杂谈

最后更新时间：2019-11-16

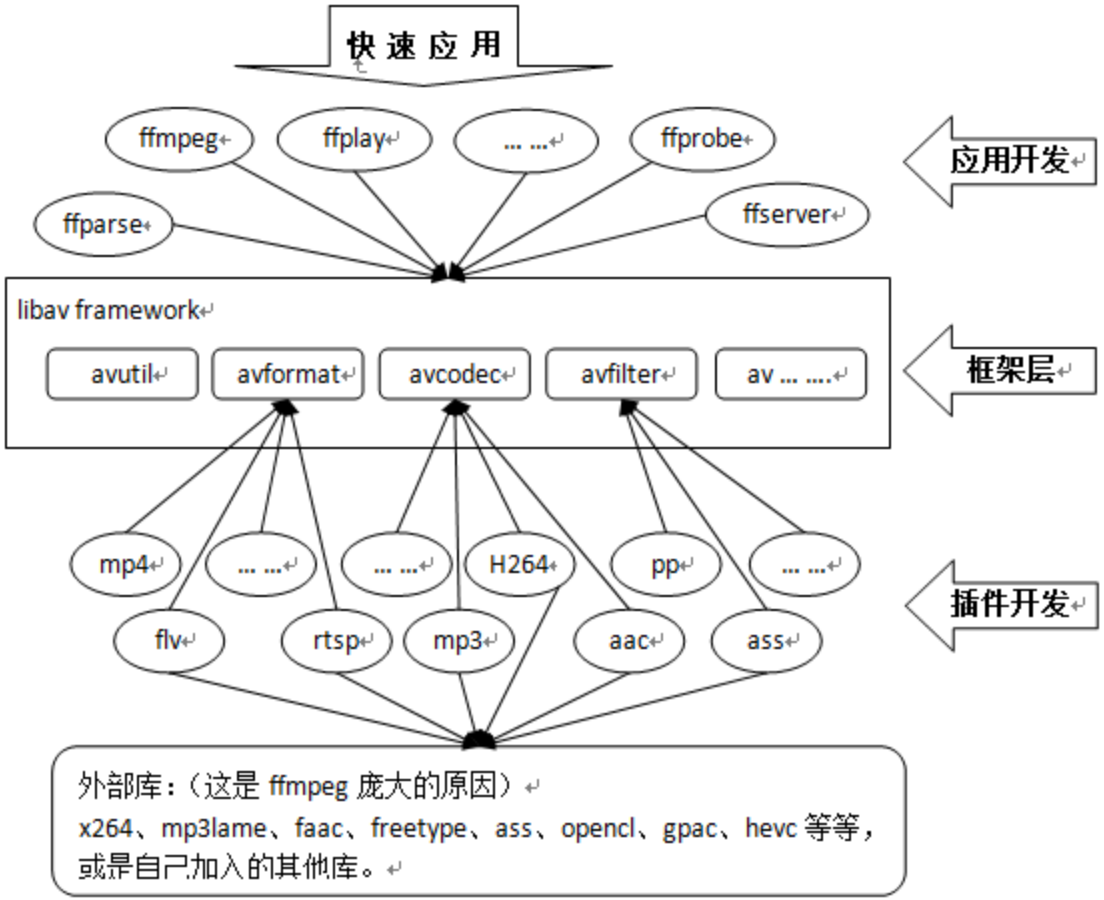
自我介绍

- 07 年参加工作，图像系统入行
- 2011 年开始涉足 Android Multimedia
- 2012 年开始全身心涉足 FFmpeg
- 2013 年开始涉足流媒体 CDN， 同时开始向 FFmpeg 贡献代码
- 2016 年开始受邀成为 FFmpeg Maintainer / Committer、Consult
- 2019 年被选为 Vote Community Committee 成员
- GSoC 2019 FFmpeg Mentor

FFmpeg History

- Introduction
- Open-source multimedia library， 遵从GPL/LGPL协议，ffmpeg只是一个商标，它的所有权属于ffmpeg org。
- History
- 由Fabrice Bellard（法国著名程序员Born in 1972）于2000年发起创建的开源项目，同时也是TinyCC(1996)、发现最快速计算圆周率算法(1997)、TinyGL(1998)、QEMU(2003)、 Jslinux(2011)等等的发起人或作者。
- 2008-Now， Ronald S Bultje 在2011年迁移一部分为Libav http://libav.org
- 2011 开始出现libavfilter身影
- 2012 开始大批量加入libavfilter
- 2013 开始高速发展
- 2015 Michael Niedermayer 辞去领导人角色，并开始大量从libav merge代码

FFmpeg Development Framework Introduction



FFmpeg Version Management

- Tool

  源码管理工具使用Git和SVN，FFmpeg最早是基于Linux上开发，可以在各个操作系统上编译运行。

- Branch

0.5.X	版本分子 最早	0.8.X	版本分子	2.8	版本分子
0.6.X	版本分子	1.2.X	版本分子	3.2	版本分子
0.7.X	版本分子	2.1.X	版本分子	3.3	版本分子

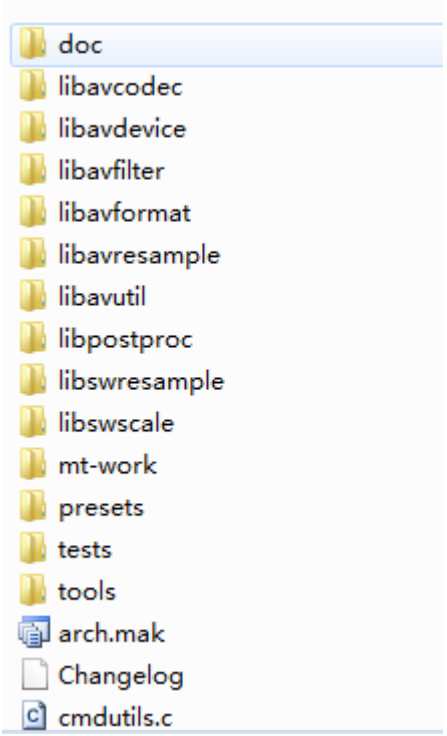
- Protect

Debug：CVE-2012-2801 format – avfreep param，user group确定bug，然后由devel 的对应Protect 人/copyright人 来修改，提交后由devel group核对提交。

Develop：由Protect对应的小组leader来开发。

FFmpeg Module

- 【libavutil 】 通用库
- 【libavcodec】 编解码
- 【libavformat】 文件格式
- 【libpostproc】 同步、时间计算的简单算法
- 【libavfilter】 filter（FileIO、FPS、 DrawText
- 【libavdevice】 硬件采集、加速、显示
- 【libavresample】 音视频封转编解码格式预设等
- 【libswscale】 原始视频格式转换
- 【libswresample】 原始音频格式转码



FFmpeg Module Version

如下是列举了FFmpeg git版本的模块版本信息:

- libavutil 55. 68.100 / 55. 68.100
- libavcodec 57.102.100 / 57.102.100
- libavformat 57. 76.100 / 57. 76.100
- libavdevice 57. 7.100 / 57. 7.100
- libavfilter 6. 95.100 / 6. 95.100
- libswscale 4. 7.101 / 4. 7.101
- libswresample 2. 8.100 / 2. 8.100
- libpostproc 54. 6.100 / 54. 6.100

FFmpeg Demuxer/Muxer/Protocol

- 协议： HTTP/HTTPS/HTTPPROXY/APPLEHTTP、 Pipe、RTMP/RTP、UDP/TCP、HLS/TLS、MMSH/MMST 、FILE等等
- 文件： AMR(DM)、DV(DM)、FLV/SWF(DM)、GIF(M)、GSM(D)、G722/G723_1(DM) 、 G729(D)、H261/H263/H264(DM)、IOC(D)、IMAGE2(DM)、IPOD(M)、M4V(DM)、MJPEG/LJPEG(DM)、MOV(DM)、MP2/MP3/MP4(DM)、MPEG2/MPEGTS(DM)、OGG/OMA(DM)、PCM系列(DM)、RM(DM)、RTP/RTSP(DM)、SRT(DM)、TG2/TGP(M)、WAV(DM)、VMD(D)等等

FFmpeg Encoder/Decoder

- 编解码器：
- V:AMV(E)、AVS(D)、BMP/GIF/JPEG/TIFF/PNG(E)、H261/H263/H264(DE)、MPEG1/2/4/4v1/4v2/4v3(DE)、PGM(E)、VC1(DE)、WMV1/2/3(DE)、RV30/RV40(DE)、VP3/VP8(D)等等。
- A: PCM/DPCM/ADPCM(DE)、ACC(DE)、AMR(DE)、COOK(DE)、G723/G726/G729(DE)、MP1/MP2/MP3(DE)、WMA(DE)、GSM(D)、AC3(DE)、OPUS等等。
- S：ass、SRT、XSUB等等。
- 硬件加速： VDA/Videotoolbox/Audiotoolbox/VAAPI/QSV/NVENC/CUVID/OMX等等。

FFmpeg Device

- 硬件方式：CDIO / DC1394 （输入设备）
- 非扩展硬件：DSHOW（输入设备）、SDL（输出设备）、X11（输入）、VFWCAP（输入）、DV1394（输入）、AVFoundation 等等。

