

=====

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

=====

打算写两篇文章记录FFmpeg中和AVOption有关的源代码。AVOption用于在FFmpeg中描述结构体中的成员变量。它最主要的作用可以概括为两个字：“赋值”。一个AVOption结构体包含了变量名称，简短的帮助，取值等等信息。

所有和AVOption有关的数据都存储在AVClass结构体中。如果一个结构体（例如AVFormatContext或者AVCodecContext）想要支持AVOption的话，它的第一个成员变量必须是一个指向AVClass结构体的指针。该AVClass中的成员变量option必须指向一个AVOption类型的静态数组。

何为AVOption？

AVOption是用来设置FFmpeg中变量的值的结构体。可能说到这个作用有的人会奇怪：设置系统中变量的值，直接使用等于号“=”就可以，为什么还要专门定义一个结构体呢？其实AVOption的特点就在于它赋值时候的灵活性。AVOption可以使用字符串为任何类型的变量赋值。传统意义上，如果变量类型为int，则需要使用整数来赋值；如果变量为double，则需要使用小数来赋值；如果变量类型为char*，才需要使用字符串来赋值。而AVOption将这些赋值“归一化”了，统一使用字符串赋值。例如给int型变量qp设定值为20，通过AVOption需要传递进去一个内容为“20”的字符串。

此外，AVOption中变量的名称也使用字符串来表示。结合上面提到的使用字符串赋值的特性，我们可以发现使用AVOption之后，传递两个字符串（一个是变量的名称，一个是变量的值）就可以改变系统中变量的值。

上文提到的这种方法的意义在哪里？我个人感觉对于直接使用C语言进行开发的人来说，作用不是很明显：完全可以使用等于号“=”就可以进行各种变量的赋值。但是对于从外部系统中调用FFmpeg的人来说，作用就很大了：从外部系统中只可以传递字符串给内部系统。比如说对于直接调用ffmpeg.exe的人来说，他们是无法修改FFmpeg内部各个变量的数值的，这种情况下只能通过输入“名称”和“值”这样的字符串，通过AVOption改变FFmpeg内部变量的值。由此可见，使用AVOption可以使FFmpeg更加适应多种多样的外部系统。

突然想到了JavaEE开发中也有这种类似的机制。互联网上只可以传输字符串，即是没有方法传输整形、浮点型这种的数据。而Java系统中却包含整形、浮点型等各种数据类型。因此开发JSP中的Servlet的时候经常需要将整数字符串手工转化成一个整型的变量。使用最多的一个函数就是Integer.parseInt()方法。例如下面代码可以将字符串“123”转化成整数123。

```
int a=Integer.parseInt("123");
```

而在使用JavaEE中的Struts2进行开发的时候，就不需要进行手动转换处理了。Struts2中包含了类似AVOption的这种数据类型自动转换机制，可以将互联网上收到的字符串“名称”和“值”的组合自动赋值给相应名称的变量。

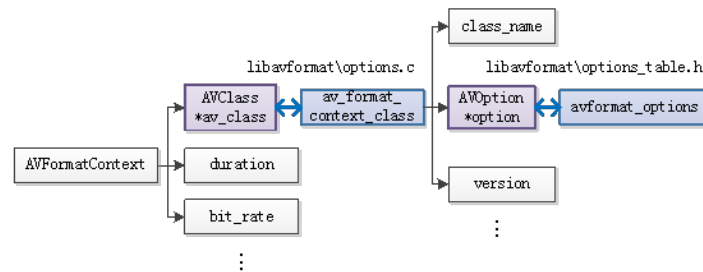
由此发现了一个结论：编程语言之间真的是相通的！

现在回到AVOption。其实除了可以对FFmpeg常用结构体AVFormatContext，AVCodecContext等进行赋值之外，还可以对它们的私有数据priv_data进行赋值。这个字段里通常存储了各种编码器特有的结构体。而这些结构体的定义在FFmpeg的SDK中是找不到的。例如使用libx264进行编码的时候，通过AVCodecContext的priv_data字段可以对X264Context结构体中的变量进行赋值，设置preset，profile等。使用libx265进行编码的时候，通过AVCodecContext的priv_data字段可以对libx265Context结构体中的变量进行赋值，设置preset，tune等。

何为AVClass？

AVClass最主要的作用就是给结构体（例如AVFormatContext等）增加AVOption功能的支持。换句话说AVClass就是AVOption和目标结构体之间的“桥梁”。AVClass要求必须声明为目标结构体的第一个变量。

AVClass中有一个option数组用于存储目标结构体的所有的AVOption。举个例子，AVFormatContext结构体，AVClass和AVOption之间的关系如下图所示。



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

图中AVFormatContext结构体的第一个变量为AVClass类型的指针av_class，它在AVFormatContext结构体初始化的时候，被赋值指向了全局静态变量av_format_context_class结构体（定义位于libavformat/options.c）。而AVClass类型的av_format_context_class结构体中的option变量指向了全局静态数组avformat_options（定义位于libavformat/options_table.h）。

AVOption

下面开始从代码的角度记录AVOption。AVOption结构体的定义如下所示。

```

1.  /**
2.   * AVOption
3.   */
4.  typedef struct AVOption {
5.      const char *name;
6.
7.      /**
8.       * short English help text
9.       * @todo What about other languages?
10.     */
11.     const char *help;
12.
13.     /**
14.      * The offset relative to the context structure where the option
15.      * value is stored. It should be 0 for named constants.
16.     */
17.     int offset;
18.     enum AVOptionType type;
19.
20.     /**
21.      * the default value for scalar options
22.     */
23.     union {
24.         int64_t i64;
25.         double dbl;
26.         const char *str;
27.         /* TODO those are unused now */
28.         AVRational q;
29.     } default_val;
30.     double min;           ///< minimum valid value for the option
31.     double max;           ///< maximum valid value for the option
32.
33.     int flags;
34.     #define AV_OPT_FLAG_ENCODING_PARAM 1 ///< a generic parameter which can be set by the user for muxing or encoding
35.     #define AV_OPT_FLAG_DECODING_PARAM 2 ///< a generic parameter which can be set by the user for demuxing or decoding
36.     #if FF_API_OPT_TYPE_METADATA
37.     #define AV_OPT_FLAG_METADATA 4 ///< some data extracted or inserted into the file like title, comment, ...
38.     #endif
39.     #define AV_OPT_FLAG_AUDIO_PARAM 8
40.     #define AV_OPT_FLAG_VIDEO_PARAM 16
41.     #define AV_OPT_FLAG_SUBTITLE_PARAM 32
42.     /**
43.      * The option is intended for exporting values to the caller.
44.     */
45.     #define AV_OPT_FLAG_EXPORT 64
46.     /**
47.      * The option may not be set through the AVOptions API, only read.
48.      * This flag only makes sense when AV_OPT_FLAG_EXPORT is also set.
49.     */
50.     #define AV_OPT_FLAG_READONLY 128
51.     #define AV_OPT_FLAG_FILTERING_PARAM (1<16) ///< a generic parameter which can be set by the user for filtering
52.     //FIXME think about enc-audio, ... style flags
53.
54.     /**
55.      * The logical unit to which the option belongs. Non-constant
56.      * options and corresponding named constants share the same
57.      * unit. May be NULL.
58.     */
59.     const char *unit;
60. } AVOption;

```

下面简单解释一下AVOption的几个成员变量：

- name：名称。
- help：简短的帮助。
- offset：选项相对结构体首部地址的偏移量（这个很重要）。
- type：选项的类型。
- default_val：选项的默认值。
- min：选项的最小值。
- max：选项的最大值。
- flags：一些标记。
- unit：该选项所属的逻辑单元，可以为空。

