

-iOS/Android/Linux



FFmpeg



QuickTime

主讲人： 陈 超

- QQ群: 563803045
- Email: chenchao_shenzhen@163.com

FFmpeg 介绍

- 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codec都是从头开发的。
- FFmpeg在Linux平台下开发，但它同样也可以在其它操作系统环境中编译运行，包括Windows、Mac OS X等。这个项目最早由Fabrice Bellard发起，现在由Michael Niedermayer维护。许多FFmpeg的开发人员都来自MPlayer项目，而且当前FFmpeg也是放在MPlayer项目组的服务器上。项目的名称来自MPEG视频编解码标准，前面的"FF"代表"Fast Forward"。

FFmpeg模块

- libavformat: 用于各种音视频封装格式的生成和解析, 包括获取解码所需信息以生成解码上下文结构
- 和读取音视频帧等功能;
- 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;
- 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;
- ffmpeg: 该项目提供的一个工具, 可用于格式转换、解码或电视卡即时编码等;
- ffserver: 一个 HTTP 多媒体即时广播串流服务器;
- ffplay: 是一个简单的播放器, 使用ffmpeg 库解析和解码, 通过SDL显示;

- 1 FFmpeg介绍
- 2 编译FFmpeg
- 3 获取H.264数据
- 4 FFmpeg解码H.264
- 5 FFmpeg解码H.264 2
- 6 OpenGL ES渲染YUV图像1
- 7 OpenGL ES渲染YUV图像2
- 8 接收音频数据
- 9 音频解码1
- 10 音频解码2
- 11 OpenAL 播放音频
- 12 H.264 NAL数据包分析1

- 13 H.264 NAL数据包分析2
- 14 QuickTime/mp4容器分析1
- 15 QuickTime/mp4容器分析2
- 16 AAC pcm等音频格式
- 17 pcm转AAC
- 18 FFmpeg合成H.264+AAC到mp4容器 1
- 19 FFmpeg合成H.264+AAC到mp4容器 2
- 20 FFmpeg合成H.264+AAC到mp4容器 3
- 21 pcm转mp3
- 22 FFmpeg其他

H.264编码原理I / B / P帧

- 三种帧的说明
- I帧:帧内编码帧，I帧表示关键帧，你可以理解为这一帧画面的完整保留；解码时只需要本帧数据就可以完成（因为包含完整画面）
- I帧特点:
 - 1.它是一个全帧压缩编码帧。它将全帧图像信息进行JPEG压缩编码及传输;
 - 2.解码时仅用I帧的数据就可重构完整图像;
 - 3.I帧描述了图像背景和运动主体的详情;
 - 4.I帧不需要参考其他画面而生成;
 - 5.I帧是P帧和B帧的参考帧(其质量直接影响到同组中以后各帧的质量);
 - 6.I帧是帧组GOP的基础帧(第一帧),在一组中只有一个I帧;
 - 7.I帧不需要考虑运动矢量;
 - 8.I帧所占数据的信息量比较大。

- P帧:前向预测编码帧。P帧表示的是这一帧跟之前的一个关键帧（或P帧）的差别，解码时需要用之前缓存的画面叠加上本帧定义的差别，生成最终画面。（也就是差别帧，P帧没有完整画面数据，只有与前一帧的画面差别的数据）
-
- P帧的预测与重构:P帧是以I帧为参考帧,在I帧中找出P帧“某点”的预测值和运动矢量,取预测差值和运动矢量一起传送。在接收端根据运动矢量从I帧中找出P帧“某点”的预测值并与差值相加以得到P帧“某点”样值,从而可得到完整的P帧。
- P帧特点:
- 1.P帧是I帧后面相隔1~2帧的编码帧;
- 2.P帧采用运动补偿的方法传送它与前面的I或P帧的差值及运动矢量(预测误差);
- 3.解码时必须将I帧中的预测值与预测误差求和后才能重构完整的P帧图像;
- 4.P帧属于前向预测的帧间编码。它只参考前面最靠近它的I帧或P帧;
- 5.P帧可以是其后面P帧的参考帧,也可以是其前后的B帧的参考帧;
- 6.由于P帧是参考帧,它可能造成解码错误的扩散;
- 7.由于是差值传送,P帧的压缩比较高。

- B帧:双向预测内插编码帧。B帧是双向差别帧，也就是B帧记录的是本帧与前后帧的差别（具体比较复杂，有4种情况，但我这样说简单些），换言之，要解码B帧，不仅要取得之前的缓存画面，还要解码之后的画面，通过前后画面的与本帧数据的叠加取得最终的画面。B帧压缩率高，但是解码时CPU会比较累。

-

- B帧的预测与重构

- B帧以前面的I或P帧和后面的P帧为参考帧，“找出”B帧“某点”的预测值和两个运动矢量，并取预测差值和运动矢量传送。接收端根据运动矢量在两个参考帧中“找出(算出)”预测值并与差值求和，得到B帧“某点”样值，从而可得到完整的B帧。

- B帧特点

- 1.B帧是由前面的I或P帧和后面的P帧来进行预测的；
- 2.B帧传送的是它与前面的I或P帧和后面的P帧之间的预测误差及运动矢量；
- 3.B帧是双向预测编码帧；
- 4.B帧压缩比最高，因为它只反映两参考帧间运动主体的变化情况，预测比较准确；
- 5.B帧不是参考帧，不会造成解码错误的扩散。

