AudioManager 架构说明

V1.0

History

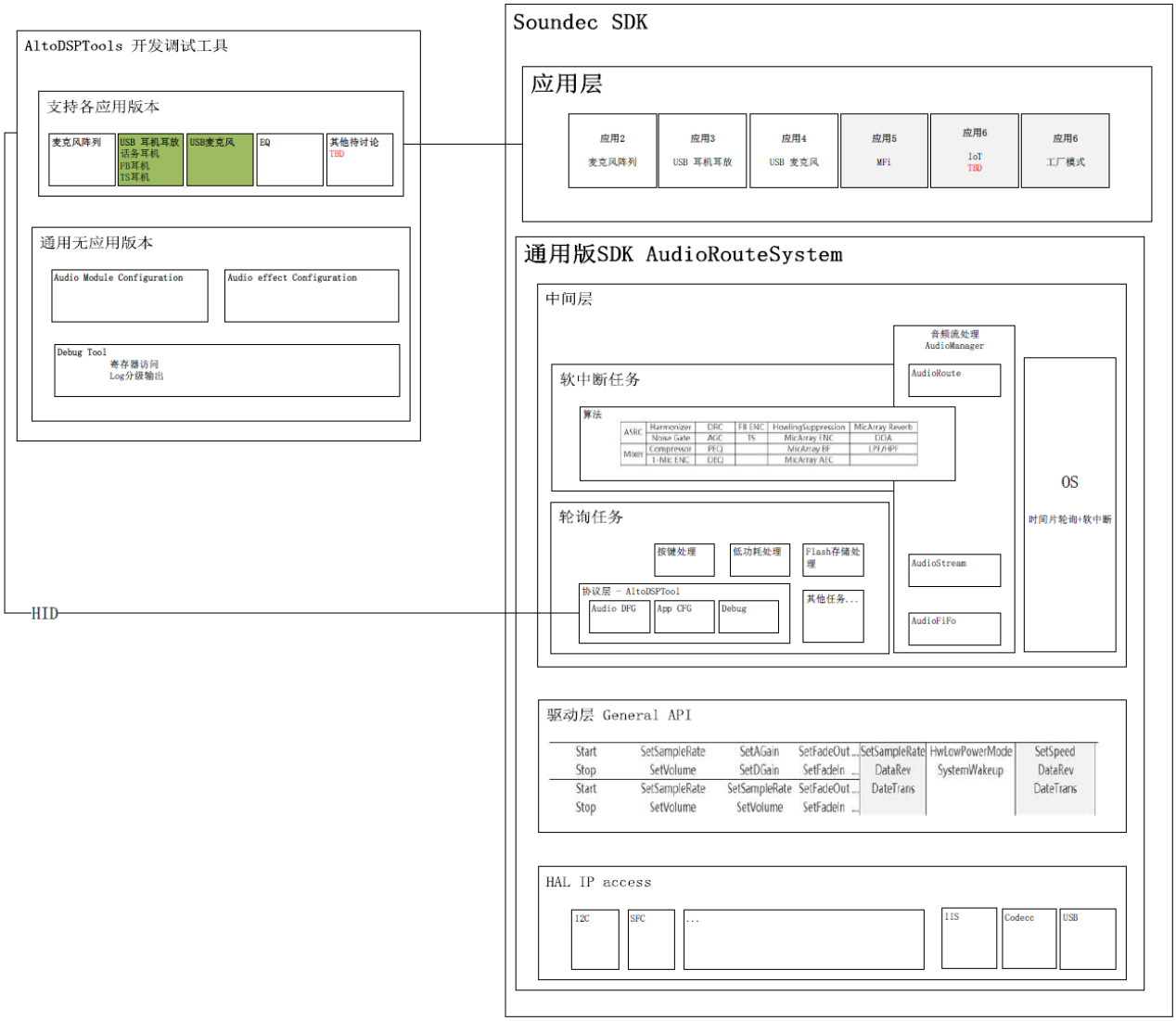
版本	发布时间	版本说明	作者	核准人
V1.0	2022-4-10	1st Release	白蓉	

