原 FFmpeg源代码简单分析：结构体成员管理系统-AVOption

2015年03月16日 09:41:56　阅读数：13832

==========

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】

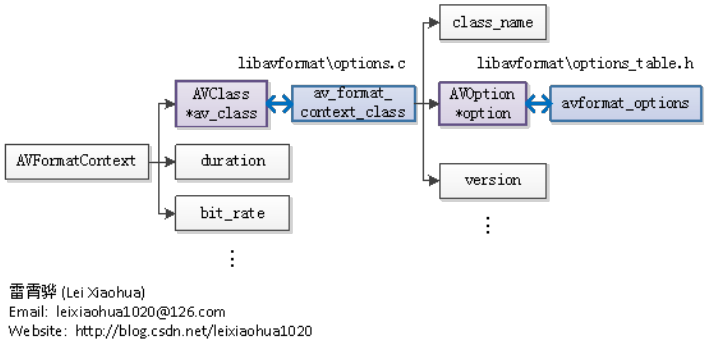
FFmpeg 的 H.264 解码器源代码简单分析：概述

==========

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。



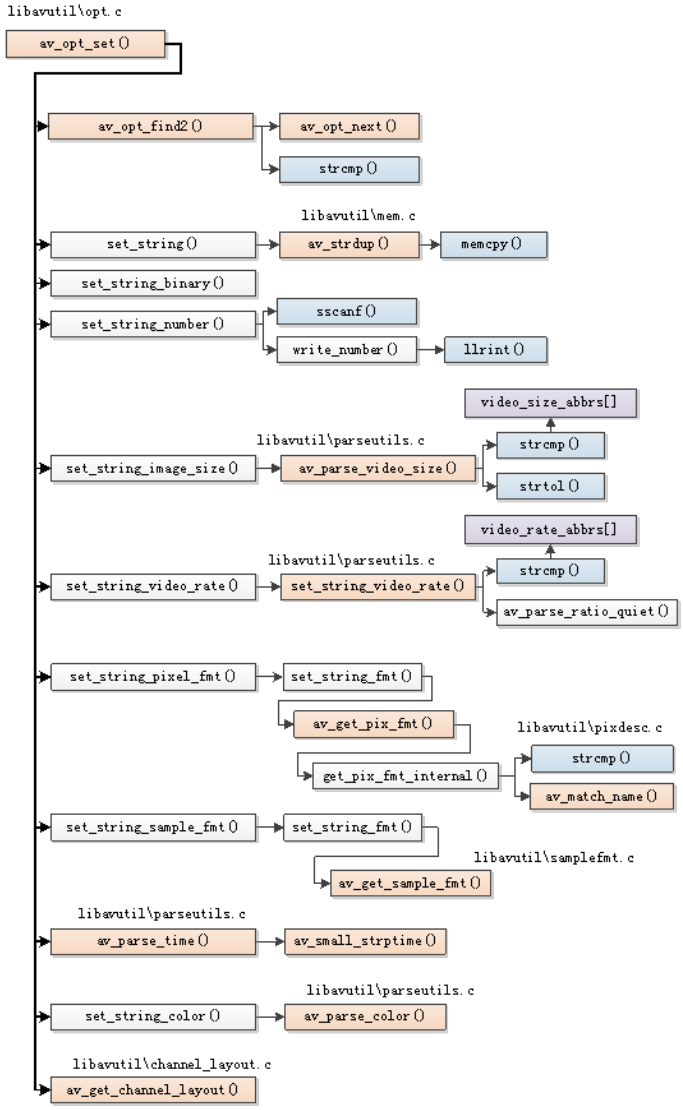
有关上篇文章的内容，这里不再重复。总体来说，上篇文章主要概括了AVClass，AVOption和目标结构体之间的从属关系，但是并没有分析有关AVOption的源代码。本文补充上一篇文章的内容，分析有关AVOption的源代码。

