T Chat

我在大厂做研发

入门音视频：从写个播放器开始

莲叔 @ 阿里

T Chat

×

莲叔

UC 短视频客户端负责人

个人兴趣：
- 函数式编程
- 音视频
- 端智能

Why

- 广泛的应用场景
- 稳定的技术基本盘

T Chat 基本概念: 以短视频生产消费为例

完整项目代码见：https://github.com/aaaron7/tinyplayer

T Chat 基本概念一览

- 封装格式：MP4 / MP3 / FLV
- 视频编解码标准: VP8 / VP9 / H.264 / H.265
- 音频编解码标准：AAC
- 音频原始数据：PCM
- 颜色空间：RGB / YUV(YCbCr)
  - 排列方式：
    - RGB / RGBA / BGR / BGRA
    - YUV422 / YUV420

完整项目代码见：***https://github.com/aaaron7/tinyplayer***

T Chat

To Do

- 搭一个跨平台的架子
- 实现最小播放器
  - tinyplayer
- 在 iOS 上跑起来
  - TinyPlayDemo

完整项目代码见：***https://github.com/aaaron7/tinyplayer***

搭架子

T Chat

跨平台

- 平台代码分离与桥接
  - 尽量避免用宏
  - 利用继承与多态
- 构建机制
  - bazel / cmake
  - cross-compile toolchain
- 开发效率
  - Demo 工程调试

T Chat 目录结构

```
tinyplayer
  libs/                 依赖库
  src/
    core/               平台无关代码
    platform/           平台相关代码
      ios/
        egl/
        audio/
  examples/             Demo 工程
    iOS/
  build/                中间产物目录
```

完整项目代码见：***https://github.com/aaaron7/tinyplayer***

T Chat

CMakeLists.txt

- #1: 声明target
- #2: 分别声明源码文件，并分开平台无关和平台有关
- #3: 声明 header search path 和 lib search path
- #4: 关联源码到target，设置team和其他编译选项

```
#1
project(tinyplayer)
cmake_minimum_required(VERSION 3.4.1)
set(DEPLOYMENT_TARGET 8.0)
set(CMAKE_CXX_STANDARD 11)
#2
FILE(GLOB SRC_FILES ${CMAKE_SOURCE_DIR}/core/
*.*)
FILE(GLOB PLATFORM_FILES ${CMAKE_SOURCE_DIR}/
platform/iOS/**/*.*)

#3
include_directories(
    ${CMAKE_SOURCE_DIR}/../libs/include/
    ${CMAKE_SOURCE_DIR}/
    ${CMAKE_SOURCE_DIR}/platform/ios/egl/
    ${CMAKE_SOURCE_DIR}/platform/ios/audio/

)
link_directories(
    ${CMAKE_SOURCE_DIR}/../libs/lib/
)
#4
set(DEVELOPMENT_TEAM_ID GH246XP5QK)
add_library(tinyplayer SHARED ${SRC_FILES} $
{PLATFORM_FILES})
SET_XCODE_PROPERTY(tinyplayer
CODE_SIGN_IDENTITY "iPhone Developer")
SET_XCODE_PROPERTY(tinyplayer DEVELOPMENT_TEAM
${DEVELOPMENT_TEAM_ID})
target_compile_options(tinyplayer PUBLIC "-
fno-objc-arc")
```

完整项目代码见：***https://github.com/aaaron7/tinyplayer***

T Chat

CMakeLists.txt

- #5: 声明一个cmake 函数，用于方便的搜索 ios 系统库的路径。
- #6: 通过#5的函数添加iOS必要的framework到target
- #7 添加各类静态库（主要是ffmpeg和相关依赖到target）

