FFmpeg学习4：音频格式转换

前段时间，在学习试用FFmpeg播放音频的时候总是有杂音，网上的很多教程是基于之前版本的FFmpeg的，而新的FFmpeg3中audio增加了平面（planar）格式，而SDL播放音频是不支持平面格式的，所以通过FFmpeg解码出来的数据不能直接发送到SDL进行播放，需要进行一个格式转换。通过网上一些资料，也能够正确的播放音频了，但是对其体的音频转换过程不是很了解，这里就对FFmpeg的对音频的存储格式及格式转换做个总结。本文主要有以下几个方面的内容：

- AVSampleFormat 音频sample的存储格式
- channel layout 各个通道存储顺序
- 使用FFmpeg对音频数据进行格式转换
- 音频解码API `avcodec_decode_audio4` 在新版中已废弃，替换为使用更为简单的 `avcodec_send_packet` 和 `avcodec_receive_frame` 。本文简单的介绍了该API的使用。

AVSampleFormat

在FFmpeg中使用枚举 `AVSampleFormat` 表示音频的采样格式，其声明如下：

```
enum AVSampleFormat {
    AV_SAMPLE_FMT_NONE = -1,
    AV_SAMPLE_FMT_U8,          ///< unsigned 8 bits
    AV_SAMPLE_FMT_S16,         ///< signed 16 bits
    AV_SAMPLE_FMT_S32,         ///< signed 32 bits
    AV_SAMPLE_FMT_FLT,         ///< float
    AV_SAMPLE_FMT_DBL,         ///< double

    AV_SAMPLE_FMT_U8P,         ///< unsigned 8 bits, planar
    AV_SAMPLE_FMT_S16P,        ///< signed 16 bits, planar
    AV_SAMPLE_FMT_S32P,        ///< signed 32 bits, planar
    AV_SAMPLE_FMT_FLTP,        ///< float, planar
    AV_SAMPLE_FMT_DBLP,        ///< double, planar

    AV_SAMPLE_FMT_NB           ///< Number of sample formats. DO NOT USE if linking dynamically
};
```

和图像的像素存储格式类似，可以使用8位无符号整数，16位有符号整数，32位有符号整数以及单精度浮点数，双精度浮点数表示一个采样。但是，没有使用
24位的有符号整数，这是因为这些不同的格式使用的是原生的C类型，而C中是没有24位的长度的类型的。

> Sample value can be expressed by native C types,hence the lack of a signed 24-bit sample format even though
> it is a common raw audio data format.

对于浮点格式，其值在[-1.0,1.0]之间，任何在该区间之外的值都超过了最大音量的范围。
和YUV的图像格式格式，音频的采样格式分为平面（planar）和打包（packed）两种类型，在枚举值中上半部分是packed类型，后面（有P后缀的）是planar类型。
对于planar格式的，每一个通道的值都有一个单独的plane，所有的plane必须有相同的大小；对于packed类型，所有的数据在同一个数据平面中，不同通道的数据
交叉保存。
另外，在 `AVFrame` 中表示音频采样格式的字段 `format` 是一个int型，在使用 `AVSampleFormat` 时候需要进行一个类型转换，将int转换为 `AVSampleFormat` 枚举值。

在头文件 `samplefmt.h` 提供了和音频采样格式相关的一些函数，现列举一些如下：

- `const char *av_get_sample_fmt_name(enum AVSampleFormat sample_fmt)`
  根据枚举值获取其相应的格式名称（字符串）
- `enum AVSampleFormat av_get_sample_fmt(const char *name)`
  根据格式名字（字符串）获取相应的枚举值
- `enum AVSampleFormat av_get_packed_sample_fmt(enum AVSampleFormat sample_fmt)`
  传入planar类型的采样格式，返回其可转换的packed类型的采样格式。例如传入 `AV_SAMPLE_FMT_S32P` ，其返回值为 `AV_SAMPLE_FMT_S32` 。
- `enum AVSampleFormat av_get_planar_sample_fmt(enum AVSampleFormat sample_fmt)`
  和上面函数类似，不同的是传入的是packed类型的格式。
- `int av_sample_fmt_is_planar(enum AVSampleFormat sample_fmt`
  判断一个采样格式是不是planar类型的
- `int av_get_bytes_per_sample(enum AVSampleFormat sample_fmt)`
  每个采样值所占用的字节数
- `int av_samples_get_buffer_size(int *linesize, int nb_channels, int nb_samples,enum AVSampleFormat sample_fmt, int align)`
  根据输入的参数，计算其所占用空间的大小（字节数）。 `linesize` 可设为null，align是buff空间的对齐格式（0=default，1 = no alignment）