其中，default_val是一个union类型的变量，可以根据选项数据类型的不同，取int，double，char*，AVRational（表示分数）几种类型。type是一个AVOptionType类型的变量。AVOptionType是一个枚举类型，定义如下。



```
[cpp]
1.  enum AVOptionType{
2.      AV_OPT_TYPE_FLAGS,
3.      AV_OPT_TYPE_INT,
4.      AV_OPT_TYPE_INT64,
5.      AV_OPT_TYPE_DOUBLE,
6.      AV_OPT_TYPE_FLOAT,
7.      AV_OPT_TYPE_STRING,
8.      AV_OPT_TYPE_RATIONAL,
9.      AV_OPT_TYPE_BINARY, ///< offset must point to a pointer immediately followed by an int for the length
10.     AV_OPT_TYPE_DICT,
11.     AV_OPT_TYPE_CONST = 128,
12.     AV_OPT_TYPE_IMAGE_SIZE = MKBETAG('S','I','Z','E'), ///< offset must point to two consecutive integers
13.     AV_OPT_TYPE_PIXEL_FMT = MKBETAG('P','F','M','T'),
14.     AV_OPT_TYPE_SAMPLE_FMT = MKBETAG('S','F','M','T'),
15.     AV_OPT_TYPE_VIDEO_RATE = MKBETAG('V','R','A','T'), ///< offset must point to AVRational
16.     AV_OPT_TYPE_DURATION = MKBETAG('D','U','R',' '),
17.     AV_OPT_TYPE_COLOR = MKBETAG('C','O','L','R'),
18.     AV_OPT_TYPE_CHANNEL_LAYOUT = MKBETAG('C','H','L','A'),
19. #if FF_API_OLD_AVOPTIONS
20.     FF_OPT_TYPE_FLAGS = 0,
21.     FF_OPT_TYPE_INT,
22.     FF_OPT_TYPE_INT64,
23.     FF_OPT_TYPE_DOUBLE,
24.     FF_OPT_TYPE_FLOAT,
25.     FF_OPT_TYPE_STRING,
26.     FF_OPT_TYPE_RATIONAL,
27.     FF_OPT_TYPE_BINARY, ///< offset must point to a pointer immediately followed by an int for the length
28.     FF_OPT_TYPE_CONST=128,
29. #endif
30. };
```

AVClass

AVClass中存储了AVOption类型的数组option，用于存储选项信息。AVClass有一个特点就是它必须位于其支持的结构体的第一个位置。例如，AVFormatContext和AVCodecContext都支持AVClass，观察它们结构体的定义可以发现他们结构体的第一个变量都是AVClass。截取一小段AVFormatContext的定义的开头部分，如下所示。

```
[cpp]
1.  typedef struct AVFormatContext {
2.      /**
3.       * A class for logging and @ref avoptions. Set by avformat_alloc_context().
4.       * Exports (de)muxer private options if they exist.
5.       */
6.      const AVClass *av_class;
7.
8.      /**
9.       * The input container format.
10.      *
11.      * Demuxing only, set by avformat_open_input().
12.      */
13.      struct AVInputFormat *iformat;
14.
15.      /**
16.       * The output container format.
17.       *
18.       * Muxing only, must be set by the caller before avformat_write_header().
19.       */
20.      struct AVOutputFormat *oformat;
21.      //后文略
```

截取一小段AVCodecContext的定义的开头部分，如下所示。

```
[cpp]    
1. typedef struct AVCodecContext {  
2.     /**  
3.      * information on struct for av_log  
4.      * - set by avcodec_alloc_context3  
5.      */  
6.     const AVClass *av_class;  
7.     int log_level_offset;  
8.  
9.     enum AVMediaType codec_type; /* see AVMEDIA_TYPE_xxx */  
10.    const struct AVCodec *codec;  
11.    //后文略
```

下面来看一下AVClass的定义，如下所示。

```

1.  /**
2.   * Describe the class of an AVClass context structure. That is an
3.   * arbitrary struct of which the first field is a pointer to an
4.   * AVClass struct (e.g. AVCodecContext, AVFormatContext etc.).
5.   */
6.  typedef struct AVClass {
7.      /**
8.       * The name of the class; usually it is the same name as the
9.       * context structure type to which the AVClass is associated.
10.      */
11.      const char* class_name;
12.
13.      /**
14.       * A pointer to a function which returns the name of a context
15.       * instance ctx associated with the class.
16.      */
17.      const char* (*item_name)(void* ctx);
18.
19.      /**
20.       * a pointer to the first option specified in the class if any or NULL
21.       *
22.       * @see av_set_default_options()
23.       */
24.      const struct AVOption *option;
25.
26.      /**
27.       * LIBAVUTIL_VERSION with which this structure was created.
28.       * This is used to allow fields to be added without requiring major
29.       * version bumps everywhere.
30.       */
31.
32.      int version;
33.
34.      /**
35.       * Offset in the structure where log_level_offset is stored.
36.       * 0 means there is no such variable
37.       */
38.      int log_level_offset_offset;
39.
40.      /**
41.       * Offset in the structure where a pointer to the parent context for
42.       * logging is stored. For example a decoder could pass its AVCodecContext
43.       * to eval as such a parent context, which an av_log() implementation
44.       * could then leverage to display the parent context.
45.       * The offset can be NULL.
46.       */
47.      int parent_log_context_offset;
48.
49.      /**
50.       * Return next AVOptions-enabled child or NULL
51.       */
52.      void* (*child_next)(void *obj, void *prev);
53.
54.      /**
55.       * Return an AVClass corresponding to the next potential
56.       * AVOptions-enabled child.
57.       *
58.       * The difference between child_next and this is that
59.       * child_next iterates over _already existing_ objects, while
60.       * child_class_next iterates over _all possible_ children.
61.       */
62.      const struct AVClass* (*child_class_next)(const struct AVClass *prev);
63.
64.      /**
65.       * Category used for visualization (like color)
66.       * This is only set if the category is equal for all objects using this class.
67.       * available since version (51 <= 16 | 56 <= 8 | 100)
68.       */
69.      AVClassCategory category;
70.
71.      /**
72.       * Callback to return the category.
73.       * available since version (51 <= 16 | 59 <= 8 | 100)
74.       */
75.      AVClassCategory (*get_category)(void* ctx);
76.
77.      /**
78.       * Callback to return the supported/allowed ranges.
79.       * available since version (52.12)
80.       */
81.      int (*query_ranges)(struct AVOptionRanges **, void *obj, const char *key, int flags);
82.  } AVClass;

```

下面简单解释一下AVClass的几个已经理解的成员变量：

class_name：AVClass名称。

item_name：函数，获取与AVClass相关联的结构体实例的名称。

option：AVOption类型的数组（最重要）。

version：完成该AVClass的时候的LIBAVUTIL_VERSION。

category：AVClass的类型，是一个类型为AVClassCategory的枚举型变量。

其中AVClassCategory定义如下。

```
[cpp]
1. typedef enum {
2.     AV_CLASS_CATEGORY_NA = 0,
3.     AV_CLASS_CATEGORY_INPUT,
4.     AV_CLASS_CATEGORY_OUTPUT,
5.     AV_CLASS_CATEGORY_MUXER,
6.     AV_CLASS_CATEGORY_DEMUXER,
7.     AV_CLASS_CATEGORY_ENCODER,
8.     AV_CLASS_CATEGORY_DECODER,
9.     AV_CLASS_CATEGORY_FILTER,
10.    AV_CLASS_CATEGORY_BITSTREAM_FILTER,
11.    AV_CLASS_CATEGORY_SWSCALER,
12.    AV_CLASS_CATEGORY_SWRESAMPLER,
13.    AV_CLASS_CATEGORY_DEVICE_VIDEO_OUTPUT = 40,
14.    AV_CLASS_CATEGORY_DEVICE_VIDEO_INPUT,
15.    AV_CLASS_CATEGORY_DEVICE_AUDIO_OUTPUT,
16.    AV_CLASS_CATEGORY_DEVICE_AUDIO_INPUT,
17.    AV_CLASS_CATEGORY_DEVICE_OUTPUT,
18.    AV_CLASS_CATEGORY_DEVICE_INPUT,
19.    AV_CLASS_CATEGORY_NB, ///< not part of ABI/API
20. }AVClassCategory;
```

上面解释字段还是比较抽象的，下面通过具体的例子看一下AVClass这个结构体。我们看几个具体的例子：

- AVFormatContext中的AVClass
- AVCodecContext中的AVClass
- AVFrame中的AVClass
- 各种组件（libRTMP，libx264，libx265）里面特有的AVClass。

AVFormatContext

AVFormatContext 中的AVClass定义位于libavformatoptions.c中，是一个名称为av_format_context_class的静态结构体。如下所示。

```
[cpp]
1. static const AVClass av_format_context_class = {
2.     .class_name     = "AVFormatContext",
3.     .item_name      = format_to_name,
4.     .option         = avformat_options,
5.     .version        = LIBAVUTIL_VERSION_INT,
6.     .child_next     = format_child_next,
7.     .child_class_next = format_child_class_next,
8.     .category       = AV_CLASS_CATEGORY_MUXER,
9.     .get_category   = get_category,
10. };
```

