【[**架构**](http://lib.csdn.net/base/architecture)图】

[FFmpeg源代码结构图 - 解码](http://blog.csdn.net/leixiaohua1020/article/details/44220151)

[FFmpeg源代码结构图 - 编码](http://blog.csdn.net/leixiaohua1020/article/details/44226355)

【通用】

[FFmpeg 源代码简单分析：av\_register\_all()](http://blog.csdn.net/leixiaohua1020/article/details/12677129)

[FFmpeg 源代码简单分析：avcodec\_register\_all()](http://blog.csdn.net/leixiaohua1020/article/details/12677265)

[FFmpeg 源代码简单分析：内存的分配和释放（av\_malloc()、av\_free()等）](http://blog.csdn.net/leixiaohua1020/article/details/41176777)

[FFmpeg 源代码简单分析：常见结构体的初始化和销毁（AVFormatContext，AVFrame等）](http://blog.csdn.net/leixiaohua1020/article/details/41181155)

[FFmpeg 源代码简单分析：avio\_open2()](http://blog.csdn.net/leixiaohua1020/article/details/41199947)

[FFmpeg 源代码简单分析：av\_find\_decoder()和av\_find\_encoder()](http://blog.csdn.net/leixiaohua1020/article/details/44084557)

[FFmpeg 源代码简单分析：avcodec\_open2()](http://blog.csdn.net/leixiaohua1020/article/details/44117891)

[FFmpeg 源代码简单分析：avcodec\_close()](http://blog.csdn.net/leixiaohua1020/article/details/44206699)

【解码】

[图解FFMPEG打开媒体的函数avformat\_open\_input](http://blog.csdn.net/leixiaohua1020/article/details/8661601)

[FFmpeg 源代码简单分析：avformat\_open\_input()](http://blog.csdn.net/leixiaohua1020/article/details/44064715)

[FFmpeg 源代码简单分析：avformat\_find\_stream\_info()](http://blog.csdn.net/leixiaohua1020/article/details/44084321)

[FFmpeg 源代码简单分析：av\_read\_frame()](http://blog.csdn.net/leixiaohua1020/article/details/12678577)

[FFmpeg 源代码简单分析：avcodec\_decode\_video2()](http://blog.csdn.net/leixiaohua1020/article/details/12679719)

[FFmpeg 源代码简单分析：avformat\_close\_input()](http://blog.csdn.net/leixiaohua1020/article/details/44110683)

【编码】

[FFmpeg 源代码简单分析：avformat\_alloc\_output\_context2()](http://blog.csdn.net/leixiaohua1020/article/details/41198929)

[FFmpeg 源代码简单分析：avformat\_write\_header()](http://blog.csdn.net/leixiaohua1020/article/details/44116215)

[FFmpeg 源代码简单分析：avcodec\_encode\_video()](http://blog.csdn.net/leixiaohua1020/article/details/44206485)

[FFmpeg 源代码简单分析：av\_write\_frame()](http://blog.csdn.net/leixiaohua1020/article/details/44199673)

[FFmpeg 源代码简单分析：av\_write\_trailer()](http://blog.csdn.net/leixiaohua1020/article/details/44201645)

【其它】

[FFmpeg源代码简单分析：日志输出系统（av\_log()等）](http://blog.csdn.net/leixiaohua1020/article/details/44243155)

[FFmpeg源代码简单分析：结构体成员管理系统-AVClass](http://blog.csdn.net/leixiaohua1020/article/details/44268323)

[FFmpeg源代码简单分析：结构体成员管理系统-AVOption](http://blog.csdn.net/leixiaohua1020/article/details/44279329)

[FFmpeg源代码简单分析：libswscale的sws\_getContext()](http://blog.csdn.net/leixiaohua1020/article/details/44305697)

[FFmpeg源代码简单分析：libswscale的sws\_scale()](http://blog.csdn.net/leixiaohua1020/article/details/44346687)

[FFmpeg源代码简单分析：libavdevice的avdevice\_register\_all()](http://blog.csdn.net/leixiaohua1020/article/details/41211121)

[FFmpeg源代码简单分析：libavdevice的gdigrab](http://blog.csdn.net/leixiaohua1020/article/details/44597955)

【脚本】

[FFmpeg源代码简单分析：makefile](http://blog.csdn.net/leixiaohua1020/article/details/44556525)

[FFmpeg源代码简单分析：configure](http://blog.csdn.net/leixiaohua1020/article/details/44587465)

【H.264】

[FFmpeg的H.264解码器源代码简单分析：概述](http://blog.csdn.net/leixiaohua1020/article/details/44864509)

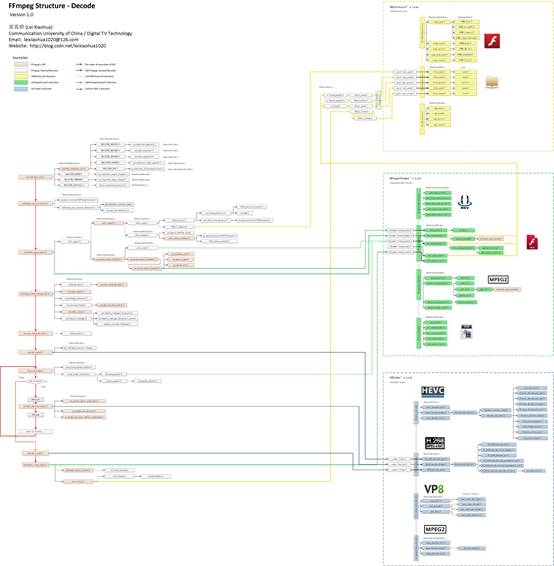
# [FFmpeg源代码结构图 - 解码](http://blog.csdn.net/leixiaohua1020/article/details/44220151)

近期研究了一下FFmpeg的内部的源代码。之前对于FFmpeg的研究主要在它的应用层面上，因此制作的很多示例程序都是调用的FFmpeg的API。但是一直感觉这样对FFmpeg的理解还是比较浅，所以打算剖析一下它的源代码，理一下它内部结构的“脉络”。但是有一个很难办的问题：FFmpeg自带的三个工程：ffplay， ffmpeg， ffprobe的代码量非常的大，其中包含了成百上千的API；而这些API背后又包含了大量的FFmpeg内部函数。如此一来，几乎是不可能理清他们之间的关系的。经过一番思考之后，打算选择FFmpeg编码和解码过程中的最核心的API进行分析。在编码或者解码的过程中，核心的API数量不多，一共大约10个左右，这样一来就可以剖析其内部的源代码了。

* FFmpeg解码过程核心的API可以参考：《[最简单的基于FFmpeg+SDL的视频播放器](http://blog.csdn.net/leixiaohua1020/article/details/38868499)》
* 编码过程核心的API可以参考：《[最简单的基于FFmpeg的视频编码器](http://blog.csdn.net/leixiaohua1020/article/details/25430425)》

## FFmpeg源代码结构图-解码

首先呈现分析出来的FFmpeg源代码结构图。这张图的尺寸非常的大，尺寸大约有4000x4000，有点像一张地图（因此最好选择“查看更清晰的图片”之后，右键保存图片到本机之后再查看）。它表明了FFmpeg在解码一个视频的时候的函数调用流程。为了保证结构清晰，其中仅列出了最关键的函数，剔除了其它不是特别重要的函数。

[](http://img.my.csdn.net/uploads/201503/12/1426134989_1189.jpg)

[单击查看更清晰的图片](https://my.csdn.net/leixiaohua1020/album/detail/1805165)

下面解释一下图中关键标记的含义。

### 函数背景色

函数在图中以方框的形式表现出来。不同的背景色标志了该函数不同的作用：

* 粉红色背景函数：FFmpeg的API函数。
* 白色背景的函数：FFmpeg的内部函数。
* 黄色背景的函数：URLProtocol结构体中的函数，包含处理协议（Protocol）的功能。
* 绿色背景的函数：AVInputFormat结构体中的函数，包含处理封装格式（Format）的功能。
* 蓝色背景的函数：AVCodec结构体中的函数，包含了编解码器（Codec）的功能。

PS：URLProtocol，AVInputFormat，AVCodec在FFmpeg开始运行并且注册完组件之后，都会分别被连接成一个个的链表。因此实际上是有很多的URLProtocol，AVInputFormat，AVCodec的。图中画出了解码一个输入协议是“文件”（其实就是打开一个文件。“文件”也被当做是一种广义的协议），封装格式为FLV，视频编码格式是H.264的数据的函数调用关系。

### 区域

整个架构图可以分为以下几个区域：

* 左边区域——架构函数区域：这些函数并不针对某一特定的视频格式。
* 右上方黄色区域——协议处理函数区域：不同的协议（RTP，RTMP，FILE）会调用不同的协议处理函数。
* 右边中间绿色区域——封装格式处理函数区域：不同的封装格式（MKV，FLV，MPEGTS，AVI）会调用不同的封装格式处理函数。
* 右边下方蓝色区域——编解码函数区域：不同的编码标准（HEVC，H.264，MPEG2）会调用不同的编解码函数。

### 箭头线

为了把调用关系表示的更明显，图中的箭头线也使用了不同的颜色：

黑色箭头线：标志了函数之间的调用关系。

红色的箭头线：标志了解码的流程。

其他颜色的箭头线：标志了函数之间的调用关系。其中：

调用URLProtocol结构体中的函数用黄色箭头线标识；

调用AVInputFormat结构体中的函数用绿色箭头线标识；

调用AVCodec结构体中的函数用蓝色箭头线标识。

### 函数所在的文件

每个函数旁边标识了它所在的文件的路径。

此外，还有一点需要注意的是，一些API函数内部也调用了另一些API函数。也就是说，API函数并不一定全部都调用FFmpeg的内部函数，他也有可能调用其他的API函数。例如从图中可以看出来，avformat\_close\_input()调用了avformat\_free\_context()和avio\_close()。这些在内部代码中被调用的API函数也标记为粉红色。

## 函数调用关系

下面简单列出几个区域中函数之间的调用关系（函数之间的调用关系使用缩进的方式表现出来）。详细的函数分析可以参考相关的《FFmpeg源代码分析》系列文章。

### 左边区域（FFmpeg架构函数）

**1.  av\_register\_all()【**[**函数简单分析**](http://blog.csdn.net/leixiaohua1020/article/details/12677129)**】**

**1)  avcodec\_register\_all()**

**(a) REGISTER\_HWACCEL()**

**(b) REGISTER\_ENCODER()**

**(c) REGISTER\_DECODER()**

**(d) REGISTER\_PARSER()**

**(e) REGISTER\_BSF()**

**2)  REGISTER\_MUXER()**

**3)  REGISTER\_DEMUXER()**

**4)  REGISTER\_PROTOCOL()**

**2.  avformat\_alloc\_context()【**[**函数简单分析**](http://blog.csdn.net/leixiaohua1020/article/details/41181155)**】**

**1) av\_malloc(sizeof(AVFormatContext))**

**2) avformat\_get\_context\_defaults()**

**(a) av\_opt\_set\_defaults()**

**3.  avformat\_open\_input()【**[**函数简单分析**](http://blog.csdn.net/leixiaohua1020/article/details/44064715)**】**

**1) init\_input()**

**(a) avio\_open2()【**[**函数简单分析**](http://blog.csdn.net/leixiaohua1020/article/details/41199947)**】**

**a) ffurl\_open()**

**i. ffurl\_alloc()**

l  **url\_find\_protocol()**

l  **url\_alloc\_for\_protocol()**

**ii. ffurl\_connect()**

**URLProtocol->url\_open()**

**b) ffio\_fdopen()**

**i. av\_malloc(buffer\_size)**

**ii. avio\_alloc\_context()**

l  **av\_mallocz(sizeof(AVIOContext))**

l  **ffio\_init\_context()**

**(b) av\_probe\_input\_buffer2()**

**a) avio\_read()**

**i.  AVInputFormat->read\_packet()**

**b) av\_probe\_input\_format2()**

**c) av\_probe\_input\_format3()**

**i. av\_iformat\_next()**

**ii. av\_match\_name()**

**iii. av\_match\_ext()**

**iv. AVInputFormat->read\_probe()**

**2) AVInputFormat->read\_header()**

**4. avformat\_find\_stream\_info()【**[**函数简单分析**](http://blog.csdn.net/leixiaohua1020/article/details/44084321)**】**

**1) find\_decoder()**

**(a) avcodec\_find\_decoder()**

**2) avcodec\_open2()**

**3) read\_frame\_internal()**

**4) try\_decode\_frame()**

**(a) avcodec\_decode\_video2()**

**5) avcodec\_close()**

**6) estimate\_timings()**

**(a)  estimate\_timings\_from\_pts()**

**(b)  estimate\_timings\_from\_bit\_rate()**

**(c)  update\_stream\_timings()**

**5. avcodec\_find\_decoder()【**[**函数简单分析**](http://blog.csdn.net/leixiaohua1020/article/details/44084557)**】**

**1) find\_encdec()**

**6. avcodec\_open2()【**[**函数简单分析**](http://blog.csdn.net/leixiaohua1020/article/details/44117891)**】**

**1) AVCodec->init()**

**7. av\_read\_frame()【**[**函数简单分析**](http://blog.csdn.net/leixiaohua1020/article/details/12678577)**】**

**1) read\_from\_packet\_buffer()**

**2) read\_frame\_internal()**

**(a) ff\_read\_packet()**

**a) AVInputFormat->read\_packet()**

**(b) parse\_packet()**

**a) av\_parser\_parse2()**

**8. avcodec\_decode\_video2()【**[**函数简单分析**](http://blog.csdn.net/leixiaohua1020/article/details/12679719)**】**

**1) av\_packet\_split\_side\_data()**

**2) AVCodec->** **decode()**

**3) av\_frame\_set\_pkt\_pos()**

**4) av\_frame\_set\_best\_effort\_timestamp()**

**9. avcodec\_close()【**[**函数简单分析**](http://blog.csdn.net/leixiaohua1020/article/details/44206699)**】**

**1) AVCodec->close()**

**10. avformat\_close\_input()【**[**函数简单分析**](http://blog.csdn.net/leixiaohua1020/article/details/44110683)**】**

**1) AVInputFormat->read\_close()**

**2) avformat\_free\_context()**

**(a) ff\_free\_stream()**

**3) avio\_close()**

**(a) avio\_flush()**

**a) flush\_buffer()**

**(b) ffurl\_close()**

**a) ffurl\_closep()**

**URLProtocol->url\_close()**

### 右上区域（URLProtocol协议处理函数）

URLProtocol结构体包含如下协议处理函数指针：

**url\_open()：打开  
url\_read()：读取  
url\_write()：写入  
url\_seek()：调整进度  
url\_close()：关闭**

【例子】不同的协议对应着上述接口有不同的实现函数，举几个例子：

**File协议（即文件）对应的URLProtocol结构体ff\_file\_protocol：**

url\_open() -> file\_open() -> open()  
url\_read() -> file\_read() -> read()  
url\_write() -> file\_write() -> write()  
url\_seek() -> file\_seek() -> lseek()  
url\_close() -> file\_close() -> close()

**RTMP协议（libRTMP）对应的URLProtocol结构体ff\_librtmp\_protocol：**

url\_open() -> rtmp\_open() -> RTMP\_Init(), RTMP\_SetupURL(), RTMP\_Connect(), RTMP\_ConnectStream()  
url\_read() -> rtmp\_read() -> RTMP\_Read()  
url\_write() -> rtmp\_write() -> RTMP\_Write()  
url\_seek() -> rtmp\_read\_seek() -> RTMP\_SendSeek()  
url\_close() -> rtmp\_close() -> RTMP\_Close()

**UDP协议对应的URLProtocol结构体ff\_udp\_protocol：**

url\_open() -> udp\_open()  
url\_read() -> udp\_read()  
url\_write() -> udp\_write()  
url\_seek() -> udp\_close()

url\_close() -> udp\_close()

### 右中区域（AVInputFormat封装格式处理函数）

AVInputFormat包含如下封装格式处理函数指针：

**read\_probe()：检查格式  
read\_header()：读取文件头  
read\_packet()：读取一帧数据  
read\_seek()：调整进度  
read\_close()：关闭**

【例子】不同的封装格式对应着上述接口有不同的实现函数，举几个例子：

**FLV封装格式对应的AVInputFormat结构体ff\_flv\_demuxer：**

read\_probe() -> flv\_probe() –> probe()  
read\_header() -> flv\_read\_header() -> create\_stream() -> avformat\_new\_stream()  
read\_packet() -> flv\_read\_packet()  
read\_seek() -> flv\_read\_seek()  
read\_close() -> flv\_read\_close()

**MKV封装格式对应的AVInputFormat结构体ff\_matroska\_demuxer：**

read\_probe() -> matroska\_probe()  
read\_header() -> matroska\_read\_header()  
read\_packet() -> matroska\_read\_packet()  
read\_seek() -> matroska\_read\_seek()  
read\_close() -> matroska\_read\_close()

**MPEG2TS封装格式对应的AVInputFormat结构体ff\_mpegts\_demuxer：**

read\_probe() -> mpegts\_probe()  
read\_header() -> mpegts\_read\_header()  
read\_packet() -> mpegts\_read\_packet()   
read\_close() -> mpegts\_read\_close()

**AVI封装格式对应的AVInputFormat结构体ff\_avi\_demuxer：**

read\_probe() -> avi\_probe()  
read\_header() -> avi\_read\_header()  
read\_packet() -> avi\_read\_packet()  
read\_seek() -> avi\_read\_seek()  
read\_close() -> avi\_read\_close()

### 右下区域（AVCodec编解码函数）

AVCodec包含如下编解码函数指针：

**init()：初始化  
decode()：解码一帧数据  
close()：关闭**

【例子】不同的编解码器对应着上述接口有不同的实现函数，举几个例子：

**HEVC解码对应的AVCodec结构体ff\_hevc\_decoder：**

init() -> hevc\_decode\_init()  
decode() -> hevc\_decode\_frame() -> decode\_nal\_units()  
close() -> hevc\_decode\_free()

**H.264解码对应的AVCodec结构体ff\_h264\_decoder：**

init() -> ff\_h264\_decode\_init()  
decode() -> h264\_decode\_frame() -> decode\_nal\_units()  
close() -> h264\_decode\_end()

**VP8解码（libVPX）对应的AVCodec结构体ff\_libvpx\_vp8\_decoder：**

init() -> vpx\_init() -> vpx\_codec\_dec\_init()  
decode() -> vp8\_decode() -> vpx\_codec\_decode(), vpx\_codec\_get\_frame()  
close() -> vp8\_free() -> vpx\_codec\_destroy()

**MPEG2解码对应的AVCodec结构体ff\_mpeg2video\_decoder：**

init() -> mpeg\_decode\_init()  
decode() -> mpeg\_decode\_frame()

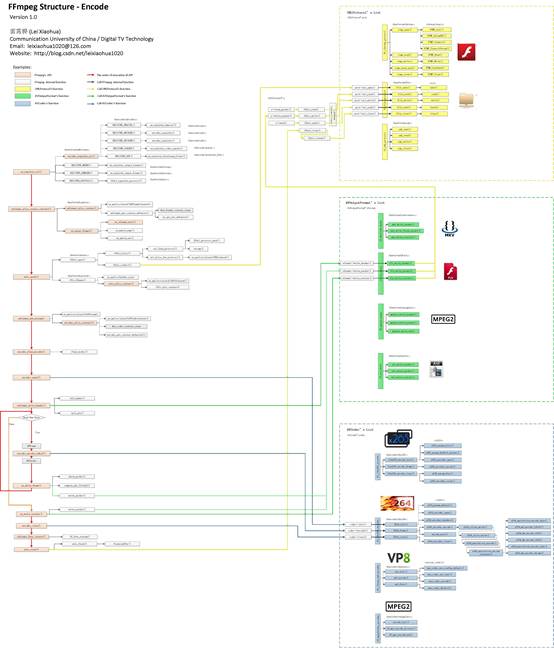
close() -> mpeg\_decode\_end()

# [FFmpeg源代码结构图 - 编码](http://blog.csdn.net/leixiaohua1020/article/details/44226355)

上一篇文章深入分析了FFmpeg解码过程中核心API的内部源代码，本文继续分析FFmpeg编码过程中核心API的内部源代码。本文的编码流程可以参考程序《[最简单的基于FFmpeg的视频编码器](http://blog.csdn.net/leixiaohua1020/article/details/25430425)》。

## 函数调用关系图

首先呈现分析的结果，如图所示。这张图的尺寸很大（大于4000x4000），因此需要打开图片链接之后将图片保存为本地文件，然后才能查看。它表明了FFmpeg在编码一个视频的时候的函数调用流程。为了保证结构清晰，其中仅列出了最关键的函数，剔除了其它不是特别重要的函数。

[](http://img.my.csdn.net/uploads/201503/13/1426229411_4898.jpg)

[单击查看更清晰的图片](https://my.csdn.net/leixiaohua1020/album/detail/1805195)

下面解释一下图中关键标记的含义。

### 函数背景色

函数在图中以方框的形式表现出来。不同的背景色标志了该函数不同的作用：

粉红色背景函数：FFmpeg的API函数。  
白色背景的函数：FFmpeg的内部函数。  
黄色背景的函数：URLProtocol结构体中的函数，包含了读写各种协议的功能。  
绿色背景的函数：AVOutputFormat结构体中的函数，包含了读写各种封装格式的功能。  
蓝色背景的函数：AVCodec结构体中的函数，包含了编解码的功能。

### 区域

整个关系图可以分为以下几个区域：

左边区域——架构函数区域：这些函数并不针对某一特定的视频格式。  
右上方黄色区域——协议处理函数区域：不同的协议（RTP，RTMP，FILE）会调用不同的协议处理函数。  
右边中间绿色区域——封装格式处理函数区域：不同的封装格式（MKV，FLV，MPEG2TS，AVI）会调用不同的封装格式处理函数。  
右边下方蓝色区域——编解码函数区域：不同的编码标准（HEVC，H.264，MPEG2）会调用不同的编解码函数。

### 箭头线

为了把调用关系表示的更明显，图中的箭头线也使用了不同的颜色：

红色的箭头线：标志了编码的流程。  
其他颜色的箭头线：标志了函数之间的调用关系。其中：

调用URLProtocol结构体中的函数用黄色箭头线标识；

调用AVOutputFormat结构体中的函数用绿色箭头线标识；

调用AVCodec结构体中的函数用蓝色箭头线标识。

### 函数所在的文件

每个函数标识了它所在的文件路径。

## 函数功能简述

下面简单列出几个区域中函数之间的调用关系（函数之间的调用关系使用缩进的方式表现出来）。详细的函数分析可以参考相关的《FFmpeg源代码分析》系列文章。

### 左边区域（架构函数）

**1. av\_register\_all()【**[**函数简单分析**](http://blog.csdn.net/leixiaohua1020/article/details/12677129)**】**

**1) avcodec\_register\_all()**

**(a) REGISTER\_HWACCEL()**

**(b) REGISTER\_ENCODER()**

**(c) REGISTER\_DECODER()**

**(d) REGISTER\_PARSER()**

**(e) REGISTER\_BSF()**

**2) REGISTER\_MUXER()**

**3) REGISTER\_DEMUXER()**

**4) REGISTER\_PROTOCOL()**

**2. avformat\_alloc\_output\_context2()【**[**函数简单分析**](http://blog.csdn.net/leixiaohua1020/article/details/41198929)**】**

**1) avformat\_alloc\_context()**

**(a) av\_malloc(sizeof(AVFormatContext))**

**(b) avformat\_get\_context\_defaults()**

**a) av\_opt\_set\_defaults()**

**2) av\_guess\_format()**

**(a) av\_oformat\_next()**

**(b) av\_match\_name()**

**(c) av\_match\_ext()**

**3. avio\_open2()【**[**函数简单分析**](http://blog.csdn.net/leixiaohua1020/article/details/41199947)**】**

**1) ffurl\_open()**

**(a) ffurl\_alloc()**

**a) url\_find\_protocol()**

**b) url\_alloc\_for\_protocol()**

**(b) ffurl\_connect()**

**a) URLProtocol->url\_open()**

**2) ffio\_fdopen()**

**(a) av\_malloc(buffer\_size)**

**(b) avio\_alloc\_context()**

**a) av\_mallocz(sizeof(AVIOContext))**

**b) ffio\_init\_context()**

**4. avformat\_new\_stream()【**[**函数简单分析**](http://blog.csdn.net/leixiaohua1020/article/details/41181155)**】**

**1) av\_mallocz(sizeof(AVStream))**

**2) avcodec\_alloc\_context3()**

**(a) av\_malloc(sizeof(AVCodecContext))**

**(b) avcodec\_get\_context\_defaults3()**

**5. avcodec\_find\_encoder()【**[**函数简单分析**](http://blog.csdn.net/leixiaohua1020/article/details/44084557)**】**

**1) find\_encdec()**

**6. avcodec\_open2()【**[**函数简单分析**](http://blog.csdn.net/leixiaohua1020/article/details/44117891)**】**

**1) AVCodec->init()**

**7. avformat\_write\_header()【**[**函数简单分析**](http://blog.csdn.net/leixiaohua1020/article/details/44116215)**】**

**1) init\_muxer()**

**2) AVOutputFormat->write\_header()**

**3) init\_pts()**

**8. avcodec\_encode\_video2()【**[**函数简单分析**](http://blog.csdn.net/leixiaohua1020/article/details/44206485)**】**

**1) AVCodec->encode2()**

**9. av\_write\_frame()【**[**函数简单分析**](http://blog.csdn.net/leixiaohua1020/article/details/44199673)**】**

**1) check\_packet()**

**2) compute\_pkt\_fields2()**

**3) write\_packet()**

**(a) AVOutputFormat->write\_packet()**

**10. av\_write\_trailer()【**[**函数简单分析**](http://blog.csdn.net/leixiaohua1020/article/details/44201645)**】**

**1) write\_packet()**

**2) AVOutputFormat->write\_trailer()**

**11. avcodec\_close()【**[**函数简单分析**](http://blog.csdn.net/leixiaohua1020/article/details/44206699)**】**

**1) AVCodec->close()**

**12. avformat\_free\_context()【**[**函数简单分析**](http://blog.csdn.net/leixiaohua1020/article/details/44110683)**】**

**1) ff\_free\_stream()**

**13. avio\_close()【**[**函数简单分析**](http://blog.csdn.net/leixiaohua1020/article/details/44110683)**】**

**1) avio\_flush()**

**(a) flush\_buffer()**

**2) ffurl\_close()**

**(a) ffurl\_closep()**

**a) URLProtocol->url\_close()**

### 右上区域（URLProtocol协议处理函数）

URLProtocol结构体包含如下协议处理函数指针：

**url\_open()：打开  
url\_read()：读取  
url\_write()：写入  
url\_seek()：调整进度  
url\_close()：关闭**

【例子】不同的协议对应着上述接口有不同的实现函数，举几个例子：

**File协议（即文件）对应的URLProtocol结构体ff\_file\_protocol：**  
url\_open() -> file\_open() -> open()  
url\_read() -> file\_read() -> read()  
url\_write() -> file\_write() -> write()  
url\_seek() -> file\_seek() -> lseek()

url\_close() -> file\_close() -> close()

**RTMP协议（libRTMP）对应的URLProtocol结构体ff\_librtmp\_protocol：**  
url\_open() -> rtmp\_open() -> RTMP\_Init(), RTMP\_SetupURL(), RTMP\_Connect(), RTMP\_ConnectStream()  
url\_read() -> rtmp\_read() -> RTMP\_Read()  
url\_write() -> rtmp\_write() -> RTMP\_Write()  
url\_seek() -> rtmp\_read\_seek() -> RTMP\_SendSeek()

url\_close() -> rtmp\_close() -> RTMP\_Close()

**UDP协议对应的URLProtocol结构体ff\_udp\_protocol：**  
url\_open() -> udp\_open()  
url\_read() -> udp\_read()  
url\_write() -> udp\_write()  
url\_seek() -> udp\_close()  
url\_close() -> udp\_close()

### 右中区域（AVOutputFormat封装格式处理函数）

AVOutputFormat包含如下封装格式处理函数指针：

write\_header()：写文件头  
write\_packet()：写一帧数据  
write\_trailer()：写文件尾

【例子】不同的封装格式对应着上述接口有不同的实现函数，举几个例子：

**FLV封装格式对应的AVOutputFormat结构体ff\_flv\_muxer：**  
write\_header() -> flv\_write\_header()  
write\_packet() –> flv\_write\_packet()

write\_trailer() -> flv\_write\_trailer()

**MKV封装格式对应的AVOutputFormat结构体ff\_matroska\_muxer：**  
write\_header() -> mkv\_write\_header()  
write\_packet() –> mkv\_write\_flush\_packet()

write\_trailer() -> mkv\_write\_trailer()

**MPEG2TS封装格式对应的AVOutputFormat结构体ff\_mpegts\_muxer：**  
write\_header() -> mpegts\_write\_header()  
write\_packet() –> mpegts\_write\_packet()

write\_trailer() -> mpegts\_write\_end()

**AVI封装格式对应的AVOutputFormat结构体ff\_avi\_muxer：**  
write\_header() -> avi\_write\_header()  
write\_packet() –> avi\_write\_packet()  
write\_trailer() -> avi\_write\_trailer()

### 右下区域（AVCodec编解码函数）

AVCodec包含如下编解码函数指针：

**init()：初始化  
encode2()：编码一帧数据  
close()：关闭**

【例子】不同的编解码器对应着上述接口有不同的实现函数，举几个例子：

**HEVC编码器对应的AVCodec结构体ff\_libx265\_encoder：**  
init() -> libx265\_encode\_init() -> x265\_param\_alloc(), x265\_param\_default\_preset(), x265\_encoder\_open()  
encode2() -> libx265\_encode\_frame() -> x265\_encoder\_encode()

close() -> libx265\_encode\_close() -> x265\_param\_free(), x265\_encoder\_close()

**H.264编码器对应的AVCodec结构体ff\_libx264\_encoder：**  
init() -> X264\_init() -> x264\_param\_default(), x264\_encoder\_open(), x264\_encoder\_headers()  
encode2() -> X264\_frame() -> x264\_encoder\_encode()

close() -> X264\_close() -> x264\_encoder\_close()

**VP8编码器（libVPX）对应的AVCodec结构体ff\_libvpx\_vp8\_encoder：**  
init() -> vpx\_init() -> vpx\_codec\_enc\_config\_default()  
encode2() -> vp8\_encode() -> vpx\_codec\_enc\_init(), vpx\_codec\_encode()

close() -> vp8\_free() -> vpx\_codec\_destroy()

**MPEG2编码器对应的AVCodec结构体ff\_mpeg2video\_encoder：**  
init() -> encode\_init()  
encode2() -> ff\_mpv\_encode\_picture()  
close() -> ff\_mpv\_encode\_end()

# [ffmpeg 源代码简单分析 ： av\_register\_all()](http://blog.csdn.net/leixiaohua1020/article/details/12677129)

前一阵子看了一下ffmpeg的源代码，并且做了一些注释，在此贴出来以作备忘。

本文分析一下ffmpeg注册复用器，编码器等的函数av\_register\_all()。该函数在所有基于ffmpeg的应用程序中几乎都是第一个被调用的。只有调用了该函数，才能使用复用器，编码器等。

可见**解复用器**注册都是用

REGISTER\_DEMUXER  (X,x)

例如：

REGISTER\_DEMUXER  (AAC, aac)

可见**复用器**注册都是用

REGISTER\_MUXER    (X,x))

例如：

REGISTER\_MUXER    (ADTS, adts)

**既有解复用器又有复用器**的话，可以用

REGISTER\_MUXDEMUX (X,x));

例如：

REGISTER\_MUXDEMUX (AC3, ac3);

我们来看一下宏的定义，这里以**解复用器**为例：

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/12677129) [copy](http://blog.csdn.net/leixiaohua1020/article/details/12677129)

[在CODE上查看代码片](https://code.csdn.net/snippets/513242)

1. #define REGISTER\_DEMUXER(X,x) { \
2. **extern** AVInputFormat ff\_##x##\_demuxer; \
3. **if**(CONFIG\_##X##\_DEMUXER) av\_register\_input\_format(&ff\_##x##\_demuxer); }

注意：define里面的##可能不太常见，它的含义就是拼接两个字符串，比如

#define Conn(x,y) x##y

那么

int  n = Conn(123,456);  结果就是n=123456;

我们以REGISTER\_DEMUXER  (AAC, aac)为例，则它等效于

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/12677129) [copy](http://blog.csdn.net/leixiaohua1020/article/details/12677129)

[在CODE上查看代码片](https://code.csdn.net/snippets/513242)

1. **extern** AVInputFormat ff\_aac\_demuxer;
2. **if**(CONFIG\_AAC\_DEMUXER) av\_register\_input\_format(&ff\_aac\_demuxer);

从上面这段代码我们可以看出，真正注册的函数是av\_register\_input\_format(&ff\_aac\_demuxer)，那我就看看这个和函数的作用，查看一下av\_register\_input\_format()的代码：

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/12677129) [copy](http://blog.csdn.net/leixiaohua1020/article/details/12677129)

[在CODE上查看代码片](https://code.csdn.net/snippets/513242)

1. **void** av\_register\_input\_format(AVInputFormat \*format)
2. {
3. AVInputFormat \*\*p;
4. p = &first\_iformat;
5. **while** (\*p != NULL) p = &(\*p)->next;
6. \*p = format;
7. format->next = NULL;
8. }

这段代码是比较容易理解的，首先先提一点，first\_iformat是个什么东东呢？其实它是Input Format链表的头部地址，是一个全局静态变量，定义如下：

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/12677129) [copy](http://blog.csdn.net/leixiaohua1020/article/details/12677129)

[在CODE上查看代码片](https://code.csdn.net/snippets/513242)

1. /\*\* head of registered input format linked list \*/
2. **static** AVInputFormat \*first\_iformat = NULL;

由此我们可以分析出av\_register\_input\_format()的含义，一句话概括就是：遍历链表并把当前的Input Format加到链表的尾部。  
至此REGISTER\_DEMUXER  (X, x)分析完毕。

同理，**复用器**道理是一样的，只是注册函数改为av\_register\_output\_format()；

**既有解复用器又有复用器**的话，有一个宏定义：

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/12677129) [copy](http://blog.csdn.net/leixiaohua1020/article/details/12677129)

[在CODE上查看代码片](https://code.csdn.net/snippets/513242)

1. #define REGISTER\_MUXDEMUX(X,x)  REGISTER\_MUXER(X,x); REGISTER\_DEMUXER(X,x)

可见是分别注册了复用器和解复用器。

此外还有网络协议的注册，注册函数为ffurl\_register\_protocol()，在此不再详述。

下面贴出它的源代码（allformats.c）

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/12677129) [copy](http://blog.csdn.net/leixiaohua1020/article/details/12677129)

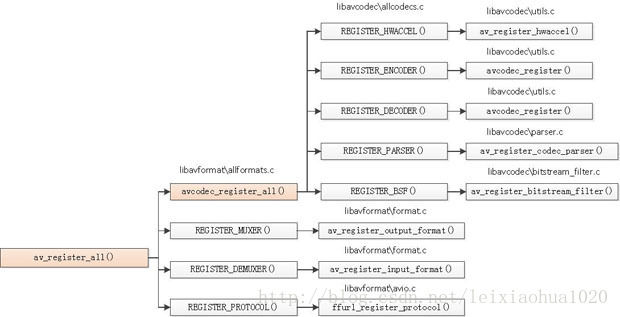
[在CODE上查看代码片](https://code.csdn.net/snippets/513242)

1. /\*
2. \*雷霄骅
3. \*leixiaohua1020@126.com
4. \*中国传媒大学/数字电视技术
5. \*/
6. /\*
7. \* Register all the formats and protocols
8. \* Copyright (c) 2000, 2001, 2002 Fabrice Bellard
9. \*
10. \* This file is part of FFmpeg.
11. \*
12. \* FFmpeg is free software; you can redistribute it and/or
13. \* modify it under the terms of the GNU Lesser General Public
14. \* License as published by the Free Software Foundation; either
15. \* version 2.1 of the License, or (at your option) any later version.
16. \*
17. \* FFmpeg is distributed in the hope that it will be useful,
18. \* but WITHOUT ANY WARRANTY; without even the implied warranty of
19. \* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
20. \* Lesser General Public License for more details.
21. \*
22. \* You should have received a copy of the GNU Lesser General Public
23. \* License along with FFmpeg; if not, write to the Free Software
24. \* Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA
25. \*/
26. #include "avformat.h"
27. #include "rtp.h"
28. #include "rdt.h"
29. #include "url.h"
30. //定义的宏？宏的速度会快一点？注册AVOutputFormat
31. //define中，#用来把参数转换成字符串，##则用来连接前后两个参数，把它们变成一个字符串。
32. //感觉有点像JAva中的EL，可以随意拼接字符串
33. #define REGISTER\_MUXER(X,x) { \
34. **extern** AVOutputFormat ff\_##x##\_muxer; \
35. **if**(CONFIG\_##X##\_MUXER) av\_register\_output\_format(&ff\_##x##\_muxer); }
36. //定义的宏？宏的速度会快一点？注册AVInputFormat
37. #define REGISTER\_DEMUXER(X,x) { \
38. **extern** AVInputFormat ff\_##x##\_demuxer; \
39. **if**(CONFIG\_##X##\_DEMUXER) av\_register\_input\_format(&ff\_##x##\_demuxer); }
40. //注册函数av\_register\_input\_format
42. //定义的宏？宏的速度会快一点？两个一起注册！
43. #define REGISTER\_MUXDEMUX(X,x)  REGISTER\_MUXER(X,x); REGISTER\_DEMUXER(X,x)
44. //定义的宏？宏的速度会快一点？注册URLProtocol
45. //extern URLProtocol ff\_##x##\_protocol;
46. //在librtmp中，对应的就是ff\_rtmp\_protocol
47. //这样就把librtmp整合起来了
48. //由此可见URLProtocol的名字是固定的
49. #define REGISTER\_PROTOCOL(X,x) { \
50. **extern** URLProtocol ff\_##x##\_protocol; \
51. **if**(CONFIG\_##X##\_PROTOCOL) ffurl\_register\_protocol(&ff\_##x##\_protocol, **sizeof**(ff\_##x##\_protocol)); }
52. //注册函数ffurl\_register\_protocol
53. **void** av\_register\_all(**void**)
54. {
55. **static** **int** initialized;
57. **if** (initialized)
58. **return**;
59. initialized = 1;
60. //注册所有的codec
61. avcodec\_register\_all();
62. //注册所有的MUXER（复用器和解复用器）
63. /\* (de)muxers \*/
64. REGISTER\_MUXER    (A64, a64);
65. REGISTER\_DEMUXER  (AAC, aac);
66. REGISTER\_MUXDEMUX (AC3, ac3);
67. REGISTER\_DEMUXER  (ACT, act);
68. REGISTER\_DEMUXER  (ADF, adf);
69. REGISTER\_MUXER    (ADTS, adts);
70. REGISTER\_MUXDEMUX (ADX, adx);
71. REGISTER\_DEMUXER  (AEA, aea);
72. REGISTER\_MUXDEMUX (AIFF, aiff);
73. REGISTER\_MUXDEMUX (AMR, amr);
74. REGISTER\_DEMUXER  (ANM, anm);
75. REGISTER\_DEMUXER  (APC, apc);
76. REGISTER\_DEMUXER  (APE, ape);
77. REGISTER\_DEMUXER  (APPLEHTTP, applehttp);
78. REGISTER\_MUXDEMUX (ASF, asf);
79. REGISTER\_MUXDEMUX (ASS, ass);
80. REGISTER\_MUXER    (ASF\_STREAM, asf\_stream);
81. REGISTER\_MUXDEMUX (AU, au);
82. REGISTER\_MUXDEMUX (AVI, avi);
83. REGISTER\_DEMUXER  (AVISYNTH, avisynth);
84. REGISTER\_MUXER    (AVM2, avm2);
85. REGISTER\_DEMUXER  (AVS, avs);
86. REGISTER\_DEMUXER  (BETHSOFTVID, bethsoftvid);
87. REGISTER\_DEMUXER  (BFI, bfi);
88. REGISTER\_DEMUXER  (BINTEXT, bintext);
89. REGISTER\_DEMUXER  (BINK, bink);
90. REGISTER\_MUXDEMUX (BIT, bit);
91. REGISTER\_DEMUXER  (BMV, bmv);
92. REGISTER\_DEMUXER  (C93, c93);
93. REGISTER\_MUXDEMUX (CAF, caf);
94. REGISTER\_MUXDEMUX (CAVSVIDEO, cavsvideo);
95. REGISTER\_DEMUXER  (CDG, cdg);
96. REGISTER\_MUXER    (CRC, crc);
97. REGISTER\_MUXDEMUX (DAUD, daud);
98. REGISTER\_DEMUXER  (DFA, dfa);
99. REGISTER\_MUXDEMUX (DIRAC, dirac);
100. REGISTER\_MUXDEMUX (DNXHD, dnxhd);
101. REGISTER\_DEMUXER  (DSICIN, dsicin);
102. REGISTER\_MUXDEMUX (DTS, dts);
103. REGISTER\_MUXDEMUX (DV, dv);
104. REGISTER\_DEMUXER  (DXA, dxa);
105. REGISTER\_DEMUXER  (EA, ea);
106. REGISTER\_DEMUXER  (EA\_CDATA, ea\_cdata);
107. REGISTER\_MUXDEMUX (EAC3, eac3);
108. REGISTER\_MUXDEMUX (FFM, ffm);
109. REGISTER\_MUXDEMUX (FFMETADATA, ffmetadata);
110. REGISTER\_MUXDEMUX (FILMSTRIP, filmstrip);
111. REGISTER\_MUXDEMUX (FLAC, flac);
112. REGISTER\_DEMUXER  (FLIC, flic);
113. REGISTER\_MUXDEMUX (FLV, flv);
114. REGISTER\_DEMUXER  (FOURXM, fourxm);
115. REGISTER\_MUXER    (FRAMECRC, framecrc);
116. REGISTER\_MUXER    (FRAMEMD5, framemd5);
117. REGISTER\_MUXDEMUX (G722, g722);
118. REGISTER\_MUXDEMUX (G723\_1, g723\_1);
119. REGISTER\_DEMUXER  (G729, g729);
120. REGISTER\_MUXER    (GIF, gif);
121. REGISTER\_DEMUXER  (GSM, gsm);
122. REGISTER\_MUXDEMUX (GXF, gxf);
123. REGISTER\_MUXDEMUX (H261, h261);
124. REGISTER\_MUXDEMUX (H263, h263);
125. REGISTER\_MUXDEMUX (H264, h264);
126. REGISTER\_DEMUXER  (ICO, ico);
127. REGISTER\_DEMUXER  (IDCIN, idcin);
128. REGISTER\_DEMUXER  (IDF, idf);
129. REGISTER\_DEMUXER  (IFF, iff);
130. REGISTER\_MUXDEMUX (IMAGE2, image2);
131. REGISTER\_MUXDEMUX (IMAGE2PIPE, image2pipe);
132. REGISTER\_DEMUXER  (INGENIENT, ingenient);
133. REGISTER\_DEMUXER  (IPMOVIE, ipmovie);
134. REGISTER\_MUXER    (IPOD, ipod);
135. REGISTER\_MUXER    (ISMV, ismv);
136. REGISTER\_DEMUXER  (ISS, iss);
137. REGISTER\_DEMUXER  (IV8, iv8);
138. REGISTER\_MUXDEMUX (IVF, ivf);
139. REGISTER\_DEMUXER  (JV, jv);
140. REGISTER\_MUXDEMUX (LATM, latm);
141. REGISTER\_DEMUXER  (LMLM4, lmlm4);
142. REGISTER\_DEMUXER  (LOAS, loas);
143. REGISTER\_DEMUXER  (LXF, lxf);
144. REGISTER\_MUXDEMUX (M4V, m4v);
145. REGISTER\_MUXER    (MD5, md5);
146. REGISTER\_MUXDEMUX (MATROSKA, matroska);
147. REGISTER\_MUXER    (MATROSKA\_AUDIO, matroska\_audio);
148. REGISTER\_MUXDEMUX (MICRODVD, microdvd);
149. REGISTER\_MUXDEMUX (MJPEG, mjpeg);
150. REGISTER\_MUXDEMUX (MLP, mlp);
151. REGISTER\_DEMUXER  (MM, mm);
152. REGISTER\_MUXDEMUX (MMF, mmf);
153. REGISTER\_MUXDEMUX (MOV, mov);
154. REGISTER\_MUXER    (MP2, mp2);
155. REGISTER\_MUXDEMUX (MP3, mp3);
156. REGISTER\_MUXER    (MP4, mp4);
157. REGISTER\_DEMUXER  (MPC, mpc);
158. REGISTER\_DEMUXER  (MPC8, mpc8);
159. REGISTER\_MUXER    (MPEG1SYSTEM, mpeg1system);
160. REGISTER\_MUXER    (MPEG1VCD, mpeg1vcd);
161. REGISTER\_MUXER    (MPEG1VIDEO, mpeg1video);
162. REGISTER\_MUXER    (MPEG2DVD, mpeg2dvd);
163. REGISTER\_MUXER    (MPEG2SVCD, mpeg2svcd);
164. REGISTER\_MUXER    (MPEG2VIDEO, mpeg2video);
165. REGISTER\_MUXER    (MPEG2VOB, mpeg2vob);
166. REGISTER\_DEMUXER  (MPEGPS, mpegps);
167. REGISTER\_MUXDEMUX (MPEGTS, mpegts);
168. REGISTER\_DEMUXER  (MPEGTSRAW, mpegtsraw);
169. REGISTER\_DEMUXER  (MPEGVIDEO, mpegvideo);
170. REGISTER\_MUXER    (MPJPEG, mpjpeg);
171. REGISTER\_DEMUXER  (MSNWC\_TCP, msnwc\_tcp);
172. REGISTER\_DEMUXER  (MTV, mtv);
173. REGISTER\_DEMUXER  (MVI, mvi);
174. REGISTER\_MUXDEMUX (MXF, mxf);
175. REGISTER\_MUXER    (MXF\_D10, mxf\_d10);
176. REGISTER\_DEMUXER  (MXG, mxg);
177. REGISTER\_DEMUXER  (NC, nc);
178. REGISTER\_DEMUXER  (NSV, nsv);
179. REGISTER\_MUXER    (NULL, null);
180. REGISTER\_MUXDEMUX (NUT, nut);
181. REGISTER\_DEMUXER  (NUV, nuv);
182. REGISTER\_MUXDEMUX (OGG, ogg);
183. REGISTER\_MUXDEMUX (OMA, oma);
184. REGISTER\_MUXDEMUX (PCM\_ALAW,  pcm\_alaw);
185. REGISTER\_MUXDEMUX (PCM\_MULAW, pcm\_mulaw);
186. REGISTER\_MUXDEMUX (PCM\_F64BE, pcm\_f64be);
187. REGISTER\_MUXDEMUX (PCM\_F64LE, pcm\_f64le);
188. REGISTER\_MUXDEMUX (PCM\_F32BE, pcm\_f32be);
189. REGISTER\_MUXDEMUX (PCM\_F32LE, pcm\_f32le);
190. REGISTER\_MUXDEMUX (PCM\_S32BE, pcm\_s32be);
191. REGISTER\_MUXDEMUX (PCM\_S32LE, pcm\_s32le);
192. REGISTER\_MUXDEMUX (PCM\_S24BE, pcm\_s24be);
193. REGISTER\_MUXDEMUX (PCM\_S24LE, pcm\_s24le);
194. REGISTER\_MUXDEMUX (PCM\_S16BE, pcm\_s16be);
195. REGISTER\_MUXDEMUX (PCM\_S16LE, pcm\_s16le);
196. REGISTER\_MUXDEMUX (PCM\_S8,    pcm\_s8);
197. REGISTER\_MUXDEMUX (PCM\_U32BE, pcm\_u32be);
198. REGISTER\_MUXDEMUX (PCM\_U32LE, pcm\_u32le);
199. REGISTER\_MUXDEMUX (PCM\_U24BE, pcm\_u24be);
200. REGISTER\_MUXDEMUX (PCM\_U24LE, pcm\_u24le);
201. REGISTER\_MUXDEMUX (PCM\_U16BE, pcm\_u16be);
202. REGISTER\_MUXDEMUX (PCM\_U16LE, pcm\_u16le);
203. REGISTER\_MUXDEMUX (PCM\_U8,    pcm\_u8);
204. REGISTER\_DEMUXER  (PMP, pmp);
205. REGISTER\_MUXER    (PSP, psp);
206. REGISTER\_DEMUXER  (PVA, pva);
207. REGISTER\_DEMUXER  (QCP, qcp);
208. REGISTER\_DEMUXER  (R3D, r3d);
209. REGISTER\_MUXDEMUX (RAWVIDEO, rawvideo);
210. REGISTER\_DEMUXER  (RL2, rl2);
211. REGISTER\_MUXDEMUX (RM, rm);
212. REGISTER\_MUXDEMUX (ROQ, roq);
213. REGISTER\_DEMUXER  (RPL, rpl);
214. REGISTER\_MUXDEMUX (RSO, rso);
215. REGISTER\_MUXDEMUX (RTP, rtp);
216. REGISTER\_MUXDEMUX (RTSP, rtsp);
217. REGISTER\_MUXDEMUX (SAP, sap);
218. REGISTER\_DEMUXER  (SBG, sbg);
219. REGISTER\_DEMUXER  (SDP, sdp);
220. #if CONFIG\_RTPDEC
221. av\_register\_rtp\_dynamic\_payload\_handlers();
222. av\_register\_rdt\_dynamic\_payload\_handlers();
223. #endif
224. REGISTER\_DEMUXER  (SEGAFILM, segafilm);
225. REGISTER\_MUXER    (SEGMENT, segment);
226. REGISTER\_DEMUXER  (SHORTEN, shorten);
227. REGISTER\_DEMUXER  (SIFF, siff);
228. REGISTER\_DEMUXER  (SMACKER, smacker);
229. REGISTER\_MUXDEMUX (SMJPEG, smjpeg);
230. REGISTER\_DEMUXER  (SOL, sol);
231. REGISTER\_MUXDEMUX (SOX, sox);
232. REGISTER\_MUXDEMUX (SPDIF, spdif);
233. REGISTER\_MUXDEMUX (SRT, srt);
234. REGISTER\_DEMUXER  (STR, str);
235. REGISTER\_MUXDEMUX (SWF, swf);
236. REGISTER\_MUXER    (TG2, tg2);
237. REGISTER\_MUXER    (TGP, tgp);
238. REGISTER\_DEMUXER  (THP, thp);
239. REGISTER\_DEMUXER  (TIERTEXSEQ, tiertexseq);
240. REGISTER\_MUXER    (MKVTIMESTAMP\_V2, mkvtimestamp\_v2);
241. REGISTER\_DEMUXER  (TMV, tmv);
242. REGISTER\_MUXDEMUX (TRUEHD, truehd);
243. REGISTER\_DEMUXER  (TTA, tta);
244. REGISTER\_DEMUXER  (TXD, txd);
245. REGISTER\_DEMUXER  (TTY, tty);
246. REGISTER\_DEMUXER  (VC1, vc1);
247. REGISTER\_MUXDEMUX (VC1T, vc1t);
248. REGISTER\_DEMUXER  (VMD, vmd);
249. REGISTER\_MUXDEMUX (VOC, voc);
250. REGISTER\_DEMUXER  (VQF, vqf);
251. REGISTER\_DEMUXER  (W64, w64);
252. REGISTER\_MUXDEMUX (WAV, wav);
253. REGISTER\_DEMUXER  (WC3, wc3);
254. REGISTER\_MUXER    (WEBM, webm);
255. REGISTER\_DEMUXER  (WSAUD, wsaud);
256. REGISTER\_DEMUXER  (WSVQA, wsvqa);
257. REGISTER\_MUXDEMUX (WTV, wtv);
258. REGISTER\_DEMUXER  (WV, wv);
259. REGISTER\_DEMUXER  (XA, xa);
260. REGISTER\_DEMUXER  (XBIN, xbin);
261. REGISTER\_DEMUXER  (XMV, xmv);
262. REGISTER\_DEMUXER  (XWMA, xwma);
263. REGISTER\_DEMUXER  (YOP, yop);
264. REGISTER\_MUXDEMUX (YUV4MPEGPIPE, yuv4mpegpipe);
266. /\* external libraries \*/
267. #if CONFIG\_LIBMODPLUG
268. REGISTER\_DEMUXER  (LIBMODPLUG, libmodplug);
269. #endif
270. REGISTER\_MUXDEMUX (LIBNUT, libnut);
271. //注册所有的Protocol（位于DEMUXER之前（我的理解~~））
272. //文件也是一种Protocol
273. /\* protocols \*/
274. REGISTER\_PROTOCOL (APPLEHTTP, applehttp);
275. REGISTER\_PROTOCOL (CACHE, cache);
276. REGISTER\_PROTOCOL (CONCAT, concat);
277. REGISTER\_PROTOCOL (CRYPTO, crypto);
278. REGISTER\_PROTOCOL (**FILE**, file);
279. REGISTER\_PROTOCOL (GOPHER, gopher);
280. REGISTER\_PROTOCOL (HTTP, http);
281. REGISTER\_PROTOCOL (HTTPPROXY, httpproxy);
282. REGISTER\_PROTOCOL (HTTPS, https);
283. REGISTER\_PROTOCOL (MMSH, mmsh);
284. REGISTER\_PROTOCOL (MMST, mmst);
285. REGISTER\_PROTOCOL (MD5,  md5);
286. REGISTER\_PROTOCOL (PIPE, pipe);
287. REGISTER\_PROTOCOL (RTMP, rtmp);
288. //如果包含了LibRTMP
289. #if CONFIG\_LIBRTMP
290. REGISTER\_PROTOCOL (RTMP, rtmpt);
291. REGISTER\_PROTOCOL (RTMP, rtmpe);
292. REGISTER\_PROTOCOL (RTMP, rtmpte);
293. REGISTER\_PROTOCOL (RTMP, rtmps);
294. #endif
295. REGISTER\_PROTOCOL (RTP, rtp);
296. REGISTER\_PROTOCOL (TCP, tcp);
297. REGISTER\_PROTOCOL (TLS, tls);
298. REGISTER\_PROTOCOL (UDP, udp);
299. }

整个代码没太多可说的，首先确定是不是已经初始化过了（initialized），如果没有，就调用avcodec\_register\_all()注册编解码器（这个先不分析），然后就是注册，注册，注册...直到完成所有注册。

PS：曾经研究过一阵子RTMP协议，以及对应的开源工程librtmp。在这里发现有一点值得注意，ffmpeg自带了RTMP协议的支持，只有使用 rtmpt://, rtmpe://, rtmpte://等的时候才会使用librtmp库。

函数调用关系图如下图所示。av\_register\_all()调用了avcodec\_register\_all()。avcodec\_register\_all()注册了和编解码器有关的组件：硬件加速器，解码器，编码器，Parser，Bitstream Filter。av\_register\_all()除了调用avcodec\_register\_all()之外，还注册了复用器，解复用器，协议处理器。



下面附上复用器，解复用器，协议处理器的代码。

注册复用器的函数是av\_register\_output\_format()。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/12677129) [copy](http://blog.csdn.net/leixiaohua1020/article/details/12677129)

[在CODE上查看代码片](https://code.csdn.net/snippets/513242)

1. **void** av\_register\_output\_format(AVOutputFormat \*format)
2. {
3. AVOutputFormat \*\*p;
4. p = &first\_oformat;
5. **while** (\*p != NULL) p = &(\*p)->next;
6. \*p = format;
7. format->next = NULL;
8. }

注册解复用器的函数是av\_register\_input\_format()。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/12677129) [copy](http://blog.csdn.net/leixiaohua1020/article/details/12677129)

[在CODE上查看代码片](https://code.csdn.net/snippets/513242)

1. **void** av\_register\_input\_format(AVInputFormat \*format)
2. {
3. AVInputFormat \*\*p;
4. p = &first\_iformat;
5. **while** (\*p != NULL) p = &(\*p)->next;
6. \*p = format;
7. format->next = NULL;
8. }

注册协议处理器的函数是ffurl\_register\_protocol()。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/12677129) [copy](http://blog.csdn.net/leixiaohua1020/article/details/12677129)

[在CODE上查看代码片](https://code.csdn.net/snippets/513242)

1. **int** ffurl\_register\_protocol(URLProtocol \*protocol)
2. {
3. URLProtocol \*\*p;
4. p = &first\_protocol;
5. **while** (\*p)
6. p = &(\*p)->next;
7. \*p             = protocol;
8. protocol->next = NULL;
9. **return** 0;
10. }

# [ffmpeg 源代码简单分析 ： avcodec\_register\_all()](http://blog.csdn.net/leixiaohua1020/article/details/12677265)

本文分析一下ffmpeg注册编解码器等的函数avcodec\_register\_all()（注意不是av\_register\_all()，那是注册所有东西的）。该函数在所有基于ffmpeg的应用程序中几乎都是第一个被调用的。只有调用了该函数，才能使用编解码器等。

之前已经写过一篇文章了：

[ffmpeg 源代码简单分析 ： av\_register\_all()](http://blog.csdn.net/leixiaohua1020/article/details/12677129)

其实注册编解码器和注册复用器解复用器道理是差不多的，重复的内容不再多说。

**编码器**的注册是：

REGISTER\_ENCODER (X,x);

例如：

REGISTER\_ENCODER (LJPEG, ljpeg);

**解码器**的注册是：

REGISTER\_DECODER (X,x);

例如：

REGISTER\_DECODER (H264, h264);

**既包含编码器有包含解码器**的注册是：

REGISTER\_ENCDEC  (X,x);

例如：

REGISTER\_ENCDEC  (BMP, bmp);

此外还有几种注册：

**Parser：**

REGISTER\_PARSER  (X,x);

例如：

REGISTER\_PARSER  (H264, h264);

**BSF（bitstream filters，比特流滤镜，有一个常用：h264\_mp4toannexb）：**

REGISTER\_BSF     (X,x);

例如：

REGISTER\_BSF     (H264\_MP4TOANNEXB, h264\_mp4toannexb);

**HWACCEL（hardware accelerators，硬件加速器）：**

REGISTER\_HWACCEL (X,x);

例如：

REGISTER\_HWACCEL (H264\_DXVA2, h264\_dxva2);

我们来看一下宏的定义，这里以编解码器为例：

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/12677265) [copy](http://blog.csdn.net/leixiaohua1020/article/details/12677265)

[在CODE上查看代码片](https://code.csdn.net/snippets/513247)

1. #define REGISTER\_ENCODER(X,x) { \
2. **extern** AVCodec ff\_##x##\_encoder; \
3. **if**(CONFIG\_##X##\_ENCODER)  avcodec\_register(&ff\_##x##\_encoder); }
4. #define REGISTER\_DECODER(X,x) { \
5. **extern** AVCodec ff\_##x##\_decoder; \
6. **if**(CONFIG\_##X##\_DECODER)  avcodec\_register(&ff\_##x##\_decoder); }
7. #define REGISTER\_ENCDEC(X,x)  REGISTER\_ENCODER(X,x); REGISTER\_DECODER(X,x)

在这里，我发现其实编码器和解码器用的注册函数都是一样的：avcodec\_register()

以REGISTER\_DECODER (H264, h264)为例，就是等效于

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/12677265) [copy](http://blog.csdn.net/leixiaohua1020/article/details/12677265)

[在CODE上查看代码片](https://code.csdn.net/snippets/513247)

1. **extern** AVCodec ff\_h264\_decoder;
2. **if**(CONFIG\_H264\_DECODER)  avcodec\_register(&ff\_h264\_decoder);

下面看一下avcodec\_register()的源代码：

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/12677265) [copy](http://blog.csdn.net/leixiaohua1020/article/details/12677265)

[在CODE上查看代码片](https://code.csdn.net/snippets/513247)

1. //注册所有的AVCodec
2. **void** avcodec\_register(AVCodec \*codec)
3. {
4. AVCodec \*\*p;
5. //初始化
6. avcodec\_init();
7. //从第一个开始
8. p = &first\_avcodec;
9. **while** (\*p != NULL) p = &(\*p)->next;
10. \*p = codec;
11. codec->next = NULL;
13. **if** (codec->init\_static\_data)
14. codec->init\_static\_data(codec);
15. }

这段代码是比较容易理解的。首先先提一点，first\_avcdec是就是AVCodec链表的头部地址，是一个全局静态变量，定义如下：

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/12677265) [copy](http://blog.csdn.net/leixiaohua1020/article/details/12677265)

[在CODE上查看代码片](https://code.csdn.net/snippets/513247)

1. /\* encoder management \*/
2. **static** AVCodec \*first\_avcodec = NULL;

由此我们可以分析出avcodec\_register()的含义，一句话概括就是：遍历链表并把当前的AVCodec加到链表的尾部。

同理，**Parser**，**BSF（bitstream filters，比特流滤镜）**，**HWACCEL（hardware accelerators，硬件加速器）**的注册方式都是类似的。不再详述。

下面贴出它的原代码：

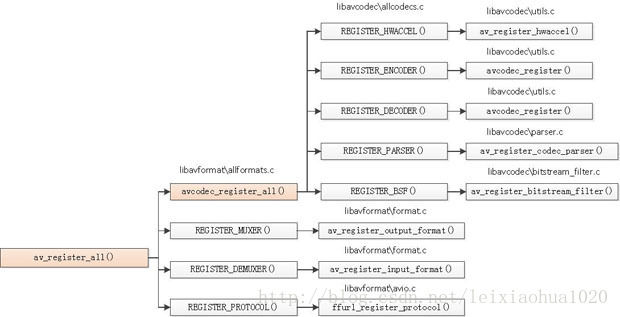
**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/12677265) [copy](http://blog.csdn.net/leixiaohua1020/article/details/12677265)

[在CODE上查看代码片](https://code.csdn.net/snippets/513247)

1. /\*
2. \*雷霄骅
3. \*leixiaohua1020@126.com
4. \*中国传媒大学/数字电视技术
5. \*/
6. /\*
7. \* Provide registration of all codecs, parsers and bitstream filters for libavcodec.
8. \* Copyright (c) 2002 Fabrice Bellard
9. \*
10. \* This file is part of FFmpeg.
11. \*
12. \* FFmpeg is free software; you can redistribute it and/or
13. \* modify it under the terms of the GNU Lesser General Public
14. \* License as published by the Free Software Foundation; either
15. \* version 2.1 of the License, or (at your option) any later version.
16. \*
17. \* FFmpeg is distributed in the hope that it will be useful,
18. \* but WITHOUT ANY WARRANTY; without even the implied warranty of
19. \* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
20. \* Lesser General Public License for more details.
21. \*
22. \* You should have received a copy of the GNU Lesser General Public
23. \* License along with FFmpeg; if not, write to the Free Software
24. \* Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA
25. \*/
27. /\*\*
28. \* @file
29. \* Provide registration of all codecs, parsers and bitstream filters for libavcodec.
30. \*/
32. #include "avcodec.h"
33. //硬件加速
34. #define REGISTER\_HWACCEL(X,x) { \
35. **extern** AVHWAccel ff\_##x##\_hwaccel; \
36. **if**(CONFIG\_##X##\_HWACCEL) av\_register\_hwaccel(&ff\_##x##\_hwaccel); }
38. #define REGISTER\_ENCODER(X,x) { \
39. **extern** AVCodec ff\_##x##\_encoder; \
40. **if**(CONFIG\_##X##\_ENCODER)  avcodec\_register(&ff\_##x##\_encoder); }
41. //定义的宏？宏的速度会快一点？注册AVCodec
42. //extern AVCodec ff\_##x##\_decoder;
43. //注意：extern表明全局唯一
44. //在h264中，对应的就是ff\_h264\_decoder
45. //由此可见AVCodecParser的名字是固定的
46. #define REGISTER\_DECODER(X,x) { \
47. **extern** AVCodec ff\_##x##\_decoder; \
48. **if**(CONFIG\_##X##\_DECODER)  avcodec\_register(&ff\_##x##\_decoder); }
49. #define REGISTER\_ENCDEC(X,x)  REGISTER\_ENCODER(X,x); REGISTER\_DECODER(X,x)
50. //定义的宏？宏的速度会快一点？注册AVCodecParser
51. //extern AVCodecParser ff\_##x##\_parser;
52. //在h264中，对应的就是ff\_h264\_parser
53. //由此可见AVCodecParser的名字是固定的
54. #define REGISTER\_PARSER(X,x) { \
55. **extern** AVCodecParser ff\_##x##\_parser; \
56. **if**(CONFIG\_##X##\_PARSER)  av\_register\_codec\_parser(&ff\_##x##\_parser); }
57. #define REGISTER\_BSF(X,x) { \
58. **extern** AVBitStreamFilter ff\_##x##\_bsf; \
59. **if**(CONFIG\_##X##\_BSF)     av\_register\_bitstream\_filter(&ff\_##x##\_bsf); }
61. **void** avcodec\_register\_all(**void**)
62. {
63. **static** **int** initialized;
65. **if** (initialized)
66. **return**;
67. initialized = 1;
69. /\* hardware accelerators \*/
70. REGISTER\_HWACCEL (H263\_VAAPI, h263\_vaapi);
71. REGISTER\_HWACCEL (H264\_DXVA2, h264\_dxva2);
72. REGISTER\_HWACCEL (H264\_VAAPI, h264\_vaapi);
73. REGISTER\_HWACCEL (H264\_VDA, h264\_vda);
74. REGISTER\_HWACCEL (MPEG1\_VDPAU, mpeg1\_vdpau);
75. REGISTER\_HWACCEL (MPEG2\_DXVA2, mpeg2\_dxva2);
76. REGISTER\_HWACCEL (MPEG2\_VAAPI, mpeg2\_vaapi);
77. REGISTER\_HWACCEL (MPEG2\_VDPAU, mpeg2\_vdpau);
78. REGISTER\_HWACCEL (MPEG4\_VAAPI, mpeg4\_vaapi);
79. REGISTER\_HWACCEL (VC1\_DXVA2, vc1\_dxva2);
80. REGISTER\_HWACCEL (VC1\_VAAPI, vc1\_vaapi);
81. REGISTER\_HWACCEL (WMV3\_DXVA2, wmv3\_dxva2);
82. REGISTER\_HWACCEL (WMV3\_VAAPI, wmv3\_vaapi);
84. /\* video codecs \*/
85. REGISTER\_ENCODER (A64MULTI, a64multi);
86. REGISTER\_ENCODER (A64MULTI5, a64multi5);
87. REGISTER\_DECODER (AASC, aasc);
88. REGISTER\_ENCDEC  (AMV, amv);
89. REGISTER\_DECODER (ANM, anm);
90. REGISTER\_DECODER (ANSI, ansi);
91. REGISTER\_ENCDEC  (ASV1, asv1);
92. REGISTER\_ENCDEC  (ASV2, asv2);
93. REGISTER\_DECODER (AURA, aura);
94. REGISTER\_DECODER (AURA2, aura2);
95. REGISTER\_ENCDEC  (AVRP, avrp);
96. REGISTER\_DECODER (AVS, avs);
97. REGISTER\_DECODER (BETHSOFTVID, bethsoftvid);
98. REGISTER\_DECODER (BFI, bfi);
99. REGISTER\_DECODER (BINK, bink);
100. REGISTER\_ENCDEC  (BMP, bmp);
101. REGISTER\_DECODER (BMV\_VIDEO, bmv\_video);
102. REGISTER\_DECODER (C93, c93);
103. REGISTER\_DECODER (CAVS, cavs);
104. REGISTER\_DECODER (CDGRAPHICS, cdgraphics);
105. REGISTER\_DECODER (CINEPAK, cinepak);
106. REGISTER\_ENCDEC  (CLJR, cljr);
107. REGISTER\_DECODER (CSCD, cscd);
108. REGISTER\_DECODER (CYUV, cyuv);
109. REGISTER\_DECODER (DFA, dfa);
110. REGISTER\_DECODER (DIRAC, dirac);
111. REGISTER\_ENCDEC  (DNXHD, dnxhd);
112. REGISTER\_ENCDEC  (DPX, dpx);
113. REGISTER\_DECODER (DSICINVIDEO, dsicinvideo);
114. REGISTER\_ENCDEC  (DVVIDEO, dvvideo);
115. REGISTER\_DECODER (DXA, dxa);
116. REGISTER\_DECODER (DXTORY, dxtory);
117. REGISTER\_DECODER (EACMV, eacmv);
118. REGISTER\_DECODER (EAMAD, eamad);
119. REGISTER\_DECODER (EATGQ, eatgq);
120. REGISTER\_DECODER (EATGV, eatgv);
121. REGISTER\_DECODER (EATQI, eatqi);
122. REGISTER\_DECODER (EIGHTBPS, eightbps);
123. REGISTER\_DECODER (EIGHTSVX\_EXP, eightsvx\_exp);
124. REGISTER\_DECODER (EIGHTSVX\_FIB, eightsvx\_fib);
125. REGISTER\_DECODER (ESCAPE124, escape124);
126. REGISTER\_DECODER (ESCAPE130, escape130);
127. REGISTER\_ENCDEC  (FFV1, ffv1);
128. REGISTER\_ENCDEC  (FFVHUFF, ffvhuff);
129. REGISTER\_ENCDEC  (FLASHSV, flashsv);
130. REGISTER\_ENCDEC  (FLASHSV2, flashsv2);
131. REGISTER\_DECODER (FLIC, flic);
132. REGISTER\_ENCDEC  (FLV, flv);
133. REGISTER\_DECODER (FOURXM, fourxm);
134. REGISTER\_DECODER (FRAPS, fraps);
135. REGISTER\_DECODER (FRWU, frwu);
136. REGISTER\_ENCDEC  (GIF, gif);
137. REGISTER\_ENCDEC  (H261, h261);
138. REGISTER\_ENCDEC  (H263, h263);
139. REGISTER\_DECODER (H263I, h263i);
140. REGISTER\_ENCODER (H263P, h263p);
141. REGISTER\_DECODER (H264, h264);
142. REGISTER\_DECODER (H264\_CRYSTALHD, h264\_crystalhd);
143. REGISTER\_DECODER (H264\_VDPAU, h264\_vdpau);
144. REGISTER\_ENCDEC  (HUFFYUV, huffyuv);
145. REGISTER\_DECODER (IDCIN, idcin);
146. REGISTER\_DECODER (IFF\_BYTERUN1, iff\_byterun1);
147. REGISTER\_DECODER (IFF\_ILBM, iff\_ilbm);
148. REGISTER\_DECODER (INDEO2, indeo2);
149. REGISTER\_DECODER (INDEO3, indeo3);
150. REGISTER\_DECODER (INDEO4, indeo4);
151. REGISTER\_DECODER (INDEO5, indeo5);
152. REGISTER\_DECODER (INTERPLAY\_VIDEO, interplay\_video);
153. REGISTER\_ENCDEC  (JPEG2000, jpeg2000);
154. REGISTER\_ENCDEC  (JPEGLS, jpegls);
155. REGISTER\_DECODER (JV, jv);
156. REGISTER\_DECODER (KGV1, kgv1);
157. REGISTER\_DECODER (KMVC, kmvc);
158. REGISTER\_DECODER (LAGARITH, lagarith);
159. REGISTER\_ENCODER (LJPEG, ljpeg);
160. REGISTER\_DECODER (LOCO, loco);
161. REGISTER\_DECODER (MDEC, mdec);
162. REGISTER\_DECODER (MIMIC, mimic);
163. REGISTER\_ENCDEC  (MJPEG, mjpeg);
164. REGISTER\_DECODER (MJPEGB, mjpegb);
165. REGISTER\_DECODER (MMVIDEO, mmvideo);
166. REGISTER\_DECODER (MOTIONPIXELS, motionpixels);
167. REGISTER\_DECODER (MPEG\_XVMC, mpeg\_xvmc);
168. REGISTER\_ENCDEC  (MPEG1VIDEO, mpeg1video);
169. REGISTER\_ENCDEC  (MPEG2VIDEO, mpeg2video);
170. REGISTER\_ENCDEC  (MPEG4, mpeg4);
171. REGISTER\_DECODER (MPEG4\_CRYSTALHD, mpeg4\_crystalhd);
172. REGISTER\_DECODER (MPEG4\_VDPAU, mpeg4\_vdpau);
173. REGISTER\_DECODER (MPEGVIDEO, mpegvideo);
174. REGISTER\_DECODER (MPEG\_VDPAU, mpeg\_vdpau);
175. REGISTER\_DECODER (MPEG1\_VDPAU, mpeg1\_vdpau);
176. REGISTER\_DECODER (MPEG2\_CRYSTALHD, mpeg2\_crystalhd);
177. REGISTER\_DECODER (MSMPEG4\_CRYSTALHD, msmpeg4\_crystalhd);
178. REGISTER\_DECODER (MSMPEG4V1, msmpeg4v1);
179. REGISTER\_ENCDEC  (MSMPEG4V2, msmpeg4v2);
180. REGISTER\_ENCDEC  (MSMPEG4V3, msmpeg4v3);
181. REGISTER\_DECODER (MSRLE, msrle);
182. REGISTER\_ENCDEC  (MSVIDEO1, msvideo1);
183. REGISTER\_DECODER (MSZH, mszh);
184. REGISTER\_DECODER (MXPEG, mxpeg);
185. REGISTER\_DECODER (NUV, nuv);
186. REGISTER\_ENCDEC  (PAM, pam);
187. REGISTER\_ENCDEC  (PBM, pbm);
188. REGISTER\_ENCDEC  (PCX, pcx);
189. REGISTER\_ENCDEC  (PGM, pgm);
190. REGISTER\_ENCDEC  (PGMYUV, pgmyuv);
191. REGISTER\_DECODER (PICTOR, pictor);
192. REGISTER\_ENCDEC  (PNG, png);
193. REGISTER\_ENCDEC  (PPM, ppm);
194. REGISTER\_ENCDEC  (PRORES, prores);
195. REGISTER\_DECODER (PRORES\_LGPL, prores\_lgpl);
196. REGISTER\_DECODER (PTX, ptx);
197. REGISTER\_DECODER (QDRAW, qdraw);
198. REGISTER\_DECODER (QPEG, qpeg);
199. REGISTER\_ENCDEC  (QTRLE, qtrle);
200. REGISTER\_ENCDEC  (R10K,  r10k);
201. REGISTER\_ENCDEC  (R210,  r210);
202. REGISTER\_ENCDEC  (RAWVIDEO, rawvideo);
203. REGISTER\_DECODER (RL2, rl2);
204. REGISTER\_ENCDEC  (ROQ, roq);
205. REGISTER\_DECODER (RPZA, rpza);
206. REGISTER\_ENCDEC  (RV10, rv10);
207. REGISTER\_ENCDEC  (RV20, rv20);
208. REGISTER\_DECODER (RV30, rv30);
209. REGISTER\_DECODER (RV40, rv40);
210. REGISTER\_DECODER (S302M, s302m);
211. REGISTER\_ENCDEC  (SGI, sgi);
212. REGISTER\_DECODER (SMACKER, smacker);
213. REGISTER\_DECODER (SMC, smc);
214. REGISTER\_ENCDEC  (SNOW, snow);
215. REGISTER\_DECODER (SP5X, sp5x);
216. REGISTER\_DECODER (SUNRAST, sunrast);
217. REGISTER\_ENCDEC  (SVQ1, svq1);
218. REGISTER\_DECODER (SVQ3, svq3);
219. REGISTER\_ENCDEC  (TARGA, targa);
220. REGISTER\_DECODER (THEORA, theora);
221. REGISTER\_DECODER (THP, thp);
222. REGISTER\_DECODER (TIERTEXSEQVIDEO, tiertexseqvideo);
223. REGISTER\_ENCDEC  (TIFF, tiff);
224. REGISTER\_DECODER (TMV, tmv);
225. REGISTER\_DECODER (TRUEMOTION1, truemotion1);
226. REGISTER\_DECODER (TRUEMOTION2, truemotion2);
227. REGISTER\_DECODER (TSCC, tscc);
228. REGISTER\_DECODER (TXD, txd);
229. REGISTER\_DECODER (ULTI, ulti);
230. REGISTER\_DECODER (UTVIDEO, utvideo);
231. REGISTER\_ENCDEC  (V210,  v210);
232. REGISTER\_DECODER (V210X, v210x);
233. REGISTER\_ENCDEC  (V308, v308);
234. REGISTER\_ENCDEC  (V410, v410);
235. REGISTER\_DECODER (VB, vb);
236. REGISTER\_DECODER (VBLE, vble);
237. REGISTER\_DECODER (VC1, vc1);
238. REGISTER\_DECODER (VC1\_CRYSTALHD, vc1\_crystalhd);
239. REGISTER\_DECODER (VC1\_VDPAU, vc1\_vdpau);
240. REGISTER\_DECODER (VC1IMAGE, vc1image);
241. REGISTER\_DECODER (VCR1, vcr1);
242. REGISTER\_DECODER (VMDVIDEO, vmdvideo);
243. REGISTER\_DECODER (VMNC, vmnc);
244. REGISTER\_DECODER (VP3, vp3);
245. REGISTER\_DECODER (VP5, vp5);
246. REGISTER\_DECODER (VP6, vp6);
247. REGISTER\_DECODER (VP6A, vp6a);
248. REGISTER\_DECODER (VP6F, vp6f);
249. REGISTER\_DECODER (VP8, vp8);
250. REGISTER\_DECODER (VQA, vqa);
251. REGISTER\_ENCDEC  (WMV1, wmv1);
252. REGISTER\_ENCDEC  (WMV2, wmv2);
253. REGISTER\_DECODER (WMV3, wmv3);
254. REGISTER\_DECODER (WMV3\_CRYSTALHD, wmv3\_crystalhd);
255. REGISTER\_DECODER (WMV3\_VDPAU, wmv3\_vdpau);
256. REGISTER\_DECODER (WMV3IMAGE, wmv3image);
257. REGISTER\_DECODER (WNV1, wnv1);
258. REGISTER\_DECODER (XAN\_WC3, xan\_wc3);
259. REGISTER\_DECODER (XAN\_WC4, xan\_wc4);
260. REGISTER\_DECODER (XL, xl);
261. REGISTER\_ENCDEC  (XWD, xwd);
262. REGISTER\_ENCDEC  (Y41P, y41p);
263. REGISTER\_DECODER (YOP, yop);
264. REGISTER\_ENCDEC  (YUV4, yuv4);
265. REGISTER\_ENCDEC  (ZLIB, zlib);
266. REGISTER\_ENCDEC  (ZMBV, zmbv);
268. /\* audio codecs \*/
269. REGISTER\_ENCDEC  (AAC, aac);
270. REGISTER\_DECODER (AAC\_LATM, aac\_latm);
271. REGISTER\_ENCDEC  (AC3, ac3);
272. REGISTER\_ENCODER (AC3\_FIXED, ac3\_fixed);
273. REGISTER\_ENCDEC  (ALAC, alac);
274. REGISTER\_DECODER (ALS, als);
275. REGISTER\_DECODER (AMRNB, amrnb);
276. REGISTER\_DECODER (AMRWB, amrwb);
277. REGISTER\_DECODER (APE, ape);
278. REGISTER\_DECODER (ATRAC1, atrac1);
279. REGISTER\_DECODER (ATRAC3, atrac3);
280. REGISTER\_DECODER (BINKAUDIO\_DCT, binkaudio\_dct);
281. REGISTER\_DECODER (BINKAUDIO\_RDFT, binkaudio\_rdft);
282. REGISTER\_DECODER (BMV\_AUDIO, bmv\_audio);
283. REGISTER\_DECODER (COOK, cook);
284. REGISTER\_ENCDEC  (DCA, dca);
285. REGISTER\_DECODER (DSICINAUDIO, dsicinaudio);
286. REGISTER\_ENCDEC  (EAC3, eac3);
287. REGISTER\_DECODER (FFWAVESYNTH, ffwavesynth);
288. REGISTER\_ENCDEC  (FLAC, flac);
289. REGISTER\_ENCDEC  (G723\_1, g723\_1);
290. REGISTER\_DECODER (G729, g729);
291. REGISTER\_DECODER (GSM, gsm);
292. REGISTER\_DECODER (GSM\_MS, gsm\_ms);
293. REGISTER\_DECODER (IMC, imc);
294. REGISTER\_DECODER (MACE3, mace3);
295. REGISTER\_DECODER (MACE6, mace6);
296. REGISTER\_DECODER (MLP, mlp);
297. REGISTER\_DECODER (MP1, mp1);
298. REGISTER\_DECODER (MP1FLOAT, mp1float);
299. REGISTER\_ENCDEC  (MP2, mp2);
300. REGISTER\_DECODER (MP2FLOAT, mp2float);
301. REGISTER\_DECODER (MP3, mp3);
302. REGISTER\_DECODER (MP3FLOAT, mp3float);
303. REGISTER\_DECODER (MP3ADU, mp3adu);
304. REGISTER\_DECODER (MP3ADUFLOAT, mp3adufloat);
305. REGISTER\_DECODER (MP3ON4, mp3on4);
306. REGISTER\_DECODER (MP3ON4FLOAT, mp3on4float);
307. REGISTER\_DECODER (MPC7, mpc7);
308. REGISTER\_DECODER (MPC8, mpc8);
309. REGISTER\_ENCDEC  (NELLYMOSER, nellymoser);
310. REGISTER\_DECODER (QCELP, qcelp);
311. REGISTER\_DECODER (QDM2, qdm2);
312. REGISTER\_ENCDEC  (RA\_144, ra\_144);
313. REGISTER\_DECODER (RA\_288, ra\_288);
314. REGISTER\_DECODER (SHORTEN, shorten);
315. REGISTER\_DECODER (SIPR, sipr);
316. REGISTER\_DECODER (SMACKAUD, smackaud);
317. REGISTER\_ENCDEC  (SONIC, sonic);
318. REGISTER\_ENCODER (SONIC\_LS, sonic\_ls);
319. REGISTER\_DECODER (TRUEHD, truehd);
320. REGISTER\_DECODER (TRUESPEECH, truespeech);
321. REGISTER\_DECODER (TTA, tta);
322. REGISTER\_DECODER (TWINVQ, twinvq);
323. REGISTER\_DECODER (VMDAUDIO, vmdaudio);
324. REGISTER\_ENCDEC  (VORBIS, vorbis);
325. REGISTER\_DECODER (WAVPACK, wavpack);
326. REGISTER\_DECODER (WMALOSSLESS, wmalossless);
327. REGISTER\_DECODER (WMAPRO, wmapro);
328. REGISTER\_ENCDEC  (WMAV1, wmav1);
329. REGISTER\_ENCDEC  (WMAV2, wmav2);
330. REGISTER\_DECODER (WMAVOICE, wmavoice);
331. REGISTER\_DECODER (WS\_SND1, ws\_snd1);
333. /\* PCM codecs \*/
334. REGISTER\_ENCDEC  (PCM\_ALAW, pcm\_alaw);
335. REGISTER\_DECODER (PCM\_BLURAY, pcm\_bluray);
336. REGISTER\_DECODER (PCM\_DVD, pcm\_dvd);
337. REGISTER\_ENCDEC  (PCM\_F32BE, pcm\_f32be);
338. REGISTER\_ENCDEC  (PCM\_F32LE, pcm\_f32le);
339. REGISTER\_ENCDEC  (PCM\_F64BE, pcm\_f64be);
340. REGISTER\_ENCDEC  (PCM\_F64LE, pcm\_f64le);
341. REGISTER\_DECODER (PCM\_LXF, pcm\_lxf);
342. REGISTER\_ENCDEC  (PCM\_MULAW, pcm\_mulaw);
343. REGISTER\_ENCDEC  (PCM\_S8, pcm\_s8);
344. REGISTER\_DECODER (PCM\_S8\_PLANAR, pcm\_s8\_planar);
345. REGISTER\_ENCDEC  (PCM\_S16BE, pcm\_s16be);
346. REGISTER\_ENCDEC  (PCM\_S16LE, pcm\_s16le);
347. REGISTER\_DECODER (PCM\_S16LE\_PLANAR, pcm\_s16le\_planar);
348. REGISTER\_ENCDEC  (PCM\_S24BE, pcm\_s24be);
349. REGISTER\_ENCDEC  (PCM\_S24DAUD, pcm\_s24daud);
350. REGISTER\_ENCDEC  (PCM\_S24LE, pcm\_s24le);
351. REGISTER\_ENCDEC  (PCM\_S32BE, pcm\_s32be);
352. REGISTER\_ENCDEC  (PCM\_S32LE, pcm\_s32le);
353. REGISTER\_ENCDEC  (PCM\_U8, pcm\_u8);
354. REGISTER\_ENCDEC  (PCM\_U16BE, pcm\_u16be);
355. REGISTER\_ENCDEC  (PCM\_U16LE, pcm\_u16le);
356. REGISTER\_ENCDEC  (PCM\_U24BE, pcm\_u24be);
357. REGISTER\_ENCDEC  (PCM\_U24LE, pcm\_u24le);
358. REGISTER\_ENCDEC  (PCM\_U32BE, pcm\_u32be);
359. REGISTER\_ENCDEC  (PCM\_U32LE, pcm\_u32le);
360. REGISTER\_DECODER (PCM\_ZORK , pcm\_zork);
362. /\* DPCM codecs \*/
363. REGISTER\_DECODER (INTERPLAY\_DPCM, interplay\_dpcm);
364. REGISTER\_ENCDEC  (ROQ\_DPCM, roq\_dpcm);
365. REGISTER\_DECODER (SOL\_DPCM, sol\_dpcm);
366. REGISTER\_DECODER (XAN\_DPCM, xan\_dpcm);
368. /\* ADPCM codecs \*/
369. REGISTER\_DECODER (ADPCM\_4XM, adpcm\_4xm);
370. REGISTER\_ENCDEC  (ADPCM\_ADX, adpcm\_adx);
371. REGISTER\_DECODER (ADPCM\_CT, adpcm\_ct);
372. REGISTER\_DECODER (ADPCM\_EA, adpcm\_ea);
373. REGISTER\_DECODER (ADPCM\_EA\_MAXIS\_XA, adpcm\_ea\_maxis\_xa);
374. REGISTER\_DECODER (ADPCM\_EA\_R1, adpcm\_ea\_r1);
375. REGISTER\_DECODER (ADPCM\_EA\_R2, adpcm\_ea\_r2);
376. REGISTER\_DECODER (ADPCM\_EA\_R3, adpcm\_ea\_r3);
377. REGISTER\_DECODER (ADPCM\_EA\_XAS, adpcm\_ea\_xas);
378. REGISTER\_ENCDEC  (ADPCM\_G722, adpcm\_g722);
379. REGISTER\_ENCDEC  (ADPCM\_G726, adpcm\_g726);
380. REGISTER\_DECODER (ADPCM\_IMA\_AMV, adpcm\_ima\_amv);
381. REGISTER\_DECODER (ADPCM\_IMA\_APC, adpcm\_ima\_apc);
382. REGISTER\_DECODER (ADPCM\_IMA\_DK3, adpcm\_ima\_dk3);
383. REGISTER\_DECODER (ADPCM\_IMA\_DK4, adpcm\_ima\_dk4);
384. REGISTER\_DECODER (ADPCM\_IMA\_EA\_EACS, adpcm\_ima\_ea\_eacs);
385. REGISTER\_DECODER (ADPCM\_IMA\_EA\_SEAD, adpcm\_ima\_ea\_sead);
386. REGISTER\_DECODER (ADPCM\_IMA\_ISS, adpcm\_ima\_iss);
387. REGISTER\_ENCDEC  (ADPCM\_IMA\_QT, adpcm\_ima\_qt);
388. REGISTER\_DECODER (ADPCM\_IMA\_SMJPEG, adpcm\_ima\_smjpeg);
389. REGISTER\_ENCDEC  (ADPCM\_IMA\_WAV, adpcm\_ima\_wav);
390. REGISTER\_DECODER (ADPCM\_IMA\_WS, adpcm\_ima\_ws);
391. REGISTER\_ENCDEC  (ADPCM\_MS, adpcm\_ms);
392. REGISTER\_DECODER (ADPCM\_SBPRO\_2, adpcm\_sbpro\_2);
393. REGISTER\_DECODER (ADPCM\_SBPRO\_3, adpcm\_sbpro\_3);
394. REGISTER\_DECODER (ADPCM\_SBPRO\_4, adpcm\_sbpro\_4);
395. REGISTER\_ENCDEC  (ADPCM\_SWF, adpcm\_swf);
396. REGISTER\_DECODER (ADPCM\_THP, adpcm\_thp);
397. REGISTER\_DECODER (ADPCM\_XA, adpcm\_xa);
398. REGISTER\_ENCDEC  (ADPCM\_YAMAHA, adpcm\_yamaha);
400. /\* subtitles \*/
401. REGISTER\_ENCDEC  (ASS, ass);
402. REGISTER\_ENCDEC  (DVBSUB, dvbsub);
403. REGISTER\_ENCDEC  (DVDSUB, dvdsub);
404. REGISTER\_DECODER (PGSSUB, pgssub);
405. REGISTER\_ENCDEC  (SRT, srt);
406. REGISTER\_ENCDEC  (XSUB, xsub);
408. /\* external libraries \*/
409. REGISTER\_ENCODER (LIBAACPLUS, libaacplus);
410. REGISTER\_DECODER (LIBCELT, libcelt);
411. REGISTER\_ENCDEC  (LIBDIRAC, libdirac);
412. REGISTER\_ENCODER (LIBFAAC, libfaac);
413. REGISTER\_ENCDEC  (LIBGSM, libgsm);
414. REGISTER\_ENCDEC  (LIBGSM\_MS, libgsm\_ms);
415. REGISTER\_ENCODER (LIBMP3LAME, libmp3lame);
416. REGISTER\_ENCDEC  (LIBOPENCORE\_AMRNB, libopencore\_amrnb);
417. REGISTER\_DECODER (LIBOPENCORE\_AMRWB, libopencore\_amrwb);
418. REGISTER\_ENCDEC (LIBOPENJPEG, libopenjpeg);
419. REGISTER\_ENCDEC  (LIBSCHROEDINGER, libschroedinger);
420. REGISTER\_ENCDEC  (LIBSPEEX, libspeex);
421. REGISTER\_DECODER (LIBSTAGEFRIGHT\_H264, libstagefright\_h264);
422. REGISTER\_ENCODER (LIBTHEORA, libtheora);
423. REGISTER\_DECODER (LIBUTVIDEO, libutvideo);
424. REGISTER\_ENCODER (LIBVO\_AACENC, libvo\_aacenc);
425. REGISTER\_ENCODER (LIBVO\_AMRWBENC, libvo\_amrwbenc);
426. REGISTER\_ENCODER (LIBVORBIS, libvorbis);
427. REGISTER\_ENCDEC  (LIBVPX, libvpx);
428. REGISTER\_ENCODER (LIBX264, libx264);
429. REGISTER\_ENCODER (LIBX264RGB, libx264rgb);
430. REGISTER\_ENCODER (LIBXAVS, libxavs);
431. REGISTER\_ENCODER (LIBXVID, libxvid);
433. /\* text \*/
434. REGISTER\_DECODER (BINTEXT, bintext);
435. REGISTER\_DECODER  (XBIN, xbin);
436. REGISTER\_DECODER  (IDF, idf);
438. /\* parsers \*/
439. REGISTER\_PARSER  (AAC, aac);
440. REGISTER\_PARSER  (AAC\_LATM, aac\_latm);
441. REGISTER\_PARSER  (AC3, ac3);
442. REGISTER\_PARSER  (ADX, adx);
443. REGISTER\_PARSER  (CAVSVIDEO, cavsvideo);
444. REGISTER\_PARSER  (DCA, dca);
445. REGISTER\_PARSER  (DIRAC, dirac);
446. REGISTER\_PARSER  (DNXHD, dnxhd);
447. REGISTER\_PARSER  (DVBSUB, dvbsub);
448. REGISTER\_PARSER  (DVDSUB, dvdsub);
449. REGISTER\_PARSER  (FLAC, flac);
450. REGISTER\_PARSER  (GSM, gsm);
451. REGISTER\_PARSER  (H261, h261);
452. REGISTER\_PARSER  (H263, h263);
453. REGISTER\_PARSER  (H264, h264);
454. REGISTER\_PARSER  (MJPEG, mjpeg);
455. REGISTER\_PARSER  (MLP, mlp);
456. REGISTER\_PARSER  (MPEG4VIDEO, mpeg4video);
457. REGISTER\_PARSER  (MPEGAUDIO, mpegaudio);
458. REGISTER\_PARSER  (MPEGVIDEO, mpegvideo);
459. REGISTER\_PARSER  (PNM, pnm);
460. REGISTER\_PARSER  (RV30, rv30);
461. REGISTER\_PARSER  (RV40, rv40);
462. REGISTER\_PARSER  (VC1, vc1);
463. REGISTER\_PARSER  (VP3, vp3);
464. REGISTER\_PARSER  (VP8, vp8);
466. /\* bitstream filters \*/
467. REGISTER\_BSF     (AAC\_ADTSTOASC, aac\_adtstoasc);
468. REGISTER\_BSF     (CHOMP, chomp);
469. REGISTER\_BSF     (DUMP\_EXTRADATA, dump\_extradata);
470. REGISTER\_BSF     (H264\_MP4TOANNEXB, h264\_mp4toannexb);
471. REGISTER\_BSF     (IMX\_DUMP\_HEADER, imx\_dump\_header);
472. REGISTER\_BSF     (MJPEG2JPEG, mjpeg2jpeg);
473. REGISTER\_BSF     (MJPEGA\_DUMP\_HEADER, mjpega\_dump\_header);
474. REGISTER\_BSF     (MP3\_HEADER\_COMPRESS, mp3\_header\_compress);
475. REGISTER\_BSF     (MP3\_HEADER\_DECOMPRESS, mp3\_header\_decompress);
476. REGISTER\_BSF     (MOV2TEXTSUB, mov2textsub);
477. REGISTER\_BSF     (NOISE, noise);
478. REGISTER\_BSF     (REMOVE\_EXTRADATA, remove\_extradata);
479. REGISTER\_BSF     (TEXT2MOVSUB, text2movsub);
480. }

整个代码的过程就是首先确定是不是已经初始化过了（initialized），如果没有，就注册，注册，注册...直到完成所有注册。

函数的调用关系图如下图所示。av\_register\_all()调用了avcodec\_register\_all()。因此如果调用过av\_register\_all()的话就不需要再调用avcodec\_register\_all()了。



下面附上硬件加速器，编码器/解码器，parser，Bitstream Filter的注册代码。

硬件加速器注册函数是av\_register\_hwaccel()。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/12677265) [copy](http://blog.csdn.net/leixiaohua1020/article/details/12677265)

[在CODE上查看代码片](https://code.csdn.net/snippets/513247)

1. **void** av\_register\_hwaccel(AVHWAccel \*hwaccel)
2. {
3. AVHWAccel \*\*p = last\_hwaccel;
4. hwaccel->next = NULL;
5. **while**(\*p || avpriv\_atomic\_ptr\_cas((**void** \* **volatile** \*)p, NULL, hwaccel))
6. p = &(\*p)->next;
7. last\_hwaccel = &hwaccel->next;
8. }

编解码器注册函数是avcodec\_register()。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/12677265) [copy](http://blog.csdn.net/leixiaohua1020/article/details/12677265)

[在CODE上查看代码片](https://code.csdn.net/snippets/513247)

1. av\_cold **void** avcodec\_register(AVCodec \*codec)
2. {
3. AVCodec \*\*p;
4. avcodec\_init();
5. p = last\_avcodec;
6. codec->next = NULL;
8. **while**(\*p || avpriv\_atomic\_ptr\_cas((**void** \* **volatile** \*)p, NULL, codec))
9. p = &(\*p)->next;
10. last\_avcodec = &codec->next;
12. **if** (codec->init\_static\_data)
13. codec->init\_static\_data(codec);
14. }

parser注册函数是av\_register\_codec\_parser()。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/12677265) [copy](http://blog.csdn.net/leixiaohua1020/article/details/12677265)

[在CODE上查看代码片](https://code.csdn.net/snippets/513247)

1. **void** av\_register\_codec\_parser(AVCodecParser \*parser)
2. {
3. **do** {
4. parser->next = av\_first\_parser;
5. } **while** (parser->next != avpriv\_atomic\_ptr\_cas((**void** \* **volatile** \*)&av\_first\_parser, parser->next, parser));
6. }

Bitstream Filter注册函数是av\_register\_bitstream\_filter()。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/12677265) [copy](http://blog.csdn.net/leixiaohua1020/article/details/12677265)

[在CODE上查看代码片](https://code.csdn.net/snippets/513247)

1. **void** av\_register\_bitstream\_filter(AVBitStreamFilter \*bsf)
2. {
3. **do** {
4. bsf->next = first\_bitstream\_filter;
5. } **while**(bsf->next != avpriv\_atomic\_ptr\_cas((**void** \* **volatile** \*)&first\_bitstream\_filter, bsf->next, bsf));
6. }

后两个函数中的avpriv\_atomic\_ptr\_cas()定义如下。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/12677265) [copy](http://blog.csdn.net/leixiaohua1020/article/details/12677265)

[在CODE上查看代码片](https://code.csdn.net/snippets/513247)

1. **void** \*avpriv\_atomic\_ptr\_cas(**void** \* **volatile** \*ptr, **void** \*oldval, **void** \*newval)
2. {
3. **if** (\*ptr == oldval) {
4. \*ptr = newval;
5. **return** oldval;
6. }
7. **return** \*ptr;
8. }

# [FFmpeg源代码简单分析：内存的分配和释放（av\_malloc()、av\_free()等）](http://blog.csdn.net/leixiaohua1020/article/details/41176777)

本文简单记录一下FFmpeg中内存操作的函数。

内存操作的常见函数位于libavutil\mem.c中。本文记录FFmpeg开发中最常使用的几个函数：av\_malloc()，av\_realloc()，av\_mallocz()，av\_calloc()，av\_free()，av\_freep()。

### av\_malloc()

av\_malloc()是FFmpeg中最常见的内存分配函数。它的定义如下。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/41176777) [copy](http://blog.csdn.net/leixiaohua1020/article/details/41176777)

1. #define FF\_MEMORY\_POISON 0x2a
3. #define ALIGN (HAVE\_AVX ? 32 : 16)
5. **static** **size\_t** max\_alloc\_size= INT\_MAX;
7. **void** \*av\_malloc(**size\_t** size)
8. {
9. **void** \*ptr = NULL;
10. #if CONFIG\_MEMALIGN\_HACK
11. **long** diff;
12. #endif

15. /\* let's disallow possibly ambiguous cases \*/
16. **if** (size > (max\_alloc\_size - 32))
17. **return** NULL;

20. #if CONFIG\_MEMALIGN\_HACK
21. ptr = malloc(size + ALIGN);
22. **if** (!ptr)
23. **return** ptr;
24. diff              = ((~(**long**)ptr)&(ALIGN - 1)) + 1;
25. ptr               = (**char** \*)ptr + diff;
26. ((**char** \*)ptr)[-1] = diff;
27. #elif HAVE\_POSIX\_MEMALIGN
28. **if** (size) //OS X on SDK 10.6 has a broken posix\_memalign implementation
29. **if** (posix\_memalign(&ptr, ALIGN, size))
30. ptr = NULL;
31. #elif HAVE\_ALIGNED\_MALLOC
32. ptr = \_aligned\_malloc(size, ALIGN);
33. #elif HAVE\_MEMALIGN
34. #ifndef \_\_DJGPP\_\_
35. ptr = memalign(ALIGN, size);
36. #else
37. ptr = memalign(size, ALIGN);
38. #endif
39. /\* Why 64?
40. \* Indeed, we should align it:
41. \*   on  4 for 386
42. \*   on 16 for 486
43. \*   on 32 for 586, PPro - K6-III
44. \*   on 64 for K7 (maybe for P3 too).
45. \* Because L1 and L2 caches are aligned on those values.
46. \* But I don't want to code such logic here!
47. \*/
48. /\* Why 32?
49. \* For AVX ASM. SSE / NEON needs only 16.
50. \* Why not larger? Because I did not see a difference in benchmarks ...
51. \*/
52. /\* benchmarks with P3
53. \* memalign(64) + 1          3071, 3051, 3032
54. \* memalign(64) + 2          3051, 3032, 3041
55. \* memalign(64) + 4          2911, 2896, 2915
56. \* memalign(64) + 8          2545, 2554, 2550
57. \* memalign(64) + 16         2543, 2572, 2563
58. \* memalign(64) + 32         2546, 2545, 2571
59. \* memalign(64) + 64         2570, 2533, 2558
60. \*
61. \* BTW, malloc seems to do 8-byte alignment by default here.
62. \*/
63. #else
64. ptr = malloc(size);
65. #endif
66. **if**(!ptr && !size) {
67. size = 1;
68. ptr= av\_malloc(1);
69. }
70. #if CONFIG\_MEMORY\_POISONING
71. **if** (ptr)
72. memset(ptr, FF\_MEMORY\_POISON, size);
73. #endif
74. **return** ptr;
75. }

如果不考虑上述代码中的一大堆宏定义（即类似CONFIG\_MEMALIGN\_HACK这类的宏都采用默认值0），av\_malloc()的代码可以简化成如下形式。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/41176777) [copy](http://blog.csdn.net/leixiaohua1020/article/details/41176777)

[在CODE上查看代码片](https://code.csdn.net/snippets/611113)

1. **void** \*av\_malloc(**size\_t** size)
2. {
3. **void** \*ptr = NULL;
4. /\* let's disallow possibly ambiguous cases \*/
5. **if** (size > (max\_alloc\_size - 32))
6. **return** NULL;
7. ptr = malloc(size);
8. **if**(!ptr && !size) {
9. size = 1;
10. ptr= av\_malloc(1);
11. }
12. **return** ptr;
13. }

可以看出，此时的av\_malloc()就是简单的封装了系统函数malloc()，并做了一些错误检查工作。

#### 关于size\_t

size \_t 这个类型在FFmpeg中多次出现，简单解释一下其作用。size \_t是为了增强程序的可移植性而定义的。不同系统上，定义size\_t可能不一样。它实际上就是unsigned int。

#### 为什么要内存对齐？

FFmpeg内存分配方面多次涉及到“内存对齐”（memory alignment）的概念。  
这方面内容在IBM的网站上有一篇文章，讲的挺通俗易懂的，在此简单转述一下。

程序员通常认为内存就是一个字节数组，每次可以一个一个字节存取内存。例如在[**C语言**](http://lib.csdn.net/base/c)中使用char \*指代“一块内存”，[**Java**](http://lib.csdn.net/base/javase)中使用byte[]指代一块内存。如下所示。

http://img.blog.csdn.net/20141116213746581

但那实际上计算机处理器却不是这样认为的。处理器相对比较“懒惰”，它会以2字节，4字节，8字节，16字节甚至32字节来存取内存。例如下图显示了以4字节为单位读写内存的处理器“看待”上述内存的方式。

http://img.blog.csdn.net/20141116213757782

上述的存取单位的大小称之为内存存取粒度。  
下面看一个实例，分别从地址0，和地址1读取4个字节到寄存器。

从程序员的角度来看，读取方式如下图所示。

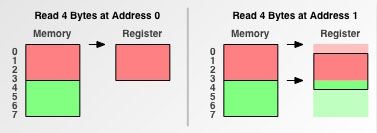


而2字节存取粒度的处理器的读取方式如下图所示。



可以看出2字节存取粒度的处理器从地址0读取4个字节一共读取2次；从地址1读取4个字节一共读取了3次。由于每次读取的开销是固定的，因此从地址1读取4字节的效率有所下降。

4字节存取粒度的处理器的读取方式如下图所示。



可以看出4字节存取粒度的处理器从地址0读取4个字节一共读取1次；从地址1读取4个字节一共读取了2次。从地址1读取的开销比从地址0读取多了一倍。由此可见内存不对齐对CPU的性能是有影响的。

### av\_realloc()

av\_realloc()用于对申请的内存的大小进行调整。它的定义如下。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/41176777) [copy](http://blog.csdn.net/leixiaohua1020/article/details/41176777)

[在CODE上查看代码片](https://code.csdn.net/snippets/611113)

1. **void** \*av\_realloc(**void** \*ptr, **size\_t** size)
2. {
3. #if CONFIG\_MEMALIGN\_HACK
4. **int** diff;
5. #endif

8. /\* let's disallow possibly ambiguous cases \*/
9. **if** (size > (max\_alloc\_size - 32))
10. **return** NULL;

13. #if CONFIG\_MEMALIGN\_HACK
14. //FIXME this isn't aligned correctly, though it probably isn't needed
15. **if** (!ptr)
16. **return** av\_malloc(size);
17. diff = ((**char** \*)ptr)[-1];
18. av\_assert0(diff>0 && diff<=ALIGN);
19. ptr = realloc((**char** \*)ptr - diff, size + diff);
20. **if** (ptr)
21. ptr = (**char** \*)ptr + diff;
22. **return** ptr;
23. #elif HAVE\_ALIGNED\_MALLOC
24. **return** \_aligned\_realloc(ptr, size + !size, ALIGN);
25. #else
26. **return** realloc(ptr, size + !size);
27. #endif
28. }

默认情况下（CONFIG\_MEMALIGN\_HACK这些宏使用默认值0）的代码：

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/41176777) [copy](http://blog.csdn.net/leixiaohua1020/article/details/41176777)

[在CODE上查看代码片](https://code.csdn.net/snippets/611113)

1. **void** \*av\_realloc(**void** \*ptr, **size\_t** size)
2. {
3. /\* let's disallow possibly ambiguous cases \*/
4. **if** (size > (max\_alloc\_size - 32))
5. **return** NULL;
6. **return** realloc(ptr, size + !size);
7. }

可以看出av\_realloc()简单封装了系统的realloc()函数。

### av\_mallocz()

av\_mallocz()可以理解为av\_malloc()+zeromemory。代码如下。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/41176777) [copy](http://blog.csdn.net/leixiaohua1020/article/details/41176777)

[在CODE上查看代码片](https://code.csdn.net/snippets/611113)

1. **void** \*av\_mallocz(**size\_t** size)
2. {
3. **void** \*ptr = av\_malloc(size);
4. **if** (ptr)
5. memset(ptr, 0, size);
6. **return** ptr;
7. }

从源代码可以看出av\_mallocz()中调用了av\_malloc()之后，又调用memset()将分配的内存设置为0。

### av\_calloc()

av\_calloc()则是简单封装了av\_mallocz()，定义如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/41176777) [copy](http://blog.csdn.net/leixiaohua1020/article/details/41176777)

[在CODE上查看代码片](https://code.csdn.net/snippets/611113)

1. **void** \*av\_calloc(**size\_t** nmemb, **size\_t** size)
2. {
3. **if** (size <= 0 || nmemb >= INT\_MAX / size)
4. **return** NULL;
5. **return** av\_mallocz(nmemb \* size);
6. }

从代码中可以看出，它调用av\_mallocz()分配了nmemb\*size个字节的内存。

### av\_free()

av\_free()用于释放申请的内存。它的定义如下。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/41176777) [copy](http://blog.csdn.net/leixiaohua1020/article/details/41176777)

[在CODE上查看代码片](https://code.csdn.net/snippets/611113)

1. **void** av\_free(**void** \*ptr)
2. {
3. #if CONFIG\_MEMALIGN\_HACK
4. **if** (ptr) {
5. **int** v= ((**char** \*)ptr)[-1];
6. av\_assert0(v>0 && v<=ALIGN);
7. free((**char** \*)ptr - v);
8. }
9. #elif HAVE\_ALIGNED\_MALLOC
10. \_aligned\_free(ptr);
11. #else
12. free(ptr);
13. #endif
14. }

默认情况下（CONFIG\_MEMALIGN\_HACK这些宏使用默认值0）的代码：

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/41176777) [copy](http://blog.csdn.net/leixiaohua1020/article/details/41176777)

[在CODE上查看代码片](https://code.csdn.net/snippets/611113)

1. **void** av\_free(**void** \*ptr)
2. {
3. free(ptr);
4. }

可以看出av\_free()简单的封装了free()。

### av\_freep()

av\_freep()简单封装了av\_free()。并且在释放内存之后将目标指针设置为NULL。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/41176777) [copy](http://blog.csdn.net/leixiaohua1020/article/details/41176777)

[在CODE上查看代码片](https://code.csdn.net/snippets/611113)

1. **void** av\_freep(**void** \*arg)
2. {
3. **void** \*\*ptr = (**void** \*\*)arg;
4. av\_free(\*ptr);
5. \*ptr = NULL;
6. }

# [FFmpeg源代码简单分析：常见结构体的初始化和销毁（AVFormatContext，AVFrame等）](http://blog.csdn.net/leixiaohua1020/article/details/41181155)

本文简单分析FFmpeg常见结构体的初始化和销毁函数的源代码。常见的结构体在文章：

《[FFMPEG中最关键的结构体之间的关系](http://blog.csdn.net/leixiaohua1020/article/details/11693997)》中已经有过叙述，包括：

AVFormatContext：统领全局的基本结构体。主要用于处理封装格式（FLV/MKV/RMVB等）。

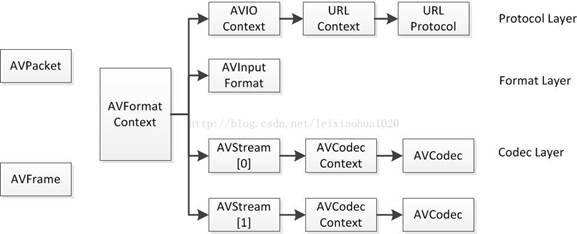
AVIOContext：输入输出对应的结构体，用于输入输出（读写文件，RTMP协议等）。

AVStream，AVCodecContext：视音频流对应的结构体，用于视音频编解码。

AVFrame：存储非压缩的数据（视频对应RGB/YUV像素数据，音频对应PCM采样数据）

AVPacket：存储压缩数据（视频对应H.264等码流数据，音频对应AAC/MP3等码流数据）

他们之间的关系如下图所示（详细信息可以参考上文提到的文章）。



下文简单分析一下上述几个结构体的初始化和销毁函数。这些函数列表如下。

|  |  |  |
| --- | --- | --- |
| 结构体 | 初始化 | 销毁 |
| AVFormatContext | avformat\_alloc\_context() | avformat\_free\_context() |
| AVIOContext | avio\_alloc\_context() |  |
| AVStream | avformat\_new\_stream() |  |
| AVCodecContext | avcodec\_alloc\_context3() |  |
| AVFrame | av\_frame\_alloc();  av\_image\_fill\_arrays() | av\_frame\_free() |
| AVPacket | av\_init\_packet();  av\_new\_packet() | av\_free\_packet() |

下面进入正文。

## AVFormatContext

AVFormatContext的初始化函数是avformat\_alloc\_context()，销毁函数是avformat\_free\_context()。

### avformat\_alloc\_context()

avformat\_alloc\_context()的声明位于libavformat\avformat.h，如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/41181155) [copy](http://blog.csdn.net/leixiaohua1020/article/details/41181155)

1. /\*\*
2. \* Allocate an AVFormatContext.
3. \* avformat\_free\_context() can be used to freethe context and everything
4. \* allocated by the framework within it.
5. \*/
6. AVFormatContext\*avformat\_alloc\_context(**void**);

avformat\_alloc\_context()的定义位于libavformat\options.c。代码如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/41181155) [copy](http://blog.csdn.net/leixiaohua1020/article/details/41181155)

[在CODE上查看代码片](https://code.csdn.net/snippets/611187)

1. AVFormatContext \*avformat\_alloc\_context(**void**)
2. {
3. AVFormatContext \*ic;
4. ic = av\_malloc(**sizeof**(AVFormatContext));
5. **if** (!ic) **return** ic;
6. avformat\_get\_context\_defaults(ic);

9. ic->internal = av\_mallocz(**sizeof**(\*ic->internal));
10. **if** (!ic->internal) {
11. avformat\_free\_context(ic);
12. **return** NULL;
13. }

16. **return** ic;
17. }

从代码中可以看出，avformat\_alloc\_context()调用av\_malloc()为AVFormatContext结构体分配了内存，而且同时也给AVFormatContext中的internal字段分配内存（这个字段是FFmpeg内部使用的，先不分析）。此外调用了一个avformat\_get\_context\_defaults()函数。该函数用于设置AVFormatContext的字段的默认值。它的定义也位于libavformat\options.c，确切的说就位于avformat\_alloc\_context()上面。我们看一下该函数的定义。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/41181155) [copy](http://blog.csdn.net/leixiaohua1020/article/details/41181155)

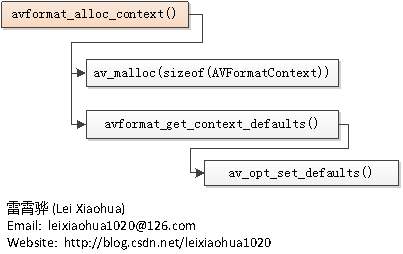
[在CODE上查看代码片](https://code.csdn.net/snippets/611187)

1. **static** **void** avformat\_get\_context\_defaults(AVFormatContext \*s)
2. {
3. memset(s, 0, **sizeof**(AVFormatContext));

6. s->av\_class = &av\_format\_context\_class;

9. av\_opt\_set\_defaults(s);
10. }

从代码可以看出，avformat\_get\_context\_defaults()首先调用memset()将AVFormatContext的所有字段置0。而后调用了一个函数av\_opt\_set\_defaults()。av\_opt\_set\_defaults()用于给字段设置默认值。  
avformat\_alloc\_context()代码的函数调用关系如下图所示。



### avformat\_free\_context()

avformat\_free\_context()的声明位于libavformat\avformat.h，如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/41181155) [copy](http://blog.csdn.net/leixiaohua1020/article/details/41181155)

[在CODE上查看代码片](https://code.csdn.net/snippets/611187)

1. /\*\*
2. \* Free an AVFormatContext and all its streams.
3. \* @param s context to free
4. \*/
5. **void** avformat\_free\_context(AVFormatContext \*s);

avformat\_free\_context()的定义位于libavformat\options.c。代码如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/41181155) [copy](http://blog.csdn.net/leixiaohua1020/article/details/41181155)

[在CODE上查看代码片](https://code.csdn.net/snippets/611187)

1. **void** avformat\_free\_context(AVFormatContext \*s)
2. {
3. **int** i;

6. **if** (!s)
7. **return**;

10. av\_opt\_free(s);
11. **if** (s->iformat && s->iformat->priv\_class && s->priv\_data)
12. av\_opt\_free(s->priv\_data);
13. **if** (s->oformat && s->oformat->priv\_class && s->priv\_data)
14. av\_opt\_free(s->priv\_data);

17. **for** (i = s->nb\_streams - 1; i >= 0; i--) {
18. ff\_free\_stream(s, s->streams[i]);
19. }
20. **for** (i = s->nb\_programs - 1; i >= 0; i--) {
21. av\_dict\_free(&s->programs[i]->metadata);
22. av\_freep(&s->programs[i]->stream\_index);
23. av\_freep(&s->programs[i]);
24. }
25. av\_freep(&s->programs);
26. av\_freep(&s->priv\_data);
27. **while** (s->nb\_chapters--) {
28. av\_dict\_free(&s->chapters[s->nb\_chapters]->metadata);
29. av\_freep(&s->chapters[s->nb\_chapters]);
30. }
31. av\_freep(&s->chapters);
32. av\_dict\_free(&s->metadata);
33. av\_freep(&s->streams);
34. av\_freep(&s->internal);
35. flush\_packet\_queue(s);
36. av\_free(s);
37. }

从代码中可以看出，avformat\_free\_context()调用了各式各样的销毁函数：av\_opt\_free()，av\_freep()，av\_dict\_free()。这些函数分别用于释放不同种类的变量，在这里不再详细讨论。在这里看一个释放AVStream的函数ff\_free\_stream()。该函数的定义位于libavformat\options.c（其实就在avformat\_free\_context()上方）。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/41181155) [copy](http://blog.csdn.net/leixiaohua1020/article/details/41181155)

[在CODE上查看代码片](https://code.csdn.net/snippets/611187)

1. **void** ff\_free\_stream(AVFormatContext \*s, AVStream \*st) {
2. **int** j;
3. av\_assert0(s->nb\_streams>0);
4. av\_assert0(s->streams[ s->nb\_streams - 1 ] == st);

7. **for** (j = 0; j < st->nb\_side\_data; j++)
8. av\_freep(&st->side\_data[j].data);
9. av\_freep(&st->side\_data);
10. st->nb\_side\_data = 0;

13. **if** (st->parser) {
14. av\_parser\_close(st->parser);
15. }
16. **if** (st->attached\_pic.data)
17. av\_free\_packet(&st->attached\_pic);
18. av\_dict\_free(&st->metadata);
19. av\_freep(&st->probe\_data.buf);
20. av\_freep(&st->index\_entries);
21. av\_freep(&st->codec->extradata);
22. av\_freep(&st->codec->subtitle\_header);
23. av\_freep(&st->codec);
24. av\_freep(&st->priv\_data);
25. **if** (st->info)
26. av\_freep(&st->info->duration\_error);
27. av\_freep(&st->info);
28. av\_freep(&s->streams[ --s->nb\_streams ]);
29. }

从代码中可以看出，与释放AVFormatContext类似，释放AVStream的时候，也是调用了av\_freep()，av\_dict\_free()这些函数释放有关的字段。如果使用了parser的话，会调用av\_parser\_close()关闭该parser。

## AVIOContext

### avio\_alloc\_context()

AVIOContext的初始化函数是avio\_alloc\_context()，销毁的时候使用av\_free()释放掉其中的缓存即可。它的声明位于libavformat\avio.h中，如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/41181155) [copy](http://blog.csdn.net/leixiaohua1020/article/details/41181155)

[在CODE上查看代码片](https://code.csdn.net/snippets/611187)

1. /\*\*
2. \* Allocate and initialize an AVIOContext for buffered I/O. It must be later
3. \* freed with av\_free().
4. \*
5. \* @param buffer Memory block for input/output operations via AVIOContext.
6. \*        The buffer must be allocated with av\_malloc() and friends.
7. \* @param buffer\_size The buffer size is very important for performance.
8. \*        For protocols with fixed blocksize it should be set to this blocksize.
9. \*        For others a typical size is a cache page, e.g. 4kb.
10. \* @param write\_flag Set to 1 if the buffer should be writable, 0 otherwise.
11. \* @param opaque An opaque pointer to user-specific data.
12. \* @param read\_packet  A function for refilling the buffer, may be NULL.
13. \* @param write\_packet A function for writing the buffer contents, may be NULL.
14. \*        The function may not change the input buffers content.
15. \* @param seek A function for seeking to specified byte position, may be NULL.
16. \*
17. \* @return Allocated AVIOContext or NULL on failure.
18. \*/
19. AVIOContext \*avio\_alloc\_context(
20. unsigned **char** \*buffer,
21. **int** buffer\_size,
22. **int** write\_flag,
23. **void** \*opaque,
24. **int** (\*read\_packet)(**void** \*opaque, uint8\_t \*buf, **int** buf\_size),
25. **int** (\*write\_packet)(**void** \*opaque, uint8\_t \*buf, **int** buf\_size),
26. int64\_t (\*seek)(**void** \*opaque, int64\_t offset, **int** whence));

avio\_alloc\_context()定义位于libavformat\aviobuf.c中，如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/41181155) [copy](http://blog.csdn.net/leixiaohua1020/article/details/41181155)

[在CODE上查看代码片](https://code.csdn.net/snippets/611187)

1. AVIOContext \*avio\_alloc\_context(
2. unsigned **char** \*buffer,
3. **int** buffer\_size,
4. **int** write\_flag,
5. **void** \*opaque,
6. **int** (\*read\_packet)(**void** \*opaque, uint8\_t \*buf, **int** buf\_size),
7. **int** (\*write\_packet)(**void** \*opaque, uint8\_t \*buf, **int** buf\_size),
8. int64\_t (\*seek)(**void** \*opaque, int64\_t offset, **int** whence))
9. {
10. AVIOContext \*s = av\_mallocz(**sizeof**(AVIOContext));
11. **if** (!s)
12. **return** NULL;
13. ffio\_init\_context(s, buffer, buffer\_size, write\_flag, opaque,
14. read\_packet, write\_packet, seek);
15. **return** s;
16. }

从代码中可以看出，avio\_alloc\_context()首先调用av\_mallocz()为AVIOContext分配内存。而后调用了一个函数ffio\_init\_context()。该函数完成了真正的初始化工作。我们看一下ffio\_init\_context()函数的定义。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/41181155) [copy](http://blog.csdn.net/leixiaohua1020/article/details/41181155)

[在CODE上查看代码片](https://code.csdn.net/snippets/611187)

1. **int** ffio\_init\_context(AVIOContext \*s,
2. unsigned **char** \*buffer,
3. **int** buffer\_size,
4. **int** write\_flag,
5. **void** \*opaque,
6. **int** (\*read\_packet)(**void** \*opaque, uint8\_t \*buf, **int** buf\_size),
7. **int** (\*write\_packet)(**void** \*opaque, uint8\_t \*buf, **int** buf\_size),
8. int64\_t (\*seek)(**void** \*opaque, int64\_t offset, **int** whence))
9. {
10. s->buffer      = buffer;
11. s->orig\_buffer\_size =
12. s->buffer\_size = buffer\_size;
13. s->buf\_ptr     = buffer;
14. s->opaque      = opaque;
15. s->direct      = 0;

18. url\_resetbuf(s, write\_flag ? AVIO\_FLAG\_WRITE : AVIO\_FLAG\_READ);

21. s->write\_packet    = write\_packet;
22. s->read\_packet     = read\_packet;
23. s->seek            = seek;
24. s->pos             = 0;
25. s->must\_flush      = 0;
26. s->eof\_reached     = 0;
27. s->error           = 0;
28. s->seekable        = seek ? AVIO\_SEEKABLE\_NORMAL : 0;
29. s->max\_packet\_size = 0;
30. s->update\_checksum = NULL;

33. **if** (!read\_packet && !write\_flag) {
34. s->pos     = buffer\_size;
35. s->buf\_end = s->buffer + buffer\_size;
36. }
37. s->read\_pause = NULL;
38. s->read\_seek  = NULL;

41. **return** 0;
42. }

从函数的代码可以看出，ffio\_init\_context()对AVIOContext中的缓存，函数指针等等进行了赋值。

## AVStream，AVCodecContext

AVStream的初始化函数是avformat\_new\_stream()，销毁函数使用销毁AVFormatContext的avformat\_free\_context()就可以了。

### avformat\_new\_stream()

avformat\_new\_stream()的声明位于libavformat\avformat.h中，如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/41181155) [copy](http://blog.csdn.net/leixiaohua1020/article/details/41181155)

[在CODE上查看代码片](https://code.csdn.net/snippets/611187)

1. /\*\*
2. \* Add a new stream to a media file.
3. \*
4. \* When demuxing, it is called by the demuxer in read\_header(). If the
5. \* flag AVFMTCTX\_NOHEADER is set in s.ctx\_flags, then it may also
6. \* be called in read\_packet().
7. \*
8. \* When muxing, should be called by the user before avformat\_write\_header().
9. \*
10. \* User is required to call avcodec\_close() and avformat\_free\_context() to
11. \* clean up the allocation by avformat\_new\_stream().
12. \*
13. \* @param s media file handle
14. \* @param c If non-NULL, the AVCodecContext corresponding to the new stream
15. \* will be initialized to use this codec. This is needed for e.g. codec-specific
16. \* defaults to be set, so codec should be provided if it is known.
17. \*
18. \* @return newly created stream or NULL on error.
19. \*/
20. AVStream \*avformat\_new\_stream(AVFormatContext \*s, **const** AVCodec \*c);

avformat\_new\_stream()的定义位于libavformat\utils.c中，如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/41181155) [copy](http://blog.csdn.net/leixiaohua1020/article/details/41181155)

[在CODE上查看代码片](https://code.csdn.net/snippets/611187)

1. AVStream \*avformat\_new\_stream(AVFormatContext \*s, **const** AVCodec \*c)
2. {
3. AVStream \*st;
4. **int** i;
5. AVStream \*\*streams;

8. **if** (s->nb\_streams >= INT\_MAX/**sizeof**(\*streams))
9. **return** NULL;
10. streams = av\_realloc\_array(s->streams, s->nb\_streams + 1, **sizeof**(\*streams));
11. **if** (!streams)
12. **return** NULL;
13. s->streams = streams;

16. st = av\_mallocz(**sizeof**(AVStream));
17. **if** (!st)
18. **return** NULL;
19. **if** (!(st->info = av\_mallocz(**sizeof**(\*st->info)))) {
20. av\_free(st);
21. **return** NULL;
22. }
23. st->info->last\_dts = AV\_NOPTS\_VALUE;

26. st->codec = avcodec\_alloc\_context3(c);
27. **if** (s->iformat) {
28. /\* no default bitrate if decoding \*/
29. st->codec->bit\_rate = 0;

32. /\* default pts setting is MPEG-like \*/
33. avpriv\_set\_pts\_info(st, 33, 1, 90000);
34. }

37. st->index      = s->nb\_streams;
38. st->start\_time = AV\_NOPTS\_VALUE;
39. st->duration   = AV\_NOPTS\_VALUE;
40. /\* we set the current DTS to 0 so that formats without any timestamps
41. \* but durations get some timestamps, formats with some unknown
42. \* timestamps have their first few packets buffered and the
43. \* timestamps corrected before they are returned to the user \*/
44. st->cur\_dts       = s->iformat ? RELATIVE\_TS\_BASE : 0;
45. st->first\_dts     = AV\_NOPTS\_VALUE;
46. st->probe\_packets = MAX\_PROBE\_PACKETS;
47. st->pts\_wrap\_reference = AV\_NOPTS\_VALUE;
48. st->pts\_wrap\_behavior = AV\_PTS\_WRAP\_IGNORE;

51. st->last\_IP\_pts = AV\_NOPTS\_VALUE;
52. st->last\_dts\_for\_order\_check = AV\_NOPTS\_VALUE;
53. **for** (i = 0; i < MAX\_REORDER\_DELAY + 1; i++)
54. st->pts\_buffer[i] = AV\_NOPTS\_VALUE;

57. st->sample\_aspect\_ratio = (AVRational) { 0, 1 };

60. #if FF\_API\_R\_FRAME\_RATE
61. st->info->last\_dts      = AV\_NOPTS\_VALUE;
62. #endif
63. st->info->fps\_first\_dts = AV\_NOPTS\_VALUE;
64. st->info->fps\_last\_dts  = AV\_NOPTS\_VALUE;

67. st->inject\_global\_side\_data = s->internal->inject\_global\_side\_data;

70. s->streams[s->nb\_streams++] = st;
71. **return** st;
72. }

从代码中可以看出，avformat\_new\_stream()首先调用av\_mallocz()为AVStream分配内存。接着给新分配的AVStream的各个字段赋上默认值。然后调用了另一个函数avcodec\_alloc\_context3()初始化AVStream中的AVCodecContext。

### avcodec\_alloc\_context3()

avcodec\_alloc\_context3()的声明位于libavcodec\avcodec.h中，如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/41181155) [copy](http://blog.csdn.net/leixiaohua1020/article/details/41181155)

[在CODE上查看代码片](https://code.csdn.net/snippets/611187)

1. /\*\*
2. \* Allocate an AVCodecContext and set its fields to default values. The
3. \* resulting struct should be freed with avcodec\_free\_context().
4. \*
5. \* @param codec if non-NULL, allocate private data and initialize defaults
6. \*              for the given codec. It is illegal to then call avcodec\_open2()
7. \*              with a different codec.
8. \*              If NULL, then the codec-specific defaults won't be initialized,
9. \*              which may result in suboptimal default settings (this is
10. \*              important mainly for encoders, e.g. libx264).
11. \*
12. \* @return An AVCodecContext filled with default values or NULL on failure.
13. \* @see avcodec\_get\_context\_defaults
14. \*/
15. AVCodecContext \*avcodec\_alloc\_context3(**const** AVCodec \*codec);

下面我们看一下avcodec\_alloc\_context3()的定义。下面我们看一下avcodec\_alloc\_context3()的定义。avcodec\_alloc\_context3()的定义位于libavcodec\options.c中。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/41181155) [copy](http://blog.csdn.net/leixiaohua1020/article/details/41181155)

[在CODE上查看代码片](https://code.csdn.net/snippets/611187)

1. AVCodecContext \*avcodec\_alloc\_context3(**const** AVCodec \*codec)
2. {
3. AVCodecContext \*avctx= av\_malloc(**sizeof**(AVCodecContext));

6. **if** (!avctx)
7. **return** NULL;

10. **if**(avcodec\_get\_context\_defaults3(avctx, codec) < 0){
11. av\_free(avctx);
12. **return** NULL;
13. }

16. **return** avctx;
17. }

从代码中可以看出，avcodec\_alloc\_context3()首先调用av\_malloc()为AVCodecContext分配存储空间，然后调用了一个函数avcodec\_get\_context\_defaults3()用于设置该AVCodecContext的默认值。avcodec\_get\_context\_defaults3()的定义如下。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/41181155) [copy](http://blog.csdn.net/leixiaohua1020/article/details/41181155)

[在CODE上查看代码片](https://code.csdn.net/snippets/611187)

1. **int** avcodec\_get\_context\_defaults3(AVCodecContext \*s, **const** AVCodec \*codec)
2. {
3. **int** flags=0;
4. memset(s, 0, **sizeof**(AVCodecContext));

7. s->av\_class = &av\_codec\_context\_class;

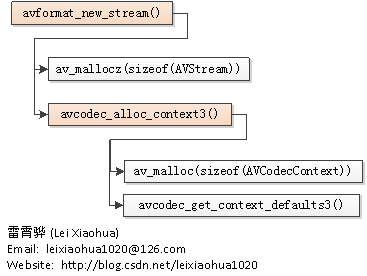
10. s->codec\_type = codec ? codec->type : AVMEDIA\_TYPE\_UNKNOWN;
11. **if** (codec)
12. s->codec\_id = codec->id;

15. **if**(s->codec\_type == AVMEDIA\_TYPE\_AUDIO)
16. flags= AV\_OPT\_FLAG\_AUDIO\_PARAM;
17. **else** **if**(s->codec\_type == AVMEDIA\_TYPE\_VIDEO)
18. flags= AV\_OPT\_FLAG\_VIDEO\_PARAM;
19. **else** **if**(s->codec\_type == AVMEDIA\_TYPE\_SUBTITLE)
20. flags= AV\_OPT\_FLAG\_SUBTITLE\_PARAM;
21. av\_opt\_set\_defaults2(s, flags, flags);

24. s->time\_base           = (AVRational){0,1};
25. s->get\_buffer2         = avcodec\_default\_get\_buffer2;
26. s->get\_format          = avcodec\_default\_get\_format;
27. s->execute             = avcodec\_default\_execute;
28. s->execute2            = avcodec\_default\_execute2;
29. s->sample\_aspect\_ratio = (AVRational){0,1};
30. s->pix\_fmt             = AV\_PIX\_FMT\_NONE;
31. s->sample\_fmt          = AV\_SAMPLE\_FMT\_NONE;
32. s->timecode\_frame\_start = -1;

35. s->reordered\_opaque    = AV\_NOPTS\_VALUE;
36. **if**(codec && codec->priv\_data\_size){
37. **if**(!s->priv\_data){
38. s->priv\_data= av\_mallocz(codec->priv\_data\_size);
39. **if** (!s->priv\_data) {
40. **return** AVERROR(ENOMEM);
41. }
42. }
43. **if**(codec->priv\_class){
44. \*(**const** AVClass\*\*)s->priv\_data = codec->priv\_class;
45. av\_opt\_set\_defaults(s->priv\_data);
46. }
47. }
48. **if** (codec && codec->defaults) {
49. **int** ret;
50. **const** AVCodecDefault \*d = codec->defaults;
51. **while** (d->key) {
52. ret = av\_opt\_set(s, d->key, d->value, 0);
53. av\_assert0(ret >= 0);
54. d++;
55. }
56. }
57. **return** 0;
58. }

avformat\_new\_stream()函数的调用结构如下所示。



## AVFrame

AVFrame的初始化函数是av\_frame\_alloc()，销毁函数是av\_frame\_free()。在这里有一点需要注意，旧版的FFmpeg都是使用avcodec\_alloc\_frame()初始化AVFrame的，但是我在写这篇文章的时候，avcodec\_alloc\_frame()已经被标记为“过时的”了，为了保证与时俱进，决定分析新的API——av\_frame\_alloc()。

### av\_frame\_alloc()

av\_frame\_alloc()的声明位于libavutil\frame.h，如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/41181155) [copy](http://blog.csdn.net/leixiaohua1020/article/details/41181155)

[在CODE上查看代码片](https://code.csdn.net/snippets/611187)

1. /\*\*
2. \* Allocate an AVFrame and set its fields to default values.  The resulting
3. \* struct must be freed using av\_frame\_free().
4. \*
5. \* @return An AVFrame filled with default values or NULL on failure.
6. \*
7. \* @note this only allocates the AVFrame itself, not the data buffers. Those
8. \* must be allocated through other means, e.g. with av\_frame\_get\_buffer() or
9. \* manually.
10. \*/
11. AVFrame \*av\_frame\_alloc(**void**);

av\_frame\_alloc()的定义位于libavutil\frame.c。代码如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/41181155) [copy](http://blog.csdn.net/leixiaohua1020/article/details/41181155)

[在CODE上查看代码片](https://code.csdn.net/snippets/611187)

1. AVFrame \*av\_frame\_alloc(**void**)
2. {
3. AVFrame \*frame = av\_mallocz(**sizeof**(\*frame));

6. **if** (!frame)
7. **return** NULL;

10. frame->extended\_data = NULL;
11. get\_frame\_defaults(frame);

14. **return** frame;
15. }

从代码可以看出，av\_frame\_alloc()首先调用av\_mallocz()为AVFrame结构体分配内存。而后调用了一个函数get\_frame\_defaults()用于设置一些默认参数。get\_frame\_defaults()定义如下。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/41181155) [copy](http://blog.csdn.net/leixiaohua1020/article/details/41181155)

[在CODE上查看代码片](https://code.csdn.net/snippets/611187)

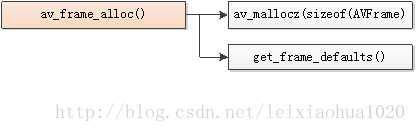
1. **static** **void** get\_frame\_defaults(AVFrame \*frame)
2. {
3. **if** (frame->extended\_data != frame->data)
4. av\_freep(&frame->extended\_data);

7. memset(frame, 0, **sizeof**(\*frame));

10. frame->pts                   =
11. frame->pkt\_dts               =
12. frame->pkt\_pts               = AV\_NOPTS\_VALUE;
13. av\_frame\_set\_best\_effort\_timestamp(frame, AV\_NOPTS\_VALUE);
14. av\_frame\_set\_pkt\_duration         (frame, 0);
15. av\_frame\_set\_pkt\_pos              (frame, -1);
16. av\_frame\_set\_pkt\_size             (frame, -1);
17. frame->key\_frame           = 1;
18. frame->sample\_aspect\_ratio = (AVRational){ 0, 1 };
19. frame->format              = -1; /\* unknown \*/
20. frame->extended\_data       = frame->data;
21. frame->color\_primaries     = AVCOL\_PRI\_UNSPECIFIED;
22. frame->color\_trc           = AVCOL\_TRC\_UNSPECIFIED;
23. frame->colorspace          = AVCOL\_SPC\_UNSPECIFIED;
24. frame->color\_range         = AVCOL\_RANGE\_UNSPECIFIED;
25. frame->chroma\_location     = AVCHROMA\_LOC\_UNSPECIFIED;
26. }

从av\_frame\_alloc()的代码我们可以看出，该函数并没有为AVFrame的像素数据分配空间。因此AVFrame中的像素数据的空间需要自行分配空间，例如使用avpicture\_fill()，av\_image\_fill\_arrays()等函数。

av\_frame\_alloc()函数的调用结构如下所示。



### avpicture\_fill()

avpicture\_fill()的声明位于libavcodec\avcodec.h，如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/41181155) [copy](http://blog.csdn.net/leixiaohua1020/article/details/41181155)

[在CODE上查看代码片](https://code.csdn.net/snippets/611187)

1. /\*\*
2. \* Setup the picture fields based on the specified image parameters
3. \* and the provided image data buffer.
4. \*
5. \* The picture fields are filled in by using the image data buffer
6. \* pointed to by ptr.
7. \*
8. \* If ptr is NULL, the function will fill only the picture linesize
9. \* array and return the required size for the image buffer.
10. \*
11. \* To allocate an image buffer and fill the picture data in one call,
12. \* use avpicture\_alloc().
13. \*
14. \* @param picture       the picture to be filled in
15. \* @param ptr           buffer where the image data is stored, or NULL
16. \* @param pix\_fmt       the pixel format of the image
17. \* @param width         the width of the image in pixels
18. \* @param height        the height of the image in pixels
19. \* @return the size in bytes required for src, a negative error code
20. \* in case of failure
21. \*
22. \* @see av\_image\_fill\_arrays()
23. \*/
24. **int** avpicture\_fill(AVPicture \*picture, **const** uint8\_t \*ptr,
25. **enum** AVPixelFormat pix\_fmt, **int** width, **int** height);

avpicture\_fill()的定义位于libavcodec\avpicture.c，如下所示。

PS：目测这个函数未来也有可能成为“过时的”函数，因为通过观察这一年FFmpeg代码的变化，发现FFmpeg组织似乎想把AVFrame相关的函数（原先定义在AVCodec的头文件中）从AVCodec的代码中分离出来，形成一套单独的API。所以很多和AVFrame相关的名称为avcodec\_XXX()的函数都被标记上了“过时的”标记。当然，上述推测也是我自己猜测的。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/41181155) [copy](http://blog.csdn.net/leixiaohua1020/article/details/41181155)

[在CODE上查看代码片](https://code.csdn.net/snippets/611187)

1. **int** avpicture\_fill(AVPicture \*picture, **const** uint8\_t \*ptr,
2. **enum** AVPixelFormat pix\_fmt, **int** width, **int** height)
3. {
4. **return** av\_image\_fill\_arrays(picture->data, picture->linesize,
5. ptr, pix\_fmt, width, height, 1);
6. }

从代码中可以看出，avpicture\_fill()仅仅是简单调用了一下av\_image\_fill\_arrays()。也就是说这两个函数实际上是等同的。

### av\_image\_fill\_arrays()

av\_image\_fill\_arrays()的声明位于libavutil\imgutils.h中，如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/41181155) [copy](http://blog.csdn.net/leixiaohua1020/article/details/41181155)

[在CODE上查看代码片](https://code.csdn.net/snippets/611187)

1. /\*\*
2. \* Setup the data pointers and linesizes based on the specified image
3. \* parameters and the provided array.
4. \*
5. \* The fields of the given image are filled in by using the src
6. \* address which points to the image data buffer. Depending on the
7. \* specified pixel format, one or multiple image data pointers and
8. \* line sizes will be set.  If a planar format is specified, several
9. \* pointers will be set pointing to the different picture planes and
10. \* the line sizes of the different planes will be stored in the
11. \* lines\_sizes array. Call with !src to get the required
12. \* size for the src buffer.
13. \*
14. \* To allocate the buffer and fill in the dst\_data and dst\_linesize in
15. \* one call, use av\_image\_alloc().
16. \*
17. \* @param dst\_data      data pointers to be filled in
18. \* @param dst\_linesizes linesizes for the image in dst\_data to be filled in
19. \* @param src           buffer which will contain or contains the actual image data, can be NULL
20. \* @param pix\_fmt       the pixel format of the image
21. \* @param width         the width of the image in pixels
22. \* @param height        the height of the image in pixels
23. \* @param align         the value used in src for linesize alignment
24. \* @return the size in bytes required for src, a negative error code
25. \* in case of failure
26. \*/
27. **int** av\_image\_fill\_arrays(uint8\_t \*dst\_data[4], **int** dst\_linesize[4],
28. **const** uint8\_t \*src,
29. **enum** AVPixelFormat pix\_fmt, **int** width, **int** height, **int** align);

av\_image\_fill\_arrays()的定义位于libavutil\imgutils.c中。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/41181155) [copy](http://blog.csdn.net/leixiaohua1020/article/details/41181155)

[在CODE上查看代码片](https://code.csdn.net/snippets/611187)

1. **int** av\_image\_fill\_arrays(uint8\_t \*dst\_data[4], **int** dst\_linesize[4],
2. **const** uint8\_t \*src,
3. **enum** AVPixelFormat pix\_fmt, **int** width, **int** height, **int** align)
4. {
5. **int** ret, i;

8. **if** ((ret = av\_image\_check\_size(width, height, 0, NULL)) < 0)
9. **return** ret;

12. **if** ((ret = av\_image\_fill\_linesizes(dst\_linesize, pix\_fmt, width)) < 0)
13. **return** ret;

16. **for** (i = 0; i < 4; i++)
17. dst\_linesize[i] = FFALIGN(dst\_linesize[i], align);

20. **if** ((ret = av\_image\_fill\_pointers(dst\_data, pix\_fmt, width, NULL, dst\_linesize)) < 0)
21. **return** ret;

24. **return** av\_image\_fill\_pointers(dst\_data, pix\_fmt, height, (uint8\_t \*)src, dst\_linesize);
25. }

av\_image\_fill\_arrays()函数中包含3个函数：av\_image\_check\_size()，av\_image\_fill\_linesizes()，av\_image\_fill\_pointers()。av\_image\_check\_size()用于检查输入的宽高参数是否合理，即不能太大或者为负数。av\_image\_fill\_linesizes()用于填充dst\_linesize。av\_image\_fill\_pointers()则用于填充dst\_data。它们的定义相对比较简单，不再详细分析。  
av\_image\_check\_size()代码如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/41181155) [copy](http://blog.csdn.net/leixiaohua1020/article/details/41181155)

[在CODE上查看代码片](https://code.csdn.net/snippets/611187)

1. **int** av\_image\_check\_size(unsigned **int** w, unsigned **int** h, **int** log\_offset, **void** \*log\_ctx)
2. {
3. ImgUtils imgutils = { &imgutils\_class, log\_offset, log\_ctx };

6. **if** ((**int**)w>0 && (**int**)h>0 && (w+128)\*(uint64\_t)(h+128) < INT\_MAX/8)
7. **return** 0;

10. av\_log(&imgutils, AV\_LOG\_ERROR, "Picture size %ux%u is invalid\n", w, h);
11. **return** AVERROR(EINVAL);
12. }

av\_image\_fill\_linesizes()代码如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/41181155) [copy](http://blog.csdn.net/leixiaohua1020/article/details/41181155)

[在CODE上查看代码片](https://code.csdn.net/snippets/611187)

1. **int** av\_image\_fill\_linesizes(**int** linesizes[4], **enum** AVPixelFormat pix\_fmt, **int** width)
2. {
3. **int** i, ret;
4. **const** AVPixFmtDescriptor \*desc = av\_pix\_fmt\_desc\_get(pix\_fmt);
5. **int** max\_step     [4];       /\* max pixel step for each plane \*/
6. **int** max\_step\_comp[4];       /\* the component for each plane which has the max pixel step \*/

9. memset(linesizes, 0, 4\***sizeof**(linesizes[0]));

12. **if** (!desc || desc->flags & AV\_PIX\_FMT\_FLAG\_HWACCEL)
13. **return** AVERROR(EINVAL);

16. av\_image\_fill\_max\_pixsteps(max\_step, max\_step\_comp, desc);
17. **for** (i = 0; i < 4; i++) {
18. **if** ((ret = image\_get\_linesize(width, i, max\_step[i], max\_step\_comp[i], desc)) < 0)
19. **return** ret;
20. linesizes[i] = ret;
21. }

24. **return** 0;
25. }

av\_image\_fill\_pointers()代码如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/41181155) [copy](http://blog.csdn.net/leixiaohua1020/article/details/41181155)

[在CODE上查看代码片](https://code.csdn.net/snippets/611187)

1. **int** av\_image\_fill\_pointers(uint8\_t \*data[4], **enum** AVPixelFormat pix\_fmt, **int** height,
2. uint8\_t \*ptr, **const** **int** linesizes[4])
3. {
4. **int** i, total\_size, size[4] = { 0 }, has\_plane[4] = { 0 };

7. **const** AVPixFmtDescriptor \*desc = av\_pix\_fmt\_desc\_get(pix\_fmt);
8. memset(data     , 0, **sizeof**(data[0])\*4);

11. **if** (!desc || desc->flags & AV\_PIX\_FMT\_FLAG\_HWACCEL)
12. **return** AVERROR(EINVAL);

15. data[0] = ptr;
16. **if** (linesizes[0] > (INT\_MAX - 1024) / height)
17. **return** AVERROR(EINVAL);
18. size[0] = linesizes[0] \* height;

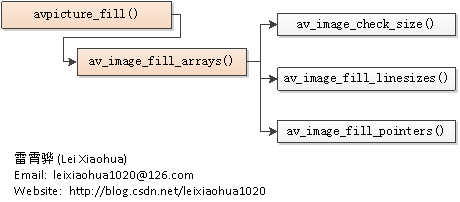
21. **if** (desc->flags & AV\_PIX\_FMT\_FLAG\_PAL ||
22. desc->flags & AV\_PIX\_FMT\_FLAG\_PSEUDOPAL) {
23. size[0] = (size[0] + 3) & ~3;
24. data[1] = ptr + size[0]; /\* palette is stored here as 256 32 bits words \*/
25. **return** size[0] + 256 \* 4;
26. }

29. **for** (i = 0; i < 4; i++)
30. has\_plane[desc->comp[i].plane] = 1;

33. total\_size = size[0];
34. **for** (i = 1; i < 4 && has\_plane[i]; i++) {
35. **int** h, s = (i == 1 || i == 2) ? desc->log2\_chroma\_h : 0;
36. data[i] = data[i-1] + size[i-1];
37. h = (height + (1 << s) - 1) >> s;
38. **if** (linesizes[i] > INT\_MAX / h)
39. **return** AVERROR(EINVAL);
40. size[i] = h \* linesizes[i];
41. **if** (total\_size > INT\_MAX - size[i])
42. **return** AVERROR(EINVAL);
43. total\_size += size[i];
44. }

47. **return** total\_size;
48. }

avpicture\_fill()函数调用关系如下图所示。



### av\_frame\_free()

av\_frame\_free ()的声明位于libavutil\frame.h，如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/41181155) [copy](http://blog.csdn.net/leixiaohua1020/article/details/41181155)

[在CODE上查看代码片](https://code.csdn.net/snippets/611187)

1. /\*\*
2. \* Free the frame and any dynamically allocated objects in it,
3. \* e.g. extended\_data. If the frame is reference counted, it will be
4. \* unreferenced first.
5. \*
6. \* @param frame frame to be freed. The pointer will be set to NULL.
7. \*/
8. **void** av\_frame\_free(AVFrame \*\*frame);

av\_frame\_free ()的定义位于libavutil\frame.c。代码如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/41181155) [copy](http://blog.csdn.net/leixiaohua1020/article/details/41181155)

[在CODE上查看代码片](https://code.csdn.net/snippets/611187)

1. **void** av\_frame\_free(AVFrame \*\*frame)
2. {
3. **if** (!frame || !\*frame)
4. **return**;

7. av\_frame\_unref(\*frame);
8. av\_freep(frame);
9. }

在释放AVFrame结构体之前，首先调用了一个函数av\_frame\_unref()。av\_frame\_unref()也是一个FFmpeg的API，它的作用是释放AVFrame中参考的缓存（还没完全弄懂），并且重置AVFrame中的字段。调用这个函数的目的应该是为了确保AVFrame可以被正常释放。代码如下。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/41181155) [copy](http://blog.csdn.net/leixiaohua1020/article/details/41181155)

[在CODE上查看代码片](https://code.csdn.net/snippets/611187)

1. **void** av\_frame\_unref(AVFrame \*frame)
2. {
3. **int** i;

6. **for** (i = 0; i < frame->nb\_side\_data; i++) {
7. free\_side\_data(&frame->side\_data[i]);
8. }
9. av\_freep(&frame->side\_data);

12. **for** (i = 0; i < FF\_ARRAY\_ELEMS(frame->buf); i++)
13. av\_buffer\_unref(&frame->buf[i]);
14. **for** (i = 0; i < frame->nb\_extended\_buf; i++)
15. av\_buffer\_unref(&frame->extended\_buf[i]);
16. av\_freep(&frame->extended\_buf);
17. av\_dict\_free(&frame->metadata);
18. av\_buffer\_unref(&frame->qp\_table\_buf);

21. get\_frame\_defaults(frame);
22. }

## AVPacket

AVPacket的初始化函数有两个：av\_init\_packet()，av\_new\_packet()。销毁函数是av\_free\_packet()。在初始化函数中av\_init\_packet()比较简单，初始化一些字段；而av\_new\_packet()相对“高级”一些，除了包含av\_init\_packet()的功能之外，还包含了AVPacket内部内存的分配。下面分别看看这些函数。

### av\_init\_packet()

av\_init\_packet()的声明位于libavcodec\avcodec.h，如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/41181155) [copy](http://blog.csdn.net/leixiaohua1020/article/details/41181155)

[在CODE上查看代码片](https://code.csdn.net/snippets/611187)

1. /\*\*
2. \* Initialize optional fields of a packet with default values.
3. \*
4. \* Note, this does not touch the data and size members, which have to be
5. \* initialized separately.
6. \*
7. \* @param pkt packet
8. \*/
9. **void** av\_init\_packet(AVPacket \*pkt);

av\_init\_packet()的定义位于libavcodec\avpacket.c。如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/41181155) [copy](http://blog.csdn.net/leixiaohua1020/article/details/41181155)

[在CODE上查看代码片](https://code.csdn.net/snippets/611187)

1. **void** av\_init\_packet(AVPacket \*pkt)
2. {
3. pkt->pts                  = AV\_NOPTS\_VALUE;
4. pkt->dts                  = AV\_NOPTS\_VALUE;
5. pkt->pos                  = -1;
6. pkt->duration             = 0;
7. pkt->convergence\_duration = 0;
8. pkt->flags                = 0;
9. pkt->stream\_index         = 0;
10. #if FF\_API\_DESTRUCT\_PACKET
11. FF\_DISABLE\_DEPRECATION\_WARNINGS
12. pkt->destruct             = NULL;
13. FF\_ENABLE\_DEPRECATION\_WARNINGS
14. #endif
15. pkt->buf                  = NULL;
16. pkt->side\_data            = NULL;
17. pkt->side\_data\_elems      = 0;
18. }

### av\_new\_packet()

av\_new\_packet()的声明位于libavcodec\avcodec.h。如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/41181155) [copy](http://blog.csdn.net/leixiaohua1020/article/details/41181155)

[在CODE上查看代码片](https://code.csdn.net/snippets/611187)

1. /\*\*
2. \* Allocate the payload of a packet and initialize its fields with
3. \* default values.
4. \*
5. \* @param pkt packet
6. \* @param size wanted payload size
7. \* @return 0 if OK, AVERROR\_xxx otherwise
8. \*/
9. **int** av\_new\_packet(AVPacket \*pkt, **int** size);

av\_new\_packet()的定义位于libavcodec\avpacket.c。如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/41181155) [copy](http://blog.csdn.net/leixiaohua1020/article/details/41181155)

[在CODE上查看代码片](https://code.csdn.net/snippets/611187)

1. **int** av\_new\_packet(AVPacket \*pkt, **int** size)
2. {
3. AVBufferRef \*buf = NULL;
4. **int** ret = packet\_alloc(&buf, size);
5. **if** (ret < 0)
6. **return** ret;

9. av\_init\_packet(pkt);
10. pkt->buf      = buf;
11. pkt->data     = buf->data;
12. pkt->size     = size;
13. #if FF\_API\_DESTRUCT\_PACKET
14. FF\_DISABLE\_DEPRECATION\_WARNINGS
15. pkt->destruct = dummy\_destruct\_packet;
16. FF\_ENABLE\_DEPRECATION\_WARNINGS
17. #endif

20. **return** 0;
21. }

从代码可以看出，av\_new\_packet()调用了av\_init\_packet(pkt)。此外还调用了一个函数packet\_alloc()。packet\_alloc()函数的定义如下。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/41181155) [copy](http://blog.csdn.net/leixiaohua1020/article/details/41181155)

[在CODE上查看代码片](https://code.csdn.net/snippets/611187)

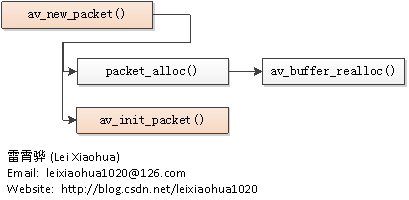
1. **static** **int** packet\_alloc(AVBufferRef \*\*buf, **int** size)
2. {
3. **int** ret;
4. **if** ((unsigned)size >= (unsigned)size + FF\_INPUT\_BUFFER\_PADDING\_SIZE)
5. **return** AVERROR(EINVAL);

8. ret = av\_buffer\_realloc(buf, size + FF\_INPUT\_BUFFER\_PADDING\_SIZE);
9. **if** (ret < 0)
10. **return** ret;

13. memset((\*buf)->data + size, 0, FF\_INPUT\_BUFFER\_PADDING\_SIZE);

16. **return** 0;
17. }

packet\_alloc()中调用av\_buffer\_realloc()为AVPacket分配内存。然后调用memset()将分配的内存置0。  
PS：发现AVPacket的结构随着FFmpeg的发展越发复杂了。原先AVPacket中的数据仅仅存在一个uint8\_t类型的数组里，而现在已经使用一个专门的结构体AVBufferRef存储数据。  
av\_new\_packet()代码的函数调用关系如下图所示。



