FFmpeg 常用命令与材料

AIHGF 2020 年 10 月 06 日 2330次浏览 暂无评论 18565字数 操作系统 Python

> FFmpeg 官方文档 - https://www.ffmpeg.org/about.html

FFmpeg 是视频处理最常用的开源软件. 这里汇总了一些常用命令及材料，以记录备忘.

Ubuntu 安装:

```
sudo apt install ffmpeg
```

查看后台运行的 ffmpeg 命令:

```
ps -ef|grep ffmpeg
```

1. FFmpeg 基础知识

[1] - 支持格式

```
ffmpeg -formats
```

如：AVI, MP4, MKV, DV, 3GP 等.

[2] - 编码格式

视频和音频都需要经过编码，才能保存成文件.

不同的编码格式$CODEC$，有不同的压缩率，会导致文件大小和清晰度的差异.

```
ffmpeg -codecs
```

如，H.262，H.264，H.265，MP3 等.

[3] - 编码器

编码器是实现某种编码格式的库文件. 只有安装了某种格式的编码器，才能实现该格式视频/音频的编码和解码.

```
ffmpeg -encoders
```

如内置的音视频编码器有:

```
#视频
libx264:最流行的开源 H.264 编码器
NVENC:基于 NVIDIA GPU 的 H.264 编码器
libx265:开源的 HEVC 编码器
libvpx:谷歌的 VP8 和 VP9 编码器
libaom:AV1 编码器
#音频
libfdk-aac
aac
```

2. FFmpeg 命令行格式

```
ffmpeg {arg1} {arg2} -i {arg3} {arg4} {arg5}
```

其中，

- arg1 - 全局参数
- arg2 - 输入文件参数
- arg3 - 输入文件
- arg4 - 输出文件参数
- arg5 - 输出文件

例如：

```
ffmpeg \
    -y \ # 全局参数
    -c:a libfdk_aac -c:v libx264 \ # 输入文件参数
    -i input.mp4 \ # 输入文件
    -c:v libvpx-vp9 -c:a libvorbis \ # 输出文件参数
    output.webm # 输出文件
```

其作用为，将 mp4 文件转成 webm 文件. 输入的 mp4 文件的音频编码格式是 aac，视频编码格式是 H.264；输出的 webm 文件的视频编码格式是 VP9，音频格式是 Vorbis.

也可以不指名编码格式，FFmpeg 会自行判断输入文件的编码，如：

```
ffmpeg -i input.avi output.mp4
```

3. FFmpeg 常用命令

FFmpeg 常用的命令行参数有：

```
-c:指定编码器
-c copy:直接复制，不经过重新编码(这样比较快)
-c:v:指定视频编码器
-c:a:指定音频编码器
-i:指定输入文件
-an:去除音频流,不处理音频
-vn:去除视频流,不处理视频
-preset:指定输出的视频质量，会影响文件的生成速度，候选值有:ultrafast, superfast, veryfast, faster, fast, medium, slow, slower, veryslow.
-y:不经过确认，输出时直接覆盖同名文件
-vcodec: 设定视频编解码器，未设定时则使用与输入流相同的编解码器
-b: 设定视频流量，默认为200Kbit/s
-r: 设定帧速率，默认为25
-s: 设定画面的宽与高
```

```
-aspect: 设定画面的比例
-ss: 开始时间

# 音频参数
-ar: 设定采样率
-ac: 设定声音的Channel数
-acodec: 设定声音编解码器，未设定时则使用与输入流相同的编解码器
-an: 不处理音频
```

[1] - **查看视频文件的元信息**

比如编码格式和比特率.

```
ffmpeg -i input.mp4
```

去除元信息外的冗余信息, 只显示元信息：

```
ffmpeg -i input.mp4 -hide_banner
```

[2] - **转换视频编码格式**

将视频文件从一种编码转成另一种编码.

