原 FFmpeg源代码简单分析：avformat_alloc_output_context2()

2015年03月03日 22:13:57　阅读数：23196

==========

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

==========

本文简单分析FFmpeg中常用的一个函数：avformat_alloc_output_context2()。在基于FFmpeg的视音频编码器程序中，该函数通常是第一个调用的函数（除了组件注册函数av_register_all()）。avformat_alloc_output_context2()函数可以初始化一个用于输出的AVFormatContext结构体。它的声明位于libavformat\avformat.h，如下所示。

```cpp
/**
 * Allocate an AVFormatContext for an output format.
 * avformat_free_context() can be used to free the context and
 * everything allocated by the framework within it.
 *
 * @param *ctx is set to the created format context, or to NULL in
 * case of failure
 * @param oformat format to use for allocating the context, if NULL
 * format_name and filename are used instead
 * @param format_name the name of output format to use for allocating the
 * context, if NULL filename is used instead
 * @param filename the name of the filename to use for allocating the
 * context, may be NULL
 * @return >= 0 in case of success, a negative AVERROR code in case of
 * failure
 */
int avformat_alloc_output_context2(AVFormatContext **ctx, AVOutputFormat *oformat,
                                   const char *format_name, const char *filename);
```

代码中的英文注释写的已经比较详细了，在这里拿中文简单叙述一下。

ctx：函数调用成功之后创建的AVFormatContext结构体。

oformat：指定AVFormatContext中的AVOutputFormat，用于确定输出格式。如果指定为NULL，可以设定后两个参数（format_name或者filename）由FFmpeg猜测输出格式。

PS：使用该参数需要自己手动获取AVOutputFormat，相对于使用后两个参数来说要麻烦一些。

format_name：指定输出格式的名称。根据格式名称，FFmpeg会推测输出格式。输出格式可以是“flv”，“mkv”等等。

filename：指定输出文件的名称。根据文件名称，FFmpeg会推测输出格式。文件名称可以是“xx.flv”，“yy.mkv”等等。

函数执行成功的话，其返回值大于等于0。

该函数最典型的例子可以参考： 最简单的基于FFMPEG的视频编码器（YUV编码为H.264）

函数调用结构图

首先贴出来最终分析得出的函数调用结构图，如下所示。

单击查看更清晰图片

avformat_alloc_output_context2()

下面看一下avformat_alloc_output_context2()的函数定义。该函数的定义位于libavformat\mux.c中，如下所示。

```cpp
int avformat_alloc_output_context2(AVFormatContext **avctx, AVOutputFormat *oformat,
                                   const char *format, const char *filename)
{
    AVFormatContext *s = avformat_alloc_context();
    int ret = 0;


    *avctx = NULL;
    if (!s)
        goto nomem;


    if (!oformat) {
        if (format) {
            oformat = av_guess_format(format, NULL, NULL);
            if (!oformat) {
                av_log(s, AV_LOG_ERROR, "Requested output format '%s' is not a suitable output format\n", format);
                ret = AVERROR(EINVAL);
                goto error;
            }
        } else {
            oformat = av_guess_format(NULL, filename, NULL);
            if (!oformat) {
                ret = AVERROR(EINVAL);
                av_log(s, AV_LOG_ERROR, "Unable to find a suitable output format for '%s'\n",
                       filename);
                goto error;
            }
        }
    }


    s->oformat = oformat;
    if (s->oformat->priv_data_size > 0) {
        s->priv_data = av_mallocz(s->oformat->priv_data_size);
        if (!s->priv_data)
            goto nomem;
        if (s->oformat->priv_class) {
            *(const AVClass**)s->priv_data= s->oformat->priv_class;
            av_opt_set_defaults(s->priv_data);
        }
    } else
        s->priv_data = NULL;


    if (filename)
        av_strlcpy(s->filename, filename, sizeof(s->filename));
    *avctx = s;
    return 0;
nomem:
    av_log(s, AV_LOG_ERROR, "Out of memory\n");
    ret = AVERROR(ENOMEM);
error:
    avformat_free_context(s);
    return ret;
}
```

从代码中可以看出，avformat_alloc_output_context2()的流程如要包含以下2步：

1) 调用avformat_alloc_context()初始化一个默认的AVFormatContext。

2) 如果指定了输入的AVOutputFormat，则直接将输入的AVOutputFormat赋值给AVOutputFormat的oformat。如果没有指定输入的AVOutputFormat，就需要根据文件格式名称或者文件名推测输出的AVOutputFormat。无论是通过文件格式名称还是文件名推测输出格式，都会调用一个函数av_guess_format()。