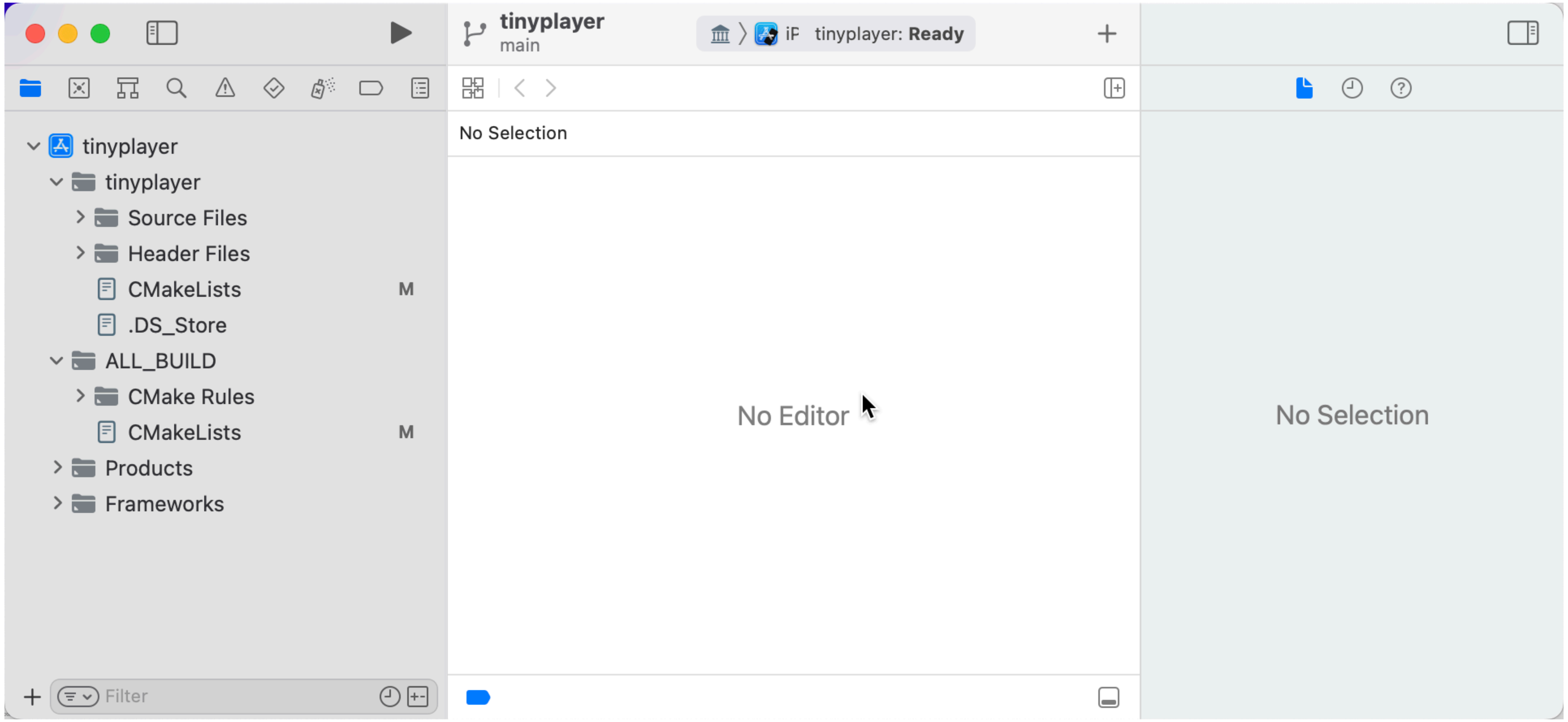
PS：这一part约等于Xcode link binary with libraries

```
#5
macro(ADD_FRAMEWORK PROJECT_NAME
FRAMEWORK_NAME)
    …
endmacro(ADD_FRAMEWORK)

#6
ADD_FRAMEWORK(tinyplayer VideoToolBox)
ADD_FRAMEWORK(tinyplayer CoreMedia)
ADD_FRAMEWORK(tinyplayer CoreVideo)
ADD_FRAMEWORK(tinyplayer CoreFoundation)
ADD_FRAMEWORK(tinyplayer Security)
ADD_FRAMEWORK(tinyplayer AVFoundation)
ADD_FRAMEWORK(tinyplayer AudioToolBox)
ADD_FRAMEWORK(tinyplayer OpenGLES)
ADD_FRAMEWORK(tinyplayer UIKit)
ADD_FRAMEWORK(tinyplayer QuartzCore)
ADD_FRAMEWORK(tinyplayer Foundation)
ADD_FRAMEWORK(tinyplayer Accelerate)

#7
target_link_libraries(tinyplayer
"-Wl"
avformat avcodec avdevice avfilter avutil
swresample swscale
z bz2 iconv x264 fdk-aac
)
```

完整项目代码见：***https://github.com/aaaron7/tinyplayer***

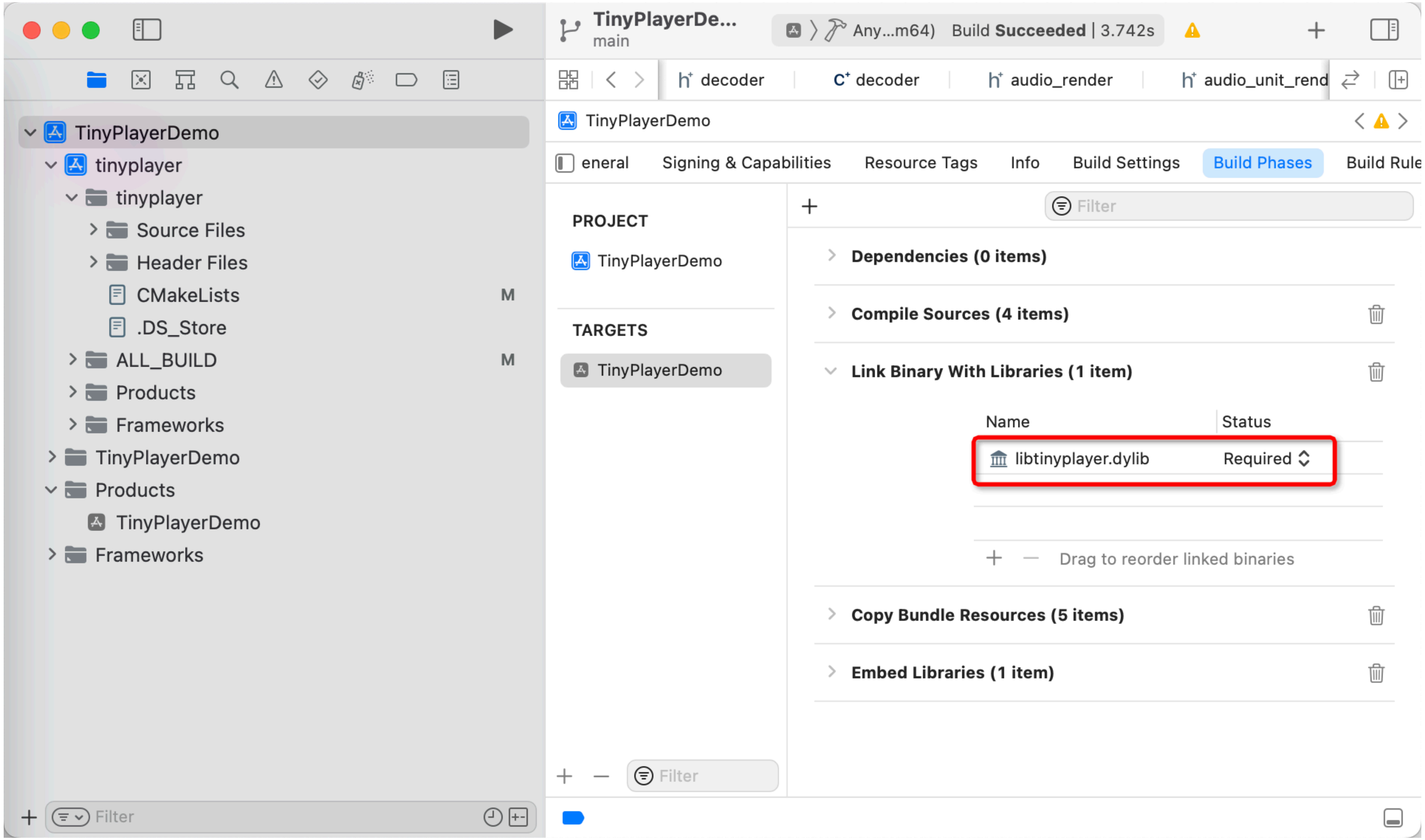
T Chat 生成 lib 工程

```
cmake -G Xcode -DCMAKE_TOOLCHAIN_FILE=./ios.cmake -DIOS_PLATFORM=OS ../src
```