如：

```
#转码为码流原始文件
ffmpeg -i test.mp4 -vcodec h264 -s 352*278 -an -f m4v test.264
#转码为码流原始文件
ffmpeg -i test.mp4 -vcodec h264 -bf 0 -g 25 -s 352*278 -an -f m4v test.264
#转码为封装文件
ffmpeg -i test.avi -vcodec mpeg4 -vtag xvid -qsame test_xvid.avi
#-bf B帧数目控制，-g 关键帧间隔控制，-s 分辨率控制
```

如转成 H.264 编码，一般使用编码器 `libx264`，所以只需指定输出文件的视频编码器.

```
ffmpeg -i input_file -c:v libx264 output.mp4 #转换成标准的264编码
```

转成 H.265 编码:

```
ffmpeg -i input_file -c:v libx265 output.mp4
```

[3] - **调整码率**

视频码率就是数据传输时单位时间传送的数据位数，一般用的单位是kbps即千位每秒. 通俗一点的理解就是取样率，单位时间内取样率越大，精度就越高，处理出来的文件就越接近原始文件.

码率：影响体积，与体积成正比：码率越大，体积越大；码率越小，体积越小.

如，指定码率最小为964K，最大为3856K，缓冲区大小为 2000K.

```
ffmpeg -i input.mp4 -minrate 964K -maxrate 3856K -bufsize 2000K output.mp4
```

[4] - **改变分辨率**

如 1080p 转 480p.

```
ffmpeg -i input.mp4 -vf scale=480:-1 output.mp4
```

[5] - **提取视频中的音频**

如：

```
ffmpeg -i input.mp4 -vn -c:a copy output.aac
```

`-vn`表示去掉视频，`-c:a copy`表示不改变音频编码，直接拷贝.

[6] - **分离视频音频流**

```
ffmpeg -i input_file -vcodec copy -an output_file_video  //分离视频流
ffmpeg -i input_file -acodec copy -vn output_file_audio  //分离音频流
```

[7] - **视频抽帧截图**

如，从指定时间开始，连续对1秒钟的视频进行截图：

```
ffmpeg -y -i input.mp4 -ss 08:01:57 -t 00:00:01 output_%3d.jpg
```

如，指定只截取一帧：

```
ffmpeg -ss 08:01:57 -i input -vframes 1 -q:v 2 output.jpg
```

`-vframes 1`指定只截取一帧，`-q:v 2`表示输出的图片质量，一般是1到5之间（1为画质最高）

[8] - **视频片段裁剪**

截取原始视频里面的一个片段，输出为一个新视频.

可以指定开始时间$start$和持续时间$duration$，也可以指定结束时间end.

如，

```
#ffmpeg -ss [start] -i [input] -t [duration] -c copy [output]
#ffmpeg -ss [start] -i [input] -to [end] -c copy [output]

ffmpeg -ss 00:01:50 -i [input] -t 10.5 -c copy [output]
ffmpeg -ss 2.5 -i [input] -to 10 -c copy [output]
```

[9] - **视频封装**

```
ffmpeg –i video_file –i audio_file –vcodec copy –acodec copy output_file
```

[10] - **音频添加封面**

可以将音频文件，转为带封面的视频文件，如：

```
ffmpeg -loop 1 -i cover.jpg -i input.mp3 -c:v libx264 -c:a aac -b:a 192k -shortest output.mp4
```

两个输入文件，封面图片`cover.jpg`和音频文件`input.mp3`.

`-loop 1`参数表示图片无限循环，`-shortest`参数表示音频文件结束，输出视频就结束.

4. FFmpeg 常用视频流命令

[1] - 视频流保存

```
ffmpeg –i rtsp://192.168.3.205:5555/test –vcodec copy out.avi
ffmpeg -i rtmp://server/live/streamName -c copy out.flv
```

[2] - 保存hls$_{m3u8}$/rtmp的视频/直播流

```
ffmpeg -i http://example.com/xxx.m3u8 -c copy merged.ts
ffmpeg -i rtmp://example.com/xxx -c copy -f mp4 output.mp4
ffmpeg -i rtmp://example.com/xxx -c copy -f segment -segment_time 60 /video/output_video_%d.flv
```

[3] - 将文件当做直播送至live

```
ffmpeg -re -i local_file.mp4 -c copy -f flv rtmp://server/live/streamName
```

[4] - 将其中一个直播流，视频改用h264压缩，音频不变，送至另外一个直播服务流

```
ffmpeg -i rtmp://server/live/originalStream -c:a copy -c:v libx264 -vpre slow -f flv rtmp://server/live/h264Stream
```