下面我们分别看看上文步骤中提到的两个重要的函数：avformat_alloc_context()和av_guess_format()。

avformat_alloc_context()

avformat_alloc_context()的是一个FFmpeg的API，它的定义如下。

```cpp
AVFormatContext *avformat_alloc_context(void)
{
    AVFormatContext *ic;
    ic = av_malloc(sizeof(AVFormatContext));
    if (!ic) return ic;
    avformat_get_context_defaults(ic);


    ic->internal = av_mallocz(sizeof(*ic->internal));
    if (!ic->internal) {
        avformat_free_context(ic);
        return NULL;
    }


    return ic;
}
```

从代码中可以看出，avformat_alloc_context()首先调用av_malloc()为AVFormatContext分配一块内存。然后调用了一个函数avformat_get_context_defaults()用于给AVFormatContext设置默认值。avformat_get_context_defaults()的定义如下。

```cpp
static void avformat_get_context_defaults(AVFormatContext *s)
{
    memset(s, 0, sizeof(AVFormatContext));


    s->av_class = &av_format_context_class;


    av_opt_set_defaults(s);
}
```

从代码中可以看出，avformat_alloc_context()首先调用memset()将AVFormatContext的内存置零；然后指定它的AVClass（指定了AVClass之后，该结构体就支持和AVOption相关的功能）；最后调用av_opt_set_defaults()给AVFormatContext的成员变量设置默认值（av_opt_set_defaults()就是和AVOption有关的一个函数，专门用于给指定的结构体设定默认值，此处暂不分析）。

av_guess_format()

av_guess_format()是FFmpeg的一个API。它的声明如下。

```cpp
/**
 * Return the output format in the list of registered output formats
 * which best matches the provided parameters, or return NULL if
 * there is no match.
 *
 * @param short_name if non-NULL checks if short_name matches with the
 * names of the registered formats
 * @param filename if non-NULL checks if filename terminates with the
 * extensions of the registered formats
 * @param mime_type if non-NULL checks if mime_type matches with the
 * MIME type of the registered formats
 */
AVOutputFormat *av_guess_format(const char *short_name,
                                const char *filename,
                                const char *mime_type);
```

拿中文简单解释一下参数。

short_name：格式的名称。

filename：文件的名称。

mime_type：MIME类型。

返回最匹配的AVOutputFormat。如果没有很匹配的AVOutputFormat，则返回NULL。

av_guess_format()的代码如下所示。

```cpp
AVOutputFormat *av_guess_format(const char *short_name, const char *filename,
                                const char *mime_type)
{
    AVOutputFormat *fmt = NULL, *fmt_found;
    int score_max, score;


    /* specific test for image sequences */
#if CONFIG_IMAGE2_MUXER
    if (!short_name && filename &&
        av_filename_number_test(filename) &&
        ff_guess_image2_codec(filename) != AV_CODEC_ID_NONE) {
        return av_guess_format("image2", NULL, NULL);
    }
#endif
    /* Find the proper file type. */
    fmt_found = NULL;
    score_max = 0;
    while ((fmt = av_oformat_next(fmt))) {
        score = 0;
        if (fmt->name && short_name && av_match_name(short_name, fmt->name))
            score += 100;
        if (fmt->mime_type && mime_type && !strcmp(fmt->mime_type, mime_type))
            score += 10;
        if (filename && fmt->extensions &&
            av_match_ext(filename, fmt->extensions)) {
            score += 5;
        }
        if (score > score_max) {
            score_max = score;
            fmt_found = fmt;
        }
    }
    return fmt_found;
}
```

从代码中可以看出，av_guess_format()中使用一个整型变量score记录每种输出格式的匹配程度。函数中包含了一个while()循环，该循环利用函数av_oformat_next()遍历FFmpeg中所有的AVOutputFormat，并逐一计算每个输出格式的score。具体的计算过程分成如下几步：

1) 如果封装格式名称匹配，score增加100。匹配中使用了函数av_match_name()。

2) 如果mime类型匹配，score增加10。匹配直接使用字符串比较函数strcmp()。

3) 如果文件名称的后缀匹配，score增加5。匹配中使用了函数av_match_ext()。

while()循环结束后，得到得分最高的格式，就是最匹配的格式。

下面看一下一个AVOutputFormat的实例，就可以理解“封装格式名称”，“mine类型”，“文件名称后缀”这些概念了。下面是flv格式的视音频复用器（Muxer）对应的AVOutputFormat格式的变量ff_flv_muxer。

```cpp
AVOutputFormat ff_flv_muxer = {
    .name           = "flv",
    .long_name      = NULL_IF_CONFIG_SMALL("FLV (Flash Video)"),
    .mime_type      = "video/x-flv",
    .extensions     = "flv",
    .priv_data_size = sizeof(FLVContext),
    .audio_codec    = CONFIG_LIBMP3LAME ? AV_CODEC_ID_MP3 : AV_CODEC_ID_ADPCM_SWF,
    .video_codec    = AV_CODEC_ID_FLV1,
    .write_header   = flv_write_header,
    .write_packet   = flv_write_packet,
    .write_trailer  = flv_write_trailer,
    .codec_tag      = (const AVCodecTag* const []) {
                          flv_video_codec_ids, flv_audio_codec_ids, 0
                      },
    .flags          = AVFMT_GLOBALHEADER | AVFMT_VARIABLE_FPS |
                      AVFMT_TS_NONSTRICT,
};
```