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()。

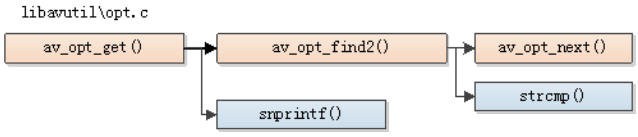
函数调用关系图

av_opt_set()的函数调用关系图如下所示。



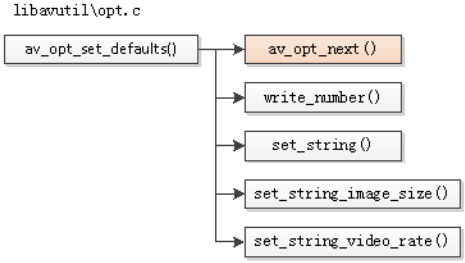
雷霄骅 (Lei Xiaohua)
Email: leixiaohua1020@126.com
Website: http://blog.csdn.net/leixiaohua1020

av_opt_get()的函数调用关系图如下所示。



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

av_opt_set_defaults()的函数调用关系图如下所示。



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

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
/**
 * @defgroup opt_set_funcs Option setting functions
 * @{
 * Those functions set the field of obj with the given name to value.
 *
 * @param[in] obj A struct whose first element is a pointer to an AVClass.
 * @param[in] name the name of the field to set
 * @param[in] val The value to set. In case of av_opt_set() if the field is not
 * of a string type, then the given string is parsed.
 * SI postfixes and some named scalars are supported.
 * If the field is of a numeric type, it has to be a numeric or named
 * scalar. Behavior with more than one scalar and +- infix operators
 * is undefined.
 * If the field is of a flags type, it has to be a sequence of numeric
 * scalars or named flags separated by '+' or '-'. Prefixing a flag
 * with '+' causes it to be set without affecting the other flags;
 * similarly, '-' unsets a flag.
 * @param search_flags flags passed to av_opt_find2. I.e. if AV_OPT_SEARCH_CHILDREN
 * is passed here, then the option may be set on a child of obj.
 *
 * @return 0 if the value has been set, or an AVERROR code in case of
 * error:
 * AVERROR_OPTION_NOT_FOUND if no matching option exists
 * AVERROR(ERANGE) if the value is out of range
 * AVERROR(EINVAL) if the value is not valid
 */
int av_opt_set         (void *obj, const char *name, const char *val, int search_flags);
int av_opt_set_int     (void *obj, const char *name, int64_t     val, int search_flags);
int av_opt_set_double  (void *obj, const char *name, double      val, int search_flags);
int av_opt_set_q       (void *obj, const char *name, AVRational  val, int search_flags);
int av_opt_set_bin     (void *obj, const char *name, const uint8_t *val, int size, int search_flags);
int av_opt_set_image_size(void *obj, const char *name, int w, int h, int search_flags);
int av_opt_set_pixel_fmt (void *obj, const char *name, enum AVPixelFormat fmt, int search_flags);
int av_opt_set_sample_fmt(void *obj, const char *name, enum AVSampleFormat fmt, int search_flags);
int av_opt_set_video_rate(void *obj, const char *name, AVRational val, int search_flags);
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。

```cpp
int av_opt_set(void *obj, const char *name, const char *val, int search_flags)
{
    int ret = 0;
    void *dst, *target_obj;
    //查找
    const AVOption *o = av_opt_find2(obj, name, NULL, 0, search_flags, &target_obj);
    if (!o || !target_obj)
        return AVERROR_OPTION_NOT_FOUND;
    if (!val && (o->type != AV_OPT_TYPE_STRING &&
                 o->type != AV_OPT_TYPE_PIXEL_FMT && o->type != AV_OPT_TYPE_SAMPLE_FMT &&
                 o->type != AV_OPT_TYPE_IMAGE_SIZE && o->type != AV_OPT_TYPE_VIDEO_RATE &&
                 o->type != AV_OPT_TYPE_DURATION && o->type != AV_OPT_TYPE_COLOR &&
                 o->type != AV_OPT_TYPE_CHANNEL_LAYOUT))
        return AVERROR(EINVAL);

    if (o->flags & AV_OPT_FLAG_READONLY)
        return AVERROR(EINVAL);
    //dst指向具体的变量
    //注意：offset的作用
    dst = ((uint8_t*)target_obj) + o->offset;
    //根据AVOption不同的类型，调用不同的设置函数
    switch (o->type) {
    case AV_OPT_TYPE_STRING:   return set_string(obj, o, val, dst);
    case AV_OPT_TYPE_BINARY:   return set_string_binary(obj, o, val, dst);
    case AV_OPT_TYPE_FLAGS:
    case AV_OPT_TYPE_INT:
    case AV_OPT_TYPE_INT64:
    case AV_OPT_TYPE_FLOAT:
    case AV_OPT_TYPE_DOUBLE:
    case AV_OPT_TYPE_RATIONAL: return set_string_number(obj, target_obj, o, val, dst);
    case AV_OPT_TYPE_IMAGE_SIZE: return set_string_image_size(obj, o, val, dst);
    case AV_OPT_TYPE_VIDEO_RATE: return set_string_video_rate(obj, o, val, dst);
    case AV_OPT_TYPE_PIXEL_FMT:  return set_string_pixel_fmt(obj, o, val, dst);
    case AV_OPT_TYPE_SAMPLE_FMT: return set_string_sample_fmt(obj, o, val, dst);
    case AV_OPT_TYPE_DURATION:
        if (!val) {
            *(int64_t *)dst = 0;
            return 0;
        } else {
            if ((ret = av_parse_time(dst, val, 1)) < 0)
                av_log(obj, AV_LOG_ERROR, "Unable to parse option value \"%s\" as duration\n", val);
            return ret;
        }
        break;
    case AV_OPT_TYPE_COLOR:      return set_string_color(obj, o, val, dst);
    case AV_OPT_TYPE_CHANNEL_LAYOUT:
        if (!val || !strcmp(val, "none")) {
            *(int64_t *)dst = 0;
        } else {
#if FF_API_GET_CHANNEL_LAYOUT_COMPAT
            int64_t cl = ff_get_channel_layout(val, 0);
#else
            int64_t cl = av_get_channel_layout(val);
#endif
            if (!cl) {
                av_log(obj, AV_LOG_ERROR, "Unable to parse option value \"%s\" as channel layout\n", val);
                ret = AVERROR(EINVAL);
            }
            *(int64_t *)dst = cl;
            return ret;
        }
        break;
    }

    av_log(obj, AV_LOG_ERROR, "Invalid option type.\n");
    return AVERROR(EINVAL);
}
```

