

## 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。

```
1.  /**
2.   * Allocate an AVFormatContext for an output format.
3.   * avformat_free_context() can be used to free the context and
4.   * everything allocated by the framework within it.
5.   *
6.   * @param *ctx is set to the created format context, or to NULL in
7.   * case of failure
8.   * @param oformat format to use for allocating the context, if NULL
9.   * format_name and filename are used instead
10.  * @param format_name the name of output format to use for allocating the
11.  * context, if NULL filename is used instead
12.  * @param filename the name of the filename to use for allocating the
13.  * context, may be NULL
14.  * @return >= 0 in case of success, a negative AERROR code in case of
15.  * failure
16.  */
17.  int avformat_alloc_output_context2(AVFormatContext **ctx, AVOutputFormat *oformat,
18.                                     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。

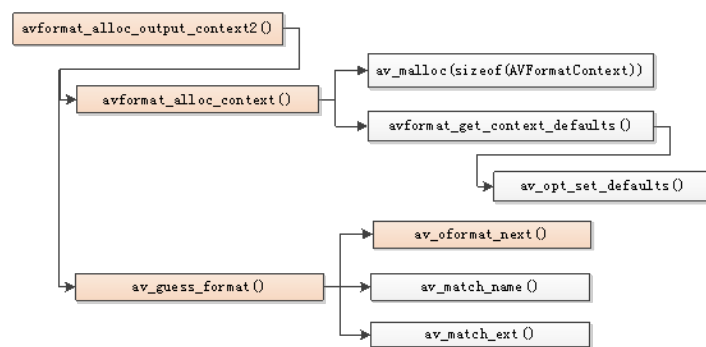
filename：指定输出文件的名称。根据文件名称，FFmpeg会推测输出格式。文件名称可以是“xx.flv”，“yy.mkv”等等。

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。



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

[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]
1.  int avformat_alloc_output_context2(AVFormatContext **avctx, AVOutputFormat *oformat,
2.                                  const char *format, const char *filename)
3.  {
4.      AVFormatContext *s = avformat_alloc_context();
5.      int ret = 0;
6.
7.
8.      *avctx = NULL;
9.      if (!s)
10.         goto nomem;
11.
12.
13.      if (!oformat) {
14.          if (format) {
15.              oformat = av_guess_format(format, NULL, NULL);
16.              if (!oformat) {
17.                  av_log(s, AV_LOG_ERROR, "Requested output format '%s' is not a suitable output format\n", format);
18.                  ret = AERROR(EINVAL);
19.                  goto error;
20.              }
21.          } else {
22.              oformat = av_guess_format(NULL, filename, NULL);
23.              if (!oformat) {
24.                  ret = AERROR(EINVAL);
25.                  av_log(s, AV_LOG_ERROR, "Unable to find a suitable output format for '%s'\n",
26.                      filename);
27.                  goto error;
28.              }
29.          }
30.      }
31.
32.
33.      s->oformat = oformat;
34.      if (s->oformat->priv_data_size > 0) {
35.          s->priv_data = av_mallocz(s->oformat->priv_data_size);
36.          if (!s->priv_data)
37.              goto nomem;
38.          if (s->oformat->priv_class) {
39.              *(const AVClass**)s->priv_data= s->oformat->priv_class;
40.              av_opt_set_defaults(s->priv_data);
41.          }
42.      } else
43.          s->priv_data = NULL;
44.
45.
46.      if (filename)
47.          av_strlcpy(s->filename, filename, sizeof(s->filename));
48.      *avctx = s;
49.      return 0;
50. nomem:
51.      av_log(s, AV_LOG_ERROR, "Out of memory\n");
52.      ret = AERROR(ENOMEM);
53. error:
54.      avformat_free_context(s);
55.      return ret;
56. }
```

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]
1. AVFormatContext *avformat_alloc_context(void)
2. {
3.     AVFormatContext *ic;
4.     ic = av_malloc(sizeof(AVFormatContext));
5.     if (!ic) return ic;
6.     avformat_get_context_defaults(ic);
7.
8.
9.     ic->internal = av_mallocz(sizeof(*ic->internal));
10.    if (!ic->internal) {
11.        avformat_free_context(ic);
12.        return NULL;
13.    }
14.
15.
16.    return ic;
17. }
```

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]
1. static void avformat_get_context_defaults(AVFormatContext *s)
2. {
3.     memset(s, 0, sizeof(AVFormatContext));
4.
5.
6.     s->av_class = &av_format_context_class;
7.
8.
9.     av_opt_set_defaults(s);
10. }
```

