libav 是由ffmpeg中“政变”失败的部分开发者发展而来的一个完整的、跨平台的用于音频和视频录制、转换的解决方案,包含 libavcodec 编码器。

libav项目中的avconv 用于视频转换

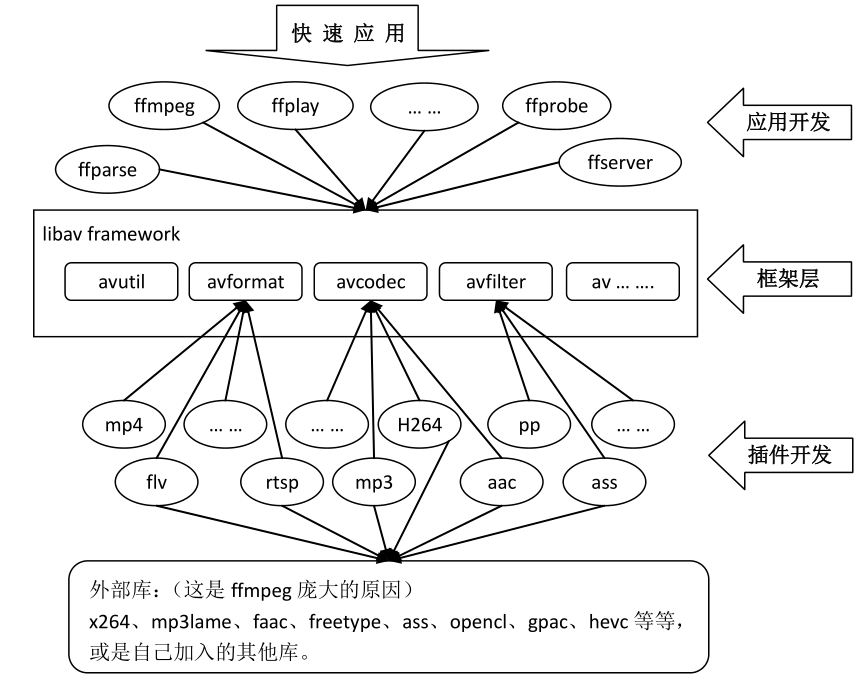
ffprobe，主要用于查看媒体文件信息，用法为ffprobe -i 文件名

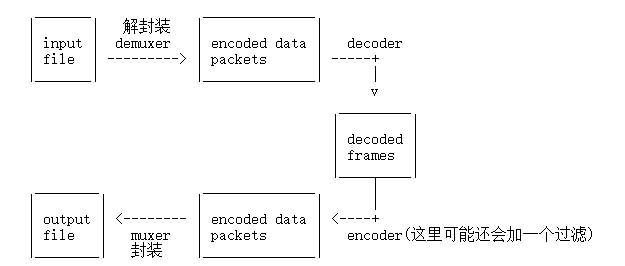
还有ffserver，这个用的很少，就不多说了

ffplay -vcodec h264\_mmal video.mp4 ffplay也可以进行硬件加速

https://trac.ffmpeg.org/wiki/HWAccelIntro 这里是对引擎的支持，可以知道对于树莓派支持MMAL和OpenMax。

FFmpeg 是多媒体程序核心开源库。FFmpeg 之所以是很好的多媒体开发的入门程序，因为它的开发人员也都慢慢成为其他开源工程的核心工作者，例如从 FFmpeg 开发者中分离到 x264、lamemp3、live555、aac、hevc、librtmp、sip、mplay、vlc、gpac、flv、preprocessing、postprocessing 等等。





ffmpeg is a very fast video and audio converter that can also grab from a live audio/video source.

1.可以改变视频的播放速度，改变视频的帧频（视频就是图像流（单位帧）与声音流）

2.分离视频音频流

3.视频解复用

4.视频转码

4.5 基本命令介绍 <https://gist.github.com/protrolium/e0dbd4bb0f1a396fcb55>

5.视频截图、剪切：

ffmpeg -i test.avi -r 1 -f image2 image-%3d.jpeg //提取图片,%3d意味着最后会是001、002、003...

