

=====

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\(\)](#)

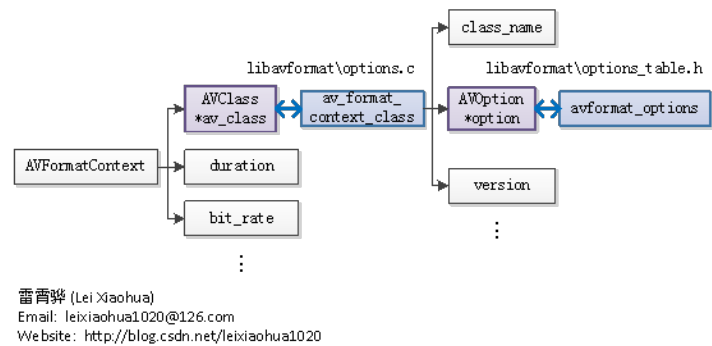
[FFmpeg 源代码简单分析：libavdevice 的 avdevice\\_register\\_all\(\)](#)

=====

本文继续上篇文章《FFmpeg源代码分析：结构体成员管理系统-AVClass》中的内容，记录FFmpeg中和AVOption相关的源代码。AVOption用于在FFmpeg中描述结构体中的成员变量。一个AVOption可以包含名称，简短的帮助信息，取值等等。

## 上篇文章简单回顾

上篇文章中概括了AVClass，AVOption和目标结构体之间的关系。以AVFormatContext为例，可以表示为下图。



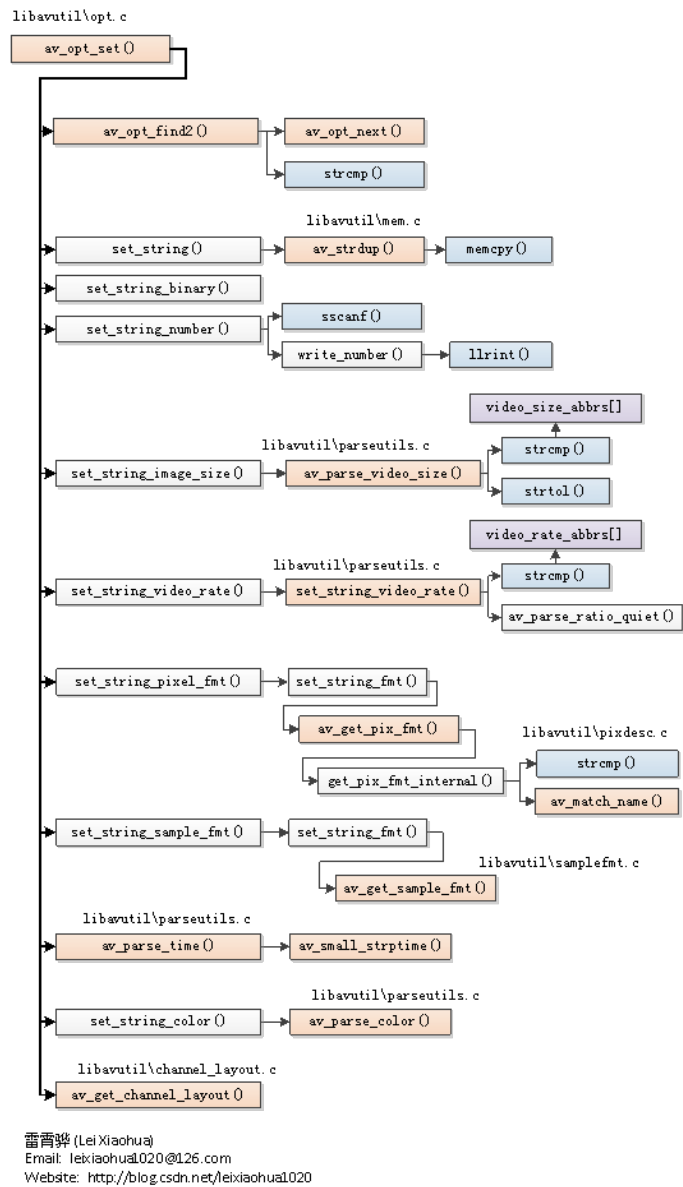
有关上篇文章的内容，这里不再重复。总体来说，上篇文章主要概括了AVClass，AVOption和目标结构体之间的从属关系，但是并没有分析有关AVOption的源代码。本文补充上一篇文章的内容，分析有关AVOption的源代码。

## AVOption有关的API

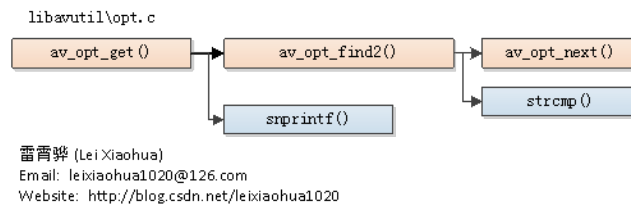
AVOption常用的API可以分成两类：用于设置参数的API和用于读取参数的API。其中最具有代表性的用于设置参数的API就是av\_opt\_set()；而最有代表性的用于读取参数的API就是av\_opt\_get()。除了记录以上两个函数之外，本文再记录一个在FFmpeg的结构体初始化代码中最常用的用于设置默认值的函数av\_opt\_set\_defaults()。