FFmpeg Filter

- AV-Filter: volume、crop、draw、frame等等。
- Codec-Filter:acc、h264、mp3、mpeg4-gh(h264-sps_pps)/mjpeg-huffman_tables/rv10-affitional_flags、mov_sub等等
- F-Filter:src、buffersink、fifo/avio、format等等。

FFmpeg Extension Library

bzip http://www.bzip.org/ CELT http://www.celt-codec.org/

FreeType http://www.freetype.org/ Frei0r http://www.piksel.org/frei0r/

LAME http://lame.sourceforge.net/ libass http://code.google.com/p/libass/

libgsm http://libgsm.sourcearchive.com/

Theora http://www.theora.org/ Vorbis http://www.vorbis.com/

libvpx http://www.webmproject.org/ NUT <git://git.ffmpeg.org/nut>

OpenCORE AMR http://sourceforge.net/projects/opencore-amr/

OpenJPEG http://www.openjpeg.org/ RTMPDump http://rtmpdump.mplayerhq.hu/

Schroedinger http://diracvideo.org/ Speexhttp://www.speex.org/

Ut Videohttp://umezawa.dyndns.info/archive/utvideo/

VisualOn AAC http://sourceforge.net/projects/opencore-amr/

VisualOn AMR-WB http://sourceforge.net/projects/opencore-amr/

x264http://www.videolan.org/developers/x264.html

XAVS http://xavs.sourceforge.net/ Xvid http://www.xvid.org/ zlib http://zlib.net/

FFmpeg 编译

主流OS支持:Win NT/CE/Phone、linux/unix、android(linux)、ios/mac os(改后linux)。

Windows编译选择:

1. Ubuntu(linux) + mingw 模拟编译
2. Cygwin + mingw + ms交叉编译
3. Msys + mingw + ms交叉编译
4. Intel + ms 本地化编译(部分)
5. VS直接编译需要改源码。

FFmpeg 汇编

在linux上开发，gcc编译的，分为2类，嵌入式汇编格式(.c/.S);汇编文件格式(.asm).gcc 使用AT&T语法格式的汇编非Intel.

指令支持:

1.ppc

2.x86(IA芯片)

3.arm

4.Sparc(Sun)

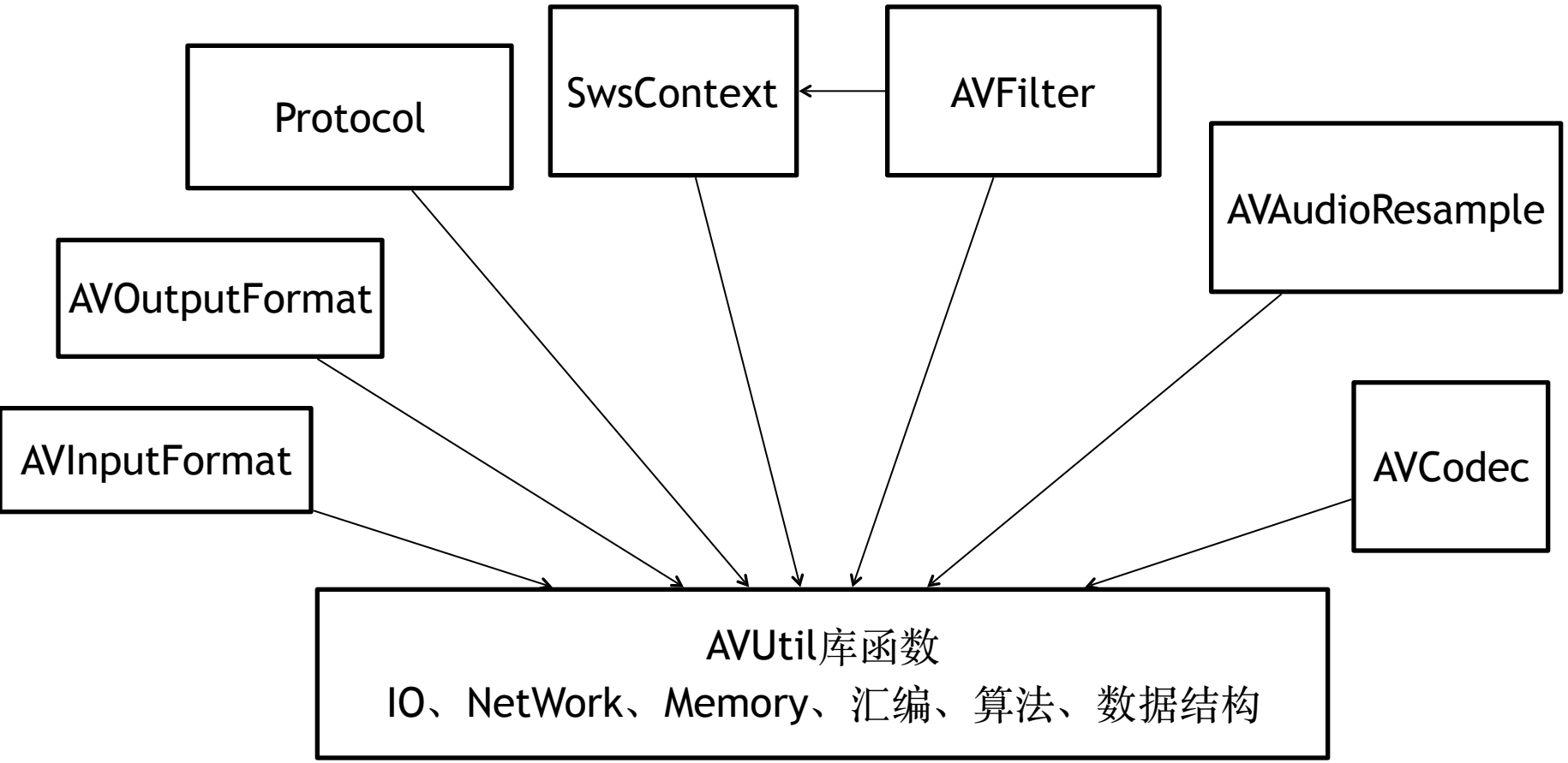
5.bfin、sh4(DSP芯片)

6.Mips、avr32(单片机)

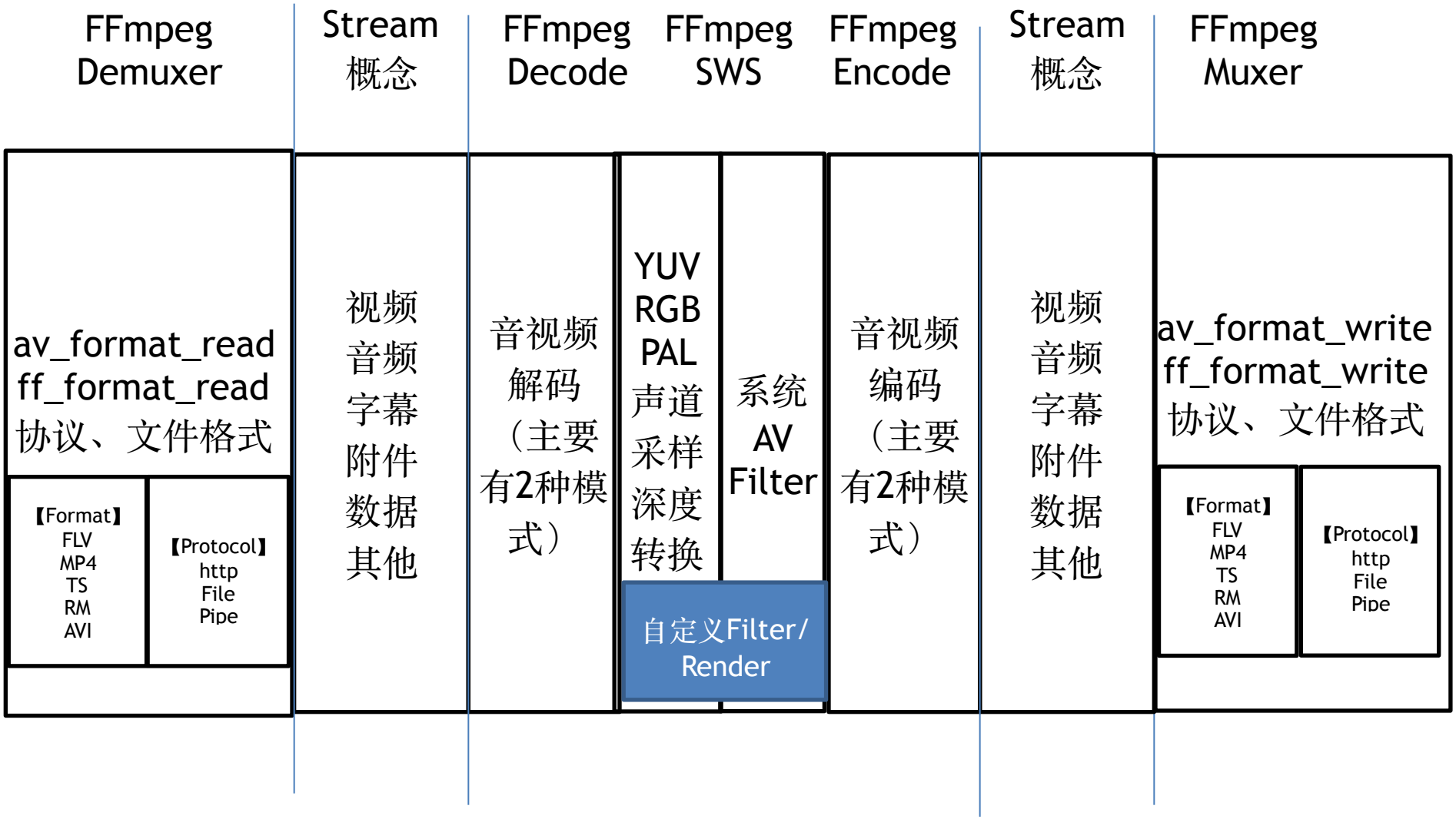
7.alpha(小型机64 cpu架构)

