

麦克风阵列	版本: 1.9
	日期: 2020-09-14
MIC_DOC	

文档密级: A

科大讯飞(苏州)科技有限公司
iFLYTEK_SZ

麦克风阵列使用说明
V1.9

麦克风阵列	版本: 1.9
	日期: 2020-09-14
MIC_DOC	

目录

目录.....	2
1. 产品简介.....	4
1.1 功能介绍.....	4
1.2 硬件介绍.....	4
1.3 其他说明.....	5
2. Linux-SDK 介绍.....	5
2.1 概述.....	5
2.2 SDK 介绍与集成.....	6
2.2.1 SDK 介绍.....	6
2.2.2 集成方式.....	8
2.3 麦克风阵列接口说明.....	8
2.3.1 变量声明.....	8
2.3.2 类型声明.....	9
2.3.3 接口函数列表.....	10
2.3.4 接口描述.....	10
2.4 基础功能测试用例.....	19
2.5 离线命令词识别用例.....	21
2.6 在线人机交互案例.....	23
3. ROS 功能包集.....	26
3.1 功能包介绍.....	26
3.2 麦克风相关 ROS 接口描述.....	28
3.2.1 上传音频和关闭上传音频.....	28
3.2.2 音频流.....	28
3.2.3 唤醒角度(被动).....	29
3.2.4 唤醒角度(主动).....	29
3.2.5 点亮和关闭灯光.....	29
3.2.6 设置主麦克风.....	30
3.2.7 获取主麦克风.....	30
3.2.8 更改唤醒词.....	30
3.2.9 获取离线命令词识别结果.....	31
3.3 使用步骤.....	32
3.3.1 概述.....	32
3.3.2 使用说明.....	32
3.3.3 启动方式.....	33
4. 讯飞开放平台应用创建与集成.....	34
4.1 集成离线命令词识别.....	35
4.1.1 添加离线命令词识别应用.....	35

麦克风阵列	版本: 1.9
	日期: 2020-09-14
MIC_DOC	

4.1.2 集成方法.....	36
4.2 集成 AIUI.....	36
4.2.1 添加 AIUI 应用.....	36
4.2.2 集成方法.....	39
4.3 合并下载 SDK.....	40
5. 镜像烧录.....	41
6. 版本升级.....	43
7. 常见错误.....	44
8. 版本记录.....	45

麦克风阵列	版本: 1.9
	日期: 2020-09-14
MIC_DOC	

麦克风阵列开发者文档

1. 产品简介

1.1 功能介绍

麦克风阵列是由一定数目的声学传感器(一般为麦克风)组成,对声场的空间特性进行采样并处理的系统。其主要作用有声源定位,抑制背景噪声、干扰、混响、回声,信号提取与分离。声源定位是指利用麦克风阵列计算声源距离阵列的角度和距离,基于 TDOA(Time Difference Of Arrival, 到达时间差)实现对目标声源的跟踪;信号的提取与分离是指在期望方向上有效地形成一个波束,仅拾取波束内的信号,从而达到同时提取声源和抑制噪声的目的;此外利用麦克风阵列提供的信息基于深度神经网络可实现有效的混响去除,从而极大程度上提升了真实应用场景中语音交互的效果。

本麦克风阵列采用平面式分布结构,包含 6 个麦克风,可实现 360 度等效拾音,唤醒分辨率为 1 度。用户可以使用麦克风阵列获取原始和降噪音频,获取唤醒角度,主麦编号;也可以设置主麦编号,灯光点亮和关闭。

1.2 硬件介绍

麦克风阵列结构如图 1 所示,其中 0-5 表示麦克风的编号。配备的按键和接口如下:

- ☑ **USB 口**: 用于与 PC 或嵌入式设备连接;
- ☑ **ADFU**: 用于刷机,按住该键时将 USB 插入到主机上,即可进入开发者模式;
- ☑ **参考信号接口**: 可用于回声消除;
- ☑ **麦克风编号**: 已标注在麦克风上,分别对应 0-5,如图示红色数字所示;
- ☑ **LED 灯编号**: 从 0 号麦克风开始,led 顺时针编号依次为 0-11,如图示黑色数字所示;

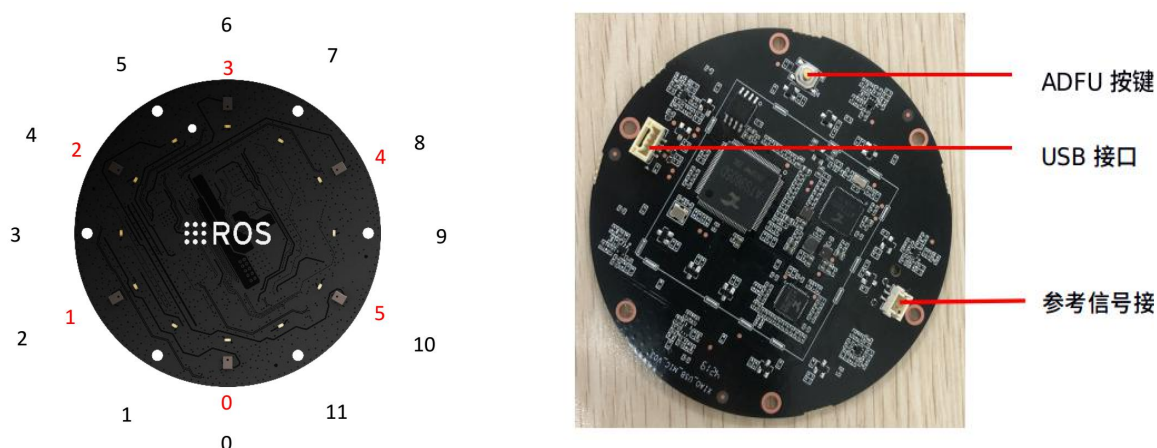


图 1 麦克风阵列硬件结构图

接线图: MIC 阵列板的 J1 通过 USB 通信线接上位机的 USB 接口为标准 USB type-A 接口,电压 5V,电流 0.5A。用于音频数据传输和控制信号传输。MIC 阵列板的 J2 通过一根 2pin 的参考信号线接上位机功放后端,即功放喇叭的 SPK+与 SPK-,用于回声消除参考信号的传输。要求输入参考信号的峰峰值不能超过 300mV。

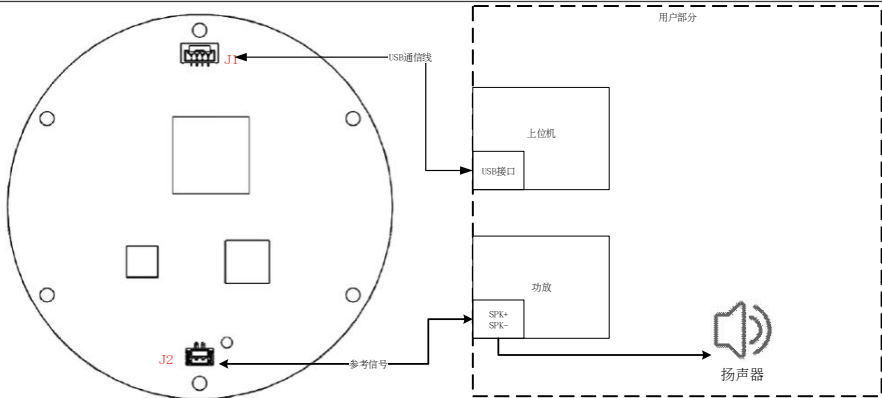


图 2 接线图示

1.3 其他说明

麦克风获取的音频类型及唤醒角度定义如下：

- ☑ **降噪音频：** 采样率 16khz， 16bit；
- ☑ **原始音频：** 采样率 16khz， 32bit，为八通道，其中 1-6 通道对应 6 个麦克风，7-8 是参考信号；
- ☑ **唤醒角度：** 从 0 号麦克风开始，顺时针依次为 0-359；
- ☑ **默认唤醒词：** "小微小微"。

麦克风的一般使用流程如下：

1. 麦克风启动并进入工作状态；
2. 设置麦克风的主麦方向，可唤醒或手动设置；
3. 获取降噪音频送入识别引擎进行识别和处理；

具体实现以及其他功能展示详见 2.3 节。

注意：在麦克风阵列中，我们一般指定一个主麦来实现指定方向声音加强，其他方向抑制的目的，且降噪音频的获取也基于该主麦方向的。若未主动设置主麦，则主麦方向随机，其获取到的降噪音频不一定基于你说话的方向，这时生成的降噪音频可能不是最优的。用户可通过唤醒或手动设置主麦的方式来设置主麦，以此来提高录音质量。