- 注:I、B、P各帧是根据压缩算法的需要，是人为定义的，它们都是实实在在的物理帧。一般来说，I帧的压缩率是7（跟JPG差不多），P帧是20，B帧可以达到50。可见使用B帧能节省大量空间，节省出来的空间可以用来保存多一些I帧，这样在相同码率下，可以提供更好的画质。

- h264的压缩方法:
- 1.分组:把几帧图像分为一组(GOP, 也就是一个序列),为防止运动变化,帧数不宜取多。
- 2.定义帧:将每组内各帧图像定义为三种类型,即I帧、B帧和P帧;
- 3.预测帧:以I帧做为基础帧,以I帧预测P帧,再由I帧和P帧预测B帧;
- 4.数据传输:最后将I帧数据与预测的差值信息进行存储和传输。
- 帧内 (Intraframe) 压缩也称为空间压缩 (Spatial compression) 。当压缩一帧图像时, 仅考虑本帧的数据而不考虑相邻帧之间的冗余信息, 这实际上与静态图像压缩类似。帧内一般采用有损压缩算法, 由于帧内压缩是编码一个完整的图像, 所以可以独立的解码、显示。帧内压缩一般达不到很高的压缩, 跟编码jpeg差不多。
- 帧间 (Interframe) 压缩的原理是: 相邻几帧的数据有很大的相关性, 或者说前后两帧信息变化很小的特点。也即连续的视频其相邻帧之间具有冗余信息,根据这一特性, 压缩相邻帧之间的冗余量就可以进一步提高压缩量, 减小压缩比。帧间压缩也称为时间压缩 (Temporal compression) , 它通过比较时间轴上不同帧之间的数据进行压缩。帧间压缩一般是无损的。帧差值 (Frame differencing) 算法是一种典型的时间压缩法, 它通过比较本帧与相邻帧之间的差异, 仅记录本帧与其相邻帧的差值, 这样可以大大减少数据量。
- 顺便说下有损 (Lossy) 压缩和无损 (Lossless) 压缩。无损压缩也即压缩前和解压缩后的数据完全一致。多数的无损压缩都采用RLE行程编码算法。有损压缩意味着解压缩后的数据与压缩前的数据不一致。在压缩的过程中要丢失一些人眼和人耳所不敏感的图像或音频信息,而且丢失的信息不可恢复。几乎所有高压缩的算法都采用有损压缩,这样才能达到低数据率的目标。丢失的数据率与压缩比有关,压缩比越小, 丢失的数据越多,解压缩后的效果一般越差。此外,某些有损压缩算法采用多次重复压缩的方式,这样还会引起额外的数据丢失。

H264 NAL头解析

如果NALU对应的Slice为一帧的开始，则用4字节表示，即0x00000001；否则用3字节表示，0x000001。

NAL Header: forbidden_bit, nal_reference_bit (优先级) 2bit, nal_unit_type (类型) 5bit。标识NAL单元中的RBSP数据类型，其中，nal_unit_type为1, 2, 3, 4, 5的NAL单元称为VCL的NAL单元，其他类型的NAL单元为非VCL的NAL单元。

0: 未规定

1: 非IDR图像中不采用数据划分的片段

2: 非IDR图像中A类数据划分片段

3: 非IDR图像中B类数据划分片段

4: 非IDR图像中C类数据划分片段

5: IDR图像的片段

6: 补充增强信息 (SEI)

- 7: 序列参数集 (SPS)
- 8: 图像参数集 (PPS)
- 9: 分割符
- 10: 序列结束符
- 11: 流结束符
- 12: 填充数据
- 13: 序列参数集扩展
- 14: 带前缀的NAL单元
- 15: 子序列参数集
- 16 – 18: 保留
- 19: 不采用数据划分的辅助编码图像片段
- 20: 编码片段扩展
- 21 – 23: 保留
- 24 – 31: 未规定

H.264的SPS和PPS串，包含了初始化H.264解码器所需要的信息参数，包括编码所用的profile，level，图像的宽和高，deblock滤波器。

码率：256~512 kb/s

帧率：15~20fps

分辨率：1280x720(HD) 640x368(VGA) 1920x1080(UHD)

- AAC (Advanced Audio Coding) , 中文名: 高级音频编码, 出现于1997年, 基于MPEG-2的音频编码技术。由Fraunhofer IIS、杜比实验室、AT&T、Sony等公司共同开发, 目的是取代MP3格式。2000年, MPEG-4标准出现后, AAC重新集成了其特性, 加入了SBR技术和PS技术, 为了区别于传统的MPEG-2 AAC又称为MPEG-4 AAC。

- 优点：相对于mp3，AAC格式的音质更佳，文件更小。
- 不足：AAC属于有损压缩的格式，与时下流行的APE、FLAC等无损格式相比音质存在“本质上”的差距。加之，传输速度更快的USB3.0和16G以上大容量MP3正在加速普及，也使得AAC头上“小巧”的光环不复存在了。

- 音频采样率是指录音设备在一秒钟内对声音信号的采样次数，采样频率越高声音的还原就越真实越自然。在当今的主流采集卡上，采样频率一般共分为22.05KHz、44.1KHz、48KHz三个等级，22.05KHz只能达到FM广播的声音品质，44.1KHz则是理论上的CD音质界限，48KHz则更加精确一些。