完整项目代码见：*https://github.com/aaaron7/tinyplayer*

T Chat 创建 Demo 工程

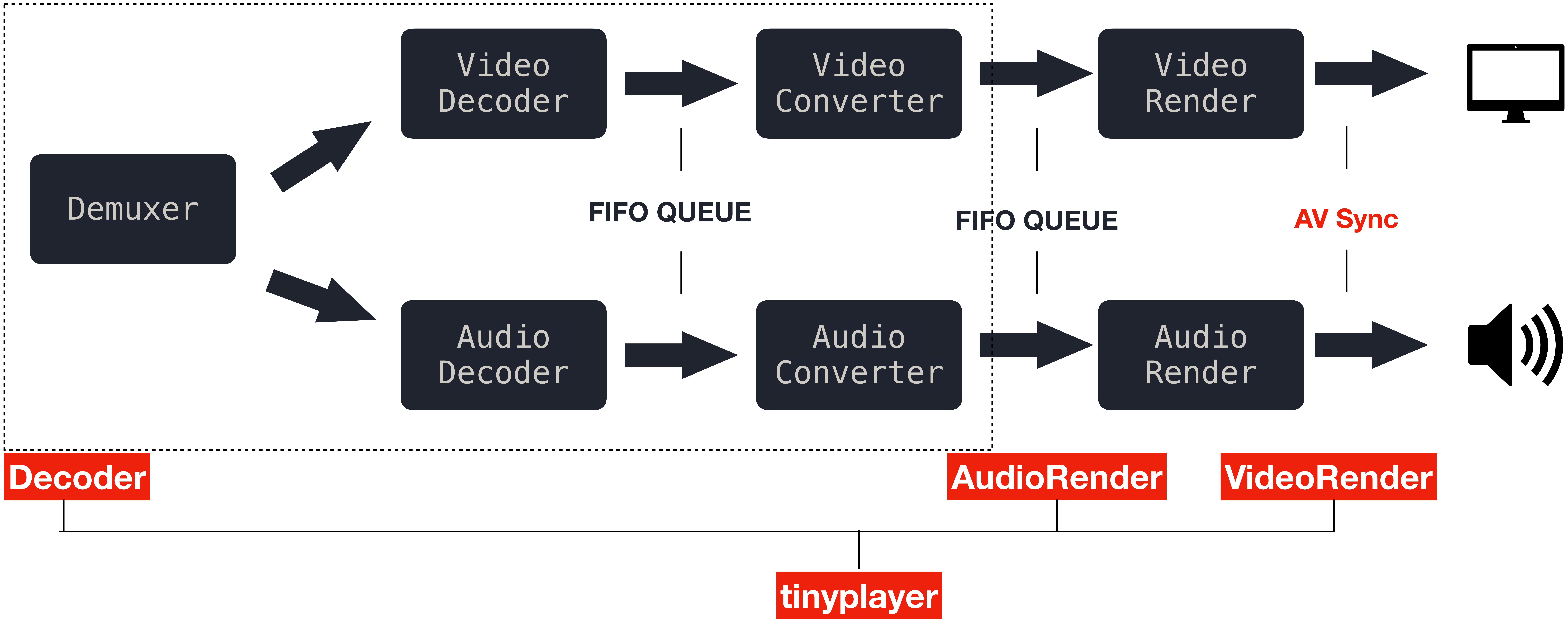


完整项目代码见：***https://github.com/aaaron7/tinyplayer***

播放器

T Chat 架构设计

- 多级生产-消费模式
- 通过并发和Queue提升吞吐量



完整项目代码见：https://github.com/aaaron7/tinyplayer

T Chat 技术栈

播放器首选

FFmpeg

vs

WebRTC

- 相同点
  - 开源 + 基础音视频处理能力
  - 都能做直播
- FFmpeg
  - 音视频处理基础库
  - 重点：多格式（封装格式、编码协议）的兼容性
- WebRTC
  - 实时音视频通信引擎
  - 重点：音频处理算法(3A控制)，弱网对抗算法
  - 部分功能依赖 FFmpeg

完整项目代码见：***https://github.com/aaaron7/tinyplayer***

T Chat FFmpeg

"A complete cross-platform solution to record, convert, and stream audio and video…the leading multimedia framework able to decode, encode, transcode, mux, demux, stream, filter, and play pretty much anything that humans and machines have created."