麦克风阵列上可接参考信号，一般将扬声器的声音信号作为参考信号，以实现回声消除，这是在实时对话中是必须的，否则会出现“自言自语”现象。

2. Linux-SDK 介绍

2.1 概述

本麦克风阵列板载系统为 Linux 系统，用户可以使用任一搭载 Linux 系统的主机进行通信，主机与麦克风阵列之间的通信方式为基于 USB 的自定义通信协议。基于这些协议用户可以在主机端进行诸如麦克风板开机、获取音频、设置主麦方向、获取唤醒角度等操作。为便于用户使用，本麦克风阵列配备 SDK 供使用，主要是对一些协议进行 API 的封装，以便于用户使用，除此，还提供了三个案例，供用户学习如何操作麦克风阵列以及利用麦克风阵列进行离线命令词识别和人机交互。用户也可以参照这几个案例进行其他自定义功能的编写。

sdk 功能案例	支持平台	操作系统
----------	------	------

麦克风阵列	版本: 1.9
	日期: 2020-09-14
MIC_DOC	

1. 麦克风基本功能演示案例
2. 离线命令词识别案例(仅供参考)
3. 在线语音交互案例(仅供参考)

X64:通用 PC 平台等
arm64:如 nvidia 系列等

Linux:支持
Windows:不支持
安卓:待支持

2.2 SDK 介绍与集成

拷贝 sdk 拷贝到系统任一目录中, 切换到该目录中。可以看到 sdk -vvui 以及在 ros 中使用的功能包集 vvui_ros-master, 本节仅描述 vvui, ros 功能包使用说明见第 3 部分。

若为初次使用, 且未配置过本麦克风的 udev 规则, 则执行如下指令进行配置, 若已配置过, 可忽略该步骤。注: xf_mic.rules 位于根目录。

```
$ sudo cp xf_mic.rules /etc/udev/rules.d/
```

然后执行如下指令重启 udev 服务:

```
$ sudo service udev restart
```

再次重启设备或插拔麦克风设备即可。

2.2.1 SDK 介绍

其目录结构如下:

```
├── audio
│   ├── mic_demo_vvui_deno.pcm
│   └── vvui_deno.pcm
├── bin
│   ├── mic_demo_sample
│   └── offline_command_sample
├── config
│   ├── call.bnf
│   ├── mic_offline_params.yaml
│   └── msc
├── include
│   ├── aiui
│   ├── asr_offline_record_sample.h
│   ├── cJSON.h
│   ├── cJSON_Utils.h
│   ├── hidapi.h
│   ├── libusb.h
│   ├── linuxrec.h
│   ├── msp_cmh.h
│   ├── msp_errors.h
│   ├── msp_types.h
│   ├── protocol_proc_unit.h
│   ├── qise.h
│   ├── qisr.h
│   ├── qtts.h
│   ├── queue_internal.h
│   ├── queue_simple.h
│   ├── speech_recognizer.h
│   └── user_interface.h
├── lib
└── arm32
```

麦克风阵列	版本: 1.9
	日期: 2020-09-14
MIC_DOC	

```

├── arm64
├── x64
├── x86
├── readme.md
├── sample
│   ├── aiui_sample
│   ├── mic_demo_sample
│   └── offline_command_sample
├── tmp
│   ├── all.pcm
│   ├── config.txt
│   ├── in.pcm
│   └── system.tar

```

其中:

☒ **audio**

用于存放录制的音频文件，该命名是自定义的。以 sample 文件夹中给出的 mic_demo_sample 为例，其程序中定义的降噪后音频文件的命名为 mic_demo_vvui_deno.pcm。

☒ **bin:**

用于存放可执行文件。

☒ **include**

包含案例中需要的头文件，其中 user_interface.h 为离线命令词识别案例中的用户接口。其中 char *ASR_RES_PATH 参数需要将“fo|../config/msc/res/asr/common.jet”更改为用户系统里的绝对路径，即如下样式：“fo|/home/用户名/..vvui/config/msc/res/asr/common.jet”。

☒ **lib**

包含麦克风阵列启动、给定案例中需要的动态库文件。为了兼容不同的平台，给定了兼容 Jetson 系列的 arm64 版动态库，树莓派 arm32 版以及兼容 x64 系统、x86 系统的动态库。

☒ **sample**

包含给出的三个演示案例，分别是：

1) mic_demo_sample: 麦克风基本功能测试用例。熟悉麦克风开机、录音、设置主麦方向、获取唤醒角度等基础功能演示 demo；

2) offline_command_sample: 离线命令词识别用例。通过离线命令词识别进行控制机器人运动，使用技术为讯飞离线命令词识别功能；

3) aiui_sample: 在线交互用例。通过在线 AIUI 进行实时交互，可提供查询、闲聊等能力，该功能基于讯飞 AIUI 平台。该案例仅供参考，用户可自行学习 AIUI 平台，根据自己需要改写。

注：在每一个案例中都包含了该案例实现的源码文件，以及编译用的 Makefile 文件或 CMakeLists.txt，用户可直接执行相应主机系统的.sh 文件进行编译和链接以生成可执行文件。

☒ **tmp**

麦克风阵列	版本: 1.9
	日期: 2020-09-14
MIC_DOC	

包含文件麦克风阵列板所需的资源文件，用户可忽略，在版本升级时需要用到。

☑ config:

存放配置文件，其中 msc 文件夹是离线命令词识别引擎需要的资源文件，如更换 appid，需要更换该文件夹或仅更换其中的 res/asr/下的 common.jet 文件。

call.bnf 为用户自定义的离线命令词语法文件，可根据自己实际场景进行更改。在语法文件里面用户需要根据规则来定义关键语料。离线识别的命令词是开发者自己定义，命令词最大长度为 16 个汉字，需要先构建语法，然后指定使用的语法。语法文件开发文档请参考 [bnf 语法规则编写指南](#)，建议用户在编写 bnf 前，先将准备对机器人说的命令罗列下来，然后一条一条添加到 call.bnf 中，加一条测一条，这样可避免出现语法错误。简单示例：

例如，开发一个简单的语音拨号应用，可定义如下语法：

```
.....
<commands>:(找一下|打电话给) <name>;
<name>: 张三|李四;
.....
```

该语法使识别引擎可以支持以下说法：找一下张三、打电话给张三、找一下李四、打电话给李四。凡是用户说出这个范围中的任意一句话，均可被识别系统识别。如果用户说的话不在上述范围中，识别系统可能拒绝识别。

2.2.2 集成方式

首先进入主机(搭载 linux 系统的 PC 或 ARM 板)，将 SDK 拷贝到自定义的目录中。切换到该目录中，执行如下命令解压：

```
$ tar xvf vvui.tar.gz
```

然后安装必要的声卡库：

```
$ sudo apt-get install libasound2-dev
```

安装必要的音频播放库：

```
$ sudo apt-get install sox
```

```
$ sudo apt-get install mplayer
```

完成之后，将麦克风阵列通过 usb 口插到主机上，然后在主机上打开终端，运行如下命令，查看是否检测到设备：

```
$ lsusb
```

若检测到 VID:PID 为 10d6:b003 的设备，则设备读取成功。可以运行案例以熟悉麦克风阵列的使用过程。

2.3 麦克风阵列接口说明

2.3.1 变量声明

变量	类型	含义	取值范围
is_boot	int	麦克风是否启动标志	1:启动; 0:未启动
is_reboot	int	麦克风是否重启成功	1:启动; 0:未启动
major_mic_id	int	麦克风主麦编号	0-5 表示 6 个麦克风, -1 表示不指定
led_id	int	灯光编号	0-11 表示 12 个灯光, 99 表示不指定
mic_angle	int	当前唤醒角度	0-359

麦克风阵列	版本: 1.9
	日期: 2020-09-14
MIC_DOC	

if_awake	int	是否被唤醒	1 表示被唤醒, 0 表示未被唤醒
PID	unsigned short	产品识别码	0xb003
VID	unsigned short	供应商识别码	0x10d6
mic_open_status	int	设备打开成功标志	0 成功, -1 无设备, -2 设备被占用