- 比特率是指每秒传送的比特(bit)数。单位为 bps(Bit Per Second), 比特率越高, 传送数据速度越快。声音中的比特率是指将模拟声音信号转换成数字声音信号后, 单位时间内的二进制数据量, 是间接衡量音频质量的一个指标。视频中的比特率 (码率) 原理与声音中的相同, 都是指由模拟信号转换为数字信号后, 单位时间内的二进制数据量。
- 信道编码中, K 符号大小的信源数据块通过编码映射为 N 符号大小的码字, 则 K/N 成为码率, 其中假设编码前后的符号表没有变化。

-

- FFMPEG中结构体很多。最关键的结构体可以分成以下几类：
- 解协议 (http,rtsp,rtmp,mms)
- AVIOContext, URLProtocol, URLContext主要存储视音频使用的协议的类型以及状态。URLProtocol存储输入视音频使用的封装格式。每种协议都对应一个URLProtocol结构。（注意：FFMPEG中文件也被当做一种协议“file”）
- 解封装 (flv,avi,rmvb,mp4)
- AVFormatContext主要存储视音频封装格式中包含的信息；AVInputFormat存储输入视音频使用的封装格式。每种视音频封装格式都对应一个AVInputFormat 结构。
- 解码 (h264,mpeg2,aac,mp3)
- 每个AVStream存储一个视频/音频流的相关数据；每个AVStream对应一个AVCodecContext，存储该视频/音频流使用解码方式的相关数据；每个AVCodecContext中对应一个AVCodec，包含该视频/音频对应的解码器。每种解码器都对应一个AVCodec结构。
- 存数据
- 视频的话，每个结构一般是存一帧；音频可能有好几帧
- 解码前数据：AVPacket
- 解码后数据：AVFrame

- AVCodec
- AVCodec是存储编解码器信息的结构体
- `const char *name`: 编解码器的名字, 比较短
- `const char *long_name`: 编解码器的名字, 全称, 比较长
- `enum AVMediaType type`: 指明了类型, 是视频, 音频, 还是字幕
- `enum AVCodecID id`: ID, 不重复
- `const AVRational *supported_framerates`: 支持的帧率 (仅视频)
- `const enum AVPixelFormat *pix_fmts`: 支持的像素格式 (仅视频)
- `const int *supported_samplerates`: 支持的采样率 (仅音频)
- `const enum AVSampleFormat *sample_fmts`: 支持的采样格式 (仅音频)
- `const uint64_t *channel_layouts`: 支持的声道数 (仅音频)
- `int priv_data_size`: 私有数据的大小
- 1.注册所有编解码器: `av_register_all()`;
- 2.声明一个AVCodec类型的指针, 比如说`AVCodec* first_c`;
- 3.调用`av_codec_next()`函数, 即可获得指向链表下一个解码器的指针, 循环往复可以获得所有解码器的信息。注意, 如果想要获得指向第一个解码器的指针, 则需要将该函数的参数设置为NULL。

- AVCodecContext
- 这是一个描述编解码器上下文的数据结构，包含了众多编解码器需要的参数信息
- 如果是单纯使用libavcodec，这部分信息需要调用者进行初始化；如果是使用整个FFMPEG库，这部分信息在调用 av_open_input_file和av_find_stream_info的过程中根据文件的头信息及媒体流内的头部信息完成初始化。其中几个主要域的释义如下：
 - extradata/extradata_size：这个buffer中存放了解码器可能会用到的额外信息，在av_read_frame中填充。一般来说，首先，某种具体格式的demuxer在读取格式头信息的时候会填充extradata，其次，如果demuxer没有做这个事情，比如可能在头部压根儿就没有相关的编解码信息，则相应的parser会继续从已经解复用出来的媒体流中继续寻找。在没有找到任何额外信息的情况下，这个buffer指针为空。
 - time_base：
 - width/height：视频的宽和高。
 - sample_rate/channels：音频的采样率和信道数目。
 - sample_fmt：音频的原始采样格式。
 - codec_name/codec_type/codec_id/codec_tag：编解码器的信息。

- AVStream
- 该结构体描述一个媒体流
- 主要域的释义如下，其中大部分域的值可以由av_open_input_file根据文件头的信息确定，缺少的信息需要通过调用av_find_stream_info读帧及软解码进一步获取：
 - index/id: index对应流的索引，这个数字是自动生成的，根据index可以从AVFormatContext::streams表中索引到该流；而id则是流的标识，依赖于具体的容器格式。比如对于MPEG TS格式，id就是pid。
 - time_base: 流的时间基准，是一个实数，该流中媒体数据的pts和dts都将以这个时间基准为粒度。通常，使用av_rescale/av_rescale_q可以实现不同时间基准的转换。
 - start_time: 流的起始时间，以流的时间基准为单位，通常是该流中第一个帧的pts。
 - duration: 流的总时间，以流的时间基准为单位。
 - need_parsing: 对该流parsing过程的控制域。
 - nb_frames: 流内的帧数目。
 - r_frame_rate/framerate/avg_frame_rate: 帧率相关。
 - codec: 指向该流对应的AVCodecContext结构，调用av_open_input_file时生成。
 - parser: 指向该流对应的AVCodecParserContext结构，调用av_find_stream_info时生成。