- 命令行工具
  - ffprobe
  - ffplay
  - ffmpeg
- 开发库
  - libavutil
  - libavcodec
  - libavformat
  - libavdevice
  - libavfilter
  - libswscale
  - libswresample **AudioConverter**

Decoder

完整项目代码见：*https://github.com/aaaron7/tinyplayer*

Player

T Chat Player

接口类

```cpp
class Player{
public:
    Player();
    ~Player();

    void SetVideoURL(std::string
file_url);

    void Play();
    void Pause();
    void Stop();
    bool Open();

    PlayerViewPlatform *GetPlayerView();

private:
    ...
private:
    void Init();
    void ReadThreadLoop();
    void RenderThreadLoop();
    void ReadFrames();
    void Render();
    void CountFPS();
    void ReleaseThread();
```

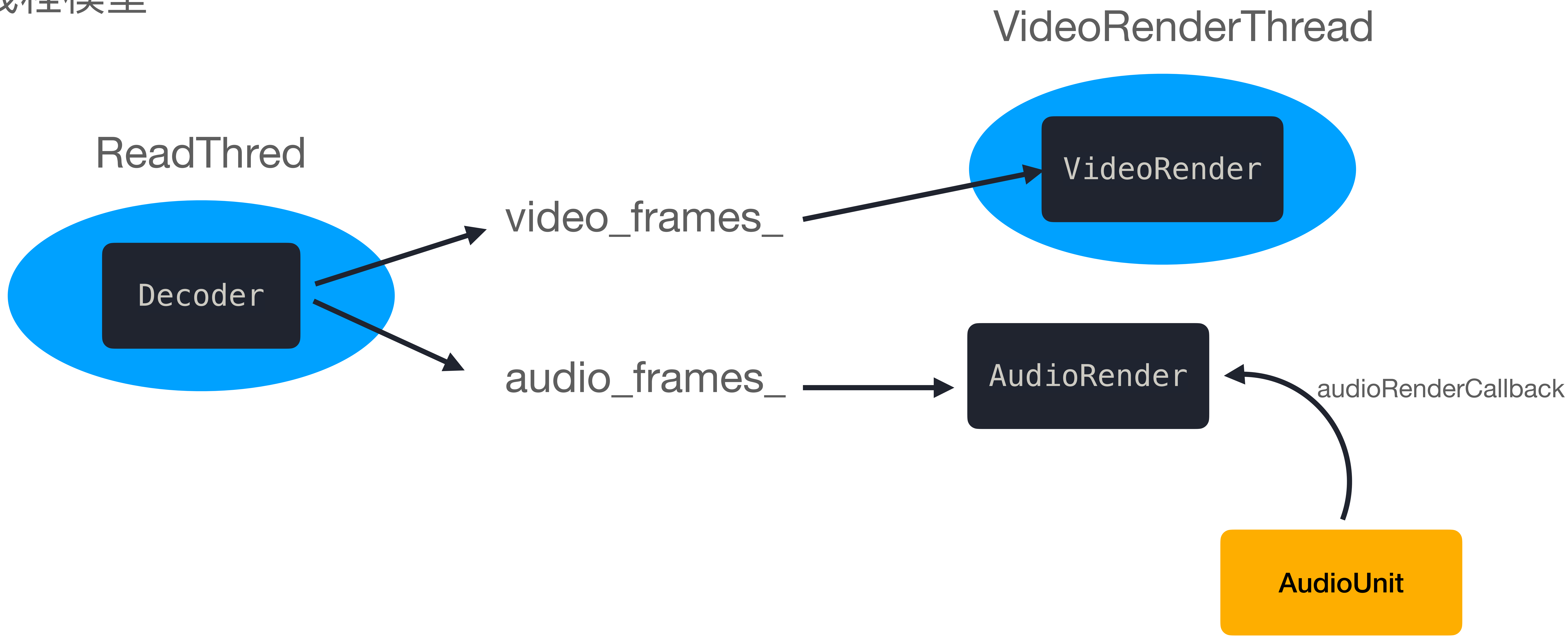
用法案例

```objc
    tinyplayer::Player *tinyPlayer_;
    tinyPlayer_ = new tinyplayer::Player();

    UIView *view = (__bridge UIView *)tinyPlayer_-
>GetPlayerView()->PlatformView();
    view.frame = self.view.bounds;
    [self.view addSubview:view];

    NSString *path = [[NSBundle mainBundle]
pathForResource:@"02_Skater" ofType:@“mp4"];

    tinyPlayer_->SetVideoURL([path UTF8String]);
    if (tinyPlayer_->Open()){
        tinyPlayer_->Play();
    }else{
        assert(0);
    }
```

