd = { filename ? filename : "", NULL, -offset };
    unsigned char *buf = NULL;
    int ret = 0, probe_size;

    //计算最多探测数据的字节数
    if (!max_probe_size) {
        max_probe_size = PROBE_BUF_MAX;
    } else if (max_probe_size > PROBE_BUF_MAX) {
        max_probe_size = PROBE_BUF_MAX;
    } else if (max_probe_size < PROBE_BUF_MIN) {
        return AVERROR(EINVAL);
    }

    if (offset >= max_probe_size) {
        return AVERROR(EINVAL);
    }

    //循环直到探测完指定的数据
    for (probe_size = PROBE_BUF_MIN;
            probe_size <= max_probe_size && !*fmt;
            probe_size =
                    FFMIN(probe_size<<1, FFMAX(max_probe_size, probe_size+1))) {
        int score = probe_size < max_probe_size ? AVPROBE_SCORE_MAX / 4 : 0;
        int buf_offset = (probe_size == PROBE_BUF_MIN) ? 0 : probe_size >> 1;
        void *buftmp;

        if (probe_size < offset) {
            continue;
        }

        /* read probe data */
        //分配读取数据存放的缓冲
        buftmp = av_realloc(buf, probe_size + AVPROBE_PADDING_SIZE);
        if (!buftmp) {
            av_free(buf);
            return AVERROR(ENOMEM);
        }
        buf = buftmp;
        //利用pb读数据到缓冲的剩余空间中
        if ((ret = avio_read(pb, buf + buf_offset, probe_size - buf_offset))
                < 0) {
            /* fail if error was not end of file, otherwise, lower score */
            if (ret != AVERROR_EOF) {
                av_free(buf);
                return ret;
            }
            score = 0;
            ret = 0; /* error was end of file, nothing read */
        }
```

```
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