- AVFormatContext
- 这个结构体描述了一个媒体文件或媒体流的构成和基本信息
- 这是FFMpeg中最为基本的一个结构，是所有其他结构的根，是一个多媒体文件或流的根本抽象。其中:nb_streams和streams所表示的AVStream结构指针数组包含了所有内嵌媒体流的描述；iformat和oformat指向对应的demuxer和muxer指针；pb则指向一个控制底层数据读写的ByteIOContext结构。
- start_time和duration是从streams数组的各个AVStream中推断出的多媒体文件的起始时间和长度，以微妙为单位。
- 通常，这个结构由av_open_input_file在内部创建并以缺省值初始化部分成员。但是，如果调用者希望自己创建该结构，则需要显式地为该结构的一些成员置缺省值——如果没有缺省值的话，会导致之后的动作产生异常。以下成员需要被关注：
 - probesize
 - mux_rate
 - packet_size
 - flags
 - max_analyze_duration
 - key
 - max_index_size
 - max_picture_buffer
 - max_delay

- AVPacket
- AVPacket定义在avcodec.h中
- FFMPEG使用AVPacket来暂存解复用之后、解码之前的媒体数据（一个音/视频帧、一个字幕包等）及附加信息（解码时间戳、显示时间戳、时长等）。其中：
 - dts 表示解码时间戳，pts表示显示时间戳，它们的单位是所属媒体流的时间基准。
 - stream_index 给出所属媒体流的索引；
 - data 为数据缓冲区指针，size为长度；
 - duration 为数据的时长，也是以所属媒体流的时间基准为单位；
 - pos 表示该数据在媒体流中的字节偏移量；
 - destruct 为用于释放数据缓冲区的函数指针；
 - flags 为标志域，其中，最低为置1表示该数据是一个关键帧。
- AVPacket 结构本身只是个容器，它使用data成员指向实际的数据缓冲区，这个缓冲区可以通过av_new_packet创建，可以通过 av_dup_packet 拷贝，也可以由FFMPEG的API产生（如av_read_frame），使用之后需要通过调用av_free_packet释放。
- av_free_packet调用的是结构体本身的destruct函数，它的值有两种情况：(1)av_destruct_packet_nofree或0；(2)av_destruct_packet，其中，前者仅仅是将data和size的值清0而已，后者才会真正地释放缓冲区。FFMPEG内部使用 AVPacket结构建立缓冲区装载数据，同时提供destruct函数，如果FFMPEG打算自己维护缓冲区，则将destruct设为av_destruct_packet_nofree，用户调用av_free_packet清理缓冲区时并不能够将其释放；如果FFMPEG不会再使用 该缓冲区，则将destruct设为av_destruct_packet，表示它能够被释放。对于缓冲区不能够被释放的AVPacket，用户在使用之前 最好调用av_dup_packet进行缓冲区的克隆，将其转化为缓冲区能够被释放的AVPacket，以免对缓冲区的不当占用造成异常错误。而 av_dup_packet会为destruct指针为av_destruct_packet_nofree的AVPacket新建一个缓冲区，然后将原 缓冲区的数据拷贝至新缓冲区，置data的值为新缓冲区的地址，同时设destruct指针为av_destruct_packet。

AVFrame

- 构体保存的是解码后和原始的音视频信息。AVFrame通过函数av_frame_alloc()初始化，该函数仅仅分配AVFrame实例本身，而没有分配其内部的缓存。AVFrame实例由av_frame_free()释放；AVFrame实例通常分配一次，重复使用，如分配一个AVFrame实例来保留解码器中输出的视频帧（此时应在恰当的时候使用av_frame_unref()清理参考帧并将AVFrame归零）。该类所描述的数据通常由AVBuffer的API来保存一个引用计数，并保存于AVFrame.buf
- /AVFrame.extended_buf，在至少存在一个参考的时候（如AVFrame.buf[0] != NULL），则该对象被标记为“被引用”。在此情况下，AVFrame所包含的每一组数据必须包含于AVFrame的缓存中。

AAC格式ADTS

ADTS流 跟Raw流,

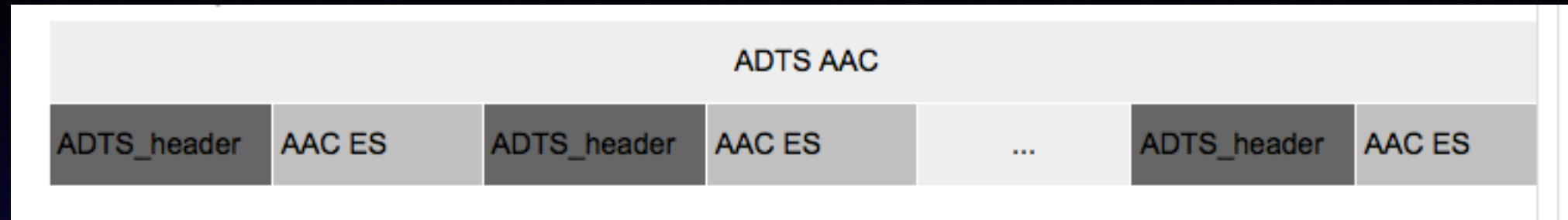
1.ADTS是个啥

ADTS全称是(Audio Data Transport Stream), 是AAC的一种十分常见的传输格式。

AAC解码器都需要把AAC的ES流打包成ADTS的格式, 一般是在AAC ES流前添加7个字节的ADTS header。也就是说你可以把ADTS这个头看作是AAC的frameheader。