完整

T Chat Player

线程模型



完整项目代码见：*https://github.com/aaaron7/tinyplayer*

Decoder

T Chat 数据结构

```
#1
class DecodedFrame{
public:
    FrameType type;
    void *buf;
    uint32_t length;
    double duration;
    double position;
    DecodedFrame(){};
    virtual ~DecodedFrame() = default;
};
#2
class DecodedVideoFrame : public DecodedFrame{
public:
    int height;
    int width;

    unique_ptr<uint8_t[]> y_data;
    unique_ptr<uint8_t[]> u_data;
    unique_ptr<uint8_t[]> v_data;

};
#3
typedef vector<shared_ptr<DecodedFrame>> FrameVec;
```

- #1: 通用帧
- #2: 视频帧需要额外的yuv data pointer
- #3: 一组帧的type alias，后面会多次用到

完整项目代码见：***https://github.com/aaaron7/tinyplayer***

T Chat Decoder

```
class Decoder{
public:
    int Open(string file_url);
    int GetVideoHeight();
    int GetVideoWidth();

    void ReadNewFrames(FrameVec &result,
FrameType *type);
private:
    ...
private:
    ...
    FrameVec GetVideoFrameFromPacket(AVPacket
*packet);
    FrameVec GetAudioFrameFromPacket(AVPacket
*packet);

};
```

- Open: 打开文件，初始化context
  - 具体可参考源码
- ReadNewFrame：播放器调用，获取最近解码的帧和类型
- GetXXXFromPacket：内部调用，从ffmpeg packet中读取frame

完整项目代码见：***https://github.com/aaaron7/tinyplayer***

T Chat Decoder

```
void Decoder::ReadNewFrames(FrameVec &result,FrameType *type)
{
    ...
    while(isReading){
        #1
        int ret = av_read_frame(format_context_, &packet);
        ...
        #2
        FrameVec frames;
        if (packet.stream_index == video_index_){
            frames = GetVideoFrameFromPacket(&packet);
        }else{
            frames = GetAudioFrameFromPacket(&packet);
        }

        #3
        if (frames.size() > 0){
            result.insert(result.end(), frames.begin(),
frames.end());

            frames.clear();
        }
        av_packet_unref(&packet);
    }
}
```

- #1: 使用 **av_read_frame** 从解封装器读取下一个packet
- #2: 从packet 中解码出一帧或几帧音频或视频帧
  - 一个packet只会有一种类型的帧数据
- #3: 添加到结果的vector中

完整项目代码见：*https://github.com/aaaron7/tinyplayer*

T Chat Decoder

- GetVideoFrameFromPacket
  - #1: 使用 **avcodec_send_packet** 将解封装的packet 发给解码器
  - #2: 通过循环执行 **avcodec_receive_frame** 不断读取从packet 中解码完成的frame
  - #3: 基于解码出来的数据，创建我们的DecodedVideoFrame
    - 本次以YUV视频为例，所以需要读取为3个data buffer

```
    #1
    int ret = avcodec_send_packet(video_codec_context_,
packet);

    do{
        #2
        ret = avcodec_receive_frame(video_codec_context_,
vframe_);
        ... //根据ret的返回值做一些逻辑控制

        int width = video_codec_context_ ->width;
        int height = video_codec_context_ ->height;

        #3
        FramePtr frame(new DecodedVideoFrame());
        VideoFramePtr vf =
dynamic_pointer_cast<DecodedVideoFrame>(frame);
        vf->width = video_codec_context_->width;
        vf->height = video_codec_context_->height;
        vf->type = FrameTypeVideo;
        vf->duration = vframe_->pkt_duration>0?:1/fps_;
        vf->position = vframe_->best_effort_timestamp *
video_timebase_ * 1000;
        vf->y_data = GetDataFromVideoFrame(vframe_->data[0],
vframe_->linesize[0], width, height);
        vf->u_data = GetDataFromVideoFrame(vframe_->data[1],
vframe_->linesize[1], width / 2, height / 2);
        vf->v_data = GetDataFromVideoFrame(vframe_->data[2],
vframe_->linesize[2], width / 2, height / 2);


        vec.push_back(frame);
```

完整项目代码见：*https://github.com/aaaron7/tinyplayer*

T Chat Decoder

- GetAudioFrameFromPacket
  - #1: 和视频一样，**avcodec_send_packet** 和 **avcodec_receive_frame** 获得解码后的音频帧
  - #2: 使用 swr_convert 对音频进行重采样
    - 不同音频输出设备对于采样率和数据格式有不同的要求
  - #3: 创建音频frame，直接用 DecodedFrame 即可

```
    #1
    int ret = avcodec_send_packet(audio_codec_context_,
packet);
    if (ret != 0){
        LOG("decoder error: %", ret);
    }

    do {
        ret =avcodec_receive_frame(audio_codec_context_,
aframe_);
        ...
        #2
        if (swr_context_ != NULL){
            ...
            uint8_t *o[2] = { (uint8_t*)audio_swr_buffer_,
0 };
            sample_per_channel = swr_convert(swr_context_, o,
samples, (const uint8_t **)aframe_->data, aframe_-
>nb_samples);
        }
        #3
        FramePtr frame(new DecodedFrame());
        int data_length = sample_per_channel *
audio_channels_ * sizeof(float);
        int elements =sample_per_channel * audio_channels_;
        frame->position = aframe_->best_effort_timestamp *
audio_timebase_ *1000;
        frame->buf = new uint8_t[audio_swr_buffer_size_];
        memcpy(frame->buf, data, audio_swr_buffer_size_);
        frame->length = audio_swr_buffer_size_;
```