### av\_free\_packet()

av\_free\_packet()的声明位于libavcodec\avcodec.h，如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/41181155) [copy](http://blog.csdn.net/leixiaohua1020/article/details/41181155)

[在CODE上查看代码片](https://code.csdn.net/snippets/611187)

1. /\*\*
2. \* Free a packet.
3. \*
4. \* @param pkt packet to free
5. \*/
6. **void** av\_free\_packet(AVPacket \*pkt);

av\_free\_packet()的定义位于libavcodec\avpacket.c。如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/41181155) [copy](http://blog.csdn.net/leixiaohua1020/article/details/41181155)

[在CODE上查看代码片](https://code.csdn.net/snippets/611187)

1. **void** av\_free\_packet(AVPacket \*pkt)
2. {
3. **if** (pkt) {
4. FF\_DISABLE\_DEPRECATION\_WARNINGS
5. **if** (pkt->buf)
6. av\_buffer\_unref(&pkt->buf);
7. #if FF\_API\_DESTRUCT\_PACKET
8. **else** **if** (pkt->destruct)
9. pkt->destruct(pkt);
10. pkt->destruct = NULL;
11. #endif
12. FF\_ENABLE\_DEPRECATION\_WARNINGS
13. pkt->data            = NULL;
14. pkt->size            = 0;

17. av\_packet\_free\_side\_data(pkt);
18. }
19. }

从代码可以看出，av\_free\_packet()调用av\_buffer\_unref()释放AVPacket中的数据，而后还调用了av\_packet\_free\_side\_data()释放了side\_data（存储封装格式可以提供的额外的数据）。

# [FFmpeg源代码简单分析：avio\_open2()](http://blog.csdn.net/leixiaohua1020/article/details/41199947)

本文简单分析FFmpeg中一个常用的函数avio\_open2()。该函数用于打开FFmpeg的输入输出文件。avio\_open2()的声明位于libavformat\avio.h文件中，如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/41199947) [copy](http://blog.csdn.net/leixiaohua1020/article/details/41199947)

[在CODE上查看代码片](https://code.csdn.net/snippets/611806)

1. /\*\*
2. \* Create and initialize a AVIOContext for accessing the
3. \* resource indicated by url.
4. \* @note When the resource indicated by url has been opened in
5. \* read+write mode, the AVIOContext can be used only for writing.
6. \*
7. \* @param s Used to return the pointer to the created AVIOContext.
8. \* In case of failure the pointed to value is set to NULL.
9. \* @param url resource to access
10. \* @param flags flags which control how the resource indicated by url
11. \* is to be opened
12. \* @param int\_cb an interrupt callback to be used at the protocols level
13. \* @param options  A dictionary filled with protocol-private options. On return
14. \* this parameter will be destroyed and replaced with a dict containing options
15. \* that were not found. May be NULL.
16. \* @return >= 0 in case of success, a negative value corresponding to an
17. \* AVERROR code in case of failure
18. \*/
19. **int** avio\_open2(AVIOContext \*\*s, **const** **char** \*url, **int** flags,
20. **const** AVIOInterruptCB \*int\_cb, AVDictionary \*\*options);

avio\_open2()函数参数的含义如下：

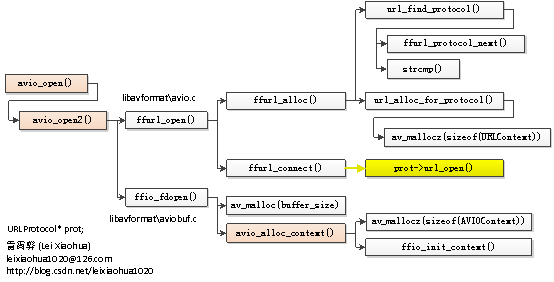
s：函数调用成功之后创建的AVIOContext结构体。  
url：输入输出协议的地址（文件也是一种“广义”的协议，对于文件来说就是文件的路径）。  
flags：打开地址的方式。可以选择只读，只写，或者读写。取值如下。  
 AVIO\_FLAG\_READ：只读。  
 AVIO\_FLAG\_WRITE：只写。  
 AVIO\_FLAG\_READ\_WRITE：读写。  
int\_cb：目前还没有用过。

options：目前还没有用过。

该函数最典型的例子可以参考：[最简单的基于FFMPEG的视频编码器（YUV编码为H.264）](http://blog.csdn.net/leixiaohua1020/article/details/25430425)

## 函数调用结构图

首先贴出来最终分析得出的函数调用结构图，如下所示。

[](https://my.csdn.net/leixiaohua1020/album/detail/1804801)

[单击查看更清晰的图片](https://my.csdn.net/leixiaohua1020/album/detail/1804801)

## avio\_open()

有一个和avio\_open2()“长得很像”的函数avio\_open()，应该是avio\_open2()的早期版本。avio\_open()比avio\_open2()少了最后2个参数。而它前面几个参数的含义和avio\_open2()是一样的。从源代码中可以看出，avio\_open()内部调用了avio\_open2()，并且把avio\_open2()的后2个参数设置成了NULL，因此它的功能实际上和avio\_open2()是一样的。avio\_open()源代码如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/41199947) [copy](http://blog.csdn.net/leixiaohua1020/article/details/41199947)

[在CODE上查看代码片](https://code.csdn.net/snippets/611806)

1. **int** avio\_open(AVIOContext \*\*s, **const** **char** \*filename, **int** flags)
2. {
3. **return** avio\_open2(s, filename, flags, NULL, NULL);
4. }

## avio\_open2()

下面看一下avio\_open2()的源代码，位于libavformat\aviobuf.c文件中。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/41199947) [copy](http://blog.csdn.net/leixiaohua1020/article/details/41199947)

[在CODE上查看代码片](https://code.csdn.net/snippets/611806)

1. **int** avio\_open2(AVIOContext \*\*s, **const** **char** \*filename, **int** flags,
2. **const** AVIOInterruptCB \*int\_cb, AVDictionary \*\*options)
3. {
4. URLContext \*h;
5. **int** err;

8. err = ffurl\_open(&h, filename, flags, int\_cb, options);
9. **if** (err < 0)
10. **return** err;
11. err = ffio\_fdopen(s, h);
12. **if** (err < 0) {
13. ffurl\_close(h);
14. **return** err;
15. }
16. **return** 0;
17. }

从avio\_open2()的源代码可以看出，它主要调用了2个函数：ffurl\_open()和ffio\_fdopen()。其中ffurl\_open()用于初始化URLContext，ffio\_fdopen()用于根据URLContext初始化AVIOContext。URLContext中包含的URLProtocol完成了具体的协议读写等工作。AVIOContext则是在URLContext的读写函数外面加上了一层“包装”（通过retry\_transfer\_wrapper()函数）。

### URLProtocol和URLContext

在查看ffurl\_open()和ffio\_fdopen()函数之前，首先查看一下URLContext和URLProtocol的定义。这两个结构体在FFmpeg的早期版本的SDK中是定义在头文件中可以直接使用的。但是近期的FFmpeg的SDK中已经找不到这两个结构体的定义了。FFmpeg把这两个结构体移动到了源代码的内部，变成了内部结构体。  
URLProtocol的定义位于libavformat\url.h，如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/41199947) [copy](http://blog.csdn.net/leixiaohua1020/article/details/41199947)

[在CODE上查看代码片](https://code.csdn.net/snippets/611806)

1. **typedef** **struct** URLProtocol {
2. **const** **char** \*name;
3. **int**     (\*url\_open)( URLContext \*h, **const** **char** \*url, **int** flags);
4. /\*\*
5. \* This callback is to be used by protocols which open further nested
6. \* protocols. options are then to be passed to ffurl\_open()/ffurl\_connect()
7. \* for those nested protocols.
8. \*/
9. **int**     (\*url\_open2)(URLContext \*h, **const** **char** \*url, **int** flags, AVDictionary \*\*options);

12. /\*\*
13. \* Read data from the protocol.
14. \* If data is immediately available (even less than size), EOF is
15. \* reached or an error occurs (including EINTR), return immediately.
16. \* Otherwise:
17. \* In non-blocking mode, return AVERROR(EAGAIN) immediately.
18. \* In blocking mode, wait for data/EOF/error with a short timeout (0.1s),
19. \* and return AVERROR(EAGAIN) on timeout.
20. \* Checking interrupt\_callback, looping on EINTR and EAGAIN and until
21. \* enough data has been read is left to the calling function; see
22. \* retry\_transfer\_wrapper in avio.c.
23. \*/
24. **int**     (\*url\_read)( URLContext \*h, unsigned **char** \*buf, **int** size);
25. **int**     (\*url\_write)(URLContext \*h, **const** unsigned **char** \*buf, **int** size);
26. int64\_t (\*url\_seek)( URLContext \*h, int64\_t pos, **int** whence);
27. **int**     (\*url\_close)(URLContext \*h);
28. **struct** URLProtocol \*next;
29. **int** (\*url\_read\_pause)(URLContext \*h, **int** pause);
30. int64\_t (\*url\_read\_seek)(URLContext \*h, **int** stream\_index,
31. int64\_t timestamp, **int** flags);
32. **int** (\*url\_get\_file\_handle)(URLContext \*h);
33. **int** (\*url\_get\_multi\_file\_handle)(URLContext \*h, **int** \*\*handles,
34. **int** \*numhandles);
35. **int** (\*url\_shutdown)(URLContext \*h, **int** flags);
36. **int** priv\_data\_size;
37. **const** AVClass \*priv\_data\_class;
38. **int** flags;
39. **int** (\*url\_check)(URLContext \*h, **int** mask);
40. } URLProtocol;

从URLProtocol的定义可以看出，其中包含了用于协议读写的函数指针。例如：  
url\_open()：打开协议。  
url\_read()：读数据。  
url\_write()：写数据。  
url\_close()：关闭协议。  
每种具体的协议都包含了一个URLProtocol结构体，例如：  
FILE协议（“文件”在FFmpeg中也被当做一种协议）的结构体ff\_file\_protocol的定义如下所示（位于libavformat\file.c）。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/41199947) [copy](http://blog.csdn.net/leixiaohua1020/article/details/41199947)

[在CODE上查看代码片](https://code.csdn.net/snippets/611806)

1. URLProtocol ff\_file\_protocol = {
2. .name                = "file",
3. .url\_open            = file\_open,
4. .url\_read            = file\_read,
5. .url\_write           = file\_write,
6. .url\_seek            = file\_seek,
7. .url\_close           = file\_close,
8. .url\_get\_file\_handle = file\_get\_handle,
9. .url\_check           = file\_check,
10. .priv\_data\_size      = **sizeof**(FileContext),
11. .priv\_data\_class     = &file\_class,
12. };

在使用FILE协议进行读写的时候，调用url\_open()实际上就是调用了file\_open()函数，这里限于篇幅不再对file\_open()的源代码进行分析。file\_open()函数实际上调用了系统的打开文件函数open()。同理，调用url\_read()实际上就是调用了file\_read()函数；file\_read()函数实际上调用了系统的读取文件函数read()。url\_write()，url\_seek()等函数的道理都是一样的。  
  
LibRTMP协议的结构体ff\_librtmp\_protocol的定义如下所示（位于libavformat\librtmp.c）。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/41199947) [copy](http://blog.csdn.net/leixiaohua1020/article/details/41199947)

[在CODE上查看代码片](https://code.csdn.net/snippets/611806)

1. URLProtocol ff\_librtmp\_protocol = {
2. .name                = "rtmp",
3. .url\_open            = rtmp\_open,
4. .url\_read            = rtmp\_read,
5. .url\_write           = rtmp\_write,
6. .url\_close           = rtmp\_close,
7. .url\_read\_pause      = rtmp\_read\_pause,
8. .url\_read\_seek       = rtmp\_read\_seek,
9. .url\_get\_file\_handle = rtmp\_get\_file\_handle,
10. .priv\_data\_size      = **sizeof**(LibRTMPContext),
11. .priv\_data\_class     = &librtmp\_class,
12. .flags               = URL\_PROTOCOL\_FLAG\_NETWORK,
13. };

UDP协议的结构体ff\_udp\_protocol的定义如下所示（位于libavformat\udp.c）。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/41199947) [copy](http://blog.csdn.net/leixiaohua1020/article/details/41199947)

[在CODE上查看代码片](https://code.csdn.net/snippets/611806)

1. URLProtocol ff\_udp\_protocol = {
2. .name                = "udp",
3. .url\_open            = udp\_open,
4. .url\_read            = udp\_read,
5. .url\_write           = udp\_write,
6. .url\_close           = udp\_close,
7. .url\_get\_file\_handle = udp\_get\_file\_handle,
8. .priv\_data\_size      = **sizeof**(UDPContext),
9. .priv\_data\_class     = &udp\_context\_class,
10. .flags               = URL\_PROTOCOL\_FLAG\_NETWORK,
11. };

上文中简单介绍了URLProtocol结构体。下面看一下URLContext结构体。URLContext的定义也位于libavformat\url.h，如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/41199947) [copy](http://blog.csdn.net/leixiaohua1020/article/details/41199947)

[在CODE上查看代码片](https://code.csdn.net/snippets/611806)

1. **typedef** **struct** URLContext {
2. **const** AVClass \*av\_class;    /\*\*< information for av\_log(). Set by url\_open(). \*/
3. **struct** URLProtocol \*prot;
4. **void** \*priv\_data;
5. **char** \*filename;             /\*\*< specified URL \*/
6. **int** flags;
7. **int** max\_packet\_size;        /\*\*< if non zero, the stream is packetized with this max packet size \*/
8. **int** is\_streamed;            /\*\*< true if streamed (no seek possible), default = false \*/
9. **int** is\_connected;
10. AVIOInterruptCB interrupt\_callback;
11. int64\_t rw\_timeout;         /\*\*< maximum time to wait for (network) read/write operation completion, in mcs \*/
12. } URLContext;

从代码中可以看出，URLProtocol结构体是URLContext结构体的一个成员。由于还没有对URLContext结构体进行详细研究，有关该结构体的代码不再做过多分析。

## ffurl\_open()

前文提到AVIOContext中主要调用了2个函数：ffurl\_open()和ffio\_fdopen()。其中ffurl\_open()用于初始化URLContext，ffio\_fdopen()用于根据URLContext初始化AVIOContext。下面首先看一下初始化URLContext的函数ffurl\_open()。  
ffurl\_open()的函数定义位于libavformat\avio.c中，如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/41199947) [copy](http://blog.csdn.net/leixiaohua1020/article/details/41199947)

[在CODE上查看代码片](https://code.csdn.net/snippets/611806)

1. **int** ffurl\_open(URLContext \*\*puc, **const** **char** \*filename, **int** flags,
2. **const** AVIOInterruptCB \*int\_cb, AVDictionary \*\*options)
3. {
4. **int** ret = ffurl\_alloc(puc, filename, flags, int\_cb);
5. **if** (ret < 0)
6. **return** ret;
7. **if** (options && (\*puc)->prot->priv\_data\_class &&
8. (ret = av\_opt\_set\_dict((\*puc)->priv\_data, options)) < 0)
9. **goto** fail;
10. **if** ((ret = av\_opt\_set\_dict(\*puc, options)) < 0)
11. **goto** fail;
12. ret = ffurl\_connect(\*puc, options);
13. **if** (!ret)
14. **return** 0;
15. fail:
16. ffurl\_close(\*puc);
17. \*puc = NULL;
18. **return** ret;
19. }

从代码中可以看出，ffurl\_open()主要调用了2个函数：ffurl\_alloc()和ffurl\_connect()。ffurl\_alloc()用于查找合适的URLProtocol，并创建一个URLContext；ffurl\_connect()用于打开获得的URLProtocol。

#### ffurl\_alloc()

ffurl\_alloc()的定义位于libavformat\avio.c中，如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/41199947) [copy](http://blog.csdn.net/leixiaohua1020/article/details/41199947)

[在CODE上查看代码片](https://code.csdn.net/snippets/611806)

1. **int** ffurl\_alloc(URLContext \*\*puc, **const** **char** \*filename, **int** flags,
2. **const** AVIOInterruptCB \*int\_cb)
3. {
4. URLProtocol \*p = NULL;

7. **if** (!first\_protocol) {
8. av\_log(NULL, AV\_LOG\_WARNING, "No URL Protocols are registered. "
9. "Missing call to av\_register\_all()?\n");
10. }

13. p = url\_find\_protocol(filename);
14. **if** (p)
15. **return** url\_alloc\_for\_protocol(puc, p, filename, flags, int\_cb);

18. \*puc = NULL;
19. **if** (av\_strstart(filename, "https:", NULL))
20. av\_log(NULL, AV\_LOG\_WARNING, "https protocol not found, recompile with openssl or gnutls enabled.\n");
21. **return** AVERROR\_PROTOCOL\_NOT\_FOUND;
22. }

从代码中可以看出，ffurl\_alloc()主要调用了2个函数：url\_find\_protocol()根据文件路径查找合适的URLProtocol，url\_alloc\_for\_protocol()为查找到的URLProtocol创建URLContext。

#### url\_find\_protocol()

先来看一下url\_find\_protocol()函数，定义如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/41199947) [copy](http://blog.csdn.net/leixiaohua1020/article/details/41199947)

[在CODE上查看代码片](https://code.csdn.net/snippets/611806)

1. #define URL\_SCHEME\_CHARS                        \
2. "abcdefghijklmnopqrstuvwxyz"                \
3. "ABCDEFGHIJKLMNOPQRSTUVWXYZ"                \
4. "0123456789+-."
6. **static** **struct** URLProtocol \*url\_find\_protocol(**const** **char** \*filename)
7. {
8. URLProtocol \*up = NULL;
9. **char** proto\_str[128], proto\_nested[128], \*ptr;
10. **size\_t** proto\_len = strspn(filename, URL\_SCHEME\_CHARS);

13. **if** (filename[proto\_len] != ':' &&
14. (filename[proto\_len] != ',' || !strchr(filename + proto\_len + 1, ':')) ||
15. is\_dos\_path(filename))
16. strcpy(proto\_str, "file");
17. **else**
18. av\_strlcpy(proto\_str, filename,
19. FFMIN(proto\_len + 1, **sizeof**(proto\_str)));

22. **if** ((ptr = strchr(proto\_str, ',')))
23. \*ptr = '\0';
24. av\_strlcpy(proto\_nested, proto\_str, **sizeof**(proto\_nested));
25. **if** ((ptr = strchr(proto\_nested, '+')))
26. \*ptr = '\0';

29. **while** (up = ffurl\_protocol\_next(up)) {
30. **if** (!strcmp(proto\_str, up->name))
31. **break**;
32. **if** (up->flags & URL\_PROTOCOL\_FLAG\_NESTED\_SCHEME &&
33. !strcmp(proto\_nested, up->name))
34. **break**;
35. }

38. **return** up;
39. }

url\_find\_protocol()函数表明了FFmpeg根据文件路径猜测协议的方法。该函数首先根据strspn()函数查找字符串中第一个“非字母或数字”的字符的位置，并保存在proto\_len中。一般情况下，协议URL中都是包含“:”的，比如说RTMP的URL格式是“rtmp://xxx…”，UDP的URL格式是“udp://…”，HTTP的URL格式是“http://...”。因此，一般情况下proto\_len的数值就是“:”的下标（代表了“:”前面的协议名称的字符的个数，例如rtmp://的proto\_len为4）。  
接下来函数将filename的前proto\_len个字节拷贝至proto\_str字符串中。

PS：

这个地方比较纠结，源代码中av\_strlcpy()函数的第3个参数size写的字符串的长度是（proto\_len+1），但是查了一下av\_strlcpy()的定义，发现该函数至多拷贝（size-1）个字符。这么一涨一消，最终还是拷贝了proto\_len个字节。例如RTMP协议就拷贝了“rtmp”，UDP协议就拷贝了“udp”。

av\_strlcpy()是FFMpeg的一个工具函数，声明位于libavutil\avstring.h，如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/41199947) [copy](http://blog.csdn.net/leixiaohua1020/article/details/41199947)

[在CODE上查看代码片](https://code.csdn.net/snippets/611806)

1. /\*\*
2. \* Copy the string src to dst, but no more than size - 1 bytes, and
3. \* null-terminate dst.
4. \*
5. \* This function is the same as BSD strlcpy().
6. \*
7. \* @param dst destination buffer
8. \* @param src source string
9. \* @param size size of destination buffer
10. \* @return the length of src
11. \*
12. \* @warning since the return value is the length of src, src absolutely
13. \* \_must\_ be a properly 0-terminated string, otherwise this will read beyond
14. \* the end of the buffer and possibly crash.
15. \*/
16. **size\_t** av\_strlcpy(**char** \*dst, **const** **char** \*src, **size\_t** size);

这里有一种例外，那就是文件路径。“文件”在FFmpeg中也是一种“协议”，并且前缀是“file”。也就是标准的文件路径应该是“file://...”格式的。但是这太不符合我们一般人的使用习惯，我们一般是不会在文件路径前面加上“file”协议名称的。所以该函数采取的方法是：一旦检测出来输入的URL是文件路径而不是网络协议，就自动向proto\_str中拷贝“file”。

其中判断文件路径那里有一个很复杂的if()语句。根据我的理解，“||”前面的语句用于判断是否是相对文件路径，“||”后面的语句用于判断是否是绝对路径。判断绝对路径的时候用到了一个函数is\_dos\_path()，定义位于libavformat\os\_support.h，如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/41199947) [copy](http://blog.csdn.net/leixiaohua1020/article/details/41199947)

[在CODE上查看代码片](https://code.csdn.net/snippets/611806)

1. **static** **inline** **int** is\_dos\_path(**const** **char** \*path)
2. {
3. #if HAVE\_DOS\_PATHS
4. **if** (path[0] && path[1] == ':')
5. **return** 1;
6. #endif
7. **return** 0;
8. }

注意“&&”优先级低于“==”。如果文件路径第1个字符不为空（一般情况下是盘符）而且第2个字符为“:”，就认为它是绝对文件路径。  
此外url\_find\_protocol()函数中还涉及到一个函数ffurl\_protocol\_next()。该函数用于获得下一个URLProtocol（所有的URLProtocol在FFmpeg初始化注册的时候形成一个链表结构）。ffurl\_protocol\_next()代码极其简单，如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/41199947) [copy](http://blog.csdn.net/leixiaohua1020/article/details/41199947)

[在CODE上查看代码片](https://code.csdn.net/snippets/611806)

1. URLProtocol \*ffurl\_protocol\_next(**const** URLProtocol \*prev)
2. {
3. **return** prev ? prev->next : first\_protocol;
4. }

#### url\_alloc\_for\_protocol()

url\_alloc\_for\_protocol()的定义位于libavformat\avio.c中，如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/41199947) [copy](http://blog.csdn.net/leixiaohua1020/article/details/41199947)

[在CODE上查看代码片](https://code.csdn.net/snippets/611806)

1. **static** **int** url\_alloc\_for\_protocol(URLContext \*\*puc, **struct** URLProtocol \*up,
2. **const** **char** \*filename, **int** flags,
3. **const** AVIOInterruptCB \*int\_cb)
4. {
5. URLContext \*uc;
6. **int** err;

9. #if CONFIG\_NETWORK
10. **if** (up->flags & URL\_PROTOCOL\_FLAG\_NETWORK && !ff\_network\_init())
11. **return** AVERROR(EIO);
12. #endif
13. **if** ((flags & AVIO\_FLAG\_READ) && !up->url\_read) {
14. av\_log(NULL, AV\_LOG\_ERROR,
15. "Impossible to open the '%s' protocol for reading\n", up->name);
16. **return** AVERROR(EIO);
17. }
18. **if** ((flags & AVIO\_FLAG\_WRITE) && !up->url\_write) {
19. av\_log(NULL, AV\_LOG\_ERROR,
20. "Impossible to open the '%s' protocol for writing\n", up->name);
21. **return** AVERROR(EIO);
22. }
23. uc = av\_mallocz(**sizeof**(URLContext) + strlen(filename) + 1);
24. **if** (!uc) {
25. err = AVERROR(ENOMEM);
26. **goto** fail;
27. }
28. uc->av\_class = &ffurl\_context\_class;
29. uc->filename = (**char** \*)&uc[1];
30. strcpy(uc->filename, filename);
31. uc->prot            = up;
32. uc->flags           = flags;
33. uc->is\_streamed     = 0; /\* default = not streamed \*/
34. uc->max\_packet\_size = 0; /\* default: stream file \*/
35. **if** (up->priv\_data\_size) {
36. uc->priv\_data = av\_mallocz(up->priv\_data\_size);
37. **if** (!uc->priv\_data) {
38. err = AVERROR(ENOMEM);
39. **goto** fail;
40. }
41. **if** (up->priv\_data\_class) {
42. **int** proto\_len= strlen(up->name);
43. **char** \*start = strchr(uc->filename, ',');
44. \*(**const** AVClass \*\*)uc->priv\_data = up->priv\_data\_class;
45. av\_opt\_set\_defaults(uc->priv\_data);
46. **if**(!strncmp(up->name, uc->filename, proto\_len) && uc->filename + proto\_len == start){
47. **int** ret= 0;
48. **char** \*p= start;
49. **char** sep= \*++p;
50. **char** \*key, \*val;
51. p++;
52. **while**(ret >= 0 && (key= strchr(p, sep)) && p<key && (val = strchr(key+1, sep))){
53. \*val= \*key= 0;
54. ret= av\_opt\_set(uc->priv\_data, p, key+1, 0);
55. **if** (ret == AVERROR\_OPTION\_NOT\_FOUND)
56. av\_log(uc, AV\_LOG\_ERROR, "Key '%s' not found.\n", p);
57. \*val= \*key= sep;
58. p= val+1;
59. }
60. **if**(ret<0 || p!=key){
61. av\_log(uc, AV\_LOG\_ERROR, "Error parsing options string %s\n", start);
62. av\_freep(&uc->priv\_data);
63. av\_freep(&uc);
64. err = AVERROR(EINVAL);
65. **goto** fail;
66. }
67. memmove(start, key+1, strlen(key));
68. }
69. }
70. }
71. **if** (int\_cb)
72. uc->interrupt\_callback = \*int\_cb;

75. \*puc = uc;
76. **return** 0;
77. fail:
78. \*puc = NULL;
79. **if** (uc)
80. av\_freep(&uc->priv\_data);
81. av\_freep(&uc);
82. #if CONFIG\_NETWORK
83. **if** (up->flags & URL\_PROTOCOL\_FLAG\_NETWORK)
84. ff\_network\_close();
85. #endif
86. **return** err;
87. }

url\_alloc\_for\_protocol()完成了以下步骤：首先，检查输入的URLProtocol是否支持指定的flag。比如flag中如果指定了AVIO\_FLAG\_READ，则URLProtocol中必须包含url\_read()；如果指定了AVIO\_FLAG\_WRITE，则URLProtocol中必须包含url\_write()。在检查无误之后，接着就可以调用av\_mallocz()为即将创建的URLContext分配内存了。接下来基本上就是各种赋值工作，在这里不再详细记录。

#### ffurl\_connect()

ffurl\_connect()用于打开获得的URLProtocol。该函数的定义位于libavformat\avio.c中，如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/41199947) [copy](http://blog.csdn.net/leixiaohua1020/article/details/41199947)

[在CODE上查看代码片](https://code.csdn.net/snippets/611806)

1. **int** ffurl\_connect(URLContext \*uc, AVDictionary \*\*options)
2. {
3. **int** err =
4. uc->prot->url\_open2 ? uc->prot->url\_open2(uc,
5. uc->filename,
6. uc->flags,
7. options) :
8. uc->prot->url\_open(uc, uc->filename, uc->flags);
9. **if** (err)
10. **return** err;
11. uc->is\_connected = 1;
12. /\* We must be careful here as ffurl\_seek() could be slow,
13. \* for example for http \*/
14. **if** ((uc->flags & AVIO\_FLAG\_WRITE) || !strcmp(uc->prot->name, "file"))
15. **if** (!uc->is\_streamed && ffurl\_seek(uc, 0, SEEK\_SET) < 0)
16. uc->is\_streamed = 1;
17. **return** 0;
18. }

该函数最重要的函数就是它的第一句：URLProtocol中是否包含url\_open2()？如果包含的话，就调用url\_open2()，否则就调用url\_open()。

url\_open()本身是URLProtocol的一个函数指针，这个地方根据不同的协议调用的url\_open()具体实现函数也是不一样的，例如file协议的url\_open()对应的是file\_open()，而file\_open()最终调用了\_wsopen()，\_sopen()（Windows下）或者open()（[**Linux**](http://lib.csdn.net/base/linux)下，类似于fopen()）这样的系统中打开文件的API函数；而libRTMP的url\_open()对应的是rtmp\_open()，而rtmp\_open()最终调用了libRTMP的API函数RTMP\_Init()，RTMP\_SetupURL()，RTMP\_Connect() 以及RTMP\_ConnectStream()。

## ffio\_fdopen()

ffio\_fdopen()使用已经获得的URLContext初始化AVIOContext。它的函数定义位于libavformat\aviobuf.c中，如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/41199947) [copy](http://blog.csdn.net/leixiaohua1020/article/details/41199947)

[在CODE上查看代码片](https://code.csdn.net/snippets/611806)

1. #define IO\_BUFFER\_SIZE 32768

4. **int** ffio\_fdopen(AVIOContext \*\*s, URLContext \*h)
5. {
6. uint8\_t \*buffer;
7. **int** buffer\_size, max\_packet\_size;

10. max\_packet\_size = h->max\_packet\_size;
11. **if** (max\_packet\_size) {
12. buffer\_size = max\_packet\_size; /\* no need to bufferize more than one packet \*/
13. } **else** {
14. buffer\_size = IO\_BUFFER\_SIZE;
15. }
16. buffer = av\_malloc(buffer\_size);
17. **if** (!buffer)
18. **return** AVERROR(ENOMEM);

21. \*s = avio\_alloc\_context(buffer, buffer\_size, h->flags & AVIO\_FLAG\_WRITE, h,
22. (**void**\*)ffurl\_read, (**void**\*)ffurl\_write, (**void**\*)ffurl\_seek);
23. **if** (!\*s) {
24. av\_free(buffer);
25. **return** AVERROR(ENOMEM);
26. }
27. (\*s)->direct = h->flags & AVIO\_FLAG\_DIRECT;
28. (\*s)->seekable = h->is\_streamed ? 0 : AVIO\_SEEKABLE\_NORMAL;
29. (\*s)->max\_packet\_size = max\_packet\_size;
30. **if**(h->prot) {
31. (\*s)->read\_pause = (**int** (\*)(**void** \*, **int**))h->prot->url\_read\_pause;
32. (\*s)->read\_seek  = (int64\_t (\*)(**void** \*, **int**, int64\_t, **int**))h->prot->url\_read\_seek;
33. }
34. (\*s)->av\_class = &ffio\_url\_class;
35. **return** 0;
36. }

ffio\_fdopen()函数首先初始化AVIOContext中的Buffer。如果URLContext中设置了max\_packet\_size，则将Buffer的大小设置为max\_packet\_size。如果没有设置的话（似乎大部分URLContext都没有设置该值），则会分配IO\_BUFFER\_SIZE个字节给Buffer。IO\_BUFFER\_SIZE取值为32768。

## avio\_alloc\_context()

ffio\_fdopen()接下来会调用avio\_alloc\_context()初始化一个AVIOContext。avio\_alloc\_context()本身是一个FFmpeg的API函数。它的声明位于libavformat\avio.h中，如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/41199947) [copy](http://blog.csdn.net/leixiaohua1020/article/details/41199947)

[在CODE上查看代码片](https://code.csdn.net/snippets/611806)

1. /\*\*
2. \* Allocate and initialize an AVIOContext for buffered I/O. It must be later
3. \* freed with av\_free().
4. \*
5. \* @param buffer Memory block for input/output operations via AVIOContext.
6. \*        The buffer must be allocated with av\_malloc() and friends.
7. \* @param buffer\_size The buffer size is very important for performance.
8. \*        For protocols with fixed blocksize it should be set to this blocksize.
9. \*        For others a typical size is a cache page, e.g. 4kb.
10. \* @param write\_flag Set to 1 if the buffer should be writable, 0 otherwise.
11. \* @param opaque An opaque pointer to user-specific data.
12. \* @param read\_packet  A function for refilling the buffer, may be NULL.
13. \* @param write\_packet A function for writing the buffer contents, may be NULL.
14. \*        The function may not change the input buffers content.
15. \* @param seek A function for seeking to specified byte position, may be NULL.
16. \*
17. \* @return Allocated AVIOContext or NULL on failure.
18. \*/
19. AVIOContext \*avio\_alloc\_context(
20. unsigned **char** \*buffer,
21. **int** buffer\_size,
22. **int** write\_flag,
23. **void** \*opaque,
24. **int** (\*read\_packet)(**void** \*opaque, uint8\_t \*buf, **int** buf\_size),
25. **int** (\*write\_packet)(**void** \*opaque, uint8\_t \*buf, **int** buf\_size),
26. int64\_t (\*seek)(**void** \*opaque, int64\_t offset, **int** whence));

avio\_alloc\_context()看上去参数很多，但实际上并不复杂。先简单解释一下它各个参数的含义：

buffer：AVIOContext中的Buffer。  
buffer\_size：AVIOContext中的Buffer的大小。  
write\_flag：设置为1则Buffer可写；否则Buffer只可读。  
opaque：用户自定义数据。  
read\_packet()：读取外部数据，填充Buffer的函数。  
write\_packet()：向Buffer中写入数据的函数。  
seek()：用于Seek的函数。

该函数成功执行的话则会返回一个创建好的AVIOContext。  
下面看一下avio\_alloc\_context()的定义，位于libavformat\aviobuf.c，如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/41199947) [copy](http://blog.csdn.net/leixiaohua1020/article/details/41199947)

[在CODE上查看代码片](https://code.csdn.net/snippets/611806)

1. AVIOContext \*avio\_alloc\_context(
2. unsigned **char** \*buffer,
3. **int** buffer\_size,
4. **int** write\_flag,
5. **void** \*opaque,
6. **int** (\*read\_packet)(**void** \*opaque, uint8\_t \*buf, **int** buf\_size),
7. **int** (\*write\_packet)(**void** \*opaque, uint8\_t \*buf, **int** buf\_size),
8. int64\_t (\*seek)(**void** \*opaque, int64\_t offset, **int** whence))
9. {
10. AVIOContext \*s = av\_mallocz(**sizeof**(AVIOContext));
11. **if** (!s)
12. **return** NULL;
13. ffio\_init\_context(s, buffer, buffer\_size, write\_flag, opaque,
14. read\_packet, write\_packet, seek);
15. **return** s;
16. }

该函数代码很简单：首先调用av\_mallocz()为AVIOContext分配一块内存空间，然后基本上将所有输入参数传递给ffio\_init\_context()。

#### ffio\_init\_context()

ffio\_init\_context()的定义如下。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/41199947) [copy](http://blog.csdn.net/leixiaohua1020/article/details/41199947)

[在CODE上查看代码片](https://code.csdn.net/snippets/611806)

1. **int** ffio\_init\_context(AVIOContext \*s,
2. unsigned **char** \*buffer,
3. **int** buffer\_size,
4. **int** write\_flag,
5. **void** \*opaque,
6. **int** (\*read\_packet)(**void** \*opaque, uint8\_t \*buf, **int** buf\_size),
7. **int** (\*write\_packet)(**void** \*opaque, uint8\_t \*buf, **int** buf\_size),
8. int64\_t (\*seek)(**void** \*opaque, int64\_t offset, **int** whence))
9. {
10. s->buffer      = buffer;
11. s->orig\_buffer\_size =
12. s->buffer\_size = buffer\_size;
13. s->buf\_ptr     = buffer;
14. s->opaque      = opaque;
15. s->direct      = 0;

18. url\_resetbuf(s, write\_flag ? AVIO\_FLAG\_WRITE : AVIO\_FLAG\_READ);

21. s->write\_packet    = write\_packet;
22. s->read\_packet     = read\_packet;
23. s->seek            = seek;
24. s->pos             = 0;
25. s->must\_flush      = 0;
26. s->eof\_reached     = 0;
27. s->error           = 0;
28. s->seekable        = seek ? AVIO\_SEEKABLE\_NORMAL : 0;
29. s->max\_packet\_size = 0;
30. s->update\_checksum = NULL;

33. **if** (!read\_packet && !write\_flag) {
34. s->pos     = buffer\_size;
35. s->buf\_end = s->buffer + buffer\_size;
36. }
37. s->read\_pause = NULL;
38. s->read\_seek  = NULL;

41. **return** 0;
42. }

可以看出，这个函数的工作就是各种赋值，不算很有“技术含量”，不再详述。

#### ffurl\_read()，ffurl\_write()，ffurl\_seek()

现在我们再回到ffio\_fdopen()，会发现它初始化AVIOContext的结构体的时候，首先将自己分配的Buffer设置为该AVIOContext的Buffer；然后将URLContext作为用户自定义数据（对应AVIOContext的opaque变量）提供给该AVIOContext；最后分别将3个函数作为该AVIOContext的读，写，跳转函数：ffurl\_read()，ffurl\_write()，ffurl\_seek()。下面我们选择一个ffurl\_read()看看它的定义。  
ffurl\_read()的定义位于libavformat\avio.c，如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/41199947) [copy](http://blog.csdn.net/leixiaohua1020/article/details/41199947)

[在CODE上查看代码片](https://code.csdn.net/snippets/611806)

1. **int** ffurl\_read(URLContext \*h, unsigned **char** \*buf, **int** size)
2. {
3. **if** (!(h->flags & AVIO\_FLAG\_READ))
4. **return** AVERROR(EIO);
5. **return** retry\_transfer\_wrapper(h, buf, size, 1, h->prot->url\_read);
6. }

该函数先判断了一下输入的URLContext是否支持“读”操作，接着调用了一个函数：retry\_transfer\_wrapper()。  
如果我们看ffurl\_write()的代码，如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/41199947) [copy](http://blog.csdn.net/leixiaohua1020/article/details/41199947)

[在CODE上查看代码片](https://code.csdn.net/snippets/611806)

1. **int** ffurl\_write(URLContext \*h, **const** unsigned **char** \*buf, **int** size)
2. {
3. **if** (!(h->flags & AVIO\_FLAG\_WRITE))
4. **return** AVERROR(EIO);
5. /\* avoid sending too big packets \*/
6. **if** (h->max\_packet\_size && size > h->max\_packet\_size)
7. **return** AVERROR(EIO);

10. **return** retry\_transfer\_wrapper(h, (unsigned **char** \*)buf, size, size, (**void**\*)h->prot->url\_write);
11. }

会发现他也调用了同样的一个函数retry\_transfer\_wrapper()。唯一的不同在于ffurl\_read()调用retry\_transfer\_wrapper()的时候，最后一个参数是URLProtocol的url\_read()，而ffurl\_write()调用retry\_transfer\_wrapper()的时候，最后一个参数是URLProtocol的url\_write()。  
下面我们看一下retry\_transfer\_wrapper()的定义，位于libavformat\avio.c，如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/41199947) [copy](http://blog.csdn.net/leixiaohua1020/article/details/41199947)

[在CODE上查看代码片](https://code.csdn.net/snippets/611806)

1. **static** **inline** **int** retry\_transfer\_wrapper(URLContext \*h, uint8\_t \*buf,
2. **int** size, **int** size\_min,
3. **int** (\*transfer\_func)(URLContext \*h,
4. uint8\_t \*buf,
5. **int** size))
6. {
7. **int** ret, len;
8. **int** fast\_retries = 5;
9. int64\_t wait\_since = 0;

12. len = 0;
13. **while** (len < size\_min) {
14. **if** (ff\_check\_interrupt(&h->interrupt\_callback))
15. **return** AVERROR\_EXIT;
16. ret = transfer\_func(h, buf + len, size - len);
17. **if** (ret == AVERROR(EINTR))
18. **continue**;
19. **if** (h->flags & AVIO\_FLAG\_NONBLOCK)
20. **return** ret;
21. **if** (ret == AVERROR(EAGAIN)) {
22. ret = 0;
23. **if** (fast\_retries) {
24. fast\_retries--;
25. } **else** {
26. **if** (h->rw\_timeout) {
27. **if** (!wait\_since)
28. wait\_since = av\_gettime\_relative();
29. **else** **if** (av\_gettime\_relative() > wait\_since + h->rw\_timeout)
30. **return** AVERROR(EIO);
31. }
32. av\_usleep(1000);
33. }
34. } **else** **if** (ret < 1)
35. **return** (ret < 0 && ret != AVERROR\_EOF) ? ret : len;
36. **if** (ret)
37. fast\_retries = FFMAX(fast\_retries, 2);
38. len += ret;
39. }
40. **return** len;
41. }

从代码中可以看出，它的核心实际上是调用了一个名称为transfer\_func()的函数。而该函数就是retry\_transfer\_wrapper()的第四个参数。该函数实际上是对URLProtocol的读写操作中的错误进行了一些“容错”处理，可以让数据的读写更加的稳定。

avio\_alloc\_context()执行完毕后，ffio\_fdopen()函数的工作就基本完成了，avio\_open2()的工作也就做完了。

# [FFmpeg源代码简单分析：av\_find\_decoder()和av\_find\_encoder()](http://blog.csdn.net/leixiaohua1020/article/details/44084557)

本文记录FFmpeg的两个API函数：avcodec\_find\_encoder()和avcodec\_find\_decoder()。avcodec\_find\_encoder()用于查找FFmpeg的编码器，avcodec\_find\_decoder()用于查找FFmpeg的解码器。

avcodec\_find\_encoder()的声明位于libavcodec\avcodec.h，如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44084557) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44084557)

[在CODE上查看代码片](https://code.csdn.net/snippets/613144)

1. /\*\*
2. \* Find a registered encoder with a matching codec ID.
3. \*
4. \* @param id AVCodecID of the requested encoder
5. \* @return An encoder if one was found, NULL otherwise.
6. \*/
7. AVCodec \*avcodec\_find\_encoder(**enum** AVCodecID id);

函数的参数是一个编码器的ID，返回查找到的编码器（没有找到就返回NULL）。  
avcodec\_find\_decoder()的声明也位于libavcodec\avcodec.h，如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44084557) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44084557)

[在CODE上查看代码片](https://code.csdn.net/snippets/613144)

1. /\*\*
2. \* Find a registered decoder with a matching codec ID.
3. \*
4. \* @param id AVCodecID of the requested decoder
5. \* @return A decoder if one was found, NULL otherwise.
6. \*/
7. AVCodec \*avcodec\_find\_decoder(**enum** AVCodecID id);

函数的参数是一个解码器的ID，返回查找到的解码器（没有找到就返回NULL）。

avcodec\_find\_encoder()函数最典型的例子可以参考：

[最简单的基于FFMPEG的视频编码器（YUV编码为H.264）](http://blog.csdn.net/leixiaohua1020/article/details/25430425)

avcodec\_find\_decoder()函数最典型的例子可以参考：

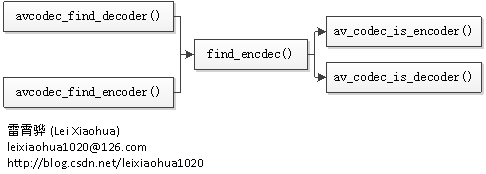
[最简单的基于FFMPEG+SDL的视频播放器 ver2 （采用SDL2.0）](http://blog.csdn.net/leixiaohua1020/article/details/38868499)

其实这两个函数的实质就是遍历AVCodec链表并且获得符合条件的元素。有关AVCodec链表的建立可以参考文章：

[ffmpeg 源代码简单分析 ： av\_register\_all()](http://blog.csdn.net/leixiaohua1020/article/details/12677129)

## 函数调用关系图

avcodec\_find\_encoder()和avcodec\_find\_decoder()的函数调用关系图如下所示。



## avcodec\_find\_encoder()

avcodec\_find\_encoder()的源代码位于libavcodec\utils.c，如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44084557) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44084557)

[在CODE上查看代码片](https://code.csdn.net/snippets/613144)

1. AVCodec \*avcodec\_find\_encoder(**enum** AVCodecID id)
2. {
3. **return** find\_encdec(id, 1);
4. }

从源代码可以看出avcodec\_find\_encoder()调用了一个find\_encdec()，注意它的第二个参数是0。

下面我们看一下find\_encdec()的定义。

## find\_encdec()

find\_encdec()的源代码位于libavcodec\utils.c，如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44084557) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44084557)

[在CODE上查看代码片](https://code.csdn.net/snippets/613144)

1. **static** AVCodec \*first\_avcodec;
3. **static** AVCodec \*find\_encdec(**enum** AVCodecID id, **int** encoder)
4. {
5. AVCodec \*p, \*experimental = NULL;
6. p = first\_avcodec;
7. id= remap\_deprecated\_codec\_id(id);
8. **while** (p) {
9. **if** ((encoder ? av\_codec\_is\_encoder(p) : av\_codec\_is\_decoder(p)) &&
10. p->id == id) {
11. **if** (p->capabilities & CODEC\_CAP\_EXPERIMENTAL && !experimental) {
12. experimental = p;
13. } **else**
14. **return** p;
15. }
16. p = p->next;
17. }
18. **return** experimental;
19. }

find\_encdec()中有一个循环，该循环会遍历AVCodec结构的链表，逐一比较输入的ID和每一个编码器的ID，直到找到ID取值相等的编码器。  
在这里有几点需要注意：  
（1）first\_avcodec是一个全局变量，存储AVCodec链表的第一个元素。  
（2）remap\_deprecated\_codec\_id()用于将一些过时的编码器ID映射到新的编码器ID。  
（3）函数的第二个参数encoder用于确定查找编码器还是解码器。当该值为1的时候，用于查找编码器，此时会调用av\_codec\_is\_encoder()判断AVCodec是否为编码器；当该值为0的时候，用于查找解码器，此时会调用av\_codec\_is\_decoder()判断AVCodec是否为解码器。

### av\_codec\_is\_encoder()

av\_codec\_is\_encoder()是一个判断AVCodec是否为编码器的函数。如果是编码器，返回非0值，否则返回0。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44084557) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44084557)

[在CODE上查看代码片](https://code.csdn.net/snippets/613144)

1. /\*\*
2. \* @return a non-zero number if codec is an encoder, zero otherwise
3. \*/
4. **int** av\_codec\_is\_encoder(**const** AVCodec \*codec);

av\_codec\_is\_encoder()源代码很简单，如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44084557) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44084557)

[在CODE上查看代码片](https://code.csdn.net/snippets/613144)

1. **int** av\_codec\_is\_encoder(**const** AVCodec \*codec)
2. {
3. **return** codec && (codec->encode\_sub || codec->encode2);
4. }

从源代码可以看出，av\_codec\_is\_encoder()判断了一下AVCodec是否包含了encode2()或者encode\_sub()接口函数。

### av\_codec\_is\_decoder()

av\_codec\_is\_decoder()是一个判断AVCodec是否为解码器的函数。如果是解码器，返回非0值，否则返回0。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44084557) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44084557)

[在CODE上查看代码片](https://code.csdn.net/snippets/613144)

1. /\*\*
2. \* @return a non-zero number if codec is a decoder, zero otherwise
3. \*/
4. **int** av\_codec\_is\_decoder(**const** AVCodec \*codec);

av\_codec\_is\_decoder()源代码也很简单，如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44084557) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44084557)

[在CODE上查看代码片](https://code.csdn.net/snippets/613144)

1. **int** av\_codec\_is\_decoder(**const** AVCodec \*codec)
2. {
3. **return** codec && codec->decode;
4. }

从源代码可以看出，av\_codec\_is\_decoder()判断了一下AVCodec是否包含了decode()接口函数。

## avcodec\_find\_decoder()

avcodec\_find\_decoder()的源代码位于libavcodec\utils.c，如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44084557) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44084557)

[在CODE上查看代码片](https://code.csdn.net/snippets/613144)

1. AVCodec \*avcodec\_find\_decoder(**enum** AVCodecID id)
2. {
3. **return** find\_encdec(id, 0);
4. }

可以看出avcodec\_find\_decoder()同样调用了find\_encdec()，只是第2个参数设置为0。因此不再详细分析。

# [FFmpeg源代码简单分析：avcodec\_open2()](http://blog.csdn.net/leixiaohua1020/article/details/44117891)

本文简单分析FFmpeg的avcodec\_open2()函数。该函数用于初始化一个视音频编解码器的AVCodecContext。avcodec\_open2()的声明位于libavcodec\avcodec.h，如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44117891) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44117891)

[在CODE上查看代码片](https://code.csdn.net/snippets/614179)

1. /\*\*
2. \* Initialize the AVCodecContext to use the given AVCodec. Prior to using this
3. \* function the context has to be allocated with avcodec\_alloc\_context3().
4. \*
5. \* The functions avcodec\_find\_decoder\_by\_name(), avcodec\_find\_encoder\_by\_name(),
6. \* avcodec\_find\_decoder() and avcodec\_find\_encoder() provide an easy way for
7. \* retrieving a codec.
8. \*
9. \* @warning This function is not thread safe!
10. \*
11. \* @code
12. \* avcodec\_register\_all();
13. \* av\_dict\_set(&opts, "b", "2.5M", 0);
14. \* codec = avcodec\_find\_decoder(AV\_CODEC\_ID\_H264);
15. \* if (!codec)
16. \*     exit(1);
17. \*
18. \* context = avcodec\_alloc\_context3(codec);
19. \*
20. \* if (avcodec\_open2(context, codec, opts) < 0)
21. \*     exit(1);
22. \* @endcode
23. \*
24. \* @param avctx The context to initialize.
25. \* @param codec The codec to open this context for. If a non-NULL codec has been
26. \*              previously passed to avcodec\_alloc\_context3() or
27. \*              avcodec\_get\_context\_defaults3() for this context, then this
28. \*              parameter MUST be either NULL or equal to the previously passed
29. \*              codec.
30. \* @param options A dictionary filled with AVCodecContext and codec-private options.
31. \*                On return this object will be filled with options that were not found.
32. \*
33. \* @return zero on success, a negative value on error
34. \* @see avcodec\_alloc\_context3(), avcodec\_find\_decoder(), avcodec\_find\_encoder(),
35. \*      av\_dict\_set(), av\_opt\_find().
36. \*/
37. **int** avcodec\_open2(AVCodecContext \*avctx, **const** AVCodec \*codec, AVDictionary \*\*options);

用中文简单转述一下avcodec\_open2()各个参数的含义：

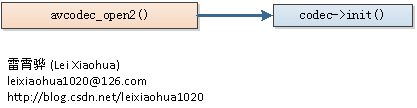
avctx：需要初始化的AVCodecContext。  
codec：输入的AVCodec  
options：一些选项。例如使用libx264编码的时候，“preset”，“tune”等都可以通过该参数设置。

该函数最典型的例子可以参考：

[最简单的基于FFMPEG的视频编码器（YUV编码为H.264）](http://blog.csdn.net/leixiaohua1020/article/details/25430425)

## 函数调用关系图

avcodec\_open2()函数调用关系非常简单，如下图所示。



## avcodec\_open2()

avcodec\_open2()的定义位于libavcodec\utils.c，如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44117891) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44117891)

[在CODE上查看代码片](https://code.csdn.net/snippets/614179)

1. **int** avcodec\_open2(AVCodecContext \*avctx, **const** AVCodec \*codec, AVDictionary \*\*options)
2. {
3. **int** ret = 0;
4. AVDictionary \*tmp = NULL;
5. //如果已经打开，直接返回
6. **if** (avcodec\_is\_open(avctx))
7. **return** 0;
9. **if** ((!codec && !avctx->codec)) {
10. av\_log(avctx, AV\_LOG\_ERROR, "No codec provided to avcodec\_open2()\n");
11. **return** AVERROR(EINVAL);
12. }
13. **if** ((codec && avctx->codec && codec != avctx->codec)) {
14. av\_log(avctx, AV\_LOG\_ERROR, "This AVCodecContext was allocated for %s, "
15. "but %s passed to avcodec\_open2()\n", avctx->codec->name, codec->name);
16. **return** AVERROR(EINVAL);
17. }
18. **if** (!codec)
19. codec = avctx->codec;
21. **if** (avctx->extradata\_size < 0 || avctx->extradata\_size >= FF\_MAX\_EXTRADATA\_SIZE)
22. **return** AVERROR(EINVAL);
24. **if** (options)
25. av\_dict\_copy(&tmp, \*options, 0);
27. ret = ff\_lock\_avcodec(avctx);
28. **if** (ret < 0)
29. **return** ret;
30. //各种Malloc
31. avctx->internal = av\_mallocz(**sizeof**(AVCodecInternal));
32. **if** (!avctx->internal) {
33. ret = AVERROR(ENOMEM);
34. **goto** end;
35. }
37. avctx->internal->pool = av\_mallocz(**sizeof**(\*avctx->internal->pool));
38. **if** (!avctx->internal->pool) {
39. ret = AVERROR(ENOMEM);
40. **goto** free\_and\_end;
41. }
43. avctx->internal->to\_free = av\_frame\_alloc();
44. **if** (!avctx->internal->to\_free) {
45. ret = AVERROR(ENOMEM);
46. **goto** free\_and\_end;
47. }
49. **if** (codec->priv\_data\_size > 0) {
50. **if** (!avctx->priv\_data) {
51. avctx->priv\_data = av\_mallocz(codec->priv\_data\_size);
52. **if** (!avctx->priv\_data) {
53. ret = AVERROR(ENOMEM);
54. **goto** end;
55. }
56. **if** (codec->priv\_class) {
57. \*(**const** AVClass \*\*)avctx->priv\_data = codec->priv\_class;
58. av\_opt\_set\_defaults(avctx->priv\_data);
59. }
60. }
61. **if** (codec->priv\_class && (ret = av\_opt\_set\_dict(avctx->priv\_data, &tmp)) < 0)
62. **goto** free\_and\_end;
63. } **else** {
64. avctx->priv\_data = NULL;
65. }
66. //将输入的AVDictionary形式的选项设置到AVCodecContext
67. **if** ((ret = av\_opt\_set\_dict(avctx, &tmp)) < 0)
68. **goto** free\_and\_end;
70. **if** (avctx->codec\_whitelist && av\_match\_list(codec->name, avctx->codec\_whitelist, ',') <= 0) {
71. av\_log(avctx, AV\_LOG\_ERROR, "Codec (%s) not on whitelist\n", codec->name);
72. ret = AVERROR(EINVAL);
73. **goto** free\_and\_end;
74. }
76. // only call ff\_set\_dimensions() for non H.264/VP6F codecs so as not to overwrite previously setup dimensions
77. **if** (!(avctx->coded\_width && avctx->coded\_height && avctx->width && avctx->height &&
78. (avctx->codec\_id == AV\_CODEC\_ID\_H264 || avctx->codec\_id == AV\_CODEC\_ID\_VP6F))) {
79. **if** (avctx->coded\_width && avctx->coded\_height)
80. ret = ff\_set\_dimensions(avctx, avctx->coded\_width, avctx->coded\_height);
81. **else** **if** (avctx->width && avctx->height)
82. ret = ff\_set\_dimensions(avctx, avctx->width, avctx->height);
83. **if** (ret < 0)
84. **goto** free\_and\_end;
85. }
86. //检查宽和高
87. **if** ((avctx->coded\_width || avctx->coded\_height || avctx->width || avctx->height)
88. && (  av\_image\_check\_size(avctx->coded\_width, avctx->coded\_height, 0, avctx) < 0
89. || av\_image\_check\_size(avctx->width,       avctx->height,       0, avctx) < 0)) {
90. av\_log(avctx, AV\_LOG\_WARNING, "Ignoring invalid width/height values\n");
91. ff\_set\_dimensions(avctx, 0, 0);
92. }
93. //检查宽高比
94. **if** (avctx->width > 0 && avctx->height > 0) {
95. **if** (av\_image\_check\_sar(avctx->width, avctx->height,
96. avctx->sample\_aspect\_ratio) < 0) {
97. av\_log(avctx, AV\_LOG\_WARNING, "ignoring invalid SAR: %u/%u\n",
98. avctx->sample\_aspect\_ratio.num,
99. avctx->sample\_aspect\_ratio.den);
100. avctx->sample\_aspect\_ratio = (AVRational){ 0, 1 };
101. }
102. }
104. /\* if the decoder init function was already called previously,
105. \* free the already allocated subtitle\_header before overwriting it \*/
106. **if** (av\_codec\_is\_decoder(codec))
107. av\_freep(&avctx->subtitle\_header);
109. **if** (avctx->channels > FF\_SANE\_NB\_CHANNELS) {
110. ret = AVERROR(EINVAL);
111. **goto** free\_and\_end;
112. }
114. avctx->codec = codec;
115. **if** ((avctx->codec\_type == AVMEDIA\_TYPE\_UNKNOWN || avctx->codec\_type == codec->type) &&
116. avctx->codec\_id == AV\_CODEC\_ID\_NONE) {
117. avctx->codec\_type = codec->type;
118. avctx->codec\_id   = codec->id;
119. }
120. **if** (avctx->codec\_id != codec->id || (avctx->codec\_type != codec->type
121. && avctx->codec\_type != AVMEDIA\_TYPE\_ATTACHMENT)) {
122. av\_log(avctx, AV\_LOG\_ERROR, "Codec type or id mismatches\n");
123. ret = AVERROR(EINVAL);
124. **goto** free\_and\_end;
125. }
126. avctx->frame\_number = 0;
127. avctx->codec\_descriptor = avcodec\_descriptor\_get(avctx->codec\_id);
128. //检查编码器是否出于“实验”阶段
129. **if** (avctx->codec->capabilities & CODEC\_CAP\_EXPERIMENTAL &&
130. avctx->strict\_std\_compliance > FF\_COMPLIANCE\_EXPERIMENTAL) {
131. **const** **char** \*codec\_string = av\_codec\_is\_encoder(codec) ? "encoder" : "decoder";
132. AVCodec \*codec2;
133. av\_log(avctx, AV\_LOG\_ERROR,
134. "The %s '%s' is experimental but experimental codecs are not enabled, "
135. "add '-strict %d' if you want to use it.\n",
136. codec\_string, codec->name, FF\_COMPLIANCE\_EXPERIMENTAL);
137. codec2 = av\_codec\_is\_encoder(codec) ? avcodec\_find\_encoder(codec->id) : avcodec\_find\_decoder(codec->id);
138. **if** (!(codec2->capabilities & CODEC\_CAP\_EXPERIMENTAL))
139. av\_log(avctx, AV\_LOG\_ERROR, "Alternatively use the non experimental %s '%s'.\n",
140. codec\_string, codec2->name);
141. ret = AVERROR\_EXPERIMENTAL;
142. **goto** free\_and\_end;
143. }
145. **if** (avctx->codec\_type == AVMEDIA\_TYPE\_AUDIO &&
146. (!avctx->time\_base.num || !avctx->time\_base.den)) {
147. avctx->time\_base.num = 1;
148. avctx->time\_base.den = avctx->sample\_rate;
149. }
151. **if** (!HAVE\_THREADS)
152. av\_log(avctx, AV\_LOG\_WARNING, "Warning: not compiled with thread support, using thread emulation\n");
154. **if** (CONFIG\_FRAME\_THREAD\_ENCODER) {
155. ff\_unlock\_avcodec(); //we will instanciate a few encoders thus kick the counter to prevent false detection of a problem
156. ret = ff\_frame\_thread\_encoder\_init(avctx, options ? \*options : NULL);
157. ff\_lock\_avcodec(avctx);
158. **if** (ret < 0)
159. **goto** free\_and\_end;
160. }
162. **if** (HAVE\_THREADS
163. && !(avctx->internal->frame\_thread\_encoder && (avctx->active\_thread\_type&FF\_THREAD\_FRAME))) {
164. ret = ff\_thread\_init(avctx);
165. **if** (ret < 0) {
166. **goto** free\_and\_end;
167. }
168. }
169. **if** (!HAVE\_THREADS && !(codec->capabilities & CODEC\_CAP\_AUTO\_THREADS))
170. avctx->thread\_count = 1;
172. **if** (avctx->codec->max\_lowres < avctx->lowres || avctx->lowres < 0) {
173. av\_log(avctx, AV\_LOG\_ERROR, "The maximum value for lowres supported by the decoder is %d\n",
174. avctx->codec->max\_lowres);
175. ret = AVERROR(EINVAL);
176. **goto** free\_and\_end;
177. }
179. #if FF\_API\_VISMV
180. **if** (avctx->debug\_mv)
181. av\_log(avctx, AV\_LOG\_WARNING, "The 'vismv' option is deprecated, "
182. "see the codecview filter instead.\n");
183. #endif
184. //检查输入参数是否符合【编码器】要求
185. **if** (av\_codec\_is\_encoder(avctx->codec)) {
186. **int** i;
187. //如果包含采样率参数（表明是音频），检查采样率是否符合要求
188. **if** (avctx->codec->sample\_fmts) {
189. //遍历编码器支持的所有采样率
190. **for** (i = 0; avctx->codec->sample\_fmts[i] != AV\_SAMPLE\_FMT\_NONE; i++) {
191. //如果设置的采样率==编码器支持的采样率，跳出循环。
192. **if** (avctx->sample\_fmt == avctx->codec->sample\_fmts[i])
193. **break**;
194. **if** (avctx->channels == 1 &&
195. av\_get\_planar\_sample\_fmt(avctx->sample\_fmt) ==
196. av\_get\_planar\_sample\_fmt(avctx->codec->sample\_fmts[i])) {
197. avctx->sample\_fmt = avctx->codec->sample\_fmts[i];
198. **break**;
199. }
200. }
201. //再检查一下采样率取值是否正确
202. //注意，此时的i值没有变化
203. **if** (avctx->codec->sample\_fmts[i] == AV\_SAMPLE\_FMT\_NONE) {
204. **char** buf[128];
205. snprintf(buf, **sizeof**(buf), "%d", avctx->sample\_fmt);
206. av\_log(avctx, AV\_LOG\_ERROR, "Specified sample format %s is invalid or not supported\n",
207. (**char** \*)av\_x\_if\_null(av\_get\_sample\_fmt\_name(avctx->sample\_fmt), buf));
208. ret = AVERROR(EINVAL);
209. **goto** free\_and\_end;
210. }
211. }
212. //检查像素格式
213. **if** (avctx->codec->pix\_fmts) {
214. **for** (i = 0; avctx->codec->pix\_fmts[i] != AV\_PIX\_FMT\_NONE; i++)
215. **if** (avctx->pix\_fmt == avctx->codec->pix\_fmts[i])
216. **break**;
217. **if** (avctx->codec->pix\_fmts[i] == AV\_PIX\_FMT\_NONE
218. && !((avctx->codec\_id == AV\_CODEC\_ID\_MJPEG || avctx->codec\_id == AV\_CODEC\_ID\_LJPEG)
219. && avctx->strict\_std\_compliance <= FF\_COMPLIANCE\_UNOFFICIAL)) {
220. **char** buf[128];
221. snprintf(buf, **sizeof**(buf), "%d", avctx->pix\_fmt);
222. av\_log(avctx, AV\_LOG\_ERROR, "Specified pixel format %s is invalid or not supported\n",
223. (**char** \*)av\_x\_if\_null(av\_get\_pix\_fmt\_name(avctx->pix\_fmt), buf));
224. ret = AVERROR(EINVAL);
225. **goto** free\_and\_end;
226. }
227. **if** (avctx->codec->pix\_fmts[i] == AV\_PIX\_FMT\_YUVJ420P ||
228. avctx->codec->pix\_fmts[i] == AV\_PIX\_FMT\_YUVJ411P ||
229. avctx->codec->pix\_fmts[i] == AV\_PIX\_FMT\_YUVJ422P ||
230. avctx->codec->pix\_fmts[i] == AV\_PIX\_FMT\_YUVJ440P ||
231. avctx->codec->pix\_fmts[i] == AV\_PIX\_FMT\_YUVJ444P)
232. avctx->color\_range = AVCOL\_RANGE\_JPEG;
233. }
234. //检查采样率
235. **if** (avctx->codec->supported\_samplerates) {
236. **for** (i = 0; avctx->codec->supported\_samplerates[i] != 0; i++)
237. **if** (avctx->sample\_rate == avctx->codec->supported\_samplerates[i])
238. **break**;
239. **if** (avctx->codec->supported\_samplerates[i] == 0) {
240. av\_log(avctx, AV\_LOG\_ERROR, "Specified sample rate %d is not supported\n",
241. avctx->sample\_rate);
242. ret = AVERROR(EINVAL);
243. **goto** free\_and\_end;
244. }
245. }
246. //检查声道布局
247. **if** (avctx->codec->channel\_layouts) {
248. **if** (!avctx->channel\_layout) {
249. av\_log(avctx, AV\_LOG\_WARNING, "Channel layout not specified\n");
250. } **else** {
251. **for** (i = 0; avctx->codec->channel\_layouts[i] != 0; i++)
252. **if** (avctx->channel\_layout == avctx->codec->channel\_layouts[i])
253. **break**;
254. **if** (avctx->codec->channel\_layouts[i] == 0) {
255. **char** buf[512];
256. av\_get\_channel\_layout\_string(buf, **sizeof**(buf), -1, avctx->channel\_layout);
257. av\_log(avctx, AV\_LOG\_ERROR, "Specified channel layout '%s' is not supported\n", buf);
258. ret = AVERROR(EINVAL);
259. **goto** free\_and\_end;
260. }
261. }
262. }
263. //检查声道数
264. **if** (avctx->channel\_layout && avctx->channels) {
265. **int** channels = av\_get\_channel\_layout\_nb\_channels(avctx->channel\_layout);
266. **if** (channels != avctx->channels) {
267. **char** buf[512];
268. av\_get\_channel\_layout\_string(buf, **sizeof**(buf), -1, avctx->channel\_layout);
269. av\_log(avctx, AV\_LOG\_ERROR,
270. "Channel layout '%s' with %d channels does not match number of specified channels %d\n",
271. buf, channels, avctx->channels);
272. ret = AVERROR(EINVAL);
273. **goto** free\_and\_end;
274. }
275. } **else** **if** (avctx->channel\_layout) {
276. avctx->channels = av\_get\_channel\_layout\_nb\_channels(avctx->channel\_layout);
277. }
278. //检查宽高
279. **if**(avctx->codec\_type == AVMEDIA\_TYPE\_VIDEO) {
280. **if** (avctx->width <= 0 || avctx->height <= 0) {
281. av\_log(avctx, AV\_LOG\_ERROR, "dimensions not set\n");
282. ret = AVERROR(EINVAL);
283. **goto** free\_and\_end;
284. }
285. }
286. //检查码率
287. **if** (   (avctx->codec\_type == AVMEDIA\_TYPE\_VIDEO || avctx->codec\_type == AVMEDIA\_TYPE\_AUDIO)
288. && avctx->bit\_rate>0 && avctx->bit\_rate<1000) {
289. av\_log(avctx, AV\_LOG\_WARNING, "Bitrate %d is extremely low, maybe you mean %dk\n", avctx->bit\_rate, avctx->bit\_rate);
290. }
292. **if** (!avctx->rc\_initial\_buffer\_occupancy)
293. avctx->rc\_initial\_buffer\_occupancy = avctx->rc\_buffer\_size \* 3 / 4;
294. }
296. avctx->pts\_correction\_num\_faulty\_pts =
297. avctx->pts\_correction\_num\_faulty\_dts = 0;
298. avctx->pts\_correction\_last\_pts =
299. avctx->pts\_correction\_last\_dts = INT64\_MIN;
300. //关键：
301. //一切检查都无误之后，调用编解码器初始化函数
302. **if** (   avctx->codec->init && (!(avctx->active\_thread\_type&FF\_THREAD\_FRAME)
303. || avctx->internal->frame\_thread\_encoder)) {
304. ret = avctx->codec->init(avctx);
305. **if** (ret < 0) {
306. **goto** free\_and\_end;
307. }
308. }
310. ret=0;
312. #if FF\_API\_AUDIOENC\_DELAY
313. **if** (av\_codec\_is\_encoder(avctx->codec))
314. avctx->delay = avctx->initial\_padding;
315. #endif
317. //【解码器】
318. //解码器的参数大部分都是由系统自动设定而不是由用户设定，因而不怎么需要检查
319. **if** (av\_codec\_is\_decoder(avctx->codec)) {
320. **if** (!avctx->bit\_rate)
321. avctx->bit\_rate = get\_bit\_rate(avctx);
322. /\* validate channel layout from the decoder \*/
323. **if** (avctx->channel\_layout) {
324. **int** channels = av\_get\_channel\_layout\_nb\_channels(avctx->channel\_layout);
325. **if** (!avctx->channels)
326. avctx->channels = channels;
327. **else** **if** (channels != avctx->channels) {
328. **char** buf[512];
329. av\_get\_channel\_layout\_string(buf, **sizeof**(buf), -1, avctx->channel\_layout);
330. av\_log(avctx, AV\_LOG\_WARNING,
331. "Channel layout '%s' with %d channels does not match specified number of channels %d: "
332. "ignoring specified channel layout\n",
333. buf, channels, avctx->channels);
334. avctx->channel\_layout = 0;
335. }
336. }
338. **if** (avctx->channels && avctx->channels < 0 ||
339. avctx->channels > FF\_SANE\_NB\_CHANNELS) {
340. ret = AVERROR(EINVAL);
341. **goto** free\_and\_end;
342. }
343. **if** (avctx->sub\_charenc) {
344. **if** (avctx->codec\_type != AVMEDIA\_TYPE\_SUBTITLE) {
345. av\_log(avctx, AV\_LOG\_ERROR, "Character encoding is only "
346. "supported with subtitles codecs\n");
347. ret = AVERROR(EINVAL);
348. **goto** free\_and\_end;
349. } **else** **if** (avctx->codec\_descriptor->props & AV\_CODEC\_PROP\_BITMAP\_SUB) {
350. av\_log(avctx, AV\_LOG\_WARNING, "Codec '%s' is bitmap-based, "
351. "subtitles character encoding will be ignored\n",
352. avctx->codec\_descriptor->name);
353. avctx->sub\_charenc\_mode = FF\_SUB\_CHARENC\_MODE\_DO\_NOTHING;
354. } **else** {
355. /\* input character encoding is set for a text based subtitle
356. \* codec at this point \*/
357. **if** (avctx->sub\_charenc\_mode == FF\_SUB\_CHARENC\_MODE\_AUTOMATIC)
358. avctx->sub\_charenc\_mode = FF\_SUB\_CHARENC\_MODE\_PRE\_DECODER;
360. **if** (avctx->sub\_charenc\_mode == FF\_SUB\_CHARENC\_MODE\_PRE\_DECODER) {
361. #if CONFIG\_ICONV
362. iconv\_t cd = iconv\_open("UTF-8", avctx->sub\_charenc);
363. **if** (cd == (iconv\_t)-1) {
364. ret = AVERROR(errno);
365. av\_log(avctx, AV\_LOG\_ERROR, "Unable to open iconv context "
366. "with input character encoding \"%s\"\n", avctx->sub\_charenc);
367. **goto** free\_and\_end;
368. }
369. iconv\_close(cd);
370. #else
371. av\_log(avctx, AV\_LOG\_ERROR, "Character encoding subtitles "
372. "conversion needs a libavcodec built with iconv support "
373. "for this codec\n");
374. ret = AVERROR(ENOSYS);
375. **goto** free\_and\_end;
376. #endif
377. }
378. }
379. }
381. #if FF\_API\_AVCTX\_TIMEBASE
382. **if** (avctx->framerate.num > 0 && avctx->framerate.den > 0)
383. avctx->time\_base = av\_inv\_q(av\_mul\_q(avctx->framerate, (AVRational){avctx->ticks\_per\_frame, 1}));
384. #endif
385. }
386. end:
387. ff\_unlock\_avcodec();
388. **if** (options) {
389. av\_dict\_free(options);
390. \*options = tmp;
391. }
393. **return** ret;
394. free\_and\_end:
395. av\_dict\_free(&tmp);
396. **if** (codec->priv\_class && codec->priv\_data\_size)
397. av\_opt\_free(avctx->priv\_data);
398. av\_freep(&avctx->priv\_data);
399. **if** (avctx->internal) {
400. av\_frame\_free(&avctx->internal->to\_free);
401. av\_freep(&avctx->internal->pool);
402. }
403. av\_freep(&avctx->internal);
404. avctx->codec = NULL;
405. **goto** end;
406. }

avcodec\_open2()的源代码量是非常长的，但是它的调用关系非常简单——它只调用了一个关键的函数，即AVCodec的init()，后文将会对这个函数进行分析。  
我们可以简单梳理一下avcodec\_open2()所做的工作，如下所列：

（1）为各种结构体分配内存（通过各种av\_malloc()实现）。  
（2）将输入的AVDictionary形式的选项设置到AVCodecContext。  
（3）其他一些零零碎碎的检查，比如说检查编解码器是否处于“实验”阶段。  
（4）如果是编码器，检查输入参数是否符合编码器的要求  
（5）调用AVCodec的init()初始化具体的解码器。

前几步比较简单，不再分析。在这里我们分析一下第4步和第5步。

## 检查输入参数是否符合编码器要求

在这里简单分析一下第4步，即“检查输入参数是否符合编码器的要求”。这一步中检查了很多的参数，在这里我们随便选一个参数pix\_fmts（像素格式）看一下，如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44117891) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44117891)

[在CODE上查看代码片](https://code.csdn.net/snippets/614179)

1. //检查像素格式
2. **if** (avctx->codec->pix\_fmts) {
3. **for** (i = 0; avctx->codec->pix\_fmts[i] != AV\_PIX\_FMT\_NONE; i++)
4. **if** (avctx->pix\_fmt == avctx->codec->pix\_fmts[i])
5. **break**;
6. **if** (avctx->codec->pix\_fmts[i] == AV\_PIX\_FMT\_NONE
7. && !((avctx->codec\_id == AV\_CODEC\_ID\_MJPEG || avctx->codec\_id == AV\_CODEC\_ID\_LJPEG)
8. && avctx->strict\_std\_compliance <= FF\_COMPLIANCE\_UNOFFICIAL)) {
9. **char** buf[128];
10. snprintf(buf, **sizeof**(buf), "%d", avctx->pix\_fmt);
11. av\_log(avctx, AV\_LOG\_ERROR, "Specified pixel format %s is invalid or not supported\n",
12. (**char** \*)av\_x\_if\_null(av\_get\_pix\_fmt\_name(avctx->pix\_fmt), buf));
13. ret = AVERROR(EINVAL);
14. **goto** free\_and\_end;
15. }
16. **if** (avctx->codec->pix\_fmts[i] == AV\_PIX\_FMT\_YUVJ420P ||
17. avctx->codec->pix\_fmts[i] == AV\_PIX\_FMT\_YUVJ411P ||
18. avctx->codec->pix\_fmts[i] == AV\_PIX\_FMT\_YUVJ422P ||
19. avctx->codec->pix\_fmts[i] == AV\_PIX\_FMT\_YUVJ440P ||
20. avctx->codec->pix\_fmts[i] == AV\_PIX\_FMT\_YUVJ444P)
21. avctx->color\_range = AVCOL\_RANGE\_JPEG;
22. }

可以看出，该代码首先进入了一个for()循环，将AVCodecContext中设定的pix\_fmt与编码器AVCodec中的pix\_fmts数组中的元素逐一比较。  
先简单介绍一下AVCodec中的pix\_fmts数组。AVCodec中的pix\_fmts数组存储了该种编码器支持的像素格式，并且规定以AV\_PIX\_FMT\_NONE（AV\_PIX\_FMT\_NONE取值为-1）为结尾。例如，libx264的pix\_fmts数组的定义位于libavcodec\libx264.c，如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44117891) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44117891)

[在CODE上查看代码片](https://code.csdn.net/snippets/614179)

1. **static** **const** **enum** AVPixelFormat pix\_fmts\_8bit[] = {
2. AV\_PIX\_FMT\_YUV420P,
3. AV\_PIX\_FMT\_YUVJ420P,
4. AV\_PIX\_FMT\_YUV422P,
5. AV\_PIX\_FMT\_YUVJ422P,
6. AV\_PIX\_FMT\_YUV444P,
7. AV\_PIX\_FMT\_YUVJ444P,
8. AV\_PIX\_FMT\_NV12,
9. AV\_PIX\_FMT\_NV16,
10. AV\_PIX\_FMT\_NONE
11. };

从pix\_fmts\_8bit的定义可以看出libx264主要支持的是以YUV为主的像素格式。  
现在回到“检查输入pix\_fmt是否符合编码器的要求”的那段代码。如果for()循环从AVCodec->pix\_fmts数组中找到了符合AVCodecContext->pix\_fmt的像素格式，或者完成了AVCodec->pix\_fmts数组的遍历，都会跳出循环。如果发现AVCodec->pix\_fmts数组中索引为i的元素是AV\_PIX\_FMT\_NONE（即最后一个元素，取值为-1）的时候，就认为没有找到合适的像素格式，并且最终提示错误信息。

## AVCodec->init()

avcodec\_open2()中最关键的一步就是调用AVCodec的init()方法初始化具体的编码器。AVCodec的init()是一个函数指针，指向具体编解码器中的初始化函数。这里我们以libx264为例，看一下它对应的AVCodec的定义。libx264对应的AVCodec的定义位于libavcodec\libx264.c，如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44117891) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44117891)

[在CODE上查看代码片](https://code.csdn.net/snippets/614179)

1. AVCodec ff\_libx264\_encoder = {
2. .name             = "libx264",
3. .long\_name        = NULL\_IF\_CONFIG\_SMALL("libx264 H.264 / AVC / MPEG-4 AVC / MPEG-4 part 10"),
4. .type             = AVMEDIA\_TYPE\_VIDEO,
5. .id               = AV\_CODEC\_ID\_H264,
6. .priv\_data\_size   = **sizeof**(X264Context),
7. .init             = X264\_init,
8. .encode2          = X264\_frame,
9. .close            = X264\_close,
10. .capabilities     = CODEC\_CAP\_DELAY | CODEC\_CAP\_AUTO\_THREADS,
11. .priv\_class       = &x264\_class,
12. .defaults         = x264\_defaults,
13. .init\_static\_data = X264\_init\_static,
14. };

可以看出在ff\_libx264\_encoder中init()指向X264\_init()。X264\_init()的定义同样位于libavcodec\libx264.c，如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44117891) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44117891)

[在CODE上查看代码片](https://code.csdn.net/snippets/614179)

1. **static** av\_cold **int** X264\_init(AVCodecContext \*avctx)
2. {
3. X264Context \*x4 = avctx->priv\_data;
4. **int** sw,sh;
6. **if** (avctx->global\_quality > 0)
7. av\_log(avctx, AV\_LOG\_WARNING, "-qscale is ignored, -crf is recommended.\n");
9. x264\_param\_default(&x4->params);
11. x4->params.b\_deblocking\_filter         = avctx->flags & CODEC\_FLAG\_LOOP\_FILTER;
13. **if** (x4->preset || x4->tune)
14. **if** (x264\_param\_default\_preset(&x4->params, x4->preset, x4->tune) < 0) {
15. **int** i;
16. av\_log(avctx, AV\_LOG\_ERROR, "Error setting preset/tune %s/%s.\n", x4->preset, x4->tune);
17. av\_log(avctx, AV\_LOG\_INFO, "Possible presets:");
18. **for** (i = 0; x264\_preset\_names[i]; i++)
19. av\_log(avctx, AV\_LOG\_INFO, " %s", x264\_preset\_names[i]);
20. av\_log(avctx, AV\_LOG\_INFO, "\n");
21. av\_log(avctx, AV\_LOG\_INFO, "Possible tunes:");
22. **for** (i = 0; x264\_tune\_names[i]; i++)
23. av\_log(avctx, AV\_LOG\_INFO, " %s", x264\_tune\_names[i]);
24. av\_log(avctx, AV\_LOG\_INFO, "\n");
25. **return** AVERROR(EINVAL);
26. }
28. **if** (avctx->level > 0)
29. x4->params.i\_level\_idc = avctx->level;
31. x4->params.pf\_log               = X264\_log;
32. x4->params.p\_log\_private        = avctx;
33. x4->params.i\_log\_level          = X264\_LOG\_DEBUG;
34. x4->params.i\_csp                = convert\_pix\_fmt(avctx->pix\_fmt);
36. OPT\_STR("weightp", x4->wpredp);
38. **if** (avctx->bit\_rate) {
39. x4->params.rc.i\_bitrate   = avctx->bit\_rate / 1000;
40. x4->params.rc.i\_rc\_method = X264\_RC\_ABR;
41. }
42. x4->params.rc.i\_vbv\_buffer\_size = avctx->rc\_buffer\_size / 1000;
43. x4->params.rc.i\_vbv\_max\_bitrate = avctx->rc\_max\_rate    / 1000;
44. x4->params.rc.b\_stat\_write      = avctx->flags & CODEC\_FLAG\_PASS1;
45. **if** (avctx->flags & CODEC\_FLAG\_PASS2) {
46. x4->params.rc.b\_stat\_read = 1;
47. } **else** {
48. **if** (x4->crf >= 0) {
49. x4->params.rc.i\_rc\_method   = X264\_RC\_CRF;
50. x4->params.rc.f\_rf\_constant = x4->crf;
51. } **else** **if** (x4->cqp >= 0) {
52. x4->params.rc.i\_rc\_method   = X264\_RC\_CQP;
53. x4->params.rc.i\_qp\_constant = x4->cqp;
54. }
56. **if** (x4->crf\_max >= 0)
57. x4->params.rc.f\_rf\_constant\_max = x4->crf\_max;
58. }
60. **if** (avctx->rc\_buffer\_size && avctx->rc\_initial\_buffer\_occupancy > 0 &&
61. (avctx->rc\_initial\_buffer\_occupancy <= avctx->rc\_buffer\_size)) {
62. x4->params.rc.f\_vbv\_buffer\_init =
63. (**float**)avctx->rc\_initial\_buffer\_occupancy / avctx->rc\_buffer\_size;
64. }
66. OPT\_STR("level", x4->level);
68. **if** (avctx->i\_quant\_factor > 0)
69. x4->params.rc.f\_ip\_factor         = 1 / fabs(avctx->i\_quant\_factor);
70. **if** (avctx->b\_quant\_factor > 0)
71. x4->params.rc.f\_pb\_factor         = avctx->b\_quant\_factor;
72. **if** (avctx->chromaoffset)
73. x4->params.analyse.i\_chroma\_qp\_offset = avctx->chromaoffset;
75. **if** (avctx->me\_method == ME\_EPZS)
76. x4->params.analyse.i\_me\_method = X264\_ME\_DIA;
77. **else** **if** (avctx->me\_method == ME\_HEX)
78. x4->params.analyse.i\_me\_method = X264\_ME\_HEX;
79. **else** **if** (avctx->me\_method == ME\_UMH)
80. x4->params.analyse.i\_me\_method = X264\_ME\_UMH;
81. **else** **if** (avctx->me\_method == ME\_FULL)
82. x4->params.analyse.i\_me\_method = X264\_ME\_ESA;
83. **else** **if** (avctx->me\_method == ME\_TESA)
84. x4->params.analyse.i\_me\_method = X264\_ME\_TESA;
86. **if** (avctx->gop\_size >= 0)
87. x4->params.i\_keyint\_max         = avctx->gop\_size;
88. **if** (avctx->max\_b\_frames >= 0)
89. x4->params.i\_bframe             = avctx->max\_b\_frames;
90. **if** (avctx->scenechange\_threshold >= 0)
91. x4->params.i\_scenecut\_threshold = avctx->scenechange\_threshold;
92. **if** (avctx->qmin >= 0)
93. x4->params.rc.i\_qp\_min          = avctx->qmin;
94. **if** (avctx->qmax >= 0)
95. x4->params.rc.i\_qp\_max          = avctx->qmax;
96. **if** (avctx->max\_qdiff >= 0)
97. x4->params.rc.i\_qp\_step         = avctx->max\_qdiff;
98. **if** (avctx->qblur >= 0)
99. x4->params.rc.f\_qblur           = avctx->qblur;     /\* temporally blur quants \*/
100. **if** (avctx->qcompress >= 0)
101. x4->params.rc.f\_qcompress       = avctx->qcompress; /\* 0.0 => cbr, 1.0 => constant qp \*/
102. **if** (avctx->refs >= 0)
103. x4->params.i\_frame\_reference    = avctx->refs;
104. **else** **if** (x4->level) {
105. **int** i;
106. **int** mbn = FF\_CEIL\_RSHIFT(avctx->width, 4) \* FF\_CEIL\_RSHIFT(avctx->height, 4);
107. **int** level\_id = -1;
108. **char** \*tail;
109. **int** scale = X264\_BUILD < 129 ? 384 : 1;
111. **if** (!strcmp(x4->level, "1b")) {
112. level\_id = 9;
113. } **else** **if** (strlen(x4->level) <= 3){
114. level\_id = av\_strtod(x4->level, &tail) \* 10 + 0.5;
115. **if** (\*tail)
116. level\_id = -1;
117. }
118. **if** (level\_id <= 0)
119. av\_log(avctx, AV\_LOG\_WARNING, "Failed to parse level\n");
121. **for** (i = 0; i<x264\_levels[i].level\_idc; i++)
122. **if** (x264\_levels[i].level\_idc == level\_id)
123. x4->params.i\_frame\_reference = av\_clip(x264\_levels[i].dpb / mbn / scale, 1, x4->params.i\_frame\_reference);
124. }
126. **if** (avctx->trellis >= 0)
127. x4->params.analyse.i\_trellis    = avctx->trellis;
128. **if** (avctx->me\_range >= 0)
129. x4->params.analyse.i\_me\_range   = avctx->me\_range;
130. **if** (avctx->noise\_reduction >= 0)
131. x4->params.analyse.i\_noise\_reduction = avctx->noise\_reduction;
132. **if** (avctx->me\_subpel\_quality >= 0)
133. x4->params.analyse.i\_subpel\_refine   = avctx->me\_subpel\_quality;
134. **if** (avctx->b\_frame\_strategy >= 0)
135. x4->params.i\_bframe\_adaptive = avctx->b\_frame\_strategy;
136. **if** (avctx->keyint\_min >= 0)
137. x4->params.i\_keyint\_min = avctx->keyint\_min;
138. **if** (avctx->coder\_type >= 0)
139. x4->params.b\_cabac = avctx->coder\_type == FF\_CODER\_TYPE\_AC;
140. **if** (avctx->me\_cmp >= 0)
141. x4->params.analyse.b\_chroma\_me = avctx->me\_cmp & FF\_CMP\_CHROMA;
143. **if** (x4->aq\_mode >= 0)
144. x4->params.rc.i\_aq\_mode = x4->aq\_mode;
145. **if** (x4->aq\_strength >= 0)
146. x4->params.rc.f\_aq\_strength = x4->aq\_strength;
147. PARSE\_X264\_OPT("psy-rd", psy\_rd);
148. PARSE\_X264\_OPT("deblock", deblock);
149. PARSE\_X264\_OPT("partitions", partitions);
150. PARSE\_X264\_OPT("stats", stats);
151. **if** (x4->psy >= 0)
152. x4->params.analyse.b\_psy  = x4->psy;
153. **if** (x4->rc\_lookahead >= 0)
154. x4->params.rc.i\_lookahead = x4->rc\_lookahead;
155. **if** (x4->weightp >= 0)
156. x4->params.analyse.i\_weighted\_pred = x4->weightp;
157. **if** (x4->weightb >= 0)
158. x4->params.analyse.b\_weighted\_bipred = x4->weightb;
159. **if** (x4->cplxblur >= 0)
160. x4->params.rc.f\_complexity\_blur = x4->cplxblur;
162. **if** (x4->ssim >= 0)
163. x4->params.analyse.b\_ssim = x4->ssim;
164. **if** (x4->intra\_refresh >= 0)
165. x4->params.b\_intra\_refresh = x4->intra\_refresh;
166. **if** (x4->bluray\_compat >= 0) {
167. x4->params.b\_bluray\_compat = x4->bluray\_compat;
168. x4->params.b\_vfr\_input = 0;
169. }
170. **if** (x4->avcintra\_class >= 0)
171. #if X264\_BUILD >= 142
172. x4->params.i\_avcintra\_class = x4->avcintra\_class;
173. #else
174. av\_log(avctx, AV\_LOG\_ERROR,
175. "x264 too old for AVC Intra, at least version 142 needed\n");
176. #endif
177. **if** (x4->b\_bias != INT\_MIN)
178. x4->params.i\_bframe\_bias              = x4->b\_bias;
179. **if** (x4->b\_pyramid >= 0)
180. x4->params.i\_bframe\_pyramid = x4->b\_pyramid;
181. **if** (x4->mixed\_refs >= 0)
182. x4->params.analyse.b\_mixed\_references = x4->mixed\_refs;
183. **if** (x4->dct8x8 >= 0)
184. x4->params.analyse.b\_transform\_8x8    = x4->dct8x8;
185. **if** (x4->fast\_pskip >= 0)
186. x4->params.analyse.b\_fast\_pskip       = x4->fast\_pskip;
187. **if** (x4->aud >= 0)
188. x4->params.b\_aud                      = x4->aud;
189. **if** (x4->mbtree >= 0)
190. x4->params.rc.b\_mb\_tree               = x4->mbtree;
191. **if** (x4->direct\_pred >= 0)
192. x4->params.analyse.i\_direct\_mv\_pred   = x4->direct\_pred;
194. **if** (x4->slice\_max\_size >= 0)
195. x4->params.i\_slice\_max\_size =  x4->slice\_max\_size;
196. **else** {
197. /\*
198. \* Allow x264 to be instructed through AVCodecContext about the maximum
199. \* size of the RTP payload. For example, this enables the production of
200. \* payload suitable for the H.264 RTP packetization-mode 0 i.e. single
201. \* NAL unit per RTP packet.
202. \*/
203. **if** (avctx->rtp\_payload\_size)
204. x4->params.i\_slice\_max\_size = avctx->rtp\_payload\_size;
205. }
207. **if** (x4->fastfirstpass)
208. x264\_param\_apply\_fastfirstpass(&x4->params);
210. /\* Allow specifying the x264 profile through AVCodecContext. \*/
211. **if** (!x4->profile)
212. **switch** (avctx->profile) {
213. **case** FF\_PROFILE\_H264\_BASELINE:
214. x4->profile = av\_strdup("baseline");
215. **break**;
216. **case** FF\_PROFILE\_H264\_HIGH:
217. x4->profile = av\_strdup("high");
218. **break**;
219. **case** FF\_PROFILE\_H264\_HIGH\_10:
220. x4->profile = av\_strdup("high10");
221. **break**;
222. **case** FF\_PROFILE\_H264\_HIGH\_422:
223. x4->profile = av\_strdup("high422");
224. **break**;
225. **case** FF\_PROFILE\_H264\_HIGH\_444:
226. x4->profile = av\_strdup("high444");
227. **break**;
228. **case** FF\_PROFILE\_H264\_MAIN:
229. x4->profile = av\_strdup("main");
230. **break**;
231. **default**:
232. **break**;
233. }
235. **if** (x4->nal\_hrd >= 0)
236. x4->params.i\_nal\_hrd = x4->nal\_hrd;
238. **if** (x4->profile)
239. **if** (x264\_param\_apply\_profile(&x4->params, x4->profile) < 0) {
240. **int** i;
241. av\_log(avctx, AV\_LOG\_ERROR, "Error setting profile %s.\n", x4->profile);
242. av\_log(avctx, AV\_LOG\_INFO, "Possible profiles:");
243. **for** (i = 0; x264\_profile\_names[i]; i++)
244. av\_log(avctx, AV\_LOG\_INFO, " %s", x264\_profile\_names[i]);
245. av\_log(avctx, AV\_LOG\_INFO, "\n");
246. **return** AVERROR(EINVAL);
247. }
249. x4->params.i\_width          = avctx->width;
250. x4->params.i\_height         = avctx->height;
251. av\_reduce(&sw, &sh, avctx->sample\_aspect\_ratio.num, avctx->sample\_aspect\_ratio.den, 4096);
252. x4->params.vui.i\_sar\_width  = sw;
253. x4->params.vui.i\_sar\_height = sh;
254. x4->params.i\_timebase\_den = avctx->time\_base.den;
255. x4->params.i\_timebase\_num = avctx->time\_base.num;
256. x4->params.i\_fps\_num = avctx->time\_base.den;
257. x4->params.i\_fps\_den = avctx->time\_base.num \* avctx->ticks\_per\_frame;
259. x4->params.analyse.b\_psnr = avctx->flags & CODEC\_FLAG\_PSNR;
261. x4->params.i\_threads      = avctx->thread\_count;
262. **if** (avctx->thread\_type)
263. x4->params.b\_sliced\_threads = avctx->thread\_type == FF\_THREAD\_SLICE;
265. x4->params.b\_interlaced   = avctx->flags & CODEC\_FLAG\_INTERLACED\_DCT;
267. x4->params.b\_open\_gop     = !(avctx->flags & CODEC\_FLAG\_CLOSED\_GOP);
269. x4->params.i\_slice\_count  = avctx->slices;
271. x4->params.vui.b\_fullrange = avctx->pix\_fmt == AV\_PIX\_FMT\_YUVJ420P ||
272. avctx->pix\_fmt == AV\_PIX\_FMT\_YUVJ422P ||
273. avctx->pix\_fmt == AV\_PIX\_FMT\_YUVJ444P ||
274. avctx->color\_range == AVCOL\_RANGE\_JPEG;
276. **if** (avctx->colorspace != AVCOL\_SPC\_UNSPECIFIED)
277. x4->params.vui.i\_colmatrix = avctx->colorspace;
278. **if** (avctx->color\_primaries != AVCOL\_PRI\_UNSPECIFIED)
279. x4->params.vui.i\_colorprim = avctx->color\_primaries;
280. **if** (avctx->color\_trc != AVCOL\_TRC\_UNSPECIFIED)
281. x4->params.vui.i\_transfer  = avctx->color\_trc;
283. **if** (avctx->flags & CODEC\_FLAG\_GLOBAL\_HEADER)
284. x4->params.b\_repeat\_headers = 0;
286. **if**(x4->x264opts){
287. **const** **char** \*p= x4->x264opts;
288. **while**(p){
289. **char** param[256]={0}, val[256]={0};
290. **if**(sscanf(p, "%255[^:=]=%255[^:]", param, val) == 1){
291. OPT\_STR(param, "1");
292. }**else**
293. OPT\_STR(param, val);
294. p= strchr(p, ':');
295. p+=!!p;
296. }
297. }
299. **if** (x4->x264\_params) {
300. AVDictionary \*dict    = NULL;
301. AVDictionaryEntry \*en = NULL;
303. **if** (!av\_dict\_parse\_string(&dict, x4->x264\_params, "=", ":", 0)) {
304. **while** ((en = av\_dict\_get(dict, "", en, AV\_DICT\_IGNORE\_SUFFIX))) {
305. **if** (x264\_param\_parse(&x4->params, en->key, en->value) < 0)
306. av\_log(avctx, AV\_LOG\_WARNING,
307. "Error parsing option '%s = %s'.\n",
308. en->key, en->value);
309. }
311. av\_dict\_free(&dict);
312. }
313. }
315. // update AVCodecContext with x264 parameters
316. avctx->has\_b\_frames = x4->params.i\_bframe ?
317. x4->params.i\_bframe\_pyramid ? 2 : 1 : 0;
318. **if** (avctx->max\_b\_frames < 0)
319. avctx->max\_b\_frames = 0;
321. avctx->bit\_rate = x4->params.rc.i\_bitrate\*1000;
323. x4->enc = x264\_encoder\_open(&x4->params);
324. **if** (!x4->enc)
325. **return** -1;
327. avctx->coded\_frame = av\_frame\_alloc();
328. **if** (!avctx->coded\_frame)
329. **return** AVERROR(ENOMEM);
331. **if** (avctx->flags & CODEC\_FLAG\_GLOBAL\_HEADER) {
332. x264\_nal\_t \*nal;
333. uint8\_t \*p;
334. **int** nnal, s, i;
336. s = x264\_encoder\_headers(x4->enc, &nal, &nnal);
337. avctx->extradata = p = av\_malloc(s);
339. **for** (i = 0; i < nnal; i++) {
340. /\* Don't put the SEI in extradata. \*/
341. **if** (nal[i].i\_type == NAL\_SEI) {
342. av\_log(avctx, AV\_LOG\_INFO, "%s\n", nal[i].p\_payload+25);
343. x4->sei\_size = nal[i].i\_payload;
344. x4->sei      = av\_malloc(x4->sei\_size);
345. memcpy(x4->sei, nal[i].p\_payload, nal[i].i\_payload);
346. **continue**;
347. }
348. memcpy(p, nal[i].p\_payload, nal[i].i\_payload);
349. p += nal[i].i\_payload;
350. }
351. avctx->extradata\_size = p - avctx->extradata;
352. }
354. **return** 0;
355. }

X264\_init()的代码以后研究X264的时候再进行细节的分析，在这里简单记录一下它做的两项工作：

（1）设置X264Context的参数。X264Context主要完成了libx264和FFmpeg对接的功能。可以看出代码主要在设置一个params结构体变量，该变量的类型即是x264中存储参数的结构体x264\_param\_t。  
（2）调用libx264的API进行编码器的初始化工作。例如调用x264\_param\_default()设置默认参数，调用x264\_param\_apply\_profile()设置profile，调用x264\_encoder\_open()打开编码器等等。

最后附上X264Context的定义，位于libavcodec\libx264.c，如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44117891) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44117891)

[在CODE上查看代码片](https://code.csdn.net/snippets/614179)

1. **typedef** **struct** X264Context {
2. AVClass        \***class**;
3. x264\_param\_t    params;
4. x264\_t         \*enc;
5. x264\_picture\_t  pic;
6. uint8\_t        \*sei;
7. **int**             sei\_size;
8. **char** \*preset;
9. **char** \*tune;
10. **char** \*profile;
11. **char** \*level;
12. **int** fastfirstpass;
13. **char** \*wpredp;
14. **char** \*x264opts;
15. **float** crf;
16. **float** crf\_max;
17. **int** cqp;
18. **int** aq\_mode;
19. **float** aq\_strength;
20. **char** \*psy\_rd;
21. **int** psy;
22. **int** rc\_lookahead;
23. **int** weightp;
24. **int** weightb;
25. **int** ssim;
26. **int** intra\_refresh;
27. **int** bluray\_compat;
28. **int** b\_bias;
29. **int** b\_pyramid;
30. **int** mixed\_refs;
31. **int** dct8x8;
32. **int** fast\_pskip;
33. **int** aud;
34. **int** mbtree;
35. **char** \*deblock;
36. **float** cplxblur;
37. **char** \*partitions;
38. **int** direct\_pred;
39. **int** slice\_max\_size;
40. **char** \*stats;
41. **int** nal\_hrd;
42. **int** avcintra\_class;
43. **char** \*x264\_params;
44. } X264Context;

# [FFmpeg源代码简单分析：avcodec\_close()](http://blog.csdn.net/leixiaohua1020/article/details/44206699)

本文简单分析FFmpeg的avcodec\_close()函数。该函数用于关闭编码器。avcodec\_close()函数的声明位于libavcodec\avcodec.h，如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44206699) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44206699)

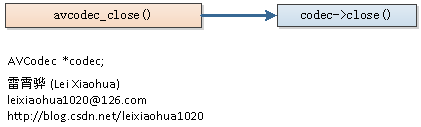
[在CODE上查看代码片](https://code.csdn.net/snippets/617441)

1. /\*\*
2. \* Close a given AVCodecContext and free all the data associated with it
3. \* (but not the AVCodecContext itself).
4. \*
5. \* Calling this function on an AVCodecContext that hasn't been opened will free
6. \* the codec-specific data allocated in avcodec\_alloc\_context3() /
7. \* avcodec\_get\_context\_defaults3() with a non-NULL codec. Subsequent calls will
8. \* do nothing.
9. \*/
10. **int** avcodec\_close(AVCodecContext \*avctx);

该函数只有一个参数，就是需要关闭的编码器的AVCodecContext。

## 函数调用关系图

函数的调用关系图如下所示。



## avcodec\_close()

avcodec\_close()的定义位于libavcodec\utils.c，如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44206699) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44206699)

[在CODE上查看代码片](https://code.csdn.net/snippets/617441)

1. av\_cold **int** avcodec\_close(AVCodecContext \*avctx)
2. {
3. **if** (!avctx)
4. **return** 0;
6. **if** (avcodec\_is\_open(avctx)) {
7. FramePool \*pool = avctx->internal->pool;
8. **int** i;
9. **if** (CONFIG\_FRAME\_THREAD\_ENCODER &&
10. avctx->internal->frame\_thread\_encoder && avctx->thread\_count > 1) {
11. ff\_frame\_thread\_encoder\_free(avctx);
12. }
13. **if** (HAVE\_THREADS && avctx->internal->thread\_ctx)
14. ff\_thread\_free(avctx);
15. //关闭编解码器
16. **if** (avctx->codec && avctx->codec->close)
17. avctx->codec->close(avctx);
18. avctx->coded\_frame = NULL;
19. avctx->internal->byte\_buffer\_size = 0;
20. av\_freep(&avctx->internal->byte\_buffer);
21. av\_frame\_free(&avctx->internal->to\_free);
22. **for** (i = 0; i < FF\_ARRAY\_ELEMS(pool->pools); i++)
23. av\_buffer\_pool\_uninit(&pool->pools[i]);
24. av\_freep(&avctx->internal->pool);
26. **if** (avctx->hwaccel && avctx->hwaccel->uninit)
27. avctx->hwaccel->uninit(avctx);
28. av\_freep(&avctx->internal->hwaccel\_priv\_data);
30. av\_freep(&avctx->internal);
31. }
33. **if** (avctx->priv\_data && avctx->codec && avctx->codec->priv\_class)
34. av\_opt\_free(avctx->priv\_data);
35. av\_opt\_free(avctx);
36. av\_freep(&avctx->priv\_data);
37. **if** (av\_codec\_is\_encoder(avctx->codec))
38. av\_freep(&avctx->extradata);
39. avctx->codec = NULL;
40. avctx->active\_thread\_type = 0;
42. **return** 0;
43. }

从avcodec\_close()的定义可以看出，该函数释放AVCodecContext中有关的变量，并且调用了AVCodec的close()关闭了解码器。

## AVCodec->close()

AVCodec的close()是一个函数指针，指向了特定编码器的关闭函数。在这里我们以libx264为例，看一下它对应的AVCodec的结构体的定义，如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44206699) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44206699)

[在CODE上查看代码片](https://code.csdn.net/snippets/617441)

1. AVCodec ff\_libx264\_encoder = {
2. .name             = "libx264",
3. .long\_name        = NULL\_IF\_CONFIG\_SMALL("libx264 H.264 / AVC / MPEG-4 AVC / MPEG-4 part 10"),
4. .type             = AVMEDIA\_TYPE\_VIDEO,
5. .id               = AV\_CODEC\_ID\_H264,
6. .priv\_data\_size   = **sizeof**(X264Context),
7. .init             = X264\_init,
8. .encode2          = X264\_frame,
9. .close            = X264\_close,
10. .capabilities     = CODEC\_CAP\_DELAY | CODEC\_CAP\_AUTO\_THREADS,
11. .priv\_class       = &x264\_class,
12. .defaults         = x264\_defaults,
13. .init\_static\_data = X264\_init\_static,
14. };

从ff\_libx264\_encoder的定义可以看出：close()函数对应的是X264\_close()函数。继续看一下X264\_close()函数的定义，如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44206699) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44206699)

[在CODE上查看代码片](https://code.csdn.net/snippets/617441)

1. **static** av\_cold **int** X264\_close(AVCodecContext \*avctx)
2. {
3. X264Context \*x4 = avctx->priv\_data;
5. av\_freep(&avctx->extradata);
6. av\_freep(&x4->sei);
7. //关闭编码器
8. **if** (x4->enc)
9. x264\_encoder\_close(x4->enc);
11. av\_frame\_free(&avctx->coded\_frame);
13. **return** 0;
14. }

从X264\_close()的定义可以看出，该函数调用了libx264的x264\_encoder\_close()关闭了libx264编码器。

# [图解FFMPEG打开媒体的函数avformat\_open\_input](http://blog.csdn.net/leixiaohua1020/article/details/8661601)

FFMPEG打开媒体的的过程开始于avformat\_open\_input，因此该函数的重要性不可忽视。

在该函数中，FFMPEG完成了：

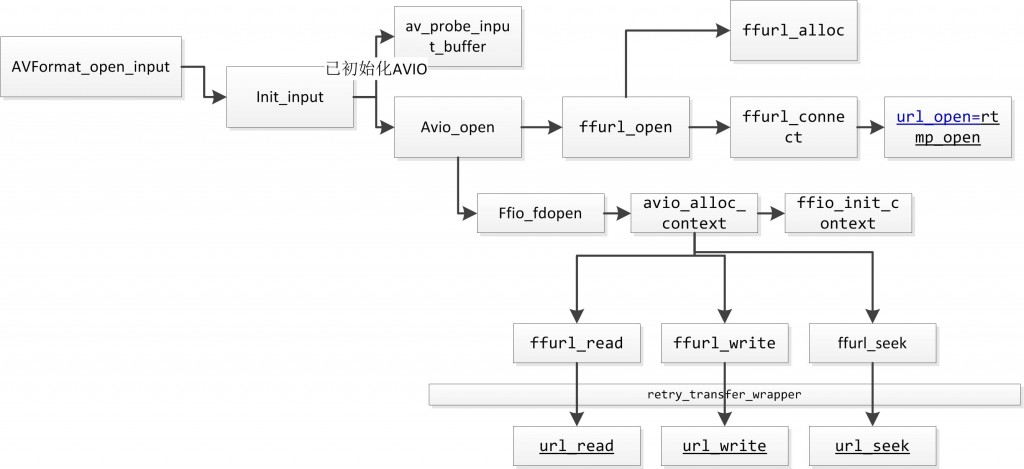
输入输出结构体AVIOContext的初始化；

输入数据的协议（例如RTMP，或者file）的识别（通过一套评分机制）:1判断文件名的后缀 2读取文件头的数据进行比对；

使用获得最高分的文件协议对应的URLProtocol，通过函数指针的方式，与FFMPEG连接（非专业用词）；

剩下的就是调用该URLProtocol的函数进行open,read等操作了

以下是通过eclipse+MinGW调试FFMPEG源代码获得的函数调用关系图



可见最终都调用了URLProtocol结构体中的函数指针。

URLProtocol结构如下，是一大堆函数指针的集合（avio.h文件）

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/8661601) [copy](http://blog.csdn.net/leixiaohua1020/article/details/8661601)

1. **typedef** **struct** URLProtocol {
2. **const** **char** \*name;
3. **int** (\*url\_open)(URLContext \*h, **const** **char** \*url, **int** flags);
4. **int** (\*url\_read)(URLContext \*h, unsigned **char** \*buf, **int** size);
5. **int** (\*url\_write)(URLContext \*h, **const** unsigned **char** \*buf, **int** size);
6. int64\_t (\*url\_seek)(URLContext \*h, int64\_t pos, **int** whence);
7. **int** (\*url\_close)(URLContext \*h);
8. **struct** URLProtocol \*next;
9. **int** (\*url\_read\_pause)(URLContext \*h, **int** pause);
10. int64\_t (\*url\_read\_seek)(URLContext \*h, **int** stream\_index,
11. int64\_t timestamp, **int** flags);
12. **int** (\*url\_get\_file\_handle)(URLContext \*h);
13. **int** priv\_data\_size;
14. **const** AVClass \*priv\_data\_class;
15. **int** flags;
16. **int** (\*url\_check)(URLContext \*h, **int** mask);
17. } URLProtocol;

URLProtocol功能就是完成各种输入协议的读写等操作

但输入协议种类繁多，它是怎样做到“大一统”的呢？

原来，每个具体的输入协议都有自己对应的URLProtocol。

比如file协议（FFMPEG把文件也当做一种特殊的协议）（\*file.c文件）

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/8661601) [copy](http://blog.csdn.net/leixiaohua1020/article/details/8661601)

1. URLProtocol ff\_pipe\_protocol = {
2. .name                = "pipe",
3. .url\_open            = pipe\_open,
4. .url\_read            = file\_read,
5. .url\_write           = file\_write,
6. .url\_get\_file\_handle = file\_get\_handle,
7. .url\_check           = file\_check,
8. };

或者rtmp协议（此处使用了librtmp）（librtmp.c文件）

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/8661601) [copy](http://blog.csdn.net/leixiaohua1020/article/details/8661601)

1. URLProtocol ff\_rtmp\_protocol = {
2. .name                = "rtmp",
3. .url\_open            = rtmp\_open,
4. .url\_read            = rtmp\_read,
5. .url\_write           = rtmp\_write,
6. .url\_close           = rtmp\_close,
7. .url\_read\_pause      = rtmp\_read\_pause,
8. .url\_read\_seek       = rtmp\_read\_seek,
9. .url\_get\_file\_handle = rtmp\_get\_file\_handle,
10. .priv\_data\_size      = **sizeof**(RTMP),
11. .flags               = URL\_PROTOCOL\_FLAG\_NETWORK,
12. };

可见它们把各自的函数指针都赋值给了URLProtocol结构体的函数指针

因此avformat\_open\_input只需调用url\_open,url\_read这些函数就可以完成各种具体输入协议的open,read等操作了

# [FFmpeg源代码简单分析：avformat\_open\_input()](http://blog.csdn.net/leixiaohua1020/article/details/44064715)

本文简单分析FFmpeg中一个常用的函数：avformat\_open\_input()。该函数用于打开多媒体数据并且获得一些相关的信息。它的声明位于libavformat\avformat.h，如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44064715) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44064715)

[在CODE上查看代码片](https://code.csdn.net/snippets/612296)

1. /\*\*
2. \* Open an input stream and read the header. The codecs are not opened.
3. \* The stream must be closed with avformat\_close\_input().
4. \*
5. \* @param ps Pointer to user-supplied AVFormatContext (allocated by avformat\_alloc\_context).
6. \*           May be a pointer to NULL, in which case an AVFormatContext is allocated by this
7. \*           function and written into ps.
8. \*           Note that a user-supplied AVFormatContext will be freed on failure.
9. \* @param filename Name of the stream to open.
10. \* @param fmt If non-NULL, this parameter forces a specific input format.
11. \*            Otherwise the format is autodetected.
12. \* @param options  A dictionary filled with AVFormatContext and demuxer-private options.
13. \*                 On return this parameter will be destroyed and replaced with a dict containing
14. \*                 options that were not found. May be NULL.
15. \*
16. \* @return 0 on success, a negative AVERROR on failure.
17. \*
18. \* @note If you want to use custom IO, preallocate the format context and set its pb field.
19. \*/
20. **int** avformat\_open\_input(AVFormatContext \*\*ps, **const** **char** \*filename, AVInputFormat \*fmt, AVDictionary \*\*options);

代码中的英文注释写的已经比较详细了，在这里拿中文简单叙述一下。

ps：函数调用成功之后处理过的AVFormatContext结构体。  
file：打开的视音频流的URL。  
fmt：强制指定AVFormatContext中AVInputFormat的。这个参数一般情况下可以设置为NULL，这样FFmpeg可以自动检测AVInputFormat。

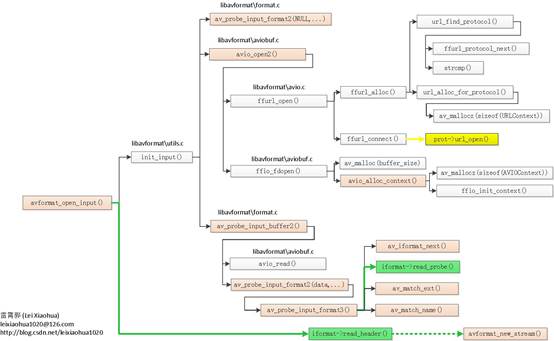
dictionay：附加的一些选项，一般情况下可以设置为NULL。

函数执行成功的话，其返回值大于等于0。  
  
 该函数最典型的例子可以参考：[最简单的基于FFMPEG+SDL的视频播放器 ver2 （采用SDL2.0）](http://blog.csdn.net/leixiaohua1020/article/details/38868499)

此前已经粗略写了1篇关于avformat\_open\_input()的文章《 [图解FFMPEG打开媒体的函数avformat\_open\_input](http://blog.csdn.net/leixiaohua1020/article/details/8661601)》，还转载了一篇注释比较详细的文章《[FFMPEG源码分析：avformat\_open\_input()（媒体打开函数](http://blog.csdn.net/leixiaohua1020/article/details/11885813)）》。但是个人感觉这个函数确实太重要了，可以算作FFmpeg的“灵魂”，所以打算再写一篇文章分析一下它的结构。

## 函数调用关系图

函数调用结构图如下所示。

[](https://my.csdn.net/leixiaohua1020/album/detail/1804831)

[单击查看更清晰的图片](https://my.csdn.net/leixiaohua1020/album/detail/1804831)

## avformat\_open\_input()

下面看一下的定义，位于libavformat\utils.c中，如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44064715) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44064715)

[在CODE上查看代码片](https://code.csdn.net/snippets/612296)

1. **int** avformat\_open\_input(AVFormatContext \*\*ps, **const** **char** \*filename,
2. AVInputFormat \*fmt, AVDictionary \*\*options)
3. {
4. AVFormatContext \*s = \*ps;
5. **int** ret = 0;
6. AVDictionary \*tmp = NULL;
7. ID3v2ExtraMeta \*id3v2\_extra\_meta = NULL;
9. **if** (!s && !(s = avformat\_alloc\_context()))
10. **return** AVERROR(ENOMEM);
11. **if** (!s->av\_class) {
12. av\_log(NULL, AV\_LOG\_ERROR, "Input context has not been properly allocated by avformat\_alloc\_context() and is not NULL either\n");
13. **return** AVERROR(EINVAL);
14. }
15. **if** (fmt)
16. s->iformat = fmt;
18. **if** (options)
19. av\_dict\_copy(&tmp, \*options, 0);
21. **if** ((ret = av\_opt\_set\_dict(s, &tmp)) < 0)
22. **goto** fail;
24. **if** ((ret = init\_input(s, filename, &tmp)) < 0)
25. **goto** fail;
26. s->probe\_score = ret;
28. **if** (s->format\_whitelist && av\_match\_list(s->iformat->name, s->format\_whitelist, ',') <= 0) {
29. av\_log(s, AV\_LOG\_ERROR, "Format not on whitelist\n");
30. ret = AVERROR(EINVAL);
31. **goto** fail;
32. }
34. avio\_skip(s->pb, s->skip\_initial\_bytes);
36. /\* Check filename in case an image number is expected. \*/
37. **if** (s->iformat->flags & AVFMT\_NEEDNUMBER) {
38. **if** (!av\_filename\_number\_test(filename)) {
39. ret = AVERROR(EINVAL);
40. **goto** fail;
41. }
42. }
44. s->duration = s->start\_time = AV\_NOPTS\_VALUE;
45. av\_strlcpy(s->filename, filename ? filename : "", **sizeof**(s->filename));
47. /\* Allocate private data. \*/
48. **if** (s->iformat->priv\_data\_size > 0) {
49. **if** (!(s->priv\_data = av\_mallocz(s->iformat->priv\_data\_size))) {
50. ret = AVERROR(ENOMEM);
51. **goto** fail;
52. }
53. **if** (s->iformat->priv\_class) {
54. \*(**const** AVClass \*\*) s->priv\_data = s->iformat->priv\_class;
55. av\_opt\_set\_defaults(s->priv\_data);
56. **if** ((ret = av\_opt\_set\_dict(s->priv\_data, &tmp)) < 0)
57. **goto** fail;
58. }
59. }
61. /\* e.g. AVFMT\_NOFILE formats will not have a AVIOContext \*/
62. **if** (s->pb)
63. ff\_id3v2\_read(s, ID3v2\_DEFAULT\_MAGIC, &id3v2\_extra\_meta, 0);
65. **if** (!(s->flags&AVFMT\_FLAG\_PRIV\_OPT) && s->iformat->read\_header)
66. **if** ((ret = s->iformat->read\_header(s)) < 0)
67. **goto** fail;
69. **if** (id3v2\_extra\_meta) {
70. **if** (!strcmp(s->iformat->name, "mp3") || !strcmp(s->iformat->name, "aac") ||
71. !strcmp(s->iformat->name, "tta")) {
72. **if** ((ret = ff\_id3v2\_parse\_apic(s, &id3v2\_extra\_meta)) < 0)
73. **goto** fail;
74. } **else**
75. av\_log(s, AV\_LOG\_DEBUG, "demuxer does not support additional id3 data, skipping\n");
76. }
77. ff\_id3v2\_free\_extra\_meta(&id3v2\_extra\_meta);
79. **if** ((ret = avformat\_queue\_attached\_pictures(s)) < 0)
80. **goto** fail;
82. **if** (!(s->flags&AVFMT\_FLAG\_PRIV\_OPT) && s->pb && !s->data\_offset)
83. s->data\_offset = avio\_tell(s->pb);
85. s->raw\_packet\_buffer\_remaining\_size = RAW\_PACKET\_BUFFER\_SIZE;
87. **if** (options) {
88. av\_dict\_free(options);
89. \*options = tmp;
90. }
91. \*ps = s;
92. **return** 0;
94. fail:
95. ff\_id3v2\_free\_extra\_meta(&id3v2\_extra\_meta);
96. av\_dict\_free(&tmp);
97. **if** (s->pb && !(s->flags & AVFMT\_FLAG\_CUSTOM\_IO))
98. avio\_close(s->pb);
99. avformat\_free\_context(s);
100. \*ps = NULL;
101. **return** ret;
102. }

avformat\_open\_input()源代码比较长，一部分是一些容错代码，比如说如果发现传入的AVFormatContext指针没有初始化过，就调用avformat\_alloc\_context()初始化该结构体；还有一部分是针对一些格式做的特殊处理，比如id3v2信息的处理等等。有关上述两种信息不再详细分析，在这里只选择它关键的两个函数进行分析：

**init\_input()**：绝大部分初始化工作都是在这里做的。

**s->iformat->read\_header()**：读取多媒体数据文件头，根据视音频流创建相应的AVStream。

下面我们逐一看看上述函数。

## init\_input()

init\_input()作为一个内部函数，竟然包含了一行注释（一般内部函数都没有注释），足可以看出它的重要性。它的主要工作就是打开输入的视频数据并且探测视频的格式。该函数的定义位于libavformat\utils.c，如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44064715) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44064715)

[在CODE上查看代码片](https://code.csdn.net/snippets/612296)

1. /\* Open input file and probe the format if necessary. \*/
2. **static** **int** init\_input(AVFormatContext \*s, **const** **char** \*filename,
3. AVDictionary \*\*options)
4. {
5. **int** ret;
6. AVProbeData pd = { filename, NULL, 0 };
7. **int** score = AVPROBE\_SCORE\_RETRY;
9. **if** (s->pb) {
10. s->flags |= AVFMT\_FLAG\_CUSTOM\_IO;
11. **if** (!s->iformat)
12. **return** av\_probe\_input\_buffer2(s->pb, &s->iformat, filename,
13. s, 0, s->format\_probesize);
14. **else** **if** (s->iformat->flags & AVFMT\_NOFILE)
15. av\_log(s, AV\_LOG\_WARNING, "Custom AVIOContext makes no sense and "
16. "will be ignored with AVFMT\_NOFILE format.\n");
17. **return** 0;
18. }
20. **if** ((s->iformat && s->iformat->flags & AVFMT\_NOFILE) ||
21. (!s->iformat && (s->iformat = av\_probe\_input\_format2(&pd, 0, &score))))
22. **return** score;
24. **if** ((ret = avio\_open2(&s->pb, filename, AVIO\_FLAG\_READ | s->avio\_flags,
25. &s->interrupt\_callback, options)) < 0)
26. **return** ret;
27. **if** (s->iformat)
28. **return** 0;
29. **return** av\_probe\_input\_buffer2(s->pb, &s->iformat, filename,
30. s, 0, s->format\_probesize);
31. }

这个函数在短短的几行代码中包含了好几个return，因此逻辑还是有点复杂的，我们可以梳理一下：  
 在函数的开头的score变量是一个判决AVInputFormat的分数的门限值，如果最后得到的AVInputFormat的分数低于该门限值，就认为没有找到合适的AVInputFormat。FFmpeg内部判断封装格式的原理实际上是对每种AVInputFormat给出一个分数，满分是100分，越有可能正确的AVInputFormat给出的分数就越高。最后选择分数最高的AVInputFormat作为推测结果。score的值是一个宏定义AVPROBE\_SCORE\_RETRY，我们可以看一下它的定义：

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44064715) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44064715)

[在CODE上查看代码片](https://code.csdn.net/snippets/612296)

1. #define AVPROBE\_SCORE\_RETRY (AVPROBE\_SCORE\_MAX/4)

其中AVPROBE\_SCORE\_MAX是score的最大值，取值是100：

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44064715) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44064715)

[在CODE上查看代码片](https://code.csdn.net/snippets/612296)

1. #define AVPROBE\_SCORE\_MAX       100 ///< maximum score

由此我们可以得出score取值是25，即如果推测后得到的最佳AVInputFormat的分值低于25，就认为没有找到合适的AVInputFormat。  
  
 整个函数的逻辑大体如下：

（1）当使用了自定义的AVIOContext的时候（AVFormatContext中的AVIOContext不为空，即s->pb!=NULL），如果指定了AVInputFormat就直接返回，如果没有指定就调用av\_probe\_input\_buffer2()推测AVInputFormat。这一情况出现的不算很多，但是当我们从内存中读取数据的时候（需要初始化自定义的AVIOContext），就会执行这一步骤。  
（2）在更一般的情况下，如果已经指定了AVInputFormat，就直接返回；如果没有指定AVInputFormat，就调用av\_probe\_input\_format(NULL,…)根据文件路径判断文件格式。这里特意把av\_probe\_input\_format()的第1个参数写成“NULL”，是为了强调这个时候实际上并没有给函数提供输入数据，此时仅仅通过文件路径推测AVInputFormat。

（3）如果发现通过文件路径判断不出来文件格式，那么就需要打开文件探测文件格式了，这个时候会首先调用avio\_open2()打开文件，然后调用av\_probe\_input\_buffer2()推测AVInputFormat。

下面分析一下av\_probe\_input\_format()，avio\_open2()，av\_probe\_input\_buffer2()这几个函数。

## av\_probe\_input\_format2()

av\_probe\_input\_format2()是一个API函数，声明位于libavformat\avformat.h，如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44064715) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44064715)

[在CODE上查看代码片](https://code.csdn.net/snippets/612296)

1. /\*\*
2. \* Guess the file format.
3. \*
4. \* @param pd        data to be probed
5. \* @param is\_opened Whether the file is already opened; determines whether
6. \*                  demuxers with or without AVFMT\_NOFILE are probed.
7. \* @param score\_max A probe score larger that this is required to accept a
8. \*                  detection, the variable is set to the actual detection
9. \*                  score afterwards.
10. \*                  If the score is <= AVPROBE\_SCORE\_MAX / 4 it is recommended
11. \*                  to retry with a larger probe buffer.
12. \*/
13. AVInputFormat \*av\_probe\_input\_format2(AVProbeData \*pd, **int** is\_opened, **int** \*score\_max);

该函数用于根据输入数据查找合适的AVInputFormat。参数含义如下所示：

pd：存储输入数据信息的AVProbeData结构体。  
is\_opened：文件是否打开。

score\_max：判决AVInputFormat的门限值。只有某格式判决分数大于该门限值的时候，函数才会返回该封装格式，否则返回NULL。

该函数中涉及到一个结构体AVProbeData，用于存储输入文件的一些信息，它的定义如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44064715) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44064715)

[在CODE上查看代码片](https://code.csdn.net/snippets/612296)

1. /\*\*
2. \* This structure contains the data a format has to probe a file.
3. \*/
4. **typedef** **struct** AVProbeData {
5. **const** **char** \*filename;
6. unsigned **char** \*buf; /\*\*< Buffer must have AVPROBE\_PADDING\_SIZE of extra allocated bytes filled with zero. \*/
7. **int** buf\_size;       /\*\*< Size of buf except extra allocated bytes \*/
8. **const** **char** \*mime\_type; /\*\*< mime\_type, when known. \*/
9. } AVProbeData;

av\_probe\_input\_format2()函数的定义位于libavformat\format.c，如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44064715) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44064715)

[在CODE上查看代码片](https://code.csdn.net/snippets/612296)

1. AVInputFormat \*av\_probe\_input\_format2(AVProbeData \*pd, **int** is\_opened, **int** \*score\_max)
2. {
3. **int** score\_ret;
4. AVInputFormat \*fmt = av\_probe\_input\_format3(pd, is\_opened, &score\_ret);
5. **if** (score\_ret > \*score\_max) {
6. \*score\_max = score\_ret;
7. **return** fmt;
8. } **else**
9. **return** NULL;
10. }

从函数中可以看出，av\_probe\_input\_format2()调用了av\_probe\_input\_format3()，并且增加了一个判断，当av\_probe\_input\_format3()返回的分数大于score\_max的时候，才会返回AVInputFormat，否则返回NULL。

下面我们看一下av\_probe\_input\_format3()。

## av\_probe\_input\_format3()

av\_probe\_input\_format3()是一个API函数，声明位于libavformat\avformat.h，如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44064715) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44064715)

[在CODE上查看代码片](https://code.csdn.net/snippets/612296)

1. /\*\*
2. \* Guess the file format.
3. \*
4. \* @param is\_opened Whether the file is already opened; determines whether
5. \*                  demuxers with or without AVFMT\_NOFILE are probed.
6. \* @param score\_ret The score of the best detection.
7. \*/
8. AVInputFormat \*av\_probe\_input\_format3(AVProbeData \*pd, **int** is\_opened, **int** \*score\_ret);

从函数声明中可以看出，av\_probe\_input\_format3()和av\_probe\_input\_format2()的区别是函数的第3个参数不同：av\_probe\_input\_format2()是一个分数的门限值，而av\_probe\_input\_format3()是一个探测后的最匹配的格式的分数值。

av\_probe\_input\_format3()的定义位于libavformat\format.c，如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44064715) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44064715)

[在CODE上查看代码片](https://code.csdn.net/snippets/612296)

1. #define AVPROBE\_PADDING\_SIZE 32             ///< extra allocated bytes at the end of the probe buffer
2. #define AVPROBE\_SCORE\_EXTENSION  50 ///< score for file extension
3. #define AVPROBE\_SCORE\_MIME       75 ///< score for file mime type
4. #define AVPROBE\_SCORE\_MAX       100 ///< maximum score
6. AVInputFormat \*av\_probe\_input\_format3(AVProbeData \*pd, **int** is\_opened,
7. **int** \*score\_ret)
8. {
9. AVProbeData lpd = \*pd;
10. AVInputFormat \*fmt1 = NULL, \*fmt;
11. **int** score, nodat = 0, score\_max = 0;
12. **const** **static** uint8\_t zerobuffer[AVPROBE\_PADDING\_SIZE];
14. **if** (!lpd.buf)
15. lpd.buf = zerobuffer;
17. **if** (lpd.buf\_size > 10 && ff\_id3v2\_match(lpd.buf, ID3v2\_DEFAULT\_MAGIC)) {
18. **int** id3len = ff\_id3v2\_tag\_len(lpd.buf);
19. **if** (lpd.buf\_size > id3len + 16) {
20. lpd.buf      += id3len;
21. lpd.buf\_size -= id3len;
22. } **else** **if** (id3len >= PROBE\_BUF\_MAX) {
23. nodat = 2;
24. } **else**
25. nodat = 1;
26. }
28. fmt = NULL;
29. **while** ((fmt1 = av\_iformat\_next(fmt1))) {
30. **if** (!is\_opened == !(fmt1->flags & AVFMT\_NOFILE) && strcmp(fmt1->name, "image2"))
31. **continue**;
32. score = 0;
33. **if** (fmt1->read\_probe) {
34. score = fmt1->read\_probe(&lpd);
35. **if** (fmt1->extensions && av\_match\_ext(lpd.filename, fmt1->extensions)) {
36. **if**      (nodat == 0) score = FFMAX(score, 1);
37. **else** **if** (nodat == 1) score = FFMAX(score, AVPROBE\_SCORE\_EXTENSION / 2 - 1);
38. **else**                 score = FFMAX(score, AVPROBE\_SCORE\_EXTENSION);
39. }
40. } **else** **if** (fmt1->extensions) {
41. **if** (av\_match\_ext(lpd.filename, fmt1->extensions))
42. score = AVPROBE\_SCORE\_EXTENSION;
43. }
44. **if** (av\_match\_name(lpd.mime\_type, fmt1->mime\_type))
45. score = FFMAX(score, AVPROBE\_SCORE\_MIME);
46. **if** (score > score\_max) {
47. score\_max = score;
48. fmt       = fmt1;
49. } **else** **if** (score == score\_max)
50. fmt = NULL;
51. }
52. **if** (nodat == 1)
53. score\_max = FFMIN(AVPROBE\_SCORE\_EXTENSION / 2 - 1, score\_max);
54. \*score\_ret = score\_max;
56. **return** fmt;
57. }

av\_probe\_input\_format3()根据输入数据查找合适的AVInputFormat。输入的数据位于AVProbeData中。前文已经提到过，AVProbeData定义如下。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44064715) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44064715)

[在CODE上查看代码片](https://code.csdn.net/snippets/612296)

1. /\*\*
2. \* This structure contains the data a format has to probe a file.
3. \*/
4. **typedef** **struct** AVProbeData {
5. **const** **char** \*filename;
6. unsigned **char** \*buf; /\*\*< Buffer must have AVPROBE\_PADDING\_SIZE of extra allocated bytes filled with zero. \*/
7. **int** buf\_size;       /\*\*< Size of buf except extra allocated bytes \*/
8. **const** **char** \*mime\_type; /\*\*< mime\_type, when known. \*/
9. } AVProbeData;

其中filename是文件路径， buf存储用于推测AVInputFormat的媒体数据，最后还有个mime\_type保存媒体的类型。其中buf可以为空，但是其后面无论如何都需要填充AVPROBE\_PADDING\_SIZE个0（AVPROBE\_PADDING\_SIZE取值为32，即32个0）。  
 该函数最主要的部分是一个循环。该循环调用av\_iformat\_next()遍历FFmpeg中所有的AVInputFormat，并根据以下规则确定AVInputFormat和输入媒体数据的匹配分数（score，反应匹配程度）：

（1）如果AVInputFormat中包含read\_probe()，就调用read\_probe()函数获取匹配分数（这一方法如果结果匹配的话，一般会获得AVPROBE\_SCORE\_MAX的分值，即100分）。如果不包含该函数，就使用av\_match\_ext()函数比较输入媒体的扩展名和AVInputFormat的扩展名是否匹配，如果匹配的话，设定匹配分数为AVPROBE\_SCORE\_EXTENSION（AVPROBE\_SCORE\_EXTENSION取值为50，即50分）。  
（2）使用av\_match\_name()比较输入媒体的mime\_type和AVInputFormat的mime\_type，如果匹配的话，设定匹配分数为AVPROBE\_SCORE\_MIME（AVPROBE\_SCORE\_MIME取值为75，即75分）。

（3）如果该AVInputFormat的匹配分数大于此前的最大匹配分数，则记录当前的匹配分数为最大匹配分数，并且记录当前的AVInputFormat为最佳匹配的AVInputFormat。

上述过程中涉及到以下几个知识点：

### AVInputFormat->read\_probe()

AVInputFormat中包含read\_probe()是用于获得匹配函数的函数指针，不同的封装格式包含不同的实现函数。例如，FLV封装格式的AVInputFormat模块定义（位于libavformat\flvdec.c）如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44064715) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44064715)

[在CODE上查看代码片](https://code.csdn.net/snippets/612296)

1. AVInputFormat ff\_flv\_demuxer = {
2. .name           = "flv",
3. .long\_name      = NULL\_IF\_CONFIG\_SMALL("FLV (Flash Video)"),
4. .priv\_data\_size = **sizeof**(FLVContext),
5. .read\_probe     = flv\_probe,
6. .read\_header    = flv\_read\_header,
7. .read\_packet    = flv\_read\_packet,
8. .read\_seek      = flv\_read\_seek,
9. .read\_close     = flv\_read\_close,
10. .extensions     = "flv",
11. .priv\_class     = &flv\_class,
12. };

其中，read\_probe()函数对应的是flv\_probe()函数。我们可以看一下flv\_probe()函数的定义：

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44064715) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44064715)

[在CODE上查看代码片](https://code.csdn.net/snippets/612296)

1. **static** **int** flv\_probe(AVProbeData \*p)
2. {
3. **return** probe(p, 0);
4. }

可见flv\_probe()调用了一个probe()函数。probe()函数的定义如下。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44064715) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44064715)

[在CODE上查看代码片](https://code.csdn.net/snippets/612296)

1. **static** **int** probe(AVProbeData \*p, **int** live)
2. {
3. **const** uint8\_t \*d = p->buf;
4. unsigned offset = AV\_RB32(d + 5);
6. **if** (d[0] == 'F' &&
7. d[1] == 'L' &&
8. d[2] == 'V' &&
9. d[3] < 5 && d[5] == 0 &&
10. offset + 100 < p->buf\_size &&
11. offset > 8) {
12. **int** is\_live = !memcmp(d + offset + 40, "NGINX RTMP", 10);
14. **if** (live == is\_live)
15. **return** AVPROBE\_SCORE\_MAX;
16. }
17. **return** 0;
18. }

从probe()函数我们可以看出，该函数做了如下工作：

（1）获得第6至第9字节的数据（对应Headersize字段）并且做大小端转换，然后存入offset变量。之所以要进行大小端转换是因为FLV是以“大端”方式存储数据，而[**操作系统**](http://lib.csdn.net/base/operatingsystem)是以“小端”方式存储数据，这一转换主要通过AV\_RB32()函数实现。AV\_RB32()是一个宏定义，其对应的函数是av\_bswap32()。

（2）检查开头3个字符（Signature）是否为“FLV”。

（3）第4个字节（Version）小于5。

（4）第6个字节（Headersize的第1个字节？）为0。

（5）offset取值大于8。

参照FLV文件头的格式可以对上述判断有一个更清晰的认识：



此外代码中还包含了有关live方式的FLV格式的判断，在这里我们不加探讨。对于我们打开FLV文件来说，live和is\_live两个变量取值都为0。也就是说满足上述5个条件的话，就可以认为输入媒体数据是FLV封装格式了。满足上述条件，probe()函数返回AVPROBE\_SCORE\_MAX（AVPROBE\_SCORE\_MAX取值为100，即100分），否则返回0（0分）。

### av\_match\_name()

av\_match\_name()是一个API函数，声明位于libavutil\avstring.h，如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44064715) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44064715)

[在CODE上查看代码片](https://code.csdn.net/snippets/612296)

1. /\*\*
2. \* Match instances of a name in a comma-separated list of names.
3. \* @param name  Name to look for.
4. \* @param names List of names.
5. \* @return 1 on match, 0 otherwise.
6. \*/
7. **int** av\_match\_name(**const** **char** \*name, **const** **char** \*names);

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

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44064715) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44064715)

[在CODE上查看代码片](https://code.csdn.net/snippets/612296)

1. **int** av\_match\_name(**const** **char** \*name, **const** **char** \*names)
2. {
3. **const** **char** \*p;
4. **int** len, namelen;
6. **if** (!name || !names)
7. **return** 0;
9. namelen = strlen(name);
10. **while** ((p = strchr(names, ','))) {
11. len = FFMAX(p - names, namelen);
12. **if** (!av\_strncasecmp(name, names, len))
13. **return** 1;
14. names = p + 1;
15. }
16. **return** !av\_strcasecmp(name, names);
17. }

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

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44064715) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44064715)

[在CODE上查看代码片](https://code.csdn.net/snippets/612296)

1. AVInputFormat ff\_matroska\_demuxer = {
2. .name           = "matroska,webm",
3. .long\_name      = NULL\_IF\_CONFIG\_SMALL("Matroska / WebM"),
4. .extensions     = "mkv,mk3d,mka,mks",
5. .priv\_data\_size = **sizeof**(MatroskaDemuxContext),
6. .read\_probe     = matroska\_probe,
7. .read\_header    = matroska\_read\_header,
8. .read\_packet    = matroska\_read\_packet,
9. .read\_close     = matroska\_read\_close,
10. .read\_seek      = matroska\_read\_seek,
11. .mime\_type      = "audio/webm,audio/x-matroska,video/webm,video/x-matroska"
12. };

从代码可以看出，ff\_matroska\_demuxer中的name字段对应“matroska,webm”，mime\_type字段对应“audio/webm,audio/x-matroska,video/webm,video/x-matroska”。av\_match\_name()函数对于这样的字符串，会把它按照“,”截断成一个个的名称，然后一一进行比较。

### av\_match\_ext()

av\_match\_ext()是一个API函数，声明位于libavformat\avformat.h（注意位置和av\_match\_name()不一样），如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44064715) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44064715)

[在CODE上查看代码片](https://code.csdn.net/snippets/612296)

1. /\*\*
2. \* Return a positive value if the given filename has one of the given
3. \* extensions, 0 otherwise.
4. \*
5. \* @param filename   file name to check against the given extensions
6. \* @param extensions a comma-separated list of filename extensions
7. \*/
8. **int** av\_match\_ext(**const** **char** \*filename, **const** **char** \*extensions);

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

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44064715) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44064715)

[在CODE上查看代码片](https://code.csdn.net/snippets/612296)

1. **int** av\_match\_ext(**const** **char** \*filename, **const** **char** \*extensions)
2. {
3. **const** **char** \*ext;
5. **if** (!filename)
6. **return** 0;
8. ext = strrchr(filename, '.');
9. **if** (ext)
10. **return** av\_match\_name(ext + 1, extensions);
11. **return** 0;
12. }

## avio\_open2()

有关avio\_open2()的分析可以参考文章：[FFmpeg源代码简单分析：avio\_open2()](http://blog.csdn.net/leixiaohua1020/article/details/41199947)

## av\_probe\_input\_buffer2()

av\_probe\_input\_buffer2()是一个API函数，它根据输入的媒体数据推测该媒体数据的AVInputFormat，声明位于libavformat\avformat.h，如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44064715) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44064715)

[在CODE上查看代码片](https://code.csdn.net/snippets/612296)

1. /\*\*
2. \* Probe a bytestream to determine the input format. Each time a probe returns
3. \* with a score that is too low, the probe buffer size is increased and another
4. \* attempt is made. When the maximum probe size is reached, the input format
5. \* with the highest score is returned.
6. \*
7. \* @param pb the bytestream to probe
8. \* @param fmt the input format is put here
9. \* @param filename the filename of the stream
10. \* @param logctx the log context
11. \* @param offset the offset within the bytestream to probe from
12. \* @param max\_probe\_size the maximum probe buffer size (zero for default)
13. \* @return the score in case of success, a negative value corresponding to an
14. \*         the maximal score is AVPROBE\_SCORE\_MAX
15. \* AVERROR code otherwise
16. \*/
17. **int** av\_probe\_input\_buffer2(AVIOContext \*pb, AVInputFormat \*\*fmt,
18. **const** **char** \*filename, **void** \*logctx,
19. unsigned **int** offset, unsigned **int** max\_probe\_size);

av\_probe\_input\_buffer2()参数的含义如下所示：

pb：用于读取数据的AVIOContext。  
fmt：输出推测出来的AVInputFormat。  
filename：输入媒体的路径。  
logctx：日志（没有研究过）。  
offset：开始推测AVInputFormat的偏移量。

max\_probe\_size：用于推测格式的媒体数据的最大值。

返回推测后的得到的AVInputFormat的匹配分数。  
  
 av\_probe\_input\_buffer2()的定义位于libavformat\format.c，如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44064715) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44064715)

[在CODE上查看代码片](https://code.csdn.net/snippets/612296)

1. **int** av\_probe\_input\_buffer2(AVIOContext \*pb, AVInputFormat \*\*fmt,
2. **const** **char** \*filename, **void** \*logctx,
3. unsigned **int** offset, unsigned **int** max\_probe\_size)
4. {
5. AVProbeData pd = { filename ? filename : "" };
6. uint8\_t \*buf = NULL;
7. **int** ret = 0, probe\_size, buf\_offset = 0;
8. **int** score = 0;
9. **int** ret2;
11. **if** (!max\_probe\_size)
12. max\_probe\_size = PROBE\_BUF\_MAX;
13. **else** **if** (max\_probe\_size < PROBE\_BUF\_MIN) {
14. av\_log(logctx, AV\_LOG\_ERROR,
15. "Specified probe size value %u cannot be < %u\n", max\_probe\_size, PROBE\_BUF\_MIN);
16. **return** AVERROR(EINVAL);
17. }
19. **if** (offset >= max\_probe\_size)
20. **return** AVERROR(EINVAL);
22. **if** (pb->av\_class) {
23. uint8\_t \*mime\_type\_opt = NULL;
24. av\_opt\_get(pb, "mime\_type", AV\_OPT\_SEARCH\_CHILDREN, &mime\_type\_opt);
25. pd.mime\_type = (**const** **char** \*)mime\_type\_opt;
26. }
27. #if 0
28. **if** (!\*fmt && pb->av\_class && av\_opt\_get(pb, "mime\_type", AV\_OPT\_SEARCH\_CHILDREN, &mime\_type) >= 0 && mime\_type) {
29. **if** (!av\_strcasecmp(mime\_type, "audio/aacp")) {
30. \*fmt = av\_find\_input\_format("aac");
31. }
32. av\_freep(&mime\_type);
33. }
34. #endif
36. **for** (probe\_size = PROBE\_BUF\_MIN; probe\_size <= max\_probe\_size && !\*fmt;
37. probe\_size = FFMIN(probe\_size << 1,
38. FFMAX(max\_probe\_size, probe\_size + 1))) {
39. score = probe\_size < max\_probe\_size ? AVPROBE\_SCORE\_RETRY : 0;
41. /\* Read probe data. \*/
42. **if** ((ret = av\_reallocp(&buf, probe\_size + AVPROBE\_PADDING\_SIZE)) < 0)
43. **goto** fail;
44. **if** ((ret = avio\_read(pb, buf + buf\_offset,
45. probe\_size - buf\_offset)) < 0) {
46. /\* Fail if error was not end of file, otherwise, lower score. \*/
47. **if** (ret != AVERROR\_EOF)
48. **goto** fail;
50. score = 0;
51. ret   = 0;          /\* error was end of file, nothing read \*/
52. }
53. buf\_offset += ret;
54. **if** (buf\_offset < offset)
55. **continue**;
56. pd.buf\_size = buf\_offset - offset;
57. pd.buf = &buf[offset];
59. memset(pd.buf + pd.buf\_size, 0, AVPROBE\_PADDING\_SIZE);
61. /\* Guess file format. \*/
62. \*fmt = av\_probe\_input\_format2(&pd, 1, &score);
63. **if** (\*fmt) {
64. /\* This can only be true in the last iteration. \*/
65. **if** (score <= AVPROBE\_SCORE\_RETRY) {
66. av\_log(logctx, AV\_LOG\_WARNING,
67. "Format %s detected only with low score of %d, "
68. "misdetection possible!\n", (\*fmt)->name, score);
69. } **else**
70. av\_log(logctx, AV\_LOG\_DEBUG,
71. "Format %s probed with size=%d and score=%d\n",
72. (\*fmt)->name, probe\_size, score);
73. #if 0
74. **FILE** \*f = fopen("probestat.tmp", "ab");
75. fprintf(f, "probe\_size:%d format:%s score:%d filename:%s\n", probe\_size, (\*fmt)->name, score, filename);
76. fclose(f);
77. #endif
78. }
79. }
81. **if** (!\*fmt)
82. ret = AVERROR\_INVALIDDATA;
84. fail:
85. /\* Rewind. Reuse probe buffer to avoid seeking. \*/
86. ret2 = ffio\_rewind\_with\_probe\_data(pb, &buf, buf\_offset);
87. **if** (ret >= 0)
88. ret = ret2;
90. av\_freep(&pd.mime\_type);
91. **return** ret < 0 ? ret : score;
92. }

av\_probe\_input\_buffer2()首先需要确定用于推测格式的媒体数据的最大值max\_probe\_size。max\_probe\_size默认为PROBE\_BUF\_MAX（PROBE\_BUF\_MAX取值为1 << 20，即1048576Byte，大约1MB）。

在确定了max\_probe\_size之后，函数就会进入到一个循环中，调用avio\_read()读取数据并且使用av\_probe\_input\_format2()（该函数前文已经记录过）推测文件格式。

肯定有人会奇怪这里为什么要使用一个循环，而不是只运行一次？其实这个循环是一个逐渐增加输入媒体数据量的过程。av\_probe\_input\_buffer2()并不是一次性读取max\_probe\_size字节的媒体数据，我个人感觉可能是因为这样做不是很经济，因为推测大部分媒体格式根本用不到1MB这么多的媒体数据。因此函数中使用一个probe\_size存储需要读取的字节数，并且随着循环次数的增加逐渐增加这个值。函数首先从PROBE\_BUF\_MIN（取值为2048）个字节开始读取，如果通过这些数据已经可以推测出AVInputFormat，那么就可以直接退出循环了（参考for循环的判断条件“!\*fmt”）；如果没有推测出来，就增加probe\_size的量为过去的2倍（参考for循环的表达式“probe\_size << 1”），继续推测AVInputFormat；如果一直读取到max\_probe\_size字节的数据依然没能确定AVInputFormat，则会退出循环并且返回错误信息。

## AVInputFormat-> read\_header()

在调用完init\_input()完成基本的初始化并且推测得到相应的AVInputFormat之后，avformat\_open\_input()会调用AVInputFormat的read\_header()方法读取媒体文件的文件头并且完成相关的初始化工作。read\_header()是一个位于AVInputFormat结构体中的一个函数指针，对于不同的封装格式，会调用不同的read\_header()的实现函数。举个例子，当输入视频的封装格式为FLV的时候，会调用FLV的AVInputFormat中的read\_header()。FLV的AVInputFormat定义位于libavformat\flvdec.c文件中，如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44064715) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44064715)

1. AVInputFormat ff\_flv\_demuxer = {
2. .name           = "flv",
3. .long\_name      = NULL\_IF\_CONFIG\_SMALL("FLV (Flash Video)"),
4. .priv\_data\_size = **sizeof**(FLVContext),
5. .read\_probe     = flv\_probe,
6. .read\_header    = flv\_read\_header,
7. .read\_packet    = flv\_read\_packet,
8. .read\_seek      = flv\_read\_seek,
9. .read\_close     = flv\_read\_close,
10. .extensions     = "flv",
11. .priv\_class     = &flv\_class,
12. };

可以看出read\_header()指向了flv\_read\_header()函数。flv\_read\_header()的实现同样位于libavformat\flvdec.c文件中，如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44064715) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44064715)

[在CODE上查看代码片](https://code.csdn.net/snippets/612296)

1. **static** **int** flv\_read\_header(AVFormatContext \*s)
2. {
3. **int** offset, flags;
5. avio\_skip(s->pb, 4);
6. flags = avio\_r8(s->pb);
8. s->ctx\_flags |= AVFMTCTX\_NOHEADER;
10. **if** (flags & FLV\_HEADER\_FLAG\_HASVIDEO)
11. **if** (!create\_stream(s, AVMEDIA\_TYPE\_VIDEO))
12. **return** AVERROR(ENOMEM);
13. **if** (flags & FLV\_HEADER\_FLAG\_HASAUDIO)
14. **if** (!create\_stream(s, AVMEDIA\_TYPE\_AUDIO))
15. **return** AVERROR(ENOMEM);
16. // Flag doesn't indicate whether or not there is script-data present. Must
17. // create that stream if it's encountered.
19. offset = avio\_rb32(s->pb);
20. avio\_seek(s->pb, offset, SEEK\_SET);
21. avio\_skip(s->pb, 4);
23. s->start\_time = 0;
25. **return** 0;
26. }

可以看出，函数读取了FLV的文件头并且判断其中是否包含视频流和音频流。如果包含视频流或者音频流，就会调用create\_stream()函数。  
 create\_stream()函数定义也位于libavformat\flvdec.c中，如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44064715) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44064715)

[在CODE上查看代码片](https://code.csdn.net/snippets/612296)

1. **static** AVStream \*create\_stream(AVFormatContext \*s, **int** codec\_type)
2. {
3. AVStream \*st = avformat\_new\_stream(s, NULL);
4. **if** (!st)
5. **return** NULL;
6. st->codec->codec\_type = codec\_type;
7. **if** (s->nb\_streams>=3 ||(   s->nb\_streams==2
8. && s->streams[0]->codec->codec\_type != AVMEDIA\_TYPE\_DATA
9. && s->streams[1]->codec->codec\_type != AVMEDIA\_TYPE\_DATA))
10. s->ctx\_flags &= ~AVFMTCTX\_NOHEADER;
12. avpriv\_set\_pts\_info(st, 32, 1, 1000); /\* 32 bit pts in ms \*/
13. **return** st;
14. }

从代码中可以看出，create\_stream()调用了API函数avformat\_new\_stream()创建相应的视频流和音频流。  
 上面这段解析FLV头的代码可以参考一下FLV封装格式的文件头格式，如下图所示。



经过上面的步骤AVInputFormat的read\_header()完成了视音频流对应的AVStream的创建。至此，avformat\_open\_input()中的主要代码分析完毕。

# [FFmpeg源代码简单分析：avformat\_find\_stream\_info()](http://blog.csdn.net/leixiaohua1020/article/details/44084321)

本文简单分析FFmpeg中一个常用的函数：avformat\_find\_stream\_info()。该函数可以读取一部分视音频数据并且获得一些相关的信息。avformat\_find\_stream\_info()的声明位于libavformat\avformat.h，如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44084321) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44084321)

[在CODE上查看代码片](https://code.csdn.net/snippets/613078)

1. /\*\*
2. \* Read packets of a media file to get stream information. This
3. \* is useful for file formats with no headers such as MPEG. This
4. \* function also computes the real framerate in case of MPEG-2 repeat
5. \* frame mode.
6. \* The logical file position is not changed by this function;
7. \* examined packets may be buffered for later processing.
8. \*
9. \* @param ic media file handle
10. \* @param options  If non-NULL, an ic.nb\_streams long array of pointers to
11. \*                 dictionaries, where i-th member contains options for
12. \*                 codec corresponding to i-th stream.
13. \*                 On return each dictionary will be filled with options that were not found.
14. \* @return >=0 if OK, AVERROR\_xxx on error
15. \*
16. \* @note this function isn't guaranteed to open all the codecs, so
17. \*       options being non-empty at return is a perfectly normal behavior.
18. \*
19. \* @todo Let the user decide somehow what information is needed so that
20. \*       we do not waste time getting stuff the user does not need.
21. \*/
22. **int** avformat\_find\_stream\_info(AVFormatContext \*ic, AVDictionary \*\*options);

简单解释一下它的参数的含义：

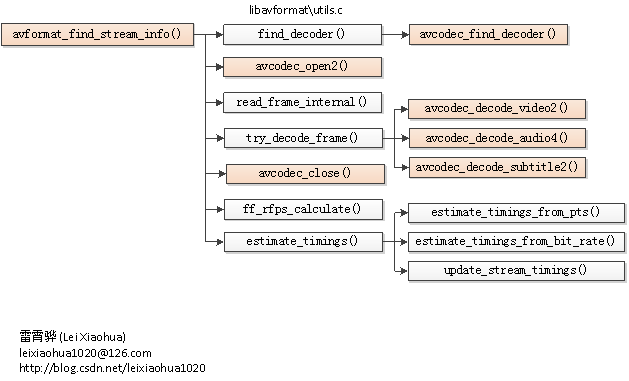
ic：输入的AVFormatContext。  
options：额外的选项，目前没有深入研究过。

函数正常执行后返回值大于等于0。  
该函数最典型的例子可以参考：[最简单的基于FFMPEG+SDL的视频播放器 ver2 （采用SDL2.0）](http://blog.csdn.net/leixiaohua1020/article/details/38868499)

PS：由于该函数比较复杂，所以只看了一部分代码，以后有时间再进一步分析。

## 函数调用关系图

函数的调用关系如下图所示。



## avformat\_find\_stream\_info()

avformat\_find\_stream\_info()的定义位于libavformat\utils.c。它的代码比较长，如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44084321) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44084321)

[在CODE上查看代码片](https://code.csdn.net/snippets/613078)

1. **int** avformat\_find\_stream\_info(AVFormatContext \*ic, AVDictionary \*\*options)
2. {
3. **int** i, count, ret = 0, j;
4. int64\_t read\_size;
5. AVStream \*st;
6. AVPacket pkt1, \*pkt;
7. int64\_t old\_offset  = avio\_tell(ic->pb);
8. // new streams might appear, no options for those
9. **int** orig\_nb\_streams = ic->nb\_streams;
10. **int** flush\_codecs;
11. int64\_t max\_analyze\_duration = ic->max\_analyze\_duration2;
12. int64\_t probesize = ic->probesize2;

15. **if** (!max\_analyze\_duration)
16. max\_analyze\_duration = ic->max\_analyze\_duration;
17. **if** (ic->probesize)
18. probesize = ic->probesize;
19. flush\_codecs = probesize > 0;

22. av\_opt\_set(ic, "skip\_clear", "1", AV\_OPT\_SEARCH\_CHILDREN);