一直再更新支持最新更多的指令集。

FFmpeg Framework



FFmpeg 转码流程


FFmpeg Demuxer
Stream 概念
FFmpeg Decode
FFmpeg SWS
FFmpeg Encode
Stream 概念
FFmpeg Muxer
av_format_read
ff_format_read
协议、文件格式
【Format】
FLV
MP4
TS
RM
AVI
【Protocol】
http
File
Pipe
视频
音频
字幕
附件
数据
其他
音视频
解码
（主要
有2种模
式）
YUV
RGB
PAL
声道
采样
深度
转换
系统
AV
Filter
自定义Filter/
Render
音视频
编码
（主要
有2种模
式）
视频
音频
字幕
附件
数据
其他
av_format_write
ff_format_write
协议、文件格式
【Format】
FLV
MP4
TS
RM
AVI
【Protocol】
http
File
Pipe

FFmpeg Review

- FFmpeg Position

  FFmpeg从层次划分：快速应用、应用开发、框架层、插件开发；FFmpeg从模块划分：avutil、avcodec、avdevice、avfilter、avformat、postproc、resample、scale；

  FFmpeg从功能划分：编解码、容器封装和解析、协议IO、音频重采样、视频色彩空间转换、音视频字幕处理。

- Deep Step

1. FFmpeg是用C语言实现的面向对象的高效框架库，要先了解 FFmpeg KS（核心数据结构），将从流媒体数据流动方向，原始数据->帧->数据包->流->容器->协议，逐步介绍。
2. 深入学习FFmpeg各个 Module的核心结构，学习容器、协议、编解码、音视频处理标准和原理，再通过流媒体串起来，流媒体非专业的核心可能还涉及：关于高中物理、统计学、概率论等一些知识。
3. FFmpeg 核心扩展功能开发。

FFmpeg KS [数据帧/数据包]

AVFrame用于存储原始音视频数据、也包含字幕数据。

原始数据是指：解码后数据、或是编码前的数据。

音频：一段原始WAV（或未压缩的PCM）音频数据。

视频：一帧图像、一个图片。

字幕：一个时间段内的一句或是几句话。

AVPacket用于存储字节流数据，包含音视频、字幕、附件信息等等。

字节流数据是指：编码后数据、或是解码前数据、或是要封装的数据、或是封装解析后的数据。

数据：字节流数据包。

FFmpeg KS [编解码]

AVCodec编解码器的实现功能类对象。

主要保存的是：函数接口实现、常量。

AVCodecContext编解码器的数据类对象。

主要保存的是：数据、动态接口等。

AVCodecParameters编解码器的数据类对象。

主要保存的是：数据、动态接口等。

FFmpeg KS [流/容器]

AVInputFormat容器解析的实现功能类对象。

例如：读取一个MP4文件，就需要一个MP4容器解析对象来实现功能。

AVOutputFormat容器封装的实现功能类对象。

例如：生成一个MP4文件，就需要一个MP4容器封装对象来实现功能。

AVFormatContext容器的数据类对象。

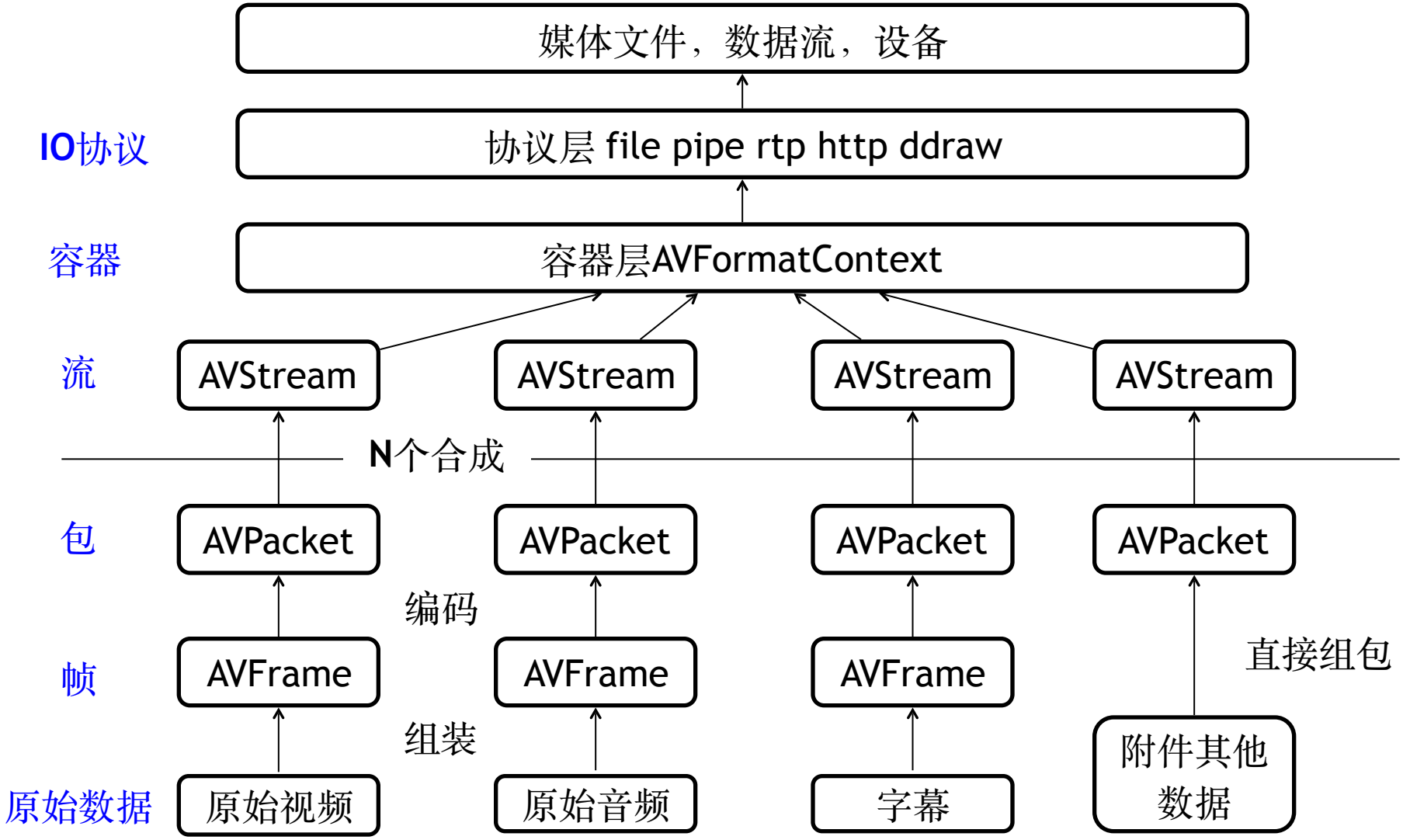
例如：读写MP4文件时，数据的存储对象。

AVStream容器内数据流对象，属于抽象对象。

主要概念：数据流、时间轴等。

- AVStream是数据流，它的基本成员有codec，context，，时长，总帧数，帧率（fps分数）/码率，metadata等等。
- AVFormatContext是封装核心，它的基本成员有源的路径，iformat（输入），oformat（输出），所有的数据流，格式分类，时间参数，包seek相关，metadata，chunk，对应对的属性等等。

FFmpeg KS数据流程图



FFmpeg KS [原始音视频数据格式转换]

SwsContext色彩空间转换对象。

是数据和功能函数统一的一个对象。

1.可以修改图像分辨率（缩放、拉伸功能）。

2.色彩空间转换（yuv-rgb-gray）。

SwrContext音频重采样转换对象。

是数据和功能函数统一的一个对象。

1.采样深度。

2.采样频率。

3.声道数和声道布局的有限转换。

FFmpeg KS[音视频处理]

AVFilterGraph 数据对象，动态管理，外部可见 滤器链路图

AVFilter 功能对象，固定分配，外部可见 音视频滤器

AVFilterContext 数据对象，动态管理，外部可见 音视频滤器上下文

AVFilterPad 功能对象，固定分配，外部可见 音视频滤器输入输出接口

AVFilterBuffer 数据对象，动态管理，外部可见 音视频数据帧

AVFilterBufferRefAudioProps 数据结构 音频数据帧，属性信息

AVFilterBufferRefVideoProps 数据结构 视频数据帧，属性信息

AVFilterBufferRef 数据对象，动态管理，外部可见 音视频数据帧引用扩展对象

AVFilterLink 数据对象，动态管理，内部使用 滤器链路（内部filter之间）

AVFilterInOut 数据对象，动态管理，外部可见 滤器链路（外部IO之间）

AVFilterCommand 数据结构 外部参数传递对象

AVFilterPool 数据结构 音视频数据帧引用池

AVFilterFormats 数据结构 滤器支持类型，视频格式

AVFilterChannelLayouts 数据结构 滤器支持类型，音频格式

AVBufferSinkParams

AVABufferSinkParams 内部 音视频参数传递

视频原始数据[AVFrame]