目录

History 2
1 Soundec32CxxSDK2.0 简介 4
2 Audio Manager 简介 4
2.1 Device/Session 架构 5
2.1.1 Device 架构 6
2.1.2 Session 架构 6
2.2 Audio Manager 的使用 7
2.2.1 Audio Manager 设备&事件注册 8

1 Soundec32CxxSDK2.0 简介

1.1 SDK 完整架构说明



2 Audio Manager 简介

Audio Manager 采用 Device/Session 的架构设计，通过 Audio Device 管理音频硬件设备的挂载和驱动，通过 Audio Session 管理 PCM 数据流的输入输出

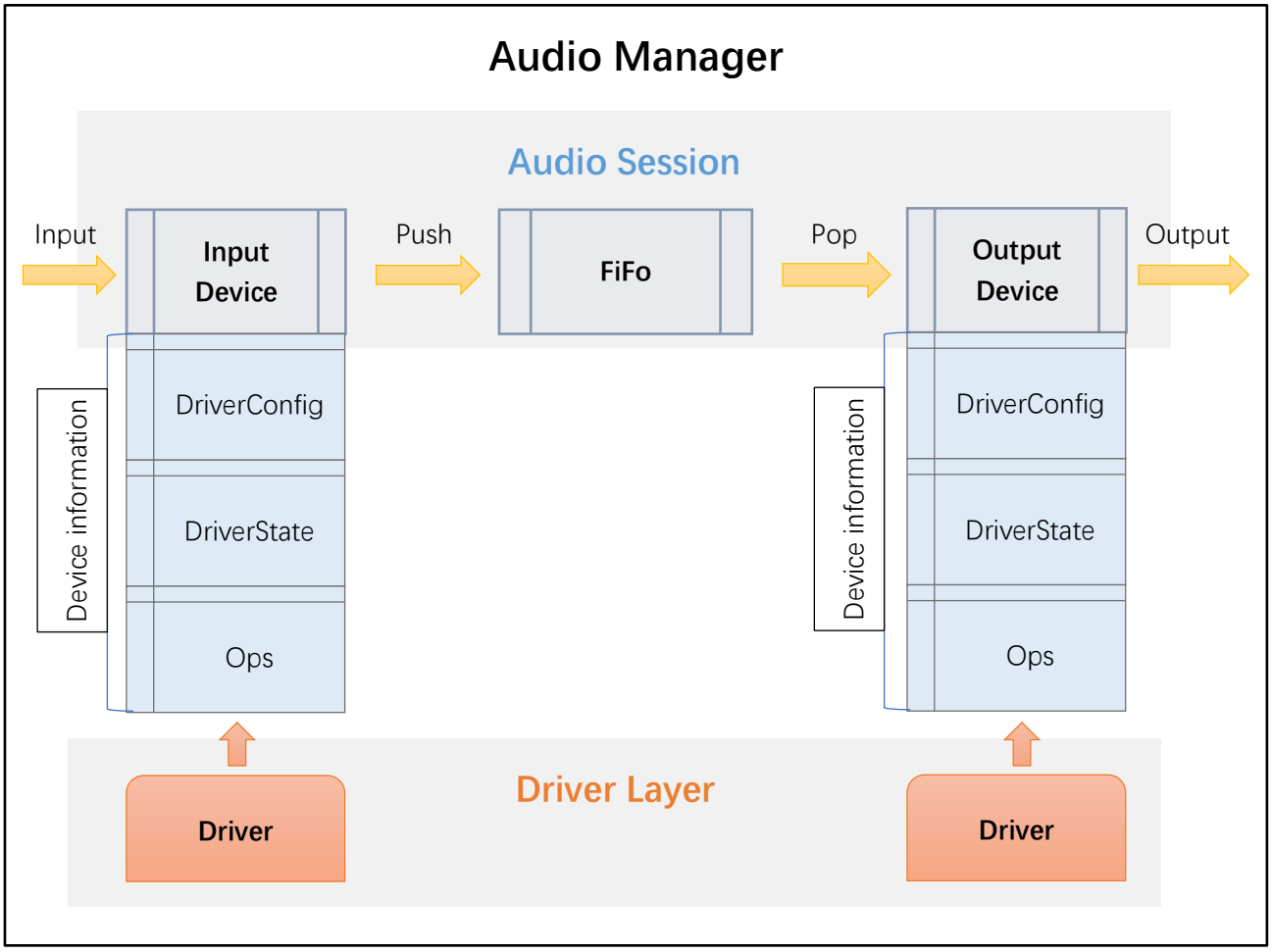
系统提供两个接口文件供用户自由配置音频流：

1） Audio Device Interface：见文件“audio_hw_desc.c”