channel_layout

从上面可知，sample有两种类型的存储方式：平面（planar）和打包（packed），在planar中每一个通道独自占用一个存储平面；在packed中，所有通道的sample交织存储在同一个
平面。但是，对于planar格式不知道具体的某一通道所在的平面；对于packed格式各个通道的数据是以怎么样的顺序交织存储的。这就需要借助于channel_layout。
首先来看下FFmpeg对channel_layout的定义：
channel_layout是一个64位整数，每个值为1的位对应一个通道。也就说， `channel_layout` 的位模式中值为1的个数等于其通道数量。

> A channel_layout is a 64-bits interget with a bit set for every channel.The number of bits set must be equal to the number of channels.

在头文件 `channel_layout.h` 中为将每个通道定义了一个mask，其定义如下：

```
#define AV_CH_FRONT_LEFT             0x00000001
#define AV_CH_FRONT_RIGHT            0x00000002
#define AV_CH_FRONT_CENTER           0x00000004
#define AV_CH_LOW_FREQUENCY          0x00000008
#define AV_CH_BACK_LEFT              0x00000010
#define AV_CH_BACK_RIGHT             0x00000020
#define AV_CH_FRONT_LEFT_OF_CENTER   0x00000040
#define AV_CH_FRONT_RIGHT_OF_CENTER  0x00000080
#define AV_CH_BACK_CENTER            0x00000100
#define AV_CH_SIDE_LEFT              0x00000200
#define AV_CH_SIDE_RIGHT             0x00000400
#define AV_CH_TOP_CENTER             0x00000800
#define AV_CH_TOP_FRONT_LEFT         0x00001000
#define AV_CH_TOP_FRONT_CENTER       0x00002000
#define AV_CH_TOP_FRONT_RIGHT        0x00004000
#define AV_CH_TOP_BACK_LEFT          0x00008000
#define AV_CH_TOP_BACK_CENTER        0x00010000
#define AV_CH_TOP_BACK_RIGHT         0x00020000
#define AV_CH_STEREO_LEFT            0x20000000  ///< Stereo downmix.
#define AV_CH_STEREO_RIGHT           0x40000000  ///< See AV_CH_STEREO_LEFT.
```

这样，一个channel_layout就是上述channel mask的组合，部分定义如下：

```
#define AV_CH_LAYOUT_MONO              (AV_CH_FRONT_CENTER)
#define AV_CH_LAYOUT_STEREO            (AV_CH_FRONT_LEFT|AV_CH_FRONT_RIGHT)
#define AV_CH_LAYOUT_2POINT1           (AV_CH_LAYOUT_STEREO|AV_CH_LOW_FREQUENCY)
#define AV_CH_LAYOUT_2_1               (AV_CH_LAYOUT_STEREO|AV_CH_BACK_CENTER)
#define AV_CH_LAYOUT_SURROUND          (AV_CH_LAYOUT_STEREO|AV_CH_FRONT_CENTER)
#define AV_CH_LAYOUT_3POINT1           (AV_CH_LAYOUT_SURROUND|AV_CH_LOW_FREQUENCY)
#define AV_CH_LAYOUT_4POINT0           (AV_CH_LAYOUT_SURROUND|AV_CH_BACK_CENTER)
#define AV_CH_LAYOUT_4POINT1           (AV_CH_LAYOUT_4POINT0|AV_CH_LOW_FREQUENCY)
#define AV_CH_LAYOUT_2_2               (AV_CH_LAYOUT_STEREO|AV_CH_SIDE_LEFT|AV_CH_SIDE_RIGHT)
#define AV_CH_LAYOUT_QUAD              (AV_CH_LAYOUT_STEREO|AV_CH_BACK_LEFT|AV_CH_BACK_RIGHT)
#define AV_CH_LAYOUT_5POINT0           (AV_CH_LAYOUT_SURROUND|AV_CH_SIDE_LEFT|AV_CH_SIDE_RIGHT)
#define AV_CH_LAYOUT_5POINT1           (AV_CH_LAYOUT_5POINT0|AV_CH_LOW_FREQUENCY)
...
```