从源代码可以看出以下几点

(1) class_name

该AVClass名称是“AVFormatContext”。

(2) item_name

item_name指向一个函数format_to_name()，该函数定义如下所示。

```
[cpp]
1. static const char* format_to_name(void* ptr)
2. {
3.     AVFormatContext* fc = (AVFormatContext*) ptr;
4.     if(fc->iformat) return fc->iformat->name;
5.     else if(fc->oformat) return fc->oformat->name;
6.     else return "NULL";
7. }
```

从函数的定义可以看出，如果AVFormatContext结构体中的AVInputFormat结构体不为空，则返回AVInputFormat的name，然后尝试返回AVOutputFormat的name，如果AVOutputFormat也为空，则返回“NULL”。

(3) option

option字段则指向一个元素个数很多的静态数组avformat_options。该数组单独定义于libavformat\options_table.h中。其中包含了AVFormatContext支持的所有的AVOption，如下所示。

[cpp]  

```
1.  /*
2.  * 雷霄骅
3.  * leixiaohua1020@126.com
4.  * 中国传媒大学/数字电视技术
5.  * http://blog.csdn.net/leixiaohua1020
6.  *
7.  */
8.
9. #ifndef AVFORMAT_OPTIONS_TABLE_H
10. #define AVFORMAT_OPTIONS_TABLE_H
11.
12. #include <limits.h>
13.
14. #include "libavutil/opt.h"
15. #include "avformat.h"
16. #include "internal.h"
17.
18. #define OFFSET(x) offsetof(AVFormatContext,x)
19. #define DEFAULT 0 //should be NAN but it does not work as it is not a constant in glibc as required by ANSI/ISO C
20. //these names are too long to be readable
21. #define E AV_OPT_FLAG_ENCODING_PARAM
22. #define D AV_OPT_FLAG_DECODING_PARAM
23.
24. static const AVOption avformat_options[] = {
25.     {"avioflags", NULL, OFFSET(avio_flags), AV_OPT_TYPE_FLAGS, {.i64 = DEFAULT }, INT_MIN, INT_MAX, D|E, "avioflags"},
26.     {"direct", "reduce buffering", 0, AV_OPT_TYPE_CONST, {.i64 = AVIO_FLAG_DIRECT }, INT_MIN, INT_MAX, D|E, "avioflags"},
27.     {"probesize", "set probing size", OFFSET(probesize2), AV_OPT_TYPE_INT64, {.i64 = 5000000 }, 32, INT64_MAX, D},
28.     {"formatprobesize", "number of bytes to probe file format", OFFSET(format_probesize), AV_OPT_TYPE_INT, {.i64 = PROBE_BUF_MAX}, 0, INT_MAX-1, D},
29.     {"packetsize", "set packet size", OFFSET(packet_size), AV_OPT_TYPE_INT, {.i64 = DEFAULT }, 0, INT_MAX, E},
30.     {"fflags", NULL, OFFSET(flags), AV_OPT_TYPE_FLAGS, {.i64 = AVFMT_FLAG_FLUSH_PACKETS }, INT_MIN, INT_MAX, D|E, "fflags"},
31.     {"flush_packets", "reduce the latency by flushing out packets immediately", 0, AV_OPT_TYPE_CONST, {.i64 = AVFMT_FLAG_FLUSH_PACKETS }, INT_MIN, INT_MAX, E, "fflags"},
32.     {"ignidx", "ignore index", 0, AV_OPT_TYPE_CONST, {.i64 = AVFMT_FLAG_IGNIDX }, INT_MIN, INT_MAX, D, "fflags"},
33.     {"genpts", "generate pts", 0, AV_OPT_TYPE_CONST, {.i64 = AVFMT_FLAG_GENPTS }, INT_MIN, INT_MAX, D, "fflags"},
34.     {"nofillin", "do not fill in missing values that can be exactly calculated", 0, AV_OPT_TYPE_CONST, {.i64 = AVFMT_FLAG_NOFILLIN }, INT_MIN, INT_MAX, D, "fflags"},
35.     {"noparse", "disable AVParasers, this needs nofillin too", 0, AV_OPT_TYPE_CONST, {.i64 = AVFMT_FLAG_NOPARSE }, INT_MIN, INT_MAX, D, "fflags"},
36.     {"igndts", "ignore dts", 0, AV_OPT_TYPE_CONST, {.i64 = AVFMT_FLAG_IGNDTS }, INT_MIN, INT_MAX, D, "fflags"},
37.     {"discardcorrupt", "discard corrupted frames", 0, AV_OPT_TYPE_CONST, {.i64 = AVFMT_FLAG_DISCARD_CORRUPT }, INT_MIN, INT_MAX, D, "fflags"},
38.     {"sortdts", "try to interleave outputted packets by dts", 0, AV_OPT_TYPE_CONST, {.i64 = AVFMT_FLAG_SORT_DTS }, INT_MIN, INT_MAX, D, "fflags"},
39.     {"keepside", "don't merge side data", 0, AV_OPT_TYPE_CONST, {.i64 = AVFMT_FLAG_KEEP_SIDE_DATA }, INT_MIN, INT_MAX, D, "fflags"},
40.     {"latm", "enable RTP MP4A-LATM payload", 0, AV_OPT_TYPE_CONST, {.i64 = AVFMT_FLAG_MP4A_LATM }, INT_MIN, INT_MAX, E, "fflags"},
41.     {"nobuffer", "reduce the latency introduced by optional buffering", 0, AV_OPT_TYPE_CONST, {.i64 = AVFMT_FLAG_NOBUFFER }, 0, INT_MAX, E, "fflags"},
42.     {"seek2any", "allow seeking to non-keyframes on demuxer level when supported", OFFSET(seek2any), AV_OPT_TYPE_INT, {.i64 = 0 }, 0, 1, D},
43.     {"bitexact", "do not write random/volatile data", 0, AV_OPT_TYPE_CONST, {.i64 = AVFMT_FLAG_BITEXACT }, 0, 0, E, "fflags"},
44.     {"analyzeduration", "specify how many microseconds are analyzed to probe the input", OFFSET(max_analyze_duration2), AV_OPT_TYPE_INT64, {.i64 = 0 }, 0, INT64_MAX, D},
45.     {"cryptokey", "decryption key", OFFSET(key), AV_OPT_TYPE_BINARY, {.dbl = 0}, 0, 0, D},
46.     {"indexmem", "max memory used for timestamp index (per stream)", OFFSET(max_index_size), AV_OPT_TYPE_INT, {.i64 = 1<<20 }, 0, INT_MAX, D},
47.     {"rtbufsize", "max memory used for buffering real-time frames", OFFSET(max_picture_buffer), AV_OPT_TYPE_INT, {.i64 = 3041280 }, 0, INT_MAX, D}, /* defaults to 1s of 15fps 352x288 YUYV 422 video */
48.     {"fdebug", "print specific debug info", OFFSET(debug), AV_OPT_TYPE_FLAGS, {.i64 = DEFAULT }, 0, INT_MAX, E|D, "fdebug"},
49.     {"ts", NULL, 0, AV_OPT_TYPE_CONST, {.i64 = FF_FDEBUG_TS }, INT_MIN, INT_MAX, E|D, "fdebug"},
50.     {"max_delay", "maximum muxing or demuxing delay in microseconds", OFFSET(max_delay), AV_OPT_TYPE_INT, {.i64 = -1 }, -1, INT_MAX, E|D},
51.     {"start_time_realtime", "wall-clock time when stream begins (PTS==0)", OFFSET(start_time_realtime), AV_OPT_TYPE_INT64, {.i64 = AV_NOPTS_VALUE}, INT64_MIN, INT64_MAX, E},
52.     {"fpsprobesize", "number of frames used to probe fps", OFFSET(fps_probe_size), AV_OPT_TYPE_INT, {.i64 = -1}, -1, INT_MAX-1, D},
53.     {"audio_preload", "microseconds by which audio packets should be interleaved earlier", OFFSET(audio_preload), AV_OPT_TYPE_INT, {.i64 = 0}, 0, INT_MAX-1, E},
54.     {"chunk_duration", "microseconds for each chunk", OFFSET(max_chunk_duration), AV_OPT_TYPE_INT, {.i64 = 0}, 0, INT_MAX-1, E},
55.     {"chunk_size", "size in bytes for each chunk", OFFSET(max_chunk_size), AV_OPT_TYPE_INT, {.i64 = 0}, 0, INT_MAX-1, E},
56.     /* this is a crutch for avconv, since it cannot deal with identically named options in different contexts.
57.      * to be removed when avconv is fixed */
58.     {"f_err_detect", "set error detection flags (deprecated; use err_detect, save via avconv)", OFFSET(error_recognition), AV_OPT_TYPE_FLAGS, {.i64 = AV_EF_CRC_CHECK }, INT_MIN, INT_MAX, D, "err_detect"},
59.     {"err_detect", "set error detection flags", OFFSET(error_recognition), AV_OPT_TYPE_FLAGS, {.i64 = AV_EF_CRC_CHECK }, INT_MIN, INT_MAX, D, "err_detect"},
60.     {"crccheck", "verify embedded CRCs", 0, AV_OPT_TYPE_CONST, {.i64 = AV_EF_CRC_CHECK }, INT_MIN, INT_MAX, D, "err_detect"},
61.     {"bitstream", "detect bitstream specification deviations", 0, AV_OPT_TYPE_CONST, {.i64 = AV_EF_BITSTREAM }, INT_MIN, INT_MAX, D, "err_detect"},
62.     {"buffer", "detect improper bitstream length", 0, AV_OPT_TYPE_CONST, {.i64 = AV_EF_BUFFER }, INT_MIN, INT_MAX, D, "err_detect"},
63.     {"explode", "abort decoding on minor error detection", 0, AV_OPT_TYPE_CONST, {.i64 = AV_EF_EXPLODE }, INT_MIN, INT_MAX, D, "err_detect"},
64.     {"ignore_err", "ignore errors", 0, AV_OPT_TYPE_CONST, {.i64 = AV_EF_IGNORE_ERR }, INT_MIN, INT_MAX, D, "err_detect"},
65.     {"careful", "consider things that violate the spec, are fast to check and have not been seen in the wild as errors", 0, AV_OPT_TYPE_CONST, {.i64 = AV_EF_CAREFUL }, INT_MIN, INT_MAX, D, "err_detect"},
66.     {"compliant", "consider all spec non-compliances as errors", 0, AV_OPT_TYPE_CONST, {.i64 = AV_EF_COMPLIANT }, INT_MIN, INT_MAX, D, "err_detect"},
67.     {"aggressive", "consider things that a sane encoder shouldn't do as an error", 0, AV_OPT_TYPE_CONST, {.i64 = AV_EF_AGGRESSIVE }, INT_MIN, INT_MAX, D, "err_detect"}