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。

```cpp
/**
 * Look for an option in an object. Consider only options which
 * have all the specified flags set.
 *
 * @param[in] obj A pointer to a struct whose first element is a
 *                pointer to an AVClass.
 *                Alternatively a double pointer to an AVClass, if
 *                AV_OPT_SEARCH_FAKE_OBJ search flag is set.
 * @param[in] name The name of the option to look for.
 * @param[in] unit When searching for named constants, name of the unit
 *                 it belongs to.
 * @param opt_flags Find only options with all the specified flags set (AV_OPT_FLAG).
 * @param search_flags A combination of AV_OPT_SEARCH_*.
 * @param[out] target_obj if non-NULL, an object to which the option belongs will be
 * written here. It may be different from obj if AV_OPT_SEARCH_CHILDREN is present
 * in search_flags. This parameter is ignored if search_flags contain
 * AV_OPT_SEARCH_FAKE_OBJ.
 *
 * @return A pointer to the option found, or NULL if no option
 *         was found.
 */
const AVOption *av_opt_find2(void *obj, const char *name, const char *unit,
                             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。

```cpp
const AVOption *av_opt_find(void *obj, const char *name, const char *unit,
                            int opt_flags, int search_flags)
{
    return av_opt_find2(obj, name, unit, opt_flags, search_flags, NULL);
}
```

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。