完整项目代码见：https://github.com/aaaron7/tinyplayer

VideoRender

T Chat 抽象的 player view

player_view.hpp

```
class PlayerView{

public:
    PlayerView():is_inited_(false){

    }

    void PrepareLayers(){}
    virtual void
PlatformRenderBufferBind(){};
    virtual void Present(){};
    virtual void Setup(){};

protected:

    virtual void SetupLayer(){};

    virtual ~PlayerView() = default;

private:
    bool is_inited_;
```

iOS Impl

TinyPlayer.h

```
class PlayerViewPlatform : public
tinyplayer::PlayerView{
```

TinyPlayer.mm

```
@interface TinyPlayerView ()
@end
using namespace tinyplayer;
@implementation TinyPlayerView
@end

PlayerViewPlatform::~PlayerViewPlatform(){
    TinyPlayerView *view = (__bridge
TinyPlayerView *)this->platform_inst_holder_;
    [view release];
}

void PlayerViewPlatform::Present(){
    TinyPlayerView *view = (__bridge
TinyPlayerView *)this->platform_inst_holder_;
    [view present];
}
```

完整项目代码见：https://github.com/aaaron7/tinyplayer

T Chat PrepareRender

```
void VideoRender::PrepareRender(){
    #1
    player_view_->PrepareLayers();
    CreateBuffer();
    #3
    GlHelper::CreateProgram(&program_handle_,
&position_handle_);
    #4
    UpdateCoords();
}

void VideoRender::CreateBuffer(){
    #2
    GlHelper::GenRenderBuffer(&render_buffer_);
    player_view_->PlatformRenderBufferBind();
    GlHelper::CreateFrameBuffer(render_buffer_,
&frame_buffer_, &render_width_, &render_height_);
}
```

- #1: 设置平台层CAEAGLLayer属性
- #2.1:创建Render Buffer，并绑定到平台层player上
- #2.2: 创建FBO，并和RenderBuffer绑定
- #3: 创建gl program
    - 核心：需要使用yuv-rgb的 shader
- #: 初始化顶点、纹理坐标

完整项目代码见：***https://github.com/aaaron7/tinyplayer***

T Chat RenderFrame

```
void VideoRender::RenderFrame(FramePtr frame){
    ...
    #1
    glClearColor(0, 1, 0, 1);
    glClear(GL_COLOR_BUFFER_BIT);
    glViewport(0,0,render_width_, render_height_);
    glPixelStorei(GL_UNPACK_ALIGNMENT, 1);
    #2
    bool succ = GlHelper::UploadTexture(frame,
program_handle_, textures_, sampler_, video_width_,
video_height_);
    if (succ){
        #3
        glVertexAttribPointer(0, 2, GL_FLOAT,
GL_FALSE, 0, &vertex_coords_[0]);
        glEnableVertexAttribArray(0);
        glVertexAttribPointer(1, 2, GL_FLOAT,
GL_FALSE, 0, &texture_coords_[0]);
        glEnableVertexAttribArray(1);
        glDrawArrays(GL_TRIANGLE_STRIP, 0, 4);
    }else{
        assert(0);
    }
    #4
    player_view_->Present();
}
```

完整

- #1: 清空绘制上下文
- #2: 将frame的yuv数据上传并绑定纹理
- #3:将纹理通过和program绑定的shader，绘制到FrameBuffer上
- #4: 通知平台层，将RenderBuffer的内容上屏

AudioRender

T Chat AudioUnit 初始化

```
...
    AudioStreamBasicDescription _clientFormat16int;
    UInt32 bytesPerSample = sizeof (SInt16);
    bzero(&_clientFormat16int, sizeof(_clientFormat16int));
    _clientFormat16int.mFormatID          = kAudioFormatLinearPCM;
    _clientFormat16int.mFormatFlags       =
kLinearPCMFormatFlagIsSignedInteger | kLinearPCMFormatFlagIsPacked;
    _clientFormat16int.mBytesPerPacket    = bytesPerSample * channels;
    _clientFormat16int.mFramesPerPacket   = 1;
    _clientFormat16int.mBytesPerFrame     = bytesPerSample * channels;
    _clientFormat16int.mChannelsPerFrame  = channels;
    _clientFormat16int.mBitsPerChannel    = 8 * bytesPerSample;
    _clientFormat16int.mSampleRate        = 48000;

    status = AudioUnitSetProperty(audiounit,
kAudioUnitProperty_StreamFormat, kAudioUnitScope_Input, 0,
&_clientFormat16int, sizeof(_clientFormat16int));
    if(status != noErr){
        assert(0);
    }
...
```

- 一定要根据音频解码后的格式来设置AudioUnit 的输入格式
- 之前设置 swr 的输出格式为：AV_SAMPLE_FMT_S16
- 所以，这里bytesPerSample就是sizeof(SInt16)

完整项目代码见：*https://github.com/aaaron7/tinyplayer*

T Chat RenderCallback

```
OSStatus AudioUnitRender::renderCallback(void* inRefCon, AudioUnitRenderActionFlags* inActionFlags,
                                         const AudioTimeStamp* inTimeStamp, UInt32 inBusNumber,
                                         UInt32 inNumberFrames, AudioBufferList* ioData)
{

    UInt32 num = ioData->mNumberBuffers;
    for (UInt32 i = 0; i < num; ++i) {
        AudioBuffer buf = ioData->mBuffers[i];
        memset(buf.mData, 0, buf.mDataByteSize);
    }
    AudioUnitRender *render = (AudioUnitRender *)inRefCon;
    if (!render ->player_ref){
        assert(0);
    }
    render->player_ref->RenderAudioFrame(render->buffer_,inNumberFrames, render->channels_per_frame_);
    for (int i = 0 ; i< ioData->mNumberBuffers ; i++){
        AudioBuffer buf = ioData->mBuffers[i];
        uint32_t channels = buf.mNumberChannels;
        memcpy(buf.mData, render->buffer_, buf.mDataByteSize);
    }
    return noErr;
}
```