```



```

68.     {"use_wallclock_as_timestamps", "use wallclock as timestamps", OFFSET(use_wallclock_as_timestamps), AV_OPT_TYPE_INT, {.i64 = 0}, 0, INT_MAX-1, D},
69.     {"avoid_negative_ts", "shift timestamps so they start at 0", OFFSET(avoid_negative_ts), AV_OPT_TYPE_INT, {.i64 = -1}, -1, 2, E, "avoid_id_negative_ts"},
70.     {"auto", "enabled when required by target format", 0, AV_OPT_TYPE_CONST, {.i64 = -1 }, INT_MIN, INT_MAX, E, "avoid_negative_ts"},
71.     {"disabled", "do not change timestamps", 0, AV_OPT_TYPE_CONST, {.i64 = 0 }, INT_MIN, INT_MAX, E, "avoid_negative_ts"},
72.     {"make_zero", "shift timestamps so they start at 0", 0, AV_OPT_TYPE_CONST, {.i64 = 2 }, INT_MIN, INT_MAX, E, "avoid_negative_ts"},
73.     {"make_non_negative", "shift timestamps so they are non negative", 0, AV_OPT_TYPE_CONST, {.i64 = 1 }, INT_MIN, INT_MAX, E, "avoid_negative_ts"},
74.     {"skip_initial_bytes", "set number of bytes to skip before reading header and frames", OFFSET(skip_initial_bytes), AV_OPT_TYPE_INT64, {.i64 = 0}, 0, INT64_MAX-1, D},
75.     {"correct_ts_overflow", "correct single timestamp overflows", OFFSET(correct_ts_overflow), AV_OPT_TYPE_INT, {.i64 = 1}, 0, 1, D},
76.     {"flush_packets", "enable flushing of the I/O context after each packet", OFFSET(flush_packets), AV_OPT_TYPE_INT, {.i64 = 1}, 0, 1, E},
77.     {"metadata_header_padding", "set number of bytes to be written as padding in a metadata header", OFFSET(metadata_header_padding), AV_OPT_TYPE_INT, {.i64 = -1}, -1, INT_MAX, E},
78.     {"output_ts_offset", "set output timestamp offset", OFFSET(output_ts_offset), AV_OPT_TYPE_DURATION, {.i64 = 0}, -INT64_MAX, INT64_MAX, E},
79.     {"max_interleave_delta", "maximum buffering duration for interleaving", OFFSET(max_interleave_delta), AV_OPT_TYPE_INT64, {.i64 = 1000000 }, 0, INT64_MAX, E },
80.     {"f_strict", "how strictly to follow the standards (deprecated; use strict, save via avconv)", OFFSET(strict_std_compliance), AV_OPT_TYPE_INT, {.i64 = DEFAULT }, INT_MIN, INT_MAX, D|E, "strict"},
81.     {"strict", "how strictly to follow the standards", OFFSET(strict_std_compliance), AV_OPT_TYPE_INT, {.i64 = DEFAULT }, INT_MIN, INT_MAX, D|E, "strict"},
82.     {"strict", "strictly conform to all the things in the spec no matter what the consequences", 0, AV_OPT_TYPE_CONST, {.i64 = FF_COMPLIANCE_STRICT }, INT_MIN, INT_MAX, D|E, "strict"},
83.     {"normal", NULL, 0, AV_OPT_TYPE_CONST, {.i64 = FF_COMPLIANCE_NORMAL }, INT_MIN, INT_MAX, D|E, "strict"},
84.     {"experimental", "allow non-standardized experimental variants", 0, AV_OPT_TYPE_CONST, {.i64 = FF_COMPLIANCE_EXPERIMENTAL }, INT_MIN, INT_MAX, D|E, "strict"},
85.     {"max_ts_probe", "maximum number of packets to read while waiting for the first timestamp", OFFSET(max_ts_probe), AV_OPT_TYPE_INT, {.i64 = 50 }, 0, INT_MAX, D },
86.     {NULL},
87. };
88.
89. #undef E
90. #undef D
91. #undef DEFAULT
92. #undef OFFSET
93.
94. #endif /* AVFORMAT_OPTIONS_TABLE_H */

```

AVCodecContext

AVFormatContext 中的AVClass定义位于libavcodec/options.c中，是一个名称为av_codec_context_class的静态结构体。如下所示。

```

1.  static const AVClass av_codec_context_class = {
2.      .class_name      = "AVCodecContext",
3.      .item_name       = context_to_name,
4.      .option          = avcodec_options,
5.      .version         = LIBAVUTIL_VERSION_INT,
6.      .log_level_offset_offset = offsetof(AVCodecContext, log_level_offset),
7.      .child_next      = codec_child_next,
8.      .child_class_next = codec_child_class_next,
9.      .category        = AV_CLASS_CATEGORY_ENCODER,
10.     .get_category      = get_category,
11. };

```

从源代码可以看出：

(1) class_name

该AVClass名称是“AVCodecContext”。

(2) item_name

item_name指向一个函数context_to_name ()，该函数定义如下所示。

```

1.  static const char* context_to_name(void* ptr) {
2.      AVCodecContext *avc= ptr;
3.
4.      if(avc && avc->codec && avc->codec->name)
5.          return avc->codec->name;
6.      else
7.          return "NULL";
8.  }

```

从函数的定义可以看出，如果AVCodecContext中的Codec结构体不为空，则返回Codec的name，否则返回“NULL”。

(3) category

option字段则指向一个元素个数极多的静态数组avcodec_options。该数组单独定义于libavcodec/options_table.h中。其中包含了AVCodecContext支持的所有的AVOptions。由于该数组定义实在是太多了，在这里仅贴出它前面的一小部分。