```cpp
const AVOption *av_opt_find2(void *obj, const char *name, const char *unit,
                             int opt_flags, int search_flags, void **target_obj)
{
    const AVClass  *c;
    const AVOption *o = NULL;

    if(!obj)
        return NULL;

    c= *(AVClass**)obj;

    if (!c)
        return NULL;
    //查找范围包含子节点的时候
    //递归调用av_opt_find2()
    if (search_flags & AV_OPT_SEARCH_CHILDREN) {
        if (search_flags & AV_OPT_SEARCH_FAKE_OBJ) {
            const AVClass *child = NULL;
            while (child = av_opt_child_class_next(c, child))
                if (o = av_opt_find2(&child, name, unit, opt_flags, search_flags, NULL))
                    return o;
        } else {
            void *child = NULL;
            while (child = av_opt_child_next(obj, child))
                if (o = av_opt_find2(child, name, unit, opt_flags, search_flags, target_obj))
                    return o;
        }
    }
    //遍历
    while (o = av_opt_next(obj, o)) {
        //比较名称
        if (!strcmp(o->name, name) && (o->flags & opt_flags) == opt_flags &&
            ((!unit && o->type != AV_OPT_TYPE_CONST) ||
             (unit  && o->type == AV_OPT_TYPE_CONST && o->unit && !strcmp(o->unit, unit)))) {
            if (target_obj) {
                if (!(search_flags & AV_OPT_SEARCH_FAKE_OBJ))
                    *target_obj = obj;
                else
                    *target_obj = NULL;
            }
            return o;
        }
    }
    return NULL;
}
```

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
/**
 * Iterate over all AVOptions belonging to obj.
 *
 * @param obj an AVOptions-enabled struct or a double pointer to an
 *            AVClass describing it.
 * @param prev result of the previous call to av_opt_next() on this object
 *             or NULL
 * @return next AVOption or NULL
 */
const AVOption *av_opt_next(void *obj, const AVOption *prev);
```

av_opt_next()的定义如下所示。

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

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
static int set_string(void *obj, const AVOption *o, const char *val, uint8_t **dst)
{
    av_freep(dst);
    *dst = av_strdup(val);
    return 0;
}
```

其中又调用了一个函数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
char *av_strdup(const char *s)
{
    char *ptr = NULL;
    if (s) {
        int len = strlen(s) + 1;
        ptr = av_realloc(NULL, len);
        if (ptr)
            memcpy(ptr, s, len);
    }
    return ptr;
}
```

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
static int set_string_image_size(void *obj, const AVOption *o, const char *val, int *dst)
{
    int ret;

    if (!val || !strcmp(val, "none")) {
        dst[0] =
        dst[1] = 0;
        return 0;
    }
    ret = av_parse_video_size(dst, dst + 1, val);
    if (ret < 0)
        av_log(obj, AV_LOG_ERROR, "Unable to parse option value \"%s\" as image size\n", val);
    return ret;
}
```

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
/**
 * Parse str and put in width_ptr and height_ptr the detected values.
 *
 * @param[in,out] width_ptr pointer to the variable which will contain the detected
 * width value
 * @param[in,out] height_ptr pointer to the variable which will contain the detected
 * height value
 * @param[in] str the string to parse: it has to be a string in the format
 * width x height or a valid video size abbreviation.
 * @return >= 0 on success, a negative error code otherwise
 */
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
//解析分辨率
int av_parse_video_size(int *width_ptr, int *height_ptr, const char *str)
{
    int i;
    int n = FF_ARRAY_ELEMS(video_size_abbrs);
    const char *p;
    int width = 0, height = 0;
    //先看看有没有“分辨率简称”相同的（例如vga，qcif等）
    for (i = 0; i < n; i++) {
        if (!strcmp(video_size_abbrs[i].abbr, str)) {
            width  = video_size_abbrs[i].width;
            height = video_size_abbrs[i].height;
            break;
        }
    }
    //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
    if (i == n) {
        //strtol()：字符串转换成整型，遇到非数字则停止
        width = strtol(str, (void*)&p, 10);
        if (*p)
            p++;
        height = strtol(p, (void*)&p, 10);

        /* trailing extraneous data detected, like in 123x345foobar */
        if (*p)
            return AVERROR(EINVAL);
    }
    //检查一下正确性
    if (width <= 0 || height <= 0)
        return AVERROR(EINVAL);
    *width_ptr  = width;
    *height_ptr = height;
    return 0;
}
```