- ffmpeg写 mp4+aac时呢， 音频有两个值得注意的地方。
- 1 写aac音频时， 要添加两个字节的的信息到AVCodecContext.
- 2 ffmpeg 写AAC音频数据不能含有ADTS头

- 在AAC ES流前添加7个字节的ADTS header。也就是说你可以把ADTS这个头看作是AAC的frameheader。



2.ADTS内容及结构

ADTS 头中相对有用的信息 采样率、声道数、帧长度。想想也是，我要是解码器的话，你给我一堆得AAC音频ES流我也解不出来。每一个带ADTS头信息的AAC流会清晰的告送解码器他需要的这些信息。

一般情况下ADTS的头信息都是7个字节，分为2部分：

adts_fixed_header();

adts_variable_header();

Syntax	No. of bits	Mnemonic
adts_fixed_header() { syncword; ID; layer; protection_absent; profile; sampling_frequency_index; private_bit; channel_configuration; original_copy; home; }	 12 1 2 1 2 4 1 3 1 1	 bslbf bslbf uimsbf bslbf uimsbf uimsbf bslbf uimsbf bslbf bslbf

syncword：同步头 总是0xFFF, all bits must be 1, 代表着一个ADTS帧的开始

ID：MPEG Version: 0 for MPEG-4, 1 for MPEG-2

Layer：always: '00'

profile：表示使用哪个级别的AAC，有些芯片只支持AAC LC 。在MPEG-2 AAC中定义了3种：

index	profile
0	Main profile
1	Low Complexity profile (LC)
2	Scalable Sampling Rate profile (SSR)
3	(reserved)

sampling_frequency_index：表示使用的采样率下标，通过这个下标在 **Sampling Frequencies[]**数组中查找得知采样率的值。

There are 13 supported frequencies:

- 0: 96000 Hz
- 1: 88200 Hz
- 2: 64000 Hz
- 3: 48000 Hz
- 4: 44100 Hz
- 5: 32000 Hz
- 6: 24000 Hz
- 7: 22050 Hz
- 8: 16000 Hz
- 9: 12000 Hz
- 10: 11025 Hz
- 11: 8000 Hz
- 12: 7350 Hz
- 13: Reserved
- 14: Reserved
- 15: frequency is written explicitly

channel_configuration: 表示声道数

- 0: Defined in AOT Specific Config
- 1: 1 channel: front-center
- 2: 2 channels: front-left, front-right
- 3: 3 channels: front-center, front-left, front-right
- 4: 4 channels: front-center, front-left, front-right, back-center
- 5: 5 channels: front-center, front-left, front-right, back-left, back-right
- 6: 6 channels: front-center, front-left, front-right, back-left, back-right, LFE-channel
- 7: 8 channels: front-center, front-left, front-right, side-left, side-right, back-left, back-right, LFE-channel
- 8-15: Reserved

Syntax	No. of bits	Mnemonic
adts_variable_header() { copyright_identification_bit; copyright_identification_start; aac_frame_length; adts_buffer_fullness; number_of_raw_data_blocks_in_frame; }	 1 1 13 11 2	 bslbf bslbf bslbf bslbf uimsfb

frame_length : 一个ADTS帧的长度包括ADTS头和AAC原始流.

adts_buffer_fullness: 0x7FF 说明是码率可变的码流

- tvOS必须要支持 bitcode. (iOS bitcode项可选的)
所以在编译的时候Makefile要加上 CFLAGS= -fembed-bitcode 。 如果用xcode编译lib, 要在 Build Settings—>custom compiler flags —>cflags 加上OTHER_CFLAGS="-fembed-bitcode" 。

Qt



- 类似微软的MFC C++
- 写桌面软件界面 也支持写嵌入式Windows,Mac Linux,iOS,Android等等软件.
- Qt在写桌面软件，工具软件用的非常多。
- 主要是其开源,跨平台的优良性能。
- 网站 qt.io


```
./configure --arch=x86_64 --target-os=darwin --  
disable-yasm --prefix=./output --disable-ffmpeg  
--disable-ffplay --disable-doc --disable-  
ffprobe --disable-bzlib
```

Mac OS 来源于FreeBSD Unix

Mac Linux

编程接口 Posix接口

- <http://download.qt.io/archive/qt/4.8/4.8.6/>

FFmpeg优化

- 1 内存优化。内存往上涨。没能及时回收。最好能够使用手动管理内存。
- 解码优化，看ffmpeg文档，最新的解码库，解码效率，稳定性，综合性考虑。
- YUV->RGB OpenGL ES shader来显示。

FFmpeg转码

- 1.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

- 2.视频解复用

- `ffmpeg -i test.mp4 -vcodec copy -an -f m4v test.264`

- `ffmpeg -i test.avi -vcodec copy -an -f m4v test.264`

- 3.视频转码

- `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` //转码为码流原始文件

- `ffmpeg -i test.avi -vcodec mpeg4 -vtag xvid -qsame test_xvid.avi` //转码为封装文件

- // -bf B帧数目控制, -g 关键帧间隔控制, -s 分辨率控制

- 4.视频封装

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

- 5.视频剪切

- `ffmpeg -i test.avi -r 1 -f image2 image-%3d.jpeg` //提取图片

- `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, -t 持续时间

- 6.视频录制

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

- 7.YUV序列播放

- `ffplay -f rawvideo -video_size 1920x1080 input.yuv`

- 8.YUV序列转AVI

- `ffmpeg -s w*h -pix_fmt yuv420p -i input.yuv -vcodec mpeg4 output.avi`

system调用

- `#include <stdio.h>`
- `#include <string.h>`
- `int main ()`
- `{`
- `char command[50];`
- `strcpy(command, "ffmpeg -s w*h -pix_fmt yuv420p -i input.yuv -vcodec mpeg4 output.avi");`
- `system(command);`
- `return(0);`
- `}`