[cpp]  

```
1.  /*
2.   * 雷霄骅
3.   * leixiaohua1020@126.com
4.   * 中国传媒大学/数字电视技术
5.   * http://blog.csdn.net/leixiaohua1020
6.   *
7.   */
8.
9. #ifndef AVCODEC_OPTIONS_TABLE_H
10. #define AVCODEC_OPTIONS_TABLE_H
11.
12. #include <float.h>
13. #include <limits.h>
14. #include <stdint.h>
15.
16. #include "libavutil/opt.h"
17. #include "avcodec.h"
18. #include "version.h"
19.
20. #define OFFSET(x) offsetof(AVCodecContext,x)
21. #define DEFAULT 0 //should be NAN but it does not work as it is not a constant in glibc as required by ANSI/ISO C
22. //these names are too long to be readable
23. #define V AV_OPT_FLAG_VIDEO_PARAM
24. #define A AV_OPT_FLAG_AUDIO_PARAM
25. #define S AV_OPT_FLAG_SUBTITLE_PARAM
26. #define E AV_OPT_FLAG_ENCODING_PARAM
27. #define D AV_OPT_FLAG_DECODING_PARAM
28.
29. #define AV_CODEC_DEFAULT_BITRATE 200*1000
30.
31. static const AVOption avcodec_options[] = {
32. {"b", "set bitrate (in bits/s)", AV_OPT_TYPE_INT, {.i64 = AV_CODEC_DEFAULT_BITRATE }, 0, INT_MAX, A|V|E},
33. {"ab", "set bitrate (in bits/s)", OFFSET(bit_rate), AV_OPT_TYPE_INT, {.i64 = 128*1000 }, 0, INT_MAX, A|E},
34. {"bt", "Set video bitrate tolerance (in bits/s). In 1-pass mode, bitrate tolerance specifies how far "
35. "ratecontrol is willing to deviate from the target average bitrate value. This is not related "
36. "to minimum/maximum bitrate. Lowering tolerance too much has an adverse effect on quality.",
37. OFFSET(bit_rate_tolerance), AV_OPT_TYPE_INT, {.i64 = AV_CODEC_DEFAULT_BITRATE*20 }, 1, INT_MAX, V|E},
38. {"flags", NULL, OFFSET(flags), AV_OPT_TYPE_FLAGS, {.i64 = DEFAULT }, 0, UINT_MAX, V|A|S|E|D, "flags"},
39. {"unaligned", "allow decoders to produce unaligned output", 0, AV_OPT_TYPE_CONST, {.i64 = CODEC_FLAG_UNALIGNED }, INT_MIN, INT_MAX,
40. | D, "flags"},
41. {"mv4", "use four motion vectors per macroblock (MPEG-4)", 0, AV_OPT_TYPE_CONST, {.i64 = CODEC_FLAG_4MV }, INT_MIN, INT_MAX, V|E, "f
42. lags"},
43. {"qpel", "use 1/4-pel motion compensation", 0, AV_OPT_TYPE_CONST, {.i64 = CODEC_FLAG_QPEL }, INT_MIN, INT_MAX, V|E, "flags"},
44. {"loop", "use loop filter", 0, AV_OPT_TYPE_CONST, {.i64 = CODEC_FLAG_LOOP_FILTER }, INT_MIN, INT_MAX, V|E, "flags"},
45. {"qscale", "use fixed qscale", 0, AV_OPT_TYPE_CONST, {.i64 = CODEC_FLAG_QSCALE }, INT_MIN, INT_MAX, 0, "flags"},
46. #if FF_API_GMC
47. {"gmc", "use gmc", 0, AV_OPT_TYPE_CONST, {.i64 = CODEC_FLAG_GMC }, INT_MIN, INT_MAX, V|E, "flags"},
48. #endif
49. #if FF_API_MV0
50. {"mv0", "always try a mb with mv=<0,0>", 0, AV_OPT_TYPE_CONST, {.i64 = CODEC_FLAG_MV0 }, INT_MIN, INT_MAX, V|E, "flags"},
51. #endif
52. #if FF_API_INPUT_PRESERVED
53. {"input_preserved", NULL, 0, AV_OPT_TYPE_CONST, {.i64 = CODEC_FLAG_INPUT_PRESERVED }, INT_MIN, INT_MAX, 0, "flags"},
54. #endif
55. {"pass1", "use internal 2-
56. pass ratecontrol in first pass mode", 0, AV_OPT_TYPE_CONST, {.i64 = CODEC_FLAG_PASS1 }, INT_MIN, INT_MAX, 0, "flags"},
57. {"pass2", "use internal 2-
58. pass ratecontrol in second pass mode", 0, AV_OPT_TYPE_CONST, {.i64 = CODEC_FLAG_PASS2 }, INT_MIN, INT_MAX, 0, "flags"},
59. {"gray", "only decode/encode grayscale", 0, AV_OPT_TYPE_CONST, {.i64 = CODEC_FLAG_GRAY }, INT_MIN, INT_MAX, V|E|D, "flags"},
60. #if FF_API_EMU_EDGE
61. {"emu_edge", "do not draw edges", 0, AV_OPT_TYPE_CONST, {.i64 = CODEC_FLAG_EMU_EDGE }, INT_MIN, INT_MAX, 0, "flags"},
62. #endif
63. {"psnr", "error[?] variables will be set during encoding", 0, AV_OPT_TYPE_CONST, {.i64 = CODEC_FLAG_PSNR }, INT_MIN, INT_MAX, V|E, "
64. flags"},
65. {"truncated", NULL, 0, AV_OPT_TYPE_CONST, {.i64 = CODEC_FLAG_TRUNCATED }, INT_MIN, INT_MAX, 0, "flags"},
66. #if FF_API_NORMALIZE_AQP
67. {"naq", "normalize adaptive quantization", 0, AV_OPT_TYPE_CONST, {.i64 = CODEC_FLAG_NORMALIZE_AQP }, INT_MIN, INT_MAX, V|E, "flags"}
68. ,
69. #endif
70. {"ildct", "use interlaced DCT", 0, AV_OPT_TYPE_CONST, {.i64 = CODEC_FLAG_INTERLACED_DCT }, INT_MIN, INT_MAX, V|E, "flags"},
71. {"low_delay", "force low delay", 0, AV_OPT_TYPE_CONST, {.i64 = CODEC_FLAG_LOW_DELAY }, INT_MIN, INT_MAX, V|D|E, "flags"},
72. {"global_header", "place global headers in extradata instead of every keyframe", 0, AV_OPT_TYPE_CONST, {.i64 = CODEC_FLAG_GLOBAL_HEAD
73. R }, INT_MIN, INT_MAX, V|A|E, "flags"},
74. {"bitexact", "use only bitexact functions (except (I)DCT)", 0, AV_OPT_TYPE_CONST, {.i64 = CODEC_FLAG_BITEXACT }, INT_MIN, INT_MAX, A|
75. |S|D|E, "flags"},
76. {"aic", "H.263 advanced intra coding / MPEG-
77. 4 AC prediction", 0, AV_OPT_TYPE_CONST, {.i64 = CODEC_FLAG_AC_PRED }, INT_MIN, INT_MAX, V|E, "flags"},
78. {"ilme", "interlaced motion estimation", 0, AV_OPT_TYPE_CONST, {.i64 = CODEC_FLAG_INTERLACED_ME }, INT_MIN, INT_MAX, V|E, "flags"},
79.
80. {"cgop", "closed GOP", 0, AV_OPT_TYPE_CONST, {.i64 = CODEC_FLAG_CLOSED_GOP }, INT_MIN, INT_MAX, V|E, "flags"},
81. {"output_corrupt", "Output even potentially corrupted frames", 0, AV_OPT_TYPE_CONST, {.i64 = CODEC_FLAG_OUTPUT_CORRUPT }, INT_MIN, IN
82. MAX, V|D, "flags"},
83. {"fast", "allow non-spec-compliant speedup tricks", 0, AV_OPT_TYPE_CONST, {.i64 = CODEC_FLAG2_FAST }, INT_MIN, INT_MAX, V|E, "flags2
84. "},
85. {"noout", "skip bitstream encoding", 0, AV_OPT_TYPE_CONST, {.i64 = CODEC_FLAG2_NO_OUTPUT }, INT_MIN, INT_MAX, V|E, "flags2"},
86. {"ignorecrop", "ignore cropping information from sps", 0, AV_OPT_TYPE_CONST, {.i64 = CODEC_FLAG2_IGNORE_CROP }, INT_MIN, INT_MAX, V|D
87. "flags2"},
88. {"local_header", "place global headers at every keyframe instead of in extradata", 0, AV_OPT_TYPE_CONST, {.i64 = CODEC_FLAG2_LOCAL_HE
89. DER }, INT_MIN, INT_MAX, V|E, "flags2"}
```

```

    DER }, INT_MIN, INT_MAX, V|E, "flags2"},
76. { "chunks", "Frame data might be split into multiple chunks", 0, AV_OPT_TYPE_CONST, {.i64 = CODEC_FLAG2_CHUNKS }, INT_MIN, INT_MAX, V|E, "flags2"},
77. { "showall", "Show all frames before the first keyframe", 0, AV_OPT_TYPE_CONST, {.i64 = CODEC_FLAG2_SHOW_ALL }, INT_MIN, INT_MAX, V|D, "flags2"},
78. { "export_mv", "export motion vectors through frame side data", 0, AV_OPT_TYPE_CONST, {.i64 = CODEC_FLAG2_EXPORT_MVS}, INT_MIN, INT_MAX, V|D, "flags2"},
79. { "me_method", "set motion estimation method", OFFSET(me_method), AV_OPT_TYPE_INT, {.i64 = ME_EPZS }, INT_MIN, INT_MAX, V|E, "me_method"},
80. { "zero", "zero motion estimation (fastest)", 0, AV_OPT_TYPE_CONST, {.i64 = ME_ZERO }, INT_MIN, INT_MAX, V|E, "me_method" },
81. { "full", "full motion estimation (slowest)", 0, AV_OPT_TYPE_CONST, {.i64 = ME_FULL }, INT_MIN, INT_MAX, V|E, "me_method" },
82. { "epzs", "EPZS motion estimation (default)", 0, AV_OPT_TYPE_CONST, {.i64 = ME_EPZS }, INT_MIN, INT_MAX, V|E, "me_method" },
83. { "esa", "esa motion estimation (alias for full)", 0, AV_OPT_TYPE_CONST, {.i64 = ME_FULL }, INT_MIN, INT_MAX, V|E, "me_method" },
84. { "tesa", "tesa motion estimation", 0, AV_OPT_TYPE_CONST, {.i64 = ME_TESA }, INT_MIN, INT_MAX, V|E, "me_method" },
85. { "dia", "diamond motion estimation (alias for EPZS)", 0, AV_OPT_TYPE_CONST, {.i64 = ME_EPZS }, INT_MIN, INT_MAX, V|E, "me_method" },

86. { "log", "log motion estimation", 0, AV_OPT_TYPE_CONST, {.i64 = ME_LOG }, INT_MIN, INT_MAX, V|E, "me_method" },
87. { "phods", "phods motion estimation", 0, AV_OPT_TYPE_CONST, {.i64 = ME_PHODS }, INT_MIN, INT_MAX, V|E, "me_method" },
88. { "x1", "X1 motion estimation", 0, AV_OPT_TYPE_CONST, {.i64 = ME_X1 }, INT_MIN, INT_MAX, V|E, "me_method" },
89. { "hex", "hex motion estimation", 0, AV_OPT_TYPE_CONST, {.i64 = ME_HEX }, INT_MIN, INT_MAX, V|E, "me_method" },
90. { "umh", "umh motion estimation", 0, AV_OPT_TYPE_CONST, {.i64 = ME_UMH }, INT_MIN, INT_MAX, V|E, "me_method" },
91. { "iter", "iter motion estimation", 0, AV_OPT_TYPE_CONST, {.i64 = ME_ITER }, INT_MIN, INT_MAX, V|E, "me_method" },
92. { "extradata_size", NULL, OFFSET(extradata_size), AV_OPT_TYPE_INT, {.i64 = DEFAULT }, INT_MIN, INT_MAX},
93. { "time_base", NULL, OFFSET(time_base), AV_OPT_TYPE_RATIONAL, {.dbl = 0}, INT_MIN, INT_MAX},
94. { "g", "set the group of picture (GOP) size", OFFSET(gop_size), AV_OPT_TYPE_INT, {.i64 = 12 }, INT_MIN, INT_MAX, V|E},
95. { "ar", "set audio sampling rate (in Hz)", OFFSET(sample_rate), AV_OPT_TYPE_INT, {.i64 = DEFAULT }, INT_MIN, INT_MAX, A|D|E},
96. { "ac", "set number of audio channels", OFFSET(channels), AV_OPT_TYPE_INT, {.i64 = DEFAULT }, INT_MIN, INT_MAX, A|D|E},