## 函数调用关系图

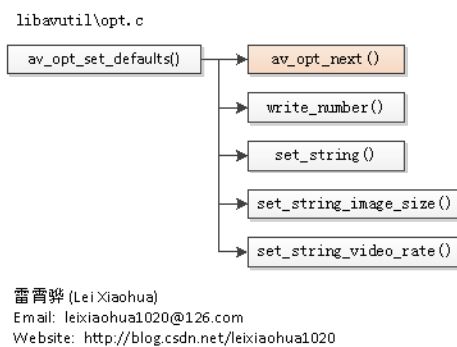
av\_opt\_set()的函数调用关系图如下所示。



av\_opt\_get()的函数调用关系图如下所示。



av\_opt\_set\_defaults()的函数调用关系图如下所示。



## av\_opt\_set()

通过AVOption设置参数最常用的函数就是av\_opt\_set()了。该函数通过字符串的方式（传入的参数是变量名称的字符串和变量值的字符串）设置一个AVOption的值。此外，还包含了它的一系列“兄弟”函数av\_opt\_set\_XXX(), 其中“XXX”代表了int, double这些数据类型。使用这些函数的时候，可以指定int, double这些类型的变量（而不是字符串）作为输入，设定相应的AVOption的值。

```
[cpp]
1.  /**
2.   * @defgroup opt_set_funcs Option setting functions
3.   * @{
4.   * Those functions set the field of obj with the given name to value.
5.   *
6.   * @param[in] obj A struct whose first element is a pointer to an AVClass.
7.   * @param[in] name the name of the field to set
8.   * @param[in] val The value to set. In case of av_opt_set() if the field is not
9.   * of a string type, then the given string is parsed.
10.  * SI postfixes and some named scalars are supported.
11.  * If the field is of a numeric type, it has to be a numeric or named
12.  * scalar. Behavior with more than one scalar and +- infix operators
13.  * is undefined.
14.  * If the field is of a flags type, it has to be a sequence of numeric
15.  * scalars or named flags separated by '+' or '-'. Prefixing a flag
16.  * with '+' causes it to be set without affecting the other flags;
17.  * similarly, '-' unsets a flag.
18.  * @param search_flags flags passed to av_opt_find2. I.e. if AV_OPT_SEARCH_CHILDREN
19.  * is passed here, then the option may be set on a child of obj.
20.  *
21.  * @return 0 if the value has been set, or an AERROR code in case of
22.  * error:
23.  * AERROR_OPTION_NOT_FOUND if no matching option exists
24.  * AERROR(ERANGE) if the value is out of range
25.  * AERROR(EINVAL) if the value is not valid
26.  */
27. int av_opt_set      (void *obj, const char *name, const char *val, int search_flags);
28. int av_opt_set_int  (void *obj, const char *name, int64_t  val, int search_flags);
29. int av_opt_set_double(void *obj, const char *name, double   val, int search_flags);
30. int av_opt_set_q    (void *obj, const char *name, AVRational val, int search_flags);
31. int av_opt_set_bin  (void *obj, const char *name, const uint8_t *val, int size, int search_flags);
32. int av_opt_set_image_size(void *obj, const char *name, int w, int h, int search_flags);
33. int av_opt_set_pixel_fmt(void *obj, const char *name, enum AVPixelFormat fmt, int search_flags);
34. int av_opt_set_sample_fmt(void *obj, const char *name, enum AVSampleFormat fmt, int search_flags);
35. int av_opt_set_video_rate(void *obj, const char *name, AVRational val, int search_flags);
36. int av_opt_set_channel_layout(void *obj, const char *name, int64_t ch_layout, int search_flags);
```