数据面个数：

rgb:1 yuv:1-3 normal-4 **ffmpeg-8** opencv-16 dx- IDirectDrawSurface

BYTE * **Data**[8]（数据面个数）

行：

1. Stride(原始的) + Heigh + ColorFormat
2. Pitch(对齐的) + Heigh + width + ColorFormat
3. Linesize(内存) + Heigh + width + PixelFormat

int Linesize[8]（行）

音频原始数据[AVFrame]

int sample_fmt 采样格式（u8bit,16bit,32bit,float,double）

int sample_rate 采样率（44100,48000,128000-mp3,192000,320000）ape-flac

int channels 通道数(单声道，双声道，四通道，5.1声道，6声道)

BYTE * buffer 音频流

int buffer_size 缓冲区大小

int nb_samples/channel_layout 平均每个通道的采样数/通道布局

FFmpeg KS [AVFrame]

```
uint8_t *data[8];                   //数据
int linesize[8];                    //数据大小
int format;                         //AVSampleFormat or AVPixelFormat
int key_frame;                      //关键帧
int64_t pts;                        //时间戳
int width, height;                  //视频宽高
enum AVPictureType pict_type;       //视频帧类型
int nb_samples;                     //当前采样数
AVRational sample_aspect_ratio;     //宽高比
int sample_rate;                    //采样率
uint64_t channel_layout;            //声道布局
int channels;                       //声道数
int64_t pkt_pts;                    //PKT - PTS
int64_t pkt_dts;                    //PKT - DTS
int64_t pkt_pos;                    //PKT - 位置
int pkt_size;                       //PKT - 大小
int64_t pkt_duration;               //PKT - 时长
void *opaque;
... ... 其他等
```

FFmpeg KS[AVPixelFormat]

- ColorSpace(ColorFormat):

分类有YUV/YCbCr/YPbPr(PAL)视差、RGB色差、CMY印刷

YUV：yuv420、yuv422、yuv444、yuv411

RGB：rgb24、rgb32(rgba)

- FourCC(PixelFormat):

根据4个字节唯一标示的PixelFormat，方便传输规定。

（标准: http://www.fourcc.org/ ）

YUV PixelFormat分为2类Packed、Planar，其实也是2种数据(传输和存储)模式,VGA 真彩色RGB24 3X5=15根引脚，123(RGB)-678(G)传输RGB24。

YV12 3个平面不好传输,时钟周期同步的NV12（2个引脚）,传输yuv420数据。

- FFmpeg PixelFormat:

没有使用FourCC标准标示，自己定义的一套内部标示，还是遵从标准的。

FFmpeg KS[AVSampleFormat]

```
enum AVSampleFormat {
    AV_SAMPLE_FMT_NONE = -1,
    AV_SAMPLE_FMT_U8,          ///< unsigned 8 bits
    AV_SAMPLE_FMT_S16,         ///< signed 16 bits
    AV_SAMPLE_FMT_S32,         ///< signed 32 bits
    AV_SAMPLE_FMT_FLT,         ///< float
    AV_SAMPLE_FMT_DBL,         ///< double

    AV_SAMPLE_FMT_U8P,         ///< unsigned 8 bits, planar
    AV_SAMPLE_FMT_S16P,        ///< signed 16 bits, planar
    AV_SAMPLE_FMT_S32P,        ///< signed 32 bits, planar
    AV_SAMPLE_FMT_FLTP,        ///< float, planar
    AV_SAMPLE_FMT_DBLP,        ///< double, planar

    AV_SAMPLE_FMT_NB           ///< Number of sample formats. DO NOT USE if linking
     dynamically
};
int
```

FFmpeg KS [PTS/DTS]

AVFrame 原始数据经过编解码后，有序存储 在 内存中：

int64 pts 编码输入顺序（显示顺序）

int64 dts 编码输出顺序 （解码顺序）

视频： int pict_type帧类型（I,P,B,S,SI,SP）

音频： 字节流 -> 编码 -> 数据帧

AVPacket 准备封装/协议处理的数据流：

int64 pts 显示时间

int64 dts 编解码时间/解码顺序

duration 持续时间

pos 容器中的位置

FFmpeg KS [AVPacket]

```
uint8_t *data;          //字节流数据
int    size;            //数据大小
int64_t pts;            //显示时间戳
int64_t dts;            //编解码时间戳
int    duration;        //时长
int64_t pos;            //位置
int    stream_index;    //数据流所属容器索引ID值
int    flags;           //包类型
void   *priv;
... ... 其他等
```

FFmpeg KS [编解码接口]

- FFmpeg编解码抽象AVCodec核心成员：

init 初始化

open 打开编解码器

decode/encode 编解码操作

flush/update 编解码刷新

close 关闭编解码器

FFmpeg KS [容器接口]

- FFmpeg封装层AVFotmat核心成员：

输入：read_header、read_probe、read_packet、read_close 、read_seek 、read_play 、read_pause 、read_timestamp等

输出：write_header 、write_packet、av_write_trailer等。

FFmpeg KS [协议接口]

- FFmpeg协议抽象层AVIO核心成员：

Open/Close 打开和关闭数据流

Seek 定位数据位置

Read/Write 读写数据

Pause/Shutdown 暂停/停止

Check 检测

FFmpeg KS [其他接口]

- FFmpeg AVHWAccel硬件加速层:

start_frame/end_frame 编码控制

decode_slice 解码控制

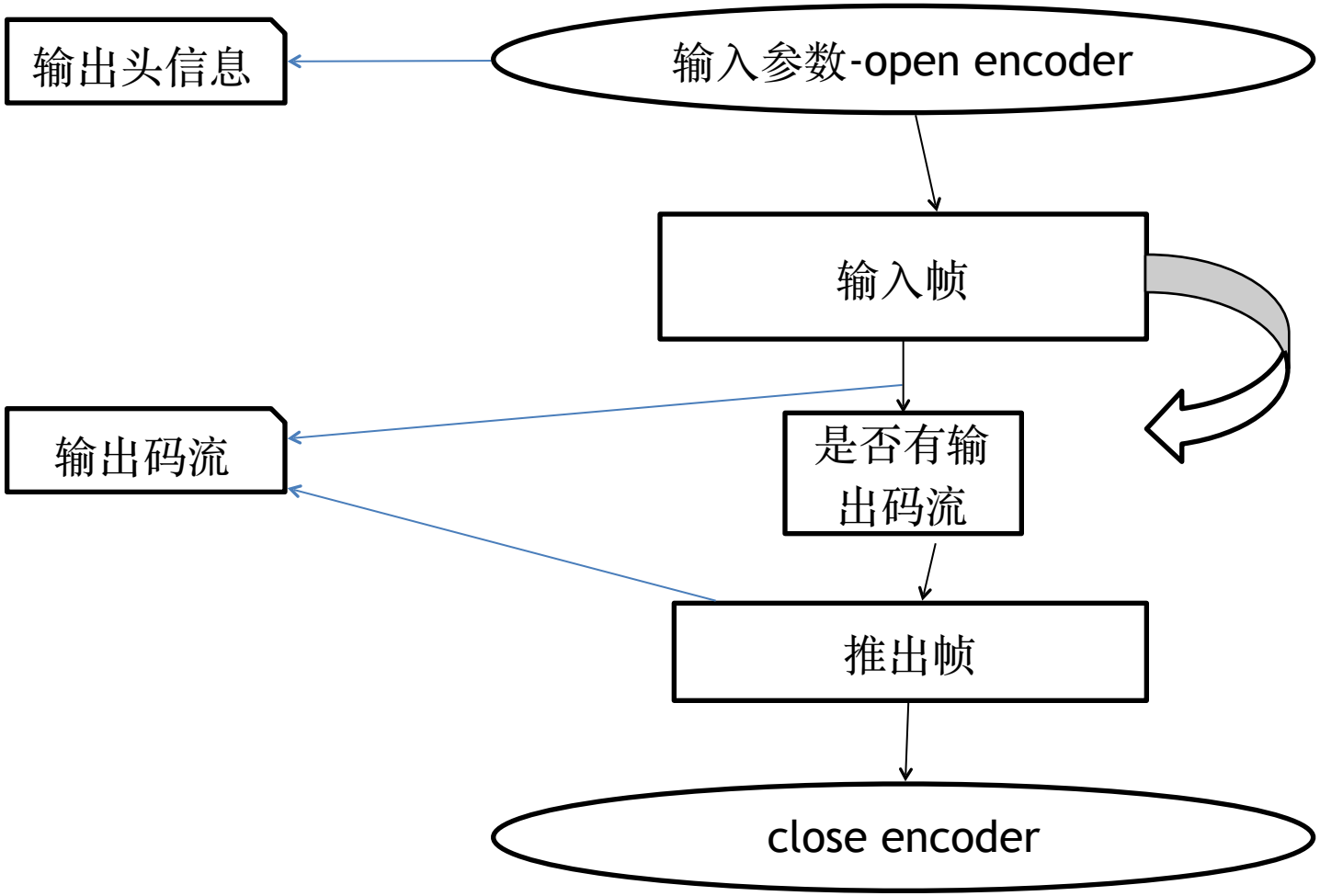
- FFmpeg网络层:

异步支持poll,iocp仅是提供接口框架;

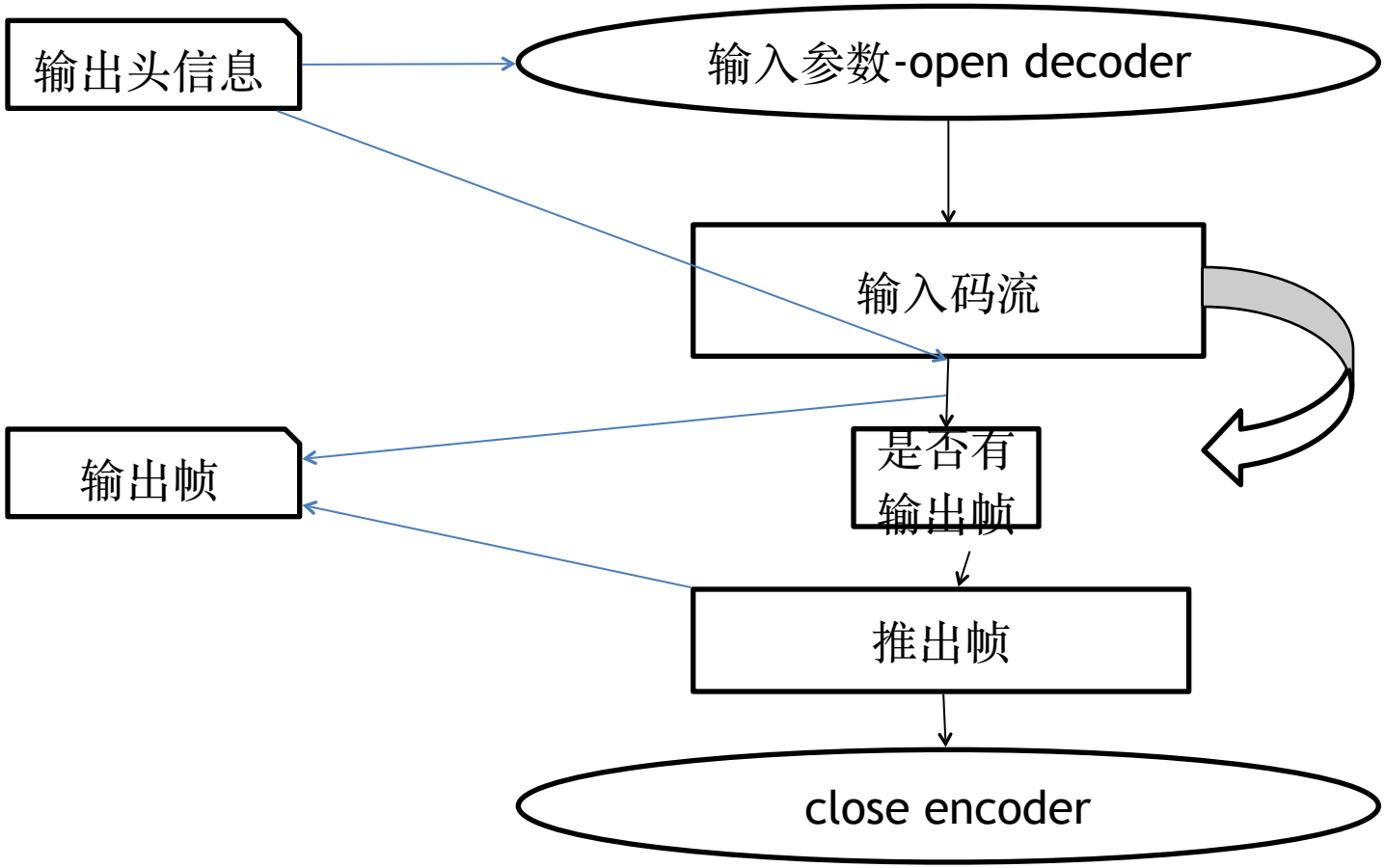
同步代码内嵌支持。

- FFmpeg输入输出设备支持视为容器。

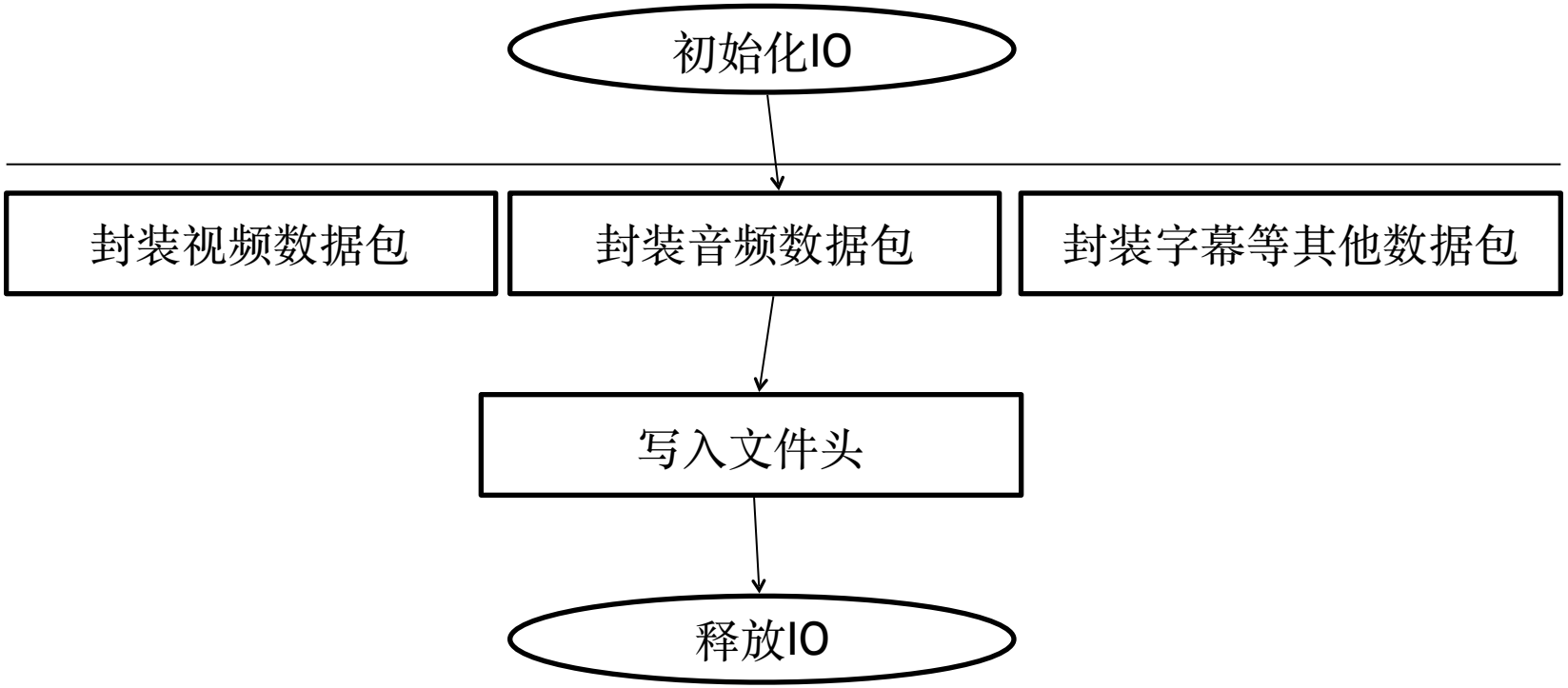
FFmpeg 编码过程



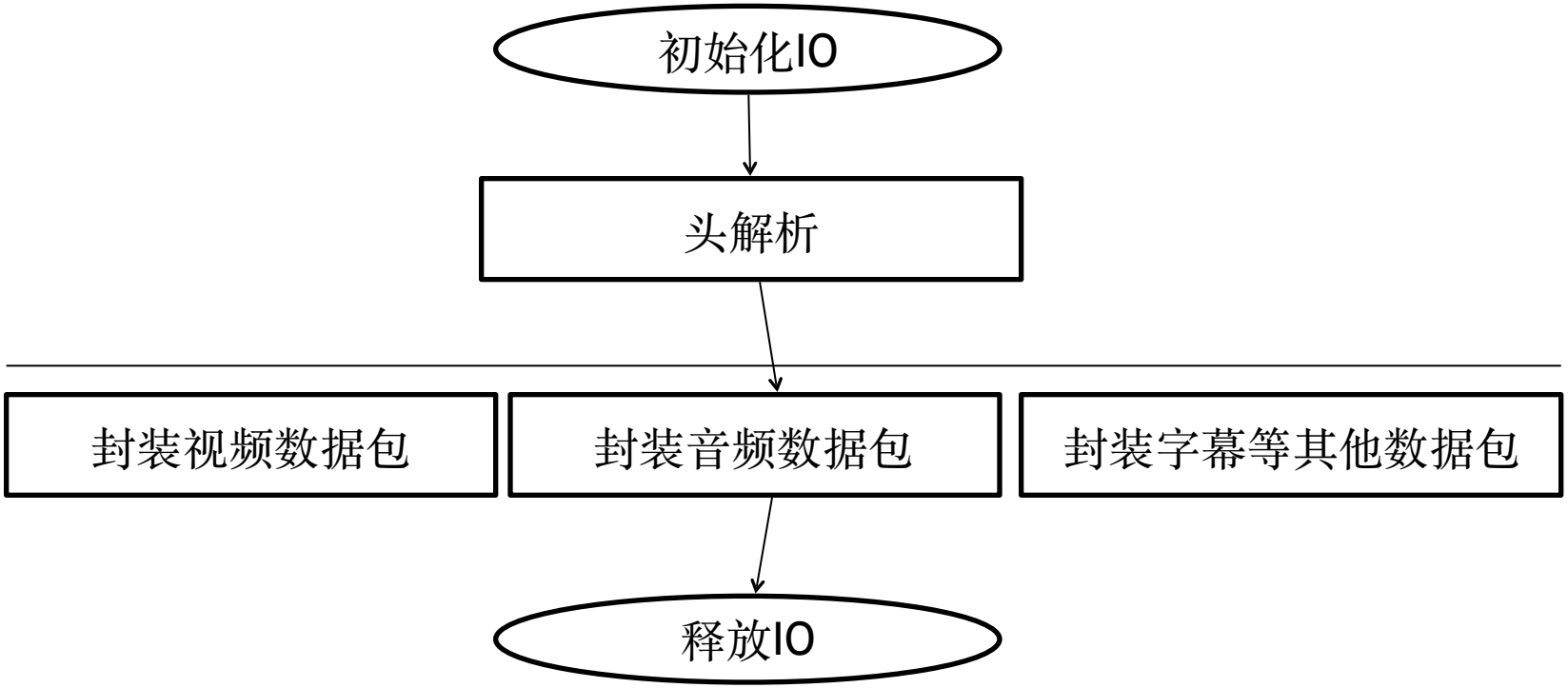
FFmpeg 解码过程



FFmpeg Muxer



FFmpeg DeMuxer



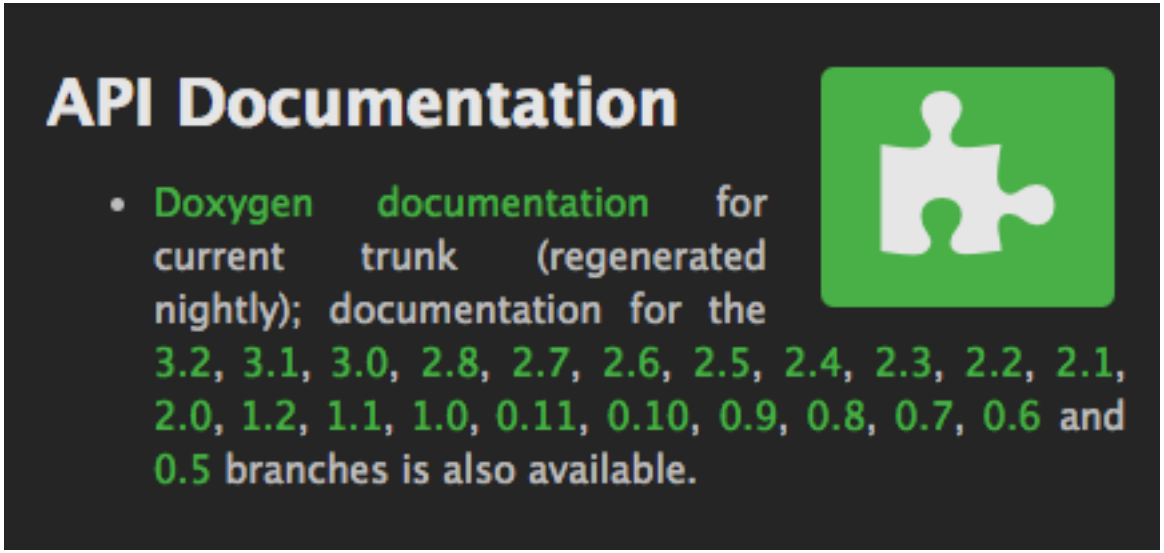
FFmpeg Codec 新接口

- 编码:
  - avcodec_send_frame
  - avcodec_receive_packet
- 解码:
  - avcodec_send_packet