25. **if** (!max\_analyze\_duration) {
26. **if** (!strcmp(ic->iformat->name, "flv") && !(ic->ctx\_flags & AVFMTCTX\_NOHEADER)) {
27. max\_analyze\_duration = 10\*AV\_TIME\_BASE;
28. } **else**
29. max\_analyze\_duration = 5\*AV\_TIME\_BASE;
30. }

33. **if** (ic->pb)
34. av\_log(ic, AV\_LOG\_DEBUG, "Before avformat\_find\_stream\_info() pos: %"PRId64" bytes read:%"PRId64" seeks:%d\n",
35. avio\_tell(ic->pb), ic->pb->bytes\_read, ic->pb->seek\_count);

38. **for** (i = 0; i < ic->nb\_streams; i++) {
39. **const** AVCodec \*codec;
40. AVDictionary \*thread\_opt = NULL;
41. st = ic->streams[i];

44. **if** (st->codec->codec\_type == AVMEDIA\_TYPE\_VIDEO ||
45. st->codec->codec\_type == AVMEDIA\_TYPE\_SUBTITLE) {
46. /\*            if (!st->time\_base.num)
47. st->time\_base = \*/
48. **if** (!st->codec->time\_base.num)
49. st->codec->time\_base = st->time\_base;
50. }
51. // only for the split stuff
52. **if** (!st->parser && !(ic->flags & AVFMT\_FLAG\_NOPARSE)) {
53. st->parser = av\_parser\_init(st->codec->codec\_id);
54. **if** (st->parser) {
55. **if** (st->need\_parsing == AVSTREAM\_PARSE\_HEADERS) {
56. st->parser->flags |= PARSER\_FLAG\_COMPLETE\_FRAMES;
57. } **else** **if** (st->need\_parsing == AVSTREAM\_PARSE\_FULL\_RAW) {
58. st->parser->flags |= PARSER\_FLAG\_USE\_CODEC\_TS;
59. }
60. } **else** **if** (st->need\_parsing) {
61. av\_log(ic, AV\_LOG\_VERBOSE, "parser not found for codec "
62. "%s, packets or times may be invalid.\n",
63. avcodec\_get\_name(st->codec->codec\_id));
64. }
65. }
66. codec = find\_decoder(ic, st, st->codec->codec\_id);

69. /\* Force thread count to 1 since the H.264 decoder will not extract
70. \* SPS and PPS to extradata during multi-threaded decoding. \*/
71. av\_dict\_set(options ? &options[i] : &thread\_opt, "threads", "1", 0);

74. **if** (ic->codec\_whitelist)
75. av\_dict\_set(options ? &options[i] : &thread\_opt, "codec\_whitelist", ic->codec\_whitelist, 0);

78. /\* Ensure that subtitle\_header is properly set. \*/
79. **if** (st->codec->codec\_type == AVMEDIA\_TYPE\_SUBTITLE
80. && codec && !st->codec->codec) {
81. **if** (avcodec\_open2(st->codec, codec, options ? &options[i] : &thread\_opt) < 0)
82. av\_log(ic, AV\_LOG\_WARNING,
83. "Failed to open codec in av\_find\_stream\_info\n");
84. }

87. // Try to just open decoders, in case this is enough to get parameters.
88. **if** (!has\_codec\_parameters(st, NULL) && st->request\_probe <= 0) {
89. **if** (codec && !st->codec->codec)
90. **if** (avcodec\_open2(st->codec, codec, options ? &options[i] : &thread\_opt) < 0)
91. av\_log(ic, AV\_LOG\_WARNING,
92. "Failed to open codec in av\_find\_stream\_info\n");
93. }
94. **if** (!options)
95. av\_dict\_free(&thread\_opt);
96. }

99. **for** (i = 0; i < ic->nb\_streams; i++) {
100. #if FF\_API\_R\_FRAME\_RATE
101. ic->streams[i]->info->last\_dts = AV\_NOPTS\_VALUE;
102. #endif
103. ic->streams[i]->info->fps\_first\_dts = AV\_NOPTS\_VALUE;
104. ic->streams[i]->info->fps\_last\_dts  = AV\_NOPTS\_VALUE;
105. }

108. count     = 0;
109. read\_size = 0;
110. **for** (;;) {
111. **if** (ff\_check\_interrupt(&ic->interrupt\_callback)) {
112. ret = AVERROR\_EXIT;
113. av\_log(ic, AV\_LOG\_DEBUG, "interrupted\n");
114. **break**;
115. }

118. /\* check if one codec still needs to be handled \*/
119. **for** (i = 0; i < ic->nb\_streams; i++) {
120. **int** fps\_analyze\_framecount = 20;

123. st = ic->streams[i];
124. **if** (!has\_codec\_parameters(st, NULL))
125. **break**;
126. /\* If the timebase is coarse (like the usual millisecond precision
127. \* of mkv), we need to analyze more frames to reliably arrive at
128. \* the correct fps. \*/
129. **if** (av\_q2d(st->time\_base) > 0.0005)
130. fps\_analyze\_framecount \*= 2;
131. **if** (!tb\_unreliable(st->codec))
132. fps\_analyze\_framecount = 0;
133. **if** (ic->fps\_probe\_size >= 0)
134. fps\_analyze\_framecount = ic->fps\_probe\_size;
135. **if** (st->disposition & AV\_DISPOSITION\_ATTACHED\_PIC)
136. fps\_analyze\_framecount = 0;
137. /\* variable fps and no guess at the real fps \*/
138. **if** (!(st->r\_frame\_rate.num && st->avg\_frame\_rate.num) &&
139. st->info->duration\_count < fps\_analyze\_framecount &&
140. st->codec->codec\_type == AVMEDIA\_TYPE\_VIDEO)
141. **break**;
142. **if** (st->parser && st->parser->parser->split &&
143. !st->codec->extradata)
144. **break**;
145. **if** (st->first\_dts == AV\_NOPTS\_VALUE &&
146. !(ic->iformat->flags & AVFMT\_NOTIMESTAMPS) &&
147. st->codec\_info\_nb\_frames < ic->max\_ts\_probe &&
148. (st->codec->codec\_type == AVMEDIA\_TYPE\_VIDEO ||
149. st->codec->codec\_type == AVMEDIA\_TYPE\_AUDIO))
150. **break**;
151. }
152. **if** (i == ic->nb\_streams) {
153. /\* NOTE: If the format has no header, then we need to read some
154. \* packets to get most of the streams, so we cannot stop here. \*/
155. **if** (!(ic->ctx\_flags & AVFMTCTX\_NOHEADER)) {
156. /\* If we found the info for all the codecs, we can stop. \*/
157. ret = count;
158. av\_log(ic, AV\_LOG\_DEBUG, "All info found\n");
159. flush\_codecs = 0;
160. **break**;
161. }
162. }
163. /\* We did not get all the codec info, but we read too much data. \*/
164. **if** (read\_size >= probesize) {
165. ret = count;
166. av\_log(ic, AV\_LOG\_DEBUG,
167. "Probe buffer size limit of %"PRId64" bytes reached\n", probesize);
168. **for** (i = 0; i < ic->nb\_streams; i++)
169. **if** (!ic->streams[i]->r\_frame\_rate.num &&
170. ic->streams[i]->info->duration\_count <= 1 &&
171. ic->streams[i]->codec->codec\_type == AVMEDIA\_TYPE\_VIDEO &&
172. strcmp(ic->iformat->name, "image2"))
173. av\_log(ic, AV\_LOG\_WARNING,
174. "Stream #%d: not enough frames to estimate rate; "
175. "consider increasing probesize\n", i);
176. **break**;
177. }

180. /\* NOTE: A new stream can be added there if no header in file
181. \* (AVFMTCTX\_NOHEADER). \*/
182. ret = read\_frame\_internal(ic, &pkt1);
183. **if** (ret == AVERROR(EAGAIN))
184. **continue**;

187. **if** (ret < 0) {
188. /\* EOF or error\*/
189. **break**;
190. }

193. **if** (ic->flags & AVFMT\_FLAG\_NOBUFFER)
194. free\_packet\_buffer(&ic->packet\_buffer, &ic->packet\_buffer\_end);
195. {
196. pkt = add\_to\_pktbuf(&ic->packet\_buffer, &pkt1,
197. &ic->packet\_buffer\_end);
198. **if** (!pkt) {
199. ret = AVERROR(ENOMEM);
200. **goto** find\_stream\_info\_err;
201. }
202. **if** ((ret = av\_dup\_packet(pkt)) < 0)
203. **goto** find\_stream\_info\_err;
204. }

207. st = ic->streams[pkt->stream\_index];
208. **if** (!(st->disposition & AV\_DISPOSITION\_ATTACHED\_PIC))
209. read\_size += pkt->size;

212. **if** (pkt->dts != AV\_NOPTS\_VALUE && st->codec\_info\_nb\_frames > 1) {
213. /\* check for non-increasing dts \*/
214. **if** (st->info->fps\_last\_dts != AV\_NOPTS\_VALUE &&
215. st->info->fps\_last\_dts >= pkt->dts) {
216. av\_log(ic, AV\_LOG\_DEBUG,
217. "Non-increasing DTS in stream %d: packet %d with DTS "
218. "%"PRId64", packet %d with DTS %"PRId64"\n",
219. st->index, st->info->fps\_last\_dts\_idx,
220. st->info->fps\_last\_dts, st->codec\_info\_nb\_frames,
221. pkt->dts);
222. st->info->fps\_first\_dts =
223. st->info->fps\_last\_dts  = AV\_NOPTS\_VALUE;
224. }
225. /\* Check for a discontinuity in dts. If the difference in dts
226. \* is more than 1000 times the average packet duration in the
227. \* sequence, we treat it as a discontinuity. \*/
228. **if** (st->info->fps\_last\_dts != AV\_NOPTS\_VALUE &&
229. st->info->fps\_last\_dts\_idx > st->info->fps\_first\_dts\_idx &&
230. (pkt->dts - st->info->fps\_last\_dts) / 1000 >
231. (st->info->fps\_last\_dts     - st->info->fps\_first\_dts) /
232. (st->info->fps\_last\_dts\_idx - st->info->fps\_first\_dts\_idx)) {
233. av\_log(ic, AV\_LOG\_WARNING,
234. "DTS discontinuity in stream %d: packet %d with DTS "
235. "%"PRId64", packet %d with DTS %"PRId64"\n",
236. st->index, st->info->fps\_last\_dts\_idx,
237. st->info->fps\_last\_dts, st->codec\_info\_nb\_frames,
238. pkt->dts);
239. st->info->fps\_first\_dts =
240. st->info->fps\_last\_dts  = AV\_NOPTS\_VALUE;
241. }

244. /\* update stored dts values \*/
245. **if** (st->info->fps\_first\_dts == AV\_NOPTS\_VALUE) {
246. st->info->fps\_first\_dts     = pkt->dts;
247. st->info->fps\_first\_dts\_idx = st->codec\_info\_nb\_frames;
248. }
249. st->info->fps\_last\_dts     = pkt->dts;
250. st->info->fps\_last\_dts\_idx = st->codec\_info\_nb\_frames;
251. }
252. **if** (st->codec\_info\_nb\_frames>1) {
253. int64\_t t = 0;

256. **if** (st->time\_base.den > 0)
257. t = av\_rescale\_q(st->info->codec\_info\_duration, st->time\_base, AV\_TIME\_BASE\_Q);
258. **if** (st->avg\_frame\_rate.num > 0)
259. t = FFMAX(t, av\_rescale\_q(st->codec\_info\_nb\_frames, av\_inv\_q(st->avg\_frame\_rate), AV\_TIME\_BASE\_Q));

262. **if** (   t == 0
263. && st->codec\_info\_nb\_frames>30
264. && st->info->fps\_first\_dts != AV\_NOPTS\_VALUE
265. && st->info->fps\_last\_dts  != AV\_NOPTS\_VALUE)
266. t = FFMAX(t, av\_rescale\_q(st->info->fps\_last\_dts - st->info->fps\_first\_dts, st->time\_base, AV\_TIME\_BASE\_Q));

269. **if** (t >= max\_analyze\_duration) {
270. av\_log(ic, AV\_LOG\_VERBOSE, "max\_analyze\_duration %"PRId64" reached at %"PRId64" microseconds\n",
271. max\_analyze\_duration,
272. t);
273. **if** (ic->flags & AVFMT\_FLAG\_NOBUFFER)
274. av\_packet\_unref(pkt);
275. **break**;
276. }
277. **if** (pkt->duration) {
278. st->info->codec\_info\_duration        += pkt->duration;
279. st->info->codec\_info\_duration\_fields += st->parser && st->need\_parsing && st->codec->ticks\_per\_frame ==2 ? st->parser->repeat\_pict + 1 : 2;
280. }
281. }
282. #if FF\_API\_R\_FRAME\_RATE
283. **if** (st->codec->codec\_type == AVMEDIA\_TYPE\_VIDEO)
284. ff\_rfps\_add\_frame(ic, st, pkt->dts);
285. #endif
286. **if** (st->parser && st->parser->parser->split && !st->codec->extradata) {
287. **int** i = st->parser->parser->split(st->codec, pkt->data, pkt->size);
288. **if** (i > 0 && i < FF\_MAX\_EXTRADATA\_SIZE) {
289. **if** (ff\_alloc\_extradata(st->codec, i))
290. **return** AVERROR(ENOMEM);
291. memcpy(st->codec->extradata, pkt->data,
292. st->codec->extradata\_size);
293. }
294. }

297. /\* If still no information, we try to open the codec and to
298. \* decompress the frame. We try to avoid that in most cases as
299. \* it takes longer and uses more memory. For MPEG-4, we need to
300. \* decompress for QuickTime.
301. \*
302. \* If CODEC\_CAP\_CHANNEL\_CONF is set this will force decoding of at
303. \* least one frame of codec data, this makes sure the codec initializes
304. \* the channel configuration and does not only trust the values from
305. \* the container. \*/
306. try\_decode\_frame(ic, st, pkt,
307. (options && i < orig\_nb\_streams) ? &options[i] : NULL);

310. **if** (ic->flags & AVFMT\_FLAG\_NOBUFFER)
311. av\_packet\_unref(pkt);

314. st->codec\_info\_nb\_frames++;
315. count++;
316. }

319. **if** (flush\_codecs) {
320. AVPacket empty\_pkt = { 0 };
321. **int** err = 0;
322. av\_init\_packet(&empty\_pkt);

325. **for** (i = 0; i < ic->nb\_streams; i++) {

328. st = ic->streams[i];

331. /\* flush the decoders \*/
332. **if** (st->info->found\_decoder == 1) {
333. **do** {
334. err = try\_decode\_frame(ic, st, &empty\_pkt,
335. (options && i < orig\_nb\_streams)
336. ? &options[i] : NULL);
337. } **while** (err > 0 && !has\_codec\_parameters(st, NULL));

340. **if** (err < 0) {
341. av\_log(ic, AV\_LOG\_INFO,
342. "decoding for stream %d failed\n", st->index);
343. }
344. }
345. }
346. }

349. // close codecs which were opened in try\_decode\_frame()
350. **for** (i = 0; i < ic->nb\_streams; i++) {
351. st = ic->streams[i];
352. avcodec\_close(st->codec);
353. }

356. ff\_rfps\_calculate(ic);

359. **for** (i = 0; i < ic->nb\_streams; i++) {
360. st = ic->streams[i];
361. **if** (st->codec->codec\_type == AVMEDIA\_TYPE\_VIDEO) {
362. **if** (st->codec->codec\_id == AV\_CODEC\_ID\_RAWVIDEO && !st->codec->codec\_tag && !st->codec->bits\_per\_coded\_sample) {
363. uint32\_t tag= avcodec\_pix\_fmt\_to\_codec\_tag(st->codec->pix\_fmt);
364. **if** (avpriv\_find\_pix\_fmt(avpriv\_get\_raw\_pix\_fmt\_tags(), tag) == st->codec->pix\_fmt)
365. st->codec->codec\_tag= tag;
366. }

369. /\* estimate average framerate if not set by demuxer \*/
370. **if** (st->info->codec\_info\_duration\_fields &&
371. !st->avg\_frame\_rate.num &&
372. st->info->codec\_info\_duration) {
373. **int** best\_fps      = 0;
374. **double** best\_error = 0.01;

377. **if** (st->info->codec\_info\_duration        >= INT64\_MAX / st->time\_base.num / 2||
378. st->info->codec\_info\_duration\_fields >= INT64\_MAX / st->time\_base.den ||
379. st->info->codec\_info\_duration        < 0)
380. **continue**;
381. av\_reduce(&st->avg\_frame\_rate.num, &st->avg\_frame\_rate.den,
382. st->info->codec\_info\_duration\_fields \* (int64\_t) st->time\_base.den,
383. st->info->codec\_info\_duration \* 2 \* (int64\_t) st->time\_base.num, 60000);

386. /\* Round guessed framerate to a "standard" framerate if it's
387. \* within 1% of the original estimate. \*/
388. **for** (j = 0; j < MAX\_STD\_TIMEBASES; j++) {
389. AVRational std\_fps = { get\_std\_framerate(j), 12 \* 1001 };
390. **double** error       = fabs(av\_q2d(st->avg\_frame\_rate) /
391. av\_q2d(std\_fps) - 1);

394. **if** (error < best\_error) {
395. best\_error = error;
396. best\_fps   = std\_fps.num;
397. }
398. }
399. **if** (best\_fps)
400. av\_reduce(&st->avg\_frame\_rate.num, &st->avg\_frame\_rate.den,
401. best\_fps, 12 \* 1001, INT\_MAX);
402. }

405. **if** (!st->r\_frame\_rate.num) {
406. **if** (    st->codec->time\_base.den \* (int64\_t) st->time\_base.num
407. <= st->codec->time\_base.num \* st->codec->ticks\_per\_frame \* (int64\_t) st->time\_base.den) {
408. st->r\_frame\_rate.num = st->codec->time\_base.den;
409. st->r\_frame\_rate.den = st->codec->time\_base.num \* st->codec->ticks\_per\_frame;
410. } **else** {
411. st->r\_frame\_rate.num = st->time\_base.den;
412. st->r\_frame\_rate.den = st->time\_base.num;
413. }
414. }
415. } **else** **if** (st->codec->codec\_type == AVMEDIA\_TYPE\_AUDIO) {
416. **if** (!st->codec->bits\_per\_coded\_sample)
417. st->codec->bits\_per\_coded\_sample =
418. av\_get\_bits\_per\_sample(st->codec->codec\_id);
419. // set stream disposition based on audio service type
420. **switch** (st->codec->audio\_service\_type) {
421. **case** AV\_AUDIO\_SERVICE\_TYPE\_EFFECTS:
422. st->disposition = AV\_DISPOSITION\_CLEAN\_EFFECTS;
423. **break**;
424. **case** AV\_AUDIO\_SERVICE\_TYPE\_VISUALLY\_IMPAIRED:
425. st->disposition = AV\_DISPOSITION\_VISUAL\_IMPAIRED;
426. **break**;
427. **case** AV\_AUDIO\_SERVICE\_TYPE\_HEARING\_IMPAIRED:
428. st->disposition = AV\_DISPOSITION\_HEARING\_IMPAIRED;
429. **break**;
430. **case** AV\_AUDIO\_SERVICE\_TYPE\_COMMENTARY:
431. st->disposition = AV\_DISPOSITION\_COMMENT;
432. **break**;
433. **case** AV\_AUDIO\_SERVICE\_TYPE\_KARAOKE:
434. st->disposition = AV\_DISPOSITION\_KARAOKE;
435. **break**;
436. }
437. }
438. }

441. **if** (probesize)
442. estimate\_timings(ic, old\_offset);

445. av\_opt\_set(ic, "skip\_clear", "0", AV\_OPT\_SEARCH\_CHILDREN);

448. **if** (ret >= 0 && ic->nb\_streams)
449. /\* We could not have all the codec parameters before EOF. \*/
450. ret = -1;
451. **for** (i = 0; i < ic->nb\_streams; i++) {
452. **const** **char** \*errmsg;
453. st = ic->streams[i];
454. **if** (!has\_codec\_parameters(st, &errmsg)) {
455. **char** buf[256];
456. avcodec\_string(buf, **sizeof**(buf), st->codec, 0);
457. av\_log(ic, AV\_LOG\_WARNING,
458. "Could not find codec parameters for stream %d (%s): %s\n"
459. "Consider increasing the value for the 'analyzeduration' and 'probesize' options\n",
460. i, buf, errmsg);
461. } **else** {
462. ret = 0;
463. }
464. }

467. compute\_chapters\_end(ic);

470. find\_stream\_info\_err:
471. **for** (i = 0; i < ic->nb\_streams; i++) {
472. st = ic->streams[i];
473. **if** (ic->streams[i]->codec->codec\_type != AVMEDIA\_TYPE\_AUDIO)
474. ic->streams[i]->codec->thread\_count = 0;
475. **if** (st->info)
476. av\_freep(&st->info->duration\_error);
477. av\_freep(&ic->streams[i]->info);
478. }
479. **if** (ic->pb)
480. av\_log(ic, AV\_LOG\_DEBUG, "After avformat\_find\_stream\_info() pos: %"PRId64" bytes read:%"PRId64" seeks:%d frames:%d\n",
481. avio\_tell(ic->pb), ic->pb->bytes\_read, ic->pb->seek\_count, count);
482. **return** ret;
483. }

由于avformat\_find\_stream\_info()代码比较长，难以全部分析，在这里只能简单记录一下它的要点。该函数主要用于给每个媒体流（音频/视频）的AVStream结构体赋值。我们大致浏览一下这个函数的代码，会发现它其实已经实现了解码器的查找，解码器的打开，视音频帧的读取，视音频帧的解码等工作。换句话说，该函数实际上已经“走通”的解码的整个流程。下面看一下除了成员变量赋值之外，该函数的几个关键流程。

1.查找解码器：find\_decoder()  
2.打开解码器：avcodec\_open2()  
3.读取完整的一帧压缩编码的数据：read\_frame\_internal()  
注：av\_read\_frame()内部实际上就是调用的read\_frame\_internal()。  
4.解码一些压缩编码数据：try\_decode\_frame()

下面选择上述流程中几个关键函数的代码简单看一下。

## find\_decoder()

find\_decoder()用于找到合适的解码器，它的定义如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44084321) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44084321)

[在CODE上查看代码片](https://code.csdn.net/snippets/613078)

1. **static** **const** AVCodec \*find\_decoder(AVFormatContext \*s, AVStream \*st, **enum** AVCodecID codec\_id)
2. {
3. **if** (st->codec->codec)
4. **return** st->codec->codec;

7. **switch** (st->codec->codec\_type) {
8. **case** AVMEDIA\_TYPE\_VIDEO:
9. **if** (s->video\_codec)    **return** s->video\_codec;
10. **break**;
11. **case** AVMEDIA\_TYPE\_AUDIO:
12. **if** (s->audio\_codec)    **return** s->audio\_codec;
13. **break**;
14. **case** AVMEDIA\_TYPE\_SUBTITLE:
15. **if** (s->subtitle\_codec) **return** s->subtitle\_codec;
16. **break**;
17. }

20. **return** avcodec\_find\_decoder(codec\_id);
21. }

从代码中可以看出，如果指定的AVStream已经包含了解码器，则函数什么也不做直接返回。否则调用avcodec\_find\_decoder()获取解码器。avcodec\_find\_decoder()是一个FFmpeg的API函数，在这里不做详细分析。

## read\_frame\_internal()

read\_frame\_internal()的功能是读取一帧压缩码流数据。FFmpeg的API函数av\_read\_frame()内部调用的就是read\_frame\_internal()。有关这方面的知识可以参考文章：

[ffmpeg 源代码简单分析 ： av\_read\_frame()](http://blog.csdn.net/leixiaohua1020/article/details/12678577)  
因此，可以认为read\_frame\_internal()和av\_read\_frame()的功能基本上是等同的。

## try\_decode\_frame()

try\_decode\_frame()的功能可以从字面上的意思进行理解：“尝试解码一些帧”，它的定义如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44084321) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44084321)

[在CODE上查看代码片](https://code.csdn.net/snippets/613078)

1. /\* returns 1 or 0 if or if not decoded data was returned, or a negative error \*/
2. **static** **int** try\_decode\_frame(AVFormatContext \*s, AVStream \*st, AVPacket \*avpkt,
3. AVDictionary \*\*options)
4. {
5. **const** AVCodec \*codec;
6. **int** got\_picture = 1, ret = 0;
7. AVFrame \*frame = av\_frame\_alloc();
8. AVSubtitle subtitle;
9. AVPacket pkt = \*avpkt;

12. **if** (!frame)
13. **return** AVERROR(ENOMEM);

16. **if** (!avcodec\_is\_open(st->codec) &&
17. st->info->found\_decoder <= 0 &&
18. (st->codec->codec\_id != -st->info->found\_decoder || !st->codec->codec\_id)) {
19. AVDictionary \*thread\_opt = NULL;

22. codec = find\_decoder(s, st, st->codec->codec\_id);

25. **if** (!codec) {
26. st->info->found\_decoder = -st->codec->codec\_id;
27. ret                     = -1;
28. **goto** fail;
29. }

32. /\* Force thread count to 1 since the H.264 decoder will not extract
33. \* SPS and PPS to extradata during multi-threaded decoding. \*/
34. av\_dict\_set(options ? options : &thread\_opt, "threads", "1", 0);
35. **if** (s->codec\_whitelist)
36. av\_dict\_set(options ? options : &thread\_opt, "codec\_whitelist", s->codec\_whitelist, 0);
37. ret = avcodec\_open2(st->codec, codec, options ? options : &thread\_opt);
38. **if** (!options)
39. av\_dict\_free(&thread\_opt);
40. **if** (ret < 0) {
41. st->info->found\_decoder = -st->codec->codec\_id;
42. **goto** fail;
43. }
44. st->info->found\_decoder = 1;
45. } **else** **if** (!st->info->found\_decoder)
46. st->info->found\_decoder = 1;

49. **if** (st->info->found\_decoder < 0) {
50. ret = -1;
51. **goto** fail;
52. }

55. **while** ((pkt.size > 0 || (!pkt.data && got\_picture)) &&
56. ret >= 0 &&
57. (!has\_codec\_parameters(st, NULL) || !has\_decode\_delay\_been\_guessed(st) ||
58. (!st->codec\_info\_nb\_frames &&
59. st->codec->codec->capabilities & CODEC\_CAP\_CHANNEL\_CONF))) {
60. got\_picture = 0;
61. **switch** (st->codec->codec\_type) {
62. **case** AVMEDIA\_TYPE\_VIDEO:
63. ret = avcodec\_decode\_video2(st->codec, frame,
64. &got\_picture, &pkt);
65. **break**;
66. **case** AVMEDIA\_TYPE\_AUDIO:
67. ret = avcodec\_decode\_audio4(st->codec, frame, &got\_picture, &pkt);
68. **break**;
69. **case** AVMEDIA\_TYPE\_SUBTITLE:
70. ret = avcodec\_decode\_subtitle2(st->codec, &subtitle,
71. &got\_picture, &pkt);
72. ret = pkt.size;
73. **break**;
74. **default**:
75. **break**;
76. }
77. **if** (ret >= 0) {
78. **if** (got\_picture)
79. st->nb\_decoded\_frames++;
80. pkt.data += ret;
81. pkt.size -= ret;
82. ret       = got\_picture;
83. }
84. }

87. **if** (!pkt.data && !got\_picture)
88. ret = -1;

91. fail:
92. av\_frame\_free(&frame);
93. **return** ret;
94. }

从try\_decode\_frame()的定义可以看出，该函数首先判断视音频流的解码器是否已经打开，如果没有打开的话，先打开相应的解码器。接下来根据视音频流类型的不同，调用不同的解码函数进行解码：视频流调用avcodec\_decode\_video2()，音频流调用avcodec\_decode\_audio4()，字幕流调用avcodec\_decode\_subtitle2()。解码的循环会一直持续下去直到满足了while()的所有条件。

while()语句的条件中有一个has\_codec\_parameters()函数，用于判断AVStream中的成员变量是否都已经设置完毕。该函数在avformat\_find\_stream\_info()中的多个地方被使用过。下面简单看一下该函数。

### has\_codec\_parameters()

has\_codec\_parameters()用于检查AVStream中的成员变量是否都已经设置完毕。函数的定义如下。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44084321) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44084321)

[在CODE上查看代码片](https://code.csdn.net/snippets/613078)

1. **static** **int** has\_codec\_parameters(AVStream \*st, **const** **char** \*\*errmsg\_ptr)
2. {
3. AVCodecContext \*avctx = st->codec;

6. #define FAIL(errmsg) do {                                         \
7. **if** (errmsg\_ptr)                                           \
8. \*errmsg\_ptr = errmsg;                                 \
9. **return** 0;                                                 \
10. } **while** (0)

13. **if** (   avctx->codec\_id == AV\_CODEC\_ID\_NONE
14. && avctx->codec\_type != AVMEDIA\_TYPE\_DATA)
15. FAIL("unknown codec");
16. **switch** (avctx->codec\_type) {
17. **case** AVMEDIA\_TYPE\_AUDIO:
18. **if** (!avctx->frame\_size && determinable\_frame\_size(avctx))
19. FAIL("unspecified frame size");
20. **if** (st->info->found\_decoder >= 0 &&
21. avctx->sample\_fmt == AV\_SAMPLE\_FMT\_NONE)
22. FAIL("unspecified sample format");
23. **if** (!avctx->sample\_rate)
24. FAIL("unspecified sample rate");
25. **if** (!avctx->channels)
26. FAIL("unspecified number of channels");
27. **if** (st->info->found\_decoder >= 0 && !st->nb\_decoded\_frames && avctx->codec\_id == AV\_CODEC\_ID\_DTS)
28. FAIL("no decodable DTS frames");
29. **break**;
30. **case** AVMEDIA\_TYPE\_VIDEO:
31. **if** (!avctx->width)
32. FAIL("unspecified size");
33. **if** (st->info->found\_decoder >= 0 && avctx->pix\_fmt == AV\_PIX\_FMT\_NONE)
34. FAIL("unspecified pixel format");
35. **if** (st->codec->codec\_id == AV\_CODEC\_ID\_RV30 || st->codec->codec\_id == AV\_CODEC\_ID\_RV40)
36. **if** (!st->sample\_aspect\_ratio.num && !st->codec->sample\_aspect\_ratio.num && !st->codec\_info\_nb\_frames)
37. FAIL("no frame in rv30/40 and no sar");
38. **break**;
39. **case** AVMEDIA\_TYPE\_SUBTITLE:
40. **if** (avctx->codec\_id == AV\_CODEC\_ID\_HDMV\_PGS\_SUBTITLE && !avctx->width)
41. FAIL("unspecified size");
42. **break**;
43. **case** AVMEDIA\_TYPE\_DATA:
44. **if** (avctx->codec\_id == AV\_CODEC\_ID\_NONE) **return** 1;
45. }

48. **return** 1;
49. }

## estimate\_timings()

estimate\_timings()位于avformat\_find\_stream\_info()最后面，用于估算AVFormatContext以及AVStream的时长duration。它的代码如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44084321) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44084321)

[在CODE上查看代码片](https://code.csdn.net/snippets/613078)

1. **static** **void** estimate\_timings(AVFormatContext \*ic, int64\_t old\_offset)
2. {
3. int64\_t file\_size;

6. /\* get the file size, if possible \*/
7. **if** (ic->iformat->flags & AVFMT\_NOFILE) {
8. file\_size = 0;
9. } **else** {
10. file\_size = avio\_size(ic->pb);
11. file\_size = FFMAX(0, file\_size);
12. }

15. **if** ((!strcmp(ic->iformat->name, "mpeg") ||
16. !strcmp(ic->iformat->name, "mpegts")) &&
17. file\_size && ic->pb->seekable) {
18. /\* get accurate estimate from the PTSes \*/
19. estimate\_timings\_from\_pts(ic, old\_offset);
20. ic->duration\_estimation\_method = AVFMT\_DURATION\_FROM\_PTS;
21. } **else** **if** (has\_duration(ic)) {
22. /\* at least one component has timings - we use them for all
23. \* the components \*/
24. fill\_all\_stream\_timings(ic);
25. ic->duration\_estimation\_method = AVFMT\_DURATION\_FROM\_STREAM;
26. } **else** {
27. /\* less precise: use bitrate info \*/
28. estimate\_timings\_from\_bit\_rate(ic);
29. ic->duration\_estimation\_method = AVFMT\_DURATION\_FROM\_BITRATE;
30. }
31. update\_stream\_timings(ic);

34. {
35. **int** i;
36. AVStream av\_unused \*st;
37. **for** (i = 0; i < ic->nb\_streams; i++) {
38. st = ic->streams[i];
39. av\_dlog(ic, "%d: start\_time: %0.3f duration: %0.3f\n", i,
40. (**double**) st->start\_time / AV\_TIME\_BASE,
41. (**double**) st->duration   / AV\_TIME\_BASE);
42. }
43. av\_dlog(ic,
44. "stream: start\_time: %0.3f duration: %0.3f bitrate=%d kb/s\n",
45. (**double**) ic->start\_time / AV\_TIME\_BASE,
46. (**double**) ic->duration   / AV\_TIME\_BASE,
47. ic->bit\_rate / 1000);
48. }
49. }

从estimate\_timings()的代码中可以看出，有3种估算方法：  
（1）通过pts（显示时间戳）。该方法调用estimate\_timings\_from\_pts()。它的基本思想就是读取视音频流中的结束位置AVPacket的PTS和起始位置AVPacket的PTS，两者相减得到时长信息。  
（2）通过已知流的时长。该方法调用fill\_all\_stream\_timings()。它的代码没有细看，但从函数的注释的意思来说，应该是当有些视音频流有时长信息的时候，直接赋值给其他视音频流。  
（3）通过bitrate（码率）。该方法调用estimate\_timings\_from\_bit\_rate()。它的基本思想就是获得整个文件大小，以及整个文件的bitrate，两者相除之后得到时长信息。

### estimate\_timings\_from\_bit\_rate()

在这里附上上述几种方法中最简单的函数estimate\_timings\_from\_bit\_rate()的代码。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44084321) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44084321)

[在CODE上查看代码片](https://code.csdn.net/snippets/613078)

1. **static** **void** estimate\_timings\_from\_bit\_rate(AVFormatContext \*ic)
2. {
3. int64\_t filesize, duration;
4. **int** i, show\_warning = 0;
5. AVStream \*st;

8. /\* if bit\_rate is already set, we believe it \*/
9. **if** (ic->bit\_rate <= 0) {
10. **int** bit\_rate = 0;
11. **for** (i = 0; i < ic->nb\_streams; i++) {
12. st = ic->streams[i];
13. **if** (st->codec->bit\_rate > 0) {
14. **if** (INT\_MAX - st->codec->bit\_rate < bit\_rate) {
15. bit\_rate = 0;
16. **break**;
17. }
18. bit\_rate += st->codec->bit\_rate;
19. }
20. }
21. ic->bit\_rate = bit\_rate;
22. }

25. /\* if duration is already set, we believe it \*/
26. **if** (ic->duration == AV\_NOPTS\_VALUE &&
27. ic->bit\_rate != 0) {
28. filesize = ic->pb ? avio\_size(ic->pb) : 0;
29. **if** (filesize > ic->data\_offset) {
30. filesize -= ic->data\_offset;
31. **for** (i = 0; i < ic->nb\_streams; i++) {
32. st      = ic->streams[i];
33. **if** (   st->time\_base.num <= INT64\_MAX / ic->bit\_rate
34. && st->duration == AV\_NOPTS\_VALUE) {
35. duration = av\_rescale(8 \* filesize, st->time\_base.den,
36. ic->bit\_rate \*
37. (int64\_t) st->time\_base.num);
38. st->duration = duration;
39. show\_warning = 1;
40. }
41. }
42. }
43. }
44. **if** (show\_warning)
45. av\_log(ic, AV\_LOG\_WARNING,
46. "Estimating duration from bitrate, this may be inaccurate\n");
47. }

从代码中可以看出，该函数做了两步工作：  
（1）如果AVFormatContext中没有bit\_rate信息，就把所有AVStream的bit\_rate加起来作为AVFormatContext的bit\_rate信息。  
（2）使用文件大小filesize除以bitrate得到时长信息。具体的方法是：

**AVStream->duration=(filesize\*8/bit\_rate)/time\_base**

PS：  
1）filesize乘以8是因为需要把Byte转换为Bit  
2）具体的实现函数是那个av\_rescale()函数。x=av\_rescale(a,b,c)的含义是x=a\*b/c。  
3）之所以要除以time\_base，是因为AVStream中的duration的单位是time\_base，注意这和AVFormatContext中的duration的单位（单位是AV\_TIME\_BASE，固定取值为1000000）是不一样的。

至此，avformat\_find\_stream\_info()主要的函数就分析完了。

# [ffmpeg 源代码简单分析 ： av\_read\_frame()](http://blog.csdn.net/leixiaohua1020/article/details/12678577)

ffmpeg中的av\_read\_frame()的作用是读取码流中的音频若干帧或者视频一帧。例如，解码视频的时候，每解码一个视频帧，需要先调用 av\_read\_frame()获得一帧视频的压缩数据，然后才能对该数据进行解码（例如H.264中一帧压缩数据通常对应一个NAL）。

对该函数源代码的分析是很久之前做的了，现在翻出来，用博客记录一下。

上代码之前，先参考了其他人对av\_read\_frame()的解释，在此做一个参考：

通过av\_read\_packet(\*\*\*)，读取一个包，需要说明的是此函数必须是包含整数帧的，不存在半帧的情况，以ts流为例，是读取一个完整的PES包（一个完整pes包包含若干视频或音频es包），读取完毕后，通过av\_parser\_parse2(\*\*\*)分析出视频一帧（或音频若干帧），返回，下次进入循环的时候，如果上次的数据没有完全取完，则st = s->cur\_st;不会是NULL，即再此进入av\_parser\_parse2(\*\*\*)流程，而不是下面的av\_read\_packet（\*\*）流程，这样就保证了，如果读取一次包含了N帧视频数据（以视频为例），则调用av\_read\_frame（\*\*\*）N次都不会去读数据，而是返回第一次读取的数据，直到全部解析完毕。

av\_read\_frame()的声明位于libavformat\avformat.h，如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/12678577) [copy](http://blog.csdn.net/leixiaohua1020/article/details/12678577)

1. /\*\*
2. \* Return the next frame of a stream.
3. \* This function returns what is stored in the file, and does not validate
4. \* that what is there are valid frames for the decoder. It will split what is
5. \* stored in the file into frames and return one for each call. It will not
6. \* omit invalid data between valid frames so as to give the decoder the maximum
7. \* information possible for decoding.
8. \*
9. \* If pkt->buf is NULL, then the packet is valid until the next
10. \* av\_read\_frame() or until avformat\_close\_input(). Otherwise the packet
11. \* is valid indefinitely. In both cases the packet must be freed with
12. \* av\_free\_packet when it is no longer needed. For video, the packet contains
13. \* exactly one frame. For audio, it contains an integer number of frames if each
14. \* frame has a known fixed size (e.g. PCM or ADPCM data). If the audio frames
15. \* have a variable size (e.g. MPEG audio), then it contains one frame.
16. \*
17. \* pkt->pts, pkt->dts and pkt->duration are always set to correct
18. \* values in AVStream.time\_base units (and guessed if the format cannot
19. \* provide them). pkt->pts can be AV\_NOPTS\_VALUE if the video format
20. \* has B-frames, so it is better to rely on pkt->dts if you do not
21. \* decompress the payload.
22. \*
23. \* @return 0 if OK, < 0 on error or end of file
24. \*/
25. **int** av\_read\_frame(AVFormatContext \*s, AVPacket \*pkt);

av\_read\_frame()使用方法在注释中写得很详细，用中文简单描述一下它的两个参数：

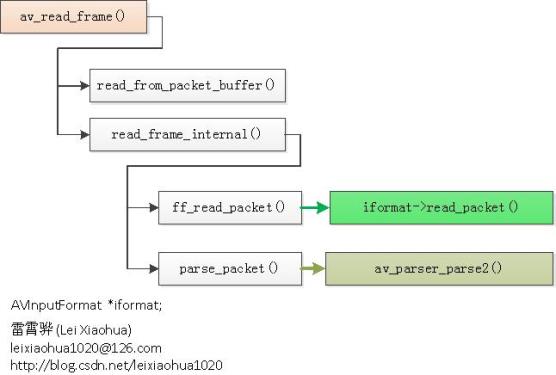
s：输入的AVFormatContext

pkt：输出的AVPacket

如果返回0则说明读取正常。

## 函数调用结构图

函数调用结构图如下所示。



## av\_read\_frame()

av\_read\_frame()的定义位于libavformat\utils.c，如下所示：

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/12678577) [copy](http://blog.csdn.net/leixiaohua1020/article/details/12678577)

1. //获取一个AVPacket
2. /\*
3. \* av\_read\_frame - 新版本的ffmpeg用的是av\_read\_frame，而老版本的是av\_read\_packet
4. \* 。区别是av\_read\_packet读出的是包，它可能是半帧或多帧，不保证帧的完整性。av\_read\_frame对
5. \* av\_read\_packet进行了封装，使读出的数据总是完整的帧
6. \*/
7. **int** av\_read\_frame(AVFormatContext \*s, AVPacket \*pkt)
8. {
9. **const** **int** genpts = s->flags & AVFMT\_FLAG\_GENPTS;
10. **int**          eof = 0;
12. **if** (!genpts)
13. /\*\*
14. \* This buffer is only needed when packets were already buffered but
15. \* not decoded, for example to get the codec parameters in MPEG
16. \* streams.
17. \* 一般情况下会调用read\_frame\_internal(s, pkt)
18. \* 直接返回
19. \*/
20. **return** s->packet\_buffer ? read\_from\_packet\_buffer(s, pkt) :
21. read\_frame\_internal(s, pkt);
23. **for** (;;) {
24. **int** ret;
25. AVPacketList \*pktl = s->packet\_buffer;
27. **if** (pktl) {
28. AVPacket \*next\_pkt = &pktl->pkt;
30. **if** (next\_pkt->dts != AV\_NOPTS\_VALUE) {
31. **int** wrap\_bits = s->streams[next\_pkt->stream\_index]->pts\_wrap\_bits;
32. **while** (pktl && next\_pkt->pts == AV\_NOPTS\_VALUE) {
33. **if** (pktl->pkt.stream\_index == next\_pkt->stream\_index &&
34. (av\_compare\_mod(next\_pkt->dts, pktl->pkt.dts, 2LL << (wrap\_bits - 1)) < 0) &&
35. av\_compare\_mod(pktl->pkt.pts, pktl->pkt.dts, 2LL << (wrap\_bits - 1))) { //not b frame
36. next\_pkt->pts = pktl->pkt.dts;
37. }
38. pktl = pktl->next;
39. }
40. pktl = s->packet\_buffer;
41. }
43. /\* read packet from packet buffer, if there is data \*/
44. **if** (!(next\_pkt->pts == AV\_NOPTS\_VALUE &&
45. next\_pkt->dts != AV\_NOPTS\_VALUE && !eof))
46. **return** read\_from\_packet\_buffer(s, pkt);
47. }
49. ret = read\_frame\_internal(s, pkt);
50. **if** (ret < 0) {
51. **if** (pktl && ret != AVERROR(EAGAIN)) {
52. eof = 1;
53. **continue**;
54. } **else**
55. **return** ret;
56. }
58. **if** (av\_dup\_packet(add\_to\_pktbuf(&s->packet\_buffer, pkt,
59. &s->packet\_buffer\_end)) < 0)
60. **return** AVERROR(ENOMEM);
61. }
62. }

可以从源代码中看出，av\_read\_frame()调用了read\_frame\_internal()。

## read\_frame\_internal()

read\_frame\_internal()代码如下所示：

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/12678577) [copy](http://blog.csdn.net/leixiaohua1020/article/details/12678577)

[在CODE上查看代码片](https://code.csdn.net/snippets/617485)

1. //av\_read\_frame对他进行了封装
2. **static** **int** read\_frame\_internal(AVFormatContext \*s, AVPacket \*pkt)
3. {
4. **int** ret = 0, i, got\_packet = 0;
5. AVDictionary \*metadata = NULL;
6. //初始化
7. av\_init\_packet(pkt);
9. **while** (!got\_packet && !s->parse\_queue) {
10. AVStream \*st;
11. AVPacket cur\_pkt;
13. /\* read next packet \*/
14. ret = ff\_read\_packet(s, &cur\_pkt);
15. **if** (ret < 0) {
16. **if** (ret == AVERROR(EAGAIN))
17. **return** ret;
18. /\* flush the parsers \*/
19. **for** (i = 0; i < s->nb\_streams; i++) {
20. st = s->streams[i];
21. //需要解析
22. **if** (st->parser && st->need\_parsing)
23. parse\_packet(s, NULL, st->index);
24. }
25. /\* all remaining packets are now in parse\_queue =>
26. \* really terminate parsing \*/
27. **break**;
28. }
29. ret = 0;
30. st  = s->streams[cur\_pkt.stream\_index];
32. **if** (cur\_pkt.pts != AV\_NOPTS\_VALUE &&
33. cur\_pkt.dts != AV\_NOPTS\_VALUE &&
34. cur\_pkt.pts < cur\_pkt.dts) {
35. av\_log(s, AV\_LOG\_WARNING,
36. "Invalid timestamps stream=%d, pts=%s, dts=%s, size=%d\n",
37. cur\_pkt.stream\_index,
38. av\_ts2str(cur\_pkt.pts),
39. av\_ts2str(cur\_pkt.dts),
40. cur\_pkt.size);
41. }
42. **if** (s->debug & FF\_FDEBUG\_TS)
43. av\_log(s, AV\_LOG\_DEBUG,
44. "ff\_read\_packet stream=%d, pts=%s, dts=%s, size=%d, duration=%d, flags=%d\n",
45. cur\_pkt.stream\_index,
46. av\_ts2str(cur\_pkt.pts),
47. av\_ts2str(cur\_pkt.dts),
48. cur\_pkt.size, cur\_pkt.duration, cur\_pkt.flags);
50. **if** (st->need\_parsing && !st->parser && !(s->flags & AVFMT\_FLAG\_NOPARSE)) {
51. st->parser = av\_parser\_init(st->codec->codec\_id);
52. **if** (!st->parser) {
53. av\_log(s, AV\_LOG\_VERBOSE, "parser not found for codec "
54. "%s, packets or times may be invalid.\n",
55. avcodec\_get\_name(st->codec->codec\_id));
56. /\* no parser available: just output the raw packets \*/
57. st->need\_parsing = AVSTREAM\_PARSE\_NONE;
58. } **else** **if** (st->need\_parsing == AVSTREAM\_PARSE\_HEADERS)
59. st->parser->flags |= PARSER\_FLAG\_COMPLETE\_FRAMES;
60. **else** **if** (st->need\_parsing == AVSTREAM\_PARSE\_FULL\_ONCE)
61. st->parser->flags |= PARSER\_FLAG\_ONCE;
62. **else** **if** (st->need\_parsing == AVSTREAM\_PARSE\_FULL\_RAW)
63. st->parser->flags |= PARSER\_FLAG\_USE\_CODEC\_TS;
64. }
65. **if** (!st->need\_parsing || !st->parser) {
66. /\* no parsing needed: we just output the packet as is \*/
67. \*pkt = cur\_pkt;
68. compute\_pkt\_fields(s, st, NULL, pkt);
69. **if** ((s->iformat->flags & AVFMT\_GENERIC\_INDEX) &&
70. (pkt->flags & AV\_PKT\_FLAG\_KEY) && pkt->dts != AV\_NOPTS\_VALUE) {
71. ff\_reduce\_index(s, st->index);
72. av\_add\_index\_entry(st, pkt->pos, pkt->dts,
73. 0, 0, AVINDEX\_KEYFRAME);
74. }
75. got\_packet = 1;
76. } **else** **if** (st->discard < AVDISCARD\_ALL) {
77. **if** ((ret = parse\_packet(s, &cur\_pkt, cur\_pkt.stream\_index)) < 0)
78. **return** ret;
79. } **else** {
80. /\* free packet \*/
81. av\_free\_packet(&cur\_pkt);
82. }
83. **if** (pkt->flags & AV\_PKT\_FLAG\_KEY)
84. st->skip\_to\_keyframe = 0;
85. **if** (st->skip\_to\_keyframe) {
86. av\_free\_packet(&cur\_pkt);
87. **if** (got\_packet) {
88. \*pkt = cur\_pkt;
89. }
90. got\_packet = 0;
91. }
92. }
94. **if** (!got\_packet && s->parse\_queue)
95. ret = read\_from\_packet\_buffer(&s->parse\_queue, &s->parse\_queue\_end, pkt);
97. **if** (ret >= 0) {
98. AVStream \*st = s->streams[pkt->stream\_index];
99. **int** discard\_padding = 0;
100. **if** (st->first\_discard\_sample && pkt->pts != AV\_NOPTS\_VALUE) {
101. int64\_t pts = pkt->pts - (is\_relative(pkt->pts) ? RELATIVE\_TS\_BASE : 0);
102. int64\_t sample = ts\_to\_samples(st, pts);
103. **int** duration = ts\_to\_samples(st, pkt->duration);
104. int64\_t end\_sample = sample + duration;
105. **if** (duration > 0 && end\_sample >= st->first\_discard\_sample &&
106. sample < st->last\_discard\_sample)
107. discard\_padding = FFMIN(end\_sample - st->first\_discard\_sample, duration);
108. }
109. **if** (st->skip\_samples || discard\_padding) {
110. uint8\_t \*p = av\_packet\_new\_side\_data(pkt, AV\_PKT\_DATA\_SKIP\_SAMPLES, 10);
111. **if** (p) {
112. AV\_WL32(p, st->skip\_samples);
113. AV\_WL32(p + 4, discard\_padding);
114. av\_log(s, AV\_LOG\_DEBUG, "demuxer injecting skip %d\n", st->skip\_samples);
115. }
116. st->skip\_samples = 0;
117. }
119. **if** (st->inject\_global\_side\_data) {
120. **for** (i = 0; i < st->nb\_side\_data; i++) {
121. AVPacketSideData \*src\_sd = &st->side\_data[i];
122. uint8\_t \*dst\_data;
124. **if** (av\_packet\_get\_side\_data(pkt, src\_sd->type, NULL))
125. **continue**;
127. dst\_data = av\_packet\_new\_side\_data(pkt, src\_sd->type, src\_sd->size);
128. **if** (!dst\_data) {
129. av\_log(s, AV\_LOG\_WARNING, "Could not inject global side data\n");
130. **continue**;
131. }
133. memcpy(dst\_data, src\_sd->data, src\_sd->size);
134. }
135. st->inject\_global\_side\_data = 0;
136. }
138. **if** (!(s->flags & AVFMT\_FLAG\_KEEP\_SIDE\_DATA))
139. av\_packet\_merge\_side\_data(pkt);
140. }
142. av\_opt\_get\_dict\_val(s, "metadata", AV\_OPT\_SEARCH\_CHILDREN, &metadata);
143. **if** (metadata) {
144. s->event\_flags |= AVFMT\_EVENT\_FLAG\_METADATA\_UPDATED;
145. av\_dict\_copy(&s->metadata, metadata, 0);
146. av\_dict\_free(&metadata);
147. av\_opt\_set\_dict\_val(s, "metadata", NULL, AV\_OPT\_SEARCH\_CHILDREN);
148. }
150. **if** (s->debug & FF\_FDEBUG\_TS)
151. av\_log(s, AV\_LOG\_DEBUG,
152. "read\_frame\_internal stream=%d, pts=%s, dts=%s, "
153. "size=%d, duration=%d, flags=%d\n",
154. pkt->stream\_index,
155. av\_ts2str(pkt->pts),
156. av\_ts2str(pkt->dts),
157. pkt->size, pkt->duration, pkt->flags);
159. **return** ret;
160. }

read\_frame\_internal()代码比较长，这里只简单看一下它前面的部分。它前面部分有2步是十分关键的：

（1）调用了ff\_read\_packet()从相应的AVInputFormat读取数据。

（2）如果媒体频流需要使用AVCodecParser，则调用parse\_packet()解析相应的AVPacket。

下面我们分成分别看一下ff\_read\_packet()和parse\_packet()的源代码。

### ff\_read\_packet()

ff\_read\_packet()的代码比较长，如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/12678577) [copy](http://blog.csdn.net/leixiaohua1020/article/details/12678577)

[在CODE上查看代码片](https://code.csdn.net/snippets/617485)

1. **int** ff\_read\_packet(AVFormatContext \*s, AVPacket \*pkt)
2. {
3. **int** ret, i, err;
4. AVStream \*st;
6. **for** (;;) {
7. AVPacketList \*pktl = s->raw\_packet\_buffer;
9. **if** (pktl) {
10. \*pkt = pktl->pkt;
11. st   = s->streams[pkt->stream\_index];
12. **if** (s->raw\_packet\_buffer\_remaining\_size <= 0)
13. **if** ((err = probe\_codec(s, st, NULL)) < 0)
14. **return** err;
15. **if** (st->request\_probe <= 0) {
16. s->raw\_packet\_buffer                 = pktl->next;
17. s->raw\_packet\_buffer\_remaining\_size += pkt->size;
18. av\_free(pktl);
19. **return** 0;
20. }
21. }
23. pkt->data = NULL;
24. pkt->size = 0;
25. av\_init\_packet(pkt);
26. //关键：读取Packet
27. ret = s->iformat->read\_packet(s, pkt);
28. **if** (ret < 0) {
29. **if** (!pktl || ret == AVERROR(EAGAIN))
30. **return** ret;
31. **for** (i = 0; i < s->nb\_streams; i++) {
32. st = s->streams[i];
33. **if** (st->probe\_packets)
34. **if** ((err = probe\_codec(s, st, NULL)) < 0)
35. **return** err;
36. av\_assert0(st->request\_probe <= 0);
37. }
38. **continue**;
39. }
41. **if** ((s->flags & AVFMT\_FLAG\_DISCARD\_CORRUPT) &&
42. (pkt->flags & AV\_PKT\_FLAG\_CORRUPT)) {
43. av\_log(s, AV\_LOG\_WARNING,
44. "Dropped corrupted packet (stream = %d)\n",
45. pkt->stream\_index);
46. av\_free\_packet(pkt);
47. **continue**;
48. }
50. **if** (pkt->stream\_index >= (unsigned)s->nb\_streams) {
51. av\_log(s, AV\_LOG\_ERROR, "Invalid stream index %d\n", pkt->stream\_index);
52. **continue**;
53. }
55. st = s->streams[pkt->stream\_index];
57. **if** (update\_wrap\_reference(s, st, pkt->stream\_index, pkt) && st->pts\_wrap\_behavior == AV\_PTS\_WRAP\_SUB\_OFFSET) {
58. // correct first time stamps to negative values
59. **if** (!is\_relative(st->first\_dts))
60. st->first\_dts = wrap\_timestamp(st, st->first\_dts);
61. **if** (!is\_relative(st->start\_time))
62. st->start\_time = wrap\_timestamp(st, st->start\_time);
63. **if** (!is\_relative(st->cur\_dts))
64. st->cur\_dts = wrap\_timestamp(st, st->cur\_dts);
65. }
67. pkt->dts = wrap\_timestamp(st, pkt->dts);
68. pkt->pts = wrap\_timestamp(st, pkt->pts);
70. force\_codec\_ids(s, st);
72. /\* TODO: audio: time filter; video: frame reordering (pts != dts) \*/
73. **if** (s->use\_wallclock\_as\_timestamps)
74. pkt->dts = pkt->pts = av\_rescale\_q(av\_gettime(), AV\_TIME\_BASE\_Q, st->time\_base);
76. **if** (!pktl && st->request\_probe <= 0)
77. **return** ret;
79. add\_to\_pktbuf(&s->raw\_packet\_buffer, pkt, &s->raw\_packet\_buffer\_end);
80. s->raw\_packet\_buffer\_remaining\_size -= pkt->size;
82. **if** ((err = probe\_codec(s, st, pkt)) < 0)
83. **return** err;
84. }
85. }

ff\_read\_packet()中最关键的地方就是调用了AVInputFormat的read\_packet()方法。AVInputFormat的read\_packet()是一个函数指针，指向当前的AVInputFormat的读取数据的函数。在这里我们以FLV封装格式对应的AVInputFormat为例，看看read\_packet()的实现函数是什么样子的。

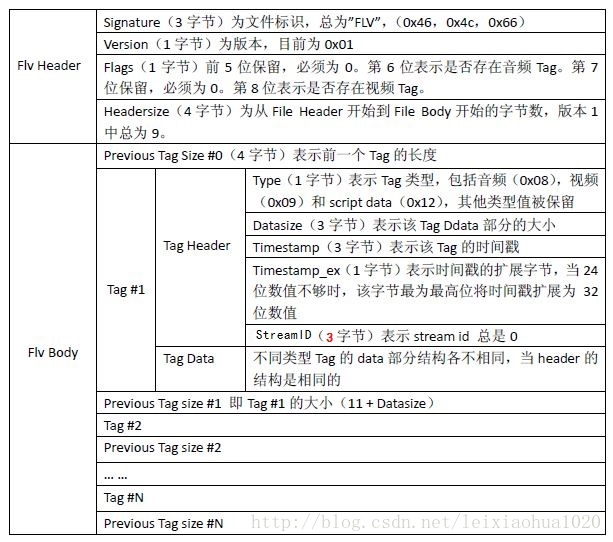
FLV封装格式对应的AVInputFormat的定义位于libavformat\flvdec.c，如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/12678577) [copy](http://blog.csdn.net/leixiaohua1020/article/details/12678577)

[在CODE上查看代码片](https://code.csdn.net/snippets/617485)

1. AVInputFormat ff\_flv\_demuxer = {
2. .name           = "flv",
3. .long\_name      = NULL\_IF\_CONFIG\_SMALL("FLV (Flash Video)"),
4. .priv\_data\_size = **sizeof**(FLVContext),
5. .read\_probe     = flv\_probe,
6. .read\_header    = flv\_read\_header,
7. .read\_packet    = flv\_read\_packet,
8. .read\_seek      = flv\_read\_seek,
9. .read\_close     = flv\_read\_close,
10. .extensions     = "flv",
11. .priv\_class     = &flv\_class,
12. };

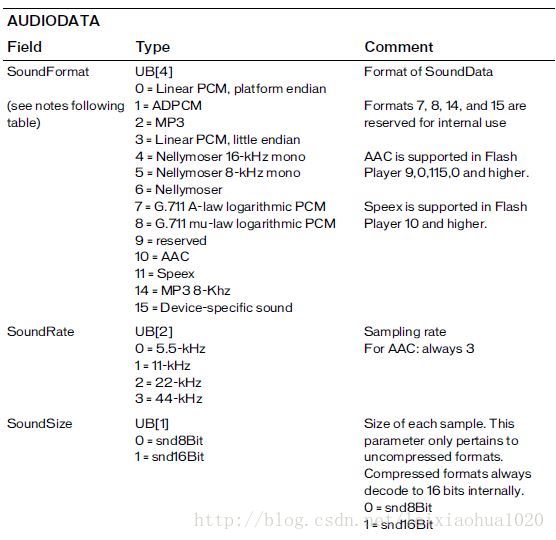
从ff\_flv\_demuxer的定义可以看出，read\_packet()对应的是flv\_read\_packet()函数。在看flv\_read\_packet()函数之前，我们先回顾一下FLV封装格式的结构，如下图所示。



从图中可以看出，FLV文件体部分是由一个一个的Tag连接起来的（中间间隔着Previous Tag Size）。每个Tag包含了Tag Header和Tag Data两个部分。Tag Data根据Tag的Type不同而不同：可以分为音频Tag Data，视频Tag Data以及Script Tag Data。下面简述一下音频Tag Data和视频Tag Data。

#### Audio Tag Data

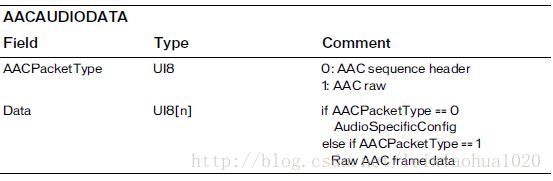
Audio Tag在官方标准中定义如下。



Audio Tag开始的第1个字节包含了音频数据的参数信息，从第2个字节开始为音频流数据。  
第1个字节的前4位的数值表示了音频数据格式：

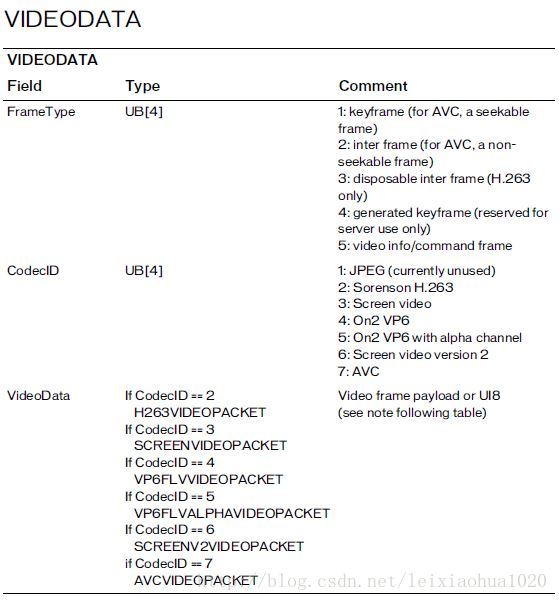
0 = Linear PCM, platform endian  
1 = ADPCM  
2 = MP3  
3 = Linear PCM, little endian  
4 = Nellymoser 16-kHz mono  
5 = Nellymoser 8-kHz mono  
6 = Nellymoser  
7 = G.711 A-law logarithmic PCM  
8 = G.711 mu-law logarithmic PCM  
9 = reserved  
10 = AAC  
14 = MP3 8-Khz  
15 = Device-specific sound

第1个字节的第5-6位的数值表示采样率：0 = 5.5kHz，1 = 11KHz，2 = 22 kHz，3 = 44 kHz。  
第1个字节的第7位表示采样精度：0 = 8bits，1 = 16bits。  
第1个字节的第8位表示音频类型：0 = sndMono，1 = sndStereo。  
其中，当音频编码为AAC的时候，第一个字节后面存储的是AACAUDIODATA，格式如下所示。



#### Video Tag Data

Video Tag在官方标准中的定义如下。



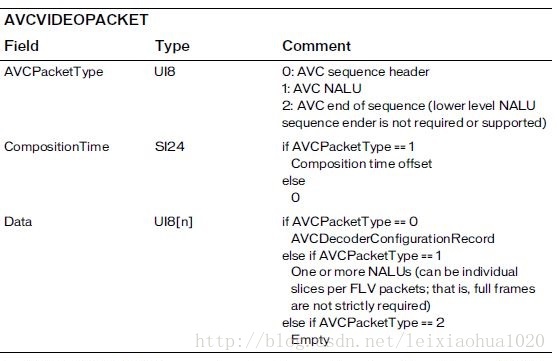
Video Tag也用开始的第1个字节包含视频数据的参数信息，从第2个字节为视频流数据。  
第1个字节的前4位的数值表示帧类型（FrameType）：

1: keyframe (for AVC, a seekableframe)（关键帧）  
2: inter frame (for AVC, a nonseekableframe)  
3: disposable inter frame (H.263only)  
4: generated keyframe (reservedfor server use only)  
5: video info/command frame

第1个字节的后4位的数值表示视频编码ID（CodecID）：

1: JPEG (currently unused)  
2: Sorenson H.263  
3: Screen video  
4: On2 VP6  
5: On2 VP6 with alpha channel  
6: Screen video version 2  
7: AVC

其中，当音频编码为AVC（H.264）的时候，第一个字节后面存储的是AVCVIDEOPACKET，格式如下所示。



了解了FLV的基本格式之后，就可以看一下FLV解析Tag的函数flv\_read\_packet()了。

### flv\_read\_packet()

flv\_read\_packet()的定义位于libavformat\flvdec.c，如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/12678577) [copy](http://blog.csdn.net/leixiaohua1020/article/details/12678577)

[在CODE上查看代码片](https://code.csdn.net/snippets/617485)

1. **static** **int** flv\_read\_packet(AVFormatContext \*s, AVPacket \*pkt)
2. {
3. FLVContext \*flv = s->priv\_data;
4. **int** ret, i, type, size, flags;
5. **int** stream\_type=-1;
6. int64\_t next, pos, meta\_pos;
7. int64\_t dts, pts = AV\_NOPTS\_VALUE;
8. **int** av\_uninit(channels);
9. **int** av\_uninit(sample\_rate);
10. AVStream \*st    = NULL;
12. /\* pkt size is repeated at end. skip it \*/
13. **for** (;; avio\_skip(s->pb, 4)) {
14. pos  = avio\_tell(s->pb);
15. //解析Tag Header==========
16. //Tag类型
17. type = (avio\_r8(s->pb) & 0x1F);
18. //Datasize数据大小
19. size = avio\_rb24(s->pb);
20. //Timstamp时间戳
21. dts  = avio\_rb24(s->pb);
22. dts |= avio\_r8(s->pb) << 24;
23. av\_dlog(s, "type:%d, size:%d, dts:%"PRId64" pos:%"PRId64"\n", type, size, dts, avio\_tell(s->pb));
24. **if** (avio\_feof(s->pb))
25. **return** AVERROR\_EOF;
26. //StreamID
27. avio\_skip(s->pb, 3); /\* stream id, always 0 \*/
28. flags = 0;
29. //========================
30. **if** (flv->validate\_next < flv->validate\_count) {
31. int64\_t validate\_pos = flv->validate\_index[flv->validate\_next].pos;
32. **if** (pos == validate\_pos) {
33. **if** (FFABS(dts - flv->validate\_index[flv->validate\_next].dts) <=
34. VALIDATE\_INDEX\_TS\_THRESH) {
35. flv->validate\_next++;
36. } **else** {
37. clear\_index\_entries(s, validate\_pos);
38. flv->validate\_count = 0;
39. }
40. } **else** **if** (pos > validate\_pos) {
41. clear\_index\_entries(s, validate\_pos);
42. flv->validate\_count = 0;
43. }
44. }
46. **if** (size == 0)
47. **continue**;
49. next = size + avio\_tell(s->pb);
51. **if** (type == FLV\_TAG\_TYPE\_AUDIO) {
52. //Type是音频
53. stream\_type = FLV\_STREAM\_TYPE\_AUDIO;
54. //Tag Data的第一个字节
55. flags    = avio\_r8(s->pb);
56. size--;
57. } **else** **if** (type == FLV\_TAG\_TYPE\_VIDEO) {
58. //Type是音频
59. stream\_type = FLV\_STREAM\_TYPE\_VIDEO;
60. //Tag Data的第一个字节
61. flags    = avio\_r8(s->pb);
62. size--;
63. **if** ((flags & FLV\_VIDEO\_FRAMETYPE\_MASK) == FLV\_FRAME\_VIDEO\_INFO\_CMD)
64. **goto** skip;
65. } **else** **if** (type == FLV\_TAG\_TYPE\_META) {
66. stream\_type=FLV\_STREAM\_TYPE\_DATA;
67. **if** (size > 13 + 1 + 4 && dts == 0) { // Header-type metadata stuff
68. meta\_pos = avio\_tell(s->pb);
69. **if** (flv\_read\_metabody(s, next) <= 0) {
70. **goto** skip;
71. }
72. avio\_seek(s->pb, meta\_pos, SEEK\_SET);
73. }
74. } **else** {
75. av\_log(s, AV\_LOG\_DEBUG,
76. "Skipping flv packet: type %d, size %d, flags %d.\n",
77. type, size, flags);
78. skip:
79. avio\_seek(s->pb, next, SEEK\_SET);
80. **continue**;
81. }
83. /\* skip empty data packets \*/
84. **if** (!size)
85. **continue**;
87. /\* now find stream \*/
88. **for** (i = 0; i < s->nb\_streams; i++) {
89. st = s->streams[i];
90. **if** (stream\_type == FLV\_STREAM\_TYPE\_AUDIO) {
91. **if** (st->codec->codec\_type == AVMEDIA\_TYPE\_AUDIO &&
92. (s->audio\_codec\_id || flv\_same\_audio\_codec(st->codec, flags)))
93. **break**;
94. } **else** **if** (stream\_type == FLV\_STREAM\_TYPE\_VIDEO) {
95. **if** (st->codec->codec\_type == AVMEDIA\_TYPE\_VIDEO &&
96. (s->video\_codec\_id || flv\_same\_video\_codec(st->codec, flags)))
97. **break**;
98. } **else** **if** (stream\_type == FLV\_STREAM\_TYPE\_DATA) {
99. **if** (st->codec->codec\_type == AVMEDIA\_TYPE\_DATA)
100. **break**;
101. }
102. }
103. **if** (i == s->nb\_streams) {
104. **static** **const** **enum** AVMediaType stream\_types[] = {AVMEDIA\_TYPE\_VIDEO, AVMEDIA\_TYPE\_AUDIO, AVMEDIA\_TYPE\_DATA};
105. av\_log(s, AV\_LOG\_WARNING, "Stream discovered after head already parsed\n");
106. st = create\_stream(s, stream\_types[stream\_type]);
107. **if** (!st)
108. **return** AVERROR(ENOMEM);
110. }
111. av\_dlog(s, "%d %X %d \n", stream\_type, flags, st->discard);
113. **if** ((flags & FLV\_VIDEO\_FRAMETYPE\_MASK) == FLV\_FRAME\_KEY ||
114. stream\_type == FLV\_STREAM\_TYPE\_AUDIO)
115. av\_add\_index\_entry(st, pos, dts, size, 0, AVINDEX\_KEYFRAME);
117. **if** (  (st->discard >= AVDISCARD\_NONKEY && !((flags & FLV\_VIDEO\_FRAMETYPE\_MASK) == FLV\_FRAME\_KEY || (stream\_type == FLV\_STREAM\_TYPE\_AUDIO)))
118. ||(st->discard >= AVDISCARD\_BIDIR  &&  ((flags & FLV\_VIDEO\_FRAMETYPE\_MASK) == FLV\_FRAME\_DISP\_INTER && (stream\_type == FLV\_STREAM\_TYPE\_VIDEO)))
119. || st->discard >= AVDISCARD\_ALL
120. ) {
121. avio\_seek(s->pb, next, SEEK\_SET);
122. **continue**;
123. }
124. **break**;
125. }
127. // if not streamed and no duration from metadata then seek to end to find
128. // the duration from the timestamps
129. **if** (s->pb->seekable && (!s->duration || s->duration == AV\_NOPTS\_VALUE) && !flv->searched\_for\_end) {
130. **int** size;
131. **const** int64\_t pos   = avio\_tell(s->pb);
132. // Read the last 4 bytes of the file, this should be the size of the
133. // previous FLV tag. Use the timestamp of its payload as duration.
134. int64\_t fsize       = avio\_size(s->pb);
135. retry\_duration:
136. avio\_seek(s->pb, fsize - 4, SEEK\_SET);
137. size = avio\_rb32(s->pb);
138. // Seek to the start of the last FLV tag at position (fsize - 4 - size)
139. // but skip the byte indicating the type.
140. avio\_seek(s->pb, fsize - 3 - size, SEEK\_SET);
141. **if** (size == avio\_rb24(s->pb) + 11) {
142. uint32\_t ts = avio\_rb24(s->pb);
143. ts         |= avio\_r8(s->pb) << 24;
144. **if** (ts)
145. s->duration = ts \* (int64\_t)AV\_TIME\_BASE / 1000;
146. **else** **if** (fsize >= 8 && fsize - 8 >= size) {
147. fsize -= size+4;
148. **goto** retry\_duration;
149. }
150. }
152. avio\_seek(s->pb, pos, SEEK\_SET);
153. flv->searched\_for\_end = 1;
154. }
156. **if** (stream\_type == FLV\_STREAM\_TYPE\_AUDIO) {
157. **int** bits\_per\_coded\_sample;
158. channels = (flags & FLV\_AUDIO\_CHANNEL\_MASK) == FLV\_STEREO ? 2 : 1;
159. sample\_rate = 44100 << ((flags & FLV\_AUDIO\_SAMPLERATE\_MASK) >>
160. FLV\_AUDIO\_SAMPLERATE\_OFFSET) >> 3;
161. bits\_per\_coded\_sample = (flags & FLV\_AUDIO\_SAMPLESIZE\_MASK) ? 16 : 8;
162. **if** (!st->codec->channels || !st->codec->sample\_rate ||
163. !st->codec->bits\_per\_coded\_sample) {
164. st->codec->channels              = channels;
165. st->codec->channel\_layout        = channels == 1
166. ? AV\_CH\_LAYOUT\_MONO
167. : AV\_CH\_LAYOUT\_STEREO;
168. st->codec->sample\_rate           = sample\_rate;
169. st->codec->bits\_per\_coded\_sample = bits\_per\_coded\_sample;
170. }
171. **if** (!st->codec->codec\_id) {
172. flv\_set\_audio\_codec(s, st, st->codec,
173. flags & FLV\_AUDIO\_CODECID\_MASK);
174. flv->last\_sample\_rate =
175. sample\_rate           = st->codec->sample\_rate;
176. flv->last\_channels    =
177. channels              = st->codec->channels;
178. } **else** {
179. AVCodecContext ctx = {0};
180. ctx.sample\_rate = sample\_rate;
181. ctx.bits\_per\_coded\_sample = bits\_per\_coded\_sample;
182. flv\_set\_audio\_codec(s, st, &ctx, flags & FLV\_AUDIO\_CODECID\_MASK);
183. sample\_rate = ctx.sample\_rate;
184. }
185. } **else** **if** (stream\_type == FLV\_STREAM\_TYPE\_VIDEO) {
186. size -= flv\_set\_video\_codec(s, st, flags & FLV\_VIDEO\_CODECID\_MASK, 1);
187. }
188. //几种特殊的格式
189. **if** (st->codec->codec\_id == AV\_CODEC\_ID\_AAC ||
190. st->codec->codec\_id == AV\_CODEC\_ID\_H264 ||
191. st->codec->codec\_id == AV\_CODEC\_ID\_MPEG4) {
192. //对应AACPacketType或者AVCPacketType
193. **int** type = avio\_r8(s->pb);
194. size--;
195. //H.264
196. **if** (st->codec->codec\_id == AV\_CODEC\_ID\_H264 || st->codec->codec\_id == AV\_CODEC\_ID\_MPEG4) {
197. // sign extension
198. //对应CompositionTime
199. int32\_t cts = (avio\_rb24(s->pb) + 0xff800000) ^ 0xff800000;
200. //计算PTS
201. pts = dts + cts;
202. **if** (cts < 0) { // dts might be wrong
203. **if** (!flv->wrong\_dts)
204. av\_log(s, AV\_LOG\_WARNING,
205. "Negative cts, previous timestamps might be wrong.\n");
206. flv->wrong\_dts = 1;
207. } **else** **if** (FFABS(dts - pts) > 1000\*60\*15) {
208. av\_log(s, AV\_LOG\_WARNING,
209. "invalid timestamps %"PRId64" %"PRId64"\n", dts, pts);
210. dts = pts = AV\_NOPTS\_VALUE;
211. }
212. }
213. //如果编码器是AAC或者H.264
214. **if** (type == 0 && (!st->codec->extradata || st->codec->codec\_id == AV\_CODEC\_ID\_AAC ||
215. st->codec->codec\_id == AV\_CODEC\_ID\_H264)) {
216. AVDictionaryEntry \*t;
218. **if** (st->codec->extradata) {
219. **if** ((ret = flv\_queue\_extradata(flv, s->pb, stream\_type, size)) < 0)
220. **return** ret;
221. ret = AVERROR(EAGAIN);
222. **goto** leave;
223. }
224. **if** ((ret = flv\_get\_extradata(s, st, size)) < 0)
225. **return** ret;
227. /\* Workaround for buggy Omnia A/XE encoder \*/
228. t = av\_dict\_get(s->metadata, "Encoder", NULL, 0);
229. **if** (st->codec->codec\_id == AV\_CODEC\_ID\_AAC && t && !strcmp(t->value, "Omnia A/XE"))
230. st->codec->extradata\_size = 2;
231. //AAC
232. **if** (st->codec->codec\_id == AV\_CODEC\_ID\_AAC && 0) {
233. MPEG4AudioConfig cfg;
235. **if** (avpriv\_mpeg4audio\_get\_config(&cfg, st->codec->extradata,
236. st->codec->extradata\_size \* 8, 1) >= 0) {
237. st->codec->channels       = cfg.channels;
238. st->codec->channel\_layout = 0;
239. **if** (cfg.ext\_sample\_rate)
240. st->codec->sample\_rate = cfg.ext\_sample\_rate;
241. **else**
242. st->codec->sample\_rate = cfg.sample\_rate;
243. av\_dlog(s, "mp4a config channels %d sample rate %d\n",
244. st->codec->channels, st->codec->sample\_rate);
245. }
246. }
248. ret = AVERROR(EAGAIN);
249. **goto** leave;
250. }
251. }
253. /\* skip empty data packets \*/
254. **if** (!size) {
255. ret = AVERROR(EAGAIN);
256. **goto** leave;
257. }
259. ret = av\_get\_packet(s->pb, pkt, size);
260. **if** (ret < 0)
261. **return** ret;
262. //设置PTS、DTS等等
263. pkt->dts          = dts;
264. pkt->pts          = pts == AV\_NOPTS\_VALUE ? dts : pts;
265. pkt->stream\_index = st->index;
266. **if** (flv->new\_extradata[stream\_type]) {
267. uint8\_t \*side = av\_packet\_new\_side\_data(pkt, AV\_PKT\_DATA\_NEW\_EXTRADATA,
268. flv->new\_extradata\_size[stream\_type]);
269. **if** (side) {
270. memcpy(side, flv->new\_extradata[stream\_type],
271. flv->new\_extradata\_size[stream\_type]);
272. av\_freep(&flv->new\_extradata[stream\_type]);
273. flv->new\_extradata\_size[stream\_type] = 0;
274. }
275. }
276. **if** (stream\_type == FLV\_STREAM\_TYPE\_AUDIO &&
277. (sample\_rate != flv->last\_sample\_rate ||
278. channels    != flv->last\_channels)) {
279. flv->last\_sample\_rate = sample\_rate;
280. flv->last\_channels    = channels;
281. ff\_add\_param\_change(pkt, channels, 0, sample\_rate, 0, 0);
282. }
283. //标记上Keyframe
284. **if** (    stream\_type == FLV\_STREAM\_TYPE\_AUDIO ||
285. ((flags & FLV\_VIDEO\_FRAMETYPE\_MASK) == FLV\_FRAME\_KEY) ||
286. stream\_type == FLV\_STREAM\_TYPE\_DATA)
287. pkt->flags |= AV\_PKT\_FLAG\_KEY;
289. leave:
290. avio\_skip(s->pb, 4);
291. **return** ret;
292. }

flv\_read\_packet()的代码比较长，但是逻辑比较简单。它的主要功能就是根据FLV文件格式的规范，逐层解析Tag以及TagData，获取Tag以及TagData中的信息。比较关键的地方已经写上了注释，不再详细叙述。

## parse\_packet()

parse\_packet()给需要AVCodecParser的媒体流提供解析AVPacket的功能。它的代码如下所示：

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/12678577) [copy](http://blog.csdn.net/leixiaohua1020/article/details/12678577)

[在CODE上查看代码片](https://code.csdn.net/snippets/617485)

1. /\*\*
2. \* Parse a packet, add all split parts to parse\_queue.
3. \*
4. \* @param pkt Packet to parse, NULL when flushing the parser at end of stream.
5. \*/
6. **static** **int** parse\_packet(AVFormatContext \*s, AVPacket \*pkt, **int** stream\_index)
7. {
8. AVPacket out\_pkt = { 0 }, flush\_pkt = { 0 };
9. AVStream \*st = s->streams[stream\_index];
10. uint8\_t \*data = pkt ? pkt->data : NULL;
11. **int** size      = pkt ? pkt->size : 0;
12. **int** ret = 0, got\_output = 0;
14. **if** (!pkt) {
15. av\_init\_packet(&flush\_pkt);
16. pkt        = &flush\_pkt;
17. got\_output = 1;
18. } **else** **if** (!size && st->parser->flags & PARSER\_FLAG\_COMPLETE\_FRAMES) {
19. // preserve 0-size sync packets
20. compute\_pkt\_fields(s, st, st->parser, pkt);
21. }
23. **while** (size > 0 || (pkt == &flush\_pkt && got\_output)) {
24. **int** len;
26. av\_init\_packet(&out\_pkt);
27. //解析
28. len = av\_parser\_parse2(st->parser, st->codec,
29. &out\_pkt.data, &out\_pkt.size, data, size,
30. pkt->pts, pkt->dts, pkt->pos);
32. pkt->pts = pkt->dts = AV\_NOPTS\_VALUE;
33. pkt->pos = -1;
34. /\* increment read pointer \*/
35. data += len;
36. size -= len;
38. got\_output = !!out\_pkt.size;
39. //继续
40. **if** (!out\_pkt.size)
41. **continue**;
43. **if** (pkt->side\_data) {
44. out\_pkt.side\_data       = pkt->side\_data;
45. out\_pkt.side\_data\_elems = pkt->side\_data\_elems;
46. pkt->side\_data          = NULL;
47. pkt->side\_data\_elems    = 0;
48. }
50. /\* set the duration \*/
51. out\_pkt.duration = 0;
52. **if** (st->codec->codec\_type == AVMEDIA\_TYPE\_AUDIO) {
53. **if** (st->codec->sample\_rate > 0) {
54. out\_pkt.duration =
55. av\_rescale\_q\_rnd(st->parser->duration,
56. (AVRational) { 1, st->codec->sample\_rate },
57. st->time\_base,
58. AV\_ROUND\_DOWN);
59. }
60. }
61. //设置属性值
62. out\_pkt.stream\_index = st->index;
63. out\_pkt.pts          = st->parser->pts;
64. out\_pkt.dts          = st->parser->dts;
65. out\_pkt.pos          = st->parser->pos;
67. **if** (st->need\_parsing == AVSTREAM\_PARSE\_FULL\_RAW)
68. out\_pkt.pos = st->parser->frame\_offset;
70. **if** (st->parser->key\_frame == 1 ||
71. (st->parser->key\_frame == -1 &&
72. st->parser->pict\_type == AV\_PICTURE\_TYPE\_I))
73. out\_pkt.flags |= AV\_PKT\_FLAG\_KEY;
75. **if** (st->parser->key\_frame == -1 && st->parser->pict\_type ==AV\_PICTURE\_TYPE\_NONE && (pkt->flags&AV\_PKT\_FLAG\_KEY))
76. out\_pkt.flags |= AV\_PKT\_FLAG\_KEY;
78. compute\_pkt\_fields(s, st, st->parser, &out\_pkt);
80. **if** (out\_pkt.data == pkt->data && out\_pkt.size == pkt->size) {
81. out\_pkt.buf = pkt->buf;
82. pkt->buf    = NULL;
83. #if FF\_API\_DESTRUCT\_PACKET
84. FF\_DISABLE\_DEPRECATION\_WARNINGS
85. out\_pkt.destruct = pkt->destruct;
86. pkt->destruct = NULL;
87. FF\_ENABLE\_DEPRECATION\_WARNINGS
88. #endif
89. }
90. **if** ((ret = av\_dup\_packet(&out\_pkt)) < 0)
91. **goto** fail;
93. **if** (!add\_to\_pktbuf(&s->parse\_queue, &out\_pkt, &s->parse\_queue\_end)) {
94. av\_free\_packet(&out\_pkt);
95. ret = AVERROR(ENOMEM);
96. **goto** fail;
97. }
98. }
100. /\* end of the stream => close and free the parser \*/
101. **if** (pkt == &flush\_pkt) {
102. av\_parser\_close(st->parser);
103. st->parser = NULL;
104. }
106. fail:
107. av\_free\_packet(pkt);
108. **return** ret;
109. }

从代码中可以看出，最终调用了相应AVCodecParser的av\_parser\_parse2()函数，解析出来AVPacket。此后根据解析的信息还进行了一系列的赋值工作，不再详细叙述。

# [ffmpeg 源代码简单分析 ： avcodec\_decode\_video2()](http://blog.csdn.net/leixiaohua1020/article/details/12679719)

ffmpeg中的avcodec\_decode\_video2()的作用是解码一帧视频数据。输入一个压缩编码的结构体AVPacket，输出一个解码后的结构体AVFrame。该函数的声明位于libavcodec\avcodec.h，如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/12679719) [copy](http://blog.csdn.net/leixiaohua1020/article/details/12679719)

1. /\*\*
2. \* Decode the video frame of size avpkt->size from avpkt->data into picture.
3. \* Some decoders may support multiple frames in a single AVPacket, such
4. \* decoders would then just decode the first frame.
5. \*
6. \* @warning The input buffer must be FF\_INPUT\_BUFFER\_PADDING\_SIZE larger than
7. \* the actual read bytes because some optimized bitstream readers read 32 or 64
8. \* bits at once and could read over the end.
9. \*
10. \* @warning The end of the input buffer buf should be set to 0 to ensure that
11. \* no overreading happens for damaged MPEG streams.
12. \*
13. \* @note Codecs which have the CODEC\_CAP\_DELAY capability set have a delay
14. \* between input and output, these need to be fed with avpkt->data=NULL,
15. \* avpkt->size=0 at the end to return the remaining frames.
16. \*
17. \* @param avctx the codec context
18. \* @param[out] picture The AVFrame in which the decoded video frame will be stored.
19. \*             Use av\_frame\_alloc() to get an AVFrame. The codec will
20. \*             allocate memory for the actual bitmap by calling the
21. \*             AVCodecContext.get\_buffer2() callback.
22. \*             When AVCodecContext.refcounted\_frames is set to 1, the frame is
23. \*             reference counted and the returned reference belongs to the
24. \*             caller. The caller must release the frame using av\_frame\_unref()
25. \*             when the frame is no longer needed. The caller may safely write
26. \*             to the frame if av\_frame\_is\_writable() returns 1.
27. \*             When AVCodecContext.refcounted\_frames is set to 0, the returned
28. \*             reference belongs to the decoder and is valid only until the
29. \*             next call to this function or until closing or flushing the
30. \*             decoder. The caller may not write to it.
31. \*
32. \* @param[in] avpkt The input AVPacket containing the input buffer.
33. \*            You can create such packet with av\_init\_packet() and by then setting
34. \*            data and size, some decoders might in addition need other fields like
35. \*            flags&AV\_PKT\_FLAG\_KEY. All decoders are designed to use the least
36. \*            fields possible.
37. \* @param[in,out] got\_picture\_ptr Zero if no frame could be decompressed, otherwise, it is nonzero.
38. \* @return On error a negative value is returned, otherwise the number of bytes
39. \* used or zero if no frame could be decompressed.
40. \*/
41. **int** avcodec\_decode\_video2(AVCodecContext \*avctx, AVFrame \*picture,
42. **int** \*got\_picture\_ptr,
43. **const** AVPacket \*avpkt);

查看源代码之后发现，这个函数竟然十分的简单，源代码位于libavcodec\utils.c，如下所示：

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/12679719) [copy](http://blog.csdn.net/leixiaohua1020/article/details/12679719)

[在CODE上查看代码片](https://code.csdn.net/snippets/617456)

1. **int** attribute\_align\_arg avcodec\_decode\_video2(AVCodecContext \*avctx, AVFrame \*picture,
2. **int** \*got\_picture\_ptr,
3. **const** AVPacket \*avpkt)
4. {
5. AVCodecInternal \*avci = avctx->internal;
6. **int** ret;
7. // copy to ensure we do not change avpkt
8. AVPacket tmp = \*avpkt;
10. **if** (!avctx->codec)
11. **return** AVERROR(EINVAL);
12. //检查是不是视频（非音频）
13. **if** (avctx->codec->type != AVMEDIA\_TYPE\_VIDEO) {
14. av\_log(avctx, AV\_LOG\_ERROR, "Invalid media type for video\n");
15. **return** AVERROR(EINVAL);
16. }
18. \*got\_picture\_ptr = 0;
19. //检查宽、高设置是否正确
20. **if** ((avctx->coded\_width || avctx->coded\_height) && av\_image\_check\_size(avctx->coded\_width, avctx->coded\_height, 0, avctx))
21. **return** AVERROR(EINVAL);
23. av\_frame\_unref(picture);
25. **if** ((avctx->codec->capabilities & CODEC\_CAP\_DELAY) || avpkt->size || (avctx->active\_thread\_type & FF\_THREAD\_FRAME)) {
26. **int** did\_split = av\_packet\_split\_side\_data(&tmp);
27. ret = apply\_param\_change(avctx, &tmp);
28. **if** (ret < 0) {
29. av\_log(avctx, AV\_LOG\_ERROR, "Error applying parameter changes.\n");
30. **if** (avctx->err\_recognition & AV\_EF\_EXPLODE)
31. **goto** fail;
32. }
34. avctx->internal->pkt = &tmp;
35. **if** (HAVE\_THREADS && avctx->active\_thread\_type & FF\_THREAD\_FRAME)
36. ret = ff\_thread\_decode\_frame(avctx, picture, got\_picture\_ptr,
37. &tmp);
38. **else** {
39. //最关键的解码函数
40. ret = avctx->codec->decode(avctx, picture, got\_picture\_ptr,
41. &tmp);
42. //设置pkt\_dts字段的值
43. picture->pkt\_dts = avpkt->dts;
45. **if**(!avctx->has\_b\_frames){
46. av\_frame\_set\_pkt\_pos(picture, avpkt->pos);
47. }
48. //FIXME these should be under if(!avctx->has\_b\_frames)
49. /\* get\_buffer is supposed to set frame parameters \*/
50. **if** (!(avctx->codec->capabilities & CODEC\_CAP\_DR1)) {
51. //对一些字段进行赋值
52. **if** (!picture->sample\_aspect\_ratio.num)    picture->sample\_aspect\_ratio = avctx->sample\_aspect\_ratio;
53. **if** (!picture->width)                      picture->width               = avctx->width;
54. **if** (!picture->height)                     picture->height              = avctx->height;
55. **if** (picture->format == AV\_PIX\_FMT\_NONE)   picture->format              = avctx->pix\_fmt;
56. }
57. }
58. add\_metadata\_from\_side\_data(avctx, picture);
60. fail:
61. emms\_c(); //needed to avoid an emms\_c() call before every return;
63. avctx->internal->pkt = NULL;
64. **if** (did\_split) {
65. av\_packet\_free\_side\_data(&tmp);
66. **if**(ret == tmp.size)
67. ret = avpkt->size;
68. }
70. **if** (\*got\_picture\_ptr) {
71. **if** (!avctx->refcounted\_frames) {
72. **int** err = unrefcount\_frame(avci, picture);
73. **if** (err < 0)
74. **return** err;
75. }
77. avctx->frame\_number++;
78. av\_frame\_set\_best\_effort\_timestamp(picture,
79. guess\_correct\_pts(avctx,
80. picture->pkt\_pts,
81. picture->pkt\_dts));
82. } **else**
83. av\_frame\_unref(picture);
84. } **else**
85. ret = 0;
87. /\* many decoders assign whole AVFrames, thus overwriting extended\_data;
88. \* make sure it's set correctly \*/
89. av\_assert0(!picture->extended\_data || picture->extended\_data == picture->data);
91. #if FF\_API\_AVCTX\_TIMEBASE
92. **if** (avctx->framerate.num > 0 && avctx->framerate.den > 0)
93. avctx->time\_base = av\_inv\_q(av\_mul\_q(avctx->framerate, (AVRational){avctx->ticks\_per\_frame, 1}));
94. #endif
96. **return** ret;
97. }

从代码中可以看出，avcodec\_decode\_video2()主要做了以下几个方面的工作：

（1）对输入的字段进行了一系列的检查工作：例如宽高是否正确，输入是否为视频等等。

（2）通过ret = avctx->codec->decode(avctx, picture, got\_picture\_ptr,&tmp)这句代码，调用了相应AVCodec的decode()函数，完成了解码操作。

（3）对得到的AVFrame的一些字段进行了赋值，例如宽高、像素格式等等。

其中第二部是关键的一步，它调用了AVCodec的decode()方法完成了解码。AVCodec的decode()方法是一个函数指针，指向了具体解码器的解码函数。在这里我们以H.264解码器为例，看一下解码的实现过程。H.264解码器对应的AVCodec的定义位于libavcodec\h264.c，如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/12679719) [copy](http://blog.csdn.net/leixiaohua1020/article/details/12679719)

[在CODE上查看代码片](https://code.csdn.net/snippets/617456)

1. AVCodec ff\_h264\_decoder = {
2. .name                  = "h264",
3. .long\_name             = NULL\_IF\_CONFIG\_SMALL("H.264 / AVC / MPEG-4 AVC / MPEG-4 part 10"),
4. .type                  = AVMEDIA\_TYPE\_VIDEO,
5. .id                    = AV\_CODEC\_ID\_H264,
6. .priv\_data\_size        = **sizeof**(H264Context),
7. .init                  = ff\_h264\_decode\_init,
8. .close                 = h264\_decode\_end,
9. .decode                = h264\_decode\_frame,
10. .capabilities          = /\*CODEC\_CAP\_DRAW\_HORIZ\_BAND |\*/ CODEC\_CAP\_DR1 |
11. CODEC\_CAP\_DELAY | CODEC\_CAP\_SLICE\_THREADS |
12. CODEC\_CAP\_FRAME\_THREADS,
13. .flush                 = flush\_dpb,
14. .init\_thread\_copy      = ONLY\_IF\_THREADS\_ENABLED(decode\_init\_thread\_copy),
15. .update\_thread\_context = ONLY\_IF\_THREADS\_ENABLED(ff\_h264\_update\_thread\_context),
16. .profiles              = NULL\_IF\_CONFIG\_SMALL(profiles),
17. .priv\_class            = &h264\_class,
18. };

从ff\_h264\_decoder的定义可以看出，decode()指向了h264\_decode\_frame()函数。继续看一下h264\_decode\_frame()函数的定义，如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/12679719) [copy](http://blog.csdn.net/leixiaohua1020/article/details/12679719)

[在CODE上查看代码片](https://code.csdn.net/snippets/617456)

1. **static** **int** h264\_decode\_frame(AVCodecContext \*avctx, **void** \*data,
2. **int** \*got\_frame, AVPacket \*avpkt)
3. {
4. **const** uint8\_t \*buf = avpkt->data;
5. **int** buf\_size       = avpkt->size;
6. H264Context \*h     = avctx->priv\_data;
7. AVFrame \*pict      = data;
8. **int** buf\_index      = 0;
9. H264Picture \*out;
10. **int** i, out\_idx;
11. **int** ret;
13. h->flags = avctx->flags;
14. /\* reset data partitioning here, to ensure GetBitContexts from previous
15. \* packets do not get used. \*/
16. h->data\_partitioning = 0;
18. /\* end of stream, output what is still in the buffers \*/
19. **if** (buf\_size == 0) {
20. out:
22. h->cur\_pic\_ptr = NULL;
23. h->first\_field = 0;
25. // FIXME factorize this with the output code below
26. out     = h->delayed\_pic[0];
27. out\_idx = 0;
28. **for** (i = 1;
29. h->delayed\_pic[i] &&
30. !h->delayed\_pic[i]->f.key\_frame &&
31. !h->delayed\_pic[i]->mmco\_reset;
32. i++)
33. **if** (h->delayed\_pic[i]->poc < out->poc) {
34. out     = h->delayed\_pic[i];
35. out\_idx = i;
36. }
38. **for** (i = out\_idx; h->delayed\_pic[i]; i++)
39. h->delayed\_pic[i] = h->delayed\_pic[i + 1];
41. **if** (out) {
42. out->reference &= ~DELAYED\_PIC\_REF;
43. ret = output\_frame(h, pict, out);
44. **if** (ret < 0)
45. **return** ret;
46. \*got\_frame = 1;
47. }
49. **return** buf\_index;
50. }
51. **if** (h->is\_avc && av\_packet\_get\_side\_data(avpkt, AV\_PKT\_DATA\_NEW\_EXTRADATA, NULL)) {
52. **int** side\_size;
53. uint8\_t \*side = av\_packet\_get\_side\_data(avpkt, AV\_PKT\_DATA\_NEW\_EXTRADATA, &side\_size);
54. **if** (is\_extra(side, side\_size))
55. ff\_h264\_decode\_extradata(h, side, side\_size);
56. }
57. **if**(h->is\_avc && buf\_size >= 9 && buf[0]==1 && buf[2]==0 && (buf[4]&0xFC)==0xFC && (buf[5]&0x1F) && buf[8]==0x67){
58. **if** (is\_extra(buf, buf\_size))
59. **return** ff\_h264\_decode\_extradata(h, buf, buf\_size);
60. }
61. //H.264解码
62. buf\_index = decode\_nal\_units(h, buf, buf\_size, 0);
63. **if** (buf\_index < 0)
64. **return** AVERROR\_INVALIDDATA;
66. **if** (!h->cur\_pic\_ptr && h->nal\_unit\_type == NAL\_END\_SEQUENCE) {
67. av\_assert0(buf\_index <= buf\_size);
68. **goto** out;
69. }
71. **if** (!(avctx->flags2 & CODEC\_FLAG2\_CHUNKS) && !h->cur\_pic\_ptr) {
72. **if** (avctx->skip\_frame >= AVDISCARD\_NONREF ||
73. buf\_size >= 4 && !memcmp("Q264", buf, 4))
74. **return** buf\_size;
75. av\_log(avctx, AV\_LOG\_ERROR, "no frame!\n");
76. **return** AVERROR\_INVALIDDATA;
77. }
79. **if** (!(avctx->flags2 & CODEC\_FLAG2\_CHUNKS) ||
80. (h->mb\_y >= h->mb\_height && h->mb\_height)) {
81. **if** (avctx->flags2 & CODEC\_FLAG2\_CHUNKS)
82. decode\_postinit(h, 1);
84. ff\_h264\_field\_end(h, 0);
86. /\* Wait for second field. \*/
87. \*got\_frame = 0;
88. **if** (h->next\_output\_pic && (
89. h->next\_output\_pic->recovered)) {
90. **if** (!h->next\_output\_pic->recovered)
91. h->next\_output\_pic->f.flags |= AV\_FRAME\_FLAG\_CORRUPT;
93. ret = output\_frame(h, pict, h->next\_output\_pic);
94. **if** (ret < 0)
95. **return** ret;
96. \*got\_frame = 1;
97. **if** (CONFIG\_MPEGVIDEO) {
98. ff\_print\_debug\_info2(h->avctx, pict, h->er.mbskip\_table,
99. h->next\_output\_pic->mb\_type,
100. h->next\_output\_pic->qscale\_table,
101. h->next\_output\_pic->motion\_val,
102. &h->low\_delay,
103. h->mb\_width, h->mb\_height, h->mb\_stride, 1);
104. }
105. }
106. }
108. assert(pict->buf[0] || !\*got\_frame);
110. **return** get\_consumed\_bytes(buf\_index, buf\_size);
111. }

从h264\_decode\_frame()的定义可以看出，它调用了decode\_nal\_units()完成了具体的H.264解码工作。有关H.264解码就不在详细分析了。

# [FFmpeg源代码简单分析：avformat\_close\_input()](http://blog.csdn.net/leixiaohua1020/article/details/44110683)

本文简单分析FFmpeg的avformat\_close\_input()函数。该函数用于关闭一个AVFormatContext，一般情况下是和avformat\_open\_input()成对使用的。

avformat\_close\_input()的声明位于libavformat\avformat.h，如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44110683) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44110683)

[在CODE上查看代码片](https://code.csdn.net/snippets/613733)

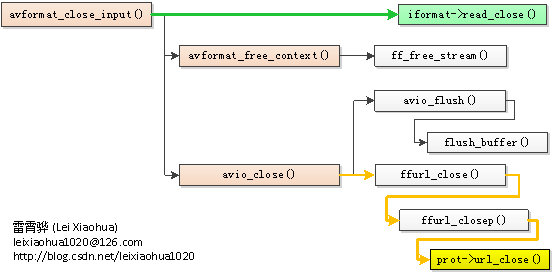
1. /\*\*
2. \* Close an opened input AVFormatContext. Free it and all its contents
3. \* and set \*s to NULL.
4. \*/
5. **void** avformat\_close\_input(AVFormatContext \*\*s);

该函数最典型的例子可以参考：

[最简单的基于FFMPEG+SDL的视频播放器 ver2 （采用SDL2.0）](http://blog.csdn.net/leixiaohua1020/article/details/38868499)

## 函数调用关系图

函数的调用关系如下图所示。



## avformat\_close\_input()

下面看一下avformat\_close\_input()的源代码，位于libavformat\utils.c文件中。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44110683) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44110683)

[在CODE上查看代码片](https://code.csdn.net/snippets/613733)

1. **void** avformat\_close\_input(AVFormatContext \*\*ps)
2. {
3. AVFormatContext \*s;
4. AVIOContext \*pb;
6. **if** (!ps || !\*ps)
7. **return**;
9. s  = \*ps;
10. pb = s->pb;
12. **if** ((s->iformat && strcmp(s->iformat->name, "image2") && s->iformat->flags & AVFMT\_NOFILE) ||
13. (s->flags & AVFMT\_FLAG\_CUSTOM\_IO))
14. pb = NULL;
16. flush\_packet\_queue(s);
18. **if** (s->iformat)
19. **if** (s->iformat->read\_close)
20. s->iformat->read\_close(s);
22. avformat\_free\_context(s);
24. \*ps = NULL;
26. avio\_close(pb);
27. }

从源代码中可以看出，avformat\_close\_input()主要做了以下几步工作：

（1）调用AVInputFormat的read\_close()方法关闭输入流  
（2）调用avformat\_free\_context()释放AVFormatContext  
（3）调用avio\_close()关闭并且释放AVIOContext

下面我们分别来看上述几个步骤。

## AVInputFormat-> read\_close()

AVInputFormat的read\_close()是一个函数指针，指向关闭输入流的函数。不同的AVInputFormat包含有不同的read\_close()方法。例如，FLV格式对应的AVInputFormat的定义如下。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44110683) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44110683)

[在CODE上查看代码片](https://code.csdn.net/snippets/613733)

1. AVInputFormat ff\_flv\_demuxer = {
2. .name           = "flv",
3. .long\_name      = NULL\_IF\_CONFIG\_SMALL("FLV (Flash Video)"),
4. .priv\_data\_size = **sizeof**(FLVContext),
5. .read\_probe     = flv\_probe,
6. .read\_header    = flv\_read\_header,
7. .read\_packet    = flv\_read\_packet,
8. .read\_seek      = flv\_read\_seek,
9. .read\_close     = flv\_read\_close,
10. .extensions     = "flv",
11. .priv\_class     = &flv\_class,
12. };

从ff\_flv\_demuxer的定义中可以看出，read\_close()指向的函数是flv\_read\_close()。我们可以看一下flv\_read\_close()的定义，如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44110683) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44110683)

[在CODE上查看代码片](https://code.csdn.net/snippets/613733)

1. **static** **int** flv\_read\_close(AVFormatContext \*s)
2. {
3. **int** i;
4. FLVContext \*flv = s->priv\_data;
5. **for** (i=0; i<FLV\_STREAM\_TYPE\_NB; i++)
6. av\_freep(&flv->new\_extradata[i]);
7. **return** 0;
8. }

从flv\_read\_close()的定义可以看出，该函数释放了FLVContext中的new\_extradata数组中每个元素指向的内存。

## avformat\_free\_context()

avformat\_free\_context()是一个FFmpeg的API函数，用于释放一个AVFormatContext。在这里要注意搞清楚avformat\_free\_context()和avformat\_close\_input()之间的区别与联系。

有关avformat\_free\_context()可以参考文章：

[FFmpeg源代码简单分析：内存的分配和释放](http://blog.csdn.net/leixiaohua1020/article/details/41176777)

## avio\_close()

avio\_close()是一个FFmpeg的API函数，用于关闭和释放AVIOContext。它的声明位于libavformat\avio.h，如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44110683) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44110683)

[在CODE上查看代码片](https://code.csdn.net/snippets/613733)

1. /\*\*
2. \* Close the resource accessed by the AVIOContext s and free it.
3. \* This function can only be used if s was opened by avio\_open().
4. \*
5. \* The internal buffer is automatically flushed before closing the
6. \* resource.
7. \*
8. \* @return 0 on success, an AVERROR < 0 on error.
9. \* @see avio\_closep
10. \*/
11. **int** avio\_close(AVIOContext \*s);

avio\_close()的定义位于libavformat\aviobuf.c，如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44110683) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44110683)

[在CODE上查看代码片](https://code.csdn.net/snippets/613733)

1. **int** avio\_close(AVIOContext \*s)
2. {
3. URLContext \*h;
5. **if** (!s)
6. **return** 0;
8. avio\_flush(s);
9. h = s->opaque;
10. av\_freep(&s->buffer);
11. **if** (s->write\_flag)
12. av\_log(s, AV\_LOG\_DEBUG, "Statistics: %d seeks, %d writeouts\n", s->seek\_count, s->writeout\_count);
13. **else**
14. av\_log(s, AV\_LOG\_DEBUG, "Statistics: %"PRId64" bytes read, %d seeks\n", s->bytes\_read, s->seek\_count);
15. av\_free(s);
16. **return** ffurl\_close(h);
17. }

从源代码可以看出，avio\_close()按照顺序做了以下几个步骤：

（1）调用avio\_flush()强制清除缓存中的数据  
（2）调用av\_freep()释放掉AVIOContext种的buffer  
（3）调用av\_free()释放掉AVIOContext结构体  
（4）调用ffurl\_close()关闭并且释放掉URLContext

下面按照顺序分别看看avio\_flush()和ffurl\_close()这两个函数。

### avio\_flush()

avio\_flush()是一个FFmpeg的API函数，声明位于libavformat\avio.h，如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44110683) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44110683)

[在CODE上查看代码片](https://code.csdn.net/snippets/613733)

1. /\*\*
2. \* Force flushing of buffered data.
3. \*
4. \* For write streams, force the buffered data to be immediately written to the output,
5. \* without to wait to fill the internal buffer.
6. \*
7. \* For read streams, discard all currently buffered data, and advance the
8. \* reported file position to that of the underlying stream. This does not
9. \* read new data, and does not perform any seeks.
10. \*/
11. **void** avio\_flush(AVIOContext \*s);

avio\_flush()的定义位于libavformat\aviobuf.c，如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44110683) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44110683)

[在CODE上查看代码片](https://code.csdn.net/snippets/613733)

1. **void** avio\_flush(AVIOContext \*s)
2. {
3. flush\_buffer(s);
4. s->must\_flush = 0;
5. }

可以看出avio\_flush()简单调用了flush\_buffer()函数。我们看一下flush\_buffer()的定义。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44110683) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44110683)

[在CODE上查看代码片](https://code.csdn.net/snippets/613733)

1. **static** **void** flush\_buffer(AVIOContext \*s)
2. {
3. **if** (s->write\_flag && s->buf\_ptr > s->buffer) {
4. writeout(s, s->buffer, s->buf\_ptr - s->buffer);
5. **if** (s->update\_checksum) {
6. s->checksum     = s->update\_checksum(s->checksum, s->checksum\_ptr,
7. s->buf\_ptr - s->checksum\_ptr);
8. s->checksum\_ptr = s->buffer;
9. }
10. }
11. s->buf\_ptr = s->buffer;
12. **if** (!s->write\_flag)
13. s->buf\_end = s->buffer;
14. }

从flush\_buffer()定义我们可以看出，该函数将当前缓存指针buf\_ptr的位置重新设置到缓存buffer的首部，然后根据AVIOContext对应的流是否可写分别做不同的处理。如果AVIOContext对应的流是只读的（write\_flag取值为0），就将缓存的尾部buf\_end设定到缓存首部位置；如果AVIOContext对应的流如果是可写的（write\_flag取值非0），则会调用writeout()函数输出缓存中剩余的数据。  
在这里我们看一下writeout()函数的定义，如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44110683) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44110683)

[在CODE上查看代码片](https://code.csdn.net/snippets/613733)

1. **static** **void** writeout(AVIOContext \*s, **const** uint8\_t \*data, **int** len)
2. {
3. **if** (s->write\_packet && !s->error) {
4. **int** ret = s->write\_packet(s->opaque, (uint8\_t \*)data, len);
5. **if** (ret < 0) {
6. s->error = ret;
7. }
8. }
9. s->writeout\_count ++;
10. s->pos += len;
11. }

从定义可以看出，writeout()调用了AVIOContext的write\_packet()方法。根据此前文章《[FFmpeg源代码简单分析：avio\_open2()](http://blog.csdn.net/leixiaohua1020/article/details/41199947)》中的分析我们可以了解到，AVIOContext的write\_packet()实际指向了ffurl\_write()函数，而ffurl\_write()经过retry\_transfer\_wrapper()函数最终调用了URLProtocol的url\_write()函数。url\_write()是一个函数指针，不同的URLProtocol的url\_write()指向不同的函数。

例如，file（文件）对应的URLProtocol的定义位于libavformat\file.c，如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44110683) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44110683)

[在CODE上查看代码片](https://code.csdn.net/snippets/613733)

1. URLProtocol ff\_file\_protocol = {
2. .name                = "file",
3. .url\_open            = file\_open,
4. .url\_read            = file\_read,
5. .url\_write           = file\_write,
6. .url\_seek            = file\_seek,
7. .url\_close           = file\_close,
8. .url\_get\_file\_handle = file\_get\_handle,
9. .url\_check           = file\_check,
10. .priv\_data\_size      = **sizeof**(FileContext),
11. .priv\_data\_class     = &file\_class,
12. };

可以看出ff\_file\_protocol中的url\_write()指向的是file\_write()函数。我们继续看一下file\_write()的源代码，如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44110683) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44110683)

[在CODE上查看代码片](https://code.csdn.net/snippets/613733)

1. **static** **int** file\_write(URLContext \*h, **const** unsigned **char** \*buf, **int** size)
2. {
3. FileContext \*c = h->priv\_data;
4. **int** r;
5. size = FFMIN(size, c->blocksize);
6. r = write(c->fd, buf, size);
7. **return** (-1 == r)?AVERROR(errno):r;
8. }

从源代码中可以看出file\_write()调用了系统的write()方法向文件中写数据（很多人可能对write()函数很陌生，可以简单理解为它等同于fwrite()）。

### ffurl\_close()和ffurl\_closep()

ffurl\_close()和ffurl\_closep()是FFmpeg内部的两个函数，它们的声明位于libavformat\url.h，如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44110683) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44110683)

[在CODE上查看代码片](https://code.csdn.net/snippets/613733)

1. /\*\*
2. \* Close the resource accessed by the URLContext h, and free the
3. \* memory used by it. Also set the URLContext pointer to NULL.
4. \*
5. \* @return a negative value if an error condition occurred, 0
6. \* otherwise
7. \*/
8. **int** ffurl\_closep(URLContext \*\*h);
9. **int** ffurl\_close(URLContext \*h);

其实这两个函数是等同的。ffurl\_close()的定义位于libavformat\avio.c，如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44110683) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44110683)

[在CODE上查看代码片](https://code.csdn.net/snippets/613733)

1. **int** ffurl\_close(URLContext \*h)
2. {
3. **return** ffurl\_closep(&h);
4. }

可见ffurl\_close()调用了ffurl\_closep()。  
ffurl\_closep()的定义如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44110683) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44110683)

[在CODE上查看代码片](https://code.csdn.net/snippets/613733)

1. **int** ffurl\_closep(URLContext \*\*hh)
2. {
3. URLContext \*h= \*hh;
4. **int** ret = 0;
5. **if** (!h)
6. **return** 0;     /\* can happen when ffurl\_open fails \*/
8. **if** (h->is\_connected && h->prot->url\_close)
9. ret = h->prot->url\_close(h);
10. #if CONFIG\_NETWORK
11. **if** (h->prot->flags & URL\_PROTOCOL\_FLAG\_NETWORK)
12. ff\_network\_close();
13. #endif
14. **if** (h->prot->priv\_data\_size) {
15. **if** (h->prot->priv\_data\_class)
16. av\_opt\_free(h->priv\_data);
17. av\_freep(&h->priv\_data);
18. }
19. av\_freep(hh);
20. **return** ret;
21. }

从ffurl\_closep()的定义可以看出，它主要做了两步工作：

（1）调用URLProtocol的url\_close()  
（2）调用av\_freep()释放URLContext结构体

其中URLProtocol的url\_close()是一个函数指针，其指向的函数与具体的URLProtocol有关，这里我们还是看一下file（文件）对应的URLProtocol，如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44110683) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44110683)

[在CODE上查看代码片](https://code.csdn.net/snippets/613733)

1. URLProtocol ff\_file\_protocol = {
2. .name                = "file",
3. .url\_open            = file\_open,
4. .url\_read            = file\_read,
5. .url\_write           = file\_write,
6. .url\_seek            = file\_seek,
7. .url\_close           = file\_close,
8. .url\_get\_file\_handle = file\_get\_handle,
9. .url\_check           = file\_check,
10. .priv\_data\_size      = **sizeof**(FileContext),
11. .priv\_data\_class     = &file\_class,
12. };

从ff\_file\_protocol中可以看出，url\_close()指向file\_close()函数。我们再看一下file\_close()的定义，如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44110683) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44110683)

[在CODE上查看代码片](https://code.csdn.net/snippets/613733)

1. **static** **int** file\_close(URLContext \*h)
2. {
3. FileContext \*c = h->priv\_data;
4. **return** close(c->fd);
5. }

可见file\_close()最终调用了系统函数close()关闭了文件指针（不熟悉close()的可以简单把它理解为fclose()）。

至此avio\_close()函数分析完毕。

# [FFmpeg源代码简单分析：avformat\_alloc\_output\_context2()](http://blog.csdn.net/leixiaohua1020/article/details/41198929)

本文简单分析FFmpeg中常用的一个函数：avformat\_alloc\_output\_context2()。在基于FFmpeg的视音频编码器程序中，该函数通常是第一个调用的函数（除了组件注册函数av\_register\_all()）。avformat\_alloc\_output\_context2()函数可以初始化一个用于输出的AVFormatContext结构体。它的声明位于libavformat\avformat.h，如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/41198929) [copy](http://blog.csdn.net/leixiaohua1020/article/details/41198929)

[在CODE上查看代码片](https://code.csdn.net/snippets/611475)

1. /\*\*
2. \* Allocate an AVFormatContext for an output format.
3. \* avformat\_free\_context() can be used to free the context and
4. \* everything allocated by the framework within it.
5. \*
6. \* @param \*ctx is set to the created format context, or to NULL in
7. \* case of failure
8. \* @param oformat format to use for allocating the context, if NULL
9. \* format\_name and filename are used instead
10. \* @param format\_name the name of output format to use for allocating the
11. \* context, if NULL filename is used instead
12. \* @param filename the name of the filename to use for allocating the
13. \* context, may be NULL
14. \* @return >= 0 in case of success, a negative AVERROR code in case of
15. \* failure
16. \*/
17. **int** avformat\_alloc\_output\_context2(AVFormatContext \*\*ctx, AVOutputFormat \*oformat,
18. **const** **char** \*format\_name, **const** **char** \*filename);

代码中的英文注释写的已经比较详细了，在这里拿中文简单叙述一下。

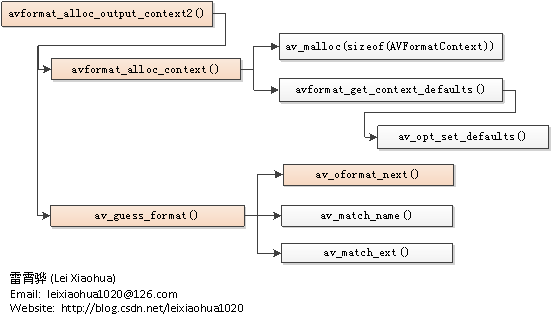
ctx：函数调用成功之后创建的AVFormatContext结构体。  
oformat：指定AVFormatContext中的AVOutputFormat，用于确定输出格式。如果指定为NULL，可以设定后两个参数（format\_name或者filename）由FFmpeg猜测输出格式。  
PS：使用该参数需要自己手动获取AVOutputFormat，相对于使用后两个参数来说要麻烦一些。  
format\_name：指定输出格式的名称。根据格式名称，FFmpeg会推测输出格式。输出格式可以是“flv”，“mkv”等等。  
filename：指定输出文件的名称。根据文件名称，FFmpeg会推测输出格式。文件名称可以是“xx.flv”，“yy.mkv”等等。

函数执行成功的话，其返回值大于等于0。

该函数最典型的例子可以参考：[最简单的基于FFMPEG的视频编码器（YUV编码为H.264）](http://blog.csdn.net/leixiaohua1020/article/details/25430425)

## 函数调用结构图

首先贴出来最终分析得出的函数调用结构图，如下所示。

[](https://my.csdn.net/leixiaohua1020/album/detail/1804803)

[单击查看更清晰图片](https://my.csdn.net/leixiaohua1020/album/detail/1804803)

## avformat\_alloc\_output\_context2()

下面看一下avformat\_alloc\_output\_context2()的函数定义。该函数的定义位于libavformat\mux.c中，如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/41198929) [copy](http://blog.csdn.net/leixiaohua1020/article/details/41198929)

[在CODE上查看代码片](https://code.csdn.net/snippets/611475)

1. **int** avformat\_alloc\_output\_context2(AVFormatContext \*\*avctx, AVOutputFormat \*oformat,
2. **const** **char** \*format, **const** **char** \*filename)
3. {
4. AVFormatContext \*s = avformat\_alloc\_context();
5. **int** ret = 0;

8. \*avctx = NULL;
9. **if** (!s)
10. **goto** nomem;

13. **if** (!oformat) {
14. **if** (format) {
15. oformat = av\_guess\_format(format, NULL, NULL);
16. **if** (!oformat) {
17. av\_log(s, AV\_LOG\_ERROR, "Requested output format '%s' is not a suitable output format\n", format);
18. ret = AVERROR(EINVAL);
19. **goto** error;
20. }
21. } **else** {
22. oformat = av\_guess\_format(NULL, filename, NULL);
23. **if** (!oformat) {
24. ret = AVERROR(EINVAL);
25. av\_log(s, AV\_LOG\_ERROR, "Unable to find a suitable output format for '%s'\n",
26. filename);
27. **goto** error;
28. }
29. }
30. }

33. s->oformat = oformat;
34. **if** (s->oformat->priv\_data\_size > 0) {
35. s->priv\_data = av\_mallocz(s->oformat->priv\_data\_size);
36. **if** (!s->priv\_data)
37. **goto** nomem;
38. **if** (s->oformat->priv\_class) {
39. \*(**const** AVClass\*\*)s->priv\_data= s->oformat->priv\_class;
40. av\_opt\_set\_defaults(s->priv\_data);
41. }
42. } **else**
43. s->priv\_data = NULL;

46. **if** (filename)
47. av\_strlcpy(s->filename, filename, **sizeof**(s->filename));
48. \*avctx = s;
49. **return** 0;
50. nomem:
51. av\_log(s, AV\_LOG\_ERROR, "Out of memory\n");
52. ret = AVERROR(ENOMEM);
53. error:
54. avformat\_free\_context(s);
55. **return** ret;
56. }

从代码中可以看出，avformat\_alloc\_output\_context2()的流程如要包含以下2步：

1) 调用avformat\_alloc\_context()初始化一个默认的AVFormatContext。

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

下面我们分别看看上文步骤中提到的两个重要的函数：avformat\_alloc\_context()和av\_guess\_format()。

## avformat\_alloc\_context()

avformat\_alloc\_context()的是一个FFmpeg的API，它的定义如下。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/41198929) [copy](http://blog.csdn.net/leixiaohua1020/article/details/41198929)

[在CODE上查看代码片](https://code.csdn.net/snippets/611475)

1. AVFormatContext \*avformat\_alloc\_context(**void**)
2. {
3. AVFormatContext \*ic;
4. ic = av\_malloc(**sizeof**(AVFormatContext));
5. **if** (!ic) **return** ic;
6. avformat\_get\_context\_defaults(ic);

9. ic->internal = av\_mallocz(**sizeof**(\*ic->internal));
10. **if** (!ic->internal) {
11. avformat\_free\_context(ic);
12. **return** NULL;
13. }

16. **return** ic;
17. }

从代码中可以看出，avformat\_alloc\_context()首先调用av\_malloc()为AVFormatContext分配一块内存。然后调用了一个函数avformat\_get\_context\_defaults()用于给AVFormatContext设置默认值。avformat\_get\_context\_defaults()的定义如下。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/41198929) [copy](http://blog.csdn.net/leixiaohua1020/article/details/41198929)

[在CODE上查看代码片](https://code.csdn.net/snippets/611475)

1. **static** **void** avformat\_get\_context\_defaults(AVFormatContext \*s)
2. {
3. memset(s, 0, **sizeof**(AVFormatContext));

6. s->av\_class = &av\_format\_context\_class;

9. av\_opt\_set\_defaults(s);
10. }

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

## av\_guess\_format()

av\_guess\_format()是FFmpeg的一个API。它的声明如下。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/41198929) [copy](http://blog.csdn.net/leixiaohua1020/article/details/41198929)

[在CODE上查看代码片](https://code.csdn.net/snippets/611475)

1. /\*\*
2. \* Return the output format in the list of registered output formats
3. \* which best matches the provided parameters, or return NULL if
4. \* there is no match.
5. \*
6. \* @param short\_name if non-NULL checks if short\_name matches with the
7. \* names of the registered formats
8. \* @param filename if non-NULL checks if filename terminates with the
9. \* extensions of the registered formats
10. \* @param mime\_type if non-NULL checks if mime\_type matches with the
11. \* MIME type of the registered formats
12. \*/
13. AVOutputFormat \*av\_guess\_format(**const** **char** \*short\_name,
14. **const** **char** \*filename,
15. **const** **char** \*mime\_type);

拿中文简单解释一下参数。

short\_name：格式的名称。  
filename：文件的名称。  
mime\_type：MIME类型。

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

av\_guess\_format()的代码如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/41198929) [copy](http://blog.csdn.net/leixiaohua1020/article/details/41198929)

[在CODE上查看代码片](https://code.csdn.net/snippets/611475)

1. AVOutputFormat \*av\_guess\_format(**const** **char** \*short\_name, **const** **char** \*filename,
2. **const** **char** \*mime\_type)
3. {
4. AVOutputFormat \*fmt = NULL, \*fmt\_found;
5. **int** score\_max, score;

8. /\* specific test for image sequences \*/
9. #if CONFIG\_IMAGE2\_MUXER
10. **if** (!short\_name && filename &&
11. av\_filename\_number\_test(filename) &&
12. ff\_guess\_image2\_codec(filename) != AV\_CODEC\_ID\_NONE) {
13. **return** av\_guess\_format("image2", NULL, NULL);
14. }
15. #endif
16. /\* Find the proper file type. \*/
17. fmt\_found = NULL;
18. score\_max = 0;
19. **while** ((fmt = av\_oformat\_next(fmt))) {
20. score = 0;
21. **if** (fmt->name && short\_name && av\_match\_name(short\_name, fmt->name))
22. score += 100;
23. **if** (fmt->mime\_type && mime\_type && !strcmp(fmt->mime\_type, mime\_type))
24. score += 10;
25. **if** (filename && fmt->extensions &&
26. av\_match\_ext(filename, fmt->extensions)) {
27. score += 5;
28. }
29. **if** (score > score\_max) {
30. score\_max = score;
31. fmt\_found = fmt;
32. }
33. }
34. **return** fmt\_found;
35. }

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

1) 如果封装格式名称匹配，score增加100。匹配中使用了函数av\_match\_name()。  
2) 如果mime类型匹配，score增加10。匹配直接使用字符串比较函数strcmp()。  
3) 如果文件名称的后缀匹配，score增加5。匹配中使用了函数av\_match\_ext()。

while()循环结束后，得到得分最高的格式，就是最匹配的格式。

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

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/41198929) [copy](http://blog.csdn.net/leixiaohua1020/article/details/41198929)

[在CODE上查看代码片](https://code.csdn.net/snippets/611475)

1. AVOutputFormat ff\_flv\_muxer = {
2. .name           = "flv",
3. .long\_name      = NULL\_IF\_CONFIG\_SMALL("FLV (Flash Video)"),
4. .mime\_type      = "video/x-flv",
5. .extensions     = "flv",
6. .priv\_data\_size = **sizeof**(FLVContext),
7. .audio\_codec    = CONFIG\_LIBMP3LAME ? AV\_CODEC\_ID\_MP3 : AV\_CODEC\_ID\_ADPCM\_SWF,
8. .video\_codec    = AV\_CODEC\_ID\_FLV1,
9. .write\_header   = flv\_write\_header,
10. .write\_packet   = flv\_write\_packet,
11. .write\_trailer  = flv\_write\_trailer,
12. .codec\_tag      = (**const** AVCodecTag\* **const** []) {
13. flv\_video\_codec\_ids, flv\_audio\_codec\_ids, 0
14. },
15. .flags          = AVFMT\_GLOBALHEADER | AVFMT\_VARIABLE\_FPS |
16. AVFMT\_TS\_NONSTRICT,
17. };

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

### av\_oformat\_next()

av\_oformat\_next()是个API函数，声明如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/41198929) [copy](http://blog.csdn.net/leixiaohua1020/article/details/41198929)

1. /\*\*
2. \* If f is NULL, returns the first registered output format,
3. \* if f is non-NULL, returns the next registered output format after f
4. \* or NULL if f is the last one.
5. \*/
6. AVOutputFormat \*av\_oformat\_next(**const** AVOutputFormat \*f);

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

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/41198929) [copy](http://blog.csdn.net/leixiaohua1020/article/details/41198929)

[在CODE上查看代码片](https://code.csdn.net/snippets/611475)

1. AVOutputFormat \*av\_oformat\_next(**const** AVOutputFormat \*f)
2. {
3. **if** (f)
4. **return** f->next;
5. **else**
6. **return** first\_oformat;
7. }

### av\_match\_name()

av\_match\_name()是一个API函数，声明如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/41198929) [copy](http://blog.csdn.net/leixiaohua1020/article/details/41198929)

1. /\*\*
2. \* Match instances of a name in a comma-separated list of names.
3. \* @param name  Name to look for.
4. \* @param names List of names.
5. \* @return 1 on match, 0 otherwise.
6. \*/
7. **int** av\_match\_name(**const** **char** \*name, **const** **char** \*names);

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

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/41198929) [copy](http://blog.csdn.net/leixiaohua1020/article/details/41198929)

[在CODE上查看代码片](https://code.csdn.net/snippets/611475)

1. **int** av\_match\_name(**const** **char** \*name, **const** **char** \*names)
2. {
3. **const** **char** \*p;
4. **int** len, namelen;

7. **if** (!name || !names)
8. **return** 0;

11. namelen = strlen(name);
12. **while** ((p = strchr(names, ','))) {
13. len = FFMAX(p - names, namelen);
14. **if** (!av\_strncasecmp(name, names, len))
15. **return** 1;
16. names = p + 1;
17. }
18. **return** !av\_strcasecmp(name, names);
19. }

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

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/41198929) [copy](http://blog.csdn.net/leixiaohua1020/article/details/41198929)

[在CODE上查看代码片](https://code.csdn.net/snippets/611475)

1. AVInputFormat ff\_matroska\_demuxer = {
2. .name           = "matroska,webm",
3. .long\_name      = NULL\_IF\_CONFIG\_SMALL("Matroska / WebM"),
4. .extensions     = "mkv,mk3d,mka,mks",
5. .priv\_data\_size = **sizeof**(MatroskaDemuxContext),
6. .read\_probe     = matroska\_probe,
7. .read\_header    = matroska\_read\_header,
8. .read\_packet    = matroska\_read\_packet,
9. .read\_close     = matroska\_read\_close,
10. .read\_seek      = matroska\_read\_seek,
11. .mime\_type      = "audio/webm,audio/x-matroska,video/webm,video/x-matroska"
12. };

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

### av\_match\_ext()

av\_match\_ext()是一个API函数，声明如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/41198929) [copy](http://blog.csdn.net/leixiaohua1020/article/details/41198929)

1. /\*\*
2. \* Return a positive value if the given filename has one of the given
3. \* extensions, 0 otherwise.
4. \*
5. \* @param filename   file name to check against the given extensions
6. \* @param extensions a comma-separated list of filename extensions
7. \*/
8. **int** av\_match\_ext(**const** **char** \*filename, **const** **char** \*extensions);

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

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/41198929) [copy](http://blog.csdn.net/leixiaohua1020/article/details/41198929)

[在CODE上查看代码片](https://code.csdn.net/snippets/611475)

1. **int** av\_match\_ext(**const** **char** \*filename, **const** **char** \*extensions)
2. {
3. **const** **char** \*ext;

6. **if** (!filename)
7. **return** 0;

10. ext = strrchr(filename, '.');
11. **if** (ext)
12. **return** av\_match\_name(ext + 1, extensions);
13. **return** 0;
14. }

经过以上几步之后，av\_guess\_format()最终可以得到最合适的AVOutputFormat并且返回给avformat\_alloc\_output\_context2()。avformat\_alloc\_output\_context2()接下来将获得的AVOutputFormat赋值给刚刚新建的AVFormatContext，即可完成初始化工作。

# [FFmpeg源代码简单分析：avformat\_write\_header()](http://blog.csdn.net/leixiaohua1020/article/details/44116215)

打算写两篇文章简单分析FFmpeg的写文件用到的3个函数：avformat\_write\_header()，av\_write\_frame()以及av\_write\_trailer()。其中av\_write\_frame()用于写视频数据，avformat\_write\_header()用于写视频文件头，而av\_write\_trailer()用于写视频文件尾。

本文首先分析avformat\_write\_header()。  
PS：  
需要注意的是，尽管这3个函数功能是配套的，但是它们的前缀却不一样，写文件头Header的函数前缀是“avformat\_”，其他两个函数前缀是“av\_”（不太明白其中的原因）。  
avformat\_write\_header()的声明位于libavformat\avformat.h，如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44116215) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44116215)

[在CODE上查看代码片](https://code.csdn.net/snippets/614513)

1. /\*\*
2. \* Allocate the stream private data and write the stream header to
3. \* an output media file.
4. \*
5. \* @param s Media file handle, must be allocated with avformat\_alloc\_context().
6. \*          Its oformat field must be set to the desired output format;
7. \*          Its pb field must be set to an already opened AVIOContext.
8. \* @param options  An AVDictionary filled with AVFormatContext and muxer-private options.
9. \*                 On return this parameter will be destroyed and replaced with a dict containing
10. \*                 options that were not found. May be NULL.
11. \*
12. \* @return 0 on success, negative AVERROR on failure.
13. \*
14. \* @see av\_opt\_find, av\_dict\_set, avio\_open, av\_oformat\_next.
15. \*/
16. **int** avformat\_write\_header(AVFormatContext \*s, AVDictionary \*\*options);

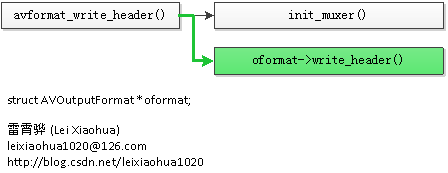
简单解释一下它的参数的含义：

s：用于输出的AVFormatContext。  
options：额外的选项，目前没有深入研究过，一般为NULL。

函数正常执行后返回值等于0。  
  
该函数最典型的例子可以参考：  
[最简单的基于FFMPEG的视频编码器（YUV编码为H.264）](http://blog.csdn.net/leixiaohua1020/article/details/25430425)

## 函数调用关系图

avformat\_write\_header()的调用关系如下图所示。



## avformat\_write\_header()

avformat\_write\_header()的定义位于libavformat\mux.c，如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44116215) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44116215)

[在CODE上查看代码片](https://code.csdn.net/snippets/614513)

1. **int** avformat\_write\_header(AVFormatContext \*s, AVDictionary \*\*options)
2. {
3. **int** ret = 0;
5. **if** (ret = init\_muxer(s, options))
6. **return** ret;
8. **if** (s->oformat->write\_header) {
9. ret = s->oformat->write\_header(s);
10. **if** (ret >= 0 && s->pb && s->pb->error < 0)
11. ret = s->pb->error;
12. **if** (ret < 0)
13. **return** ret;
14. **if** (s->flush\_packets && s->pb && s->pb->error >= 0 && s->flags & AVFMT\_FLAG\_FLUSH\_PACKETS)
15. avio\_flush(s->pb);
16. }
18. **if** ((ret = init\_pts(s)) < 0)
19. **return** ret;
21. **if** (s->avoid\_negative\_ts < 0) {
22. av\_assert2(s->avoid\_negative\_ts == AVFMT\_AVOID\_NEG\_TS\_AUTO);
23. **if** (s->oformat->flags & (AVFMT\_TS\_NEGATIVE | AVFMT\_NOTIMESTAMPS)) {
24. s->avoid\_negative\_ts = 0;
25. } **else**
26. s->avoid\_negative\_ts = AVFMT\_AVOID\_NEG\_TS\_MAKE\_NON\_NEGATIVE;
27. }
29. **return** 0;
30. }

从源代码可以看出，avformat\_write\_header()完成了以下工作：

（1）调用init\_muxer()初始化复用器  
（2）调用AVOutputFormat的write\_header()

下面看一下这两个函数。

## init\_muxer()

init\_muxer()用于初始化复用器，它的定义如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44116215) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44116215)

[在CODE上查看代码片](https://code.csdn.net/snippets/614513)

1. **static** **int** init\_muxer(AVFormatContext \*s, AVDictionary \*\*options)
2. {
3. **int** ret = 0, i;
4. AVStream \*st;
5. AVDictionary \*tmp = NULL;
6. AVCodecContext \*codec = NULL;
7. AVOutputFormat \*of = s->oformat;
8. AVDictionaryEntry \*e;
10. **if** (options)
11. av\_dict\_copy(&tmp, \*options, 0);
13. **if** ((ret = av\_opt\_set\_dict(s, &tmp)) < 0)
14. **goto** fail;
15. **if** (s->priv\_data && s->oformat->priv\_class && \*(**const** AVClass\*\*)s->priv\_data==s->oformat->priv\_class &&
16. (ret = av\_opt\_set\_dict2(s->priv\_data, &tmp, AV\_OPT\_SEARCH\_CHILDREN)) < 0)
17. **goto** fail;
19. #if FF\_API\_LAVF\_BITEXACT
20. **if** (s->nb\_streams && s->streams[0]->codec->flags & CODEC\_FLAG\_BITEXACT)
21. s->flags |= AVFMT\_FLAG\_BITEXACT;
22. #endif
24. // some sanity checks
25. **if** (s->nb\_streams == 0 && !(of->flags & AVFMT\_NOSTREAMS)) {
26. av\_log(s, AV\_LOG\_ERROR, "No streams to mux were specified\n");
27. ret = AVERROR(EINVAL);
28. **goto** fail;
29. }
31. **for** (i = 0; i < s->nb\_streams; i++) {
32. st    = s->streams[i];
33. codec = st->codec;
35. #if FF\_API\_LAVF\_CODEC\_TB
36. FF\_DISABLE\_DEPRECATION\_WARNINGS
37. **if** (!st->time\_base.num && codec->time\_base.num) {
38. av\_log(s, AV\_LOG\_WARNING, "Using AVStream.codec.time\_base as a "
39. "timebase hint to the muxer is deprecated. Set "
40. "AVStream.time\_base instead.\n");
41. avpriv\_set\_pts\_info(st, 64, codec->time\_base.num, codec->time\_base.den);
42. }
43. FF\_ENABLE\_DEPRECATION\_WARNINGS
44. #endif
46. **if** (!st->time\_base.num) {
47. /\* fall back on the default timebase values \*/
48. **if** (codec->codec\_type == AVMEDIA\_TYPE\_AUDIO && codec->sample\_rate)
49. avpriv\_set\_pts\_info(st, 64, 1, codec->sample\_rate);
50. **else**
51. avpriv\_set\_pts\_info(st, 33, 1, 90000);
52. }
54. **switch** (codec->codec\_type) {
55. **case** AVMEDIA\_TYPE\_AUDIO:
56. **if** (codec->sample\_rate <= 0) {
57. av\_log(s, AV\_LOG\_ERROR, "sample rate not set\n");
58. ret = AVERROR(EINVAL);
59. **goto** fail;
60. }
61. **if** (!codec->block\_align)
62. codec->block\_align = codec->channels \*
63. av\_get\_bits\_per\_sample(codec->codec\_id) >> 3;
64. **break**;
65. **case** AVMEDIA\_TYPE\_VIDEO:
66. **if** ((codec->width <= 0 || codec->height <= 0) &&
67. !(of->flags & AVFMT\_NODIMENSIONS)) {
68. av\_log(s, AV\_LOG\_ERROR, "dimensions not set\n");
69. ret = AVERROR(EINVAL);
70. **goto** fail;
71. }
72. **if** (av\_cmp\_q(st->sample\_aspect\_ratio, codec->sample\_aspect\_ratio)
73. && FFABS(av\_q2d(st->sample\_aspect\_ratio) - av\_q2d(codec->sample\_aspect\_ratio)) > 0.004\*av\_q2d(st->sample\_aspect\_ratio)
74. ) {
75. **if** (st->sample\_aspect\_ratio.num != 0 &&
76. st->sample\_aspect\_ratio.den != 0 &&
77. codec->sample\_aspect\_ratio.num != 0 &&
78. codec->sample\_aspect\_ratio.den != 0) {
79. av\_log(s, AV\_LOG\_ERROR, "Aspect ratio mismatch between muxer "
80. "(%d/%d) and encoder layer (%d/%d)\n",
81. st->sample\_aspect\_ratio.num, st->sample\_aspect\_ratio.den,
82. codec->sample\_aspect\_ratio.num,
83. codec->sample\_aspect\_ratio.den);
84. ret = AVERROR(EINVAL);
85. **goto** fail;
86. }
87. }
88. **break**;
89. }
91. **if** (of->codec\_tag) {
92. **if** (   codec->codec\_tag
93. && codec->codec\_id == AV\_CODEC\_ID\_RAWVIDEO
94. && (   av\_codec\_get\_tag(of->codec\_tag, codec->codec\_id) == 0
95. || av\_codec\_get\_tag(of->codec\_tag, codec->codec\_id) == MKTAG('r', 'a', 'w', ' '))
96. && !validate\_codec\_tag(s, st)) {
97. // the current rawvideo encoding system ends up setting
98. // the wrong codec\_tag for avi/mov, we override it here
99. codec->codec\_tag = 0;
100. }
101. **if** (codec->codec\_tag) {
102. **if** (!validate\_codec\_tag(s, st)) {
103. **char** tagbuf[32], tagbuf2[32];
104. av\_get\_codec\_tag\_string(tagbuf, **sizeof**(tagbuf), codec->codec\_tag);
105. av\_get\_codec\_tag\_string(tagbuf2, **sizeof**(tagbuf2), av\_codec\_get\_tag(s->oformat->codec\_tag, codec->codec\_id));
106. av\_log(s, AV\_LOG\_ERROR,
107. "Tag %s/0x%08x incompatible with output codec id '%d' (%s)\n",
108. tagbuf, codec->codec\_tag, codec->codec\_id, tagbuf2);
109. ret = AVERROR\_INVALIDDATA;
110. **goto** fail;
111. }
112. } **else**
113. codec->codec\_tag = av\_codec\_get\_tag(of->codec\_tag, codec->codec\_id);
114. }
116. **if** (of->flags & AVFMT\_GLOBALHEADER &&
117. !(codec->flags & CODEC\_FLAG\_GLOBAL\_HEADER))
118. av\_log(s, AV\_LOG\_WARNING,
119. "Codec for stream %d does not use global headers "
120. "but container format requires global headers\n", i);
122. **if** (codec->codec\_type != AVMEDIA\_TYPE\_ATTACHMENT)
123. s->internal->nb\_interleaved\_streams++;
124. }
126. **if** (!s->priv\_data && of->priv\_data\_size > 0) {
127. s->priv\_data = av\_mallocz(of->priv\_data\_size);
128. **if** (!s->priv\_data) {
129. ret = AVERROR(ENOMEM);
130. **goto** fail;
131. }
132. **if** (of->priv\_class) {
133. \*(**const** AVClass \*\*)s->priv\_data = of->priv\_class;
134. av\_opt\_set\_defaults(s->priv\_data);
135. **if** ((ret = av\_opt\_set\_dict2(s->priv\_data, &tmp, AV\_OPT\_SEARCH\_CHILDREN)) < 0)
136. **goto** fail;
137. }
138. }
140. /\* set muxer identification string \*/
141. **if** (!(s->flags & AVFMT\_FLAG\_BITEXACT)) {
142. av\_dict\_set(&s->metadata, "encoder", LIBAVFORMAT\_IDENT, 0);
143. } **else** {
144. av\_dict\_set(&s->metadata, "encoder", NULL, 0);
145. }
147. **for** (e = NULL; e = av\_dict\_get(s->metadata, "encoder-", e, AV\_DICT\_IGNORE\_SUFFIX); ) {
148. av\_dict\_set(&s->metadata, e->key, NULL, 0);
149. }
151. **if** (options) {
152. av\_dict\_free(options);
153. \*options = tmp;
154. }
156. **return** 0;
158. fail:
159. av\_dict\_free(&tmp);
160. **return** ret;
161. }

init\_muxer()代码很长，但是它所做的工作比较简单，可以概括成两个字：检查。函数的流程可以概括成以下几步：

（1）将传入的AVDictionary形式的选项设置到AVFormatContext  
（2）遍历AVFormatContext中的每个AVStream，并作如下检查：

a)AVStream的time\_base是否正确设置。如果发现AVStream的time\_base没有设置，则会调用avpriv\_set\_pts\_info()进行设置。

b)对于音频，检查采样率设置是否正确；对于视频，检查宽、高、宽高比。

c)其他一些检查，不再详述。

## AVOutputFormat->write\_header()

avformat\_write\_header()中最关键的地方就是调用了AVOutputFormat的write\_header()。write\_header()是AVOutputFormat中的一个函数指针，指向写文件头的函数。不同的AVOutputFormat有不同的write\_header()的实现方法。在这里我们举例子看一下FLV封装格式对应的AVOutputFormat，它的定义位于libavformat\flvenc.c，如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44116215) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44116215)

[在CODE上查看代码片](https://code.csdn.net/snippets/614513)

1. AVOutputFormat ff\_flv\_muxer = {
2. .name           = "flv",
3. .long\_name      = NULL\_IF\_CONFIG\_SMALL("FLV (Flash Video)"),
4. .mime\_type      = "video/x-flv",
5. .extensions     = "flv",
6. .priv\_data\_size = **sizeof**(FLVContext),
7. .audio\_codec    = CONFIG\_LIBMP3LAME ? AV\_CODEC\_ID\_MP3 : AV\_CODEC\_ID\_ADPCM\_SWF,
8. .video\_codec    = AV\_CODEC\_ID\_FLV1,
9. .write\_header   = flv\_write\_header,
10. .write\_packet   = flv\_write\_packet,
11. .write\_trailer  = flv\_write\_trailer,
12. .codec\_tag      = (**const** AVCodecTag\* **const** []) {
13. flv\_video\_codec\_ids, flv\_audio\_codec\_ids, 0
14. },
15. .flags          = AVFMT\_GLOBALHEADER | AVFMT\_VARIABLE\_FPS |
16. AVFMT\_TS\_NONSTRICT,
17. };

从ff\_flv\_muxer的定义中可以看出，write\_header()指向的函数为flv\_write\_header()。我们继续看一下flv\_write\_header()函数。flv\_write\_header()的定义同样位于libavformat\flvenc.c，如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44116215) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44116215)

[在CODE上查看代码片](https://code.csdn.net/snippets/614513)

1. **static** **int** flv\_write\_header(AVFormatContext \*s)
2. {
3. **int** i;
4. AVIOContext \*pb = s->pb;
5. FLVContext \*flv = s->priv\_data;
6. int64\_t data\_size;
7. //设置参数
8. **for** (i = 0; i < s->nb\_streams; i++) {
9. AVCodecContext \*enc = s->streams[i]->codec;
10. FLVStreamContext \*sc;
11. **switch** (enc->codec\_type) {
12. **case** AVMEDIA\_TYPE\_VIDEO:
13. **if** (s->streams[i]->avg\_frame\_rate.den &&
14. s->streams[i]->avg\_frame\_rate.num) {
15. //设置帧率，从AVStream拷贝过来
16. flv->framerate = av\_q2d(s->streams[i]->avg\_frame\_rate);
17. }
18. **if** (flv->video\_enc) {
19. av\_log(s, AV\_LOG\_ERROR,
20. "at most one video stream is supported in flv\n");
21. **return** AVERROR(EINVAL);
22. }
23. //视频编码的AVCodecContext
24. flv->video\_enc = enc;
25. **if** (enc->codec\_tag == 0) {
26. av\_log(s, AV\_LOG\_ERROR, "Video codec '%s' for stream %d is not compatible with FLV\n",
27. avcodec\_get\_name(enc->codec\_id), i);
28. **return** AVERROR(EINVAL);
29. }
30. **if** (enc->codec\_id == AV\_CODEC\_ID\_MPEG4 ||
31. enc->codec\_id == AV\_CODEC\_ID\_H263) {
32. **int** error = s->strict\_std\_compliance > FF\_COMPLIANCE\_UNOFFICIAL;
33. av\_log(s, error ? AV\_LOG\_ERROR : AV\_LOG\_WARNING,
34. "Codec %s is not supported in the official FLV specification,\n", avcodec\_get\_name(enc->codec\_id));
36. **if** (error) {
37. av\_log(s, AV\_LOG\_ERROR,
38. "use vstrict=-1 / -strict -1 to use it anyway.\n");
39. **return** AVERROR(EINVAL);
40. }
41. } **else** **if** (enc->codec\_id == AV\_CODEC\_ID\_VP6) {
42. av\_log(s, AV\_LOG\_WARNING,
43. "Muxing VP6 in flv will produce flipped video on playback.\n");
44. }
45. **break**;
46. **case** AVMEDIA\_TYPE\_AUDIO:
47. **if** (flv->audio\_enc) {
48. av\_log(s, AV\_LOG\_ERROR,
49. "at most one audio stream is supported in flv\n");
50. **return** AVERROR(EINVAL);
51. }
52. //音频编码的AVCodecContext
53. flv->audio\_enc = enc;
54. **if** (get\_audio\_flags(s, enc) < 0)
55. **return** AVERROR\_INVALIDDATA;
56. **if** (enc->codec\_id == AV\_CODEC\_ID\_PCM\_S16BE)
57. av\_log(s, AV\_LOG\_WARNING,
58. "16-bit big-endian audio in flv is valid but most likely unplayable (hardware dependent); use s16le\n");
59. **break**;
60. **case** AVMEDIA\_TYPE\_DATA:
61. **if** (enc->codec\_id != AV\_CODEC\_ID\_TEXT && enc->codec\_id != AV\_CODEC\_ID\_NONE) {
62. av\_log(s, AV\_LOG\_ERROR, "Data codec '%s' for stream %d is not compatible with FLV\n",
63. avcodec\_get\_name(enc->codec\_id), i);
64. **return** AVERROR\_INVALIDDATA;
65. }
66. flv->data\_enc = enc;
67. **break**;
68. **default**:
69. av\_log(s, AV\_LOG\_ERROR, "Codec type '%s' for stream %d is not compatible with FLV\n",
70. av\_get\_media\_type\_string(enc->codec\_type), i);
71. **return** AVERROR(EINVAL);
72. }
73. avpriv\_set\_pts\_info(s->streams[i], 32, 1, 1000); /\* 32 bit pts in ms \*/
75. sc = av\_mallocz(**sizeof**(FLVStreamContext));
76. **if** (!sc)
77. **return** AVERROR(ENOMEM);
78. s->streams[i]->priv\_data = sc;
79. sc->last\_ts = -1;
80. }
82. flv->delay = AV\_NOPTS\_VALUE;
83. //开始写入
84. //Signature
85. avio\_write(pb, "FLV", 3);
86. //Version
87. avio\_w8(pb, 1);
88. //“！！”意思是把非0转换成1
89. //Flags
90. avio\_w8(pb, FLV\_HEADER\_FLAG\_HASAUDIO \* !!flv->audio\_enc +
91. FLV\_HEADER\_FLAG\_HASVIDEO \* !!flv->video\_enc);
92. //Header size
93. avio\_wb32(pb, 9);
94. //Header结束
95. //Previous Tag Size
96. avio\_wb32(pb, 0);
98. **for** (i = 0; i < s->nb\_streams; i++)
99. **if** (s->streams[i]->codec->codec\_tag == 5) {
100. avio\_w8(pb, 8);     // message type
101. avio\_wb24(pb, 0);   // include flags
102. avio\_wb24(pb, 0);   // time stamp
103. avio\_wb32(pb, 0);   // reserved
104. avio\_wb32(pb, 11);  // size
105. flv->reserved = 5;
106. }
108. write\_metadata(s, 0);
110. **for** (i = 0; i < s->nb\_streams; i++) {
111. AVCodecContext \*enc = s->streams[i]->codec;
112. **if** (enc->codec\_id == AV\_CODEC\_ID\_AAC || enc->codec\_id == AV\_CODEC\_ID\_H264 || enc->codec\_id == AV\_CODEC\_ID\_MPEG4) {
113. int64\_t pos;
114. avio\_w8(pb, enc->codec\_type == AVMEDIA\_TYPE\_VIDEO ?
115. FLV\_TAG\_TYPE\_VIDEO : FLV\_TAG\_TYPE\_AUDIO);
116. avio\_wb24(pb, 0); // size patched later
117. avio\_wb24(pb, 0); // ts
118. avio\_w8(pb, 0);   // ts ext
119. avio\_wb24(pb, 0); // streamid
120. pos = avio\_tell(pb);
121. **if** (enc->codec\_id == AV\_CODEC\_ID\_AAC) {
122. avio\_w8(pb, get\_audio\_flags(s, enc));
123. avio\_w8(pb, 0); // AAC sequence header
124. avio\_write(pb, enc->extradata, enc->extradata\_size);
125. } **else** {
126. avio\_w8(pb, enc->codec\_tag | FLV\_FRAME\_KEY); // flags
127. avio\_w8(pb, 0); // AVC sequence header
128. avio\_wb24(pb, 0); // composition time
129. ff\_isom\_write\_avcc(pb, enc->extradata, enc->extradata\_size);
130. }
131. data\_size = avio\_tell(pb) - pos;
132. avio\_seek(pb, -data\_size - 10, SEEK\_CUR);
133. avio\_wb24(pb, data\_size);
134. avio\_skip(pb, data\_size + 10 - 3);
135. avio\_wb32(pb, data\_size + 11); // previous tag size
136. }
137. }
139. **return** 0;
140. }

从源代码可以看出，flv\_write\_header()完成了FLV文件头的写入工作。该函数的工作可以大体分为以下两部分：

（1）给FLVContext设置参数  
（2）写文件头，以及相关的Tag

写文件头的代码很短，如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44116215) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44116215)

[在CODE上查看代码片](https://code.csdn.net/snippets/614513)

1. avio\_write(pb, "FLV", 3);
2. avio\_w8(pb, 1);
3. avio\_w8(pb, FLV\_HEADER\_FLAG\_HASAUDIO \* !!flv->audio\_enc +
4. FLV\_HEADER\_FLAG\_HASVIDEO \* !!flv->video\_enc);
5. avio\_wb32(pb, 9);

可以参考下图中FLV文件头的定义比对一下上面的代码。



# [FFmpeg源代码简单分析：avcodec\_encode\_video()](http://blog.csdn.net/leixiaohua1020/article/details/44206485)

本文简单分析FFmpeg的avcodec\_encode\_video2()函数。该函数用于编码一帧视频数据。avcodec\_encode\_video2()函数的声明位于libavcodec\avcodec.h，如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44206485) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44206485)

[在CODE上查看代码片](https://code.csdn.net/snippets/617364)

1. /\*\*
2. \* Encode a frame of video.
3. \*
4. \* Takes input raw video data from frame and writes the next output packet, if
5. \* available, to avpkt. The output packet does not necessarily contain data for
6. \* the most recent frame, as encoders can delay and reorder input frames
7. \* internally as needed.
8. \*
9. \* @param avctx     codec context
10. \* @param avpkt     output AVPacket.
11. \*                  The user can supply an output buffer by setting
12. \*                  avpkt->data and avpkt->size prior to calling the
13. \*                  function, but if the size of the user-provided data is not
14. \*                  large enough, encoding will fail. All other AVPacket fields
15. \*                  will be reset by the encoder using av\_init\_packet(). If
16. \*                  avpkt->data is NULL, the encoder will allocate it.
17. \*                  The encoder will set avpkt->size to the size of the
18. \*                  output packet. The returned data (if any) belongs to the
19. \*                  caller, he is responsible for freeing it.
20. \*
21. \*                  If this function fails or produces no output, avpkt will be
22. \*                  freed using av\_free\_packet() (i.e. avpkt->destruct will be
23. \*                  called to free the user supplied buffer).
24. \* @param[in] frame AVFrame containing the raw video data to be encoded.
25. \*                  May be NULL when flushing an encoder that has the
26. \*                  CODEC\_CAP\_DELAY capability set.
27. \* @param[out] got\_packet\_ptr This field is set to 1 by libavcodec if the
28. \*                            output packet is non-empty, and to 0 if it is
29. \*                            empty. If the function returns an error, the
30. \*                            packet can be assumed to be invalid, and the
31. \*                            value of got\_packet\_ptr is undefined and should
32. \*                            not be used.
33. \* @return          0 on success, negative error code on failure
34. \*/
35. **int** avcodec\_encode\_video2(AVCodecContext \*avctx, AVPacket \*avpkt,
36. **const** AVFrame \*frame, **int** \*got\_packet\_ptr);