2.3.2 类型声明

2.3.2.1 hid_device

结构体类型, 其内容如下, 仅在设备开启时作为返回值类型使用, 可忽略, 如有兴趣可进行研究。

```
typedef struct hid_device_ hid_device;
struct hid_device_
{
    /* Handle to the actual device. */
    libusb_device_handle *device_handle;
    /* Endpoint information */
    int int_input_endpoint;
    int int_output_endpoint;
    int bulk_input_endpoint;
    int bulk_output_endpoint;
    int input_ep_max_packet_size;
    int bulk_input_ep_max_packet_size;
    int bulk_output_ep_max_packet_size;
    /* The interface number of the HID */
    int interface_n;

    /* Read thread objects */
    pthread_t thread;
    // pthread_t bulk_thread;
    int shutdown_thread;
    // int shutdown_bulk_thread;
    int cancelled;
};
```

2.3.2.2 hid_device_info

结构体类型, 用于表征设备信息, 可忽略, 其内容如下:

```
struct hid_device_info
{
    /** Platform-specific device path */
    char *path;
    /** Device Vendor ID */
    unsigned short vendor_id;
    /** Device Product ID */
    unsigned short product_id;
    /*The USB interface which this logical device*/
    int interface_number;
    /** Pointer to the next device */
};
```

麦克风阵列	版本: 1.9
	日期: 2020-09-14
MIC_DOC	

```
struct hid_device_info *next;
};
```

2.3.3 接口函数列表

与麦克风相关的函数列表如下表所示:

范围	函数名	含义
与麦克风本体相关	hid_open	打开麦克风设备
	hid_close	关闭麦克风设备
	get_system_status	获取麦克风工作状态
	get_software_version	获取软件版本
	get_protocol_version	获取协议版本
	send_resource	发送资源文件
	send_resource_info	发送资源信息
	whether_upgrade_succeed	麦克风是否启动成功
	whether_set_succeed	协议是否发送成功
	whether_set_resource_info	资源信息是否发送成功
	send_to_usb_device	向麦克风写入
	recv_from_usb_device	设备重新恢复
	business_proc_callback	全局回调函数
	err_proc	错误回调函数
与音频录制相关	start_to_record_denoised_sound	开始录制降噪音频
	finish_to_record_denoised_sound	结束录制降噪音频
	start_to_record_original_sound	开始录制原始音频
	finish_to_record_original_sound	结束录制原始音频
	get_original_sound	获取原始音频
	get_denoised_sound	获取降噪音频
与唤醒相关	get_awake_mic_id	获取唤醒主麦
	get_awake_mic_angle	获取唤醒角度
	set_awake_word	设置唤醒词
	Whether_set_awake_word	唤醒词是否设置成功
与灯光相关	get_led_based_angle	根据角度获取灯光编号
	get_led_based_mic_id	麦克风编号获取灯光编号
	set_target_led_on	设置灯光点亮和熄灭
与主麦方向相关	get_major_mic_id	获取主麦
	set_major_mic_id	设置主麦

2.3.4 接口描述

2.3.4.1 hid_open

```
HID_API_EXPORT hid_device *HID_API_CALL hid_open(void)
```

麦克风阵列	版本: 1.9
	日期: 2020-09-14
MIC_DOC	

☑ 功能

打开麦克风设备并初始化。

☑ 参数值

无

☑ 返回值

若返回为空则打开失败。否则返回结构体 hid_device 类型的变量。

☑ 说明

此接口用于打开麦克风设备，且全局仅调用一次。

☑ 示例

```
hid_device *handle;
handle = hid_open();
if (!handle)
{
    printf(">>>>>无法打开麦克风设备， 请检查设备连接\n");
    return -1;
}
printf(">>>>>成功打开麦克风设备\n");
```

☑ 参见

mic_demo_sample.cpp

2.3.4.2 hid_close

```
void HID_API_EXPORT HID_API_CALL hid_close(void)
```

☑ 功能

断开麦克风设备的连接，逆初始化。

☑ 参数值

无

☑ 返回值

无

☑ 说明

此接口可用于断开麦克风设备连接。且全局仅调用一次。

☑ 示例

```
hid_close();
```

☑ 参见

mic_demo_sample.cpp

2.3.4.3 get_system_status

```
void get_system_status()
```

☑ 功能

麦克风阵列	版本: 1.9
	日期: 2020-09-14
MIC_DOC	

获取麦克风设备是否成功开机。结果会在 `business_callback()` 中反馈。见 `businessMsg.modId` 为 0x03 且 `businessMsg.msgId` 为 0x01 项中，若查询到麦克风设备正常开机，则继续下面程序，否则下发资源包给麦克风并等待麦克风开机。

☒ **参数值**

无

☒ **返回值**

无

☒ **说明**

此接口需要运行在打开麦克风串口后，用于检测麦克风是否启动，若麦克风未启动，需要下发资源来使其启动，若不进行此步骤，程序后续的录音和唤醒功能将不会生效。

☒ **示例**

```
get_system_status();
```

☒ **参见**

`mic_demo_sample.cpp`

2.3.4.4 start_to_record_denoised_sound

```
int start_to_record_denoised_sound()
```

☒ **功能**

开启降噪音频的录制，此接口调用后，麦克风通过通信协议上传音频数据。见 `businessMsg.modId` 为 0x01 且 `businessMsg.msgId` 为 0x02 项中。通过 2.3.4.6 接口可用来将音频流保存在本地，也可将音频流直接送入语音识别或转写引擎。

☒ **参数值**

无

☒ **返回值**

若录音开启成功则返回 0，否则返回-3，其原因可能是麦克风未开启。

☒ **说明**

此接口用于降噪音频获取的开关。

☒ **示例**

```
int start_to_record_denoised_sound();
```

☒ **参见**

`mic_demo_sample.cpp`

2.3.4.5 finish_to_record_denoised_sound

```
int finish_to_record_denoised_sound()
```

☒ **功能**

关闭降噪音频的录制，此接口调用后，麦克风将不再上传音频数据。

☒ **参数值**

无

☒ **返回值**

若录音关闭成功则返回 0，否则返回-3，其原因可能是麦克风未开启。

☒ **说明**

此接口用于降噪音频获取的开关。

麦克风阵列	版本: 1.9
	日期: 2020-09-14
MIC_DOC	

☑ 示例

```
ret = finish_to_record_denoised_sound();
```

☑ 参见

mic_demo_sample.cpp

2.3.4.6 get_denoised_sound

```
Int get_denoised_sound(fileName, businessMsg.data)
```

☑ 功能

以追加地方式保存音频流到指定的文件中，文件格式为 pcm 格式。该文件为降噪后的音频，可以进一步对该文件进行识别和转写等。

☑ 参数值

fileName 为文件名，其格式为 pcm.businessMsg.data 为回调中麦克风上传的音频流。

☑ 返回值

0 表示请求送达，-3 表示异常，可能原因是麦克风暂无启动。

☑ 说明

此接口用于保存降噪音频到本地，需注意文件的路径和文件名。

☑ 示例

```
char fileName[] = DENOISE_SOUND_PATH;
get_denoised_sound(fileName, businessMsg.data);
```

☑ 参见

mic_demo_sample.cpp

2.3.4.7 start_to_record_original_sound

```
int start_to_record_original_sound()
```

☑ 功能

开启降噪音频的录制，此接口调用后，麦克风通过通信协议上传音频数据。见 businessMsg.modId 为 0x01 且 businessMsg.msgId 为 0x06 项中。

☑ 参数值

无

☑ 返回值

若录音开启成功则返回 0，否则返回-3，其原因可能是麦克风未开启。

☑ 说明

此接口用于原始音频获取的开关。

☑ 示例

```
int ret = start_to_record_original_sound();
```

☑ 参见

mic_demo_sample.cpp

2.3.4.8 finish_to_record_original_sound

```
int finish_to_record_original_sound()
```

麦克风阵列	版本: 1.9
	日期: 2020-09-14
MIC_DOC	

☑ 功能

关闭原始音频的录制，此接口调用后，麦克风将不再上传音频数据。

☑ 参数值

无

☑ 返回值

若录音开启成功则返回 0，否则返回-3，其原因可能是麦克风未开启。

☑ 说明

此接口用于原始音频获取的开关。

☑ 示例

```
int ret = finish_to_record_original_sound();
```

☑ 参见

mic_demo_sample.cpp

2.3.4.9 get_original_sound

```
int get_original_sound(char *fileName_ori, unsigned char *data)
```