  - avcodec_receive_frame

FFmpeg 代码使用示例

doc/examples

http://www.ffmpeg.org/doxygen/trunk/examples.html



小结

FFmpeg KS：

AVFrame、AVPacket、AVCodec、AVCodecContext、AVInputFormat、AVOutputFormat、AVFormatContext、AVStream、SwrContext、SwsContext、AVFilterGraph、AVFilter等等。

FFmpeg 功能：

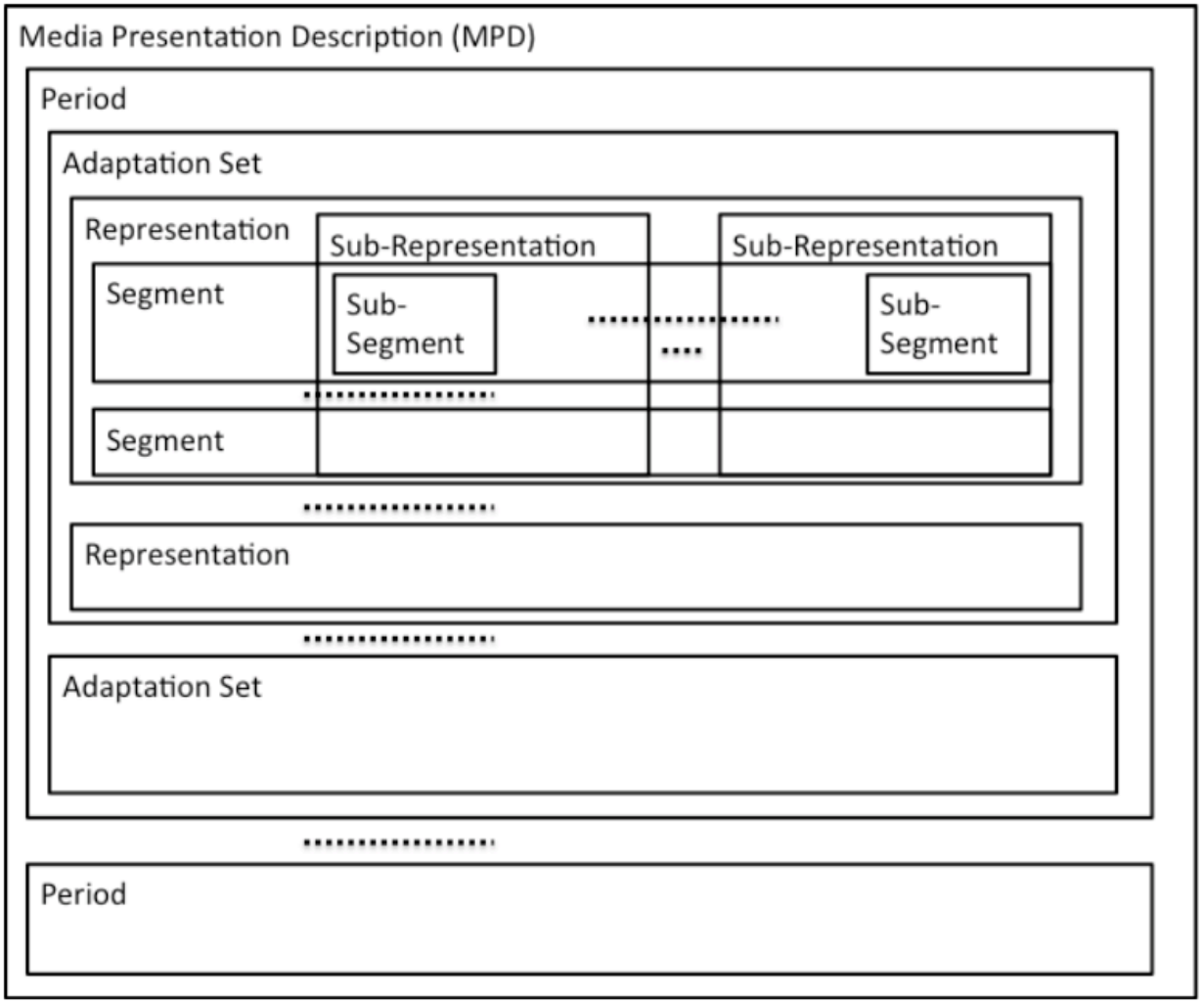
编解码、容器封装解析、AVIO、音频原始数据转换（重采样）、视频原始数据转换（色彩空间转换）、音视频处理（AVFilter）、后期处理（PostProc）。

