原创 豹纹法克米 于 2019-08-29 18:39:45 发布 9759 收藏 49 版权
分类专栏： 多媒体技术 文章标签： 多媒体 FFmpeg C++

使用FFmpeg进行多媒体开发（一）——读取视频流

最近接触到FFmpeg，需要实现一个将rtsp协议的码流读取并能显示的程序。在网上搬运代码的同时，也写一些对FFmpeg，Qt这些工具的理解。

准备

首先定义宏，其作用是避免'UINT64_C' was not declared in this scope的错误。

```
#ifndef INT64_C
#define INT64_C(c) (c ## LL)
#define UINT64_C(c) (c ## ULL)
#endif
```

加入FFmpeg和C++头文件

```
extern "C" {
    /*Include ffmpeg header file*/
#include <libavformat/avformat.h>
#include <libavcodec/avcodec.h>
#include <libswscale/swscale.h>

#include <libavutil/imgutils.h>
#include <libavutil/opt.h>
#include <libavutil/mathematics.h>
#include <libavutil/samplefmt.h>
}

#include <iostream>
using namespace std;
```

主函数

首先，定义输入输出AVFormatContext结构体，这类结构体存储音视频数据,也就是音视频文件的一种抽象和封装，注意在FFmpeg开发者只能使用指针。随后定义输入输出文件名，输入就是rtsp协议的地址，这里我用的是我自己的海康摄像头地址。输出保存为一个flv文件。avformat_network_init函数顾名思义是初始化网络。

```
int main(void)
{
 AVFormatContext* ifmt_ctx = NULL, * ofmt_ctx = NULL;
    const char* in_filename, * out_filename;
 in_filename = "rtsp://admin:WY@123456@192.168.0.64/h264/ch1/main/av_stream";
 out_filename = "output.flv";

    avformat_network_init();
```

设置一个配置字典，在FFmpeg中我们用AVDictionary结构体配置。

```
 AVDictionary* avdic = NULL;
    char option_key[] = "rtsp_transport";
    char option_value[] = "tcp";
    av_dict_set(&avdic, option_key, option_value, 0);
    char option_key2[] = "max_delay";
    char option_value2[] = "5000000";
    av_dict_set(&avdic, option_key2, option_value2, 0);
```

接下来就是打开输入流了，需要使用AV,我们先看一下它的声明：

```
int avformat_open_input(AVFormatContext **ps, const char *url, ff_const59 AVInputFormat *fmt, AVDictionary **op
```

该函数有四个参数，首先是一个指向AVFormatContext指针的指针，随后是url的指针。AVInputFormat 是 指定输入的封装格式。一般传NULL，由FFmpeg自行探测。AVDictionary **options其它参数设置，它是一个字典，用于参数传递。
打开输入视频流之前我们再定义几个参数：

```
 AVPacket pkt;
 AVOutputFormat* ofmt = NULL;
    int video_index = -1;
    int frame_index = 0;

    int i;
```

AVPacket类保存解复用后，解码之前的数据。至于什么是解复用，我们都知道信号有时分复用，频分复用等，音视频信号中经常将视频音频等进行复用，在接收端就得把他们独立分离出来，即Source->Demux->Stream的变化。详见：AVPacket分析
接下来我们打开输入流：

```
    //打开输入流
    int ret;
    if ((ret = avformat_open_input(&ifmt_ctx, in_filename, 0, &avdic)) < 0)
    {
     c o u t<<"Could not open input file."<<endl;
        goto end;
    }
    if ((ret = avformat_find_stream_info(ifmt_ctx, 0)) < 0)
    {
     c o u t<<"Failed to retrieve input stream information"<<endl;
        goto end;
    }
```

这里的end代表程序末尾，在end中做一些关闭输入流等收尾工作。如果avformat_open_input返回负值，则输出错误并结束。
avformat_open_input的作用是打开输入流并阅读文件头，但不打开解码器。而avformat_find_stream_info则阅读媒体文件中的包，获得流推送的信息。
继续看代码：

```
    //nb_streams代表有几路流

    for (i = 0; i < ifmt_ctx->nb_streams; i++)
    {
        if (ifmt_ctx->streams[i]->codecpar->codec_type == AVMEDIA_TYPE_VIDEO)
        {
            //视频流
```

```
    video_index = i;
      cout << "get videostream." << endl;
          break;
      }
  }
  av_dump_format(ifmt_ctx, 0, in_filename, 0);
```