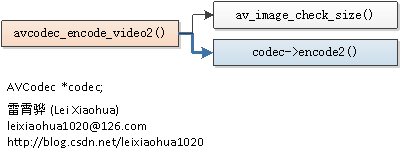
该函数每个参数的含义在注释里面已经写的很清楚了，在这里用中文简述一下：

avctx：编码器的AVCodecContext。  
avpkt：编码输出的AVPacket。  
frame：编码输入的AVFrame。  
got\_packet\_ptr：成功编码一个AVPacket的时候设置为1。

函数返回0代表编码成功。

## 函数调用关系图

函数的调用关系如下图所示。



## avcodec\_encode\_video2()

avcodec\_encode\_video2()的定义位于libavcodec\utils.c，如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44206485) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44206485)

[在CODE上查看代码片](https://code.csdn.net/snippets/617364)

1. **int** attribute\_align\_arg avcodec\_encode\_video2(AVCodecContext \*avctx,
2. AVPacket \*avpkt,
3. **const** AVFrame \*frame,
4. **int** \*got\_packet\_ptr)
5. {
6. **int** ret;
7. AVPacket user\_pkt = \*avpkt;
8. **int** needs\_realloc = !user\_pkt.data;
10. \*got\_packet\_ptr = 0;
12. **if**(CONFIG\_FRAME\_THREAD\_ENCODER &&
13. avctx->internal->frame\_thread\_encoder && (avctx->active\_thread\_type&FF\_THREAD\_FRAME))
14. **return** ff\_thread\_video\_encode\_frame(avctx, avpkt, frame, got\_packet\_ptr);
16. **if** ((avctx->flags&CODEC\_FLAG\_PASS1) && avctx->stats\_out)
17. avctx->stats\_out[0] = '\0';
19. **if** (!(avctx->codec->capabilities & CODEC\_CAP\_DELAY) && !frame) {
20. av\_free\_packet(avpkt);
21. av\_init\_packet(avpkt);
22. avpkt->size = 0;
23. **return** 0;
24. }
25. //检查输入
26. **if** (av\_image\_check\_size(avctx->width, avctx->height, 0, avctx))
27. **return** AVERROR(EINVAL);
29. av\_assert0(avctx->codec->encode2);
30. //编码
31. ret = avctx->codec->encode2(avctx, avpkt, frame, got\_packet\_ptr);
32. av\_assert0(ret <= 0);
34. **if** (avpkt->data && avpkt->data == avctx->internal->byte\_buffer) {
35. needs\_realloc = 0;
36. **if** (user\_pkt.data) {
37. **if** (user\_pkt.size >= avpkt->size) {
38. memcpy(user\_pkt.data, avpkt->data, avpkt->size);
39. } **else** {
40. av\_log(avctx, AV\_LOG\_ERROR, "Provided packet is too small, needs to be %d\n", avpkt->size);
41. avpkt->size = user\_pkt.size;
42. ret = -1;
43. }
44. avpkt->buf      = user\_pkt.buf;
45. avpkt->data     = user\_pkt.data;
46. #if FF\_API\_DESTRUCT\_PACKET
47. FF\_DISABLE\_DEPRECATION\_WARNINGS
48. avpkt->destruct = user\_pkt.destruct;
49. FF\_ENABLE\_DEPRECATION\_WARNINGS
50. #endif
51. } **else** {
52. **if** (av\_dup\_packet(avpkt) < 0) {
53. ret = AVERROR(ENOMEM);
54. }
55. }
56. }
58. **if** (!ret) {
59. **if** (!\*got\_packet\_ptr)
60. avpkt->size = 0;
61. **else** **if** (!(avctx->codec->capabilities & CODEC\_CAP\_DELAY))
62. avpkt->pts = avpkt->dts = frame->pts;
64. **if** (needs\_realloc && avpkt->data) {
65. ret = av\_buffer\_realloc(&avpkt->buf, avpkt->size + FF\_INPUT\_BUFFER\_PADDING\_SIZE);
66. **if** (ret >= 0)
67. avpkt->data = avpkt->buf->data;
68. }
70. avctx->frame\_number++;
71. }
73. **if** (ret < 0 || !\*got\_packet\_ptr)
74. av\_free\_packet(avpkt);
75. **else**
76. av\_packet\_merge\_side\_data(avpkt);
78. emms\_c();
79. **return** ret;
80. }

从函数的定义可以看出，avcodec\_encode\_video2()首先调用了av\_image\_check\_size()检查设置的宽高参数是否合理，然后调用了AVCodec的encode2()调用具体的解码器。

## av\_image\_check\_size()

av\_image\_check\_size()是一个很简单的函数，用于检查图像宽高是否正常，它的定义如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44206485) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44206485)

[在CODE上查看代码片](https://code.csdn.net/snippets/617364)

1. **int** av\_image\_check\_size(unsigned **int** w, unsigned **int** h, **int** log\_offset, **void** \*log\_ctx)
2. {
3. ImgUtils imgutils = { &imgutils\_class, log\_offset, log\_ctx };
5. **if** ((**int**)w>0 && (**int**)h>0 && (w+128)\*(uint64\_t)(h+128) < INT\_MAX/8)
6. **return** 0;
8. av\_log(&imgutils, AV\_LOG\_ERROR, "Picture size %ux%u is invalid\n", w, h);
9. **return** AVERROR(EINVAL);
10. }

从代码中可以看出，av\_image\_check\_size()主要是要求图像宽高必须为正数，而且取值不能太大。

## AVCodec->encode2()

AVCodec的encode2()是一个函数指针，指向特定编码器的编码函数。在这里我们以libx264为例，看一下它对应的AVCodec的结构体的定义，如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44206485) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44206485)

[在CODE上查看代码片](https://code.csdn.net/snippets/617364)

1. AVCodec ff\_libx264\_encoder = {
2. .name             = "libx264",
3. .long\_name        = NULL\_IF\_CONFIG\_SMALL("libx264 H.264 / AVC / MPEG-4 AVC / MPEG-4 part 10"),
4. .type             = AVMEDIA\_TYPE\_VIDEO,
5. .id               = AV\_CODEC\_ID\_H264,
6. .priv\_data\_size   = **sizeof**(X264Context),
7. .init             = X264\_init,
8. .encode2          = X264\_frame,
9. .close            = X264\_close,
10. .capabilities     = CODEC\_CAP\_DELAY | CODEC\_CAP\_AUTO\_THREADS,
11. .priv\_class       = &x264\_class,
12. .defaults         = x264\_defaults,
13. .init\_static\_data = X264\_init\_static,
14. };

从ff\_libx264\_encoder的定义可以看出，encode2()函数指向的是X264\_frame()函数。

## X264\_frame()

X264\_frame()函数的定义位于libavcodec\libx264.c，如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44206485) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44206485)

[在CODE上查看代码片](https://code.csdn.net/snippets/617364)

1. **static** **int** X264\_frame(AVCodecContext \*ctx, AVPacket \*pkt, **const** AVFrame \*frame,
2. **int** \*got\_packet)
3. {
4. X264Context \*x4 = ctx->priv\_data;
5. x264\_nal\_t \*nal;
6. **int** nnal, i, ret;
7. x264\_picture\_t pic\_out = {0};
8. AVFrameSideData \*side\_data;
10. x264\_picture\_init( &x4->pic );
11. x4->pic.img.i\_csp   = x4->params.i\_csp;
12. **if** (x264\_bit\_depth > 8)
13. x4->pic.img.i\_csp |= X264\_CSP\_HIGH\_DEPTH;
14. x4->pic.img.i\_plane = avfmt2\_num\_planes(ctx->pix\_fmt);
16. **if** (frame) {
17. **for** (i = 0; i < x4->pic.img.i\_plane; i++) {
18. x4->pic.img.plane[i]    = frame->data[i];
19. x4->pic.img.i\_stride[i] = frame->linesize[i];
20. }
22. x4->pic.i\_pts  = frame->pts;
23. x4->pic.i\_type =
24. frame->pict\_type == AV\_PICTURE\_TYPE\_I ? X264\_TYPE\_KEYFRAME :
25. frame->pict\_type == AV\_PICTURE\_TYPE\_P ? X264\_TYPE\_P :
26. frame->pict\_type == AV\_PICTURE\_TYPE\_B ? X264\_TYPE\_B :
27. X264\_TYPE\_AUTO;
29. **if** (x4->avcintra\_class < 0) {
30. **if** (x4->params.b\_interlaced && x4->params.b\_tff != frame->top\_field\_first) {
31. x4->params.b\_tff = frame->top\_field\_first;
32. x264\_encoder\_reconfig(x4->enc, &x4->params);
33. }
34. **if** (x4->params.vui.i\_sar\_height != ctx->sample\_aspect\_ratio.den ||
35. x4->params.vui.i\_sar\_width  != ctx->sample\_aspect\_ratio.num) {
36. x4->params.vui.i\_sar\_height = ctx->sample\_aspect\_ratio.den;
37. x4->params.vui.i\_sar\_width  = ctx->sample\_aspect\_ratio.num;
38. x264\_encoder\_reconfig(x4->enc, &x4->params);
39. }
41. **if** (x4->params.rc.i\_vbv\_buffer\_size != ctx->rc\_buffer\_size / 1000 ||
42. x4->params.rc.i\_vbv\_max\_bitrate != ctx->rc\_max\_rate    / 1000) {
43. x4->params.rc.i\_vbv\_buffer\_size = ctx->rc\_buffer\_size / 1000;
44. x4->params.rc.i\_vbv\_max\_bitrate = ctx->rc\_max\_rate    / 1000;
45. x264\_encoder\_reconfig(x4->enc, &x4->params);
46. }
48. **if** (x4->params.rc.i\_rc\_method == X264\_RC\_ABR &&
49. x4->params.rc.i\_bitrate != ctx->bit\_rate / 1000) {
50. x4->params.rc.i\_bitrate = ctx->bit\_rate / 1000;
51. x264\_encoder\_reconfig(x4->enc, &x4->params);
52. }
54. **if** (x4->crf >= 0 &&
55. x4->params.rc.i\_rc\_method == X264\_RC\_CRF &&
56. x4->params.rc.f\_rf\_constant != x4->crf) {
57. x4->params.rc.f\_rf\_constant = x4->crf;
58. x264\_encoder\_reconfig(x4->enc, &x4->params);
59. }
61. **if** (x4->params.rc.i\_rc\_method == X264\_RC\_CQP &&
62. x4->cqp >= 0 &&
63. x4->params.rc.i\_qp\_constant != x4->cqp) {
64. x4->params.rc.i\_qp\_constant = x4->cqp;
65. x264\_encoder\_reconfig(x4->enc, &x4->params);
66. }
68. **if** (x4->crf\_max >= 0 &&
69. x4->params.rc.f\_rf\_constant\_max != x4->crf\_max) {
70. x4->params.rc.f\_rf\_constant\_max = x4->crf\_max;
71. x264\_encoder\_reconfig(x4->enc, &x4->params);
72. }
73. }
75. side\_data = av\_frame\_get\_side\_data(frame, AV\_FRAME\_DATA\_STEREO3D);
76. **if** (side\_data) {
77. AVStereo3D \*stereo = (AVStereo3D \*)side\_data->data;
78. **int** fpa\_type;
80. **switch** (stereo->type) {
81. **case** AV\_STEREO3D\_CHECKERBOARD:
82. fpa\_type = 0;
83. **break**;
84. **case** AV\_STEREO3D\_COLUMNS:
85. fpa\_type = 1;
86. **break**;
87. **case** AV\_STEREO3D\_LINES:
88. fpa\_type = 2;
89. **break**;
90. **case** AV\_STEREO3D\_SIDEBYSIDE:
91. fpa\_type = 3;
92. **break**;
93. **case** AV\_STEREO3D\_TOPBOTTOM:
94. fpa\_type = 4;
95. **break**;
96. **case** AV\_STEREO3D\_FRAMESEQUENCE:
97. fpa\_type = 5;
98. **break**;
99. **default**:
100. fpa\_type = -1;
101. **break**;
102. }
104. **if** (fpa\_type != x4->params.i\_frame\_packing) {
105. x4->params.i\_frame\_packing = fpa\_type;
106. x264\_encoder\_reconfig(x4->enc, &x4->params);
107. }
108. }
109. }
110. **do** {
111. **if** (x264\_encoder\_encode(x4->enc, &nal, &nnal, frame? &x4->pic: NULL, &pic\_out) < 0)
112. **return** -1;
114. ret = encode\_nals(ctx, pkt, nal, nnal);
115. **if** (ret < 0)
116. **return** -1;
117. } **while** (!ret && !frame && x264\_encoder\_delayed\_frames(x4->enc));
119. pkt->pts = pic\_out.i\_pts;
120. pkt->dts = pic\_out.i\_dts;
122. **switch** (pic\_out.i\_type) {
123. **case** X264\_TYPE\_IDR:
124. **case** X264\_TYPE\_I:
125. ctx->coded\_frame->pict\_type = AV\_PICTURE\_TYPE\_I;
126. **break**;
127. **case** X264\_TYPE\_P:
128. ctx->coded\_frame->pict\_type = AV\_PICTURE\_TYPE\_P;
129. **break**;
130. **case** X264\_TYPE\_B:
131. **case** X264\_TYPE\_BREF:
132. ctx->coded\_frame->pict\_type = AV\_PICTURE\_TYPE\_B;
133. **break**;
134. }
136. pkt->flags |= AV\_PKT\_FLAG\_KEY\*pic\_out.b\_keyframe;
137. **if** (ret)
138. ctx->coded\_frame->quality = (pic\_out.i\_qpplus1 - 1) \* FF\_QP2LAMBDA;
140. \*got\_packet = ret;
141. **return** 0;
142. }

有关X264编码的代码在以后分析X264的时候再进行详细分析。在这里我们可以我们可以简单看出该函数中有一个do while循环，其中调用了x264\_encoder\_encode()完成了编码的工作。

# [FFmpeg源代码简单分析：av\_write\_frame()](http://blog.csdn.net/leixiaohua1020/article/details/44199673)

打算写两篇文章简单分析FFmpeg的写文件用到的3个函数avformat\_write\_header()，av\_write\_frame()以及av\_write\_trailer()。上篇文章已经分析了avformat\_write\_header()，这篇文章继续分析av\_write\_frame()。

av\_write\_frame()用于输出一帧视音频数据，它的声明位于libavformat\avformat.h，如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44199673) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44199673)

[在CODE上查看代码片](https://code.csdn.net/snippets/616946)

1. /\*\*
2. \* Write a packet to an output media file.
3. \*
4. \* This function passes the packet directly to the muxer, without any buffering
5. \* or reordering. The caller is responsible for correctly interleaving the
6. \* packets if the format requires it. Callers that want libavformat to handle
7. \* the interleaving should call av\_interleaved\_write\_frame() instead of this
8. \* function.
9. \*
10. \* @param s media file handle
11. \* @param pkt The packet containing the data to be written. Note that unlike
12. \*            av\_interleaved\_write\_frame(), this function does not take
13. \*            ownership of the packet passed to it (though some muxers may make
14. \*            an internal reference to the input packet).
15. \*            <br>
16. \*            This parameter can be NULL (at any time, not just at the end), in
17. \*            order to immediately flush data buffered within the muxer, for
18. \*            muxers that buffer up data internally before writing it to the
19. \*            output.
20. \*            <br>
21. \*            Packet's @ref AVPacket.stream\_index "stream\_index" field must be
22. \*            set to the index of the corresponding stream in @ref
23. \*            AVFormatContext.streams "s->streams". It is very strongly
24. \*            recommended that timing information (@ref AVPacket.pts "pts", @ref
25. \*            AVPacket.dts "dts", @ref AVPacket.duration "duration") is set to
26. \*            correct values.
27. \* @return < 0 on error, = 0 if OK, 1 if flushed and there is no more data to flush
28. \*
29. \* @see av\_interleaved\_write\_frame()
30. \*/
31. **int** av\_write\_frame(AVFormatContext \*s, AVPacket \*pkt);

简单解释一下它的参数的含义：

s：用于输出的AVFormatContext。  
pkt：等待输出的AVPacket。

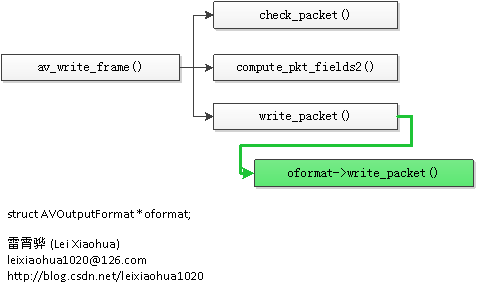
函数正常执行后返回值等于0。

这个函数最典型的例子可以参考：

[最简单的基于FFMPEG的视频编码器（YUV编码为H.264）](http://blog.csdn.net/leixiaohua1020/article/details/25430425)

## 函数调用关系图

av\_write\_frame()的调用关系如下图所示。



## av\_write\_frame()

av\_write\_frame()的定义位于libavformat\mux.c，如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44199673) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44199673)

[在CODE上查看代码片](https://code.csdn.net/snippets/616946)

1. **int** av\_write\_frame(AVFormatContext \*s, AVPacket \*pkt)
2. {
3. **int** ret;
5. ret = check\_packet(s, pkt);
6. **if** (ret < 0)
7. **return** ret;
8. //Packet为NULL，Flush Encoder
9. **if** (!pkt) {
10. **if** (s->oformat->flags & AVFMT\_ALLOW\_FLUSH) {
11. ret = s->oformat->write\_packet(s, NULL);
12. **if** (s->flush\_packets && s->pb && s->pb->error >= 0 && s->flags & AVFMT\_FLAG\_FLUSH\_PACKETS)
13. avio\_flush(s->pb);
14. **if** (ret >= 0 && s->pb && s->pb->error < 0)
15. ret = s->pb->error;
16. **return** ret;
17. }
18. **return** 1;
19. }
21. ret = compute\_pkt\_fields2(s, s->streams[pkt->stream\_index], pkt);
23. **if** (ret < 0 && !(s->oformat->flags & AVFMT\_NOTIMESTAMPS))
24. **return** ret;
25. //写入
26. ret = write\_packet(s, pkt);
27. **if** (ret >= 0 && s->pb && s->pb->error < 0)
28. ret = s->pb->error;
30. **if** (ret >= 0)
31. s->streams[pkt->stream\_index]->nb\_frames++;
32. **return** ret;
33. }

从源代码可以看出，av\_write\_frame()主要完成了以下几步工作：  
（1）调用check\_packet()做一些简单的检测  
（2）调用compute\_pkt\_fields2()设置AVPacket的一些属性值  
（3）调用write\_packet()写入数据

下面分别看一下这几个函数功能。

### check\_packet()

check\_packet()定义位于libavformat\mux.c，如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44199673) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44199673)

[在CODE上查看代码片](https://code.csdn.net/snippets/616946)

1. **static** **int** check\_packet(AVFormatContext \*s, AVPacket \*pkt)
2. {
3. **if** (!pkt)
4. **return** 0;
6. **if** (pkt->stream\_index < 0 || pkt->stream\_index >= s->nb\_streams) {
7. av\_log(s, AV\_LOG\_ERROR, "Invalid packet stream index: %d\n",
8. pkt->stream\_index);
9. **return** AVERROR(EINVAL);
10. }
12. **if** (s->streams[pkt->stream\_index]->codec->codec\_type == AVMEDIA\_TYPE\_ATTACHMENT) {
13. av\_log(s, AV\_LOG\_ERROR, "Received a packet for an attachment stream.\n");
14. **return** AVERROR(EINVAL);
15. }
17. **return** 0;
18. }

从代码中可以看出，check\_packet()的功能比较简单：首先检查一下输入的AVPacket是否为空，如果为空，则是直接返回；然后检查一下AVPacket的stream\_index（标记了该AVPacket所属的AVStream）设置是否正常，如果为负数或者大于AVStream的个数，则返回错误信息；最后检查AVPacket所属的AVStream是否属于attachment stream，这个地方没见过，目前还没有研究。

### compute\_pkt\_fields2()

compute\_pkt\_fields2()函数的定义位于libavformat\mux.c，如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44199673) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44199673)

[在CODE上查看代码片](https://code.csdn.net/snippets/616946)

1. //FIXME merge with compute\_pkt\_fields
2. **static** **int** compute\_pkt\_fields2(AVFormatContext \*s, AVStream \*st, AVPacket \*pkt)
3. {
4. **int** delay = FFMAX(st->codec->has\_b\_frames, st->codec->max\_b\_frames > 0);
5. **int** num, den, i;
6. **int** frame\_size;
8. av\_dlog(s, "compute\_pkt\_fields2: pts:%s dts:%s cur\_dts:%s b:%d size:%d st:%d\n",
9. av\_ts2str(pkt->pts), av\_ts2str(pkt->dts), av\_ts2str(st->cur\_dts), delay, pkt->size, pkt->stream\_index);
11. **if** (pkt->duration < 0 && st->codec->codec\_type != AVMEDIA\_TYPE\_SUBTITLE) {
12. av\_log(s, AV\_LOG\_WARNING, "Packet with invalid duration %d in stream %d\n",
13. pkt->duration, pkt->stream\_index);
14. pkt->duration = 0;
15. }
17. /\* duration field \*/
18. **if** (pkt->duration == 0) {
19. ff\_compute\_frame\_duration(s, &num, &den, st, NULL, pkt);
20. **if** (den && num) {
21. pkt->duration = av\_rescale(1, num \* (int64\_t)st->time\_base.den \* st->codec->ticks\_per\_frame, den \* (int64\_t)st->time\_base.num);
22. }
23. }
25. **if** (pkt->pts == AV\_NOPTS\_VALUE && pkt->dts != AV\_NOPTS\_VALUE && delay == 0)
26. pkt->pts = pkt->dts;
28. //XXX/FIXME this is a temporary hack until all encoders output pts
29. **if** ((pkt->pts == 0 || pkt->pts == AV\_NOPTS\_VALUE) && pkt->dts == AV\_NOPTS\_VALUE && !delay) {
30. **static** **int** warned;
31. **if** (!warned) {
32. av\_log(s, AV\_LOG\_WARNING, "Encoder did not produce proper pts, making some up.\n");
33. warned = 1;
34. }
35. pkt->dts =
36. //        pkt->pts= st->cur\_dts;
37. pkt->pts = st->pts.val;
38. }
40. //calculate dts from pts
41. **if** (pkt->pts != AV\_NOPTS\_VALUE && pkt->dts == AV\_NOPTS\_VALUE && delay <= MAX\_REORDER\_DELAY) {
42. st->pts\_buffer[0] = pkt->pts;
43. **for** (i = 1; i < delay + 1 && st->pts\_buffer[i] == AV\_NOPTS\_VALUE; i++)
44. st->pts\_buffer[i] = pkt->pts + (i - delay - 1) \* pkt->duration;
45. **for** (i = 0; i<delay && st->pts\_buffer[i] > st->pts\_buffer[i + 1]; i++)
46. FFSWAP(int64\_t, st->pts\_buffer[i], st->pts\_buffer[i + 1]);
48. pkt->dts = st->pts\_buffer[0];
49. }
51. **if** (st->cur\_dts && st->cur\_dts != AV\_NOPTS\_VALUE &&
52. ((!(s->oformat->flags & AVFMT\_TS\_NONSTRICT) &&
53. st->cur\_dts >= pkt->dts) || st->cur\_dts > pkt->dts)) {
54. av\_log(s, AV\_LOG\_ERROR,
55. "Application provided invalid, non monotonically increasing dts to muxer in stream %d: %s >= %s\n",
56. st->index, av\_ts2str(st->cur\_dts), av\_ts2str(pkt->dts));
57. **return** AVERROR(EINVAL);
58. }
59. **if** (pkt->dts != AV\_NOPTS\_VALUE && pkt->pts != AV\_NOPTS\_VALUE && pkt->pts < pkt->dts) {
60. av\_log(s, AV\_LOG\_ERROR,
61. "pts (%s) < dts (%s) in stream %d\n",
62. av\_ts2str(pkt->pts), av\_ts2str(pkt->dts),
63. st->index);
64. **return** AVERROR(EINVAL);
65. }
67. av\_dlog(s, "av\_write\_frame: pts2:%s dts2:%s\n",
68. av\_ts2str(pkt->pts), av\_ts2str(pkt->dts));
69. st->cur\_dts = pkt->dts;
70. st->pts.val = pkt->dts;
72. /\* update pts \*/
73. **switch** (st->codec->codec\_type) {
74. **case** AVMEDIA\_TYPE\_AUDIO:
75. frame\_size = (pkt->flags & AV\_PKT\_FLAG\_UNCODED\_FRAME) ?
76. ((AVFrame \*)pkt->data)->nb\_samples :
77. av\_get\_audio\_frame\_duration(st->codec, pkt->size);
79. /\* HACK/FIXME, we skip the initial 0 size packets as they are most
80. \* likely equal to the encoder delay, but it would be better if we
81. \* had the real timestamps from the encoder \*/
82. **if** (frame\_size >= 0 && (pkt->size || st->pts.num != st->pts.den >> 1 || st->pts.val)) {
83. frac\_add(&st->pts, (int64\_t)st->time\_base.den \* frame\_size);
84. }
85. **break**;
86. **case** AVMEDIA\_TYPE\_VIDEO:
87. frac\_add(&st->pts, (int64\_t)st->time\_base.den \* st->codec->time\_base.num);
88. **break**;
89. }
90. **return** 0;
91. }

从代码中可以看出，compute\_pkt\_fields2()主要有两方面的功能：一方面用于计算AVPacket的duration， dts等信息；另一方面用于检查pts、dts这些参数的合理性（例如PTS是否一定大于DTS）。具体的代码还没有细看，以后有时间再进行分析。

### AVOutputFormat->write\_packet()

write\_packet()函数的定义位于libavformat\mux.c，如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44199673) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44199673)

[在CODE上查看代码片](https://code.csdn.net/snippets/616946)

1. /\*\*
2. \* Make timestamps non negative, move side data from payload to internal struct, call muxer, and restore
3. \* sidedata.
4. \*
5. \* FIXME: this function should NEVER get undefined pts/dts beside when the
6. \* AVFMT\_NOTIMESTAMPS is set.
7. \* Those additional safety checks should be dropped once the correct checks
8. \* are set in the callers.
9. \*/
10. **static** **int** write\_packet(AVFormatContext \*s, AVPacket \*pkt)
11. {
12. **int** ret, did\_split;
14. **if** (s->output\_ts\_offset) {
15. AVStream \*st = s->streams[pkt->stream\_index];
16. int64\_t offset = av\_rescale\_q(s->output\_ts\_offset, AV\_TIME\_BASE\_Q, st->time\_base);
18. **if** (pkt->dts != AV\_NOPTS\_VALUE)
19. pkt->dts += offset;
20. **if** (pkt->pts != AV\_NOPTS\_VALUE)
21. pkt->pts += offset;
22. }
24. **if** (s->avoid\_negative\_ts > 0) {
25. AVStream \*st = s->streams[pkt->stream\_index];
26. int64\_t offset = st->mux\_ts\_offset;
28. **if** (s->offset == AV\_NOPTS\_VALUE && pkt->dts != AV\_NOPTS\_VALUE &&
29. (pkt->dts < 0 || s->avoid\_negative\_ts == AVFMT\_AVOID\_NEG\_TS\_MAKE\_ZERO)) {
30. s->offset = -pkt->dts;
31. s->offset\_timebase = st->time\_base;
32. }
34. **if** (s->offset != AV\_NOPTS\_VALUE && !offset) {
35. offset = st->mux\_ts\_offset =
36. av\_rescale\_q\_rnd(s->offset,
37. s->offset\_timebase,
38. st->time\_base,
39. AV\_ROUND\_UP);
40. }
42. **if** (pkt->dts != AV\_NOPTS\_VALUE)
43. pkt->dts += offset;
44. **if** (pkt->pts != AV\_NOPTS\_VALUE)
45. pkt->pts += offset;
47. av\_assert2(pkt->dts == AV\_NOPTS\_VALUE || pkt->dts >= 0 || s->max\_interleave\_delta > 0);
48. **if** (pkt->dts != AV\_NOPTS\_VALUE && pkt->dts < 0) {
49. av\_log(s, AV\_LOG\_WARNING,
50. "Packets poorly interleaved, failed to avoid negative "
51. "timestamp %s in stream %d.\n"
52. "Try -max\_interleave\_delta 0 as a possible workaround.\n",
53. av\_ts2str(pkt->dts),
54. pkt->stream\_index
55. );
56. }
57. }
59. did\_split = av\_packet\_split\_side\_data(pkt);
60. **if** ((pkt->flags & AV\_PKT\_FLAG\_UNCODED\_FRAME)) {
61. AVFrame \*frame = (AVFrame \*)pkt->data;
62. av\_assert0(pkt->size == UNCODED\_FRAME\_PACKET\_SIZE);
63. ret = s->oformat->write\_uncoded\_frame(s, pkt->stream\_index, &frame, 0);
64. av\_frame\_free(&frame);
65. } **else** {
66. //写入
67. ret = s->oformat->write\_packet(s, pkt);
68. }
70. **if** (s->flush\_packets && s->pb && ret >= 0 && s->flags & AVFMT\_FLAG\_FLUSH\_PACKETS)
71. avio\_flush(s->pb);
73. **if** (did\_split)
74. av\_packet\_merge\_side\_data(pkt);
76. **return** ret;
77. }

write\_packet()函数最关键的地方就是调用了AVOutputFormat中写入数据的方法。如果AVPacket中的flag标记中包含AV\_PKT\_FLAG\_UNCODED\_FRAME，就会调用AVOutputFormat的write\_uncoded\_frame()函数；如果不包含那个标记，就会调用write\_packet()函数。write\_packet()实际上是一个函数指针，指向特定的AVOutputFormat中的实现函数。例如，我们看一下FLV对应的AVOutputFormat，位于libavformat\flvenc.c，如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44199673) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44199673)

[在CODE上查看代码片](https://code.csdn.net/snippets/616946)

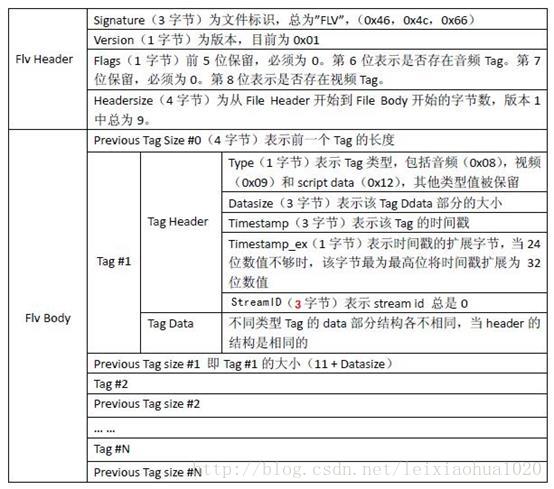
1. AVOutputFormat ff\_flv\_muxer = {
2. .name           = "flv",
3. .long\_name      = NULL\_IF\_CONFIG\_SMALL("FLV (Flash Video)"),
4. .mime\_type      = "video/x-flv",
5. .extensions     = "flv",
6. .priv\_data\_size = **sizeof**(FLVContext),
7. .audio\_codec    = CONFIG\_LIBMP3LAME ? AV\_CODEC\_ID\_MP3 : AV\_CODEC\_ID\_ADPCM\_SWF,
8. .video\_codec    = AV\_CODEC\_ID\_FLV1,
9. .write\_header   = flv\_write\_header,
10. .write\_packet   = flv\_write\_packet,
11. .write\_trailer  = flv\_write\_trailer,
12. .codec\_tag      = (**const** AVCodecTag\* **const** []) {
13. flv\_video\_codec\_ids, flv\_audio\_codec\_ids, 0
14. },
15. .flags          = AVFMT\_GLOBALHEADER | AVFMT\_VARIABLE\_FPS |
16. AVFMT\_TS\_NONSTRICT,
17. };

从ff\_flv\_muxer的定义可以看出，write\_packet()指向的是flv\_write\_packet()函数。在看flv\_write\_packet()函数的定义之前，我们先回顾一下FLV封装格式的结构。

### FLV封装格式

FLV封装格式如下图所示。

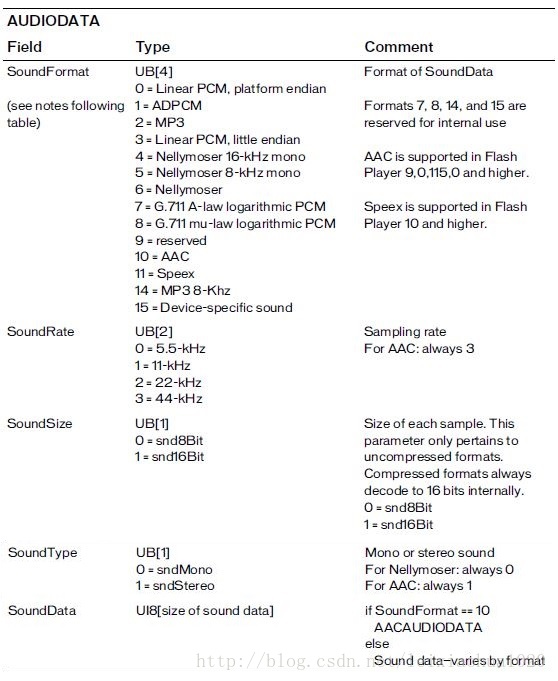
PS：原图是网上找的，感觉画的很清晰，比官方的Video File Format Specification更加通俗易懂。但是图中有一个错误，就是TagHeader中的StreamID字段的长度写错了（查看了一下官方标准，应该是3字节，现在已经改过来了）。



从FLV的封装格式结构可以看出，它的文件数据是一个一个的Tag连接起来的，中间间隔包含着Previous Tag Size。因此，flv\_write\_packet()函数的任务就是写入一个Tag和Previous Tag Size。下面简单记录一下Tag Data的格式。Tag Data根据Tag的Type不同而不同：可以分为音频Tag Data，视频Tag Data以及Script Tag Data。下面简述一下音频Tag Data和视频Tag Data。

#### Audio Tag Data

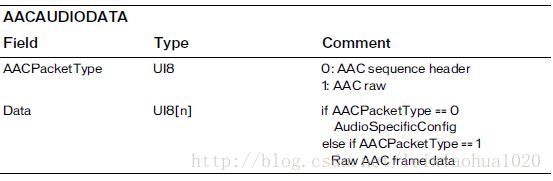
Audio Tag在官方标准中定义如下。



Audio Tag开始的第1个字节包含了音频数据的参数信息，从第2个字节开始为音频流数据。  
第1个字节的前4位的数值表示了音频数据格式：

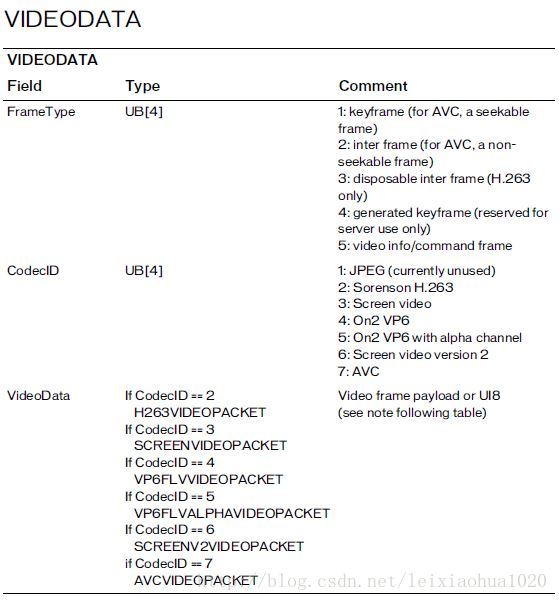
0 = Linear PCM, platform endian  
1 = ADPCM  
2 = MP3  
3 = Linear PCM, little endian  
4 = Nellymoser 16-kHz mono  
5 = Nellymoser 8-kHz mono  
6 = Nellymoser  
7 = G.711 A-law logarithmic PCM  
8 = G.711 mu-law logarithmic PCM  
9 = reserved  
10 = AAC  
14 = MP3 8-Khz  
15 = Device-specific sound

第1个字节的第5-6位的数值表示采样率：0 = 5.5kHz，1 = 11KHz，2 = 22 kHz，3 = 44 kHz。  
第1个字节的第7位表示采样精度：0 = 8bits，1 = 16bits。  
第1个字节的第8位表示音频类型：0 = sndMono，1 = sndStereo。  
其中，当音频编码为AAC的时候，第一个字节后面存储的是AACAUDIODATA，格式如下所示。



#### Video Tag Data

Video Tag在官方标准中的定义如下。



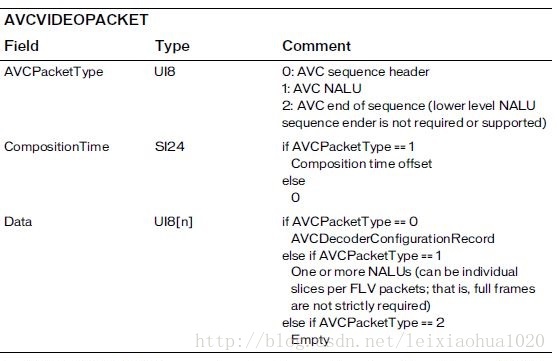
Video Tag也用开始的第1个字节包含视频数据的参数信息，从第2个字节为视频流数据。  
第1个字节的前4位的数值表示帧类型（FrameType）：

1: keyframe (for AVC, a seekableframe)（关键帧）  
2: inter frame (for AVC, a nonseekableframe)  
3: disposable inter frame (H.263only)  
4: generated keyframe (reservedfor server use only)  
5: video info/command frame

第1个字节的后4位的数值表示视频编码ID（CodecID）：

1: JPEG (currently unused)  
2: Sorenson H.263  
3: Screen video  
4: On2 VP6  
5: On2 VP6 with alpha channel  
6: Screen video version 2  
7: AVC

其中，当音频编码为AVC（H.264）的时候，第一个字节后面存储的是AVCVIDEOPACKET，格式如下所示。



## flv\_write\_packet()

下面我们看一下FLV格式中write\_packet()对应的实现函数flv\_write\_packet()的定义，位于libavformat\flvenc.c，如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44199673) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44199673)

[在CODE上查看代码片](https://code.csdn.net/snippets/616946)

1. **static** **int** flv\_write\_packet(AVFormatContext \*s, AVPacket \*pkt)
2. {
3. AVIOContext \*pb      = s->pb;
4. AVCodecContext \*enc  = s->streams[pkt->stream\_index]->codec;
5. FLVContext \*flv      = s->priv\_data;
6. FLVStreamContext \*sc = s->streams[pkt->stream\_index]->priv\_data;
7. unsigned ts;
8. **int** size = pkt->size;
9. uint8\_t \*data = NULL;
10. **int** flags = -1, flags\_size, ret;
12. **if** (enc->codec\_id == AV\_CODEC\_ID\_VP6F || enc->codec\_id == AV\_CODEC\_ID\_VP6A ||
13. enc->codec\_id == AV\_CODEC\_ID\_VP6  || enc->codec\_id == AV\_CODEC\_ID\_AAC)
14. flags\_size = 2;
15. **else** **if** (enc->codec\_id == AV\_CODEC\_ID\_H264 || enc->codec\_id == AV\_CODEC\_ID\_MPEG4)
16. flags\_size = 5;
17. **else**
18. flags\_size = 1;
20. **if** (flv->delay == AV\_NOPTS\_VALUE)
21. flv->delay = -pkt->dts;
23. **if** (pkt->dts < -flv->delay) {
24. av\_log(s, AV\_LOG\_WARNING,
25. "Packets are not in the proper order with respect to DTS\n");
26. **return** AVERROR(EINVAL);
27. }
29. ts = pkt->dts + flv->delay; // add delay to force positive dts
31. **if** (s->event\_flags & AVSTREAM\_EVENT\_FLAG\_METADATA\_UPDATED) {
32. write\_metadata(s, ts);
33. s->event\_flags &= ~AVSTREAM\_EVENT\_FLAG\_METADATA\_UPDATED;
34. }
35. //Tag Header
36. **switch** (enc->codec\_type) {
37. **case** AVMEDIA\_TYPE\_VIDEO:
38. //Type
39. avio\_w8(pb, FLV\_TAG\_TYPE\_VIDEO);
41. flags = enc->codec\_tag;
42. **if** (flags == 0) {
43. av\_log(s, AV\_LOG\_ERROR,
44. "Video codec '%s' is not compatible with FLV\n",
45. avcodec\_get\_name(enc->codec\_id));
46. **return** AVERROR(EINVAL);
47. }
48. //Key Frame?
49. flags |= pkt->flags & AV\_PKT\_FLAG\_KEY ? FLV\_FRAME\_KEY : FLV\_FRAME\_INTER;
50. **break**;
51. **case** AVMEDIA\_TYPE\_AUDIO:
53. flags = get\_audio\_flags(s, enc);
55. av\_assert0(size);
56. //Type
57. avio\_w8(pb, FLV\_TAG\_TYPE\_AUDIO);
58. **break**;
59. **case** AVMEDIA\_TYPE\_DATA:
60. //Type
61. avio\_w8(pb, FLV\_TAG\_TYPE\_META);
62. **break**;
63. **default**:
64. **return** AVERROR(EINVAL);
65. }
67. **if** (enc->codec\_id == AV\_CODEC\_ID\_H264 || enc->codec\_id == AV\_CODEC\_ID\_MPEG4) {
68. /\* check if extradata looks like mp4 formated \*/
69. **if** (enc->extradata\_size > 0 && \*(uint8\_t\*)enc->extradata != 1)
70. **if** ((ret = ff\_avc\_parse\_nal\_units\_buf(pkt->data, &data, &size)) < 0)
71. **return** ret;
72. } **else** **if** (enc->codec\_id == AV\_CODEC\_ID\_AAC && pkt->size > 2 &&
73. (AV\_RB16(pkt->data) & 0xfff0) == 0xfff0) {
74. **if** (!s->streams[pkt->stream\_index]->nb\_frames) {
75. av\_log(s, AV\_LOG\_ERROR, "Malformed AAC bitstream detected: "
76. "use the audio bitstream filter 'aac\_adtstoasc' to fix it "
77. "('-bsf:a aac\_adtstoasc' option with ffmpeg)\n");
78. **return** AVERROR\_INVALIDDATA;
79. }
80. av\_log(s, AV\_LOG\_WARNING, "aac bitstream error\n");
81. }
83. /\* check Speex packet duration \*/
84. **if** (enc->codec\_id == AV\_CODEC\_ID\_SPEEX && ts - sc->last\_ts > 160)
85. av\_log(s, AV\_LOG\_WARNING, "Warning: Speex stream has more than "
86. "8 frames per packet. Adobe Flash "
87. "Player cannot handle this!\n");
89. **if** (sc->last\_ts < ts)
90. sc->last\_ts = ts;
92. **if** (size + flags\_size >= 1<<24) {
93. av\_log(s, AV\_LOG\_ERROR, "Too large packet with size %u >= %u\n",
94. size + flags\_size, 1<<24);
95. **return** AVERROR(EINVAL);
96. }
97. //Tag Header - Datasize
98. avio\_wb24(pb, size + flags\_size);
99. //Tag Header - Timestamp
100. avio\_wb24(pb, ts & 0xFFFFFF);
101. avio\_w8(pb, (ts >> 24) & 0x7F); // timestamps are 32 bits \_signed\_
102. //StreamID
103. avio\_wb24(pb, flv->reserved);
105. **if** (enc->codec\_type == AVMEDIA\_TYPE\_DATA) {
106. **int** data\_size;
107. int64\_t metadata\_size\_pos = avio\_tell(pb);
108. **if** (enc->codec\_id == AV\_CODEC\_ID\_TEXT) {
109. // legacy FFmpeg magic?
110. avio\_w8(pb, AMF\_DATA\_TYPE\_STRING);
111. put\_amf\_string(pb, "onTextData");
112. avio\_w8(pb, AMF\_DATA\_TYPE\_MIXEDARRAY);
113. avio\_wb32(pb, 2);
114. put\_amf\_string(pb, "type");
115. avio\_w8(pb, AMF\_DATA\_TYPE\_STRING);
116. put\_amf\_string(pb, "Text");
117. put\_amf\_string(pb, "text");
118. avio\_w8(pb, AMF\_DATA\_TYPE\_STRING);
119. put\_amf\_string(pb, pkt->data);
120. put\_amf\_string(pb, "");
121. avio\_w8(pb, AMF\_END\_OF\_OBJECT);
122. } **else** {
123. // just pass the metadata through
124. avio\_write(pb, data ? data : pkt->data, size);
125. }
126. /\* write total size of tag \*/
127. data\_size = avio\_tell(pb) - metadata\_size\_pos;
128. avio\_seek(pb, metadata\_size\_pos - 10, SEEK\_SET);
129. avio\_wb24(pb, data\_size);
130. avio\_seek(pb, data\_size + 10 - 3, SEEK\_CUR);
131. avio\_wb32(pb, data\_size + 11);
132. } **else** {
133. av\_assert1(flags>=0);
134. //First Byte of Tag Data
135. avio\_w8(pb,flags);
136. **if** (enc->codec\_id == AV\_CODEC\_ID\_VP6)
137. avio\_w8(pb,0);
138. **if** (enc->codec\_id == AV\_CODEC\_ID\_VP6F || enc->codec\_id == AV\_CODEC\_ID\_VP6A) {
139. **if** (enc->extradata\_size)
140. avio\_w8(pb, enc->extradata[0]);
141. **else**
142. avio\_w8(pb, ((FFALIGN(enc->width,  16) - enc->width) << 4) |
143. (FFALIGN(enc->height, 16) - enc->height));
144. } **else** **if** (enc->codec\_id == AV\_CODEC\_ID\_AAC)
145. avio\_w8(pb, 1); // AAC raw
146. **else** **if** (enc->codec\_id == AV\_CODEC\_ID\_H264 || enc->codec\_id == AV\_CODEC\_ID\_MPEG4) {
147. //AVCVIDEOPACKET-AVCPacketType
148. avio\_w8(pb, 1); // AVC NALU
149. //AVCVIDEOPACKET-CompositionTime
150. avio\_wb24(pb, pkt->pts - pkt->dts);
151. }
152. //Data
153. avio\_write(pb, data ? data : pkt->data, size);
155. avio\_wb32(pb, size + flags\_size + 11); // previous tag size
156. flv->duration = FFMAX(flv->duration,
157. pkt->pts + flv->delay + pkt->duration);
158. }
160. av\_free(data);
162. **return** pb->error;
163. }

我们通过源代码简单梳理一下flv\_write\_packet()在写入H.264/AAC时候的流程：  
（1）写入Tag Header的Type，如果是视频，代码如下：

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44199673) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44199673)

[在CODE上查看代码片](https://code.csdn.net/snippets/616946)

1. avio\_w8(pb, FLV\_TAG\_TYPE\_VIDEO);

如果是音频，代码如下：

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44199673) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44199673)

[在CODE上查看代码片](https://code.csdn.net/snippets/616946)

1. avio\_w8(pb, FLV\_TAG\_TYPE\_AUDIO);

（2）写入Tag Header的Datasize，Timestamp和StreamID（至此完成Tag Header）：

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44199673) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44199673)

[在CODE上查看代码片](https://code.csdn.net/snippets/616946)

1. //Tag Header - Datasize
2. avio\_wb24(pb, size + flags\_size);
3. //Tag Header - Timestamp
4. avio\_wb24(pb, ts & 0xFFFFFF);
5. avio\_w8(pb, (ts >> 24) & 0x7F); // timestamps are 32 bits \_signed\_
6. //StreamID
7. avio\_wb24(pb, flv->reserved);

（3）写入Tag Data的第一字节（其中flag已经在前面的代码中设置完毕）：

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44199673) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44199673)

[在CODE上查看代码片](https://code.csdn.net/snippets/616946)

1. //First Byte of Tag Data
2. avio\_w8(pb,flags);

（4）如果编码格式VP6作相应的处理（不研究）；编码格式为AAC，写入AACAUDIODATA；编码格式为H.264，写入AVCVIDEOPACKET：

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44199673) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44199673)

[在CODE上查看代码片](https://code.csdn.net/snippets/616946)

1. **if** (enc->codec\_id == AV\_CODEC\_ID\_VP6F || enc->codec\_id == AV\_CODEC\_ID\_VP6A) {
2. **if** (enc->extradata\_size)
3. avio\_w8(pb, enc->extradata[0]);
4. **else**
5. avio\_w8(pb, ((FFALIGN(enc->width,  16) - enc->width) << 4) |
6. (FFALIGN(enc->height, 16) - enc->height));
7. } **else** **if** (enc->codec\_id == AV\_CODEC\_ID\_AAC)
8. avio\_w8(pb, 1); // AAC raw
9. **else** **if** (enc->codec\_id == AV\_CODEC\_ID\_H264 || enc->codec\_id == AV\_CODEC\_ID\_MPEG4) {
10. //AVCVIDEOPACKET-AVCPacketType
11. avio\_w8(pb, 1); // AVC NALU
12. //AVCVIDEOPACKET-CompositionTime
13. avio\_wb24(pb, pkt->pts - pkt->dts);
14. }

（5）写入数据：

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44199673) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44199673)

[在CODE上查看代码片](https://code.csdn.net/snippets/616946)

1. //Data
2. avio\_write(pb, data ? data : pkt->data, size);

（6） 写入previous tag size：

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44199673) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44199673)

[在CODE上查看代码片](https://code.csdn.net/snippets/616946)

1. avio\_wb32(pb, size + flags\_size + 11); // previous tag size

至此，flv\_write\_packet()就完成了一个Tag的写入。

# [FFmpeg源代码简单分析：av\_write\_trailer()](http://blog.csdn.net/leixiaohua1020/article/details/44201645)

打算写两篇文章简单分析FFmpeg的写文件用到的3个函数avformat\_write\_header()，av\_write\_frame()以及av\_write\_trailer()。这篇文章继续分析av\_write\_trailer()。

av\_write\_trailer()用于输出文件尾，它的声明位于libavformat\avformat.h，如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44201645) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44201645)

[在CODE上查看代码片](https://code.csdn.net/snippets/617060)

1. /\*\*
2. \* Write the stream trailer to an output media file and free the
3. \* file private data.
4. \*
5. \* May only be called after a successful call to avformat\_write\_header.
6. \*
7. \* @param s media file handle
8. \* @return 0 if OK, AVERROR\_xxx on error
9. \*/
10. **int** av\_write\_trailer(AVFormatContext \*s);

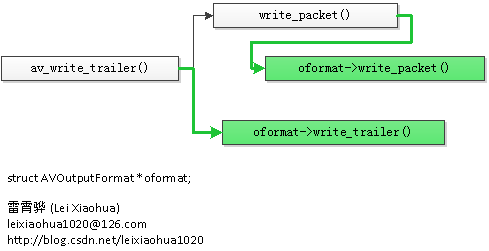
它只需要指定一个参数，即用于输出的AVFormatContext。  
函数正常执行后返回值等于0。

这2个函数最典型的例子可以参考：

[最简单的基于FFMPEG的视频编码器（YUV编码为H.264）](http://blog.csdn.net/leixiaohua1020/article/details/25430425)

## 函数调用关系图

av\_write\_trailer()的调用关系如下图所示。



## av\_write\_trailer()

av\_write\_trailer()的定义位于libavformat\mux.c，如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44201645) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44201645)

[在CODE上查看代码片](https://code.csdn.net/snippets/617060)

1. **int** av\_write\_trailer(AVFormatContext \*s)
2. {
3. **int** ret, i;
5. **for** (;; ) {
6. AVPacket pkt;
7. ret = interleave\_packet(s, &pkt, NULL, 1);
8. **if** (ret < 0)
9. **goto** fail;
10. **if** (!ret)
11. **break**;
12. //写入AVPacket
13. ret = write\_packet(s, &pkt);
14. **if** (ret >= 0)
15. s->streams[pkt.stream\_index]->nb\_frames++;
17. av\_free\_packet(&pkt);
19. **if** (ret < 0)
20. **goto** fail;
21. **if**(s->pb && s->pb->error)
22. **goto** fail;
23. }
25. fail:
26. //写文件尾
27. **if** (s->oformat->write\_trailer)
28. **if** (ret >= 0) {
29. ret = s->oformat->write\_trailer(s);
30. } **else** {
31. s->oformat->write\_trailer(s);
32. }
34. **if** (s->pb)
35. avio\_flush(s->pb);
36. **if** (ret == 0)
37. ret = s->pb ? s->pb->error : 0;
38. **for** (i = 0; i < s->nb\_streams; i++) {
39. av\_freep(&s->streams[i]->priv\_data);
40. av\_freep(&s->streams[i]->index\_entries);
41. }
42. **if** (s->oformat->priv\_class)
43. av\_opt\_free(s->priv\_data);
44. av\_freep(&s->priv\_data);
45. **return** ret;
46. }

从源代码可以看出av\_write\_trailer()主要完成了以下两步工作：

（1）循环调用interleave\_packet()以及write\_packet()，将还未输出的AVPacket输出出来。

（2）调用AVOutputFormat的write\_trailer()，输出文件尾。

其中第一步和av\_write\_frame()中的步骤大致是一样的（interleave\_packet()这一部分在并不包含在av\_write\_frame()中，而是包含在av\_interleaved\_write\_frame()中，这一部分源代码还没有分析），可以参考文章《[FFmpeg源代码简单分析：av\_write\_frame()](http://blog.csdn.net/leixiaohua1020/article/details/44199673)》。下面分析一下第二步。

## AVOutputFormat->write\_trailer()

AVOutputFormat的write\_trailer()是一个函数指针，指向特定的AVOutputFormat中的实现函数。我们以FLV对应的AVOutputFormat为例，看一下它的定义，如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44201645) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44201645)

[在CODE上查看代码片](https://code.csdn.net/snippets/617060)

1. AVOutputFormat ff\_flv\_muxer = {
2. .name           = "flv",
3. .long\_name      = NULL\_IF\_CONFIG\_SMALL("FLV (Flash Video)"),
4. .mime\_type      = "video/x-flv",
5. .extensions     = "flv",
6. .priv\_data\_size = **sizeof**(FLVContext),
7. .audio\_codec    = CONFIG\_LIBMP3LAME ? AV\_CODEC\_ID\_MP3 : AV\_CODEC\_ID\_ADPCM\_SWF,
8. .video\_codec    = AV\_CODEC\_ID\_FLV1,
9. .write\_header   = flv\_write\_header,
10. .write\_packet   = flv\_write\_packet,
11. .write\_trailer  = flv\_write\_trailer,
12. .codec\_tag      = (**const** AVCodecTag\* **const** []) {
13. flv\_video\_codec\_ids, flv\_audio\_codec\_ids, 0
14. },
15. .flags          = AVFMT\_GLOBALHEADER | AVFMT\_VARIABLE\_FPS |
16. AVFMT\_TS\_NONSTRICT,
17. };

从FLV对应的AVOutputFormat结构体的定义我们可以看出，write\_trailer()指向了flv\_write\_trailer()函数。

## flv\_write\_trailer()

flv\_write\_trailer()函数的定义位于libavformat\flvenc.c，如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44201645) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44201645)

[在CODE上查看代码片](https://code.csdn.net/snippets/617060)

1. **static** **int** flv\_write\_trailer(AVFormatContext \*s)
2. {
3. int64\_t file\_size;
5. AVIOContext \*pb = s->pb;
6. FLVContext \*flv = s->priv\_data;
7. **int** i;
9. /\* Add EOS tag \*/
10. **for** (i = 0; i < s->nb\_streams; i++) {
11. AVCodecContext \*enc = s->streams[i]->codec;
12. FLVStreamContext \*sc = s->streams[i]->priv\_data;
13. **if** (enc->codec\_type == AVMEDIA\_TYPE\_VIDEO &&
14. (enc->codec\_id == AV\_CODEC\_ID\_H264 || enc->codec\_id == AV\_CODEC\_ID\_MPEG4))
15. put\_avc\_eos\_tag(pb, sc->last\_ts);
16. }
18. file\_size = avio\_tell(pb);
20. /\* update information \*/
21. **if** (avio\_seek(pb, flv->duration\_offset, SEEK\_SET) < 0)
22. av\_log(s, AV\_LOG\_WARNING, "Failed to update header with correct duration.\n");
23. **else**
24. put\_amf\_double(pb, flv->duration / (**double**)1000);
25. **if** (avio\_seek(pb, flv->filesize\_offset, SEEK\_SET) < 0)
26. av\_log(s, AV\_LOG\_WARNING, "Failed to update header with correct filesize.\n");
27. **else**
28. put\_amf\_double(pb, file\_size);
30. avio\_seek(pb, file\_size, SEEK\_SET);
31. **return** 0;
32. }

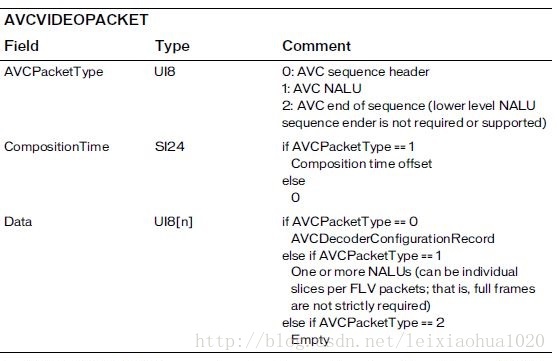
从flv\_write\_trailer()的源代码可以看出该函数做了以下两步工作：  
（1）如果视频流是H.264，则添加包含EOS（End Of Stream） NALU的Tag。  
（2）更新FLV的时长信息，以及文件大小信息。  
其中，put\_avc\_eos\_tag()函数用于添加包含EOS NALU的Tag（包含结尾的一个PreviousTagSize），如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44201645) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44201645)

[在CODE上查看代码片](https://code.csdn.net/snippets/617060)

1. **static** **void** put\_avc\_eos\_tag(AVIOContext \*pb, unsigned ts)
2. {
3. avio\_w8(pb, FLV\_TAG\_TYPE\_VIDEO);
4. avio\_wb24(pb, 5);               /\* Tag Data Size \*/
5. avio\_wb24(pb, ts);              /\* lower 24 bits of timestamp in ms \*/
6. avio\_w8(pb, (ts >> 24) & 0x7F); /\* MSB of ts in ms \*/
7. avio\_wb24(pb, 0);               /\* StreamId = 0 \*/
8. avio\_w8(pb, 23);                /\* ub[4] FrameType = 1, ub[4] CodecId = 7 \*/
9. avio\_w8(pb, 2);                 /\* AVC end of sequence \*/
10. avio\_wb24(pb, 0);               /\* Always 0 for AVC EOS. \*/
11. avio\_wb32(pb, 16);              /\* Size of FLV tag \*/
12. }

可以参考FLV封装格式理解上述函数。由于前面的文章中已经描述过FLV封装格式，在这里不再重复叙述，在这里仅在此记录一下AVCVIDEOPACKET的格式，如下所示。



可以看出包含EOS NALU的AVCVIDEOPACKET的AVCPacketType为2。在这种情况下，AVCVIDEOPACKET的CompositionTime字段取0，并且无需包含Data字段。

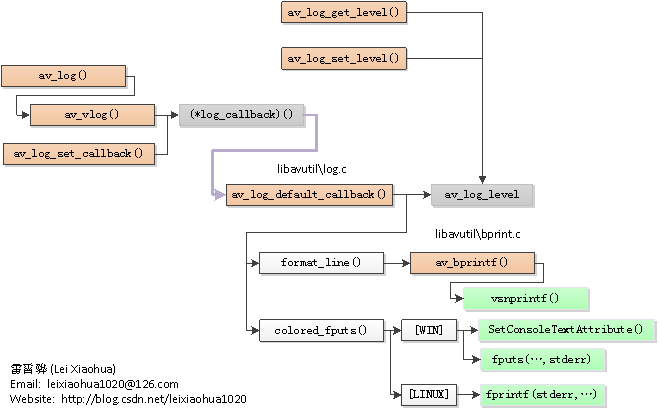
# [FFmpeg源代码简单分析：日志输出系统（av\_log()等）](http://blog.csdn.net/leixiaohua1020/article/details/44243155)

本文分析一下FFmpeg的日志（Log）输出系统的源代码。日志输出部分的核心函数只有一个：av\_log()。使用av\_log()在控制台输出日志的效果如下图所示。



## 函数调用结构图

FFmpeg日志输出系统的函数调用结构图如图所示。



## av\_log()

av\_log()是FFmpeg中输出日志的函数。随便打开一个FFmpeg的源代码文件，就会发现其中遍布着av\_log()函数。一般情况下FFmpeg类库的源代码中是不允许使用printf()这种的函数的，所有的输出一律使用av\_log()。  
av\_log()的声明位于libavutil\log.h，如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44243155) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44243155)

[在CODE上查看代码片](https://code.csdn.net/snippets/619180)

1. /\*\*
2. \* Send the specified message to the log if the level is less than or equal
3. \* to the current av\_log\_level. By default, all logging messages are sent to
4. \* stderr. This behavior can be altered by setting a different logging callback
5. \* function.
6. \* @see av\_log\_set\_callback
7. \*
8. \* @param avcl A pointer to an arbitrary struct of which the first field is a
9. \*        pointer to an AVClass struct.
10. \* @param level The importance level of the message expressed using a @ref
11. \*        lavu\_log\_constants "Logging Constant".
12. \* @param fmt The format string (printf-compatible) that specifies how
13. \*        subsequent arguments are converted to output.
14. \*/
15. **void** av\_log(**void** \*avcl, **int** level, **const** **char** \*fmt, ...) av\_printf\_format(3, 4);

这个函数的声明有两个地方比较特殊：

（1）函数最后一个参数是“…”。  
在C语言中，在函数参数数量不确定的情况下使用“…”来代表参数。例如printf()的原型定义如下

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44243155) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44243155)

[在CODE上查看代码片](https://code.csdn.net/snippets/619180)

1. **int** printf (**const** **char**\*, ...);

后文中对此再作详细分析。

（2）它的声明后面有一个av\_printf\_format(3, 4)。有关这个地方的左右还没有深入研究，网上资料中说它的作用是按照printf()的格式检查av\_log()的格式。

av\_log()每个字段的含义如下：  
avcl：指定一个包含AVClass的结构体。  
level：log的级别  
fmt：和printf()一样。

由此可见，av\_log()和printf()的不同主要在于前面多了两个参数。其中第一个参数指定该log所属的结构体，例如AVFormatContext、AVCodecContext等等。第二个参数指定log的级别，源代码中定义了如下几个级别。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44243155) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44243155)

[在CODE上查看代码片](https://code.csdn.net/snippets/619180)

1. /\*\*
2. \* Print no output.
3. \*/
4. #define AV\_LOG\_QUIET    -8
6. /\*\*
7. \* Something went really wrong and we will crash now.
8. \*/
9. #define AV\_LOG\_PANIC     0
11. /\*\*
12. \* Something went wrong and recovery is not possible.
13. \* For example, no header was found for a format which depends
14. \* on headers or an illegal combination of parameters is used.
15. \*/
16. #define AV\_LOG\_FATAL     8
18. /\*\*
19. \* Something went wrong and cannot losslessly be recovered.
20. \* However, not all future data is affected.
21. \*/
22. #define AV\_LOG\_ERROR    16
24. /\*\*
25. \* Something somehow does not look correct. This may or may not
26. \* lead to problems. An example would be the use of '-vstrict -2'.
27. \*/
28. #define AV\_LOG\_WARNING  24
30. /\*\*
31. \* Standard information.
32. \*/
33. #define AV\_LOG\_INFO     32
35. /\*\*
36. \* Detailed information.
37. \*/
38. #define AV\_LOG\_VERBOSE  40
40. /\*\*
41. \* Stuff which is only useful for libav\* developers.
42. \*/
43. #define AV\_LOG\_DEBUG    48

从定义中可以看出来，随着严重程度逐渐下降，一共包含如下级别：AV\_LOG\_PANIC，AV\_LOG\_FATAL，AV\_LOG\_ERROR，AV\_LOG\_WARNING，AV\_LOG\_INFO，AV\_LOG\_VERBOSE，AV\_LOG\_DEBUG。每个级别定义的数值代表了严重程度，数值越小代表越严重。默认的级别是AV\_LOG\_INFO。此外，还有一个级别不输出任何信息，即AV\_LOG\_QUIET。

当前系统存在着一个“Log级别”。所有严重程度高于该级别的Log信息都会输出出来。例如当前的Log级别是AV\_LOG\_WARNING，则会输出AV\_LOG\_PANIC，AV\_LOG\_FATAL，AV\_LOG\_ERROR，AV\_LOG\_WARNING级别的信息，而不会输出AV\_LOG\_INFO级别的信息。可以通过av\_log\_get\_level()获得当前Log的级别，通过另一个函数av\_log\_set\_level()设置当前的Log级别。

## av\_log\_get\_level(), av\_log\_set\_level()

av\_log\_get\_level()的定义如下所示

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44243155) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44243155)

[在CODE上查看代码片](https://code.csdn.net/snippets/619180)

1. /\*\*
2. \* Get the current log level
3. \*
4. \* @see lavu\_log\_constants
5. \*
6. \* @return Current log level
7. \*/
8. **int** av\_log\_get\_level(**void**);

可以通过av\_log\_set\_level()设置当前Log的级别。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44243155) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44243155)

[在CODE上查看代码片](https://code.csdn.net/snippets/619180)

1. /\*\*
2. \* Set the log level
3. \*
4. \* @see lavu\_log\_constants
5. \*
6. \* @param level Logging level
7. \*/
8. **void** av\_log\_set\_level(**int** level);

上述两个函数的定义十分的简单，如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44243155) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44243155)

[在CODE上查看代码片](https://code.csdn.net/snippets/619180)

1. **int** av\_log\_get\_level(**void**)
2. {
3. **return** av\_log\_level;
4. }

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44243155) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44243155)

[在CODE上查看代码片](https://code.csdn.net/snippets/619180)

1. **void** av\_log\_set\_level(**int** level)
2. {
3. av\_log\_level = level;
4. }

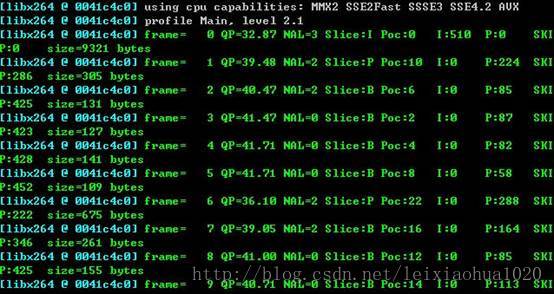
从代码中可以看出，以上两个函数主要操作了一个静态全局变量av\_log\_level。该变量用于存储当前系统Log的级别。它的定义如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44243155) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44243155)

[在CODE上查看代码片](https://code.csdn.net/snippets/619180)

1. **static** **int** av\_log\_level = AV\_LOG\_INFO;

下面我们看一下H.264编码的时候libx264的Log输出的示例：



下面回到av\_log()函数的源代码。它的源代码位于libavutil\log.c，如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44243155) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44243155)

[在CODE上查看代码片](https://code.csdn.net/snippets/619180)

1. **void** av\_log(**void**\* avcl, **int** level, **const** **char** \*fmt, ...)
2. {
3. AVClass\* avc = avcl ? \*(AVClass \*\*) avcl : NULL;
4. **va\_list** vl;
5. va\_start(vl, fmt);
6. **if** (avc && avc->version >= (50 << 16 | 15 << 8 | 2) &&
7. avc->log\_level\_offset\_offset && level >= AV\_LOG\_FATAL)
8. level += \*(**int** \*) (((uint8\_t \*) avcl) + avc->log\_level\_offset\_offset);
9. av\_vlog(avcl, level, fmt, vl);
10. va\_end(vl);
11. }

首先来提一下C语言函数中“…”参数的含义。与它相关还涉及到以下4个部分：

（1）va\_list变量  
（2）va\_start()  
（3）va\_arg()  
（4）va\_end()

va\_list是一个指向函数的参数的指针。va\_start()用于初始化va\_list变量。va\_arg()用于返回可变参数。va\_start()用于结束可变参数的获取。有关它们的用法可以参考一个小demo，如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44243155) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44243155)

[在CODE上查看代码片](https://code.csdn.net/snippets/619180)

1. #include <stdio.h>
2. #include<stdarg.h>
3. **void** fun(**int** a,...){
4. **va\_list** pp;
5. va\_start(pp,a);
6. **do**{
7. printf("param =%d\n",a);
8. a=va\_arg(pp,**int**);//使 pp 指向下一个参数，将下一个参数的值赋给变量 a
9. }
10. **while** (a!=0);//直到参数为 0 时停止循环
11. }
12. **void** main(){
13. fun(20,40,60,80,0);
14. }

有关这方面的知识很难用简短的语言描述清楚，因此不再详述。av\_log()的源代码中，在va\_start()和va\_end()之间，调用了另一个函数av\_vlog()。

## av\_vlog()

av\_vlog()是一个FFmpeg的API函数。它的声明位于libavutil\log.h中，如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44243155) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44243155)

[在CODE上查看代码片](https://code.csdn.net/snippets/619180)

1. /\*\*
2. \* Send the specified message to the log if the level is less than or equal
3. \* to the current av\_log\_level. By default, all logging messages are sent to
4. \* stderr. This behavior can be altered by setting a different logging callback
5. \* function.
6. \* @see av\_log\_set\_callback
7. \*
8. \* @param avcl A pointer to an arbitrary struct of which the first field is a
9. \*        pointer to an AVClass struct.
10. \* @param level The importance level of the message expressed using a @ref
11. \*        lavu\_log\_constants "Logging Constant".
12. \* @param fmt The format string (printf-compatible) that specifies how
13. \*        subsequent arguments are converted to output.
14. \* @param vl The arguments referenced by the format string.
15. \*/
16. **void** av\_vlog(**void** \*avcl, **int** level, **const** **char** \*fmt, **va\_list** vl);

从声明中可以看出，av\_vlog()和av\_log()的参数基本上是一模一样的。唯一的不同在于av\_log()中的“…”变成了av\_vlog()中的va\_list。

av\_vlog()的定义位于libavutil\log.c中，如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44243155) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44243155)

[在CODE上查看代码片](https://code.csdn.net/snippets/619180)

1. **void** av\_vlog(**void**\* avcl, **int** level, **const** **char** \*fmt, **va\_list** vl)
2. {
3. **void** (\*log\_callback)(**void**\*, **int**, **const** **char**\*, **va\_list**) = av\_log\_callback;
4. **if** (log\_callback)
5. log\_callback(avcl, level, fmt, vl);
6. }

从定义中可以看出，av\_vlog()简单调用了一个函数指针av\_log\_callback。av\_log\_callback是一个全局静态变量，定义如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44243155) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44243155)

[在CODE上查看代码片](https://code.csdn.net/snippets/619180)

1. **static** **void** (\*av\_log\_callback)(**void**\*, **int**, **const** **char**\*, **va\_list**) =
2. av\_log\_default\_callback;

从代码中可以看出，av\_log\_callback指针默认指向一个函数av\_log\_default\_callback()。av\_log\_default\_callback()即FFmpeg默认的Log函数。需要注意的是，这个Log函数是可以自定义的。按照指定的参数定义一个自定义的函数后，可以通过FFmpeg的另一个API函数av\_log\_set\_callback()设定为Log函数。  
av\_log\_set\_callback()的声明如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44243155) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44243155)

[在CODE上查看代码片](https://code.csdn.net/snippets/619180)

1. /\*\*
2. \* Set the logging callback
3. \*
4. \* @note The callback must be thread safe, even if the application does not use
5. \*       threads itself as some codecs are multithreaded.
6. \*
7. \* @see av\_log\_default\_callback
8. \*
9. \* @param callback A logging function with a compatible signature.
10. \*/
11. **void** av\_log\_set\_callback(**void** (\*callback)(**void**\*, **int**, **const** **char**\*, **va\_list**));

从声明中可以看出，需要指定一个参数为(void\*, int, const char\*, va\_list)，返回值为void的函数作为Log函数。  
av\_log\_set\_callback()的定义很简单，做了一个函数指针赋值的工作，如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44243155) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44243155)

[在CODE上查看代码片](https://code.csdn.net/snippets/619180)

1. **void** av\_log\_set\_callback(**void** (\*callback)(**void**\*, **int**, **const** **char**\*, **va\_list**))
2. {
3. av\_log\_callback = callback;
4. }

例如，我们可以指定一个my\_logoutput()函数作为Log的输出函数，就可以将Log信息输出到文本中（而不是屏幕上）。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44243155) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44243155)

[在CODE上查看代码片](https://code.csdn.net/snippets/619180)

1. **void** my\_logoutput(**void**\* ptr, **int** level, **const** **char**\* fmt,**va\_list** vl){
2. **FILE** \*fp = fopen("my\_log.txt","a+");
3. **if**(fp){
4. vfprintf(fp,fmt,vl);
5. fflush(fp);
6. fclose(fp);
7. }
8. }

编辑好函数之后，使用av\_log\_set\_callback()函数设置该函数为Log输出函数即可。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44243155) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44243155)

[在CODE上查看代码片](https://code.csdn.net/snippets/619180)

1. av\_log\_set\_callback(my\_logoutput);

## av\_log\_default\_callback()

下面我们看一下FFmpeg的默认Log输出函数av\_log\_default\_callback()。它的定义如下。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44243155) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44243155)

[在CODE上查看代码片](https://code.csdn.net/snippets/619180)

1. **void** av\_log\_default\_callback(**void**\* ptr, **int** level, **const** **char**\* fmt, **va\_list** vl)
2. {
3. **static** **int** print\_prefix = 1;
4. **static** **int** count;
5. **static** **char** prev[LINE\_SZ];
6. AVBPrint part[4];
7. **char** line[LINE\_SZ];
8. **static** **int** is\_atty;
9. **int** type[2];
10. unsigned tint = 0;
12. **if** (level >= 0) {
13. tint = level & 0xff00;
14. level &= 0xff;
15. }
17. **if** (level > av\_log\_level)
18. **return**;
19. #if HAVE\_PTHREADS
20. pthread\_mutex\_lock(&mutex);
21. #endif
23. format\_line(ptr, level, fmt, vl, part, &print\_prefix, type);
24. snprintf(line, **sizeof**(line), "%s%s%s%s", part[0].str, part[1].str, part[2].str, part[3].str);
26. #if HAVE\_ISATTY
27. **if** (!is\_atty)
28. is\_atty = isatty(2) ? 1 : -1;
29. #endif
31. **if** (print\_prefix && (flags & AV\_LOG\_SKIP\_REPEATED) && !strcmp(line, prev) &&
32. \*line && line[strlen(line) - 1] != '\r'){
33. count++;
34. **if** (is\_atty == 1)
35. fprintf(stderr, "    Last message repeated %d times\r", count);
36. **goto** end;
37. }
38. **if** (count > 0) {
39. fprintf(stderr, "    Last message repeated %d times\n", count);
40. count = 0;
41. }
42. strcpy(prev, line);
43. sanitize(part[0].str);
44. colored\_fputs(type[0], 0, part[0].str);
45. sanitize(part[1].str);
46. colored\_fputs(type[1], 0, part[1].str);
47. sanitize(part[2].str);
48. colored\_fputs(av\_clip(level >> 3, 0, 6), tint >> 8, part[2].str);
49. sanitize(part[3].str);
50. colored\_fputs(av\_clip(level >> 3, 0, 6), tint >> 8, part[3].str);
51. end:
52. av\_bprint\_finalize(part+3, NULL);
53. #if HAVE\_PTHREADS
54. pthread\_mutex\_unlock(&mutex);
55. #endif
56. }

av\_log\_default\_callback()的代码是比较复杂的。其实如果我们仅仅是希望把Log信息输出到屏幕上，远不需要那么多代码，只需要简单打印一下就可以了。av\_log\_default\_callback()之所以会那么复杂，主要是因为他还包含了很多的功能，比如说根据Log级别的不同将输出的文本设置成不同的颜色等等。下图显示了不同级别的Log不同的背景颜色。



下面看一下av\_log\_default\_callback()的源代码大致的流程：

（1）如果输入参数level大于系统当前的日志级别av\_log\_level，表明不需要做任何处理，直接返回。  
（2）调用format\_line()设定Log的输出格式。  
（3）调用colored\_fputs()设定Log的颜色。

### format\_line(), av\_log\_format\_line()

format\_line()用于设定Log的输出格式。它本身并不是一个FFmpeg的API，但是FFmpeg有一个API函数av\_log\_format\_line()调用了这个函数。av\_log\_format\_line()的声明如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44243155) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44243155)

[在CODE上查看代码片](https://code.csdn.net/snippets/619180)

1. /\*\*
2. \* Format a line of log the same way as the default callback.
3. \* @param line          buffer to receive the formated line
4. \* @param line\_size     size of the buffer
5. \* @param print\_prefix  used to store whether the prefix must be printed;
6. \*                      must point to a persistent integer initially set to 1
7. \*/
8. **void** av\_log\_format\_line(**void** \*ptr, **int** level, **const** **char** \*fmt, **va\_list** vl,
9. **char** \*line, **int** line\_size, **int** \*print\_prefix);

av\_log\_format\_line()的定义如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44243155) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44243155)

[在CODE上查看代码片](https://code.csdn.net/snippets/619180)

1. **void** av\_log\_format\_line(**void** \*ptr, **int** level, **const** **char** \*fmt, **va\_list** vl,
2. **char** \*line, **int** line\_size, **int** \*print\_prefix)
3. {
4. AVBPrint part[4];
5. format\_line(ptr, level, fmt, vl, part, print\_prefix, NULL);
6. snprintf(line, line\_size, "%s%s%s%s", part[0].str, part[1].str, part[2].str, part[3].str);
7. av\_bprint\_finalize(part+3, NULL);
8. }

从代码中可以看出，首先声明了一个AVBPrint类型的数组，其中包含了4个元素；接着调用format\_line()设定格式；最后将设置格式后的AVBPrint数组中的4个元素连接起来。  
在这里遇到了一个结构体AVBPrint，它的定义位于libavutil\bprint.h，如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44243155) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44243155)

[在CODE上查看代码片](https://code.csdn.net/snippets/619180)

1. /\*\*
2. \* Buffer to print data progressively
3. \*
4. \* The string buffer grows as necessary and is always 0-terminated.
5. \* The content of the string is never accessed, and thus is
6. \* encoding-agnostic and can even hold binary data.
7. \*
8. \* Small buffers are kept in the structure itself, and thus require no
9. \* memory allocation at all (unless the contents of the buffer is needed
10. \* after the structure goes out of scope). This is almost as lightweight as
11. \* declaring a local "char buf[512]".
12. \*
13. \* The length of the string can go beyond the allocated size: the buffer is
14. \* then truncated, but the functions still keep account of the actual total
15. \* length.
16. \*
17. \* In other words, buf->len can be greater than buf->size and records the
18. \* total length of what would have been to the buffer if there had been
19. \* enough memory.
20. \*
21. \* Append operations do not need to be tested for failure: if a memory
22. \* allocation fails, data stop being appended to the buffer, but the length
23. \* is still updated. This situation can be tested with
24. \* av\_bprint\_is\_complete().
25. \*
26. \* The size\_max field determines several possible behaviours:
27. \*
28. \* size\_max = -1 (= UINT\_MAX) or any large value will let the buffer be
29. \* reallocated as necessary, with an amortized linear cost.
30. \*
31. \* size\_max = 0 prevents writing anything to the buffer: only the total
32. \* length is computed. The write operations can then possibly be repeated in
33. \* a buffer with exactly the necessary size
34. \* (using size\_init = size\_max = len + 1).
35. \*
36. \* size\_max = 1 is automatically replaced by the exact size available in the
37. \* structure itself, thus ensuring no dynamic memory allocation. The
38. \* internal buffer is large enough to hold a reasonable paragraph of text,
39. \* such as the current paragraph.
40. \*/
41. **typedef** **struct** AVBPrint {
42. FF\_PAD\_STRUCTURE(1024,
43. **char** \*str;         /\*\*< string so far \*/
44. unsigned len;      /\*\*< length so far \*/
45. unsigned size;     /\*\*< allocated memory \*/
46. unsigned size\_max; /\*\*< maximum allocated memory \*/
47. **char** reserved\_internal\_buffer[1];
48. )
49. } AVBPrint;

AVBPrint的注释代码很长，不再详细叙述。在这里只要知道他是用于打印字符串的缓存就可以了。它的名称BPrint的意思应该就是“Buffer to Print”。其中的str存储了将要打印的字符串。  
format\_line()函数的定义如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44243155) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44243155)

[在CODE上查看代码片](https://code.csdn.net/snippets/619180)

1. **static** **void** format\_line(**void** \*avcl, **int** level, **const** **char** \*fmt, **va\_list** vl,
2. AVBPrint part[4], **int** \*print\_prefix, **int** type[2])
3. {
4. AVClass\* avc = avcl ? \*(AVClass \*\*) avcl : NULL;
5. av\_bprint\_init(part+0, 0, 1);
6. av\_bprint\_init(part+1, 0, 1);
7. av\_bprint\_init(part+2, 0, 1);
8. av\_bprint\_init(part+3, 0, 65536);
10. **if**(type) type[0] = type[1] = AV\_CLASS\_CATEGORY\_NA + 16;
11. **if** (\*print\_prefix && avc) {
12. **if** (avc->parent\_log\_context\_offset) {
13. AVClass\*\* parent = \*(AVClass \*\*\*) (((uint8\_t \*) avcl) +
14. avc->parent\_log\_context\_offset);
15. **if** (parent && \*parent) {
16. av\_bprintf(part+0, "[%s @ %p] ",
17. (\*parent)->item\_name(parent), parent);
18. **if**(type) type[0] = get\_category(parent);
19. }
20. }
21. av\_bprintf(part+1, "[%s @ %p] ",
22. avc->item\_name(avcl), avcl);
23. **if**(type) type[1] = get\_category(avcl);
25. **if** (flags & AV\_LOG\_PRINT\_LEVEL)
26. av\_bprintf(part+2, "[%s] ", get\_level\_str(level));
27. }
29. av\_vbprintf(part+3, fmt, vl);
31. **if**(\*part[0].str || \*part[1].str || \*part[2].str || \*part[3].str) {
32. **char** lastc = part[3].len && part[3].len <= part[3].size ? part[3].str[part[3].len - 1] : 0;
33. \*print\_prefix = lastc == '\n' || lastc == '\r';
34. }
35. }

从代码中可以看出，其分别处理了AVBPrint类型数组part的4个元素。由此我们也可以看出FFmpeg一条Log可以分成4个组成部分。在这里涉及到几个与AVBPrint相关的函数，由于篇幅的关系，不再分析它们的源代码，仅仅列出它们的定义：  
初始化AVBPrint的函数av\_bprint\_init()。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44243155) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44243155)

[在CODE上查看代码片](https://code.csdn.net/snippets/619180)

1. /\*\*
2. \* Init a print buffer.
3. \*
4. \* @param buf        buffer to init
5. \* @param size\_init  initial size (including the final 0)
6. \* @param size\_max   maximum size;
7. \*                   0 means do not write anything, just count the length;
8. \*                   1 is replaced by the maximum value for automatic storage;
9. \*                   any large value means that the internal buffer will be
10. \*                   reallocated as needed up to that limit; -1 is converted to
11. \*                   UINT\_MAX, the largest limit possible.
12. \*                   Check also AV\_BPRINT\_SIZE\_\* macros.
13. \*/
14. **void** av\_bprint\_init(AVBPrint \*buf, unsigned size\_init, unsigned size\_max);

向AVBPrint添加一个字符串的函数av\_bprintf()。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44243155) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44243155)

[在CODE上查看代码片](https://code.csdn.net/snippets/619180)

1. /\*\*
2. \* Append a formatted string to a print buffer.
3. \*/
4. **void** av\_bprintf(AVBPrint \*buf, **const** **char** \*fmt, ...) av\_printf\_format(2, 3);

向AVBPrint添加一个字符串的函数av\_vbprintf ()，注意与av\_bprintf()的不同在于其第3个参数不一样。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44243155) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44243155)

[在CODE上查看代码片](https://code.csdn.net/snippets/619180)

1. /\*\*
2. \* Append a formatted string to a print buffer.
3. \*/
4. **void** av\_vbprintf(AVBPrint \*buf, **const** **char** \*fmt, **va\_list** vl\_arg);

我们可以瞄一眼av\_bprintf()的定义，位于libavutil\bprint.c，如下所示。

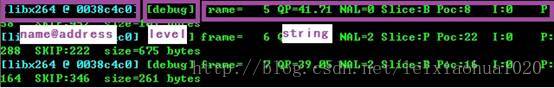
**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44243155) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44243155)

[在CODE上查看代码片](https://code.csdn.net/snippets/619180)

1. **void** av\_bprintf(AVBPrint \*buf, **const** **char** \*fmt, ...)
2. {
3. unsigned room;
4. **char** \*dst;
5. **va\_list** vl;
6. **int** extra\_len;
8. **while** (1) {
9. room = av\_bprint\_room(buf);
10. dst = room ? buf->str + buf->len : NULL;
11. va\_start(vl, fmt);
12. extra\_len = vsnprintf(dst, room, fmt, vl);
13. va\_end(vl);
14. **if** (extra\_len <= 0)
15. **return**;
16. **if** (extra\_len < room)
17. **break**;
18. **if** (av\_bprint\_alloc(buf, extra\_len))
19. **break**;
20. }
21. av\_bprint\_grow(buf, extra\_len);
22. }

可以看出av\_bprintf()实际上调用了系统的vsnprintf()完成了相应的功能。

看完以上几个与AVBPrint相关函数之后，就可以来看一下format\_line()的代码了。例如，part[0]对应的是目标结构体的父结构体的名称（如果父结构体存在的话）；其打印格式形如“[%s @ %p]”，其中前面的“%s”对应父结构体的名称，“%p”对应其所在的地址。part[1]对应的是目标结构体的名称；其打印格式形如“[%s @ %p]”，其中前面的“%s”对应本结构体的名称，“%p”对应其所在的地址。part[2]用于输出Log的级别，这个字符串只有在flag中设置AV\_LOG\_PRINT\_LEVEL的时候才能打印。part[3]则是打印原本传送进来的文本。将format\_line()函数处理后得到的4个字符串连接其来，就可以的到一条完整的Log信息。下面图显示了flag设置AV\_LOG\_PRINT\_LEVEL后的打印出来的Log的格式。



默认情况下不设置flag打印出来的格式如下所示。

http://img.blog.csdn.net/20150313172203357?watermark/2/text/aHR0cDovL2Jsb2cuY3Nkbi5uZXQvbGVpeGlhb2h1YTEwMjA=/font/5a6L5L2T/fontsize/400/fill/I0JBQkFCMA==/dissolve/70/gravity/SouthEast

### colored\_fputs()

colored\_fputs()函数用于将输出的文本“上色”并且输出。在这里有一点需要注意：Windows和[**Linux**](http://lib.csdn.net/base/linux)下控制台程序上色的方法是不一样的。Windows下是通过SetConsoleTextAttribute()方法给控制台中的文本上色；Linux下则是通过添加一些ANSI控制码完成上色。

#### Linux下控制台文字上色的方法

Linux下控制台颜色是通过添加专用数字来选择的。这些数字夹在 "\e["和 "m"之间。如果指定一个以上的数字，则用分号将它们分开。

举几个例子：

（1）第一个数字(31)为前景颜色(红色)；第二个数字为(42)背景颜色(绿色)

**[plain]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44243155) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44243155)

[在CODE上查看代码片](https://code.csdn.net/snippets/619180)

1. echo -e "\e[31;42m"

（2）使用"\e[0m"序列将颜色重新设置为正常值

**[plain]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44243155) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44243155)

[在CODE上查看代码片](https://code.csdn.net/snippets/619180)

1. echo -e "\e[0m" 或 echo -e "\033[0m"

（3）颜色对应关系如下所示：  
\e[30m -- \e[37m    设置前景色(字体颜色)  
echo -e "\e[30m"    灰色  
echo -e "\e[31m"    红色      
echo -e "\e[32m"    绿色  
echo -e "\e[33m"    黄色  
echo -e "\e[34m"    蓝色  
echo -e "\e[35m"    紫色  
echo -e "\e[36m"    淡蓝色  
echo -e "\e[37m"    白色       
  
\e[40m -- \e[47m    设置背景色  
echo -e "\e[40m"    灰色  
echo -e "\e[41m"    红色  
echo -e "\e[42m"    绿色  
echo -e "\e[43m"    黄色  
echo -e "\e[44m"    蓝色  
echo -e "\e[45m"    紫色  
echo -e "\e[46m"    淡蓝色  
echo -e "\e[47m"    白色  
  
具体到编程中，printf() 颜色设置示例代码如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44243155) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44243155)

[在CODE上查看代码片](https://code.csdn.net/snippets/619180)

1. **int** main()
2. {
3. printf("\e[31m Hello World. \e[0m \n"); // 红色字体
4. **return** 0;
5. }

#### Windows下控制台文字上色的方法

Windows下控制台颜色是通过SetConsoleTextAttribute()函数完成的。SetConsoleTextAttribute()函数的原型如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44243155) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44243155)

[在CODE上查看代码片](https://code.csdn.net/snippets/619180)

1. **BOOL** SetConsoleTextAttribute(**HANDLE** hConsoleOutput, **WORD** wAttributes);

其中2个参数的含义如下所示：

hConsoleOutput：指向控制台的句柄。  
wAttributes：文本属性。

hConsoleOutput可以选择以下3种句柄：

STD\_INPUT\_HANDLE： 标准输入的句柄  
STD\_OUTPUT\_HANDLE： 标准输出的句柄  
STD\_ERROR\_HANDLE： 标准错误的句柄

wAttributes可以控制前景色和背景色：

FOREGROUND\_BLUE： 字体颜色：蓝  
FOREGROUND\_GREEN： 字体颜色：绿  
FOREGROUND\_RED： 字体颜色：红  
FOREGROUND\_INTENSITY： 前景色高亮显示  
BACKGROUND\_BLUE： 背景颜色：蓝  
BACKGROUND\_GREEN： 背景颜色：绿  
BACKGROUND\_RED： 背景颜色：红  
BACKGROUND\_INTENSITY 背景色高亮显示

控制台文本上色demo代码如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44243155) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44243155)

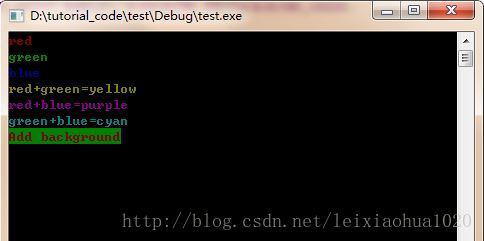
[在CODE上查看代码片](https://code.csdn.net/snippets/619180)

1. /\*\*
2. \* 雷霄骅 Lei Xiaohua
3. \* leixiaohua1020@126.com
4. \* http://blog.csdn.net/leixiaohua1020
5. \*/
6. #include <stdio.h>
7. #include <windows.h>

10. **int** main()
11. {
12. SetConsoleTextAttribute(GetStdHandle(STD\_OUTPUT\_HANDLE), FOREGROUND\_RED);
13. printf("red\n");
14. SetConsoleTextAttribute(GetStdHandle(STD\_OUTPUT\_HANDLE), FOREGROUND\_GREEN);
15. printf("green\n");
16. SetConsoleTextAttribute(GetStdHandle(STD\_OUTPUT\_HANDLE), FOREGROUND\_BLUE);
17. printf("blue\n");
18. SetConsoleTextAttribute(GetStdHandle(STD\_OUTPUT\_HANDLE), FOREGROUND\_RED|FOREGROUND\_GREEN);
19. printf("red+green=yellow\n");
20. SetConsoleTextAttribute(GetStdHandle(STD\_OUTPUT\_HANDLE), FOREGROUND\_RED|FOREGROUND\_BLUE);
21. printf("red+blue=purple\n");
22. SetConsoleTextAttribute(GetStdHandle(STD\_OUTPUT\_HANDLE), FOREGROUND\_GREEN|FOREGROUND\_BLUE);
23. printf("green+blue=cyan\n");
24. SetConsoleTextAttribute(GetStdHandle(STD\_OUTPUT\_HANDLE), FOREGROUND\_RED|BACKGROUND\_GREEN);
25. printf("Add background\n");

28. **return** 0;
29. }

程序的运行结果如下图所示。



### colored\_fputs()源代码

下面看一下colored\_fputs()函数的源代码。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44243155) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44243155)

[在CODE上查看代码片](https://code.csdn.net/snippets/619180)

1. **static** **void** colored\_fputs(**int** level, **int** tint, **const** **char** \*str)
2. {
3. **int** local\_use\_color;
4. **if** (!\*str)
5. **return**;
7. **if** (use\_color < 0)
8. check\_color\_terminal();
10. **if** (level == AV\_LOG\_INFO/8) local\_use\_color = 0;
11. **else**                        local\_use\_color = use\_color;
13. #if defined(\_WIN32) && !defined(\_\_MINGW32CE\_\_) && HAVE\_SETCONSOLETEXTATTRIBUTE
14. **if** (local\_use\_color)
15. SetConsoleTextAttribute(con, background | color[level]);
16. fputs(str, stderr);
17. **if** (local\_use\_color)
18. SetConsoleTextAttribute(con, attr\_orig);
19. #else
20. **if** (local\_use\_color == 1) {
21. fprintf(stderr,
22. "\033[%d;3%dm%s\033[0m",
23. (color[level] >> 4) & 15,
24. color[level] & 15,
25. str);
26. } **else** **if** (tint && use\_color == 256) {
27. fprintf(stderr,
28. "\033[48;5;%dm\033[38;5;%dm%s\033[0m",
29. (color[level] >> 16) & 0xff,
30. tint,
31. str);
32. } **else** **if** (local\_use\_color == 256) {
33. fprintf(stderr,
34. "\033[48;5;%dm\033[38;5;%dm%s\033[0m",
35. (color[level] >> 16) & 0xff,
36. (color[level] >> 8) & 0xff,
37. str);
38. } **else**
39. fputs(str, stderr);
40. #endif
42. }

从colored\_fputs()的源代码中可以看出如下流程：  
首先判定根据宏定义系统的类型，如果系统类型是Windows，那么就调用SetConsoleTextAttribute()方法设定控制台文本的颜色，然后调用fputs()将Log记录输出到stderr（注意不是stdout）；如果系统类型是Linux，则通过添加特定字符串的方式设定控制台文本的颜色，然后将Log记录输出到stderr。  
至此FFmpeg的日志输出系统的源代码就分析完毕了。

# [FFmpeg源代码简单分析：结构体成员管理系统-AVClass](http://blog.csdn.net/leixiaohua1020/article/details/44268323)

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***](http://lib.csdn.net/base/javase)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。

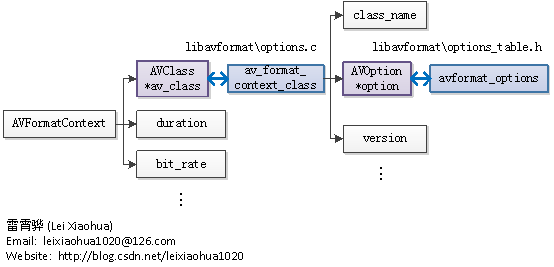
int a=Integer.parseInt("123");

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。



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。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44268323) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44268323)

[在CODE上查看代码片](https://code.csdn.net/snippets/620110)

1. /\*\*
2. \* AVOption
3. \*/
4. **typedef** **struct** AVOption {
5. **const** **char** \*name;
7. /\*\*
8. \* short English help text
9. \* @todo What about other languages?
10. \*/
11. **const** **char** \*help;
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;
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
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 inteded 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
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]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44268323) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44268323)

[在CODE上查看代码片](https://code.csdn.net/snippets/620110)

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]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44268323) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44268323)

[在CODE上查看代码片](https://code.csdn.net/snippets/620110)

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;
8. /\*\*
9. \* The input container format.
10. \*
11. \* Demuxing only, set by avformat\_open\_input().
12. \*/
13. **struct** AVInputFormat \*iformat;
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]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44268323) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44268323)

[在CODE上查看代码片](https://code.csdn.net/snippets/620110)

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;
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。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44268323) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44268323)

[在CODE上查看代码片](https://code.csdn.net/snippets/620110)

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;
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);
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;
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. \*/
32. **int** version;
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;
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;
49. /\*\*
50. \* Return next AVOptions-enabled child or NULL
51. \*/
52. **void**\* (\*child\_next)(**void** \*obj, **void** \*prev);
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);
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;
71. /\*\*
72. \* Callback to return the category.
73. \* available since version (51 << 16 | 59 << 8 | 100)
74. \*/
75. AVClassCategory (\*get\_category)(**void**\* ctx);
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]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44268323) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44268323)

[在CODE上查看代码片](https://code.csdn.net/snippets/620110)

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libavformat\options.c中，是一个名称为av\_format\_context\_class的静态结构体。如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44268323) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44268323)

[在CODE上查看代码片](https://code.csdn.net/snippets/620110)

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]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44268323) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44268323)

[在CODE上查看代码片](https://code.csdn.net/snippets/620110)

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]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44268323) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44268323)

[在CODE上查看代码片](https://code.csdn.net/snippets/620110)

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. \*/
9. #ifndef AVFORMAT\_OPTIONS\_TABLE\_H
10. #define AVFORMAT\_OPTIONS\_TABLE\_H
12. #include <limits.h>
14. #include "libavutil/opt.h"
15. #include "avformat.h"
16. #include "internal.h"
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
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 AVParsers, 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, D, "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 YUYV422 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\_CRCCHECK }, INT\_MIN, INT\_MAX, D, "err\_detect"},
59. {"err\_detect", "set error detection flags", OFFSET(error\_recognition), AV\_OPT\_TYPE\_FLAGS, {.i64 = AV\_EF\_CRCCHECK }, INT\_MIN, INT\_MAX, D, "err\_detect"},
60. {"crccheck", "verify embedded CRCs", 0, AV\_OPT\_TYPE\_CONST, {.i64 = AV\_EF\_CRCCHECK }, 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 compliancies 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\_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 = 10000000 }, 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. };
89. #undef E
90. #undef D
91. #undef DEFAULT
92. #undef OFFSET
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。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44268323) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44268323)

[在CODE上查看代码片](https://code.csdn.net/snippets/620110)

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。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44268323) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44268323)

[在CODE上查看代码片](https://code.csdn.net/snippets/620110)

1. **static** **const** **char**\* context\_to\_name(**void**\* ptr) {
2. AVCodecContext \*avc= ptr;
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AVOption。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]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44268323) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44268323)

[在CODE上查看代码片](https://code.csdn.net/snippets/620110)

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. \*/
9. #ifndef AVCODEC\_OPTIONS\_TABLE\_H
10. #define AVCODEC\_OPTIONS\_TABLE\_H
12. #include <float.h>
13. #include <limits.h>
14. #include <stdint.h>
16. #include "libavutil/opt.h"
17. #include "avcodec.h"
18. #include "version.h"
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
29. #define AV\_CODEC\_DEFAULT\_BITRATE 200\*1000
31. **static** **const** AVOption avcodec\_options[] = {
32. {"b", "set bitrate (in bits/s)", OFFSET(bit\_rate), 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, V | D, "flags" },
40. {"mv4", "use four motion vectors per macroblock (MPEG-4)", 0, AV\_OPT\_TYPE\_CONST, {.i64 = CODEC\_FLAG\_4MV }, INT\_MIN, INT\_MAX, V|E, "flags"},
41. {"qpel", "use 1/4-pel motion compensation", 0, AV\_OPT\_TYPE\_CONST, {.i64 = CODEC\_FLAG\_QPEL }, INT\_MIN, INT\_MAX, V|E, "flags"},
42. {"loop", "use loop filter", 0, AV\_OPT\_TYPE\_CONST, {.i64 = CODEC\_FLAG\_LOOP\_FILTER }, INT\_MIN, INT\_MAX, V|E, "flags"},
43. {"qscale", "use fixed qscale", 0, AV\_OPT\_TYPE\_CONST, {.i64 = CODEC\_FLAG\_QSCALE }, INT\_MIN, INT\_MAX, 0, "flags"},
44. #if FF\_API\_GMC
45. {"gmc", "use gmc", 0, AV\_OPT\_TYPE\_CONST, {.i64 = CODEC\_FLAG\_GMC }, INT\_MIN, INT\_MAX, V|E, "flags"},
46. #endif
47. #if FF\_API\_MV0
48. {"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"},
49. #endif
50. #if FF\_API\_INPUT\_PRESERVED
51. {"input\_preserved", NULL, 0, AV\_OPT\_TYPE\_CONST, {.i64 = CODEC\_FLAG\_INPUT\_PRESERVED }, INT\_MIN, INT\_MAX, 0, "flags"},
52. #endif
53. {"pass1", "use internal 2-pass ratecontrol in first  pass mode", 0, AV\_OPT\_TYPE\_CONST, {.i64 = CODEC\_FLAG\_PASS1 }, INT\_MIN, INT\_MAX, 0, "flags"},
54. {"pass2", "use internal 2-pass ratecontrol in second pass mode", 0, AV\_OPT\_TYPE\_CONST, {.i64 = CODEC\_FLAG\_PASS2 }, INT\_MIN, INT\_MAX, 0, "flags"},
55. {"gray", "only decode/encode grayscale", 0, AV\_OPT\_TYPE\_CONST, {.i64 = CODEC\_FLAG\_GRAY }, INT\_MIN, INT\_MAX, V|E|D, "flags"},
56. #if FF\_API\_EMU\_EDGE
57. {"emu\_edge", "do not draw edges", 0, AV\_OPT\_TYPE\_CONST, {.i64 = CODEC\_FLAG\_EMU\_EDGE }, INT\_MIN, INT\_MAX, 0, "flags"},
58. #endif
59. {"psnr", "error[?] variables will be set during encoding", 0, AV\_OPT\_TYPE\_CONST, {.i64 = CODEC\_FLAG\_PSNR }, INT\_MIN, INT\_MAX, V|E, "flags"},
60. {"truncated", NULL, 0, AV\_OPT\_TYPE\_CONST, {.i64 = CODEC\_FLAG\_TRUNCATED }, INT\_MIN, INT\_MAX, 0, "flags"},
61. #if FF\_API\_NORMALIZE\_AQP
62. {"naq", "normalize adaptive quantization", 0, AV\_OPT\_TYPE\_CONST, {.i64 = CODEC\_FLAG\_NORMALIZE\_AQP }, INT\_MIN, INT\_MAX, V|E, "flags"},
63. #endif
64. {"ildct", "use interlaced DCT", 0, AV\_OPT\_TYPE\_CONST, {.i64 = CODEC\_FLAG\_INTERLACED\_DCT }, INT\_MIN, INT\_MAX, V|E, "flags"},
65. {"low\_delay", "force low delay", 0, AV\_OPT\_TYPE\_CONST, {.i64 = CODEC\_FLAG\_LOW\_DELAY }, INT\_MIN, INT\_MAX, V|D|E, "flags"},
66. {"global\_header", "place global headers in extradata instead of every keyframe", 0, AV\_OPT\_TYPE\_CONST, {.i64 = CODEC\_FLAG\_GLOBAL\_HEADER }, INT\_MIN, INT\_MAX, V|A|E, "flags"},
67. {"bitexact", "use only bitexact functions (except (I)DCT)", 0, AV\_OPT\_TYPE\_CONST, {.i64 = CODEC\_FLAG\_BITEXACT }, INT\_MIN, INT\_MAX, A|V|S|D|E, "flags"},
68. {"aic", "H.263 advanced intra coding / MPEG-4 AC prediction", 0, AV\_OPT\_TYPE\_CONST, {.i64 = CODEC\_FLAG\_AC\_PRED }, INT\_MIN, INT\_MAX, V|E, "flags"},
69. {"ilme", "interlaced motion estimation", 0, AV\_OPT\_TYPE\_CONST, {.i64 = CODEC\_FLAG\_INTERLACED\_ME }, INT\_MIN, INT\_MAX, V|E, "flags"},
70. {"cgop", "closed GOP", 0, AV\_OPT\_TYPE\_CONST, {.i64 = CODEC\_FLAG\_CLOSED\_GOP }, INT\_MIN, INT\_MAX, V|E, "flags"},
71. {"output\_corrupt", "Output even potentially corrupted frames", 0, AV\_OPT\_TYPE\_CONST, {.i64 = CODEC\_FLAG\_OUTPUT\_CORRUPT }, INT\_MIN, INT\_MAX, V|D, "flags"},
72. {"fast", "allow non-spec-compliant speedup tricks", 0, AV\_OPT\_TYPE\_CONST, {.i64 = CODEC\_FLAG2\_FAST }, INT\_MIN, INT\_MAX, V|E, "flags2"},
73. {"noout", "skip bitstream encoding", 0, AV\_OPT\_TYPE\_CONST, {.i64 = CODEC\_FLAG2\_NO\_OUTPUT }, INT\_MIN, INT\_MAX, V|E, "flags2"},
74. {"ignorecrop", "ignore cropping information from sps", 0, AV\_OPT\_TYPE\_CONST, {.i64 = CODEC\_FLAG2\_IGNORE\_CROP }, INT\_MIN, INT\_MAX, V|D, "flags2"},
75. {"local\_header", "place global headers at every keyframe instead of in extradata", 0, AV\_OPT\_TYPE\_CONST, {.i64 = CODEC\_FLAG2\_LOCAL\_HEADER }, 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|D, "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\_mvs", "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。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44268323) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44268323)

[在CODE上查看代码片](https://code.csdn.net/snippets/620110)

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。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44268323) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44268323)

[在CODE上查看代码片](https://code.csdn.net/snippets/620110)

1. **static** **const** AVOption frame\_options[]={
2. {"best\_effort\_timestamp", "", FOFFSET(best\_effort\_timestamp), AV\_OPT\_TYPE\_INT64, {.i64 = AV\_NOPTS\_VALUE }, INT64\_MIN, INT64\_MAX, 0},
3. {"pkt\_pos", "", FOFFSET(pkt\_pos), AV\_OPT\_TYPE\_INT64, {.i64 = -1 }, INT64\_MIN, INT64\_MAX, 0},
4. {"pkt\_size", "", FOFFSET(pkt\_size), AV\_OPT\_TYPE\_INT64, {.i64 = -1 }, INT64\_MIN, INT64\_MAX, 0},
5. {"sample\_aspect\_ratio", "", FOFFSET(sample\_aspect\_ratio), AV\_OPT\_TYPE\_RATIONAL, {.dbl = 0 }, 0, INT\_MAX, 0},
6. {"width", "", FOFFSET(width), AV\_OPT\_TYPE\_INT, {.i64 = 0 }, 0, INT\_MAX, 0},
7. {"height", "", FOFFSET(height), AV\_OPT\_TYPE\_INT, {.i64 = 0 }, 0, INT\_MAX, 0},
8. {"format", "", FOFFSET(format), AV\_OPT\_TYPE\_INT, {.i64 = -1 }, 0, INT\_MAX, 0},
9. {"channel\_layout", "", FOFFSET(channel\_layout), AV\_OPT\_TYPE\_INT64, {.i64 = 0 }, 0, INT64\_MAX, 0},
10. {"sample\_rate", "", FOFFSET(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]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44268323) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44268323)

[在CODE上查看代码片](https://code.csdn.net/snippets/620110)

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]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44268323) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44268323)

[在CODE上查看代码片](https://code.csdn.net/snippets/620110)

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]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44268323) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44268323)

[在CODE上查看代码片](https://code.csdn.net/snippets/620110)

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]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44268323) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44268323)

[在CODE上查看代码片](https://code.csdn.net/snippets/620110)

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]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44268323) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44268323)

[在CODE上查看代码片](https://code.csdn.net/snippets/620110)

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 = -1 }, -1, FLT\_MAX, VE },
12. { "qp",            "Constant quantization parameter rate control method",OFFSET(cqp),        AV\_OPT\_TYPE\_INT,    { .i64 = -1 }, -1, INT\_MAX, VE },
13. { "aq-mode",       "AQ method",                                       OFFSET(aq\_mode),       AV\_OPT\_TYPE\_INT,    { .i64 = -1 }, -1, INT\_MAX, VE, "aq\_mode"},
14. { "none",          NULL,                              0, AV\_OPT\_TYPE\_CONST, {.i64 = X264\_AQ\_NONE},         INT\_MIN, INT\_MAX, VE, "aq\_mode" },
15. { "variance",      "Variance AQ (complexity mask)",   0, AV\_OPT\_TYPE\_CONST, {.i64 = X264\_AQ\_VARIANCE},     INT\_MIN, INT\_MAX, VE, "aq\_mode" },
16. { "autovariance",  "Auto-variance AQ (experimental)", 0, AV\_OPT\_TYPE\_CONST, {.i64 = X264\_AQ\_AUTOVARIANCE}, INT\_MIN, INT\_MAX, VE, "aq\_mode" },
17. { "aq-strength",   "AQ strength. Reduces blocking and blurring in flat and textured areas.", OFFSET(aq\_strength), AV\_OPT\_TYPE\_FLOAT, {.dbl = -1}, -1, FLT\_MAX, VE},
18. { "psy",           "Use psychovisual optimizations.",                 OFFSET(psy),           AV\_OPT\_TYPE\_INT,    { .i64 = -1 }, -1, 1, VE },
19. { "psy-rd",        "Strength of psychovisual optimization, in <psy-rd>:<psy-trellis> format.", OFFSET(psy\_rd), AV\_OPT\_TYPE\_STRING,  {0 }, 0, 0, VE},
20. { "rc-lookahead",  "Number of frames to look ahead for frametype and ratecontrol", OFFSET(rc\_lookahead), AV\_OPT\_TYPE\_INT, { .i64 = -1 }, -1, INT\_MAX, VE },
21. { "weightb",       "Weighted prediction for B-frames.",               OFFSET(weightb),       AV\_OPT\_TYPE\_INT,    { .i64 = -1 }, -1, 1, VE },
22. { "weightp",       "Weighted prediction analysis method.",            OFFSET(weightp),       AV\_OPT\_TYPE\_INT,    { .i64 = -1 }, -1, INT\_MAX, VE, "weightp" },
23. { "none",          NULL, 0, AV\_OPT\_TYPE\_CONST, {.i64 = X264\_WEIGHTP\_NONE},   INT\_MIN, INT\_MAX, VE, "weightp" },
24. { "simple",        NULL, 0, AV\_OPT\_TYPE\_CONST, {.i64 = X264\_WEIGHTP\_SIMPLE}, INT\_MIN, INT\_MAX, VE, "weightp" },
25. { "smart",         NULL, 0, AV\_OPT\_TYPE\_CONST, {.i64 = X264\_WEIGHTP\_SMART},  INT\_MIN, INT\_MAX, VE, "weightp" },
26. { "ssim",          "Calculate and print SSIM stats.",                 OFFSET(ssim),          AV\_OPT\_TYPE\_INT,    { .i64 = -1 }, -1, 1, VE },
27. { "intra-refresh", "Use Periodic Intra Refresh instead of IDR frames.",OFFSET(intra\_refresh),AV\_OPT\_TYPE\_INT,    { .i64 = -1 }, -1, 1, VE },
28. { "bluray-compat", "Bluray compatibility workarounds.",               OFFSET(bluray\_compat) ,AV\_OPT\_TYPE\_INT,    { .i64 = -1 }, -1, 1, VE },
29. { "b-bias",        "Influences how often B-frames are used",          OFFSET(b\_bias),        AV\_OPT\_TYPE\_INT,    { .i64 = INT\_MIN}, INT\_MIN, INT\_MAX, VE },
30. { "b-pyramid",     "Keep some B-frames as references.",               OFFSET(b\_pyramid),     AV\_OPT\_TYPE\_INT,    { .i64 = -1 }, -1, INT\_MAX, VE, "b\_pyramid" },
31. { "none",          NULL,                                  0, AV\_OPT\_TYPE\_CONST, {.i64 = X264\_B\_PYRAMID\_NONE},   INT\_MIN, INT\_MAX, VE, "b\_pyramid" },
32. { "strict",        "Strictly hierarchical pyramid",       0, AV\_OPT\_TYPE\_CONST, {.i64 = X264\_B\_PYRAMID\_STRICT}, INT\_MIN, INT\_MAX, VE, "b\_pyramid" },
33. { "normal",        "Non-strict (not Blu-ray compatible)", 0, AV\_OPT\_TYPE\_CONST, {.i64 = X264\_B\_PYRAMID\_NORMAL}, INT\_MIN, INT\_MAX, VE, "b\_pyramid" },
34. { "mixed-refs",    "One reference per partition, as opposed to one reference per macroblock", OFFSET(mixed\_refs), AV\_OPT\_TYPE\_INT, { .i64 = -1}, -1, 1, VE },
35. { "8x8dct",        "High profile 8x8 transform.",                     OFFSET(dct8x8),        AV\_OPT\_TYPE\_INT,    { .i64 = -1 }, -1, 1, VE},
36. { "fast-pskip",    NULL,                                              OFFSET(fast\_pskip),    AV\_OPT\_TYPE\_INT,    { .i64 = -1 }, -1, 1, VE},
37. { "aud",           "Use access unit delimiters.",                     OFFSET(aud),           AV\_OPT\_TYPE\_INT,    { .i64 = -1 }, -1, 1, VE},
38. { "mbtree",        "Use macroblock tree ratecontrol.",                OFFSET(mbtree),        AV\_OPT\_TYPE\_INT,    { .i64 = -1 }, -1, 1, VE},
39. { "deblock",       "Loop filter parameters, in <alpha:beta> form.",   OFFSET(deblock),       AV\_OPT\_TYPE\_STRING, { 0 },  0, 0, VE},
40. { "cplxblur",      "Reduce fluctuations in QP (before curve compression)", OFFSET(cplxblur), AV\_OPT\_TYPE\_FLOAT,  {.dbl = -1 }, -1, FLT\_MAX, VE},
41. { "partitions",    "A comma-separated list of partitions to consider. "
42. "Possible values: p8x8, p4x4, b8x8, i8x8, i4x4, none, all", OFFSET(partitions), AV\_OPT\_TYPE\_STRING, { 0 }, 0, 0, VE},
43. { "direct-pred",   "Direct MV prediction mode",                       OFFSET(direct\_pred),   AV\_OPT\_TYPE\_INT,    { .i64 = -1 }, -1, INT\_MAX, VE, "direct-pred" },
44. { "none",          NULL,      0,    AV\_OPT\_TYPE\_CONST, { .i64 = X264\_DIRECT\_PRED\_NONE },     0, 0, VE, "direct-pred" },
45. { "spatial",       NULL,      0,    AV\_OPT\_TYPE\_CONST, { .i64 = X264\_DIRECT\_PRED\_SPATIAL },  0, 0, VE, "direct-pred" },
46. { "temporal",      NULL,      0,    AV\_OPT\_TYPE\_CONST, { .i64 = X264\_DIRECT\_PRED\_TEMPORAL }, 0, 0, VE, "direct-pred" },
47. { "auto",          NULL,      0,    AV\_OPT\_TYPE\_CONST, { .i64 = X264\_DIRECT\_PRED\_AUTO },     0, 0, VE, "direct-pred" },
48. { "slice-max-size","Limit the size of each slice in bytes",           OFFSET(slice\_max\_size),AV\_OPT\_TYPE\_INT,    { .i64 = -1 }, -1, INT\_MAX, VE },
49. { "stats",         "Filename for 2 pass stats",                       OFFSET(stats),         AV\_OPT\_TYPE\_STRING, { 0 },  0,       0, VE },
50. { "nal-hrd",       "Signal HRD information (requires vbv-bufsize; "
51. "cbr not allowed in .mp4)",                        OFFSET(nal\_hrd),       AV\_OPT\_TYPE\_INT,    { .i64 = -1 }, -1, INT\_MAX, VE, "nal-hrd" },
52. { "none",          NULL, 0, AV\_OPT\_TYPE\_CONST, {.i64 = X264\_NAL\_HRD\_NONE}, INT\_MIN, INT\_MAX, VE, "nal-hrd" },
53. { "vbr",           NULL, 0, AV\_OPT\_TYPE\_CONST, {.i64 = X264\_NAL\_HRD\_VBR},  INT\_MIN, INT\_MAX, VE, "nal-hrd" },
54. { "cbr",           NULL, 0, AV\_OPT\_TYPE\_CONST, {.i64 = X264\_NAL\_HRD\_CBR},  INT\_MIN, INT\_MAX, VE, "nal-hrd" },
55. { "avcintra-class","AVC-Intra class 50/100/200",                      OFFSET(avcintra\_class),AV\_OPT\_TYPE\_INT,     { .i64 = -1 }, -1, 200   , VE},
56. { "x264-params",  "Override the x264 configuration using a :-separated list of key=value parameters", OFFSET(x264\_params), AV\_OPT\_TYPE\_STRING, { 0 }, 0, 0, VE },
57. { NULL },
58. };

### Libx265

Libx265的AVClass定义如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44268323) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44268323)

[在CODE上查看代码片](https://code.csdn.net/snippets/620110)

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]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44268323) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44268323)

[在CODE上查看代码片](https://code.csdn.net/snippets/620110)

1. **static** **const** AVOption options[] = {
2. { "preset",      "set the x265 preset",                                                         OFFSET(preset),    AV\_OPT\_TYPE\_STRING, { 0 }, 0, 0, VE },
3. { "tune",        "set the x265 tune parameter",                                                 OFFSET(tune),      AV\_OPT\_TYPE\_STRING, { 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]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44268323) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44268323)

[在CODE上查看代码片](https://code.csdn.net/snippets/620110)

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;
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. };
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]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44268323) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44268323)

[在CODE上查看代码片](https://code.csdn.net/snippets/620110)

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]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44268323) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44268323)

[在CODE上查看代码片](https://code.csdn.net/snippets/620110)

1. **const** AVClass \*avcodec\_get\_class(**void**)
2. {
3. **return** &av\_codec\_context\_class;
4. }

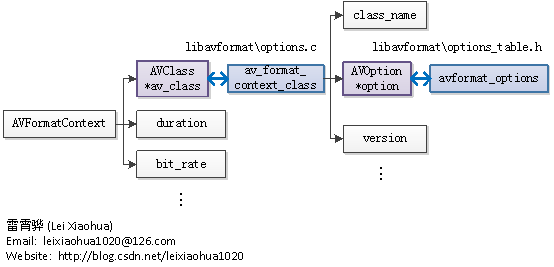
至此FFmpeg的AVClass就基本上分析完毕了。下篇文章具体分析AVOption。

# [FFmpeg源代码简单分析：结构体成员管理系统-AVOption](http://blog.csdn.net/leixiaohua1020/article/details/44279329)

本文继续上篇文章《[FFmpeg源代码分析：结构体成员管理系统-AVClass](http://blog.csdn.net/leixiaohua1020/article/details/44268323)》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。



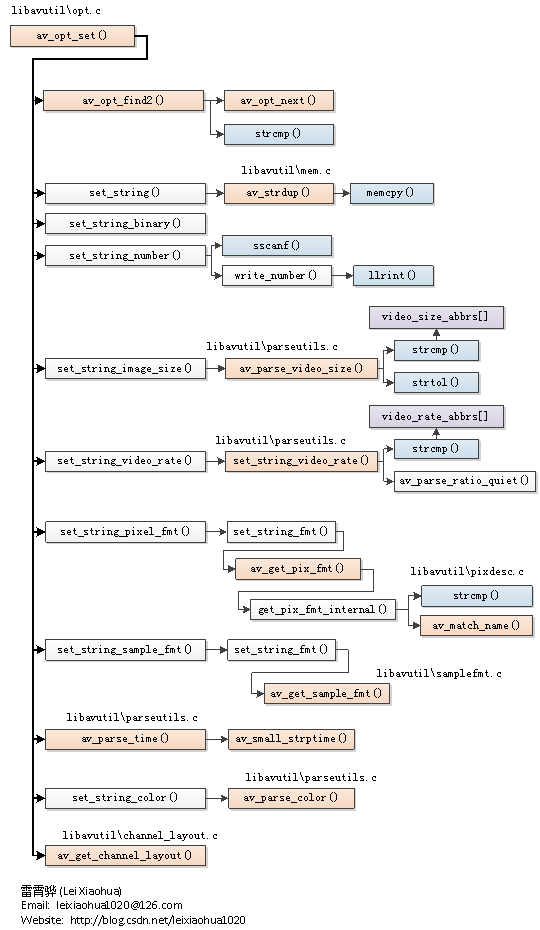
有关上篇文章的内容，这里不再重复。总体来说，上篇文章主要概括了AVClass，AVOption和目标结构体之间的从属关系，但是并没有分析有关AVOption的源代码。本文补充上一篇文章的内容，分析有关AVOption的源代码。

## AVOption有关的API

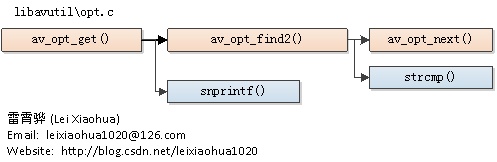
AVOption常用的API可以分成两类：用于设置参数的API和用于读取参数的API。其中最有代表性的用于设置参数的API就是av\_opt\_set()；而最有代表性的用于读取参数的API就是av\_opt\_get()。除了记录以上两个函数之外，本文再记录一个在FFmpeg的结构体初始化代码中最常用的用于设置默认值的函数av\_opt\_set\_defaults()。

## 函数调用关系图

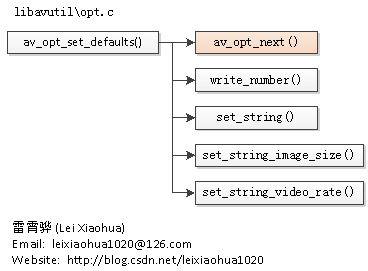
av\_opt\_set()的函数调用关系图如下所示。



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]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44279329) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44279329)

[在CODE上查看代码片](https://code.csdn.net/snippets/620400)

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 AVERROR code in case of
22. \* error:
23. \* AVERROR\_OPTION\_NOT\_FOUND if no matching option exists
24. \* AVERROR(ERANGE) if the value is out of range
25. \* AVERROR(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。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44279329) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44279329)

[在CODE上查看代码片](https://code.csdn.net/snippets/620400)

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);
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. }
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。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44279329) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44279329)

[在CODE上查看代码片](https://code.csdn.net/snippets/620400)

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。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44279329) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44279329)

[在CODE上查看代码片](https://code.csdn.net/snippets/620400)

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。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44279329) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44279329)

[在CODE上查看代码片](https://code.csdn.net/snippets/620400)

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;
7. **if**(!obj)
8. **return** NULL;
10. c= \*(AVClass\*\*)obj;
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]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44279329) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44279329)

[在CODE上查看代码片](https://code.csdn.net/snippets/620400)

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]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44279329) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44279329)

[在CODE上查看代码片](https://code.csdn.net/snippets/620400)

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]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44279329) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44279329)

[在CODE上查看代码片](https://code.csdn.net/snippets/620400)

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]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44279329) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44279329)

[在CODE上查看代码片](https://code.csdn.net/snippets/620400)

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]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44279329) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44279329)

[在CODE上查看代码片](https://code.csdn.net/snippets/620400)

1. **static** **int** set\_string\_image\_size(**void** \*obj, **const** AVOption \*o, **const** **char** \*val, **int** \*dst)
2. {
3. **int** ret;
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]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44279329) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44279329)

[在CODE上查看代码片](https://code.csdn.net/snippets/620400)

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]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44279329) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44279329)

[在CODE上查看代码片](https://code.csdn.net/snippets/620400)

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. //strtol()：字符串转换成整型，遇到非数字则停止
19. width = strtol(str, (**void**\*)&p, 10);
20. **if** (\*p)
21. p++;
22. height = strtol(p, (**void**\*)&p, 10);
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]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44279329) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44279329)

[在CODE上查看代码片](https://code.csdn.net/snippets/620400)

1. **typedef** **struct** {
2. **const** **char** \*abbr;
3. **int** width, height;
4. } VideoSizeAbbr;
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。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44279329) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44279329)

[在CODE上查看代码片](https://code.csdn.net/snippets/620400)

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。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44279329) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44279329)

[在CODE上查看代码片](https://code.csdn.net/snippets/620400)

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. }
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
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]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44279329) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44279329)

[在CODE上查看代码片](https://code.csdn.net/snippets/620400)

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]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44279329) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44279329)

[在CODE上查看代码片](https://code.csdn.net/snippets/620400)

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。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44279329) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44279329)

[在CODE上查看代码片](https://code.csdn.net/snippets/620400)

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;
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), "%"PRId64, \*(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. **case** AV\_OPT\_TYPE\_CONST:     ret = snprintf(buf, **sizeof**(buf), "%f" ,     o->default\_val.dbl);**break**;
25. **case** AV\_OPT\_TYPE\_STRING:
26. **if** (\*(uint8\_t\*\*)dst)
27. \*out\_val = av\_strdup(\*(uint8\_t\*\*)dst);
28. **else**
29. \*out\_val = av\_strdup("");
30. **return** 0;
31. **case** AV\_OPT\_TYPE\_BINARY:
32. len = \*(**int**\*)(((uint8\_t \*)dst) + **sizeof**(uint8\_t \*));
33. **if** ((uint64\_t)len\*2 + 1 > INT\_MAX)
34. **return** AVERROR(EINVAL);
35. **if** (!(\*out\_val = av\_malloc(len\*2 + 1)))
36. **return** AVERROR(ENOMEM);
37. **if** (!len) {
38. \*out\_val[0] = '\0';
39. **return** 0;
40. }
41. bin = \*(uint8\_t\*\*)dst;
42. **for** (i = 0; i < len; i++)
43. snprintf(\*out\_val + i\*2, 3, "%02X", bin[i]);
44. **return** 0;
45. **case** AV\_OPT\_TYPE\_IMAGE\_SIZE:
46. //分辨率
47. ret = snprintf(buf, **sizeof**(buf), "%dx%d", ((**int** \*)dst)[0], ((**int** \*)dst)[1]);
48. **break**;
49. **case** AV\_OPT\_TYPE\_PIXEL\_FMT:
50. //像素格式
51. ret = snprintf(buf, **sizeof**(buf), "%s", (**char** \*)av\_x\_if\_null(av\_get\_pix\_fmt\_name(\*(**enum** AVPixelFormat \*)dst), "none"));
52. **break**;
53. **case** AV\_OPT\_TYPE\_SAMPLE\_FMT:
54. //采样格式
55. ret = snprintf(buf, **sizeof**(buf), "%s", (**char** \*)av\_x\_if\_null(av\_get\_sample\_fmt\_name(\*(**enum** AVSampleFormat \*)dst), "none"));
56. **break**;
57. **case** AV\_OPT\_TYPE\_DURATION:
58. //时长
59. i64 = \*(int64\_t \*)dst;
60. ret = snprintf(buf, **sizeof**(buf), "%"PRIi64":%02d:%02d.%06d",
61. i64 / 3600000000, (**int**)((i64 / 60000000) % 60),
62. (**int**)((i64 / 1000000) % 60), (**int**)(i64 % 1000000));
63. **break**;
64. **case** AV\_OPT\_TYPE\_COLOR:
65. ret = snprintf(buf, **sizeof**(buf), "0x%02x%02x%02x%02x",
66. (**int**)((uint8\_t \*)dst)[0], (**int**)((uint8\_t \*)dst)[1],
67. (**int**)((uint8\_t \*)dst)[2], (**int**)((uint8\_t \*)dst)[3]);
68. **break**;
69. **case** AV\_OPT\_TYPE\_CHANNEL\_LAYOUT:
70. i64 = \*(int64\_t \*)dst;
71. ret = snprintf(buf, **sizeof**(buf), "0x%"PRIx64, i64);
72. **break**;
73. **default**:
74. **return** AVERROR(EINVAL);
75. }
77. **if** (ret >= **sizeof**(buf))
78. **return** AVERROR(EINVAL);
79. //拷贝
80. \*out\_val = av\_strdup(buf);
81. **return** 0;
82. }

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。

# [FFmpeg源代码简单分析：libswscale的sws\_getContext()](http://blog.csdn.net/leixiaohua1020/article/details/44305697)

打算写两篇文章记录FFmpeg中的图像处理（缩放，YUV/RGB格式转换）类库libswsscale的源代码。libswscale是一个主要用于处理图片像素数据的类库。可以完成图片像素格式的转换，图片的拉伸等工作。有关libswscale的使用可以参考文章：

《[最简单的基于FFmpeg的libswscale的示例（YUV转RGB）](http://blog.csdn.net/leixiaohua1020/article/details/42134965)》

libswscale常用的函数数量很少，一般情况下就3个：

sws\_getContext()：初始化一个SwsContext。

sws\_scale()：处理图像数据。

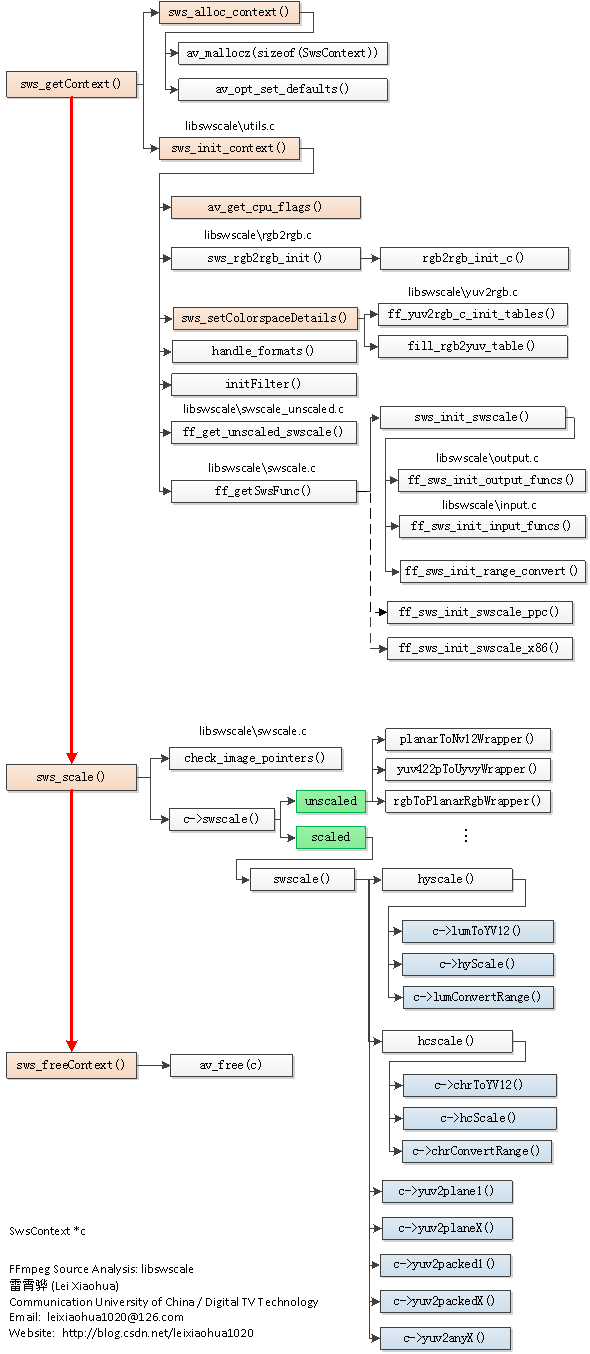
sws\_freeContext()：释放一个SwsContext。

其中sws\_getContext()也可以用sws\_getCachedContext()取代。

尽管libswscale从表面上看常用函数的个数不多，它的内部却有一个大大的“世界”。做为一个几乎“万能”的图片像素数据处理类库，它的内部包含了大量的代码。因此计划写两篇文章分析它的源代码。本文首先分析它的初始化函数sws\_getContext()，而下一篇文章则分析它的数据处理函数sws\_scale()。

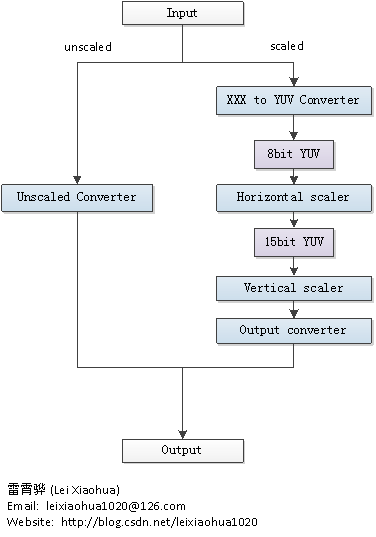
## 函数调用结构图

分析得到的libswscale的函数调用关系如下图所示。



## Libswscale处理数据流程

Libswscale处理像素数据的流程可以概括为下图。



从图中可以看出，libswscale处理数据有两条最主要的方式：unscaled和scaled。unscaled用于处理不需要拉伸的像素数据（属于比较特殊的情况），scaled用于处理需要拉伸的像素数据。Unscaled只需要对图像像素格式进行转换；而Scaled则除了对像素格式进行转换之外，还需要对图像进行缩放。Scaled方式可以分成以下几个步骤：

* XXX to YUV Converter：首相将数据像素数据转换为8bitYUV格式；
* Horizontal scaler：水平拉伸图像，并且转换为15bitYUV；
* Vertical scaler：垂直拉伸图像；
* Output converter：转换为输出像素格式。

## SwsContext

SwsContext是使用libswscale时候一个贯穿始终的结构体。但是我们在使用FFmpeg的类库进行开发的时候，是无法看到它的内部结构的。在libswscale\swscale.h中只能看到一行定义：

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44305697) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44305697)

[在CODE上查看代码片](https://code.csdn.net/snippets/621402)

1. **struct** SwsContext;

一般人看到这个只有一行定义的结构体，会猜测它的内部一定十分简单。但是假使我们看一下FFmpeg的源代码，会发现这个猜测是完全错误的——SwsContext的定义是十分复杂的。它的定义位于libswscale\swscale\_internal.h中，如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44305697) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44305697)

[在CODE上查看代码片](https://code.csdn.net/snippets/621402)

1. /\* This struct should be aligned on at least a 32-byte boundary. \*/
2. **typedef** **struct** SwsContext {
3. /\*\*
4. \* info on struct for av\_log
5. \*/
6. **const** AVClass \*av\_class;
8. /\*\*
9. \* Note that src, dst, srcStride, dstStride will be copied in the
10. \* sws\_scale() wrapper so they can be freely modified here.
11. \*/
12. SwsFunc swscale;
13. **int** srcW;                     ///< Width  of source      luma/alpha planes.
14. **int** srcH;                     ///< Height of source      luma/alpha planes.
15. **int** dstH;                     ///< Height of destination luma/alpha planes.
16. **int** chrSrcW;                  ///< Width  of source      chroma     planes.
17. **int** chrSrcH;                  ///< Height of source      chroma     planes.
18. **int** chrDstW;                  ///< Width  of destination chroma     planes.
19. **int** chrDstH;                  ///< Height of destination chroma     planes.
20. **int** lumXInc, chrXInc;
21. **int** lumYInc, chrYInc;
22. **enum** AVPixelFormat dstFormat; ///< Destination pixel format.
23. **enum** AVPixelFormat srcFormat; ///< Source      pixel format.
24. **int** dstFormatBpp;             ///< Number of bits per pixel of the destination pixel format.
25. **int** srcFormatBpp;             ///< Number of bits per pixel of the source      pixel format.
26. **int** dstBpc, srcBpc;
27. **int** chrSrcHSubSample;         ///< Binary logarithm of horizontal subsampling factor between luma/alpha and chroma planes in source      image.
28. **int** chrSrcVSubSample;         ///< Binary logarithm of vertical   subsampling factor between luma/alpha and chroma planes in source      image.
29. **int** chrDstHSubSample;         ///< Binary logarithm of horizontal subsampling factor between luma/alpha and chroma planes in destination image.
30. **int** chrDstVSubSample;         ///< Binary logarithm of vertical   subsampling factor between luma/alpha and chroma planes in destination image.
31. **int** vChrDrop;                 ///< Binary logarithm of extra vertical subsampling factor in source image chroma planes specified by user.
32. **int** sliceDir;                 ///< Direction that slices are fed to the scaler (1 = top-to-bottom, -1 = bottom-to-top).
33. **double** param[2];              ///< Input parameters for scaling algorithms that need them.
35. /\* The cascaded\_\* fields allow spliting a scaler task into multiple
36. \* sequential steps, this is for example used to limit the maximum
37. \* downscaling factor that needs to be supported in one scaler.
38. \*/
39. **struct** SwsContext \*cascaded\_context[2];
40. **int** cascaded\_tmpStride[4];
41. uint8\_t \*cascaded\_tmp[4];
43. uint32\_t pal\_yuv[256];
44. uint32\_t pal\_rgb[256];
46. /\*\*
47. \* @name Scaled horizontal lines ring buffer.
48. \* The horizontal scaler keeps just enough scaled lines in a ring buffer
49. \* so they may be passed to the vertical scaler. The pointers to the
50. \* allocated buffers for each line are duplicated in sequence in the ring
51. \* buffer to simplify indexing and avoid wrapping around between lines
52. \* inside the vertical scaler code. The wrapping is done before the
53. \* vertical scaler is called.
54. \*/
55. //@{
56. int16\_t \*\*lumPixBuf;          ///< Ring buffer for scaled horizontal luma   plane lines to be fed to the vertical scaler.
57. int16\_t \*\*chrUPixBuf;         ///< Ring buffer for scaled horizontal chroma plane lines to be fed to the vertical scaler.
58. int16\_t \*\*chrVPixBuf;         ///< Ring buffer for scaled horizontal chroma plane lines to be fed to the vertical scaler.
59. int16\_t \*\*alpPixBuf;          ///< Ring buffer for scaled horizontal alpha  plane lines to be fed to the vertical scaler.
60. **int** vLumBufSize;              ///< Number of vertical luma/alpha lines allocated in the ring buffer.
61. **int** vChrBufSize;              ///< Number of vertical chroma     lines allocated in the ring buffer.
62. **int** lastInLumBuf;             ///< Last scaled horizontal luma/alpha line from source in the ring buffer.
63. **int** lastInChrBuf;             ///< Last scaled horizontal chroma     line from source in the ring buffer.
64. **int** lumBufIndex;              ///< Index in ring buffer of the last scaled horizontal luma/alpha line from source.
65. **int** chrBufIndex;              ///< Index in ring buffer of the last scaled horizontal chroma     line from source.
66. //@}
68. uint8\_t \*formatConvBuffer;
70. /\*\*
71. \* @name Horizontal and vertical filters.
72. \* To better understand the following fields, here is a pseudo-code of
73. \* their usage in filtering a horizontal line:
74. \* @code
75. \* for (i = 0; i < width; i++) {
76. \*     dst[i] = 0;
77. \*     for (j = 0; j < filterSize; j++)
78. \*         dst[i] += src[ filterPos[i] + j ] \* filter[ filterSize \* i + j ];
79. \*     dst[i] >>= FRAC\_BITS; // The actual implementation is fixed-point.
80. \* }
81. \* @endcode
82. \*/
83. //@{
84. int16\_t \*hLumFilter;          ///< Array of horizontal filter coefficients for luma/alpha planes.
85. int16\_t \*hChrFilter;          ///< Array of horizontal filter coefficients for chroma     planes.
86. int16\_t \*vLumFilter;          ///< Array of vertical   filter coefficients for luma/alpha planes.
87. int16\_t \*vChrFilter;          ///< Array of vertical   filter coefficients for chroma     planes.
88. int32\_t \*hLumFilterPos;       ///< Array of horizontal filter starting positions for each dst[i] for luma/alpha planes.
89. int32\_t \*hChrFilterPos;       ///< Array of horizontal filter starting positions for each dst[i] for chroma     planes.
90. int32\_t \*vLumFilterPos;       ///< Array of vertical   filter starting positions for each dst[i] for luma/alpha planes.
91. int32\_t \*vChrFilterPos;       ///< Array of vertical   filter starting positions for each dst[i] for chroma     planes.
92. **int** hLumFilterSize;           ///< Horizontal filter size for luma/alpha pixels.
93. **int** hChrFilterSize;           ///< Horizontal filter size for chroma     pixels.
94. **int** vLumFilterSize;           ///< Vertical   filter size for luma/alpha pixels.
95. **int** vChrFilterSize;           ///< Vertical   filter size for chroma     pixels.
96. //@}
98. **int** lumMmxextFilterCodeSize;  ///< Runtime-generated MMXEXT horizontal fast bilinear scaler code size for luma/alpha planes.
99. **int** chrMmxextFilterCodeSize;  ///< Runtime-generated MMXEXT horizontal fast bilinear scaler code size for chroma planes.
100. uint8\_t \*lumMmxextFilterCode; ///< Runtime-generated MMXEXT horizontal fast bilinear scaler code for luma/alpha planes.
101. uint8\_t \*chrMmxextFilterCode; ///< Runtime-generated MMXEXT horizontal fast bilinear scaler code for chroma planes.
103. **int** canMMXEXTBeUsed;
105. **int** dstY;                     ///< Last destination vertical line output from last slice.
106. **int** flags;                    ///< Flags passed by the user to select scaler algorithm, optimizations, subsampling, etc...
107. **void** \*yuvTable;             // pointer to the yuv->rgb table start so it can be freed()
108. // alignment ensures the offset can be added in a single
109. // instruction on e.g. ARM
110. DECLARE\_ALIGNED(16, **int**, table\_gV)[256 + 2\*YUVRGB\_TABLE\_HEADROOM];
111. uint8\_t \*table\_rV[256 + 2\*YUVRGB\_TABLE\_HEADROOM];
112. uint8\_t \*table\_gU[256 + 2\*YUVRGB\_TABLE\_HEADROOM];
113. uint8\_t \*table\_bU[256 + 2\*YUVRGB\_TABLE\_HEADROOM];
114. DECLARE\_ALIGNED(16, int32\_t, input\_rgb2yuv\_table)[16+40\*4]; // This table can contain both C and SIMD formatted values, the C vales are always at the XY\_IDX points
115. #define RY\_IDX 0
116. #define GY\_IDX 1
117. #define BY\_IDX 2
118. #define RU\_IDX 3
119. #define GU\_IDX 4
120. #define BU\_IDX 5
121. #define RV\_IDX 6
122. #define GV\_IDX 7
123. #define BV\_IDX 8
124. #define RGB2YUV\_SHIFT 15
126. **int** \*dither\_error[4];
128. //Colorspace stuff
129. **int** contrast, brightness, saturation;    // for sws\_getColorspaceDetails
130. **int** srcColorspaceTable[4];
131. **int** dstColorspaceTable[4];
132. **int** srcRange;                 ///< 0 = MPG YUV range, 1 = JPG YUV range (source      image).
133. **int** dstRange;                 ///< 0 = MPG YUV range, 1 = JPG YUV range (destination image).
134. **int** src0Alpha;
135. **int** dst0Alpha;
136. **int** srcXYZ;
137. **int** dstXYZ;
138. **int** src\_h\_chr\_pos;
139. **int** dst\_h\_chr\_pos;
140. **int** src\_v\_chr\_pos;
141. **int** dst\_v\_chr\_pos;
142. **int** yuv2rgb\_y\_offset;
143. **int** yuv2rgb\_y\_coeff;
144. **int** yuv2rgb\_v2r\_coeff;
145. **int** yuv2rgb\_v2g\_coeff;
146. **int** yuv2rgb\_u2g\_coeff;
147. **int** yuv2rgb\_u2b\_coeff;
149. #define RED\_DITHER            "0\*8"
150. #define GREEN\_DITHER          "1\*8"
151. #define BLUE\_DITHER           "2\*8"
152. #define Y\_COEFF               "3\*8"
153. #define VR\_COEFF              "4\*8"
154. #define UB\_COEFF              "5\*8"
155. #define VG\_COEFF              "6\*8"
156. #define UG\_COEFF              "7\*8"
157. #define Y\_OFFSET              "8\*8"
158. #define U\_OFFSET              "9\*8"
159. #define V\_OFFSET              "10\*8"
160. #define LUM\_MMX\_FILTER\_OFFSET "11\*8"
161. #define CHR\_MMX\_FILTER\_OFFSET "11\*8+4\*4\*"AV\_STRINGIFY(MAX\_FILTER\_SIZE)
162. #define DSTW\_OFFSET           "11\*8+4\*4\*"AV\_STRINGIFY(MAX\_FILTER\_SIZE)"\*2"
163. #define ESP\_OFFSET            "11\*8+4\*4\*"AV\_STRINGIFY(MAX\_FILTER\_SIZE)"\*2+8"
164. #define VROUNDER\_OFFSET       "11\*8+4\*4\*"AV\_STRINGIFY(MAX\_FILTER\_SIZE)"\*2+16"
165. #define U\_TEMP                "11\*8+4\*4\*"AV\_STRINGIFY(MAX\_FILTER\_SIZE)"\*2+24"
166. #define V\_TEMP                "11\*8+4\*4\*"AV\_STRINGIFY(MAX\_FILTER\_SIZE)"\*2+32"
167. #define Y\_TEMP                "11\*8+4\*4\*"AV\_STRINGIFY(MAX\_FILTER\_SIZE)"\*2+40"
168. #define ALP\_MMX\_FILTER\_OFFSET "11\*8+4\*4\*"AV\_STRINGIFY(MAX\_FILTER\_SIZE)"\*2+48"
169. #define UV\_OFF\_PX             "11\*8+4\*4\*"AV\_STRINGIFY(MAX\_FILTER\_SIZE)"\*3+48"
170. #define UV\_OFF\_BYTE           "11\*8+4\*4\*"AV\_STRINGIFY(MAX\_FILTER\_SIZE)"\*3+56"
171. #define DITHER16              "11\*8+4\*4\*"AV\_STRINGIFY(MAX\_FILTER\_SIZE)"\*3+64"
172. #define DITHER32              "11\*8+4\*4\*"AV\_STRINGIFY(MAX\_FILTER\_SIZE)"\*3+80"
173. #define DITHER32\_INT          (11\*8+4\*4\*MAX\_FILTER\_SIZE\*3+80) // value equal to above, used for checking that the struct hasn't been changed by mistake
175. DECLARE\_ALIGNED(8, uint64\_t, redDither);
176. DECLARE\_ALIGNED(8, uint64\_t, greenDither);
177. DECLARE\_ALIGNED(8, uint64\_t, blueDither);
179. DECLARE\_ALIGNED(8, uint64\_t, yCoeff);
180. DECLARE\_ALIGNED(8, uint64\_t, vrCoeff);
181. DECLARE\_ALIGNED(8, uint64\_t, ubCoeff);
182. DECLARE\_ALIGNED(8, uint64\_t, vgCoeff);
183. DECLARE\_ALIGNED(8, uint64\_t, ugCoeff);
184. DECLARE\_ALIGNED(8, uint64\_t, yOffset);
185. DECLARE\_ALIGNED(8, uint64\_t, uOffset);
186. DECLARE\_ALIGNED(8, uint64\_t, vOffset);
187. int32\_t lumMmxFilter[4 \* MAX\_FILTER\_SIZE];
188. int32\_t chrMmxFilter[4 \* MAX\_FILTER\_SIZE];
189. **int** dstW;                     ///< Width  of destination luma/alpha planes.
190. DECLARE\_ALIGNED(8, uint64\_t, esp);
191. DECLARE\_ALIGNED(8, uint64\_t, vRounder);
192. DECLARE\_ALIGNED(8, uint64\_t, u\_temp);
193. DECLARE\_ALIGNED(8, uint64\_t, v\_temp);
194. DECLARE\_ALIGNED(8, uint64\_t, y\_temp);
195. int32\_t alpMmxFilter[4 \* MAX\_FILTER\_SIZE];
196. // alignment of these values is not necessary, but merely here
197. // to maintain the same offset across x8632 and x86-64. Once we
198. // use proper offset macros in the asm, they can be removed.
199. DECLARE\_ALIGNED(8, **ptrdiff\_t**, uv\_off); ///< offset (in pixels) between u and v planes
200. DECLARE\_ALIGNED(8, **ptrdiff\_t**, uv\_offx2); ///< offset (in bytes) between u and v planes
201. DECLARE\_ALIGNED(8, uint16\_t, dither16)[8];
202. DECLARE\_ALIGNED(8, uint32\_t, dither32)[8];
204. **const** uint8\_t \*chrDither8, \*lumDither8;
206. #if HAVE\_ALTIVEC
207. vector **signed** **short**   CY;
208. vector **signed** **short**   CRV;
209. vector **signed** **short**   CBU;
210. vector **signed** **short**   CGU;
211. vector **signed** **short**   CGV;
212. vector **signed** **short**   OY;
213. vector unsigned **short** CSHIFT;
214. vector **signed** **short**  \*vYCoeffsBank, \*vCCoeffsBank;
215. #endif
217. **int** use\_mmx\_vfilter;
219. /\* pre defined color-spaces gamma \*/
220. #define XYZ\_GAMMA (2.6f)
221. #define RGB\_GAMMA (2.2f)
222. int16\_t \*xyzgamma;
223. int16\_t \*rgbgamma;
224. int16\_t \*xyzgammainv;
225. int16\_t \*rgbgammainv;
226. int16\_t xyz2rgb\_matrix[3][4];
227. int16\_t rgb2xyz\_matrix[3][4];
229. /\* function pointers for swscale() \*/
230. yuv2planar1\_fn yuv2plane1;
231. yuv2planarX\_fn yuv2planeX;
232. yuv2interleavedX\_fn yuv2nv12cX;
233. yuv2packed1\_fn yuv2packed1;
234. yuv2packed2\_fn yuv2packed2;
235. yuv2packedX\_fn yuv2packedX;
236. yuv2anyX\_fn yuv2anyX;
238. /// Unscaled conversion of luma plane to YV12 for horizontal scaler.
239. **void** (\*lumToYV12)(uint8\_t \*dst, **const** uint8\_t \*src, **const** uint8\_t \*src2, **const** uint8\_t \*src3,
240. **int** width, uint32\_t \*pal);
241. /// Unscaled conversion of alpha plane to YV12 for horizontal scaler.
242. **void** (\*alpToYV12)(uint8\_t \*dst, **const** uint8\_t \*src, **const** uint8\_t \*src2, **const** uint8\_t \*src3,
243. **int** width, uint32\_t \*pal);
244. /// Unscaled conversion of chroma planes to YV12 for horizontal scaler.
245. **void** (\*chrToYV12)(uint8\_t \*dstU, uint8\_t \*dstV,
246. **const** uint8\_t \*src1, **const** uint8\_t \*src2, **const** uint8\_t \*src3,
247. **int** width, uint32\_t \*pal);
249. /\*\*
250. \* Functions to read planar input, such as planar RGB, and convert
251. \* internally to Y/UV/A.
252. \*/
253. /\*\* @{ \*/
254. **void** (\*readLumPlanar)(uint8\_t \*dst, **const** uint8\_t \*src[4], **int** width, int32\_t \*rgb2yuv);
255. **void** (\*readChrPlanar)(uint8\_t \*dstU, uint8\_t \*dstV, **const** uint8\_t \*src[4],
256. **int** width, int32\_t \*rgb2yuv);
257. **void** (\*readAlpPlanar)(uint8\_t \*dst, **const** uint8\_t \*src[4], **int** width, int32\_t \*rgb2yuv);
258. /\*\* @} \*/
260. /\*\*
261. \* Scale one horizontal line of input data using a bilinear filter
262. \* to produce one line of output data. Compared to SwsContext->hScale(),
263. \* please take note of the following caveats when using these:
264. \* - Scaling is done using only 7bit instead of 14bit coefficients.
265. \* - You can use no more than 5 input pixels to produce 4 output
266. \*   pixels. Therefore, this filter should not be used for downscaling
267. \*   by more than ~20% in width (because that equals more than 5/4th
268. \*   downscaling and thus more than 5 pixels input per 4 pixels output).
269. \* - In general, bilinear filters create artifacts during downscaling
270. \*   (even when <20%), because one output pixel will span more than one
271. \*   input pixel, and thus some pixels will need edges of both neighbor
272. \*   pixels to interpolate the output pixel. Since you can use at most
273. \*   two input pixels per output pixel in bilinear scaling, this is
274. \*   impossible and thus downscaling by any size will create artifacts.
275. \* To enable this type of scaling, set SWS\_FLAG\_FAST\_BILINEAR
276. \* in SwsContext->flags.
277. \*/
278. /\*\* @{ \*/
279. **void** (\*hyscale\_fast)(**struct** SwsContext \*c,
280. int16\_t \*dst, **int** dstWidth,
281. **const** uint8\_t \*src, **int** srcW, **int** xInc);
282. **void** (\*hcscale\_fast)(**struct** SwsContext \*c,
283. int16\_t \*dst1, int16\_t \*dst2, **int** dstWidth,
284. **const** uint8\_t \*src1, **const** uint8\_t \*src2,
285. **int** srcW, **int** xInc);
286. /\*\* @} \*/
288. /\*\*
289. \* Scale one horizontal line of input data using a filter over the input
290. \* lines, to produce one (differently sized) line of output data.
291. \*
292. \* @param dst        pointer to destination buffer for horizontally scaled
293. \*                   data. If the number of bits per component of one
294. \*                   destination pixel (SwsContext->dstBpc) is <= 10, data
295. \*                   will be 15bpc in 16bits (int16\_t) width. Else (i.e.
296. \*                   SwsContext->dstBpc == 16), data will be 19bpc in
297. \*                   32bits (int32\_t) width.
298. \* @param dstW       width of destination image
299. \* @param src        pointer to source data to be scaled. If the number of
300. \*                   bits per component of a source pixel (SwsContext->srcBpc)
301. \*                   is 8, this is 8bpc in 8bits (uint8\_t) width. Else
302. \*                   (i.e. SwsContext->dstBpc > 8), this is native depth
303. \*                   in 16bits (uint16\_t) width. In other words, for 9-bit
304. \*                   YUV input, this is 9bpc, for 10-bit YUV input, this is
305. \*                   10bpc, and for 16-bit RGB or YUV, this is 16bpc.
306. \* @param filter     filter coefficients to be used per output pixel for
307. \*                   scaling. This contains 14bpp filtering coefficients.
308. \*                   Guaranteed to contain dstW \* filterSize entries.
309. \* @param filterPos  position of the first input pixel to be used for
310. \*                   each output pixel during scaling. Guaranteed to
311. \*                   contain dstW entries.
312. \* @param filterSize the number of input coefficients to be used (and
313. \*                   thus the number of input pixels to be used) for
314. \*                   creating a single output pixel. Is aligned to 4
315. \*                   (and input coefficients thus padded with zeroes)
316. \*                   to simplify creating SIMD code.
317. \*/
318. /\*\* @{ \*/
319. **void** (\*hyScale)(**struct** SwsContext \*c, int16\_t \*dst, **int** dstW,
320. **const** uint8\_t \*src, **const** int16\_t \*filter,
321. **const** int32\_t \*filterPos, **int** filterSize);
322. **void** (\*hcScale)(**struct** SwsContext \*c, int16\_t \*dst, **int** dstW,
323. **const** uint8\_t \*src, **const** int16\_t \*filter,
324. **const** int32\_t \*filterPos, **int** filterSize);
325. /\*\* @} \*/
327. /// Color range conversion function for luma plane if needed.
328. **void** (\*lumConvertRange)(int16\_t \*dst, **int** width);
329. /// Color range conversion function for chroma planes if needed.
330. **void** (\*chrConvertRange)(int16\_t \*dst1, int16\_t \*dst2, **int** width);
332. **int** needs\_hcscale; ///< Set if there are chroma planes to be converted.
334. SwsDither dither;
335. } SwsContext;