☑ 功能

以追加地方式保存原始音频流到指定的文件中，文件格式为 pcm 格式。该文件为原始音频，通过 audacity 等音频软件打开后可看到其由 8 路信号组成。即 6 个麦克风对应的通道数据，以及麦克风上的两路参考信号(若未接，按静音处理)。

☑ 参数值

fileName 为文件名，其格式为 pcm.businessMsg.data 为回调中麦克风上传的音频流。

☑ 返回值

0 表示请求送达，-3 表示异常，可能原因是麦克风暂无启动。

☑ 说明

此接口用于保存原始音频到本地，需注意文件的路径和文件名。

☑ 示例

```
char fileName_ori[] = ORIGINAL_SOUND_PATH;
get_original_sound(fileName_ori, businessMsg.data);
```

☑ 参见

mic_demo_sample.cpp

2.3.4.10 get_awake_mic_id

```
int get_awake_mic_id(unsigned char *data, unsigned char *key)
```

☑ 功能

在麦克风被唤醒时获取主麦编号。在后续的音频处理中，将以该麦克风为主麦来做降噪，即增强在该麦克风范围内的音频，抑制其他麦克风范围内的音频。

☑ 参数值

data 为麦克风实时上传的音频数据，key 表征通信协议中的键，使用协议中的不同的键可以获取其对应的值。

☑ 返回值

麦克风阵列	版本: 1.9
	日期: 2020-09-14
MIC_DOC	

0, 1, 2, 3, 4, 5 中的某一值, 表示麦克风编号; -3 表示麦克风未启动。

☑ 说明

此接口用于获取唤醒时的主麦编号, 可以与唤醒角度一起用于机器人方向控制以及灯光点亮等。

☑ 示例

```
unsigned char key1[] = "beam";
int mic_id = get_awake_mic_id(businessMsg.data, key1);
```

☑ 参见

mic_demo_sample.cpp

2.3.4.11 get_awake_mic_angle

```
int get_awake_mic_angle(unsigned char *data, unsigned char *key)
```

☑ 功能

在麦克风被唤醒时获取唤醒角度。

☑ 参数值

data 为麦克风实时上传的音频数据, key 表征通信协议中的键, 使用协议中的不同的键可以获取其对应的值。

☑ 返回值

0-359 中的某一整数值, 表示唤醒角度; -3 表示麦克风未启动。

☑ 说明

此接口用于获取唤醒时的唤醒角度, 可精确到 1 度, 可以与唤醒主麦方向一起用于机器人方向控制以及灯光点亮等。

☑ 示例

```
unsigned char key2[] = "angle";
int mic_angle = get_awake_mic_angle(businessMsg.data, key2);
```

☑ 参见

mic_demo_sample.cpp

2.3.4.12 set_awake_word

```
int set_awake_word(char *awake_words)
```

☑ 功能

给麦克风设置自定义的词。

☑ 参数值

字符数组形式的唤醒词。

☑ 返回值

0 表示命令执行成功, -3 表示异常, 如麦克风未启动。

☑ 说明

唤醒词支持 4-6 个汉字, 或不超过 2 个单词的英文词组, 在设置唤醒词前请参照[唤醒词设置](#)规则并进行评分, 值得注意的是低质量的唤醒词会影响唤醒效果。

☑ 示例

```
char word[30] = "小明小明; nCM:600";
```

麦克风阵列	版本: 1.9
	日期: 2020-09-14
MIC_DOC	

set_away_word(word);

☑ 参见

mic_demo_sample.cpp

2.3.4.13 whether_set_away_word

int whether_set_away_word(unsigned char *data, unsigned char *key)

☑ 功能

查询唤醒词是否设置成功, 设置唤醒词后, 其设置结果将在回调函数体现。见 businessMsg.modId 为 0x02 且 businessMsg.msgId 为 0x08 项中。

☑ 参数值

data 为麦克风实时上传的音频数据, key 表征通信协议中的键, 使用协议中的不同的键可以获取其对应的值。

☑ 返回值

0 为设置成功; -3 表示麦克风未启动; -2 为设置失败。

☑ 说明

查询自定义唤醒词是否设置成功。

☑ 示例

```
unsigned char key1[] = "errstring";
int result = whether_set_away_word(businessMsg.data, key1);
if (result == 0)
{
    printf("唤醒词设置成功\n");
}
else if (result == -2)
{
    printf("唤醒词设置失败\n");
}
```

☑ 参见

mic_demo_sample.cpp

2.3.4.14 get_led_based_angle

int get_led_based_angle(int mic_angle)

☑ 功能

根据唤醒角度获取哪个灯需要被点亮。

☑ 参数值

mic_angle 为唤醒角度, 取值为 0-359。

☑ 返回值

0-11 之间的整数, 表示灯的编号; -3 表示麦克风未启动。

☑ 说明

此接口一般用在唤醒后, 在获取唤醒角度后来点亮相应的灯光。

☑ 示例

麦克风阵列	版本: 1.9
	日期: 2020-09-14
MIC_DOC	

```
int led_id = get_led_based_angle(mic_angle);
```

☒ **参见**

mic_demo_sample.cpp

2.3.4.15 get_led_based_id

```
int get_led_based_mic_id(int mic_id)
```

☒ **功能**

在人为主动设置主麦时，用于获取该点亮的灯光编号。

☒ **参数值**

mic_id 为唤醒角度，取值为 0-5。

☒ **返回值**

0-11 之间的整数，表示灯的编号；-3 表示麦克风未启动。

☒ **说明**

此接口一般用在主动设置主麦方向，在获取唤醒角度后来点亮相应的灯光。

☒ **示例**

```
int led_id = get_led_based_id(mic_id);
```

☒ **参见**

mic_demo_sample.cpp

2.3.4.16 set_major_mic_id

```
int set_major_mic_id(int id)
```

☒ **功能**

设置主麦方向。

☒ **参数值**

id 为当前主麦 id。

☒ **返回值**

0 表示 id 符合要求，-3 表示麦克风未开启。

☒ **说明**

此接口一般用在唤醒后，设置主麦方向，也可在需要的时候用本函数进行设置。

☒ **示例**

```
ret = set_major_mic_id(mic_id);
```

☒ **参见**

mic_demo_sample.cpp

2.3.4.17 set_target_led_on

```
int set_target_led_on(int led_id)
```

☒ **功能**

点亮指定编号的灯光。

☒ **参数值**

麦克风阵列	版本: 1.9
	日期: 2020-09-14
MIC_DOC	

led_id 为灯光编号, 0-11 或 99。

返回

0 表示灯光 id 符合要求, -3 表示灯光未开启。

说明

此接口一般用在唤醒后, 在获取唤醒角度后来点亮相应的灯光。特殊地, 当 led_id 为 99 时, 可以关闭灯光。

示例

```
set_target_led_on(led_id);
```

参见

mic_demo_sample.cpp

2.3.4.18 get_major_mic_id

```
void get_major_mic_id()
```

功能

获取当前主麦编号。其结果将在回调函数中返回。见 businessMsg.modId 为 0x01 且 businessMsg.msgId 为 0x07 项中。

参数值

无。

返回值

0 表示请求发送成功, -3 表示异常, 如麦克风未开启。

说明

该接口调用后, 会在回调函数中进行返回。其返回值为 0-5 的值或-1。特别地, -1 表示未设置主麦, 常出现在唤醒或设置主麦方向前。

示例

```
get_major_mic_id();
```

参见

mic_demo_sample.cpp

2.3.4.19 get_software_version

```
void get_software_version()
```

功能

获取当前麦克风软件版本号。

参数值

无。

返回值

版本号, 如"R-2.1.1"。

说明

无

示例

麦克风阵列	版本: 1.9
	日期: 2020-09-14
MIC_DOC	

```
get_software_version();
```

☑ 参见

mic_demo_sample.cpp

2.3.4.20 get_protocol_version

```
int get_protocol_version(businessMsg.data, protocol_version)
```

☑ 功能

获取当前麦克风协议版本号。

☑ 参数值

回调返回值以及版本号变量。

☑ 返回值

0 表示获取成功, -1 表示获取失败。

☑ 说明

无。

☑ 示例

```
char protocol_version[40];
int ret = get_protocol_version(businessMsg.data, protocol_version);
```

☑ 参见

mic_demo_sample.cpp

2.4 基础功能测试用例

本案例用于帮助用户熟悉麦克风阵列的操作过程、通信协议及 API。使用过程如下, 注意你只需要根据自己的操作系统来执行命令:

☑ 动态库配置

首先切换到 lib 目录下:

```
$ cd ./vvui/lib
```

若主机是 arm64 操作系统, 如英伟达 jetson 系列, 则执行如下命令:

```
$ cd arm64 && sudo cp lib* /usr/lib
```

若主机是 arm32 操作系统, 如树莓派, 则执行如下命令:

```
$ cd arm32 && sudo cp lib* /usr/lib
```

若主机是 X64 操作系统, 则执行如下命令:

```
$ cd x64 && sudo cp lib* /usr/lib
```

☑ 生成可执行文件

切换到 mic_demo_sample 所在的目录:

```
$ cd vvui/sample/mic_demo_sample
```

依照自己的运行平台进行编译脚本文件的选择, 选择以下其一即可:

若为 arm64 平台, 如英伟达 jetson 系列, 则运行如下指令可生成可执行文件:

```
$ sh ./arm64_make.sh
```

若为 arm32 平台, 如树莓派, 则运行如下指令可生成可执行文件:

```
$ sh ./arm32_make.sh
```

麦克风阵列	版本: 1.9
	日期: 2020-09-14
MIC_DOC	

若平台为 x64 位操作系统，则运行如下指令：

```
$ sh ./x64bit_make.sh
```

若编译正常，则可在 bin 目录下生成可执行文件 mic_demo_sample。

测试

切换到 bin 目录下，执行生成的可执行文件。

```
$ cd ../../bin
```

```
$ sudo ./mic_demo_sample
```

注：若已设置过 udev 规则，sudo 也可省略。

如图所示，运行完成后，会提示 1-11 的命令，其中：

```
inano@xiao-sn-1234567890:~/vvui/bin$ ./mic_demo_sample
>>>>成功打开麦克风设备
demo示例为输入命令，调用对应的函数，可以简单熟悉麦克风相关的基本功能，如：
1命令，获取麦克风的工作状态，若麦克风未启动，则麦克风会自动启动至工作状态
2命令，开启降噪音频录音功能，执行该命令前请务必保持设备为唤醒状态
3命令，停止降噪音频录音功能，在指定的路径中保存音频文件
4命令，开启原始音频录音功能，执行该命令前请务必保持设备为唤醒状态
5命令，停止原始音频录音功能，在指定的路径保存音频文件
6命令，同时开启降噪音频和原始音频录音功能，执行该命令前请务必保持设备为唤醒状态
7命令，同时停止降噪音频和原始音频录音功能，在指定的路径保存音频文件
8命令，设置主麦克风id
9命令，获取主麦克风id
10命令，点亮和关闭灯光
11命令，设置自定义唤醒词
: send result to client,stop the offline_recognise mode
>>>>请输入指令 32755485]: close the offline recognise mode ...
:■
```

图 3 mic_demo_sample 测试过程

1) 1 命令是麦克风阵列开机操作。在将麦克风阵列插入到设备上时，麦克风阵列软件层并未开机和运行。此时，麦克风有一个灯光为常亮状态表示硬件已供电。当执行本执行时，系统会首先检测麦克风是否已开始工作，若已正常工作，将反馈“麦克风正常工作”；否则，系统会自动与麦克风进行通信来让麦克风进入工作状态，此时反馈“麦克风正在启动，”。启动时间大约在 10s。之后常亮灯灭掉。若此时且再次输入”1”命令，终端描述为“麦克风正常工作，”。在此之后，若不出现拔插麦克风操作，输入”1”命令时终端描述均为“麦克风正常工作，”。

2) 2 命令是录制麦克风降噪音频的操作，在此之前请务必用唤醒词进行唤醒，否则是无法获取音频的，唤醒成功后，按下 2，则开始进行录音，且在 audio 文件夹中会看到录制的降噪后音频 mic_demo_vvui_deno.pcm 你可修改程序中保存文件的路径和名称。

3) 3 命令是停止录制麦克风降噪音频，按下 3，则停止录制降噪后音频。

4) 4 命令是录制麦克风原始音频的操作，若在此之前麦克风阵列未被唤醒过，仍然需要再唤醒后进行操作，按下 4，则开始正常录音，且在 audio 文件夹中会看到录制的降噪后音频 mic_demo_vvui_ori.pcm。

5) 5 命令是停止录制麦克风原始音频，按下 5 即停止。

6) 6 命令是同时录制原始和降噪音频。

7) 7 命令是停止同时录制原始和降噪音频。

8) 8 命令设置麦克风的主麦，可设置 0-5 中的任意一个为主麦。

9) 9 命令获取主麦克风 id，在终端会打印出来。打印结果为以 0-5 表示的麦克风 id，或者-1，表示未设置。

10)10 命令点亮和关闭灯光，灯光 id 为 0-11。输入 0 表示点亮 1 号灯光，特别地 99 表示关闭灯光。

11)11 命令设置自定义唤醒词，唤醒词设置要在 4 — 6 个汉字为佳，唤醒词的质量影响唤醒率，可在讯飞开放平台上对唤醒词进行评分。

麦克风阵列	版本: 1.9
	日期: 2020-09-14
MIC_DOC	

特殊地，若要退出本命令，可输入”0”后按回车来退出。在退出时建议手动设置停止录音，避免出现在录音过程中强制中断的现象，因为强制中断无法中断麦克风的数据传输。

2.5 离线命令词识别用例

本案例(offline_command_sample)为离线命令词识别用例，用户可使用自定义的离线命令词的关键字进行离线下的命令下发。使用过程如下：

☑ 动态库配置

注：此步骤同案例 mic_demo_sample 中的配置过程，若已配置过动态库，可直接跳过本动态库配置过程。首先切换到 lib 目录下：

```
$ cd ./vvui/lib
```

若主机是 arm64 操作系统，如英伟达 jetson 系列，则执行如下命令：

```
$ cd arm64 && sudo cp lib* /usr/lib
```

若主机是 arm32 操作系统，如树莓派，则执行如下命令：

```
$ cd arm32 && sudo cp lib* /usr/lib
```

若主机是 X64 操作系统，则执行如下命令：

```
$ cd x64 && sudo cp lib* /usr/lib
```

☑ 参数配置

在执行生成的可执行文件之前，需要检查一下离线资源路径是否正确设置，ASR_RES_PATH 要设置为用户系统里的绝对路径，sdk 提供了一个用户接口 user_interface.h 用来设置调试参数。包括语法识别资源路径、语法路径、音频文件保存路径、置信度和一次命令时长等参数。

■ **ASR_RES_PATH**：离线命令词识别引擎资源路径，为 config/msc/res/asr/中的 common.jet。你需要修改为你系统下的**绝对路径**。否则在程序执行时会出现”10102”等错误。

■ **GRM_BUILD_PATH**：离线命令词识别引擎在构建离线识别网络时，生成的数据可保存的路径。

■ **GRM_FILE**：这个为用户自定义的语法文件存放的地址。一般在 config 中。

■ **DENOISE_SOUND_PATH**：降噪音频保存的地方，一般保存在 audio 文件夹中。

■ **SYSTEM_PATH**：与麦克风通信相关的资源包的路径。

■ **SYSTEM_CONFIG_PATH**：上述资源包的配置文件的路径。

除路径外，还有几个与离线命令词使用相关的参数，对其进行简要介绍：

■ **APPID**：用户在科大讯飞开放平台上注册账号，并创建应用后会得到一个 appid，该 appid 标识一个应用，在该应用中，用户可添加离线命令词识别，语音听写，语音合成等引擎。在使用离线命令词识别时，appid 与离线命令词识别引擎绑定。若在使用离线命令词识别时出现“11210”或“11212”等错误时，可能原因 appid 与引擎不匹配或者 appid 过期。

■ **LEX_NAME**：表示语法规则的名字，离线命令词需要用户创建 bnf 语法文件，LEX_NAME 实际为 bnf 中的规则槽名称。

■ **confidence**：为识别的置信度阈值，若离线识别引擎识别到的关键词的置信度大于该阈值则认为其识别的命令词是可信的，否则是不可信的，用户可根据实际测试效果，来调节该参数，如多次出现误识别，则可调大该参数等等。

麦克风阵列	版本: 1.9
	日期: 2020-09-14
MIC_DOC	

■ **time_per_order**: 本案例可实现连续离线命令词交互, 每唤醒一次, 可交互 `awake_count` 次, 每交互 `awake_count` 次, 麦克风即进入休眠模式, 需要二次唤醒才可进行交互。用户可根据自己实际需求, 增大或减小该值。