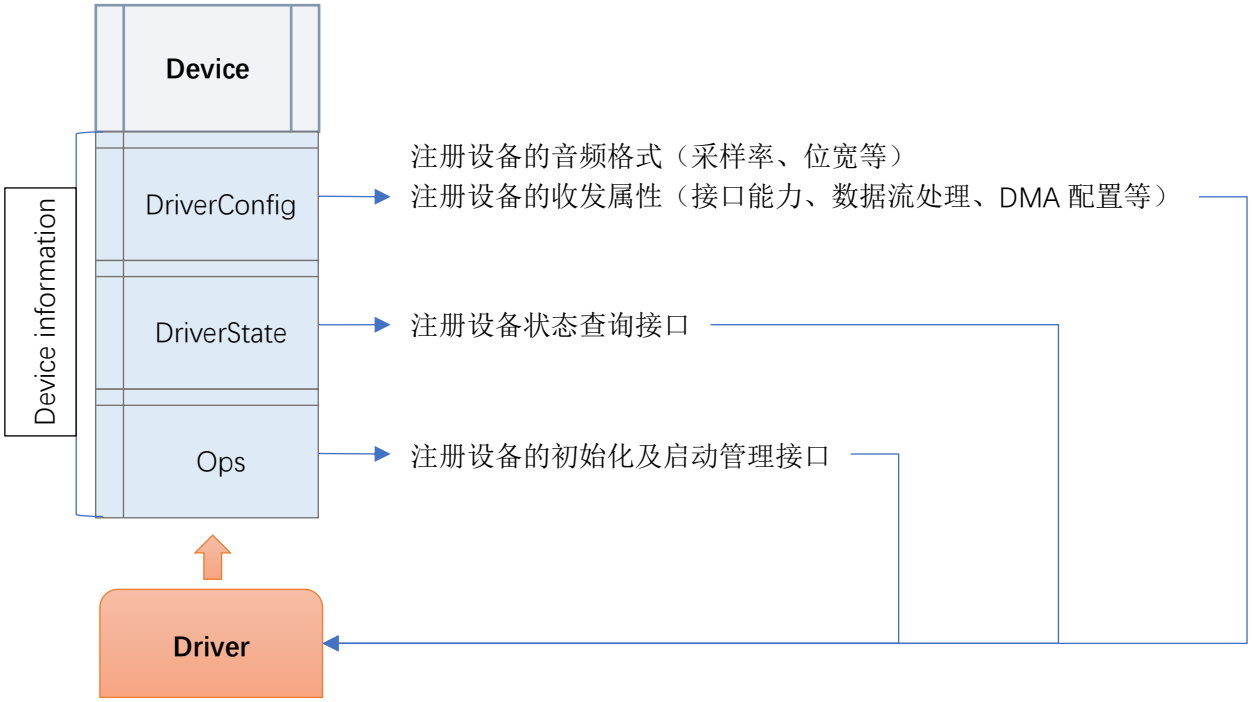
2） Audio Session Interface：见文件”audio_session_desc_xxx.c”