有关av\_opt\_set\_XXX()函数的定义不再详细分析，在这里详细看一下av\_opt\_set()的源代码。av\_opt\_set()的定义位于libavutil/opt.c，如下所示。

```

1. int av_opt_set(void *obj, const char *name, const char *val, int search_flags)
2. {
3.     int ret = 0;
4.     void *dst, *target_obj;
5.     //查找
6.     const AVOption *o = av_opt_find2(obj, name, NULL, 0, search_flags, &target_obj);
7.     if (!o || !target_obj)
8.         return AVERROR_OPTION_NOT_FOUND;
9.     if (!val && (o->type != AV_OPT_TYPE_STRING &&
10.                o->type != AV_OPT_TYPE_PIXEL_FMT && o->type != AV_OPT_TYPE_SAMPLE_FMT &&
11.                o->type != AV_OPT_TYPE_IMAGE_SIZE && o->type != AV_OPT_TYPE_VIDEO_RATE &&
12.                o->type != AV_OPT_TYPE_DURATION && o->type != AV_OPT_TYPE_COLOR &&
13.                o->type != AV_OPT_TYPE_CHANNEL_LAYOUT))
14.         return AVERROR(EINVAL);
15.
16.     if (o->flags & AV_OPT_FLAG_READONLY)
17.         return AVERROR(EINVAL);
18.     //dst指向具体的变量
19.     //注意：offset的作用
20.     dst = ((uint8_t*)target_obj) + o->offset;
21.     //根据AVOption不同的类型，调用不同的设置函数
22.     switch (o->type) {
23.     case AV_OPT_TYPE_STRING:     return set_string(obj, o, val, dst);
24.     case AV_OPT_TYPE_BINARY:     return set_string_binary(obj, o, val, dst);
25.     case AV_OPT_TYPE_FLAGS:
26.     case AV_OPT_TYPE_INT:
27.     case AV_OPT_TYPE_INT64:
28.     case AV_OPT_TYPE_FLOAT:
29.     case AV_OPT_TYPE_DOUBLE:
30.     case AV_OPT_TYPE_RATIONAL:   return set_string_number(obj, target_obj, o, val, dst);
31.     case AV_OPT_TYPE_IMAGE_SIZE: return set_string_image_size(obj, o, val, dst);
32.     case AV_OPT_TYPE_VIDEO_RATE: return set_string_video_rate(obj, o, val, dst);
33.     case AV_OPT_TYPE_PIXEL_FMT:  return set_string_pixel_fmt(obj, o, val, dst);
34.     case AV_OPT_TYPE_SAMPLE_FMT: return set_string_sample_fmt(obj, o, val, dst);
35.     case AV_OPT_TYPE_DURATION:
36.         if (!val) {
37.             *(int64_t *)dst = 0;
38.             return 0;
39.         } else {
40.             if ((ret = av_parse_time(dst, val, 1)) < 0)
41.                 av_log(obj, AV_LOG_ERROR, "Unable to parse option value \"%s\" as duration\n", val);
42.             return ret;
43.         }
44.         break;
45.     case AV_OPT_TYPE_COLOR:       return set_string_color(obj, o, val, dst);
46.     case AV_OPT_TYPE_CHANNEL_LAYOUT:
47.         if (!val || !strcmp(val, "none")) {
48.             *(int64_t *)dst = 0;
49.         } else {
50.             #if FF_API_GET_CHANNEL_LAYOUT_COMPAT
51.                 int64_t cl = ff_get_channel_layout(val, 0);
52.             #else
53.                 int64_t cl = av_get_channel_layout(val);
54.             #endif
55.             if (!cl) {
56.                 av_log(obj, AV_LOG_ERROR, "Unable to parse option value \"%s\" as channel layout\n", val);
57.                 ret = AVERROR(EINVAL);
58.             }
59.             *(int64_t *)dst = cl;
60.             return ret;
61.         }
62.         break;
63.     }
64.
65.     av_log(obj, AV_LOG_ERROR, "Invalid option type.\n");
66.     return AVERROR(EINVAL);
67. }

```

从源代码可以看出，av\_opt\_set()首先调用av\_opt\_find2()查找AVOption。如果找到了，则根据AVOption的type，调用不同的函数（set\_string(), set\_string\_number(), set\_string\_image\_size()等等）将输入的字符串转化为相应type的数据并对该AVOption进行赋值。如果没有找到，则立即返回“没有找到AVOption”的错误。

## av\_opt\_find2() / av\_opt\_find()

av\_opt\_find2()本身也是一个API函数，用于查找AVOption。它的声明位于libavutil/opt.h中，如下所示。

```

1.  /**
2.   * Look for an option in an object. Consider only options which
3.   * have all the specified flags set.
4.   *
5.   * @param[in] obj A pointer to a struct whose first element is a
6.   *               pointer to an AVClass.
7.   *               Alternatively a double pointer to an AVClass, if
8.   *               AV_OPT_SEARCH_FAKE_OBJ search flag is set.
9.   * @param[in] name The name of the option to look for.
10.  * @param[in] unit When searching for named constants, name of the unit
11.  *                it belongs to.
12.  * @param opt_flags Find only options with all the specified flags set (AV_OPT_FLAG).
13.  * @param search_flags A combination of AV_OPT_SEARCH_*.
14.  * @param[out] target_obj if non-NULL, an object to which the option belongs will be
15.  * written here. It may be different from obj if AV_OPT_SEARCH_CHILDREN is present
16.  * in search_flags. This parameter is ignored if search_flags contain
17.  * AV_OPT_SEARCH_FAKE_OBJ.
18.  *
19.  * @return A pointer to the option found, or NULL if no option
20.  *         was found.
21.  */
22.  const AVOption *av_opt_find2(void *obj, const char *name, const char *unit,
23.                               int opt_flags, int search_flags, void **target_obj);

```

此外还有一个和av\_opt\_find2()“长得很像”的API函数av\_opt\_find(), 功能与av\_opt\_find2()基本类似, 与av\_opt\_find2()相比少了最后一个参数。从源代码中可以看出它只是简单调用了av\_opt\_find2()并把所有的输入参数原封不动的传递过去, 并把最后一个参数设置成NULL。

```

1.  const AVOption *av_opt_find(void *obj, const char *name, const char *unit,
2.                               int opt_flags, int search_flags)
3.  {
4.      return av_opt_find2(obj, name, unit, opt_flags, search_flags, NULL);
5.  }