下面看看av_guess_format()匹配最佳格式的过程中涉及到的几个函数。

av_oformat_next()

av_oformat_next()是个API函数，声明如下所示。

```cpp
/**
 * If f is NULL, returns the first registered output format,
 * if f is non-NULL, returns the next registered output format after f
 * or NULL if f is the last one.
 */
AVOutputFormat *av_oformat_next(const AVOutputFormat *f);
```

av_oformat_next()参数不为NULL的时候用于获得下一个AVOutputFormat，否则获得第一个AVOutputFormat。定义如下。

```cpp
AVOutputFormat *av_oformat_next(const AVOutputFormat *f)
{
    if (f)
        return f->next;
    else
        return first_oformat;
}
```

av_match_name()

av_match_name()是一个API函数，声明如下所示。

```cpp
/**
 * Match instances of a name in a comma-separated list of names.
 * @param name  Name to look for.
 * @param names List of names.
 * @return 1 on match, 0 otherwise.
 */
int av_match_name(const char *name, const char *names);
```

av_match_name()用于比较两个格式的名称。简单地说就是比较字符串。注意该函数的字符串是不区分大小写的：字符都转换为小写进行比较。

```cpp
int av_match_name(const char *name, const char *names)
{
    const char *p;
    int len, namelen;


    if (!name || !names)
        return 0;


    namelen = strlen(name);
    while ((p = strchr(names, ','))) {
        len = FFMAX(p - names, namelen);
        if (!av_strncasecmp(name, names, len))
            return 1;
        names = p + 1;
    }
    return !av_strcasecmp(name, names);
}
```

上述函数还有一点需要注意，其中使用了一个while()循环，用于搜索“,”。这是因为FFmpeg中有些格式是对应多种格式名称的，例如MKV格式的解复用器（Demuxer）的定义如下。

```cpp
AVInputFormat ff_matroska_demuxer = {
    .name           = "matroska,webm",
    .long_name      = NULL_IF_CONFIG_SMALL("Matroska / WebM"),
    .extensions     = "mkv,mk3d,mka,mks",
    .priv_data_size = sizeof(MatroskaDemuxContext),
    .read_probe     = matroska_probe,
    .read_header    = matroska_read_header,
    .read_packet    = matroska_read_packet,
    .read_close     = matroska_read_close,
    .read_seek      = matroska_read_seek,
    .mime_type      = "audio/webm,audio/x-matroska,video/webm,video/x-matroska"
};
```

从代码可以看出，ff_matroska_demuxer中的name字段对应“matroska,webm”。av_match_name()函数对于这样的字符串，会把它按照“,”截断成一个个封装格式名称，然后一一进行比较。

av_match_ext()

av_match_ext()是一个API函数，声明如下所示。

```cpp
/**
 * Return a positive value if the given filename has one of the given
 * extensions, 0 otherwise.
 *
 * @param filename   file name to check against the given extensions
 * @param extensions a comma-separated list of filename extensions
 */
int av_match_ext(const char *filename, const char *extensions);
```

av_match_ext()用于比较文件的后缀。该函数首先通过反向查找的方式找到输入文件名中的“.”，就可以通过获取“.”后面的字符串来得到该文件的后缀。然后调用av_match_name()，采用和比较格式名称的方法比较两个后缀。

```cpp
int av_match_ext(const char *filename, const char *extensions)
{
    const char *ext;


    if (!filename)
        return 0;


    ext = strrchr(filename, '.');
    if (ext)
        return av_match_name(ext + 1, extensions);
    return 0;
}
```

经过以上几步之后，av_guess_format()最终可以得到最合适的AVOutputFormat并且返回给avformat_alloc_output_context2()。avformat_alloc_output_context2()接下来将获得的AVOutputFormat赋值给刚刚新建的AVFormatContext，即可完成初始化工作。

雷霄骅 (Lei Xiaohua)
leixiaohua1020@126.com
http://blog.csdn.net/leixiaohua1020

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/41198929

文章标签： FFMpeg 输出 源代码分析 初始化 AVFormatContext

个人分类： FFMPEG

所属专栏： FFmpeg

此PDF由spygg生成,请尊重原作者版权!!!
我的邮箱:liushidc@163.com