**一、ffmpeg简介**

FFmpeg是一套可以用来记录、转换数字音频、视频，并能将其转化为流的开源计算机程序。采用LGPL或GPL许可证。它提供了录制、转换以及流化音视频的完整解决方案。它包含了非常先进的音频/视频编解码库libavcodec，为了保证高可移植性和编解码质量，libavcodec里很多code都是从头开发的。

**FFmpeg有非常强大的功能包括视频采集功能、视频格式转换、视频抓图、给视频加水印等，其项目组成为：**

libavcodec：用于各种类型声音/图像编解码；

libavutil：包含一些公共的工具函数；

libavformat：包含多种多媒体容器格式的封装、解封装工具

libswscale：用于视频场景比例缩放、色彩映射转换；

libpostproc：用于后期效果处理；

libavdevice：用于音视频数据采集和渲染等功能的设备相关

ibswresample：用于音频重采样和格式转换等功能

libavfilter：包含多媒体处理常用的滤镜功能

1. ****相关代码****

**在qt中利用ffmpeg获取h264流视频并显示：**

**#include "ffmpegdecode.h"#include <QDateTime>#include <QDebug>#include <QCoreApplication>#include "stdio.h"**

*//static int interrupt\_cb( void \*ctx)*

*//{// // do something// time\_out++;// if (time\_out > 40) {// time\_out=0;// if (firsttimeplay) {// firsttimeplay=0;// return 1; //�������ǳ�ʱ�ķ���// }// }// return 0;//}*

ffmpegDecode::ffmpegDecode(QObject \*parent):

QObject(parent)

{

pCodecCtx = NULL;

videoStreamIndex=-1;

av\_register\_all();*//ע���������п��õ��ļ���ʽ�ͽ�����*

avformat\_network\_init();*//��ʼ����������ʽ,ʹ��RTSP������ʱ������ִ��*

pFormatCtx = avformat\_alloc\_context();*//����һ��AVFormatContext�ṹ���ڴ�,�����м*

o\_pFormatCtx = NULL;

*//out*

ofmt = NULL;

*// s pFormatCtx->interrupt\_callback = interrupt\_cb; //ע���ص�����*

pFrame=av\_frame\_alloc();

*//outFileName = "test.mp4";*

isRecord = false;

flag = true;

}

ffmpegDecode::~ffmpegDecode()

{

avformat\_free\_context(o\_pFormatCtx);

avformat\_free\_context(pFormatCtx);

av\_frame\_free(&pFrame);

sws\_freeContext(pSwsCtx);

}

**void** ffmpegDecode::h264Decodec()

{

*//һ֡һ֡��ȡ��Ƶ*

**int** frameFinished=0;

**while**(av\_read\_frame(pFormatCtx, &packet) >= 0 && flag){

**if**(packet.stream\_index==videoStreamIndex)

{

*//qDebug()<<"��ʼ����"<<QDateTime::currentDateTime().toString("HH:mm:ss zzz");*

avcodec\_decode\_video2(pCodecCtx, pFrame, &frameFinished, &packet);

**if** (frameFinished)

{

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*ffmpeg decodec\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

mutex.lock();

**if**(isRecord)

{

*//output*

AVStream \*in\_stream, \*out\_stream;

in\_stream = pFormatCtx->streams[packet.stream\_index];

out\_stream = o\_pFormatCtx->streams[packet.stream\_index];

*//Convert PTS/DTS*

packet.pts = av\_rescale\_q\_rnd(packet.pts, in\_stream->time\_base, out\_stream->time\_base, (AVRounding)(AV\_ROUND\_NEAR\_INF|AV\_ROUND\_PASS\_MINMAX));

packet.dts = av\_rescale\_q\_rnd(packet.dts, in\_stream->time\_base, out\_stream->time\_base, (AVRounding)(AV\_ROUND\_NEAR\_INF|AV\_ROUND\_PASS\_MINMAX));

packet.duration = av\_rescale\_q(packet.duration, in\_stream->time\_base, out\_stream->time\_base);

packet.pos = -1;

*//Write*

**if** (av\_interleaved\_write\_frame(o\_pFormatCtx, &packet) < 0) { *//��AVPacket���洢��Ƶѹ���������ݣ�д���ļ���*

printf( "Error muxing packet\n");

**break**;

}

}

**int** rs = sws\_scale(pSwsCtx, (**const** **uint8\_t**\* **const** \*) pFrame->data,

pFrame->linesize, 0,

height, pAVPicture.data, pAVPicture.linesize);

*//���ͻ�ȡһ֡ͼ���ź�*

QImage **image**(pAVPicture.data[0],width,height,QImage::Format\_RGB888);

emit **GetImage**(image);

mutex.unlock();

}

}

av\_free\_packet(&packet);*//�ͷ���Դ,�����ڴ���һֱ����*

}

**if**(isRecord)

{

wFileTrailer();

}

}