`AV_CH_LAYOUT_STEREO` 是立体声（2通道），其通道的存放顺序为 `LEFT | RIGHT` ； `AV_CH_LAYOUT_4POINT0` 是4通道，其通道的存放顺序为
`LEFT|RIGHT|FRONT-CENTER|BACK-CENTER` ；其它数量的声道与此类似。
下面列举一些和channel_layout相关的函数

- `uint64_t av_get_channel_layout(const char *name)` 根据传入的字符串，返回相对应的channel_layout。传入的参数可以是：
  - 常用的channel layout的名称：mono,stereo,4.0,quad,5.0,5.0(side),5.1等。
  - 一个单通道的名称：FL,FR,FC,BL,BR,FLC,FRC等
  - 通道的数量
  - channel_layout mask,以"0x"开头的十六进制串。
    更多详细的说明，参见该函数的文档。
- `int av_get_channel_layout_nb_channels(uint64_t channel_layout)` 根据通道的layout返回通道的个数
- `int64_t av_get_default_channel_layout(int nb_channels)` 根据通道的个数返回默认的layout
- `int av_get_channel_layout_channel_index(uint64_t channel_layout,uint64_t channel);` 返回通道在layout中的index，也就是某一通道
  在layout的存储位置。
  `av_get_channel_layout_channel_index` 的实现如下：

```
int av_get_channel_layout_channel_index(uint64_t channel_layout,
                                        uint64_t channel)
{
if (!(channel_layout & channel) ||
    av_get_channel_layout_nb_channels(channel) != 1)
    return AVERROR(EINVAL);
channel_layout &= channel - 1;
return av_get_channel_layout_nb_channels(channel_layout);
}
```

首先判断传入的layout包含该通道，并且保证该传入的通道是一个单通道。
以4通道 `AV_CH_LAYOUT_4POINT0` 为例，说明下计算方法。 `AV_CH_LAYOUT_4POINT0 = AV_CH_FRONT_LEFT | AV_CH_FRONT_RIGHT | AV_CH_FRONT_CENTER | AV_CH_BACK_CENTER`
其二进制表示为 `0001,0000,0111` ，假如想找 `AV_CH_BACK_CENTER` 在该layout中的index， `AV_CH_BACK_CENTER` 的十六进制为 `0x0100` ，二进制为 `0001,0000,0000` ，那么
`AV_CH_BACK_CENTER - 1 = 1111,1111` = `0001,0000,0111 & 0000,1111,1111 = 0111` ，函数 `av_get_channel_layout_nb_channels` 是获取某个layout对应的通道的数量，
前面提到，layout中值为1的位的个数和通道的数量相等，所以 `AV_CH_BACK_CENTER` 在layout `AV_CH_LAYOUT_4POINT0` 的index为3。

Audio 格式转换

在FFmpeg中进行音频的格式转换主要有三个步骤

昵称：laughingQing
园龄：1年8个月
粉丝：2
关注：0
+加关注

< 2017年2月 >

日 一 二 三 四 五 六
29 30 31 1 2 3 4
5 6 7 8 9 10 11
12 13 14 15 16 17 18
19 20 21 22 23 24 25
26 27 28 1 2 3 4
5 6 7 8 9 10 11

搜索
找找看
谷歌搜索

常用链接
我的随笔
我的评论
我的参与
最新评论
我的标签
更多链接

随笔分类
android(76)
Java(12)
流媒体开发(9)

随笔档案
2016年12月 (1)
2016年9月 (16)
2016年8月 (3)
2016年7月 (1)
2016年6月 (1)
2016年5月 (1)
2016年4月 (2)
2016年3月 (1)
2015年11月 (2)
2015年10月 (5)
2015年9月 (17)
2015年8月 (3)
2015年7月 (2)
2015年6月 (44)

