OneMO | 中国移动 China Mobile

ML307A
音频开发指导手册

版本：V1.0.0
发布日期：2022/12/9

China Mobile IoT Co., Ltd.
iot.10086.cn

服务与支持

如果您有任何关于模组产品及产品手册的评论、疑问、想法，或者任何无法从本手册中找到答案的疑问，请通过以下方式联系我们。

OneMO官网： onemo10086.com

邮箱： SmartModule@cmiot.chinamobile.com

客户服务热线：400-110-0866

文档声明

注意

本手册描述的产品及其附件特性和功能，取决于当地网络设计或网络性能，同时也取决于用户预先安装的各种软件。由于当地网络运营商、ISP，或当地网络设置等原因，可能也会造成本手册中描述的全部或部分产品及其附件特性和功能未包含在您的购买或使用范围之内。

责任限制

除非合同另有约定，中移物联网有限公司对本文档内容不做任何明示或暗示的声明或保证，并且不对特定目的适销性及适用性或者任何间接的、特殊的或连带的损失承担任何责任。

在适用法律允许的范围内，在任何情况下，中移物联网有限公司均不对用户因使用本手册内容和本手册中描述的产品而引起的任何特殊的、间接的、附带的或后果性的损坏、利润损失、数据丢失、声誉和预期的节省而负责。

因使用本手册中所述的产品而引起的中移物联网有限公司对用户的最大赔偿（除在涉及#身伤害的情况中根据适用法律规定的损害赔偿外），不应超过用户为购买此产品而支付的金额。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。公司保留随时修改本手册中任何信息的权利，无需进行提前通知且不承担任何责任。

商标声明

中国移动 China Mobile 为中国移动注册商标。

本手册和本手册描述的产品中出现的其他商标、产品名称、服务名称和公司名称，均为其各自所有者的财产。

进出口法规

出口、转口或进口本手册中描述的产品（包括但不限于产品软件和技术数据），用户应遵守相关进出口法律和法规。

隐私保护

关于我们如何保护用户的个人信息等隐私情况，请查看相关隐私政策。

操作系统更新声明

操作系统仅支持官方升级；如用户自己刷非官方系统，导致安全风险和损失由用户负责。

固件包完整性风险声明

固件仅支持官方升级；如用户自己刷非官方固件，导致安全风险和损失由用户负责。

版权所有©中移物联网有限公司。保留一切权利。

关于文档

修订记录

版本	描述
V1.0.0	初版

目录

1. 概述

本文档介绍了OpenCPU SDK下音频功能，旨在为客户使用SDK进行应用程序开发提供参考，客户可参照本文档和SDK中的示例程序进行应用程序编写。

模组内部已集成codec电路，可直接使用扬声器、麦克风等音频设备。

1.1. 适用范围

Table 1. 适用模组

模组系列	模组子型号
ML307A	ML307A-DCLN/ML307A-DSLN

2. 功能说明

OpenCPU SDK支持音频功能开发，包括音频播放和录制音频功能。

2.1. 播放音频说明

本节主要介绍播放音频功能。

OpenCPU SDK音频播放支持一路输出通道（DSLN支持音频播放和在线TTS播放，DCLN支持在线TTS播放）。

OpenCPU SDK音频播放支持以下格式：

- pcm格式：采样率、声道等设置详见cm_audio_common.h文件；
- mp3格式：支持mp3格式，无限制；
- amr格式：支持amr格式，无限制。

2.2. 录制音频说明

本节主要介绍了录制音频功能。

OpenCPU SDK音频录制支持一路输入通道（DSLN支持音频录制，DCLN不支持音频录制）。

OpenCPU SDK音频录制支持如下音频格式：

- pcm格式：采样率、声道等设置详见cm_audio_common.h文件；
- amr格式：支持amr格式，编码方式设置详见cm_audio_common.h文件。

3. 硬件电路设计

硬件电路设计参考对应型号硬件设计手册。

4. API说明

SDK提供一套完整的音频功能用户编程接口，具体接口定义和说明请查阅cm_audio_player.h、cm_audio_recorder.h和cm_audio_common.h头文件。对于支持TTS功能的模组，具体接口定义和说明请查阅cm_tts.h头文件。