```

下面先看一下av\_opt\_find2()函数的定义。该函数的定义位于libavutil/opt.c中, 如下所示。

```

1.  const AVOption *av_opt_find2(void *obj, const char *name, const char *unit,
2.                               int opt_flags, int search_flags, void **target_obj)
3.  {
4.      const AVClass *c;
5.      const AVOption *o = NULL;
6.
7.      if(!obj)
8.          return NULL;
9.
10.     c = *(AVClass**)obj;
11.
12.     if (!c)
13.         return NULL;
14.     //查找范围包含子节点的时候
15.     //递归调用av_opt_find2()
16.     if (search_flags & AV_OPT_SEARCH_CHILDREN) {
17.         if (search_flags & AV_OPT_SEARCH_FAKE_OBJ) {
18.             const AVClass *child = NULL;
19.             while (child = av_opt_child_class_next(c, child))
20.                 if (o = av_opt_find2(&child, name, unit, opt_flags, search_flags, NULL))
21.                     return o;
22.         } else {
23.             void *child = NULL;
24.             while (child = av_opt_child_next(obj, child))
25.                 if (o = av_opt_find2(child, name, unit, opt_flags, search_flags, target_obj))
26.                     return o;
27.         }
28.     }
29.     //遍历
30.     while (o = av_opt_next(obj, o)) {
31.         //比较名称
32.         if (!strcmp(o->name, name) && (o->flags & opt_flags) == opt_flags &&
33.             ((unit && o->type != AV_OPT_TYPE_CONST) ||
34.              (unit && o->type == AV_OPT_TYPE_CONST && o->unit && !strcmp(o->unit, unit)))) {
35.             if (target_obj) {
36.                 if (!(search_flags & AV_OPT_SEARCH_FAKE_OBJ))
37.                     *target_obj = obj;
38.                 else
39.                     *target_obj = NULL;
40.             }
41.             return o;
42.         }
43.     }
44.     return NULL;
45. }

```

这段代码的前半部分暂时不关注，前半部分的if()语句中的内容只有在search\_flags指定为AV\_OPT\_SEARCH\_CHILDREN的时候才会执行。后半部分代码是重点。后半部分代码是一个while()循环，该循环的条件是一个函数av\_opt\_next()。

## av\_opt\_next()

av\_opt\_next()也是一个FFmpeg的API函数。使用它可以循环遍历目标结构体的所有AVOption，它的声明如下。

```
[cpp]
1.  /**
2.   * Iterate over all AVOptions belonging to obj.
3.   *
4.   * @param obj an AVOptions-enabled struct or a double pointer to an
5.   *           AVClass describing it.
6.   * @param prev result of the previous call to av_opt_next() on this object
7.   *           or NULL
8.   * @return next AVOption or NULL
9.   */
10. const AVOption *av_opt_next(void *obj, const AVOption *prev);
```

av\_opt\_next()的定义如下所示。

```
[cpp]
1.  const AVOption *av_opt_next(void *obj, const AVOption *last)
2.  {
3.      AVClass *class = *(AVClass**)obj;
4.      if (!last && class && class->option && class->option[0].name)
5.          return class->option;
6.      if (last && last[1].name)
7.          return ++last;
8.      return NULL;
9.  }
```

从av\_opt\_next()的代码可以看出，输入的AVOption类型的last变量为空的时候，会返回该AVClass的option数组的第一个元素，否则会返回数组的下一个元素。

现在再回到av\_opt\_find2()函数。我们发现在while()循环中有一个strcmp()函数，正是这个函数比较输入的AVOption的name和AVClass的option数组中每个元素的name，当上述两个name相等的时候，就代表查找到了AVOption，接着就可以返回获得的AVOption。

现在再回到刚才的av\_opt\_set()函数。该函数有一个void型的变量dst用于确定需要设定的AVOption对应的变量的位置。具体的方法就是将输入的AVClass结构体的首地址加上该AVOption的偏移量offset。确定了AVOption对应的变量的位置之后，就可以根据该AVOption的类型type的不同调用不同的字符串转换函数设置相应的值了。我们可以看几个设置值的简单例子：

### 1. AV\_OPT\_TYPE\_STRING

当AVOption的type为AV\_OPT\_TYPE\_STRING的时候，调用set\_string()方法设置相应的值。set\_string()的定义如下：

```
[cpp]
1.  static int set_string(void *obj, const AVOption *o, const char *val, uint8_t **dst)
2.  {
3.      av_freep(dst);
4.      *dst = av_strdup(val);
5.      return 0;
6.  }
```

其中又调用了函数av\_strdup()，这是一个FFmpeg的API函数，用于拷贝字符串。它的代码如下所示，其中调用了memcpy()。

```
[cpp]
1.  char *av_strdup(const char *s)
2.  {
3.      char *ptr = NULL;
4.      if (s) {
5.          int len = strlen(s) + 1;
6.          ptr = av_realloc(NULL, len);
7.          if (ptr)
8.              memcpy(ptr, s, len);
9.      }
10.     return ptr;
11. }
```

### 2.

#### AV\_OPT\_TYPE\_IMAGE\_SIZE

当AVOption的type为AV\_OPT\_TYPE\_IMAGE\_SIZE的时候，调用set\_string\_image\_size()方法设置相应的值。set\_string\_image\_size()的定义如下。

```
[cpp]
1. static int set_string_image_size(void *obj, const AVOption *o, const char *val, int *dst)
2. {
3.     int ret;
4.
5.     if (!val || !strcmp(val, "none")) {
6.         dst[0] =
7.         dst[1] = 0;
8.         return 0;
9.     }
10.    ret = av_parse_video_size(dst, dst + 1, val);
11.    if (ret < 0)
12.        av_log(obj, AV_LOG_ERROR, "Unable to parse option value \"%s\" as image size\n", val);
13.    return ret;
14. }
```