- Darwin Streaming Server

- Darwin Streaming Server简称DSS。DSS是Apple公司提供的开源实时流媒体播放服务器程序。整个程序使用C++编写，在设计上遵循高性能，简单，模块化等程序设计原则，务求做到程序高效，可扩充性好。并且DSS是一个开放源代码的，基于标准的流媒体服务器，可以运行在Windows NT和Windows 2000，以及几个UNIX实现上，包括Mac OS X，Linux，FreeBSD，和Solaris操作系统上的。

- 如果要使用QuickTime流媒体服务器的编程接口，您应该熟悉该服务器实现的互联网工程组织（Internet Engineering Task Force，简称IETF）协议，列举如下：
- 实时流媒体协议（Real Time Streaming Protocol，简称RTSP）
- 实时传输协议（Real Time Transport Protocol，简称RTP）
- 实时传输控制协议（Real Time Transport Control Protocol，简称RTCP）
- 会话描述协议（Session Description Protocol，简称SDP）
- <http://dss.macosforge.org/>

流媒体转发

- 消耗带宽
- 直播。带宽费用
- 国内用户带宽上下行是不对等。上512 下20M
- amazon,阿里云。vps。
- 延时 3~4秒

RTMP

- Adobe公司开发的一种实时传输协议。
- 网页游戏。Adobe方案。客户端flash。
- 付费。 scala dropcam 流媒体转发
- 1000万人同时在线看，大并发 长时间监控录像

开源的RTMP服务器

- SRS (simple RTMP Server) 国人开发的开源rtmp服务器。
- <https://github.com/ossrs/srs>
- 苹果解决方案 hls
- 强烈推荐 安防摄像头。

libVLC

- libVLC is the core engine and the interface to the multimedia framework on which VLC media player is based.
- It allows developers to create a wide range of multimedia applications using the VLC features.

mencoder

- mencoder 是一款命令行方式的视频处理软件，是Mplayer自带的编码工具（Mplayer是Linux下的播放器，开源，支持几乎所有视频格式的播放，有windows和Mac版本）。
- mencoder支持几乎所有的格式的视频转换，可以将任意格式转换到任意格式，转换功能可以说是相当强大。市面上流行的格式转换器，都是基于mencoder开发的GUI，比如暴风转码，格式工厂等。可以说转换器能办到的，mencoder都能办到，但mencoder能办到的，转换器就不一定能办到了。
- 很多初学者宁肯去用转换器，也不用mencoder的很大原因是mencoder只支持在命令行下操作。

WebRTC

- WebRTC，名称源自网页实时通信（Web Real-Time Communication）的缩写，是一个支持网页浏览器进行实时语音对话或视频对话的技术，是谷歌2010年以6820万美元收购Global IP Solutions公司而获得的一项技术。2011年5月开放了工程的源代码，在行业内得到了广泛的支持和应用，成为下一代视频通话的标准。

- 服务器不需要消耗那么高的带宽。
- p2p 服务 打洞 udp穿透
- 视频通话
- iPhone,android,linux ,windows,Mac web browser。IE
firefox chrome. safari.
- 强烈推荐。视频会议，视频通话。监控摄像头 withings
- 不适合大并发，一般三四个同时在线看。
- 一般情况下延时较流媒体转发小一些。1~2秒

Linphone

- <http://www.linphone.org/>
- Linphone is an open source SIP Phone, available on mobile and desktop environments (iOS, Android, Windows Phone 8, GNU/Linux, Windows Desktop, MAC OSX) and on web browsers.
- Linphone has inside a separation between the user interfaces and the core engine, allowing to create various kinds of user interface on top of the same functionalities.

- 类似于skype
- 基于SIP协议
- 打电话，视频聊天。
- p2p
- 可视门铃
- 拨号很快。

屏幕录制

- FFmpeg中有一个和多媒体设备交互的类库：Libavdevice。使用这个库可以读取电脑的多媒体设备的数据，或者输出数据到指定的多媒体设备上。
- `ffmpeg -f dshow -i video="Integrated Camera" -vcodec libx264 mycamera.mkv`

Qt 提交Mac Store

- 签名问题
- QtGuiFramework
- Sandbox 沙盒机制 避免一些病毒.