[5] - 将其中一个直播流，视频改用h264压缩，音频改用faac压缩，送至另外一个直播服务流

```
ffmpeg -i rtmp://server/live/originalStream -c:a libfaac -ar 44100 -ab 48k -c:v libx264 -vpre slow -vpre baseline -f flv rtmp://server/live/h264Stream
```

[6] - 将其中一个直播流，视频不变，音频改用faac压缩，送至另外一个直播服务流

```
ffmpeg -i rtmp://server/live/originalStream -acodec libfaac -ar 44100 -ab 48k -vcodec copy -f flv rtmp://server/live/h264_AAC_Stream
```

[7] - 将一个高清流，复制为几个不同视频清晰度的流重新发布，其中音频不变

```
ffmpeg -re -i rtmp://server/live/high_FMLE_stream -acodec copy -vcodec x264lib -s 640×360 -b 500k -vpre medium -vpre baseline rtmp://server/live/baseline_50
```

采用`-x264opts`选项

```
ffmpeg -re -i rtmp://server/live/high_FMLE_stream -c:a copy -c:v x264lib -s 640×360 -x264opts bitrate=500:profile=baseline:preset=slow rtmp://server/live/ba
```

[8] - 将一个JPG图片经过h264压缩循环输出为mp4视频

```
ffmpeg -i input.jpg -an -vcodec libx264 -coder 1 -flags +loop -cmp +chroma -subq 10 -qcomp 0.6 -qmin 10 -qmax 51 -qdiff 4 -flags2 +dct8x8 -trellis 2 -partit
```

[9] - 将普通流视频改用h264压缩，音频不变，送至高清流服务新版本$FMSlive=1$

```

ffmpeg -i rtmp://server/live/originalStream -c:a copy -c:v libx264 -vpre slow -f flv rtmp://server/live/h264Stream live=1
```

5. python 实现实时推流

> 原文：Python 通过ffmpeg实现实时推流$ubuntu16+ffmpeg+nginx$ - 2019.08.27

主要场景是：安防项目，前端实时展示监控摄像机的经过AI模型处理后画面.

如果前端只要求展示原始画面，只需要在接入摄像机的时候，把视频流推送到一个服务器地址上，前端可根据地址获取视频流，前端借助的是一个视频流插件video.js，可拉取rtmp格式的视频流.

> 如果接入多路的摄像头，可以借助服务器Nginx + ffmpeg,具体的安装配置可参考：
>
> 利用nginx搭建RTMP视频点播、直播、HLS服务器

这里主要是关于推流实现：

```
import cv2
import queue
import os
import numpy as np
from threading import Thread
import datetime,_thread
import subprocess as sp
import time

# 使用线程锁，防止线程死锁
mutex = _thread.allocate_lock()
# 存图片的队列
frame_queue = queue.Queue()
# 推流的地址，前端通过这个地址拉流，主机的IP，2019是ffmpeg在nginx中设置的端口号
rtmpUrl="rtmp://192.168.40.145:2019/live/1"
# 用于推流的配置,参数比较多，可网上查询理解
command=['ffmpeg',
                '-y',
                '-f', 'rawvideo',
                '-vcodec','rawvideo',
                '-pix_fmt', 'bgr24',
                '-s', "{}x{}".format(640, 480),# 图片分辨率
                '-r', str(25.0), #视频帧率
                '-i', '-',
                '-c:v', 'libx264',
                '-pix_fmt', 'yuv420p',
                '-preset', 'ultrafast',
                '-f', 'flv',
                rtmpUrl]

def Video():
    # 调用相机拍图的函数
    vid = cv2.VideoCapture(0)
    if not vid.isOpened():
        raise IOError("Couldn't open webcam or video")
```

```
    while (vid.isOpened()):
        return_value, frame = vid.read()
        # 原始图片推入队列中
        frame_queue.put(frame)

def push_frame():
    # 推流函数
    accum_time = 0
    curr_fps = 0
    fps = "FPS: ??"
    prev_time = time()

    # 防止多线程时 command 未被设置
    while True:
        if len(command) > 0:
            # 管道配置, 其中用到管道
            p = sp.Popen(command, stdin=sp.PIPE)
            break
```

在处理图像的时候，最好是将原图存到队列中，再从队列中取出来做处理，之前试过将处理后的图片存到队列中，然后直接推送，发现推送的进程占用了所有的资源，导致处理图像的进程无法执行. 所以顺序不对，很容易产生资源占用的情况.