```

AVFrame

AVFrame 中的AVClass定义位于libavcodec/options.c中，是一个名称为av_frame_class的静态结构体。如下所示。

```

1. static const AVClass av_frame_class = {
2.     .class_name      = "AVFrame",
3.     .item_name       = NULL,
4.     .option           = frame_options,
5.     .version          = LIBAVUTIL_VERSION_INT,
6. };

```

option字段则指向一个元素个数极多的静态数组frame_options。frame_options定义如下所示。

```

1. static const AVOption frame_options[]={
2.     { "best_effort_timestamp", "", FFFSET(best_effort_timestamp), AV_OPT_TYPE_INT64, {.i64 = AV_NOPTS_VALUE }, INT64_MIN, INT64_MAX, 0},
3.     { "pkt_pos", "", FFFSET(pkt_pos), AV_OPT_TYPE_INT64, {.i64 = -1 }, INT64_MIN, INT64_MAX, 0},
4.     { "pkt_size", "", FFFSET(pkt_size), AV_OPT_TYPE_INT64, {.i64 = -1 }, INT64_MIN, INT64_MAX, 0},
5.     { "sample_aspect_ratio", "", FFFSET(sample_aspect_ratio), AV_OPT_TYPE_RATIONAL, {.dbl = 0 }, 0, INT_MAX, 0},
6.     { "width", "", FFFSET(width), AV_OPT_TYPE_INT, {.i64 = 0 }, 0, INT_MAX, 0},
7.     { "height", "", FFFSET(height), AV_OPT_TYPE_INT, {.i64 = 0 }, 0, INT_MAX, 0},
8.     { "format", "", FFFSET(format), AV_OPT_TYPE_INT, {.i64 = -1 }, 0, INT_MAX, 0},
9.     { "channel_layout", "", FFFSET(channel_layout), AV_OPT_TYPE_INT64, {.i64 = 0 }, 0, INT64_MAX, 0},
10.    { "sample_rate", "", FFFSET(sample_rate), AV_OPT_TYPE_INT, {.i64 = 0 }, 0, INT_MAX, 0},
11.    { NULL},
12. };

```

可以看出AVFrame的选项数组中包含了“width”，“height”这类用于视频帧的选项，以及“channel_layout”，“sample_rate”这类用于音频帧的选项。

各种组件特有的AVClass

除了Ffmpeg中通用的AVFormatContext，AVCodecContext，AVFrame这类的结构体之外，每种特定的组件也包含自己的AVClass。下面举例几个。

LibRTMP

libRTMP中根据协议类型的不同定义了多种的AVClass。由于这些AVClass除了名字不一样之外，其他的字段一模一样，所以AVClass的声明写成了一个名称为RTMP_CLASS的宏。

```
[cpp]
1. #define RTMP_CLASS(flavor)\
2. static const AVClass lib ## flavor ## _class = {\
3.     .class_name = "lib" #flavor " protocol",\
4.     .item_name  = av_default_item_name,\
5.     .option     = options,\
6.     .version    = LIBAVUTIL_VERSION_INT,\
7. };
```

而后定义了多种AVClass：

```
[cpp]
1. RTMP_CLASS(rtmp)
2. RTMP_CLASS(rtmpt)
3. RTMP_CLASS(rtmpe)
4. RTMP_CLASS(rtmpte)
5. RTMP_CLASS(rtmps)
```