常用音频功能参数结构体如下（不同模组或SDK版本间可能存在差异，请以SDK头文件为准）。

```
/** PCM音频采样通道 */
typedef enum
{
  CM_AUDIO_SOUND_MONO = 1, /*!< （默认）单通道，录音时默认且只能使用CM_AUDIO_SOUND_MONO */
  CM_AUDIO_SOUND_STEREO = 2, /*!< 双通道（立体声） */
} cm_audio_sound_channel_e;

/** 音频采样参数结构体 */
typedef struct
{
  cm_audio_sample_format_e sample_format; /*!< 采样格式（PCM） */
  cm_audio_sample_rate_e rate; /*!< 采样率（PCM） */
  cm_audio_sound_channel_e num_channels; /*!< 采样声道（PCM） */
} cm_audio_sample_param_t;
```

常用音频功能接口定义如下（不同模组或SDK版本间可能存在差异，请以SDK头文件为准）。

```
/**
 * @brief 设置播放参数
 *
 * @param [in] type 设置参数类型
 * @param [in] value 设置参数数值
 *
 * @return
 * = 0 - 成功 \n
 * = -1 - 失败
 */
int32_t cm_audio_play_set_cfg(cm_audio_play_cfg_type_e type, void *value);


/**
 * @brief 读取播放参数
 *
 * @param [in] type 读取参数类型
 * @param [out] value 读取参数数值
 *
 * @return
 * = 0 - 成功 \n
 * = -1 - 失败
 */
int32_t cm_audio_play_get_cfg(cm_audio_play_cfg_type_e type, void *value);


/**
```

```
 * @brief 从内存中播放音频数据
 *
 * @param [in] data 播放音频数据
 * @param [in] size 播放音频数据长度
 * @param [in] format 播放音频格式
 * @param [in] sample_param 播放音频PCM采样参数（format参数为CM_AUDIO_PLAY_FORMAT_PCM使用，其余情况传入NULL）
 * @param [in] cb 音频播放回调函数（回调函数在音频处理线程中被执行）
 * @param [in] cb_param 用户参数（与cm_audio_play_callback回调函数中param参数相对应，长度最大为8）
 *
 * @return
 * = 0 - 成功 \n
 * = -1 - 失败
 *
 * @details 在播放器启动之前，整个流应该位于内存中
 */
int32_t cm_audio_play(const void *data, uint32_t size, cm_audio_play_format_e format, cm_audio_sample_param_t
*sample_param, cm_audio_play_callback cb, void *cb_param);


/**
 * @brief 从文件系统播放音频文件
 *
 * @param [in] path 文件路径/名称
 * @param [in] format 播放格式
 * @param [in] sample_param 播放音频PCM采样参数（format参数为CM_AUDIO_PLAY_FORMAT_PCM使用，其余情况传入NULL）
 * @param [in] cb 音频播放回调函数（回调函数在音频处理线程中被执行）
 * @param [in] cb_param 用户参数（与cm_audio_play_callback回调函数中param参数相对应，长度最大为8）
 *
 * @return
 * = 0 - 成功 \n
 * = -1 - 失败
 *
 * @details 从文件系统播放mp3文件时，不支持暂停播放和继续播放操作
 */
int32_t cm_audio_play_file(const char *path, cm_audio_play_format_e format, cm_audio_sample_param_t *sample_param,
cm_audio_play_callback cb, void *cb_param);