清空入参的ioData

调用Player的RenderAudioFrame方法
将数据填到render->buffer_中

将render->buffer_的数据填到ioData中。
因为这里左右声道的数据是一样的，所以copy
的数据也是一样的

完整项目代码见：*https://github.com/aaaron7/tinyplayer*

```cpp
void Player::RenderAudioFrame(short *data, uint32_t frames, uint32_t channels){
    ...
    memset(data, 0, frames * channels * sizeof(short));

    while (frames > 0) {
        if (!current_audio_frame_){
            if (audio_frames_.size() <= 0){ return; }
            lock_guard<mutex> guard(audio_lock_);
            current_audio_frame_ = audio_frames_[0];
            current_audio_frame_offset_ = 0;
            audio_frames_.erase(audio_frames_.begin());
        }

        int pos = current_audio_frame_offset_;

        void *bytes = (uint8_t *)current_audio_frame_->buf + pos;
        uint32_t remain = current_audio_frame_->length - pos;
        uint32_t channel_size = channels * sizeof(short);
        uint32_t bytes_to_copy = min(frames * channel_size, remain);
        uint32_t frames_to_copy = bytes_to_copy / channel_size;

        memcpy(data, bytes, bytes_to_copy );
        frames -= frames_to_copy;
        data += bytes_to_copy;
        current_audio_position_ = current_audio_frame_->position;

        if (bytes_to_copy < remain){
            current_audio_frame_offset_ += bytes_to_copy;
        }else{
            current_audio_frame_ = nullptr;
        }
```

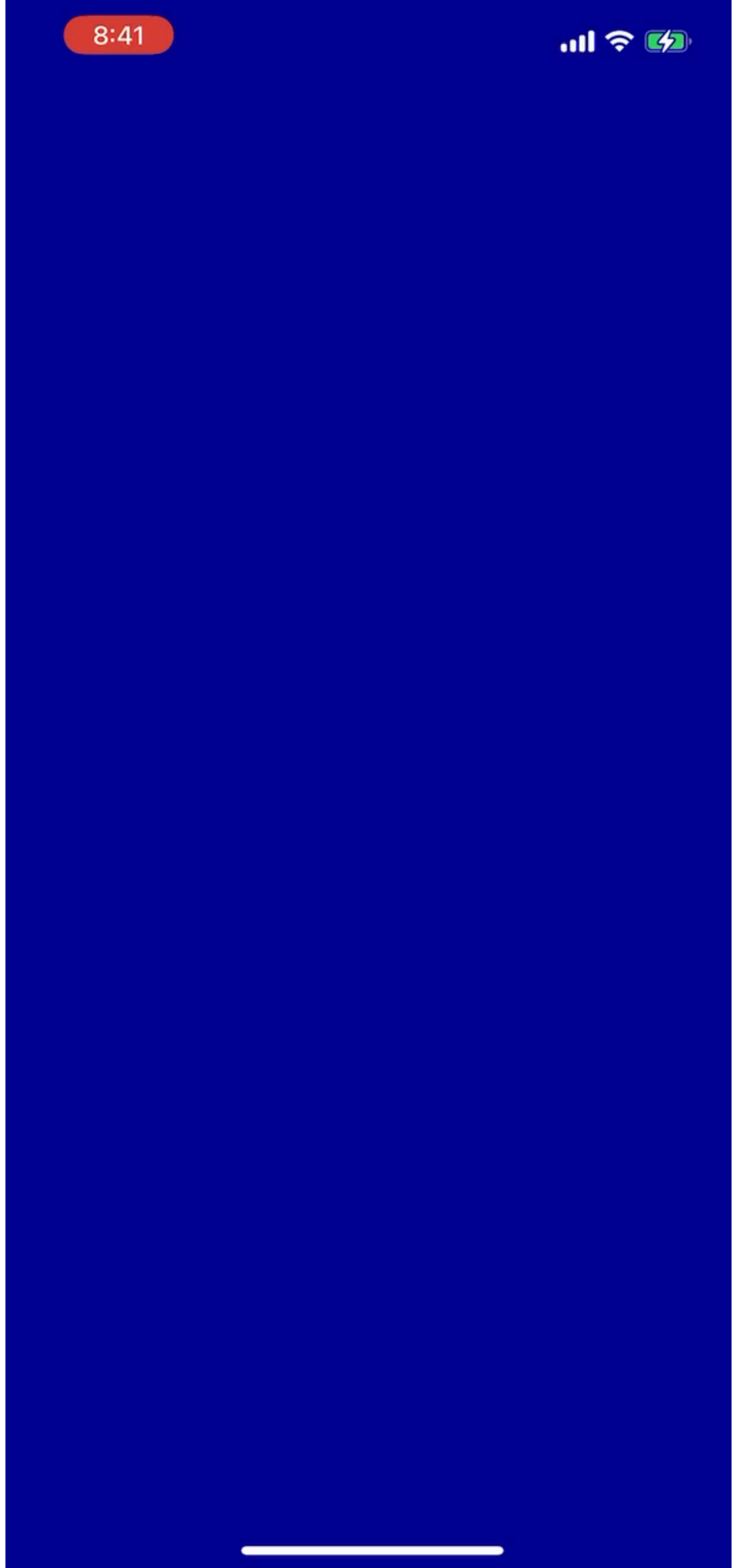
清空入参的buffer

获取待处理的音频帧，也就是audio_frames_[0]