以上文件的详细使用说明见后文 Audio Manager 设备&事件注册

2.1 Device/Session 架构

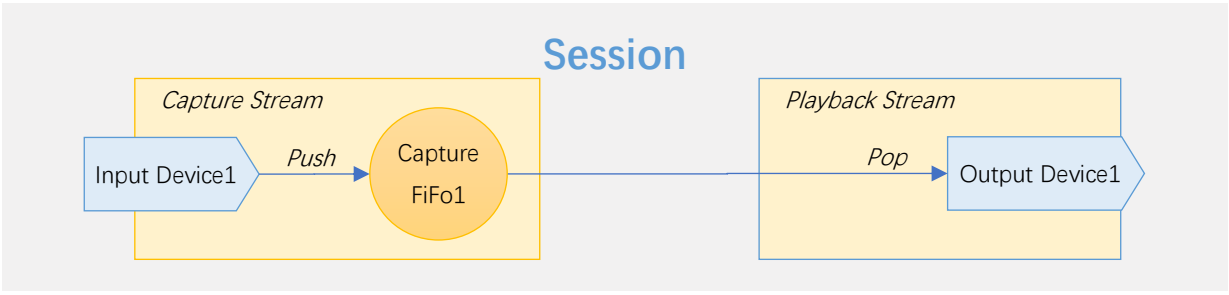


2.1.1 Device 架构

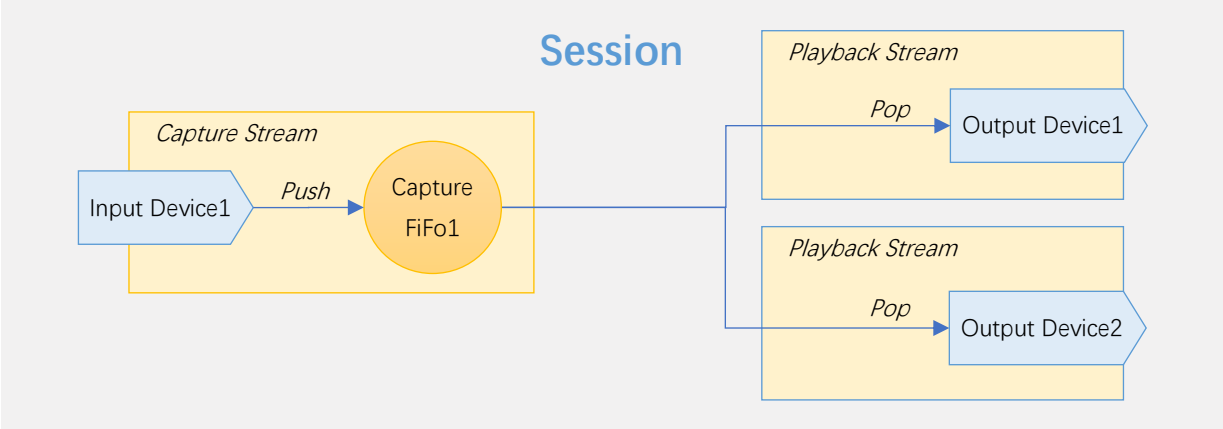


2.1.2 Session 架构

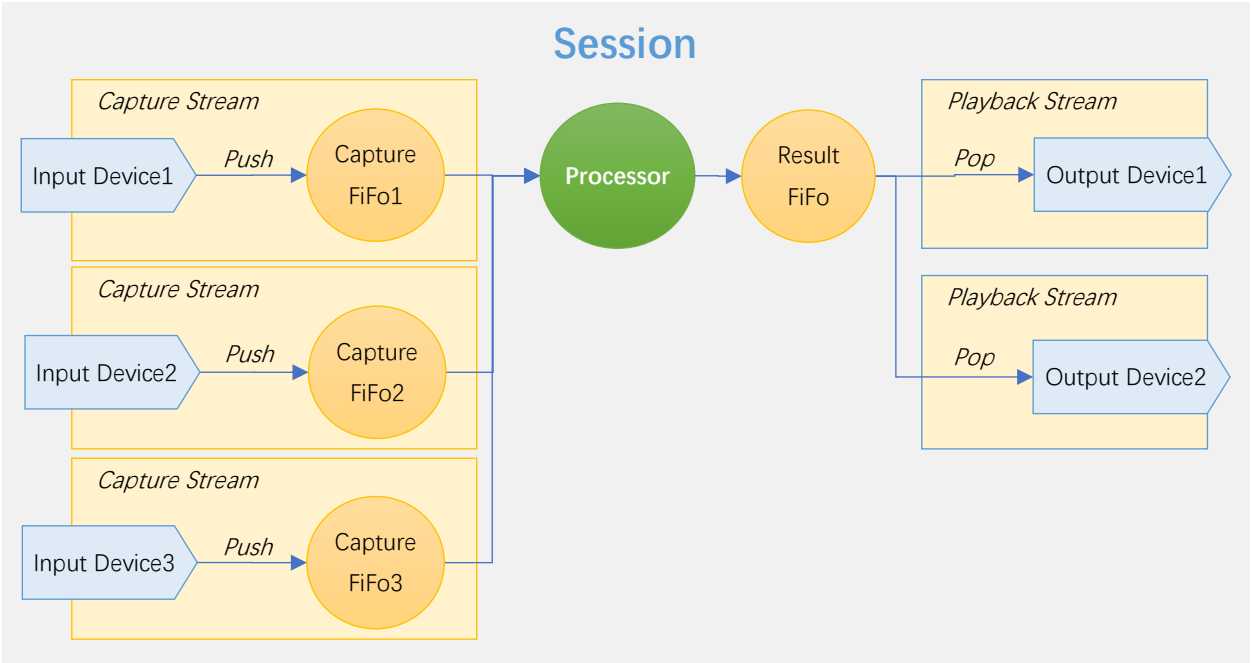
单入单出 Session



单入多出 Session 通路