• SDL

- SDL (Simple DirectMedia Layer) 是一套开放源代码的跨平台多媒体开发库，使用C语言写成。SDL提供了数种控制图像、声音、输出入的函数，让开发者只要用相同或是相似的代码就可以开发出跨多个平台（Linux、Windows、Mac OS X等）的应用软件。目前SDL多用于开发游戏、模拟器、媒体播放器等多媒体应用领域。

- 播放音频 pcm 除了openAL 还可以使用SDL
- 渲染视频
- 游戏, opengl linux开源游戏

• OpenCV

- OpenCV的全称是：Open Source Computer Vision Library。OpenCV是一个基于BSD许可（开源）发行的跨平台计算机视觉库，可以运行在Linux、Windows和Mac OS操作系统上。它轻量级而且高效——由一系列 C 函数和少量 C++ 类构成，同时提供了Python、Ruby、MATLAB等语言的接口，实现了图像处理和计算机视觉方面的很多通用算法。
- OpenCV用C++语言编写，它的主要接口也是C++语言，但是依然保留了大量的C语言接口。该库也有大量的Python, Java and MATLAB/OCTAVE (版本2.5) 的接口。这些语言的API接口函数可以通过在线文档获得。如今也提供对于C#, Ch, Ruby的支持。
- 所有新的开发和算法都是用C++接口。一个使用CUDA的GPU接口也于2010年9月开始实现。

- 机器视觉 人脸识别 上班打卡机
- 摄像头车牌识别
- 医疗器械 CT扫描。
- 物体追踪，统计。
- 3D手势识别，3D扫描 OpenGL
- 自动驾驶技术。
- 深度学习 人工智能，机器人
- 隆重推荐 支持iOS,Android等平台。

- 计算机图像
- OpenGL
- FFmpeg
- OpenCV
- SDL OpenAL
- Qt
- SRS(RTMP) WebRTC SIP

- HLS Http Live streaming
- RTMP (adobe), 网页游戏。flash flv流。

VPS

- <https://my.vultr.com/>
- Ubuntu14.04 website VPN(翻墙)。

SRS

- <https://github.com/ossrs/srs>
- https://github.com/ossrs/srs/wiki/v3_CN_Home

Linux服务器

- Redhat企业版 yum
- fedora (红帽社区版本)
- Debian apt-get
- Ubuntu(Debian)
- ls rm cp mv

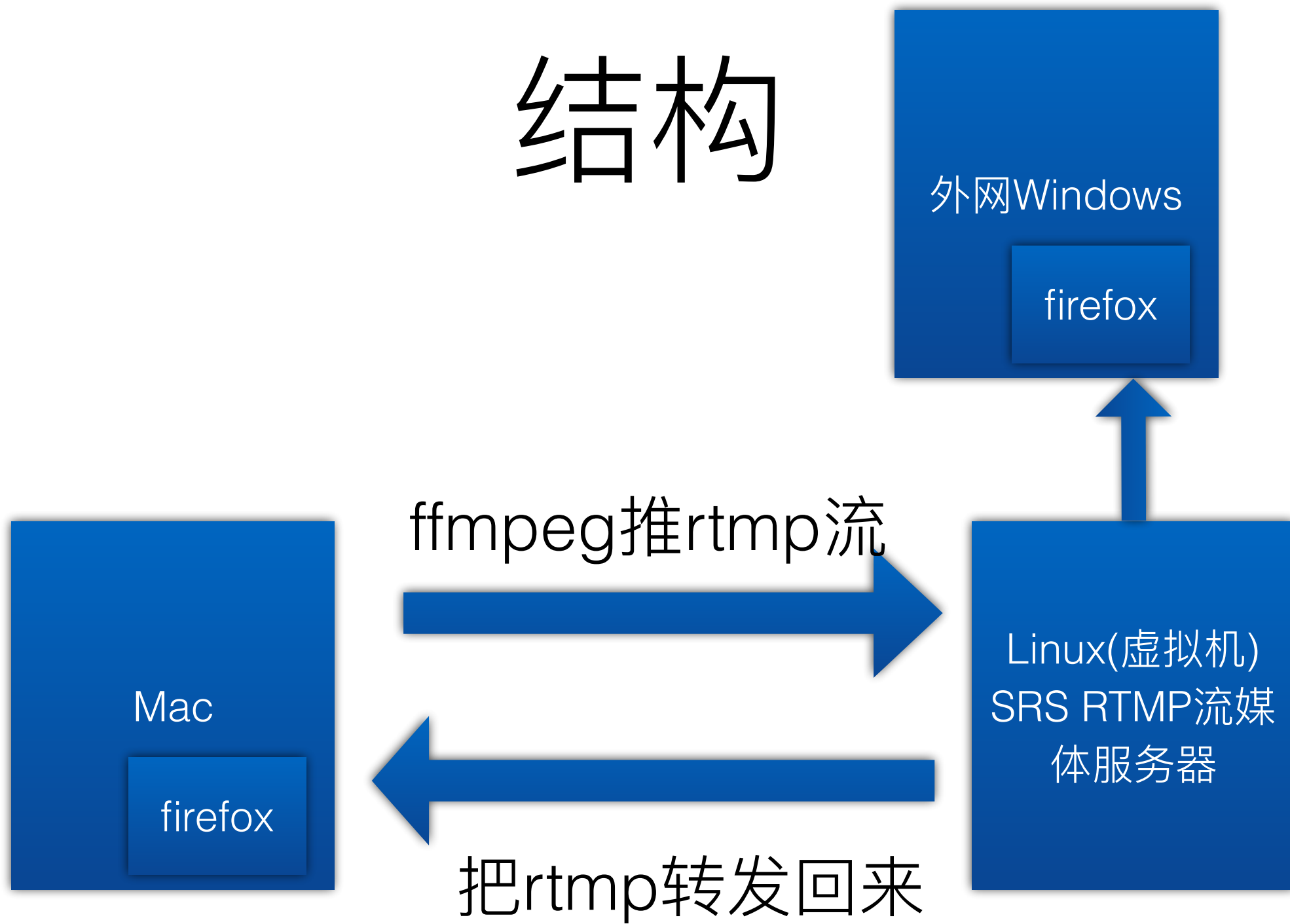
Ubuntu

- `sudo apt-get update` 更新软院源
- `sudo apt-get install vim` 安装vim编辑器
- `sudo apt-get install build-essential` 安装编译环境
c/c++
- AppStore ubuntu(debian)