可见其中调用了另一个函数av\_parse\_video\_size()。

## av\_parse\_video\_size()

av\_parse\_video\_size()是一个FFmpeg的API函数，用于解析出输入的分辨率字符串的宽高信息。例如，输入的字符串为“1920x1080”或者“1920\*1080”，经过av\_parse\_video\_size()的处理之后，可以得到宽度为1920，高度为1080；此外，输入一个“特定分辨率”字符串例如“vga”，也可以得到宽度为640，高度为480。该函数不属于AVOption这部分的内容，而是整个FFmpeg通用的一个字符串解析函数。声明位于libavutil\parseutils.h中，如下所示。

```
[cpp]
1. /**
2.  * Parse str and put in width_ptr and height_ptr the detected values.
3.  *
4.  * @param[in,out] width_ptr pointer to the variable which will contain the detected
5.  * width value
6.  * @param[in,out] height_ptr pointer to the variable which will contain the detected
7.  * height value
8.  * @param[in] str the string to parse: it has to be a string in the format
9.  * width x height or a valid video size abbreviation.
10. * @return >= 0 on success, a negative error code otherwise
11. */
12. int av_parse_video_size(int *width_ptr, int *height_ptr, const char *str);
```

从声明中可以看出，该函数输入一个字符串str，输出结果保存在width\_ptr和height\_ptr所指向的内存中。av\_parse\_video\_size()定义位于libavutil\parseutils.c中，代码如下。

```
[cpp]
1. //解析分辨率
2. int av_parse_video_size(int *width_ptr, int *height_ptr, const char *str)
3. {
4.     int i;
5.     int n = FF_ARRAY_ELEMS(video_size_abbrs);
6.     const char *p;
7.     int width = 0, height = 0;
8.     //先看看有没有“分辨率简称”相同的（例如vga, qcif等）
9.     for (i = 0; i < n; i++) {
10.        if (!strcmp(video_size_abbrs[i].abbr, str)) {
11.            width = video_size_abbrs[i].width;
12.            height = video_size_abbrs[i].height;
13.            break;
14.        }
15.    }
16.    //如果没有使用“分辨率简称”，而是使用具体的数值（例如“1920x1080”），则执行下面的步骤
17.    if (i == n) {
18.        //strtoul(): 字符串转换成无符号长整型，遇到非数字则停止
19.        width = strtoul(str, (void*)&p, 10);
20.        if (*p)
21.            p++;
22.        height = strtoul(p, (void*)&p, 10);
23.
24.        /* trailing extraneous data detected, like in 123x345foobar */
25.        if (*p)
26.            return AVERROR(EINVAL);
27.    }
28.    //检查一下正确性
29.    if (width <= 0 || height <= 0)
30.        return AVERROR(EINVAL);
31.    *width_ptr = width;
32.    *height_ptr = height;
33.    return 0;
34. }
```



上述代码中包含了FFmpeg中两种解析视频分辨率的方法。FFmpeg中包含两种设定视频分辨率的方法：通过已经定义好的“分辨率简称”，或者通过具体的数值。代码中首先遍历“特定分辨率”的数组video\_size\_abbrs。该数组定义如下所示。

```
[cpp]
1. typedef struct {
2.     const char *abbr;
3.     int width, height;
4. } VideoSizeAbbr;
5.
6. static const VideoSizeAbbr video_size_abbrs[] = {
7.     { "ntsc",      720, 480 },
8.     { "pal",       720, 576 },
9.     { "qntsc",     352, 240 }, /* VCD compliant NTSC */
10.    { "qpal",      352, 288 }, /* VCD compliant PAL */
11.    { "sntsc",     640, 480 }, /* square pixel NTSC */
12.    { "spal",      768, 576 }, /* square pixel PAL */
13.    { "film",      352, 240 },
14.    { "ntsc-film", 352, 240 },
15.    { "sqcif",     128,  96 },
16.    { "qcif",      176, 144 },
17.    { "cif",       352, 288 },
18.    { "4cif",      704, 576 },
19.    { "16cif",     1408,1152 },
20.    { "qqvga",     160, 120 },
21.    { "qvga",      320, 240 },
22.    { "vga",       640, 480 },
23.    { "svga",      800, 600 },
24.    { "xga",       1024, 768 },
25.    { "uxga",      1600,1200 },
26.    { "qxga",      2048,1536 },
27.    { "sxga",      1280,1024 },
28.    { "qsxga",     2560,2048 },
29.    { "hsxga",     5120,4096 },
30.    { "wvga",      852, 480 },
31.    { "wxga",      1366, 768 },
32.    { "wsxga",     1600,1024 },
33.    { "wuxga",     1920,1200 },
34.    { "woxga",     2560,1600 },
35.    { "wqsxga",    3200,2048 },
36.    { "wquxga",    3840,2400 },
37.    { "whsxga",    6400,4096 },
38.    { "whuxga",    7680,4800 },
39.    { "cga",       320, 200 },
40.    { "ega",       640, 350 },
41.    { "hd480",     852, 480 },
42.    { "hd720",     1280, 720 },
43.    { "hd1080",    1920,1080 },
44.    { "2k",        2048,1080 }, /* Digital Cinema System Specification */
45.    { "2kflat",    1998,1080 },
46.    { "2kscope",   2048, 858 },
47.    { "4k",        4096,2160 }, /* Digital Cinema System Specification */
48.    { "4kflat",    3996,2160 },
49.    { "4kscope",   4096,1716 },
50.    { "nhd",       640,360 },
51.    { "hqvga",     240,160 },
52.    { "wqvga",     400,240 },
53.    { "fwqvga",    432,240 },
54.    { "hvga",      480,320 },
55.    { "qhd",       960,540 },
56. };
```