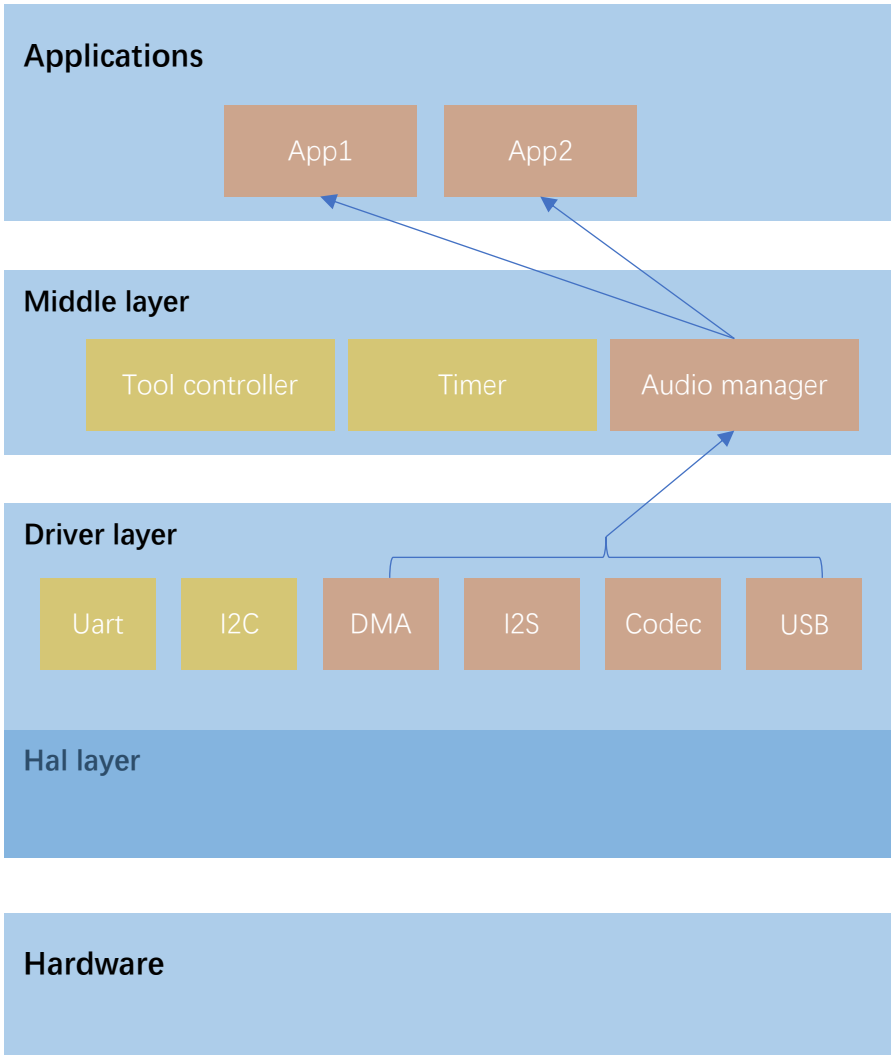


多入多出 Session 通路



2.2 Audio Manager 的使用

使用 Audio Manager 配置音频通路，不需要考虑内部实现逻辑，也不需要考虑音频设备的初始化调用等流程。用户只需要根据 Session 架构，在 audio_session_desc_xxx.c 中描述音频走向及处理格式：音频的数据来源、音频源数量、源数据的处理（注册算法处理函数）、选择输出设备即可。