■ **time_per_order**: 每次交互的时候, 麦克风允许有 `time_per_order` 时间(单位为秒)来说出命令词, 用户可根据自己实际需求, 增大或减小该值。

☑ 生成可执行文件

切换到 `offline_command_sample` 所在的目录:

```
$ cd vvui/sample/offline_command_sample
```

依照自己的运行平台进行编译脚本文件的选择:

若为 arm64 平台, 如英伟达 jetson 系列, 则运行如下指令可生成可执行文件:

```
$ sh ./arm64_make.sh
```

若为 arm32 平台, 如树莓派, 则运行如下指令可生成可执行文件:

```
$ sh ./arm32_make.sh
```

若平台为 x64 位操作系统, 则运行如下指令:

```
$ sh ./x64bit_make.sh
```

若编译正常, 则可在 `bin` 目录下生成可执行文件 `offline_command_sample`。

☑ 测试

切换到 `bin` 目录下, 可以执行生成的可执行文件。

```
$ cd ../../bin
```

```
$ sudo ./offline_command_sample
```

注: 若已设置过 `udev` 规则, `sudo` 也可省略。

测试过程如图所示, 与 `mic_demo_sample` 不同的是, 本案例不再是分步操作, 而是连续地自动操作, 即首先检测麦克风阵列是否开机, 若未开机就等待其开机, 开机完成后则进入待唤醒状态。此时会提醒用户进行唤醒, 若检测到唤醒成功则自动开启录音, 此时说出离线命令词即可进行识别。

```
>>>>麦克风系统正常工作
>>>>开机成功!
>>>>待唤醒, 请用唤醒词进行唤醒!

>>>>第4个麦克风被唤醒

>>>>唤醒角度为:210

>>>>已点亮7灯

>>>>设置主麦克风成功

>>>>设置灯光成功
>>>>唤醒成功, 已开启录音!

>>>>设置主麦克风成功
>>>>开始录制降噪音频

>>>>设置灯光成功
>>>>停止录制降噪音频
>>>>正在识别
>>>>是否识别成功: [ 是 ]
>>>>全部返回结果: [ <?xml version='1.0' encoding='utf-8' standalone='yes' ?><nlp>
<version>1.1</version>
<rawtext>向前走</rawtext>
<confidence>76</confidence>
<engine>local</engine>
<result>
<focus>want|direction|do</focus>
<confidence>46|100|82</confidence>
<object>
<want id="65535">向</want>
<direction id="10001">前</direction>
<do id="10001">走</do>
</object>
</result>
</nlp>
]
>>>>关键字的置信度: [ 76 ]
>>>>关键字识别结果: [ 向前走 ]
```

图4 离线命令词识别运行测试

麦克风阵列	版本: 1.9
	日期: 2020-09-14
MIC_DOC	

注：本 sdk 使用的离线命令词是根据自定义的应用场景所创建的，在本例中是为控制机器人运动，其命令词包含：【你】【往、向】【前、后、左、右】【走、转】，用户说出这四组关键字中任一组合均可识别，如“往前走”、“向后走”、“向左转”、“左转”等。若要对关键字进行修改，可修改 config 目录中的 call.bnf 文件，其格式如下：

```

1  #BNF+IAT 1.0 UTF-8;
2  !grammar move;
3  !slot <want>;
4  !slot <direction>;
5  !slot <do>;
6  !start <movestart>;
7
8  <callstart>:[<want>]<dowhat>;
9  <want>:向|往|你往|你向;
10 <dowhat>:<direction><do>;
11 <direction>:左!id(10001)|右!id(10001)|前!id(10001)|后!id(10001);
12 <do>:走!id(20002)|移动!id(20002)|转!id(20002)|动!id(20002)|退!id(20002);

```

图 5 离线命令词语法

☑ 进阶

本例中离线命令词识别的功能基于动态库 liboffline_record_lib.so，该动态库存在于上述的 lib 目录下，你也可以根据自己需要修改这个动态库的实现，实现方法是：

修改路径 vvui/sample/offline_command_sample/src 中的代码，并根据自己平台类型，修改 CmakeList.txt 中的如下字段，如为 x64 则修改下面路径为 x64：

```

link_directories(
  ../../lib/arm64
  ./
)

```

修改完毕后，我们可以重新生成动态库 liboffline_record_lib.so，方法如下：

```

$ cd ./vvui/sample/offline_command_sample
$ sh ./offline_lib_make.sh

```

该过程生成的动态库位于 vvui/lib/指定平台/目录下，你需要重新进行动态库配置来使其生效。

离线命令词识别本例中给出的离线命令词是需要输入录制音频文件后进行识别，即给定固定的时间进行录音，然后对录音文件进行命令词识别。用户可参照本例和讯飞开放平台中离线命令词识别 SDK 进行更改，以实现不保存录音文件，直接对用户语音进行处理，即根据 VAD 来判断语音的开头和结尾。

2.6 在线人机交互案例

在 vvui 下的 sample 中，使用如下命令切换到 aiui_sample 所在的目录：

```
$ cd aiui_sample
```

☑ 修改 CMakeLists.txt 文件

该项主要是根据主机系统来修改动态库路径，若 x64 系统，则修改 16-20 行内容如下：

```

'''
link_directories(
  ../../lib/x64
)
'''

```

若为 arm64 系统，则修改 16-20 行内容如下：

麦克风阵列	版本: 1.9
	日期: 2020-09-14
MIC_DOC	

```
...
link_directories(
../lib/arm64
)
...
```

☑ 生成可执行文件

执行如下命令，生成可执行文件：

```
$ sh ./aiui_make.sh
```

编译完成后，可在 `vvui/bin` 目录中生成可执行文件 `aiui_sample`。

☑ 测试

执行如下命令即可体验 `aiui` 交互：

```
$ cd ../bin
$ sudo ./aiui_sample
```

使用唤醒词“小微小微”即可对麦克风板进行唤醒，唤醒后即进入交互状态，本案例默认提供的交互方式是 `continues` 方式，即一次唤醒多次交互的方式，用户在该方式下可以体验到中途打断的功能，在它回答问题中，可以接着说出下一个问题，这时会自动回答下一个问题。值得注意的是，服务端保存用户交互历史的时间最长为 120s，即如果用户说“明天天气怎么样”，接下来说“后天呢？”，会默认回答后天天气状况，多轮对话最长时间为 120s，如下图所示：

```
>>>>检测到对话开始
>>>>问题是:"明天天气怎么样"

>>>>答案是:"合肥市明天全天中雨转雷阵雨，出门记得带伞，气温6℃~ 22℃ 有南风转北风3-4级，温度适宜。"

>>>>检测到本轮对话结束
>>>>检测到对话开始
>>>>问题是:"后天呢"

>>>>答案是:"后天合肥市全天阴转多云，气温6℃~ 12℃ 空气质量优，有东北风3-4级，温度适宜。"
```

图 6 在线语音交互过程

值得注意的是，`AIUI` 引擎在长时间无有效交互时会进入休眠模式，此时需要重新唤醒进行交互。此时间默认为 180s，可在 `aiui.cfg` 中进行修改。

在正常交互过程中事件的状态会标识为“`tts`”，即文字到语音的转化。当用户使用技能命令词“你去休息吧”，事件的状态标识为“`nlp`”，此时麦克风会进入休眠模式，不会进行交互。若要重新进入交互模式需要再次使用“小微小微”进行唤醒。示例如下：

```
>>>>当前事件状态:nlp
>>>>问题是:"你去休息吧"

>>>>答案是:null

>>>>麦克风准备进入休眠模式，将休眠
EVENT_STATE:READY
>>>>当前事件状态:tts
>>>>当前事件状态:tts
>>>>当前事件状态:tts
```

图 7 使用技能

在执行本案例的过程中，你可选择将每次交互过程中产生的音频文件都保存下来，为了避免文件过大，该样例中已设置在每次唤醒时会自动清除上次交互产生的文件。

若要修改唤醒词，可以对机器人说“更改/修改唤醒词”，然后按照机器人提示进行设置。

特别地，可以按“`q`”键和回车来退出本程序。

☑ 常见错误

麦克风阵列	版本: 1.9
	日期: 2020-09-14
MIC_DOC	

在使用过程中,可能会出现不能交互且报错代码为 11201 的情况,其原因是本案例绑定的 appid 单日的交互次数已达上限,用户可以替换为自己已开通 aiui 引擎的 appid。