我们用一个for循环，如果第i个流的codecpar参数中的codec_type 为AVMEDIA_TYPE_VIDEO我们就知道获取了视频流了。随后我们将视频流这一支的i赋给video_index，并将"get videostream."信息打印在控制台上。av_dump_format函数打印其他一些流的信息。如下图所示：

```
Input #0, rtsp, from 'rtsp://admin:WY@123456@192.168.0.64/h264/ch1/main/av_stream':
  Metadata:
    title           : Media Presentation
  Duration: N/A, start: 0.000000, bitrate: N/A
    Stream #0:0: Video: h264 (High), yuv420p(progressive), 1280x1024, 25 fps, 25 tbr, 90k tbn, 50 tbc
    Stream #0:1: Audio: pcm_alaw, 8000 Hz, 1 channels, s16, 64 kb/s
```

接下来，我们打开输出流：

```
  avformat_alloc_output_context2(&ofmt_ctx, NULL, NULL, out_filename);

  if (!ofmt_ctx)
  {
      printf("Could not create output context\n");
  ret = AVERROR_UNKNOWN;
      goto end;
  }
```

这个函数的作用是根据文件名寻找合适的AVFormatContext管理结构。接下来写文件头到输出文件：

```
  //写文件头到输出文件
  ret = avformat_write_header(ofmt_ctx, NULL);
  if (ret < 0)
  {
      printf("Error occured when opening output URL\n");
      goto end;
  }
```

接下来就是把数据存入视频文件啦。我们使用一个while循环：

```
  while (1)
  {
  AVStream* in_stream, * out_stream;
      //从输入流获取一个数据包
  ret = av_read_frame(ifmt_ctx, &pkt);//读一帧并放到pkt中去
      if (ret < 0)
          break;//读取失败

  in_stream = ifmt_ctx->streams[pkt.stream_index];
  out_stream = ofmt_ctx->streams[pkt.stream_index];
      //copy packet
      //转换 PTS/DTS 时序
   pkt.pts = av_rescale_q_rnd(pkt.pts, in_stream->time_base, out_stream->time_base, (enum AVRounding)(AV_R
   pkt.dts = av_rescale_q_rnd(pkt.dts, in_stream->time_base, out_stream->time_base, (enum AVRounding)(AV_R
      //printf("pts %d dts %d base %d\n",pkt.pts,pkt.dts, in_stream->time_base);
   pkt.duration = av_rescale_q(pkt.duration, in_stream->time_base, out_stream->time_base);
   pkt.pos = -1;

      //此while循环中并非所有packet都是视频帧，当收到视频帧时记录一下，仅此而已
      if (pkt.stream_index == video_index)
      {
          printf("Receive %8d video frames from input URL\n", frame_index);
  frame_index++;
      }

      //将包数据写入到文件。
  ret = av_interleaved_write_frame(ofmt_ctx, &pkt);
      if (ret < 0)
      {
          if (ret == -22) {
              continue;
          }
          else {
              printf("Error muxing packet.error code %d\n", ret);
              break;
          }
      }
```

对网络视频文件的播放需要经过以下步骤：解协议，解封装，解码视音频，视音频同步。这其中，接协议后的输出就是AVStream类。我们首先用av_read_frame读取一帧的数据。紧接着，我们用指针in_stream指向packet中某一个流的数据，out_stream指向另一个流的数据。获取流的指针后，进行pts/dts转换。pts，dts分别是视频播放和解码时间戳，下面这篇文章叙述地比较详细：

pts，dts的概念——作者:SamirChen

若收到视频帧，则打印到控制台：

```
      if (pkt.stream_index == video_index)
      {
          printf("Receive %8d video frames from input URL\n", frame_index);
  frame_index++;
      }
```