- 注册硬件/加载音频设备
- 注册 Session/打开对应通路
- 数据流写数据

2.2.1 Audio Manager 设备&事件注册

2.2.1.1 注册硬件/加载音频设备

硬件设备的注册接口见文件 audio_hw_desc.c，通过其中 auDeviceIfs 数组定义了包含 Codec（ADC&DAC）、I2S（3 组）、USB（Mic&Speaker）音频设备及其描述接口

```
audioDevIf_t auDeviceIfs[AUDIO_DEVICE_MAX] = {
#if (CODEC_ADC_ENABLE)
    {
        .id = AUDIO_DEVICE_ADC_IN,
        .load = FALSE,
        .name = "adc",
        .rxCfgs = &auDrvCfg_adc,
        .txCfgs = NULL,
        .state = &auDrvSt_adc_in,
        .ops = &auDrvOps_adc_in,
        /*.userData = NULL;*/
    },
#endif
#if (CODEC_DAC_ENABLE)
    {
        .id = AUDIO_DEVICE_CODEC_OUT,
        .name = "codec",
        .load = FALSE,
        .rxCfgs = NULL,
        .txCfgs = &auDrvCfg_dac,
        .state = &auDrvSt_dac,
        .ops = &auDrvOps_dac,
        /*.userData = NULL;*/
    },
#endif
```

...其他音频模块配置描述...

```
#if (USB_ENABLE == 1)
    {
        .id = AUDIO_DEVICE_USB,
        .name = "usb",
        .load = FALSE,
        .rxCfgs = &auDrvCfg_usb_speaker,
        .txCfgs = &auDrvCfg_usb_mic,
        .state = &auDrvSt_usb,
        .ops = &auDrvOps_usb,
        /*.userData = NULL;*/
    },
#endif
};
```

2.2.1.2 注册 Session/打开对应通路

音频通路 Session 的注册接口见文件 audio_session_desc_xx.c，以双麦会议音箱应用为例，说明音频描述文件 audio_session_desc_2mic_meetingbox.c 的配置。此配置支持两路音频通路：AUDIO_SESSION_0 和 AUDIO_SESSION_1。

- AUDIO_SESSION_0 支持 2 路输入（capture），1 路输出（playback），输入信号经过（session0Proc）处理后输出；
- AUDIO_SESSION_1 支持 1 路输入（capture），1 路输出（playback），输入信号不需要处理，直接输出：

```
auSessionDesc_t auSessionDesc[AUDIO_SESSION_MAX] = {
    {
        .id = AUDIO_SESSION_0,
        .sessionPolicy = &session0Policy,
        .captureNum = 2,
        .capture = session0StreamIn,

        .procIf = &session0Proc,

        .playbackNum = 1,
        .playback = session0StreamOut,

    },
    {
        .id = AUDIO_SESSION_1,
        .captureNum = 1,
        .capture = session1StreamIn,

        .procIf = NULL,

        .playbackNum = 1,
        .playback = session1StreamOut,

    },
};
```

通过调用 audioManager_init()接口，系统会根据 Session 中的通路描述，以及 capture 和 playback 中的格式说明，自动挂载对应的音频设备，完成 Audio manager 的初始化。