这个结构体的定义确实比较复杂，里面包含了libswscale所需要的全部变量。一一分析这些变量是不太现实的，在后文中会简单分析其中的几个变量。

## sws\_getContext()

sws\_getContext()是初始化SwsContext的函数。sws\_getContext()的声明位于libswscale\swscale.h，如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44305697) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44305697)

[在CODE上查看代码片](https://code.csdn.net/snippets/621402)

1. /\*\*
2. \* Allocate and return an SwsContext. You need it to perform
3. \* scaling/conversion operations using sws\_scale().
4. \*
5. \* @param srcW the width of the source image
6. \* @param srcH the height of the source image
7. \* @param srcFormat the source image format
8. \* @param dstW the width of the destination image
9. \* @param dstH the height of the destination image
10. \* @param dstFormat the destination image format
11. \* @param flags specify which algorithm and options to use for rescaling
12. \* @return a pointer to an allocated context, or NULL in case of error
13. \* @note this function is to be removed after a saner alternative is
14. \*       written
15. \*/
16. **struct** SwsContext \*sws\_getContext(**int** srcW, **int** srcH, **enum** AVPixelFormat srcFormat,
17. **int** dstW, **int** dstH, **enum** AVPixelFormat dstFormat,
18. **int** flags, SwsFilter \*srcFilter,
19. SwsFilter \*dstFilter, **const** **double** \*param);

该函数包含以下参数：

srcW：源图像的宽  
srcH：源图像的高  
srcFormat：源图像的像素格式  
dstW：目标图像的宽  
dstH：目标图像的高  
dstFormat：目标图像的像素格式  
flags：设定图像拉伸使用的算法

成功执行的话返回生成的SwsContext，否则返回NULL。  
sws\_getContext()的定义位于libswscale\utils.c，如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44305697) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44305697)

[在CODE上查看代码片](https://code.csdn.net/snippets/621402)

1. SwsContext \*sws\_getContext(**int** srcW, **int** srcH, **enum** AVPixelFormat srcFormat,
2. **int** dstW, **int** dstH, **enum** AVPixelFormat dstFormat,
3. **int** flags, SwsFilter \*srcFilter,
4. SwsFilter \*dstFilter, **const** **double** \*param)
5. {
6. SwsContext \*c;
8. **if** (!(c = sws\_alloc\_context()))
9. **return** NULL;
11. c->flags     = flags;
12. c->srcW      = srcW;
13. c->srcH      = srcH;
14. c->dstW      = dstW;
15. c->dstH      = dstH;
16. c->srcFormat = srcFormat;
17. c->dstFormat = dstFormat;
19. **if** (param) {
20. c->param[0] = param[0];
21. c->param[1] = param[1];
22. }
24. **if** (sws\_init\_context(c, srcFilter, dstFilter) < 0) {
25. sws\_freeContext(c);
26. **return** NULL;
27. }
29. **return** c;
30. }

从sws\_getContext()的定义中可以看出，它首先调用了一个函数sws\_alloc\_context()用于给SwsContext分配内存。然后将传入的源图像，目标图像的宽高，像素格式，以及标志位分别赋值给该SwsContext相应的字段。最后调用一个函数sws\_init\_context()完成初始化工作。下面我们分别看一下sws\_alloc\_context()和sws\_init\_context()这两个函数。

## sws\_alloc\_context()

sws\_alloc\_context()是FFmpeg的一个API，用于给SwsContext分配内存，它的声明如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44305697) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44305697)

[在CODE上查看代码片](https://code.csdn.net/snippets/621402)

1. /\*\*
2. \* Allocate an empty SwsContext. This must be filled and passed to
3. \* sws\_init\_context(). For filling see AVOptions, options.c and
4. \* sws\_setColorspaceDetails().
5. \*/
6. **struct** SwsContext \*sws\_alloc\_context(**void**);

sws\_alloc\_context()的定义位于libswscale\utils.c，如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44305697) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44305697)

[在CODE上查看代码片](https://code.csdn.net/snippets/621402)

1. SwsContext \*sws\_alloc\_context(**void**)
2. {
3. SwsContext \*c = av\_mallocz(**sizeof**(SwsContext));
5. av\_assert0(offsetof(SwsContext, redDither) + DITHER32\_INT == offsetof(SwsContext, dither32));
7. **if** (c) {
8. c->av\_class = &sws\_context\_class;
9. av\_opt\_set\_defaults(c);
10. }
12. **return** c;
13. }

从代码中可以看出，sws\_alloc\_context()首先调用av\_mallocz()为SwsContext结构体分配了一块内存；然后设置了该结构体的AVClass，并且给该结构体的字段设置了默认值。

## sws\_init\_context()

sws\_init\_context()的是FFmpeg的一个API，用于初始化SwsContext。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44305697) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44305697)

[在CODE上查看代码片](https://code.csdn.net/snippets/621402)

1. /\*\*
2. \* Initialize the swscaler context sws\_context.
3. \*
4. \* @return zero or positive value on success, a negative value on
5. \* error
6. \*/
7. **int** sws\_init\_context(**struct** SwsContext \*sws\_context, SwsFilter \*srcFilter, SwsFilter \*dstFilter);

sws\_init\_context()的函数定义非常的长，位于libswscale\utils.c，如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44305697) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44305697)

[在CODE上查看代码片](https://code.csdn.net/snippets/621402)

1. av\_cold **int** sws\_init\_context(SwsContext \*c, SwsFilter \*srcFilter,
2. SwsFilter \*dstFilter)
3. {
4. **int** i, j;
5. **int** usesVFilter, usesHFilter;
6. **int** unscaled;
7. SwsFilter dummyFilter = { NULL, NULL, NULL, NULL };
8. **int** srcW              = c->srcW;
9. **int** srcH              = c->srcH;
10. **int** dstW              = c->dstW;
11. **int** dstH              = c->dstH;
12. **int** dst\_stride        = FFALIGN(dstW \* **sizeof**(int16\_t) + 66, 16);
13. **int** flags, cpu\_flags;
14. **enum** AVPixelFormat srcFormat = c->srcFormat;
15. **enum** AVPixelFormat dstFormat = c->dstFormat;
16. **const** AVPixFmtDescriptor \*desc\_src;
17. **const** AVPixFmtDescriptor \*desc\_dst;
18. **int** ret = 0;
19. //获取
20. cpu\_flags = av\_get\_cpu\_flags();
21. flags     = c->flags;
22. emms\_c();
23. **if** (!rgb15to16)
24. sws\_rgb2rgb\_init();
25. //如果输入的宽高和输出的宽高一样，则做特殊处理
26. unscaled = (srcW == dstW && srcH == dstH);
27. //如果是JPEG标准（Y取值0-255），则需要设置这两项
28. c->srcRange |= handle\_jpeg(&c->srcFormat);
29. c->dstRange |= handle\_jpeg(&c->dstFormat);
31. **if**(srcFormat!=c->srcFormat || dstFormat!=c->dstFormat)
32. av\_log(c, AV\_LOG\_WARNING, "deprecated pixel format used, make sure you did set range correctly\n");
33. //设置Colorspace
34. **if** (!c->contrast && !c->saturation && !c->dstFormatBpp)
35. sws\_setColorspaceDetails(c, ff\_yuv2rgb\_coeffs[SWS\_CS\_DEFAULT], c->srcRange,
36. ff\_yuv2rgb\_coeffs[SWS\_CS\_DEFAULT],
37. c->dstRange, 0, 1 << 16, 1 << 16);
39. handle\_formats(c);
40. srcFormat = c->srcFormat;
41. dstFormat = c->dstFormat;
42. desc\_src = av\_pix\_fmt\_desc\_get(srcFormat);
43. desc\_dst = av\_pix\_fmt\_desc\_get(dstFormat);
44. //转换大小端？
45. **if** (!(unscaled && sws\_isSupportedEndiannessConversion(srcFormat) &&
46. av\_pix\_fmt\_swap\_endianness(srcFormat) == dstFormat)) {
47. //检查输入格式是否支持
48. **if** (!sws\_isSupportedInput(srcFormat)) {
49. av\_log(c, AV\_LOG\_ERROR, "%s is not supported as input pixel format\n",
50. av\_get\_pix\_fmt\_name(srcFormat));
51. **return** AVERROR(EINVAL);
52. }
53. //检查输出格式是否支持
54. **if** (!sws\_isSupportedOutput(dstFormat)) {
55. av\_log(c, AV\_LOG\_ERROR, "%s is not supported as output pixel format\n",
56. av\_get\_pix\_fmt\_name(dstFormat));
57. **return** AVERROR(EINVAL);
58. }
59. }
60. //检查拉伸的方法
61. i = flags & (SWS\_POINT         |
62. SWS\_AREA          |
63. SWS\_BILINEAR      |
64. SWS\_FAST\_BILINEAR |
65. SWS\_BICUBIC       |
66. SWS\_X             |
67. SWS\_GAUSS         |
68. SWS\_LANCZOS       |
69. SWS\_SINC          |
70. SWS\_SPLINE        |
71. SWS\_BICUBLIN);
73. /\* provide a default scaler if not set by caller \*/
74. //如果没有指定，就使用默认的
75. **if** (!i) {
76. **if** (dstW < srcW && dstH < srcH)
77. flags |= SWS\_BICUBIC;
78. **else** **if** (dstW > srcW && dstH > srcH)
79. flags |= SWS\_BICUBIC;
80. **else**
81. flags |= SWS\_BICUBIC;
82. c->flags = flags;
83. } **else** **if** (i & (i - 1)) {
84. av\_log(c, AV\_LOG\_ERROR,
85. "Exactly one scaler algorithm must be chosen, got %X\n", i);
86. **return** AVERROR(EINVAL);
87. }
88. /\* sanity check \*/
89. //检查宽高参数
90. **if** (srcW < 1 || srcH < 1 || dstW < 1 || dstH < 1) {
91. /\* FIXME check if these are enough and try to lower them after
92. \* fixing the relevant parts of the code \*/
93. av\_log(c, AV\_LOG\_ERROR, "%dx%d -> %dx%d is invalid scaling dimension\n",
94. srcW, srcH, dstW, dstH);
95. **return** AVERROR(EINVAL);
96. }
98. **if** (!dstFilter)
99. dstFilter = &dummyFilter;
100. **if** (!srcFilter)
101. srcFilter = &dummyFilter;
103. c->lumXInc      = (((int64\_t)srcW << 16) + (dstW >> 1)) / dstW;
104. c->lumYInc      = (((int64\_t)srcH << 16) + (dstH >> 1)) / dstH;
105. c->dstFormatBpp = av\_get\_bits\_per\_pixel(desc\_dst);
106. c->srcFormatBpp = av\_get\_bits\_per\_pixel(desc\_src);
107. c->vRounder     = 4 \* 0x0001000100010001ULL;
109. usesVFilter = (srcFilter->lumV && srcFilter->lumV->length > 1) ||
110. (srcFilter->chrV && srcFilter->chrV->length > 1) ||
111. (dstFilter->lumV && dstFilter->lumV->length > 1) ||
112. (dstFilter->chrV && dstFilter->chrV->length > 1);
113. usesHFilter = (srcFilter->lumH && srcFilter->lumH->length > 1) ||
114. (srcFilter->chrH && srcFilter->chrH->length > 1) ||
115. (dstFilter->lumH && dstFilter->lumH->length > 1) ||
116. (dstFilter->chrH && dstFilter->chrH->length > 1);
118. av\_pix\_fmt\_get\_chroma\_sub\_sample(srcFormat, &c->chrSrcHSubSample, &c->chrSrcVSubSample);
119. av\_pix\_fmt\_get\_chroma\_sub\_sample(dstFormat, &c->chrDstHSubSample, &c->chrDstVSubSample);
121. **if** (isAnyRGB(dstFormat) && !(flags&SWS\_FULL\_CHR\_H\_INT)) {
122. **if** (dstW&1) {
123. av\_log(c, AV\_LOG\_DEBUG, "Forcing full internal H chroma due to odd output size\n");
124. flags |= SWS\_FULL\_CHR\_H\_INT;
125. c->flags = flags;
126. }
128. **if** (   c->chrSrcHSubSample == 0
129. && c->chrSrcVSubSample == 0
130. && c->dither != SWS\_DITHER\_BAYER //SWS\_FULL\_CHR\_H\_INT is currently not supported with SWS\_DITHER\_BAYER
131. && !(c->flags & SWS\_FAST\_BILINEAR)
132. ) {
133. av\_log(c, AV\_LOG\_DEBUG, "Forcing full internal H chroma due to input having non subsampled chroma\n");
134. flags |= SWS\_FULL\_CHR\_H\_INT;
135. c->flags = flags;
136. }
137. }
139. **if** (c->dither == SWS\_DITHER\_AUTO) {
140. **if** (flags & SWS\_ERROR\_DIFFUSION)
141. c->dither = SWS\_DITHER\_ED;
142. }
144. **if**(dstFormat == AV\_PIX\_FMT\_BGR4\_BYTE ||
145. dstFormat == AV\_PIX\_FMT\_RGB4\_BYTE ||
146. dstFormat == AV\_PIX\_FMT\_BGR8 ||
147. dstFormat == AV\_PIX\_FMT\_RGB8) {
148. **if** (c->dither == SWS\_DITHER\_AUTO)
149. c->dither = (flags & SWS\_FULL\_CHR\_H\_INT) ? SWS\_DITHER\_ED : SWS\_DITHER\_BAYER;
150. **if** (!(flags & SWS\_FULL\_CHR\_H\_INT)) {
151. **if** (c->dither == SWS\_DITHER\_ED || c->dither == SWS\_DITHER\_A\_DITHER || c->dither == SWS\_DITHER\_X\_DITHER) {
152. av\_log(c, AV\_LOG\_DEBUG,
153. "Desired dithering only supported in full chroma interpolation for destination format '%s'\n",
154. av\_get\_pix\_fmt\_name(dstFormat));
155. flags   |= SWS\_FULL\_CHR\_H\_INT;
156. c->flags = flags;
157. }
158. }
159. **if** (flags & SWS\_FULL\_CHR\_H\_INT) {
160. **if** (c->dither == SWS\_DITHER\_BAYER) {
161. av\_log(c, AV\_LOG\_DEBUG,
162. "Ordered dither is not supported in full chroma interpolation for destination format '%s'\n",
163. av\_get\_pix\_fmt\_name(dstFormat));
164. c->dither = SWS\_DITHER\_ED;
165. }
166. }
167. }
168. **if** (isPlanarRGB(dstFormat)) {
169. **if** (!(flags & SWS\_FULL\_CHR\_H\_INT)) {
170. av\_log(c, AV\_LOG\_DEBUG,
171. "%s output is not supported with half chroma resolution, switching to full\n",
172. av\_get\_pix\_fmt\_name(dstFormat));
173. flags   |= SWS\_FULL\_CHR\_H\_INT;
174. c->flags = flags;
175. }
176. }
178. /\* reuse chroma for 2 pixels RGB/BGR unless user wants full
179. \* chroma interpolation \*/
180. **if** (flags & SWS\_FULL\_CHR\_H\_INT &&
181. isAnyRGB(dstFormat)        &&
182. !isPlanarRGB(dstFormat)    &&
183. dstFormat != AV\_PIX\_FMT\_RGBA  &&
184. dstFormat != AV\_PIX\_FMT\_ARGB  &&
185. dstFormat != AV\_PIX\_FMT\_BGRA  &&
186. dstFormat != AV\_PIX\_FMT\_ABGR  &&
187. dstFormat != AV\_PIX\_FMT\_RGB24 &&
188. dstFormat != AV\_PIX\_FMT\_BGR24 &&
189. dstFormat != AV\_PIX\_FMT\_BGR4\_BYTE &&
190. dstFormat != AV\_PIX\_FMT\_RGB4\_BYTE &&
191. dstFormat != AV\_PIX\_FMT\_BGR8 &&
192. dstFormat != AV\_PIX\_FMT\_RGB8
193. ) {
194. av\_log(c, AV\_LOG\_WARNING,
195. "full chroma interpolation for destination format '%s' not yet implemented\n",
196. av\_get\_pix\_fmt\_name(dstFormat));
197. flags   &= ~SWS\_FULL\_CHR\_H\_INT;
198. c->flags = flags;
199. }
200. **if** (isAnyRGB(dstFormat) && !(flags & SWS\_FULL\_CHR\_H\_INT))
201. c->chrDstHSubSample = 1;
203. // drop some chroma lines if the user wants it
204. c->vChrDrop          = (flags & SWS\_SRC\_V\_CHR\_DROP\_MASK) >>
205. SWS\_SRC\_V\_CHR\_DROP\_SHIFT;
206. c->chrSrcVSubSample += c->vChrDrop;
208. /\* drop every other pixel for chroma calculation unless user
209. \* wants full chroma \*/
210. **if** (isAnyRGB(srcFormat) && !(flags & SWS\_FULL\_CHR\_H\_INP)   &&
211. srcFormat != AV\_PIX\_FMT\_RGB8 && srcFormat != AV\_PIX\_FMT\_BGR8 &&
212. srcFormat != AV\_PIX\_FMT\_RGB4 && srcFormat != AV\_PIX\_FMT\_BGR4 &&
213. srcFormat != AV\_PIX\_FMT\_RGB4\_BYTE && srcFormat != AV\_PIX\_FMT\_BGR4\_BYTE &&
214. srcFormat != AV\_PIX\_FMT\_GBRP9BE   && srcFormat != AV\_PIX\_FMT\_GBRP9LE  &&
215. srcFormat != AV\_PIX\_FMT\_GBRP10BE  && srcFormat != AV\_PIX\_FMT\_GBRP10LE &&
216. srcFormat != AV\_PIX\_FMT\_GBRP12BE  && srcFormat != AV\_PIX\_FMT\_GBRP12LE &&
217. srcFormat != AV\_PIX\_FMT\_GBRP14BE  && srcFormat != AV\_PIX\_FMT\_GBRP14LE &&
218. srcFormat != AV\_PIX\_FMT\_GBRP16BE  && srcFormat != AV\_PIX\_FMT\_GBRP16LE &&
219. ((dstW >> c->chrDstHSubSample) <= (srcW >> 1) ||
220. (flags & SWS\_FAST\_BILINEAR)))
221. c->chrSrcHSubSample = 1;
223. // Note the FF\_CEIL\_RSHIFT is so that we always round toward +inf.
224. c->chrSrcW = FF\_CEIL\_RSHIFT(srcW, c->chrSrcHSubSample);
225. c->chrSrcH = FF\_CEIL\_RSHIFT(srcH, c->chrSrcVSubSample);
226. c->chrDstW = FF\_CEIL\_RSHIFT(dstW, c->chrDstHSubSample);
227. c->chrDstH = FF\_CEIL\_RSHIFT(dstH, c->chrDstVSubSample);
229. FF\_ALLOC\_OR\_GOTO(c, c->formatConvBuffer, FFALIGN(srcW\*2+78, 16) \* 2, fail);
231. c->srcBpc = 1 + desc\_src->comp[0].depth\_minus1;
232. **if** (c->srcBpc < 8)
233. c->srcBpc = 8;
234. c->dstBpc = 1 + desc\_dst->comp[0].depth\_minus1;
235. **if** (c->dstBpc < 8)
236. c->dstBpc = 8;
237. **if** (isAnyRGB(srcFormat) || srcFormat == AV\_PIX\_FMT\_PAL8)
238. c->srcBpc = 16;
239. **if** (c->dstBpc == 16)
240. dst\_stride <<= 1;
242. **if** (INLINE\_MMXEXT(cpu\_flags) && c->srcBpc == 8 && c->dstBpc <= 14) {
243. c->canMMXEXTBeUsed = dstW >= srcW && (dstW & 31) == 0 &&
244. c->chrDstW >= c->chrSrcW &&
245. (srcW & 15) == 0;
246. **if** (!c->canMMXEXTBeUsed && dstW >= srcW && c->chrDstW >= c->chrSrcW && (srcW & 15) == 0
248. && (flags & SWS\_FAST\_BILINEAR)) {
249. **if** (flags & SWS\_PRINT\_INFO)
250. av\_log(c, AV\_LOG\_INFO,
251. "output width is not a multiple of 32 -> no MMXEXT scaler\n");
252. }
253. **if** (usesHFilter || isNBPS(c->srcFormat) || is16BPS(c->srcFormat) || isAnyRGB(c->srcFormat))
254. c->canMMXEXTBeUsed = 0;
255. } **else**
256. c->canMMXEXTBeUsed = 0;
258. c->chrXInc = (((int64\_t)c->chrSrcW << 16) + (c->chrDstW >> 1)) / c->chrDstW;
259. c->chrYInc = (((int64\_t)c->chrSrcH << 16) + (c->chrDstH >> 1)) / c->chrDstH;
261. /\* Match pixel 0 of the src to pixel 0 of dst and match pixel n-2 of src
262. \* to pixel n-2 of dst, but only for the FAST\_BILINEAR mode otherwise do
263. \* correct scaling.
264. \* n-2 is the last chrominance sample available.
265. \* This is not perfect, but no one should notice the difference, the more
266. \* correct variant would be like the vertical one, but that would require
267. \* some special code for the first and last pixel \*/
268. **if** (flags & SWS\_FAST\_BILINEAR) {
269. **if** (c->canMMXEXTBeUsed) {
270. c->lumXInc += 20;
271. c->chrXInc += 20;
272. }
273. // we don't use the x86 asm scaler if MMX is available
274. **else** **if** (INLINE\_MMX(cpu\_flags) && c->dstBpc <= 14) {
275. c->lumXInc = ((int64\_t)(srcW       - 2) << 16) / (dstW       - 2) - 20;
276. c->chrXInc = ((int64\_t)(c->chrSrcW - 2) << 16) / (c->chrDstW - 2) - 20;
277. }
278. }
280. **if** (isBayer(srcFormat)) {
281. **if** (!unscaled ||
282. (dstFormat != AV\_PIX\_FMT\_RGB24 && dstFormat != AV\_PIX\_FMT\_YUV420P)) {
283. **enum** AVPixelFormat tmpFormat = AV\_PIX\_FMT\_RGB24;
285. ret = av\_image\_alloc(c->cascaded\_tmp, c->cascaded\_tmpStride,
286. srcW, srcH, tmpFormat, 64);
287. **if** (ret < 0)
288. **return** ret;
290. c->cascaded\_context[0] = sws\_getContext(srcW, srcH, srcFormat,
291. srcW, srcH, tmpFormat,
292. flags, srcFilter, NULL, c->param);
293. **if** (!c->cascaded\_context[0])
294. **return** -1;
296. c->cascaded\_context[1] = sws\_getContext(srcW, srcH, tmpFormat,
297. dstW, dstH, dstFormat,
298. flags, NULL, dstFilter, c->param);
299. **if** (!c->cascaded\_context[1])
300. **return** -1;
301. **return** 0;
302. }
303. }
305. #define USE\_MMAP (HAVE\_MMAP && HAVE\_MPROTECT && defined MAP\_ANONYMOUS)
307. /\* precalculate horizontal scaler filter coefficients \*/
308. {
309. #if HAVE\_MMXEXT\_INLINE
310. // can't downscale !!!
311. **if** (c->canMMXEXTBeUsed && (flags & SWS\_FAST\_BILINEAR)) {
312. c->lumMmxextFilterCodeSize = ff\_init\_hscaler\_mmxext(dstW, c->lumXInc, NULL,
313. NULL, NULL, 8);
314. c->chrMmxextFilterCodeSize = ff\_init\_hscaler\_mmxext(c->chrDstW, c->chrXInc,
315. NULL, NULL, NULL, 4);
317. #if USE\_MMAP
318. c->lumMmxextFilterCode = mmap(NULL, c->lumMmxextFilterCodeSize,
319. PROT\_READ | PROT\_WRITE,
320. MAP\_PRIVATE | MAP\_ANONYMOUS,
321. -1, 0);
322. c->chrMmxextFilterCode = mmap(NULL, c->chrMmxextFilterCodeSize,
323. PROT\_READ | PROT\_WRITE,
324. MAP\_PRIVATE | MAP\_ANONYMOUS,
325. -1, 0);
326. #elif HAVE\_VIRTUALALLOC
327. c->lumMmxextFilterCode = VirtualAlloc(NULL,
328. c->lumMmxextFilterCodeSize,
329. MEM\_COMMIT,
330. PAGE\_EXECUTE\_READWRITE);
331. c->chrMmxextFilterCode = VirtualAlloc(NULL,
332. c->chrMmxextFilterCodeSize,
333. MEM\_COMMIT,
334. PAGE\_EXECUTE\_READWRITE);
335. #else
336. c->lumMmxextFilterCode = av\_malloc(c->lumMmxextFilterCodeSize);
337. c->chrMmxextFilterCode = av\_malloc(c->chrMmxextFilterCodeSize);
338. #endif
340. #ifdef MAP\_ANONYMOUS
341. **if** (c->lumMmxextFilterCode == MAP\_FAILED || c->chrMmxextFilterCode == MAP\_FAILED)
342. #else
343. **if** (!c->lumMmxextFilterCode || !c->chrMmxextFilterCode)
344. #endif
345. {
346. av\_log(c, AV\_LOG\_ERROR, "Failed to allocate MMX2FilterCode\n");
347. **return** AVERROR(ENOMEM);
348. }
350. FF\_ALLOCZ\_OR\_GOTO(c, c->hLumFilter,    (dstW           / 8 + 8) \* **sizeof**(int16\_t), fail);
351. FF\_ALLOCZ\_OR\_GOTO(c, c->hChrFilter,    (c->chrDstW     / 4 + 8) \* **sizeof**(int16\_t), fail);
352. FF\_ALLOCZ\_OR\_GOTO(c, c->hLumFilterPos, (dstW       / 2 / 8 + 8) \* **sizeof**(int32\_t), fail);
353. FF\_ALLOCZ\_OR\_GOTO(c, c->hChrFilterPos, (c->chrDstW / 2 / 4 + 8) \* **sizeof**(int32\_t), fail);
355. ff\_init\_hscaler\_mmxext(      dstW, c->lumXInc, c->lumMmxextFilterCode,
356. c->hLumFilter, (uint32\_t\*)c->hLumFilterPos, 8);
357. ff\_init\_hscaler\_mmxext(c->chrDstW, c->chrXInc, c->chrMmxextFilterCode,
358. c->hChrFilter, (uint32\_t\*)c->hChrFilterPos, 4);
360. #if USE\_MMAP
361. **if** (   mprotect(c->lumMmxextFilterCode, c->lumMmxextFilterCodeSize, PROT\_EXEC | PROT\_READ) == -1
362. || mprotect(c->chrMmxextFilterCode, c->chrMmxextFilterCodeSize, PROT\_EXEC | PROT\_READ) == -1) {
363. av\_log(c, AV\_LOG\_ERROR, "mprotect failed, cannot use fast bilinear scaler\n");
364. **goto** fail;
365. }
366. #endif
367. } **else**
368. #endif /\* HAVE\_MMXEXT\_INLINE \*/
369. {
370. **const** **int** filterAlign = X86\_MMX(cpu\_flags)     ? 4 :
371. PPC\_ALTIVEC(cpu\_flags) ? 8 : 1;
373. **if** ((ret = initFilter(&c->hLumFilter, &c->hLumFilterPos,
374. &c->hLumFilterSize, c->lumXInc,
375. srcW, dstW, filterAlign, 1 << 14,
376. (flags & SWS\_BICUBLIN) ? (flags | SWS\_BICUBIC) : flags,
377. cpu\_flags, srcFilter->lumH, dstFilter->lumH,
378. c->param,
379. get\_local\_pos(c, 0, 0, 0),
380. get\_local\_pos(c, 0, 0, 0))) < 0)
381. **goto** fail;
382. **if** ((ret = initFilter(&c->hChrFilter, &c->hChrFilterPos,
383. &c->hChrFilterSize, c->chrXInc,
384. c->chrSrcW, c->chrDstW, filterAlign, 1 << 14,
385. (flags & SWS\_BICUBLIN) ? (flags | SWS\_BILINEAR) : flags,
386. cpu\_flags, srcFilter->chrH, dstFilter->chrH,
387. c->param,
388. get\_local\_pos(c, c->chrSrcHSubSample, c->src\_h\_chr\_pos, 0),
389. get\_local\_pos(c, c->chrDstHSubSample, c->dst\_h\_chr\_pos, 0))) < 0)
390. **goto** fail;
391. }
392. } // initialize horizontal stuff
394. /\* precalculate vertical scaler filter coefficients \*/
395. {
396. **const** **int** filterAlign = X86\_MMX(cpu\_flags)     ? 2 :
397. PPC\_ALTIVEC(cpu\_flags) ? 8 : 1;
399. **if** ((ret = initFilter(&c->vLumFilter, &c->vLumFilterPos, &c->vLumFilterSize,
400. c->lumYInc, srcH, dstH, filterAlign, (1 << 12),
401. (flags & SWS\_BICUBLIN) ? (flags | SWS\_BICUBIC) : flags,
402. cpu\_flags, srcFilter->lumV, dstFilter->lumV,
403. c->param,
404. get\_local\_pos(c, 0, 0, 1),
405. get\_local\_pos(c, 0, 0, 1))) < 0)
406. **goto** fail;
407. **if** ((ret = initFilter(&c->vChrFilter, &c->vChrFilterPos, &c->vChrFilterSize,
408. c->chrYInc, c->chrSrcH, c->chrDstH,
409. filterAlign, (1 << 12),
410. (flags & SWS\_BICUBLIN) ? (flags | SWS\_BILINEAR) : flags,
411. cpu\_flags, srcFilter->chrV, dstFilter->chrV,
412. c->param,
413. get\_local\_pos(c, c->chrSrcVSubSample, c->src\_v\_chr\_pos, 1),
414. get\_local\_pos(c, c->chrDstVSubSample, c->dst\_v\_chr\_pos, 1))) < 0)
416. **goto** fail;
418. #if HAVE\_ALTIVEC
419. FF\_ALLOC\_OR\_GOTO(c, c->vYCoeffsBank, **sizeof**(vector **signed** **short**) \* c->vLumFilterSize \* c->dstH,    fail);
420. FF\_ALLOC\_OR\_GOTO(c, c->vCCoeffsBank, **sizeof**(vector **signed** **short**) \* c->vChrFilterSize \* c->chrDstH, fail);
422. **for** (i = 0; i < c->vLumFilterSize \* c->dstH; i++) {
423. **int** j;
424. **short** \*p = (**short** \*)&c->vYCoeffsBank[i];
425. **for** (j = 0; j < 8; j++)
426. p[j] = c->vLumFilter[i];
427. }
429. **for** (i = 0; i < c->vChrFilterSize \* c->chrDstH; i++) {
430. **int** j;
431. **short** \*p = (**short** \*)&c->vCCoeffsBank[i];
432. **for** (j = 0; j < 8; j++)
433. p[j] = c->vChrFilter[i];
434. }
435. #endif
436. }
438. // calculate buffer sizes so that they won't run out while handling these damn slices
439. c->vLumBufSize = c->vLumFilterSize;
440. c->vChrBufSize = c->vChrFilterSize;
441. **for** (i = 0; i < dstH; i++) {
442. **int** chrI      = (int64\_t)i \* c->chrDstH / dstH;
443. **int** nextSlice = FFMAX(c->vLumFilterPos[i] + c->vLumFilterSize - 1,
444. ((c->vChrFilterPos[chrI] + c->vChrFilterSize - 1)
445. << c->chrSrcVSubSample));
447. nextSlice >>= c->chrSrcVSubSample;
448. nextSlice <<= c->chrSrcVSubSample;
449. **if** (c->vLumFilterPos[i] + c->vLumBufSize < nextSlice)
450. c->vLumBufSize = nextSlice - c->vLumFilterPos[i];
451. **if** (c->vChrFilterPos[chrI] + c->vChrBufSize <
452. (nextSlice >> c->chrSrcVSubSample))
453. c->vChrBufSize = (nextSlice >> c->chrSrcVSubSample) -
454. c->vChrFilterPos[chrI];
455. }
457. **for** (i = 0; i < 4; i++)
458. FF\_ALLOCZ\_OR\_GOTO(c, c->dither\_error[i], (c->dstW+2) \* **sizeof**(**int**), fail);
460. /\* Allocate pixbufs (we use dynamic allocation because otherwise we would
461. \* need to allocate several megabytes to handle all possible cases) \*/
462. FF\_ALLOC\_OR\_GOTO(c, c->lumPixBuf,  c->vLumBufSize \* 3 \* **sizeof**(int16\_t \*), fail);
463. FF\_ALLOC\_OR\_GOTO(c, c->chrUPixBuf, c->vChrBufSize \* 3 \* **sizeof**(int16\_t \*), fail);
464. FF\_ALLOC\_OR\_GOTO(c, c->chrVPixBuf, c->vChrBufSize \* 3 \* **sizeof**(int16\_t \*), fail);
465. **if** (CONFIG\_SWSCALE\_ALPHA && isALPHA(c->srcFormat) && isALPHA(c->dstFormat))
466. FF\_ALLOCZ\_OR\_GOTO(c, c->alpPixBuf, c->vLumBufSize \* 3 \* **sizeof**(int16\_t \*), fail);
467. /\* Note we need at least one pixel more at the end because of the MMX code
468. \* (just in case someone wants to replace the 4000/8000). \*/
469. /\* align at 16 bytes for AltiVec \*/
470. **for** (i = 0; i < c->vLumBufSize; i++) {
471. FF\_ALLOCZ\_OR\_GOTO(c, c->lumPixBuf[i + c->vLumBufSize],
472. dst\_stride + 16, fail);
473. c->lumPixBuf[i] = c->lumPixBuf[i + c->vLumBufSize];
474. }
475. // 64 / c->scalingBpp is the same as 16 / sizeof(scaling\_intermediate)
476. c->uv\_off   = (dst\_stride>>1) + 64 / (c->dstBpc &~ 7);
477. c->uv\_offx2 = dst\_stride + 16;
478. **for** (i = 0; i < c->vChrBufSize; i++) {
479. FF\_ALLOC\_OR\_GOTO(c, c->chrUPixBuf[i + c->vChrBufSize],
480. dst\_stride \* 2 + 32, fail);
481. c->chrUPixBuf[i] = c->chrUPixBuf[i + c->vChrBufSize];
482. c->chrVPixBuf[i] = c->chrVPixBuf[i + c->vChrBufSize]
483. = c->chrUPixBuf[i] + (dst\_stride >> 1) + 8;
484. }
485. **if** (CONFIG\_SWSCALE\_ALPHA && c->alpPixBuf)
486. **for** (i = 0; i < c->vLumBufSize; i++) {
487. FF\_ALLOCZ\_OR\_GOTO(c, c->alpPixBuf[i + c->vLumBufSize],
488. dst\_stride + 16, fail);
489. c->alpPixBuf[i] = c->alpPixBuf[i + c->vLumBufSize];
490. }
492. // try to avoid drawing green stuff between the right end and the stride end
493. **for** (i = 0; i < c->vChrBufSize; i++)
494. **if**(desc\_dst->comp[0].depth\_minus1 == 15){
495. av\_assert0(c->dstBpc > 14);
496. **for**(j=0; j<dst\_stride/2+1; j++)
497. ((int32\_t\*)(c->chrUPixBuf[i]))[j] = 1<<18;
498. } **else**
499. **for**(j=0; j<dst\_stride+1; j++)
500. ((int16\_t\*)(c->chrUPixBuf[i]))[j] = 1<<14;
502. av\_assert0(c->chrDstH <= dstH);
503. //是否要输出
504. **if** (flags & SWS\_PRINT\_INFO) {
505. **const** **char** \*scaler = NULL, \*cpucaps;
507. **for** (i = 0; i < FF\_ARRAY\_ELEMS(scale\_algorithms); i++) {
508. **if** (flags & scale\_algorithms[i].flag) {
509. scaler = scale\_algorithms[i].description;
510. **break**;
511. }
512. }
513. **if** (!scaler)
514. scaler =  "ehh flags invalid?!";
515. av\_log(c, AV\_LOG\_INFO, "%s scaler, from %s to %s%s ",
516. scaler,
517. av\_get\_pix\_fmt\_name(srcFormat),
518. #ifdef DITHER1XBPP
519. dstFormat == AV\_PIX\_FMT\_BGR555   || dstFormat == AV\_PIX\_FMT\_BGR565   ||
520. dstFormat == AV\_PIX\_FMT\_RGB444BE || dstFormat == AV\_PIX\_FMT\_RGB444LE ||
521. dstFormat == AV\_PIX\_FMT\_BGR444BE || dstFormat == AV\_PIX\_FMT\_BGR444LE ?
522. "dithered " : "",
523. #else
524. "",
525. #endif
526. av\_get\_pix\_fmt\_name(dstFormat));
528. **if** (INLINE\_MMXEXT(cpu\_flags))
529. cpucaps = "MMXEXT";
530. **else** **if** (INLINE\_AMD3DNOW(cpu\_flags))
531. cpucaps = "3DNOW";
532. **else** **if** (INLINE\_MMX(cpu\_flags))
533. cpucaps = "MMX";
534. **else** **if** (PPC\_ALTIVEC(cpu\_flags))
535. cpucaps = "AltiVec";
536. **else**
537. cpucaps = "C";
539. av\_log(c, AV\_LOG\_INFO, "using %s\n", cpucaps);
541. av\_log(c, AV\_LOG\_VERBOSE, "%dx%d -> %dx%d\n", srcW, srcH, dstW, dstH);
542. av\_log(c, AV\_LOG\_DEBUG,
543. "lum srcW=%d srcH=%d dstW=%d dstH=%d xInc=%d yInc=%d\n",
544. c->srcW, c->srcH, c->dstW, c->dstH, c->lumXInc, c->lumYInc);
545. av\_log(c, AV\_LOG\_DEBUG,
546. "chr srcW=%d srcH=%d dstW=%d dstH=%d xInc=%d yInc=%d\n",
547. c->chrSrcW, c->chrSrcH, c->chrDstW, c->chrDstH,
548. c->chrXInc, c->chrYInc);
549. }
551. /\* unscaled special cases \*/
552. //不拉伸的情况
553. **if** (unscaled && !usesHFilter && !usesVFilter &&
554. (c->srcRange == c->dstRange || isAnyRGB(dstFormat))) {
555. //不许拉伸的情况下，初始化相应的函数
556. ff\_get\_unscaled\_swscale(c);
558. **if** (c->swscale) {
559. **if** (flags & SWS\_PRINT\_INFO)
560. av\_log(c, AV\_LOG\_INFO,
561. "using unscaled %s -> %s special converter\n",
562. av\_get\_pix\_fmt\_name(srcFormat), av\_get\_pix\_fmt\_name(dstFormat));
563. **return** 0;
564. }
565. }
566. //关键：设置SwsContext中的swscale()指针
567. c->swscale = ff\_getSwsFunc(c);
568. **return** 0;
569. fail: // FIXME replace things by appropriate error codes
570. **if** (ret == RETCODE\_USE\_CASCADE)  {
571. **int** tmpW = sqrt(srcW \* (int64\_t)dstW);
572. **int** tmpH = sqrt(srcH \* (int64\_t)dstH);
573. **enum** AVPixelFormat tmpFormat = AV\_PIX\_FMT\_YUV420P;
575. **if** (srcW\*(int64\_t)srcH <= 4LL\*dstW\*dstH)
576. **return** AVERROR(EINVAL);
578. ret = av\_image\_alloc(c->cascaded\_tmp, c->cascaded\_tmpStride,
579. tmpW, tmpH, tmpFormat, 64);
580. **if** (ret < 0)
581. **return** ret;
583. c->cascaded\_context[0] = sws\_getContext(srcW, srcH, srcFormat,
584. tmpW, tmpH, tmpFormat,
585. flags, srcFilter, NULL, c->param);
586. **if** (!c->cascaded\_context[0])
587. **return** -1;
589. c->cascaded\_context[1] = sws\_getContext(tmpW, tmpH, tmpFormat,
590. dstW, dstH, dstFormat,
591. flags, NULL, dstFilter, c->param);
592. **if** (!c->cascaded\_context[1])
593. **return** -1;
594. **return** 0;
595. }
596. **return** -1;
597. }

sws\_init\_context()除了对SwsContext中的各种变量进行赋值之外，主要按照顺序完成了以下一些工作：

1. 通过sws\_rgb2rgb\_init()初始化RGB转RGB（或者YUV转YUV）的函数（注意不包含RGB与YUV相互转换的函数）。  
2. 通过判断输入输出图像的宽高来判断图像是否需要拉伸。如果图像需要拉伸，那么unscaled变量会被标记为1。  
3. 通过sws\_setColorspaceDetails()初始化颜色空间。  
4. 一些输入参数的检测。例如：如果没有设置图像拉伸方法的话，默认设置为SWS\_BICUBIC；如果输入和输出图像的宽高小于等于0的话，也会返回错误信息。  
5. 初始化Filter。这一步根据拉伸方法的不同，初始化不同的Filter。  
6. 如果flags中设置了“打印信息”选项SWS\_PRINT\_INFO，则输出信息。  
7. 如果不需要拉伸的话，调用ff\_get\_unscaled\_swscale()将特定的像素转换函数的指针赋值给SwsContext中的swscale指针。  
8. 如果需要拉伸的话，调用ff\_getSwsFunc()将通用的swscale()赋值给SwsContext中的swscale指针（这个地方有点绕，但是确实是这样的）。

下面分别记录一下上述步骤的实现。

**1.初始化RGB转RGB（或者YUV转YUV）的函数。注意这部分函数不包含RGB与YUV相互转换的函数。**

### sws\_rgb2rgb\_init()

sws\_rgb2rgb\_init()的定义位于libswscale\rgb2rgb.c，如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44305697) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44305697)

[在CODE上查看代码片](https://code.csdn.net/snippets/621402)

1. av\_cold **void** sws\_rgb2rgb\_init(**void**){
2. rgb2rgb\_init\_c();
3. **if** (ARCH\_X86)
4. rgb2rgb\_init\_x86();
5. }

从sws\_rgb2rgb\_init()代码中可以看出，有两个初始化函数：rgb2rgb\_init\_c()是初始化C语言版本的RGB互转（或者YUV互转）的函数，rgb2rgb\_init\_x86()则是初始化X86汇编版本的RGB互转的函数。  
PS：在libswscale中有一点需要注意：很多的函数名称中包含类似“\_c”这样的字符串，代表了该函数是C语言写的。与之对应的还有其它标记，比如“\_mmx”，“sse2”等。

#### rgb2rgb\_init\_c()

首先来看一下C语言版本的RGB互转函数的初始化函数rgb2rgb\_init\_c()，定义位于libswscale\rgb2rgb\_template.c，如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44305697) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44305697)

[在CODE上查看代码片](https://code.csdn.net/snippets/621402)

1. **static** av\_cold **void** rgb2rgb\_init\_c(**void**)
2. {
3. rgb15to16          = rgb15to16\_c;
4. rgb15tobgr24       = rgb15tobgr24\_c;
5. rgb15to32          = rgb15to32\_c;
6. rgb16tobgr24       = rgb16tobgr24\_c;
7. rgb16to32          = rgb16to32\_c;
8. rgb16to15          = rgb16to15\_c;
9. rgb24tobgr16       = rgb24tobgr16\_c;
10. rgb24tobgr15       = rgb24tobgr15\_c;
11. rgb24tobgr32       = rgb24tobgr32\_c;
12. rgb32to16          = rgb32to16\_c;
13. rgb32to15          = rgb32to15\_c;
14. rgb32tobgr24       = rgb32tobgr24\_c;
15. rgb24to15          = rgb24to15\_c;
16. rgb24to16          = rgb24to16\_c;
17. rgb24tobgr24       = rgb24tobgr24\_c;
18. shuffle\_bytes\_2103 = shuffle\_bytes\_2103\_c;
19. rgb32tobgr16       = rgb32tobgr16\_c;
20. rgb32tobgr15       = rgb32tobgr15\_c;
21. yv12toyuy2         = yv12toyuy2\_c;
22. yv12touyvy         = yv12touyvy\_c;
23. yuv422ptoyuy2      = yuv422ptoyuy2\_c;
24. yuv422ptouyvy      = yuv422ptouyvy\_c;
25. yuy2toyv12         = yuy2toyv12\_c;
26. planar2x           = planar2x\_c;
27. ff\_rgb24toyv12     = ff\_rgb24toyv12\_c;
28. interleaveBytes    = interleaveBytes\_c;
29. deinterleaveBytes  = deinterleaveBytes\_c;
30. vu9\_to\_vu12        = vu9\_to\_vu12\_c;
31. yvu9\_to\_yuy2       = yvu9\_to\_yuy2\_c;
33. uyvytoyuv420       = uyvytoyuv420\_c;
34. uyvytoyuv422       = uyvytoyuv422\_c;
35. yuyvtoyuv420       = yuyvtoyuv420\_c;
36. yuyvtoyuv422       = yuyvtoyuv422\_c;
37. }

可以看出rgb2rgb\_init\_c()执行后，会把[**C语言**](http://lib.csdn.net/base/c)版本的图像格式转换函数赋值给系统的函数指针。

下面我们选择几个函数看一下这些转换函数的定义。

#### rgb24tobgr24\_c()

rgb24tobgr24\_c()完成了RGB24向BGR24格式的转换。函数的定义如下所示。从代码中可以看出，该函数实现了“R”与“B”之间位置的对调，从而完成了这两种格式之间的转换。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44305697) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44305697)

[在CODE上查看代码片](https://code.csdn.net/snippets/621402)

1. **static** **inline** **void** rgb24tobgr24\_c(**const** uint8\_t \*src, uint8\_t \*dst, **int** src\_size)
2. {
3. unsigned i;
5. **for** (i = 0; i < src\_size; i += 3) {
6. **register** uint8\_t x = src[i + 2];
7. dst[i + 1]         = src[i + 1];
8. dst[i + 2]         = src[i + 0];
9. dst[i + 0]         = x;
10. }
11. }

#### rgb24to16\_c()

rgb24to16\_c()完成了RGB24向RGB16像素格式的转换。函数的定义如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44305697) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44305697)

[在CODE上查看代码片](https://code.csdn.net/snippets/621402)

1. **static** **inline** **void** rgb24to16\_c(**const** uint8\_t \*src, uint8\_t \*dst, **int** src\_size)
2. {
3. uint16\_t \*d        = (uint16\_t \*)dst;
4. **const** uint8\_t \*s   = src;
5. **const** uint8\_t \*end = s + src\_size;
7. **while** (s < end) {
8. **const** **int** r = \*s++;
9. **const** **int** g = \*s++;
10. **const** **int** b = \*s++;
11. \*d++        = (b >> 3) | ((g & 0xFC) << 3) | ((r & 0xF8) << 8);
12. }
13. }

#### yuyvtoyuv422\_c()

yuyvtoyuv422\_c()完成了YUYV向YUV422像素格式的转换。函数的定义如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44305697) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44305697)

[在CODE上查看代码片](https://code.csdn.net/snippets/621402)

1. **static** **void** yuyvtoyuv422\_c(uint8\_t \*ydst, uint8\_t \*udst, uint8\_t \*vdst,
2. **const** uint8\_t \*src, **int** width, **int** height,
3. **int** lumStride, **int** chromStride, **int** srcStride)
4. {
5. **int** y;
6. **const** **int** chromWidth = FF\_CEIL\_RSHIFT(width, 1);
8. **for** (y = 0; y < height; y++) {
9. extract\_even\_c(src, ydst, width);
10. extract\_odd2\_c(src, udst, vdst, chromWidth);
12. src  += srcStride;
13. ydst += lumStride;
14. udst += chromStride;
15. vdst += chromStride;
16. }
17. }

该函数将YUYV像素数据分离成为Y，U，V三个分量的像素数据。其中extract\_even\_c()用于获取一行像素中序数为偶数的像素，对应提取了YUYV像素格式中的“Y”。extract\_odd2\_c()用于获取一行像素中序数为奇数的像素，并且把这些像素值再次按照奇偶的不同，存储于两个数组中。对应提取了YUYV像素格式中的“U”和“V”。  
extract\_even\_c()定义如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44305697) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44305697)

[在CODE上查看代码片](https://code.csdn.net/snippets/621402)

1. **static** **void** extract\_even\_c(**const** uint8\_t \*src, uint8\_t \*dst, **int** count)
2. {
3. dst   +=  count;
4. src   +=  count \* 2;
5. count  = -count;
6. **while** (count < 0) {
7. dst[count] = src[2 \* count];
8. count++;
9. }
10. }

extract\_odd2\_c()定义如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44305697) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44305697)

[在CODE上查看代码片](https://code.csdn.net/snippets/621402)

1. **static** **void** extract\_even2\_c(**const** uint8\_t \*src, uint8\_t \*dst0, uint8\_t \*dst1,
2. **int** count)
3. {
4. dst0  +=  count;
5. dst1  +=  count;
6. src   +=  count \* 4;
7. count  = -count;
8. **while** (count < 0) {
9. dst0[count] = src[4 \* count + 0];
10. dst1[count] = src[4 \* count + 2];
11. count++;
12. }
13. }

### rgb2rgb\_init\_x86()

rgb2rgb\_init\_x86()用于初始化基于X86汇编语言的RGB互转的代码。由于对汇编不是很熟，不再作详细分析，出于和rgb2rgb\_init\_c()相对比的目的，列出它的代码。它的代码位于libswscale\x86\rgb2rgb.c，如下所示。

PS：所有和汇编有关的代码都位于libswscale目录的x86子目录下。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44305697) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44305697)

[在CODE上查看代码片](https://code.csdn.net/snippets/621402)

1. av\_cold **void** rgb2rgb\_init\_x86(**void**)
2. {
3. #if HAVE\_INLINE\_ASM
4. **int** cpu\_flags = av\_get\_cpu\_flags();
6. **if** (INLINE\_MMX(cpu\_flags))
7. rgb2rgb\_init\_mmx();
8. **if** (INLINE\_AMD3DNOW(cpu\_flags))
9. rgb2rgb\_init\_3dnow();
10. **if** (INLINE\_MMXEXT(cpu\_flags))
11. rgb2rgb\_init\_mmxext();
12. **if** (INLINE\_SSE2(cpu\_flags))
13. rgb2rgb\_init\_sse2();
14. **if** (INLINE\_AVX(cpu\_flags))
15. rgb2rgb\_init\_avx();
16. #endif /\* HAVE\_INLINE\_ASM \*/
17. }

可以看出，rgb2rgb\_init\_x86()首先调用了av\_get\_cpu\_flags()获取CPU支持的特性，根据特性调用rgb2rgb\_init\_mmx()，rgb2rgb\_init\_3dnow()，rgb2rgb\_init\_mmxext()，rgb2rgb\_init\_sse2()，rgb2rgb\_init\_avx()等函数。

**2.判断图像是否需要拉伸。**

这一步主要通过比较输入图像和输出图像的宽高实现。系统使用一个unscaled变量记录图像是否需要拉伸，如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44305697) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44305697)

[在CODE上查看代码片](https://code.csdn.net/snippets/621402)

1. unscaled = (srcW == dstW && srcH == dstH);

**3.初始化颜色空间。**

初始化颜色空间通过函数sws\_setColorspaceDetails()完成。sws\_setColorspaceDetails()是FFmpeg的一个API函数，它的声明如下所示：

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44305697) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44305697)

[在CODE上查看代码片](https://code.csdn.net/snippets/621402)

1. /\*\*
2. \* @param dstRange flag indicating the while-black range of the output (1=jpeg / 0=mpeg)
3. \* @param srcRange flag indicating the while-black range of the input (1=jpeg / 0=mpeg)
4. \* @param table the yuv2rgb coefficients describing the output yuv space, normally ff\_yuv2rgb\_coeffs[x]
5. \* @param inv\_table the yuv2rgb coefficients describing the input yuv space, normally ff\_yuv2rgb\_coeffs[x]
6. \* @param brightness 16.16 fixed point brightness correction
7. \* @param contrast 16.16 fixed point contrast correction
8. \* @param saturation 16.16 fixed point saturation correction
9. \* @return -1 if not supported
10. \*/
11. **int** sws\_setColorspaceDetails(**struct** SwsContext \*c, **const** **int** inv\_table[4],
12. **int** srcRange, **const** **int** table[4], **int** dstRange,
13. **int** brightness, **int** contrast, **int** saturation);

简单解释一下几个参数的含义：

c：需要设定的SwsContext。  
inv\_table：描述输出YUV颜色空间的参数表。  
srcRange：输入图像的取值范围（“1”代表JPEG标准，取值范围是0-255；“0”代表MPEG标准，取值范围是16-235）。  
table：描述输入YUV颜色空间的参数表。  
dstRange：输出图像的取值范围。  
brightness：未研究。  
contrast：未研究。  
saturation：未研究。

如果返回-1代表设置不成功。  
其中描述颜色空间的参数表可以通过sws\_getCoefficients()获取。该函数在后文中再详细记录。  
sws\_setColorspaceDetails()的定义位于libswscale\utils.c，如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44305697) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44305697)

[在CODE上查看代码片](https://code.csdn.net/snippets/621402)

1. **int** sws\_setColorspaceDetails(**struct** SwsContext \*c, **const** **int** inv\_table[4],
2. **int** srcRange, **const** **int** table[4], **int** dstRange,
3. **int** brightness, **int** contrast, **int** saturation)
4. {
5. **const** AVPixFmtDescriptor \*desc\_dst;
6. **const** AVPixFmtDescriptor \*desc\_src;
7. **int** need\_reinit = 0;
8. memmove(c->srcColorspaceTable, inv\_table, **sizeof**(**int**) \* 4);
9. memmove(c->dstColorspaceTable, table, **sizeof**(**int**) \* 4);
11. handle\_formats(c);
12. desc\_dst = av\_pix\_fmt\_desc\_get(c->dstFormat);
13. desc\_src = av\_pix\_fmt\_desc\_get(c->srcFormat);
15. **if**(!isYUV(c->dstFormat) && !isGray(c->dstFormat))
16. dstRange = 0;
17. **if**(!isYUV(c->srcFormat) && !isGray(c->srcFormat))
18. srcRange = 0;
20. c->brightness = brightness;
21. c->contrast   = contrast;
22. c->saturation = saturation;
23. **if** (c->srcRange != srcRange || c->dstRange != dstRange)
24. need\_reinit = 1;
25. c->srcRange   = srcRange;
26. c->dstRange   = dstRange;
28. //The srcBpc check is possibly wrong but we seem to lack a definitive reference to test this
29. //and what we have in ticket 2939 looks better with this check
30. **if** (need\_reinit && (c->srcBpc == 8 || !isYUV(c->srcFormat)))
31. ff\_sws\_init\_range\_convert(c);
33. **if** ((isYUV(c->dstFormat) || isGray(c->dstFormat)) && (isYUV(c->srcFormat) || isGray(c->srcFormat)))
34. **return** -1;
36. c->dstFormatBpp = av\_get\_bits\_per\_pixel(desc\_dst);
37. c->srcFormatBpp = av\_get\_bits\_per\_pixel(desc\_src);
39. **if** (!isYUV(c->dstFormat) && !isGray(c->dstFormat)) {
40. ff\_yuv2rgb\_c\_init\_tables(c, inv\_table, srcRange, brightness,
41. contrast, saturation);
42. // FIXME factorize
44. **if** (ARCH\_PPC)
45. ff\_yuv2rgb\_init\_tables\_ppc(c, inv\_table, brightness,
46. contrast, saturation);
47. }
49. fill\_rgb2yuv\_table(c, table, dstRange);
51. **return** 0;
52. }

从sws\_setColorspaceDetails()定义中可以看出，该函数将输入的参数分别赋值给了相应的变量，并且在最后调用了一个函数fill\_rgb2yuv\_table()。fill\_rgb2yuv\_table()函数还没有弄懂，暂时不记录。

### sws\_getCoefficients()

sws\_getCoefficients()用于获取描述颜色空间的参数表。它的声明如下。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44305697) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44305697)

[在CODE上查看代码片](https://code.csdn.net/snippets/621402)

1. /\*\*
2. \* Return a pointer to yuv<->rgb coefficients for the given colorspace
3. \* suitable for sws\_setColorspaceDetails().
4. \*
5. \* @param colorspace One of the SWS\_CS\_\* macros. If invalid,
6. \* SWS\_CS\_DEFAULT is used.
7. \*/
8. **const** **int** \*sws\_getCoefficients(**int** colorspace);

其中colorspace可以取值如下变量。默认的取值SWS\_CS\_DEFAULT等同于SWS\_CS\_ITU601或者SWS\_CS\_SMPTE170M。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44305697) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44305697)

[在CODE上查看代码片](https://code.csdn.net/snippets/621402)

1. #define SWS\_CS\_ITU709         1
2. #define SWS\_CS\_FCC            4
3. #define SWS\_CS\_ITU601         5
4. #define SWS\_CS\_ITU624         5
5. #define SWS\_CS\_SMPTE170M      5
6. #define SWS\_CS\_SMPTE240M      7
7. #define SWS\_CS\_DEFAULT        5

下面看一下sws\_getCoefficients()的定义，位于libswscale\yuv2rgb.c，如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44305697) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44305697)

[在CODE上查看代码片](https://code.csdn.net/snippets/621402)

1. **const** **int** \*sws\_getCoefficients(**int** colorspace)
2. {
3. **if** (colorspace > 7 || colorspace < 0)
4. colorspace = SWS\_CS\_DEFAULT;
5. **return** ff\_yuv2rgb\_coeffs[colorspace];
6. }

可以看出它返回了一个名称为ff\_yuv2rgb\_coeffs的数组中的一个元素，该数组的定义如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44305697) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44305697)

[在CODE上查看代码片](https://code.csdn.net/snippets/621402)

1. **const** int32\_t ff\_yuv2rgb\_coeffs[8][4] = {
2. { 117504, 138453, 13954, 34903 }, /\* no sequence\_display\_extension \*/
3. { 117504, 138453, 13954, 34903 }, /\* ITU-R Rec. 709 (1990) \*/
4. { 104597, 132201, 25675, 53279 }, /\* unspecified \*/
5. { 104597, 132201, 25675, 53279 }, /\* reserved \*/
6. { 104448, 132798, 24759, 53109 }, /\* FCC \*/
7. { 104597, 132201, 25675, 53279 }, /\* ITU-R Rec. 624-4 System B, G \*/
8. { 104597, 132201, 25675, 53279 }, /\* SMPTE 170M \*/
9. { 117579, 136230, 16907, 35559 }  /\* SMPTE 240M (1987) \*/
10. };

**4.一些输入参数的检测。**

例如：如果没有设置图像拉伸方法的话，默认设置为SWS\_BICUBIC；如果输入和输出图像的宽高小于等于0的话，也会返回错误信息。有关这方面的代码比较多，简单举个例子。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44305697) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44305697)

[在CODE上查看代码片](https://code.csdn.net/snippets/621402)

1. i = flags & (SWS\_POINT         |
2. SWS\_AREA          |
3. SWS\_BILINEAR      |
4. SWS\_FAST\_BILINEAR |
5. SWS\_BICUBIC       |
6. SWS\_X             |
7. SWS\_GAUSS         |
8. SWS\_LANCZOS       |
9. SWS\_SINC          |
10. SWS\_SPLINE        |
11. SWS\_BICUBLIN);
13. /\* provide a default scaler if not set by caller \*/
14. **if** (!i) {
15. **if** (dstW < srcW && dstH < srcH)
16. flags |= SWS\_BICUBIC;
17. **else** **if** (dstW > srcW && dstH > srcH)
18. flags |= SWS\_BICUBIC;
19. **else**
20. flags |= SWS\_BICUBIC;
21. c->flags = flags;
22. } **else** **if** (i & (i - 1)) {
23. av\_log(c, AV\_LOG\_ERROR,
24. "Exactly one scaler algorithm must be chosen, got %X\n", i);
25. **return** AVERROR(EINVAL);
26. }
27. /\* sanity check \*/
28. **if** (srcW < 1 || srcH < 1 || dstW < 1 || dstH < 1) {
29. /\* FIXME check if these are enough and try to lower them after
30. \* fixing the relevant parts of the code \*/
31. av\_log(c, AV\_LOG\_ERROR, "%dx%d -> %dx%d is invalid scaling dimension\n",
32. srcW, srcH, dstW, dstH);
33. **return** AVERROR(EINVAL);
34. }

**5.初始化Filter。这一步根据拉伸方法的不同，初始化不同的Filter。**

这一部分的工作在函数initFilter()中完成，暂时不详细分析。

**6.如果flags中设置了“打印信息”选项SWS\_PRINT\_INFO，则输出信息。**

SwsContext初始化的时候，可以给flags设置SWS\_PRINT\_INFO标记。这样SwsContext初始化完成的时候就可以打印出一些配置信息。与打印相关的代码如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44305697) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44305697)

[在CODE上查看代码片](https://code.csdn.net/snippets/621402)

1. **if** (flags & SWS\_PRINT\_INFO) {
2. **const** **char** \*scaler = NULL, \*cpucaps;
4. **for** (i = 0; i < FF\_ARRAY\_ELEMS(scale\_algorithms); i++) {
5. **if** (flags & scale\_algorithms[i].flag) {
6. scaler = scale\_algorithms[i].description;
7. **break**;
8. }
9. }
10. **if** (!scaler)
11. scaler =  "ehh flags invalid?!";
12. av\_log(c, AV\_LOG\_INFO, "%s scaler, from %s to %s%s ",
13. scaler,
14. av\_get\_pix\_fmt\_name(srcFormat),
15. #ifdef DITHER1XBPP
16. dstFormat == AV\_PIX\_FMT\_BGR555   || dstFormat == AV\_PIX\_FMT\_BGR565   ||
17. dstFormat == AV\_PIX\_FMT\_RGB444BE || dstFormat == AV\_PIX\_FMT\_RGB444LE ||
18. dstFormat == AV\_PIX\_FMT\_BGR444BE || dstFormat == AV\_PIX\_FMT\_BGR444LE ?
19. "dithered " : "",
20. #else
21. "",
22. #endif
23. av\_get\_pix\_fmt\_name(dstFormat));
25. **if** (INLINE\_MMXEXT(cpu\_flags))
26. cpucaps = "MMXEXT";
27. **else** **if** (INLINE\_AMD3DNOW(cpu\_flags))
28. cpucaps = "3DNOW";
29. **else** **if** (INLINE\_MMX(cpu\_flags))
30. cpucaps = "MMX";
31. **else** **if** (PPC\_ALTIVEC(cpu\_flags))
32. cpucaps = "AltiVec";
33. **else**
34. cpucaps = "C";
36. av\_log(c, AV\_LOG\_INFO, "using %s\n", cpucaps);
38. av\_log(c, AV\_LOG\_VERBOSE, "%dx%d -> %dx%d\n", srcW, srcH, dstW, dstH);
39. av\_log(c, AV\_LOG\_DEBUG,
40. "lum srcW=%d srcH=%d dstW=%d dstH=%d xInc=%d yInc=%d\n",
41. c->srcW, c->srcH, c->dstW, c->dstH, c->lumXInc, c->lumYInc);
42. av\_log(c, AV\_LOG\_DEBUG,
43. "chr srcW=%d srcH=%d dstW=%d dstH=%d xInc=%d yInc=%d\n",
44. c->chrSrcW, c->chrSrcH, c->chrDstW, c->chrDstH,
45. c->chrXInc, c->chrYInc);
46. }

**7.如果不需要拉伸的话，就会调用ff\_get\_unscaled\_swscale()将特定的像素转换函数的指针赋值给SwsContext中的swscale指针。**

### ff\_get\_unscaled\_swscale()

ff\_get\_unscaled\_swscale()的定义如下所示。该函数根据输入图像像素格式和输出图像像素格式，选择不同的像素格式转换函数。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44305697) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44305697)

[在CODE上查看代码片](https://code.csdn.net/snippets/621402)

1. **void** ff\_get\_unscaled\_swscale(SwsContext \*c)
2. {
3. **const** **enum** AVPixelFormat srcFormat = c->srcFormat;
4. **const** **enum** AVPixelFormat dstFormat = c->dstFormat;
5. **const** **int** flags = c->flags;
6. **const** **int** dstH = c->dstH;
7. **int** needsDither;
9. needsDither = isAnyRGB(dstFormat) &&
10. c->dstFormatBpp < 24 &&
11. (c->dstFormatBpp < c->srcFormatBpp || (!isAnyRGB(srcFormat)));
13. /\* yv12\_to\_nv12 \*/
14. **if** ((srcFormat == AV\_PIX\_FMT\_YUV420P || srcFormat == AV\_PIX\_FMT\_YUVA420P) &&
15. (dstFormat == AV\_PIX\_FMT\_NV12 || dstFormat == AV\_PIX\_FMT\_NV21)) {
16. c->swscale = planarToNv12Wrapper;
17. }
18. /\* nv12\_to\_yv12 \*/
19. **if** (dstFormat == AV\_PIX\_FMT\_YUV420P &&
20. (srcFormat == AV\_PIX\_FMT\_NV12 || srcFormat == AV\_PIX\_FMT\_NV21)) {
21. c->swscale = nv12ToPlanarWrapper;
22. }
23. /\* yuv2bgr \*/
24. **if** ((srcFormat == AV\_PIX\_FMT\_YUV420P || srcFormat == AV\_PIX\_FMT\_YUV422P ||
25. srcFormat == AV\_PIX\_FMT\_YUVA420P) && isAnyRGB(dstFormat) &&
26. !(flags & SWS\_ACCURATE\_RND) && (c->dither == SWS\_DITHER\_BAYER || c->dither == SWS\_DITHER\_AUTO) && !(dstH & 1)) {
27. c->swscale = ff\_yuv2rgb\_get\_func\_ptr(c);
28. }
30. **if** (srcFormat == AV\_PIX\_FMT\_YUV410P && !(dstH & 3) &&
31. (dstFormat == AV\_PIX\_FMT\_YUV420P || dstFormat == AV\_PIX\_FMT\_YUVA420P) &&
32. !(flags & SWS\_BITEXACT)) {
33. c->swscale = yvu9ToYv12Wrapper;
34. }
36. /\* bgr24toYV12 \*/
37. **if** (srcFormat == AV\_PIX\_FMT\_BGR24 &&
38. (dstFormat == AV\_PIX\_FMT\_YUV420P || dstFormat == AV\_PIX\_FMT\_YUVA420P) &&
39. !(flags & SWS\_ACCURATE\_RND))
40. c->swscale = bgr24ToYv12Wrapper;
42. /\* RGB/BGR -> RGB/BGR (no dither needed forms) \*/
43. **if** (isAnyRGB(srcFormat) && isAnyRGB(dstFormat) && findRgbConvFn(c)
44. && (!needsDither || (c->flags&(SWS\_FAST\_BILINEAR|SWS\_POINT))))
45. c->swscale = rgbToRgbWrapper;
47. **if** ((srcFormat == AV\_PIX\_FMT\_GBRP && dstFormat == AV\_PIX\_FMT\_GBRAP) ||
48. (srcFormat == AV\_PIX\_FMT\_GBRAP && dstFormat == AV\_PIX\_FMT\_GBRP))
49. c->swscale = planarRgbToplanarRgbWrapper;
51. #define isByteRGB(f) (             \
52. f == AV\_PIX\_FMT\_RGB32   || \
53. f == AV\_PIX\_FMT\_RGB32\_1 || \
54. f == AV\_PIX\_FMT\_RGB24   || \
55. f == AV\_PIX\_FMT\_BGR32   || \
56. f == AV\_PIX\_FMT\_BGR32\_1 || \
57. f == AV\_PIX\_FMT\_BGR24)
59. **if** (srcFormat == AV\_PIX\_FMT\_GBRP && isPlanar(srcFormat) && isByteRGB(dstFormat))
60. c->swscale = planarRgbToRgbWrapper;
62. **if** ((srcFormat == AV\_PIX\_FMT\_RGB48LE  || srcFormat == AV\_PIX\_FMT\_RGB48BE  ||
63. srcFormat == AV\_PIX\_FMT\_BGR48LE  || srcFormat == AV\_PIX\_FMT\_BGR48BE  ||
64. srcFormat == AV\_PIX\_FMT\_RGBA64LE || srcFormat == AV\_PIX\_FMT\_RGBA64BE ||
65. srcFormat == AV\_PIX\_FMT\_BGRA64LE || srcFormat == AV\_PIX\_FMT\_BGRA64BE) &&
66. (dstFormat == AV\_PIX\_FMT\_GBRP9LE  || dstFormat == AV\_PIX\_FMT\_GBRP9BE  ||
67. dstFormat == AV\_PIX\_FMT\_GBRP10LE || dstFormat == AV\_PIX\_FMT\_GBRP10BE ||
68. dstFormat == AV\_PIX\_FMT\_GBRP12LE || dstFormat == AV\_PIX\_FMT\_GBRP12BE ||
69. dstFormat == AV\_PIX\_FMT\_GBRP14LE || dstFormat == AV\_PIX\_FMT\_GBRP14BE ||
70. dstFormat == AV\_PIX\_FMT\_GBRP16LE || dstFormat == AV\_PIX\_FMT\_GBRP16BE ||
71. dstFormat == AV\_PIX\_FMT\_GBRAP16LE || dstFormat == AV\_PIX\_FMT\_GBRAP16BE ))
72. c->swscale = Rgb16ToPlanarRgb16Wrapper;
74. **if** ((srcFormat == AV\_PIX\_FMT\_GBRP9LE  || srcFormat == AV\_PIX\_FMT\_GBRP9BE  ||
75. srcFormat == AV\_PIX\_FMT\_GBRP16LE || srcFormat == AV\_PIX\_FMT\_GBRP16BE ||
76. srcFormat == AV\_PIX\_FMT\_GBRP10LE || srcFormat == AV\_PIX\_FMT\_GBRP10BE ||
77. srcFormat == AV\_PIX\_FMT\_GBRP12LE || srcFormat == AV\_PIX\_FMT\_GBRP12BE ||
78. srcFormat == AV\_PIX\_FMT\_GBRP14LE || srcFormat == AV\_PIX\_FMT\_GBRP14BE ||
79. srcFormat == AV\_PIX\_FMT\_GBRAP16LE || srcFormat == AV\_PIX\_FMT\_GBRAP16BE) &&
80. (dstFormat == AV\_PIX\_FMT\_RGB48LE  || dstFormat == AV\_PIX\_FMT\_RGB48BE  ||
81. dstFormat == AV\_PIX\_FMT\_BGR48LE  || dstFormat == AV\_PIX\_FMT\_BGR48BE  ||
82. dstFormat == AV\_PIX\_FMT\_RGBA64LE || dstFormat == AV\_PIX\_FMT\_RGBA64BE ||
83. dstFormat == AV\_PIX\_FMT\_BGRA64LE || dstFormat == AV\_PIX\_FMT\_BGRA64BE))
84. c->swscale = planarRgb16ToRgb16Wrapper;
86. **if** (av\_pix\_fmt\_desc\_get(srcFormat)->comp[0].depth\_minus1 == 7 &&
87. isPackedRGB(srcFormat) && dstFormat == AV\_PIX\_FMT\_GBRP)
88. c->swscale = rgbToPlanarRgbWrapper;
90. **if** (isBayer(srcFormat)) {
91. **if** (dstFormat == AV\_PIX\_FMT\_RGB24)
92. c->swscale = bayer\_to\_rgb24\_wrapper;
93. **else** **if** (dstFormat == AV\_PIX\_FMT\_YUV420P)
94. c->swscale = bayer\_to\_yv12\_wrapper;
95. **else** **if** (!isBayer(dstFormat)) {
96. av\_log(c, AV\_LOG\_ERROR, "unsupported bayer conversion\n");
97. av\_assert0(0);
98. }
99. }
101. /\* bswap 16 bits per pixel/component packed formats \*/
102. **if** (IS\_DIFFERENT\_ENDIANESS(srcFormat, dstFormat, AV\_PIX\_FMT\_BAYER\_BGGR16) ||
103. IS\_DIFFERENT\_ENDIANESS(srcFormat, dstFormat, AV\_PIX\_FMT\_BAYER\_RGGB16) ||
104. IS\_DIFFERENT\_ENDIANESS(srcFormat, dstFormat, AV\_PIX\_FMT\_BAYER\_GBRG16) ||
105. IS\_DIFFERENT\_ENDIANESS(srcFormat, dstFormat, AV\_PIX\_FMT\_BAYER\_GRBG16) ||
106. IS\_DIFFERENT\_ENDIANESS(srcFormat, dstFormat, AV\_PIX\_FMT\_BGR444) ||
107. IS\_DIFFERENT\_ENDIANESS(srcFormat, dstFormat, AV\_PIX\_FMT\_BGR48)  ||
108. IS\_DIFFERENT\_ENDIANESS(srcFormat, dstFormat, AV\_PIX\_FMT\_BGRA64) ||
109. IS\_DIFFERENT\_ENDIANESS(srcFormat, dstFormat, AV\_PIX\_FMT\_BGR555) ||
110. IS\_DIFFERENT\_ENDIANESS(srcFormat, dstFormat, AV\_PIX\_FMT\_BGR565) ||
111. IS\_DIFFERENT\_ENDIANESS(srcFormat, dstFormat, AV\_PIX\_FMT\_BGRA64) ||
112. IS\_DIFFERENT\_ENDIANESS(srcFormat, dstFormat, AV\_PIX\_FMT\_GRAY16) ||
113. IS\_DIFFERENT\_ENDIANESS(srcFormat, dstFormat, AV\_PIX\_FMT\_YA16)   ||
114. IS\_DIFFERENT\_ENDIANESS(srcFormat, dstFormat, AV\_PIX\_FMT\_GBRP9)  ||
115. IS\_DIFFERENT\_ENDIANESS(srcFormat, dstFormat, AV\_PIX\_FMT\_GBRP10) ||
116. IS\_DIFFERENT\_ENDIANESS(srcFormat, dstFormat, AV\_PIX\_FMT\_GBRP12) ||
117. IS\_DIFFERENT\_ENDIANESS(srcFormat, dstFormat, AV\_PIX\_FMT\_GBRP14) ||
118. IS\_DIFFERENT\_ENDIANESS(srcFormat, dstFormat, AV\_PIX\_FMT\_GBRP16) ||
119. IS\_DIFFERENT\_ENDIANESS(srcFormat, dstFormat, AV\_PIX\_FMT\_GBRAP16) ||
120. IS\_DIFFERENT\_ENDIANESS(srcFormat, dstFormat, AV\_PIX\_FMT\_RGB444) ||
121. IS\_DIFFERENT\_ENDIANESS(srcFormat, dstFormat, AV\_PIX\_FMT\_RGB48)  ||
122. IS\_DIFFERENT\_ENDIANESS(srcFormat, dstFormat, AV\_PIX\_FMT\_RGBA64) ||
123. IS\_DIFFERENT\_ENDIANESS(srcFormat, dstFormat, AV\_PIX\_FMT\_RGB555) ||
124. IS\_DIFFERENT\_ENDIANESS(srcFormat, dstFormat, AV\_PIX\_FMT\_RGB565) ||
125. IS\_DIFFERENT\_ENDIANESS(srcFormat, dstFormat, AV\_PIX\_FMT\_RGBA64) ||
126. IS\_DIFFERENT\_ENDIANESS(srcFormat, dstFormat, AV\_PIX\_FMT\_XYZ12)  ||
127. IS\_DIFFERENT\_ENDIANESS(srcFormat, dstFormat, AV\_PIX\_FMT\_YUV420P9)  ||
128. IS\_DIFFERENT\_ENDIANESS(srcFormat, dstFormat, AV\_PIX\_FMT\_YUV420P10) ||
129. IS\_DIFFERENT\_ENDIANESS(srcFormat, dstFormat, AV\_PIX\_FMT\_YUV420P12) ||
130. IS\_DIFFERENT\_ENDIANESS(srcFormat, dstFormat, AV\_PIX\_FMT\_YUV420P14) ||
131. IS\_DIFFERENT\_ENDIANESS(srcFormat, dstFormat, AV\_PIX\_FMT\_YUV420P16) ||
132. IS\_DIFFERENT\_ENDIANESS(srcFormat, dstFormat, AV\_PIX\_FMT\_YUV422P9)  ||
133. IS\_DIFFERENT\_ENDIANESS(srcFormat, dstFormat, AV\_PIX\_FMT\_YUV422P10) ||
134. IS\_DIFFERENT\_ENDIANESS(srcFormat, dstFormat, AV\_PIX\_FMT\_YUV422P12) ||
135. IS\_DIFFERENT\_ENDIANESS(srcFormat, dstFormat, AV\_PIX\_FMT\_YUV422P14) ||
136. IS\_DIFFERENT\_ENDIANESS(srcFormat, dstFormat, AV\_PIX\_FMT\_YUV422P16) ||
137. IS\_DIFFERENT\_ENDIANESS(srcFormat, dstFormat, AV\_PIX\_FMT\_YUV444P9)  ||
138. IS\_DIFFERENT\_ENDIANESS(srcFormat, dstFormat, AV\_PIX\_FMT\_YUV444P10) ||
139. IS\_DIFFERENT\_ENDIANESS(srcFormat, dstFormat, AV\_PIX\_FMT\_YUV444P12) ||
140. IS\_DIFFERENT\_ENDIANESS(srcFormat, dstFormat, AV\_PIX\_FMT\_YUV444P14) ||
141. IS\_DIFFERENT\_ENDIANESS(srcFormat, dstFormat, AV\_PIX\_FMT\_YUV444P16))
142. c->swscale = packed\_16bpc\_bswap;
144. **if** (usePal(srcFormat) && isByteRGB(dstFormat))
145. c->swscale = palToRgbWrapper;
147. **if** (srcFormat == AV\_PIX\_FMT\_YUV422P) {
148. **if** (dstFormat == AV\_PIX\_FMT\_YUYV422)
149. c->swscale = yuv422pToYuy2Wrapper;
150. **else** **if** (dstFormat == AV\_PIX\_FMT\_UYVY422)
151. c->swscale = yuv422pToUyvyWrapper;
152. }
154. /\* LQ converters if -sws 0 or -sws 4\*/
155. **if** (c->flags&(SWS\_FAST\_BILINEAR|SWS\_POINT)) {
156. /\* yv12\_to\_yuy2 \*/
157. **if** (srcFormat == AV\_PIX\_FMT\_YUV420P || srcFormat == AV\_PIX\_FMT\_YUVA420P) {
158. **if** (dstFormat == AV\_PIX\_FMT\_YUYV422)
159. c->swscale = planarToYuy2Wrapper;
160. **else** **if** (dstFormat == AV\_PIX\_FMT\_UYVY422)
161. c->swscale = planarToUyvyWrapper;
162. }
163. }
164. **if** (srcFormat == AV\_PIX\_FMT\_YUYV422 &&
165. (dstFormat == AV\_PIX\_FMT\_YUV420P || dstFormat == AV\_PIX\_FMT\_YUVA420P))
166. c->swscale = yuyvToYuv420Wrapper;
167. **if** (srcFormat == AV\_PIX\_FMT\_UYVY422 &&
168. (dstFormat == AV\_PIX\_FMT\_YUV420P || dstFormat == AV\_PIX\_FMT\_YUVA420P))
169. c->swscale = uyvyToYuv420Wrapper;
170. **if** (srcFormat == AV\_PIX\_FMT\_YUYV422 && dstFormat == AV\_PIX\_FMT\_YUV422P)
171. c->swscale = yuyvToYuv422Wrapper;
172. **if** (srcFormat == AV\_PIX\_FMT\_UYVY422 && dstFormat == AV\_PIX\_FMT\_YUV422P)
173. c->swscale = uyvyToYuv422Wrapper;
175. #define isPlanarGray(x) (isGray(x) && (x) != AV\_PIX\_FMT\_YA8 && (x) != AV\_PIX\_FMT\_YA16LE && (x) != AV\_PIX\_FMT\_YA16BE)
176. /\* simple copy \*/
177. **if** ( srcFormat == dstFormat ||
178. (srcFormat == AV\_PIX\_FMT\_YUVA420P && dstFormat == AV\_PIX\_FMT\_YUV420P) ||
179. (srcFormat == AV\_PIX\_FMT\_YUV420P && dstFormat == AV\_PIX\_FMT\_YUVA420P) ||
180. (isPlanarYUV(srcFormat) && isPlanarGray(dstFormat)) ||
181. (isPlanarYUV(dstFormat) && isPlanarGray(srcFormat)) ||
182. (isPlanarGray(dstFormat) && isPlanarGray(srcFormat)) ||
183. (isPlanarYUV(srcFormat) && isPlanarYUV(dstFormat) &&
184. c->chrDstHSubSample == c->chrSrcHSubSample &&
185. c->chrDstVSubSample == c->chrSrcVSubSample &&
186. dstFormat != AV\_PIX\_FMT\_NV12 && dstFormat != AV\_PIX\_FMT\_NV21 &&
187. srcFormat != AV\_PIX\_FMT\_NV12 && srcFormat != AV\_PIX\_FMT\_NV21))
188. {
189. **if** (isPacked(c->srcFormat))
190. c->swscale = packedCopyWrapper;
191. **else** /\* Planar YUV or gray \*/
192. c->swscale = planarCopyWrapper;
193. }
195. **if** (ARCH\_PPC)
196. ff\_get\_unscaled\_swscale\_ppc(c);
197. //     if (ARCH\_ARM)
198. //         ff\_get\_unscaled\_swscale\_arm(c);
199. }

从ff\_get\_unscaled\_swscale()源代码中可以看出，赋值给SwsContext的swscale指针的函数名称大多数为XXXWrapper()。实际上这些函数封装了一些基本的像素格式转换函数。例如yuyvToYuv422Wrapper()的定义如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44305697) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44305697)

[在CODE上查看代码片](https://code.csdn.net/snippets/621402)

1. **static** **int** yuyvToYuv422Wrapper(SwsContext \*c, **const** uint8\_t \*src[],
2. **int** srcStride[], **int** srcSliceY, **int** srcSliceH,
3. uint8\_t \*dstParam[], **int** dstStride[])
4. {
5. uint8\_t \*ydst = dstParam[0] + dstStride[0] \* srcSliceY;
6. uint8\_t \*udst = dstParam[1] + dstStride[1] \* srcSliceY;
7. uint8\_t \*vdst = dstParam[2] + dstStride[2] \* srcSliceY;
9. yuyvtoyuv422(ydst, udst, vdst, src[0], c->srcW, srcSliceH, dstStride[0],
10. dstStride[1], srcStride[0]);
12. **return** srcSliceH;
13. }

从yuyvToYuv422Wrapper()的定义中可以看出，它调用了yuyvtoyuv422()。而yuyvtoyuv422()则是rgb2rgb.c中的一个函数，用于将YUVU转换为YUV422（该函数在前文中已经记录）。  
  
**8.如果需要拉伸的话，就会调用ff\_getSwsFunc()将通用的swscale()赋值给SwsContext中的swscale指针，然后返回。**  
上一步骤（图像不用缩放）实际上是一种不太常见的情况，更多的情况下会执行本步骤。这个时候就会调用ff\_getSwsFunc()获取图像的缩放函数。

### ff\_getSwsFunc()

ff\_getSwsFunc()用于获取通用的swscale()函数。该函数的定义如下。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44305697) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44305697)

[在CODE上查看代码片](https://code.csdn.net/snippets/621402)

1. SwsFunc ff\_getSwsFunc(SwsContext \*c)
2. {
3. sws\_init\_swscale(c);
5. **if** (ARCH\_PPC)
6. ff\_sws\_init\_swscale\_ppc(c);
7. **if** (ARCH\_X86)
8. ff\_sws\_init\_swscale\_x86(c);
10. **return** swscale;
11. }

从源代码中可以看出ff\_getSwsFunc()调用了函数sws\_init\_swscale()。如果系统支持X86汇编的话，还会调用ff\_sws\_init\_swscale\_x86()。

### sws\_init\_swscale()

sws\_init\_swscale()的定义位于libswscale\swscale.c，如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44305697) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44305697)

[在CODE上查看代码片](https://code.csdn.net/snippets/621402)

1. **static** av\_cold **void** sws\_init\_swscale(SwsContext \*c)
2. {
3. **enum** AVPixelFormat srcFormat = c->srcFormat;
5. ff\_sws\_init\_output\_funcs(c, &c->yuv2plane1, &c->yuv2planeX,
6. &c->yuv2nv12cX, &c->yuv2packed1,
7. &c->yuv2packed2, &c->yuv2packedX, &c->yuv2anyX);
9. ff\_sws\_init\_input\_funcs(c);

12. **if** (c->srcBpc == 8) {
13. **if** (c->dstBpc <= 14) {
14. c->hyScale = c->hcScale = hScale8To15\_c;
15. **if** (c->flags & SWS\_FAST\_BILINEAR) {
16. c->hyscale\_fast = ff\_hyscale\_fast\_c;
17. c->hcscale\_fast = ff\_hcscale\_fast\_c;
18. }
19. } **else** {
20. c->hyScale = c->hcScale = hScale8To19\_c;
21. }
22. } **else** {
23. c->hyScale = c->hcScale = c->dstBpc > 14 ? hScale16To19\_c
24. : hScale16To15\_c;
25. }
27. ff\_sws\_init\_range\_convert(c);
29. **if** (!(isGray(srcFormat) || isGray(c->dstFormat) ||
30. srcFormat == AV\_PIX\_FMT\_MONOBLACK || srcFormat == AV\_PIX\_FMT\_MONOWHITE))
31. c->needs\_hcscale = 1;
32. }

从函数中可以看出，sws\_init\_swscale()主要调用了3个函数：ff\_sws\_init\_output\_funcs()，ff\_sws\_init\_input\_funcs()，ff\_sws\_init\_range\_convert()。其中，ff\_sws\_init\_output\_funcs()用于初始化输出的函数，ff\_sws\_init\_input\_funcs()用于初始化输入的函数，ff\_sws\_init\_range\_convert()用于初始化像素值范围转换的函数。

### ff\_sws\_init\_output\_funcs()

ff\_sws\_init\_output\_funcs()用于初始化“输出函数”。“输出函数”在libswscale中的作用就是将处理后的一行像素数据输出出来。ff\_sws\_init\_output\_funcs()的定义位于libswscale\output.c，如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44305697) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44305697)

[在CODE上查看代码片](https://code.csdn.net/snippets/621402)

1. av\_cold **void** ff\_sws\_init\_output\_funcs(SwsContext \*c,
2. yuv2planar1\_fn \*yuv2plane1,
3. yuv2planarX\_fn \*yuv2planeX,
4. yuv2interleavedX\_fn \*yuv2nv12cX,
5. yuv2packed1\_fn \*yuv2packed1,
6. yuv2packed2\_fn \*yuv2packed2,
7. yuv2packedX\_fn \*yuv2packedX,
8. yuv2anyX\_fn \*yuv2anyX)
9. {
10. **enum** AVPixelFormat dstFormat = c->dstFormat;
11. **const** AVPixFmtDescriptor \*desc = av\_pix\_fmt\_desc\_get(dstFormat);
13. **if** (is16BPS(dstFormat)) {
14. \*yuv2planeX = isBE(dstFormat) ? yuv2planeX\_16BE\_c  : yuv2planeX\_16LE\_c;
15. \*yuv2plane1 = isBE(dstFormat) ? yuv2plane1\_16BE\_c  : yuv2plane1\_16LE\_c;
16. } **else** **if** (is9\_OR\_10BPS(dstFormat)) {
17. **if** (desc->comp[0].depth\_minus1 == 8) {
18. \*yuv2planeX = isBE(dstFormat) ? yuv2planeX\_9BE\_c  : yuv2planeX\_9LE\_c;
19. \*yuv2plane1 = isBE(dstFormat) ? yuv2plane1\_9BE\_c  : yuv2plane1\_9LE\_c;
20. } **else** **if** (desc->comp[0].depth\_minus1 == 9) {
21. \*yuv2planeX = isBE(dstFormat) ? yuv2planeX\_10BE\_c  : yuv2planeX\_10LE\_c;
22. \*yuv2plane1 = isBE(dstFormat) ? yuv2plane1\_10BE\_c  : yuv2plane1\_10LE\_c;
23. } **else** **if** (desc->comp[0].depth\_minus1 == 11) {
24. \*yuv2planeX = isBE(dstFormat) ? yuv2planeX\_12BE\_c  : yuv2planeX\_12LE\_c;
25. \*yuv2plane1 = isBE(dstFormat) ? yuv2plane1\_12BE\_c  : yuv2plane1\_12LE\_c;
26. } **else** **if** (desc->comp[0].depth\_minus1 == 13) {
27. \*yuv2planeX = isBE(dstFormat) ? yuv2planeX\_14BE\_c  : yuv2planeX\_14LE\_c;
28. \*yuv2plane1 = isBE(dstFormat) ? yuv2plane1\_14BE\_c  : yuv2plane1\_14LE\_c;
29. } **else**
30. av\_assert0(0);
31. } **else** {
32. \*yuv2plane1 = yuv2plane1\_8\_c;
33. \*yuv2planeX = yuv2planeX\_8\_c;
34. **if** (dstFormat == AV\_PIX\_FMT\_NV12 || dstFormat == AV\_PIX\_FMT\_NV21)
35. \*yuv2nv12cX = yuv2nv12cX\_c;
36. }
38. **if**(c->flags & SWS\_FULL\_CHR\_H\_INT) {
39. **switch** (dstFormat) {
40. **case** AV\_PIX\_FMT\_RGBA:
41. #if CONFIG\_SMALL
42. \*yuv2packedX = yuv2rgba32\_full\_X\_c;
43. \*yuv2packed2 = yuv2rgba32\_full\_2\_c;
44. \*yuv2packed1 = yuv2rgba32\_full\_1\_c;
45. #else
46. #if CONFIG\_SWSCALE\_ALPHA
47. **if** (c->alpPixBuf) {
48. \*yuv2packedX = yuv2rgba32\_full\_X\_c;
49. \*yuv2packed2 = yuv2rgba32\_full\_2\_c;
50. \*yuv2packed1 = yuv2rgba32\_full\_1\_c;
51. } **else**
52. #endif /\* CONFIG\_SWSCALE\_ALPHA \*/
53. {
54. \*yuv2packedX = yuv2rgbx32\_full\_X\_c;
55. \*yuv2packed2 = yuv2rgbx32\_full\_2\_c;
56. \*yuv2packed1 = yuv2rgbx32\_full\_1\_c;
57. }
58. #endif /\* !CONFIG\_SMALL \*/
59. **break**;
60. **case** AV\_PIX\_FMT\_ARGB:
61. #if CONFIG\_SMALL
62. \*yuv2packedX = yuv2argb32\_full\_X\_c;
63. \*yuv2packed2 = yuv2argb32\_full\_2\_c;
64. \*yuv2packed1 = yuv2argb32\_full\_1\_c;
65. #else
66. #if CONFIG\_SWSCALE\_ALPHA
67. **if** (c->alpPixBuf) {
68. \*yuv2packedX = yuv2argb32\_full\_X\_c;
69. \*yuv2packed2 = yuv2argb32\_full\_2\_c;
70. \*yuv2packed1 = yuv2argb32\_full\_1\_c;
71. } **else**
72. #endif /\* CONFIG\_SWSCALE\_ALPHA \*/
73. {
74. \*yuv2packedX = yuv2xrgb32\_full\_X\_c;
75. \*yuv2packed2 = yuv2xrgb32\_full\_2\_c;
76. \*yuv2packed1 = yuv2xrgb32\_full\_1\_c;
77. }
78. #endif /\* !CONFIG\_SMALL \*/
79. **break**;
80. **case** AV\_PIX\_FMT\_BGRA:
81. #if CONFIG\_SMALL
82. \*yuv2packedX = yuv2bgra32\_full\_X\_c;
83. \*yuv2packed2 = yuv2bgra32\_full\_2\_c;
84. \*yuv2packed1 = yuv2bgra32\_full\_1\_c;
85. #else
86. #if CONFIG\_SWSCALE\_ALPHA
87. **if** (c->alpPixBuf) {
88. \*yuv2packedX = yuv2bgra32\_full\_X\_c;
89. \*yuv2packed2 = yuv2bgra32\_full\_2\_c;
90. \*yuv2packed1 = yuv2bgra32\_full\_1\_c;
91. } **else**
92. #endif /\* CONFIG\_SWSCALE\_ALPHA \*/
93. {
94. \*yuv2packedX = yuv2bgrx32\_full\_X\_c;
95. \*yuv2packed2 = yuv2bgrx32\_full\_2\_c;
96. \*yuv2packed1 = yuv2bgrx32\_full\_1\_c;
97. }
98. #endif /\* !CONFIG\_SMALL \*/
99. **break**;
100. **case** AV\_PIX\_FMT\_ABGR:
101. #if CONFIG\_SMALL
102. \*yuv2packedX = yuv2abgr32\_full\_X\_c;
103. \*yuv2packed2 = yuv2abgr32\_full\_2\_c;
104. \*yuv2packed1 = yuv2abgr32\_full\_1\_c;
105. #else
106. #if CONFIG\_SWSCALE\_ALPHA
107. **if** (c->alpPixBuf) {
108. \*yuv2packedX = yuv2abgr32\_full\_X\_c;
109. \*yuv2packed2 = yuv2abgr32\_full\_2\_c;
110. \*yuv2packed1 = yuv2abgr32\_full\_1\_c;
111. } **else**
112. #endif /\* CONFIG\_SWSCALE\_ALPHA \*/
113. {
114. \*yuv2packedX = yuv2xbgr32\_full\_X\_c;
115. \*yuv2packed2 = yuv2xbgr32\_full\_2\_c;
116. \*yuv2packed1 = yuv2xbgr32\_full\_1\_c;
117. }
118. #endif /\* !CONFIG\_SMALL \*/
119. **break**;
120. **case** AV\_PIX\_FMT\_RGB24:
121. \*yuv2packedX = yuv2rgb24\_full\_X\_c;
122. \*yuv2packed2 = yuv2rgb24\_full\_2\_c;
123. \*yuv2packed1 = yuv2rgb24\_full\_1\_c;
124. **break**;
125. **case** AV\_PIX\_FMT\_BGR24:
126. \*yuv2packedX = yuv2bgr24\_full\_X\_c;
127. \*yuv2packed2 = yuv2bgr24\_full\_2\_c;
128. \*yuv2packed1 = yuv2bgr24\_full\_1\_c;
129. **break**;
130. **case** AV\_PIX\_FMT\_BGR4\_BYTE:
131. \*yuv2packedX = yuv2bgr4\_byte\_full\_X\_c;
132. \*yuv2packed2 = yuv2bgr4\_byte\_full\_2\_c;
133. \*yuv2packed1 = yuv2bgr4\_byte\_full\_1\_c;
134. **break**;
135. **case** AV\_PIX\_FMT\_RGB4\_BYTE:
136. \*yuv2packedX = yuv2rgb4\_byte\_full\_X\_c;
137. \*yuv2packed2 = yuv2rgb4\_byte\_full\_2\_c;
138. \*yuv2packed1 = yuv2rgb4\_byte\_full\_1\_c;
139. **break**;
140. **case** AV\_PIX\_FMT\_BGR8:
141. \*yuv2packedX = yuv2bgr8\_full\_X\_c;
142. \*yuv2packed2 = yuv2bgr8\_full\_2\_c;
143. \*yuv2packed1 = yuv2bgr8\_full\_1\_c;
144. **break**;
145. **case** AV\_PIX\_FMT\_RGB8:
146. \*yuv2packedX = yuv2rgb8\_full\_X\_c;
147. \*yuv2packed2 = yuv2rgb8\_full\_2\_c;
148. \*yuv2packed1 = yuv2rgb8\_full\_1\_c;
149. **break**;
150. **case** AV\_PIX\_FMT\_GBRP:
151. **case** AV\_PIX\_FMT\_GBRP9BE:
152. **case** AV\_PIX\_FMT\_GBRP9LE:
153. **case** AV\_PIX\_FMT\_GBRP10BE:
154. **case** AV\_PIX\_FMT\_GBRP10LE:
155. **case** AV\_PIX\_FMT\_GBRP12BE:
156. **case** AV\_PIX\_FMT\_GBRP12LE:
157. **case** AV\_PIX\_FMT\_GBRP14BE:
158. **case** AV\_PIX\_FMT\_GBRP14LE:
159. **case** AV\_PIX\_FMT\_GBRP16BE:
160. **case** AV\_PIX\_FMT\_GBRP16LE:
161. **case** AV\_PIX\_FMT\_GBRAP:
162. \*yuv2anyX = yuv2gbrp\_full\_X\_c;
163. **break**;
164. }
165. **if** (!\*yuv2packedX && !\*yuv2anyX)
166. **goto** YUV\_PACKED;
167. } **else** {
168. YUV\_PACKED:
169. **switch** (dstFormat) {
170. **case** AV\_PIX\_FMT\_RGBA64LE:
171. #if CONFIG\_SWSCALE\_ALPHA
172. **if** (c->alpPixBuf) {
173. \*yuv2packed1 = yuv2rgba64le\_1\_c;
174. \*yuv2packed2 = yuv2rgba64le\_2\_c;
175. \*yuv2packedX = yuv2rgba64le\_X\_c;
176. } **else**
177. #endif /\* CONFIG\_SWSCALE\_ALPHA \*/
178. {
179. \*yuv2packed1 = yuv2rgbx64le\_1\_c;
180. \*yuv2packed2 = yuv2rgbx64le\_2\_c;
181. \*yuv2packedX = yuv2rgbx64le\_X\_c;
182. }
183. **break**;
184. **case** AV\_PIX\_FMT\_RGBA64BE:
185. #if CONFIG\_SWSCALE\_ALPHA
186. **if** (c->alpPixBuf) {
187. \*yuv2packed1 = yuv2rgba64be\_1\_c;
188. \*yuv2packed2 = yuv2rgba64be\_2\_c;
189. \*yuv2packedX = yuv2rgba64be\_X\_c;
190. } **else**
191. #endif /\* CONFIG\_SWSCALE\_ALPHA \*/
192. {
193. \*yuv2packed1 = yuv2rgbx64be\_1\_c;
194. \*yuv2packed2 = yuv2rgbx64be\_2\_c;
195. \*yuv2packedX = yuv2rgbx64be\_X\_c;
196. }
197. **break**;
198. **case** AV\_PIX\_FMT\_BGRA64LE:
199. #if CONFIG\_SWSCALE\_ALPHA
200. **if** (c->alpPixBuf) {
201. \*yuv2packed1 = yuv2bgra64le\_1\_c;
202. \*yuv2packed2 = yuv2bgra64le\_2\_c;
203. \*yuv2packedX = yuv2bgra64le\_X\_c;
204. } **else**
205. #endif /\* CONFIG\_SWSCALE\_ALPHA \*/
206. {
207. \*yuv2packed1 = yuv2bgrx64le\_1\_c;
208. \*yuv2packed2 = yuv2bgrx64le\_2\_c;
209. \*yuv2packedX = yuv2bgrx64le\_X\_c;
210. }
211. **break**;
212. **case** AV\_PIX\_FMT\_BGRA64BE:
213. #if CONFIG\_SWSCALE\_ALPHA
214. **if** (c->alpPixBuf) {
215. \*yuv2packed1 = yuv2bgra64be\_1\_c;
216. \*yuv2packed2 = yuv2bgra64be\_2\_c;
217. \*yuv2packedX = yuv2bgra64be\_X\_c;
218. } **else**
219. #endif /\* CONFIG\_SWSCALE\_ALPHA \*/
220. {
221. \*yuv2packed1 = yuv2bgrx64be\_1\_c;
222. \*yuv2packed2 = yuv2bgrx64be\_2\_c;
223. \*yuv2packedX = yuv2bgrx64be\_X\_c;
224. }
225. **break**;
226. **case** AV\_PIX\_FMT\_RGB48LE:
227. \*yuv2packed1 = yuv2rgb48le\_1\_c;
228. \*yuv2packed2 = yuv2rgb48le\_2\_c;
229. \*yuv2packedX = yuv2rgb48le\_X\_c;
230. **break**;
231. **case** AV\_PIX\_FMT\_RGB48BE:
232. \*yuv2packed1 = yuv2rgb48be\_1\_c;
233. \*yuv2packed2 = yuv2rgb48be\_2\_c;
234. \*yuv2packedX = yuv2rgb48be\_X\_c;
235. **break**;
236. **case** AV\_PIX\_FMT\_BGR48LE:
237. \*yuv2packed1 = yuv2bgr48le\_1\_c;
238. \*yuv2packed2 = yuv2bgr48le\_2\_c;
239. \*yuv2packedX = yuv2bgr48le\_X\_c;
240. **break**;
241. **case** AV\_PIX\_FMT\_BGR48BE:
242. \*yuv2packed1 = yuv2bgr48be\_1\_c;
243. \*yuv2packed2 = yuv2bgr48be\_2\_c;
244. \*yuv2packedX = yuv2bgr48be\_X\_c;
245. **break**;
246. **case** AV\_PIX\_FMT\_RGB32:
247. **case** AV\_PIX\_FMT\_BGR32:
248. #if CONFIG\_SMALL
249. \*yuv2packed1 = yuv2rgb32\_1\_c;
250. \*yuv2packed2 = yuv2rgb32\_2\_c;
251. \*yuv2packedX = yuv2rgb32\_X\_c;
252. #else
253. #if CONFIG\_SWSCALE\_ALPHA
254. **if** (c->alpPixBuf) {
255. \*yuv2packed1 = yuv2rgba32\_1\_c;
256. \*yuv2packed2 = yuv2rgba32\_2\_c;
257. \*yuv2packedX = yuv2rgba32\_X\_c;
258. } **else**
259. #endif /\* CONFIG\_SWSCALE\_ALPHA \*/
260. {
261. \*yuv2packed1 = yuv2rgbx32\_1\_c;
262. \*yuv2packed2 = yuv2rgbx32\_2\_c;
263. \*yuv2packedX = yuv2rgbx32\_X\_c;
264. }
265. #endif /\* !CONFIG\_SMALL \*/
266. **break**;
267. **case** AV\_PIX\_FMT\_RGB32\_1:
268. **case** AV\_PIX\_FMT\_BGR32\_1:
269. #if CONFIG\_SMALL
270. \*yuv2packed1 = yuv2rgb32\_1\_1\_c;
271. \*yuv2packed2 = yuv2rgb32\_1\_2\_c;
272. \*yuv2packedX = yuv2rgb32\_1\_X\_c;
273. #else
274. #if CONFIG\_SWSCALE\_ALPHA
275. **if** (c->alpPixBuf) {
276. \*yuv2packed1 = yuv2rgba32\_1\_1\_c;
277. \*yuv2packed2 = yuv2rgba32\_1\_2\_c;
278. \*yuv2packedX = yuv2rgba32\_1\_X\_c;
279. } **else**
280. #endif /\* CONFIG\_SWSCALE\_ALPHA \*/
281. {
282. \*yuv2packed1 = yuv2rgbx32\_1\_1\_c;
283. \*yuv2packed2 = yuv2rgbx32\_1\_2\_c;
284. \*yuv2packedX = yuv2rgbx32\_1\_X\_c;
285. }
286. #endif /\* !CONFIG\_SMALL \*/
287. **break**;
288. **case** AV\_PIX\_FMT\_RGB24:
289. \*yuv2packed1 = yuv2rgb24\_1\_c;
290. \*yuv2packed2 = yuv2rgb24\_2\_c;
291. \*yuv2packedX = yuv2rgb24\_X\_c;
292. **break**;
293. **case** AV\_PIX\_FMT\_BGR24:
294. \*yuv2packed1 = yuv2bgr24\_1\_c;
295. \*yuv2packed2 = yuv2bgr24\_2\_c;
296. \*yuv2packedX = yuv2bgr24\_X\_c;
297. **break**;
298. **case** AV\_PIX\_FMT\_RGB565LE:
299. **case** AV\_PIX\_FMT\_RGB565BE:
300. **case** AV\_PIX\_FMT\_BGR565LE:
301. **case** AV\_PIX\_FMT\_BGR565BE:
302. \*yuv2packed1 = yuv2rgb16\_1\_c;
303. \*yuv2packed2 = yuv2rgb16\_2\_c;
304. \*yuv2packedX = yuv2rgb16\_X\_c;
305. **break**;
306. **case** AV\_PIX\_FMT\_RGB555LE:
307. **case** AV\_PIX\_FMT\_RGB555BE:
308. **case** AV\_PIX\_FMT\_BGR555LE:
309. **case** AV\_PIX\_FMT\_BGR555BE:
310. \*yuv2packed1 = yuv2rgb15\_1\_c;
311. \*yuv2packed2 = yuv2rgb15\_2\_c;
312. \*yuv2packedX = yuv2rgb15\_X\_c;
313. **break**;
314. **case** AV\_PIX\_FMT\_RGB444LE:
315. **case** AV\_PIX\_FMT\_RGB444BE:
316. **case** AV\_PIX\_FMT\_BGR444LE:
317. **case** AV\_PIX\_FMT\_BGR444BE:
318. \*yuv2packed1 = yuv2rgb12\_1\_c;
319. \*yuv2packed2 = yuv2rgb12\_2\_c;
320. \*yuv2packedX = yuv2rgb12\_X\_c;
321. **break**;
322. **case** AV\_PIX\_FMT\_RGB8:
323. **case** AV\_PIX\_FMT\_BGR8:
324. \*yuv2packed1 = yuv2rgb8\_1\_c;
325. \*yuv2packed2 = yuv2rgb8\_2\_c;
326. \*yuv2packedX = yuv2rgb8\_X\_c;
327. **break**;
328. **case** AV\_PIX\_FMT\_RGB4:
329. **case** AV\_PIX\_FMT\_BGR4:
330. \*yuv2packed1 = yuv2rgb4\_1\_c;
331. \*yuv2packed2 = yuv2rgb4\_2\_c;
332. \*yuv2packedX = yuv2rgb4\_X\_c;
333. **break**;
334. **case** AV\_PIX\_FMT\_RGB4\_BYTE:
335. **case** AV\_PIX\_FMT\_BGR4\_BYTE:
336. \*yuv2packed1 = yuv2rgb4b\_1\_c;
337. \*yuv2packed2 = yuv2rgb4b\_2\_c;
338. \*yuv2packedX = yuv2rgb4b\_X\_c;
339. **break**;
340. }
341. }
342. **switch** (dstFormat) {
343. **case** AV\_PIX\_FMT\_MONOWHITE:
344. \*yuv2packed1 = yuv2monowhite\_1\_c;
345. \*yuv2packed2 = yuv2monowhite\_2\_c;
346. \*yuv2packedX = yuv2monowhite\_X\_c;
347. **break**;
348. **case** AV\_PIX\_FMT\_MONOBLACK:
349. \*yuv2packed1 = yuv2monoblack\_1\_c;
350. \*yuv2packed2 = yuv2monoblack\_2\_c;
351. \*yuv2packedX = yuv2monoblack\_X\_c;
352. **break**;
353. **case** AV\_PIX\_FMT\_YUYV422:
354. \*yuv2packed1 = yuv2yuyv422\_1\_c;
355. \*yuv2packed2 = yuv2yuyv422\_2\_c;
356. \*yuv2packedX = yuv2yuyv422\_X\_c;
357. **break**;
358. **case** AV\_PIX\_FMT\_YVYU422:
359. \*yuv2packed1 = yuv2yvyu422\_1\_c;
360. \*yuv2packed2 = yuv2yvyu422\_2\_c;
361. \*yuv2packedX = yuv2yvyu422\_X\_c;
362. **break**;
363. **case** AV\_PIX\_FMT\_UYVY422:
364. \*yuv2packed1 = yuv2uyvy422\_1\_c;
365. \*yuv2packed2 = yuv2uyvy422\_2\_c;
366. \*yuv2packedX = yuv2uyvy422\_X\_c;
367. **break**;
368. }
369. }

ff\_sws\_init\_output\_funcs()根据输出像素格式的不同，对以下几个函数指针进行赋值：

yuv2plane1：是yuv2planar1\_fn类型的函数指针。该函数用于输出一行水平拉伸后的planar格式数据。数据没有使用垂直拉伸。  
yuv2planeX：是yuv2planarX\_fn类型的函数指针。该函数用于输出一行水平拉伸后的planar格式数据。数据使用垂直拉伸。  
yuv2packed1：是yuv2packed1\_fn类型的函数指针。该函数用于输出一行水平拉伸后的packed格式数据。数据没有使用垂直拉伸。  
yuv2packed2：是yuv2packed2\_fn类型的函数指针。该函数用于输出一行水平拉伸后的packed格式数据。数据使用两行数据进行垂直拉伸。  
yuv2packedX：是yuv2packedX\_fn类型的函数指针。该函数用于输出一行水平拉伸后的packed格式数据。数据使用垂直拉伸。  
yuv2nv12cX：是yuv2interleavedX\_fn类型的函数指针。还没有研究该函数。  
yuv2anyX：是yuv2anyX\_fn类型的函数指针。还没有研究该函数。

### ff\_sws\_init\_input\_funcs()

ff\_sws\_init\_input\_funcs()用于初始化“输入函数”。“输入函数”在libswscale中的作用就是任意格式的像素转换为YUV格式以供后续的处理。ff\_sws\_init\_input\_funcs()的定义位于libswscale\input.c，如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44305697) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44305697)

[在CODE上查看代码片](https://code.csdn.net/snippets/621402)

1. av\_cold **void** ff\_sws\_init\_input\_funcs(SwsContext \*c)
2. {
3. **enum** AVPixelFormat srcFormat = c->srcFormat;
5. c->chrToYV12 = NULL;
6. **switch** (srcFormat) {
7. **case** AV\_PIX\_FMT\_YUYV422:
8. c->chrToYV12 = yuy2ToUV\_c;
9. **break**;
10. **case** AV\_PIX\_FMT\_YVYU422:
11. c->chrToYV12 = yvy2ToUV\_c;
12. **break**;
13. **case** AV\_PIX\_FMT\_UYVY422:
14. c->chrToYV12 = uyvyToUV\_c;
15. **break**;
16. **case** AV\_PIX\_FMT\_NV12:
17. c->chrToYV12 = nv12ToUV\_c;
18. **break**;
19. **case** AV\_PIX\_FMT\_NV21:
20. c->chrToYV12 = nv21ToUV\_c;
21. **break**;
22. **case** AV\_PIX\_FMT\_RGB8:
23. **case** AV\_PIX\_FMT\_BGR8:
24. **case** AV\_PIX\_FMT\_PAL8:
25. **case** AV\_PIX\_FMT\_BGR4\_BYTE:
26. **case** AV\_PIX\_FMT\_RGB4\_BYTE:
27. c->chrToYV12 = palToUV\_c;
28. **break**;
29. **case** AV\_PIX\_FMT\_GBRP9LE:
30. c->readChrPlanar = planar\_rgb9le\_to\_uv;
31. **break**;
32. **case** AV\_PIX\_FMT\_GBRP10LE:
33. c->readChrPlanar = planar\_rgb10le\_to\_uv;
34. **break**;
35. **case** AV\_PIX\_FMT\_GBRP12LE:
36. c->readChrPlanar = planar\_rgb12le\_to\_uv;
37. **break**;
38. **case** AV\_PIX\_FMT\_GBRP14LE:
39. c->readChrPlanar = planar\_rgb14le\_to\_uv;
40. **break**;
41. **case** AV\_PIX\_FMT\_GBRAP16LE:
42. **case** AV\_PIX\_FMT\_GBRP16LE:
43. c->readChrPlanar = planar\_rgb16le\_to\_uv;
44. **break**;
45. **case** AV\_PIX\_FMT\_GBRP9BE:
46. c->readChrPlanar = planar\_rgb9be\_to\_uv;
47. **break**;
48. **case** AV\_PIX\_FMT\_GBRP10BE:
49. c->readChrPlanar = planar\_rgb10be\_to\_uv;
50. **break**;
51. **case** AV\_PIX\_FMT\_GBRP12BE:
52. c->readChrPlanar = planar\_rgb12be\_to\_uv;
53. **break**;
54. **case** AV\_PIX\_FMT\_GBRP14BE:
55. c->readChrPlanar = planar\_rgb14be\_to\_uv;
56. **break**;
57. **case** AV\_PIX\_FMT\_GBRAP16BE:
58. **case** AV\_PIX\_FMT\_GBRP16BE:
59. c->readChrPlanar = planar\_rgb16be\_to\_uv;
60. **break**;
61. **case** AV\_PIX\_FMT\_GBRAP:
62. **case** AV\_PIX\_FMT\_GBRP:
63. c->readChrPlanar = planar\_rgb\_to\_uv;
64. **break**;
65. #if HAVE\_BIGENDIAN
66. **case** AV\_PIX\_FMT\_YUV444P9LE:
67. **case** AV\_PIX\_FMT\_YUV422P9LE:
68. **case** AV\_PIX\_FMT\_YUV420P9LE:
69. **case** AV\_PIX\_FMT\_YUV422P10LE:
70. **case** AV\_PIX\_FMT\_YUV444P10LE:
71. **case** AV\_PIX\_FMT\_YUV420P10LE:
72. **case** AV\_PIX\_FMT\_YUV422P12LE:
73. **case** AV\_PIX\_FMT\_YUV444P12LE:
74. **case** AV\_PIX\_FMT\_YUV420P12LE:
75. **case** AV\_PIX\_FMT\_YUV422P14LE:
76. **case** AV\_PIX\_FMT\_YUV444P14LE:
77. **case** AV\_PIX\_FMT\_YUV420P14LE:
78. **case** AV\_PIX\_FMT\_YUV420P16LE:
79. **case** AV\_PIX\_FMT\_YUV422P16LE:
80. **case** AV\_PIX\_FMT\_YUV444P16LE:
82. **case** AV\_PIX\_FMT\_YUVA444P9LE:
83. **case** AV\_PIX\_FMT\_YUVA422P9LE:
84. **case** AV\_PIX\_FMT\_YUVA420P9LE:
85. **case** AV\_PIX\_FMT\_YUVA444P10LE:
86. **case** AV\_PIX\_FMT\_YUVA422P10LE:
87. **case** AV\_PIX\_FMT\_YUVA420P10LE:
88. **case** AV\_PIX\_FMT\_YUVA420P16LE:
89. **case** AV\_PIX\_FMT\_YUVA422P16LE:
90. **case** AV\_PIX\_FMT\_YUVA444P16LE:
91. c->chrToYV12 = bswap16UV\_c;
92. **break**;
93. #else
94. **case** AV\_PIX\_FMT\_YUV444P9BE:
95. **case** AV\_PIX\_FMT\_YUV422P9BE:
96. **case** AV\_PIX\_FMT\_YUV420P9BE:
97. **case** AV\_PIX\_FMT\_YUV444P10BE:
98. **case** AV\_PIX\_FMT\_YUV422P10BE:
99. **case** AV\_PIX\_FMT\_YUV420P10BE:
100. **case** AV\_PIX\_FMT\_YUV444P12BE:
101. **case** AV\_PIX\_FMT\_YUV422P12BE:
102. **case** AV\_PIX\_FMT\_YUV420P12BE:
103. **case** AV\_PIX\_FMT\_YUV444P14BE:
104. **case** AV\_PIX\_FMT\_YUV422P14BE:
105. **case** AV\_PIX\_FMT\_YUV420P14BE:
106. **case** AV\_PIX\_FMT\_YUV420P16BE:
107. **case** AV\_PIX\_FMT\_YUV422P16BE:
108. **case** AV\_PIX\_FMT\_YUV444P16BE:
110. **case** AV\_PIX\_FMT\_YUVA444P9BE:
111. **case** AV\_PIX\_FMT\_YUVA422P9BE:
112. **case** AV\_PIX\_FMT\_YUVA420P9BE:
113. **case** AV\_PIX\_FMT\_YUVA444P10BE:
114. **case** AV\_PIX\_FMT\_YUVA422P10BE:
115. **case** AV\_PIX\_FMT\_YUVA420P10BE:
116. **case** AV\_PIX\_FMT\_YUVA420P16BE:
117. **case** AV\_PIX\_FMT\_YUVA422P16BE:
118. **case** AV\_PIX\_FMT\_YUVA444P16BE:
119. c->chrToYV12 = bswap16UV\_c;
120. **break**;
121. #endif
122. }
123. **if** (c->chrSrcHSubSample) {
124. **switch** (srcFormat) {
125. **case** AV\_PIX\_FMT\_RGBA64BE:
126. c->chrToYV12 = rgb64BEToUV\_half\_c;
127. **break**;
128. **case** AV\_PIX\_FMT\_RGBA64LE:
129. c->chrToYV12 = rgb64LEToUV\_half\_c;
130. **break**;
131. **case** AV\_PIX\_FMT\_BGRA64BE:
132. c->chrToYV12 = bgr64BEToUV\_half\_c;
133. **break**;
134. **case** AV\_PIX\_FMT\_BGRA64LE:
135. c->chrToYV12 = bgr64LEToUV\_half\_c;
136. **break**;
137. **case** AV\_PIX\_FMT\_RGB48BE:
138. c->chrToYV12 = rgb48BEToUV\_half\_c;
139. **break**;
140. **case** AV\_PIX\_FMT\_RGB48LE:
141. c->chrToYV12 = rgb48LEToUV\_half\_c;
142. **break**;
143. **case** AV\_PIX\_FMT\_BGR48BE:
144. c->chrToYV12 = bgr48BEToUV\_half\_c;
145. **break**;
146. **case** AV\_PIX\_FMT\_BGR48LE:
147. c->chrToYV12 = bgr48LEToUV\_half\_c;
148. **break**;
149. **case** AV\_PIX\_FMT\_RGB32:
150. c->chrToYV12 = bgr32ToUV\_half\_c;
151. **break**;
152. **case** AV\_PIX\_FMT\_RGB32\_1:
153. c->chrToYV12 = bgr321ToUV\_half\_c;
154. **break**;
155. **case** AV\_PIX\_FMT\_BGR24:
156. c->chrToYV12 = bgr24ToUV\_half\_c;
157. **break**;
158. **case** AV\_PIX\_FMT\_BGR565LE:
159. c->chrToYV12 = bgr16leToUV\_half\_c;
160. **break**;
161. **case** AV\_PIX\_FMT\_BGR565BE:
162. c->chrToYV12 = bgr16beToUV\_half\_c;
163. **break**;
164. **case** AV\_PIX\_FMT\_BGR555LE:
165. c->chrToYV12 = bgr15leToUV\_half\_c;
166. **break**;
167. **case** AV\_PIX\_FMT\_BGR555BE:
168. c->chrToYV12 = bgr15beToUV\_half\_c;
169. **break**;
170. **case** AV\_PIX\_FMT\_GBRAP:
171. **case** AV\_PIX\_FMT\_GBRP:
172. c->chrToYV12 = gbr24pToUV\_half\_c;
173. **break**;
174. **case** AV\_PIX\_FMT\_BGR444LE:
175. c->chrToYV12 = bgr12leToUV\_half\_c;
176. **break**;
177. **case** AV\_PIX\_FMT\_BGR444BE:
178. c->chrToYV12 = bgr12beToUV\_half\_c;
179. **break**;
180. **case** AV\_PIX\_FMT\_BGR32:
181. c->chrToYV12 = rgb32ToUV\_half\_c;
182. **break**;
183. **case** AV\_PIX\_FMT\_BGR32\_1:
184. c->chrToYV12 = rgb321ToUV\_half\_c;
185. **break**;
186. **case** AV\_PIX\_FMT\_RGB24:
187. c->chrToYV12 = rgb24ToUV\_half\_c;
188. **break**;
189. **case** AV\_PIX\_FMT\_RGB565LE:
190. c->chrToYV12 = rgb16leToUV\_half\_c;
191. **break**;
192. **case** AV\_PIX\_FMT\_RGB565BE:
193. c->chrToYV12 = rgb16beToUV\_half\_c;
194. **break**;
195. **case** AV\_PIX\_FMT\_RGB555LE:
196. c->chrToYV12 = rgb15leToUV\_half\_c;
197. **break**;
198. **case** AV\_PIX\_FMT\_RGB555BE:
199. c->chrToYV12 = rgb15beToUV\_half\_c;
200. **break**;
201. **case** AV\_PIX\_FMT\_RGB444LE:
202. c->chrToYV12 = rgb12leToUV\_half\_c;
203. **break**;
204. **case** AV\_PIX\_FMT\_RGB444BE:
205. c->chrToYV12 = rgb12beToUV\_half\_c;
206. **break**;
207. }
208. } **else** {
209. **switch** (srcFormat) {
210. **case** AV\_PIX\_FMT\_RGBA64BE:
211. c->chrToYV12 = rgb64BEToUV\_c;
212. **break**;
213. **case** AV\_PIX\_FMT\_RGBA64LE:
214. c->chrToYV12 = rgb64LEToUV\_c;
215. **break**;
216. **case** AV\_PIX\_FMT\_BGRA64BE:
217. c->chrToYV12 = bgr64BEToUV\_c;
218. **break**;
219. **case** AV\_PIX\_FMT\_BGRA64LE:
220. c->chrToYV12 = bgr64LEToUV\_c;
221. **break**;
222. **case** AV\_PIX\_FMT\_RGB48BE:
223. c->chrToYV12 = rgb48BEToUV\_c;
224. **break**;
225. **case** AV\_PIX\_FMT\_RGB48LE:
226. c->chrToYV12 = rgb48LEToUV\_c;
227. **break**;
228. **case** AV\_PIX\_FMT\_BGR48BE:
229. c->chrToYV12 = bgr48BEToUV\_c;
230. **break**;
231. **case** AV\_PIX\_FMT\_BGR48LE:
232. c->chrToYV12 = bgr48LEToUV\_c;
233. **break**;
234. **case** AV\_PIX\_FMT\_RGB32:
235. c->chrToYV12 = bgr32ToUV\_c;
236. **break**;
237. **case** AV\_PIX\_FMT\_RGB32\_1:
238. c->chrToYV12 = bgr321ToUV\_c;
239. **break**;
240. **case** AV\_PIX\_FMT\_BGR24:
241. c->chrToYV12 = bgr24ToUV\_c;
242. **break**;
243. **case** AV\_PIX\_FMT\_BGR565LE:
244. c->chrToYV12 = bgr16leToUV\_c;
245. **break**;
246. **case** AV\_PIX\_FMT\_BGR565BE:
247. c->chrToYV12 = bgr16beToUV\_c;
248. **break**;
249. **case** AV\_PIX\_FMT\_BGR555LE:
250. c->chrToYV12 = bgr15leToUV\_c;
251. **break**;
252. **case** AV\_PIX\_FMT\_BGR555BE:
253. c->chrToYV12 = bgr15beToUV\_c;
254. **break**;
255. **case** AV\_PIX\_FMT\_BGR444LE:
256. c->chrToYV12 = bgr12leToUV\_c;
257. **break**;
258. **case** AV\_PIX\_FMT\_BGR444BE:
259. c->chrToYV12 = bgr12beToUV\_c;
260. **break**;
261. **case** AV\_PIX\_FMT\_BGR32:
262. c->chrToYV12 = rgb32ToUV\_c;
263. **break**;
264. **case** AV\_PIX\_FMT\_BGR32\_1:
265. c->chrToYV12 = rgb321ToUV\_c;
266. **break**;
267. **case** AV\_PIX\_FMT\_RGB24:
268. c->chrToYV12 = rgb24ToUV\_c;
269. **break**;
270. **case** AV\_PIX\_FMT\_RGB565LE:
271. c->chrToYV12 = rgb16leToUV\_c;
272. **break**;
273. **case** AV\_PIX\_FMT\_RGB565BE:
274. c->chrToYV12 = rgb16beToUV\_c;
275. **break**;
276. **case** AV\_PIX\_FMT\_RGB555LE:
277. c->chrToYV12 = rgb15leToUV\_c;
278. **break**;
279. **case** AV\_PIX\_FMT\_RGB555BE:
280. c->chrToYV12 = rgb15beToUV\_c;
281. **break**;
282. **case** AV\_PIX\_FMT\_RGB444LE:
283. c->chrToYV12 = rgb12leToUV\_c;
284. **break**;
285. **case** AV\_PIX\_FMT\_RGB444BE:
286. c->chrToYV12 = rgb12beToUV\_c;
287. **break**;
288. }
289. }
291. c->lumToYV12 = NULL;
292. c->alpToYV12 = NULL;
293. **switch** (srcFormat) {
294. **case** AV\_PIX\_FMT\_GBRP9LE:
295. c->readLumPlanar = planar\_rgb9le\_to\_y;
296. **break**;
297. **case** AV\_PIX\_FMT\_GBRP10LE:
298. c->readLumPlanar = planar\_rgb10le\_to\_y;
299. **break**;
300. **case** AV\_PIX\_FMT\_GBRP12LE:
301. c->readLumPlanar = planar\_rgb12le\_to\_y;
302. **break**;
303. **case** AV\_PIX\_FMT\_GBRP14LE:
304. c->readLumPlanar = planar\_rgb14le\_to\_y;
305. **break**;
306. **case** AV\_PIX\_FMT\_GBRAP16LE:
307. **case** AV\_PIX\_FMT\_GBRP16LE:
308. c->readLumPlanar = planar\_rgb16le\_to\_y;
309. **break**;
310. **case** AV\_PIX\_FMT\_GBRP9BE:
311. c->readLumPlanar = planar\_rgb9be\_to\_y;
312. **break**;
313. **case** AV\_PIX\_FMT\_GBRP10BE:
314. c->readLumPlanar = planar\_rgb10be\_to\_y;
315. **break**;
316. **case** AV\_PIX\_FMT\_GBRP12BE:
317. c->readLumPlanar = planar\_rgb12be\_to\_y;
318. **break**;
319. **case** AV\_PIX\_FMT\_GBRP14BE:
320. c->readLumPlanar = planar\_rgb14be\_to\_y;
321. **break**;
322. **case** AV\_PIX\_FMT\_GBRAP16BE:
323. **case** AV\_PIX\_FMT\_GBRP16BE:
324. c->readLumPlanar = planar\_rgb16be\_to\_y;
325. **break**;
326. **case** AV\_PIX\_FMT\_GBRAP:
327. c->readAlpPlanar = planar\_rgb\_to\_a;
328. **case** AV\_PIX\_FMT\_GBRP:
329. c->readLumPlanar = planar\_rgb\_to\_y;
330. **break**;
331. #if HAVE\_BIGENDIAN
332. **case** AV\_PIX\_FMT\_YUV444P9LE:
333. **case** AV\_PIX\_FMT\_YUV422P9LE:
334. **case** AV\_PIX\_FMT\_YUV420P9LE:
335. **case** AV\_PIX\_FMT\_YUV444P10LE:
336. **case** AV\_PIX\_FMT\_YUV422P10LE:
337. **case** AV\_PIX\_FMT\_YUV420P10LE:
338. **case** AV\_PIX\_FMT\_YUV444P12LE:
339. **case** AV\_PIX\_FMT\_YUV422P12LE:
340. **case** AV\_PIX\_FMT\_YUV420P12LE:
341. **case** AV\_PIX\_FMT\_YUV444P14LE:
342. **case** AV\_PIX\_FMT\_YUV422P14LE:
343. **case** AV\_PIX\_FMT\_YUV420P14LE:
344. **case** AV\_PIX\_FMT\_YUV420P16LE:
345. **case** AV\_PIX\_FMT\_YUV422P16LE:
346. **case** AV\_PIX\_FMT\_YUV444P16LE:
348. **case** AV\_PIX\_FMT\_GRAY16LE:
349. c->lumToYV12 = bswap16Y\_c;
350. **break**;
351. **case** AV\_PIX\_FMT\_YUVA444P9LE:
352. **case** AV\_PIX\_FMT\_YUVA422P9LE:
353. **case** AV\_PIX\_FMT\_YUVA420P9LE:
354. **case** AV\_PIX\_FMT\_YUVA444P10LE:
355. **case** AV\_PIX\_FMT\_YUVA422P10LE:
356. **case** AV\_PIX\_FMT\_YUVA420P10LE:
357. **case** AV\_PIX\_FMT\_YUVA420P16LE:
358. **case** AV\_PIX\_FMT\_YUVA422P16LE:
359. **case** AV\_PIX\_FMT\_YUVA444P16LE:
360. c->lumToYV12 = bswap16Y\_c;
361. c->alpToYV12 = bswap16Y\_c;
362. **break**;
363. #else
364. **case** AV\_PIX\_FMT\_YUV444P9BE:
365. **case** AV\_PIX\_FMT\_YUV422P9BE:
366. **case** AV\_PIX\_FMT\_YUV420P9BE:
367. **case** AV\_PIX\_FMT\_YUV444P10BE:
368. **case** AV\_PIX\_FMT\_YUV422P10BE:
369. **case** AV\_PIX\_FMT\_YUV420P10BE:
370. **case** AV\_PIX\_FMT\_YUV444P12BE:
371. **case** AV\_PIX\_FMT\_YUV422P12BE:
372. **case** AV\_PIX\_FMT\_YUV420P12BE:
373. **case** AV\_PIX\_FMT\_YUV444P14BE:
374. **case** AV\_PIX\_FMT\_YUV422P14BE:
375. **case** AV\_PIX\_FMT\_YUV420P14BE:
376. **case** AV\_PIX\_FMT\_YUV420P16BE:
377. **case** AV\_PIX\_FMT\_YUV422P16BE:
378. **case** AV\_PIX\_FMT\_YUV444P16BE:
380. **case** AV\_PIX\_FMT\_GRAY16BE:
381. c->lumToYV12 = bswap16Y\_c;
382. **break**;
383. **case** AV\_PIX\_FMT\_YUVA444P9BE:
384. **case** AV\_PIX\_FMT\_YUVA422P9BE:
385. **case** AV\_PIX\_FMT\_YUVA420P9BE:
386. **case** AV\_PIX\_FMT\_YUVA444P10BE:
387. **case** AV\_PIX\_FMT\_YUVA422P10BE:
388. **case** AV\_PIX\_FMT\_YUVA420P10BE:
389. **case** AV\_PIX\_FMT\_YUVA420P16BE:
390. **case** AV\_PIX\_FMT\_YUVA422P16BE:
391. **case** AV\_PIX\_FMT\_YUVA444P16BE:
392. c->lumToYV12 = bswap16Y\_c;
393. c->alpToYV12 = bswap16Y\_c;
394. **break**;
395. #endif
396. **case** AV\_PIX\_FMT\_YA16LE:
397. c->lumToYV12 = read\_ya16le\_gray\_c;
398. c->alpToYV12 = read\_ya16le\_alpha\_c;
399. **break**;
400. **case** AV\_PIX\_FMT\_YA16BE:
401. c->lumToYV12 = read\_ya16be\_gray\_c;
402. c->alpToYV12 = read\_ya16be\_alpha\_c;
403. **break**;
404. **case** AV\_PIX\_FMT\_YUYV422:
405. **case** AV\_PIX\_FMT\_YVYU422:
406. **case** AV\_PIX\_FMT\_YA8:
407. c->lumToYV12 = yuy2ToY\_c;
408. **break**;
409. **case** AV\_PIX\_FMT\_UYVY422:
410. c->lumToYV12 = uyvyToY\_c;
411. **break**;
412. **case** AV\_PIX\_FMT\_BGR24:
413. c->lumToYV12 = bgr24ToY\_c;
414. **break**;
415. **case** AV\_PIX\_FMT\_BGR565LE:
416. c->lumToYV12 = bgr16leToY\_c;
417. **break**;
418. **case** AV\_PIX\_FMT\_BGR565BE:
419. c->lumToYV12 = bgr16beToY\_c;
420. **break**;
421. **case** AV\_PIX\_FMT\_BGR555LE:
422. c->lumToYV12 = bgr15leToY\_c;
423. **break**;
424. **case** AV\_PIX\_FMT\_BGR555BE:
425. c->lumToYV12 = bgr15beToY\_c;
426. **break**;
427. **case** AV\_PIX\_FMT\_BGR444LE:
428. c->lumToYV12 = bgr12leToY\_c;
429. **break**;
430. **case** AV\_PIX\_FMT\_BGR444BE:
431. c->lumToYV12 = bgr12beToY\_c;
432. **break**;
433. **case** AV\_PIX\_FMT\_RGB24:
434. c->lumToYV12 = rgb24ToY\_c;
435. **break**;
436. **case** AV\_PIX\_FMT\_RGB565LE:
437. c->lumToYV12 = rgb16leToY\_c;
438. **break**;
439. **case** AV\_PIX\_FMT\_RGB565BE:
440. c->lumToYV12 = rgb16beToY\_c;
441. **break**;
442. **case** AV\_PIX\_FMT\_RGB555LE:
443. c->lumToYV12 = rgb15leToY\_c;
444. **break**;
445. **case** AV\_PIX\_FMT\_RGB555BE:
446. c->lumToYV12 = rgb15beToY\_c;
447. **break**;
448. **case** AV\_PIX\_FMT\_RGB444LE:
449. c->lumToYV12 = rgb12leToY\_c;
450. **break**;
451. **case** AV\_PIX\_FMT\_RGB444BE:
452. c->lumToYV12 = rgb12beToY\_c;
453. **break**;
454. **case** AV\_PIX\_FMT\_RGB8:
455. **case** AV\_PIX\_FMT\_BGR8:
456. **case** AV\_PIX\_FMT\_PAL8:
457. **case** AV\_PIX\_FMT\_BGR4\_BYTE:
458. **case** AV\_PIX\_FMT\_RGB4\_BYTE:
459. c->lumToYV12 = palToY\_c;
460. **break**;
461. **case** AV\_PIX\_FMT\_MONOBLACK:
462. c->lumToYV12 = monoblack2Y\_c;
463. **break**;
464. **case** AV\_PIX\_FMT\_MONOWHITE:
465. c->lumToYV12 = monowhite2Y\_c;
466. **break**;
467. **case** AV\_PIX\_FMT\_RGB32:
468. c->lumToYV12 = bgr32ToY\_c;
469. **break**;
470. **case** AV\_PIX\_FMT\_RGB32\_1:
471. c->lumToYV12 = bgr321ToY\_c;
472. **break**;
473. **case** AV\_PIX\_FMT\_BGR32:
474. c->lumToYV12 = rgb32ToY\_c;
475. **break**;
476. **case** AV\_PIX\_FMT\_BGR32\_1:
477. c->lumToYV12 = rgb321ToY\_c;
478. **break**;
479. **case** AV\_PIX\_FMT\_RGB48BE:
480. c->lumToYV12 = rgb48BEToY\_c;
481. **break**;
482. **case** AV\_PIX\_FMT\_RGB48LE:
483. c->lumToYV12 = rgb48LEToY\_c;
484. **break**;
485. **case** AV\_PIX\_FMT\_BGR48BE:
486. c->lumToYV12 = bgr48BEToY\_c;
487. **break**;
488. **case** AV\_PIX\_FMT\_BGR48LE:
489. c->lumToYV12 = bgr48LEToY\_c;
490. **break**;
491. **case** AV\_PIX\_FMT\_RGBA64BE:
492. c->lumToYV12 = rgb64BEToY\_c;
493. **break**;
494. **case** AV\_PIX\_FMT\_RGBA64LE:
495. c->lumToYV12 = rgb64LEToY\_c;
496. **break**;
497. **case** AV\_PIX\_FMT\_BGRA64BE:
498. c->lumToYV12 = bgr64BEToY\_c;
499. **break**;
500. **case** AV\_PIX\_FMT\_BGRA64LE:
501. c->lumToYV12 = bgr64LEToY\_c;
502. }
503. **if** (c->alpPixBuf) {
504. **if** (is16BPS(srcFormat) || isNBPS(srcFormat)) {
505. **if** (HAVE\_BIGENDIAN == !isBE(srcFormat))
506. c->alpToYV12 = bswap16Y\_c;
507. }
508. **switch** (srcFormat) {
509. **case** AV\_PIX\_FMT\_BGRA64LE:
510. **case** AV\_PIX\_FMT\_BGRA64BE:
511. **case** AV\_PIX\_FMT\_RGBA64LE:
512. **case** AV\_PIX\_FMT\_RGBA64BE:  c->alpToYV12 = rgba64ToA\_c; **break**;
513. **case** AV\_PIX\_FMT\_BGRA:
514. **case** AV\_PIX\_FMT\_RGBA:
515. c->alpToYV12 = rgbaToA\_c;
516. **break**;
517. **case** AV\_PIX\_FMT\_ABGR:
518. **case** AV\_PIX\_FMT\_ARGB:
519. c->alpToYV12 = abgrToA\_c;
520. **break**;
521. **case** AV\_PIX\_FMT\_YA8:
522. c->alpToYV12 = uyvyToY\_c;
523. **break**;
524. **case** AV\_PIX\_FMT\_PAL8 :
525. c->alpToYV12 = palToA\_c;
526. **break**;
527. }
528. }
529. }

ff\_sws\_init\_input\_funcs()根据输入像素格式的不同，对以下几个函数指针进行赋值：

lumToYV12：转换得到Y分量。  
chrToYV12：转换得到UV分量。  
alpToYV12：转换得到Alpha分量。  
readLumPlanar：读取planar格式的数据转换为Y。  
readChrPlanar：读取planar格式的数据转换为UV。

下面看几个例子。

当输入像素格式为AV\_PIX\_FMT\_RGB24的时候，lumToYV12()指针指向的函数是rgb24ToY\_c()，如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44305697) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44305697)

[在CODE上查看代码片](https://code.csdn.net/snippets/621402)

1. **case** AV\_PIX\_FMT\_RGB24:
2. c->lumToYV12 = rgb24ToY\_c;
3. **break**;

#### rgb24ToY\_c()

rgb24ToY\_c()的定义如下。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44305697) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44305697)

[在CODE上查看代码片](https://code.csdn.net/snippets/621402)

1. **static** **void** rgb24ToY\_c(uint8\_t \*\_dst, **const** uint8\_t \*src, **const** uint8\_t \*unused1, **const** uint8\_t \*unused2, **int** width,
2. uint32\_t \*rgb2yuv)
3. {
4. int16\_t \*dst = (int16\_t \*)\_dst;
5. int32\_t ry = rgb2yuv[RY\_IDX], gy = rgb2yuv[GY\_IDX], by = rgb2yuv[BY\_IDX];
6. **int** i;
7. **for** (i = 0; i < width; i++) {
8. **int** r = src[i \* 3 + 0];
9. **int** g = src[i \* 3 + 1];
10. **int** b = src[i \* 3 + 2];
12. dst[i] = ((ry\*r + gy\*g + by\*b + (32<<(RGB2YUV\_SHIFT-1)) + (1<<(RGB2YUV\_SHIFT-7)))>>(RGB2YUV\_SHIFT-6));
13. }
14. }

从源代码中可以看出，该函数主要完成了以下三步：

1. 取系数。通过读取rgb2yuv数组中存储的参数获得R，G，B每个分量的系数。  
2. 取像素值。分别读取R，G，B每个分量的像素值。  
3. 计算得到亮度值。根据R，G，B的系数和值，计算得到亮度值Y。

当输入像素格式为AV\_PIX\_FMT\_RGB24的时候，chrToYV12 ()指针指向的函数是rgb24ToUV\_half\_c()，如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44305697) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44305697)

[在CODE上查看代码片](https://code.csdn.net/snippets/621402)

1. **case** AV\_PIX\_FMT\_RGB24:
2. c->chrToYV12 = rgb24ToUV\_half\_c;
3. **break**;

#### rgb24ToUV\_half\_c()

rgb24ToUV\_half\_c()定义如下。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44305697) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44305697)

[在CODE上查看代码片](https://code.csdn.net/snippets/621402)

1. **static** **void** rgb24ToUV\_half\_c(uint8\_t \*\_dstU, uint8\_t \*\_dstV, **const** uint8\_t \*unused0, **const** uint8\_t \*src1,
2. **const** uint8\_t \*src2, **int** width, uint32\_t \*rgb2yuv)
3. {
4. int16\_t \*dstU = (int16\_t \*)\_dstU;
5. int16\_t \*dstV = (int16\_t \*)\_dstV;
6. **int** i;
7. int32\_t ru = rgb2yuv[RU\_IDX], gu = rgb2yuv[GU\_IDX], bu = rgb2yuv[BU\_IDX];
8. int32\_t rv = rgb2yuv[RV\_IDX], gv = rgb2yuv[GV\_IDX], bv = rgb2yuv[BV\_IDX];
9. av\_assert1(src1 == src2);
10. **for** (i = 0; i < width; i++) {
11. **int** r = src1[6 \* i + 0] + src1[6 \* i + 3];
12. **int** g = src1[6 \* i + 1] + src1[6 \* i + 4];
13. **int** b = src1[6 \* i + 2] + src1[6 \* i + 5];
15. dstU[i] = (ru\*r + gu\*g + bu\*b + (256<<RGB2YUV\_SHIFT) + (1<<(RGB2YUV\_SHIFT-6)))>>(RGB2YUV\_SHIFT-5);
16. dstV[i] = (rv\*r + gv\*g + bv\*b + (256<<RGB2YUV\_SHIFT) + (1<<(RGB2YUV\_SHIFT-6)))>>(RGB2YUV\_SHIFT-5);
17. }
18. }

rgb24ToUV\_half\_c()的过程相比rgb24ToY\_c()要稍微复杂些。这主要是因为U，V取值的数量只有Y的一半。因此需要首先求出每2个像素点的平均值之后，再进行计算。  
当输入像素格式为AV\_PIX\_FMT\_GBRP（注意这个是planar格式，三个分量分别为G，B，R）的时候，readLumPlanar指向的函数是planar\_rgb\_to\_y()，如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44305697) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44305697)

[在CODE上查看代码片](https://code.csdn.net/snippets/621402)

1. **case** AV\_PIX\_FMT\_GBRP:
2. c->readLumPlanar = planar\_rgb\_to\_y;
3. **break**;

#### planar\_rgb\_to\_y()

planar\_rgb\_to\_y()定义如下。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44305697) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44305697)

[在CODE上查看代码片](https://code.csdn.net/snippets/621402)

1. **static** **void** planar\_rgb\_to\_y(uint8\_t \*\_dst, **const** uint8\_t \*src[4], **int** width, int32\_t \*rgb2yuv)
2. {
3. uint16\_t \*dst = (uint16\_t \*)\_dst;
4. int32\_t ry = rgb2yuv[RY\_IDX], gy = rgb2yuv[GY\_IDX], by = rgb2yuv[BY\_IDX];
5. **int** i;
6. **for** (i = 0; i < width; i++) {
7. **int** g = src[0][i];
8. **int** b = src[1][i];
9. **int** r = src[2][i];
11. dst[i] = (ry\*r + gy\*g + by\*b + (0x801<<(RGB2YUV\_SHIFT-7))) >> (RGB2YUV\_SHIFT-6);
12. }
13. }

可以看出处理planar格式的GBR数据和处理packed格式的RGB数据的方法是基本一样的，在这里不再重复。

### ff\_sws\_init\_range\_convert()

ff\_sws\_init\_range\_convert()用于初始化像素值范围转换的函数，它的定义位于libswscale\swscale.c，如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44305697) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44305697)

[在CODE上查看代码片](https://code.csdn.net/snippets/621402)

1. av\_cold **void** ff\_sws\_init\_range\_convert(SwsContext \*c)
2. {
3. c->lumConvertRange = NULL;
4. c->chrConvertRange = NULL;
5. **if** (c->srcRange != c->dstRange && !isAnyRGB(c->dstFormat)) {
6. **if** (c->dstBpc <= 14) {
7. **if** (c->srcRange) {
8. c->lumConvertRange = lumRangeFromJpeg\_c;
9. c->chrConvertRange = chrRangeFromJpeg\_c;
10. } **else** {
11. c->lumConvertRange = lumRangeToJpeg\_c;
12. c->chrConvertRange = chrRangeToJpeg\_c;
13. }
14. } **else** {
15. **if** (c->srcRange) {
16. c->lumConvertRange = lumRangeFromJpeg16\_c;
17. c->chrConvertRange = chrRangeFromJpeg16\_c;
18. } **else** {
19. c->lumConvertRange = lumRangeToJpeg16\_c;
20. c->chrConvertRange = chrRangeToJpeg16\_c;
21. }
22. }
23. }
24. }

ff\_sws\_init\_range\_convert()包含了两种像素取值范围的转换：

lumConvertRange：亮度分量取值范围的转换。  
chrConvertRange：色度分量取值范围的转换。

从JPEG标准转换为MPEG标准的函数有：lumRangeFromJpeg\_c()和chrRangeFromJpeg\_c()。

#### lumRangeFromJpeg\_c()

亮度转换（0-255转换为16-235）函数lumRangeFromJpeg\_c()如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44305697) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44305697)

[在CODE上查看代码片](https://code.csdn.net/snippets/621402)

1. **static** **void** lumRangeFromJpeg\_c(int16\_t \*dst, **int** width)
2. {
3. **int** i;
4. **for** (i = 0; i < width; i++)
5. dst[i] = (dst[i] \* 14071 + 33561947) >> 14;
6. }

可以简单代入一个数字验证一下上述函数的正确性。该函数将亮度值“0”映射成“16”，“255”映射成“235”，因此我们可以代入一个“255”看看转换后的数值是否为“235”。在这里需要注意，dst中存储的像素数值是15bit的亮度值。因此我们需要将8bit的数值“255”左移7位后带入。经过计算，255左移7位后取值为32640，计算后得到的数值为30080，右移7位后得到的8bit亮度值即为235。

后续几个函数都可以用上面描述的方法进行验证，就不再重复了。

#### chrRangeFromJpeg\_c()

色度转换（0-255转换为16-240）函数chrRangeFromJpeg\_c()如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44305697) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44305697)

[在CODE上查看代码片](https://code.csdn.net/snippets/621402)

1. **static** **void** chrRangeFromJpeg\_c(int16\_t \*dstU, int16\_t \*dstV, **int** width)
2. {
3. **int** i;
4. **for** (i = 0; i < width; i++) {
5. dstU[i] = (dstU[i] \* 1799 + 4081085) >> 11; // 1469
6. dstV[i] = (dstV[i] \* 1799 + 4081085) >> 11; // 1469
7. }
8. }

从MPEG标准转换为JPEG标准的函数有：lumRangeToJpeg\_c()和chrRangeToJpeg\_c()。

#### lumRangeToJpeg\_c()

亮度转换（16-235转换为0-255）函数lumRangeToJpeg\_c()定义如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44305697) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44305697)

[在CODE上查看代码片](https://code.csdn.net/snippets/621402)

1. **static** **void** lumRangeToJpeg\_c(int16\_t \*dst, **int** width)
2. {
3. **int** i;
4. **for** (i = 0; i < width; i++)
5. dst[i] = (FFMIN(dst[i], 30189) \* 19077 - 39057361) >> 14;
6. }

#### chrRangeToJpeg\_c()

色度转换（16-240转换为0-255）函数chrRangeToJpeg\_c()定义如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44305697) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44305697)

[在CODE上查看代码片](https://code.csdn.net/snippets/621402)

1. **static** **void** chrRangeToJpeg\_c(int16\_t \*dstU, int16\_t \*dstV, **int** width)
2. {
3. **int** i;
4. **for** (i = 0; i < width; i++) {
5. dstU[i] = (FFMIN(dstU[i], 30775) \* 4663 - 9289992) >> 12; // -264
6. dstV[i] = (FFMIN(dstV[i], 30775) \* 4663 - 9289992) >> 12; // -264
7. }
8. }

至此sws\_getContext()的源代码就基本上分析完毕了。

# [FFmpeg源代码简单分析：libswscale的sws\_scale()](http://blog.csdn.net/leixiaohua1020/article/details/44346687)

本文继续上一篇文章《[FFmpeg源代码分析：sws\_getContext()](http://blog.csdn.net/leixiaohua1020/article/details/44305697)》的内容，简单分析FFmpeg的图像处理（缩放，YUV/RGB格式转换）类库libswsscale中的sws\_scale()函数。libswscale是一个主要用于处理图片像素数据的类库。可以完成图片像素格式的转换，图片的拉伸等工作。有关libswscale的使用可以参考文章：  
《[最简单的基于FFmpeg的libswscale的示例（YUV转RGB）](http://blog.csdn.net/leixiaohua1020/article/details/42134965)》

该类库常用的函数数量很少，一般情况下就3个：

sws\_getContext()：初始化一个SwsContext。

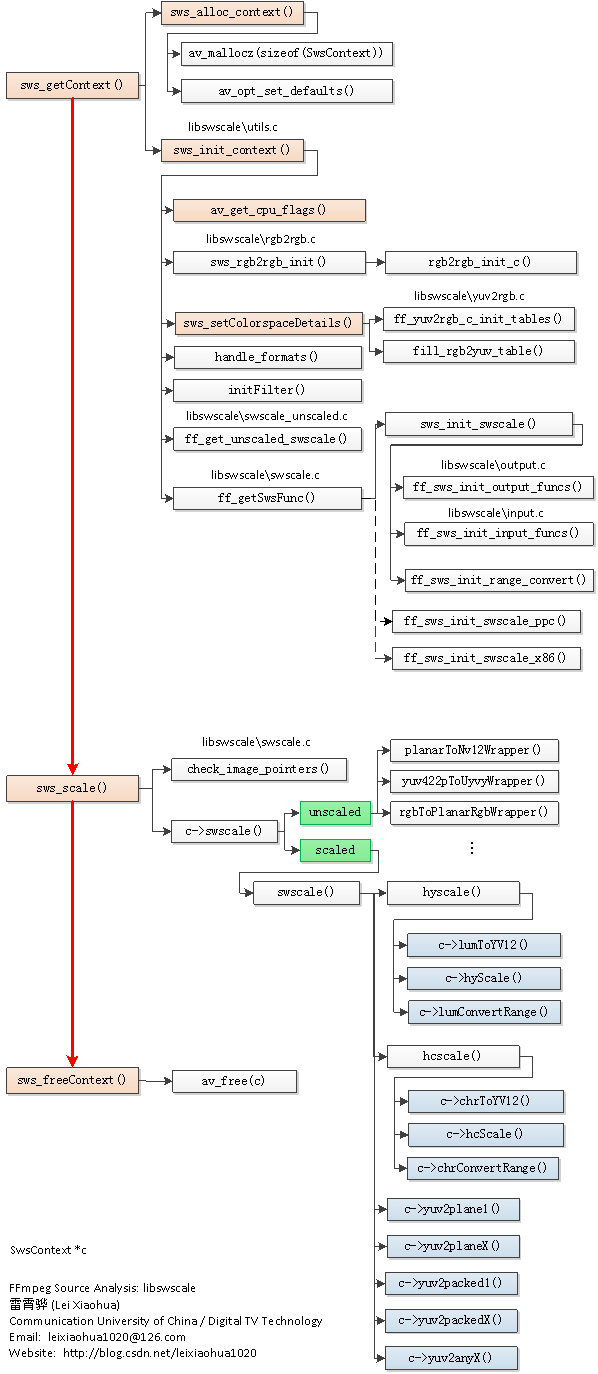
sws\_scale()：处理图像数据。

sws\_freeContext()：释放一个SwsContext。

在分析sws\_scale()的源代码之前，先简单回顾一下上篇文章中分析得到的两张图。

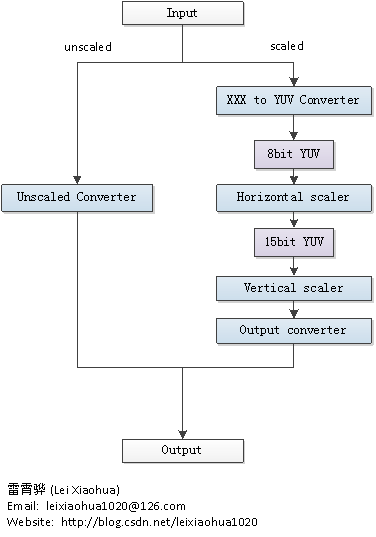
## 函数调用结构图

分析得到的libswscale的函数调用关系如下图所示。



## Libswscale处理数据流程

Libswscale处理像素数据的流程可以概括为下图。



从图中可以看出，libswscale处理数据有两条最主要的方式：unscaled和scaled。unscaled用于处理不需要拉伸的像素数据（属于比较特殊的情况），scaled用于处理需要拉伸的像素数据。Unscaled只需要对图像像素格式进行转换；而Scaled则除了对像素格式进行转换之外，还需要对图像进行缩放。Scaled方式可以分成以下几个步骤：