-qscale:v (or the alias -q:v) 使用这个参数似乎可以加强输出图片文件的质量，反正肯定是有参数的

H264---(解码)-->YUV420p（转换像素格式）---(编码).---(封装)--->JPEG

ffmpeg -ss 0:1:30 -t 0:0:20 -i input.avi -vcodec copy -acodec copy output.avi //剪切视频

//-r 提取图像的频率，-ss 开始时间（摄像头一开始的开启从暗转亮），-t 持续时间

6.视频录制/获取：

ffmpeg –i rtsp://192.168.3.205:5555/test –vcodec copy out.avi

可以直接 -i /dev/video0来指定摄像头，但是树莓派的csi摄像头要先更改/etc/modules才能在/dev/videoxxx

ffmpeg -f video4linux2 -list\_formats all -i /dev/video0来查询支持设备

ffmpeg -i input%03d.jpg -f framehash - #我只是举一个例子告诉可以这样输入图片

ffmpeg输出到shell不是标准输出，并不能被重定向，而且它的刷新显示仅仅针对最后一行的。

-an disable audio 所以产生的就是无声视频，没有-an似乎会出现问题

-i - 前面是指解码，后面是编码插件。-c:v或者-vcodec指定中间件

ffmeg转mov：-i a.mov -c copy a.mp4

软编码：使用CPU进行编码

硬编码：使用非CPU进行编码，如显卡GPU、专用的DSP、FPGA、ASIC芯片等。GPU可以支持视频解码,而且在ffmpeg中也提供了解码的接口,常见的一个是NVIDIA的VDPAU,一个是intel发明的VAAPI。

ffmpeg就可以自己调用啦。在编译ffmpeg的时候需要注意其中有两个比较关键的东西叫omx-rpi和mmal：

omx-rpi 树莓派的硬编码加速

mmal 树莓派的硬解码加速（raspivid、raspistill使用）

FFmpeg 3.1 adds support for OpenMAX encoding on Raspberry Pi, VA-API H.264 & H.265 Encoding, and more。使用了硬编码并不是说CPU的消耗基本没有，有的命令会以最高的帧率来编码，会耗费大量的cpu在内存拷贝上，也会耗费cpu。OMX只是一个框架，所以解码器是硬解还是软解都没关系，而且大多说厂家的硬解码都是使用OMX框架。openMAX是推出的一个标准，主要统一多媒体来使用codec。主要由 OMXCore 和OMXComponent组成，Core和Component分别需要实现一些接口，OMX\_Core.h OMX\_Component.h。解码出Video 需要进行 render 之后 再送到硬件输出。

开启硬编码ffmpeg配置：h264\_omx h264\_mmal 还开了一些必要的，关闭了一些不要的工具，从而直接在树莓派上编译

git clone git://git.videolan.org/x264.git

./configure \

--disable-shared --enable-static \

--enable-strip \

--disable-cli

git clone git://source.ffmpeg.org/ffmpeg.git

另外ffmpeg中的播放器程序ffplay依赖sdl来显示视频，sdl是一个图形显示库，安装命令：

sudo apt-get install libsdl2-dev

./configure --enable-gpl --enable-version3 --enable-nonfree --enable-static --enable-libx264 --enable-encoder=libx264 --enable-decoder=h264 --enable-encoder=aac --enable-decoder=aac --enable-encoder=ac3 --enable-decoder=ac3 --enable-encoder=rawvideo --enable-decoder=rawvideo --enable-encoder=mjpeg --enable-decoder=mjpeg --enable-muxer=flv --enable-demuxer=flv --enable-muxer=image2 --enable-demuxer=image2 --enable-muxer=mp4 --enable-demuxer=mpegvideo --enable-muxer=matroska --enable-demuxer=matroska --enable-muxer=wav --enable-demuxer=wav --enable-muxer='pcm\*' --enable-demuxer='pcm\*' --enable-muxer=rawvideo --enable-demuxer=rawvideo --enable-parser=h264 --enable-parser=aac --enable-protocol=file --enable-protocol=tcp --enable-protocol=rtmp --enable-protocol=cache --enable-protocol=pipe --enable-filter=aresample --enable-filter=allyuv --enable-filter=scale --enable-indev=v4l2 --enable-indev=alsa --enable-omx --enable-omx-rpi --enable-encoder=h264\_omx --enable-mmal --enable-hwaccel=h264\_mmal --enable-decoder=h264\_mmal 没想到生成编译的配置文件都需要一段时间