通过调用 audioManager_open_session(sessionID) 接口，实现相关通路的开启。

通过调用 audioManager_close_session(sessionID) 接口，实现相关通路的关闭。

2.2.1.3 数据流的读写处理

经过 audioManager_init()的调用，所有的已挂载的音频设备的数据接收和发送，都将在统一的接口中进行处理：fifo_push/fifo_pop

```
int8_t auRxCompleteCB(void *arg, void *data, uint32_t *len, auSlotSize_t bitSlot)
{
    if (arg == NULL)  return AUDIO_MNGR_FAIL;
    auStream_t *stream = (auStream_t *)arg;

    if (stream->fifo && (stream->fifo->fuc->push))
    {
#ifdef TEST_AU_MANAGER ...
#endif
        stream->fifo->fuc->push(stream->fifo, data, *len, bitSlot);

        if (stream->ops->tuneSampleRate)
            stream->ops->tuneSampleRate(stream->fifo->fuc->get_data_len(stream->fifo), stream->fifo->fuc->get_fifo_size(stream->fifo));
    }
    return AUDIO_MNGR_FAIL;
}

/* Len: in Bytes*/
int8_t auTxCompleteCB(void *arg, void *data, uint32_t *len, auSlotSize_t bitSlot)
{
    if (arg == NULL)  return AUDIO_MNGR_FAIL;
    auStream_t *stream = (auStream_t *)arg;

    if (stream->fifo && stream->fifo->fuc->pop)
    {
#ifdef TEST_AU_MANAGER ...
#endif
        stream->fifo->fuc->pop(stream->fifo, data, len, bitSlot);

        if (stream->ops->tuneSampleRate)
            stream->ops->tuneSampleRate(stream->fifo->fuc->get_data_len(stream->fifo), stream->fifo->fuc->get_fifo_size(stream->fifo));
    }
    return AUDIO_MNGR_FAIL;
}
```

2.2.1.4 数据缓存处理和优化

所有音频数据的缓存区的创建、数据缓存、格式转换、优化管理，通过 fifo_manager 实现；

```
typedef struct {
    void (*init)                    (struct fifo_handle_st *p_fifo);    /*len in byte*/
    void (*reset)                   (struct fifo_handle_st *p_fifo, auStrmFormat_t streamFormat);
    void (*skip)                    (struct fifo_handle_st *p_fifo, uint32_t pos, uint32_t len);
    void (*insert)                  (struct fifo_handle_st *p_fifo, uint32_t pos, uint32_t len);

    uint32_t (*push)                (struct fifo_handle_st *p_fifo, uint8_t *__restrict data, uint32_t sizeInByte, auSlotSize_t bitWidth);
    uint32_t (*pop)                 (struct fifo_handle_st *p_fifo, uint8_t *data, uint32_t *len, auSlotSize_t bitWidth);
    uint32_t (*push_skip)           (struct fifo_handle_st *p_fifo, uint32_t len, auSlotSize_t bitWidth);
    uint32_t (*pop_skip)            (struct fifo_handle_st *p_fifo, uint32_t *len, auSlotSize_t bitWidth);
    uint32_t (*get_fifo_head)       (struct fifo_handle_st *p_fifo);//return readPtr
    uint32_t (*get_fifo_tail)       (struct fifo_handle_st *p_fifo);//return writePtr

//  uint32_t (*get_fifo)            (struct fifo_handle_st *p_fifo);
    bool (*is_empty)                (struct fifo_handle_st *p_fifo);
    bool (*is_full)                 (struct fifo_handle_st *p_fifo);
    bool (*is_enough)               (struct fifo_handle_st *p_fifo);

    uint32_t (*get_free_len)        (struct fifo_handle_st *p_fifo);
    uint32_t (*get_data_len)        (struct fifo_handle_st *p_fifo);
    uint32_t (*get_fifo_size)       (struct fifo_handle_st *p_fifo);
    uint32_t (*get_fifo_thresholdLevel)     (struct fifo_handle_st *p_fifo);
    uint8_t (*get_fifo_share)       (struct fifo_handle_st *p_fifo);
    fifoOffset_t (*get_status)      (struct fifo_handle_st *p_fifo);
    uint32_t (*add_share_fifo)      (struct fifo_handle_st *p_srcFifo, struct fifo_handle_st *p_shareFifo);

    uint32_t (*get_sdata)               (struct fifo_handle_st *p_fifo);

} « end {anonfifo_ctrl_t} » fifo_ctrl_t;
```

通过 AudioManager 中 stream 的相关控制，调用到底层 fifo_manager；以上为 fifo 公开接口，可在应用需要的地方进行调用