通过调用strcmp()方法比对输入字符串的值与video\_size\_abbrs数组中每个VideoSizeAbbr元素的abbr字段的值，判断输入的字符串是否指定了这些标准的分辨率。如果指定了的话，则返回该分辨率的宽和高。

如果从上述列表中没有找到相应的“特定分辨率”，则说明输入的字符串应该是一个具体的分辨率的值，形如“1920\*1020”，“1280x720”这样的字符串。这个时候就需要对这个字符串进行解析，并从中提取出数字信息。通过两次调用strtol()方法，从字符串中提取出宽高信息（第一次提取出宽，第二次提取出高）。

PS1：strtol()用于将字符串转换成整型，遇到非数字则停止。

PS2：从这种解析方法可以得到一个信息——FFmpeg并不管“宽{X}高”中间的那个{X}是什么字符，也就是说中间那个字符不一定非得是“\*”或者“x”。后来试了一下，中间那个字符使用其他字母也是可以的。

## av\_opt\_set\_defaults()

av\_opt\_set\_defaults()是一个FFmpeg的API，作用是给一个结构体的成员变量设定默认值。在FFmpeg初始化其各种结构体（AVFormatContext，AVCodecContext等）的时候，通常会调用该函数设置结构体中的默认值。av\_opt\_set\_defaults()的声明如下所示。

```

1.  /**
2.   * Set the values of all AVOption fields to their default values.
3.   *
4.   * @param s an AVOption-enabled struct (its first member must be a pointer to AVClass)
5.   */
6.  void av_opt_set_defaults(void *s);

```

可见只需要把包含AVOption功能的结构体（第一个变量是一个AVClass类型的指针）的指针提供给av\_opt\_set\_defaults(), 就可以初始化该结构体的默认值了。下面看一下av\_opt\_set\_defaults()的源代码，位于libavutil/opt.c，如下所示。

```

1.  void av_opt_set_defaults(void *s)
2.  {
3.      //奇怪的#if...#endif
4.      #if FF_API_OLD_AVOPTIONS
5.          av_opt_set_defaults2(s, 0, 0);
6.      }
7.
8.  void av_opt_set_defaults2(void *s, int mask, int flags)
9.  {
10.     #endif
11.     const AVOption *opt = NULL;
12.     //遍历所有的AVOption
13.     while ((opt = av_opt_next(s, opt))) {
14.         //注意:offset的使用
15.         void *dst = ((uint8_t*)s) + opt->offset;
16.         #if FF_API_OLD_AVOPTIONS
17.             if ((opt->flags & mask) != flags)
18.                 continue;
19.         #endif
20.
21.         if (opt->flags & AV_OPT_FLAG_READONLY)
22.             continue;
23.         //读取各种default_val
24.         switch (opt->type) {
25.             case AV_OPT_TYPE_CONST:
26.                 /* Nothing to be done here */
27.                 break;
28.             case AV_OPT_TYPE_FLAGS:
29.             case AV_OPT_TYPE_INT:
30.             case AV_OPT_TYPE_INT64:
31.             case AV_OPT_TYPE_DURATION:
32.             case AV_OPT_TYPE_CHANNEL_LAYOUT:
33.                 write_number(s, opt, dst, 1, 1, opt->default_val.i64);
34.                 break;
35.             case AV_OPT_TYPE_DOUBLE:
36.             case AV_OPT_TYPE_FLOAT: {
37.                 double val;
38.                 val = opt->default_val.dbl;
39.                 write_number(s, opt, dst, val, 1, 1);
40.             }
41.             break;
42.             case AV_OPT_TYPE_RATIONAL: {
43.                 AVRational val;
44.                 val = av_d2q(opt->default_val.dbl, INT_MAX);
45.                 write_number(s, opt, dst, 1, val.den, val.num);
46.             }
47.             break;
48.             case AV_OPT_TYPE_COLOR:
49.                 set_string_color(s, opt, opt->default_val.str, dst);
50.                 break;
51.             case AV_OPT_TYPE_STRING:
52.                 set_string(s, opt, opt->default_val.str, dst);
53.                 break;
54.             case AV_OPT_TYPE_IMAGE_SIZE:
55.                 set_string_image_size(s, opt, opt->default_val.str, dst);
56.                 break;
57.             case AV_OPT_TYPE_VIDEO_RATE:
58.                 set_string_video_rate(s, opt, opt->default_val.str, dst);
59.                 break;
60.             case AV_OPT_TYPE_PIXEL_FMT:
61.                 write_number(s, opt, dst, 1, 1, opt->default_val.i64);
62.                 break;
63.             case AV_OPT_TYPE_SAMPLE_FMT:
64.                 write_number(s, opt, dst, 1, 1, opt->default_val.i64);
65.                 break;
66.             case AV_OPT_TYPE_BINARY:
67.                 set_string_binary(s, opt, opt->default_val.str, dst);
68.                 break;
69.             case AV_OPT_TYPE_DICT:
70.                 /* Cannot set defaults for these types */
71.                 break;
72.             default:
73.                 av_log(s, AV_LOG_DEBUG, "AVOption type %d of option %s not implemented yet\n", opt->type, opt->name);
74.         }
75.     }
76. }

