7-10-音频处理基本概念及音频重采样

0 重点问题

- 如何进行重采样
- 采样率不一样的时候pts怎么处理

官方参考文档：http://ffmpeg.org/doxygen/trunk/group__lswr.html

1重采样

1.1 什么是重采样

所谓的重采样，就是改变音频的采样率、sample format、声道数等参数，使之按照我们期望的参数输出。

1.2 为什么要重采样

为什么要重采样？当然是原有的音频参数不满足我们的需求，比如在FFmpeg解码音频的时候，不同的音源有不同的格式，采样率等，在解码后的数据中的这些参数也会不一致(最新FFmpeg 解码音频后，音频格式为AV_SAMPLE_FMT_FLTP，这个参数应该是一致的)，如果我们接下来需要使用解码后的音频数据做其他操作，而这些参数的不一致导致会有很多额外工作，此时直接对其进行重采样，获取我们制定的音频参数，这样就会方便很多。
再比如在将音频进行SDL播放时候，因为当前的SDL2.0不支持planar格式，也不支持浮点型的，而最新的FFMPEG 16年会将音频解码为AV_SAMPLE_FMT_FLTP格式，因此此时就需要我们对其重采样，使之可以在SDL2.0上进行播放。

1.3 可调节的参数

通过重采样，我们可以对：

1. sample rate(采样率)
2. sample format(采样格式)
3. channel layout(通道布局，可以通过此参数获取声道数

2 对应参数解析

2.1 采样率

采样设备每秒抽取样本的次数

2.2 采样格式及量化精度（位宽）

每种音频格式有不同的量化精度（位宽），位数越多，表示值就越精确，声音表现自然就越精准。FFMpeg中音频格式有以下几种，每种格式有其占用的字节数信息（libavutil/samplefmt.h）：

```
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_S64,         ///< signed 64 bits
```

```
    AV_SAMPLE_FMT_S64P,        ///< signed 64 bits, planar
    AV_SAMPLE_FMT_NB           ///< Number of sample formats. DO NOT USE if linking
dynamically
};
```

2.3 分片（plane）和打包（packed）

以双声道为例，带P（plane）的数据格式在存储时，其左声道和右声道的数据是分开存储的，左声道的数据存储在data[0]，右声道的数据存储在data[1]，每个声道的所占用的字节数为linesize[0]和linesize[1]；

不带P（packed）的音频数据在存储时，是按照LRLRLR...的格式交替存储在data[0]中，linesize[0]表示总的数据量。

2.4 声道分布（channel_layout）

声道分布在FFmpeg\libavutil\channel_layout.h中有定义，一般来说用的比较多的是AV_CH_LAYOUT_STEREO（双声道）和AV_CH_LAYOUT_SURROUND（三声道），这两者的定义如下：

```
#define AV_CH_LAYOUT_STEREO            (AV_CH_FRONT_LEFT|AV_CH_FRONT_RIGHT)
#define AV_CH_LAYOUT_SURROUND
(AV_CH_LAYOUT_STEREO|AV_CH_FRONT_CENTER)
```

2.5 音频帧的数据量计算

一帧音频的数据量（**字节**）=channel数 * nb_samples样本数 * 每个样本占用的字节数

如果该音频帧是FLTP格式的PCM数据，包含1024个样本，双声道，那么该音频帧包含的音频数据量是2*1024*4=8192字节。

AV_SAMPLE_FMT_DBL ： 2*1024*8 = 16384

2.6 音频播放时间计算

以采样率44100Hz来计算，每秒44100个sample，而正常一帧为1024个sample，可知每帧播放时间/1024=1000ms/44100，得到每帧播放时间=1024*1000/44100=23.2ms （更精确的是23.21995464852608）。

一帧播放时间（毫秒） = nb_samples样本数 *1000/采样率 =

（1）1024*1000/44100=23.21995464852608ms ->约等于 **23**.2ms，精度损失了0.011995464852608ms，如果累计10万帧，误差>1199毫秒，如果有视频一起的就会有音视频同步的问题。 如果按着23.2去计算pts（0 23.2 46.4 ）就会有累积误差。

（2）1024*1000/48000=**21.3**3333333333333ms

3 FFmpeg重采样API

分配音频重采样的上下文

```
struct SwrContext *swr_alloc(void);
```

当设置好相关的参数后，使用此函数来初始化SwrContext结构体

```
int swr_init(struct SwrContext *s);
```

分配SwrContext并设置/重置常用的参数。

```
struct SwrContext *swr_alloc_set_opts(struct SwrContext *s,    // 音频重采样上下文
                      int64_t out_ch_layout,     // 输出的layout, 如：5.1声道
                      enum AVSampleFormat out_sample_fmt, // 输出的采样格式。Float, S16,一般选用是s16  绝大部分声卡支持
                      int out_sample_rate,    //输出采样率
                      int64_t  in_ch_layout,     // 输入的layout
                      enum AVSampleFormat  in_sample_fmt,  // 输入的采样格式
                      int  in_sample_rate,        // 输入的采样率
                         int log_offset,     // 日志相关，不用管先，直接为0
                      void *log_ctx // 日志相关，不用管先，直接为NULL
                      );
```

将输入的音频按照定义的参数进行转换并输出

```
int swr_convert(struct SwrContext *s,     // 音频重采样的上下文
                      uint8_t **out,     // 输出的指针。传递的输出的数组
                      int out_count,    //输出的样本数量，不是字节数。单通道的样本数量。
                         const uint8_t **in , //输入的数组，AVFrame解码出来的DATA
                      int in_count    // 输入的单通道的样本数量。
                      );
                      返回值 <= out_count
```

in和in_count可以设置为0，以最后刷新最后几个样本。

释放掉SwrContext结构体并将此结构体置为NULL;

```
void swr_free(struct SwrContext **s);
```

音频重采样，采样格式转换和混合库。

与lswr的交互是通过SwrContext完成的，SwrContext被分配给swr_alloc（）或swr_alloc_set_opts（）。 它是不透明的，所以所有参数必须使用AVOptions API设置。
为了使用lswr，你需要做的第一件事就是分配SwrContext。 这可以使用swr_alloc（）或swr_alloc_set_opts（）来完成。 如果您使用前者，则必须通过AVOptions API设置选项。 后一个函数提供了相同的功能，但它允许您在同一语句中设置一些常用选项。
例如，以下代码将设置从平面浮动样本格式到交织的带符号16位整数的转换，从48kHz到44.1kHz的下采样，以及从5.1声道到立体声的下混合（使用默认混合矩阵）。 这是使用swr_alloc（）函数。

```
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);
```

同样的工作也可以使用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
```

一旦设置了所有值，它必须用swr_init（）初始化。 如果需要更改转换参数，可以使用AVOptions来更改参数，如上面第一个例子所述; 或者使用swr_alloc_set_opts（），但是第一个参数是分配的上下文。 您必须再次调用swr_init（）。

转换本身通过重复调用swr_convert（）来完成。 请注意，如果提供的输出空间不足或采样率转换完成后，样本可能会在swr中缓冲，这需要“未来”样本。 可以随时通过使用swr_convert（）（in_count可以设置为0）来检索不需要将来输入的样本。 在转换结束时，可以通过调用具有NULL in和in incount的swr_convert（）来刷新重采样缓冲区。

4. 音频重采样工程范例

4.1 简单范例（resample）

FFMpeg自带的resample例子：FFmpeg\doc\examples\resampling_audio.c，这里把最核心的resample代码贴一下，在工程中使用时，注意设置的各种参数，给定的输入数据都不能错。

4.2 复杂范例

结合AVAudioFifo进行封装。