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
typedef struct {
    const char *abbr;
    int width, height;
} VideoSizeAbbr;

static const VideoSizeAbbr video_size_abbrs[] = {
    { "ntsc",      720, 480 },
    { "pal",       720, 576 },
    { "qntsc",     352, 240 }, /* VCD compliant NTSC */
    { "qpal",      352, 288 }, /* VCD compliant PAL */
    { "sntsc",     640, 480 }, /* square pixel NTSC */
    { "spal",      768, 576 }, /* square pixel PAL */
    { "film",      352, 240 },
    { "ntsc-film", 352, 240 },
    { "sqcif",     128,  96 },
    { "qcif",      176, 144 },
    { "cif",       352, 288 },
    { "4cif",      704, 576 },
    { "16cif",    1408,1152 },
    { "qqvga",     160, 120 },
    { "qvga",      320, 240 },
    { "vga",       640, 480 },
    { "svga",      800, 600 },
    { "xga",      1024, 768 },
    { "uxga",     1600,1200 },
    { "qxga",     2048,1536 },
    { "sxga",     1280,1024 },
    { "qsxga",    2560,2048 },
    { "hsxga",    5120,4096 },
    { "wvga",      852, 480 },
    { "wxga",     1366, 768 },
    { "wsxga",    1600,1024 },
    { "wuxga",    1920,1200 },
    { "woxga",    2560,1600 },
    { "wqsxga",   3200,2048 },
    { "wquxga",   3840,2400 },
    { "whsxga",   6400,4096 },
    { "whuxga",   7680,4800 },
    { "cga",       320, 200 },
    { "ega",       640, 350 },
    { "hd480",     852, 480 },
    { "hd720",    1280, 720 },
    { "hd1080",   1920,1080 },
    { "2k",       2048,1080 }, /* Digital Cinema System Specification */
    { "2kflat",   1998,1080 },
    { "2kscope",  2048, 858 },
    { "4k",       4096,2160 }, /* Digital Cinema System Specification */
    { "4kflat",   3996,2160 },
    { "4kscope",  4096,1716 },
    { "nhd",       640,360  },
    { "hqvga",     240,160  },
    { "wqvga",     400,240  },
    { "fwqvga",    432,240  },
    { "hvga",      480,320  },
    { "qhd",       960,540  },
};
```

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。