最新评论
1. Re:android音视频点/直播模块开发
[img]
版主，请问有遇到截图所示的问题吗
--jwolf_tcl
2. Re:如何让Service自动重启而不被kill掉
写的好，不错，真心不错
--laughingQing
3. Re:android设备休眠机制
哇擦，这供上能还有如何棒的博客，我算是见识了！
--xmTan
4. Re:Android网络编程系列 一 TCP/IP协议概之传输层
写的不错，加油
--xmTan

阅读排行榜
1. android音视频点/直播模块开发(1338)
2. TCP/IP、Http、Socket、XMPP的区别(607)
3. android sdk 更新那些文件(458)
4. Android Gradle 完整指南(331)
5. 使用Ant打包工具 基本介绍(231)

评论排行榜
1. android设备休眠机制(1)
2. Android网络编程系列 一 TCP/IP协议概之传输层(1)
3. android音视频点/直播模块开发(1)
4. 如何让Service自动重启而不被kill掉(1)

推荐排行榜
1. android设备休眠机制(1)

1. 实例化 `SwrContext` ，并设置转换所需的参数：通道数量、channel layout、sample rate

有以下两种方式来实例 `SwrContext` ，并设置参数：

- 使用 `swr_alloc`

```
SwrContext *swr = swr_alloc();
av_opt_set_channel_layout(swr, "in_channel_layout",  AV_CH_LAYOUT_5POINT1, 0);
av_opt_set_channel_layout(swr, "out_channel_layout", AV_CH_LAYOUT_STEREO,  0);
av_opt_set_int(swr, "in_sample_rate",     48000,                0);
av_opt_set_int(swr, "out_sample_rate",    44100,                0);
av_opt_set_sample_fmt(swr, "in_sample_fmt",  AV_SAMPLE_FMT_FLTP, 0);
av_opt_set_sample_fmt(swr, "out_sample_fmt", AV_SAMPLE_FMT_S16,  0);
```

- 使用 `swr_alloc_set_opts`

```
SwrContext *swr = swr_alloc_set_opts(NULL,  // we're allocating a new context
                      AV_CH_LAYOUT_STEREO,  // out_ch_layout
                      AV_SAMPLE_FMT_S16,    // out_sample_fmt
                      44100,                // out_sample_rate
                      AV_CH_LAYOUT_5POINT1, // in_ch_layout
                      AV_SAMPLE_FMT_FLTP,   // in_sample_fmt
                      48000,                // in_sample_rate
                      0,                    // log_offset
                      NULL);                // log_ctx
```

上述两种方法设置那个的参数是将5.1声道，channel layout为AV_CH_LAYOUT_5POINT1，采样率为48KHz转换为2声道，channel_layout为AV_SAMPLE_FMT_S16，采样率为44.1KHz。

1. 计算转换后的**sample**个数
转后后的sample个数的计算公式为：src_nb_samples * dst_sample_rate / src_sample_rate。其计算如下：

```
int dst_nb_samples = av_rescale_rnd(swr_get_delay(swr_ctx, frame->sample_rate) + frame->nb_samples, frame->sample_rate, frame->sample_rate, AVRounding(1));
```

函数 `av_rescale_rnd` 是按照指定的舍入方式计算**a * b / c** 。

函数 `swr_get_delay` 得到输入sample和输出sample之间的延迟，并且其返回值的根据传入的第二个参数不同而不同。如果是输入的采样率，则返回值是输入sample个数；如果输入的是输出采样率，则返回值是输出sample个数。

2. 调用 `swr_convert` 进行转换

```
int nb = swr_convert(swr_ctx, &audio_buf, dst_nb_samples, (const uint8_t**)frame->data, frame->nb_samples);
```

其返回值为转换的sample个数。

SDL播放音频时的格式转换

- 首先使用 `avcodec_send_packet` 和 `avcodec_receive_frame` 获取解码后的原始数据

```
int ret = avcodec_send_packet(aCodecCtx, &pkt);
if (ret < 0 && ret != AVERROR(EAGAIN) && ret != AVERROR_EOF)
    return -1;

ret = avcodec_receive_frame(aCodecCtx, frame);
if (ret < 0 && ret != AVERROR_EOF)
    return -1;
```

