前言

本文档最终解释权归珠海杰理科技股份有限公司。

修订记录

修订时间	修订版本	修订人	修订描述
2020/02/18	V1.0	梁泳新	
2021/08/25	V2.0	梁泳新	

目录

1 音频解码的使用流程

1.1 打开解码服务

```
struct server *dec_server = server_open("audio_server", "dec");
```

1.2 注册解码服务事件回调

注意：服务事件回调是通过任务的队列消息传递的，此消息是不允许丢失的，使用者需要做好相应的异步处理，哪个线程注册该回调函数就是该线程负责接收，特别需要注意不能出现消息队列填满引起的死锁问题。

```
static void dec_server_event_handler(void *priv, int argc, int *argv)
{
    switch (argv[0]) {
    case AUDIO_SERVER_EVENT_END:    //解码结束
        //建议在解码结束时加上暂停操作，避免出现死锁问题
        union audio_req r = {0};
        r.dec.cmd = AUDIO_DEC_PAUSE;
        server_request(dec_server, AUDIO_REQ_DEC, &r);
        break;
    case AUDIO_SERVER_EVENT_CURR_TIME:  //当前播放时间
        log_d("play_time: %d\n", argv[1]);
        break;
    }
}

server_register_event_handler(dec_server, priv, dec_server_event_handler);
```

1.3 解码请求参数解析

```
struct audio_dec_req {
    u8 cmd; //请求操作类型
    u8 status;  //请求后返回的解码状态
    u8 channel; //解码通道数
    u8 volume;  //解码音量(0-100)
    u8 priority;    //解码优先级，暂时没用到
    u8 priority; //解码优先级，暂时没用到
    u16 pitchV; // >32768是音调变高，<32768音调变低，建议范围20000到50000
    u16 attr;   //解码附加属性
    u8 digital_gain_mul;    //数字增益乘值
    u8 digital_gain_div;    //数字增益除值
    u32 output_buf_len; //解码buffer大小
    u32 orig_sr;    //原始采样率，强制变采样时才使用
    u32 sample_rate;    //实际的解码采样率
    u32 ff_fr_step; //快进快退级数
    u32 total_time; //解码的总共时长
    u32 play_time;  //断点恢复时的当前播放时间
    void *output_buf;   //解码缓存buffer，默认填NULL，由解码器自己实现分配和释放
    FILE *file; //需要解码的文件
    char *dec_type; //解码格式
    const char *sample_source;  //播放源
    struct audio_dec_breakpoint *bp;    //断点播放句柄
    const struct audio_vfs_ops *vfs_ops;    //虚拟文件操作句柄
```

```
    void *eq_attr;  //eq属性设置
    void *eq_hdl;   //预先申请好的的eq句柄
    struct audio_cbuf_t *virtual_audio; //虚拟解码句柄，供外部读写使用
    int (*dec_callback)(u8 *buf, u32 len);  //解码后的PCM数据回调
    int (*dec_sync)(void *priv, u32 data_size, u16 *in_rate, u16 *out_rate); //解码对端采样率同步，常用于蓝牙解码
};
```

1.3.1 cmd

完整的解码命令使用流程应该是AUDIO_DEC_OPEN->AUDIO_DEC_START->AUDIO_DEC_PAUSE->AUDIO_DEC_STOP，每一次解码结束后一定要主动调用AUDIO_DEC_STOP释放当前的解码资源，才能再次调用AUDIO_DEC_OPEN，其他指令除了AUDIO_DEC_GET_STATUS外，使用提前是已经调用AUDIO_DEC_OPEN。

```
#define AUDIO_DEC_OPEN                  0    //打开解码
#define AUDIO_DEC_START                 1    //开始解码
#define AUDIO_DEC_PAUSE                 2    //暂停解码
#define AUDIO_DEC_STOP                  3    //停止解码
#define AUDIO_DEC_FF                    4    //快进
#define AUDIO_DEC_FR                    5    //快退
#define AUDIO_DEC_GET_BREAKPOINT        6    //获取断点数据
#define AUDIO_DEC_PP                    7    //暂停/播放
#define AUDIO_DEC_SET_VOLUME            8    //设置解码音量值
#define AUDIO_DEC_DIGITAL_GAIN_SET      9    //设置当前解码的数字增益
#define AUDIO_DEC_PS_PARM_SET           10   //设置变速变调的参数
#define AUDIO_DEC_GET_STATUS            11   //获取当前的解码器状态
```