ffmpeg -hwaccels 查看支持的硬件加速方法（树莓派是支持vaapi硬件加速方法）

ffmpeg -vaapi\_device /dev/dri/renderD128 -hwaccel vaapi -hwaccel\_output\_format yuv420p ……

ffmpeg -f v4l2 -r 15 -video\_size vga -pix\_fmt yuv420p -i /dev/video0 -b:v 500k -c:v h264\_omx -an  -f flv /tmp/test.flv 2%

ffmpeg -f v4l2 -r 15 -video\_size vga -input\_format mjpeg -i /dev/video0 -b:v 500k -c:v h264\_omx -an -f flv /tmp/test.flv -y会占用63%

#ffmpeg -f video4linux2 -list\_formats all -i /dev/video0这个可以查看到mjpeg，h264都是Compressed，而yuv420p是raw

[video4linux2,v4l2 @ 0x3180250] Raw : yuv420p : Planar YUV 4:2:0 : {32-2592, 2}x{32-1944, 2}

[video4linux2,v4l2 @ 0x3180250] Raw : yuyv422 : YUYV 4:2:2 : {32-2592, 2}x{32-1944, 2}

[video4linux2,v4l2 @ 0x3180250] Raw : rgb24 : 24-bit RGB 8-8-8 : {32-2592, 2}x{32-1944, 2}

[video4linux2,v4l2 @ 0x3180250] Compressed: mjpeg : JFIF JPEG : {32-2592, 2}x{32-1944, 2}

[video4linux2,v4l2 @ 0x3180250] Compressed: h264 : H.264 : {32-2592, 2}x{32-1944, 2}

[video4linux2,v4l2 @ 0x3180250] Compressed: mjpeg : Motion-JPEG : {32-2592, 2}x{32-1944, 2}

[video4linux2,v4l2 @ 0x3180250] Raw : Unsupported : YVYU 4:2:2 : {32-2592, 2}x{32-1944, 2}

[video4linux2,v4l2 @ 0x3180250] Raw : Unsupported : VYUY 4:2:2 : {32-2592, 2}x{32-1944, 2}

[video4linux2,v4l2 @ 0x3180250] Raw : uyvy422 : UYVY 4:2:2 : {32-2592, 2}x{32-1944, 2}

[video4linux2,v4l2 @ 0x3180250] Raw : nv12 : Y/CbCr 4:2:0 : {32-2592, 2}x{32-1944, 2}

[video4linux2,v4l2 @ 0x3180250] Raw : bgr24 : 24-bit BGR 8-8-8 : {32-2592, 2}x{32-1944, 2}

[video4linux2,v4l2 @ 0x3180250] Raw : yuv420p : Planar YVU 4:2:0 : {32-2592, 2}x{32-1944, 2}

[video4linux2,v4l2 @ 0x3180250] Raw : Unsupported : Y/CrCb 4:2:0 : {32-2592, 2}x{32-1944, 2}

[video4linux2,v4l2 @ 0x3180250] Raw : bgr0 : 32-bit BGRA/X 8-8-8-8 : {32-2592, 2}x{32-1944, 2}

所以我就认为将原来的流压缩转换为mjpeg流是CPU在做，于是会有不小的CPU消耗。

ffmpeg -hwaccel vaapi -hwaccel\_device /dev/dri/renderD128 -hwaccel\_output\_format yuv420p -i - -r 0.2 -f image2 /tmp/webCam/%d.jpg

如果使用了-hwaccel 但是不给出自己的-hwaccel\_device

The device selection for VAAPI works as follows, stopping at the first usable device it finds:

1) If the user passed a device name, try to open that name an X11 display.

2) If they didn't, try to open the default X11 display (i.e. $DISPLAY).

3) If the user passed a device name, try to open that name as the path to a DRM device.

4) If they didn't, try to open /dev/dri/renderD128 as a DRM device.

rasp-config 调节GL driver 就能出现/dev/dri/renderD128了（启用 VC4 开源驱动），KMS refers to Kernel Mode-Setting。

ffmpeg -vcodec h264\_mmal -i - -r 2 -f image2 /tmp/webCam/%d.jpg