```

av\_opt\_set\_defaults()代码开始的时候有一个预编译指令还是挺奇怪的。怪就怪在#if和#endif竟然横跨在了两个函数之间。简单解读一下这个定义的意思：当定义了FF\_API\_OLD\_AVOPTIONS的时候，存在两个函数av\_opt\_set\_defaults()和av\_opt\_set\_defaults2()，而这两个函数的作用是一样的；当没有定义FF\_API\_OLD\_AVOPTIONS的时候，只存在一个函数av\_opt\_set\_defaults()。估计FFmpeg这么做主要是考虑到兼容性的问题。

av\_opt\_set\_defaults()主体部分是一个while()循环。该循环的判断条件是一个av\_opt\_next()，其作用是获得下一个AVOption。该函数的定义在前文中已经做过分析，这里再重复一下。定义如下所示。

```
[cpp]
1.  const AVOption *av_opt_next(void *obj, const AVOption *last)
2.  {
3.      AVClass *class = *(AVClass**)obj;
4.      if (!last && class && class->option && class->option[0].name)
5.          return class->option;
6.      if (last && last[1].name)
7.          return ++last;
8.      return NULL;
9.  }
```

从av\_opt\_next()的代码可以看出，输入的AVOption类型的last为空的时候，会返回该AVClass的option数组的第一个元素，否则会返回下一个元素。

下面简单解读一下av\_opt\_set\_defaults()中while()循环语句里面的内容。有一个void类型的指针dst用于确定当前AVOption代表的变量的位置。该指针的位置有结构体的首地址和变量的偏移量offset确定。然后根据AVOption代表的变量的类型type，调用不同的函数设定相应的值。例如type为AV\_OPT\_TYPE\_INT的话，则会调用write\_number()；type为AV\_OPT\_TYPE\_STRING的时候，则会调用set\_string()；type为AV\_OPT\_TYPE\_IMAGE\_SIZE的时候，则会调用set\_string\_image\_size()。有关这些设置值的函数在前文中已经有所叙述，不再重复。需要注意的是，该函数中设置的值都是AVOption中的default\_val变量的值。

## av\_opt\_get()

av\_opt\_get()用于获取一个AVOption变量的值。需要注意的是，不论是何种类型的变量，通过av\_opt\_get()取出来的值都是字符串类型的。此外，还包含了它的一系列“兄弟”函数av\_opt\_get\_XXX()（其中“XXX”代表了int，double这些数据类型）。通过这些“兄弟”函数可以直接取出int，double类型的数值。av\_opt\_get()的声明如下所示。

```
[cpp]
1.  /**
2.   * @defgroup opt_get_funcs Option getting functions
3.   * @{
4.   * Those functions get a value of the option with the given name from an object.
5.   *
6.   * @param[in] obj a struct whose first element is a pointer to an AVClass.
7.   * @param[in] name name of the option to get.
8.   * @param[in] search_flags flags passed to av_opt_find2. I.e. if AV_OPT_SEARCH_CHILDREN
9.   * is passed here, then the option may be found in a child of obj.
10.  * @param[out] out_val value of the option will be written here
11.  * @return >=0 on success, a negative error code otherwise
12.  */
13. /**
14.  * @note the returned string will be av_malloc()ed and must be av_free()ed by the caller
15.  */
16.  int av_opt_get      (void *obj, const char *name, int search_flags, uint8_t **out_val);
17.  int av_opt_get_int  (void *obj, const char *name, int search_flags, int64_t *out_val);
18.  int av_opt_get_double(void *obj, const char *name, int search_flags, double *out_val);
19.  int av_opt_get_q    (void *obj, const char *name, int search_flags, AVRational *out_val);
20.  int av_opt_get_image_size(void *obj, const char *name, int search_flags, int *w_out, int *h_out);
21.  int av_opt_get_pixel_fmt(void *obj, const char *name, int search_flags, enum AVPixelFormat *out_fmt);
22.  int av_opt_get_sample_fmt(void *obj, const char *name, int search_flags, enum AVSampleFormat *out_fmt);
23.  int av_opt_get_video_rate(void *obj, const char *name, int search_flags, AVRational *out_val);
24.  int av_opt_get_channel_layout(void *obj, const char *name, int search_flags, int64_t *ch_layout);
```