最后，调用av_interleaved_write_frame写数据，av_interleaved_write_frame函数相较于av_write_frame提供了对 packet 进行缓存和 pts 检查的功能。

```
      //将包数据写入到文件。
  ret = av_interleaved_write_frame(ofmt_ctx, &pkt);
      if (ret < 0)
      {
          if (ret == -22) {
              continue;
          }
          else {
              printf("Error muxing packet.error code %d\n", ret);
              break;
          }
      }
          av_packet_unref(&pkt);
  }
```

最后，我们写文件尾和对之前的错误进行后续处理：

```
  //写文件尾
  av_write_trailer(ofmt_ctx);

end:
  av_dict_free(&avdic);
  avformat_close_input(&ifmt_ctx);
  //Close input
  if (ofmt_ctx && !(ofmt->flags & AVFMT_NOFILE))
      avio_close(ofmt_ctx->pb);
```

```
    avformat_free_context(ofmt_ctx);
    if (ret < 0 && ret != AVERROR_EOF)
    {
    cout<<"Error occured."<<endl;
        return -1;
    }

    return 0;
}
```

在使用visual studio运行时，需要关闭SDL检查，否则会因为版本原因报错

总结

第一次使用FFmpeg进行多媒体开发，虽然代码是搬运的，但过程还是很有趣的，以后有时间会继续学习这一块。
原文链接:
FFmpeg从rtsp抓取流
完整项目代码:

```
#ifndef INT64_C
#define INT64_C(c) (c ## LL)
#define UINT64_C(c) (c ## ULL)
#endif

extern "C" {
    /*Include ffmpeg header file*/
#include <libavformat/avformat.h>
#include <libavcodec/avcodec.h>
#include <libswscale/swscale.h>

#include <libavutil/imgutils.h>
#include <libavutil/opt.h>
#include <libavutil/mathematics.h>
#include <libavutil/samplefmt.h>
}

#include <iostream>
using namespace std;

int main(void)
{
 AVFormatContext* ifmt_ctx = NULL, * ofmt_ctx = NULL;
    const char* in_filename, * out_filename;
 in_filename = "rtsp://admin:WY@123456@192.168.0.64/h264/ch1/main/av_stream";
 out_filename = "output.flv";

    avformat_network_init();

 AVDictionary* avdic = NULL;
    char option_key[] = "rtsp_transport";
    char option_value[] = "tcp";
    av_dict_set(&avdic, option_key, option_value, 0);
    char option_key2[] = "max_delay";
    char option_value2[] = "5000000";
    av_dict_set(&avdic, option_key2, option_value2, 0);

 AVPacket pkt;
 AVOutputFormat* ofmt = NULL;
    int video_index = -1;
    int frame_index = 0;

    int i;

    //打开输入流
    int ret;
    if ((ret = avformat_open_input(&ifmt_ctx, in_filename, 0, &avdic)) < 0)
    {
    cout<<"Could not open input file."<<endl;
        goto end;
    }
    if ((ret = avformat_find_stream_info(ifmt_ctx, 0)) < 0)
    {
    cout<<"Failed to retrieve input stream information"<<endl;
        goto end;
    }


    //nb_streams代表有几路流

    for (i = 0; i < ifmt_ctx->nb_streams; i++)
    {
        if (ifmt_ctx->streams[i]->codecpar->codec_type == AVMEDIA_TYPE_VIDEO)
        {
            //视频流
     video_index = i;
      cout << "get videostream." << endl;
            break;
        }
    }

    av_dump_format(ifmt_ctx, 0, in_filename, 0);

    //打开输出流
    avformat_alloc_output_context2(&ofmt_ctx, NULL, NULL, out_filename);

    if (!ofmt_ctx)
    {
        printf("Could not create output context\n");
    ret = AVERROR_UNKNOWN;
        goto end;
    }
 ofmt = ofmt_ctx->oformat;

    for (i = 0; i < ifmt_ctx->nb_streams; i++)
    {
   AVStream* in_stream = ifmt_ctx->streams[i];
   AVStream* out_stream = avformat_new_stream(ofmt_ctx, in_stream->codec->codec);

        if (!out_stream)
        {
            printf("Failed allocating output stream.\n");