DASH Demuxer

ISO-23009

XML

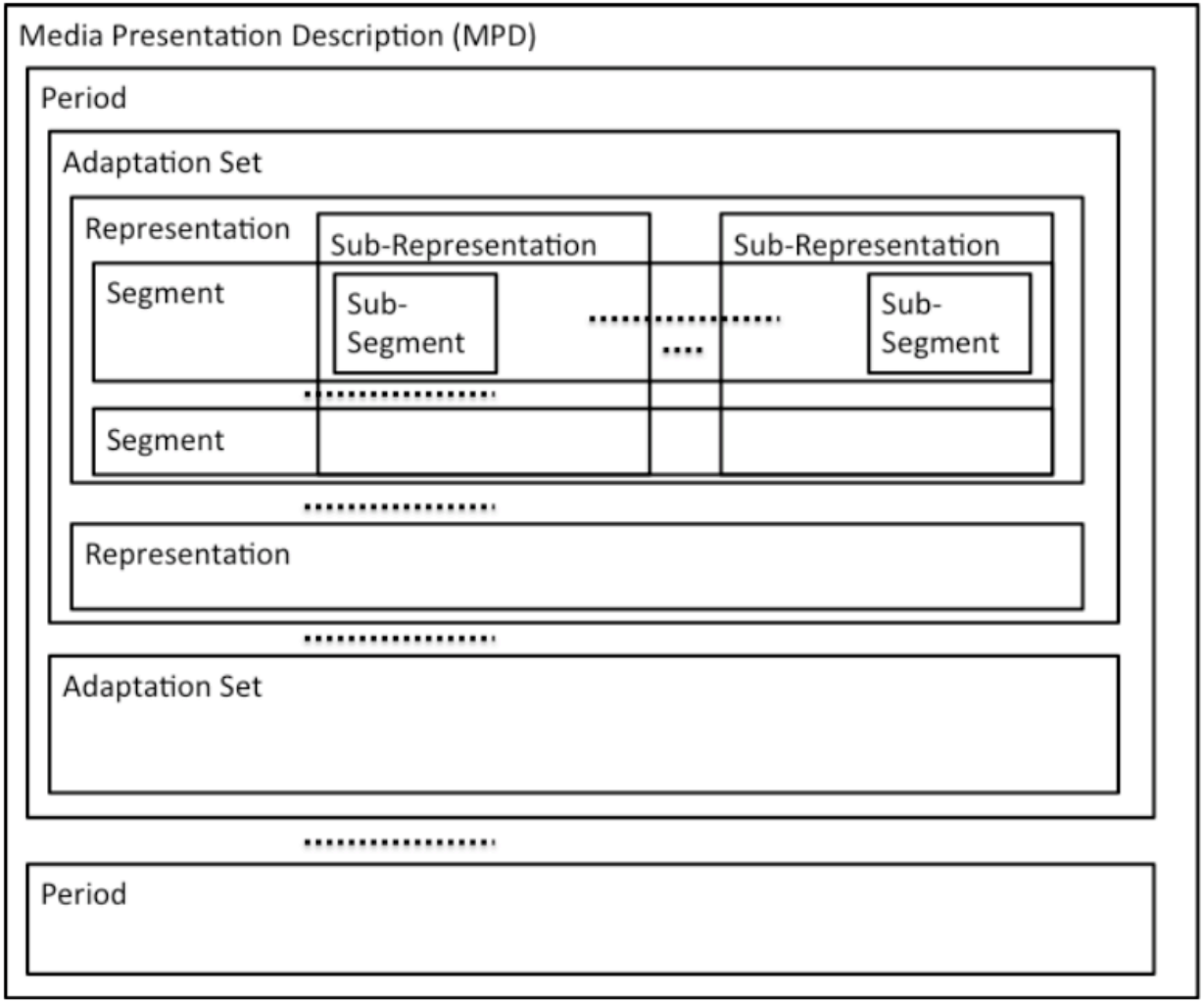
HLS



DASH Demuxer

SegmentBase
SegmentList
SegmentList

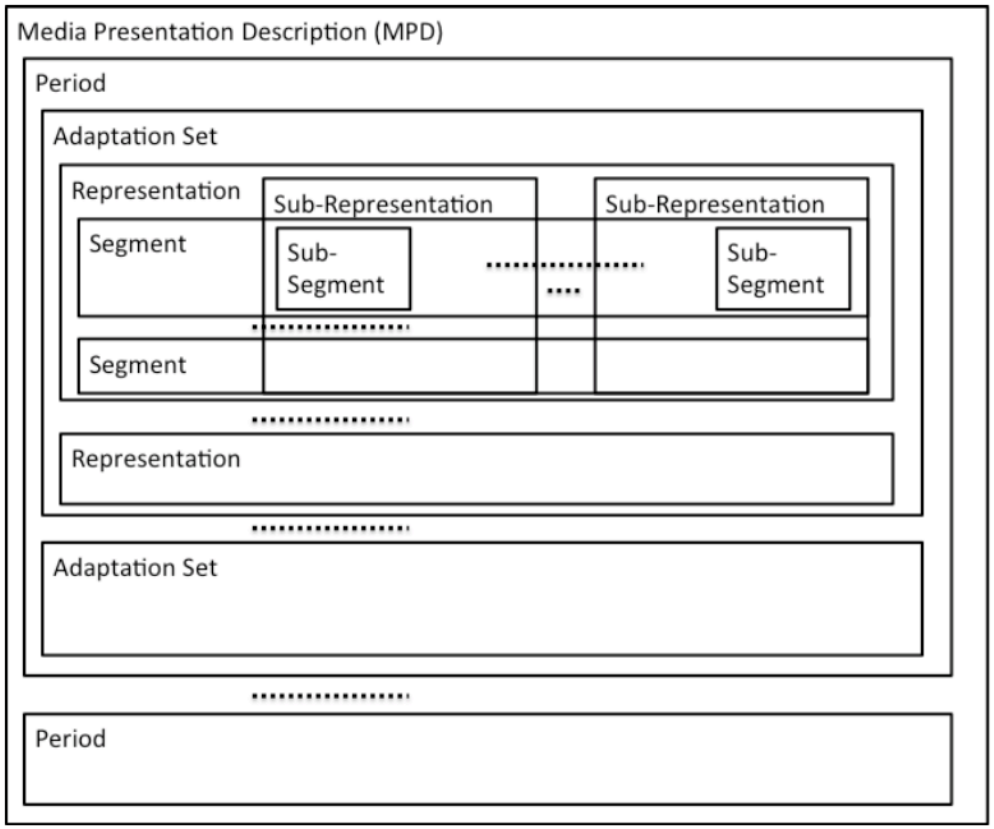
Template:
$Number$
$Time$
$Bandwidth$



DASH Demuxer

Timeline

MPD-timeShiftBufferDepth



FFmpeg - 社区能力

https://trac.ffmpeg.org

https://ffmpeg.org/documentation.html

https://wiki.multimedia.cx/index.php/Main_Page

https://trac.ffmpeg.org/timeline

https://ffmpeg.org/contact.html

https://bbs.chinaffmpeg.com

FFmpeg - maillist

ffmpeg-user

libav-user

ffmpeg-devel

FFmpeg - 交流基本规则

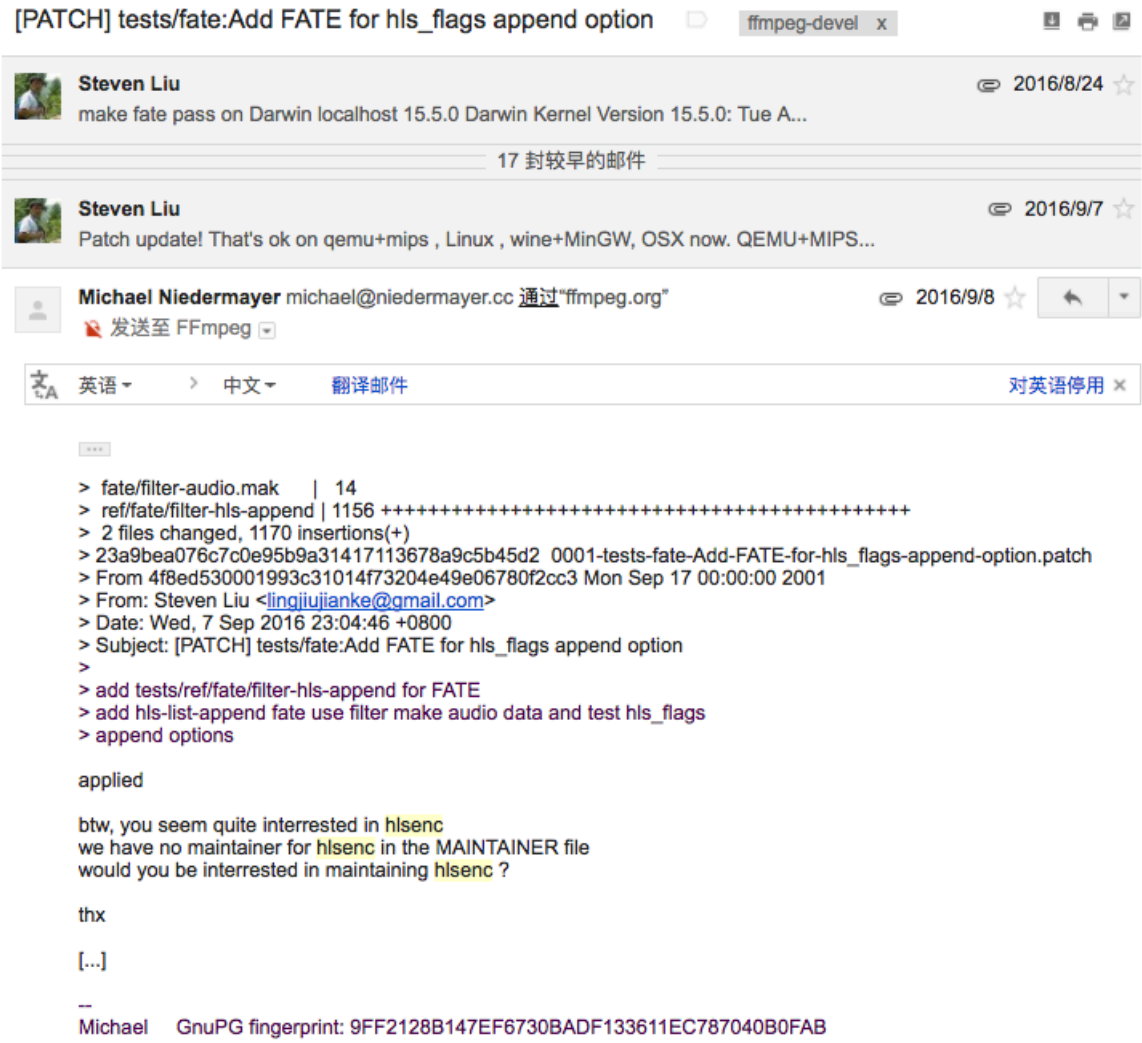
- 不在邮件最上回复
- 生成 patch 使用 git format-patch
- 提交 patch 使用 git send-email
- 提交 patch 前使用 tools/patcheck 确认规则