```cpp
/**
 * Set the values of all AVOption fields to their default values.
 *
 * @param s an AVOption-enabled struct (its first member must be a pointer to AVClass)
 */
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。

```cpp
void av_opt_set_defaults(void *s)
{
//奇怪的#if...#endif
#if FF_API_OLD_AVOPTIONS
    av_opt_set_defaults2(s, 0, 0);
}

void av_opt_set_defaults2(void *s, int mask, int flags)
{
#endif
    const AVOption *opt = NULL;
    //遍历所有的AVOption
    while ((opt = av_opt_next(s, opt))) {
        //注意：offset的使用
        void *dst = ((uint8_t*)s) + opt->offset;
#if FF_API_OLD_AVOPTIONS
        if ((opt->flags & mask) != flags)
            continue;
#endif

        if (opt->flags & AV_OPT_FLAG_READONLY)
            continue;
        //读取各种default_val
        switch (opt->type) {
            case AV_OPT_TYPE_CONST:
                /* Nothing to be done here */
            break;
            case AV_OPT_TYPE_FLAGS:
            case AV_OPT_TYPE_INT:
            case AV_OPT_TYPE_INT64:
            case AV_OPT_TYPE_DURATION:
            case AV_OPT_TYPE_CHANNEL_LAYOUT:
                write_number(s, opt, dst, 1, 1, opt->default_val.i64);
            break;
            case AV_OPT_TYPE_DOUBLE:
            case AV_OPT_TYPE_FLOAT: {
                double val;
                val = opt->default_val.dbl;
                write_number(s, opt, dst, val, 1, 1);
            }
            break;
            case AV_OPT_TYPE_RATIONAL: {
                AVRational val;
                val = av_d2q(opt->default_val.dbl, INT_MAX);
                write_number(s, opt, dst, 1, val.den, val.num);
            }
            break;
            case AV_OPT_TYPE_COLOR:
                set_string_color(s, opt, opt->default_val.str, dst);
                break;
            case AV_OPT_TYPE_STRING:
                set_string(s, opt, opt->default_val.str, dst);
                break;
            case AV_OPT_TYPE_IMAGE_SIZE:
                set_string_image_size(s, opt, opt->default_val.str, dst);
                break;
            case AV_OPT_TYPE_VIDEO_RATE:
                set_string_video_rate(s, opt, opt->default_val.str, dst);
                break;
            case AV_OPT_TYPE_PIXEL_FMT:
                write_number(s, opt, dst, 1, 1, opt->default_val.i64);
                break;
            case AV_OPT_TYPE_SAMPLE_FMT:
                write_number(s, opt, dst, 1, 1, opt->default_val.i64);
                break;
            case AV_OPT_TYPE_BINARY:
                set_string_binary(s, opt, opt->default_val.str, dst);
                break;
            case AV_OPT_TYPE_DICT:
                /* Cannot set defaults for these types */
            break;
            default:
                av_log(s, AV_LOG_DEBUG, "AVOption type %d of option %s not implemented yet\n", opt->type, opt->name);
        }
    }
}
```

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
const AVOption *av_opt_next(void *obj, const AVOption *last)
{
    AVClass *class = *(AVClass**)obj;
    if (!last && class && class->option && class->option[0].name)
        return class->option;
    if (last && last[1].name)
        return ++last;
    return NULL;
}
```

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
/**
 * @defgroup opt_get_funcs Option getting functions
 * @{
 * Those functions get a value of the option with the given name from an object.
 *
 * @param[in] obj a struct whose first element is a pointer to an AVClass.
 * @param[in] name name of the option to get.
 * @param[in] search_flags flags passed to av_opt_find2. I.e. if AV_OPT_SEARCH_CHILDREN
 * is passed here, then the option may be found in a child of obj.
 * @param[out] out_val value of the option will be written here
 * @return >=0 on success, a negative error code otherwise
 */
/**
 * @note the returned string will be av_malloc()ed and must be av_free()ed by the caller
 */
int av_opt_get         (void *obj, const char *name, int search_flags, uint8_t   **out_val);
int av_opt_get_int     (void *obj, const char *name, int search_flags, int64_t    *out_val);
int av_opt_get_double  (void *obj, const char *name, int search_flags, double     *out_val);
int av_opt_get_q       (void *obj, const char *name, int search_flags, AVRational *out_val);
int av_opt_get_image_size(void *obj, const char *name, int search_flags, int *w_out, int *h_out);
int av_opt_get_pixel_fmt (void *obj, const char *name, int search_flags, enum AVPixelFormat *out_fmt);
int av_opt_get_sample_fmt(void *obj, const char *name, int search_flags, enum AVSampleFormat *out_fmt);
int av_opt_get_video_rate(void *obj, const char *name, int search_flags, AVRational *out_val);
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。