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]
1. /**
2.  * Return the output format in the list of registered output formats
3.  * which best matches the provided parameters, or return NULL if
4.  * there is no match.
5.  *
6.  * @param short_name if non-NULL checks if short_name matches with the
7.  * names of the registered formats
8.  * @param filename if non-NULL checks if filename terminates with the
9.  * extensions of the registered formats
10. * @param mime_type if non-NULL checks if mime_type matches with the
11. * MIME type of the registered formats
12. */
13. AVOutputFormat *av_guess_format(const char *short_name,
14.                                const char *filename,
15.                                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。

```

1. AVOutputFormat *av_guess_format(const char *short_name, const char *filename,
2.                                const char *mime_type)
3. {
4.     AVOutputFormat *fmt = NULL, *fmt_found;
5.     int score_max, score;
6.
7.
8.     /* specific test for image sequences */
9.     #if CONFIG_IMAGE2_MUXER
10.    if (!short_name && filename &&
11.        av_filename_number_test(filename) &&
12.        ff_guess_image2_codec(filename) != AV_CODEC_ID_NONE) {
13.        return av_guess_format("image2", NULL, NULL);
14.    }
15.    #endif
16.    /* Find the proper file type. */
17.    fmt_found = NULL;
18.    score_max = 0;
19.    while ((fmt = av_oformat_next(fmt))) {
20.        score = 0;
21.        if (fmt->name && short_name && av_match_name(short_name, fmt->name))
22.            score += 100;
23.        if (fmt->mime_type && mime_type && !strcmp(fmt->mime_type, mime_type))
24.            score += 10;
25.        if (filename && fmt->extensions &&
26.            av_match_ext(filename, fmt->extensions)) {
27.            score += 5;
28.        }
29.        if (score > score_max) {
30.            score_max = score;
31.            fmt_found = fmt;
32.        }
33.    }
34.    return fmt_found;
35. }

```

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”，“mime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。

```

1. AVOutputFormat ff_flv_muxer = {
2.     .name           = "flv",
3.     .long_name      = NULL_IF_CONFIG_SMALL("FLV (Flash Video)"),
4.     .mime_type      = "video/x-flv",
5.     .extensions     = "flv",
6.     .priv_data_size = sizeof(FLVContext),
7.     .audio_codec     = CONFIG_LIBMP3LAME ? AV_CODEC_ID_MP3 : AV_CODEC_ID_ADPCM_SWF,
8.     .video_codec     = AV_CODEC_ID_FLV1,
9.     .write_header    = flv_write_header,
10.    .write_packet     = flv_write_packet,
11.    .write_trailer    = flv_write_trailer,
12.    .codec_tag        = (const AVCodecTag* const []) {
13.        flv_video_codec_ids, flv_audio_codec_ids, 0
14.    },
15.    .flags            = AVFMT_GLOBALHEADER | AVFMT_VARIABLE_FPS |
16.        AVFMT_TS_NONSTRICT,
17. };

```

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]
1.  /**
2.   * If f is NULL, returns the first registered output format,
3.   * if f is non-NULL, returns the next registered output format after f
4.   * or NULL if f is the last one.
5.   */
6.  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]
1.  AVOutputFormat *av_oformat_next(const AVOutputFormat *f)
2.  {
3.      if (f)
4.          return f->next;
5.      else
6.          return first_oformat;
7.  }
```

## av\_match\_name()

av\_match\_name()是一个API函数，声明如下所示。

```
[cpp]
1.  /**
2.   * Match instances of a name in a comma-separated list of names.
3.   * @param name Name to look for.
4.   * @param names List of names.
5.   * @return 1 on match, 0 otherwise.
6.   */
7.  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]
1.  int av_match_name(const char *name, const char *names)
2.  {
3.      const char *p;
4.      int len, namelen;
5.
6.
7.      if (!name || !names)
8.          return 0;
9.
10.
11.     namelen = strlen(name);
12.     while ((p = strchr(names, ',')) {
13.         len = FFMAX(p - names, namelen);
14.         if (!av_strncasecmp(name, names, len))
15.             return 1;
16.         names = p + 1;
17.     }
18.     return !av_strcasecmp(name, names);
19. }
```

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]
1.  AVInputFormat ff_matroska_demuxer = {
2.      .name      = "matroska,webm",
3.      .long_name  = NULL_IF_CONFIG_SMALL("Matroska / WebM"),
4.      .extensions = "mkv,mk3d,mka,mks",
5.      .priv_data_size = sizeof(MatroskaDemuxContext),
6.      .read_probe  = matroska_probe,
7.      .read_header = matroska_read_header,
8.      .read_packet = matroska_read_packet,
9.      .read_close  = matroska_read_close,
10.     .read_seek   = matroska_read_seek,
11.     .mime_type    = "audio/webm,audio/x-matroska,video/webm,video/x-matroska"
12. };
```

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]
1.  /**
2.   * Return a positive value if the given filename has one of the given
3.   * extensions, 0 otherwise.
4.   *
5.   * @param filename  file name to check against the given extensions
6.   * @param extensions a comma-separated list of filename extensions
7.   */
8.  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]
1.  int av_match_ext(const char *filename, const char *extensions)
2.  {
3.      const char *ext;
4.
5.
6.      if (!filename)
7.          return 0;
8.
9.
10.     ext = strrchr(filename, '.');
11.     if (ext)
12.         return av_match_name(ext + 1, extensions);
13.     return 0;
14. }
```

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