        ret = AVERROR_UNKNOWN;
            goto end;
        }

        //将输出流的编码信息复制到输
    ret = avcodec_copy_cont                    odec, in_stream->codec);
        if (ret < 0)
        {
            printf("Failed to copy context from input to output stream codec context\n");
            goto end;
        }
```

```
    out_stream->codec->codec_tag = 0;

        if (ofmt_ctx->oformat->flags & AVFMT_GLOBALHEADER)
      out_stream->codec->flags |= AV_CODEC_FLAG_GLOBAL_HEADER;

    }

    av_dump_format(ofmt_ctx, 0, out_filename, 1);

    if (!(ofmt->flags & AVFMT_NOFILE))
    {
     ret  = avio_open(&ofmt_ctx->pb, out_filename, AVIO_FLAG_WRITE);
        if (ret < 0)
        {
            printf("Could not open output URL '%s'", out_filename);
            goto end;
        }
    }

    //写文件头到输出文件
   ret  = avformat_write_header(ofmt_ctx, NULL);
    if (ret < 0)
    {
        printf("Error occured when opening output URL\n");
        goto end;
    }


    //while循环中持续获取数据包，不管音频视频都存入文件
    while (1)
    {
    AVStream* in_stream, * out_stream;
        //从输入流获取一个数据包
     ret  = av_read_frame(ifmt_ctx, &pkt);
        if (ret < 0)
            break;

   in_stream  = ifmt_ctx->streams[pkt.stream_index];
   out_stream  = ofmt_ctx->streams[pkt.stream_index];
        //copy packet
        //转换 PTS/DTS 时序
     pkt.pts = av_rescale_q_rnd(pkt.pts, in_stream->time_base, out_stream->time_base, (enum AVRounding)(AV_R
     pkt.dts = av_rescale_q_rnd(pkt.dts, in_stream->time_base, out_stream->time_base, (enum AVRounding)(AV_R
        //printf("pts %d dts %d base %d\n",pkt.pts,pkt.dts, in_stream->time_base);
     pkt.duration = av_rescale_q(pkt.duration, in_stream->time_base, out_stream->time_base);
     pkt.pos = -1;

        //此while循环中并非所有packet都是视频帧，当收到视频帧时记录一下
        if (pkt.stream_index == video_index)
        {
            printf("Receive %8d video frames from input URL\n", frame_index);
     frame_index++;
        }

        //将包数据写入到文件。
     ret  = av_interleaved_write_frame(ofmt_ctx, &pkt);
        if (ret < 0)
        {
            if (ret == -22) {
                continue;
            }
            else {
                printf("Error muxing packet.error code %d\n", ret);
                break;
            }

        }

        //av_free_packet(&pkt); //此句在新版本中已deprecated 由av_packet_unref代替
        av_packet_unref(&pkt);
    }


    //写文件尾
    av_write_trailer(ofmt_ctx);

end:
    av_dict_free(&avdic);
    avformat_close_input(&ifmt_ctx);
    //Close input
    if (ofmt_ctx && !(ofmt->flags & AVFMT_NOFILE))
        avio_close(ofmt_ctx->pb);
    avformat_free_context(ofmt_ctx);
    if (ret < 0 && ret != AVERROR_EOF)
    {
    cout<<"Error occured."<<endl;
        return -1;
    }

    return 0;
}

```

从mp4中读取视频信息，并rtmp推流

基于FFmpeg，读取RTSP, 编解码音视频流，视频流添加文字，保存mp4 12-27
1.基于FFmpeg实现rtsp访问 2.编解码音视频 3.视频流添加文字 4.音视频同步mp4 5.兼容Windows和Linux

...实现实时推流和拉流保存的功能_ffmpeg拉流并保存_TSINGSEE的博客-CSDN... 7-31
拉流保存命令: ffmpeg -i rtmp://server/live/streamName -c copy dump.flv 该命令就是将rtmp://server/live/streamName视频流保存为dump.flv文件 实时推流...

...使用FFmpeg保存网络流到本地文件_android ffmpeg保存视频流... 8-1
在Amdroid直播开发之旅(5):详解ffmpeg编译与在Android平台上的移植和Android直播开发之旅(12):初探FFmpeg开源框架文章中,我们分别探讨了FFmpeg...