这些AVClass的option字段指向的数组是一样的，如下所示。

```
[cpp]
1. static const AVOption options[] = {
2.     {"rtmp_app", "Name of application to connect to on the RTMP server", OFFSET(app), AV_OPT_TYPE_STRING, {.str = NULL }, 0, 0, DEC|ENC},
3.     {"rtmp_buffer", "Set buffer time in milliseconds. The default is 3000.", OFFSET(client_buffer_time), AV_OPT_TYPE_STRING, {.str = "3000"}, 0, 0, DEC|ENC},
4.     {"rtmp_conn", "Append arbitrary AMF data to the Connect message", OFFSET(conn), AV_OPT_TYPE_STRING, {.str = NULL }, 0, 0, DEC|ENC},
5.     {"rtmp_flashver", "Version of the Flash plugin used to run the SWF player.", OFFSET(flashver), AV_OPT_TYPE_STRING, {.str = NULL }, 0, 0, DEC|ENC},
6.     {"rtmp_live", "Specify that the media is a live stream.", OFFSET(live), AV_OPT_TYPE_INT, {.i64 = 0}, INT_MIN, INT_MAX, DEC, "rtmp_live"},
7.     {"any", "both", 0, AV_OPT_TYPE_CONST, {.i64 = -2}, 0, 0, DEC, "rtmp_live"},
8.     {"live", "live stream", 0, AV_OPT_TYPE_CONST, {.i64 = -1}, 0, 0, DEC, "rtmp_live"},
9.     {"recorded", "recorded stream", 0, AV_OPT_TYPE_CONST, {.i64 = 0}, 0, 0, DEC, "rtmp_live"},
10.    {"rtmp_pageurl", "URL of the web page in which the media was embedded. By default no value will be sent.", OFFSET(pageurl), AV_OPT_TYPE_STRING, {.str = NULL }, 0, 0, DEC},
11.    {"rtmp_playpath", "Stream identifier to play or to publish", OFFSET(playpath), AV_OPT_TYPE_STRING, {.str = NULL }, 0, 0, DEC|ENC},
12.    {"rtmp_subscribe", "Name of live stream to subscribe to. Defaults to rtmp_playpath.", OFFSET(subscribe), AV_OPT_TYPE_STRING, {.str = NULL }, 0, 0, DEC},
13.    {"rtmp_swfurl", "URL of the SWF player. By default no value will be sent", OFFSET(swfurl), AV_OPT_TYPE_STRING, {.str = NULL }, 0, 0, DEC|ENC},
14.    {"rtmp_swfverify", "URL to player swf file, compute hash/size automatically. (unimplemented)", OFFSET(swfverify), AV_OPT_TYPE_STRING, {.str = NULL }, 0, 0, DEC},
15.    {"rtmp_tcurl", "URL of the target stream. Defaults to proto://host[:port]/app.", OFFSET(tcurl), AV_OPT_TYPE_STRING, {.str = NULL }, 0, 0, DEC|ENC},
16.    { NULL },
17. };
```

Libx264

Libx264的AVClass定义如下所示。

```
[cpp]
1. static const AVClass x264_class = {
2.     .class_name = "libx264",
3.     .item_name  = av_default_item_name,
4.     .option     = options,
5.     .version    = LIBAVUTIL_VERSION_INT,
6. };
```

其中option字段指向的数组定义如下所示。这些option的使用频率还是比较高的。

```
[cpp]
1. static const AVOption options[] = {
2.     {"preset", "Set the encoding preset (cf. x264 --fullhelp)", OFFSET(preset), AV_OPT_TYPE_STRING, {.str = "medium"}, 0, 0, VE},
3.     {"tune", "Tune the encoding params (cf. x264 --fullhelp)", OFFSET(tune), AV_OPT_TYPE_STRING, { 0 }, 0, 0, VE},
4.     {"profile", "Set profile restrictions (cf. x264 --fullhelp)", OFFSET(profile), AV_OPT_TYPE_STRING, { 0 }, 0, 0, VE},
5.     {"fastfirstpass", "Use fast settings when encoding first pass", OFFSET(fastfirstpass), AV_OPT_TYPE_INT, {.i64 = 1 }, 0, 1, VE},
6.     {"level", "Specify level (as defined by Annex A)", OFFSET(level), AV_OPT_TYPE_STRING, {.str=NULL}, 0, 0, VE},
7.     {"passlogfile", "Filename for 2 pass stats", OFFSET(stats), AV_OPT_TYPE_STRING, {.str=NULL}, 0, 0, VE},
8.     {"wpredp", "Weighted prediction for P-frames", OFFSET(wpredp), AV_OPT_TYPE_STRING, {.str=NULL}, 0, 0, VE},
9.     {"x264opts", "x264 options", OFFSET(x264opts), AV_OPT_TYPE_STRING, {.str=NULL}, 0, 0, VE},
10.    {"crf", "Select the quality for constant quality mode", OFFSET(crf), AV_OPT_TYPE_FLOAT, {.dbl = -1 }, -1, FLT_MAX, VE},
```

```

11.     { "crf_max",      "In CRF mode, prevents VBV from lowering quality beyond this point.",OFFSET(crf_max), AV_OPT_TYPE_FLOAT, {.dbl
12.     = -1 }, -1, FLT_MAX, VE },
13.     { "qp",          "Constant quantization parameter rate control method",OFFSET(cqp),          AV_OPT_TYPE_INT,    { .i64 = -1 },
14.     -1, INT_MAX, VE },
15.     { "aq-mode",      "AQ method",                                OFFSET(aq_mode),    AV_OPT_TYPE_INT,    { .i64 = -1 },
16.     -1, INT_MAX, VE, "aq_mode"},
17.     { "none",         NULL,                                0, AV_OPT_TYPE_CONST, {.i64 = X264_AQ_NONE},    INT_MIN, INT_MAX, VE,
18.     "aq_mode" },
19.     { "variance",      "Variance AQ (complexity mask)",    0, AV_OPT_TYPE_CONST, {.i64 = X264_AQ_VARIANCE},    INT_MIN, INT_MAX, VE,
20.     "aq_mode" },
21.     { "autovariance",  "Auto-
22.     variance AQ (experimental)", 0, AV_OPT_TYPE_CONST, {.i64 = X264_AQ_AUTOVARIANCE}, INT_MIN, INT_MAX, VE, "aq_mode" },
23.     { "aq-
24.     strength",         "AQ strength. Reduces blocking and blurring in flat and textured areas.", OFFSET(aq_strength), AV_OPT_TYPE_FLOAT, {.dbl
25.     = -1}, -1, FLT_MAX, VE},
26.     { "psy",          "Use psychovisual optimizations.",    OFFSET(psy),        AV_OPT_TYPE_INT,    { .i64 = -1 },
27.     -1, 1, VE },
28.     { "psy-rd",        "Strength of psychovisual optimization, in <psy-rd>:<psy-
29.     trellis> format.", OFFSET(psy_rd), AV_OPT_TYPE_STRING, { 0 }, 0, 0, VE},
30.     { "rc-
31.     lookahead",        "Number of frames to look ahead for frametype and ratecontrol", OFFSET(rc_lookahead), AV_OPT_TYPE_INT, { .i64 = -1 }, -
32.     1, INT_MAX, VE },
33.     { "weightb",       "Weighted prediction for B-frames.",    OFFSET(weightb),    AV_OPT_TYPE_INT,    { .i64 = -1 },
34.     -1, 1, VE },
35.     { "weightp",       "Weighted prediction analysis method.",    OFFSET(weightp),    AV_OPT_TYPE_INT,    { .i64 = -1 },
36.     -1, INT_MAX, VE, "weightp" },
37.     { "none",         NULL, 0, AV_OPT_TYPE_CONST, {.i64 = X264_WEIGHTP_NONE},    INT_MIN, INT_MAX, VE, "weightp" },
38.     { "simple",        NULL, 0, AV_OPT_TYPE_CONST, {.i64 = X264_WEIGHTP_SIMPLE}, INT_MIN, INT_MAX, VE, "weightp" },
39.     { "smart",        NULL, 0, AV_OPT_TYPE_CONST, {.i64 = X264_WEIGHTP_SMART},    INT_MIN, INT_MAX, VE, "weightp" },
40.     { "ssim",         "Calculate and print SSIM stats.",    OFFSET(ssim),        AV_OPT_TYPE_INT,    { .i64 = -1 },
41.     -1, 1, VE },
42.     { "intra-refresh", "Use Periodic Intra Refresh instead of IDR frames.",OFFSET(intra_refresh),AV_OPT_TYPE_INT,    { .i64 = -1 },
43.     -1, 1, VE },
44.     { "bluray-compat", "Bluray compatibility workarounds.",    OFFSET(bluray_compat), AV_OPT_TYPE_INT,    { .i64 = -1 },
45.     -1, 1, VE },
46.     { "b-bias",        "Influences how often B-
47.     frames are used",    OFFSET(b_bias),      AV_OPT_TYPE_INT,    { .i64 = INT_MIN}, INT_MIN, INT_MAX, VE },
48.     { "b-pyramid",     "Keep some B-frames as references.",    OFFSET(b_pyramid),   AV_OPT_TYPE_INT,    { .i64 = -1 },
49.     -1, INT_MAX, VE, "b_pyramid" },
50.     { "none",         NULL, 0, AV_OPT_TYPE_CONST, {.i64 = X264_B_PYRAMID_NONE},    INT_MIN, INT_MAX,
51.     , "b_pyramid" },
52.     { "strict",        "Strictly hierarchical pyramid",    0, AV_OPT_TYPE_CONST, {.i64 = X264_B_PYRAMID_STRICT}, INT_MIN, INT_MAX,
53.     VE, "b_pyramid" },
54.     { "normal",        "Non-strict (not Blu-
55.     ray compatible)", 0, AV_OPT_TYPE_CONST, {.i64 = X264_B_PYRAMID_NORMAL}, INT_MIN, INT_MAX, VE, "b_pyramid" },
56.     { "mixed-
57.     refs",            "One reference per partition, as opposed to one reference per macroblock", OFFSET(mixed_refs), AV_OPT_TYPE_INT, { .i64 = -
58.     1}, -1, 1, VE },
59.     { "8x8dct",        "High profile 8x8 transform.",    OFFSET(dct8x8),      AV_OPT_TYPE_INT,    { .i64 = -1 },
60.     -1, 1, VE},
61.     { "fast-pskip",    NULL,                                OFFSET(fast_pskip),   AV_OPT_TYPE_INT,    { .i64 = -1 },
62.     -1, 1, VE},
63.     { "aud",           "Use access unit delimiters.",    OFFSET(aud),         AV_OPT_TYPE_INT,    { .i64 = -1 },
64.     -1, 1, VE},
65.     { "mbtree",        "Use macroblock tree ratecontrol.",    OFFSET(mbtree),      AV_OPT_TYPE_INT,    { .i64 = -1 },
66.     -1, 1, VE},
67.     { "deblock",       "Loop filter parameters, in <alpha:beta> form.",    OFFSET(deblock),     AV_OPT_TYPE_STRING, { 0 }, 0, 0, VE
68.     ,
69.     { "cplxblur",      "Reduce fluctuations in QP (before curve compression)", OFFSET(cplxblur), AV_OPT_TYPE_FLOAT, { .dbl = -1 }, -
70.     1, FLT_MAX, VE},
71.     { "partitions",    "A comma-separated list of partitions to consider. "
72.     "Possible values: p8x8, p4x4, b8x8, i8x8, i4x4, none, all", OFFSET(partitions), AV_OPT_TYPE_STRING, { 0 }, 0, 0,
73.     0, VE},
74.     { "direct-pred",   "Direct MV prediction mode",    OFFSET(direct_pred), AV_OPT_TYPE_INT,    { .i64 = -1 },
75.     -1, INT_MAX, VE, "direct-pred" },
76.     { "none",         NULL, 0, AV_OPT_TYPE_CONST, {.i64 = X264_DIRECT_PRED_NONE },    0, 0, VE, "direct-pred" },
77.     { "spatial",      NULL, 0, AV_OPT_TYPE_CONST, {.i64 = X264_DIRECT_PRED_SPATIAL },    0, 0, VE, "direct-pred" },
78.     { "temporal",     NULL, 0, AV_OPT_TYPE_CONST, {.i64 = X264_DIRECT_PRED_TEMPORAL }, 0, 0, VE, "direct-pred" },
79.     { "auto",         NULL, 0, AV_OPT_TYPE_CONST, {.i64 = X264_DIRECT_PRED_AUTO },    0, 0, VE, "direct-pred" },
80.     { "slice-max-size", "Limit the size of each slice in bytes",    OFFSET(slice_max_size),AV_OPT_TYPE_INT,    { .i64 = -1 },
81.     -1, INT_MAX, VE },
82.     { "stats",        "Filename for 2 pass stats",    OFFSET(stats),        AV_OPT_TYPE_STRING, { 0 }, 0,
83.     0, VE },
84.     { "nal-hrd",       "Signal HRD information (requires vbv-buFSIZE; "
85.     "cbr not allowed in .mp4)",    OFFSET(nal_hrd),      AV_OPT_TYPE_INT,    { .i64 = -1 },
86.     -1, INT_MAX, VE, "nal-hrd" },
87.     { "none",         NULL, 0, AV_OPT_TYPE_CONST, {.i64 = X264_NAL_HRD_NONE},    INT_MIN, INT_MAX, VE, "nal-hrd" },
88.     { "vbr",          NULL, 0, AV_OPT_TYPE_CONST, {.i64 = X264_NAL_HRD_VBR},    INT_MIN, INT_MAX, VE, "nal-hrd" },
89.     { "cbr",          NULL, 0, AV_OPT_TYPE_CONST, {.i64 = X264_NAL_HRD_CBR},    INT_MIN, INT_MAX, VE, "nal-hrd" },
90.     { "avcintra-class", "AVC-Intra class 50/100/200",    OFFSET(avcintra_class),AV_OPT_TYPE_INT,    { .i64 = -1 },
91.     -1, 200 , VE},
92.     { "x264-params",   "Override the x264 configuration using a :-
93.     separated list of key=value parameters", OFFSET(x264_params), AV_OPT_TYPE_STRING, { 0 }, 0, 0, VE },
94.     { NULL },
95. };