Makefile

- 把各个源代码关联起来。
- automake
- cmake

结构



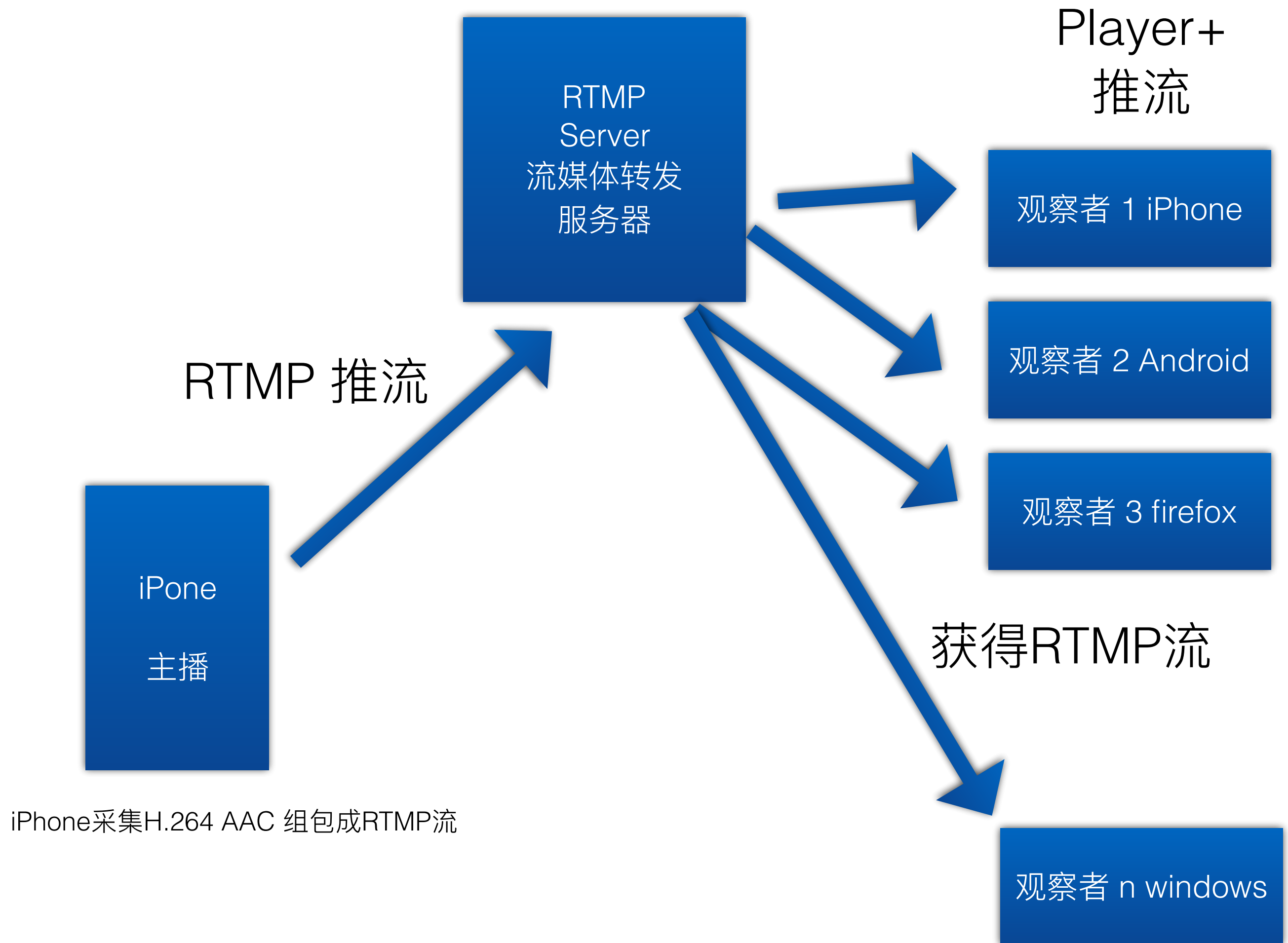
Linux SRS
RTMP流媒体服
务器

rtmp转发过来

iPhone
播放rtmp
(flv里面的
H.264
AAC)

- 1 接收RTMP
- 2 拆分flv
- 3 得到H.264 pps sps AAC
- 4 FFmpeg解码H.264
- 5 AAC转pcm
- 6 OpenGL ES显示
- 7 OpenAL 播放

- 1 接收RTMP
- 2 丢给FFmpeg
- 3 FFmpeg自动分离音视频
- 4 FFmpeg解码H.264 视频
- 5 FFmpeg 自动解码AAC
- 6 OpenGL ES显示
- 7 OpenAL/AudioQueue 播放



iPhone



Linux



Mac Firefox