FFMPEG保存视频流数据至本地（rtsp转mp4） 爱捣蛋的小仙儿。 1万+
将rtsp流中的h264视频流在没解码之前获取下来，并保存到本地文件mp4中的h264流中，h264->mp4 流程图： 源码： #include "rtspstreammuxtask.h" #i...

ffmpeg 解复用 fengfeifengfei的专栏 605
#include #include #include int main(int argc ,char *argv[]) { //注册所有类型 av_register_all(); // if(argc>=3&&argc { printf("please input video f...

ffmpeg(7):将h264编码的视频流保存为BMP或者JPEG图片_js h264 转jpg... 7-29
一般我们知道播放视频流的时候是有截图功能的。 所以我想是否可以将视频流保存为BMP或者JPEG 参考: 1.最简单的基于FFMPEG的图像编码器(YUV编...

FFmpeg —— 10.示例程序（四）：音视频分离（分离为AAC、H264格式） 天空的博客 1万+
参考：https://blog.csdn.net/leixiaohua1020/article/details/39802819 流程图 程序源码 /* * * 本程序可以将封装格式中的视频码流数据和音频码流数据分离...

FFmpeg常用参数详解 程序猿老樊的博客 8082
FFmpeg是非常强大的媒体文件转换工具，可以转换任何媒体文件，还可以用自己的 AudioFilter 以及 VideoFliter 进行处理和编辑。在使用的时候吗，我们...

基于python-ffmpeg库的摄像机拉流与视频保存 weixin_44287933的博客 167
之前妄图通过定时器的方式解决opencv的拉流问题。后来发现不如直接用ffmpeg拉流。刚好得知有可用的包，故试之。

c++使用cmake链接ffmpeg并读取视频文件显示出来 04-24
一个简单的通过c++接口调用ffmpeg实现了读取一段视频并通过opencv显示出来，以及写入文档文件中

C# 利用ffmpeg进行推流 04-12
C# 利用ffmpeg进行推流，推流方式涉及：读取本地文件，以入读取指定内存数据；示例项目中使用开源项目 FFmpeg.AutoGen，其对ffmpeg的API做了封...

Javacv使用ffmpeg实现音视频同步播放 08-28
主要介绍了Javacv使用ffmpeg实现音视频同步播放，小编觉得挺不错的，现在分享给大家，也给大家做个参考。一起跟随小编过来看看吧

代码例子：如何用FFmpeg读取内存的PS/TS流，分离出视频和音频 12-08
演示如何用FFmpeg读取内存的PS/TS流，分离出视频和音频，用VC2008编译的，支持读取TS文件或PS文件到内存，支持分离、解码

FFmpeg 接收 UDP 视频流 热门推荐 Big_Wong的博客 1万+
主要目的是想用 FFmpeg 接收 UDP 的视频流。 考虑过用 FFmpeg 的库自己编程，一堆坑：（1）雷霄骅：最简单的推流器 里用到的一些函数和结构体...

C# 调用ffmpeg把rtsp视频流保存为MP4文件 唐伯虎点纹香的博客 1108
【代码】C# 调用ffmpeg把rtsp视频流保存为MP4文件。

ffmpeg之rtsp保存为mp4文件 ZONGXP的博客 1万+
参考：https://www.cnblogs.com/wenjingu/p/3990071.html 参考了很多网上现有的例子，流程大同小异，基本上都是上述链接的翻版，但实际测试时存在一...

ffmpeg 读取rtp h264视频流解码C代码 06-10
以下是使用FFmpeg读取RTP H.264视频流并解码的示例代码： ``` #include <stdio.h> #include <stdlib.h> #include <string.h> #include <errno.h> #includ...

“相关推荐”对你有帮助么？
非常没帮助 没帮助 一般 有帮助 非常有帮助

豹纹法克米
码龄8年 暂无认证
2 原创 | 14万+ 周排名 | 157万+ 总排名 | 1万+ 访问 | 等级
134 积分 | 8 粉丝 | 5 获赞 | 4 评论 | 54 收藏
私信 关注