下面我们看一下av\_opt\_get()的定义，如下所示。

```

1. int av_opt_get(void *obj, const char *name, int search_flags, uint8_t **out_val)
2. {
3.     void *dst, *target_obj;
4.     //查找
5.     const AVOption *o = av_opt_find2(obj, name, NULL, 0, search_flags, &target_obj);
6.     uint8_t *bin, buf[128];
7.     int len, i, ret;
8.     int64_t i64;
9.
10.    if (!o || !target_obj || (o->offset <= 0 && o->type != AV_OPT_TYPE_CONST))
11.        return AVERROR_OPTION_NOT_FOUND;
12.    //注意:offset的使用
13.    dst = (uint8_t *)target_obj + o->offset;
14.    //使用sprintf()转换成字符串, 存入buf
15.    buf[0] = 0;
16.    switch (o->type) {
17.        case AV_OPT_TYPE_FLAGS:    ret = snprintf(buf, sizeof(buf), "0x%08X", *(int *)dst);break;
18.        case AV_OPT_TYPE_INT:      ret = snprintf(buf, sizeof(buf), "%d", *(int *)dst);break;
19.        case AV_OPT_TYPE_INT64:    ret = snprintf(buf, sizeof(buf), "%"PRIi64, *(int64_t *)dst);break;
20.        case AV_OPT_TYPE_FLOAT:    ret = snprintf(buf, sizeof(buf), "%f", *(float *)dst);break;
21.        case AV_OPT_TYPE_DOUBLE:   ret = snprintf(buf, sizeof(buf), "%f", *(double *)dst);break;
22.        case AV_OPT_TYPE_VIDEO_RATE:
23.        case AV_OPT_TYPE_RATIONAL: ret = snprintf(buf, sizeof(buf), "%d/%d", ((AVRational *)dst)->num, ((AVRational *)dst)->den);break;
24.
25.        case AV_OPT_TYPE_CONST:    ret = snprintf(buf, sizeof(buf), "%f", o->default_val.dbl);break;
26.        case AV_OPT_TYPE_STRING:
27.            if (*(uint8_t **)dst)
28.                *out_val = av_strdup(*(uint8_t **)dst);
29.            else
30.                *out_val = av_strdup("");
31.            return 0;
32.        case AV_OPT_TYPE_BINARY:
33.            len = *(int *)(((uint8_t *)dst) + sizeof(uint8_t *));
34.            if ((uint64_t)len*2 + 1 > INT_MAX)
35.                return AVERROR(EINVAL);
36.            if (!(*out_val = av_malloc(len*2 + 1)))
37.                return AVERROR(ENOMEM);
38.            if (!len) {
39.                *out_val[0] = '\0';
40.                return 0;
41.            }
42.            bin = *(uint8_t **)dst;
43.            for (i = 0; i < len; i++)
44.                snprintf(*out_val + i*2, 3, "%02X", bin[i]);
45.            return 0;
46.        case AV_OPT_TYPE_IMAGE_SIZE:
47.            //分辨率
48.            ret = snprintf(buf, sizeof(buf), "%dx%d", ((int *)dst)[0], ((int *)dst)[1]);
49.            break;
50.        case AV_OPT_TYPE_PIXEL_FMT:
51.            //像素格式
52.            ret = snprintf(buf, sizeof(buf), "%s", (char *)av_x_if_null(av_get_pix_fmt_name(*(enum AVPixelFormat *)dst), "none"));
53.            break;
54.        case AV_OPT_TYPE_SAMPLE_FMT:
55.            //采样格式
56.            ret = snprintf(buf, sizeof(buf), "%s", (char *)av_x_if_null(av_get_sample_fmt_name(*(enum AVSampleFormat *)dst), "none"));
57.            break;
58.        case AV_OPT_TYPE_DURATION:
59.            //时长
60.            i64 = *(int64_t *)dst;
61.            ret = snprintf(buf, sizeof(buf), "%"PRIi64":%02d:%02d.%06d",
62.                i64 / 3600000000, (int)((i64 / 60000000) % 60),
63.                (int)((i64 / 1000000) % 60), (int)(i64 % 1000000));
64.            break;
65.        case AV_OPT_TYPE_COLOR:
66.            ret = snprintf(buf, sizeof(buf), "0x%02x%02x%02x%02x",
67.                (int)((uint8_t *)dst)[0], (int)((uint8_t *)dst)[1],
68.                (int)((uint8_t *)dst)[2], (int)((uint8_t *)dst)[3]);
69.            break;
70.        case AV_OPT_TYPE_CHANNEL_LAYOUT:
71.            i64 = *(int64_t *)dst;
72.            ret = snprintf(buf, sizeof(buf), "0x%"PRIx64, i64);
73.            break;
74.        default:
75.            return AVERROR(EINVAL);
76.    }
77.
78.    if (ret >= sizeof(buf))
79.        return AVERROR(EINVAL);
80.    //拷贝
81.    *out_val = av_strdup(buf);
82.    return 0;
83. }

```

从av\_opt\_get()的定义可以看出, 该函数首先通过av\_opt\_find2()查相应的AVOption, 然后取出该变量的值, 最后通过sprintf()将变量的值转化为字符串(各种各样类型的变量都这样处理)并且输出出来。

至此FFmpeg中和AVOption相关的源代码基本上就分析完毕了。

**雷霄骅**

**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/44279329>

文章标签：[FFmpeg](#) [AVOption](#) [AVClass](#) [源代码](#)

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](mailto:liushidc@163.com)