- 根据入参(frames,channels) 计算要copy的数据 bytes_to_copy
- 当bytes_to_copy 小于frame的数据量，则记录offset，下一次callback从offset处继续读取

完整项目代码见：https://github.com/aaaron7/tinyplayer

T Chat 跑起来试试



完整项目代码见：*https://github.com/aaaron7/tinyplayer*

音视频同步

T Chat 音视频同步

- 以视频为准
- 以音频为准
- 以系统时钟为准

RenderThreadLoop:

```
{
    lock_guard<mutex> guard(video_lock_);
    FramePtr frame = video_frames_[0];

    double diff = frame->position - current_audio_position_;
    if(diff > 200){
        usleep(1000);
        return;
    } else if (diff < -200){
        //drop current frame
        video_frames_.erase(video_frames_.begin());
        return;
    } else {
        video_frames_.erase(video_frames_.begin());

        video_render_->RenderFrame(frame);
    }

}
```

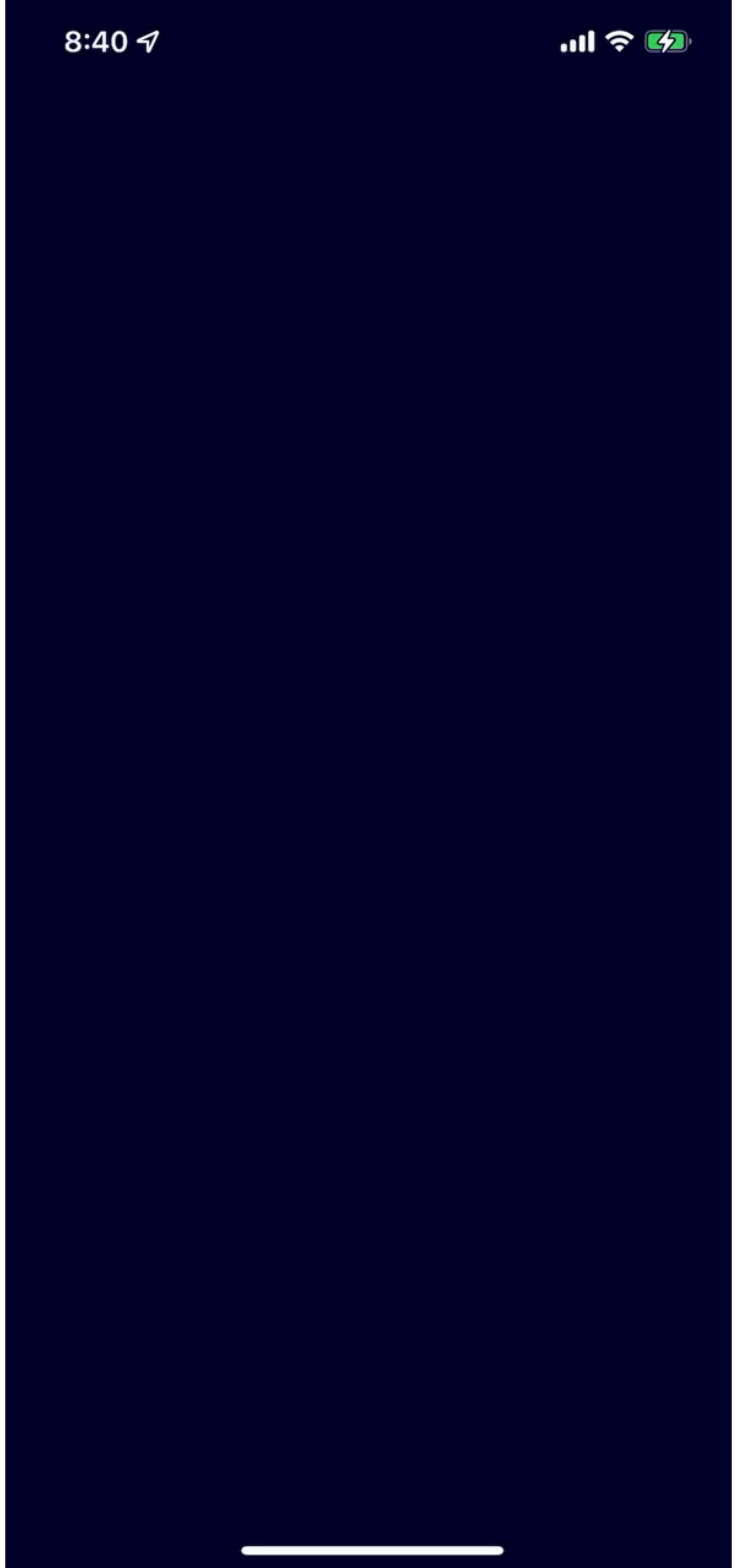
画面比声音快， sleep 等待声音

画面比声音慢，跳过当前帧

两边差不多，则渲染视频帧

完整项目代码见：*https://github.com/aaaron7/tinyplayer*

T Chat 再跑起来试试



完整项目代码见：***https://github.com/aaaron7/tinyplayer***

THANK YOU