XXX to YUV Converter：首相将数据像素数据转换为8bitYUV格式；

Horizontal scaler：水平拉伸图像，并且转换为15bitYUV；

Vertical scaler：垂直拉伸图像；

Output converter：转换为输出像素格式。

## sws\_scale()

sws\_scale()是用于转换像素的函数。它的声明位于libswscale\swscale.h，如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44346687) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44346687)

[在CODE上查看代码片](https://code.csdn.net/snippets/621825)

1. /\*\*
2. \* Scale the image slice in srcSlice and put the resulting scaled
3. \* slice in the image in dst. A slice is a sequence of consecutive
4. \* rows in an image.
5. \*
6. \* Slices have to be provided in sequential order, either in
7. \* top-bottom or bottom-top order. If slices are provided in
8. \* non-sequential order the behavior of the function is undefined.
9. \*
10. \* @param c         the scaling context previously created with
11. \*                  sws\_getContext()
12. \* @param srcSlice  the array containing the pointers to the planes of
13. \*                  the source slice
14. \* @param srcStride the array containing the strides for each plane of
15. \*                  the source image
16. \* @param srcSliceY the position in the source image of the slice to
17. \*                  process, that is the number (counted starting from
18. \*                  zero) in the image of the first row of the slice
19. \* @param srcSliceH the height of the source slice, that is the number
20. \*                  of rows in the slice
21. \* @param dst       the array containing the pointers to the planes of
22. \*                  the destination image
23. \* @param dstStride the array containing the strides for each plane of
24. \*                  the destination image
25. \* @return          the height of the output slice
26. \*/
27. **int** sws\_scale(**struct** SwsContext \*c, **const** uint8\_t \***const** srcSlice[],
28. **const** **int** srcStride[], **int** srcSliceY, **int** srcSliceH,
29. uint8\_t \***const** dst[], **const** **int** dstStride[]);

sws\_scale()的定义位于libswscale\swscale.c，如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44346687) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44346687)

[在CODE上查看代码片](https://code.csdn.net/snippets/621825)

1. /\*\*
2. \* swscale wrapper, so we don't need to export the SwsContext.
3. \* Assumes planar YUV to be in YUV order instead of YVU.
4. \*/
5. **int** sws\_scale(**struct** SwsContext \*c,
6. **const** uint8\_t \* **const** srcSlice[],
7. **const** **int** srcStride[], **int** srcSliceY,
8. **int** srcSliceH, uint8\_t \***const** dst[],
9. **const** **int** dstStride[])
10. {
11. **int** i, ret;
12. **const** uint8\_t \*src2[4];
13. uint8\_t \*dst2[4];
14. uint8\_t \*rgb0\_tmp = NULL;
15. //检查输入参数
16. **if** (!srcStride || !dstStride || !dst || !srcSlice) {
17. av\_log(c, AV\_LOG\_ERROR, "One of the input parameters to sws\_scale() is NULL, please check the calling code\n");
18. **return** 0;
19. }
20. **if** (c->cascaded\_context[0] && srcSliceY == 0 && srcSliceH == c->cascaded\_context[0]->srcH) {
21. ret = sws\_scale(c->cascaded\_context[0],
22. srcSlice, srcStride, srcSliceY, srcSliceH,
23. c->cascaded\_tmp, c->cascaded\_tmpStride);
24. **if** (ret < 0)
25. **return** ret;
26. ret = sws\_scale(c->cascaded\_context[1],
27. (**const** uint8\_t \* **const** \* )c->cascaded\_tmp, c->cascaded\_tmpStride, 0, c->cascaded\_context[0]->dstH,
28. dst, dstStride);
29. **return** ret;
30. }
32. memcpy(src2, srcSlice, **sizeof**(src2));
33. memcpy(dst2, dst, **sizeof**(dst2));
35. // do not mess up sliceDir if we have a "trailing" 0-size slice
36. **if** (srcSliceH == 0)
37. **return** 0;
38. //检查
39. **if** (!check\_image\_pointers(srcSlice, c->srcFormat, srcStride)) {
40. av\_log(c, AV\_LOG\_ERROR, "bad src image pointers\n");
41. **return** 0;
42. }
43. **if** (!check\_image\_pointers((**const** uint8\_t\* **const**\*)dst, c->dstFormat, dstStride)) {
44. av\_log(c, AV\_LOG\_ERROR, "bad dst image pointers\n");
45. **return** 0;
46. }
48. **if** (c->sliceDir == 0 && srcSliceY != 0 && srcSliceY + srcSliceH != c->srcH) {
49. av\_log(c, AV\_LOG\_ERROR, "Slices start in the middle!\n");
50. **return** 0;
51. }
52. **if** (c->sliceDir == 0) {
53. **if** (srcSliceY == 0) c->sliceDir = 1; **else** c->sliceDir = -1;
54. }
55. //使用调色板palette的特殊处理？应该不常见
56. **if** (usePal(c->srcFormat)) {
57. **for** (i = 0; i < 256; i++) {
58. **int** r, g, b, y, u, v, a = 0xff;
59. **if** (c->srcFormat == AV\_PIX\_FMT\_PAL8) {
60. uint32\_t p = ((**const** uint32\_t \*)(srcSlice[1]))[i];
61. a = (p >> 24) & 0xFF;
62. r = (p >> 16) & 0xFF;
63. g = (p >>  8) & 0xFF;
64. b =  p        & 0xFF;
65. } **else** **if** (c->srcFormat == AV\_PIX\_FMT\_RGB8) {
66. r = ( i >> 5     ) \* 36;
67. g = ((i >> 2) & 7) \* 36;
68. b = ( i       & 3) \* 85;
69. } **else** **if** (c->srcFormat == AV\_PIX\_FMT\_BGR8) {
70. b = ( i >> 6     ) \* 85;
71. g = ((i >> 3) & 7) \* 36;
72. r = ( i       & 7) \* 36;
73. } **else** **if** (c->srcFormat == AV\_PIX\_FMT\_RGB4\_BYTE) {
74. r = ( i >> 3     ) \* 255;
75. g = ((i >> 1) & 3) \* 85;
76. b = ( i       & 1) \* 255;
77. } **else** **if** (c->srcFormat == AV\_PIX\_FMT\_GRAY8 || c->srcFormat == AV\_PIX\_FMT\_GRAY8A) {
78. r = g = b = i;
79. } **else** {
80. av\_assert1(c->srcFormat == AV\_PIX\_FMT\_BGR4\_BYTE);
81. b = ( i >> 3     ) \* 255;
82. g = ((i >> 1) & 3) \* 85;
83. r = ( i       & 1) \* 255;
84. }
85. #define RGB2YUV\_SHIFT 15
86. #define BY ( (int) (0.114 \* 219 / 255 \* (1 << RGB2YUV\_SHIFT) + 0.5))
87. #define BV (-(int) (0.081 \* 224 / 255 \* (1 << RGB2YUV\_SHIFT) + 0.5))
88. #define BU ( (int) (0.500 \* 224 / 255 \* (1 << RGB2YUV\_SHIFT) + 0.5))
89. #define GY ( (int) (0.587 \* 219 / 255 \* (1 << RGB2YUV\_SHIFT) + 0.5))
90. #define GV (-(int) (0.419 \* 224 / 255 \* (1 << RGB2YUV\_SHIFT) + 0.5))
91. #define GU (-(int) (0.331 \* 224 / 255 \* (1 << RGB2YUV\_SHIFT) + 0.5))
92. #define RY ( (int) (0.299 \* 219 / 255 \* (1 << RGB2YUV\_SHIFT) + 0.5))
93. #define RV ( (int) (0.500 \* 224 / 255 \* (1 << RGB2YUV\_SHIFT) + 0.5))
94. #define RU (-(int) (0.169 \* 224 / 255 \* (1 << RGB2YUV\_SHIFT) + 0.5))
96. y = av\_clip\_uint8((RY \* r + GY \* g + BY \* b + ( 33 << (RGB2YUV\_SHIFT - 1))) >> RGB2YUV\_SHIFT);
97. u = av\_clip\_uint8((RU \* r + GU \* g + BU \* b + (257 << (RGB2YUV\_SHIFT - 1))) >> RGB2YUV\_SHIFT);
98. v = av\_clip\_uint8((RV \* r + GV \* g + BV \* b + (257 << (RGB2YUV\_SHIFT - 1))) >> RGB2YUV\_SHIFT);
99. c->pal\_yuv[i]= y + (u<<8) + (v<<16) + ((unsigned)a<<24);
101. **switch** (c->dstFormat) {
102. **case** AV\_PIX\_FMT\_BGR32:
103. #if !HAVE\_BIGENDIAN
104. **case** AV\_PIX\_FMT\_RGB24:
105. #endif
106. c->pal\_rgb[i]=  r + (g<<8) + (b<<16) + ((unsigned)a<<24);
107. **break**;
108. **case** AV\_PIX\_FMT\_BGR32\_1:
109. #if HAVE\_BIGENDIAN
110. **case** AV\_PIX\_FMT\_BGR24:
111. #endif
112. c->pal\_rgb[i]= a + (r<<8) + (g<<16) + ((unsigned)b<<24);
113. **break**;
114. **case** AV\_PIX\_FMT\_RGB32\_1:
115. #if HAVE\_BIGENDIAN
116. **case** AV\_PIX\_FMT\_RGB24:
117. #endif
118. c->pal\_rgb[i]= a + (b<<8) + (g<<16) + ((unsigned)r<<24);
119. **break**;
120. **case** AV\_PIX\_FMT\_RGB32:
121. #if !HAVE\_BIGENDIAN
122. **case** AV\_PIX\_FMT\_BGR24:
123. #endif
124. **default**:
125. c->pal\_rgb[i]=  b + (g<<8) + (r<<16) + ((unsigned)a<<24);
126. }
127. }
128. }
129. //Alpha的特殊处理?
130. **if** (c->src0Alpha && !c->dst0Alpha && isALPHA(c->dstFormat)) {
131. uint8\_t \*base;
132. **int** x,y;
133. rgb0\_tmp = av\_malloc(FFABS(srcStride[0]) \* srcSliceH + 32);
134. **if** (!rgb0\_tmp)
135. **return** AVERROR(ENOMEM);
137. base = srcStride[0] < 0 ? rgb0\_tmp - srcStride[0] \* (srcSliceH-1) : rgb0\_tmp;
138. **for** (y=0; y<srcSliceH; y++){
139. memcpy(base + srcStride[0]\*y, src2[0] + srcStride[0]\*y, 4\*c->srcW);
140. **for** (x=c->src0Alpha-1; x<4\*c->srcW; x+=4) {
141. base[ srcStride[0]\*y + x] = 0xFF;
142. }
143. }
144. src2[0] = base;
145. }
146. //XYZ的特殊处理?
147. **if** (c->srcXYZ && !(c->dstXYZ && c->srcW==c->dstW && c->srcH==c->dstH)) {
148. uint8\_t \*base;
149. rgb0\_tmp = av\_malloc(FFABS(srcStride[0]) \* srcSliceH + 32);
150. **if** (!rgb0\_tmp)
151. **return** AVERROR(ENOMEM);
153. base = srcStride[0] < 0 ? rgb0\_tmp - srcStride[0] \* (srcSliceH-1) : rgb0\_tmp;
155. xyz12Torgb48(c, (uint16\_t\*)base, (**const** uint16\_t\*)src2[0], srcStride[0]/2, srcSliceH);
156. src2[0] = base;
157. }
159. **if** (!srcSliceY && (c->flags & SWS\_BITEXACT) && c->dither == SWS\_DITHER\_ED && c->dither\_error[0])
160. **for** (i = 0; i < 4; i++)
161. memset(c->dither\_error[i], 0, **sizeof**(c->dither\_error[0][0]) \* (c->dstW+2));

164. // copy strides, so they can safely be modified
165. // sliceDir: 1 = top-to-bottom; -1 = bottom-to-top;
166. **if** (c->sliceDir == 1) {
167. // slices go from top to bottom
168. **int** srcStride2[4] = { srcStride[0], srcStride[1], srcStride[2],
169. srcStride[3] };
170. **int** dstStride2[4] = { dstStride[0], dstStride[1], dstStride[2],
171. dstStride[3] };
173. reset\_ptr(src2, c->srcFormat);
174. reset\_ptr((**void**\*)dst2, c->dstFormat);
176. /\* reset slice direction at end of frame \*/
177. **if** (srcSliceY + srcSliceH == c->srcH)
178. c->sliceDir = 0;
179. //关键：调用
180. ret = c->swscale(c, src2, srcStride2, srcSliceY, srcSliceH, dst2,
181. dstStride2);
182. } **else** {
183. // slices go from bottom to top => we flip the image internally
184. **int** srcStride2[4] = { -srcStride[0], -srcStride[1], -srcStride[2],
185. -srcStride[3] };
186. **int** dstStride2[4] = { -dstStride[0], -dstStride[1], -dstStride[2],
187. -dstStride[3] };
189. src2[0] += (srcSliceH - 1) \* srcStride[0];
190. **if** (!usePal(c->srcFormat))
191. src2[1] += ((srcSliceH >> c->chrSrcVSubSample) - 1) \* srcStride[1];
192. src2[2] += ((srcSliceH >> c->chrSrcVSubSample) - 1) \* srcStride[2];
193. src2[3] += (srcSliceH - 1) \* srcStride[3];
194. dst2[0] += ( c->dstH                         - 1) \* dstStride[0];
195. dst2[1] += ((c->dstH >> c->chrDstVSubSample) - 1) \* dstStride[1];
196. dst2[2] += ((c->dstH >> c->chrDstVSubSample) - 1) \* dstStride[2];
197. dst2[3] += ( c->dstH                         - 1) \* dstStride[3];
199. reset\_ptr(src2, c->srcFormat);
200. reset\_ptr((**void**\*)dst2, c->dstFormat);
202. /\* reset slice direction at end of frame \*/
203. **if** (!srcSliceY)
204. c->sliceDir = 0;
205. //关键：调用
206. ret = c->swscale(c, src2, srcStride2, c->srcH-srcSliceY-srcSliceH,
207. srcSliceH, dst2, dstStride2);
208. }

211. **if** (c->dstXYZ && !(c->srcXYZ && c->srcW==c->dstW && c->srcH==c->dstH)) {
212. /\* replace on the same data \*/
213. rgb48Toxyz12(c, (uint16\_t\*)dst2[0], (**const** uint16\_t\*)dst2[0], dstStride[0]/2, ret);
214. }
216. av\_free(rgb0\_tmp);
217. **return** ret;
218. }

从sws\_scale()的定义可以看出，它封装了SwsContext中的swscale()（注意这个函数中间没有“\_”）。函数最重要的一句代码就是“c->swscale()”。除此之外，函数还做了一些增加“兼容性”的一些处理。函数的主要步骤如下所示。

**1.检查输入的图像参数的合理性。**

这一步骤首先检查输入输出的参数是否为空，然后通过调用check\_image\_pointers()检查输入输出图像的内存是否正确分配。check\_image\_pointers()的定义如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44346687) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44346687)

[在CODE上查看代码片](https://code.csdn.net/snippets/621825)

1. **static** **int** check\_image\_pointers(**const** uint8\_t \* **const** data[4], **enum** AVPixelFormat pix\_fmt,
2. **const** **int** linesizes[4])
3. {
4. **const** AVPixFmtDescriptor \*desc = av\_pix\_fmt\_desc\_get(pix\_fmt);
5. **int** i;
7. **for** (i = 0; i < 4; i++) {
8. **int** plane = desc->comp[i].plane;
9. **if** (!data[plane] || !linesizes[plane])
10. **return** 0;
11. }
13. **return** 1;
14. }

从check\_image\_pointers()的定义可以看出，在特定像素格式前提下，如果该像素格式应该包含像素的分量为空，就返回0，否则返回1。  
**2.如果输入像素数据中使用了“调色板”（palette），则进行一些相应的处理。这一步通过函数usePal()来判定。**usePal()的定义如下。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44346687) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44346687)

[在CODE上查看代码片](https://code.csdn.net/snippets/621825)

1. **static** av\_always\_inline **int** usePal(**enum** AVPixelFormat pix\_fmt)
2. {
3. **const** AVPixFmtDescriptor \*desc = av\_pix\_fmt\_desc\_get(pix\_fmt);
4. av\_assert0(desc);
5. **return** (desc->flags & AV\_PIX\_FMT\_FLAG\_PAL) || (desc->flags & AV\_PIX\_FMT\_FLAG\_PSEUDOPAL);
6. }

从定义可以看出该函数通过判定AVPixFmtDescriptor中的flag是否包含AV\_PIX\_FMT\_FLAG\_PAL来断定像素格式是否使用了“调色板”。  
**3.其它一些特殊格式的处理，比如说Alpha，XYZ等的处理（这方面没有研究过）。  
4.如果输入的图像的扫描方式是从底部到顶部的（一般情况下是从顶部到底部），则将图像进行反转。  
5.调用SwsContext中的swscale()。**

## SwsContext中的swscale()

swscale这个变量的类型是SwsFunc，实际上就是一个函数指针。它是整个类库的**核心**。当我们从外部调用swscale()函数的时候，实际上就是调用了SwsContext中的这个名称为swscale的变量（注意外部函数接口和这个内部函数指针的名字是一样的，但不是一回事）。  
可以看一下SwsFunc这个类型的定义：

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44346687) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44346687)

[在CODE上查看代码片](https://code.csdn.net/snippets/621825)

1. **typedef** **int** (\*SwsFunc)(**struct** SwsContext \*context, **const** uint8\_t \*src[],
2. **int** srcStride[], **int** srcSliceY, **int** srcSliceH,
3. uint8\_t \*dst[], **int** dstStride[]);

可以看出SwsFunc的定义的参数类型和libswscale类库外部接口函数swscale()的参数类型一模一样。  
在libswscale中，该指针的指向可以分成2种情况：

1.图像没有伸缩的时候，指向专有的像素转换函数

2.图像有伸缩的时候，指向swscale()函数。

在调用sws\_getContext()初始化SwsContext的时候，会在其子函数sws\_init\_context()中对swscale指针进行赋值。如果图像没有进行拉伸，则会调用ff\_get\_unscaled\_swscale()对其进行赋值；如果图像进行了拉伸，则会调用ff\_getSwsFunc()对其进行赋值。下面分别看一下这2种情况。

## 没有拉伸--专有的像素转换函数

如果图像没有进行拉伸，则会调用ff\_get\_unscaled\_swscale()对SwsContext的swscale进行赋值。上篇文章中记录了这个函数，在这里回顾一下。

## ff\_get\_unscaled\_swscale()

ff\_get\_unscaled\_swscale()的定义如下。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44346687) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44346687)

[在CODE上查看代码片](https://code.csdn.net/snippets/621825)

1. **void** ff\_get\_unscaled\_swscale(SwsContext \*c)
2. {
3. **const** **enum** AVPixelFormat srcFormat = c->srcFormat;
4. **const** **enum** AVPixelFormat dstFormat = c->dstFormat;
5. **const** **int** flags = c->flags;
6. **const** **int** dstH = c->dstH;
7. **int** needsDither;
9. needsDither = isAnyRGB(dstFormat) &&
10. c->dstFormatBpp < 24 &&
11. (c->dstFormatBpp < c->srcFormatBpp || (!isAnyRGB(srcFormat)));
13. /\* yv12\_to\_nv12 \*/
14. **if** ((srcFormat == AV\_PIX\_FMT\_YUV420P || srcFormat == AV\_PIX\_FMT\_YUVA420P) &&
15. (dstFormat == AV\_PIX\_FMT\_NV12 || dstFormat == AV\_PIX\_FMT\_NV21)) {
16. c->swscale = planarToNv12Wrapper;
17. }
18. /\* nv12\_to\_yv12 \*/
19. **if** (dstFormat == AV\_PIX\_FMT\_YUV420P &&
20. (srcFormat == AV\_PIX\_FMT\_NV12 || srcFormat == AV\_PIX\_FMT\_NV21)) {
21. c->swscale = nv12ToPlanarWrapper;
22. }
23. /\* yuv2bgr \*/
24. **if** ((srcFormat == AV\_PIX\_FMT\_YUV420P || srcFormat == AV\_PIX\_FMT\_YUV422P ||
25. srcFormat == AV\_PIX\_FMT\_YUVA420P) && isAnyRGB(dstFormat) &&
26. !(flags & SWS\_ACCURATE\_RND) && (c->dither == SWS\_DITHER\_BAYER || c->dither == SWS\_DITHER\_AUTO) && !(dstH & 1)) {
27. c->swscale = ff\_yuv2rgb\_get\_func\_ptr(c);
28. }
30. **if** (srcFormat == AV\_PIX\_FMT\_YUV410P && !(dstH & 3) &&
31. (dstFormat == AV\_PIX\_FMT\_YUV420P || dstFormat == AV\_PIX\_FMT\_YUVA420P) &&
32. !(flags & SWS\_BITEXACT)) {
33. c->swscale = yvu9ToYv12Wrapper;
34. }
36. /\* bgr24toYV12 \*/
37. **if** (srcFormat == AV\_PIX\_FMT\_BGR24 &&
38. (dstFormat == AV\_PIX\_FMT\_YUV420P || dstFormat == AV\_PIX\_FMT\_YUVA420P) &&
39. !(flags & SWS\_ACCURATE\_RND))
40. c->swscale = bgr24ToYv12Wrapper;
42. /\* RGB/BGR -> RGB/BGR (no dither needed forms) \*/
43. **if** (isAnyRGB(srcFormat) && isAnyRGB(dstFormat) && findRgbConvFn(c)
44. && (!needsDither || (c->flags&(SWS\_FAST\_BILINEAR|SWS\_POINT))))
45. c->swscale = rgbToRgbWrapper;
47. **if** ((srcFormat == AV\_PIX\_FMT\_GBRP && dstFormat == AV\_PIX\_FMT\_GBRAP) ||
48. (srcFormat == AV\_PIX\_FMT\_GBRAP && dstFormat == AV\_PIX\_FMT\_GBRP))
49. c->swscale = planarRgbToplanarRgbWrapper;
51. #define isByteRGB(f) (             \
52. f == AV\_PIX\_FMT\_RGB32   || \
53. f == AV\_PIX\_FMT\_RGB32\_1 || \
54. f == AV\_PIX\_FMT\_RGB24   || \
55. f == AV\_PIX\_FMT\_BGR32   || \
56. f == AV\_PIX\_FMT\_BGR32\_1 || \
57. f == AV\_PIX\_FMT\_BGR24)
59. **if** (srcFormat == AV\_PIX\_FMT\_GBRP && isPlanar(srcFormat) && isByteRGB(dstFormat))
60. c->swscale = planarRgbToRgbWrapper;
62. **if** ((srcFormat == AV\_PIX\_FMT\_RGB48LE  || srcFormat == AV\_PIX\_FMT\_RGB48BE  ||
63. srcFormat == AV\_PIX\_FMT\_BGR48LE  || srcFormat == AV\_PIX\_FMT\_BGR48BE  ||
64. srcFormat == AV\_PIX\_FMT\_RGBA64LE || srcFormat == AV\_PIX\_FMT\_RGBA64BE ||
65. srcFormat == AV\_PIX\_FMT\_BGRA64LE || srcFormat == AV\_PIX\_FMT\_BGRA64BE) &&
66. (dstFormat == AV\_PIX\_FMT\_GBRP9LE  || dstFormat == AV\_PIX\_FMT\_GBRP9BE  ||
67. dstFormat == AV\_PIX\_FMT\_GBRP10LE || dstFormat == AV\_PIX\_FMT\_GBRP10BE ||
68. dstFormat == AV\_PIX\_FMT\_GBRP12LE || dstFormat == AV\_PIX\_FMT\_GBRP12BE ||
69. dstFormat == AV\_PIX\_FMT\_GBRP14LE || dstFormat == AV\_PIX\_FMT\_GBRP14BE ||
70. dstFormat == AV\_PIX\_FMT\_GBRP16LE || dstFormat == AV\_PIX\_FMT\_GBRP16BE ||
71. dstFormat == AV\_PIX\_FMT\_GBRAP16LE || dstFormat == AV\_PIX\_FMT\_GBRAP16BE ))
72. c->swscale = Rgb16ToPlanarRgb16Wrapper;
74. **if** ((srcFormat == AV\_PIX\_FMT\_GBRP9LE  || srcFormat == AV\_PIX\_FMT\_GBRP9BE  ||
75. srcFormat == AV\_PIX\_FMT\_GBRP16LE || srcFormat == AV\_PIX\_FMT\_GBRP16BE ||
76. srcFormat == AV\_PIX\_FMT\_GBRP10LE || srcFormat == AV\_PIX\_FMT\_GBRP10BE ||
77. srcFormat == AV\_PIX\_FMT\_GBRP12LE || srcFormat == AV\_PIX\_FMT\_GBRP12BE ||
78. srcFormat == AV\_PIX\_FMT\_GBRP14LE || srcFormat == AV\_PIX\_FMT\_GBRP14BE ||
79. srcFormat == AV\_PIX\_FMT\_GBRAP16LE || srcFormat == AV\_PIX\_FMT\_GBRAP16BE) &&
80. (dstFormat == AV\_PIX\_FMT\_RGB48LE  || dstFormat == AV\_PIX\_FMT\_RGB48BE  ||
81. dstFormat == AV\_PIX\_FMT\_BGR48LE  || dstFormat == AV\_PIX\_FMT\_BGR48BE  ||
82. dstFormat == AV\_PIX\_FMT\_RGBA64LE || dstFormat == AV\_PIX\_FMT\_RGBA64BE ||
83. dstFormat == AV\_PIX\_FMT\_BGRA64LE || dstFormat == AV\_PIX\_FMT\_BGRA64BE))
84. c->swscale = planarRgb16ToRgb16Wrapper;
86. **if** (av\_pix\_fmt\_desc\_get(srcFormat)->comp[0].depth\_minus1 == 7 &&
87. isPackedRGB(srcFormat) && dstFormat == AV\_PIX\_FMT\_GBRP)
88. c->swscale = rgbToPlanarRgbWrapper;
90. **if** (isBayer(srcFormat)) {
91. **if** (dstFormat == AV\_PIX\_FMT\_RGB24)
92. c->swscale = bayer\_to\_rgb24\_wrapper;
93. **else** **if** (dstFormat == AV\_PIX\_FMT\_YUV420P)
94. c->swscale = bayer\_to\_yv12\_wrapper;
95. **else** **if** (!isBayer(dstFormat)) {
96. av\_log(c, AV\_LOG\_ERROR, "unsupported bayer conversion\n");
97. av\_assert0(0);
98. }
99. }
101. /\* bswap 16 bits per pixel/component packed formats \*/
102. **if** (IS\_DIFFERENT\_ENDIANESS(srcFormat, dstFormat, AV\_PIX\_FMT\_BAYER\_BGGR16) ||
103. IS\_DIFFERENT\_ENDIANESS(srcFormat, dstFormat, AV\_PIX\_FMT\_BAYER\_RGGB16) ||
104. IS\_DIFFERENT\_ENDIANESS(srcFormat, dstFormat, AV\_PIX\_FMT\_BAYER\_GBRG16) ||
105. IS\_DIFFERENT\_ENDIANESS(srcFormat, dstFormat, AV\_PIX\_FMT\_BAYER\_GRBG16) ||
106. IS\_DIFFERENT\_ENDIANESS(srcFormat, dstFormat, AV\_PIX\_FMT\_BGR444) ||
107. IS\_DIFFERENT\_ENDIANESS(srcFormat, dstFormat, AV\_PIX\_FMT\_BGR48)  ||
108. IS\_DIFFERENT\_ENDIANESS(srcFormat, dstFormat, AV\_PIX\_FMT\_BGRA64) ||
109. IS\_DIFFERENT\_ENDIANESS(srcFormat, dstFormat, AV\_PIX\_FMT\_BGR555) ||
110. IS\_DIFFERENT\_ENDIANESS(srcFormat, dstFormat, AV\_PIX\_FMT\_BGR565) ||
111. IS\_DIFFERENT\_ENDIANESS(srcFormat, dstFormat, AV\_PIX\_FMT\_BGRA64) ||
112. IS\_DIFFERENT\_ENDIANESS(srcFormat, dstFormat, AV\_PIX\_FMT\_GRAY16) ||
113. IS\_DIFFERENT\_ENDIANESS(srcFormat, dstFormat, AV\_PIX\_FMT\_YA16)   ||
114. IS\_DIFFERENT\_ENDIANESS(srcFormat, dstFormat, AV\_PIX\_FMT\_GBRP9)  ||
115. IS\_DIFFERENT\_ENDIANESS(srcFormat, dstFormat, AV\_PIX\_FMT\_GBRP10) ||
116. IS\_DIFFERENT\_ENDIANESS(srcFormat, dstFormat, AV\_PIX\_FMT\_GBRP12) ||
117. IS\_DIFFERENT\_ENDIANESS(srcFormat, dstFormat, AV\_PIX\_FMT\_GBRP14) ||
118. IS\_DIFFERENT\_ENDIANESS(srcFormat, dstFormat, AV\_PIX\_FMT\_GBRP16) ||
119. IS\_DIFFERENT\_ENDIANESS(srcFormat, dstFormat, AV\_PIX\_FMT\_GBRAP16) ||
120. IS\_DIFFERENT\_ENDIANESS(srcFormat, dstFormat, AV\_PIX\_FMT\_RGB444) ||
121. IS\_DIFFERENT\_ENDIANESS(srcFormat, dstFormat, AV\_PIX\_FMT\_RGB48)  ||
122. IS\_DIFFERENT\_ENDIANESS(srcFormat, dstFormat, AV\_PIX\_FMT\_RGBA64) ||
123. IS\_DIFFERENT\_ENDIANESS(srcFormat, dstFormat, AV\_PIX\_FMT\_RGB555) ||
124. IS\_DIFFERENT\_ENDIANESS(srcFormat, dstFormat, AV\_PIX\_FMT\_RGB565) ||
125. IS\_DIFFERENT\_ENDIANESS(srcFormat, dstFormat, AV\_PIX\_FMT\_RGBA64) ||
126. IS\_DIFFERENT\_ENDIANESS(srcFormat, dstFormat, AV\_PIX\_FMT\_XYZ12)  ||
127. IS\_DIFFERENT\_ENDIANESS(srcFormat, dstFormat, AV\_PIX\_FMT\_YUV420P9)  ||
128. IS\_DIFFERENT\_ENDIANESS(srcFormat, dstFormat, AV\_PIX\_FMT\_YUV420P10) ||
129. IS\_DIFFERENT\_ENDIANESS(srcFormat, dstFormat, AV\_PIX\_FMT\_YUV420P12) ||
130. IS\_DIFFERENT\_ENDIANESS(srcFormat, dstFormat, AV\_PIX\_FMT\_YUV420P14) ||
131. IS\_DIFFERENT\_ENDIANESS(srcFormat, dstFormat, AV\_PIX\_FMT\_YUV420P16) ||
132. IS\_DIFFERENT\_ENDIANESS(srcFormat, dstFormat, AV\_PIX\_FMT\_YUV422P9)  ||
133. IS\_DIFFERENT\_ENDIANESS(srcFormat, dstFormat, AV\_PIX\_FMT\_YUV422P10) ||
134. IS\_DIFFERENT\_ENDIANESS(srcFormat, dstFormat, AV\_PIX\_FMT\_YUV422P12) ||
135. IS\_DIFFERENT\_ENDIANESS(srcFormat, dstFormat, AV\_PIX\_FMT\_YUV422P14) ||
136. IS\_DIFFERENT\_ENDIANESS(srcFormat, dstFormat, AV\_PIX\_FMT\_YUV422P16) ||
137. IS\_DIFFERENT\_ENDIANESS(srcFormat, dstFormat, AV\_PIX\_FMT\_YUV444P9)  ||
138. IS\_DIFFERENT\_ENDIANESS(srcFormat, dstFormat, AV\_PIX\_FMT\_YUV444P10) ||
139. IS\_DIFFERENT\_ENDIANESS(srcFormat, dstFormat, AV\_PIX\_FMT\_YUV444P12) ||
140. IS\_DIFFERENT\_ENDIANESS(srcFormat, dstFormat, AV\_PIX\_FMT\_YUV444P14) ||
141. IS\_DIFFERENT\_ENDIANESS(srcFormat, dstFormat, AV\_PIX\_FMT\_YUV444P16))
142. c->swscale = packed\_16bpc\_bswap;
144. **if** (usePal(srcFormat) && isByteRGB(dstFormat))
145. c->swscale = palToRgbWrapper;
147. **if** (srcFormat == AV\_PIX\_FMT\_YUV422P) {
148. **if** (dstFormat == AV\_PIX\_FMT\_YUYV422)
149. c->swscale = yuv422pToYuy2Wrapper;
150. **else** **if** (dstFormat == AV\_PIX\_FMT\_UYVY422)
151. c->swscale = yuv422pToUyvyWrapper;
152. }
154. /\* LQ converters if -sws 0 or -sws 4\*/
155. **if** (c->flags&(SWS\_FAST\_BILINEAR|SWS\_POINT)) {
156. /\* yv12\_to\_yuy2 \*/
157. **if** (srcFormat == AV\_PIX\_FMT\_YUV420P || srcFormat == AV\_PIX\_FMT\_YUVA420P) {
158. **if** (dstFormat == AV\_PIX\_FMT\_YUYV422)
159. c->swscale = planarToYuy2Wrapper;
160. **else** **if** (dstFormat == AV\_PIX\_FMT\_UYVY422)
161. c->swscale = planarToUyvyWrapper;
162. }
163. }
164. **if** (srcFormat == AV\_PIX\_FMT\_YUYV422 &&
165. (dstFormat == AV\_PIX\_FMT\_YUV420P || dstFormat == AV\_PIX\_FMT\_YUVA420P))
166. c->swscale = yuyvToYuv420Wrapper;
167. **if** (srcFormat == AV\_PIX\_FMT\_UYVY422 &&
168. (dstFormat == AV\_PIX\_FMT\_YUV420P || dstFormat == AV\_PIX\_FMT\_YUVA420P))
169. c->swscale = uyvyToYuv420Wrapper;
170. **if** (srcFormat == AV\_PIX\_FMT\_YUYV422 && dstFormat == AV\_PIX\_FMT\_YUV422P)
171. c->swscale = yuyvToYuv422Wrapper;
172. **if** (srcFormat == AV\_PIX\_FMT\_UYVY422 && dstFormat == AV\_PIX\_FMT\_YUV422P)
173. c->swscale = uyvyToYuv422Wrapper;
175. #define isPlanarGray(x) (isGray(x) && (x) != AV\_PIX\_FMT\_YA8 && (x) != AV\_PIX\_FMT\_YA16LE && (x) != AV\_PIX\_FMT\_YA16BE)
176. /\* simple copy \*/
177. **if** ( srcFormat == dstFormat ||
178. (srcFormat == AV\_PIX\_FMT\_YUVA420P && dstFormat == AV\_PIX\_FMT\_YUV420P) ||
179. (srcFormat == AV\_PIX\_FMT\_YUV420P && dstFormat == AV\_PIX\_FMT\_YUVA420P) ||
180. (isPlanarYUV(srcFormat) && isPlanarGray(dstFormat)) ||
181. (isPlanarYUV(dstFormat) && isPlanarGray(srcFormat)) ||
182. (isPlanarGray(dstFormat) && isPlanarGray(srcFormat)) ||
183. (isPlanarYUV(srcFormat) && isPlanarYUV(dstFormat) &&
184. c->chrDstHSubSample == c->chrSrcHSubSample &&
185. c->chrDstVSubSample == c->chrSrcVSubSample &&
186. dstFormat != AV\_PIX\_FMT\_NV12 && dstFormat != AV\_PIX\_FMT\_NV21 &&
187. srcFormat != AV\_PIX\_FMT\_NV12 && srcFormat != AV\_PIX\_FMT\_NV21))
188. {
189. **if** (isPacked(c->srcFormat))
190. c->swscale = packedCopyWrapper;
191. **else** /\* Planar YUV or gray \*/
192. c->swscale = planarCopyWrapper;
193. }
195. **if** (ARCH\_PPC)
196. ff\_get\_unscaled\_swscale\_ppc(c);
197. //     if (ARCH\_ARM)
198. //         ff\_get\_unscaled\_swscale\_arm(c);
199. }

从代码中可以看出，它根据输入输出像素格式的不同，选择了不同的转换函数。例如YUV420P转换NV12的时候，就会将planarToNv12Wrapper()赋值给SwsContext的swscale指针。

## 有拉伸--swscale()

如果图像进行了拉伸，则会调用ff\_getSwsFunc()对SwsContext的swscale进行赋值。上篇文章中记录了这个函数，在这里回顾一下。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44346687) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44346687)

[在CODE上查看代码片](https://code.csdn.net/snippets/621825)

1. SwsFunc ff\_getSwsFunc(SwsContext \*c)
2. {
3. sws\_init\_swscale(c);
5. **if** (ARCH\_PPC)
6. ff\_sws\_init\_swscale\_ppc(c);
7. **if** (ARCH\_X86)
8. ff\_sws\_init\_swscale\_x86(c);
10. **return** swscale;
11. }

注意，sws\_init\_context()对SwsContext的swscale进行赋值的语句是：

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44346687) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44346687)

[在CODE上查看代码片](https://code.csdn.net/snippets/621825)

1. c->swscale = ff\_getSwsFunc(c);

即把ff\_getSwsFunc()的返回值赋值给SwsContext的swscale指针；而ff\_getSwsFunc()的返回值是一个静态函数，名称就叫做“swscale”。  
下面我们看一下这个swscale()静态函数的定义。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44346687) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44346687)

[在CODE上查看代码片](https://code.csdn.net/snippets/621825)

1. **static** **int** swscale(SwsContext \*c, **const** uint8\_t \*src[],
2. **int** srcStride[], **int** srcSliceY,
3. **int** srcSliceH, uint8\_t \*dst[], **int** dstStride[])
4. {
5. /\* load a few things into local vars to make the code more readable?
6. \* and faster \*/
7. //注意一下这些参数
8. //以亮度为准
9. **const** **int** srcW                   = c->srcW;
10. **const** **int** dstW                   = c->dstW;
11. **const** **int** dstH                   = c->dstH;
12. //以色度为准
13. **const** **int** chrDstW                = c->chrDstW;
14. **const** **int** chrSrcW                = c->chrSrcW;
15. **const** **int** lumXInc                = c->lumXInc;
16. **const** **int** chrXInc                = c->chrXInc;
17. **const** **enum** AVPixelFormat dstFormat = c->dstFormat;
18. **const** **int** flags                  = c->flags;
19. int32\_t \*vLumFilterPos           = c->vLumFilterPos;
20. int32\_t \*vChrFilterPos           = c->vChrFilterPos;
21. int32\_t \*hLumFilterPos           = c->hLumFilterPos;
22. int32\_t \*hChrFilterPos           = c->hChrFilterPos;
23. int16\_t \*hLumFilter              = c->hLumFilter;
24. int16\_t \*hChrFilter              = c->hChrFilter;
25. int32\_t \*lumMmxFilter            = c->lumMmxFilter;
26. int32\_t \*chrMmxFilter            = c->chrMmxFilter;
27. **const** **int** vLumFilterSize         = c->vLumFilterSize;
28. **const** **int** vChrFilterSize         = c->vChrFilterSize;
29. **const** **int** hLumFilterSize         = c->hLumFilterSize;
30. **const** **int** hChrFilterSize         = c->hChrFilterSize;
31. int16\_t \*\*lumPixBuf              = c->lumPixBuf;
32. int16\_t \*\*chrUPixBuf             = c->chrUPixBuf;
33. int16\_t \*\*chrVPixBuf             = c->chrVPixBuf;
34. int16\_t \*\*alpPixBuf              = c->alpPixBuf;
35. **const** **int** vLumBufSize            = c->vLumBufSize;
36. **const** **int** vChrBufSize            = c->vChrBufSize;
37. uint8\_t \*formatConvBuffer        = c->formatConvBuffer;
38. uint32\_t \*pal                    = c->pal\_yuv;
39. yuv2planar1\_fn yuv2plane1        = c->yuv2plane1;
40. yuv2planarX\_fn yuv2planeX        = c->yuv2planeX;
41. yuv2interleavedX\_fn yuv2nv12cX   = c->yuv2nv12cX;
42. yuv2packed1\_fn yuv2packed1       = c->yuv2packed1;
43. yuv2packed2\_fn yuv2packed2       = c->yuv2packed2;
44. yuv2packedX\_fn yuv2packedX       = c->yuv2packedX;
45. yuv2anyX\_fn yuv2anyX             = c->yuv2anyX;
46. **const** **int** chrSrcSliceY           =                srcSliceY >> c->chrSrcVSubSample;
47. **const** **int** chrSrcSliceH           = FF\_CEIL\_RSHIFT(srcSliceH,   c->chrSrcVSubSample);
48. **int** should\_dither                = is9\_OR\_10BPS(c->srcFormat) ||
49. is16BPS(c->srcFormat);
50. **int** lastDstY;
52. /\* vars which will change and which we need to store back in the context \*/
53. **int** dstY         = c->dstY;
54. **int** lumBufIndex  = c->lumBufIndex;
55. **int** chrBufIndex  = c->chrBufIndex;
56. **int** lastInLumBuf = c->lastInLumBuf;
57. **int** lastInChrBuf = c->lastInChrBuf;
59. **if** (!usePal(c->srcFormat)) {
60. pal = c->input\_rgb2yuv\_table;
61. }
63. **if** (isPacked(c->srcFormat)) {
64. src[0] =
65. src[1] =
66. src[2] =
67. src[3] = src[0];
68. srcStride[0] =
69. srcStride[1] =
70. srcStride[2] =
71. srcStride[3] = srcStride[0];
72. }
73. srcStride[1] <<= c->vChrDrop;
74. srcStride[2] <<= c->vChrDrop;
76. DEBUG\_BUFFERS("swscale() %p[%d] %p[%d] %p[%d] %p[%d] -> %p[%d] %p[%d] %p[%d] %p[%d]\n",
77. src[0], srcStride[0], src[1], srcStride[1],
78. src[2], srcStride[2], src[3], srcStride[3],
79. dst[0], dstStride[0], dst[1], dstStride[1],
80. dst[2], dstStride[2], dst[3], dstStride[3]);
81. DEBUG\_BUFFERS("srcSliceY: %d srcSliceH: %d dstY: %d dstH: %d\n",
82. srcSliceY, srcSliceH, dstY, dstH);
83. DEBUG\_BUFFERS("vLumFilterSize: %d vLumBufSize: %d vChrFilterSize: %d vChrBufSize: %d\n",
84. vLumFilterSize, vLumBufSize, vChrFilterSize, vChrBufSize);
86. **if** (dstStride[0]&15 || dstStride[1]&15 ||
87. dstStride[2]&15 || dstStride[3]&15) {
88. **static** **int** warnedAlready = 0; // FIXME maybe move this into the context
89. **if** (flags & SWS\_PRINT\_INFO && !warnedAlready) {
90. av\_log(c, AV\_LOG\_WARNING,
91. "Warning: dstStride is not aligned!\n"
92. "         ->cannot do aligned memory accesses anymore\n");
93. warnedAlready = 1;
94. }
95. }
97. **if** (   (**uintptr\_t**)dst[0]&15 || (**uintptr\_t**)dst[1]&15 || (**uintptr\_t**)dst[2]&15
98. || (**uintptr\_t**)src[0]&15 || (**uintptr\_t**)src[1]&15 || (**uintptr\_t**)src[2]&15
99. || dstStride[0]&15 || dstStride[1]&15 || dstStride[2]&15 || dstStride[3]&15
100. || srcStride[0]&15 || srcStride[1]&15 || srcStride[2]&15 || srcStride[3]&15
101. ) {
102. **static** **int** warnedAlready=0;
103. **int** cpu\_flags = av\_get\_cpu\_flags();
104. **if** (HAVE\_MMXEXT && (cpu\_flags & AV\_CPU\_FLAG\_SSE2) && !warnedAlready){
105. av\_log(c, AV\_LOG\_WARNING, "Warning: data is not aligned! This can lead to a speedloss\n");
106. warnedAlready=1;
107. }
108. }
110. /\* Note the user might start scaling the picture in the middle so this
111. \* will not get executed. This is not really intended but works
112. \* currently, so people might do it. \*/
113. **if** (srcSliceY == 0) {
114. lumBufIndex  = -1;
115. chrBufIndex  = -1;
116. dstY         = 0;
117. lastInLumBuf = -1;
118. lastInChrBuf = -1;
119. }
121. **if** (!should\_dither) {
122. c->chrDither8 = c->lumDither8 = sws\_pb\_64;
123. }
124. lastDstY = dstY;
125. //逐行循环，一次循环代表处理一行
126. //注意dstY和dstH两个变量
127. **for** (; dstY < dstH; dstY++) {
128. //色度的和亮度之间的关系
129. **const** **int** chrDstY = dstY >> c->chrDstVSubSample;
130. uint8\_t \*dest[4]  = {
131. dst[0] + dstStride[0] \* dstY,
132. dst[1] + dstStride[1] \* chrDstY,
133. dst[2] + dstStride[2] \* chrDstY,
134. (CONFIG\_SWSCALE\_ALPHA && alpPixBuf) ? dst[3] + dstStride[3] \* dstY : NULL,
135. };
136. **int** use\_mmx\_vfilter= c->use\_mmx\_vfilter;
138. // First line needed as input
139. **const** **int** firstLumSrcY  = FFMAX(1 - vLumFilterSize, vLumFilterPos[dstY]);
140. **const** **int** firstLumSrcY2 = FFMAX(1 - vLumFilterSize, vLumFilterPos[FFMIN(dstY | ((1 << c->chrDstVSubSample) - 1), dstH - 1)]);
141. // First line needed as input
142. **const** **int** firstChrSrcY  = FFMAX(1 - vChrFilterSize, vChrFilterPos[chrDstY]);
144. // Last line needed as input
145. **int** lastLumSrcY  = FFMIN(c->srcH,    firstLumSrcY  + vLumFilterSize) - 1;
146. **int** lastLumSrcY2 = FFMIN(c->srcH,    firstLumSrcY2 + vLumFilterSize) - 1;
147. **int** lastChrSrcY  = FFMIN(c->chrSrcH, firstChrSrcY  + vChrFilterSize) - 1;
148. **int** enough\_lines;
150. // handle holes (FAST\_BILINEAR & weird filters)
151. **if** (firstLumSrcY > lastInLumBuf)
152. lastInLumBuf = firstLumSrcY - 1;
153. **if** (firstChrSrcY > lastInChrBuf)
154. lastInChrBuf = firstChrSrcY - 1;
155. av\_assert0(firstLumSrcY >= lastInLumBuf - vLumBufSize + 1);
156. av\_assert0(firstChrSrcY >= lastInChrBuf - vChrBufSize + 1);
158. DEBUG\_BUFFERS("dstY: %d\n", dstY);
159. DEBUG\_BUFFERS("\tfirstLumSrcY: %d lastLumSrcY: %d lastInLumBuf: %d\n",
160. firstLumSrcY, lastLumSrcY, lastInLumBuf);
161. DEBUG\_BUFFERS("\tfirstChrSrcY: %d lastChrSrcY: %d lastInChrBuf: %d\n",
162. firstChrSrcY, lastChrSrcY, lastInChrBuf);
164. // Do we have enough lines in this slice to output the dstY line
165. enough\_lines = lastLumSrcY2 < srcSliceY + srcSliceH &&
166. lastChrSrcY < FF\_CEIL\_RSHIFT(srcSliceY + srcSliceH, c->chrSrcVSubSample);
168. **if** (!enough\_lines) {
169. lastLumSrcY = srcSliceY + srcSliceH - 1;
170. lastChrSrcY = chrSrcSliceY + chrSrcSliceH - 1;
171. DEBUG\_BUFFERS("buffering slice: lastLumSrcY %d lastChrSrcY %d\n",
172. lastLumSrcY, lastChrSrcY);
173. }
175. // Do horizontal scaling
176. //水平拉伸
177. //亮度
178. **while** (lastInLumBuf < lastLumSrcY) {
179. **const** uint8\_t \*src1[4] = {
180. src[0] + (lastInLumBuf + 1 - srcSliceY) \* srcStride[0],
181. src[1] + (lastInLumBuf + 1 - srcSliceY) \* srcStride[1],
182. src[2] + (lastInLumBuf + 1 - srcSliceY) \* srcStride[2],
183. src[3] + (lastInLumBuf + 1 - srcSliceY) \* srcStride[3],
184. };
185. lumBufIndex++;
186. av\_assert0(lumBufIndex < 2 \* vLumBufSize);
187. av\_assert0(lastInLumBuf + 1 - srcSliceY < srcSliceH);
188. av\_assert0(lastInLumBuf + 1 - srcSliceY >= 0);
189. //关键：拉伸
190. hyscale(c, lumPixBuf[lumBufIndex], dstW, src1, srcW, lumXInc,
191. hLumFilter, hLumFilterPos, hLumFilterSize,
192. formatConvBuffer, pal, 0);
193. **if** (CONFIG\_SWSCALE\_ALPHA && alpPixBuf)
194. hyscale(c, alpPixBuf[lumBufIndex], dstW, src1, srcW,
195. lumXInc, hLumFilter, hLumFilterPos, hLumFilterSize,
196. formatConvBuffer, pal, 1);
197. lastInLumBuf++;
198. DEBUG\_BUFFERS("\t\tlumBufIndex %d: lastInLumBuf: %d\n",
199. lumBufIndex, lastInLumBuf);
200. }
201. //水平拉伸
202. //色度
203. **while** (lastInChrBuf < lastChrSrcY) {
204. **const** uint8\_t \*src1[4] = {
205. src[0] + (lastInChrBuf + 1 - chrSrcSliceY) \* srcStride[0],
206. src[1] + (lastInChrBuf + 1 - chrSrcSliceY) \* srcStride[1],
207. src[2] + (lastInChrBuf + 1 - chrSrcSliceY) \* srcStride[2],
208. src[3] + (lastInChrBuf + 1 - chrSrcSliceY) \* srcStride[3],
209. };
210. chrBufIndex++;
211. av\_assert0(chrBufIndex < 2 \* vChrBufSize);
212. av\_assert0(lastInChrBuf + 1 - chrSrcSliceY < (chrSrcSliceH));
213. av\_assert0(lastInChrBuf + 1 - chrSrcSliceY >= 0);
214. // FIXME replace parameters through context struct (some at least)
215. //关键：拉伸
216. **if** (c->needs\_hcscale)
217. hcscale(c, chrUPixBuf[chrBufIndex], chrVPixBuf[chrBufIndex],
218. chrDstW, src1, chrSrcW, chrXInc,
219. hChrFilter, hChrFilterPos, hChrFilterSize,
220. formatConvBuffer, pal);
221. lastInChrBuf++;
222. DEBUG\_BUFFERS("\t\tchrBufIndex %d: lastInChrBuf: %d\n",
223. chrBufIndex, lastInChrBuf);
224. }
225. // wrap buf index around to stay inside the ring buffer
226. **if** (lumBufIndex >= vLumBufSize)
227. lumBufIndex -= vLumBufSize;
228. **if** (chrBufIndex >= vChrBufSize)
229. chrBufIndex -= vChrBufSize;
230. **if** (!enough\_lines)
231. **break**;  // we can't output a dstY line so let's try with the next slice
233. #if HAVE\_MMX\_INLINE
234. updateMMXDitherTables(c, dstY, lumBufIndex, chrBufIndex,
235. lastInLumBuf, lastInChrBuf);
236. #endif
237. **if** (should\_dither) {
238. c->chrDither8 = ff\_dither\_8x8\_128[chrDstY & 7];
239. c->lumDither8 = ff\_dither\_8x8\_128[dstY    & 7];
240. }
241. **if** (dstY >= dstH - 2) {
242. /\* hmm looks like we can't use MMX here without overwriting
243. \* this array's tail \*/
244. ff\_sws\_init\_output\_funcs(c, &yuv2plane1, &yuv2planeX, &yuv2nv12cX,
245. &yuv2packed1, &yuv2packed2, &yuv2packedX, &yuv2anyX);
246. use\_mmx\_vfilter= 0;
247. }
249. {
250. **const** int16\_t \*\*lumSrcPtr  = (**const** int16\_t \*\*)(**void**\*) lumPixBuf  + lumBufIndex + firstLumSrcY - lastInLumBuf + vLumBufSize;
251. **const** int16\_t \*\*chrUSrcPtr = (**const** int16\_t \*\*)(**void**\*) chrUPixBuf + chrBufIndex + firstChrSrcY - lastInChrBuf + vChrBufSize;
252. **const** int16\_t \*\*chrVSrcPtr = (**const** int16\_t \*\*)(**void**\*) chrVPixBuf + chrBufIndex + firstChrSrcY - lastInChrBuf + vChrBufSize;
253. **const** int16\_t \*\*alpSrcPtr  = (CONFIG\_SWSCALE\_ALPHA && alpPixBuf) ?
254. (**const** int16\_t \*\*)(**void**\*) alpPixBuf + lumBufIndex + firstLumSrcY - lastInLumBuf + vLumBufSize : NULL;
255. int16\_t \*vLumFilter = c->vLumFilter;
256. int16\_t \*vChrFilter = c->vChrFilter;
258. **if** (isPlanarYUV(dstFormat) ||
259. (isGray(dstFormat) && !isALPHA(dstFormat))) { // YV12 like
260. **const** **int** chrSkipMask = (1 << c->chrDstVSubSample) - 1;
262. vLumFilter +=    dstY \* vLumFilterSize;
263. vChrFilter += chrDstY \* vChrFilterSize;
265. //                 av\_assert0(use\_mmx\_vfilter != (
266. //                                yuv2planeX == yuv2planeX\_10BE\_c
267. //                             || yuv2planeX == yuv2planeX\_10LE\_c
268. //                             || yuv2planeX == yuv2planeX\_9BE\_c
269. //                             || yuv2planeX == yuv2planeX\_9LE\_c
270. //                             || yuv2planeX == yuv2planeX\_16BE\_c
271. //                             || yuv2planeX == yuv2planeX\_16LE\_c
272. //                             || yuv2planeX == yuv2planeX\_8\_c) || !ARCH\_X86);
274. **if**(use\_mmx\_vfilter){
275. vLumFilter= (int16\_t \*)c->lumMmxFilter;
276. vChrFilter= (int16\_t \*)c->chrMmxFilter;
277. }
278. //输出一行水平拉伸过的像素
279. //亮度
280. //是否垂直拉伸？
281. **if** (vLumFilterSize == 1) {
282. //亮度-不垂直拉伸-分量模式（planar）-输出一行水平拉伸的像素
283. yuv2plane1(lumSrcPtr[0], dest[0], dstW, c->lumDither8, 0);
284. } **else** {
285. //亮度-垂直拉伸-分量模式（planar）-输出一行水平拉伸的像素
286. yuv2planeX(vLumFilter, vLumFilterSize,
287. lumSrcPtr, dest[0],
288. dstW, c->lumDither8, 0);
289. }
290. //色度
291. //是否垂直拉伸？
292. **if** (!((dstY & chrSkipMask) || isGray(dstFormat))) {
293. **if** (yuv2nv12cX) {
294. yuv2nv12cX(c, vChrFilter,
295. vChrFilterSize, chrUSrcPtr, chrVSrcPtr,
296. dest[1], chrDstW);
297. } **else** **if** (vChrFilterSize == 1) {
298. //色度-不垂直拉伸-分量模式（planar）-输出一行水平拉伸的像素
299. //注意是2个分量
300. yuv2plane1(chrUSrcPtr[0], dest[1], chrDstW, c->chrDither8, 0);
301. yuv2plane1(chrVSrcPtr[0], dest[2], chrDstW, c->chrDither8, 3);
302. } **else** {
303. //色度-垂直拉伸-分量模式（planar）-输出一行水平拉伸的像素
304. //注意是2个分量
305. yuv2planeX(vChrFilter,
306. vChrFilterSize, chrUSrcPtr, dest[1],
307. chrDstW, c->chrDither8, 0);
308. yuv2planeX(vChrFilter,
309. vChrFilterSize, chrVSrcPtr, dest[2],
310. chrDstW, c->chrDither8, use\_mmx\_vfilter ? (c->uv\_offx2 >> 1) : 3);
311. }
312. }
314. **if** (CONFIG\_SWSCALE\_ALPHA && alpPixBuf) {
315. **if**(use\_mmx\_vfilter){
316. vLumFilter= (int16\_t \*)c->alpMmxFilter;
317. }
318. **if** (vLumFilterSize == 1) {
319. yuv2plane1(alpSrcPtr[0], dest[3], dstW,
320. c->lumDither8, 0);
321. } **else** {
322. yuv2planeX(vLumFilter,
323. vLumFilterSize, alpSrcPtr, dest[3],
324. dstW, c->lumDither8, 0);
325. }
326. }
327. } **else** **if** (yuv2packedX) {
328. av\_assert1(lumSrcPtr  + vLumFilterSize - 1 < (**const** int16\_t \*\*)lumPixBuf  + vLumBufSize \* 2);
329. av\_assert1(chrUSrcPtr + vChrFilterSize - 1 < (**const** int16\_t \*\*)chrUPixBuf + vChrBufSize \* 2);
330. **if** (c->yuv2packed1 && vLumFilterSize == 1 &&
331. vChrFilterSize <= 2) { // unscaled RGB
332. **int** chrAlpha = vChrFilterSize == 1 ? 0 : vChrFilter[2 \* dstY + 1];
333. //不垂直拉伸-打包模式（packed）-输出一行水平拉伸的像素
334. yuv2packed1(c, \*lumSrcPtr, chrUSrcPtr, chrVSrcPtr,
335. alpPixBuf ? \*alpSrcPtr : NULL,
336. dest[0], dstW, chrAlpha, dstY);
337. } **else** **if** (c->yuv2packed2 && vLumFilterSize == 2 &&
338. vChrFilterSize == 2) { // bilinear upscale RGB
339. **int** lumAlpha = vLumFilter[2 \* dstY + 1];
340. **int** chrAlpha = vChrFilter[2 \* dstY + 1];
341. lumMmxFilter[2] =
342. lumMmxFilter[3] = vLumFilter[2 \* dstY]    \* 0x10001;
343. chrMmxFilter[2] =
344. chrMmxFilter[3] = vChrFilter[2 \* chrDstY] \* 0x10001;
345. yuv2packed2(c, lumSrcPtr, chrUSrcPtr, chrVSrcPtr,
346. alpPixBuf ? alpSrcPtr : NULL,
347. dest[0], dstW, lumAlpha, chrAlpha, dstY);
348. } **else** { // general RGB
349. //垂直拉伸-打包模式（packed）-输出一行水平拉伸的像素
350. yuv2packedX(c, vLumFilter + dstY \* vLumFilterSize,
351. lumSrcPtr, vLumFilterSize,
352. vChrFilter + dstY \* vChrFilterSize,
353. chrUSrcPtr, chrVSrcPtr, vChrFilterSize,
354. alpSrcPtr, dest[0], dstW, dstY);
355. }
356. } **else** {
357. av\_assert1(!yuv2packed1 && !yuv2packed2);
358. yuv2anyX(c, vLumFilter + dstY \* vLumFilterSize,
359. lumSrcPtr, vLumFilterSize,
360. vChrFilter + dstY \* vChrFilterSize,
361. chrUSrcPtr, chrVSrcPtr, vChrFilterSize,
362. alpSrcPtr, dest, dstW, dstY);
363. }
364. }
365. }
366. **if** (isPlanar(dstFormat) && isALPHA(dstFormat) && !alpPixBuf) {
367. **int** length = dstW;
368. **int** height = dstY - lastDstY;
370. **if** (is16BPS(dstFormat) || isNBPS(dstFormat)) {
371. **const** AVPixFmtDescriptor \*desc = av\_pix\_fmt\_desc\_get(dstFormat);
372. fillPlane16(dst[3], dstStride[3], length, height, lastDstY,
373. 1, desc->comp[3].depth\_minus1,
374. isBE(dstFormat));
375. } **else**
376. fillPlane(dst[3], dstStride[3], length, height, lastDstY, 255);
377. }
379. #if HAVE\_MMXEXT\_INLINE
380. **if** (av\_get\_cpu\_flags() & AV\_CPU\_FLAG\_MMXEXT)
381. \_\_asm\_\_ **volatile** ("sfence" ::: "memory");
382. #endif
383. emms\_c();
385. /\* store changed local vars back in the context \*/
386. c->dstY         = dstY;
387. c->lumBufIndex  = lumBufIndex;
388. c->chrBufIndex  = chrBufIndex;
389. c->lastInLumBuf = lastInLumBuf;
390. c->lastInChrBuf = lastInChrBuf;
392. **return** dstY - lastDstY;
393. }

可以看出swscale()是一行一行的进行图像缩放工作的。其中每行数据的处理按照“先水平拉伸，然后垂直拉伸”的方式进行处理。具体的实现函数如下所示：

1. 水平拉伸

a) 亮度水平拉伸：hyscale()

b) 色度水平拉伸：hcscale()

2. 垂直拉伸

a) Planar

i. 亮度垂直拉伸-不拉伸：yuv2plane1()

ii. 亮度垂直拉伸-拉伸：yuv2planeX()

iii. 色度垂直拉伸-不拉伸：yuv2plane1()

iv. 色度垂直拉伸-拉伸：yuv2planeX()

b) Packed

i. 垂直拉伸-不拉伸：yuv2packed1()

ii. 垂直拉伸-拉伸：yuv2packedX()

下面具体看看这几个函数的定义。

### hyscale()

水平亮度拉伸函数hyscale()的定义位于libswscale\swscale.c，如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44346687) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44346687)

[在CODE上查看代码片](https://code.csdn.net/snippets/621825)

1. // \*\*\* horizontal scale Y line to temp buffer
2. **static** av\_always\_inline **void** hyscale(SwsContext \*c, int16\_t \*dst, **int** dstWidth,
3. **const** uint8\_t \*src\_in[4],
4. **int** srcW, **int** xInc,
5. **const** int16\_t \*hLumFilter,
6. **const** int32\_t \*hLumFilterPos,
7. **int** hLumFilterSize,
8. uint8\_t \*formatConvBuffer,
9. uint32\_t \*pal, **int** isAlpha)
10. {
11. **void** (\*toYV12)(uint8\_t \*, **const** uint8\_t \*, **const** uint8\_t \*, **const** uint8\_t \*, **int**, uint32\_t \*) =
12. isAlpha ? c->alpToYV12 : c->lumToYV12;
13. **void** (\*convertRange)(int16\_t \*, **int**) = isAlpha ? NULL : c->lumConvertRange;
14. **const** uint8\_t \*src = src\_in[isAlpha ? 3 : 0];
16. **if** (toYV12) {
17. toYV12(formatConvBuffer, src, src\_in[1], src\_in[2], srcW, pal);
18. src = formatConvBuffer;
19. } **else** **if** (c->readLumPlanar && !isAlpha) {
20. //读取
21. c->readLumPlanar(formatConvBuffer, src\_in, srcW, c->input\_rgb2yuv\_table);
22. //赋值
23. src = formatConvBuffer;
24. } **else** **if** (c->readAlpPlanar && isAlpha) {
25. c->readAlpPlanar(formatConvBuffer, src\_in, srcW, NULL);
26. src = formatConvBuffer;
27. }
29. **if** (!c->hyscale\_fast) {
30. //亮度-水平拉伸
31. c->hyScale(c, dst, dstWidth, src, hLumFilter,
32. hLumFilterPos, hLumFilterSize);
33. } **else** { // fast bilinear upscale / crap downscale
34. c->hyscale\_fast(c, dst, dstWidth, src, srcW, xInc);
35. }
36. //如果需要取值范围的转换（0-255和16-235之间）
37. **if** (convertRange)
38. convertRange(dst, dstWidth);
39. }

从hyscale()的源代码可以看出，它的流程如下所示。  
**1.转换成Y（亮度）**  
如果SwsContext的toYV12()函数存在，调用用该函数将数据转换为Y。如果该函数不存在，则调用SwsContext的readLumPlanar()读取Y。  
**2.拉伸**  
拉伸通过SwsContext的hyScale ()函数完成。如果存在hyscale\_fast()方法的话，系统会优先调用hyscale\_fast()。  
**3.转换范围（如果需要的话）**  
如果需要转换亮度的取值范围（例如需要进行16-235的MPEG标准与0-255的JPEG标准之间的转换），则会调用SwsContext的lumConvertRange ()函数。  
上述几个步骤的涉及到的函数在上一篇文章中几经介绍过了，在这里重复一下。

#### toYV12() [SwsContext ->lumToYV12()]

toYV12()的实现函数是在ff\_sws\_init\_input\_funcs()中初始化的。在这里举几种具体的输入像素格式。

**输入格式为YUYV422/ YVYU422**  
ff\_sws\_init\_input\_funcs()中，输入像素格式为YUYV422/ YVYU422的时候，toYV12()指向yuy2ToY\_c()函数。源代码如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44346687) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44346687)

[在CODE上查看代码片](https://code.csdn.net/snippets/621825)

1. **case** AV\_PIX\_FMT\_YUYV422:
2. **case** AV\_PIX\_FMT\_YVYU422:
3. **case** AV\_PIX\_FMT\_YA8:
4. c->lumToYV12 = yuy2ToY\_c;
5. **break**;

yuy2ToY\_c()的定义如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44346687) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44346687)

[在CODE上查看代码片](https://code.csdn.net/snippets/621825)

1. **static** **void** yuy2ToY\_c(uint8\_t \*dst, **const** uint8\_t \*src, **const** uint8\_t \*unused1, **const** uint8\_t \*unused2,  **int** width,
2. uint32\_t \*unused)
3. {
4. **int** i;
5. **for** (i = 0; i < width; i++)
6. dst[i] = src[2 \* i];
7. }

从yuy2ToY\_c()的定义可以看出，该函数取出了所有的Y值（Y值在src[]数组中的下标为偶数）。  
  
  
**输入格式为RGB24**  
ff\_sws\_init\_input\_funcs()中，输入像素格式为RGB24的时候，toYV12()指向yuy2ToY\_c()函数。源代码如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44346687) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44346687)

[在CODE上查看代码片](https://code.csdn.net/snippets/621825)

1. **case** AV\_PIX\_FMT\_RGB24:
2. c->lumToYV12 = rgb24ToY\_c;
3. **break**;

rgb24ToY\_c()的定义如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44346687) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44346687)

[在CODE上查看代码片](https://code.csdn.net/snippets/621825)

1. **static** **void** rgb24ToY\_c(uint8\_t \*\_dst, **const** uint8\_t \*src, **const** uint8\_t \*unused1, **const** uint8\_t \*unused2, **int** width,
2. uint32\_t \*rgb2yuv)
3. {
4. int16\_t \*dst = (int16\_t \*)\_dst;
5. int32\_t ry = rgb2yuv[RY\_IDX], gy = rgb2yuv[GY\_IDX], by = rgb2yuv[BY\_IDX];
6. **int** i;
7. **for** (i = 0; i < width; i++) {
8. **int** r = src[i \* 3 + 0];
9. **int** g = src[i \* 3 + 1];
10. **int** b = src[i \* 3 + 2];
12. dst[i] = ((ry\*r + gy\*g + by\*b + (32<<(RGB2YUV\_SHIFT-1)) + (1<<(RGB2YUV\_SHIFT-7)))>>(RGB2YUV\_SHIFT-6));
13. }
14. }

从rgb24ToY\_c()的定义可以看出，该函数通过R、G、B三个元素计算Y的值。其中R、G、B的系数取自于数组rgb2yuv[]（这个地方还没有研究）；RGB2YUV\_SHIFT似乎代表了转换后YUV的位数，取值为15（这个地方也还没有深入看）。

#### SwsContext -> hyScale ()

SwsContext -> hyScale ()的实现函数是在sws\_init\_swscale ()中初始化的。可以回顾一下sws\_init\_swscale ()的定义，如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44346687) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44346687)

[在CODE上查看代码片](https://code.csdn.net/snippets/621825)

1. **static** av\_cold **void** sws\_init\_swscale(SwsContext \*c)
2. {
3. **enum** AVPixelFormat srcFormat = c->srcFormat;
5. ff\_sws\_init\_output\_funcs(c, &c->yuv2plane1, &c->yuv2planeX,
6. &c->yuv2nv12cX, &c->yuv2packed1,
7. &c->yuv2packed2, &c->yuv2packedX, &c->yuv2anyX);
9. ff\_sws\_init\_input\_funcs(c);

12. **if** (c->srcBpc == 8) {
13. **if** (c->dstBpc <= 14) {
14. c->hyScale = c->hcScale = hScale8To15\_c;
15. **if** (c->flags & SWS\_FAST\_BILINEAR) {
16. c->hyscale\_fast = ff\_hyscale\_fast\_c;
17. c->hcscale\_fast = ff\_hcscale\_fast\_c;
18. }
19. } **else** {
20. c->hyScale = c->hcScale = hScale8To19\_c;
21. }
22. } **else** {
23. c->hyScale = c->hcScale = c->dstBpc > 14 ? hScale16To19\_c
24. : hScale16To15\_c;
25. }
27. ff\_sws\_init\_range\_convert(c);
29. **if** (!(isGray(srcFormat) || isGray(c->dstFormat) ||
30. srcFormat == AV\_PIX\_FMT\_MONOBLACK || srcFormat == AV\_PIX\_FMT\_MONOWHITE))
31. c->needs\_hcscale = 1;
32. }

从sws\_init\_swscale ()的定义可以看出，ff\_sws\_init\_input\_funcs()和ff\_sws\_init\_range\_convert()之间的代码完成了hyScale()的初始化。根据srcBpc和dstBpc取值的不同，有几种不同的拉伸函数。根据我的理解，srcBpc代表了输入的每个像素单个分量的位数，dstBpc代表了输出的每个像素单个分量的位数。最常见的像素单个分量的位数是8位。从代码中可以看出，在输入像素单个分量的位数为8位，而且输出像素单个分量的位数也为8位的时候，SwsContext 的 hyScale ()会指向hScale8To15\_c()函数。

#### hScale8To15\_c()

hScale8To15\_c()的定义如下所示。有关这个方面的代码还没有详细研究，日后再作补充。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44346687) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44346687)

[在CODE上查看代码片](https://code.csdn.net/snippets/621825)

1. // bilinear / bicubic scaling
2. **static** **void** hScale8To15\_c(SwsContext \*c, int16\_t \*dst, **int** dstW,
3. **const** uint8\_t \*src, **const** int16\_t \*filter,
4. **const** int32\_t \*filterPos, **int** filterSize)
5. {
6. **int** i;
7. **for** (i = 0; i < dstW; i++) {
8. **int** j;
9. **int** srcPos = filterPos[i];
10. **int** val    = 0;
11. **for** (j = 0; j < filterSize; j++) {
12. val += ((**int**)src[srcPos + j]) \* filter[filterSize \* i + j];
13. }
14. dst[i] = FFMIN(val >> 7, (1 << 15) - 1); // the cubic equation does overflow ...
15. }
16. }

#### lumConvertRange () [SwsContext -> lumConvertRange()]

SwsContext -> hyScale ()的实现函数是在ff\_sws\_init\_range\_convert()中初始化的。可以回顾一下ff\_sws\_init\_range\_convert ()的定义，如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44346687) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44346687)

[在CODE上查看代码片](https://code.csdn.net/snippets/621825)

1. av\_cold **void** ff\_sws\_init\_range\_convert(SwsContext \*c)
2. {
3. c->lumConvertRange = NULL;
4. c->chrConvertRange = NULL;
5. **if** (c->srcRange != c->dstRange && !isAnyRGB(c->dstFormat)) {
6. **if** (c->dstBpc <= 14) {
7. **if** (c->srcRange) {
8. c->lumConvertRange = lumRangeFromJpeg\_c;
9. c->chrConvertRange = chrRangeFromJpeg\_c;
10. } **else** {
11. c->lumConvertRange = lumRangeToJpeg\_c;
12. c->chrConvertRange = chrRangeToJpeg\_c;
13. }
14. } **else** {
15. **if** (c->srcRange) {
16. c->lumConvertRange = lumRangeFromJpeg16\_c;
17. c->chrConvertRange = chrRangeFromJpeg16\_c;
18. } **else** {
19. c->lumConvertRange = lumRangeToJpeg16\_c;
20. c->chrConvertRange = chrRangeToJpeg16\_c;
21. }
22. }
23. }
24. }

SwsContext 的lumConvertRange()函数主要用于JPEG标准像素取值范围（0-255）和MPEG标准像素取值范围（16-235）之间的转换。有关这方面的分析在上一篇文章中一斤详细叙述过，在这里不再重复。简单看一下其中的一个函数。

#### lumRangeFromJpeg\_c()

把亮度从JPEG标准转换为MPEG标准（0-255转换为16-235）的函数lumRangeFromJpeg\_c()的定义如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44346687) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44346687)

[在CODE上查看代码片](https://code.csdn.net/snippets/621825)

1. **static** **void** lumRangeFromJpeg\_c(int16\_t \*dst, **int** width)
2. {
3. **int** i;
4. **for** (i = 0; i < width; i++)
5. dst[i] = (dst[i] \* 14071 + 33561947) >> 14;
6. }

其实这个函数就是做了一个（0-255）到（16-235）的映射。它将亮度值“0”映射成“16”，“255”映射成“235”，因此我们可以代入一个“255”看看转换后的数值是否为“235”。在这里需要注意，dst中存储的像素数值是15bit的亮度值。因此我们需要将8bit的数值“255”左移7位后带入。经过计算，255左移7位后取值为32640，计算后得到的数值为30080，右移7位后得到的8bit亮度值即为235。

### hcscale()

水平色度拉伸函数hcscale()的定义位于libswscale\swscale.c，如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44346687) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44346687)

[在CODE上查看代码片](https://code.csdn.net/snippets/621825)

1. **static** av\_always\_inline **void** hcscale(SwsContext \*c, int16\_t \*dst1,
2. int16\_t \*dst2, **int** dstWidth,
3. **const** uint8\_t \*src\_in[4],
4. **int** srcW, **int** xInc,
5. **const** int16\_t \*hChrFilter,
6. **const** int32\_t \*hChrFilterPos,
7. **int** hChrFilterSize,
8. uint8\_t \*formatConvBuffer, uint32\_t \*pal)
9. {
10. **const** uint8\_t \*src1 = src\_in[1], \*src2 = src\_in[2];
11. **if** (c->chrToYV12) {
12. uint8\_t \*buf2 = formatConvBuffer +
13. FFALIGN(srcW\*2+78, 16);
14. //转换
15. c->chrToYV12(formatConvBuffer, buf2, src\_in[0], src1, src2, srcW, pal);
16. src1= formatConvBuffer;
17. src2= buf2;
18. } **else** **if** (c->readChrPlanar) {
19. uint8\_t \*buf2 = formatConvBuffer +
20. FFALIGN(srcW\*2+78, 16);
21. //读取
22. c->readChrPlanar(formatConvBuffer, buf2, src\_in, srcW, c->input\_rgb2yuv\_table);
23. //赋值
24. src1 = formatConvBuffer;
25. src2 = buf2;
26. }
28. **if** (!c->hcscale\_fast) {
29. //色度-水平拉伸
30. c->hcScale(c, dst1, dstWidth, src1, hChrFilter, hChrFilterPos, hChrFilterSize);
31. c->hcScale(c, dst2, dstWidth, src2, hChrFilter, hChrFilterPos, hChrFilterSize);
32. } **else** { // fast bilinear upscale / crap downscale
33. c->hcscale\_fast(c, dst1, dst2, dstWidth, src1, src2, srcW, xInc);
34. }
35. //如果需要取值范围的转换（0-255和16-235之间）
36. **if** (c->chrConvertRange)
37. c->chrConvertRange(dst1, dst2, dstWidth);
38. }

从hcscale()的源代码可以看出，它的流程如下所示。  
**1.转换成UV**  
该功能通过SwsContext的chrToYV12 ()函数完成。如果该函数不存在，则调用SwsContext的readChrPlanar ()读取UV。

**2.拉伸**  
拉伸通过SwsContext的hcScale ()函数完成。如果存在hcscale\_fast()方法的话，系统会优先调用hcscale\_fast ()。

**3.转换范围（如果需要的话）**  
如果需要转换色度的取值范围（例如色度取值范围从0-255转换为16-240），则会调用SwsContext的chrConvertRange ()函数。

hcscale()的原理和hyScale ()的原理基本上是一样的，在这里既不再详细研究了。  
  
  
还有几个函数没有分析，但是时间有限，以后有机会再进行补充。

# [FFmpeg源代码简单分析：libavdevice的avdevice\_register\_all()](http://blog.csdn.net/leixiaohua1020/article/details/41211121)

本文简单记录一下FFmpeg中libavdevice注册设备的函数avdevice\_register\_all()。avdevice\_register\_all()在编程中的使用示例可以参考文章：

《[最简单的基于FFmpeg的AVDevice例子（读取摄像头）](http://blog.csdn.net/leixiaohua1020/article/details/39702113)》

在使用libavdevice之前，必须先运行avdevice\_register\_all()对设备进行注册，否则就会出错。avdevice\_register\_all()的注册方式和av\_register\_all()、avcodec\_register\_all()这几个函数是类似的。可以参考文章：

《[FFmpeg 源代码简单分析：av\_register\_all()](http://blog.csdn.net/leixiaohua1020/article/details/12677129)》

avdevice\_register\_all()代码如下。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/41211121) [copy](http://blog.csdn.net/leixiaohua1020/article/details/41211121)

[在CODE上查看代码片](https://code.csdn.net/snippets/627057)

1. /\*
2. \* 雷霄骅
3. \* http://blog.csdn.net/leixiaohua1020
4. \*/
6. #include "config.h"
7. #include "avdevice.h"
8. //输出设备
9. #define REGISTER\_OUTDEV(X, x)                                           \
10. {                                                                   \
11. **extern** AVOutputFormat ff\_##x##\_muxer;                           \
12. **if** (CONFIG\_##X##\_OUTDEV)                                        \
13. av\_register\_output\_format(&ff\_##x##\_muxer);                 \
14. }
15. //输入设备
16. #define REGISTER\_INDEV(X, x)                                            \
17. {                                                                   \
18. **extern** AVInputFormat ff\_##x##\_demuxer;                          \
19. **if** (CONFIG\_##X##\_INDEV)                                         \
20. av\_register\_input\_format(&ff\_##x##\_demuxer);                \
21. }
22. //输入输出设备
23. #define REGISTER\_INOUTDEV(X, x) REGISTER\_OUTDEV(X, x); REGISTER\_INDEV(X, x)
25. **void** avdevice\_register\_all(**void**)
26. {
27. **static** **int** initialized;
29. **if** (initialized)
30. **return**;
31. initialized = 1;
33. /\* devices \*/
34. REGISTER\_INOUTDEV(ALSA,             alsa);
35. REGISTER\_INDEV   (AVFOUNDATION,     avfoundation);
36. REGISTER\_INDEV   (BKTR,             bktr);
37. REGISTER\_OUTDEV  (CACA,             caca);
38. REGISTER\_OUTDEV  (DECKLINK,         decklink);
39. REGISTER\_INDEV   (DSHOW,            dshow);
40. REGISTER\_INDEV   (DV1394,           dv1394);
41. REGISTER\_INOUTDEV(FBDEV,            fbdev);
42. REGISTER\_INDEV   (GDIGRAB,          gdigrab);
43. REGISTER\_INDEV   (IEC61883,         iec61883);
44. REGISTER\_INDEV   (JACK,             jack);
45. REGISTER\_INDEV   (LAVFI,            lavfi);
46. REGISTER\_INDEV   (OPENAL,           openal);
47. REGISTER\_OUTDEV  (OPENGL,           opengl);
48. REGISTER\_INOUTDEV(OSS,              oss);
49. REGISTER\_INOUTDEV(PULSE,            pulse);
50. REGISTER\_INDEV   (QTKIT,            qtkit);
51. REGISTER\_OUTDEV  (SDL,              sdl);
52. REGISTER\_INOUTDEV(SNDIO,            sndio);
53. REGISTER\_INOUTDEV(V4L2,             v4l2);
54. //    REGISTER\_INDEV   (V4L,              v4l
55. REGISTER\_INDEV   (VFWCAP,           vfwcap);
56. REGISTER\_INDEV   (X11GRAB,          x11grab);
57. REGISTER\_OUTDEV  (XV,               xv);
59. /\* external libraries \*/
60. REGISTER\_INDEV   (LIBCDIO,          libcdio);
61. REGISTER\_INDEV   (LIBDC1394,        libdc1394);
62. }

从代码中可以看出，avdevice\_register\_all()调用3个函数进行设备组建的注册：REGISTER\_INDEV()，REGISTER\_OUTDEV()，REGISTER\_INOUTDEV()。上述3个函数实际上是预定义的3个宏：  
REGISTER\_INDEV()：注册输入设备。实际上调用了av\_register\_input\_format()将输入设备注册成一个AVInputFormat。  
REGISTER\_OUTDEV()：注册输出设备。实际上调用了av\_register\_output\_format()将输出设备注册成一个AVOutputFormat。  
REGISTER\_INOUTDEV()：注册输入设备和输出设备。实际上将上述两个宏定义合并了。

# [FFmpeg源代码简单分析：libavdevice的gdigrab](http://blog.csdn.net/leixiaohua1020/article/details/44597955)

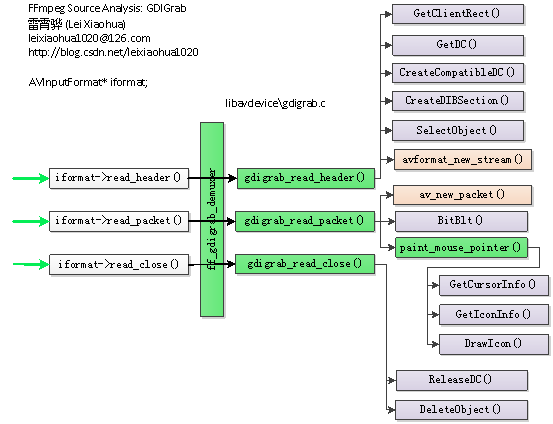
本文记录FFmpeg的libavdevice中GDIGrab组件的源代码。GDIGrab用于在Windows下屏幕录像（抓屏）。在ffmpeg.exe中使用可以参考文章：

[FFmpeg获取DirectShow设备数据（摄像头，录屏）](http://blog.csdn.net/leixiaohua1020/article/details/38284961)

编程使用可以参考文章：

[最简单的基于FFmpeg的AVDevice例子（屏幕录制）](http://blog.csdn.net/leixiaohua1020/article/details/39706721)

gdigrab的源代码位于libavdevice\gdigrab.c。关键函数的调用关系图如下图所示。图中绿色背景的函数代表源代码中自己声明的函数，紫色背景的函数代表Win32的API函数。



## ff\_gdigrab\_demuxer

在FFmpeg中Device也被当做是一种Format，因为GDIGrab是一个输入设备，因此被当作一个AVInputFormat。GDIGrab对应的AVInputFormat结构体如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44597955) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44597955)

[在CODE上查看代码片](https://code.csdn.net/snippets/627428)

1. AVInputFormat ff\_gdigrab\_demuxer = {
2. .name           = "gdigrab",
3. .long\_name      = NULL\_IF\_CONFIG\_SMALL("GDI API Windows frame grabber"),
4. .priv\_data\_size = **sizeof**(**struct** gdigrab),
5. .read\_header    = gdigrab\_read\_header,
6. .read\_packet    = gdigrab\_read\_packet,
7. .read\_close     = gdigrab\_read\_close,
8. .flags          = AVFMT\_NOFILE,
9. .priv\_class     = &gdigrab\_class,
10. };

从该结构体可以看出：

设备名称是“gdigrab”；  
设备完整名称是“GDI API Windows frame grabber”；  
初始化函数指针read\_header()指向gdigrab\_read\_header()；  
读取数据函数指针read\_packet()指向gdigrab\_read\_packet()；  
关闭函数指针read\_close()指向gdigrab\_read\_close()；  
Flags设置为AVFMT\_NOFILE；  
AVClass指定为gdigrab\_class。

下面分析一下这些数据。

## gdigrab\_class

ff\_gdigrab\_demuxer指定它的AVClass为一个名称为“gdigrab\_class”的静态变量。有关AVClass的概念之前已经记录过，在这里不再重复。gdigrab\_class的定义如下。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44597955) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44597955)

[在CODE上查看代码片](https://code.csdn.net/snippets/627428)

1. **static** **const** AVClass gdigrab\_class = {
2. .class\_name = "GDIgrab indev",
3. .item\_name  = av\_default\_item\_name,
4. .option     = options,
5. .version    = LIBAVUTIL\_VERSION\_INT,
6. };

从gdigrab\_class的定义可以看出，它指定了一个名称为“options”的数组作为它的选项数组（赋值给AVClass的option变量）。

### options

下面看一下这个options数组的定义，如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44597955) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44597955)

[在CODE上查看代码片](https://code.csdn.net/snippets/627428)

1. #define OFFSET(x) offsetof(struct gdigrab, x)
2. #define DEC AV\_OPT\_FLAG\_DECODING\_PARAM
3. **static** **const** AVOption options[] = {
4. { "draw\_mouse", "draw the mouse pointer", OFFSET(draw\_mouse), AV\_OPT\_TYPE\_INT, {.i64 = 1}, 0, 1, DEC },
5. { "show\_region", "draw border around capture area", OFFSET(show\_region), AV\_OPT\_TYPE\_INT, {.i64 = 0}, 0, 1, DEC },
6. { "framerate", "set video frame rate", OFFSET(framerate), AV\_OPT\_TYPE\_VIDEO\_RATE, {.str = "ntsc"}, 0, 0, DEC },
7. { "video\_size", "set video frame size", OFFSET(width), AV\_OPT\_TYPE\_IMAGE\_SIZE, {.str = NULL}, 0, 0, DEC },
8. { "offset\_x", "capture area x offset", OFFSET(offset\_x), AV\_OPT\_TYPE\_INT, {.i64 = 0}, INT\_MIN, INT\_MAX, DEC },
9. { "offset\_y", "capture area y offset", OFFSET(offset\_y), AV\_OPT\_TYPE\_INT, {.i64 = 0}, INT\_MIN, INT\_MAX, DEC },
10. { NULL },
11. };

options数组中包含了该Device支持的选项。可以看出GDIGrab支持如下选项：

draw\_mouse：画出鼠标指针。  
show\_region：划出抓屏区域的边界。  
framerate：抓屏帧率。  
video\_size：抓屏的大小。  
offset\_x：抓屏起始点x轴坐标。  
offset\_y：抓屏起始点y轴坐标。

从宏定义“#define OFFSET(x) offsetof(struct gdigrab, x)”中可以看出，这些选项都存储在一个名称为“gdigrab”的结构体中。

## Gdigrab 上下文结构体

Gdigrab上下文结构体中存储了GDIGrab设备用到的各种变量，定义如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44597955) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44597955)

[在CODE上查看代码片](https://code.csdn.net/snippets/627428)

1. /\*\*
2. \* GDI Device Demuxer context
3. \*/
4. **struct** gdigrab {
5. **const** AVClass \***class**;   /\*\*< Class for private options \*/
7. **int**        frame\_size;  /\*\*< Size in bytes of the frame pixel data \*/
8. **int**        header\_size; /\*\*< Size in bytes of the DIB header \*/
9. AVRational time\_base;   /\*\*< Time base \*/
10. int64\_t    time\_frame;  /\*\*< Current time \*/
12. **int**        draw\_mouse;  /\*\*< Draw mouse cursor (private option) \*/
13. **int**        show\_region; /\*\*< Draw border (private option) \*/
14. AVRational framerate;   /\*\*< Capture framerate (private option) \*/
15. **int**        width;       /\*\*< Width of the grab frame (private option) \*/
16. **int**        height;      /\*\*< Height of the grab frame (private option) \*/
17. **int**        offset\_x;    /\*\*< Capture x offset (private option) \*/
18. **int**        offset\_y;    /\*\*< Capture y offset (private option) \*/
20. **HWND**       hwnd;        /\*\*< Handle of the window for the grab \*/
21. **HDC**        source\_hdc;  /\*\*< Source device context \*/
22. **HDC**        dest\_hdc;    /\*\*< Destination, source-compatible DC \*/
23. BITMAPINFO bmi;         /\*\*< Information describing DIB format \*/
24. **HBITMAP**    hbmp;        /\*\*< Information on the bitmap captured \*/
25. **void**      \*buffer;      /\*\*< The buffer containing the bitmap image data \*/
26. RECT       clip\_rect;   /\*\*< The subarea of the screen or window to clip \*/
28. **HWND**       region\_hwnd; /\*\*< Handle of the region border window \*/
30. **int** cursor\_error\_printed;
31. };

## gdigrab\_read\_header()

gdigrab\_read\_header()用于初始化gdigrab。函数的定义如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44597955) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44597955)

[在CODE上查看代码片](https://code.csdn.net/snippets/627428)

1. /\*\*
2. \* Initializes the gdi grab device demuxer (public device demuxer API).
3. \*
4. \* @param s1 Context from avformat core
5. \* @return AVERROR\_IO error, 0 success
6. \*/
7. **static** **int**
8. gdigrab\_read\_header(AVFormatContext \*s1)
9. {
10. **struct** gdigrab \*gdigrab = s1->priv\_data;
11. //窗口句柄
12. **HWND** hwnd;
13. **HDC** source\_hdc = NULL;
14. **HDC** dest\_hdc   = NULL;
15. BITMAPINFO bmi;
16. **HBITMAP** hbmp   = NULL;
17. **void** \*buffer   = NULL;
19. **const** **char** \*filename = s1->filename;
20. **const** **char** \*name     = NULL;
21. AVStream   \*st       = NULL;
23. **int** bpp;
24. RECT virtual\_rect;
25. //窗口的位置和大小
26. RECT clip\_rect;
27. BITMAP bmp;
28. **int** ret;
29. //filename为窗口名称
30. **if** (!strncmp(filename, "title=", 6)) {
31. name = filename + 6;
32. //查找窗口的句柄
33. hwnd = FindWindow(NULL, name);
34. **if** (!hwnd) {
35. av\_log(s1, AV\_LOG\_ERROR,
36. "Can't find window '%s', aborting.\n", name);
37. ret = AVERROR(EIO);
38. **goto** error;
39. }
40. **if** (gdigrab->show\_region) {
41. av\_log(s1, AV\_LOG\_WARNING,
42. "Can't show region when grabbing a window.\n");
43. gdigrab->show\_region = 0;
44. }
45. //filename为desktop
46. } **else** **if** (!strcmp(filename, "desktop")) {
47. //窗口句柄为NULL
48. hwnd = NULL;
49. } **else** {
50. av\_log(s1, AV\_LOG\_ERROR,
51. "Please use \"desktop\" or \"title=<windowname>\" to specify your target.\n");
52. ret = AVERROR(EIO);
53. **goto** error;
54. }
56. **if** (hwnd) {
57. GetClientRect(hwnd, &virtual\_rect);
58. } **else** {
59. //窗口句柄为NULL，代表是全屏
60. virtual\_rect.left = GetSystemMetrics(SM\_XVIRTUALSCREEN);
61. virtual\_rect.top = GetSystemMetrics(SM\_YVIRTUALSCREEN);
62. virtual\_rect.right = virtual\_rect.left + GetSystemMetrics(SM\_CXVIRTUALSCREEN);
63. virtual\_rect.bottom = virtual\_rect.top + GetSystemMetrics(SM\_CYVIRTUALSCREEN);
64. }
66. /\* If no width or height set, use full screen/window area \*/
67. **if** (!gdigrab->width || !gdigrab->height) {
68. clip\_rect.left = virtual\_rect.left;
69. clip\_rect.top = virtual\_rect.top;
70. clip\_rect.right = virtual\_rect.right;
71. clip\_rect.bottom = virtual\_rect.bottom;
72. } **else** {
73. clip\_rect.left = gdigrab->offset\_x;
74. clip\_rect.top = gdigrab->offset\_y;
75. clip\_rect.right = gdigrab->width + gdigrab->offset\_x;
76. clip\_rect.bottom = gdigrab->height + gdigrab->offset\_y;
77. }
79. **if** (clip\_rect.left < virtual\_rect.left ||
80. clip\_rect.top < virtual\_rect.top ||
81. clip\_rect.right > virtual\_rect.right ||
82. clip\_rect.bottom > virtual\_rect.bottom) {
83. av\_log(s1, AV\_LOG\_ERROR,
84. "Capture area (%li,%li),(%li,%li) extends outside window area (%li,%li),(%li,%li)",
85. clip\_rect.left, clip\_rect.top,
86. clip\_rect.right, clip\_rect.bottom,
87. virtual\_rect.left, virtual\_rect.top,
88. virtual\_rect.right, virtual\_rect.bottom);
89. ret = AVERROR(EIO);
90. **goto** error;
91. }
93. /\* This will get the device context for the selected window, or if
94. \* none, the primary screen \*/
95. //得到某个窗口句柄的DC
96. source\_hdc = GetDC(hwnd);
97. **if** (!source\_hdc) {
98. WIN32\_API\_ERROR("Couldn't get window device context");
99. ret = AVERROR(EIO);
100. **goto** error;
101. }
102. bpp = GetDeviceCaps(source\_hdc, BITSPIXEL);
104. **if** (name) {
105. av\_log(s1, AV\_LOG\_INFO,
106. "Found window %s, capturing %lix%lix%i at (%li,%li)\n",
107. name,
108. clip\_rect.right - clip\_rect.left,
109. clip\_rect.bottom - clip\_rect.top,
110. bpp, clip\_rect.left, clip\_rect.top);
111. } **else** {
112. av\_log(s1, AV\_LOG\_INFO,
113. "Capturing whole desktop as %lix%lix%i at (%li,%li)\n",
114. clip\_rect.right - clip\_rect.left,
115. clip\_rect.bottom - clip\_rect.top,
116. bpp, clip\_rect.left, clip\_rect.top);
117. }
119. **if** (clip\_rect.right - clip\_rect.left <= 0 ||
120. clip\_rect.bottom - clip\_rect.top <= 0 || bpp%8) {
121. av\_log(s1, AV\_LOG\_ERROR, "Invalid properties, aborting\n");
122. ret = AVERROR(EIO);
123. **goto** error;
124. }
125. //创建一个与指定设备兼容的HDC
126. dest\_hdc = CreateCompatibleDC(source\_hdc);
127. **if** (!dest\_hdc) {
128. WIN32\_API\_ERROR("Screen DC CreateCompatibleDC");
129. ret = AVERROR(EIO);
130. **goto** error;
131. }
133. /\* Create a DIB and select it into the dest\_hdc \*/
134. //BMP
135. bmi.bmiHeader.biSize          = **sizeof**(BITMAPINFOHEADER);
136. bmi.bmiHeader.biWidth         = clip\_rect.right - clip\_rect.left;
137. bmi.bmiHeader.biHeight        = -(clip\_rect.bottom - clip\_rect.top);
138. bmi.bmiHeader.biPlanes        = 1;
139. bmi.bmiHeader.biBitCount      = bpp;
140. bmi.bmiHeader.biCompression   = BI\_RGB;
141. bmi.bmiHeader.biSizeImage     = 0;
142. bmi.bmiHeader.biXPelsPerMeter = 0;
143. bmi.bmiHeader.biYPelsPerMeter = 0;
144. bmi.bmiHeader.biClrUsed       = 0;
145. bmi.bmiHeader.biClrImportant  = 0;
146. hbmp = CreateDIBSection(dest\_hdc, &bmi, DIB\_RGB\_COLORS,
147. &buffer, NULL, 0);
148. **if** (!hbmp) {
149. WIN32\_API\_ERROR("Creating DIB Section");
150. ret = AVERROR(EIO);
151. **goto** error;
152. }
154. **if** (!SelectObject(dest\_hdc, hbmp)) {
155. WIN32\_API\_ERROR("SelectObject");
156. ret = AVERROR(EIO);
157. **goto** error;
158. }
160. /\* Get info from the bitmap \*/
161. GetObject(hbmp, **sizeof**(BITMAP), &bmp);
162. //创建AVStream
163. st = avformat\_new\_stream(s1, NULL);
164. **if** (!st) {
165. ret = AVERROR(ENOMEM);
166. **goto** error;
167. }
168. avpriv\_set\_pts\_info(st, 64, 1, 1000000); /\* 64 bits pts in us \*/
169. //保存信息到GDIGrab上下文结构体
170. gdigrab->frame\_size  = bmp.bmWidthBytes \* bmp.bmHeight \* bmp.bmPlanes;
171. gdigrab->header\_size = **sizeof**(BITMAPFILEHEADER) + **sizeof**(BITMAPINFOHEADER) +
172. (bpp <= 8 ? (1 << bpp) : 0) \* **sizeof**(RGBQUAD) /\* palette size \*/;
173. gdigrab->time\_base   = av\_inv\_q(gdigrab->framerate);
174. gdigrab->time\_frame  = av\_gettime() / av\_q2d(gdigrab->time\_base);
176. gdigrab->hwnd       = hwnd;
177. gdigrab->source\_hdc = source\_hdc;
178. gdigrab->dest\_hdc   = dest\_hdc;
179. gdigrab->hbmp       = hbmp;
180. gdigrab->bmi        = bmi;
181. gdigrab->buffer     = buffer;
182. gdigrab->clip\_rect  = clip\_rect;
184. gdigrab->cursor\_error\_printed = 0;
186. **if** (gdigrab->show\_region) {
187. **if** (gdigrab\_region\_wnd\_init(s1, gdigrab)) {
188. ret = AVERROR(EIO);
189. **goto** error;
190. }
191. }
193. st->codec->codec\_type = AVMEDIA\_TYPE\_VIDEO;
194. st->codec->codec\_id   = AV\_CODEC\_ID\_BMP;
195. st->codec->time\_base  = gdigrab->time\_base;
196. st->codec->bit\_rate   = (gdigrab->header\_size + gdigrab->frame\_size) \* 1/av\_q2d(gdigrab->time\_base) \* 8;
198. **return** 0;
200. error:
201. //如果出错了
202. **if** (source\_hdc)
203. ReleaseDC(hwnd, source\_hdc);
204. **if** (dest\_hdc)
205. DeleteDC(dest\_hdc);
206. **if** (hbmp)
207. DeleteObject(hbmp);
208. **if** (source\_hdc)
209. DeleteDC(source\_hdc);
210. **return** ret;
211. }

从源代码可以看出，gdigrab\_read\_header()的流程大致如下所示：  
（1）确定窗口的句柄hwnd。如果指定了“title=”的话，调用FindWindow()获取hwnd；如果指定了“desktop”，则设定hwnd为NULL。  
（2）根据窗口的句柄hwnd确定抓屏的矩形区域。如果抓取指定窗口，则通过GetClientRect()函数；否则就抓取整个屏幕。  
（3）调用GDI的API完成抓屏的一些初始化工作。包括：

a)通过GetDC()获得某个窗口句柄的HDC（在这里是source\_hdc）。  
b)通过CreateCompatibleDC()创建一个与指定设备兼容的HDC（在这里是dest\_hdc）  
c)通过CreateDIBSection()创建HBITMAP  
d)通过SelectObject()绑定HBITMAP和HDC（指的是dest\_hdc）

（4）通过avformat\_new\_stream()创建一个AVStream。  
（5）将初始化时候的一些参数保存至GDIGrab的上下文结构体。

## gdigrab\_read\_packet()

gdigrab\_read\_packet()用于读取一帧抓屏数据。该函数的定义如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44597955) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44597955)

[在CODE上查看代码片](https://code.csdn.net/snippets/627428)

1. /\*\*
2. \* Grabs a frame from gdi (public device demuxer API).
3. \*
4. \* @param s1 Context from avformat core
5. \* @param pkt Packet holding the grabbed frame
6. \* @return frame size in bytes
7. \*/
8. **static** **int** gdigrab\_read\_packet(AVFormatContext \*s1, AVPacket \*pkt)
9. {
10. **struct** gdigrab \*gdigrab = s1->priv\_data;
11. //读取参数
12. **HDC**        dest\_hdc   = gdigrab->dest\_hdc;
13. **HDC**        source\_hdc = gdigrab->source\_hdc;
14. RECT       clip\_rect  = gdigrab->clip\_rect;
15. AVRational time\_base  = gdigrab->time\_base;
16. int64\_t    time\_frame = gdigrab->time\_frame;
18. BITMAPFILEHEADER bfh;
19. **int** file\_size = gdigrab->header\_size + gdigrab->frame\_size;
21. int64\_t curtime, delay;
23. /\* Calculate the time of the next frame \*/
24. time\_frame += INT64\_C(1000000);
26. /\* Run Window message processing queue \*/
27. **if** (gdigrab->show\_region)
28. gdigrab\_region\_wnd\_update(s1, gdigrab);
30. /\* wait based on the frame rate \*/
31. //延时
32. **for** (;;) {
33. curtime = av\_gettime();
34. delay = time\_frame \* av\_q2d(time\_base) - curtime;
35. **if** (delay <= 0) {
36. **if** (delay < INT64\_C(-1000000) \* av\_q2d(time\_base)) {
37. time\_frame += INT64\_C(1000000);
38. }
39. **break**;
40. }
41. **if** (s1->flags & AVFMT\_FLAG\_NONBLOCK) {
42. **return** AVERROR(EAGAIN);
43. } **else** {
44. av\_usleep(delay);
45. }
46. }
47. //新建一个AVPacket
48. **if** (av\_new\_packet(pkt, file\_size) < 0)
49. **return** AVERROR(ENOMEM);
50. pkt->pts = curtime;
52. /\* Blit screen grab \*/
53. //关键：BitBlt()完成抓屏功能
54. **if** (!BitBlt(dest\_hdc, 0, 0,
55. clip\_rect.right - clip\_rect.left,
56. clip\_rect.bottom - clip\_rect.top,
57. source\_hdc,
58. clip\_rect.left, clip\_rect.top, SRCCOPY | CAPTUREBLT)) {
59. WIN32\_API\_ERROR("Failed to capture image");
60. **return** AVERROR(EIO);
61. }
62. //画鼠标指针？
63. **if** (gdigrab->draw\_mouse)
64. paint\_mouse\_pointer(s1, gdigrab);
66. /\* Copy bits to packet data \*/
67. //BMP文件头BITMAPFILEHEADER
68. bfh.bfType = 0x4d42; /\* "BM" in little-endian \*/
69. bfh.bfSize = file\_size;
70. bfh.bfReserved1 = 0;
71. bfh.bfReserved2 = 0;
72. bfh.bfOffBits = gdigrab->header\_size;
73. //往AVPacket中拷贝数据
74. //拷贝BITMAPFILEHEADER
75. memcpy(pkt->data, &bfh, **sizeof**(bfh));
76. //拷贝BITMAPINFOHEADER
77. memcpy(pkt->data + **sizeof**(bfh), &gdigrab->bmi.bmiHeader, **sizeof**(gdigrab->bmi.bmiHeader));
78. //不常见
79. **if** (gdigrab->bmi.bmiHeader.biBitCount <= 8)
80. GetDIBColorTable(dest\_hdc, 0, 1 << gdigrab->bmi.bmiHeader.biBitCount,
81. (RGBQUAD \*) (pkt->data + **sizeof**(bfh) + **sizeof**(gdigrab->bmi.bmiHeader)));
82. //拷贝像素数据
83. memcpy(pkt->data + gdigrab->header\_size, gdigrab->buffer, gdigrab->frame\_size);
85. gdigrab->time\_frame = time\_frame;
87. **return** gdigrab->header\_size + gdigrab->frame\_size;
88. }

从源代码可以看出，gdigrab\_read\_packet()的流程大致如下所示：  
（1）从GDIGrab上下文结构体读取初始化时候设定的参数。  
（2）根据帧率参数进行延时。  
（3）通过av\_new\_packet()新建一个AVPacket。  
（4）通过BitBlt()完成抓屏功能。  
（5）如果需要画鼠标指针的话，调用paint\_mouse\_pointer()，这里不做分析。  
（6）按照顺序拷贝以下3项内容至AVPacket的data指向的内存：

a)BITMAPFILEHEADER  
b)BITMAPINFOHEADER  
c)抓屏的到的像素数据

## gdigrab\_read\_close()

gdigrab\_read\_close()用于关闭gdigrab。该函数的定义如下所示。

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44597955) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44597955)

[在CODE上查看代码片](https://code.csdn.net/snippets/627428)

1. /\*\*
2. \* Closes gdi frame grabber (public device demuxer API).
3. \*
4. \* @param s1 Context from avformat core
5. \* @return 0 success, !0 failure
6. \*/
7. **static** **int** gdigrab\_read\_close(AVFormatContext \*s1)
8. {
9. **struct** gdigrab \*s = s1->priv\_data;
11. **if** (s->show\_region)
12. gdigrab\_region\_wnd\_destroy(s1, s);
14. **if** (s->source\_hdc)
15. ReleaseDC(s->hwnd, s->source\_hdc);
16. **if** (s->dest\_hdc)
17. DeleteDC(s->dest\_hdc);
18. **if** (s->hbmp)
19. DeleteObject(s->hbmp);
20. **if** (s->source\_hdc)
21. DeleteDC(s->source\_hdc);
23. **return** 0;
24. }

从源代码可以看出，gdigrab\_read\_close ()完成了各种变量的清理工作。

# [FFmpeg源代码简单分析：makefile](http://blog.csdn.net/leixiaohua1020/article/details/44556525)

本文记录FFmpeg的Makefile的源代码。Makefile用于编译FFmpeg的源代码。通过分析Makefile文件，可以了解FFmpeg的源代码生成的流程。有关Makefile这部分我本身基础不算很牢，很多地方还在慢慢摸索，所以分析的内容不能保证完全准确。以后有时间和其他朋友多交流再慢慢完善这篇文章。

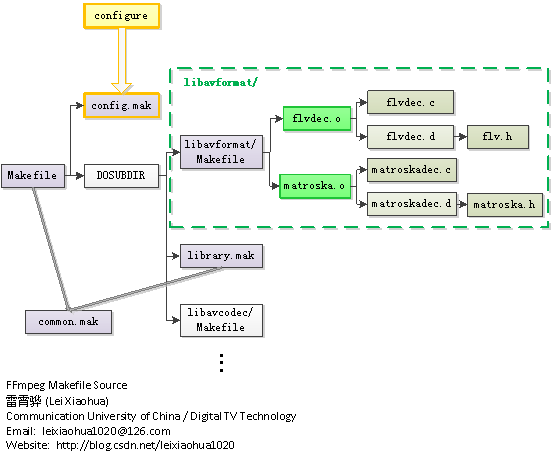
PS：make有一个很有用的命令：“make -n”。该选项会显示命令，但不会执行命令，十分有助于分析Makefile

## FFmpeg中与Makefile相关的文件

FFmpeg中与Makefile相关的文件主要有以下几个：

根目录Makefile：最基本的Makefile；  
config.mak：由configure生成的Makefile，保存了Configure的设置信息；  
libavXXXX/Makefile：每个类库的Makefile（仅仅设置了几个变量）；  
library.mak：编译类库的Makefile（和libavXXXX/Makefile配合使用）；  
common.mak：包含一些通用代码的Makefile；

它们之间的关系如下图所示。

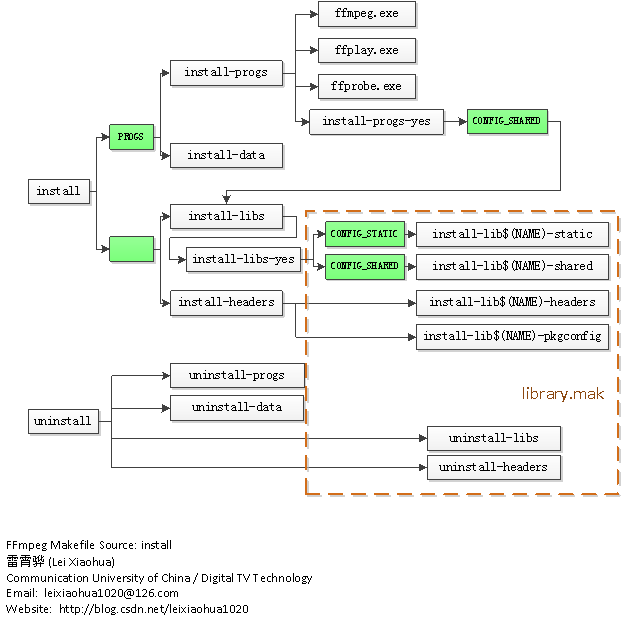


图中除了画出了Makefile之外，还画出了和Makefile有关的一些文件：

XXX.c：C语言文件；  
XXX.h：C语言文件用到的头文件；  
XXX.o：C语言文件对应的目标文件；  
XXX.d：C语言文件对应的依赖关系文件；

## Make Install 之间的关系

简单分析了一下Makefile中的make install之间的关系，如下图所示（使用的是MinGW编译器）。



从图中可以看出，install伪目标依赖于4个伪目标：

install-progs：安装应用程序ffmpeg.exe，ffplay.exe，ffprobe.exe；  
install-data：安装数据（\*.ffpreset之类的文件，没研究过）  
install-libs：安装类库（libavcodec.dll等文件）  
install-headers：安装头文件（libavcodec/avcodec.h等文件）

install-progs依赖于ffmpeg.exe，ffplay.exe，ffprobe.exe以及install-progs-yes伪目标。在CONFIG\_SHARED取值为yes的情况下，install-progs-yes依赖于install-libs。

install-libs依赖于伪目标install-libs-yes。install-libs-yes的依赖关系位于library.mak文件中。如果CONFIG\_STATIC取值为yes，install-libs-yes依赖于install-lib$(NAME)-static（其中${NAME}为类库文件名，例如avformat）；如果CONFIG\_SHARED取值为yes，install-libs-yes依赖于install-lib$(NAME)-shared。

install-headers依赖于伪目标install-lib$(NAME)-headers和install-lib$(NAME)-pkgconfig。

和install相对应，uninstall伪目标依赖于4个伪目标：

uninstall-progs：卸载应用程序；  
uninstall-data：卸载数据；  
uninstall-libs：卸载类库；  
uninstall-headers：卸载头文件；

其中uninstall-libs和uninstall-headers的依赖关系位于library.mak中。

## 根目录Makefile

根目录Makefile是最重要的。简单注释过的Makefile如下所示。

**[python]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44556525) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44556525)

[在CODE上查看代码片](https://code.csdn.net/snippets/625700)

1. # FFmpeg Main Makefile
2. #
3. # 注释：雷霄骅
4. # leixiaohua1020@126.com
5. # http://blog.csdn.net/leixiaohua1020
6. #
7. # FFmpeg的 Main Makefile。最重要。
9. MAIN\_MAKEFILE=1
10. #重要：包含了configure信息
11. include config.mak
13. #config.mak中：
14. #SRC\_PATH=.
15. #在SRC\_PATH搜索各种类型的文件
16. vpath %.c    $(SRC\_PATH)
17. vpath %.cpp  $(SRC\_PATH)
18. vpath %.h    $(SRC\_PATH)
19. vpath %.S    $(SRC\_PATH)
20. vpath %.asm  $(SRC\_PATH)
21. vpath %.v    $(SRC\_PATH)
22. vpath %.texi $(SRC\_PATH)
23. vpath %/fate\_config.sh.template $(SRC\_PATH)
25. #CONFIG\_XXX取值为yes
26. #PROGS-yes= ffmpeg ffplay ffprobe
27. PROGS-$(CONFIG\_FFMPEG)   += ffmpeg
28. PROGS-$(CONFIG\_FFPLAY)   += ffplay
29. PROGS-$(CONFIG\_FFPROBE)  += ffprobe
30. PROGS-$(CONFIG\_FFSERVER) += ffserver
32. #config.mak中：
33. #EXESUF=.exe
34. #PROGSSUF=
35. #$(var:%.c=%.o)意思是把.c 为结尾的变量替换成.o。
36. #没有“.c”的时候，代表匹配所有
37. PROGS      := $(PROGS-yes:%=%$(EXESUF))
38. INSTPROGS   = $(PROGS-yes:%=%$(PROGSSUF)$(EXESUF))
39. OBJS        = $(PROGS-yes:%=%.o) cmdutils.o
40. TESTTOOLS   = audiogen videogen rotozoom tiny\_psnr base64
41. HOSTPROGS  := $(TESTTOOLS:%=tests/%)
42. TOOLS       = qt-faststart trasher
43. TOOLS-$(CONFIG\_ZLIB) += cws2fws
45. #PROGS= ffmpeg.exe ffplay.exe ffprobe.exe
46. #INSTPROGS= ffmpeg.exe ffplay.exe ffprobe.exe
47. #OBJS= ffmpeg.o ffplay.o ffprobe.o
49. BASENAMES   = ffmpeg ffplay ffprobe ffserver
50. ALLPROGS    = $(BASENAMES:%=%$(PROGSSUF)$(EXESUF))
51. ALLPROGS\_G  = $(BASENAMES:%=%$(PROGSSUF)\_g$(EXESUF))
52. ALLMANPAGES = $(BASENAMES:%=%.1)
54. #ALLPROGS= ffmpeg.exe ffplay.exe ffprobe.exe ffserver.exe
55. #ALLPROGS\_G= ffmpeg\_g.exe ffplay\_g.exe ffprobe\_g.exe ffserver\_g.exe
56. #ALLMANPAGES=ffmpeg.1 ffplay.1 ffprobe.1 ffserver.1
57. FFLIBS-$(CONFIG\_AVDEVICE) += avdevice
58. FFLIBS-$(CONFIG\_AVFILTER) += avfilter
59. FFLIBS-$(CONFIG\_AVFORMAT) += avformat
60. FFLIBS-$(CONFIG\_AVCODEC)  += avcodec
61. FFLIBS-$(CONFIG\_POSTPROC) += postproc
62. FFLIBS-$(CONFIG\_SWRESAMPLE)+= swresample
63. FFLIBS-$(CONFIG\_SWSCALE)  += swscale
65. #FFLIBS-yes= avdevice avfilter avformat avcodec postproc swresample swscale
66. #一定需要libavutil
67. FFLIBS := avutil
68. #让通配符在变量中展开，需要使用wildcard关键字
69. DATA\_FILES := $(wildcard $(SRC\_PATH)/presets/\*.ffpreset) $(SRC\_PATH)/doc/ffprobe.xsd
71. SKIPHEADERS = cmdutils\_common\_opts.h
72. #重要
73. include $(SRC\_PATH)/common.mak
74. #依赖的类库
75. FF\_EXTRALIBS := $(FFEXTRALIBS)
76. FF\_DEP\_LIBS  := $(DEP\_LIBS)
78. #伪目标
79. #all是最关键的，生成最后的程序
80. #all: ffmpeg.exe ffplay.exe ffprobe.exe
81. #
82. all: $(PROGS)
83. #config.mak中：
84. #EXESUF=.exe
85. #PROGSSUF=
86. #$@是一个自动化变量。可以简单理解为目标的集合。
87. #$<是一个自动化变量。可以简单理解为依赖目标的集合。
88. #%是通配符
89. #两个冒号，“静态模式规则”。
90. #ffmpeg\_g.exe生成ffmpeg.exe；ffplay\_g.exe生成ffplay.exe；ffprobe\_g.exe生成ffprobe.exe
91. #strip经常用来去除目标文件中的一些符号表、调试符号表信息,以减小程序的大小
92. $(PROGS): %$(EXESUF): %$(PROGSSUF)\_g$(EXESUF)
93. $(CP) $< $@$(PROGSSUF)
94. $(STRIP) $@$(PROGSSUF)
96. $(TOOLS): %$(EXESUF): %.o
97. $(LD) $(LDFLAGS) -o $@ $< $(ELIBS)
99. tools/cws2fws$(EXESUF): ELIBS = -lz
101. config.h: .config
102. .config: $(wildcard $(FFLIBS:%=$(SRC\_PATH)/lib%/all\*.c))
103. @-tput bold 2>/dev/null
104. @-printf '\nWARNING: $(?F) newer than config.h, rerun configure\n\n'
105. @-tput sgr0 2>/dev/null
106. #给子目录中的Makefile使用的变量
107. SUBDIR\_VARS := OBJS FFLIBS CLEANFILES DIRS TESTPROGS EXAMPLES SKIPHEADERS \
108. ALTIVEC-OBJS MMX-OBJS NEON-OBJS X86-OBJS YASM-OBJS-FFT YASM-OBJS \
109. HOSTPROGS BUILT\_HEADERS TESTOBJS ARCH\_HEADERS ARMV6-OBJS TOOLS
111. define RESET
112. $(1) :=
113. $(1)-yes :=
114. endef
116. #$(call <expression>,<parm1>,<parm2>,<parm3>...)
117. #当make执行这个函数时，<expression>参数中的变量，如$(1)，$(2)，$(3)等，会被参数
118. #<parm1>，<parm2>，<parm3>依次取代。而<expression>的返回值就是call函数的返回值。
120. #命令包===========================
121. #用于编译每个库
122. #$(1)取值为libavcodec，libavcodec等等
123. define DOSUBDIR
124. $(foreach V,$(SUBDIR\_VARS),$(eval $(call RESET,$(V))))
125. SUBDIR := $(1)/
126. #每个库目录下的Makefile
127. include $(SRC\_PATH)/$(1)/Makefile
128. #注：make一般情况下如果在中途检测到有执行出错的情况(返回非 0 状态)，那么就会放弃对当前规则后续命令的执行。
129. #在命令前面加上“-”号之后，就算执行错误了，也会继续执行下去
130. -include $(SRC\_PATH)/$(1)/$(ARCH)/Makefile
131. #编译类库
132. include $(SRC\_PATH)/library.mak
133. endef
134. #=================================
136. #$(foreach <var>,<list>,<text>)
137. #把参数<list>中的单词逐一取出放到参数<var>所指定的变量中，
138. #然后再执行<text>所包含的表达式。
139. #$(eval text)
140. #text的内容将作为makefile的一部分而被make解析和执行
141. #
142. #循环调用DOSUBDIR命令包
143. #这一步会将libavcodec，libavformat等文件夹下的Makefile包含进来。
144. $(foreach D,$(FFLIBS),$(eval $(call DOSUBDIR,lib$(D))))
145. #ffplay需要SDL
146. ffplay.o: CFLAGS += $(SDL\_CFLAGS)
147. ffplay\_g$(EXESUF): FF\_EXTRALIBS += $(SDL\_LIBS)
148. ffserver\_g$(EXESUF): LDFLAGS += $(FFSERVERLDFLAGS)
149. #链接生成ffmpeg\_g.exe等等
150. #FF\_DEP\_LIBS= libavcodec/libavcodec.a libavutil/libavutil.a ....
151. %$(PROGSSUF)\_g$(EXESUF): %.o cmdutils.o $(FF\_DEP\_LIBS)
152. $(LD) $(LDFLAGS) -o $@ $< cmdutils.o $(FF\_EXTRALIBS)
154. OBJDIRS += tools
156. -include $(wildcard tools/\*.d)
158. VERSION\_SH  = $(SRC\_PATH)/version.sh
159. GIT\_LOG     = $(SRC\_PATH)/.git/logs/HEAD
161. .version: $(wildcard $(GIT\_LOG)) $(VERSION\_SH) config.mak
162. .version: M=@
164. version.h .version:
165. $(M)$(VERSION\_SH) $(SRC\_PATH) version.h $(EXTRA\_VERSION)
166. $(Q)touch .version
168. # force version.sh to run whenever version might have changed
169. -include .version
170. #安装install
171. #安装程序
172. ifdef PROGS
173. install: install-progs install-data
174. endif
175. #安装类库和头文件
176. install: install-libs install-headers
177. #install-libs-yes位于library.mak
178. install-libs: install-libs-yes
180. install-progs-yes:
181. install-progs-$(CONFIG\_SHARED): install-libs
183. #config.mak中：
184. #BINDIR=$(DESTDIR)${prefix}/bin
185. #INSTALL=install
186. #cp与install区别：
187. #cp会先清空文件后往里写入新文件，而install则会先删除掉原先的文件然后写入新文件。
188. install-progs: install-progs-yes $(PROGS)
189. $(Q)mkdir -p "$(BINDIR)"
190. $(INSTALL) -c -m 755 $(INSTPROGS) "$(BINDIR)"
192. install-data: $(DATA\_FILES)
193. $(Q)mkdir -p "$(DATADIR)"
194. $(INSTALL) -m 644 $(DATA\_FILES) "$(DATADIR)"
195. #卸载
196. uninstall: uninstall-libs uninstall-headers uninstall-progs uninstall-data
197. #addprefix()用于加前缀
198. #在这里获取ffmpeg.exe等的完整路径（用于删除）
199. uninstall-progs:
200. $(RM) $(addprefix "$(BINDIR)/", $(ALLPROGS))
202. uninstall-data:
203. $(RM) -r "$(DATADIR)"
204. #清空
205. clean::
206. $(RM) $(ALLPROGS) $(ALLPROGS\_G)
207. $(RM) $(CLEANSUFFIXES)
208. $(RM) $(TOOLS)
209. $(RM) $(CLEANSUFFIXES:%=tools/%)
210. $(RM) coverage.info
211. $(RM) -r coverage-html
213. distclean::
214. $(RM) $(DISTCLEANSUFFIXES)
215. $(RM) config.\* .version version.h libavutil/avconfig.h
217. config:
218. $(SRC\_PATH)/configure $(value FFMPEG\_CONFIGURATION)
220. # Without the sed genthml thinks "libavutil" and "./libavutil" are two different things
221. coverage.info: $(wildcard \*.gcda \*.gcno \*/\*.gcda \*/\*.gcno \*/\*/\*.gcda \*/\*/\*.gcno)
222. $(Q)lcov -c -d . -b . | sed -e 's#/./#/#g' > $@
224. coverage-html: coverage.info
225. $(Q)mkdir -p $@
226. $(Q)genhtml -o $@ $<
227. $(Q)touch $@
229. include $(SRC\_PATH)/doc/Makefile
230. include $(SRC\_PATH)/tests/Makefile
232. $(sort $(OBJDIRS)):
233. $(Q)mkdir -p $@
235. # Dummy rule to stop make trying to rebuild removed or renamed headers
236. %.h:
237. @:
239. # Disable suffix rules.  Most of the builtin rules are suffix rules,
240. # so this saves some time on slow systems.
241. .SUFFIXES:
242. #显示地指明一个目标是“伪目标”
243. .PHONY: all all-yes alltools \*clean config examples install\*
244. .PHONY: testprogs uninstall\*

根目录Makefile代码一开始的时候包含了config.mak文件。这个文件是运行./configure的后生成的配置文件，包含了所有的配置信息。

随后代码定义了ffplay.exe，ffmpeg.exe，ffprobe.exe与ffplay\_g.exe，ffmpeg\_g.exe，ffprobe\_g.exe的依赖关系。然后定义了ffplay\_g.exe，ffmpeg\_g.exe，ffprobe\_g.exe与libavformat，libavcodec等这些类库的依赖关系。

根目录Makefile中也定义了all，install，uninstall，clean等等一系列的伪目标，这样可以通过给Makefile指定不同的目标来完成不同的事。

此外根目录的Makefile中有一个很重要的命令包DOSUBDIR。在该命令包通过包含libavXXX/Makefile和library.mak等文件，定义了FFmpeg类库（例如libavformat，libavcodec，libavutil等）的依赖关系。

## config.mak

config.mak文件是运行./configure的后生成的配置文件，包含了所有的配置信息。简单注释过的config.mak的代码如下所示。

**[python]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44556525) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44556525)

[在CODE上查看代码片](https://code.csdn.net/snippets/625700)

1. # FFmpeg config.mak
2. #
3. # 注释：雷霄骅
4. # leixiaohua1020@126.com
5. # http://blog.csdn.net/leixiaohua1020
6. #
7. # Configure脚本生成的Makefile，包含了各种配置信息。
8. #
9. # Automatically generated by configure - do not modify!
10. #基本信息
11. ifndef FFMPEG\_CONFIG\_MAK
12. FFMPEG\_CONFIG\_MAK=1
13. FFMPEG\_CONFIGURATION=
14. #各种路径========================================
15. prefix=/usr/local
16. LIBDIR=$(DESTDIR)${prefix}/lib
17. SHLIBDIR=$(DESTDIR)${prefix}/bin
18. INCDIR=$(DESTDIR)${prefix}/include
19. BINDIR=$(DESTDIR)${prefix}/bin
20. DATADIR=$(DESTDIR)${prefix}/share/ffmpeg
21. MANDIR=$(DESTDIR)${prefix}/share/man
22. #是个相对路径
23. SRC\_PATH=.
24. ifndef MAIN\_MAKEFILE
25. SRC\_PATH:=$(SRC\_PATH:.%=..%)
26. endif
27. #工具集==========================================
28. CC\_IDENT=gcc 4.6.2 (GCC)
29. #架构
30. ARCH=x86
31. #编译器
32. CC=gcc
33. CXX=g++
34. AS=gcc
35. #链接器
36. LD=gcc
37. DEPCC=gcc
38. #汇编器
39. YASM=yasm
40. YASMDEP=yasm
41. #生成静态库.a工具
42. AR=ar
43. RANLIB=ranlib
44. CP=cp -p
45. LN\_S=ln -sf
46. STRIP=strip
47. #参数集==========================================
48. #编译器的参数
49. CPPFLAGS= -D\_ISOC99\_SOURCE -D\_FILE\_OFFSET\_BITS=64 -D\_LARGEFILE\_SOURCE -U\_\_STRICT\_ANSI\_\_
50. CFLAGS=   -std=c99 -fno-common -fomit-frame-pointer -I/include/SDL -D\_GNU\_SOURCE=1 -Dmain=SDL\_main -g -Wdeclaration-after-statement -Wall -Wno-parentheses -Wno-switch -Wno-format-zero-length -Wdisabled-optimization -Wpointer-arith -Wredundant-decls -Wno-pointer-sign -Wcast-qual -Wwrite-strings -Wtype-limits -Wundef -Wmissing-prototypes -Wno-pointer-to-int-cast -Wstrict-prototypes -O3 -fno-math-errno -fno-signed-zeros -fno-tree-vectorize -Werror=implicit-function-declaration -Werror=missing-prototypes
51. CXXFLAGS=  -D\_\_STDC\_CONSTANT\_MACROS
52. ASFLAGS=   -g
53. #目标文件有关的参数
54. AS\_O=-o $@
55. CC\_O=-o $@
56. CXX\_O=-o $@
57. #链接器有关的参数
58. LDFLAGS= -Wl,--as-needed -Wl,--warn-common -Wl,-rpath-link=libpostproc:libswresample:libswscale:libavfilter:libavdevice:libavformat:libavcodec:libavutil
59. FFSERVERLDFLAGS=-Wl,-E
60. SHFLAGS=-shared -Wl,--output-**def**,$$(@:$(SLIBSUF)=.**def**) -Wl,--out-implib,$(SUBDIR)lib$(SLIBNAME:$(SLIBSUF)=.dll.a) -Wl,--enable-runtime-pseudo-reloc -Wl,--enable-auto-image-base -Wl,-Bsymbolic -Wl,--version-script,$(SUBDIR)lib$(NAME).ver
61. YASMFLAGS=-f win32  -DPREFIX
62. #前缀后缀========================================
63. BUILDSUF=
64. PROGSSUF=
65. #${NAME}位于每个liavXXX/Makefile中，例如avformat
66. FULLNAME=$(NAME)$(BUILDSUF)
67. LIBPREF=lib
68. LIBSUF=.a
69. #例如libavformat.a
70. LIBNAME=$(LIBPREF)$(FULLNAME)$(LIBSUF)
71. SLIBPREF=
72. SLIBSUF=.dll
73. EXESUF=.exe
74. EXTRA\_VERSION=
75. DEPFLAGS=$(CPPFLAGS) $(CFLAGS) -MM
76. CCDEP=
77. CXXDEP=$(DEPCC) $(DEPFLAGS) $< | sed -e "/^\#.\*/d" -e "s,^[[:space:]]\*$(\*F)\\.o,$(@D)/$(\*F).o," > $(@:.o=.d)
78. ASDEP=
79. CC\_DEPFLAGS=-MMD -MF $(@:.o=.d) -MT $@
80. AS\_DEPFLAGS=-MMD -MF $(@:.o=.d) -MT $@
81. HOSTCC=gcc
82. HOSTCFLAGS=-D\_ISOC99\_SOURCE -O3 -g -std=c99 -Wall
83. HOSTEXESUF=.exe
84. HOSTLDFLAGS=
85. HOSTLIBS=-lm
86. TARGET\_EXEC=
87. TARGET\_PATH=$(CURDIR)
88. #SDL
89. SDL\_LIBS=-L/lib -lmingw32 -lSDLmain -lSDL -mwindows
90. SDL\_CFLAGS=-I/include/SDL -D\_GNU\_SOURCE=1 -Dmain=SDL\_main
91. LIB\_INSTALL\_EXTRA\_CMD=$$(RANLIB) "$(LIBDIR)/$(LIBNAME)"
92. #链接
93. EXTRALIBS=-lavicap32 -lws2\_32 -L/lib -lmingw32 -lSDLmain -lSDL -mwindows -lm -lz -lpsapi
94. INSTALL=install
95. LIBTARGET=i386
96. #例如libavformat.dll
97. SLIBNAME=$(SLIBPREF)$(FULLNAME)$(SLIBSUF)
98. #LIBVERSION变量位于library.mak
99. #例如libavformat-53.dll
100. #生成的Dll似乎就是这个版本的
101. SLIBNAME\_WITH\_VERSION=$(SLIBPREF)$(FULLNAME)-$(LIBVERSION)$(SLIBSUF)
102. #例如libavformat-53.31.100.dll
103. SLIBNAME\_WITH\_MAJOR=$(SLIBPREF)$(FULLNAME)-$(LIBMAJOR)$(SLIBSUF)
104. SLIB\_CREATE\_DEF\_CMD=
105. #生成导出库lib，会调用lib.exe
106. SLIB\_EXTRA\_CMD=-lib.exe /machine:$(LIBTARGET) /**def**:$$(@:$(SLIBSUF)=.**def**) /out:$(SUBDIR)$(SLIBNAME:$(SLIBSUF)=.lib)
107. SLIB\_INSTALL\_NAME=$(SLIBNAME\_WITH\_MAJOR)
108. SLIB\_INSTALL\_LINKS=
109. SLIB\_INSTALL\_EXTRA\_LIB=lib$(SLIBNAME:$(SLIBSUF)=.dll.a) $(SLIBNAME\_WITH\_MAJOR:$(SLIBSUF)=.**def**)
110. SLIB\_INSTALL\_EXTRA\_SHLIB=$(SLIBNAME:$(SLIBSUF)=.lib)
111. SAMPLES:=$(FATE\_SAMPLES)
112. NOREDZONE\_FLAGS=-mno-red-zone
113. #版本信息========================================
114. libavcodec\_VERSION=53.60.100
115. libavcodec\_VERSION\_MAJOR=53
116. libavdevice\_VERSION=53.4.100
117. libavdevice\_VERSION\_MAJOR=53
118. libavfilter\_VERSION=2.60.100
119. libavfilter\_VERSION\_MAJOR=2
120. libavformat\_VERSION=53.31.100
121. libavformat\_VERSION\_MAJOR=53
122. libavutil\_VERSION=51.34.101
123. libavutil\_VERSION\_MAJOR=51
124. libpostproc\_VERSION=52.0.100
125. libpostproc\_VERSION\_MAJOR=52
126. libswresample\_VERSION=0.6.100
127. libswresample\_VERSION\_MAJOR=0
128. libswscale\_VERSION=2.1.100
129. libswscale\_VERSION\_MAJOR=2
130. #组件配置========================================
131. #ARCH\_
132. !ARCH\_ALPHA=yes
133. !ARCH\_ARM=yes
134. !ARCH\_AVR32=yes
135. !ARCH\_AVR32\_AP=yes
136. !ARCH\_AVR32\_UC=yes
137. !ARCH\_BFIN=yes
138. !ARCH\_IA64=yes
139. !ARCH\_M68K=yes
140. !ARCH\_MIPS=yes
141. !ARCH\_MIPS64=yes
142. !ARCH\_PARISC=yes
143. !ARCH\_PPC=yes
144. !ARCH\_PPC64=yes
145. !ARCH\_S390=yes
146. !ARCH\_SH4=yes
147. !ARCH\_SPARC=yes
148. !ARCH\_SPARC64=yes
149. !ARCH\_TOMI=yes
150. ARCH\_X86=yes
151. ARCH\_X86\_32=yes
152. !ARCH\_X86\_64=yes
153. #HAVE\_
154. !HAVE\_ALTIVEC=yes
155. HAVE\_AMD3DNOW=yes
156. HAVE\_AMD3DNOWEXT=yes
157. !HAVE\_ARMV5TE=yes
158. !HAVE\_ARMV6=yes
159. !HAVE\_ARMV6T2=yes
160. !HAVE\_ARMVFP=yes
161. HAVE\_AVX=yes
162. !HAVE\_IWMMXT=yes
163. !HAVE\_MMI=yes
164. HAVE\_MMX=yes
165. HAVE\_MMX2=yes
166. !HAVE\_NEON=yes
167. !HAVE\_PPC4XX=yes
168. HAVE\_SSE=yes
169. HAVE\_SSSE3=yes
170. !HAVE\_VFPV3=yes
171. !HAVE\_VIS=yes
172. !HAVE\_BIGENDIAN=yes
173. HAVE\_FAST\_UNALIGNED=yes
174. !HAVE\_PTHREADS=yes
175. HAVE\_W32THREADS=yes
176. !HAVE\_OS2THREADS=yes
177. HAVE\_ALIGNED\_STACK=yes
178. !HAVE\_ALSA\_ASOUNDLIB\_H=yes
179. !HAVE\_ALTIVEC\_H=yes
180. !HAVE\_ARPA\_INET\_H=yes
181. !HAVE\_ASM\_MOD\_Y=yes
182. !HAVE\_ASM\_TYPES\_H=yes
183. HAVE\_ATTRIBUTE\_MAY\_ALIAS=yes
184. HAVE\_ATTRIBUTE\_PACKED=yes
185. HAVE\_CBRTF=yes
186. HAVE\_CLOSESOCKET=yes
187. !HAVE\_CMOV=yes
188. !HAVE\_DCBZL=yes
189. !HAVE\_DEV\_BKTR\_IOCTL\_BT848\_H=yes
190. !HAVE\_DEV\_BKTR\_IOCTL\_METEOR\_H=yes
191. !HAVE\_DEV\_IC\_BT8XX\_H=yes
192. !HAVE\_DEV\_VIDEO\_BKTR\_IOCTL\_BT848\_H=yes
193. !HAVE\_DEV\_VIDEO\_METEOR\_IOCTL\_METEOR\_H=yes
194. !HAVE\_DLFCN\_H=yes
195. !HAVE\_DLOPEN=yes
196. HAVE\_DOS\_PATHS=yes
197. HAVE\_EBP\_AVAILABLE=yes
198. HAVE\_EBX\_AVAILABLE=yes
199. HAVE\_EXP2=yes
200. HAVE\_EXP2F=yes
201. !HAVE\_FAST\_64BIT=yes
202. HAVE\_FAST\_CLZ=yes
203. !HAVE\_FAST\_CMOV=yes
204. !HAVE\_FCNTL=yes
205. !HAVE\_FORK=yes
206. !HAVE\_GETADDRINFO=yes
207. !HAVE\_GETHRTIME=yes
208. HAVE\_GETPROCESSAFFINITYMASK=yes
209. HAVE\_GETPROCESSMEMORYINFO=yes
210. HAVE\_GETPROCESSTIMES=yes
211. !HAVE\_GETRUSAGE=yes
212. HAVE\_GNU\_AS=yes
213. !HAVE\_IBM\_ASM=yes
214. !HAVE\_INET\_ATON=yes
215. HAVE\_INLINE\_ASM=yes
216. HAVE\_ISATTY=yes
217. HAVE\_KBHIT=yes
218. !HAVE\_LDBRX=yes
219. HAVE\_LLRINT=yes
220. HAVE\_LLRINTF=yes
221. HAVE\_LOCAL\_ALIGNED\_16=yes
222. HAVE\_LOCAL\_ALIGNED\_8=yes
223. !HAVE\_LOCALTIME\_R=yes
224. HAVE\_LOG2=yes
225. HAVE\_LOG2F=yes
226. !HAVE\_LOONGSON=yes
227. HAVE\_LRINT=yes
228. HAVE\_LRINTF=yes
229. !HAVE\_LZO1X\_999\_COMPRESS=yes
230. !HAVE\_MACHINE\_IOCTL\_BT848\_H=yes
231. !HAVE\_MACHINE\_IOCTL\_METEOR\_H=yes
232. HAVE\_MAKEINFO=yes
233. HAVE\_MALLOC\_H=yes
234. HAVE\_MAPVIEWOFFILE=yes
235. !HAVE\_MEMALIGN=yes
236. !HAVE\_MKSTEMP=yes
237. !HAVE\_MMAP=yes
238. HAVE\_PEEKNAMEDPIPE=yes
239. !HAVE\_POLL\_H=yes
240. !HAVE\_POSIX\_MEMALIGN=yes
241. HAVE\_ROUND=yes
242. HAVE\_ROUNDF=yes
243. !HAVE\_SCHED\_GETAFFINITY=yes
244. HAVE\_SDL=yes
245. HAVE\_SDL\_VIDEO\_SIZE=yes
246. HAVE\_SETMODE=yes
247. !HAVE\_SETRLIMIT=yes
248. !HAVE\_SNDIO\_H=yes
249. HAVE\_SOCKLEN\_T=yes
250. !HAVE\_SOUNDCARD\_H=yes
251. !HAVE\_STRERROR\_R=yes
252. !HAVE\_STRPTIME=yes
253. HAVE\_STRUCT\_ADDRINFO=yes
254. HAVE\_STRUCT\_IPV6\_MREQ=yes
255. !HAVE\_STRUCT\_RUSAGE\_RU\_MAXRSS=yes
256. HAVE\_STRUCT\_SOCKADDR\_IN6=yes
257. !HAVE\_STRUCT\_SOCKADDR\_SA\_LEN=yes
258. HAVE\_STRUCT\_SOCKADDR\_STORAGE=yes
259. !HAVE\_STRUCT\_V4L2\_FRMIVALENUM\_DISCRETE=yes
260. HAVE\_SYMVER=yes
261. HAVE\_SYMVER\_ASM\_LABEL=yes
262. !HAVE\_SYMVER\_GNU\_ASM=yes
263. !HAVE\_SYSCONF=yes
264. !HAVE\_SYSCTL=yes
265. !HAVE\_SYS\_MMAN\_H=yes
266. HAVE\_SYS\_PARAM\_H=yes
267. !HAVE\_SYS\_RESOURCE\_H=yes
268. !HAVE\_SYS\_SELECT\_H=yes
269. !HAVE\_SYS\_SOUNDCARD\_H=yes
270. !HAVE\_SYS\_VIDEOIO\_H=yes
271. !HAVE\_TERMIOS\_H=yes
272. HAVE\_THREADS=yes
273. HAVE\_TRUNC=yes
274. HAVE\_TRUNCF=yes
275. !HAVE\_VFP\_ARGS=yes
276. HAVE\_VIRTUALALLOC=yes
277. HAVE\_WINSOCK2\_H=yes
278. !HAVE\_XFORM\_ASM=yes
279. !HAVE\_XMM\_CLOBBERS=yes
280. HAVE\_YASM=yes
281. #CONFIG\_
282. CONFIG\_BSFS=yes
283. CONFIG\_DECODERS=yes
284. CONFIG\_DEMUXERS=yes
285. CONFIG\_ENCODERS=yes
286. CONFIG\_FILTERS=yes
287. !CONFIG\_HWACCELS=yes
288. CONFIG\_INDEVS=yes
289. CONFIG\_MUXERS=yes
290. CONFIG\_OUTDEVS=yes
291. CONFIG\_PARSERS=yes
292. CONFIG\_PROTOCOLS=yes
293. CONFIG\_FFPLAY=yes
294. CONFIG\_FFPROBE=yes
295. !CONFIG\_FFSERVER=yes
296. CONFIG\_FFMPEG=yes
297. !CONFIG\_AVPLAY=yes
298. !CONFIG\_AVPROBE=yes
299. !CONFIG\_AVSERVER=yes
300. CONFIG\_AANDCT=yes
301. CONFIG\_AC3DSP=yes
302. CONFIG\_AVCODEC=yes
303. CONFIG\_AVDEVICE=yes
304. CONFIG\_AVFILTER=yes
305. CONFIG\_AVFORMAT=yes
306. !CONFIG\_AVISYNTH=yes
307. !CONFIG\_BZLIB=yes
308. !CONFIG\_CRYSTALHD=yes
309. CONFIG\_DCT=yes
310. !CONFIG\_DOC=yes
311. CONFIG\_DWT=yes
312. !CONFIG\_DXVA2=yes
313. CONFIG\_FASTDIV=yes
314. CONFIG\_FFT=yes
315. !CONFIG\_FREI0R=yes
316. !CONFIG\_GNUTLS=yes
317. CONFIG\_GOLOMB=yes
318. !CONFIG\_GPL=yes
319. !CONFIG\_GRAY=yes
320. CONFIG\_H264CHROMA=yes
321. CONFIG\_H264DSP=yes
322. CONFIG\_H264PRED=yes
323. !CONFIG\_HARDCODED\_TABLES=yes
324. CONFIG\_HUFFMAN=yes
325. !CONFIG\_LIBAACPLUS=yes
326. !CONFIG\_LIBASS=yes
327. !CONFIG\_LIBCDIO=yes
328. !CONFIG\_LIBCELT=yes
329. !CONFIG\_LIBDC1394=yes
330. !CONFIG\_LIBDIRAC=yes
331. !CONFIG\_LIBFAAC=yes
332. !CONFIG\_LIBFREETYPE=yes
333. !CONFIG\_LIBGSM=yes
334. !CONFIG\_LIBMODPLUG=yes
335. !CONFIG\_LIBMP3LAME=yes
336. !CONFIG\_LIBNUT=yes
337. !CONFIG\_LIBOPENCORE\_AMRNB=yes
338. !CONFIG\_LIBOPENCORE\_AMRWB=yes
339. !CONFIG\_LIBOPENCV=yes
340. !CONFIG\_LIBOPENJPEG=yes
341. !CONFIG\_LIBPULSE=yes
342. !CONFIG\_LIBRTMP=yes
343. !CONFIG\_LIBSCHROEDINGER=yes
344. !CONFIG\_LIBSPEEX=yes
345. !CONFIG\_LIBSTAGEFRIGHT\_H264=yes
346. !CONFIG\_LIBTHEORA=yes
347. !CONFIG\_LIBUTVIDEO=yes
348. !CONFIG\_LIBV4L2=yes
349. !CONFIG\_LIBVO\_AACENC=yes
350. !CONFIG\_LIBVO\_AMRWBENC=yes
351. !CONFIG\_LIBVORBIS=yes
352. !CONFIG\_LIBVPX=yes
353. !CONFIG\_LIBX264=yes
354. !CONFIG\_LIBXAVS=yes
355. !CONFIG\_LIBXVID=yes
356. CONFIG\_LPC=yes
357. CONFIG\_LSP=yes
358. CONFIG\_MDCT=yes
359. CONFIG\_MEMALIGN\_HACK=yes
360. !CONFIG\_MLIB=yes
361. CONFIG\_MPEGAUDIODSP=yes
362. CONFIG\_NETWORK=yes
363. !CONFIG\_NONFREE=yes
364. !CONFIG\_OPENAL=yes
365. !CONFIG\_OPENSSL=yes
366. !CONFIG\_PIC=yes
367. !CONFIG\_POSTPROC=yes
368. CONFIG\_RDFT=yes
369. CONFIG\_RTPDEC=yes
370. !CONFIG\_RUNTIME\_CPUDETECT=yes
371. CONFIG\_SAFE\_BITSTREAM\_READER=yes
372. !CONFIG\_SHARED=yes
373. CONFIG\_SINEWIN=yes
374. !CONFIG\_SMALL=yes
375. !CONFIG\_SRAM=yes
376. CONFIG\_STATIC=yes
377. CONFIG\_SWRESAMPLE=yes
378. CONFIG\_SWSCALE=yes
379. CONFIG\_SWSCALE\_ALPHA=yes
380. !CONFIG\_THUMB=yes
381. !CONFIG\_VAAPI=yes
382. !CONFIG\_VDA=yes
383. !CONFIG\_VDPAU=yes
384. !CONFIG\_VERSION3=yes
385. !CONFIG\_X11GRAB=yes
386. CONFIG\_ZLIB=yes
387. CONFIG\_AVUTIL=yes
388. !CONFIG\_GPLV3=yes
389. !CONFIG\_LGPLV3=yes
390. CONFIG\_AAC\_ADTSTOASC\_BSF=yes
391. CONFIG\_CHOMP\_BSF=yes
392. CONFIG\_DUMP\_EXTRADATA\_BSF=yes
393. CONFIG\_H264\_MP4TOANNEXB\_BSF=yes
394. CONFIG\_IMX\_DUMP\_HEADER\_BSF=yes
395. CONFIG\_MJPEG2JPEG\_BSF=yes
396. CONFIG\_MJPEGA\_DUMP\_HEADER\_BSF=yes
397. CONFIG\_MP3\_HEADER\_COMPRESS\_BSF=yes
398. CONFIG\_MP3\_HEADER\_DECOMPRESS\_BSF=yes
399. CONFIG\_MOV2TEXTSUB\_BSF=yes
400. CONFIG\_NOISE\_BSF=yes
401. CONFIG\_REMOVE\_EXTRADATA\_BSF=yes
402. CONFIG\_TEXT2MOVSUB\_BSF=yes
403. CONFIG\_AASC\_DECODER=yes
404. CONFIG\_AMV\_DECODER=yes
405. CONFIG\_ANM\_DECODER=yes
406. CONFIG\_ANSI\_DECODER=yes
407. CONFIG\_ASV1\_DECODER=yes
408. CONFIG\_ASV2\_DECODER=yes
409. CONFIG\_AURA\_DECODER=yes
410. CONFIG\_AURA2\_DECODER=yes
411. CONFIG\_AVRP\_DECODER=yes
412. CONFIG\_AVS\_DECODER=yes
413. CONFIG\_BETHSOFTVID\_DECODER=yes
414. CONFIG\_BFI\_DECODER=yes
415. CONFIG\_BINK\_DECODER=yes
416. CONFIG\_BMP\_DECODER=yes
417. CONFIG\_BMV\_VIDEO\_DECODER=yes
418. CONFIG\_C93\_DECODER=yes
419. CONFIG\_CAVS\_DECODER=yes
420. CONFIG\_CDGRAPHICS\_DECODER=yes
421. CONFIG\_CINEPAK\_DECODER=yes
422. CONFIG\_CLJR\_DECODER=yes
423. CONFIG\_CSCD\_DECODER=yes
424. CONFIG\_CYUV\_DECODER=yes
425. CONFIG\_DFA\_DECODER=yes
426. CONFIG\_DIRAC\_DECODER=yes
427. CONFIG\_DNXHD\_DECODER=yes
428. CONFIG\_DPX\_DECODER=yes
429. CONFIG\_DSICINVIDEO\_DECODER=yes
430. CONFIG\_DVVIDEO\_DECODER=yes
431. CONFIG\_DXA\_DECODER=yes
432. CONFIG\_DXTORY\_DECODER=yes
433. CONFIG\_EACMV\_DECODER=yes
434. CONFIG\_EAMAD\_DECODER=yes
435. CONFIG\_EATGQ\_DECODER=yes
436. CONFIG\_EATGV\_DECODER=yes
437. CONFIG\_EATQI\_DECODER=yes
438. CONFIG\_EIGHTBPS\_DECODER=yes
439. CONFIG\_EIGHTSVX\_EXP\_DECODER=yes
440. CONFIG\_EIGHTSVX\_FIB\_DECODER=yes
441. CONFIG\_ESCAPE124\_DECODER=yes
442. CONFIG\_ESCAPE130\_DECODER=yes
443. CONFIG\_FFV1\_DECODER=yes
444. CONFIG\_FFVHUFF\_DECODER=yes
445. CONFIG\_FLASHSV\_DECODER=yes
446. CONFIG\_FLASHSV2\_DECODER=yes
447. CONFIG\_FLIC\_DECODER=yes
448. CONFIG\_FLV\_DECODER=yes
449. CONFIG\_FOURXM\_DECODER=yes
450. CONFIG\_FRAPS\_DECODER=yes
451. CONFIG\_FRWU\_DECODER=yes
452. CONFIG\_GIF\_DECODER=yes
453. CONFIG\_H261\_DECODER=yes
454. CONFIG\_H263\_DECODER=yes
455. CONFIG\_H263I\_DECODER=yes
456. CONFIG\_H264\_DECODER=yes
457. #此处省略若干条…
458. CONFIG\_RTMP\_PROTOCOL=yes
459. CONFIG\_RTMPT\_PROTOCOL=yes
460. CONFIG\_RTMPE\_PROTOCOL=yes
461. CONFIG\_RTMPTE\_PROTOCOL=yes
462. CONFIG\_RTMPS\_PROTOCOL=yes
463. CONFIG\_RTP\_PROTOCOL=yes
464. CONFIG\_TCP\_PROTOCOL=yes
465. !CONFIG\_TLS\_PROTOCOL=yes
466. CONFIG\_UDP\_PROTOCOL=yes
467. #Test
468. ACODEC\_TESTS=ac3\_fixed adpcm\_adx adpcm\_ima\_qt adpcm\_ima\_wav adpcm\_ms adpcm\_swf adpcm\_yam alac aref flac g722 g723\_1 g726 mp2 pcm\_alaw pcm\_f32be pcm\_f32le pcm\_f64be pcm\_f64le pcm\_mulaw pcm\_s16be pcm\_s16le pcm\_s24be pcm\_s24daud pcm\_s24le pcm\_s32be pcm\_s32le pcm\_s8 pcm\_u8 wmav1 wmav2
469. VCODEC\_TESTS=amv asv1 asv2 cljr dnxhd\_1080i dnxhd\_720p dnxhd\_720p\_10bit dnxhd\_720p\_rd dv dv50 dv\_411 error ffv1 flashsv flashsv2 flv h261 h263 h263p huffyuv jpeg2000 jpegls ljpeg mjpeg mpeg mpeg1b mpeg2 mpeg2\_422 mpeg2\_idct\_int mpeg2\_ilace mpeg2\_ivlc\_qprd mpeg2thread mpeg2thread\_ilace mpeg4 mpeg4\_adap mpeg4\_qpel mpeg4\_qprd mpeg4adv mpeg4nr mpeg4thread mpng msmpeg4 msmpeg4v2 msvideo1 prores qtrle qtrlegray rc rgb roq rv10 rv20 snow snowll svq1 v210 vref wmv1 wmv2 yuv zlib zmbv
470. LAVF\_TESTS=aiff alaw asf au avi bmp caf dpx dv\_fmt ffm flv\_fmt gif gxf jpg mkv mmf mov mpg mulaw mxf mxf\_d10 nut ogg pbmpipe pcx pgm pgmpipe pixfmt png ppm ppmpipe rm rso sgi sox swf tga tiff ts voc voc\_s16 wav wtv yuv4mpeg
471. LAVFI\_TESTS=crop crop\_scale crop\_scale\_vflip crop\_vflip null pixdesc pixfmts\_copy pixfmts\_crop pixfmts\_hflip pixfmts\_null pixfmts\_pad pixfmts\_scale pixfmts\_vflip scale200 scale500 vflip vflip\_crop vflip\_vflip
472. SEEK\_TESTS=seek\_ac3\_rm seek\_adpcm\_ima\_wav seek\_adpcm\_ms\_wav seek\_adpcm\_qt\_aiff seek\_adpcm\_swf\_flv seek\_adpcm\_yam\_wav seek\_alac\_m4a seek\_asv1\_avi seek\_asv2\_avi seek\_dnxhd\_1080i\_mov seek\_dnxhd\_720p\_dnxhd seek\_dnxhd\_720p\_rd\_dnxhd seek\_dv411\_dv seek\_dv50\_dv seek\_dv\_dv seek\_error\_mpeg4\_adv\_avi seek\_ffv1\_avi seek\_flac\_flac seek\_flashsv\_flv seek\_flv\_flv seek\_g726\_wav seek\_h261\_avi seek\_h263\_avi seek\_h263p\_avi seek\_huffyuv\_avi seek\_image\_bmp seek\_image\_jpg seek\_image\_pcx seek\_image\_pgm seek\_image\_ppm seek\_image\_sgi seek\_image\_tga seek\_image\_tiff seek\_jpegls\_avi seek\_lavf\_aif seek\_lavf\_al seek\_lavf\_asf seek\_lavf\_au seek\_lavf\_avi seek\_lavf\_dv seek\_lavf\_ffm seek\_lavf\_flv seek\_lavf\_gif seek\_lavf\_gxf seek\_lavf\_mkv seek\_lavf\_mmf seek\_lavf\_mov seek\_lavf\_mpg seek\_lavf\_mxf seek\_lavf\_mxf\_d10 seek\_lavf\_nut seek\_lavf\_ogg seek\_lavf\_rm seek\_lavf\_swf seek\_lavf\_ts seek\_lavf\_ul seek\_lavf\_voc seek\_lavf\_wav seek\_lavf\_wtv seek\_lavf\_y4m seek\_ljpeg\_avi seek\_mjpeg\_avi seek\_mp2\_mp2 seek\_mpeg1\_mpg seek\_mpeg1b\_mpg seek\_mpeg2\_422\_mpg seek\_mpeg2\_idct\_int\_mpg seek\_mpeg2i\_mpg seek\_mpeg2ivlc\_qprd\_mpg seek\_mpeg2reuse\_mpg seek\_mpeg2thread\_mpg seek\_mpeg2threadivlc\_mpg seek\_mpeg4\_adap\_avi seek\_mpeg4\_adv\_avi seek\_mpeg4\_nr\_avi seek\_mpeg4\_qprd\_avi seek\_mpeg4\_rc\_avi seek\_mpeg4\_thread\_avi seek\_msmpeg4\_avi seek\_msmpeg4v2\_avi seek\_odivx\_mp4 seek\_pbmpipe\_pbm seek\_pcm\_alaw\_wav seek\_pcm\_f32be\_au seek\_pcm\_f32le\_wav seek\_pcm\_f64be\_au seek\_pcm\_f64le\_wav seek\_pcm\_mulaw\_wav seek\_pcm\_s16be\_mov seek\_pcm\_s16le\_wav seek\_pcm\_s24be\_mov seek\_pcm\_s24daud\_302 seek\_pcm\_s24le\_wav seek\_pcm\_s32be\_mov seek\_pcm\_s32le\_wav seek\_pcm\_s8\_mov seek\_pcm\_u8\_wav seek\_pgmpipe\_pgm seek\_ppmpipe\_ppm seek\_rgb\_avi seek\_roqav\_roq seek\_rv10\_rm seek\_rv20\_rm seek\_snow53\_avi seek\_snow\_avi seek\_svq1\_mov seek\_wmav1\_asf seek\_wmav2\_asf seek\_wmv1\_avi seek\_wmv2\_avi seek\_yuv\_avi
473. endif # FFMPEG\_CONFIG\_MAK

config.mak代码大致可以分为以下几类信息：

（1）各种路径（prefix等）  
（2）工具集（arch、cc、ld、yasm等）  
（3）参数集（cppflag、cflag、ldflag等）  
（4）前缀后缀（.a、.dll、.exe等）  
（5）类库版本（libavXXX\_version信息）  
（6）组件配置。这一部分信息使用{组件名}=yes的方式进行书写。对于不支持的组件，则在该组件所在行的前面标记上“！”号（感叹号似乎在Makefile语法中并没有什么特殊的用意，此处可能仅仅是作为一种标记？）。这一部分可以分为3类信息：

a) ARCH\_信息

b) HAVE\_信息

c) CONFIG\_信息。这一部分内容最多，将近有1000行。

（7）Test信息（测试组件的结果？还没研究）

## libavXXXX/Makefile

libavXXXX/Makefile指的是FFmpeg类库（libavformat、libavcodec、libavutil等）所在的文件夹下的Makefile。例如libavformat文件夹下的Makefile代码如下所示。