1.3.2 status

返回当前的解码状态

```
#define AUDIO_DEC_OPEN                  0    //解码已打开
#define AUDIO_DEC_START                 1    //解码已开始
#define AUDIO_DEC_PAUSE                 2    //解码已暂停
#define AUDIO_DEC_STOP                  3    //解码已停止
```

1.3.3 channel

解码通道数
0: 从解码器的格式检查中自动获取   1:单通道   2:双通道
！使用命令->AUDIO_DEC_OPEN

1.3.4 volume

音量取值范围为0-100
！使用命令->AUDIO_DEC_OPEN或AUDIO_DEC_SET_VOLUME

1.3.5 attr

```
AUDIO_ATTR_REAL_TIME = BIT(0),  //保证解码的实时性，解码读数不能堵塞，仅限于蓝牙播歌时时钟同步使用
AUDIO_ATTR_LR_SUB    = BIT(1),  //伴奏功能，只支持双声道
AUDIO_ATTR_PS_EN     = BIT(2),  //变速变声功能开关
AUDIO_ATTR_LR_ADD    = BIT(3),  //左右通道数据叠加
AUDIO_ATTR_DECRYPT_DEC = BIT(4),//文件解密播放，需要配合对应的加密工具
AUDIO_ATTR_FADE_INOUT  = BIT(5),//模拟音量淡入淡出，解码开始和暂停时使用
AUDIO_ATTR_EQ_EN       = BIT(6),//EQ功能开关
AUDIO_ATTR_DRC_EN      = BIT(7),//DRC功能开关，使能时需要打开EQ功能
AUDIO_ATTR_EQ32BIT_EN  = BIT(8),//EQ 32bit输出
AUDIO_ATTR_BT_AAC_EN   = BIT(9),//蓝牙AAC解码
```

1.3.6 digital_gain_mul和digital_gain_div

数字增益乘值和除值
！使用命令->AUDIO_DEC_DIGITAL_GAIN_SET

digital_gain_mul	digital_gain_div	效果
0	0	关闭数字增益
0xff	0xff	静音
1 ~ 0xfe	0	value * mul
0	1 ~ 0xfe	value / div
1 ~ 0xfe	1 ~ 0xfe	value * mul /div

1.3.7 output_buf_len和output_buf

output_buf_len必须填非0值，output_buf默认填NULL，由解码器自己实现分配和释放资源
！使用命令->AUDIO_DEC_OPEN

1.3.8 orig_sr和sample_rate

orig_sr为非0值时，启用强制变采样解码，orig_sr为原始采样率，sample_rate为变采样后的采样率，目前仅用于混响功能上。
！使用命令->AUDIO_DEC_OPEN

1.3.9 total_time

当请求打开解码后，该参数保存当前解码的播放总时长，一般是从解码器的格式检查中获取。
！使用命令->AUDIO_DEC_OPEN

1.3.10 vfs_ops和file

当vfs_ops为空时，默认为解码文件操作，此时file不能为空，file需要赋值为fopen操作成功后返回的文件句柄，当解码结束后用户自己需要调用fclose关闭文件。
当vfs_ops非空时，解码器的解码数据源读取操作都通过该虚拟文件操作句柄获取，此时file参数可传入用户的私有数据指针，具体例子如下代码的net_audio_dec_vfs_ops。
！使用命令->AUDIO_DEC_OPEN

```
static const struct audio_vfs_ops net_audio_dec_vfs_ops = {
    .fread = net_download_read,
    .fseek = net_download_seek,
    .flen  = net_download_get_file_len,
};
```

1.3.11 dec_type

当前解码格式支持mp3、m4a、ape、flac、wav、amr、pcm、adpcm、wma、aac、spx、sbc、cvsd、msbc、opus。
！使用命令->AUDIO_DEC_OPEN

1.3.12 sample_source

播放源默认为"dac"，还支持"IIS0"和"IIS1"硬件输出。
！使用命令->AUDIO_DEC_OPEN

1.3.13 bp

！使用命令->AUDIO_DEC_OPEN，bp非空时作用是恢复该断点播放。
！使用命令->AUDIO_DEC_GET_BREAKPOINT，bp保存下当前解码的断点数据，获取后的bp->data需要用户自行释放内存。