FFmpeg - 项目管理

- 自测 - FATE
- 代码检测 - Coverity
- Fuzz
- BUG 及 wiki - Trac
- CodeReview 及 沟通 - Maillist
- Patch 过程管理 PatchWork

如何成为maintainer

- 多撸代码



[PATCH] tests/fate:Add FATE for hls_flags append option

ffmpeg-devel x

Steven Liu 2016/8/24

make fate pass on Darwin localhost 15.5.0 Darwin Kernel Version 15.5.0: Tue A...

17 封较早的邮件

Steven Liu 2016/9/7

Patch update! That's ok on qemu+mips , Linux , wine+MinGW, OSX now. QEMU+MIPS...

Michael Niedermayer michael@niedermayer.cc 通过"ffmpeg.org" 2016/9/8

发送至 FFmpeg

英语 › 中文 翻译邮件 对英语停用 ×

> fate/filter-audio.mak | 14
> ref/fate/filter-hls-append | 1156 ++
> 2 files changed, 1170 insertions(+)
> 23a9bea076c7c0e95b9a31417113678a9c5b45d2 0001-tests-fate-Add-FATE-for-hls_flags-append-option.patch
> From 4f8ed530001993c31014f73204e49e06780f2cc3 Mon Sep 17 00:00:00 2001
> From: Steven Liu <lingjiujianke@gmail.com>
> Date: Wed, 7 Sep 2016 23:04:46 +0800
> Subject: [PATCH] tests/fate:Add FATE for hls_flags append option
>
> add tests/ref/fate/filter-hls-append for FATE
> add hls-list-append fate use filter make audio data and test hls_flags
> append options

applied

btw, you seem quite interrested in hlsenc
we have no maintainer for hlsenc in the MAINTAINER file
would you be interrested in maintaining hlsenc ?

thx

[...]

--
Michael GnuPG fingerprint: 9FF2128B147EF6730BADF133611EC787040B0FAB

如何成为maintainer

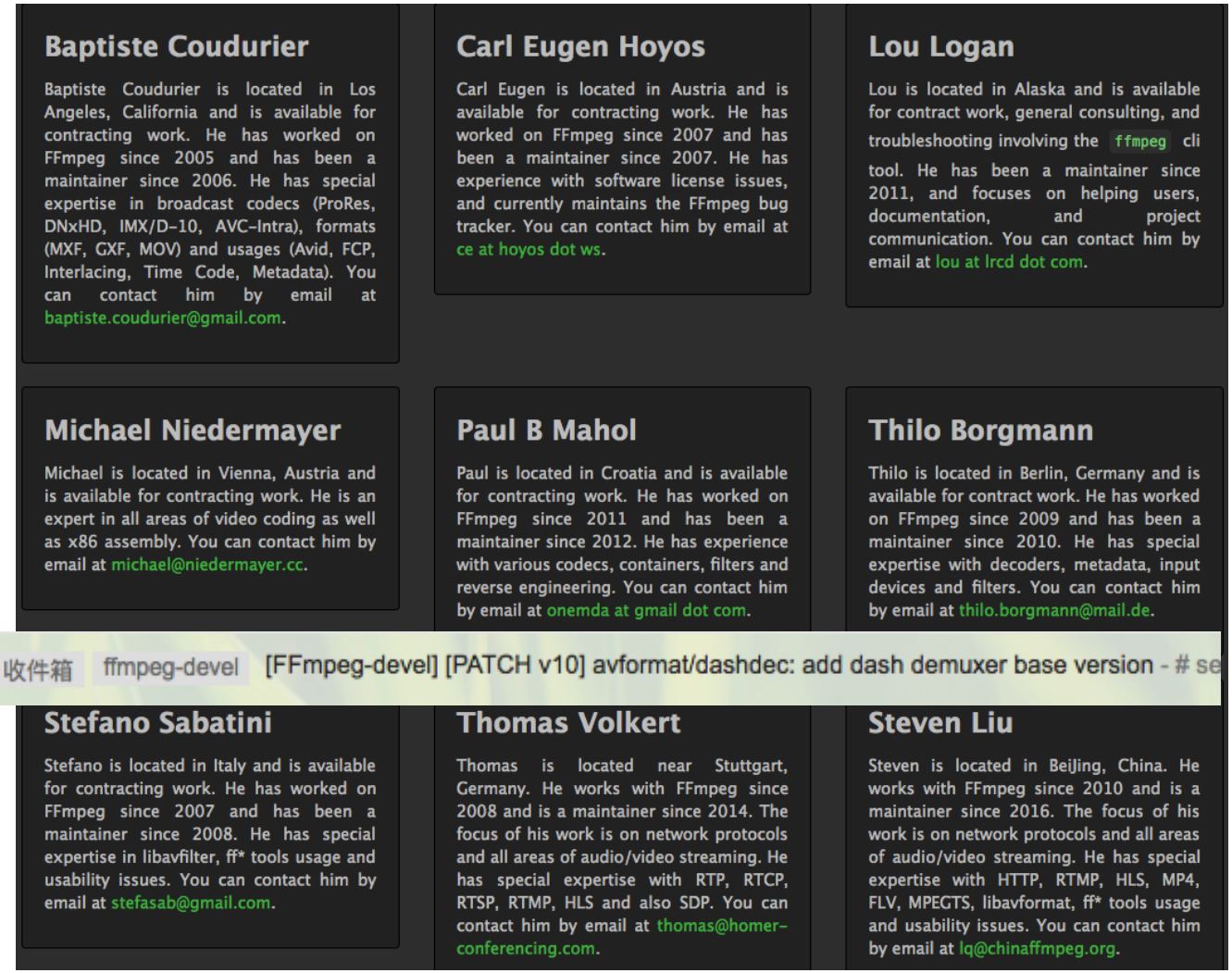
- 多撸代码

```
liuqideMBP:ffmpeg liuqi$ git log | grep  "Author: Steven Liu" | wc -l
      81
liuqideMBP:ffmpeg liuqi$ git config -l
credential.helper=osxkeychain
user.email=lingjiujianke@gmail.com
user.name=Steven Liu
core.repositoryformatversion=0
core.filemode=true
core.bare=false
core.logallrefupdates=true
core.ignorecase=true
core.precomposeunicode=true
remote.origin.url=git://source.ffmpeg.org/ffmpeg.git
remote.origin.fetch=+refs/heads/*:refs/remotes/origin/*
branch.master.remote=origin
branch.master.merge=refs/heads/master
branch.cmts.remote=origin
branch.cmts.merge=refs/heads/master
liuqideMBP:ffmpeg liuqi$
```

```
403   flvdec.c                          Michael Niedermayer
404   flvenc.c                          Michael Niedermayer, Steven Liu
405   gxf.c                             Reimar Doeffinger
406   gxfenc.c                          Baptiste Coudurier
407   hls.c                             Anssi Hannula
408   hls encryption (hlsenc.c)         Christian Suloway, Steven Liu
409   idcin.c                           Mike Melanson
410   idroqdec.c                        Mike Melanson
411   iff.c                             Jaikrishnan Menon
```

如何成为maintainer

- 多撸代码



如何开始

- 把官方文档撸一遍
- 学会使用
- 发现不足
- 改善不足
- Review别人的代码

学生如何开始更好

- GSoC（Google Summer of Code）

Derain Filter

Description: Use deep learning derain techniques to implement a filter which can remove the rain of the input image or video. The student should evaluate the best suited method based on state of the art methods

Expected results: A working drain filter

Prerequisites: Good C and C++ coding skills, knowledge of machine learning and deep learning, basic familiarity with git, familiarity with at least one deep learning framework.

Qualification Task: Implement a ffmpeg derain filter which shows good performance in deraining process.

Mentor: Steven Liu (lq [at] chinaffmpeg [dot] org)

Backup Mentor: Pedro Souza (bygrandao [at] gmail [dot] com)

学生如何开始更好

- GSoC（Google Summer of Code）

Thanks