怎样查看推流是否成功，可借助vlc软件.

业余时候和几个朋友讨论过推流的问题，如果一帧一帧往前端推送，方法比较傻，前端小伙伴估计也不愿意，这里就考虑到代理服务器，服务器类似于一个容器，将图片以流的形式放到容器中，容器可以做到均衡负载，高访问量. 当然与服务器的通信协议要以UDP的形式，不容易丢包，ffmpeg内部就封装好了UDP协议，不需要自己额外的实现.

6. python 视频处理并推送流直播

> 原文：基于python2.7的opencv3.3-ffmpeg-rtmp视频处理并推送流直播 - 2018.05.21

设想：通过opencv获取视频摄像的图片帧，图像处理识别之后加工绘制图片，并把该图片作为视频流的一帧推送rtmp，然后远端直播.

- 图片帧采集 视频/ 摄像头
- 图片帧加工 识别/检验/绘制信息
- 图片帧写入 写入文件/分 写入管道直播

> 参考：
> https://stackoverflow.com/questions/36422211/processing-camera-stream-in-opencv-pushing-it-over-rtmp-nginx-rtmp-module-usi
>
> 项目github -
>
> 关键代码文件 - ServerCamera.py

ServerCamera.py 如：

```
rtmpUrl = 'rtmp://39.107.26.100:1935/myapp/test1'

mycv = CvHelp() #opencv工具类, 提供绘图识别工具

# 视频来源
filePath='/mnt/e/nginx-rtmp/'
camera = cv2.VideoCapture(filePath+"test2.mp4") # 从文件读取视频

#这里的摄像头可以在树莓派3b上使用
# camera = cv2.VideoCapture(0) # 参数0表示第一个摄像头 摄像头读取视频
```

```
# if (camera.isOpened()):# 判断视频是否打开
#     print 'Open camera'
# else:
#     print 'Fail to open camera!'
#     return
# camera.set(cv2.CAP_PROP_FRAME_WIDTH, 1280)  # 2560x1920 2217x2217 2952x1944 1920x1080
# camera.set(cv2.CAP_PROP_FRAME_HEIGHT, 720)
# camera.set(cv2.CAP_PROP_FPS, 5)

# 视频属性
size = (int(camera.get(cv2.CAP_PROP_FRAME_WIDTH)), int(camera.get(cv2.CAP_PROP_FRAME_HEIGHT)))
sizeStr = str(size[0]) + 'x' + str(size[1])
fps = camera.get(cv2.CAP_PROP_FPS)  # 30p/self
fps = int(fps)
hz = int(1000.0 / fps)
print('size:'+ sizeStr + ' fps:' + str(fps) + ' hz:' + str(hz))

# 视频文件输出
fourcc = cv2.VideoWriter_fourcc(*'XVID')
out = cv2.VideoWriter(filePath+'res_mv.avi',fourcc, fps, size)
# 直播管道输出
# ffmpeg推送rtmp 重点：通过管道共享数据的方式
command = ['ffmpeg',
    '-y',
    '-f', 'rawvideo',
    '-vcodec','rawvideo',
    '-pix_fmt', 'bgr24',
    '-s', sizeStr,
    '-r', str(fps),
    '-i', '-',
    '-c:v', 'libx264',
    '-pix_fmt', 'yuv420p',
    '-preset', 'ultrafast',
    '-f', 'flv',
    rtmpUrl]

#管道特性配置
# pipe = sp.Popen(command, stdout = sp.PIPE, bufsize=10**8)
pipe = sp.Popen(command, stdin=sp.PIPE) #,shell=False
# pipe.stdin.write(frame.tostring())

#业务数据计算
lineWidth = 1 + int((size[1]-400) / 400)# 400 1 800 2 1080 3
```

大概就是：

只通过管道pipe来使用了ffmpeg提供的rtmp推流工具. 目前也还比较快，直播延时大概2sp_c 上还行

7. python 生成RTMP流

> 原文：使用python读取网络视频流或者本地视频进行RTMP流的生成，并对视频源的每一帧做剪切处理 - 2020.04.21

RTMP推流的功能，其实是可以用**ffmpeg+ffserver**通过命令行方式实现的，但是为了对原视频流的帧做剪切处理，所以使用**python调用ffmpeg来推流**，这里只是生成推流. 这是对的需要有一个代理服务器将视频推送到网络关开代理服务器配置可参考文