1.3.14 eq_attr和eq_hdl

eq_attr为空时，启用eq功能，用户需要配置好合适的eq参数，请求后eq_hdl返回唯一的eq句柄，所有解码器都是共用同一个eq句柄。
！使用命令->AUDIO_DEC_OPEN
注意：这两个函数仅限于AC790X旧EQ工具使用，新EQ工具无效

1.3.15 pitchV和speedV

```
u16 pitchV; // >32768是音调变高，<32768音调变低，建议范围20000到50000
u8 speedV;  // >80是变快，<80是变慢，建议范围：30到130
```
！使用命令->AUDIO_DEC_OPEN和AUDIO_DEC_PS_PARM_SET

1.4 关闭解码服务

```
server_close(dec_server);
```

1.5 DAC详细配置

```
//一路DAC输出
static const struct dac_platform_data dac_data = {
    .pa_auto_mute = 1,
    .pa_mute_port = IO_PORTC_03,    //功放的MUTE IO引脚
    .pa_mute_value = 1, //高电平MUTE
    .differ_output = 0,
    .hw_channel = 0x01,
    .ch_num = 1,
};

//两路DAC输出，双声道模式
static const struct dac_platform_data dac_data = {
    .pa_auto_mute = 1,
    .pa_mute_port = IO_PORTC_03,    //功放的MUTE IO引脚
    .pa_mute_value = 1, //高电平MUTE
    .differ_output = 0,
    .hw_channel = 0x03,
    .ch_num = 2,
};

//两路DAC输出，差分模式
static const struct dac_platform_data dac_data = {
    .pa_auto_mute = 1,
    .pa_mute_port = IO_PORTC_03,    //功放的MUTE IO引脚
    .pa_mute_value = 1, //高电平MUTE
    .differ_output = 1, //差分输出
    .hw_channel = 0x01,
    .ch_num = 1,    //差分只需开一个通道
};

//四路DAC输出，双声道模式
static const struct dac_platform_data dac_data = {
    .pa_auto_mute = 1,
    .pa_mute_port = IO_PORTC_03,    //功放的MUTE IO引脚
    .pa_mute_value = 1, //高电平MUTE
    .differ_output = 0,
    .hw_channel = 0x0f,
    .ch_num = 4,
};

//四路DAC输出，差分模式
static const struct dac_platform_data dac_data = {
    .pa_auto_mute = 1,
    .pa_mute_port = IO_PORTC_03,    //功放的MUTE IO引脚
    .pa_mute_value = 1, //高电平MUTE
    .differ_output = 1, //差分输出
    .hw_channel = 0x05,
    .ch_num = 2,    //差分只需开一个通道
};
```

2 音频编码的使用流程

2.1 打开编码服务

```
struct server *enc_server = server_open("audio_server", "enc");
```

2.2 注册编码服务事件回调

注意：服务事件回调是通过任务的队列消息传递的，此消息是不允许丢失的，使用者需要做好相应的异步处理，哪个线程注册该回调函数就是该线程负责接收，特别需要注意不能出现消息队列填满引起的死锁问题。

```
static void enc_server_event_handler(void *priv, int argc, int *argv)
{
    switch (argv[0]) {
    case AUDIO_SERVER_EVENT_END:    //编码结束
        break;
    case AUDIO_SERVER_EVENT_ERR:    //编码错误
        break;
    case AUDIO_SERVER_EVENT_SPEAK_START: //VAD检测到开始说话
        break;
    case AUDIO_SERVER_EVENT_SPEAK_STOP: //VAD检测到停止说话
        break;
    }
}

server_register_event_handler(enc_server, priv, enc_server_event_handler);
```