/**
 * @brief 暂停播放
 *
 * @return
 * = 0 - 成功 \n
 * = -1 - 失败
 */
int32_t cm_audio_player_pause(void);


/**
 * @brief 继续播放
 *
 * @return
 * = 0 - 成功 \n
 * = -1 - 失败
 */
int32_t cm_audio_player_resume(void);
```

```
/**
 * @brief 停止播放
 *
 * @return
 * = 0 - 成功 \n
 * = -1 - 失败
 */
int32_t cm_audio_player_stop(void);


/**
 * @brief 从管道/消息队列中播放音频（开启）
 *
 * @param [in] format 播放格式（暂只支持PCM）
 * @param [in] sample_param 播放音频PCM采样参数（format参数为CM_AUDIO_PLAY_FORMAT_PCM使用，其余情况传入NULL）
 *
 * @return
 * = 0 - 成功 \n
 * = -1 - 失败
 *
 * @details 播放缓冲区长度为8192
 */
int32_t cm_audio_player_stream_open(cm_audio_play_format_e format, cm_audio_sample_param_t *sample_param);


/**
 * @brief 往管道/消息队列中发送要播放的音频数据
 *
 * @param [in] data 播放的数据
 * @param [in] size 播放数据的长度
 *
 * @return
 * >= 0 - 实际写入的数据长度 \n
 * = -1 - 失败
 *
 * @details 第一次调用时必须包含完整的帧头信息用于解析（非PCM格式数据的情况）
 */
int32_t cm_audio_player_stream_push(uint8_t *data, uint32_t size);


/**
 * @brief 从管道/消息队列中播放音频（关闭）
 *
 * @details More details
 */
void cm_audio_player_stream_close(void);


/**
 * @brief 设置录音参数
 *
 * @param [in] type 设置参数类型
 * @param [in] value 设置参数数值
 *
```

```
 * @return
 * = 0 - 成功 \n
 * = -1 - 失败
 *
 * @details 本版暂不支持增益设置功能
 */
int32_t cm_audio_record_set_cfg(cm_audio_record_cfg_type_e type, void *value);


/**
 * @brief 读取录音参数
 *
 * @param [in] type 读取参数类型
 * @param [out] value 读取参数数值
 *
 * @return
 * = 0 - 成功 \n
 * = -1 - 失败
 *
 * @details 本版暂不支持增益获取功能
 */
int32_t cm_audio_record_get_cfg(cm_audio_record_cfg_type_e type, void *value);


/**
 * @brief 开始录音
 *
 * @param [in] format 录制音频格式
 * @param [in] sample_param 录制音频PCM采样参数
 * @param [in] cb 录音回调函数（回调函数不能阻塞）
 * @param [in] cb_param 用户参数（参见cm_audio_record_callback回调函数中param参数描述，长度最大为8）
 *
 * @return
 * = 0 - 成功 \n
 * = -1 - 失败
 *
 * @details 回调函数不能阻塞
 */
int32_t cm_audio_recorder_start(cm_audio_record_format_e format, cm_audio_sample_param_t *sample_param,
 cm_audio_record_callback cb, void *cb_param);


/**
 * @brief 结束录音
 */
void cm_audio_recorder_stop(void);
```