这里不再使用 `avcodec_decode_audio4` 进行音频的解码，在FFmpeg3中该函数已被废弃，使用 `avcodec_send_packet` 和 `avcodec_receive_frame` 替代，新的解码API使用更为方便，
具体参见官方文档send/receive encoding and decoding API overview。

- 设置通道数量和channel layout
在编码的时候有可能丢失通道数量或者channel layout 。这里根据获取的参数设置其默认值

```
if (frame->channels > 0 && frame->channel_layout == 0)
    frame->channel_layout = av_get_default_channel_layout(frame->channels);
else if (frame->channels == 0 && frame->channel_layout > 0)
    frame->channels = av_get_channel_layout_nb_channels(frame->channel_layout);
```

如果channel layout未知（channel_layout = 0），根据通道数量获取其默认的channel layout；如同通道的数量未知，则根据其channel layout得到其通道数量。

- 设置输出格式
由于SDL2的sample格式不支持浮点型（FFmpeg中是支持的浮点型的），这里简单的设置输出格式为 `AV_SAMPLE_FMT_S16` （16位有符号整型），输出的channel layout也
根据通道数量设置为默认值 `dst_layout = av_get_default_channel_layout(frame->channels)` （SDL2不支持planar格式），实例化 `SwrContext`

```
swr_ctx = swr_alloc_set_opts(nullptr, dst_layout, dst_format, frame->sample_rate,
    frame->channel_layout, (AVSampleFormat)frame->format, frame->sample_rate, 0, nullptr);
if (!swr_ctx || swr_init(swr_ctx) < 0)
    return -1;
```

在设置完参数后，一定要调用 `swr_init` 进行初始化。

- 转换

```
// 计算转换后的sample个数 a * b / c
int dst_nb_samples = av_rescale_rnd(swr_get_delay(swr_ctx, frame->sample_rate) + frame->nb_samples, frame->sample_rate, frame->sample_rate, AVRounding(1));
// 转换，返回值为转换后的sample个数
int nb = swr_convert(swr_ctx, &audio_buf, dst_nb_samples, (const uint8_t**)frame->data, frame->nb_samples);
data_size = frame->channels * nb * av_get_bytes_per_sample(dst_format);
```

最后 `data_size` 中保存的是转换的数据的字节数：通道数 * sample个数 * 每个sample的字节数。

总结

本文主要介绍了在FFmepg中对音频两个重要属性：采样格式和channel layout的表示方法，并简单的实现了一个音频的格式转换。

- 采样格式 使用AVSampleFormat枚举值表示，并可分为planar和packed两类。
- channel layout 是一个64位的整数，表示各个通道数据的存放顺序，其二进制位中1的个数等于其通道的数量。

本文代码 FFmpeg-playAudio.cpp

分类：流媒体开发

好文要顶 关注我 收藏该文

laughingQing
关注 - 0
粉丝 - 2
+加关注

« 上一篇：FFmpeg数据结构：AVPacket解析
» 下一篇：FFmpeg学习5：手残程播放视音频

0 0

posted @ 2016-09-23 21:28 laughingQing 阅读(68) 评论(0) 编辑 收藏

刷新评论 刷新页面 返回顶部

注册用户登录后才能发表评论，请 登录 或 注册，访问网站首页。

【推荐】50万行VC++源码: 大型组态工控、电力仿真CAD与GIS源码库
【活动】一元专享1500元微软智能云Azure

最新IT新闻:
· Facebook开发者大会日期确定：4月18日将在加州圣何塞举行
· 苹果核心系统高级工程师谢孟军：开源如何影响程序员？
· Galaxy S8谍照再曝光：屏占比约90% 确认USB-C端口
· Google Brain创造像素化图像技术
· YouTube正在向更多用户开放其移动即时串流功能
» 更多新闻...

最新知识库文章:
· 「代码家」的学习过程和学习经验分享
· 写给未来的程序媛
· 高质量的工程代码为什么难写
· 循序渐进地代码重构
· 技术的正宗与野路子
» 更多知识库文章...

Copyright ©2017 laughingQing