```

Libx265的AVClass定义如下所示。

```
[cpp]
1. static const AVClass class = {
2.     .class_name = "libx265",
3.     .item_name  = av_default_item_name,
4.     .option     = options,
5.     .version    = LIBAVUTIL_VERSION_INT,
6. };
```

其中option字段指向的数组定义如下所示。

```
[cpp]
1. static const AVOption options[] = {
2.     { "preset", "set the x265 preset", OFFSET(preset), AV_OPT_TYPE_ST
   NG, { 0 }, 0, 0, VE },
3.     { "tune", "set the x265 tune parameter", OFFSET(tune), AV_OPT_TYPE_ST
   NG, { 0 }, 0, 0, VE },
4.     { "x265-params", "set the x265 configuration using a :-
   separated list of key=value parameters", OFFSET(x265_opts), AV_OPT_TYPE_STRING, { 0 }, 0, 0, VE },
5.     { NULL }
6. };
```

官方代码中有关AVClass和AVOption的示例

官方代码中给出了一小段示例代码，演示了如何给一个普通的结构体添加AVOption的支持。如下所示。

```
[cpp]
1. typedef struct test_struct {
2.     AVClass *class;
3.     int     int_opt;
4.     char    str_opt;
5.     uint8_t bin_opt;
6.     int     bin_len;
7. } test_struct;
8.
9. static const AVOption test_options[] = {
10.     { "test_int", "This is a test option of int type.", offsetof(test_struct, int_opt),
11.       AV_OPT_TYPE_INT, { .i64 = -1 }, INT_MIN, INT_MAX },
12.     { "test_str", "This is a test option of string type.", offsetof(test_struct, str_opt),
13.       AV_OPT_TYPE_STRING,
14.       { "test_bin", "This is a test option of binary type.", offsetof(test_struct, bin_opt),
15.         AV_OPT_TYPE_BINARY,
16.         { NULL },
17.     };
18.
19. static const AVClass test_class = {
20.     .class_name = "test class",
21.     .item_name  = av_default_item_name,
22.     .option     = test_options,
23.     .version    = LIBAVUTIL_VERSION_INT,
24. };
```

AVClass有关的API

与AVClass相关的API很少。AVFormatContext提供了一个获取当前AVClass的函数avformat_get_class()。它的代码很简单，直接返回全局静态变量av_format_context_class。定义如下所示。

```
[cpp]
1. const AVClass *avformat_get_class(void)
2. {
3.     return &av_format_context_class;
4. }
```

同样，AVCodecContext也提供了一个获取当前AVClass的函数avcodec_get_class()。它直接返回静态变量av_codec_context_class。定义如下所示。

```
[cpp]
1. const AVClass *avcodec_get_class(void)
2. {
3.     return &av_codec_context_class;
4. }
```

至此FFmpeg的AVClass就基本上分析完毕了。下篇文章具体分析AVOption。

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/44268323>

文章标签：

ffmpeg

AVClass

AVOption

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