5. 应用指导

本章节介绍了常见的应用示例。

SDK提供在线TTS（文本转语音）播放方式，DSLN还提供从文件系统中播放音频文件、流播放两种播放方式。

使用从文件系统中播放音频文件方式时，应在确保文件中存在播放文件的前提下调用cm_audio_play_file()接口并传入回调函数。音频播放完毕后须调用cm_audio_player_stop()接口结束播放。在音频播放期间，用户可调用cm_audio_player_pause()和cm_audio_player_resume()接口执行暂停和恢复播放操作。

在使用在线TTS播放文本时（目前仅支持百度在线TTS），须先调用cm_tts_init()接口进行TTS初始化，之后调用cm_tts_play()播放文本并等待TTS播放完毕，详细使用参见cm_tts.h头文件（因接口使用存在诸多限制，使用时必须完整阅读头文件中说明）。

使用流播放模式播放数据前应调用cm_audio_player_stream_open()接口启动播放器并且传入格式数据。之后可多次调用cm_audio_player_stream_push()接口往播放器中传入数据，播放器即会播放传入的音频数据。完成流播放后应调用cm_audio_player_stream_close()接口关闭播放器。[1]

1. 仅适用于ML307A-DSLN，不适用于ML307A-DCLN。

5.1. 播放音频示例

本节介绍播放音频功能的示例程序。

从文件系统中播放音频文件

```
else if (0 == strcmp(operation, "FILEMP3")) //播报内置的MP3文件，CM:AUDIO_PLAY:FILEMP3
{
  cm_audio_play_file(TEST_MP3_FILE_PATH, CM_AUDIO_PLAY_FORMAT_MP3, NULL,
 _cm_player_process_event, "FILEMP3");
}
else if (0 == strcmp(operation, "FILEAMR")) //播报内置的AMR文件，CM:AUDIO_PLAY:FILEAMR
{
  cm_audio_play_file(TEST_AMR_FILE_PATH, CM_AUDIO_PLAY_FORMAT_AMRNB, NULL,
 _cm_player_process_event, "FILEAMR");
}
```

```
//播放回调函数
static void _cm_player_process_event(cm_audio_play_event_e event, void *param)
{
  if (event == CM_AUDIO_PLAY_EVENT_FINISHED) //判断播放结束
  {
    /* 通知事件为中断触发，不可阻塞，不可做耗时较长的操作，例如不可使用UART打印log */
    //todo
  }
}
```

播放TTS文本内容（仅限支持TTS功能的模组）

```
static void __cm_tts_callback(cm_tts_event_e event, void *param)
{
  switch(event)
  {
    case CM_TTS_EVENT_SYNTH_DATA:
    case CM_TTS_EVENT_SYNTH_FAIL:
    case CM_TTS_EVENT_SYNTH_INTERRUPT:
    case CM_TTS_EVENT_SYNTH_FINISH:
      break;
    case CM_TTS_EVENT_PLAY_FAIL:
      cm_demo_printf("[TTS] [%s] CM_TTS_EVENT_PLAY_FAIL\n", (char *)param);
      break;
    case CM_TTS_EVENT_PLAY_INTERRUPT:
      cm_demo_printf("[TTS] [%s] CM_TTS_EVENT_PLAY_INTERRUPT\n", (char *)param);
      break;
    case CM_TTS_EVENT_PLAY_FINISH:
      break;
    default:
      break;
  }
}

/**
* UART口TTS功能调试使用示例
* CM:TTS:INIT //初始化TTS配置
```

```
* CM:TTS:PLAY  //播放TTS语音
*/
void cm_test_tts(unsigned char **cmd,int len)
{
  unsigned char operation[20] = {0};
  sprintf(operation, "%s", cmd[2]);

  if (0 == strcmp(operation, "INIT"))
  {
    cm_tts_deinit();
    cm_tts_cfg_t cfg = {0};

    cfg.speed = 5;
    cfg.volume = 9;
    cfg.encode = CM_TTS_ENCODE_TYPE_UTF8;
 //底层限制，仅可配置为CM_TTS_ENCODE_TYPE_UTF8，且该配置并不生效，实际使用百度在线TTS需使用urlencode格式编码
    cfg.rdn = CM_TTS_RDN_AUTO;  //仅可配置为CM_TTS_RDN_AUTO
    cfg.pitch = CM_TTS_PITCH_5;
    cfg.voice = CM_TTS_VOICE_0;
    cfg.channel = CM_TTS_CHANNEL_2;
 //底层限制，仅可配置为CM_TTS_CHANNEL_2，且该配置并不生效，本平台仅一个音频输出通道
    cfg.url = tts_url;  //仅支持百度在线TTS，配置其他URL不生效
    cfg.apiKey = tts_apiKey;
    cfg.secretKey = tts_secretKey;

    if (0 == cm_tts_init(&cfg))
    {
      cm_demo_printf("[TTS] init success");
    }
    else
    {
      cm_demo_printf("[TTS] init error\n");
    }
  }
  else if (0 == strcmp(operation, "PLAY"))
  {
    /* 播放TTS，注意百度平台仅支持urlencode格式 */
    if (0 ==
 cm_tts_play("%E7%99%BE%E5%BA%A6%E4%BD%A0%E5%A5%BD%EF%BC%8C%E4%BB%8A
%E5%A4%A9%E6%98%AF%E4%B8%AA%E5%A5%BD%E6%97%A5%E5%AD%90",
strlen("%E7%99%BE%E5%BA%A6%E4%BD%A0%E5%A5%BD%EF%BC%8C%E4%BB%8A
%E5%A4%A9%E6%98%AF%E4%B8%AA%E5%A5%BD%E6%97%A5%E5%AD%90"), __cm_tts_callback, "OpenCPU"))  //播放TTS语
音
    {
      cm_demo_printf("[TTS]:cm_tts_play() success\n");
    }
    else
    {
      cm_demo_printf("[TTS]:cm_tts_play() fail\n");
    }
  }
  else
  {
    cm_demo_printf("[TTS] Illegal operation\n");
  }
}
```