2.3 编码请求参数解析

```
struct audio_enc_req {
    u8 cmd; //请求操作类型
    u8 status; //编码器状态
    u8 channel; //同时编码的通道数
    u8 channel_bit_map; //ADC通道选择
    u8 volume; //ADC增益(0-100)，编码过程中可以通过AUDIO_ENC_SET_VOLUME动态调整增益
    u8 priority; //编码优先级，暂时没用到
    u8 use_vad : 1; //是否使用VAD功能
    u8 vad_auto_refresh : 1; //是否自动刷新VAD状态，赋值1表示SPEAK_START->SPEAK_STOP->SPEAK_START->SPEAK_STOP->....循环
    u8 direct2dac : 1;  //AUDIO_AD直通DAC功能
    u8 high_gain : 1;  //直通DAC时是否打开强增益
    u8 amr_src : 1;    //amr编码时的强制16k变采样为8kpcm数据，因为amr编码器暂时只支持8k编码
    u8 aec_enable : 1; //AEC回声消除功能开关，常用于蓝牙通话
    u8 ch_data_exchange : 1; //用于AEC差分回采时和MIC的通道数据交换
    u8 no_header : 1; //用于opus编码时是否需要添加头部格式
    u8 vir_data_wait : 1; //虚拟编码时是否允许丢失数据
    u8 no_auto_start : 1; //请求AUDIO_ENC_OPEN时不自动运行编码器，需要主动调用AUDIO_ENC_START
    u8 sample_depth : 6; //采样深度16bit或者24bit
    u16 vad_start_threshold;    //VAD连续检测到声音的阈值，表示开始说话，回调AUDIO_SERVER_EVENT_SPEAK_START，单位ms，填0使用库内默认值
    u16 vad_stop_threshold;     //VAD连续检测到静音的阈值，表示停止说话，回调AUDIO_SERVER_EVENT_SPEAK_STOP，单位ms,填0使用库内默认值
    u16 frame_size; //编码器输出的每一帧帧长大小，只有pcm格式编码时才有效
```

```
    u16 frame_head_reserve_len; //编码输出的帧预留头部的大小
    u32 bitrate; //编码码率大小
    u32 output_buf_len; //编码buffer大小
    u32 sample_rate; //采样率
    u32 msec; //编码时长，填0表示一直编码，单位ms，编码结束会回调AUDIO_SERVER_EVENT_END消
息
    FILE *file; //编码输出文件句柄
    u8 *output_buf; //编码buffer，默认填NULL，由编码器自动分配和释放资源
    const char *format;          //编码格式
    const char *sample_source;  //采样源，支
持"mic","linein","plnk0","plnk1"，"virtual"，“iis0”，“iis1”
    const struct audio_vfs_ops *vfs_ops; //虚拟文件操作句柄
    u32(*read_input)(u8 *buf, u32 len); //用于虚拟采样源"virtual"编码时的数据读取操作
    void *aec_attr; //AEC回声消除算法配置参数
};
```

2.3.1 cmd

完整的编码命令使用流程应该是AUDIO_ENC_OPEN->AUDIO_ENC_CLOSE，其他命令暂时无效，每一次编码结束后一定要主动调用AUDIO_ENC_CLOSE释放当前的资源，才能再次调用AUDIO_ENC_OPEN。

2.3.2 channel和channel_bit_map

编码通道数同时支持四路，需要哪一路数据就填BIT(x)

2.3.3 format

当前编码格式支持spx、opus、wav、amr、pcm、cvsd、msb、sbc。

2.3.4 vfs_ops和file

当vfs_ops为空时，默认编码封装成文件，此时file不能为空，file需要赋值为fopen操作成功后返回的文件句柄，当编码结束后用户自己需要调用fclose关闭文件。
当vfs_ops非空时，编码器编码后的数据写入操作都通过该虚拟文件操作句柄，此时file参数可传入用户的私有数据指针，具体例子如下代码的reverberation_vfs_ops。

```
static int reverberation_vfs_fwrite(void *file, void *data, u32 len)
{
    //此函数内一定不能堵塞
    return len; //返回0可以强制触发编码结束，会有回调消息AUDIO_SERVER_EVENT_ERR
}

static int reverberation_vfs_fclose(void *file)
{
    return 0;
}

static const struct audio_vfs_ops reverberation_vfs_ops = {
    .fwrite = reverberation_vfs_fwrite,
    .fclose = reverberation_vfs_fclose,
};
```

2.4 关闭编码服务

```
server_close(enc_server);
```

2.5 ADC详细配置

```
具体请阅读audio_demo使用说明.pdf
```