**[python]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44556525) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44556525)

[在CODE上查看代码片](https://code.csdn.net/snippets/625700)

1. # FFmpeg Libavformat Makefile
2. #
3. # 注释：雷霄骅
4. # leixiaohua1020@126.com
5. # http://blog.csdn.net/leixiaohua1020
6. #
7. # FFmpeg中libavformat的Makefile。
8. # 注意该Makefile并没有定义类库的编译规则（这一部分统一在library.mak中完成）。
9. # 该Makefile中只是赋值了几个重要的字符串：
10. #       NAME，FFLIBS，HEADERS，OBJS，OBJS-yes
12. #重要：包含了configure信息，位于上一级目录
13. include $(SUBDIR)../config.mak
14. #名称
15. NAME = avformat
16. #用到的库？
17. FFLIBS = avcodec avutil
18. #SDK中的头文件
19. HEADERS = avformat.h avio.h version.h
21. #OBJS存储的是必须的目标文件
22. OBJS = allformats.o         \
23. cutils.o             \
24. id3v1.o              \
25. id3v2.o              \
26. metadata.o           \
27. options.o            \
28. os\_support.o         \
29. sdp.o                \
30. seek.o               \
31. utils.o              \
33. #OBJS-yes存储的是可选的目标文件
34. OBJS-$(CONFIG\_NETWORK)                   += network.o
36. # muxers/demuxers
37. OBJS-$(CONFIG\_A64\_MUXER)                 += a64.o
38. OBJS-$(CONFIG\_AAC\_DEMUXER)               += aacdec.o rawdec.o
39. OBJS-$(CONFIG\_AC3\_DEMUXER)               += ac3dec.o rawdec.o
40. OBJS-$(CONFIG\_AC3\_MUXER)                 += rawenc.o
41. OBJS-$(CONFIG\_ACT\_DEMUXER)               += act.o
42. OBJS-$(CONFIG\_ADF\_DEMUXER)               += bintext.o sauce.o
43. OBJS-$(CONFIG\_ADX\_DEMUXER)               += adxdec.o
44. OBJS-$(CONFIG\_ADX\_MUXER)                 += rawenc.o
45. OBJS-$(CONFIG\_ADTS\_MUXER)                += adtsenc.o
46. OBJS-$(CONFIG\_AEA\_DEMUXER)               += aea.o pcm.o
47. OBJS-$(CONFIG\_AIFF\_DEMUXER)              += aiffdec.o riff.o pcm.o isom.o
48. OBJS-$(CONFIG\_AIFF\_MUXER)                += aiffenc.o riff.o isom.o
49. OBJS-$(CONFIG\_AMR\_DEMUXER)               += amr.o
50. OBJS-$(CONFIG\_AMR\_MUXER)                 += amr.o
51. OBJS-$(CONFIG\_ANM\_DEMUXER)               += anm.o
52. OBJS-$(CONFIG\_APC\_DEMUXER)               += apc.o
53. OBJS-$(CONFIG\_APE\_DEMUXER)               += ape.o apetag.o
54. OBJS-$(CONFIG\_APPLEHTTP\_DEMUXER)         += applehttp.o
55. OBJS-$(CONFIG\_ASF\_DEMUXER)               += asfdec.o asf.o asfcrypt.o \
56. riff.o avlanguage.o
57. OBJS-$(CONFIG\_ASF\_MUXER)                 += asfenc.o asf.o riff.o
58. OBJS-$(CONFIG\_ASS\_DEMUXER)               += assdec.o
59. OBJS-$(CONFIG\_ASS\_MUXER)                 += assenc.o
60. OBJS-$(CONFIG\_AU\_DEMUXER)                += au.o pcm.o
61. OBJS-$(CONFIG\_AU\_MUXER)                  += au.o
62. OBJS-$(CONFIG\_AVI\_DEMUXER)               += avidec.o riff.o
63. OBJS-$(CONFIG\_AVI\_MUXER)                 += avienc.o riff.o
64. OBJS-$(CONFIG\_AVISYNTH)                  += avisynth.o
65. OBJS-$(CONFIG\_AVM2\_MUXER)                += swfenc.o
66. OBJS-$(CONFIG\_AVS\_DEMUXER)               += avs.o vocdec.o voc.o
67. OBJS-$(CONFIG\_BETHSOFTVID\_DEMUXER)       += bethsoftvid.o
68. OBJS-$(CONFIG\_BFI\_DEMUXER)               += bfi.o
69. OBJS-$(CONFIG\_BINK\_DEMUXER)              += bink.o
70. OBJS-$(CONFIG\_BINTEXT\_DEMUXER)           += bintext.o sauce.o
71. OBJS-$(CONFIG\_BIT\_DEMUXER)               += bit.o
72. OBJS-$(CONFIG\_BIT\_MUXER)                 += bit.o
73. OBJS-$(CONFIG\_BMV\_DEMUXER)               += bmv.o
74. OBJS-$(CONFIG\_C93\_DEMUXER)               += c93.o vocdec.o voc.o
75. OBJS-$(CONFIG\_CAF\_DEMUXER)               += cafdec.o caf.o mov.o mov\_chan.o \
76. riff.o isom.o
77. OBJS-$(CONFIG\_CAF\_MUXER)                 += cafenc.o caf.o riff.o isom.o
78. OBJS-$(CONFIG\_CAVSVIDEO\_DEMUXER)         += cavsvideodec.o rawdec.o
79. OBJS-$(CONFIG\_CAVSVIDEO\_MUXER)           += rawenc.o
80. OBJS-$(CONFIG\_CDG\_DEMUXER)               += cdg.o
81. OBJS-$(CONFIG\_CRC\_MUXER)                 += crcenc.o
82. OBJS-$(CONFIG\_DAUD\_DEMUXER)              += daud.o
83. OBJS-$(CONFIG\_DAUD\_MUXER)                += daud.o
84. OBJS-$(CONFIG\_DFA\_DEMUXER)               += dfa.o
85. OBJS-$(CONFIG\_DIRAC\_DEMUXER)             += diracdec.o rawdec.o
86. OBJS-$(CONFIG\_DIRAC\_MUXER)               += rawenc.o
87. OBJS-$(CONFIG\_DNXHD\_DEMUXER)             += dnxhddec.o rawdec.o
88. OBJS-$(CONFIG\_DNXHD\_MUXER)               += rawenc.o
89. OBJS-$(CONFIG\_DSICIN\_DEMUXER)            += dsicin.o
90. OBJS-$(CONFIG\_DTS\_DEMUXER)               += dtsdec.o rawdec.o
91. OBJS-$(CONFIG\_DTS\_MUXER)                 += rawenc.o
92. OBJS-$(CONFIG\_DV\_DEMUXER)                += dv.o
93. OBJS-$(CONFIG\_DV\_MUXER)                  += dvenc.o
94. OBJS-$(CONFIG\_DXA\_DEMUXER)               += dxa.o riff.o
95. OBJS-$(CONFIG\_EA\_CDATA\_DEMUXER)          += eacdata.o
96. OBJS-$(CONFIG\_EA\_DEMUXER)                += electronicarts.o
97. OBJS-$(CONFIG\_EAC3\_DEMUXER)              += ac3dec.o rawdec.o
98. OBJS-$(CONFIG\_EAC3\_MUXER)                += rawenc.o
99. OBJS-$(CONFIG\_FFM\_DEMUXER)               += ffmdec.o
100. OBJS-$(CONFIG\_FFM\_MUXER)                 += ffmenc.o
101. OBJS-$(CONFIG\_FFMETADATA\_DEMUXER)        += ffmetadec.o
102. OBJS-$(CONFIG\_FFMETADATA\_MUXER)          += ffmetaenc.o
103. OBJS-$(CONFIG\_FILMSTRIP\_DEMUXER)         += filmstripdec.o
104. OBJS-$(CONFIG\_FILMSTRIP\_MUXER)           += filmstripenc.o
105. OBJS-$(CONFIG\_FLAC\_DEMUXER)              += flacdec.o rawdec.o \
106. oggparsevorbis.o \
107. vorbiscomment.o
108. OBJS-$(CONFIG\_FLAC\_MUXER)                += flacenc.o flacenc\_header.o \
109. vorbiscomment.o
110. OBJS-$(CONFIG\_FLIC\_DEMUXER)              += flic.o
111. OBJS-$(CONFIG\_FLV\_DEMUXER)               += flvdec.o
112. OBJS-$(CONFIG\_FLV\_MUXER)                 += flvenc.o avc.o
113. OBJS-$(CONFIG\_FOURXM\_DEMUXER)            += 4xm.o
114. OBJS-$(CONFIG\_FRAMECRC\_MUXER)            += framecrcenc.o
115. OBJS-$(CONFIG\_FRAMEMD5\_MUXER)            += md5enc.o
116. OBJS-$(CONFIG\_GIF\_MUXER)                 += gif.o
117. OBJS-$(CONFIG\_GSM\_DEMUXER)               += gsmdec.o
118. OBJS-$(CONFIG\_GXF\_DEMUXER)               += gxf.o
119. OBJS-$(CONFIG\_GXF\_MUXER)                 += gxfenc.o audiointerleave.o
120. OBJS-$(CONFIG\_G722\_DEMUXER)              += rawdec.o
121. OBJS-$(CONFIG\_G722\_MUXER)                += rawenc.o
122. OBJS-$(CONFIG\_G723\_1\_DEMUXER)            += g723\_1.o
123. OBJS-$(CONFIG\_G723\_1\_MUXER)              += rawenc.o
124. OBJS-$(CONFIG\_G729\_DEMUXER)              += g729dec.o
125. OBJS-$(CONFIG\_H261\_DEMUXER)              += h261dec.o rawdec.o
126. OBJS-$(CONFIG\_H261\_MUXER)                += rawenc.o
127. OBJS-$(CONFIG\_H263\_DEMUXER)              += h263dec.o rawdec.o
128. OBJS-$(CONFIG\_H263\_MUXER)                += rawenc.o
129. OBJS-$(CONFIG\_H264\_DEMUXER)              += h264dec.o rawdec.o
130. OBJS-$(CONFIG\_H264\_MUXER)                += rawenc.o
131. OBJS-$(CONFIG\_ICO\_DEMUXER)               += icodec.o
132. OBJS-$(CONFIG\_IDCIN\_DEMUXER)             += idcin.o
133. OBJS-$(CONFIG\_IDF\_DEMUXER)               += bintext.o
134. OBJS-$(CONFIG\_IFF\_DEMUXER)               += iff.o
135. OBJS-$(CONFIG\_IMAGE2\_DEMUXER)            += img2.o
136. OBJS-$(CONFIG\_IMAGE2\_MUXER)              += img2.o
137. OBJS-$(CONFIG\_IMAGE2PIPE\_DEMUXER)        += img2.o
138. OBJS-$(CONFIG\_IMAGE2PIPE\_MUXER)          += img2.o
139. OBJS-$(CONFIG\_INGENIENT\_DEMUXER)         += ingenientdec.o rawdec.o
140. OBJS-$(CONFIG\_IPMOVIE\_DEMUXER)           += ipmovie.o
141. OBJS-$(CONFIG\_ISS\_DEMUXER)               += iss.o
142. OBJS-$(CONFIG\_IV8\_DEMUXER)               += iv8.o
143. OBJS-$(CONFIG\_IVF\_DEMUXER)               += ivfdec.o riff.o
144. OBJS-$(CONFIG\_IVF\_MUXER)                 += ivfenc.o
145. OBJS-$(CONFIG\_JV\_DEMUXER)                += jvdec.o
146. OBJS-$(CONFIG\_LATM\_DEMUXER)              += rawdec.o
147. OBJS-$(CONFIG\_LATM\_MUXER)                += latmenc.o
148. OBJS-$(CONFIG\_LMLM4\_DEMUXER)             += lmlm4.o
149. OBJS-$(CONFIG\_LOAS\_DEMUXER)              += loasdec.o
150. OBJS-$(CONFIG\_LXF\_DEMUXER)               += lxfdec.o
151. OBJS-$(CONFIG\_M4V\_DEMUXER)               += m4vdec.o rawdec.o
152. OBJS-$(CONFIG\_M4V\_MUXER)                 += rawenc.o
153. OBJS-$(CONFIG\_MATROSKA\_DEMUXER)          += matroskadec.o matroska.o \
154. riff.o isom.o rmdec.o rm.o
155. OBJS-$(CONFIG\_MATROSKA\_MUXER)            += matroskaenc.o matroska.o \
156. riff.o isom.o avc.o \
157. flacenc\_header.o avlanguage.o
158. OBJS-$(CONFIG\_MD5\_MUXER)                 += md5enc.o
159. OBJS-$(CONFIG\_MICRODVD\_DEMUXER)          += microdvddec.o
160. OBJS-$(CONFIG\_MICRODVD\_MUXER)            += microdvdenc.o rawenc.o
161. OBJS-$(CONFIG\_MJPEG\_DEMUXER)             += rawdec.o
162. OBJS-$(CONFIG\_MJPEG\_MUXER)               += rawenc.o
163. OBJS-$(CONFIG\_MLP\_DEMUXER)               += rawdec.o
164. OBJS-$(CONFIG\_MLP\_MUXER)                 += rawenc.o
165. OBJS-$(CONFIG\_MM\_DEMUXER)                += mm.o
166. OBJS-$(CONFIG\_MMF\_DEMUXER)               += mmf.o pcm.o
167. OBJS-$(CONFIG\_MMF\_MUXER)                 += mmf.o riff.o
168. OBJS-$(CONFIG\_MOV\_DEMUXER)               += mov.o riff.o isom.o mov\_chan.o
169. OBJS-$(CONFIG\_MOV\_MUXER)                 += movenc.o riff.o isom.o avc.o \
170. movenchint.o rtpenc\_chain.o \
171. mov\_chan.o
172. OBJS-$(CONFIG\_MP2\_MUXER)                 += mp3enc.o rawenc.o
173. OBJS-$(CONFIG\_MP3\_DEMUXER)               += mp3dec.o
174. OBJS-$(CONFIG\_MP3\_MUXER)                 += mp3enc.o rawenc.o id3v2enc.o
175. OBJS-$(CONFIG\_MPC\_DEMUXER)               += mpc.o apetag.o
176. OBJS-$(CONFIG\_MPC8\_DEMUXER)              += mpc8.o
177. OBJS-$(CONFIG\_MPEG1SYSTEM\_MUXER)         += mpegenc.o
178. OBJS-$(CONFIG\_MPEG1VCD\_MUXER)            += mpegenc.o
179. OBJS-$(CONFIG\_MPEG2DVD\_MUXER)            += mpegenc.o
180. OBJS-$(CONFIG\_MPEG2VOB\_MUXER)            += mpegenc.o
181. OBJS-$(CONFIG\_MPEG2SVCD\_MUXER)           += mpegenc.o
182. OBJS-$(CONFIG\_MPEG1VIDEO\_MUXER)          += rawenc.o
183. OBJS-$(CONFIG\_MPEG2VIDEO\_MUXER)          += rawenc.o
184. OBJS-$(CONFIG\_MPEGPS\_DEMUXER)            += mpeg.o
185. OBJS-$(CONFIG\_MPEGTS\_DEMUXER)            += mpegts.o isom.o
186. OBJS-$(CONFIG\_MPEGTS\_MUXER)              += mpegtsenc.o adtsenc.o
187. OBJS-$(CONFIG\_MPEGVIDEO\_DEMUXER)         += mpegvideodec.o rawdec.o
188. OBJS-$(CONFIG\_MPJPEG\_MUXER)              += mpjpeg.o
189. OBJS-$(CONFIG\_MSNWC\_TCP\_DEMUXER)         += msnwc\_tcp.o
190. OBJS-$(CONFIG\_MTV\_DEMUXER)               += mtv.o
191. OBJS-$(CONFIG\_MVI\_DEMUXER)               += mvi.o
192. OBJS-$(CONFIG\_MXF\_DEMUXER)               += mxfdec.o mxf.o
193. OBJS-$(CONFIG\_MXF\_MUXER)                 += mxfenc.o mxf.o audiointerleave.o
194. OBJS-$(CONFIG\_MXG\_DEMUXER)               += mxg.o
195. OBJS-$(CONFIG\_NC\_DEMUXER)                += ncdec.o
196. OBJS-$(CONFIG\_NSV\_DEMUXER)               += nsvdec.o
197. OBJS-$(CONFIG\_NULL\_MUXER)                += nullenc.o
198. OBJS-$(CONFIG\_NUT\_DEMUXER)               += nutdec.o nut.o riff.o
199. OBJS-$(CONFIG\_NUT\_MUXER)                 += nutenc.o nut.o riff.o
200. OBJS-$(CONFIG\_NUV\_DEMUXER)               += nuv.o riff.o
201. OBJS-$(CONFIG\_OGG\_DEMUXER)               += oggdec.o         \
202. oggparsecelt.o   \
203. oggparsedirac.o  \
204. oggparseflac.o   \
205. oggparseogm.o    \
206. oggparseskeleton.o \
207. oggparsespeex.o  \
208. oggparsetheora.o \
209. oggparsevorbis.o \
210. riff.o \
211. vorbiscomment.o
212. OBJS-$(CONFIG\_OGG\_MUXER)                 += oggenc.o \
213. vorbiscomment.o
214. OBJS-$(CONFIG\_OMA\_DEMUXER)               += omadec.o pcm.o oma.o
215. OBJS-$(CONFIG\_OMA\_MUXER)                 += omaenc.o rawenc.o oma.o id3v2enc.o
216. OBJS-$(CONFIG\_PCM\_ALAW\_DEMUXER)          += pcmdec.o pcm.o rawdec.o
217. OBJS-$(CONFIG\_PCM\_ALAW\_MUXER)            += pcmenc.o rawenc.o
218. OBJS-$(CONFIG\_PCM\_F32BE\_DEMUXER)         += pcmdec.o pcm.o rawdec.o
219. OBJS-$(CONFIG\_PCM\_F32BE\_MUXER)           += pcmenc.o rawenc.o
220. OBJS-$(CONFIG\_PCM\_F32LE\_DEMUXER)         += pcmdec.o pcm.o rawdec.o
221. OBJS-$(CONFIG\_PCM\_F32LE\_MUXER)           += pcmenc.o rawenc.o
222. OBJS-$(CONFIG\_PCM\_F64BE\_DEMUXER)         += pcmdec.o pcm.o rawdec.o
223. OBJS-$(CONFIG\_PCM\_F64BE\_MUXER)           += pcmenc.o rawenc.o
224. OBJS-$(CONFIG\_PCM\_F64LE\_DEMUXER)         += pcmdec.o pcm.o rawdec.o
225. OBJS-$(CONFIG\_PCM\_F64LE\_MUXER)           += pcmenc.o rawenc.o
226. OBJS-$(CONFIG\_PCM\_MULAW\_DEMUXER)         += pcmdec.o pcm.o rawdec.o
227. OBJS-$(CONFIG\_PCM\_MULAW\_MUXER)           += pcmenc.o rawenc.o
228. OBJS-$(CONFIG\_PCM\_S16BE\_DEMUXER)         += pcmdec.o pcm.o rawdec.o
229. OBJS-$(CONFIG\_PCM\_S16BE\_MUXER)           += pcmenc.o rawenc.o
230. OBJS-$(CONFIG\_PCM\_S16LE\_DEMUXER)         += pcmdec.o pcm.o rawdec.o
231. OBJS-$(CONFIG\_PCM\_S16LE\_MUXER)           += pcmenc.o rawenc.o
232. OBJS-$(CONFIG\_PCM\_S24BE\_DEMUXER)         += pcmdec.o pcm.o rawdec.o
233. OBJS-$(CONFIG\_PCM\_S24BE\_MUXER)           += pcmenc.o rawenc.o
234. OBJS-$(CONFIG\_PCM\_S24LE\_DEMUXER)         += pcmdec.o pcm.o rawdec.o
235. OBJS-$(CONFIG\_PCM\_S24LE\_MUXER)           += pcmenc.o rawenc.o
236. OBJS-$(CONFIG\_PCM\_S32BE\_DEMUXER)         += pcmdec.o pcm.o rawdec.o
237. OBJS-$(CONFIG\_PCM\_S32BE\_MUXER)           += pcmenc.o rawenc.o
238. OBJS-$(CONFIG\_PCM\_S32LE\_DEMUXER)         += pcmdec.o pcm.o rawdec.o
239. OBJS-$(CONFIG\_PCM\_S32LE\_MUXER)           += pcmenc.o rawenc.o
240. OBJS-$(CONFIG\_PCM\_S8\_DEMUXER)            += pcmdec.o pcm.o rawdec.o
241. OBJS-$(CONFIG\_PCM\_S8\_MUXER)              += pcmenc.o rawenc.o
242. OBJS-$(CONFIG\_PCM\_U16BE\_DEMUXER)         += pcmdec.o pcm.o rawdec.o
243. OBJS-$(CONFIG\_PCM\_U16BE\_MUXER)           += pcmenc.o rawenc.o
244. OBJS-$(CONFIG\_PCM\_U16LE\_DEMUXER)         += pcmdec.o pcm.o rawdec.o
245. OBJS-$(CONFIG\_PCM\_U16LE\_MUXER)           += pcmenc.o rawenc.o
246. OBJS-$(CONFIG\_PCM\_U24BE\_DEMUXER)         += pcmdec.o pcm.o rawdec.o
247. OBJS-$(CONFIG\_PCM\_U24BE\_MUXER)           += pcmenc.o rawenc.o
248. OBJS-$(CONFIG\_PCM\_U24LE\_DEMUXER)         += pcmdec.o pcm.o rawdec.o
249. OBJS-$(CONFIG\_PCM\_U24LE\_MUXER)           += pcmenc.o rawenc.o
250. OBJS-$(CONFIG\_PCM\_U32BE\_DEMUXER)         += pcmdec.o pcm.o rawdec.o
251. OBJS-$(CONFIG\_PCM\_U32BE\_MUXER)           += pcmenc.o rawenc.o
252. OBJS-$(CONFIG\_PCM\_U32LE\_DEMUXER)         += pcmdec.o pcm.o rawdec.o
253. OBJS-$(CONFIG\_PCM\_U32LE\_MUXER)           += pcmenc.o rawenc.o
254. OBJS-$(CONFIG\_PCM\_U8\_DEMUXER)            += pcmdec.o pcm.o rawdec.o
255. OBJS-$(CONFIG\_PCM\_U8\_MUXER)              += pcmenc.o rawenc.o
256. OBJS-$(CONFIG\_PMP\_DEMUXER)               += pmpdec.o
257. OBJS-$(CONFIG\_PVA\_DEMUXER)               += pva.o
258. OBJS-$(CONFIG\_QCP\_DEMUXER)               += qcp.o
259. OBJS-$(CONFIG\_R3D\_DEMUXER)               += r3d.o
260. OBJS-$(CONFIG\_RAWVIDEO\_DEMUXER)          += rawvideodec.o rawdec.o
261. OBJS-$(CONFIG\_RAWVIDEO\_MUXER)            += rawenc.o
262. OBJS-$(CONFIG\_RL2\_DEMUXER)               += rl2.o
263. OBJS-$(CONFIG\_RM\_DEMUXER)                += rmdec.o rm.o
264. OBJS-$(CONFIG\_RM\_MUXER)                  += rmenc.o rm.o
265. OBJS-$(CONFIG\_ROQ\_DEMUXER)               += idroqdec.o
266. OBJS-$(CONFIG\_ROQ\_MUXER)                 += idroqenc.o rawenc.o
267. OBJS-$(CONFIG\_RSO\_DEMUXER)               += rsodec.o rso.o pcm.o
268. OBJS-$(CONFIG\_RSO\_MUXER)                 += rsoenc.o rso.o
269. OBJS-$(CONFIG\_RPL\_DEMUXER)               += rpl.o
270. OBJS-$(CONFIG\_RTP\_MUXER)                 += rtp.o         \
271. rtpenc\_aac.o     \
272. rtpenc\_latm.o    \
273. rtpenc\_amr.o     \
274. rtpenc\_h263.o    \
275. rtpenc\_mpv.o     \
276. rtpenc.o      \
277. rtpenc\_h264.o \
278. rtpenc\_vp8.o  \
279. rtpenc\_xiph.o \
280. avc.o
281. OBJS-$(CONFIG\_RTPDEC)                    += rdt.o         \
282. rtp.o         \
283. rtpdec.o      \
284. rtpdec\_amr.o  \
285. rtpdec\_asf.o  \
286. rtpdec\_g726.o \
287. rtpdec\_h263.o \
288. rtpdec\_h264.o \
289. rtpdec\_latm.o \
290. rtpdec\_mpeg4.o \
291. rtpdec\_qcelp.o \
292. rtpdec\_qdm2.o \
293. rtpdec\_qt.o   \
294. rtpdec\_svq3.o \
295. rtpdec\_vp8.o  \
296. rtpdec\_xiph.o
297. OBJS-$(CONFIG\_RTSP\_DEMUXER)              += rtsp.o rtspdec.o httpauth.o
298. OBJS-$(CONFIG\_RTSP\_MUXER)                += rtsp.o rtspenc.o httpauth.o \
299. rtpenc\_chain.o
300. OBJS-$(CONFIG\_SAP\_DEMUXER)               += sapdec.o
301. OBJS-$(CONFIG\_SAP\_MUXER)                 += sapenc.o rtpenc\_chain.o
302. OBJS-$(CONFIG\_SBG\_DEMUXER)               += sbgdec.o
303. OBJS-$(CONFIG\_SDP\_DEMUXER)               += rtsp.o
304. OBJS-$(CONFIG\_SEGAFILM\_DEMUXER)          += segafilm.o
305. OBJS-$(CONFIG\_SEGMENT\_MUXER)             += segment.o
306. OBJS-$(CONFIG\_SHORTEN\_DEMUXER)           += rawdec.o
307. OBJS-$(CONFIG\_SIFF\_DEMUXER)              += siff.o
308. OBJS-$(CONFIG\_SMACKER\_DEMUXER)           += smacker.o
309. OBJS-$(CONFIG\_SMJPEG\_DEMUXER)            += smjpegdec.o smjpeg.o
310. OBJS-$(CONFIG\_SMJPEG\_MUXER)              += smjpegenc.o smjpeg.o
311. OBJS-$(CONFIG\_SOL\_DEMUXER)               += sol.o pcm.o
312. OBJS-$(CONFIG\_SOX\_DEMUXER)               += soxdec.o pcm.o
313. OBJS-$(CONFIG\_SOX\_MUXER)                 += soxenc.o
314. OBJS-$(CONFIG\_SPDIF\_DEMUXER)             += spdif.o spdifdec.o
315. OBJS-$(CONFIG\_SPDIF\_MUXER)               += spdif.o spdifenc.o
316. OBJS-$(CONFIG\_SRT\_DEMUXER)               += srtdec.o
317. OBJS-$(CONFIG\_SRT\_MUXER)                 += rawenc.o
318. OBJS-$(CONFIG\_STR\_DEMUXER)               += psxstr.o
319. OBJS-$(CONFIG\_SWF\_DEMUXER)               += swfdec.o
320. OBJS-$(CONFIG\_SWF\_MUXER)                 += swfenc.o
321. OBJS-$(CONFIG\_THP\_DEMUXER)               += thp.o
322. OBJS-$(CONFIG\_TIERTEXSEQ\_DEMUXER)        += tiertexseq.o
323. OBJS-$(CONFIG\_MKVTIMESTAMP\_V2\_MUXER)     += mkvtimestamp\_v2.o
324. OBJS-$(CONFIG\_TMV\_DEMUXER)               += tmv.o
325. OBJS-$(CONFIG\_TRUEHD\_DEMUXER)            += rawdec.o
326. OBJS-$(CONFIG\_TRUEHD\_MUXER)              += rawenc.o
327. OBJS-$(CONFIG\_TTA\_DEMUXER)               += tta.o
328. OBJS-$(CONFIG\_TTY\_DEMUXER)               += tty.o sauce.o
329. OBJS-$(CONFIG\_TXD\_DEMUXER)               += txd.o
330. OBJS-$(CONFIG\_VC1\_DEMUXER)               += rawdec.o
331. OBJS-$(CONFIG\_VC1T\_DEMUXER)              += vc1test.o
332. OBJS-$(CONFIG\_VC1T\_MUXER)                += vc1testenc.o
333. OBJS-$(CONFIG\_VMD\_DEMUXER)               += sierravmd.o
334. OBJS-$(CONFIG\_VOC\_DEMUXER)               += vocdec.o voc.o
335. OBJS-$(CONFIG\_VOC\_MUXER)                 += vocenc.o voc.o
336. OBJS-$(CONFIG\_VQF\_DEMUXER)               += vqf.o
337. OBJS-$(CONFIG\_W64\_DEMUXER)               += wav.o riff.o pcm.o
338. OBJS-$(CONFIG\_WAV\_DEMUXER)               += wav.o riff.o pcm.o
339. OBJS-$(CONFIG\_WAV\_MUXER)                 += wav.o riff.o
340. OBJS-$(CONFIG\_WC3\_DEMUXER)               += wc3movie.o
341. OBJS-$(CONFIG\_WEBM\_MUXER)                += matroskaenc.o matroska.o \
342. riff.o isom.o avc.o \
343. flacenc\_header.o avlanguage.o
344. OBJS-$(CONFIG\_WSAUD\_DEMUXER)             += westwood\_aud.o
345. OBJS-$(CONFIG\_WSVQA\_DEMUXER)             += westwood\_vqa.o
346. OBJS-$(CONFIG\_WTV\_DEMUXER)               += wtvdec.o wtv.o asfdec.o asf.o asfcrypt.o \
347. avlanguage.o mpegts.o isom.o riff.o
348. OBJS-$(CONFIG\_WTV\_MUXER)                 += wtvenc.o wtv.o asf.o asfenc.o riff.o
349. OBJS-$(CONFIG\_WV\_DEMUXER)                += wv.o apetag.o
350. OBJS-$(CONFIG\_XA\_DEMUXER)                += xa.o
351. OBJS-$(CONFIG\_XBIN\_DEMUXER)              += bintext.o sauce.o
352. OBJS-$(CONFIG\_XMV\_DEMUXER)               += xmv.o riff.o
353. OBJS-$(CONFIG\_XWMA\_DEMUXER)              += xwma.o riff.o
354. OBJS-$(CONFIG\_YOP\_DEMUXER)               += yop.o
355. OBJS-$(CONFIG\_YUV4MPEGPIPE\_MUXER)        += yuv4mpeg.o
356. OBJS-$(CONFIG\_YUV4MPEGPIPE\_DEMUXER)      += yuv4mpeg.o
358. # external libraries
359. OBJS-$(CONFIG\_LIBMODPLUG\_DEMUXER)        += libmodplug.o
360. OBJS-$(CONFIG\_LIBNUT\_DEMUXER)            += libnut.o riff.o
361. OBJS-$(CONFIG\_LIBNUT\_MUXER)              += libnut.o riff.o
363. # protocols I/O
364. OBJS+= avio.o aviobuf.o
366. OBJS-$(CONFIG\_APPLEHTTP\_PROTOCOL)        += applehttpproto.o
367. OBJS-$(CONFIG\_CACHE\_PROTOCOL)            += cache.o
368. OBJS-$(CONFIG\_CONCAT\_PROTOCOL)           += concat.o
369. OBJS-$(CONFIG\_CRYPTO\_PROTOCOL)           += crypto.o
370. OBJS-$(CONFIG\_FILE\_PROTOCOL)             += file.o
371. OBJS-$(CONFIG\_GOPHER\_PROTOCOL)           += gopher.o
372. OBJS-$(CONFIG\_HTTP\_PROTOCOL)             += http.o httpauth.o
373. OBJS-$(CONFIG\_HTTPPROXY\_PROTOCOL)        += http.o httpauth.o
374. OBJS-$(CONFIG\_HTTPS\_PROTOCOL)            += http.o httpauth.o
375. OBJS-$(CONFIG\_MMSH\_PROTOCOL)             += mmsh.o mms.o asf.o
376. OBJS-$(CONFIG\_MMST\_PROTOCOL)             += mmst.o mms.o asf.o
377. OBJS-$(CONFIG\_MD5\_PROTOCOL)              += md5proto.o
378. OBJS-$(CONFIG\_PIPE\_PROTOCOL)             += file.o
380. # external or internal rtmp
381. RTMP-OBJS-$(CONFIG\_LIBRTMP)               = librtmp.o
382. RTMP-OBJS-$(!CONFIG\_LIBRTMP)              = rtmpproto.o rtmppkt.o
383. OBJS-$(CONFIG\_RTMP\_PROTOCOL)             += $(RTMP-OBJS-yes)
385. OBJS-$(CONFIG\_RTP\_PROTOCOL)              += rtpproto.o
386. OBJS-$(CONFIG\_TCP\_PROTOCOL)              += tcp.o
387. OBJS-$(CONFIG\_TLS\_PROTOCOL)              += tls.o
388. OBJS-$(CONFIG\_UDP\_PROTOCOL)              += udp.o
390. SKIPHEADERS-$(CONFIG\_NETWORK)            += network.h rtsp.h
391. TESTPROGS = seek
392. TOOLS     = aviocat ismindex pktdumper probetest

从代码可以看出，libavformat文件夹下的Makefile的规则十分简单，并不包含文件之间的依赖关系（依赖关系位于library.mak中），仅仅是设置了几个变量的值：

NAME：类库名称。注意不包含类库前面的“lib”以及类库的后缀。在这里是“avformat”。  
FFLIBS：该类库依赖的类库名称。在这里用到了“avcodec”和“avutil”。  
HEADERS：该类库导出的头文件。在这里是“avformat.h”，“avio.h”，“version.h”。  
OBJS：该类库依赖的目标文件（必须的）。在这里是“utils.o”等等。  
OBJS-yes：该类库依赖的目标文件（可选的）。在这里是“flvdec.o”、“flvenc.o”等等。

## library.mak

library.mak专门用于存储编译类库的规则，是和libavXXX/Makefile配合使用的。它的源代码如下所示。

**[python]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44556525) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44556525)

[在CODE上查看代码片](https://code.csdn.net/snippets/625700)

1. # FFmpeg library.mak
2. #
3. # 注释：雷霄骅
4. # leixiaohua1020@126.com
5. # http://blog.csdn.net/leixiaohua1020
6. #
7. # 编译类库(libavformat等)专用的Makefile，其中包含了编译类库的规则。
9. #【NAME位于每个类库的Makefile】，可以取avcodec，avformat等等
10. SRC\_DIR := $(SRC\_PATH)/lib$(NAME)
12. include $(SRC\_PATH)/common.mak
14. #这些信息都位于config.mak中
15. #例如：
16. # libavformat\_VERSION=53.31.100
17. # libavformat\_VERSION\_MAJOR=53
19. LIBVERSION := $(lib$(NAME)\_VERSION)
20. LIBMAJOR   := $(lib$(NAME)\_VERSION\_MAJOR)
21. INCINSTDIR := $(INCDIR)/lib$(NAME)
22. THIS\_LIB   := $(SUBDIR)$($(CONFIG\_SHARED:yes=S)LIBNAME)
24. all-$(CONFIG\_STATIC): $(SUBDIR)$(LIBNAME)
25. all-$(CONFIG\_SHARED): $(SUBDIR)$(SLIBNAME)

28. $(SUBDIR)%-test.o: $(SUBDIR)%-test.c
29. $(COMPILE\_C)
31. $(SUBDIR)%-test.o: $(SUBDIR)%.c
32. $(COMPILE\_C)
33. #汇编？
34. $(SUBDIR)x86/%.o: $(SUBDIR)x86/%.asm
35. $(YASMDEP) $(YASMFLAGS) -I $(<D)/ -M -o $@ $< > $(@:.o=.d)
36. $(YASM) $(YASMFLAGS) -I $(<D)/ -o $@ $<
38. $(OBJS) $(OBJS:.o=.s) $(SUBDIR)%.ho $(TESTOBJS): CPPFLAGS += -DHAVE\_AV\_CONFIG\_H
39. $(TESTOBJS): CPPFLAGS += -DTEST
41. #【OBJS来自于每个类库的Makefile】
42. #$@  表示规则中的目标文件集
43. #$^  所有的依赖目标的集合。
44. #生成静态库？
45. $(SUBDIR)$(LIBNAME): $(OBJS)
46. $(RM) $@
47. $(AR) rc $@ $^ $(EXTRAOBJS)
48. $(RANLIB) $@
49. #安转头文件，根目录的Makefile调用
50. install-headers: install-lib$(NAME)-headers install-lib$(NAME)-pkgconfig
51. #install-libs-yes被install-libs（位于根目录Makefile）调用
52. install-libs-$(CONFIG\_STATIC): install-lib$(NAME)-static
53. install-libs-$(CONFIG\_SHARED): install-lib$(NAME)-shared
55. define RULES
56. $(EXAMPLES) $(TESTPROGS) $(TOOLS): %$(EXESUF): %.o

(LD)$(LDFLAGS)−o

@

−l$(FULLNAME)$(FFEXTRALIBS)

(ELIBS)

2. $(SUBDIR)$(SLIBNAME): $(SUBDIR)$(SLIBNAME\_WITH\_MAJOR)
3. $(Q)cd ./$(SUBDIR) && $(LN\_S) $(SLIBNAME\_WITH\_MAJOR) $(SLIBNAME)
5. $(SUBDIR)$(SLIBNAME\_WITH\_MAJOR): $(OBJS) $(SUBDIR)lib$(NAME).ver
6. $(SLIB\_CREATE\_DEF\_CMD)

(LD)$(SHFLAGS)$(LDFLAGS)−o

@

(filter

^) $(FFEXTRALIBS) $(EXTRAOBJS)

1. $(SLIB\_EXTRA\_CMD)
3. #SLIBNAME\_WITH\_MAJOR包含了Major版本号。例如：libavformat-53.dll
4. ifdef SUBDIR
5. $(SUBDIR)$(SLIBNAME\_WITH\_MAJOR): $(DEP\_LIBS)
6. endif
7. #清空
8. clean::
9. $(RM) $(addprefix $(SUBDIR),\*-example$(EXESUF) \*-test$(EXESUF) $(CLEANFILES) $(CLEANSUFFIXES) $(LIBSUFFIXES)) \
10. $(foreach dir,$(DIRS),$(CLEANSUFFIXES:%=$(SUBDIR)$(dir)/%)) \
11. $(HOSTOBJS) $(HOSTPROGS)
13. distclean:: clean
14. $(RM) $(DISTCLEANSUFFIXES:%=$(SUBDIR)%) \
15. $(foreach dir,$(DIRS),$(DISTCLEANSUFFIXES:%=$(SUBDIR)$(dir)/%))
16. #安装库文件=====================
17. install-lib$(NAME)-shared: $(SUBDIR)$(SLIBNAME)
18. $(Q)mkdir -p "$(SHLIBDIR)"
19. $$(INSTALL) -m 755 $$< "$(SHLIBDIR)/$(SLIB\_INSTALL\_NAME)"
20. $$(STRIP) "$(SHLIBDIR)/$(SLIB\_INSTALL\_NAME)"
21. $(Q)$(foreach F,$(SLIB\_INSTALL\_LINKS),cd "$(SHLIBDIR)" && $(LN\_S) $(SLIB\_INSTALL\_NAME) $(F);)
22. $(**if** $(SLIB\_INSTALL\_EXTRA\_SHLIB),$$(INSTALL) -m 644 $(SLIB\_INSTALL\_EXTRA\_SHLIB:%=$(SUBDIR)%) "$(SHLIBDIR)")
23. $(**if** $(SLIB\_INSTALL\_EXTRA\_LIB),$(Q)mkdir -p "$(LIBDIR)")
24. $(**if** $(SLIB\_INSTALL\_EXTRA\_LIB),$$(INSTALL) -m 644 $(SLIB\_INSTALL\_EXTRA\_LIB:%=$(SUBDIR)%) "$(LIBDIR)")
26. install-lib$(NAME)-static: $(SUBDIR)$(LIBNAME)
27. $(Q)mkdir -p "$(LIBDIR)"
28. $$(INSTALL) -m 644 $$< "$(LIBDIR)"
29. $(LIB\_INSTALL\_EXTRA\_CMD)
30. #安装头文件=====================
31. #-m
32. #权限：644,755,777
33. #644 rw-r--r--
34. #755 rwxr-xr-x
35. #777 rwxrwxrwx
36. #从左至右，1-3位数字代表文件所有者的权限，4-6位数字代表同组用户的权限，7-9数字代表其他用户的权限。
37. #通过4、2、1的组合，得到以下几种权限：0（没有权限）；4（读取权限）；5（4+1 | 读取+执行）；6（4+2 | 读取+写入）；7（4+2+1 | 读取+写入+执行）
38. #addprefix()
39. #$(addprefix src/,foo bar)返回值是“src/foo src/bar”。
41. #【HEADERS来自于每个类库的Makefile】
42. #例如libavformat中HEADERS = avformat.h avio.h version.h
43. install-lib$(NAME)-headers: $(addprefix $(SUBDIR),$(HEADERS) $(BUILT\_HEADERS))
44. $(Q)mkdir -p "$(INCINSTDIR)"
45. $$(INSTALL) -m 644 $$^ "$(INCINSTDIR)"
47. install-lib$(NAME)-pkgconfig: $(SUBDIR)lib$(NAME).pc
48. $(Q)mkdir -p "$(LIBDIR)/pkgconfig"
49. $$(INSTALL) -m 644 $$^ "$(LIBDIR)/pkgconfig"
51. #卸载
52. uninstall-libs::
53. -$(RM) "$(SHLIBDIR)/$(SLIBNAME\_WITH\_MAJOR)" \
54. "$(SHLIBDIR)/$(SLIBNAME)"            \
55. "$(SHLIBDIR)/$(SLIBNAME\_WITH\_VERSION)"
56. -$(RM) $(SLIB\_INSTALL\_EXTRA\_SHLIB:%="$(SHLIBDIR)"%)
57. -$(RM) $(SLIB\_INSTALL\_EXTRA\_LIB:%="$(LIBDIR)"%)
58. -$(RM) "$(LIBDIR)/$(LIBNAME)"
60. uninstall-headers::
61. $(RM) $(addprefix "$(INCINSTDIR)/",$(HEADERS)) $(addprefix "$(INCINSTDIR)/",$(BUILT\_HEADERS))
62. $(RM) "$(LIBDIR)/pkgconfig/lib$(NAME).pc"
63. -rmdir "$(INCINSTDIR)"
64. endef
66. $(eval $(RULES))
68. $(EXAMPLES) $(TESTPROGS) $(TOOLS): $(THIS\_LIB) $(DEP\_LIBS)
69. $(TESTPROGS): $(SUBDIR)$(LIBNAME)
71. examples: $(EXAMPLES)
72. testprogs: $(TESTPROGS)

library.mak代码中首先包含了common.mak文件。这个文件定义了通用的一些编译规则。然后定义了类库的依赖关系。  
此外library.mak中也定义了install-headers，install-lib$(NAME)-shared，install-lib$(NAME)-static，install-lib$(NAME)-headers，clean等等一系列的伪目标（NAME取值avformat、avcodec等）。这些目标主要配合根目录的Makefile使用。

## common.mak

common.mak文件定义了通用的一些编译规则。代码如下所示。

**[python]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44556525) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44556525)

[在CODE上查看代码片](https://code.csdn.net/snippets/625700)

1. # FFmpeg common.mak
2. #
3. # 注释：雷霄骅
4. # leixiaohua1020@126.com
5. # http://blog.csdn.net/leixiaohua1020
6. #
7. # 通用的Makefile，其中包含了通用的编译规则。
8. #
9. # common bits used by all libraries
10. #
12. # first so "all" becomes default target
13. all: all-yes
15. ifndef SUBDIR
16. #在控制台打印信息
17. ifndef V
18. Q      = @
19. #输出
20. ECHO   = printf "$(1)\t%s\n" $(2)
21. BRIEF  = CC CXX AS YASM AR LD HOSTCC STRIP CP
22. SILENT = DEPCC YASMDEP RM RANLIB
23. MSG    = $@
24. M      = @$(call ECHO,$(TAG),$@);
25. $(foreach VAR,$(BRIEF), \
26. $(eval override $(VAR) = @

(callECHO,$(VAR),

(MSG)); $($(VAR))))

1. $(foreach VAR,$(SILENT),$(eval override $(VAR) = @$($(VAR))))
2. $(eval INSTALL = @$(call ECHO,INSTALL,$$(^:$(SRC\_DIR)/%=%)); $(INSTALL))
3. endif
4. #所有的lib
5. ALLFFLIBS = avcodec avdevice avfilter avformat avutil postproc swscale swresample
7. # NASM requires -I path terminated with /
8. #各种Flag
9. #SRC\_PATH=.
10. IFLAGS     := -I. -I$(SRC\_PATH)/
11. CPPFLAGS   := $(IFLAGS) $(CPPFLAGS)
12. CFLAGS     += $(ECFLAGS)
13. CCFLAGS     = $(CFLAGS)
14. CXXFLAGS   := $(CFLAGS) $(CXXFLAGS)
15. YASMFLAGS  += $(IFLAGS) -I$(SRC\_PATH)/libavutil/x86/ -Pconfig.asm
16. HOSTCFLAGS += $(IFLAGS)
17. #avcodec处理后成为-Llibavcodec
18. #config.mak文件中：
19. #LDFLAGS= -Wl,--as-needed -Wl,--warn-common -Wl,
20. #-rpath-link=libpostproc:libswresample:libswscale:libavfilter:libavdevice:libavformat:libavcodec:libavutil
21. LDFLAGS    := $(ALLFFLIBS:%=-Llib%) $(LDFLAGS)
23. #命令包
24. #具体编译命令
25. #
26. #$(1)可以取CC、CXX等
27. #例如取$(1)取CC
28. #config.mak文件中：
29. #SRC\_PATH=.
30. #CC=gcc
31. #
32. #CCFLAGS=$(CFLAGS)
33. #CFLAGS=   -std=c99 -fno-common -fomit-frame-pointer -I/include/SDL -D\_GNU\_SOURCE=1 -Dmain=SDL\_main
34. # -g -Wdeclaration-after-statement -Wall -Wno-parentheses -Wno-switch -Wno-format-zero-length
35. # -Wdisabled-optimization -Wpointer-arith -Wredundant-decls -Wno-pointer-sign -Wcast-qual -Wwrite-strings
36. # -Wtype-limits -Wundef -Wmissing-prototypes -Wno-pointer-to-int-cast -Wstrict-prototypes
37. # -O3 -fno-math-errno -fno-signed-zeros -fno-tree-vectorize -Werror=implicit-function-declaration -Werror=missing-prototypes
38. #
39. #CPPFLAGS= -D\_ISOC99\_SOURCE -D\_FILE\_OFFSET\_BITS=64 -D\_LARGEFILE\_SOURCE -U\_\_STRICT\_ANSI\_\_
40. #CC\_O=-o $@
41. #CC\_DEPFLAGS=-MMD -MF $(@:.o=.d) -MT $@
42. #举例：
43. #gcc -I. -Itest/ -c -o $@ $<
44. #再例如$(1)取CXX
45. #CXXFLAGS=  -D\_\_STDC\_CONSTANT\_MACROS
47. define COMPILE
48. $($(1)DEP)
49. $($(1)) $(CPPFLAGS) $($(1)FLAGS) $($(1)\_DEPFLAGS) -c $($(1)\_O) $<
50. endef
52. #编译命令
53. #$(call <expression>,<parm1>,<parm2>,<parm3>...)
54. #当make执行这个函数时，<expression>参数中的变量，如$(1)，$(2)，$(3)等，会被参数
55. #<parm1>，<parm2>，<parm3>依次取代。而<expression>的返回值就是call函数的返回值。
56. COMPILE\_C = $(call COMPILE,CC)
57. COMPILE\_CXX = $(call COMPILE,CXX)
58. COMPILE\_S = $(call COMPILE,AS)
60. #COMPILE\_C为：
61. #$(CC DEP)
62. #$($(CC) $(CPPFLAGS) $($(1)FLAGS) $($(1)\_DEPFLAGS) -c $($(1)\_O) $<
64. #依赖关系
65. #C语言
66. %.o: %.c
67. #编译
68. $(COMPILE\_C)
70. #C++
71. %.o: %.cpp
72. $(COMPILE\_CXX)
74. %.s: %.c
75. $(CC) $(CPPFLAGS) $(CFLAGS) -S -o $@ $<
77. %.o: %.S
78. $(COMPILE\_S)
80. %.ho: %.h
81. $(CC) $(CPPFLAGS) $(CFLAGS) -Wno-unused -c -o $@ -x c $<
83. %.ver: %.v
84. $(Q)sed 's/$$MAJOR/$($(basename $(@F))\_VERSION\_MAJOR)/' $^ > $@
86. %.c %.h: TAG = GEN
88. # Dummy rule to stop make trying to rebuild removed or renamed headers
89. %.h:
90. @:
92. # Disable suffix rules.  Most of the builtin rules are suffix rules,
93. # so this saves some time on slow systems.
94. .SUFFIXES:
96. # Do not delete intermediate files from chains of implicit rules
97. $(OBJS):
98. endif
100. OBJS-$(HAVE\_MMX) +=  $(MMX-OBJS-yes)
102. #源自Makefile
103. #OBJS：该类库必须的目标文件
104. #OBJS-yes：该类库可配置的目标文件
105. OBJS      += $(OBJS-yes)
106. #FFLIBS：必须的类库
107. #FFLIBS-yes：可选的类库
108. #FFLIBS = avcodec avutil ....
109. FFLIBS    := $(FFLIBS-yes) $(FFLIBS)
110. TESTPROGS += $(TESTPROGS-yes)
112. FFEXTRALIBS := $(FFLIBS:%=-l%$(BUILDSUF)) $(EXTRALIBS)
114. EXAMPLES  := $(EXAMPLES:%=$(SUBDIR)%-example$(EXESUF))
115. #排序？
116. OBJS      := $(sort $(OBJS:%=$(SUBDIR)%))
117. TESTOBJS  := $(TESTOBJS:%=$(SUBDIR)%) $(TESTPROGS:%=$(SUBDIR)%-test.o)
118. TESTPROGS := $(TESTPROGS:%=$(SUBDIR)%-test$(EXESUF))
119. HOSTOBJS  := $(HOSTPROGS:%=$(SUBDIR)%.o)
120. HOSTPROGS := $(HOSTPROGS:%=$(SUBDIR)%$(HOSTEXESUF))
121. TOOLS     += $(TOOLS-yes)
122. TOOLOBJS  := $(TOOLS:%=tools/%.o)
123. TOOLS     := $(TOOLS:%=tools/%$(EXESUF))
125. #DEP\_LIBS= libavcodec/libavcodec.a libavutil/libavutil.a ....
126. DEP\_LIBS := $(foreach NAME,$(FFLIBS),lib$(NAME)/$($(CONFIG\_SHARED:yes=S)LIBNAME))
128. ALLHEADERS := $(subst $(SRC\_DIR)/,$(SUBDIR),$(wildcard $(SRC\_DIR)/\*.h $(SRC\_DIR)/$(ARCH)/\*.h))
129. SKIPHEADERS += $(ARCH\_HEADERS:%=$(ARCH)/%) $(SKIPHEADERS-)
130. SKIPHEADERS := $(SKIPHEADERS:%=$(SUBDIR)%)
131. checkheaders: $(filter-out $(SKIPHEADERS:.h=.ho),$(ALLHEADERS:.h=.ho))
133. alltools: $(TOOLS)
135. $(HOSTOBJS): %.o: %.c
136. $(HOSTCC) $(HOSTCFLAGS) -c -o $@ $<
138. $(HOSTPROGS): %$(HOSTEXESUF): %.o
139. $(HOSTCC) $(HOSTLDFLAGS) -o $@ $< $(HOSTLIBS)
141. $(OBJS):     | $(sort $(dir $(OBJS)))
142. $(HOSTOBJS): | $(sort $(dir $(HOSTOBJS)))
143. $(TESTOBJS): | $(sort $(dir $(TESTOBJS)))
144. $(TOOLOBJS): | tools
146. OBJDIRS := $(OBJDIRS) $(dir $(OBJS) $(HOSTOBJS) $(TESTOBJS))
148. CLEANSUFFIXES     = \*.d \*.o \*~ \*.ho \*.map \*.ver \*.gcno \*.gcda
149. DISTCLEANSUFFIXES = \*.pc
150. LIBSUFFIXES       = \*.a \*.lib \*.so \*.so.\* \*.dylib \*.dll \*.**def** \*.dll.a \*.exp
152. #依赖文件.d（dependence）
153. -include $(wildcard $(OBJS:.o=.d) $(TESTOBJS:.o=.d))

从代码中可以看出，common.mak定义了一些通用的编译规则，例如编译时候的控制台输出格式，编译命令COMPILE\_C、COMPILE\_CXX、COMPILE\_S，以及.c、.o等文件之间的依赖关系等。

# [FFmpeg源代码简单分析：configure](http://blog.csdn.net/leixiaohua1020/article/details/44587465)

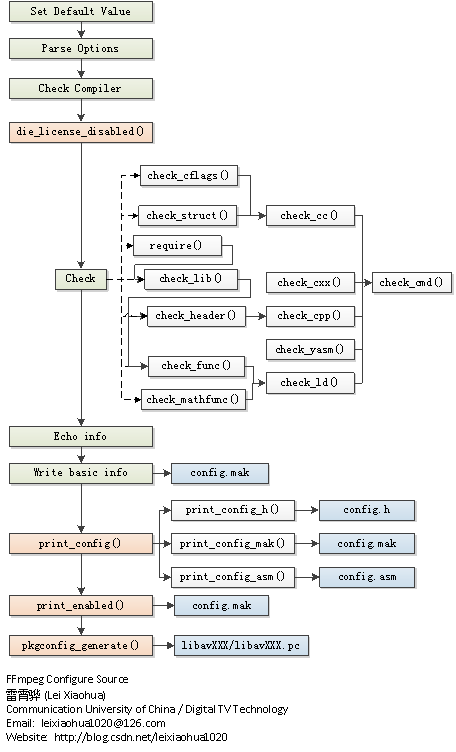
本文记录FFmpeg的Configure脚本的源代码。Configure一方面用于检测FFmpeg的编译环境，另一方面根据用户配置的选项生成config.mak，config.h文件（可能还有config.asm），提供给Makefile使用。由于FFmpeg的configure脚本很复杂（一个4000-5000行的Shell脚本），难以逐行细致的分析，因此本文简单梳理一下它的结构。

PS1：Configure的日志位于config.log文件中。查看该文件有助于分析Configure的过程。

PS2：使用“sh -x script\_name.sh”可以调试Shell脚本。

## Configure文件的整体流程

Configure文件的整体流程如下所示。



Configure的整体流程可以分成以下几步：

Set Default Value：设置各个变量默认值；

Parse Options：解析输入的选项；

Check Compiler：检查编译器；

die\_license\_disabled()：检查GPL等协议的设置情况；

Check：检查编译环境（数学函数，第三方类库等）；

Echo info：控制台上打印配置信息；

Write basic info：向config.mak中写入一些基本信息；

print\_config()：向config.h、config.mak、config.asm中写入所有配置信息；

print\_enabled()：向config.mak写入所有enabled的组件信息；

pkgconfig\_generate()：向libavXXX/libavXXX.pc中写入pkgconfig信息（XXX代表avcodec，avformat等）；

下文简单梳理一下这些步骤。

## Set Default Value

Set Default Value部分设置一些Configure的默认值。例如下面的代码。

**[python]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44587465) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44587465)

[在CODE上查看代码片](https://code.csdn.net/snippets/626422)

1. # 默认参数 default parameters
2. # 日志
3. logfile="config.log"
5. # 安装路径 installation paths
6. prefix\_default="/usr/local"
7. bindir\_default='${prefix}/bin'
8. datadir\_default='${prefix}/share/ffmpeg'
9. incdir\_default='${prefix}/include'
10. libdir\_default='${prefix}/lib'
11. mandir\_default='${prefix}/share/man'
12. shlibdir\_default="$libdir\_default"
13. postproc\_version\_default="current"
15. # 工具链 toolchain
16. ar\_default="ar"
17. cc\_default="gcc"
18. cxx\_default="g++"
19. cc\_version=\"unknown\"
20. host\_cc\_default="gcc"
21. install="install"
22. ln\_s="ln -sf"
23. nm\_default="nm"
24. objformat="elf"
25. pkg\_config\_default=pkg-config
26. ranlib="ranlib"
27. strip\_default="strip"
28. yasmexe\_default="yasm"
30. nm\_opts='-g'
31. nogas=":"
33. # 机器 machine
34. arch\_default=$(uname -m)
35. cpu="generic"
37. # 操作系统 OS
38. target\_os\_default=$(tolower $(uname -s))
39. host\_os=$target\_os\_default
41. # alternative libpostproc version
42. ALT\_PP\_VER\_MAJOR=51
43. ALT\_PP\_VER\_MINOR=2
44. ALT\_PP\_VER\_MICRO=101
45. ALT\_PP\_VER=$ALT\_PP\_VER\_MAJOR.$ALT\_PP\_VER\_MINOR.$ALT\_PP\_VER\_MICRO
47. # 选项 configurable options
48. # PROGRAM\_LIST内容是 ffplay ffprobe ffserver ffmpeg
49. enable $PROGRAM\_LIST
51. enable avcodec
52. enable avdevice
53. enable avfilter
54. enable avformat
55. enable avutil
56. enable postproc
57. enable stripping
58. enable swresample
59. enable swscale
61. enable asm
62. enable debug
63. enable doc
64. enable fastdiv
65. enable network
66. enable optimizations
67. enable safe\_bitstream\_reader
68. enable static
69. enable swscale\_alpha
71. # 编译选项 build settings
72. SHFLAGS='-shared -Wl,-soname,$$(@F)'
73. FFSERVERLDFLAGS=-Wl,-E
74. # 前缀后缀
75. LIBPREF="lib"
76. LIBSUF=".a"
77. FULLNAME='$(NAME)$(BUILDSUF)'
78. # 名称
79. LIBNAME='$(LIBPREF)$(FULLNAME)$(LIBSUF)'
80. # 动态库前缀后缀
81. SLIBPREF="lib"
82. SLIBSUF=".so"
83. # 名称
84. SLIBNAME='$(SLIBPREF)$(FULLNAME)$(SLIBSUF)'
85. SLIBNAME\_WITH\_VERSION='$(SLIBNAME).$(LIBVERSION)'
86. SLIBNAME\_WITH\_MAJOR='$(SLIBNAME).$(LIBMAJOR)'
87. LIB\_INSTALL\_EXTRA\_CMD='$$(RANLIB) "$(LIBDIR)/$(LIBNAME)"'
88. SLIB\_INSTALL\_NAME='$(SLIBNAME\_WITH\_VERSION)'
89. SLIB\_INSTALL\_LINKS='$(SLIBNAME\_WITH\_MAJOR) $(SLIBNAME)'
91. AS\_O='-o $@'
92. CC\_O='-o $@'
93. CXX\_O='-o $@'
95. host\_cflags='-D\_ISOC99\_SOURCE -O3 -g'
96. host\_libs='-lm'
98. target\_path='$(CURDIR)'

需要注意的是，“enable avcodec”，“enable avformat”，“enable avutil”等中的enable()本身是一个函数。enable()的定义如下。

**[python]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44587465) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44587465)

[在CODE上查看代码片](https://code.csdn.net/snippets/626422)

1. #把所有输入参数的值设置为"yes"
2. enable(){
3. set\_all yes $\*
4. }

可以看出enable()调用了set\_all()函数。并且将第1个参数设置为“yes”，并且将调用enable()时候的参数传递给set\_all()。set\_all()函数的定义如下所示。

**[python]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44587465) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44587465)

[在CODE上查看代码片](https://code.csdn.net/snippets/626422)

1. #第一个参数为值，后面的参数为变量
2. set\_all(){
3. value=$1
4. shift
5. **for** var **in** $\*; do
6. eval $var=$value
7. done
8. }

可以看出set\_all()将传入的参数全部进行赋值。特定于enable()函数来说，就是将所有的输入变量赋值为“yes”。由此可见，“enable avcodec”实际上相当于执行了：

**[python]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44587465) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44587465)

[在CODE上查看代码片](https://code.csdn.net/snippets/626422)

1. avcodec=”yes”

## Parse Options

Parse Options部分用于解析Configure的附加参数。该部分的代码如下所示。

**[python]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44587465) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44587465)

[在CODE上查看代码片](https://code.csdn.net/snippets/626422)

1. #注意：opt不是参数列表（实际上也没有看见opt变量的定义）
2. #原因是处在for循环中，当你没有为in指定列表时，for会默认取命令行参数列表。
3. #因此“opt”这个名字实际上是可以随便取的
4. **for** opt do
5. # "#"用于去除特定字符前面的字符串
6. # optval内容为opt去掉"="以及其前面字符串之后的内容
7. optval="${opt#\*=}"
8. case "$opt" **in**
9. # 不同的选项
10. --extra-ldflags=\*) add\_ldflags $optval
11. ;;
12. --extra-libs=\*) add\_extralibs $optval
13. ;;
14. --disable-devices) disable $INDEV\_LIST $OUTDEV\_LIST
15. ;;
16. --enable-debug=\*) debuglevel="$optval"
17. ;;
18. --disable-everything)
19. map 'eval unset \${$(toupper ${v%s})\_LIST}' $COMPONENT\_LIST
20. ;;
21. --enable-\*=\*|--disable-\*=\*)
22. eval $(echo "${opt%%=\*}" | sed 's/--/action=/;s/-/ thing=/')
23. is\_in "${thing}s" $COMPONENT\_LIST || die\_unknown "$opt"
24. eval list=\$$(toupper $thing)\_LIST
25. name=$(echo "${optval}" | sed "s/,/\_${thing}|/g")\_${thing}
26. $action $(filter "$name" $list)
27. ;;
28. --enable-?\*|--disable-?\*)
29. eval $(echo "$opt" | sed 's/--/action=/;s/-/ option=/;s/-/\_/g')
30. **if** is\_in $option $COMPONENT\_LIST; then
31. test $action = disable && action=unset
32. eval $action \$$(toupper ${option%s})\_LIST
33. **elif** is\_in $option $CMDLINE\_SELECT; then
34. $action $option
35. **else**
36. die\_unknown $opt
37. fi
38. ;;
39. --list-\*)
40. NAME="${opt#--list-}"
41. is\_in $NAME $COMPONENT\_LIST || die\_unknown $opt
42. NAME=${NAME%s}
43. eval show\_list $NAME \$$(toupper $NAME)\_LIST
44. ;;
45. --help|-h) show\_help
46. ;;
47. \*)
48. #% 就是从右边开始删除符合条件的字符串（符合条件的最短字符串）
49. #%%是删除符合条件的最长的字符串
51. #删除“=”右边的内容
52. optname="${opt%%=\*}"
53. #删除左边的“--”
54. optname="${optname#--}"
55. optname=$(echo "$optname" | sed 's/-/\_/g')
56. #看看是否在opt列表中，不在的话就会返回错误
57. **if** is\_in $optname $CMDLINE\_SET; then
58. eval $optname='$optval'
59. **elif** is\_in $optname $CMDLINE\_APPEND; then
60. append $optname "$optval"
61. **else**
62. die\_unknown $opt
63. fi
64. ;;
65. esac
66. done

在这里需要注意，取出opt的值一般都是“--extra-ldflags=XXX”的形式，通过“${opt#\*=}”截取获得“=”号后面的内容作为optval，对于“--extra-ldflags=XXX”来说，optval取值为“XXX”。  
然后根据opt种类的不同，以及optval取值的不同，分别作不同的处理。

## Check Compiler

Check Compiler用于检查编译器。这部分代码还没有细看，暂时不做分析。

## die\_license\_disabled()

die\_license\_disabled()用于检查是否指定了特定了License。像libx264、libfaac这些第三方类库，都需要指定特定的License才可以使用（例如libfaac必须指定nonfree）。开启这些第三方类库后如果没有指定License，Configure会立刻退出。这部分代码如下所示。

**[python]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44587465) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44587465)

[在CODE上查看代码片](https://code.csdn.net/snippets/626422)

1. #检查License
2. #GPL
3. die\_license\_disabled gpl libcdio
4. die\_license\_disabled gpl libx264
5. die\_license\_disabled gpl libxavs
6. die\_license\_disabled gpl libxvid
7. die\_license\_disabled gpl x11grab
8. #nonfree
9. die\_license\_disabled nonfree libaacplus
10. die\_license\_disabled nonfree libfaac
11. die\_license\_disabled nonfree openssl
12. #Version3
13. die\_license\_disabled version3 libopencore\_amrnb
14. die\_license\_disabled version3 libopencore\_amrwb
15. die\_license\_disabled version3 libvo\_aacenc
16. die\_license\_disabled version3 libvo\_amrwbenc

其中涉及到一个函数die\_license\_disabled()，它的定义如下所示。

**[python]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44587465) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44587465)

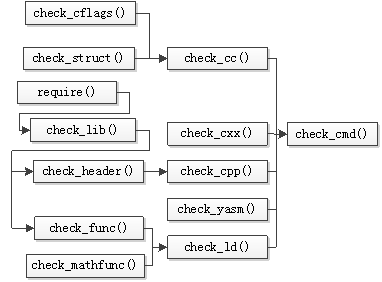
[在CODE上查看代码片](https://code.csdn.net/snippets/626422)

1. #不符合License则立刻结束
2. die\_license\_disabled() {
3. enabled $1 || { enabled $2 && die "$2 is $1 and --enable-$1 is not specified."; }
4. }

从定义可以看出，die\_license\_disabled()首先会看第1个参数（对应“gpl”，“nonfree”）对应的组件是否enable，如果已经enable，则正常运行完函数；如果没有enable，则会检查第2个参数（对应“libx264”，“libfaac”）是否enable，如果第2个参数enable了，就会报错退出。

## Check

Check部分是Configure中最重要的部分。该部分用于检查编译环境（例如数学函数，第三方类库等）。这一部分涉及到很多的函数。包括check\_cflags()、check\_struct()、require()、check\_lib()、check\_header()、check\_func()、check\_mathfunc()等等。这些函数之间的调用关系如下图所示。



下面简单举例下面几个函数：

check\_func()：用于检查函数。

check\_header()：用于检查头文件。

check\_func\_headers()：用于同时检查头文件和函数。

check\_mathfunc()：用于检查数学类函数。

require()：检查第三方类库。

check\_cflags ()：用于检查编译器的cflags标志参数。

下面详细看看这几个函数。

### check\_func()

check\_func()用于检查函数。它的输入参数一个函数名。Configure中与check\_func()有关的代码如下所示。

**[python]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44587465) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44587465)

[在CODE上查看代码片](https://code.csdn.net/snippets/626422)

1. check\_func  isatty
2. check\_func  localtime\_r
3. check\_func  ${malloc\_prefix}memalign            && enable memalign
4. check\_func  mkstemp
5. check\_func  mmap
6. check\_func  ${malloc\_prefix}posix\_memalign      && enable posix\_memalign
7. check\_func  setrlimit
8. check\_func  strerror\_r
9. check\_func  strptime
10. check\_func  sched\_getaffinity
11. check\_func  sysconf
12. check\_func  sysctl

check\_func()的定义如下所示。

**[python]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44587465) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44587465)

[在CODE上查看代码片](https://code.csdn.net/snippets/626422)

1. check\_func(){
2. log check\_func "$@"
3. func=$1
4. shift
5. disable $func
6. check\_ld "cc" "$@" <<EOF && enable $func
7. extern int $func();
8. int main(void){ $func(); }
9. EOF
10. }

从check\_func()的定义可以看出，该函数首先将输入的第1个参数赋值给func，然后生成一个下述内容的C语言文件。

**[python]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44587465) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44587465)

[在CODE上查看代码片](https://code.csdn.net/snippets/626422)

1. extern int $func();
2. int main(void){ $func(); }

最后调用check\_ld()完成编译测试。check\_ld()的定义如下所示。

**[python]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44587465) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44587465)

[在CODE上查看代码片](https://code.csdn.net/snippets/626422)

1. check\_ld(){
2. log check\_ld "$@"
3. type=$1
4. shift 1
5. flags=''
6. libs=''
7. **for** f; do
8. test "${f}" = "${f#-l}" && flags="$flags $f" || libs="$libs $f"
9. done
10. check\_$type $($filter\_cflags $flags) || **return**
11. check\_cmd $ld $LDFLAGS $flags -o $TMPE $TMPO $libs $extralibs
12. }

其中check\_cmd()是个很简单的函数，可以输出日志，如下所示。

**[python]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44587465) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44587465)

[在CODE上查看代码片](https://code.csdn.net/snippets/626422)

1. check\_cmd(){
2. log "$@"
3. "$@" >> $logfile 2>&1
4. }

例如，“check\_func  mkstemp”相当于编译了下述代码。

**[python]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44587465) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44587465)

[在CODE上查看代码片](https://code.csdn.net/snippets/626422)

1. extern int mkstemp();
2. int main(void){ mkstemp(); }

### check\_header()

check\_header()用于检查头文件。Configure中与check\_header()有关的代码如下所示。

**[python]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44587465) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44587465)

[在CODE上查看代码片](https://code.csdn.net/snippets/626422)

1. check\_header dlfcn.h
2. check\_header dxva2api.h -D\_WIN32\_WINNT=0x0600
3. check\_header libcrystalhd/libcrystalhd\_if.h
4. check\_header malloc.h
5. check\_header poll.h
6. check\_header sys/mman.h
7. check\_header sys/param.h
8. check\_header sys/resource.h
9. check\_header sys/select.h
10. check\_header termios.h
11. check\_header vdpau/vdpau.h
12. check\_header vdpau/vdpau\_x11.h
13. check\_header X11/extensions/XvMClib.h
14. check\_header asm/types.h

check\_header()的定义如下所示。

**[python]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44587465) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44587465)

[在CODE上查看代码片](https://code.csdn.net/snippets/626422)

1. check\_header(){
2. log check\_header "$@"
3. header=$1
4. shift
5. disable\_safe $header
6. check\_cpp "$@" <<EOF && enable\_safe $header
7. #include <$header>
8. int x;
9. EOF
10. }

从check\_header()的定义可以看出，该函数首先将输入的第1个参数赋值给header，然后生成一个下述内容的C语言文件。

**[python]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44587465) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44587465)

[在CODE上查看代码片](https://code.csdn.net/snippets/626422)

1. #include <$header>
2. int x;

最后调用check\_cpp()完成编译测试。check\_cpp()的定义如下所示。

**[python]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44587465) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44587465)

[在CODE上查看代码片](https://code.csdn.net/snippets/626422)

1. check\_cpp(){
2. log check\_cpp "$@"
3. cat > $TMPC
4. log\_file $TMPC
5. #-E选项，可以让编译器在预处理后停止，并输出预处理结果。
6. check\_cmd $cc $CPPFLAGS $CFLAGS "$@" -E -o $TMPO $TMPC
7. }

例如，“check\_header malloc.h”相当于处理以下C语言文件。

**[python]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44587465) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44587465)

[在CODE上查看代码片](https://code.csdn.net/snippets/626422)

1. #include <malloc.h>
2. int x;

### check\_func\_headers()

check\_func\_headers()用于同时检查头文件和函数。Configure中与check\_header()有关的代码如下所示。

**[python]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44587465) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44587465)

[在CODE上查看代码片](https://code.csdn.net/snippets/626422)

1. check\_func\_headers conio.h kbhit
2. check\_func\_headers windows.h PeekNamedPipe
3. check\_func\_headers io.h setmode
4. check\_func\_headers lzo/lzo1x.h lzo1x\_999\_compress
5. check\_func\_headers windows.h GetProcessAffinityMask
6. check\_func\_headers windows.h GetProcessTimes
7. check\_func\_headers windows.h MapViewOfFile
8. check\_func\_headers windows.h VirtualAlloc

check\_func\_headers()的定义如下所示。

**[python]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44587465) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44587465)

[在CODE上查看代码片](https://code.csdn.net/snippets/626422)

1. check\_func\_headers(){
2. log check\_func\_headers "$@"
3. headers=$1
4. funcs=$2
5. shift 2
6. {
7. **for** hdr **in** $headers; do
8. echo "#include <$hdr>"
9. done
10. **for** func **in** $funcs; do
11. echo "long check\_$func(void) { return (long) $func; }"
12. done
13. echo "int main(void) { return 0; }"
14. } | check\_ld "cc" "$@" && enable $funcs && enable\_safe $headers
15. }

从check\_func\_headers()的定义可以看出，该函数首先将输入的第1个参数赋值给header，第2个参数赋值给funcs，然后生成一个下述内容的C语言文件。

**[python]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44587465) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44587465)

[在CODE上查看代码片](https://code.csdn.net/snippets/626422)

1. #include <$hdr>
2. long check\_$func (void) { **return** (long) $func; }
3. int main(void) { **return** 0; }

例如，“check\_func\_headers windows.h PeekNamedPipe”相当于处理以下C语言文件。

**[python]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44587465) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44587465)

[在CODE上查看代码片](https://code.csdn.net/snippets/626422)

1. #include <windows.h>
2. long check\_PeekNamedPipe (void) { **return** (long) PeekNamedPipe; }
3. int main(void) { **return** 0; }

### check\_mathfunc()

check\_mathfunc()用于检查数学类函数。Configure中与check\_mathfunc()有关的代码如下所示。

**[python]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44587465) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44587465)

[在CODE上查看代码片](https://code.csdn.net/snippets/626422)

1. check\_mathfunc cbrtf
2. check\_mathfunc exp2
3. check\_mathfunc exp2f
4. check\_mathfunc llrint
5. check\_mathfunc llrintf
6. check\_mathfunc log2
7. check\_mathfunc log2f
8. check\_mathfunc lrint
9. check\_mathfunc lrintf
10. check\_mathfunc round
11. check\_mathfunc roundf
12. check\_mathfunc trunc
13. check\_mathfunc truncf

check\_mathfunc()的定义如下所示。

**[python]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44587465) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44587465)

[在CODE上查看代码片](https://code.csdn.net/snippets/626422)

1. check\_mathfunc(){
2. log check\_mathfunc "$@"
3. #数学函数名称
4. func=$1
5. shift
6. disable $func
7. check\_ld "cc" "$@" <<EOF && enable $func
8. #include <math.h>
9. float foo(float f) { **return** $func(f); }
10. int main(void){ **return** (int) foo; }
11. EOF
12. }

从check\_mathfunc()的定义可以看出，该函数首先将输入的第1个参数赋值给func，然后生成一个下述内容的C语言文件。

**[python]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44587465) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44587465)

[在CODE上查看代码片](https://code.csdn.net/snippets/626422)

1. #include <math.h>
2. float foo(float f) { **return** $func(f); }
3. int main(void){ **return** (int) foo; }

最后调用check\_ld()完成编译测试。  
例如，“check\_mathfunc exp2”相当于编译连接了下面这个C文件。

**[python]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44587465) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44587465)

[在CODE上查看代码片](https://code.csdn.net/snippets/626422)

1. #include <math.h>
2. float foo(float f) { **return** exp2(f); }
3. int main(void){ **return** (int) foo; }

### require()

require()用于检查第三方类库。Configure中与require()有关的代码如下所示。

**[python]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44587465) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44587465)

[在CODE上查看代码片](https://code.csdn.net/snippets/626422)

1. #检查第三方类库
2. # these are off by default, so fail if requested and not available
3. #require()函数参数的规范：（名称，头文件，函数名，附加选项）
4. #require2()函数参数规范类似
5. enabled avisynth   && require2 vfw32 "windows.h vfw.h" AVIFileInit -lavifil32
6. enabled frei0r     && { check\_header frei0r.h || die "ERROR: frei0r.h header not found"; }
7. enabled gnutls     && require\_pkg\_config gnutls gnutls/gnutls.h gnutls\_global\_init
8. enabled libaacplus && require  "libaacplus >= 2.0.0" aacplus.h aacplusEncOpen -laacplus
9. enabled libass     && require\_pkg\_config libass ass/ass.h ass\_library\_init
10. enabled libcelt    && require libcelt celt/celt.h celt\_decode -lcelt0 &&
11. { check\_lib celt/celt.h celt\_decoder\_create\_custom -lcelt0 ||
12. die "ERROR: libcelt version must be >= 0.11.0."; }
13. enabled libdc1394  && require\_pkg\_config libdc1394-2 dc1394/dc1394.h dc1394\_new
14. enabled libdirac   && require\_pkg\_config dirac                          \
15. "libdirac\_decoder/dirac\_parser.h libdirac\_encoder/dirac\_encoder.h"  \
16. "dirac\_decoder\_init dirac\_encoder\_init"
17. #测试libfaac
18. enabled libfaac    && require2 libfaac "stdint.h faac.h" faacEncGetVersion -lfaac
19. enabled libfreetype && require\_pkg\_config freetype2 "ft2build.h freetype/freetype.h" FT\_Init\_FreeType
20. enabled libgsm     && require  libgsm gsm/gsm.h gsm\_create -lgsm
21. enabled libmodplug && require  libmodplug libmodplug/modplug.h ModPlug\_Load -lmodplug
22. enabled libmp3lame && require  "libmp3lame >= 3.98.3" lame/lame.h lame\_set\_VBR\_quality -lmp3lame
23. enabled libnut     && require  libnut libnut.h nut\_demuxer\_init -lnut
24. enabled libopencore\_amrnb  && require libopencore\_amrnb opencore-amrnb/interf\_dec.h Decoder\_Interface\_init -lopencore-amrnb
25. enabled libopencore\_amrwb  && require libopencore\_amrwb opencore-amrwb/dec\_if.h D\_IF\_init -lopencore-amrwb
26. enabled libopencv  && require\_pkg\_config opencv opencv/cxcore.h cvCreateImageHeader
27. enabled libopenjpeg && require libopenjpeg openjpeg.h opj\_version -lopenjpeg
28. enabled libpulse && require\_pkg\_config libpulse-simple pulse/simple.h pa\_simple\_new
29. enabled librtmp    && require\_pkg\_config librtmp librtmp/rtmp.h RTMP\_Socket
30. enabled libschroedinger && require\_pkg\_config schroedinger-1.0 schroedinger/schro.h schro\_init
31. enabled libspeex   && require  libspeex speex/speex.h speex\_decoder\_init -lspeex
32. enabled libstagefright\_h264  && require\_cpp libstagefright\_h264 "binder/ProcessState.h media/stagefright/MetaData.h
33. media/stagefright/MediaBufferGroup.h media/stagefright/MediaDebug.h media/stagefright/MediaDefs.h
34. media/stagefright/OMXClient.h media/stagefright/OMXCodec.h" android::OMXClient -lstagefright -lmedia -lutils -lbinder
35. enabled libtheora  && require  libtheora theora/theoraenc.h th\_info\_init -ltheoraenc -ltheoradec -logg
36. enabled libutvideo    && require\_cpp utvideo "stdint.h stdlib.h utvideo/utvideo.h utvideo/Codec.h" 'CCodec\*' -lutvideo -lstdc++
37. enabled libv4l2    && require\_pkg\_config libv4l2 libv4l2.h v4l2\_ioctl
38. enabled libvo\_aacenc && require libvo\_aacenc vo-aacenc/voAAC.h voGetAACEncAPI -lvo-aacenc
39. enabled libvo\_amrwbenc && require libvo\_amrwbenc vo-amrwbenc/enc\_if.h E\_IF\_init -lvo-amrwbenc
40. enabled libvorbis  && require  libvorbis vorbis/vorbisenc.h vorbis\_info\_init -lvorbisenc -lvorbis -logg
41. enabled libvpx     && {
42. enabled libvpx\_decoder && { check\_lib2 "vpx/vpx\_decoder.h vpx/vp8dx.h" vpx\_codec\_dec\_init\_ver -lvpx ||
43. die "ERROR: libvpx decoder version must be >=0.9.1"; }
44. enabled libvpx\_encoder && { check\_lib2 "vpx/vpx\_encoder.h vpx/vp8cx.h" "vpx\_codec\_enc\_init\_ver VPX\_CQ" -lvpx ||
45. die "ERROR: libvpx encoder version must be >=0.9.6"; } }
46. #测试libx264
47. enabled libx264    && require  libx264 x264.h x264\_encoder\_encode -lx264 &&
48. { check\_cpp\_condition x264.h "X264\_BUILD >= 118" ||
49. die "ERROR: libx264 version must be >= 0.118."; }
50. enabled libxavs    && require  libxavs xavs.h xavs\_encoder\_encode -lxavs
51. enabled libxvid    && require  libxvid xvid.h xvid\_global -lxvidcore
52. enabled openal     && { { **for** al\_libs **in** "${OPENAL\_LIBS}" "-lopenal" "-lOpenAL32"; do
53. check\_lib 'AL/al.h' alGetError "${al\_libs}" && **break**; done } ||
54. die "ERROR: openal not found"; } &&
55. { check\_cpp\_condition "AL/al.h" "defined(AL\_VERSION\_1\_1)" ||
56. die "ERROR: openal version must be 1.1 or compatible"; }
57. enabled mlib       && require  mediaLib mlib\_types.h mlib\_VectorSub\_S16\_U8\_Mod -lmlib
58. enabled openssl    && { check\_lib openssl/ssl.h SSL\_library\_init -lssl -lcrypto ||
59. check\_lib openssl/ssl.h SSL\_library\_init -lssl32 -leay32 ||
60. check\_lib openssl/ssl.h SSL\_library\_init -lssl -lcrypto -lws2\_32 -lgdi32 ||
61. die "ERROR: openssl not found"; }

由于上述代码量比较大。在这里我们只选择一个典型的例子——libx264来看一下。require()检测libx264的定义如下所示。

**[python]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44587465) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44587465)

[在CODE上查看代码片](https://code.csdn.net/snippets/626422)

1. #测试libx264
2. require libx264 x264.h x264\_encoder\_encode -lx264 &&
3. { check\_cpp\_condition x264.h "X264\_BUILD >= 118" ||
4. die "ERROR: libx264 version must be >= 0.118."; }

从测试libx264的代码可以看出，require()函数的使用方式为：

**[plain]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44587465) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44587465)

[在CODE上查看代码片](https://code.csdn.net/snippets/626422)

1. require {名称} {头文件} {函数名} {附加选项}

require()函数定义如下所示。

**[python]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44587465) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44587465)

[在CODE上查看代码片](https://code.csdn.net/snippets/626422)

1. #检查依赖项的时候使用
2. require(){
3. name="$1"
4. header="$2"
5. func="$3"
6. shift 3
7. check\_lib $header $func "$@" || die "ERROR: $name not found"
8. }

从require()的定义可以看出，该函数将第1个参数赋值给name，第2个参数赋值给header，第3个参数赋值给func。最后调用check\_lib()函数。  
check\_lib()的定义如下所示。

**[python]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44587465) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44587465)

[在CODE上查看代码片](https://code.csdn.net/snippets/626422)

1. #检查类库
2. check\_lib(){
3. log check\_lib "$@"
4. header="$1"
5. func="$2"
6. shift 2
7. check\_header $header && check\_func $func "$@" && add\_extralibs "$@"
8. }

可以看出check\_lib()调用了check\_header()、check\_func()等几个函数完成检查工作。这两个函数在前文中已经介绍过，就不再重复了。  
例如检测libx264的时候调用check\_header()会生成以下临时文件：

**[python]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44587465) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44587465)

[在CODE上查看代码片](https://code.csdn.net/snippets/626422)

1. #include <x264.h>
2. int x;

### check\_cflags ()

check\_cflags()用于检查编译器的cflags标志参数。Configure中与check\_cflags()有关的代码如下所示。

**[python]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44587465) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44587465)

[在CODE上查看代码片](https://code.csdn.net/snippets/626422)

1. #添加一些编译选项
2. # add some useful compiler flags if supported
3. check\_cflags -Wdeclaration-after-statement
4. check\_cflags -Wall
5. check\_cflags -Wno-parentheses
6. check\_cflags -Wno-switch
7. check\_cflags -Wno-format-zero-length
8. check\_cflags -Wdisabled-optimization
9. check\_cflags -Wpointer-arith
10. check\_cflags -Wredundant-decls
11. check\_cflags -Wno-pointer-sign
12. check\_cflags -Wcast-qual
13. check\_cflags -Wwrite-strings
14. check\_cflags -Wtype-limits
15. check\_cflags -Wundef
16. check\_cflags -Wmissing-prototypes
17. check\_cflags -Wno-pointer-to-int-cast
18. check\_cflags -Wstrict-prototypes
19. check\_cflags()函数的定义如下所示。
20. check\_cflags(){
21. log check\_cflags "$@"
22. set -- $($filter\_cflags "$@")
23. check\_cc "$@" <<EOF && append CFLAGS "$@"
24. int x;
25. EOF
26. }

从定义可以看出，check\_cflags()调用了check\_cc()执行命令。整个代码只有一行：

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44587465) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44587465)

[在CODE上查看代码片](https://code.csdn.net/snippets/626422)

1. **int** x;

check\_cc()的定义如下所示。

**[python]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44587465) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44587465)

[在CODE上查看代码片](https://code.csdn.net/snippets/626422)

1. check\_cc(){
2. log check\_cc "$@"
3. cat > $TMPC
4. log\_file $TMPC
5. #很多检查都调用了这个check\_cmd
6. #-c 只编译不链接
7. check\_cmd $cc $CPPFLAGS $CFLAGS "$@" -c -o $TMPO $TMPC
8. }

可以看出check\_cflags()将输入参数设置到命令行中，并最终调用了check\_cmd。  
除了上述几个函数之外，还有其他的一些检查编译环境的函数，在这里就不一一列举了。

## Echo info

Echo info用于在控制台上打印配置信息。Configure中该部分的代码如下所示。

**[python]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44587465) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44587465)

[在CODE上查看代码片](https://code.csdn.net/snippets/626422)

1. #在控制台输出信息
2. echo "install prefix            $prefix"
3. echo "source path               $source\_path"
4. echo "C compiler                $cc"
5. echo "ARCH                      $arch ($cpu)"
6. **if** test "$build\_suffix" != ""; then
7. echo "build suffix              $build\_suffix"
8. fi
9. **if** test "$progs\_suffix" != ""; then
10. echo "progs suffix              $progs\_suffix"
11. fi
12. **if** test "$extra\_version" != ""; then
13. echo "version string suffix     $extra\_version"
14. fi
15. #${}的特异功能：
16. #${file-my.file.txt}假如 $file 为空值，则使用 my.file.txt 作默认值。(保留没设定及非空值)
17. #在这里，如果某个变量为空值，则取默认值为no
18. echo "big-endian                ${bigendian-no}"
19. echo "runtime cpu detection     ${runtime\_cpudetect-no}"
20. **if** enabled x86; then
21. echo "${yasmexe}                      ${yasm-no}"
22. echo "MMX enabled               ${mmx-no}"
23. echo "MMX2 enabled              ${mmx2-no}"
24. echo "3DNow! enabled            ${amd3dnow-no}"
25. echo "3DNow! extended enabled   ${amd3dnowext-no}"
26. echo "SSE enabled               ${sse-no}"
27. echo "SSSE3 enabled             ${ssse3-no}"
28. echo "AVX enabled               ${avx-no}"
29. echo "CMOV enabled              ${cmov-no}"
30. echo "CMOV is fast              ${fast\_cmov-no}"
31. echo "EBX available             ${ebx\_available-no}"
32. echo "EBP available             ${ebp\_available-no}"
33. fi
34. **if** enabled arm; then
35. echo "ARMv5TE enabled           ${armv5te-no}"
36. echo "ARMv6 enabled             ${armv6-no}"
37. echo "ARMv6T2 enabled           ${armv6t2-no}"
38. echo "ARM VFP enabled           ${armvfp-no}"
39. echo "IWMMXT enabled            ${iwmmxt-no}"
40. echo "NEON enabled              ${neon-no}"
41. fi
42. **if** enabled mips; then
43. echo "MMI enabled               ${mmi-no}"
44. fi
45. **if** enabled ppc; then
46. echo "AltiVec enabled           ${altivec-no}"
47. echo "PPC 4xx optimizations     ${ppc4xx-no}"
48. echo "dcbzl available           ${dcbzl-no}"
49. fi
50. **if** enabled sparc; then
51. echo "VIS enabled               ${vis-no}"
52. fi
53. echo "debug symbols             ${debug-no}"
54. echo "strip symbols             ${stripping-no}"
55. echo "optimize for size         ${small-no}"
56. echo "optimizations             ${optimizations-no}"
57. echo "static                    ${static-no}"
58. echo "shared                    ${shared-no}"
59. echo "postprocessing support    ${postproc-no}"
60. echo "new filter support        ${avfilter-no}"
61. echo "network support           ${network-no}"
62. echo "threading support         ${thread\_type-no}"
63. echo "safe bitstream reader     ${safe\_bitstream\_reader-no}"
64. echo "SDL support               ${sdl-no}"
65. echo "Sun medialib support      ${mlib-no}"
66. echo "libdxva2 enabled          ${dxva2-no}"
67. echo "libva enabled             ${vaapi-no}"
68. echo "libvdpau enabled          ${vdpau-no}"
69. echo "AVISynth enabled          ${avisynth-no}"
70. echo "frei0r enabled            ${frei0r-no}"
71. echo "gnutls enabled            ${gnutls-no}"
72. echo "libaacplus enabled        ${libaacplus-no}"
73. echo "libass enabled            ${libass-no}"
74. echo "libcdio support           ${libcdio-no}"
75. echo "libcelt enabled           ${libcelt-no}"
76. echo "libdc1394 support         ${libdc1394-no}"
77. echo "libdirac enabled          ${libdirac-no}"
78. echo "libfaac enabled           ${libfaac-no}"
79. echo "libgsm enabled            ${libgsm-no}"
80. echo "libmodplug enabled        ${libmodplug-no}"
81. echo "libmp3lame enabled        ${libmp3lame-no}"
82. echo "libnut enabled            ${libnut-no}"
83. echo "libopencore-amrnb support ${libopencore\_amrnb-no}"
84. echo "libopencore-amrwb support ${libopencore\_amrwb-no}"
85. echo "libopencv support         ${libopencv-no}"
86. echo "libopenjpeg enabled       ${libopenjpeg-no}"
87. echo "libpulse enabled          ${libpulse-no}"
88. echo "librtmp enabled           ${librtmp-no}"
89. echo "libschroedinger enabled   ${libschroedinger-no}"
90. echo "libspeex enabled          ${libspeex-no}"
91. echo "libstagefright-h264 enabled    ${libstagefright\_h264-no}"
92. echo "libtheora enabled         ${libtheora-no}"
93. echo "libutvideo enabled        ${libutvideo-no}"
94. echo "libv4l2 enabled           ${libv4l2-no}"
95. echo "libvo-aacenc support      ${libvo\_aacenc-no}"
96. echo "libvo-amrwbenc support    ${libvo\_amrwbenc-no}"
97. echo "libvorbis enabled         ${libvorbis-no}"
98. echo "libvpx enabled            ${libvpx-no}"
99. echo "libx264 enabled           ${libx264-no}"
100. echo "libxavs enabled           ${libxavs-no}"
101. echo "libxvid enabled           ${libxvid-no}"
102. echo "openal enabled            ${openal-no}"
103. echo "openssl enabled           ${openssl-no}"
104. echo "zlib enabled              ${zlib-no}"
105. echo "bzlib enabled             ${bzlib-no}"
106. echo
108. **for** type **in** decoder encoder hwaccel parser demuxer muxer protocol filter bsf indev outdev; do
109. echo "Enabled ${type}s:"
110. eval list=\$$(toupper $type)\_LIST
111. print\_enabled '\_\*' $list | sort | pr -r -3 -t
112. echo
113. done
115. license="LGPL version 2.1 or later"
116. **if** enabled nonfree; then
117. license="nonfree and unredistributable"
118. **elif** enabled gplv3; then
119. license="GPL version 3 or later"
120. **elif** enabled lgplv3; then
121. license="LGPL version 3 or later"
122. **elif** enabled gpl; then
123. license="GPL version 2 or later"
124. fi
126. echo "License: $license"

有关这段代码，有一个地方需要注意：很多的${}符号中的字符为“XXX-no”，这种格式的意思是如果XXX取值为空，则使用默认值“no”（这个规则比较奇特）。

## Write basic info

Write basic info用于向config.mak中写入一些基本信息。Configure中该部分的代码如下所示。

**[python]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44587465) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44587465)

[在CODE上查看代码片](https://code.csdn.net/snippets/626422)

1. #创建config.mak和config.h
2. #根据情况也会创建config.asm
3. echo "Creating config.mak and config.h..."
5. test -e Makefile || $ln\_s "$source\_path/Makefile" .
7. enabled stripping || strip="echo skipping strip"
8. #重要：需要输出的文件
9. #TMPH是一个临时文件，最终会拷贝给config.h
10. config\_files="$TMPH config.mak"
11. #写入config.mak文件
12. #首先写入一些基本信息
13. #"<<EOF"表示后续的输入作为子命令或子shell的输入，直到遇到"EOF"，再次返回到
14. #主调shell，可将其理解为分界符（delimiter）。
15. #最后的"EOF"必须单独占一行
16. cat > config.mak <<EOF
17. # Automatically generated by configure - do not modify!
18. ifndef FFMPEG\_CONFIG\_MAK
19. FFMPEG\_CONFIG\_MAK=1
20. FFMPEG\_CONFIGURATION=$FFMPEG\_CONFIGURATION
21. prefix=$prefix
22. LIBDIR=\$(DESTDIR)$libdir
23. SHLIBDIR=\$(DESTDIR)$shlibdir
24. INCDIR=\$(DESTDIR)$incdir
25. BINDIR=\$(DESTDIR)$bindir
26. DATADIR=\$(DESTDIR)$datadir
27. MANDIR=\$(DESTDIR)$mandir
28. SRC\_PATH=$source\_path
29. ifndef MAIN\_MAKEFILE
30. SRC\_PATH:=\$(SRC\_PATH:.%=..%)
31. endif
32. CC\_IDENT=$cc\_ident
33. ARCH=$arch
34. CC=$cc
35. CXX=$cxx
36. AS=$as
37. LD=$ld
38. DEPCC=$dep\_cc
39. YASM=$yasmexe
40. YASMDEP=$yasmexe
41. AR=$ar
42. RANLIB=$ranlib
43. CP=cp -p
44. LN\_S=$ln\_s
45. STRIP=$strip
46. CPPFLAGS=$CPPFLAGS
47. CFLAGS=$CFLAGS
48. CXXFLAGS=$CXXFLAGS
49. ASFLAGS=$ASFLAGS
50. AS\_O=$CC\_O
51. CC\_O=$CC\_O
52. CXX\_O=$CXX\_O
53. LDFLAGS=$LDFLAGS
54. FFSERVERLDFLAGS=$FFSERVERLDFLAGS
55. SHFLAGS=$SHFLAGS
56. YASMFLAGS=$YASMFLAGS
57. BUILDSUF=$build\_suffix
58. PROGSSUF=$progs\_suffix
59. FULLNAME=$FULLNAME
60. LIBPREF=$LIBPREF
61. LIBSUF=$LIBSUF
62. LIBNAME=$LIBNAME
63. SLIBPREF=$SLIBPREF
64. SLIBSUF=$SLIBSUF
65. EXESUF=$EXESUF
66. EXTRA\_VERSION=$extra\_version
67. DEPFLAGS=$DEPFLAGS
68. CCDEP=$CCDEP
69. CXXDEP=$CXXDEP
70. ASDEP=$ASDEP
71. CC\_DEPFLAGS=$CC\_DEPFLAGS
72. AS\_DEPFLAGS=$AS\_DEPFLAGS
73. HOSTCC=$host\_cc
74. HOSTCFLAGS=$host\_cflags
75. HOSTEXESUF=$HOSTEXESUF
76. HOSTLDFLAGS=$host\_ldflags
77. HOSTLIBS=$host\_libs
78. TARGET\_EXEC=$target\_exec
79. TARGET\_PATH=$target\_path
80. SDL\_LIBS=$sdl\_libs
81. SDL\_CFLAGS=$sdl\_cflags
82. LIB\_INSTALL\_EXTRA\_CMD=$LIB\_INSTALL\_EXTRA\_CMD
83. EXTRALIBS=$extralibs
84. INSTALL=$install
85. LIBTARGET=${LIBTARGET}
86. SLIBNAME=${SLIBNAME}
87. SLIBNAME\_WITH\_VERSION=${SLIBNAME\_WITH\_VERSION}
88. SLIBNAME\_WITH\_MAJOR=${SLIBNAME\_WITH\_MAJOR}
89. SLIB\_CREATE\_DEF\_CMD=${SLIB\_CREATE\_DEF\_CMD}
90. SLIB\_EXTRA\_CMD=${SLIB\_EXTRA\_CMD}
91. SLIB\_INSTALL\_NAME=${SLIB\_INSTALL\_NAME}
92. SLIB\_INSTALL\_LINKS=${SLIB\_INSTALL\_LINKS}
93. SLIB\_INSTALL\_EXTRA\_LIB=${SLIB\_INSTALL\_EXTRA\_LIB}
94. SLIB\_INSTALL\_EXTRA\_SHLIB=${SLIB\_INSTALL\_EXTRA\_SHLIB}
95. SAMPLES:=${samples:-\$(FATE\_SAMPLES)}
96. NOREDZONE\_FLAGS=$noredzone\_flags
97. EOF
98. #获取版本
99. #主要通过各个类库文件夹中的version.h文件
100. #读取XXX\_VERSION（相当于把头文件当成一个文本来读）
101. get\_version(){
102. name=$1
103. file=$source\_path/$2
104. # This condition will be removed when we stop supporting old libpostproc versions
105. **if** ! test "$name" = LIBPOSTPROC || test "$postproc\_version" = current; then
106. eval $(grep "#define ${name}\_VERSION\_M" "$file" | awk '{ print $2"="$3 }')
107. eval ${name}\_VERSION=\$${name}\_VERSION\_MAJOR.\$${name}\_VERSION\_MINOR.\$${name}\_VERSION\_MICRO
108. fi
109. lcname=$(tolower $name)
110. eval echo "${lcname}\_VERSION=\$${name}\_VERSION" >> config.mak
111. eval echo "${lcname}\_VERSION\_MAJOR=\$${name}\_VERSION\_MAJOR" >> config.mak
112. }
113. #获取版本
114. get\_version LIBAVCODEC  libavcodec/version.h
115. get\_version LIBAVDEVICE libavdevice/avdevice.h
116. get\_version LIBAVFILTER libavfilter/version.h
117. get\_version LIBAVFORMAT libavformat/version.h
118. get\_version LIBAVUTIL   libavutil/avutil.h
119. get\_version LIBPOSTPROC libpostproc/postprocess.h
120. get\_version LIBSWRESAMPLE libswresample/swresample.h
121. get\_version LIBSWSCALE  libswscale/swscale.h

关于这段代码，有以下几点需要注意：  
（1）“cat > config.mak <<EOF”的作用就是往config.mak中写入文本，当遇到“EOF”的时候写入结束  
（2）get\_version()用于获取当前的FFmpeg源代码中各个类库的版本。通过把各个类库文件夹下的version.h当作文本读取之后，分析字符串并且得到版本号，最终写入config.mak文件。

## print\_config()

print\_config()用于向config.h、config.mak、config.asm中写入所有配置信息。Configure中该部分的代码如下所示。

**[python]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44587465) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44587465)

[在CODE上查看代码片](https://code.csdn.net/snippets/626422)

1. #输出所有的配置信息包含3类：
2. #以“ARCH\_”开头，包含系统架构信息
3. #以“HAVE\_”开头，包含系统特征信息
4. #以“CONFIG\_”开头，包含编译配置（数量最多，包含协议、复用器、编解码器等的配置，将近1000行）
5. #config\_files
6. print\_config ARCH\_   "$config\_files" $ARCH\_LIST
7. print\_config HAVE\_   "$config\_files" $HAVE\_LIST
8. print\_config CONFIG\_ "$config\_files" $CONFIG\_LIST       \
9. $CONFIG\_EXTRA      \
10. $ALL\_COMPONENTS    \

从源代码中可以看出，其中调用了一个函数print\_config()。print\_config()的源代码如下所示。

**[python]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44587465) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44587465)

[在CODE上查看代码片](https://code.csdn.net/snippets/626422)

1. # 输出文本到config.mak，config.h等文件
2. # 该函数的示例调用方法：print\_config CONFIG\_ "$config\_files" $CONFIG\_LIST
3. print\_config(){
4. # 前缀
5. pfx=$1
6. # 文件列表
7. files=$2
8. # 位置参数可以用shift命令左移。比如shift 3表示原来的$4现在变成$1
9. shift 2
10. #for循环中，当没有in指定列表时，for会默认取命令行参数列表。
11. #在这里取的就是$CONFIG\_LIST 等
12. **for** cfg; do
13. # toupper()：转换为大写
14. ucname="$(toupper $cfg)"
15. # files= config.h config.mak config.asm
16. # 循环输出
17. **for** f **in** $files; do
18. # "x#\*/"代表去取x的第一个slash之后的所有内容（不包括slash）
19. # "#"代表删除从前往后最小匹配的内容
20. # "f##\*."代表去取f的第一个"."之后的所有内容。在这里是“h”、“mak”等
21. # 在这里print\_config\_h(),print\_config\_mak(),print\_config\_asm()
22. "print\_config\_${f##\*.}" $cfg ${pfx}${ucname} >>$f
23. done
24. done
25. }

可以看出print\_config()的第1个参数是写入参数的前缀（例如可以取“ARCH\_”、“HAVE\_”、“CONFIG\_”）；第2个参数是文件列表（例如可以是“config.h config.mak config.asm”）；第3个以后的的参数就是需要写入的变量（例如$ARCH\_LIST、$CONFIG\_LIST等）。  
print\_config()有两层循环：外层循环遍历了所有的变量（例如$CONFIG\_LIST），内层循环遍历了所有文件（例如“config.h config.mak”），其中调用了一个函数print\_config\_XXX()，其中“XXX”根据文件后缀的不同可以取不同的值（例如“h”、“mak”）。下面举例看两个函数：print\_config\_h()和print\_config\_mak()。

### print\_config\_h()

print\_config\_h()用于输出配置信息至config.h。该函数的源代码如下所示。

**[python]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44587465) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44587465)

[在CODE上查看代码片](https://code.csdn.net/snippets/626422)

1. #输出config.h的时候使用
2. #调用示例：print\_config\_h ffplay CONFIG\_FFPLAY
3. print\_config\_h(){
4. #command1  && command2
5. #&&左边的命令（命令1）返回真(即返回0，成功被执行）后，&&右边的命令（命令2）才能够被执行
6. #command1 || command2
7. #||左边的命令（命令1）未执行成功，那么就执行||右边的命令（命令2）
8. enabled $1 && v=1 || v=0
9. #示例：#define CONFIG\_FFPLAY 1
10. echo "#define $2 $v"
11. }

从源代码中可以看出，参数1是变量名称，参数2是经过处理后准备写入文件的变量名称（变量名转换成了大写并且添加了前缀）。如果参数1所指向的变量是enabled的，那么v取值为1，那么写入文件的格式就是：

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44587465) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44587465)

[在CODE上查看代码片](https://code.csdn.net/snippets/626422)

1. #define {处理后变量名称} 1

如果参数1所指向的变量不是enabled的，那么v取值为0，那么写入文件的格式就是：

**[python]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44587465) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44587465)

[在CODE上查看代码片](https://code.csdn.net/snippets/626422)

1. #define {处理后变量名称} 0

### print\_config\_mak()

print\_config\_mak()用于输出配置信息至config.mak。该函数的源代码如下所示。

**[python]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44587465) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44587465)

[在CODE上查看代码片](https://code.csdn.net/snippets/626422)

1. #输出config.mak的时候使用
2. print\_config\_mak(){
3. enabled $1 && v= || v=!
4. echo "$v$2=yes"
5. }

从源代码中可以看出print\_config\_mak()的原理和print\_config\_h()是类似的。如果变量是enabled的，那么写入文件的格式就是：

**[plain]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44587465) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44587465)

[在CODE上查看代码片](https://code.csdn.net/snippets/626422)

1. {处理后变量名称}=yes

如果变量不是enabled的，那么写入文件的格式就是：

**[plain]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44587465) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44587465)

[在CODE上查看代码片](https://code.csdn.net/snippets/626422)

1. !{处理后变量名称}=yes

## print\_enabled()

print\_enabled()用于向config.mak写入所有enabled的组件信息。这方面功能通过print\_enabled()函数完成，就不再详细分析了。

## pkgconfig\_generate()

pkgconfig\_generate()用于向libavXXX/libavXXX.pc中写入pkgconfig信息（XXX代表avcodec，avformat等）。这方面的代码还没有细看，以后有机会再进行补充。

## 源代码（包含注释）

至此，FFmpeg的Configure的流程就大致梳理完毕了，最后附上和Configure有关的config.mak、config.h以及Configure本身的源代码。

### config.mak源代码

**[python]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44587465) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44587465)

[在CODE上查看代码片](https://code.csdn.net/snippets/626422)

1. # FFmpeg config.mak
2. #
3. # 注释：雷霄骅
4. # leixiaohua1020@126.com
5. # http://blog.csdn.net/leixiaohua1020
6. #
7. # Configure脚本生成的Makefile，包含了各种配置信息。
8. #
9. # Automatically generated by configure - do not modify!
10. #基本信息
11. ifndef FFMPEG\_CONFIG\_MAK
12. FFMPEG\_CONFIG\_MAK=1
13. FFMPEG\_CONFIGURATION=
14. #各种路径========================================
15. prefix=/usr/local
16. LIBDIR=$(DESTDIR)${prefix}/lib
17. SHLIBDIR=$(DESTDIR)${prefix}/bin
18. INCDIR=$(DESTDIR)${prefix}/include
19. BINDIR=$(DESTDIR)${prefix}/bin
20. DATADIR=$(DESTDIR)${prefix}/share/ffmpeg
21. MANDIR=$(DESTDIR)${prefix}/share/man
22. #是个相对路径
23. SRC\_PATH=.
24. ifndef MAIN\_MAKEFILE
25. SRC\_PATH:=$(SRC\_PATH:.%=..%)
26. endif
27. #工具集==========================================
28. CC\_IDENT=gcc 4.6.2 (GCC)
29. #架构
30. ARCH=x86
31. #编译器
32. CC=gcc
33. CXX=g++
34. AS=gcc
35. #链接器
36. LD=gcc
37. DEPCC=gcc
38. #汇编器
39. YASM=yasm
40. YASMDEP=yasm
41. #生成静态库.a工具
42. AR=ar
43. RANLIB=ranlib
44. CP=cp -p
45. LN\_S=ln -sf
46. STRIP=strip
47. #参数集==========================================
48. #编译器的参数
49. CPPFLAGS= -D\_ISOC99\_SOURCE -D\_FILE\_OFFSET\_BITS=64 -D\_LARGEFILE\_SOURCE -U\_\_STRICT\_ANSI\_\_
50. CFLAGS=   -std=c99 -fno-common -fomit-frame-pointer -I/include/SDL -D\_GNU\_SOURCE=1 -Dmain=SDL\_main -g -Wdeclaration-after-statement -Wall -Wno-parentheses -Wno-switch -Wno-format-zero-length -Wdisabled-optimization -Wpointer-arith -Wredundant-decls -Wno-pointer-sign -Wcast-qual -Wwrite-strings -Wtype-limits -Wundef -Wmissing-prototypes -Wno-pointer-to-int-cast -Wstrict-prototypes -O3 -fno-math-errno -fno-signed-zeros -fno-tree-vectorize -Werror=implicit-function-declaration -Werror=missing-prototypes
51. CXXFLAGS=  -D\_\_STDC\_CONSTANT\_MACROS
52. ASFLAGS=   -g
53. #目标文件有关的参数
54. AS\_O=-o $@
55. CC\_O=-o $@
56. CXX\_O=-o $@
57. #链接器有关的参数
58. LDFLAGS= -Wl,--as-needed -Wl,--warn-common -Wl,-rpath-link=libpostproc:libswresample:libswscale:libavfilter:libavdevice:libavformat:libavcodec:libavutil
59. FFSERVERLDFLAGS=-Wl,-E
60. SHFLAGS=-shared -Wl,--output-**def**,$$(@:$(SLIBSUF)=.**def**) -Wl,--out-implib,$(SUBDIR)lib$(SLIBNAME:$(SLIBSUF)=.dll.a) -Wl,--enable-runtime-pseudo-reloc -Wl,--enable-auto-image-base -Wl,-Bsymbolic -Wl,--version-script,$(SUBDIR)lib$(NAME).ver
61. YASMFLAGS=-f win32  -DPREFIX
62. #前缀后缀========================================
63. BUILDSUF=
64. PROGSSUF=
65. #${NAME}位于每个liavXXX/Makefile中，例如avformat
66. FULLNAME=$(NAME)$(BUILDSUF)
67. LIBPREF=lib
68. LIBSUF=.a
69. #例如libavformat.a
70. LIBNAME=$(LIBPREF)$(FULLNAME)$(LIBSUF)
71. SLIBPREF=
72. SLIBSUF=.dll
73. EXESUF=.exe
74. EXTRA\_VERSION=
75. DEPFLAGS=$(CPPFLAGS) $(CFLAGS) -MM
76. CCDEP=
77. CXXDEP=$(DEPCC) $(DEPFLAGS) $< | sed -e "/^\#.\*/d" -e "s,^[[:space:]]\*$(\*F)\\.o,$(@D)/$(\*F).o," > $(@:.o=.d)
78. ASDEP=
79. CC\_DEPFLAGS=-MMD -MF $(@:.o=.d) -MT $@
80. AS\_DEPFLAGS=-MMD -MF $(@:.o=.d) -MT $@
81. HOSTCC=gcc
82. HOSTCFLAGS=-D\_ISOC99\_SOURCE -O3 -g -std=c99 -Wall
83. HOSTEXESUF=.exe
84. HOSTLDFLAGS=
85. HOSTLIBS=-lm
86. TARGET\_EXEC=
87. TARGET\_PATH=$(CURDIR)
88. #SDL
89. SDL\_LIBS=-L/lib -lmingw32 -lSDLmain -lSDL -mwindows
90. SDL\_CFLAGS=-I/include/SDL -D\_GNU\_SOURCE=1 -Dmain=SDL\_main
91. LIB\_INSTALL\_EXTRA\_CMD=$$(RANLIB) "$(LIBDIR)/$(LIBNAME)"
92. #链接
93. EXTRALIBS=-lavicap32 -lws2\_32 -L/lib -lmingw32 -lSDLmain -lSDL -mwindows -lm -lz -lpsapi
94. INSTALL=install
95. LIBTARGET=i386
96. #例如libavformat.dll
97. SLIBNAME=$(SLIBPREF)$(FULLNAME)$(SLIBSUF)
98. #LIBVERSION变量位于library.mak
99. #例如libavformat-53.dll
100. #生成的Dll似乎就是这个版本的
101. SLIBNAME\_WITH\_VERSION=$(SLIBPREF)$(FULLNAME)-$(LIBVERSION)$(SLIBSUF)
102. #例如libavformat-53.31.100.dll
103. SLIBNAME\_WITH\_MAJOR=$(SLIBPREF)$(FULLNAME)-$(LIBMAJOR)$(SLIBSUF)
104. SLIB\_CREATE\_DEF\_CMD=
105. #生成导出库lib，会调用lib.exe
106. SLIB\_EXTRA\_CMD=-lib.exe /machine:$(LIBTARGET) /**def**:$$(@:$(SLIBSUF)=.**def**) /out:$(SUBDIR)$(SLIBNAME:$(SLIBSUF)=.lib)
107. SLIB\_INSTALL\_NAME=$(SLIBNAME\_WITH\_MAJOR)
108. SLIB\_INSTALL\_LINKS=
109. SLIB\_INSTALL\_EXTRA\_LIB=lib$(SLIBNAME:$(SLIBSUF)=.dll.a) $(SLIBNAME\_WITH\_MAJOR:$(SLIBSUF)=.**def**)
110. SLIB\_INSTALL\_EXTRA\_SHLIB=$(SLIBNAME:$(SLIBSUF)=.lib)
111. SAMPLES:=$(FATE\_SAMPLES)
112. NOREDZONE\_FLAGS=-mno-red-zone
113. #版本信息========================================
114. libavcodec\_VERSION=53.60.100
115. libavcodec\_VERSION\_MAJOR=53
116. libavdevice\_VERSION=53.4.100
117. libavdevice\_VERSION\_MAJOR=53
118. libavfilter\_VERSION=2.60.100
119. libavfilter\_VERSION\_MAJOR=2
120. libavformat\_VERSION=53.31.100
121. libavformat\_VERSION\_MAJOR=53
122. libavutil\_VERSION=51.34.101
123. libavutil\_VERSION\_MAJOR=51
124. libpostproc\_VERSION=52.0.100
125. libpostproc\_VERSION\_MAJOR=52
126. libswresample\_VERSION=0.6.100
127. libswresample\_VERSION\_MAJOR=0
128. libswscale\_VERSION=2.1.100
129. libswscale\_VERSION\_MAJOR=2
130. #组件配置========================================
131. #ARCH\_
132. !ARCH\_ALPHA=yes
133. !ARCH\_ARM=yes
134. !ARCH\_AVR32=yes
135. !ARCH\_AVR32\_AP=yes
136. !ARCH\_AVR32\_UC=yes
137. !ARCH\_BFIN=yes
138. !ARCH\_IA64=yes
139. !ARCH\_M68K=yes
140. !ARCH\_MIPS=yes
141. !ARCH\_MIPS64=yes
142. !ARCH\_PARISC=yes
143. !ARCH\_PPC=yes
144. !ARCH\_PPC64=yes
145. !ARCH\_S390=yes
146. !ARCH\_SH4=yes
147. !ARCH\_SPARC=yes
148. !ARCH\_SPARC64=yes
149. !ARCH\_TOMI=yes
150. ARCH\_X86=yes
151. ARCH\_X86\_32=yes
152. !ARCH\_X86\_64=yes
153. #HAVE\_
154. !HAVE\_ALTIVEC=yes
155. HAVE\_AMD3DNOW=yes
156. HAVE\_AMD3DNOWEXT=yes
157. !HAVE\_ARMV5TE=yes
158. !HAVE\_ARMV6=yes
159. !HAVE\_ARMV6T2=yes
160. !HAVE\_ARMVFP=yes
161. HAVE\_AVX=yes
162. !HAVE\_IWMMXT=yes
163. !HAVE\_MMI=yes
164. HAVE\_MMX=yes
165. HAVE\_MMX2=yes
166. //略……
167. HAVE\_YASM=yes
168. #CONFIG\_
169. CONFIG\_BSFS=yes
170. CONFIG\_DECODERS=yes
171. CONFIG\_DEMUXERS=yes
172. CONFIG\_ENCODERS=yes
173. CONFIG\_FILTERS=yes
174. !CONFIG\_HWACCELS=yes
175. CONFIG\_INDEVS=yes
176. CONFIG\_MUXERS=yes
177. CONFIG\_OUTDEVS=yes
178. CONFIG\_PARSERS=yes
179. CONFIG\_PROTOCOLS=yes
180. CONFIG\_FFPLAY=yes
181. CONFIG\_FFPROBE=yes
182. !CONFIG\_FFSERVER=yes
183. CONFIG\_FFMPEG=yes
184. !CONFIG\_AVPLAY=yes
185. !CONFIG\_AVPROBE=yes
186. !CONFIG\_AVSERVER=yes
187. CONFIG\_AANDCT=yes
188. CONFIG\_AC3DSP=yes
189. CONFIG\_AVCODEC=yes
190. CONFIG\_AVDEVICE=yes
191. CONFIG\_AVFILTER=yes
192. CONFIG\_AVFORMAT=yes
193. !CONFIG\_AVISYNTH=yes
194. !CONFIG\_BZLIB=yes
195. !CONFIG\_CRYSTALHD=yes
196. CONFIG\_DCT=yes
197. !CONFIG\_DOC=yes
198. CONFIG\_DWT=yes
199. !CONFIG\_DXVA2=yes
200. CONFIG\_FASTDIV=yes
201. CONFIG\_FFT=yes
202. !CONFIG\_FREI0R=yes
203. !CONFIG\_GNUTLS=yes
204. CONFIG\_GOLOMB=yes
205. !CONFIG\_GPL=yes
206. !CONFIG\_GRAY=yes
207. CONFIG\_H264CHROMA=yes
208. CONFIG\_H264DSP=yes
209. CONFIG\_H264PRED=yes
210. !CONFIG\_HARDCODED\_TABLES=yes
211. CONFIG\_HUFFMAN=yes
212. !CONFIG\_LIBAACPLUS=yes
213. !CONFIG\_LIBASS=yes
214. !CONFIG\_LIBCDIO=yes
215. !CONFIG\_LIBCELT=yes
216. !CONFIG\_LIBDC1394=yes
217. !CONFIG\_LIBDIRAC=yes
218. !CONFIG\_LIBFAAC=yes
219. !CONFIG\_LIBFREETYPE=yes
220. !CONFIG\_LIBGSM=yes
221. !CONFIG\_LIBMODPLUG=yes
222. !CONFIG\_LIBMP3LAME=yes
223. !CONFIG\_LIBNUT=yes
224. !CONFIG\_LIBOPENCORE\_AMRNB=yes
225. !CONFIG\_LIBOPENCORE\_AMRWB=yes
226. !CONFIG\_LIBOPENCV=yes
227. !CONFIG\_LIBOPENJPEG=yes
228. !CONFIG\_LIBPULSE=yes
229. !CONFIG\_LIBRTMP=yes
230. !CONFIG\_LIBSCHROEDINGER=yes
231. !CONFIG\_LIBSPEEX=yes
232. !CONFIG\_LIBSTAGEFRIGHT\_H264=yes
233. !CONFIG\_LIBTHEORA=yes
234. !CONFIG\_LIBUTVIDEO=yes
235. !CONFIG\_LIBV4L2=yes
236. !CONFIG\_LIBVO\_AACENC=yes
237. !CONFIG\_LIBVO\_AMRWBENC=yes
238. !CONFIG\_LIBVORBIS=yes
239. !CONFIG\_LIBVPX=yes
240. !CONFIG\_LIBX264=yes
241. !CONFIG\_LIBXAVS=yes
242. !CONFIG\_LIBXVID=yes
243. #此处省略将近1000条…
244. CONFIG\_RTMP\_PROTOCOL=yes
245. CONFIG\_RTMPT\_PROTOCOL=yes
246. CONFIG\_RTMPE\_PROTOCOL=yes
247. CONFIG\_RTMPTE\_PROTOCOL=yes
248. CONFIG\_RTMPS\_PROTOCOL=yes
249. CONFIG\_RTP\_PROTOCOL=yes
250. CONFIG\_TCP\_PROTOCOL=yes
251. !CONFIG\_TLS\_PROTOCOL=yes
252. CONFIG\_UDP\_PROTOCOL=yes
253. #Test
254. ACODEC\_TESTS=ac3\_fixed adpcm\_adx adpcm\_ima\_qt adpcm\_ima\_wav adpcm\_ms adpcm\_swf adpcm\_yam alac aref flac g722 g723\_1 g726 mp2 pcm\_alaw pcm\_f32be pcm\_f32le pcm\_f64be pcm\_f64le pcm\_mulaw pcm\_s16be pcm\_s16le pcm\_s24be pcm\_s24daud pcm\_s24le pcm\_s32be pcm\_s32le pcm\_s8 pcm\_u8 wmav1 wmav2
255. VCODEC\_TESTS=amv asv1 asv2 cljr dnxhd\_1080i dnxhd\_720p dnxhd\_720p\_10bit dnxhd\_720p\_rd dv dv50 dv\_411 error ffv1 flashsv flashsv2 flv h261 h263 h263p huffyuv jpeg2000 jpegls ljpeg mjpeg mpeg mpeg1b mpeg2 mpeg2\_422 mpeg2\_idct\_int mpeg2\_ilace mpeg2\_ivlc\_qprd mpeg2thread mpeg2thread\_ilace mpeg4 mpeg4\_adap mpeg4\_qpel mpeg4\_qprd mpeg4adv mpeg4nr mpeg4thread mpng msmpeg4 msmpeg4v2 msvideo1 prores qtrle qtrlegray rc rgb roq rv10 rv20 snow snowll svq1 v210 vref wmv1 wmv2 yuv zlib zmbv
256. LAVF\_TESTS=aiff alaw asf au avi bmp caf dpx dv\_fmt ffm flv\_fmt gif gxf jpg mkv mmf mov mpg mulaw mxf mxf\_d10 nut ogg pbmpipe pcx pgm pgmpipe pixfmt png ppm ppmpipe rm rso sgi sox swf tga tiff ts voc voc\_s16 wav wtv yuv4mpeg
257. LAVFI\_TESTS=crop crop\_scale crop\_scale\_vflip crop\_vflip null pixdesc pixfmts\_copy pixfmts\_crop pixfmts\_hflip pixfmts\_null pixfmts\_pad pixfmts\_scale pixfmts\_vflip scale200 scale500 vflip vflip\_crop vflip\_vflip
258. SEEK\_TESTS=seek\_ac3\_rm seek\_adpcm\_ima\_wav seek\_adpcm\_ms\_wav seek\_adpcm\_qt\_aiff seek\_adpcm\_swf\_flv seek\_adpcm\_yam\_wav seek\_alac\_m4a seek\_asv1\_avi seek\_asv2\_avi seek\_dnxhd\_1080i\_mov seek\_dnxhd\_720p\_dnxhd seek\_dnxhd\_720p\_rd\_dnxhd seek\_dv411\_dv seek\_dv50\_dv seek\_dv\_dv seek\_error\_mpeg4\_adv\_avi seek\_ffv1\_avi seek\_flac\_flac seek\_flashsv\_flv seek\_flv\_flv seek\_g726\_wav seek\_h261\_avi seek\_h263\_avi seek\_h263p\_avi seek\_huffyuv\_avi seek\_image\_bmp seek\_image\_jpg seek\_image\_pcx seek\_image\_pgm seek\_image\_ppm seek\_image\_sgi seek\_image\_tga seek\_image\_tiff seek\_jpegls\_avi seek\_lavf\_aif seek\_lavf\_al seek\_lavf\_asf seek\_lavf\_au seek\_lavf\_avi seek\_lavf\_dv seek\_lavf\_ffm seek\_lavf\_flv seek\_lavf\_gif seek\_lavf\_gxf seek\_lavf\_mkv seek\_lavf\_mmf seek\_lavf\_mov seek\_lavf\_mpg seek\_lavf\_mxf seek\_lavf\_mxf\_d10 seek\_lavf\_nut seek\_lavf\_ogg seek\_lavf\_rm seek\_lavf\_swf seek\_lavf\_ts seek\_lavf\_ul seek\_lavf\_voc seek\_lavf\_wav seek\_lavf\_wtv seek\_lavf\_y4m seek\_ljpeg\_avi seek\_mjpeg\_avi seek\_mp2\_mp2 seek\_mpeg1\_mpg seek\_mpeg1b\_mpg seek\_mpeg2\_422\_mpg seek\_mpeg2\_idct\_int\_mpg seek\_mpeg2i\_mpg seek\_mpeg2ivlc\_qprd\_mpg seek\_mpeg2reuse\_mpg seek\_mpeg2thread\_mpg seek\_mpeg2threadivlc\_mpg seek\_mpeg4\_adap\_avi seek\_mpeg4\_adv\_avi seek\_mpeg4\_nr\_avi seek\_mpeg4\_qprd\_avi seek\_mpeg4\_rc\_avi seek\_mpeg4\_thread\_avi seek\_msmpeg4\_avi seek\_msmpeg4v2\_avi seek\_odivx\_mp4 seek\_pbmpipe\_pbm seek\_pcm\_alaw\_wav seek\_pcm\_f32be\_au seek\_pcm\_f32le\_wav seek\_pcm\_f64be\_au seek\_pcm\_f64le\_wav seek\_pcm\_mulaw\_wav seek\_pcm\_s16be\_mov seek\_pcm\_s16le\_wav seek\_pcm\_s24be\_mov seek\_pcm\_s24daud\_302 seek\_pcm\_s24le\_wav seek\_pcm\_s32be\_mov seek\_pcm\_s32le\_wav seek\_pcm\_s8\_mov seek\_pcm\_u8\_wav seek\_pgmpipe\_pgm seek\_ppmpipe\_ppm seek\_rgb\_avi seek\_roqav\_roq seek\_rv10\_rm seek\_rv20\_rm seek\_snow53\_avi seek\_snow\_avi seek\_svq1\_mov seek\_wmav1\_asf seek\_wmav2\_asf seek\_wmv1\_avi seek\_wmv2\_avi seek\_yuv\_avi
259. endif # FFMPEG\_CONFIG\_MAK

### config.h源代码

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44587465) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44587465)

1. /\*\*
2. \* FFmpeg config.h
3. \*
4. \* 注释：雷霄骅
5. \* leixiaohua1020@126.com
6. \* http://blog.csdn.net/leixiaohua1020
7. \*
8. \* Configure脚本生成的config.h，包含了各种配置信息。
9. /
10. /\* Automatically generated by configure - do not modify! \*/
11. #ifndef FFMPEG\_CONFIG\_H
12. #define FFMPEG\_CONFIG\_H
13. #define FFMPEG\_CONFIGURATION ""
14. #define FFMPEG\_LICENSE "LGPL version 2.1 or later"
15. #define FFMPEG\_DATADIR "/usr/local/share/ffmpeg"
16. #define AVCONV\_DATADIR "/usr/local/share/ffmpeg"
17. #define CC\_TYPE "gcc"
18. #define CC\_VERSION \_\_VERSION\_\_
19. #define restrict restrict
20. #define EXTERN\_PREFIX "\_"
21. #define EXTERN\_ASM \_
22. #define SLIBSUF ".dll"
23. #define ARCH\_ALPHA 0
24. #define ARCH\_ARM 0
25. #define ARCH\_AVR32 0
26. #define ARCH\_AVR32\_AP 0
27. #define ARCH\_AVR32\_UC 0
28. #define ARCH\_BFIN 0
29. #define ARCH\_IA64 0
30. #define ARCH\_M68K 0
31. #define ARCH\_MIPS 0
32. #define ARCH\_MIPS64 0
33. #define ARCH\_PARISC 0
34. #define ARCH\_PPC 0
35. #define ARCH\_PPC64 0
36. #define ARCH\_S390 0
37. #define ARCH\_SH4 0
38. #define ARCH\_SPARC 0
39. #define ARCH\_SPARC64 0
40. #define ARCH\_TOMI 0
41. #define ARCH\_X86 1
42. #define ARCH\_X86\_32 1
43. #define ARCH\_X86\_64 0
44. #define HAVE\_ALTIVEC 0
45. #define HAVE\_AMD3DNOW 1
46. #define HAVE\_AMD3DNOWEXT 1
47. #define HAVE\_ARMV5TE 0
48. #define HAVE\_ARMV6 0
49. #define HAVE\_ARMV6T2 0
50. #define HAVE\_ARMVFP 0
51. #define HAVE\_AVX 1
52. #define HAVE\_IWMMXT 0
53. #define HAVE\_MMI 0
54. #define HAVE\_MMX 1
55. #define HAVE\_MMX2 1
56. //略……
57. #define HAVE\_YASM 1
58. #define CONFIG\_BSFS 1
59. #define CONFIG\_DECODERS 1
60. #define CONFIG\_DEMUXERS 1
61. #define CONFIG\_ENCODERS 1
62. #define CONFIG\_FILTERS 1
63. #define CONFIG\_HWACCELS 0
64. #define CONFIG\_INDEVS 1
65. #define CONFIG\_MUXERS 1
66. #define CONFIG\_OUTDEVS 1
67. #define CONFIG\_PARSERS 1
68. #define CONFIG\_PROTOCOLS 1
69. #define CONFIG\_FFPLAY 1
70. #define CONFIG\_FFPROBE 1
71. #define CONFIG\_FFSERVER 0
72. #define CONFIG\_FFMPEG 1
73. #define CONFIG\_AVPLAY 0
74. #define CONFIG\_AVPROBE 0
75. #define CONFIG\_AVSERVER 0
76. #define CONFIG\_AANDCT 1
77. #define CONFIG\_AC3DSP 1
78. #define CONFIG\_AVCODEC 1
79. #define CONFIG\_AVDEVICE 1
80. #define CONFIG\_AVFILTER 1
81. #define CONFIG\_AVFORMAT 1
82. #define CONFIG\_AVISYNTH 0
83. #define CONFIG\_BZLIB 0
84. #define CONFIG\_CRYSTALHD 0
85. #define CONFIG\_DCT 1
86. #define CONFIG\_DOC 0
87. #define CONFIG\_DWT 1
88. #define CONFIG\_DXVA2 0
89. #define CONFIG\_FASTDIV 1
90. #define CONFIG\_FFT 1
91. #define CONFIG\_FREI0R 0
92. #define CONFIG\_GNUTLS 0
93. #define CONFIG\_GOLOMB 1
94. #define CONFIG\_GPL 0
95. #define CONFIG\_GRAY 0
96. #define CONFIG\_H264CHROMA 1
97. #define CONFIG\_H264DSP 1
98. #define CONFIG\_H264PRED 1
99. #define CONFIG\_HARDCODED\_TABLES 0
100. #define CONFIG\_HUFFMAN 1
101. #define CONFIG\_LIBAACPLUS 0
102. #define CONFIG\_LIBASS 0
103. #define CONFIG\_LIBCDIO 0
104. #define CONFIG\_LIBCELT 0
105. #define CONFIG\_LIBDC1394 0
106. #define CONFIG\_LIBDIRAC 0
107. #define CONFIG\_LIBFAAC 0
108. #define CONFIG\_LIBFREETYPE 0
109. #define CONFIG\_LIBGSM 0
110. #define CONFIG\_LIBMODPLUG 0
111. #define CONFIG\_LIBMP3LAME 0
112. #define CONFIG\_LIBNUT 0
113. #define CONFIG\_LIBOPENCORE\_AMRNB 0
114. #define CONFIG\_LIBOPENCORE\_AMRWB 0
115. #define CONFIG\_LIBOPENCV 0
116. #define CONFIG\_LIBOPENJPEG 0
117. #define CONFIG\_LIBPULSE 0
118. #define CONFIG\_LIBRTMP 0
119. #define CONFIG\_LIBSCHROEDINGER 0
120. #define CONFIG\_LIBSPEEX 0
121. #define CONFIG\_LIBSTAGEFRIGHT\_H264 0
122. #define CONFIG\_LIBTHEORA 0
123. #define CONFIG\_LIBUTVIDEO 0
124. #define CONFIG\_LIBV4L2 0
125. #define CONFIG\_LIBVO\_AACENC 0
126. #define CONFIG\_LIBVO\_AMRWBENC 0
127. #define CONFIG\_LIBVORBIS 0
128. #define CONFIG\_LIBVPX 0
129. #define CONFIG\_LIBX264 0
130. //此处省略将近1000条
131. #define CONFIG\_RTMP\_PROTOCOL 1
132. #define CONFIG\_RTMPT\_PROTOCOL 1
133. #define CONFIG\_RTMPE\_PROTOCOL 1
134. #define CONFIG\_RTMPTE\_PROTOCOL 1
135. #define CONFIG\_RTMPS\_PROTOCOL 1
136. #define CONFIG\_RTP\_PROTOCOL 1
137. #define CONFIG\_TCP\_PROTOCOL 1
138. #define CONFIG\_TLS\_PROTOCOL 0
139. #define CONFIG\_UDP\_PROTOCOL 1
140. #endif /\* FFMPEG\_CONFIG\_H \*/

### Configure的源代码

**[plain]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/44587465) [copy](http://blog.csdn.net/leixiaohua1020/article/details/44587465)