使用流播放模式播放PCM数据

```
else if(0 == strcmp(operation, "STARTSTREAM"))
{
  cm_audio_sample_param_t frame = {.sample_format = CM_AUDIO_SAMPLE_FORMAT_16BIT, .rate =
 CM_AUDIO_SAMPLE_RATE_8000HZ, .num_channels = CM_AUDIO_SOUND_MONO};
  cm_audio_player_stream_open(CM_AUDIO_PLAY_FORMAT_PCM, &frame);  //从pipe中播放音频（开启）
}
else if(0 == strcmp(operation, "STOPSTREAM"))
{
  cm_audio_player_stream_close();  //pipe中播放音频（关闭）
}
else if(0 == strcmp(operation, "STREAMFRAME"))
{
  cm_audio_player_stream_push(pcm_alarm, ARRAY_SIZE(pcm_alarm));  //往pipe中发送要播放的PCM数据
}
```

5.2. 录制音频示例

本节介绍录制音频功能示例程序。

录音并将录制的PCM或AMR数据存于buff中

```
else if (0 == strcmp(operation, "PCMSTART"))
{
  //录音数据存放数组的长度清零
  u32BufferFramesLength = 0;
  cm_audio_sample_param_t frame = {.sample_format = CM_AUDIO_SAMPLE_FORMAT_16BIT, .rate =
 CM_AUDIO_SAMPLE_RATE_8000HZ, .num_channels = CM_AUDIO_SOUND_MONO};
  cm_audio_recorder_start(CM_AUDIO_RECORD_FORMAT_PCM, &frame,
 (cm_audio_record_callback)_cm_record_cb, "PCM");
  cm_demo_printf("[AUDIO] record start\n");
}
else if (0 == strcmp(operation, "AMR475START"))
{
  //录音数据存放数组的长度清零
  u32BufferFramesLength = 0;
  cm_audio_recorder_start(CM_AUDIO_RECORD_FORMAT_AMRNB_475, NULL,
 (cm_audio_record_callback)_cm_record_cb, "475");
  cm_demo_printf("[AUDIO] record start\n");
}
else if (0 == strcmp(operation, "STOP"))
{
  //结束录音
  cm_audio_recorder_stop();
  cm_demo_printf("[AUDIO] record end\n");
}
```

```
static cm_audio_record_callback _cm_record_cb(cm_audio_record_event_e event, void *param)
{
  /* 通知事件为中断触发，不可阻塞，不可做耗时较长的操作，例如不可使用UART打印log */

  cm_audio_record_data_t *record_data = (cm_audio_record_data_t *)param;

  if (CM_AUDIO_RECORD_EVENT_DATA == event)
  {
    int32_t len = record_data->len;

    if (AUDIO_TEST_FRAME_BUFFER_SIZE > (len + u32BufferFramesLength)) {
memcpy (u8BufferFrames + u32BufferFramesLength, record_data->data, len);  //录音数据存储于数组中
      u32BufferFramesLength += len;  //录制的音频数据长度+length
    }
  }
}
```

6. 编程设计注意

开发设计音频功能时需注意以下几点：

- >流播放模式下每次调用cm_audio_player_stream_push()时应传入完整的帧数据，传输数据若包含不完整帧数据可能出现异常；
- >流播放模式播放和录音可同时进行；
- >语音通话过程中，不可使用播放和录音接口。

7. 附录

Table 2. 缩略语

缩写	英文全称	中文描述
TTS	Text To Speech	文本转语音
API	Application Programming Interface	应用程序编程接口