```cpp
int av_opt_get(void *obj, const char *name, int search_flags, uint8_t **out_val)
{
    void *dst, *target_obj;
    //查找
    const AVOption *o = av_opt_find2(obj, name, NULL, 0, search_flags, &target_obj);
    uint8_t *bin, buf[128];
    int len, i, ret;
    int64_t i64;

    if (!o || !target_obj || (o->offset<=0 && o->type != AV_OPT_TYPE_CONST))
        return AVERROR_OPTION_NOT_FOUND;
    //注意：offset的使用
    dst = (uint8_t*)target_obj + o->offset;
    //使用sprintf()转换成字符串，存入buf
    buf[0] = 0;
    switch (o->type) {
    case AV_OPT_TYPE_FLAGS:     ret = snprintf(buf, sizeof(buf), "0x%08X",  *(int    *)dst);break;
    case AV_OPT_TYPE_INT:       ret = snprintf(buf, sizeof(buf), "%d" ,     *(int    *)dst);break;
    case AV_OPT_TYPE_INT64:     ret = snprintf(buf, sizeof(buf), "%"PRId64, *(int64_t*)dst);break;
    case AV_OPT_TYPE_FLOAT:     ret = snprintf(buf, sizeof(buf), "%f" ,     *(float  *)dst);break;
    case AV_OPT_TYPE_DOUBLE:    ret = snprintf(buf, sizeof(buf), "%f" ,     *(double *)dst);break;
    case AV_OPT_TYPE_VIDEO_RATE:
    case AV_OPT_TYPE_RATIONAL:  ret = snprintf(buf, sizeof(buf), "%d/%d",   ((AVRational*)dst)->num, ((AVRational*)dst)->den);break;

    case AV_OPT_TYPE_CONST:     ret = snprintf(buf, sizeof(buf), "%f" ,     o->default_val.dbl);break;
    case AV_OPT_TYPE_STRING:
        if (*(uint8_t**)dst)
            *out_val = av_strdup(*(uint8_t**)dst);
        else
            *out_val = av_strdup("");
        return 0;
    case AV_OPT_TYPE_BINARY:
        len = *(int*)(((uint8_t *)dst) + sizeof(uint8_t *));
        if ((uint64_t)len*2 + 1 > INT_MAX)
            return AVERROR(EINVAL);
        if (!(*out_val = av_malloc(len*2 + 1)))
            return AVERROR(ENOMEM);
        if (!len) {
            *out_val[0] = '\0';
            return 0;
        }
        bin = *(uint8_t**)dst;
        for (i = 0; i < len; i++)
            snprintf(*out_val + i*2, 3, "%02X", bin[i]);
        return 0;
    case AV_OPT_TYPE_IMAGE_SIZE:
        //分辨率
        ret = snprintf(buf, sizeof(buf), "%dx%d", ((int *)dst)[0], ((int *)dst)[1]);
        break;
    case AV_OPT_TYPE_PIXEL_FMT:
        //像素格式
        ret = snprintf(buf, sizeof(buf), "%s", (char *)av_x_if_null(av_get_pix_fmt_name(*(enum AVPixelFormat *)dst), "none"));
        break;
    case AV_OPT_TYPE_SAMPLE_FMT:
        //采样格式
        ret = snprintf(buf, sizeof(buf), "%s", (char *)av_x_if_null(av_get_sample_fmt_name(*(enum AVSampleFormat *)dst), "none"));
        break;
    case AV_OPT_TYPE_DURATION:
        //时长
        i64 = *(int64_t *)dst;
        ret = snprintf(buf, sizeof(buf), "%"PRIi64":%02d:%02d.%06d",
                       i64 / 3600000000, (int)((i64 / 60000000) % 60),
                       (int)((i64 / 1000000) % 60), (int)(i64 % 1000000));
        break;
    case AV_OPT_TYPE_COLOR:
        ret = snprintf(buf, sizeof(buf), "0x%02x%02x%02x%02x",
                       (int)((uint8_t *)dst)[0], (int)((uint8_t *)dst)[1],
                       (int)((uint8_t *)dst)[2], (int)((uint8_t *)dst)[3]);
        break;
    case AV_OPT_TYPE_CHANNEL_LAYOUT:
        i64 = *(int64_t *)dst;
        ret = snprintf(buf, sizeof(buf), "0x%"PRIx64, i64);
        break;
    default:
        return AVERROR(EINVAL);
    }

    if (ret >= sizeof(buf))
        return AVERROR(EINVAL);
    //拷贝
    *out_val = av_strdup(buf);
    return 0;
}
```

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snprintf()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