1. #!/bin/sh
2. #
3. # FFmpeg configure script
4. #
5. # Copyright (c) 2000-2002 Fabrice Bellard
6. # Copyright (c) 2005-2008 Diego Biurrun
7. # Copyright (c) 2005-2008 Mans Rullgard
8. #
9. # 注释：雷霄骅
10. # leixiaohua1020@126.com
11. # http://blog.csdn.net/leixiaohua1020
12. #
13. # 添加了注释的FFmpeg的Configure文件
14. #
15. # Prevent locale nonsense from breaking basic text processing.
16. LC\_ALL=C
17. export LC\_ALL
19. # make sure we are running under a compatible shell
20. # try to make this part work with most shells
22. try\_exec(){
23. echo "Trying shell $1"
24. type "$1" > /dev/null 2>&1 && exec "$@"
25. }
27. unset foo
28. (: ${foo%%bar}) 2> /dev/null
29. E1="$?"
31. (: ${foo?}) 2> /dev/null
32. E2="$?"
34. if test "$E1" != 0 || test "$E2" = 0; then
35. echo "Broken shell detected.  Trying alternatives."
36. export FF\_CONF\_EXEC
37. if test "0$FF\_CONF\_EXEC" -lt 1; then
38. FF\_CONF\_EXEC=1
39. try\_exec bash "$0" "$@"
40. fi
41. if test "0$FF\_CONF\_EXEC" -lt 2; then
42. FF\_CONF\_EXEC=2
43. try\_exec ksh "$0" "$@"
44. fi
45. if test "0$FF\_CONF\_EXEC" -lt 3; then
46. FF\_CONF\_EXEC=3
47. try\_exec /usr/xpg4/bin/sh "$0" "$@"
48. fi
49. echo "No compatible shell script interpreter found."
50. echo "This configure script requires a POSIX-compatible shell"
51. echo "such as bash or ksh."
52. echo "THIS IS NOT A BUG IN FFMPEG, DO NOT REPORT IT AS SUCH."
53. echo "Instead, install a working POSIX-compatible shell."
54. echo "Disabling this configure test will create a broken FFmpeg."
55. if test "$BASH\_VERSION" = '2.04.0(1)-release'; then
56. echo "This bash version ($BASH\_VERSION) is broken on your platform."
57. echo "Upgrade to a later version if available."
58. fi
59. exit 1
60. fi
61. #帮助菜单
62. show\_help(){
63. cat <<EOF
64. Usage: configure [options]
65. Options: [defaults in brackets after descriptions]
67. Standard options:
68. --help                   print this message
69. --logfile=FILE           log tests and output to FILE [config.log]
70. --disable-logging        do not log configure debug information
71. --prefix=PREFIX          install in PREFIX [$prefix]
72. --bindir=DIR             install binaries in DIR [PREFIX/bin]
73. --datadir=DIR            install data files in DIR [PREFIX/share/ffmpeg]
74. --libdir=DIR             install libs in DIR [PREFIX/lib]
75. --shlibdir=DIR           install shared libs in DIR [PREFIX/lib]
76. --incdir=DIR             install includes in DIR [PREFIX/include]
77. --mandir=DIR             install man page in DIR [PREFIX/share/man]
79. Configuration options:
80. --disable-static         do not build static libraries [no]
81. --enable-shared          build shared libraries [no]
82. --enable-gpl             allow use of GPL code, the resulting libs
83. and binaries will be under GPL [no]
84. --enable-version3        upgrade (L)GPL to version 3 [no]
85. --enable-nonfree         allow use of nonfree code, the resulting libs
86. and binaries will be unredistributable [no]
87. --disable-doc            do not build documentation
88. --disable-ffmpeg         disable ffmpeg build
89. --disable-ffplay         disable ffplay build
90. --disable-ffprobe        disable ffprobe build
91. --disable-ffserver       disable ffserver build
92. --disable-avdevice       disable libavdevice build
93. --disable-avcodec        disable libavcodec build
94. --disable-avformat       disable libavformat build
95. --disable-swresample     disable libswresample build
96. --disable-swscale        disable libswscale build
97. --disable-postproc       disable libpostproc build
98. --disable-avfilter       disable video filter support [no]
99. --disable-pthreads       disable pthreads [auto]
100. --disable-w32threads     disable Win32 threads [auto]
101. --disable-os2threads     disable OS/2 threads [auto]
102. --enable-x11grab         enable X11 grabbing [no]
103. --disable-network        disable network support [no]
104. --enable-gray            enable full grayscale support (slower color)
105. --disable-swscale-alpha  disable alpha channel support in swscale
106. --disable-fastdiv        disable table-based division
107. --enable-small           optimize for size instead of speed
108. --disable-aandct         disable AAN DCT code
109. --disable-dct            disable DCT code
110. --disable-fft            disable FFT code
111. --disable-golomb         disable Golomb code
112. --disable-huffman        disable Huffman code
113. --disable-lpc            disable LPC code
114. --disable-mdct           disable MDCT code
115. --disable-rdft           disable RDFT code
116. --enable-vaapi           enable VAAPI code [autodetect]
117. --enable-vda             enable VDA code [autodetect]
118. --enable-vdpau           enable VDPAU code [autodetect]
119. --disable-dxva2          disable DXVA2 code
120. --disable-vda            disable VDA code
121. --enable-runtime-cpudetect detect cpu capabilities at runtime (bigger binary)
122. --enable-hardcoded-tables use hardcoded tables instead of runtime generation
123. --disable-safe-bitstream-reader
124. disable buffer boundary checking in bitreaders
125. (faster, but may crash)
126. --enable-memalign-hack   emulate memalign, interferes with memory debuggers
127. --disable-everything     disable all components listed below
128. --disable-encoder=NAME   disable encoder NAME
129. --enable-encoder=NAME    enable encoder NAME
130. --disable-encoders       disable all encoders
131. --disable-decoder=NAME   disable decoder NAME
132. --enable-decoder=NAME    enable decoder NAME
133. --disable-decoders       disable all decoders
134. --disable-hwaccel=NAME   disable hwaccel NAME
135. --enable-hwaccel=NAME    enable hwaccel NAME
136. --disable-hwaccels       disable all hwaccels
137. --disable-muxer=NAME     disable muxer NAME
138. --enable-muxer=NAME      enable muxer NAME
139. --disable-muxers         disable all muxers
140. --disable-demuxer=NAME   disable demuxer NAME
141. --enable-demuxer=NAME    enable demuxer NAME
142. --disable-demuxers       disable all demuxers
143. --enable-parser=NAME     enable parser NAME
144. --disable-parser=NAME    disable parser NAME
145. --disable-parsers        disable all parsers
146. --enable-bsf=NAME        enable bitstream filter NAME
147. --disable-bsf=NAME       disable bitstream filter NAME
148. --disable-bsfs           disable all bitstream filters
149. --enable-protocol=NAME   enable protocol NAME
150. --disable-protocol=NAME  disable protocol NAME
151. --disable-protocols      disable all protocols
152. --disable-indev=NAME     disable input device NAME
153. --disable-outdev=NAME    disable output device NAME
154. --disable-indevs         disable input devices
155. --disable-outdevs        disable output devices
156. --disable-devices        disable all devices
157. --enable-filter=NAME     enable filter NAME
158. --disable-filter=NAME    disable filter NAME
159. --disable-filters        disable all filters
160. --list-decoders          show all available decoders
161. --list-encoders          show all available encoders
162. --list-hwaccels          show all available hardware accelerators
163. --list-muxers            show all available muxers
164. --list-demuxers          show all available demuxers
165. --list-parsers           show all available parsers
166. --list-protocols         show all available protocols
167. --list-bsfs              show all available bitstream filters
168. --list-indevs            show all available input devices
169. --list-outdevs           show all available output devices
170. --list-filters           show all available filters
172. External library support:
173. --enable-avisynth        enable reading of AVISynth script files [no]
174. --enable-bzlib           enable bzlib [autodetect]
175. --enable-frei0r          enable frei0r video filtering
176. --enable-gnutls          enable gnutls [no]
177. --enable-libaacplus      enable AAC+ encoding via libaacplus [no]
178. --enable-libass          enable libass subtitles rendering [no]
179. --enable-libcelt         enable CELT decoding via libcelt [no]
180. --enable-libopencore-amrnb enable AMR-NB de/encoding via libopencore-amrnb [no]
181. --enable-libopencore-amrwb enable AMR-WB decoding via libopencore-amrwb [no]
182. --enable-libopencv       enable video filtering via libopencv [no]
183. --enable-libcdio         enable audio CD grabbing with libcdio
184. --enable-libdc1394       enable IIDC-1394 grabbing using libdc1394
185. and libraw1394 [no]
186. --enable-libdirac        enable Dirac support via libdirac [no]
187. --enable-libfaac         enable FAAC support via libfaac [no]
188. --enable-libfreetype     enable libfreetype [no]
189. --enable-libgsm          enable GSM support via libgsm [no]
190. --enable-libmodplug      enable ModPlug via libmodplug [no]
191. --enable-libmp3lame      enable MP3 encoding via libmp3lame [no]
192. --enable-libnut          enable NUT (de)muxing via libnut,
193. native (de)muxer exists [no]
194. --enable-libopenjpeg     enable JPEG 2000 encoding/decoding via OpenJPEG [no]
195. --enable-libpulse        enable Pulseaudio input via libpulse [no]
196. --enable-librtmp         enable RTMP[E] support via librtmp [no]
197. --enable-libschroedinger enable Dirac support via libschroedinger [no]
198. --enable-libspeex        enable Speex support via libspeex [no]
199. --enable-libstagefright-h264  enable H.264 decoding via libstagefright [no]
200. --enable-libtheora       enable Theora encoding via libtheora [no]
201. --enable-libutvideo      enable Ut Video decoding via libutvideo [no]
202. --enable-libv4l2         enable libv4l2/v4l-utils [no]
203. --enable-libvo-aacenc    enable AAC encoding via libvo-aacenc [no]
204. --enable-libvo-amrwbenc  enable AMR-WB encoding via libvo-amrwbenc [no]
205. --enable-libvorbis       enable Vorbis encoding via libvorbis,
206. native implementation exists [no]
207. --enable-libvpx          enable VP8 support via libvpx [no]
208. --enable-libx264         enable H.264 encoding via x264 [no]
209. --enable-libxavs         enable AVS encoding via xavs [no]
210. --enable-libxvid         enable Xvid encoding via xvidcore,
211. native MPEG-4/Xvid encoder exists [no]
212. --enable-openal          enable OpenAL 1.1 capture support [no]
213. --enable-mlib            enable Sun medialib [no]
214. --enable-openssl         enable openssl [no]
215. --enable-zlib            enable zlib [autodetect]
217. Advanced options (experts only):
218. --cross-prefix=PREFIX    use PREFIX for compilation tools [$cross\_prefix]
219. --enable-cross-compile   assume a cross-compiler is used
220. --sysroot=PATH           root of cross-build tree
221. --sysinclude=PATH        location of cross-build system headers
222. --target-os=OS           compiler targets OS [$target\_os]
223. --target-exec=CMD        command to run executables on target
224. --target-path=DIR        path to view of build directory on target
225. --nm=NM                  use nm tool NM [$nm\_default]
226. --ar=AR                  use archive tool AR [$ar\_default]
227. --as=AS                  use assembler AS [$as\_default]
228. --yasmexe=EXE            use yasm-compatible assembler EXE [$yasmexe\_default]
229. --cc=CC                  use C compiler CC [$cc\_default]
230. --cxx=CXX                use C compiler CXX [$cxx\_default]
231. --ld=LD                  use linker LD [$ld\_default]
232. --host-cc=HOSTCC         use host C compiler HOSTCC
233. --host-cflags=HCFLAGS    use HCFLAGS when compiling for host
234. --host-ldflags=HLDFLAGS  use HLDFLAGS when linking for host
235. --host-libs=HLIBS        use libs HLIBS when linking for host
236. --extra-cflags=ECFLAGS   add ECFLAGS to CFLAGS [$CFLAGS]
237. --extra-cxxflags=ECFLAGS add ECFLAGS to CXXFLAGS [$CXXFLAGS]
238. --extra-ldflags=ELDFLAGS add ELDFLAGS to LDFLAGS [$LDFLAGS]
239. --extra-libs=ELIBS       add ELIBS [$ELIBS]
240. --extra-version=STRING   version string suffix []
241. --build-suffix=SUFFIX    library name suffix []
242. --progs-suffix=SUFFIX    program name suffix []
243. --arch=ARCH              select architecture [$arch]
244. --cpu=CPU                select the minimum required CPU (affects
245. instruction selection, may crash on older CPUs)
246. --disable-asm            disable all assembler optimizations
247. --disable-altivec        disable AltiVec optimizations
248. --disable-amd3dnow       disable 3DNow! optimizations
249. --disable-amd3dnowext    disable 3DNow! extended optimizations
250. --disable-mmx            disable MMX optimizations
251. --disable-mmx2           disable MMX2 optimizations
252. --disable-sse            disable SSE optimizations
253. --disable-ssse3          disable SSSE3 optimizations
254. --disable-avx            disable AVX optimizations
255. --disable-armv5te        disable armv5te optimizations
256. --disable-armv6          disable armv6 optimizations
257. --disable-armv6t2        disable armv6t2 optimizations
258. --disable-armvfp         disable ARM VFP optimizations
259. --disable-iwmmxt         disable iwmmxt optimizations
260. --disable-mmi            disable MMI optimizations
261. --disable-neon           disable NEON optimizations
262. --disable-vis            disable VIS optimizations
263. --disable-yasm           disable use of yasm assembler
264. --enable-pic             build position-independent code
265. --malloc-prefix=PFX      prefix malloc and related names with PFX
266. --enable-sram            allow use of on-chip SRAM
267. --disable-symver         disable symbol versioning
268. --optflags               override optimization-related compiler flags
269. --postproc-version=V     build libpostproc version V.
270. Where V can be '$ALT\_PP\_VER\_MAJOR.$ALT\_PP\_VER\_MINOR.$ALT\_PP\_VER\_MICRO' or 'current'. [$postproc\_version\_default]
272. Developer options (useful when working on FFmpeg itself):
273. --enable-coverage        build with test coverage instrumentation
274. --disable-debug          disable debugging symbols
275. --enable-debug=LEVEL     set the debug level [$debuglevel]
276. --disable-optimizations  disable compiler optimizations
277. --enable-extra-warnings  enable more compiler warnings
278. --disable-stripping      disable stripping of executables and shared libraries
279. --valgrind=VALGRIND      run "make fate" tests through valgrind to detect memory
280. leaks and errors, using the specified valgrind binary.
281. Cannot be combined with --target-exec
282. --samples=PATH           location of test samples for FATE, if not set use
283. \$FATE\_SAMPLES at make invocation time.
285. NOTE: Object files are built at the place where configure is launched.
286. EOF
287. exit 0
288. }
290. quotes='""'
292. #日志config.log
293. log(){
294. echo "$@" >> $logfile
295. }
297. log\_file(){
298. log BEGIN $1
299. pr -n -t $1 >> $logfile
300. log END $1
301. }
303. echolog(){
304. log "$@"
305. echo "$@"
306. }
308. warn(){
309. log "WARNING: $\*"
310. WARNINGS="${WARNINGS}WARNING: $\*\n"
311. }
313. #出错了
314. die(){
315. echolog "$@"
316. cat <<EOF
318. If you think configure made a mistake, make sure you are using the latest
319. version from Git.  If the latest version fails, report the problem to the
320. ffmpeg-user@ffmpeg.org mailing list or IRC #ffmpeg on irc.freenode.net.
321. EOF
322. if disabled logging; then
323. cat <<EOF
324. Rerun configure with logging enabled (do not use --disable-logging), and
325. include the log this produces with your report.
326. EOF
327. else
328. cat <<EOF
329. Include the log file "$logfile" produced by configure as this will help
330. solving the problem.
331. EOF
332. fi
333. exit 1
334. }
336. # Avoid locale weirdness, besides we really just want to translate ASCII.
337. toupper(){
338. echo "$@" | tr abcdefghijklmnopqrstuvwxyz ABCDEFGHIJKLMNOPQRSTUVWXYZ
339. }
341. tolower(){
342. echo "$@" | tr ABCDEFGHIJKLMNOPQRSTUVWXYZ abcdefghijklmnopqrstuvwxyz
343. }
345. c\_escape(){
346. echo "$\*" | sed 's/["\\]/\\\0/g'
347. }
349. sh\_quote(){
350. v=$(echo "$1" | sed "s/'/'\\\\''/g")
351. test "x$v" = "x${v#\*[!A-Za-z0-9\_/.+-]}" || v="'$v'"
352. echo "$v"
353. }
355. cleanws(){
356. echo "$@" | sed 's/^ \*//;s/  \*/ /g;s/ \*$//'
357. }
359. filter(){
360. pat=$1
361. shift
362. for v; do
363. eval "case $v in $pat) echo $v ;; esac"
364. done
365. }
367. filter\_out(){
368. pat=$1
369. shift
370. for v; do
371. eval "case $v in $pat) ;; \*) echo $v ;; esac"
372. done
373. }
375. map(){
376. m=$1
377. shift
378. for v; do eval $m; done
379. }
381. #第一个参数为值，后面的参数为变量
382. set\_all(){
383. value=$1
384. shift
385. for var in $\*; do
386. eval $var=$value
387. done
388. }
390. set\_weak(){
391. value=$1
392. shift
393. for var; do
394. eval : \${$var:=$value}
395. done
396. }
398. set\_safe(){
399. var=$1
400. shift
401. eval $(echo "$var" | sed 's/[^A-Za-z0-9\_]/\_/g')='$\*'
402. }
404. get\_safe(){
405. eval echo \$$(echo "$1" | sed 's/[^A-Za-z0-9\_]/\_/g')
406. }
408. pushvar(){
409. for var in $\*; do
410. eval level=\${${var}\_level:=0}
411. eval ${var}\_${level}="\$$var"
412. eval ${var}\_level=$(($level+1))
413. done
414. }
416. popvar(){
417. for var in $\*; do
418. eval level=\${${var}\_level:-0}
419. test $level = 0 && continue
420. eval level=$(($level-1))
421. eval $var="\${${var}\_${level}}"
422. eval ${var}\_level=$level
423. eval unset ${var}\_${level}
424. done
425. }
426. #把所有输入参数的值设置为"yes"
427. enable(){
428. set\_all yes $\*
429. }
430. #把所有输入参数的值设置为"no"
431. disable(){
432. set\_all no $\*
433. }
435. enable\_weak(){
436. set\_weak yes $\*
437. }
439. disable\_weak(){
440. set\_weak no $\*
441. }
443. enable\_safe(){
444. for var; do
445. enable $(echo "$var" | sed 's/[^A-Za-z0-9\_]/\_/g')
446. done
447. }
449. disable\_safe(){
450. for var; do
451. disable $(echo "$var" | sed 's/[^A-Za-z0-9\_]/\_/g')
452. done
453. }
455. do\_enable\_deep(){
456. for var; do
457. enabled $var && continue
458. eval sel="\$${var}\_select"
459. eval sgs="\$${var}\_suggest"
460. pushvar var sgs
461. enable\_deep $sel
462. popvar sgs
463. enable\_deep\_weak $sgs
464. popvar var
465. done
466. }
468. enable\_deep(){
469. do\_enable\_deep $\*
470. enable $\*
471. }
473. enable\_deep\_weak(){
474. do\_enable\_deep $\*
475. enable\_weak $\*
476. }
477. #是否开启该组件
478. #主要通过看该组件的取值为"yes"还是"no"
479. #示例 enabled yasm
480. enabled(){
481. # "#"代表删除从前往后最小匹配的内容
482. # 在这里，即删除"!"，去掉${1}的"!"?
483. # 这句话的意思可能是，检查去掉"!"之后，${1}是否相等？即检查${1}中是否包含"!"
484. # PS：一般很少有包含"!"的情况吧 = =!
485. # 如果包含"!"，op取值为"="，否则取值"!="
486. test "${1#!}" = "$1" && op== || op=!=
487. # 进行比较，看看取值是否为"yes"
488. # 为什么要在两边的前面都加一个"x"?
489. eval test "x\$${1#!}" $op "xyes"
490. }
491. #是否关闭该组件
492. #和enabled相对应
493. disabled(){
494. test "${1#!}" = "$1" && op== || op=!=
495. #看看取值是否为"no"
496. eval test "x\$${1#!}" $op "xno"
497. }
499. enabled\_all(){
500. for opt; do
501. enabled $opt || return 1
502. done
503. }
505. disabled\_all(){
506. for opt; do
507. disabled $opt || return 1
508. done
509. }
511. enabled\_any(){
512. for opt; do
513. enabled $opt && return 0
514. done
515. }
517. disabled\_any(){
518. for opt; do
519. disabled $opt && return 0
520. done
521. return 1
522. }
523. #设置默认值
524. set\_default(){
525. for opt; do
526. eval : \${$opt:=\$${opt}\_default}
527. done
528. }
530. is\_in(){
531. value=$1
532. shift
533. for var in $\*; do
534. [ $var = $value ] && return 0
535. done
536. return 1
537. }
539. check\_deps(){
540. for cfg; do
541. cfg="${cfg#!}"
542. enabled ${cfg}\_checking && die "Circular dependency for $cfg."
543. disabled ${cfg}\_checking && continue
544. enable ${cfg}\_checking
546. eval dep\_all="\$${cfg}\_deps"
547. eval dep\_any="\$${cfg}\_deps\_any"
548. eval dep\_sel="\$${cfg}\_select"
549. eval dep\_sgs="\$${cfg}\_suggest"
550. eval dep\_ifa="\$${cfg}\_if"
551. eval dep\_ifn="\$${cfg}\_if\_any"
553. pushvar cfg dep\_all dep\_any dep\_sel dep\_sgs dep\_ifa dep\_ifn
554. check\_deps $dep\_all $dep\_any $dep\_sel $dep\_sgs $dep\_ifa $dep\_ifn
555. popvar cfg dep\_all dep\_any dep\_sel dep\_sgs dep\_ifa dep\_ifn
557. [ -n "$dep\_ifa" ] && { enabled\_all $dep\_ifa && enable\_weak $cfg; }
558. [ -n "$dep\_ifn" ] && { enabled\_any $dep\_ifn && enable\_weak $cfg; }
559. enabled\_all  $dep\_all || disable $cfg
560. enabled\_any  $dep\_any || disable $cfg
561. disabled\_any $dep\_sel && disable $cfg
563. if enabled $cfg; then
564. eval dep\_extralibs="\$${cfg}\_extralibs"
565. test -n "$dep\_extralibs" && add\_extralibs $dep\_extralibs
566. enable\_deep $dep\_sel
567. enable\_deep\_weak $dep\_sgs
568. fi
570. disable ${cfg}\_checking
571. done
572. }
573. #输出config.h的时候使用
574. #调用示例：print\_config\_h ffplay CONFIG\_FFPLAY
575. print\_config\_h(){
576. #command1  && command2
577. #&&左边的命令（命令1）返回真(即返回0，成功被执行）后，&&右边的命令（命令2）才能够被执行
578. #command1 || command2
579. #||左边的命令（命令1）未执行成功，那么就执行||右边的命令（命令2）
580. enabled $1 && v=1 || v=0
581. #示例：#define CONFIG\_FFPLAY 1
582. echo "#define $2 $v"
583. }
584. #输出config.mak的时候使用
585. print\_config\_mak(){
586. enabled $1 && v= || v=!
587. echo "$v$2=yes"
588. }
589. # 输出config.asm的时候使用
590. print\_config\_asm(){
591. enabled $1 && echo "%define $2"
592. }
593. # 输出文本到config.mak，config.h等文件
594. # 该函数的示例调用方法：print\_config CONFIG\_ "$config\_files" $CONFIG\_LIST
595. print\_config(){
596. # 前缀
597. pfx=$1
598. # 文件列表
599. files=$2
600. # 位置参数可以用shift命令左移。比如shift 3表示原来的$4现在变成$1
601. shift 2
602. #for循环中，当没有in指定列表时，for会默认取命令行参数列表。
603. #在这里取的就是$CONFIG\_LIST 等
604. for cfg; do
605. # toupper()：转换为大写
606. ucname="$(toupper $cfg)"
607. # files= config.h config.mak config.asm
608. # 循环输出
609. for f in $files; do
610. # "x#\*/"代表去取x的第一个slash之后的所有内容（不包括slash）
611. # "#"代表删除从前往后最小匹配的内容
612. # "f##\*."代表去取f的第一个"."之后的所有内容。在这里是“h”、“mak”等
613. # 在这里print\_config\_h(),print\_config\_mak(),print\_config\_asm()
614. "print\_config\_${f##\*.}" $cfg ${pfx}${ucname} >>$f
615. done
616. done
617. }
619. print\_enabled(){
620. test "$1" = -n && end=" " && shift || end="\n"
621. suf=$1
622. shift
623. for v; do
624. enabled $v && printf "%s$end" ${v%$suf};
625. done
626. }
627. # 添加
628. # 示例append config\_files "config.asm"
629. append(){
630. var=$1
631. shift
632. # eval命令将会首先扫描命令行进行所有的置换，然后再执行该命令
633. # 按照上面的示例，置换后为 config\_files="$config\_files config.asm"
634. eval "$var=\"\$$var $\*\""
635. }
637. prepend(){
638. var=$1
639. shift
640. eval "$var=\"$\* \$$var\""
641. }
643. add\_cppflags(){
644. append CPPFLAGS $($filter\_cppflags "$@")
645. }
647. add\_cflags(){
648. append CFLAGS $($filter\_cflags "$@")
649. }
651. add\_cxxflags(){
652. append CXXFLAGS $($filter\_cflags "$@")
653. }
655. add\_asflags(){
656. append ASFLAGS $($filter\_asflags "$@")
657. }
659. add\_ldflags(){
660. append LDFLAGS "$@"
661. }
663. add\_extralibs(){
664. prepend extralibs "$@"
665. }
666. #2> 代表的是错误输出的重定向
667. #标准的输入、输出、和错误输出分别表示STDIN STDOUT STDERR 也可以用数字表示 0  1  2
668. #在这里也就是标准输出STDOUT 和 标准错误输出STDERR 都输入到了$logfile文件
669. check\_cmd(){
670. log "$@"
671. "$@" >> $logfile 2>&1
672. }
673. #检查CC编译器
674. check\_cc(){
675. log check\_cc "$@"
676. cat > $TMPC
677. log\_file $TMPC
678. #很多检查都调用了这个check\_cmd
679. #-c 只编译不链接
680. check\_cmd $cc $CPPFLAGS $CFLAGS "$@" -c -o $TMPO $TMPC
681. }
683. check\_cxx(){
684. log check\_cxx "$@"
685. cat > $TMPCPP
686. log\_file $TMPCPP
687. check\_cmd $cxx $CPPFLAGS $CFLAGS $CXXFLAGS "$@" -c -o $TMPO $TMPCPP
688. }
690. check\_cpp(){
691. log check\_cpp "$@"
692. cat > $TMPC
693. log\_file $TMPC
694. #-E选项，可以让编译器在预处理后停止，并输出预处理结果。
695. check\_cmd $cc $CPPFLAGS $CFLAGS "$@" -E -o $TMPO $TMPC
696. }
698. check\_as(){
699. log check\_as "$@"
700. cat > $TMPC
701. log\_file $TMPC
702. check\_cmd $as $CPPFLAGS $ASFLAGS "$@" -c -o $TMPO $TMPC
703. }
705. check\_asm(){
706. log check\_asm "$@"
707. name="$1"
708. code="$2"
709. shift 2
710. disable $name
711. check\_as "$@" <<EOF && enable $name
712. void foo(void){ \_\_asm\_\_ volatile($code); }
713. EOF
714. }
716. check\_yasm(){
717. log check\_yasm "$@"
718. echo "$1" > $TMPS
719. log\_file $TMPS
720. shift 1
721. check\_cmd $yasmexe $YASMFLAGS "$@" -o $TMPO $TMPS
722. }
724. check\_ld(){
725. log check\_ld "$@"
726. type=$1
727. shift 1
728. flags=''
729. libs=''
730. for f; do
731. test "${f}" = "${f#-l}" && flags="$flags $f" || libs="$libs $f"
732. done
733. check\_$type $($filter\_cflags $flags) || return
734. #编译连接
735. check\_cmd $ld $LDFLAGS $flags -o $TMPE $TMPO $libs $extralibs
736. }
738. check\_cppflags(){
739. log check\_cppflags "$@"
740. set -- $($filter\_cppflags "$@")
741. check\_cc "$@" <<EOF && append CPPFLAGS "$@"
742. int x;
743. EOF
744. }
746. check\_cflags(){
747. log check\_cflags "$@"
748. set -- $($filter\_cflags "$@")
749. check\_cc "$@" <<EOF && append CFLAGS "$@"
750. int x;
751. EOF
752. }
754. check\_cxxflags(){
755. log check\_cxxflags "$@"
756. set -- $($filter\_cflags "$@")
757. check\_cxx "$@" <<EOF && append CXXFLAGS "$@"
758. int x;
759. EOF
760. }
762. test\_ldflags(){
763. log test\_ldflags "$@"
764. check\_ld "cc" "$@" <<EOF
765. int main(void){ return 0; }
766. EOF
767. }
769. check\_ldflags(){
770. log check\_ldflags "$@"
771. test\_ldflags "$@" && add\_ldflags "$@"
772. }
773. #检查头文件
774. #生成一个简单的源代码文件
775. check\_header(){
776. log check\_header "$@"
777. header=$1
778. shift
779. disable\_safe $header
780. check\_cpp "$@" <<EOF && enable\_safe $header
781. #include <$header>
782. int x;
783. EOF
784. }
786. check\_func(){
787. log check\_func "$@"
788. func=$1
789. shift
790. disable $func
791. check\_ld "cc" "$@" <<EOF && enable $func
792. extern int $func();
793. int main(void){ $func(); }
794. EOF
795. }
796. #检查数学函数
797. check\_mathfunc(){
798. log check\_mathfunc "$@"
799. #数学函数名称
800. func=$1
801. shift
802. disable $func
803. check\_ld "cc" "$@" <<EOF && enable $func
804. #include <math.h>
805. float foo(float f) { return $func(f); }
806. int main(void){ return (int) foo; }
807. EOF
808. }
810. check\_func\_headers(){
811. log check\_func\_headers "$@"
812. headers=$1
813. funcs=$2
814. shift 2
815. {
816. for hdr in $headers; do
817. echo "#include <$hdr>"
818. done
819. for func in $funcs; do
820. echo "long check\_$func(void) { return (long) $func; }"
821. done
822. echo "int main(void) { return 0; }"
823. } | check\_ld "cc" "$@" && enable $funcs && enable\_safe $headers
824. }
826. check\_class\_headers\_cpp(){
827. log check\_class\_headers\_cpp "$@"
828. headers=$1
829. classes=$2
830. shift 2
831. {
832. for hdr in $headers; do
833. echo "#include <$hdr>"
834. done
835. echo "int main(void) { "
836. i=1
837. for class in $classes; do
838. echo "$class obj$i;"
839. i=$(expr $i + 1)
840. done
841. echo "return 0; }"
842. } | check\_ld "cxx" "$@" && enable $funcs && enable\_safe $headers
843. }
845. check\_cpp\_condition(){
846. log check\_cpp\_condition "$@"
847. header=$1
848. condition=$2
849. shift 2
850. check\_cpp $($filter\_cppflags "$@") <<EOF
851. #include <$header>
852. #if !($condition)
853. #error "unsatisfied condition: $condition"
854. #endif
855. EOF
856. }
857. #检查类库
858. check\_lib(){
859. log check\_lib "$@"
860. header="$1"
861. func="$2"
862. shift 2
863. check\_header $header && check\_func $func "$@" && add\_extralibs "$@"
864. }
866. check\_lib2(){
867. log check\_lib2 "$@"
868. headers="$1"
869. funcs="$2"
870. shift 2
871. check\_func\_headers "$headers" "$funcs" "$@" && add\_extralibs "$@"
872. }
874. check\_lib\_cpp(){
875. log check\_lib\_cpp "$@"
876. headers="$1"
877. classes="$2"
878. shift 2
879. check\_class\_headers\_cpp "$headers" "$classes" "$@" && add\_extralibs "$@"
880. }
882. check\_pkg\_config(){
883. log check\_pkg\_config "$@"
884. pkg="$1"
885. headers="$2"
886. funcs="$3"
887. shift 3
888. $pkg\_config --exists $pkg 2>/dev/null || return
889. pkg\_cflags=$($pkg\_config --cflags $pkg)
890. pkg\_libs=$($pkg\_config --libs $pkg)
891. check\_func\_headers "$headers" "$funcs" $pkg\_cflags $pkg\_libs "$@" &&
892. set\_safe ${pkg}\_cflags $pkg\_cflags   &&
893. set\_safe ${pkg}\_libs   $pkg\_libs
894. }
896. check\_exec(){
897. check\_ld "cc" "$@" && { enabled cross\_compile || $TMPE >> $logfile 2>&1; }
898. }
900. check\_exec\_crash(){
901. code=$(cat)
903. # exit() is not async signal safe.  \_Exit (C99) and \_exit (POSIX)
904. # are safe but may not be available everywhere.  Thus we use
905. # raise(SIGTERM) instead.  The check is run in a subshell so we
906. # can redirect the "Terminated" message from the shell.  SIGBUS
907. # is not defined by standard C so it is used conditionally.
909. (check\_exec "$@") >> $logfile 2>&1 <<EOF
910. #include <signal.h>
911. static void sighandler(int sig){
912. raise(SIGTERM);
913. }
914. int func(void){
915. $code
916. }
917. int main(void){
918. signal(SIGILL, sighandler);
919. signal(SIGFPE, sighandler);
920. signal(SIGSEGV, sighandler);
921. #ifdef SIGBUS
922. signal(SIGBUS, sighandler);
923. #endif
924. return func();
925. }
926. EOF
927. }
929. check\_type(){
930. log check\_type "$@"
931. headers=$1
932. type=$2
933. shift 2
934. disable\_safe "$type"
935. incs=""
936. for hdr in $headers; do
937. incs="$incs
938. #include <$hdr>"
939. done
940. check\_cc "$@" <<EOF && enable\_safe "$type"
941. $incs
942. $type v;
943. EOF
944. }
946. check\_struct(){
947. log check\_type "$@"
948. headers=$1
949. struct=$2
950. member=$3
951. shift 3
952. disable\_safe "${struct}\_${member}"
953. incs=""
954. for hdr in $headers; do
955. incs="$incs
956. #include <$hdr>"
957. done
958. check\_cc "$@" <<EOF && enable\_safe "${struct}\_${member}"
959. $incs
960. const void \*p = &(($struct \*)0)->$member;
961. EOF
962. }
963. #检查依赖项的时候使用
964. require(){
965. name="$1"
966. header="$2"
967. func="$3"
968. shift 3
969. check\_lib $header $func "$@" || die "ERROR: $name not found"
970. }
972. require2(){
973. name="$1"
974. headers="$2"
975. func="$3"
976. shift 3
977. check\_lib2 "$headers" $func "$@" || die "ERROR: $name not found"
978. }
980. require\_cpp(){
981. name="$1"
982. headers="$2"
983. classes="$3"
984. shift 3
985. check\_lib\_cpp "$headers" "$classes" "$@" || die "ERROR: $name not found"
986. }
988. require\_pkg\_config(){
989. pkg="$1"
990. check\_pkg\_config "$@" || die "ERROR: $pkg not found"
991. add\_cflags    $(get\_safe ${pkg}\_cflags)
992. add\_extralibs $(get\_safe ${pkg}\_libs)
993. }
995. check\_host\_cc(){
996. log check\_host\_cc "$@"
997. cat > $TMPC
998. log\_file $TMPC
999. check\_cmd $host\_cc $host\_cflags "$@" -c -o $TMPO $TMPC
1000. }
1002. check\_host\_cflags(){
1003. log check\_host\_cflags "$@"
1004. check\_host\_cc "$@" <<EOF && append host\_cflags "$@"
1005. int x;
1006. EOF
1007. }
1009. apply(){
1010. file=$1
1011. shift
1012. "$@" < "$file" > "$file.tmp" && mv "$file.tmp" "$file" || rm "$file.tmp"
1013. }
1014. #比较两个文件${1}和${2}，如果两个文件发生了变化，则将${1}强制覆盖${2}
1015. #该函数主要用于生成config.h
1016. cp\_if\_changed(){
1017. #cmp是二进制文件比较命令
1018. #-s：只返回退出值。值0（真）指示相同的文件；值1（假）指示不同的文件；值 2 指示不可访问的文件或缺少选项。
1019. cmp -s "$1" "$2" && echo "$2 is unchanged" && return
1020. mkdir -p "$(dirname $2)"
1021. cp -f "$1" "$2"
1022. }
1024. # CONFIG\_LIST contains configurable options, while HAVE\_LIST is for
1025. # system-dependent things.
1026. #各种List
1027. COMPONENT\_LIST="
1028. bsfs
1029. decoders
1030. demuxers
1031. encoders
1032. filters
1033. hwaccels
1034. indevs
1035. muxers
1036. outdevs
1037. parsers
1038. protocols
1039. "
1041. PROGRAM\_LIST="
1042. ffplay
1043. ffprobe
1044. ffserver
1045. ffmpeg
1046. "
1048. CONFIG\_LIST="
1049. #组件
1050. $COMPONENT\_LIST
1051. #可执行程序
1052. $PROGRAM\_LIST
1053. avplay
1054. avprobe
1055. avserver
1056. aandct
1057. ac3dsp
1058. avcodec
1059. avdevice
1060. avfilter
1061. avformat
1062. avisynth
1063. bzlib
1064. crystalhd
1065. dct
1066. doc
1067. dwt
1068. dxva2
1069. fastdiv
1070. fft
1071. frei0r
1072. gnutls
1073. golomb
1074. gpl
1075. gray
1076. h264chroma
1077. h264dsp
1078. h264pred
1079. hardcoded\_tables
1080. huffman
1081. libaacplus
1082. libass
1083. libcdio
1084. libcelt
1085. libdc1394
1086. libdirac
1087. libfaac
1088. libfreetype
1089. libgsm
1090. libmodplug
1091. libmp3lame
1092. libnut
1093. libopencore\_amrnb
1094. libopencore\_amrwb
1095. libopencv
1096. libopenjpeg
1097. libpulse
1098. librtmp
1099. libschroedinger
1100. libspeex
1101. libstagefright\_h264
1102. libtheora
1103. libutvideo
1104. libv4l2
1105. libvo\_aacenc
1106. libvo\_amrwbenc
1107. libvorbis
1108. libvpx
1109. libx264
1110. libxavs
1111. libxvid
1112. lpc
1113. lsp
1114. mdct
1115. memalign\_hack
1116. mlib
1117. mpegaudiodsp
1118. network
1119. nonfree
1120. openal
1121. openssl
1122. pic
1123. postproc
1124. rdft
1125. rtpdec
1126. runtime\_cpudetect
1127. safe\_bitstream\_reader
1128. shared
1129. sinewin
1130. small
1131. sram
1132. static
1133. swresample
1134. swscale
1135. swscale\_alpha
1136. thumb
1137. vaapi
1138. vda
1139. vdpau
1140. version3
1141. x11grab
1142. zlib
1143. "
1145. THREADS\_LIST='
1146. pthreads
1147. w32threads
1148. os2threads
1149. '
1151. ARCH\_LIST='
1152. alpha
1153. arm
1154. avr32
1155. avr32\_ap
1156. avr32\_uc
1157. bfin
1158. ia64
1159. m68k
1160. mips
1161. mips64
1162. parisc
1163. ppc
1164. ppc64
1165. s390
1166. sh4
1167. sparc
1168. sparc64
1169. tomi
1170. x86
1171. x86\_32
1172. x86\_64
1173. '
1175. ARCH\_EXT\_LIST='
1176. altivec
1177. amd3dnow
1178. amd3dnowext
1179. armv5te
1180. armv6
1181. armv6t2
1182. armvfp
1183. avx
1184. iwmmxt
1185. mmi
1186. mmx
1187. mmx2
1188. neon
1189. ppc4xx
1190. sse
1191. ssse3
1192. vfpv3
1193. vis
1194. '
1196. HAVE\_LIST\_PUB='
1197. bigendian
1198. fast\_unaligned
1199. '
1201. HAVE\_LIST="
1202. $ARCH\_EXT\_LIST
1203. $HAVE\_LIST\_PUB
1204. $THREADS\_LIST
1205. aligned\_stack
1206. alsa\_asoundlib\_h
1207. altivec\_h
1208. arpa\_inet\_h
1209. asm\_mod\_y
1210. asm\_types\_h
1211. attribute\_may\_alias
1212. attribute\_packed
1213. cbrtf
1214. closesocket
1215. cmov
1216. dcbzl
1217. dev\_bktr\_ioctl\_bt848\_h
1218. dev\_bktr\_ioctl\_meteor\_h
1219. dev\_ic\_bt8xx\_h
1220. dev\_video\_bktr\_ioctl\_bt848\_h
1221. dev\_video\_meteor\_ioctl\_meteor\_h
1222. dlfcn\_h
1223. dlopen
1224. dos\_paths
1225. ebp\_available
1226. ebx\_available
1227. exp2
1228. exp2f
1229. fast\_64bit
1230. fast\_clz
1231. fast\_cmov
1232. fcntl
1233. fork
1234. getaddrinfo
1235. gethrtime
1236. GetProcessAffinityMask
1237. GetProcessMemoryInfo
1238. GetProcessTimes
1239. getrusage
1240. gnu\_as
1241. ibm\_asm
1242. inet\_aton
1243. inline\_asm
1244. isatty
1245. kbhit
1246. ldbrx
1247. llrint
1248. llrintf
1249. local\_aligned\_16
1250. local\_aligned\_8
1251. localtime\_r
1252. log2
1253. log2f
1254. loongson
1255. lrint
1256. lrintf
1257. lzo1x\_999\_compress
1258. machine\_ioctl\_bt848\_h
1259. machine\_ioctl\_meteor\_h
1260. makeinfo
1261. malloc\_h
1262. MapViewOfFile
1263. memalign
1264. mkstemp
1265. mmap
1266. PeekNamedPipe
1267. poll\_h
1268. posix\_memalign
1269. round
1270. roundf
1271. sched\_getaffinity
1272. sdl
1273. sdl\_video\_size
1274. setmode
1275. setrlimit
1276. sndio\_h
1277. socklen\_t
1278. soundcard\_h
1279. strerror\_r
1280. strptime
1281. struct\_addrinfo
1282. struct\_ipv6\_mreq
1283. struct\_rusage\_ru\_maxrss
1284. struct\_sockaddr\_in6
1285. struct\_sockaddr\_sa\_len
1286. struct\_sockaddr\_storage
1287. struct\_v4l2\_frmivalenum\_discrete
1288. symver
1289. symver\_asm\_label
1290. symver\_gnu\_asm
1291. sysconf
1292. sysctl
1293. sys\_mman\_h
1294. sys\_param\_h
1295. sys\_resource\_h
1296. sys\_select\_h
1297. sys\_soundcard\_h
1298. sys\_videoio\_h
1299. termios\_h
1300. threads
1301. trunc
1302. truncf
1303. vfp\_args
1304. VirtualAlloc
1305. winsock2\_h
1306. xform\_asm
1307. xmm\_clobbers
1308. yasm
1309. "
1311. # options emitted with CONFIG\_ prefix but not available on command line
1312. CONFIG\_EXTRA="
1313. avutil
1314. gplv3
1315. lgplv3
1316. "
1318. CMDLINE\_SELECT="
1319. $ARCH\_EXT\_LIST
1320. $CONFIG\_LIST
1321. $THREADS\_LIST
1322. asm
1323. coverage
1324. cross\_compile
1325. debug
1326. extra\_warnings
1327. logging
1328. optimizations
1329. stripping
1330. symver
1331. yasm
1332. "
1334. PATHS\_LIST='
1335. bindir
1336. datadir
1337. incdir
1338. libdir
1339. mandir
1340. prefix
1341. shlibdir
1342. '
1344. CMDLINE\_SET="
1345. $PATHS\_LIST
1346. ar
1347. arch
1348. as
1349. build\_suffix
1350. progs\_suffix
1351. cc
1352. cpu
1353. cross\_prefix
1354. cxx
1355. dep\_cc
1356. extra\_version
1357. host\_cc
1358. host\_cflags
1359. host\_ldflags
1360. host\_libs
1361. host\_os
1362. install
1363. ld
1364. logfile
1365. malloc\_prefix
1366. nm
1367. optflags
1368. pkg\_config
1369. samples
1370. strip
1371. sysinclude
1372. sysroot
1373. target\_exec
1374. target\_os
1375. target\_path
1376. postproc\_version
1377. valgrind
1378. yasmexe
1379. "
1381. CMDLINE\_APPEND="
1382. extra\_cflags
1383. extra\_cxxflags
1384. "
1386. # code dependency declarations
1388. # architecture extensions
1390. armv5te\_deps="arm"
1391. armv6\_deps="arm"
1392. armv6t2\_deps="arm"
1393. armvfp\_deps="arm"
1394. iwmmxt\_deps="arm"
1395. neon\_deps="arm"
1396. vfpv3\_deps="armvfp"
1398. mmi\_deps="mips"
1400. altivec\_deps="ppc"
1401. ppc4xx\_deps="ppc"
1403. vis\_deps="sparc"
1405. x86\_64\_suggest="cmov fast\_cmov"
1406. amd3dnow\_deps="mmx"
1407. amd3dnowext\_deps="amd3dnow"
1408. mmx\_deps="x86"
1409. mmx2\_deps="mmx"
1410. sse\_deps="mmx"
1411. ssse3\_deps="sse"
1412. avx\_deps="ssse3"
1414. aligned\_stack\_if\_any="ppc x86"
1415. fast\_64bit\_if\_any="alpha ia64 mips64 parisc64 ppc64 sparc64 x86\_64"
1416. fast\_clz\_if\_any="alpha armv5te avr32 mips ppc x86"
1417. fast\_unaligned\_if\_any="armv6 ppc x86"
1419. inline\_asm\_deps="!tms470"
1420. need\_memalign="altivec neon sse"
1422. symver\_if\_any="symver\_asm\_label symver\_gnu\_asm"
1424. # subsystems
1425. dct\_select="rdft"
1426. mdct\_select="fft"
1427. rdft\_select="fft"
1428. mpegaudiodsp\_select="dct"
1430. # decoders / encoders / hardware accelerators
1431. aac\_decoder\_select="mdct sinewin"
1432. aac\_encoder\_select="mdct sinewin"
1433. aac\_latm\_decoder\_select="aac\_decoder aac\_latm\_parser"
1434. ac3\_decoder\_select="mdct ac3dsp ac3\_parser"
1435. ac3\_encoder\_select="mdct ac3dsp"
1436. ac3\_fixed\_encoder\_select="mdct ac3dsp"
1437. alac\_encoder\_select="lpc"
1438. amrnb\_decoder\_select="lsp"
1439. amrwb\_decoder\_select="lsp"
1440. atrac1\_decoder\_select="mdct sinewin"
1441. atrac3\_decoder\_select="mdct"
1442. binkaudio\_dct\_decoder\_select="mdct rdft dct sinewin"
1443. binkaudio\_rdft\_decoder\_select="mdct rdft sinewin"
1444. cavs\_decoder\_select="golomb"
1445. cook\_decoder\_select="mdct sinewin"
1446. cscd\_decoder\_suggest="zlib"
1447. dca\_decoder\_select="mdct"
1448. dnxhd\_encoder\_select="aandct"
1449. dxa\_decoder\_select="zlib"
1450. eac3\_decoder\_select="ac3\_decoder"
1451. eac3\_encoder\_select="mdct ac3dsp"
1452. eamad\_decoder\_select="aandct"
1453. eatgq\_decoder\_select="aandct"
1454. eatqi\_decoder\_select="aandct"
1455. ffv1\_decoder\_select="golomb"
1456. flac\_decoder\_select="golomb"
1457. flac\_encoder\_select="golomb lpc"
1458. flashsv\_decoder\_select="zlib"
1459. flashsv\_encoder\_select="zlib"
1460. flashsv2\_encoder\_select="zlib"
1461. flashsv2\_decoder\_select="zlib"
1462. flv\_decoder\_select="h263\_decoder"
1463. flv\_encoder\_select="h263\_encoder"
1464. fraps\_decoder\_select="huffman"
1465. h261\_encoder\_select="aandct"
1466. h263\_decoder\_select="h263\_parser"
1467. h263\_encoder\_select="aandct"
1468. h263\_vaapi\_hwaccel\_select="vaapi h263\_decoder"
1469. h263i\_decoder\_select="h263\_decoder"
1470. h263p\_encoder\_select="h263\_encoder"
1471. h264\_crystalhd\_decoder\_select="crystalhd h264\_mp4toannexb\_bsf h264\_parser"
1472. h264\_decoder\_select="golomb h264chroma h264dsp h264pred"
1473. h264\_dxva2\_hwaccel\_deps="dxva2api\_h"
1474. h264\_dxva2\_hwaccel\_select="dxva2 h264\_decoder"
1475. h264\_vaapi\_hwaccel\_select="vaapi h264\_decoder"
1476. h264\_vda\_hwaccel\_deps="VideoDecodeAcceleration\_VDADecoder\_h pthreads"
1477. h264\_vda\_hwaccel\_select="vda h264\_decoder"
1478. h264\_vdpau\_decoder\_select="vdpau h264\_decoder"
1479. imc\_decoder\_select="fft mdct sinewin"
1480. jpegls\_decoder\_select="golomb"
1481. jpegls\_encoder\_select="golomb"
1482. ljpeg\_encoder\_select="aandct"
1483. loco\_decoder\_select="golomb"
1484. mjpeg\_encoder\_select="aandct"
1485. mlp\_decoder\_select="mlp\_parser"
1486. mp1\_decoder\_select="mpegaudiodsp"
1487. mp1float\_decoder\_select="mpegaudiodsp"
1488. mp2\_decoder\_select="mpegaudiodsp"
1489. mp2float\_decoder\_select="mpegaudiodsp"
1490. mp3\_decoder\_select="mpegaudiodsp"
1491. mp3adu\_decoder\_select="mpegaudiodsp"
1492. mp3adufloat\_decoder\_select="mpegaudiodsp"
1493. mp3float\_decoder\_select="mpegaudiodsp"
1494. mp3on4\_decoder\_select="mpegaudiodsp"
1495. mp3on4float\_decoder\_select="mpegaudiodsp"
1496. mpc7\_decoder\_select="mpegaudiodsp"
1497. mpc8\_decoder\_select="mpegaudiodsp"
1498. mpeg\_vdpau\_decoder\_select="vdpau mpegvideo\_decoder"
1499. mpeg\_xvmc\_decoder\_deps="X11\_extensions\_XvMClib\_h"
1500. mpeg\_xvmc\_decoder\_select="mpegvideo\_decoder"
1501. mpeg1\_vdpau\_decoder\_select="vdpau mpeg1video\_decoder"
1502. mpeg1\_vdpau\_hwaccel\_select="vdpau mpeg1video\_decoder"
1503. mpeg1video\_encoder\_select="aandct"
1504. mpeg2\_crystalhd\_decoder\_select="crystalhd"
1505. mpeg2\_dxva2\_hwaccel\_deps="dxva2api\_h"
1506. mpeg2\_dxva2\_hwaccel\_select="dxva2 mpeg2video\_decoder"
1507. mpeg2\_vdpau\_hwaccel\_select="vdpau mpeg2video\_decoder"
1508. mpeg2\_vaapi\_hwaccel\_select="vaapi mpeg2video\_decoder"
1509. mpeg2video\_encoder\_select="aandct"
1510. mpeg4\_crystalhd\_decoder\_select="crystalhd"
1511. mpeg4\_decoder\_select="h263\_decoder mpeg4video\_parser"
1512. mpeg4\_encoder\_select="h263\_encoder"
1513. mpeg4\_vaapi\_hwaccel\_select="vaapi mpeg4\_decoder"
1514. mpeg4\_vdpau\_decoder\_select="vdpau mpeg4\_decoder"
1515. msmpeg4\_crystalhd\_decoder\_select="crystalhd"
1516. msmpeg4v1\_decoder\_select="h263\_decoder"
1517. msmpeg4v1\_encoder\_select="h263\_encoder"
1518. msmpeg4v2\_decoder\_select="h263\_decoder"
1519. msmpeg4v2\_encoder\_select="h263\_encoder"
1520. msmpeg4v3\_decoder\_select="h263\_decoder"
1521. msmpeg4v3\_encoder\_select="h263\_encoder"
1522. nellymoser\_decoder\_select="mdct sinewin"
1523. nellymoser\_encoder\_select="mdct sinewin"
1524. png\_decoder\_select="zlib"
1525. png\_encoder\_select="zlib"
1526. qcelp\_decoder\_select="lsp"
1527. qdm2\_decoder\_select="mdct rdft mpegaudiodsp"
1528. ra\_144\_encoder\_select="lpc"
1529. rv10\_decoder\_select="h263\_decoder"
1530. rv10\_encoder\_select="h263\_encoder"
1531. rv20\_decoder\_select="h263\_decoder"
1532. rv20\_encoder\_select="h263\_encoder"
1533. rv30\_decoder\_select="golomb h264chroma h264pred"
1534. rv40\_decoder\_select="golomb h264chroma h264pred"
1535. shorten\_decoder\_select="golomb"
1536. sipr\_decoder\_select="lsp"
1537. snow\_decoder\_select="dwt"
1538. snow\_encoder\_select="aandct dwt"
1539. sonic\_decoder\_select="golomb"
1540. sonic\_encoder\_select="golomb"
1541. sonic\_ls\_encoder\_select="golomb"
1542. svq1\_encoder\_select="aandct"
1543. svq3\_decoder\_select="golomb h264chroma h264dsp h264pred"
1544. svq3\_decoder\_suggest="zlib"
1545. theora\_decoder\_select="vp3\_decoder"
1546. tiff\_decoder\_suggest="zlib"
1547. tiff\_encoder\_suggest="zlib"
1548. truehd\_decoder\_select="mlp\_decoder"
1549. tscc\_decoder\_select="zlib"
1550. twinvq\_decoder\_select="mdct lsp sinewin"
1551. vc1\_crystalhd\_decoder\_select="crystalhd"
1552. vc1\_decoder\_select="h263\_decoder h264chroma"
1553. vc1\_dxva2\_hwaccel\_deps="dxva2api\_h"
1554. vc1\_dxva2\_hwaccel\_select="dxva2 vc1\_decoder"
1555. vc1\_vaapi\_hwaccel\_select="vaapi vc1\_decoder"
1556. vc1\_vdpau\_decoder\_select="vdpau vc1\_decoder"
1557. vc1image\_decoder\_select="vc1\_decoder"
1558. vorbis\_decoder\_select="mdct"
1559. vorbis\_encoder\_select="mdct"
1560. vp6\_decoder\_select="huffman"
1561. vp6a\_decoder\_select="vp6\_decoder"
1562. vp6f\_decoder\_select="vp6\_decoder"
1563. vp8\_decoder\_select="h264pred"
1564. wmapro\_decoder\_select="mdct sinewin"
1565. wmav1\_decoder\_select="mdct sinewin"
1566. wmav1\_encoder\_select="mdct sinewin"
1567. wmav2\_decoder\_select="mdct sinewin"
1568. wmav2\_encoder\_select="mdct sinewin"
1569. wmavoice\_decoder\_select="lsp rdft dct mdct sinewin"
1570. wmv1\_decoder\_select="h263\_decoder"
1571. wmv1\_encoder\_select="h263\_encoder"
1572. wmv2\_decoder\_select="h263\_decoder"
1573. wmv2\_encoder\_select="h263\_encoder"
1574. wmv3\_decoder\_select="vc1\_decoder"
1575. wmv3\_crystalhd\_decoder\_select="crystalhd"
1576. wmv3\_dxva2\_hwaccel\_select="vc1\_dxva2\_hwaccel"
1577. wmv3\_vaapi\_hwaccel\_select="vc1\_vaapi\_hwaccel"
1578. wmv3\_vdpau\_decoder\_select="vc1\_vdpau\_decoder"
1579. wmv3image\_decoder\_select="wmv3\_decoder"
1580. zlib\_decoder\_select="zlib"
1581. zlib\_encoder\_select="zlib"
1582. zmbv\_decoder\_select="zlib"
1583. zmbv\_encoder\_select="zlib"
1585. crystalhd\_deps="libcrystalhd\_libcrystalhd\_if\_h"
1586. vaapi\_deps="va\_va\_h"
1587. vda\_deps="VideoDecodeAcceleration\_VDADecoder\_h pthreads"
1588. vdpau\_deps="vdpau\_vdpau\_h vdpau\_vdpau\_x11\_h"
1590. # parsers
1591. h264\_parser\_select="golomb h264chroma h264dsp h264pred"
1593. # external libraries
1594. libaacplus\_encoder\_deps="libaacplus"
1595. libcelt\_decoder\_deps="libcelt"
1596. libdirac\_decoder\_deps="libdirac !libschroedinger"
1597. libdirac\_encoder\_deps="libdirac"
1598. libfaac\_encoder\_deps="libfaac"
1599. libgsm\_decoder\_deps="libgsm"
1600. libgsm\_encoder\_deps="libgsm"
1601. libgsm\_ms\_decoder\_deps="libgsm"
1602. libgsm\_ms\_encoder\_deps="libgsm"
1603. libmodplug\_demuxer\_deps="libmodplug"
1604. libmp3lame\_encoder\_deps="libmp3lame"
1605. libopencore\_amrnb\_decoder\_deps="libopencore\_amrnb"
1606. libopencore\_amrnb\_encoder\_deps="libopencore\_amrnb"
1607. libopencore\_amrwb\_decoder\_deps="libopencore\_amrwb"
1608. libopenjpeg\_decoder\_deps="libopenjpeg"
1609. libopenjpeg\_encoder\_deps="libopenjpeg"
1610. libschroedinger\_decoder\_deps="libschroedinger"
1611. libschroedinger\_encoder\_deps="libschroedinger"
1612. libspeex\_decoder\_deps="libspeex"
1613. libspeex\_encoder\_deps="libspeex"
1614. libstagefright\_h264\_decoder\_deps="libstagefright\_h264"
1615. libtheora\_encoder\_deps="libtheora"
1616. libvo\_aacenc\_encoder\_deps="libvo\_aacenc"
1617. libvo\_amrwbenc\_encoder\_deps="libvo\_amrwbenc"
1618. libvorbis\_encoder\_deps="libvorbis"
1619. libvpx\_decoder\_deps="libvpx"
1620. libvpx\_encoder\_deps="libvpx"
1621. libx264\_encoder\_deps="libx264"
1622. libx264rgb\_encoder\_deps="libx264"
1623. libxavs\_encoder\_deps="libxavs"
1624. libxvid\_encoder\_deps="libxvid"
1625. libutvideo\_decoder\_deps="libutvideo gpl"
1627. # demuxers / muxers
1628. ac3\_demuxer\_select="ac3\_parser"
1629. asf\_stream\_muxer\_select="asf\_muxer"
1630. avisynth\_demuxer\_deps="avisynth"
1631. dirac\_demuxer\_select="dirac\_parser"
1632. eac3\_demuxer\_select="ac3\_parser"
1633. flac\_demuxer\_select="flac\_parser"
1634. ipod\_muxer\_select="mov\_muxer"
1635. libnut\_demuxer\_deps="libnut"
1636. libnut\_muxer\_deps="libnut"
1637. matroska\_audio\_muxer\_select="matroska\_muxer"
1638. matroska\_demuxer\_suggest="zlib bzlib"
1639. mov\_demuxer\_suggest="zlib"
1640. mp3\_demuxer\_select="mpegaudio\_parser"
1641. mp4\_muxer\_select="mov\_muxer"
1642. mpegtsraw\_demuxer\_select="mpegts\_demuxer"
1643. mxf\_d10\_muxer\_select="mxf\_muxer"
1644. ogg\_demuxer\_select="golomb"
1645. psp\_muxer\_select="mov\_muxer"
1646. rtp\_demuxer\_select="sdp\_demuxer"
1647. rtpdec\_select="asf\_demuxer rm\_demuxer rtp\_protocol mpegts\_demuxer mov\_demuxer"
1648. rtsp\_demuxer\_select="http\_protocol rtpdec"
1649. rtsp\_muxer\_select="rtp\_muxer http\_protocol rtp\_protocol"
1650. sap\_demuxer\_select="sdp\_demuxer"
1651. sap\_muxer\_select="rtp\_muxer rtp\_protocol"
1652. sdp\_demuxer\_select="rtpdec"
1653. spdif\_muxer\_select="aac\_parser"
1654. tg2\_muxer\_select="mov\_muxer"
1655. tgp\_muxer\_select="mov\_muxer"
1656. w64\_demuxer\_deps="wav\_demuxer"
1658. # indevs / outdevs
1659. alsa\_indev\_deps="alsa\_asoundlib\_h snd\_pcm\_htimestamp"
1660. alsa\_outdev\_deps="alsa\_asoundlib\_h"
1661. bktr\_indev\_deps\_any="dev\_bktr\_ioctl\_bt848\_h machine\_ioctl\_bt848\_h dev\_video\_bktr\_ioctl\_bt848\_h dev\_ic\_bt8xx\_h"
1662. dshow\_indev\_deps="IBaseFilter"
1663. dshow\_indev\_extralibs="-lpsapi -lole32 -lstrmiids -luuid"
1664. dv1394\_indev\_deps="dv1394 dv\_demuxer"
1665. fbdev\_indev\_deps="linux\_fb\_h"
1666. jack\_indev\_deps="jack\_jack\_h sem\_timedwait"
1667. lavfi\_indev\_deps="avfilter"
1668. libcdio\_indev\_deps="libcdio"
1669. libdc1394\_indev\_deps="libdc1394"
1670. libv4l2\_indev\_deps="libv4l2"
1671. openal\_indev\_deps="openal"
1672. oss\_indev\_deps\_any="soundcard\_h sys\_soundcard\_h"
1673. oss\_outdev\_deps\_any="soundcard\_h sys\_soundcard\_h"
1674. pulse\_indev\_deps="libpulse"
1675. sdl\_outdev\_deps="sdl"
1676. sndio\_indev\_deps="sndio\_h"
1677. sndio\_outdev\_deps="sndio\_h"
1678. v4l\_indev\_deps="linux\_videodev\_h"
1679. v4l2\_indev\_deps\_any="linux\_videodev2\_h sys\_videoio\_h"
1680. vfwcap\_indev\_deps="capCreateCaptureWindow vfwcap\_defines"
1681. vfwcap\_indev\_extralibs="-lavicap32"
1682. x11\_grab\_device\_indev\_deps="x11grab XShmCreateImage"
1683. x11\_grab\_device\_indev\_extralibs="-lX11 -lXext -lXfixes"
1685. # protocols
1686. gopher\_protocol\_deps="network"
1687. httpproxy\_protocol\_deps="network"
1688. httpproxy\_protocol\_select="tcp\_protocol"
1689. http\_protocol\_deps="network"
1690. http\_protocol\_select="tcp\_protocol"
1691. https\_protocol\_select="tls\_protocol"
1692. mmsh\_protocol\_select="http\_protocol"
1693. mmst\_protocol\_deps="network"
1694. rtmp\_protocol\_select="tcp\_protocol"
1695. rtp\_protocol\_select="udp\_protocol"
1696. tcp\_protocol\_deps="network"
1697. tls\_protocol\_deps\_any="openssl gnutls"
1698. tls\_protocol\_select="tcp\_protocol"
1699. udp\_protocol\_deps="network"
1701. # filters
1702. amovie\_filter\_deps="avcodec avformat"
1703. ass\_filter\_deps="libass"
1704. blackframe\_filter\_deps="gpl"
1705. boxblur\_filter\_deps="gpl"
1706. cropdetect\_filter\_deps="gpl"
1707. delogo\_filter\_deps="gpl"
1708. drawtext\_filter\_deps="libfreetype"
1709. frei0r\_filter\_deps="frei0r dlopen"
1710. frei0r\_src\_filter\_deps="frei0r dlopen"
1711. hqdn3d\_filter\_deps="gpl"
1712. movie\_filter\_deps="avcodec avformat"
1713. mp\_filter\_deps="gpl avcodec"
1714. mptestsrc\_filter\_deps="gpl"
1715. negate\_filter\_deps="lut\_filter"
1716. ocv\_filter\_deps="libopencv"
1717. pan\_filter\_deps="swresample"
1718. scale\_filter\_deps="swscale"
1719. tinterlace\_filter\_deps="gpl"
1720. yadif\_filter\_deps="gpl"
1722. # libraries
1723. avdevice\_deps="avcodec avformat"
1724. avformat\_deps="avcodec"
1725. postproc\_deps="gpl"
1727. # programs
1728. ffplay\_deps="avcodec avformat swscale sdl"
1729. ffplay\_select="buffersink\_filter rdft"
1730. ffprobe\_deps="avcodec avformat"
1731. ffserver\_deps="avformat ffm\_muxer fork rtp\_protocol rtsp\_demuxer"
1732. ffserver\_extralibs='$ldl'
1733. ffmpeg\_deps="avcodec avformat swscale swresample"
1734. ffmpeg\_select="buffersink\_filter"
1736. doc\_deps="texi2html"
1738. # tests
1740. test\_deps(){
1741. suf1=$1
1742. suf2=$2
1743. shift 2
1744. for v; do
1745. dep=${v%=\*}
1746. tests=${v#\*=}
1747. for name in ${tests}; do
1748. append ${name}\_test\_deps ${dep}$suf1 ${dep}$suf2
1749. done
1750. done
1751. }
1753. mxf\_d10\_test\_deps="avfilter"
1754. seek\_lavf\_mxf\_d10\_test\_deps="mxf\_d10\_test"
1756. test\_deps \_encoder \_decoder                                             \
1757. adpcm\_ima\_qt                                                        \
1758. adpcm\_ima\_wav                                                       \
1759. adpcm\_ms                                                            \
1760. adpcm\_swf                                                           \
1761. adpcm\_yamaha=adpcm\_yam                                              \
1762. alac                                                                \
1763. asv1                                                                \
1764. asv2                                                                \
1765. bmp                                                                 \
1766. dnxhd="dnxhd\_1080i dnxhd\_720p dnxhd\_720p\_rd"                        \
1767. dvvideo="dv dv\_411 dv50"                                            \
1768. ffv1                                                                \
1769. flac                                                                \
1770. flashsv                                                             \
1771. flv                                                                 \
1772. adpcm\_g726=g726                                                     \
1773. gif                                                                 \
1774. h261                                                                \
1775. h263="h263 h263p"                                                   \
1776. huffyuv                                                             \
1777. jpegls                                                              \
1778. mjpeg="jpg mjpeg ljpeg"                                             \
1779. mp2                                                                 \
1780. mpeg1video="mpeg mpeg1b"                                            \
1781. mpeg2video="mpeg2 mpeg2\_422 mpeg2\_idct\_int mpeg2\_ilace mpeg2\_ivlc\_qprd" \
1782. mpeg2video="mpeg2thread mpeg2thread\_ilace"                          \
1783. mpeg4="mpeg4 mpeg4\_adap mpeg4\_qpel mpeg4\_qprd mpeg4adv mpeg4nr"     \
1784. mpeg4="mpeg4thread error rc"                                        \
1785. msmpeg4v3=msmpeg4                                                   \
1786. msmpeg4v2                                                           \
1787. pbm=pbmpipe                                                         \
1788. pcx                                                                 \
1789. pgm="pgm pgmpipe"                                                   \
1790. png                                                                 \
1791. ppm="ppm ppmpipe"                                                   \
1792. rawvideo="rgb yuv"                                                  \
1793. roq                                                                 \
1794. rv10                                                                \
1795. rv20                                                                \
1796. sgi                                                                 \
1797. snow="snow snowll"                                                  \
1798. svq1                                                                \
1799. targa=tga                                                           \
1800. tiff                                                                \
1801. wmav1                                                               \
1802. wmav2                                                               \
1803. wmv1                                                                \
1804. wmv2                                                                \
1806. test\_deps \_muxer \_demuxer                                               \
1807. aiff                                                                \
1808. pcm\_alaw=alaw                                                       \
1809. asf                                                                 \
1810. au                                                                  \
1811. avi                                                                 \
1812. dv=dv\_fmt                                                           \
1813. ffm                                                                 \
1814. flv=flv\_fmt                                                         \
1815. gxf                                                                 \
1816. matroska=mkv                                                        \
1817. mmf                                                                 \
1818. mov                                                                 \
1819. pcm\_mulaw=mulaw                                                     \
1820. mxf="mxf mxf\_d10"                                                   \
1821. nut                                                                 \
1822. ogg                                                                 \
1823. rawvideo=pixfmt                                                     \
1824. rm                                                                  \
1825. swf                                                                 \
1826. mpegts=ts                                                           \
1827. voc                                                                 \
1828. wav                                                                 \
1829. yuv4mpegpipe=yuv4mpeg                                               \
1831. ac3\_fixed\_test\_deps="ac3\_fixed\_encoder ac3\_decoder rm\_muxer rm\_demuxer"
1832. mpg\_test\_deps="mpeg1system\_muxer mpegps\_demuxer"
1834. # 默认参数 default parameters
1835. # 日志
1836. logfile="config.log"
1838. # 安装路径 installation paths
1839. prefix\_default="/usr/local"
1840. bindir\_default='${prefix}/bin'
1841. datadir\_default='${prefix}/share/ffmpeg'
1842. incdir\_default='${prefix}/include'
1843. libdir\_default='${prefix}/lib'
1844. mandir\_default='${prefix}/share/man'
1845. shlibdir\_default="$libdir\_default"
1846. postproc\_version\_default="current"
1848. # 工具链 toolchain
1849. ar\_default="ar"
1850. cc\_default="gcc"
1851. cxx\_default="g++"
1852. cc\_version=\"unknown\"
1853. host\_cc\_default="gcc"
1854. install="install"
1855. ln\_s="ln -sf"
1856. nm\_default="nm"
1857. objformat="elf"
1858. pkg\_config\_default=pkg-config
1859. ranlib="ranlib"
1860. strip\_default="strip"
1861. yasmexe\_default="yasm"
1863. nm\_opts='-g'
1864. nogas=":"
1866. # 机器 machine
1867. arch\_default=$(uname -m)
1868. cpu="generic"
1870. # 操作系统 OS
1871. target\_os\_default=$(tolower $(uname -s))
1872. host\_os=$target\_os\_default
1874. # alternative libpostproc version
1875. ALT\_PP\_VER\_MAJOR=51
1876. ALT\_PP\_VER\_MINOR=2
1877. ALT\_PP\_VER\_MICRO=101
1878. ALT\_PP\_VER=$ALT\_PP\_VER\_MAJOR.$ALT\_PP\_VER\_MINOR.$ALT\_PP\_VER\_MICRO
1880. # 选项 configurable options
1881. # PROGRAM\_LIST内容是 ffplay ffprobe ffserver ffmpeg
1882. enable $PROGRAM\_LIST
1884. enable avcodec
1885. enable avdevice
1886. enable avfilter
1887. enable avformat
1888. enable avutil
1889. enable postproc
1890. enable stripping
1891. enable swresample
1892. enable swscale
1894. enable asm
1895. enable debug
1896. enable doc
1897. enable fastdiv
1898. enable network
1899. enable optimizations
1900. enable safe\_bitstream\_reader
1901. enable static
1902. enable swscale\_alpha
1904. # 编译选项 build settings
1905. SHFLAGS='-shared -Wl,-soname,$$(@F)'
1906. FFSERVERLDFLAGS=-Wl,-E
1907. # 前缀后缀
1908. LIBPREF="lib"
1909. LIBSUF=".a"
1910. FULLNAME='$(NAME)$(BUILDSUF)'
1911. # 名称
1912. LIBNAME='$(LIBPREF)$(FULLNAME)$(LIBSUF)'
1913. # 动态库前缀后缀
1914. SLIBPREF="lib"
1915. SLIBSUF=".so"
1916. # 名称
1917. SLIBNAME='$(SLIBPREF)$(FULLNAME)$(SLIBSUF)'
1918. SLIBNAME\_WITH\_VERSION='$(SLIBNAME).$(LIBVERSION)'
1919. SLIBNAME\_WITH\_MAJOR='$(SLIBNAME).$(LIBMAJOR)'
1920. LIB\_INSTALL\_EXTRA\_CMD='$$(RANLIB) "$(LIBDIR)/$(LIBNAME)"'
1921. SLIB\_INSTALL\_NAME='$(SLIBNAME\_WITH\_VERSION)'
1922. SLIB\_INSTALL\_LINKS='$(SLIBNAME\_WITH\_MAJOR) $(SLIBNAME)'
1924. AS\_O='-o $@'
1925. CC\_O='-o $@'
1926. CXX\_O='-o $@'
1928. host\_cflags='-D\_ISOC99\_SOURCE -O3 -g'
1929. host\_libs='-lm'
1931. target\_path='$(CURDIR)'
1933. # since the object filename is not given with the -MM flag, the compiler
1934. # is only able to print the basename, and we must add the path ourselves
1935. DEPEND\_CMD='$(DEPCC) $(DEPFLAGS) $< | sed -e "/^\#.\*/d" -e "s,^[[:space:]]\*$(\*F)\\.o,$(@D)/$(\*F).o," > $(@:.o=.d)'
1936. DEPFLAGS='$(CPPFLAGS) $(CFLAGS) -MM'
1938. # find source path
1939. # $0就是该bash文件名
1940. # dirname /home/lxh/test.txt 输出/home/lxh
1941. if test -f configure; then
1942. source\_path=.
1943. else
1944. source\_path=$(cd $(dirname "$0"); pwd)
1945. echo "$source\_path" | grep -q '[[:blank:]]' &&
1946. die "Out of tree builds are impossible with whitespace in source path."
1947. test -e "$source\_path/config.h" &&
1948. die "Out of tree builds are impossible with config.h in source dir."
1949. fi
1950. # 脚本名称叫test.sh
1951. # 入参三个: 1 2 3
1952. # 运行test.sh 1 2 3后
1953. # $\*为"1 2 3"（一起被引号包住）
1954. # $@为"1" "2" "3"（分别被包住）
1955. # $#为3（参数数量）
1956. for v in "$@"; do
1957. r=${v#\*=}
1958. l=${v%"$r"}
1959. r=$(sh\_quote "$r")
1960. FFMPEG\_CONFIGURATION="${FFMPEG\_CONFIGURATION# } ${l}${r}"
1961. done
1962. # ${数字} 一般是位置参数的用法。
1963. # 如果运行脚本的时候带参数，那么可以在脚本里通过 $1 获取第一个参数，$2 获取第二个参数
1964. # 例如以ENCODER\_LIST为例，$1为"encoder"，$2为"ENC"，$3为"libavcodec/allcodecs.c"
1965. find\_things(){
1966. thing=$1
1967. pattern=$2
1968. file=$source\_path/$3
1969. # 处理一行字符串？挺复杂
1970. sed -n "s/^[^#]\*$pattern.\*([^,]\*, \*[,]∗,.∗\*).\*/\1\_$thing/p" "$file"
1971. }
1972. #从allcodecs.c等文件中提取编解码器
1973. ENCODER\_LIST=$(find\_things  encoder  ENC      libavcodec/allcodecs.c)
1974. DECODER\_LIST=$(find\_things  decoder  DEC      libavcodec/allcodecs.c)
1975. HWACCEL\_LIST=$(find\_things  hwaccel  HWACCEL  libavcodec/allcodecs.c)
1976. PARSER\_LIST=$(find\_things   parser   PARSER   libavcodec/allcodecs.c)
1977. BSF\_LIST=$(find\_things      bsf      BSF      libavcodec/allcodecs.c)
1978. MUXER\_LIST=$(find\_things    muxer    \_MUX     libavformat/allformats.c)
1979. DEMUXER\_LIST=$(find\_things  demuxer  DEMUX    libavformat/allformats.c)
1980. OUTDEV\_LIST=$(find\_things   outdev   OUTDEV   libavdevice/alldevices.c)
1981. INDEV\_LIST=$(find\_things    indev    \_IN      libavdevice/alldevices.c)
1982. PROTOCOL\_LIST=$(find\_things protocol PROTOCOL libavformat/allformats.c)
1983. FILTER\_LIST=$(find\_things   filter   FILTER   libavfilter/allfilters.c)
1985. # 所有组件
1986. ALL\_COMPONENTS="
1987. $BSF\_LIST
1988. $DECODER\_LIST
1989. $DEMUXER\_LIST
1990. $ENCODER\_LIST
1991. $FILTER\_LIST
1992. $HWACCEL\_LIST
1993. $INDEV\_LIST
1994. $MUXER\_LIST
1995. $OUTDEV\_LIST
1996. $PARSER\_LIST
1997. $PROTOCOL\_LIST
1998. "
2000. find\_tests(){
2001. map "echo ${2}\${v}\_test" $(ls "$source\_path"/tests/ref/$1 | grep -v '[^-a-z0-9\_]')
2002. }
2004. ACODEC\_TESTS=$(find\_tests acodec)
2005. VCODEC\_TESTS=$(find\_tests vsynth1)
2006. LAVF\_TESTS=$(find\_tests lavf)
2007. LAVFI\_TESTS=$(find\_tests lavfi)
2008. SEEK\_TESTS=$(find\_tests seek seek\_)
2010. ALL\_TESTS="$ACODEC\_TESTS $VCODEC\_TESTS $LAVF\_TESTS $LAVFI\_TESTS $SEEK\_TESTS"
2012. pcm\_test\_deps=$(map 'echo ${v%\_\*}\_decoder $v' $(filter pcm\_\* $ENCODER\_LIST))
2014. for n in $COMPONENT\_LIST; do
2015. v=$(toupper ${n%s})\_LIST
2016. eval enable \$$v
2017. eval ${n}\_if\_any="\$$v"
2018. done
2020. enable $ARCH\_EXT\_LIST $ALL\_TESTS
2022. die\_unknown(){
2023. echo "Unknown option \"$1\"."
2024. echo "See $0 --help for available options."
2025. exit 1
2026. }
2028. show\_list() {
2029. suffix=\_$1
2030. shift
2031. echo $\* | sed s/$suffix//g | tr ' ' '\n' | sort | pr -3 -t
2032. exit 0
2033. }
2034. # 解析各种各样的选项
2035. #
2036. # case分支语句的格式如下：
2037. #   case $变量名 in
2038. #       模式1）
2039. #   命令序列1
2040. #   ;;
2041. #       模式2）
2042. #   命令序列2
2043. #   ;;
2044. #       \*）
2045. #   默认执行的命令序列
2046. #   ;;
2047. #   esac
2048. # case语句结构特点如下：
2049. # case行尾必须为单词“in”，每一个模式必须以右括号“）”结束。
2050. # 双分号“;;”表示命令序列结束。
2051. # 最后的“\*）”表示默认模式，当使用前面的各种模式均无法匹配该变量时，将执行“\*）”后的命令序列。
2052. #
2053. #注意：opt不是参数列表（实际上也没有看见opt变量的定义）
2054. #原因是处在for循环中，当你没有为in指定列表时，for会默认取命令行参数列表。
2055. #因此“opt”这个名字实际上是可以随便取的
2056. for opt do
2057. # "#"用于去除特定字符前面的字符串
2058. # optval内容为opt去掉"="以及其前面字符串之后的内容
2059. optval="${opt#\*=}"
2060. case "$opt" in
2061. # 不同的选项
2062. --extra-ldflags=\*) add\_ldflags $optval
2063. ;;
2064. --extra-libs=\*) add\_extralibs $optval
2065. ;;
2066. --disable-devices) disable $INDEV\_LIST $OUTDEV\_LIST
2067. ;;
2068. --enable-debug=\*) debuglevel="$optval"
2069. ;;
2070. --disable-everything)
2071. map 'eval unset \${$(toupper ${v%s})\_LIST}' $COMPONENT\_LIST
2072. ;;
2073. --enable-\*=\*|--disable-\*=\*)
2074. eval $(echo "${opt%%=\*}" | sed 's/--/action=/;s/-/ thing=/')
2075. is\_in "${thing}s" $COMPONENT\_LIST || die\_unknown "$opt"
2076. eval list=\$$(toupper $thing)\_LIST
2077. name=$(echo "${optval}" | sed "s/,/\_${thing}|/g")\_${thing}
2078. $action $(filter "$name" $list)
2079. ;;
2080. --enable-?\*|--disable-?\*)
2081. eval $(echo "$opt" | sed 's/--/action=/;s/-/ option=/;s/-/\_/g')
2082. if is\_in $option $COMPONENT\_LIST; then
2083. test $action = disable && action=unset
2084. eval $action \$$(toupper ${option%s})\_LIST
2085. elif is\_in $option $CMDLINE\_SELECT; then
2086. $action $option
2087. else
2088. die\_unknown $opt
2089. fi
2090. ;;
2091. --list-\*)
2092. NAME="${opt#--list-}"
2093. is\_in $NAME $COMPONENT\_LIST || die\_unknown $opt
2094. NAME=${NAME%s}
2095. eval show\_list $NAME \$$(toupper $NAME)\_LIST
2096. ;;
2097. --help|-h) show\_help
2098. ;;
2099. \*)
2100. #% 就是从右边开始删除符合条件的字符串（符合条件的最短字符串）
2101. #%%是删除符合条件的最长的字符串
2103. #删除“=”右边的内容
2104. optname="${opt%%=\*}"
2105. #删除左边的“--”
2106. optname="${optname#--}"
2107. optname=$(echo "$optname" | sed 's/-/\_/g')
2108. #看看是否在opt列表中，不在的话就会返回错误
2109. if is\_in $optname $CMDLINE\_SET; then
2110. eval $optname='$optval'
2111. elif is\_in $optname $CMDLINE\_APPEND; then
2112. append $optname "$optval"
2113. else
2114. die\_unknown $opt
2115. fi
2116. ;;
2117. esac
2118. done
2120. disabled logging && logfile=/dev/null
2122. echo "# $0 $FFMPEG\_CONFIGURATION" > $logfile
2123. set >> $logfile
2125. test -n "$cross\_prefix" && enable cross\_compile
2127. if enabled cross\_compile; then
2128. test -n "$arch" && test -n "$target\_os" ||
2129. die "Must specify target arch and OS when cross-compiling"
2130. fi
2132. set\_default arch target\_os postproc\_version
2134. # Check if we should build alternative libpostproc version instead of current
2135. if   test "$postproc\_version" = $ALT\_PP\_VER; then
2136. LIBPOSTPROC\_VERSION=$ALT\_PP\_VER
2137. LIBPOSTPROC\_VERSION\_MAJOR=$ALT\_PP\_VER\_MAJOR
2138. LIBPOSTPROC\_VERSION\_MINOR=$ALT\_PP\_VER\_MINOR
2139. LIBPOSTPROC\_VERSION\_MICRO=$ALT\_PP\_VER\_MICRO
2140. elif test "$postproc\_version" != current; then
2141. die "Invalid argument to --postproc-version. See --help output."
2142. fi
2144. ar\_default="${cross\_prefix}${ar\_default}"
2145. cc\_default="${cross\_prefix}${cc\_default}"
2146. cxx\_default="${cross\_prefix}${cxx\_default}"
2147. nm\_default="${cross\_prefix}${nm\_default}"
2148. pkg\_config\_default="${cross\_prefix}${pkg\_config\_default}"
2149. ranlib="${cross\_prefix}${ranlib}"
2150. strip\_default="${cross\_prefix}${strip\_default}"
2152. sysinclude\_default="${sysroot}/usr/include"
2154. set\_default cc cxx nm pkg\_config strip sysinclude yasmexe
2155. enabled cross\_compile || host\_cc\_default=$cc
2156. set\_default host\_cc
2158. if ! $pkg\_config --version >/dev/null 2>&1; then
2159. warn "$pkg\_config not found, library detection may fail."
2160. pkg\_config=false
2161. fi
2163. exesuf() {
2164. case $1 in
2165. mingw32\*|cygwin\*|\*-dos|freedos|opendos|os/2\*|symbian) echo .exe ;;
2166. esac
2167. }
2169. EXESUF=$(exesuf $target\_os)
2170. HOSTEXESUF=$(exesuf $host\_os)
2172. # set temporary file name
2173. : ${TMPDIR:=$TEMPDIR}
2174. : ${TMPDIR:=$TMP}
2175. : ${TMPDIR:=/tmp}
2177. if ! check\_cmd mktemp -u XXXXXX; then
2178. # simple replacement for missing mktemp
2179. # NOT SAFE FOR GENERAL USE
2180. mktemp(){
2181. echo "${2%%XXX\*}.${HOSTNAME}.${UID}.$$"
2182. }
2183. fi
2184. #生成临时文件
2185. #${2}为该文件的后缀
2186. tmpfile(){
2187. tmp=$(mktemp -u "${TMPDIR}/ffconf.XXXXXXXX")$2 &&
2188. (set -C; exec > $tmp) 2>/dev/null ||
2189. die "Unable to create temporary file in $TMPDIR."
2190. append TMPFILES $tmp
2191. eval $1=$tmp
2192. }
2194. trap 'rm -f -- $TMPFILES' EXIT
2195. #各种临时文件
2196. tmpfile TMPASM .asm
2197. tmpfile TMPC   .c
2198. tmpfile TMPCPP .cpp
2199. tmpfile TMPE   $EXESUF
2200. tmpfile TMPH   .h
2201. tmpfile TMPO   .o
2202. tmpfile TMPS   .S
2203. tmpfile TMPSH  .sh
2204. tmpfile TMPV   .ver
2206. unset -f mktemp
2208. chmod +x $TMPE
2210. # make sure we can execute files in $TMPDIR
2211. cat > $TMPSH 2>> $logfile <<EOF
2212. #! /bin/sh
2213. EOF
2214. chmod +x $TMPSH >> $logfile 2>&1
2215. if ! $TMPSH >> $logfile 2>&1; then
2216. cat <<EOF
2217. Unable to create and execute files in $TMPDIR.  Set the TMPDIR environment
2218. variable to another directory and make sure that it is not mounted noexec.
2219. EOF
2220. die "Sanity test failed."
2221. fi
2223. filter\_asflags=echo
2224. filter\_cflags=echo
2225. filter\_cppflags=echo
2226. #检查编译器
2227. if   $cc -v 2>&1 | grep -q '^gcc.\*LLVM'; then
2228. cc\_type=llvm\_gcc
2229. cc\_version=\_\_VERSION\_\_
2230. gcc\_extra\_ver=$(expr "$($cc --version | head -n1)" : '.\*(.∗)')
2231. cc\_ident="llvm-gcc $($cc -dumpversion) $gcc\_extra\_ver"
2232. CC\_DEPFLAGS='-MMD -MF $(@:.o=.d) -MT $@'
2233. AS\_DEPFLAGS='-MMD -MF $(@:.o=.d) -MT $@'
2234. speed\_cflags='-O3'
2235. size\_cflags='-Os'
2236. elif $cc -v 2>&1 | grep -qi ^gcc; then
2237. cc\_type=gcc
2238. cc\_version=\_\_VERSION\_\_
2239. gcc\_version=$($cc --version | head -n1)
2240. gcc\_basever=$($cc -dumpversion)
2241. gcc\_pkg\_ver=$(expr "$gcc\_version" : '[^ ]\* ([)]∗)')
2242. gcc\_ext\_ver=$(expr "$gcc\_version" : ".\*$gcc\_pkg\_ver $gcc\_basever \\(.\*\\)")
2243. cc\_ident=$(cleanws "gcc $gcc\_basever $gcc\_pkg\_ver $gcc\_ext\_ver")
2244. if ! $cc -dumpversion | grep -q '^2\.'; then
2245. CC\_DEPFLAGS='-MMD -MF $(@:.o=.d) -MT $@'
2246. AS\_DEPFLAGS='-MMD -MF $(@:.o=.d) -MT $@'
2247. fi
2248. speed\_cflags='-O3'
2249. size\_cflags='-Os'
2250. elif $cc --version 2>/dev/null | grep -q Intel; then
2251. cc\_type=icc
2252. cc\_version="AV\_STRINGIFY(\_\_INTEL\_COMPILER)"
2253. cc\_ident=$($cc --version | head -n1)
2254. icc\_version=$($cc -dumpversion)
2255. CC\_DEPFLAGS='-MMD'
2256. AS\_DEPFLAGS='-MMD'
2257. speed\_cflags='-O3'
2258. size\_cflags='-Os'
2259. noopt\_cflags='-O1'
2260. elif $cc -v 2>&1 | grep -q xlc; then
2261. cc\_type=xlc
2262. cc\_version="AV\_STRINGIFY(\_\_IBMC\_\_)"
2263. cc\_ident=$($cc -qversion 2>/dev/null | head -n1)
2264. speed\_cflags='-O5'
2265. size\_cflags='-O5 -qcompact'
2266. elif $cc -V 2>/dev/null | grep -q Compaq; then
2267. cc\_type=ccc
2268. cc\_version="AV\_STRINGIFY(\_\_DECC\_VER)"
2269. cc\_ident=$($cc -V | head -n1 | cut -d' ' -f1-3)
2270. DEPFLAGS='$(CPPFLAGS) $(CFLAGS) -M'
2271. debuglevel=3
2272. add\_ldflags -Wl,-z,now # calls to libots crash without this
2273. speed\_cflags='-fast'
2274. size\_cflags='-O1'
2275. elif $cc --vsn 2>/dev/null | grep -q "ARM C/C++ Compiler"; then
2276. test -d "$sysroot" || die "No valid sysroot specified."
2277. cc\_type=armcc
2278. cc\_version="AV\_STRINGIFY(\_\_ARMCC\_VERSION)"
2279. cc\_ident=$($cc --vsn | head -n1)
2280. armcc\_conf="$PWD/armcc.conf"
2281. $cc --arm\_linux\_configure                 \
2282. --arm\_linux\_config\_file="$armcc\_conf" \
2283. --configure\_sysroot="$sysroot"        \
2284. --configure\_cpp\_headers="$sysinclude" >>$logfile 2>&1 ||
2285. die "Error creating armcc configuration file."
2286. $cc --vsn | grep -q RVCT && armcc\_opt=rvct || armcc\_opt=armcc
2287. cc="$cc --arm\_linux\_config\_file=$armcc\_conf --translate\_gcc"
2288. as\_default="${cross\_prefix}gcc"
2289. CC\_DEPFLAGS='-MMD'
2290. AS\_DEPFLAGS='-MMD'
2291. speed\_cflags='-O3'
2292. size\_cflags='-Os'
2293. filter\_asflags="filter\_out -W${armcc\_opt}\*"
2294. elif $cc -version 2>/dev/null | grep -q TMS470; then
2295. cc\_type=tms470
2296. cc\_version="AV\_STRINGIFY(\_\_TI\_COMPILER\_VERSION\_\_)"
2297. cc\_ident=$($cc -version | head -n1 | tr -s ' ')
2298. cc="$cc --gcc --abi=eabi -eo=.o -mc -me"
2299. CC\_O='-fr=$(@D)'
2300. as\_default="${cross\_prefix}gcc"
2301. ld\_default="${cross\_prefix}gcc"
2302. TMPO=$(basename $TMPC .c).o
2303. append TMPFILES $TMPO
2304. add\_cflags -D\_\_gnuc\_va\_list=va\_list -D\_\_USER\_LABEL\_PREFIX\_\_=
2305. CC\_DEPFLAGS='-ppa -ppd=$(@:.o=.d)'
2306. AS\_DEPFLAGS='-MMD'
2307. speed\_cflags='-O3 -mf=5'
2308. size\_cflags='-O3 -mf=2'
2309. filter\_cflags=tms470\_flags
2310. tms470\_flags(){
2311. for flag; do
2312. case $flag in
2313. -march=\*|-mcpu=\*)
2314. case "${flag#\*=}" in
2315. armv7-a|cortex-a\*)      echo -mv=7a8 ;;
2316. armv7-r|cortex-r\*)      echo -mv=7r4 ;;
2317. armv7-m|cortex-m\*)      echo -mv=7m3 ;;
2318. armv6\*|arm11\*)          echo -mv=6   ;;
2319. armv5\*e|arm[79]\*e\*|arm9[24]6\*|arm96\*|arm102[26])
2320. echo -mv=5e  ;;
2321. armv4\*|arm7\*|arm9[24]\*) echo -mv=4   ;;
2322. esac
2323. ;;
2324. -mfpu=neon)     echo --float\_support=vfpv3 --neon ;;
2325. -mfpu=vfp)      echo --float\_support=vfpv2        ;;
2326. -mfpu=vfpv3)    echo --float\_support=vfpv3        ;;
2327. -msoft-float)   echo --float\_support=vfplib       ;;
2328. -O[0-3]|-mf=\*)  echo $flag                        ;;
2329. -g)             echo -g -mn                       ;;
2330. -pds=\*)         echo $flag                        ;;
2331. esac
2332. done
2333. }
2334. elif $cc -v 2>&1 | grep -q clang; then
2335. cc\_type=clang
2336. $cc -dM -E $TMPC | grep -q \_\_clang\_version\_\_ &&
2337. cc\_version=\_\_clang\_version\_\_ || cc\_version=\_\_VERSION\_\_
2338. cc\_ident=$($cc --version | head -n1)
2339. CC\_DEPFLAGS='-MMD'
2340. AS\_DEPFLAGS='-MMD'
2341. speed\_cflags='-O3'
2342. size\_cflags='-Os'
2343. elif $cc -V 2>&1 | grep -q Sun; then
2344. cc\_type=suncc
2345. cc\_version="AV\_STRINGIFY(\_\_SUNPRO\_C)"
2346. cc\_ident=$($cc -V 2>&1 | head -n1 | cut -d' ' -f 2-)
2347. DEPEND\_CMD='$(DEPCC) $(DEPFLAGS) $< | sed -e "1s,^.\*: ,$@: ," -e "\$$!s,\$$, \\\," -e "1!s,^.\*: , ," > $(@:.o=.d)'
2348. DEPFLAGS='$(CPPFLAGS) $(CFLAGS) -xM1'
2349. add\_ldflags -xc99
2350. speed\_cflags='-O5'
2351. size\_cflags='-O5 -xspace'
2352. filter\_cflags=suncc\_flags
2353. suncc\_flags(){
2354. for flag; do
2355. case $flag in
2356. -march=\*|-mcpu=\*)
2357. case "${flag#\*=}" in
2358. native)                   echo -xtarget=native       ;;
2359. v9|niagara)               echo -xarch=sparc          ;;
2360. ultrasparc)               echo -xarch=sparcvis       ;;
2361. ultrasparc3|niagara2)     echo -xarch=sparcvis2      ;;
2362. i586|pentium)             echo -xchip=pentium        ;;
2363. i686|pentiumpro|pentium2) echo -xtarget=pentium\_pro  ;;
2364. pentium3\*|c3-2)           echo -xtarget=pentium3     ;;
2365. pentium-m)          echo -xarch=sse2 -xchip=pentium3 ;;
2366. pentium4\*)          echo -xtarget=pentium4           ;;
2367. prescott|nocona)    echo -xarch=sse3 -xchip=pentium4 ;;
2368. \*-sse3)             echo -xarch=sse3                 ;;
2369. core2)              echo -xarch=ssse3 -xchip=core2   ;;
2370. amdfam10|barcelona)       echo -xarch=sse4\_1         ;;
2371. athlon-4|athlon-[mx]p)    echo -xarch=ssea           ;;
2372. k8|opteron|athlon64|athlon-fx)
2373. echo -xarch=sse2a          ;;
2374. athlon\*)                  echo -xarch=pentium\_proa   ;;
2375. esac
2376. ;;
2377. -std=c99)             echo -xc99              ;;
2378. -fomit-frame-pointer) echo -xregs=frameptr    ;;
2379. -fPIC)                echo -KPIC -xcode=pic32 ;;
2380. -W\*,\*)                echo $flag              ;;
2381. -f\*-\*|-W\*)                                    ;;
2382. \*)                    echo $flag              ;;
2383. esac
2384. done
2385. }
2386. elif $cc -v 2>&1 | grep -q 'PathScale\|Path64'; then
2387. cc\_type=pathscale
2388. cc\_version=\_\_PATHSCALE\_\_
2389. cc\_ident=$($cc -v 2>&1 | head -n1 | tr -d :)
2390. CC\_DEPFLAGS='-MMD -MF $(@:.o=.d) -MT $@'
2391. AS\_DEPFLAGS='-MMD -MF $(@:.o=.d) -MT $@'
2392. speed\_cflags='-O2'
2393. size\_cflags='-Os'
2394. filter\_cflags='filter\_out -Wdisabled-optimization'
2395. elif $cc -v 2>&1 | grep -q Open64; then
2396. cc\_type=open64
2397. cc\_version=\_\_OPEN64\_\_
2398. cc\_ident=$($cc -v 2>&1 | head -n1 | tr -d :)
2399. CC\_DEPFLAGS='-MMD -MF $(@:.o=.d) -MT $@'
2400. AS\_DEPFLAGS='-MMD -MF $(@:.o=.d) -MT $@'
2401. speed\_cflags='-O2'
2402. size\_cflags='-Os'
2403. filter\_cflags='filter\_out -Wdisabled-optimization|-Wtype-limits|-fno-signed-zeros'
2404. fi
2406. test -n "$cc\_type" && enable $cc\_type ||
2407. warn "Unknown C compiler $cc, unable to select optimal CFLAGS"
2409. : ${as\_default:=$cc}
2410. : ${dep\_cc\_default:=$cc}
2411. : ${ld\_default:=$cc}
2412. set\_default ar as dep\_cc ld
2414. test -n "$CC\_DEPFLAGS" || CCDEP=$DEPEND\_CMD
2415. test -n "$CXX\_DEPFLAGS" || CXXDEP=$DEPEND\_CMD
2416. test -n "$AS\_DEPFLAGS" || ASDEP=$DEPEND\_CMD
2418. add\_cflags $extra\_cflags
2419. add\_cxxflags $extra\_cxxflags
2420. add\_asflags $extra\_cflags
2422. if test -n "$sysroot"; then
2423. case "$cc\_type" in
2424. gcc|llvm\_gcc|clang)
2425. add\_cppflags --sysroot="$sysroot"
2426. add\_ldflags --sysroot="$sysroot"
2427. ;;
2428. tms470)
2429. add\_cppflags -I"$sysinclude"
2430. add\_ldflags  --sysroot="$sysroot"
2431. ;;
2432. esac
2433. fi
2435. if test "$cpu" = host; then
2436. enabled cross\_compile && die "--cpu=host makes no sense when cross-compiling."
2438. case "$cc\_type" in
2439. gcc|llvm\_gcc)
2440. check\_native(){
2441. $cc $1=native -v -c -o $TMPO $TMPC >$TMPE 2>&1 || return
2442. sed -n "/cc1.\*$1=/{
2443. s/.\*$1=\\([^ ]\*\\).\*/\\1/
2444. p
2445. q
2446. }" $TMPE
2447. }
2448. cpu=$(check\_native -march || check\_native -mcpu)
2449. ;;
2450. esac
2452. test "${cpu:-host}" = host && die "--cpu=host not supported with compiler $cc"
2453. fi
2455. # Deal with common $arch aliases
2456. case "$arch" in
2457. arm\*|iPad\*)
2458. arch="arm"
2459. ;;
2460. mips|mipsel|IP\*)
2461. arch="mips"
2462. ;;
2463. mips64\*)
2464. arch="mips"
2465. subarch="mips64"
2466. ;;
2467. parisc|hppa)
2468. arch="parisc"
2469. ;;
2470. parisc64|hppa64)
2471. arch="parisc"
2472. subarch="parisc64"
2473. ;;
2474. "Power Macintosh"|ppc|powerpc|ppc64|powerpc64)
2475. arch="ppc"
2476. ;;
2477. s390|s390x)
2478. arch="s390"
2479. ;;
2480. sh4|sh)
2481. arch="sh4"
2482. ;;
2483. sun4u|sparc64)
2484. arch="sparc"
2485. subarch="sparc64"
2486. ;;
2487. i[3-6]86|i86pc|BePC|x86pc|x86\_64|x86\_32|amd64)
2488. arch="x86"
2489. ;;
2490. esac
2492. is\_in $arch $ARCH\_LIST || warn "unknown architecture $arch"
2493. enable $arch
2495. # Add processor-specific flags
2496. #根据CPU类型的不同，进行cpuflag的设置
2497. if test "$cpu" = generic; then
2498. : do nothing
2499. elif enabled ppc; then
2501. case $(tolower $cpu) in
2502. 601|ppc601|powerpc601)
2503. cpuflags="-mcpu=601"
2504. disable altivec
2505. ;;
2506. 603\*|ppc603\*|powerpc603\*)
2507. cpuflags="-mcpu=603"
2508. disable altivec
2509. ;;
2510. 604\*|ppc604\*|powerpc604\*)
2511. cpuflags="-mcpu=604"
2512. disable altivec
2513. ;;
2514. g3|75\*|ppc75\*|powerpc75\*)
2515. cpuflags="-mcpu=750 -mpowerpc-gfxopt"
2516. disable altivec
2517. ;;
2518. g4|745\*|ppc745\*|powerpc745\*)
2519. cpuflags="-mcpu=7450 -mpowerpc-gfxopt"
2520. ;;
2521. 74\*|ppc74\*|powerpc74\*)
2522. cpuflags="-mcpu=7400 -mpowerpc-gfxopt"
2523. ;;
2524. g5|970|ppc970|powerpc970|power4\*)
2525. cpuflags="-mcpu=970 -mpowerpc-gfxopt -mpowerpc64"
2526. ;;
2527. cell)
2528. cpuflags="-mcpu=cell"
2529. enable ldbrx
2530. ;;
2531. e500v2)
2532. cpuflags="-mcpu=8548 -mhard-float -mfloat-gprs=double"
2533. disable altivec
2534. ;;
2535. e500)
2536. cpuflags="-mcpu=8540 -mhard-float"
2537. disable altivec
2538. ;;
2539. esac
2540. #X86架构
2541. elif enabled x86; then
2543. case $cpu in
2544. i[345]86|pentium)
2545. cpuflags="-march=$cpu"
2546. disable mmx
2547. ;;
2548. # targets that do NOT support conditional mov (cmov)
2549. pentium-mmx|k6|k6-[23]|winchip-c6|winchip2|c3)
2550. cpuflags="-march=$cpu"
2551. disable cmov
2552. ;;
2553. # targets that do support conditional mov (cmov)
2554. i686|pentiumpro|pentium[23]|pentium-m|athlon|athlon-tbird|athlon-4|athlon-[mx]p|athlon64\*|k8\*|opteron\*|athlon-fx|core2|amdfam10|barcelona|atom)
2555. cpuflags="-march=$cpu"
2556. enable cmov
2557. enable fast\_cmov
2558. ;;
2559. # targets that do support conditional mov but on which it's slow
2560. pentium4|pentium4m|prescott|nocona)
2561. cpuflags="-march=$cpu"
2562. enable cmov
2563. disable fast\_cmov
2564. ;;
2565. esac
2567. elif enabled sparc; then
2569. case $cpu in
2570. niagara)
2571. cpuflags="-mcpu=$cpu"
2572. disable vis
2573. ;;
2574. sparc64)
2575. cpuflags="-mcpu=v9"
2576. ;;
2577. esac
2578. #ARM架构
2579. elif enabled arm; then
2581. case $cpu in
2582. armv\*)
2583. cpuflags="-march=$cpu"
2584. subarch=$(echo $cpu | sed 's/[^a-z0-9]//g')
2585. ;;
2586. \*)
2587. cpuflags="-mcpu=$cpu"
2588. case $cpu in
2589. cortex-a\*)                               subarch=armv7a  ;;
2590. cortex-r\*)                               subarch=armv7r  ;;
2591. cortex-m\*)                 enable thumb; subarch=armv7m  ;;
2592. arm11\*)                                  subarch=armv6   ;;
2593. arm[79]\*e\*|arm9[24]6\*|arm96\*|arm102[26]) subarch=armv5te ;;
2594. armv4\*|arm7\*|arm9[24]\*)                  subarch=armv4   ;;
2595. esac
2596. ;;
2597. esac
2599. elif enabled alpha; then
2601. enabled ccc && cpuflags="-arch $cpu" || cpuflags="-mcpu=$cpu"
2603. elif enabled bfin; then
2605. cpuflags="-mcpu=$cpu"
2607. elif enabled mips; then
2609. cpuflags="-march=$cpu"
2611. elif enabled avr32; then
2613. case $cpu in
2614. ap7[02]0[0-2])
2615. subarch="avr32\_ap"
2616. cpuflags="-mpart=$cpu"
2617. ;;
2618. ap)
2619. subarch="avr32\_ap"
2620. cpuflags="-march=$cpu"
2621. ;;
2622. uc3[ab]\*)
2623. subarch="avr32\_uc"
2624. cpuflags="-mcpu=$cpu"
2625. ;;
2626. uc)
2627. subarch="avr32\_uc"
2628. cpuflags="-march=$cpu"
2629. ;;
2630. esac
2632. fi
2634. add\_cflags $cpuflags
2635. add\_asflags $cpuflags
2637. # compiler sanity check
2638. # 用个简单的main()检查能不能用
2639. check\_exec <<EOF
2640. int main(void){ return 0; }
2641. EOF
2642. if test "$?" != 0; then
2643. echo "$cc is unable to create an executable file."
2644. if test -z "$cross\_prefix" && ! enabled cross\_compile ; then
2645. echo "If $cc is a cross-compiler, use the --enable-cross-compile option."
2646. echo "Only do this if you know what cross compiling means."
2647. fi
2648. die "C compiler test failed."
2649. fi
2651. add\_cppflags -D\_ISOC99\_SOURCE
2652. add\_cxxflags -D\_\_STDC\_CONSTANT\_MACROS
2653. check\_cflags -std=c99
2654. check\_cc -D\_FILE\_OFFSET\_BITS=64 <<EOF && add\_cppflags -D\_FILE\_OFFSET\_BITS=64
2655. #include <stdlib.h>
2656. EOF
2657. check\_cc -D\_LARGEFILE\_SOURCE <<EOF && add\_cppflags -D\_LARGEFILE\_SOURCE
2658. #include <stdlib.h>
2659. EOF
2661. check\_host\_cflags -std=c99
2662. check\_host\_cflags -Wall
2663. #32位系统指针变量占用32bit（4Byte）数据（32位寻址）
2664. #64位系统指针变量占用64bit（4Byte）数据（64位寻址）
2665. case "$arch" in
2666. alpha|ia64|mips|parisc|sparc)
2667. spic=$shared
2668. ;;
2669. x86)
2670. subarch="x86\_32"
2671. check\_cc <<EOF && subarch="x86\_64"
2672. int test[(int)sizeof(char\*) - 7];
2673. EOF
2674. if test "$subarch" = "x86\_64"; then
2675. spic=$shared
2676. fi
2677. ;;
2678. ppc)
2679. check\_cc <<EOF && subarch="ppc64"
2680. int test[(int)sizeof(char\*) - 7];
2681. EOF
2682. ;;
2683. esac
2685. enable $subarch
2686. enabled spic && enable pic
2688. # 不同的操作系统 OS specific
2689. # target-os参数
2690. case $target\_os in
2691. haiku)
2692. prefix\_default="/boot/common"
2693. network\_extralibs="-lnetwork"
2694. host\_libs=
2695. ;;
2696. sunos)
2697. FFSERVERLDFLAGS=""
2698. SHFLAGS='-shared -Wl,-h,$$(@F)'
2699. enabled x86 && SHFLAGS="-mimpure-text $SHFLAGS"
2700. network\_extralibs="-lsocket -lnsl"
2701. add\_cppflags -D\_\_EXTENSIONS\_\_
2702. # When using suncc to build, the Solaris linker will mark
2703. # an executable with each instruction set encountered by
2704. # the Solaris assembler.  As our libraries contain their own
2705. # guards for processor-specific code, instead suppress
2706. # generation of the HWCAPS ELF section on Solaris x86 only.
2707. enabled\_all suncc x86 && echo "hwcap\_1 = OVERRIDE;" > mapfile && add\_ldflags -Wl,-M,mapfile
2708. nm\_opts='-P -g'
2709. ;;
2710. netbsd)
2711. disable symver
2712. oss\_indev\_extralibs="-lossaudio"
2713. oss\_outdev\_extralibs="-lossaudio"
2714. ;;
2715. openbsd)
2716. enable malloc\_aligned
2717. # On OpenBSD 4.5. the compiler does not use PIC unless
2718. # explicitly using -fPIC. FFmpeg builds fine without PIC,
2719. # however the generated executable will not do anything
2720. # (simply quits with exit-code 1, no crash, no output).
2721. # Thus explicitly enable PIC here.
2722. enable pic
2723. disable symver
2724. SHFLAGS='-shared'
2725. oss\_indev\_extralibs="-lossaudio"
2726. oss\_outdev\_extralibs="-lossaudio"
2727. ;;
2728. dragonfly)
2729. enable malloc\_aligned
2730. disable symver
2731. ;;
2732. freebsd)
2733. enable malloc\_aligned
2734. ;;
2735. bsd/os)
2736. add\_extralibs -lpoll -lgnugetopt
2737. strip="strip -d"
2738. ;;
2739. #苹果Mac操作系统
2740. darwin)
2741. enable malloc\_aligned
2742. #以前见过这个
2743. gas="gas-preprocessor.pl $cc"
2744. enabled ppc && add\_asflags -force\_cpusubtype\_ALL
2745. SHFLAGS='-dynamiclib -Wl,-single\_module -Wl,-install\_name,$(SHLIBDIR)/$(SLIBNAME),-current\_version,$(LIBVERSION),-compatibility\_version,$(LIBMAJOR)'
2746. enabled x86\_32 && append SHFLAGS -Wl,-read\_only\_relocs,suppress
2747. strip="${strip} -x"
2748. add\_ldflags -Wl,-dynamic,-search\_paths\_first
2749. #Mac下的动态库
2750. SLIBSUF=".dylib"
2751. SLIBNAME\_WITH\_VERSION='$(SLIBPREF)$(FULLNAME).$(LIBVERSION)$(SLIBSUF)'
2752. SLIBNAME\_WITH\_MAJOR='$(SLIBPREF)$(FULLNAME).$(LIBMAJOR)$(SLIBSUF)'
2753. FFSERVERLDFLAGS=-Wl,-bind\_at\_load
2754. #macho目标文件格式
2755. objformat="macho"
2756. enabled x86\_64 && objformat="macho64"
2757. enabled\_any pic shared ||
2758. { check\_cflags -mdynamic-no-pic && add\_asflags -mdynamic-no-pic; }
2759. ;;
2760. #MinGW
2761. mingw32\*)
2762. if test $target\_os = "mingw32ce"; then
2763. disable network
2764. else
2765. target\_os=mingw32
2766. fi
2767. LIBTARGET=i386
2768. if enabled x86\_64; then
2769. enable malloc\_aligned
2770. LIBTARGET=x64
2771. elif enabled arm; then
2772. LIBTARGET=arm-wince
2773. fi
2774. shlibdir\_default="$bindir\_default"
2775. SLIBPREF=""
2776. # Windows下动态库后缀
2777. SLIBSUF=".dll"
2778. SLIBNAME\_WITH\_VERSION='$(SLIBPREF)$(FULLNAME)-$(LIBVERSION)$(SLIBSUF)'
2779. SLIBNAME\_WITH\_MAJOR='$(SLIBPREF)$(FULLNAME)-$(LIBMAJOR)$(SLIBSUF)'
2780. # 借助lib.exe生成导出库lib
2781. SLIB\_EXTRA\_CMD='-lib.exe /machine:$(LIBTARGET) /def:$$(@:$(SLIBSUF)=.def) /out:$(SUBDIR)$(SLIBNAME:$(SLIBSUF)=.lib)'
2782. SLIB\_INSTALL\_NAME='$(SLIBNAME\_WITH\_MAJOR)'
2783. SLIB\_INSTALL\_LINKS=
2784. #额外的lib导入库
2785. SLIB\_INSTALL\_EXTRA\_SHLIB='$(SLIBNAME:$(SLIBSUF)=.lib)'
2786. #额外的
2787. SLIB\_INSTALL\_EXTRA\_LIB='lib$(SLIBNAME:$(SLIBSUF)=.dll.a) $(SLIBNAME\_WITH\_MAJOR:$(SLIBSUF)=.def)'
2788. SHFLAGS='-shared -Wl,--output-def,$$(@:$(SLIBSUF)=.def) -Wl,--out-implib,$(SUBDIR)lib$(SLIBNAME:$(SLIBSUF)=.dll.a) -Wl,--enable-runtime-pseudo-reloc -Wl,--enable-auto-image-base'
2789. # windows PE格式
2790. objformat="win32"
2791. enable dos\_paths
2792. check\_cflags -fno-common
2793. check\_cpp\_condition \_mingw.h "defined (\_\_MINGW64\_VERSION\_MAJOR) || (\_\_MINGW32\_MAJOR\_VERSION > 3) \
2794. || (\_\_MINGW32\_MAJOR\_VERSION == 3 && \_\_MINGW32\_MINOR\_VERSION >= 15)" ||
2795. die "ERROR: MinGW runtime version must be >= 3.15."
2796. add\_cppflags -U\_\_STRICT\_ANSI\_\_
2797. ;;
2798. #Cygwin
2799. cygwin\*)
2800. target\_os=cygwin
2801. shlibdir\_default="$bindir\_default"
2802. SLIBPREF="cyg"
2803. SLIBSUF=".dll"
2804. SLIBNAME\_WITH\_VERSION='$(SLIBPREF)$(FULLNAME)-$(LIBVERSION)$(SLIBSUF)'
2805. SLIBNAME\_WITH\_MAJOR='$(SLIBPREF)$(FULLNAME)-$(LIBMAJOR)$(SLIBSUF)'
2806. SHFLAGS='-shared -Wl,--enable-auto-image-base'
2807. objformat="win32"
2808. enable dos\_paths
2809. check\_cflags -fno-common
2810. add\_cppflags -U\_\_STRICT\_ANSI\_\_
2811. ;;
2812. \*-dos|freedos|opendos)
2813. network\_extralibs="-lsocket"
2814. objformat="coff"
2815. enable dos\_paths
2816. add\_cppflags -U\_\_STRICT\_ANSI\_\_
2817. ;;
2818. #Linux操作系统
2819. linux)
2820. add\_cppflags -D\_POSIX\_C\_SOURCE=200112 -D\_XOPEN\_SOURCE=600
2821. enable dv1394
2822. ;;
2823. irix\*)
2824. target\_os=irix
2825. ranlib="echo ignoring ranlib"
2826. ;;
2827. os/2\*)
2828. strip="lxlite -CS"
2829. ln\_s="cp -f"
2830. objformat="aout"
2831. add\_cppflags -D\_GNU\_SOURCE
2832. add\_ldflags -Zomf -Zbin-files -Zargs-wild -Zmap
2833. SHFLAGS='$(SUBDIR)$(NAME).def -Zdll -Zomf'
2834. FFSERVERLDFLAGS=""
2835. LIBSUF="\_s.a"
2836. SLIBPREF=""
2837. SLIBSUF=".dll"
2838. SLIBNAME\_WITH\_VERSION='$(SLIBPREF)$(NAME)-$(LIBVERSION)$(SLIBSUF)'
2839. SLIBNAME\_WITH\_MAJOR='$(SLIBPREF)$(shell echo $(NAME) | cut -c1-6)$(LIBMAJOR)$(SLIBSUF)'
2840. SLIB\_CREATE\_DEF\_CMD='echo LIBRARY $(SLIBNAME\_WITH\_MAJOR) INITINSTANCE TERMINSTANCE > $(SUBDIR)$(NAME).def; \
2841. echo PROTMODE >> $(SUBDIR)$(NAME).def; \
2842. echo CODE PRELOAD MOVEABLE DISCARDABLE >> $(SUBDIR)$(NAME).def; \
2843. echo DATA PRELOAD MOVEABLE MULTIPLE NONSHARED >> $(SUBDIR)$(NAME).def; \
2844. echo EXPORTS >> $(SUBDIR)$(NAME).def; \
2845. emxexp -o $(OBJS) >> $(SUBDIR)$(NAME).def'
2846. SLIB\_EXTRA\_CMD='emximp -o $(SUBDIR)$(LIBPREF)$(NAME)\_dll.a $(SUBDIR)$(NAME).def; \
2847. emximp -o $(SUBDIR)$(LIBPREF)$(NAME)\_dll.lib $(SUBDIR)$(NAME).def;'
2848. SLIB\_INSTALL\_EXTRA\_LIB='$(LIBPREF)$(NAME)\_dll.a $(LIBPREF)$(NAME)\_dll.lib'
2849. enable dos\_paths
2850. enable\_weak os2threads
2851. ;;
2852. gnu/kfreebsd)
2853. add\_cppflags -D\_POSIX\_C\_SOURCE=200112 -D\_XOPEN\_SOURCE=600 -D\_BSD\_SOURCE
2854. ;;
2855. gnu)
2856. add\_cppflags -D\_POSIX\_C\_SOURCE=200112 -D\_XOPEN\_SOURCE=600
2857. ;;
2858. qnx)
2859. add\_cppflags -D\_QNX\_SOURCE
2860. network\_extralibs="-lsocket"
2861. ;;
2862. symbian)
2863. SLIBSUF=".dll"
2864. enable dos\_paths
2865. add\_cflags --include=$sysinclude/gcce/gcce.h -fvisibility=default
2866. add\_cppflags -D\_\_GCCE\_\_ -D\_\_SYMBIAN32\_\_ -DSYMBIAN\_OE\_POSIX\_SIGNALS
2867. add\_ldflags -Wl,--target1-abs,--no-undefined \
2868. -Wl,-Ttext,0x80000,-Tdata,0x1000000 -shared \
2869. -Wl,--entry=\_E32Startup -Wl,-u,\_E32Startup
2870. add\_extralibs -l:eexe.lib -l:usrt2\_2.lib -l:dfpaeabi.dso \
2871. -l:drtaeabi.dso -l:scppnwdl.dso -lsupc++ -lgcc \
2872. -l:libc.dso -l:libm.dso -l:euser.dso -l:libcrt0.lib
2873. ;;
2874. none)
2875. ;;
2876. \*)
2877. die "Unknown OS '$target\_os'."
2878. ;;
2879. esac
2881. echo "config:$arch:$subarch:$cpu:$target\_os:$cc\_ident:$FFMPEG\_CONFIGURATION" >config.fate
2883. check\_cpp\_condition stdlib.h "defined(\_\_PIC\_\_) || defined(\_\_pic\_\_) || defined(PIC)" && enable pic
2885. set\_default $PATHS\_LIST
2887. # we need to build at least one lib type
2888. if ! enabled\_any static shared; then
2889. cat <<EOF
2890. At least one library type must be built.
2891. Specify --enable-static to build the static libraries or --enable-shared to
2892. build the shared libraries as well. To only build the shared libraries specify
2893. --disable-static in addition to --enable-shared.
2894. EOF
2895. exit 1;
2896. fi
2897. #不符合License则立刻结束
2898. die\_license\_disabled() {
2899. enabled $1 || { enabled $2 && die "$2 is $1 and --enable-$1 is not specified."; }
2900. }
2901. #检查License
2902. #GPL
2903. die\_license\_disabled gpl libcdio
2904. die\_license\_disabled gpl libx264
2905. die\_license\_disabled gpl libxavs
2906. die\_license\_disabled gpl libxvid
2907. die\_license\_disabled gpl x11grab
2908. #nonfree
2909. die\_license\_disabled nonfree libaacplus
2910. die\_license\_disabled nonfree libfaac
2911. die\_license\_disabled nonfree openssl
2912. #Version3
2913. die\_license\_disabled version3 libopencore\_amrnb
2914. die\_license\_disabled version3 libopencore\_amrwb
2915. die\_license\_disabled version3 libvo\_aacenc
2916. die\_license\_disabled version3 libvo\_amrwbenc
2918. enabled version3 && { enabled gpl && enable gplv3 || enable lgplv3; }
2920. disabled optimizations || check\_cflags -fomit-frame-pointer
2921. #添加fPIC
2922. enable\_pic() {
2923. enable pic
2924. add\_cppflags -DPIC
2925. add\_cflags   -fPIC
2926. add\_asflags  -fPIC
2927. }
2929. enabled pic && enable\_pic
2931. check\_cc <<EOF || die "Symbol mangling check failed."
2932. int ff\_extern;
2933. EOF
2934. sym=$($nm $nm\_opts $TMPO | awk '/ff\_extern/{ print substr($0, match($0, /[^ \t]\*ff\_extern/)) }')
2935. extern\_prefix=${sym%%ff\_extern\*}
2937. check\_cc <<EOF && enable inline\_asm
2938. void foo(void) { \_\_asm\_\_ volatile ("" ::); }
2939. EOF
2941. \_restrict=
2942. for restrict\_keyword in restrict \_\_restrict\_\_ \_\_restrict; do
2943. check\_cc <<EOF && \_restrict=$restrict\_keyword && break
2944. void foo(char \* $restrict\_keyword p);
2945. EOF
2946. done
2948. check\_cc <<EOF && enable attribute\_packed
2949. struct { int x; } \_\_attribute\_\_((packed)) x;
2950. EOF
2952. check\_cc <<EOF && enable attribute\_may\_alias
2953. union { int x; } \_\_attribute\_\_((may\_alias)) x;
2954. EOF
2956. check\_cc <<EOF || die "endian test failed"
2957. unsigned int endian = 'B' << 24 | 'I' << 16 | 'G' << 8 | 'E';
2958. EOF
2959. od -t x1 $TMPO | grep -q '42 \*49 \*47 \*45' && enable bigendian
2961. if enabled alpha; then
2963. check\_cflags -mieee
2965. elif enabled arm; then
2967. enabled thumb && check\_cflags -mthumb || check\_cflags -marm
2968. nogas=die
2970. if     check\_cpp\_condition stddef.h "defined \_\_ARM\_PCS\_VFP"; then
2971. enable vfp\_args
2972. elif ! check\_cpp\_condition stddef.h "defined \_\_ARM\_PCS || defined \_\_SOFTFP\_\_"; then
2973. case "${cross\_prefix:-$cc}" in
2974. \*hardfloat\*)         enable vfp\_args;   fpabi=vfp ;;
2975. \*) check\_ld "cc" <<EOF && enable vfp\_args && fpabi=vfp || fpabi=soft ;;
2976. \_\_asm\_\_ (".eabi\_attribute 28, 1");
2977. int main(void) { return 0; }
2978. EOF
2979. esac
2980. warn "Compiler does not indicate floating-point ABI, guessing $fpabi."
2981. fi
2983. enabled armv5te && check\_asm armv5te '"qadd r0, r0, r0"'
2984. enabled armv6   && check\_asm armv6   '"sadd16 r0, r0, r0"'
2985. enabled armv6t2 && check\_asm armv6t2 '"movt r0, #0"'
2986. enabled armvfp  && check\_asm armvfp  '"fadds s0, s0, s0"'
2987. enabled iwmmxt  && check\_asm iwmmxt  '"wunpckelub wr6, wr4"'
2988. enabled neon    && check\_asm neon    '"vadd.i16 q0, q0, q0"'
2989. enabled vfpv3   && check\_asm vfpv3   '"vmov.f32 s0, #1.0"'
2991. check\_asm asm\_mod\_y '"vmul.i32 d0, d0, %y0" :: "x"(0)'
2993. enabled\_all armv6t2 shared !pic && enable\_pic
2995. elif enabled mips; then
2997. check\_asm loongson '"dmult.g $1, $2, $3"'
2998. enabled mmi     && check\_asm mmi     '"lq $2, 0($2)"'
3000. elif enabled ppc; then
3002. enable local\_aligned\_8 local\_aligned\_16
3004. check\_asm dcbzl     '"dcbzl 0, %0" :: "r"(0)'
3005. check\_asm ibm\_asm   '"add 0, 0, 0"'
3006. check\_asm ppc4xx    '"maclhw r10, r11, r12"'
3007. check\_asm xform\_asm '"lwzx %1, %y0" :: "Z"(\*(int\*)0), "r"(0)'
3009. # AltiVec flags: The FSF version of GCC differs from the Apple version
3010. if enabled altivec; then
3011. nogas=warn
3012. check\_cflags -maltivec -mabi=altivec &&
3013. { check\_header altivec.h && inc\_altivec\_h="#include <altivec.h>" ; } ||
3014. check\_cflags -faltivec
3016. # check if our compiler supports Motorola AltiVec C API
3017. check\_cc <<EOF || disable altivec
3018. $inc\_altivec\_h
3019. int main(void) {
3020. vector signed int v1, v2, v3;
3021. v1 = vec\_add(v2,v3);
3022. return 0;
3023. }
3024. EOF
3026. # check if our compiler supports braces for vector declarations
3027. check\_cc <<EOF || die "You need a compiler that supports {} in AltiVec vector declarations."
3028. $inc\_altivec\_h
3029. int main (void) { (vector int) {1}; return 0; }
3030. EOF
3031. fi
3033. elif enabled sparc; then
3035. enabled vis && check\_asm vis '"pdist %f0, %f0, %f0"' -mcpu=ultrasparc &&
3036. add\_cflags -mcpu=ultrasparc -mtune=ultrasparc
3038. elif enabled x86; then
3040. enable local\_aligned\_8 local\_aligned\_16
3042. # check whether EBP is available on x86
3043. # As 'i' is stored on the stack, this program will crash
3044. # if the base pointer is used to access it because the
3045. # base pointer is cleared in the inline assembly code.
3046. check\_exec\_crash <<EOF && enable ebp\_available
3047. volatile int i=0;
3048. \_\_asm\_\_ volatile (
3049. "xorl %%ebp, %%ebp"
3050. ::: "%ebp");
3051. return i;
3052. EOF
3054. # check whether EBX is available on x86
3055. check\_asm ebx\_available '""::"b"(0)' &&
3056. check\_asm ebx\_available '"":::"%ebx"'
3058. # check whether xmm clobbers are supported
3059. check\_asm xmm\_clobbers '"":::"%xmm0"'
3061. # check whether binutils is new enough to compile SSSE3/MMX2
3062. enabled ssse3 && check\_asm ssse3 '"pabsw %xmm0, %xmm0"'
3063. enabled mmx2  && check\_asm mmx2  '"pmaxub %mm0, %mm1"'
3065. if ! disabled\_any asm mmx yasm; then
3066. if check\_cmd $yasmexe --version; then
3067. enabled x86\_64 && yasm\_extra="-m amd64"
3068. yasm\_debug="-g dwarf2"
3069. elif check\_cmd nasm -v; then
3070. yasmexe=nasm
3071. yasm\_debug="-g -F dwarf"
3072. enabled x86\_64 && test "$objformat" = elf && objformat=elf64
3073. fi
3075. YASMFLAGS="-f $objformat $yasm\_extra"
3076. enabled pic               && append YASMFLAGS "-DPIC"
3077. test -n "$extern\_prefix"  && append YASMFLAGS "-DPREFIX"
3078. case "$objformat" in
3079. elf\*) enabled debug && append YASMFLAGS $yasm\_debug ;;
3080. esac
3082. check\_yasm "pextrd [eax], xmm0, 1" && enable yasm ||
3083. die "yasm not found, use --disable-yasm for a crippled build"
3084. check\_yasm "vextractf128 xmm0, ymm0, 0" || disable avx
3085. fi
3087. case "$cpu" in
3088. athlon\*|opteron\*|k8\*|pentium|pentium-mmx|prescott|nocona|atom|geode)
3089. disable fast\_clz
3090. ;;
3091. esac
3093. fi
3095. if enabled asm; then
3096. as=${gas:=$as}
3097. check\_asm gnu\_as '".macro m n\n\\n:.int 0\n.endm\nm x"' ||
3098. $nogas "GNU assembler not found, install gas-preprocessor"
3099. fi
3101. check\_ldflags -Wl,--as-needed
3103. if check\_func dlopen; then
3104. ldl=
3105. elif check\_func dlopen -ldl; then
3106. ldl=-ldl
3107. fi
3108. #网络socket
3109. if enabled network; then
3110. check\_type "sys/types.h sys/socket.h" socklen\_t
3111. check\_type netdb.h "struct addrinfo"
3112. check\_type netinet/in.h "struct ipv6\_mreq" -D\_DARWIN\_C\_SOURCE
3113. check\_type netinet/in.h "struct sockaddr\_in6"
3114. check\_type "sys/types.h sys/socket.h" "struct sockaddr\_storage"
3115. check\_struct "sys/types.h sys/socket.h" "struct sockaddr" sa\_len
3116. # Prefer arpa/inet.h over winsock2
3117. if check\_header arpa/inet.h ; then
3118. check\_func closesocket
3119. elif check\_header winsock2.h ; then
3120. check\_func\_headers winsock2.h closesocket -lws2 && \
3121. network\_extralibs="-lws2" || \
3122. { check\_func\_headers winsock2.h closesocket -lws2\_32 && \
3123. network\_extralibs="-lws2\_32"; }
3124. check\_type ws2tcpip.h socklen\_t
3125. check\_type ws2tcpip.h "struct addrinfo"
3126. check\_type ws2tcpip.h "struct ipv6\_mreq"
3127. check\_type ws2tcpip.h "struct sockaddr\_in6"
3128. check\_type ws2tcpip.h "struct sockaddr\_storage"
3129. check\_struct winsock2.h "struct sockaddr" sa\_len
3130. else
3131. disable network
3132. fi
3133. fi
3135. # Solaris has nanosleep in -lrt, OpenSolaris no longer needs that
3136. check\_func nanosleep || { check\_func nanosleep -lrt && add\_extralibs -lrt; }
3137. #检查函数
3138. check\_func  fcntl
3139. check\_func  fork
3140. check\_func  getaddrinfo $network\_extralibs
3141. check\_func  gethrtime
3142. check\_func  getrusage
3143. check\_struct "sys/time.h sys/resource.h" "struct rusage" ru\_maxrss
3144. check\_func  inet\_aton $network\_extralibs
3145. check\_func  isatty
3146. check\_func  localtime\_r
3147. check\_func  ${malloc\_prefix}memalign            && enable memalign
3148. check\_func  mkstemp
3149. check\_func  mmap
3150. check\_func  ${malloc\_prefix}posix\_memalign      && enable posix\_memalign
3151. check\_func  setrlimit
3152. check\_func  strerror\_r
3153. check\_func  strptime
3154. check\_func  sched\_getaffinity
3155. check\_func  sysconf
3156. check\_func  sysctl
3157. check\_func\_headers conio.h kbhit
3158. check\_func\_headers windows.h PeekNamedPipe
3159. check\_func\_headers io.h setmode
3160. check\_func\_headers lzo/lzo1x.h lzo1x\_999\_compress
3161. check\_lib2 "windows.h psapi.h" GetProcessMemoryInfo -lpsapi
3162. check\_func\_headers windows.h GetProcessAffinityMask
3163. check\_func\_headers windows.h GetProcessTimes
3164. check\_func\_headers windows.h MapViewOfFile
3165. check\_func\_headers windows.h VirtualAlloc
3166. #检查头文件
3167. check\_header dlfcn.h
3168. check\_header dxva2api.h -D\_WIN32\_WINNT=0x0600
3169. check\_header libcrystalhd/libcrystalhd\_if.h
3170. check\_header malloc.h
3171. check\_header poll.h
3172. check\_header sys/mman.h
3173. check\_header sys/param.h
3174. check\_header sys/resource.h
3175. check\_header sys/select.h
3176. check\_header termios.h
3177. check\_header vdpau/vdpau.h
3178. check\_header vdpau/vdpau\_x11.h
3179. check\_header X11/extensions/XvMClib.h
3180. check\_header asm/types.h
3182. disabled  zlib || check\_lib   zlib.h      zlibVersion -lz   || disable  zlib
3183. disabled bzlib || check\_lib2 bzlib.h BZ2\_bzlibVersion -lbz2 || disable bzlib
3185. # check for VDA header
3186. if ! disabled vda; then
3187. if check\_header VideoDecodeAcceleration/VDADecoder.h; then
3188. enable vda
3189. add\_extralibs -framework CoreFoundation -framework VideoDecodeAcceleration -framework QuartzCore
3190. fi
3191. fi
3193. if ! disabled w32threads && ! enabled pthreads; then
3194. check\_func \_beginthreadex && enable w32threads
3195. fi
3197. # check for some common methods of building with pthread support
3198. # do this before the optional library checks as some of them require pthreads
3199. if ! disabled pthreads && ! enabled w32threads && ! enabled os2threads; then
3200. enable pthreads
3201. if check\_func pthread\_create; then
3202. :
3203. elif check\_func pthread\_create -pthread; then
3204. add\_cflags -pthread
3205. add\_extralibs -pthread
3206. elif check\_func pthread\_create -pthreads; then
3207. add\_cflags -pthreads
3208. add\_extralibs -pthreads
3209. elif check\_func pthread\_create -lpthreadGC2; then
3210. add\_extralibs -lpthreadGC2
3211. elif ! check\_lib pthread.h pthread\_create -lpthread; then
3212. disable pthreads
3213. fi
3214. fi
3216. for thread in $THREADS\_LIST; do
3217. if enabled $thread; then
3218. test -n "$thread\_type" &&
3219. die "ERROR: Only one thread type must be selected." ||
3220. thread\_type="$thread"
3221. fi
3222. done
3224. check\_lib math.h sin -lm && LIBM="-lm"
3225. disabled crystalhd || check\_lib libcrystalhd/libcrystalhd\_if.h DtsCrystalHDVersion -lcrystalhd || disable crystalhd
3226. enabled vaapi && require vaapi va/va.h vaInitialize -lva
3227. #检查数学函数
3228. check\_mathfunc cbrtf
3229. check\_mathfunc exp2
3230. check\_mathfunc exp2f
3231. check\_mathfunc llrint
3232. check\_mathfunc llrintf
3233. check\_mathfunc log2
3234. check\_mathfunc log2f
3235. check\_mathfunc lrint
3236. check\_mathfunc lrintf
3237. check\_mathfunc round
3238. check\_mathfunc roundf
3239. check\_mathfunc trunc
3240. check\_mathfunc truncf
3242. #检查第三方类库
3243. # these are off by default, so fail if requested and not available
3244. #require()函数参数的规范：（名称，头文件，函数名，附加选项）
3245. #require2()函数参数规范类似
3246. enabled avisynth   && require2 vfw32 "windows.h vfw.h" AVIFileInit -lavifil32
3247. enabled frei0r     && { check\_header frei0r.h || die "ERROR: frei0r.h header not found"; }
3248. enabled gnutls     && require\_pkg\_config gnutls gnutls/gnutls.h gnutls\_global\_init
3249. enabled libaacplus && require  "libaacplus >= 2.0.0" aacplus.h aacplusEncOpen -laacplus
3250. enabled libass     && require\_pkg\_config libass ass/ass.h ass\_library\_init
3251. enabled libcelt    && require libcelt celt/celt.h celt\_decode -lcelt0 &&
3252. { check\_lib celt/celt.h celt\_decoder\_create\_custom -lcelt0 ||
3253. die "ERROR: libcelt version must be >= 0.11.0."; }
3254. enabled libdc1394  && require\_pkg\_config libdc1394-2 dc1394/dc1394.h dc1394\_new
3255. enabled libdirac   && require\_pkg\_config dirac                          \
3256. "libdirac\_decoder/dirac\_parser.h libdirac\_encoder/dirac\_encoder.h"  \
3257. "dirac\_decoder\_init dirac\_encoder\_init"
3258. #测试libfaac
3259. enabled libfaac    && require2 libfaac "stdint.h faac.h" faacEncGetVersion -lfaac
3260. enabled libfreetype && require\_pkg\_config freetype2 "ft2build.h freetype/freetype.h" FT\_Init\_FreeType
3261. enabled libgsm     && require  libgsm gsm/gsm.h gsm\_create -lgsm
3262. enabled libmodplug && require  libmodplug libmodplug/modplug.h ModPlug\_Load -lmodplug
3263. enabled libmp3lame && require  "libmp3lame >= 3.98.3" lame/lame.h lame\_set\_VBR\_quality -lmp3lame
3264. enabled libnut     && require  libnut libnut.h nut\_demuxer\_init -lnut
3265. enabled libopencore\_amrnb  && require libopencore\_amrnb opencore-amrnb/interf\_dec.h Decoder\_Interface\_init -lopencore-amrnb
3266. enabled libopencore\_amrwb  && require libopencore\_amrwb opencore-amrwb/dec\_if.h D\_IF\_init -lopencore-amrwb
3267. enabled libopencv  && require\_pkg\_config opencv opencv/cxcore.h cvCreateImageHeader
3268. enabled libopenjpeg && require libopenjpeg openjpeg.h opj\_version -lopenjpeg
3269. enabled libpulse && require\_pkg\_config libpulse-simple pulse/simple.h pa\_simple\_new
3270. enabled librtmp    && require\_pkg\_config librtmp librtmp/rtmp.h RTMP\_Socket
3271. enabled libschroedinger && require\_pkg\_config schroedinger-1.0 schroedinger/schro.h schro\_init
3272. enabled libspeex   && require  libspeex speex/speex.h speex\_decoder\_init -lspeex
3273. enabled libstagefright\_h264  && require\_cpp libstagefright\_h264 "binder/ProcessState.h media/stagefright/MetaData.h
3274. media/stagefright/MediaBufferGroup.h media/stagefright/MediaDebug.h media/stagefright/MediaDefs.h
3275. media/stagefright/OMXClient.h media/stagefright/OMXCodec.h" android::OMXClient -lstagefright -lmedia -lutils -lbinder
3276. enabled libtheora  && require  libtheora theora/theoraenc.h th\_info\_init -ltheoraenc -ltheoradec -logg
3277. enabled libutvideo    && require\_cpp utvideo "stdint.h stdlib.h utvideo/utvideo.h utvideo/Codec.h" 'CCodec\*' -lutvideo -lstdc++
3278. enabled libv4l2    && require\_pkg\_config libv4l2 libv4l2.h v4l2\_ioctl
3279. enabled libvo\_aacenc && require libvo\_aacenc vo-aacenc/voAAC.h voGetAACEncAPI -lvo-aacenc
3280. enabled libvo\_amrwbenc && require libvo\_amrwbenc vo-amrwbenc/enc\_if.h E\_IF\_init -lvo-amrwbenc
3281. enabled libvorbis  && require  libvorbis vorbis/vorbisenc.h vorbis\_info\_init -lvorbisenc -lvorbis -logg
3282. enabled libvpx     && {
3283. enabled libvpx\_decoder && { check\_lib2 "vpx/vpx\_decoder.h vpx/vp8dx.h" vpx\_codec\_dec\_init\_ver -lvpx ||
3284. die "ERROR: libvpx decoder version must be >=0.9.1"; }
3285. enabled libvpx\_encoder && { check\_lib2 "vpx/vpx\_encoder.h vpx/vp8cx.h" "vpx\_codec\_enc\_init\_ver VPX\_CQ" -lvpx ||
3286. die "ERROR: libvpx encoder version must be >=0.9.6"; } }
3287. #测试libx264
3288. enabled libx264    && require  libx264 x264.h x264\_encoder\_encode -lx264 &&
3289. { check\_cpp\_condition x264.h "X264\_BUILD >= 118" ||
3290. die "ERROR: libx264 version must be >= 0.118."; }
3291. enabled libxavs    && require  libxavs xavs.h xavs\_encoder\_encode -lxavs
3292. enabled libxvid    && require  libxvid xvid.h xvid\_global -lxvidcore
3293. enabled openal     && { { for al\_libs in "${OPENAL\_LIBS}" "-lopenal" "-lOpenAL32"; do
3294. check\_lib 'AL/al.h' alGetError "${al\_libs}" && break; done } ||
3295. die "ERROR: openal not found"; } &&
3296. { check\_cpp\_condition "AL/al.h" "defined(AL\_VERSION\_1\_1)" ||
3297. die "ERROR: openal version must be 1.1 or compatible"; }
3298. enabled mlib       && require  mediaLib mlib\_types.h mlib\_VectorSub\_S16\_U8\_Mod -lmlib
3299. enabled openssl    && { check\_lib openssl/ssl.h SSL\_library\_init -lssl -lcrypto ||
3300. check\_lib openssl/ssl.h SSL\_library\_init -lssl32 -leay32 ||
3301. check\_lib openssl/ssl.h SSL\_library\_init -lssl -lcrypto -lws2\_32 -lgdi32 ||
3302. die "ERROR: openssl not found"; }
3303. #检查SDL
3304. SDL\_CONFIG="${cross\_prefix}sdl-config"
3305. if check\_pkg\_config sdl SDL\_version.h SDL\_Linked\_Version; then
3306. check\_cpp\_condition SDL.h "(SDL\_MAJOR\_VERSION<<16 | SDL\_MINOR\_VERSION<<8 | SDL\_PATCHLEVEL) >= 0x010201" $sdl\_cflags &&
3307. enable sdl &&
3308. check\_struct SDL.h SDL\_VideoInfo current\_w $sdl\_cflags && enable sdl\_video\_size
3309. else
3310. if "${SDL\_CONFIG}" --version > /dev/null 2>&1; then
3311. sdl\_cflags=$("${SDL\_CONFIG}" --cflags)
3312. sdl\_libs=$("${SDL\_CONFIG}" --libs)
3313. check\_func\_headers SDL\_version.h SDL\_Linked\_Version $sdl\_cflags $sdl\_libs &&
3314. check\_cpp\_condition SDL.h "(SDL\_MAJOR\_VERSION<<16 | SDL\_MINOR\_VERSION<<8 | SDL\_PATCHLEVEL) >= 0x010201" $sdl\_cflags &&
3315. enable sdl &&
3316. check\_struct SDL.h SDL\_VideoInfo current\_w $sdl\_cflags && enable sdl\_video\_size
3317. fi
3318. fi
3319. enabled sdl && add\_cflags $sdl\_cflags && add\_extralibs $sdl\_libs
3321. texi2html -version > /dev/null 2>&1 && enable texi2html || disable texi2html
3322. makeinfo --version > /dev/null 2>&1 && enable makeinfo  || disable makeinfo
3323. #检查头文件
3324. check\_header linux/fb.h
3325. check\_header linux/videodev.h
3326. check\_header linux/videodev2.h
3327. check\_struct linux/videodev2.h "struct v4l2\_frmivalenum" discrete
3329. check\_header sys/videoio.h
3331. check\_func\_headers "windows.h vfw.h" capCreateCaptureWindow "$vfwcap\_indev\_extralibs"
3332. # check that WM\_CAP\_DRIVER\_CONNECT is defined to the proper value
3333. # w32api 3.12 had it defined wrong
3334. check\_cpp\_condition vfw.h "WM\_CAP\_DRIVER\_CONNECT > WM\_USER" && enable vfwcap\_defines
3336. check\_type "dshow.h" IBaseFilter
3338. # check for ioctl\_meteor.h, ioctl\_bt848.h and alternatives
3339. { check\_header dev/bktr/ioctl\_meteor.h &&
3340. check\_header dev/bktr/ioctl\_bt848.h; } ||
3341. { check\_header machine/ioctl\_meteor.h &&
3342. check\_header machine/ioctl\_bt848.h; } ||
3343. { check\_header dev/video/meteor/ioctl\_meteor.h &&
3344. check\_header dev/video/bktr/ioctl\_bt848.h; } ||
3345. check\_header dev/ic/bt8xx.h
3347. check\_header sndio.h
3348. if check\_struct sys/soundcard.h audio\_buf\_info bytes; then
3349. enable\_safe sys/soundcard.h
3350. else
3351. check\_cc -D\_\_BSD\_VISIBLE -D\_\_XSI\_VISIBLE <<EOF && add\_cppflags -D\_\_BSD\_VISIBLE -D\_\_XSI\_VISIBLE && enable\_safe sys/soundcard.h
3352. #include <sys/soundcard.h>
3353. audio\_buf\_info abc;
3354. EOF
3355. fi
3356. check\_header soundcard.h
3358. enabled\_any alsa\_indev alsa\_outdev && check\_lib2 alsa/asoundlib.h snd\_pcm\_htimestamp -lasound
3360. enabled jack\_indev && check\_lib2 jack/jack.h jack\_client\_open -ljack && check\_func sem\_timedwait
3362. enabled\_any sndio\_indev sndio\_outdev && check\_lib2 sndio.h sio\_open -lsndio
3364. enabled libcdio &&
3365. check\_lib2 "cdio/cdda.h cdio/paranoia.h" cdio\_cddap\_open "-lcdio\_paranoia -lcdio\_cdda -lcdio"
3367. enabled x11grab                         &&
3368. check\_header X11/Xlib.h                 &&
3369. check\_header X11/extensions/XShm.h      &&
3370. check\_header X11/extensions/Xfixes.h    &&
3371. check\_func XOpenDisplay -lX11           &&
3372. check\_func XShmCreateImage -lX11 -lXext &&
3373. check\_func XFixesGetCursorImage -lX11 -lXext -lXfixes
3375. if ! disabled vaapi; then
3376. check\_lib va/va.h vaInitialize -lva && {
3377. check\_cpp\_condition va/va\_version.h "VA\_CHECK\_VERSION(0,32,0)" ||
3378. warn "Please upgrade to VA-API >= 0.32 if you would like full VA-API support.";
3379. } || disable vaapi
3380. fi
3382. if ! disabled vdpau && enabled vdpau\_vdpau\_h; then
3383. check\_cpp\_condition \
3384. vdpau/vdpau.h "defined VDP\_DECODER\_PROFILE\_MPEG4\_PART2\_ASP" ||
3385. { echolog "Please upgrade to libvdpau >= 0.2 if you would like vdpau support." &&
3386. disable vdpau; }
3387. fi
3389. enabled debug && add\_cflags -g"$debuglevel" && add\_asflags -g"$debuglevel"
3390. enabled coverage && add\_cflags "-fprofile-arcs -ftest-coverage" && add\_ldflags "-fprofile-arcs -ftest-coverage"
3391. test -n "$valgrind" && target\_exec="$valgrind --error-exitcode=1 --malloc-fill=0x2a --track-origins=yes --leak-check=full --gen-suppressions=all --suppressions=$source\_path/tests/fate-valgrind.supp"
3392. #添加一些编译选项
3393. # add some useful compiler flags if supported
3394. check\_cflags -Wdeclaration-after-statement
3395. check\_cflags -Wall
3396. check\_cflags -Wno-parentheses
3397. check\_cflags -Wno-switch
3398. check\_cflags -Wno-format-zero-length
3399. check\_cflags -Wdisabled-optimization
3400. check\_cflags -Wpointer-arith
3401. check\_cflags -Wredundant-decls
3402. check\_cflags -Wno-pointer-sign
3403. check\_cflags -Wcast-qual
3404. check\_cflags -Wwrite-strings
3405. check\_cflags -Wtype-limits
3406. check\_cflags -Wundef
3407. check\_cflags -Wmissing-prototypes
3408. check\_cflags -Wno-pointer-to-int-cast
3409. check\_cflags -Wstrict-prototypes
3410. enabled extra\_warnings && check\_cflags -Winline
3412. # add some linker flags
3413. check\_ldflags -Wl,--warn-common
3414. check\_ldflags -Wl,-rpath-link=libpostproc:libswresample:libswscale:libavfilter:libavdevice:libavformat:libavcodec:libavutil
3415. test\_ldflags -Wl,-Bsymbolic && append SHFLAGS -Wl,-Bsymbolic
3417. echo "X{};" > $TMPV
3418. if test\_ldflags -Wl,--version-script,$TMPV; then
3419. append SHFLAGS '-Wl,--version-script,\$(SUBDIR)lib\$(NAME).ver'
3420. check\_cc <<EOF && enable symver\_asm\_label
3421. void ff\_foo(void) \_\_asm\_\_ ("av\_foo@VERSION");
3422. void ff\_foo(void) { ${inline\_asm+\_\_asm\_\_($quotes);} }
3423. EOF
3424. check\_cc <<EOF && enable symver\_gnu\_asm
3425. \_\_asm\_\_(".symver ff\_foo,av\_foo@VERSION");
3426. void ff\_foo(void) {}
3427. EOF
3428. fi
3430. if [ -n "$optflags" ]; then
3431. add\_cflags $optflags
3432. elif enabled small; then
3433. add\_cflags $size\_cflags
3434. elif enabled optimizations; then
3435. add\_cflags $speed\_cflags
3436. else
3437. add\_cflags $noopt\_cflags
3438. fi
3439. check\_cflags -fno-math-errno
3440. check\_cflags -fno-signed-zeros
3441. check\_cc -mno-red-zone <<EOF && noredzone\_flags="-mno-red-zone"
3442. int x;
3443. EOF

3446. if enabled icc; then
3447. # Just warnings, no remarks
3448. check\_cflags -w1
3449. # -wd: Disable following warnings
3450. # 144, 167, 556: -Wno-pointer-sign
3451. # 1292: attribute "foo" ignored
3452. # 10006: ignoring unknown option -fno-signed-zeros
3453. # 10148: ignoring unknown option -Wno-parentheses
3454. # 10156: ignoring option '-W'; no argument required
3455. check\_cflags -wd144,167,556,1292,10006,10148,10156
3456. # 11030: Warning unknown option --as-needed
3457. # 10156: ignoring option '-export'; no argument required
3458. check\_ldflags -wd10156,11030
3459. # Allow to compile with optimizations
3460. check\_ldflags -march=$cpu
3461. # icc 11.0 and 11.1 work with ebp\_available, but don't pass the test
3462. enable ebp\_available
3463. if enabled x86\_32; then
3464. test ${icc\_version%%.\*} -ge 11 && \
3465. check\_cflags -falign-stack=maintain-16-byte || \
3466. disable aligned\_stack
3467. fi
3468. elif enabled ccc; then
3469. # disable some annoying warnings
3470. add\_cflags -msg\_disable cvtu32to64
3471. add\_cflags -msg\_disable embedcomment
3472. add\_cflags -msg\_disable needconstext
3473. add\_cflags -msg\_disable nomainieee
3474. add\_cflags -msg\_disable ptrmismatch1
3475. add\_cflags -msg\_disable unreachcode
3476. elif enabled gcc; then
3477. check\_cflags -fno-tree-vectorize
3478. check\_cflags -Werror=implicit-function-declaration
3479. check\_cflags -Werror=missing-prototypes
3480. elif enabled llvm\_gcc; then
3481. check\_cflags -mllvm -stack-alignment=16
3482. elif enabled clang; then
3483. check\_cflags -mllvm -stack-alignment=16
3484. check\_cflags -Qunused-arguments
3485. elif enabled armcc; then
3486. # 2523: use of inline assembler is deprecated
3487. add\_cflags -W${armcc\_opt},--diag\_suppress=2523
3488. add\_cflags -W${armcc\_opt},--diag\_suppress=1207
3489. add\_cflags -W${armcc\_opt},--diag\_suppress=1293 # assignment in condition
3490. add\_cflags -W${armcc\_opt},--diag\_suppress=3343 # hardfp compat
3491. add\_cflags -W${armcc\_opt},--diag\_suppress=167  # pointer sign
3492. add\_cflags -W${armcc\_opt},--diag\_suppress=513  # pointer sign
3493. elif enabled tms470; then
3494. add\_cflags -pds=824 -pds=837
3495. elif enabled pathscale; then
3496. add\_cflags -fstrict-overflow -OPT:wrap\_around\_unsafe\_opt=OFF
3497. fi
3499. enabled\_any $THREADS\_LIST      && enable threads
3501. check\_deps $CONFIG\_LIST       \
3502. $CONFIG\_EXTRA      \
3503. $HAVE\_LIST         \
3504. $ALL\_COMPONENTS    \
3505. $ALL\_TESTS         \
3507. enabled asm || { arch=c; disable $ARCH\_LIST $ARCH\_EXT\_LIST; }
3509. if test $target\_os = "haiku"; then
3510. disable memalign
3511. disable posix\_memalign
3512. fi
3514. ! enabled\_any memalign posix\_memalign malloc\_aligned &&
3515. enabled\_any $need\_memalign && enable memalign\_hack
3517. #在控制台输出信息
3518. echo "install prefix            $prefix"
3519. echo "source path               $source\_path"
3520. echo "C compiler                $cc"
3521. echo "ARCH                      $arch ($cpu)"
3522. if test "$build\_suffix" != ""; then
3523. echo "build suffix              $build\_suffix"
3524. fi
3525. if test "$progs\_suffix" != ""; then
3526. echo "progs suffix              $progs\_suffix"
3527. fi
3528. if test "$extra\_version" != ""; then
3529. echo "version string suffix     $extra\_version"
3530. fi
3531. #${}的特异功能：
3532. #${file-my.file.txt}假如 $file 为空值，则使用 my.file.txt 作默认值。(保留没设定及非空值)
3533. #在这里，如果某个变量为空值，则取默认值为no
3534. echo "big-endian                ${bigendian-no}"
3535. echo "runtime cpu detection     ${runtime\_cpudetect-no}"
3536. if enabled x86; then
3537. echo "${yasmexe}                      ${yasm-no}"
3538. echo "MMX enabled               ${mmx-no}"
3539. echo "MMX2 enabled              ${mmx2-no}"
3540. echo "3DNow! enabled            ${amd3dnow-no}"
3541. echo "3DNow! extended enabled   ${amd3dnowext-no}"
3542. echo "SSE enabled               ${sse-no}"
3543. echo "SSSE3 enabled             ${ssse3-no}"
3544. echo "AVX enabled               ${avx-no}"
3545. echo "CMOV enabled              ${cmov-no}"
3546. echo "CMOV is fast              ${fast\_cmov-no}"
3547. echo "EBX available             ${ebx\_available-no}"
3548. echo "EBP available             ${ebp\_available-no}"
3549. fi
3550. if enabled arm; then
3551. echo "ARMv5TE enabled           ${armv5te-no}"
3552. echo "ARMv6 enabled             ${armv6-no}"
3553. echo "ARMv6T2 enabled           ${armv6t2-no}"
3554. echo "ARM VFP enabled           ${armvfp-no}"
3555. echo "IWMMXT enabled            ${iwmmxt-no}"
3556. echo "NEON enabled              ${neon-no}"
3557. fi
3558. if enabled mips; then
3559. echo "MMI enabled               ${mmi-no}"
3560. fi
3561. if enabled ppc; then
3562. echo "AltiVec enabled           ${altivec-no}"
3563. echo "PPC 4xx optimizations     ${ppc4xx-no}"
3564. echo "dcbzl available           ${dcbzl-no}"
3565. fi
3566. if enabled sparc; then
3567. echo "VIS enabled               ${vis-no}"
3568. fi
3569. echo "debug symbols             ${debug-no}"
3570. echo "strip symbols             ${stripping-no}"
3571. echo "optimize for size         ${small-no}"
3572. echo "optimizations             ${optimizations-no}"
3573. echo "static                    ${static-no}"
3574. echo "shared                    ${shared-no}"
3575. echo "postprocessing support    ${postproc-no}"
3576. echo "new filter support        ${avfilter-no}"
3577. echo "network support           ${network-no}"
3578. echo "threading support         ${thread\_type-no}"
3579. echo "safe bitstream reader     ${safe\_bitstream\_reader-no}"
3580. echo "SDL support               ${sdl-no}"
3581. echo "Sun medialib support      ${mlib-no}"
3582. echo "libdxva2 enabled          ${dxva2-no}"
3583. echo "libva enabled             ${vaapi-no}"
3584. echo "libvdpau enabled          ${vdpau-no}"
3585. echo "AVISynth enabled          ${avisynth-no}"
3586. echo "frei0r enabled            ${frei0r-no}"
3587. echo "gnutls enabled            ${gnutls-no}"
3588. echo "libaacplus enabled        ${libaacplus-no}"
3589. echo "libass enabled            ${libass-no}"
3590. echo "libcdio support           ${libcdio-no}"
3591. echo "libcelt enabled           ${libcelt-no}"
3592. echo "libdc1394 support         ${libdc1394-no}"
3593. echo "libdirac enabled          ${libdirac-no}"
3594. echo "libfaac enabled           ${libfaac-no}"
3595. echo "libgsm enabled            ${libgsm-no}"
3596. echo "libmodplug enabled        ${libmodplug-no}"
3597. echo "libmp3lame enabled        ${libmp3lame-no}"
3598. echo "libnut enabled            ${libnut-no}"
3599. echo "libopencore-amrnb support ${libopencore\_amrnb-no}"
3600. echo "libopencore-amrwb support ${libopencore\_amrwb-no}"
3601. echo "libopencv support         ${libopencv-no}"
3602. echo "libopenjpeg enabled       ${libopenjpeg-no}"
3603. echo "libpulse enabled          ${libpulse-no}"
3604. echo "librtmp enabled           ${librtmp-no}"
3605. echo "libschroedinger enabled   ${libschroedinger-no}"
3606. echo "libspeex enabled          ${libspeex-no}"
3607. echo "libstagefright-h264 enabled    ${libstagefright\_h264-no}"
3608. echo "libtheora enabled         ${libtheora-no}"
3609. echo "libutvideo enabled        ${libutvideo-no}"
3610. echo "libv4l2 enabled           ${libv4l2-no}"
3611. echo "libvo-aacenc support      ${libvo\_aacenc-no}"
3612. echo "libvo-amrwbenc support    ${libvo\_amrwbenc-no}"
3613. echo "libvorbis enabled         ${libvorbis-no}"
3614. echo "libvpx enabled            ${libvpx-no}"
3615. echo "libx264 enabled           ${libx264-no}"
3616. echo "libxavs enabled           ${libxavs-no}"
3617. echo "libxvid enabled           ${libxvid-no}"
3618. echo "openal enabled            ${openal-no}"
3619. echo "openssl enabled           ${openssl-no}"
3620. echo "zlib enabled              ${zlib-no}"
3621. echo "bzlib enabled             ${bzlib-no}"
3622. echo
3624. for type in decoder encoder hwaccel parser demuxer muxer protocol filter bsf indev outdev; do
3625. echo "Enabled ${type}s:"
3626. eval list=\$$(toupper $type)\_LIST
3627. print\_enabled '\_\*' $list | sort | pr -r -3 -t
3628. echo
3629. done
3631. license="LGPL version 2.1 or later"
3632. if enabled nonfree; then
3633. license="nonfree and unredistributable"
3634. elif enabled gplv3; then
3635. license="GPL version 3 or later"
3636. elif enabled lgplv3; then
3637. license="LGPL version 3 or later"
3638. elif enabled gpl; then
3639. license="GPL version 2 or later"
3640. fi
3642. echo "License: $license"
3643. #创建config.mak和config.h
3644. #根据情况也会创建config.asm
3645. echo "Creating config.mak and config.h..."
3647. test -e Makefile || $ln\_s "$source\_path/Makefile" .
3649. enabled stripping || strip="echo skipping strip"
3650. #重要：需要输出的文件
3651. #TMPH是一个临时文件，最终会拷贝给config.h
3652. config\_files="$TMPH config.mak"
3653. #写入config.mak文件
3654. #首先写入一些基本信息
3655. #"<<EOF"表示后续的输入作为子命令或子shell的输入，直到遇到"EOF"，再次返回到
3656. #主调shell，可将其理解为分界符（delimiter）。
3657. #最后的"EOF"必须单独占一行
3658. cat > config.mak <<EOF
3659. # Automatically generated by configure - do not modify!
3660. ifndef FFMPEG\_CONFIG\_MAK
3661. FFMPEG\_CONFIG\_MAK=1
3662. FFMPEG\_CONFIGURATION=$FFMPEG\_CONFIGURATION
3663. prefix=$prefix
3664. LIBDIR=\$(DESTDIR)$libdir
3665. SHLIBDIR=\$(DESTDIR)$shlibdir
3666. INCDIR=\$(DESTDIR)$incdir
3667. BINDIR=\$(DESTDIR)$bindir
3668. DATADIR=\$(DESTDIR)$datadir
3669. MANDIR=\$(DESTDIR)$mandir
3670. SRC\_PATH=$source\_path
3671. ifndef MAIN\_MAKEFILE
3672. SRC\_PATH:=\$(SRC\_PATH:.%=..%)
3673. endif
3674. CC\_IDENT=$cc\_ident
3675. ARCH=$arch
3676. CC=$cc
3677. CXX=$cxx
3678. AS=$as
3679. LD=$ld
3680. DEPCC=$dep\_cc
3681. YASM=$yasmexe
3682. YASMDEP=$yasmexe
3683. AR=$ar
3684. RANLIB=$ranlib
3685. CP=cp -p
3686. LN\_S=$ln\_s
3687. STRIP=$strip
3688. CPPFLAGS=$CPPFLAGS
3689. CFLAGS=$CFLAGS
3690. CXXFLAGS=$CXXFLAGS
3691. ASFLAGS=$ASFLAGS
3692. AS\_O=$CC\_O
3693. CC\_O=$CC\_O
3694. CXX\_O=$CXX\_O
3695. LDFLAGS=$LDFLAGS
3696. FFSERVERLDFLAGS=$FFSERVERLDFLAGS
3697. SHFLAGS=$SHFLAGS
3698. YASMFLAGS=$YASMFLAGS
3699. BUILDSUF=$build\_suffix
3700. PROGSSUF=$progs\_suffix
3701. FULLNAME=$FULLNAME
3702. LIBPREF=$LIBPREF
3703. LIBSUF=$LIBSUF
3704. LIBNAME=$LIBNAME
3705. SLIBPREF=$SLIBPREF
3706. SLIBSUF=$SLIBSUF
3707. EXESUF=$EXESUF
3708. EXTRA\_VERSION=$extra\_version
3709. DEPFLAGS=$DEPFLAGS
3710. CCDEP=$CCDEP
3711. CXXDEP=$CXXDEP
3712. ASDEP=$ASDEP
3713. CC\_DEPFLAGS=$CC\_DEPFLAGS
3714. AS\_DEPFLAGS=$AS\_DEPFLAGS
3715. HOSTCC=$host\_cc
3716. HOSTCFLAGS=$host\_cflags
3717. HOSTEXESUF=$HOSTEXESUF
3718. HOSTLDFLAGS=$host\_ldflags
3719. HOSTLIBS=$host\_libs
3720. TARGET\_EXEC=$target\_exec
3721. TARGET\_PATH=$target\_path
3722. SDL\_LIBS=$sdl\_libs
3723. SDL\_CFLAGS=$sdl\_cflags
3724. LIB\_INSTALL\_EXTRA\_CMD=$LIB\_INSTALL\_EXTRA\_CMD
3725. EXTRALIBS=$extralibs
3726. INSTALL=$install
3727. LIBTARGET=${LIBTARGET}
3728. SLIBNAME=${SLIBNAME}
3729. SLIBNAME\_WITH\_VERSION=${SLIBNAME\_WITH\_VERSION}
3730. SLIBNAME\_WITH\_MAJOR=${SLIBNAME\_WITH\_MAJOR}
3731. SLIB\_CREATE\_DEF\_CMD=${SLIB\_CREATE\_DEF\_CMD}
3732. SLIB\_EXTRA\_CMD=${SLIB\_EXTRA\_CMD}
3733. SLIB\_INSTALL\_NAME=${SLIB\_INSTALL\_NAME}
3734. SLIB\_INSTALL\_LINKS=${SLIB\_INSTALL\_LINKS}
3735. SLIB\_INSTALL\_EXTRA\_LIB=${SLIB\_INSTALL\_EXTRA\_LIB}
3736. SLIB\_INSTALL\_EXTRA\_SHLIB=${SLIB\_INSTALL\_EXTRA\_SHLIB}
3737. SAMPLES:=${samples:-\$(FATE\_SAMPLES)}
3738. NOREDZONE\_FLAGS=$noredzone\_flags
3739. EOF
3740. #获取版本
3741. #主要通过各个类库文件夹中的version.h文件
3742. #读取XXX\_VERSION（相当于把头文件当成一个文本来读）
3743. get\_version(){
3744. name=$1
3745. file=$source\_path/$2
3746. # This condition will be removed when we stop supporting old libpostproc versions
3747. if ! test "$name" = LIBPOSTPROC || test "$postproc\_version" = current; then
3748. eval $(grep "#define ${name}\_VERSION\_M" "$file" | awk '{ print $2"="$3 }')
3749. eval ${name}\_VERSION=\$${name}\_VERSION\_MAJOR.\$${name}\_VERSION\_MINOR.\$${name}\_VERSION\_MICRO
3750. fi
3751. lcname=$(tolower $name)
3752. eval echo "${lcname}\_VERSION=\$${name}\_VERSION" >> config.mak
3753. eval echo "${lcname}\_VERSION\_MAJOR=\$${name}\_VERSION\_MAJOR" >> config.mak
3754. }
3755. #获取版本
3756. get\_version LIBAVCODEC  libavcodec/version.h
3757. get\_version LIBAVDEVICE libavdevice/avdevice.h
3758. get\_version LIBAVFILTER libavfilter/version.h
3759. get\_version LIBAVFORMAT libavformat/version.h
3760. get\_version LIBAVUTIL   libavutil/avutil.h
3761. get\_version LIBPOSTPROC libpostproc/postprocess.h
3762. get\_version LIBSWRESAMPLE libswresample/swresample.h
3763. get\_version LIBSWSCALE  libswscale/swscale.h
3764. #config.h前面需要添加的一些内容（TMPH是一个临时文件，最终会拷贝给config.h）
3765. cat > $TMPH <<EOF
3766. /\* Automatically generated by configure - do not modify! \*/
3767. #ifndef FFMPEG\_CONFIG\_H
3768. #define FFMPEG\_CONFIG\_H
3769. #define FFMPEG\_CONFIGURATION "$(c\_escape $FFMPEG\_CONFIGURATION)"
3770. #define FFMPEG\_LICENSE "$(c\_escape $license)"
3771. #define FFMPEG\_DATADIR "$(eval c\_escape $datadir)"
3772. #define AVCONV\_DATADIR "$(eval c\_escape $datadir)"
3773. #define CC\_TYPE "$cc\_type"
3774. #define CC\_VERSION $cc\_version
3775. #define restrict $\_restrict
3776. #define EXTERN\_PREFIX "${extern\_prefix}"
3777. #define EXTERN\_ASM ${extern\_prefix}
3778. #define SLIBSUF "$SLIBSUF"
3779. EOF
3781. test -n "$malloc\_prefix" &&
3782. echo "#define MALLOC\_PREFIX $malloc\_prefix" >>$TMPH
3784. if enabled small || disabled optimizations; then
3785. echo "#undef  av\_always\_inline"  >> $TMPH
3786. if enabled small; then
3787. echo "#define av\_always\_inline inline"  >> $TMPH
3788. else
3789. echo "#define av\_always\_inline av\_unused"  >> $TMPH
3790. fi
3791. fi
3792. #包含yasm
3793. if enabled yasm; then
3794. append config\_files $TMPASM
3795. printf '' >$TMPASM
3796. fi
3797. #输出所有的配置信息包含3类：
3798. #以“ARCH\_”开头，包含系统架构信息
3799. #以“HAVE\_”开头，包含系统特征信息
3800. #以“CONFIG\_”开头，包含编译配置（数量最多，包含协议、复用器、编解码器等的配置，将近1000行）
3801. #config\_files
3802. print\_config ARCH\_   "$config\_files" $ARCH\_LIST
3803. print\_config HAVE\_   "$config\_files" $HAVE\_LIST
3804. print\_config CONFIG\_ "$config\_files" $CONFIG\_LIST       \
3805. $CONFIG\_EXTRA      \
3806. $ALL\_COMPONENTS    \
3807. #经过测试的组件？
3808. cat >>config.mak <<EOF
3809. ACODEC\_TESTS=$(print\_enabled -n \_test $ACODEC\_TESTS)
3810. VCODEC\_TESTS=$(print\_enabled -n \_test $VCODEC\_TESTS)
3811. LAVF\_TESTS=$(print\_enabled   -n \_test $LAVF\_TESTS)
3812. LAVFI\_TESTS=$(print\_enabled  -n \_test $LAVFI\_TESTS)
3813. SEEK\_TESTS=$(print\_enabled   -n \_test $SEEK\_TESTS)
3814. EOF
3816. echo "#endif /\* FFMPEG\_CONFIG\_H \*/" >> $TMPH
3817. #结束了
3818. echo "endif # FFMPEG\_CONFIG\_MAK" >> config.mak
3820. # 关键：临时文件拷贝至config.h
3821. # Do not overwrite an unchanged config.h to avoid superfluous rebuilds.
3822. # 配置没有变化的时候，不重新生成config.h（重新生成config.h会导致大量文件需要重新编译）
3823. #
3824. cp\_if\_changed $TMPH config.h
3825. # touch fileA
3826. # 如果fileA不存在，touch指令会在当前目录下新建一个空白文件fileA。
3827. touch .config
3829. enabled yasm && cp\_if\_changed $TMPASM config.asm
3831. cat > $TMPH <<EOF
3832. /\* Generated by ffconf \*/
3833. #ifndef AVUTIL\_AVCONFIG\_H
3834. #define AVUTIL\_AVCONFIG\_H
3835. EOF
3837. test "$postproc\_version" != current && cat >> $TMPH <<EOF
3838. #define LIBPOSTPROC\_VERSION\_MAJOR $LIBPOSTPROC\_VERSION\_MAJOR
3839. #define LIBPOSTPROC\_VERSION\_MINOR $LIBPOSTPROC\_VERSION\_MINOR
3840. #define LIBPOSTPROC\_VERSION\_MICRO $LIBPOSTPROC\_VERSION\_MICRO
3841. EOF
3843. print\_config AV\_HAVE\_ $TMPH $HAVE\_LIST\_PUB
3845. echo "#endif /\* AVUTIL\_AVCONFIG\_H \*/" >> $TMPH
3847. cp\_if\_changed $TMPH libavutil/avconfig.h
3849. test -n "$WARNINGS" && printf "\n$WARNINGS"
3851. # build pkg-config files
3853. pkgconfig\_generate(){
3854. name=$1
3855. shortname=${name#lib}${build\_suffix}
3856. comment=$2
3857. version=$3
3858. libs=$4
3859. requires=$5
3860. enabled ${name#lib} || return 0
3861. mkdir -p $name
3862. cat <<EOF > $name/$name.pc
3863. prefix=$prefix
3864. exec\_prefix=\${prefix}
3865. libdir=$libdir
3866. includedir=$incdir
3868. Name: $name
3869. Description: $comment
3870. Version: $version
3871. Requires: $(enabled shared || echo $requires)
3872. Requires.private: $(enabled shared && echo $requires)
3873. Conflicts:
3874. Libs: -L\${libdir} -l${shortname} $(enabled shared || echo $libs)
3875. Libs.private: $(enabled shared && echo $libs)
3876. Cflags: -I\${includedir}
3877. EOF
3878. cat <<EOF > $name/$name-uninstalled.pc
3879. prefix=
3880. exec\_prefix=
3881. libdir=\${pcfiledir}
3882. includedir=${source\_path}
3884. Name: $name
3885. Description: $comment
3886. Version: $version
3887. Requires: $requires
3888. Conflicts:
3889. Libs: \${libdir}/${LIBPREF}${shortname}${LIBSUF} $libs
3890. Cflags: -I\${includedir}
3891. EOF
3892. }
3894. pkgconfig\_generate libavutil "FFmpeg utility library" "$LIBAVUTIL\_VERSION" "$LIBM"
3895. pkgconfig\_generate libavcodec "FFmpeg codec library" "$LIBAVCODEC\_VERSION" "$extralibs" "libavutil = $LIBAVUTIL\_VERSION"
3896. pkgconfig\_generate libavformat "FFmpeg container format library" "$LIBAVFORMAT\_VERSION" "$extralibs" "libavcodec = $LIBAVCODEC\_VERSION"
3897. pkgconfig\_generate libavdevice "FFmpeg device handling library" "$LIBAVDEVICE\_VERSION" "$extralibs" "libavformat = $LIBAVFORMAT\_VERSION"
3898. pkgconfig\_generate libavfilter "FFmpeg video filtering library" "$LIBAVFILTER\_VERSION" "$extralibs"
3899. pkgconfig\_generate libpostproc "FFmpeg postprocessing library" "$LIBPOSTPROC\_VERSION" "" "libavutil = $LIBAVUTIL\_VERSION"
3900. pkgconfig\_generate libswscale "FFmpeg image rescaling library" "$LIBSWSCALE\_VERSION" "$LIBM" "libavutil = $LIBAVUTIL\_VERSION"
3901. pkgconfig\_generate libswresample "FFmpeg audio rescaling library" "$LIBSWRESAMPLE\_VERSION" "$LIBM" "libavutil = $LIBAVUTIL\_VERSION"