以下全部代码，需要注意：

- 主体是两个函数，通过两个线程调用，线程之间通过队列进行通信，读流线程将读取到的帧放入队列中，推流线程将队列中的帧取出来进行剪切处理并推流.
- 在处理帧的时候一定要注意将处理后的帧的 fps 设置为 ffmpeg command 中设置的width,height,不然推流会失败. `image = cv2.resize(frame[int(left_x):int(right_x)][int(left_y):int(right_y)], (width, height))`
- 不知道为什么在队列不为空的时候会出现队列中帧对象为NoneType的问题，所以在处理的时候加了个判断条件
- 适应项目要求，剪切帧需要的参数是从文件中读取的，使用时可换成别的方式.

```python
import subprocess as sp
import cv2
import sys
import queue
import threading

frame_queue = queue.Queue()
rtmpUrl = "rtmp://IP地址/live/test"
camera_path = 'rtmp://58.200.131.2:1935/livetv/hunantv' #湖南台的实时直播流

#获取摄像头参数
cap = cv2.VideoCapture(camera_path)
fps = int(cap.get(cv2.CAP_PROP_FPS))
width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
#print(fps, width, height)

# ffmpeg command
command = ['ffmpeg',
           '-y',
           'rawvideo',
           '-vcodec', 'rawvideo',
           '-pix_fmt', 'bgr24',
           '-s', "{}x{}".format(width, height),
           '-r', str(fps),
           '-c:v', 'libx264',
           '-pix_fmt', 'yuv420p',
           '-preset', 'ultrafast',
           '-f', 'flv',
           '-g', '5',
          rtmpUrl]

#读流函数
def Video():
    vid = cv2.VideoCapture(camera_path)
    if not vid.isOpened():
        raise IOError("could't open webcamera or video")
    while(vid.isOpened()):
        ret,frame = vid.read()
        #下面注释的代码是为了防止摄像头打不开而造成断流
        #if not ret:
            #vid = cv2.VideoCapture(camera_path)
            #if not vid.isOpened():
                #raise IOError("couldn't open webcamera or video")
            #continue
        frame_queue.put(frame)


def push_stream(left_x,left_y,right_x,right_y):
    # 管道配置
    while True:
        if len(command)>0:
            p = sp.Popen(command, stdin=sp.PIPE)
```

> python代码参考了以下几篇博客：
> python利用ffmpeg进行rtmp推流直播
> Python 通过ffmpeg实现实时推流$ubuntu16+ffmpeg+nginx$
>
> 以下三篇博客是关于nginx服务器和RTMP配置的，都很有参考价值
> Ubuntu安装nginx+rtmp
> 利用nginx搭建RTMP视频点播、直播、HLS服务器
> 使用 nginx 和 rtmp 插件搭建视频直播和点播服务器

材料

[1] - FFmpeg 视频处理入门教程 - 2020.01.14 - 阮一峰

[2] - 码率$Bitrate$、帧率FPS、分辨率和清晰度的联系与区别 - 2018.03.20

[3] - FFmpeg常用基本命令和中文文档 - 2017.03.04

[4] - ffmpeg 推送、保存rtmp 流命令

[5] - Python 通过ffmpeg实现实时推流$_{ubuntu16+ffmpeg+nginx}$ - 2019.08.27

[6] - 基于python2.7的opencv3.3-ffmpeg-rtmp视频处理并推送流直播 - 2018.05.21

[7] - Ubuntu安装nginx+rtmp

[8] - 利用nginx搭建RTMP视频点播、直播、HLS服务器](https://blog.csdn.net/kingroc/article/details/50839994)

[9] - 使用 nginx 和 rtmp 插件搭建视频直播和点播服务器

最后修改：2020 年 10 月 06 日 05 : 02 PM

上一篇

下一篇

发表评论

评论 *

说点什么吧.....

表情

名称 *

姓名或昵称

邮箱 *

邮箱 (必填,将保密)

地址

网站或博客

发表评论