修改方法是修改 aiui_sample/src/AIUITester.cpp 文件中的 AIUITester::createAgent 函数里面的字符串 appid 以及 key 为用户自己的。在开放平台上开通常用的语料和技能进行使用,在后续使用中若再次遇到报错代码为 11201 的情况,用户可前往控制台检查对应 appid 次数是否超过限制次数,超过可在官网产品页面领取免费包或者购买套餐包,未超过可提交技术工单。离线服务接口报错其他常见错误可见 <https://www.xfyun.cn/document/error-code/>。

麦克风阵列	版本: 1.9
	日期: 2020-09-14
MIC_DOC	

3.ROS 功能包集

本功能包涵盖交互相关的多个功能包，包括集成了唤醒角度获取，音频获取，离线命令词识别的麦克风功能包。目前支持的功能如下表所示，所有的功能都以接口的形式来给出，且均以话题或服务形式。用户可直接根据 3.2 和 3.3 中的协议来发送请求即可获取结果。

功能包集提供的功能	支持平台	操作系统
麦克风录音开与关 降噪或原始音频获取 唤醒角度获取 主麦克风获取 设置主麦克风 灯光点亮与关闭 自定义唤醒词设置 获取离线命令词识别结果	X64:通用 PC 平台等 Arm64:如 nvidia 系列等	Linux:支持 Windows:待支持 安卓:待支持

3.1 功能包介绍

该功能包作为与麦克风相关的所有接口的服务端，提供包括唤醒角度，降噪音频，灯光控制，离线命令词识别等功能。其提供与麦克风相关的多个接口，见 3.2 所示。其文件目录如下：

```
├── audio
├── CMakeLists.txt
├── config
├── include
├── launch
├── lib
├── msg
├── package.xml
├── src
├── srv
└── tmp
```

☒ **include**

存放所有头文件，在这其中，用户只需要关注 user_interface.h 文件即可，该头文件为用户接口，用户可对其中的参数进行修改来进行调试或适配。

```
#define whether_print_log 0 //是否打印调试结果，默认不打印。
char awake_words[30] = "小微小微"; //使用的唤醒词,可使用接口进行修改。
int time_per_order = 3; //一次离线命令词识别命令时长。
int PCM_MSG_LEN = 1024; //在录音时会发布音频流，大小为 2048B，麦克风获取到的降噪音频采样率为 16khz，16bit，故 1s 获取 256kbit，即 32K，若每次发送 1024B，则帧率约为 31fps。可根据用户帧率需求，来调节该参数。
bool save_pcm_local = true; //保存音频到本地，在离线命令词识别时用到。
int max_pcm_size = 10240000; //录音文件最大可保存大小为 10M，超过 10M 后自动删除，以节省空间。
```

麦克风阵列	版本: 1.9
	日期: 2020-09-14
MIC_DOC	

■ 除此外，在用户头文件中还有一些路径，这些路径将在启动文件 launch 后变为绝对路径，方便用户在不同路径下启动。故不建议修改，若要修改，可观察路径规则来修改。

☒ lib

存放所有动态库文件，用户需要根据自己预使用的 linux 平台进行动态库的选择。现支持 arm64(支持英伟达 TX1 TX2 NANO 等 JETSON 系列)以及 x64(通用 PC)。

☒ launch

包含麦克风接口功能包启动文件。

☒ src

包含麦克风接口源程序。

☒ audio

存放录音文件。

☒ config

存放配置文件，其中 msc 文件夹是离线命令词识别引擎需要的资源文件，如更换 appid，需要更换该文件夹或仅更换其中的 res/asr/下的 common.jet 文件。

call.bnf 为用户自定义的离线命令词语法文件，在里面用户需要根据规则来定义关键语料。离线识别的命令词是开发者自己定义，命令词最大长度为 16 个汉字，需要先构建语法，然后指定使用的语法。

语法文件开发文档请参考 [bnf 语法规则编写指南](#)，简单示例：

例如，开发一个简单的语音拨号应用，可定义如下语法：

```
.....
<commands>:(找一下|打电话给) <name>;
<name>: 张三|李四;
.....
```

该语法使识别引擎可以支持以下说法：找一下张三 、打电话给张三 、找一下李四 、打电话给李四。凡是用户说出这个范围中的任意一句话，均可以被识别系统识别。如果用户说的话不在上述范围中，识别系统可能拒绝识别。

mic_offline_params.yaml 文件为配置参数文件，用于修改一些接口中的默认值。便于调试。

☒ msg

包含自定义的话题类型。

☒ srv

包含自定义的服务类型。

麦克风阵列	版本: 1.9
	日期: 2020-09-14
MIC_DOC	

☑ tmp

包含麦克风通信相关的资源文件。仅在版本升级时更换。

3.2 麦克风相关 ROS 接口描述

3.2.1 上传音频和关闭上传音频

□ 类 型: 服务

□ 服务名称: /xf_asr_offline_node/start_record_srv

□ 服务类型: xf_mic_asr_offline::Start_Record_srv

```
'''
int8 whether_start
---
string result
string fail_reason
'''
```

□ 内 容: 1 为开启录音, 录音开启即开始上传, 0 为关闭音频, 关闭音频则停止上传。

□ 描 述: 在开启录音后, 不要忘记关闭录音。录音结果将在话题/mic/pcm/deno 中发布。在获取音频流的时候, 系统默认保存音频文件到本地。

```
'''
bool save_pcm_local = true; //保存音频到本地。在离线命令词识别服务中需要用到。
'''
```

□ 调用示例:

```
//开启录音
$ rosservice call xf_asr_offline_node/start_record_srv 1

//关闭录音
$ rosservice call xf_asr_offline_node/start_record_srv 0
```

3.2.2 音频流

□ 类 型: 话题(发布)

□ 话题名称: /mic/pcm/deno

□ 话题类型: xf_mic_asr_offline/Pcm_msg

```
'''
Int32 length
char[] pcm_buf
'''
```

□ 内 容: 话题内容包括两部分, length 表示 pcm_buf 中的长度, 即有效音频的大小, pcm_buf 存放的是指定大小(见 user_interface.h 中 PCM_MSG_LEN 参数)的音频字节。

□ 调用示例:

```
//查看音频流
$ rostopic echo /mic/pcm/deno
```

麦克风阵列	版本: 1.9
	日期: 2020-09-14
MIC_DOC	

3.2.3 唤醒角度(被动)

- 类 型: 话题(发布)
- 话题名称: /mic/awake/angle
- 话题类型: std_msgs/Int32
- 内 容: 0-360 之间的整数, 分辨率为 1, 表示唤醒角度。
- 描 述: 只有在唤醒的时候才会有发布唤醒角度值, 用户是被动接收, 每唤醒一次, 返回一次唤醒角度。
- 调用示例:

```
//查看唤醒角度
$ rostopic echo /mic/awake/angle
```

3.2.4 唤醒角度(主动)

- 类 型: 服务
- 服务名称: xf_mic_asr_offline/Get_Awake_Angle_srv
- 服务类型: xf_mic_asr_offline::Get_Awake_Angle_srv

```
'''
int8 get_awesome_angle #1, 0
---
string result
int32 awake_angle
string fail_reason
'''
```

- 内 容: 0-360 之间的整数, 分辨率为 1, 表示唤醒角度。
- 描 述: 可在开机唤醒后的任意时刻查询。
- 调用示例:

```
//查看唤醒角度
$ rosservice call xf_asr_offline_node/get_awesome_angle_srv 1
```

3.2.5 点亮和关闭灯光

- 类 型: 服务
- 服务名称: /xf_mic_asr_offline/set_target_led_on_srv
- 服务类型: xf_mic_asr_offline::Set_Led_On_srv

```
'''
int8 led_id
---
string result
string fail_reason
'''
```

- 内 容: 0-11 之间的整数, 若输入值是 1-11, 则分别代表 0-11 号灯。特殊地, 99 表示关闭灯光。
- 描 述: 若设置成功, result 为“ok”, 其他情况均反馈为“fail”; 若反馈为“fail”, 则 fail_reason 项不再为空, 且会返回错误类型, 原因可能为输入无效的灯光编号-“incorrect_mic_id_error”, 麦克风未开机-“mic_did_not_open_error”。
- 调用示例:

麦克风阵列	版本: 1.9
	日期: 2020-09-14
MIC_DOC	

```
//调用灯光调节接口，点亮 4 号灯
$ rosservice call xf_asr_offline_node/set_target_led_on_srv 4
//调用灯光调节接口，关闭灯光
$ rosservice call xf_asr_offline_node/set_target_led_on_srv 99
```

3.2.6 设置主麦克风

□ 类 型: 服务

□ 服务名称: /xf_asr_offline_node/set_major_mic_srv

□ 服务类型: xf_mic_asr_offline::Set_Major_Mic_srv

```
'''
int8 mic_id #取值为 0-5
---
string result
string fail_reason
'''
```

□ 内 容: 输入为 0-5 时，对应编号为 0-5 的麦克风。

□ 描 述: 其中，若设置成功，result 为“ok”，其他情况均反馈为“fail”；若反馈为“fail”，则 fail_reason 项不再为空，且会返回错误类型，原因可能为输入无效的麦克风编号-“incorrect_mic_id_error”，麦克风未开机-“mic_did_not_open_error”。

□ 调用示例:

```
//设置 3 号麦克风为主麦克风
$ rosservice call /xf_asr_offline_node/set_major_mic_srv 3
```

3.2.7 获取主麦克风

□ 类 型: 服务

□ 话题名称: /xf_asr_offline_node/get_major_mic_srv

□ 话题类型: xf_mic_asr_offline::Get_Major_Mic_srv

```
'''
int8 get_major_id #1, 0
---
string result
int8 mic_id
string fail_reason
'''
```

□ 内 容: 输入为 1 时，表示获取主麦克风，其他参数无效。

□ 描 述: 若检测到麦克风编号，则会返回 0-5，返回“ok”，若在有限时间内未能获取到麦克风编号，则会返回“false”，且错误原因为“timeout_error”。其他错误类型可能是麦克风未开机-“mic_did_not_open_error”。

□ 调用示例:

```
//获取主麦克风
$ rosservice call /xf_asr_offline_node/get_major_mic_srv 1
```

3.2.8 更改唤醒词

□ 类 型: 服务

麦克风阵列	版本: 1.9
	日期: 2020-09-14
MIC_DOC	

□ **话题名称:** /xf_asr_offline_node/set_awake_word_srv

□ **话题类型:** xf_mic_asr_offline::Set_Awake_Word_srv

```
'''
string awake_word
---
string result
string fail_reason
'''
```

□ **内 容:** 输入字符串形式唤醒词, 要 4-6 个汉字, 其他参数无效。

□ **描 述:** 若设置成功, 会返回 ok; 若在有限时间内未能设置成功, 则会返回“false”, 且错误原因为“timeout_error”; 若唤醒词设置不符合要求, 则会返回“invalid_awake_word”。其他错误类型可能是麦克风未开机-“mic_did_not_open_error”。

□ **调用示例:**

//更改唤醒词

\$ rosservice call /xf_asr_offline_node/set_awake_word_srv “齐天大圣”

3.2.9 获取离线命令词识别结果

□ **类 型:** 服务

□ **服务名称:** /xf_asr_offline_node/get_offline_recognise_result_srv

□ **服务类型:** xf_mic_asr_offline::Get_Offline_Result_srv

```
'''
int8 offline_recognise_start #1 表示开启
int8 confidence #指定置信度
Int8 time_per_order #指定单次离线命令时麦克风接收语音的时长(秒)
---
string result
string fail_reason
string text #返回文字结果
'''
```

□ **内 容:** 其中, 若离线识别成功检测到文字内容, result 为“ok”, 其他情况均反馈为“fail”;

若反馈为“fail”, 则 fail_reason 项不再为空, 且会返回错误类型, 已知类型包括:

- 1) “low_confidence error or 11212_license_expired_error”, 表示识别到的文字置信度小于指定的置信度阈值, 或者离线识别功能授权过期; 可参考 4.1 节进行修改;
- 2) “no_valid_sound error”, 表示未检测到有效的声音;
- 3) “appid_error”, 表示 appid 不符合要求;
- 4) “mic_device_detect_error”, 表示使用的是非讯飞麦克风;
- 5) “login error”, 表示登录失败;
- 6) “bnf_recognise_error”, 表示离线语法识别出错, 请检查语法路径及编码格式;
- 7) “build_grammar_error”, 表示离线命令词识别引擎 common.jet 不匹配或不存在, 请检查路径。

□ **描 述:** 在调用接口时 time_per_order 参数若不设置, 将采用默认值(见 user_interface.h 中 time_per_order 参数)。其他两个参数必须给定。其中参数“offline_recognise_start”设置为 1 表示调用,

麦克风阵列	版本: 1.9
	日期: 2020-09-14
MIC_DOC	

“confidence”为识别的置信度阈值，若离线识别引擎识别到的关键词的置信度大于该阈值则认为其识别的命令词是可信的，否则是不可信的，用户可根据实际测试效果，来调节该参数，如:若多次出现误识别，调大该参数等等。在终端调用的时候，用户看到的结果可能是“!!python/str “\u5411\u524D\u8D70””样式，这是因为终端的编码方式产生的。在你编写客户端的时候，可以对返回的结果进行处理，如可以对结果进行类似“srv.response.text.c_str()”的操作。其将会以中文的形式呈现了，不会对用户依靠结果进行其他操作产生影响。

□ 调用示例:

```
//调用离线命令词识别接口，置信度为 40，单次允许交互时间 3s
$ rosservice call xf_asr_offline_node/get_offline_recognise_result_srv 1 40 3
```

3.3 使用步骤

3.3.1 概述

此功能包集为六麦克风阵列在 ros 中的使用接口，含离线命令词识别及在/离线语音合成，仅作为用户快速上手的参考，用户可进一步修改以适配自己的功能和应用方式。

3.3.2 使用说明

1) 拷贝和编译

请将 vvui_ros-master 包放到任一 ROS 工作空间中，如果你的设备中没有 ROS 工作空间，你可执行如下指令来新建工作空间，暂定工作空间名称为 mic_ws:

```
'''
$ cd && mkdir -p ~/mic_ws/src
'''
```

请将 vvui_ros-master 包放到你创建的工作空间下的 src 目录中，然后根据你运行的平台来修改功能包中的 CmakeLists.txt 文件，地址位于 /mic_ws/src/vvui_ros-master/xf_mic_asr_offline/。

修改内容如下,找到 CmakeLists.txt 中如下字段,若为 Nvidia Jetson 平台,则修改为 lib/arm64。若为 x64 平台，如笔记本电脑，则修改为 lib/x64。

```
'''
link_directories(
  lib/x64 #lib/arm64
)
'''
```

然后切换到该工作空间，若为刚创建的 mic_ws，则执行: ~

```
'''
$ cd ~/mic_ws
$ catkin_make
'''
```

此时会进入编译状态，待编译完成后会生成可执行文件，一般为绿色显示。若编译进度未到 100%，机编译中断，则要进行二次检查。一般为动态库路径设置有误。

编译完成之后，执行如下命令使工作空间生效:

```
'''
$ echo "source ~/mic_ws/devel/setup.bash" >> ~/.bashrc
$ source ~/.bashrc
'''
```


麦克风阵列	版本: 1.9
	日期: 2020-09-14
MIC_DOC	

之后关闭所有已打开的终端。

2) udev rules 检查

若为初次使用，且未配置过本麦克风的 udev 规则，则执行如下指令进行配置，若已配置过，可忽略该步骤。

```
'''
$ sudo cp xf_mic.rules /etc/udev/rules.d/
'''
```

然后执行如下指令重启 udev 服务：

```
'''
$ sudo service udev restart
'''
```

再次插拔麦克风设备即可。

3.3.3 启动方式

1) 启动麦克风接口功能包：

```
'''
$ roslaunch xf_mic_asr_offline xf_mic_asr_offline.launch
'''
```

注意：程序会主动检测麦克风连接及主麦设置情况，若主麦未设置，则需要等待用户进行设置后才可以进行操作。你可使用唤醒词唤醒来设置麦克风，也可调用 3.2.6 中的接口进行主麦设置。

2) 调用服务接口

在任一目录下新建终端，然后根据 3.2 节中各接口的调用示例程序调用服务即可，接口描述见 3.2 节。

麦克风阵列	版本: 1.9
	日期: 2020-09-14
MIC_DOC	

4. 讯飞开放平台应用创建与集成

首先到[讯飞开放平台](#)注册账户，然后点击右上角“控制台”，进入自己的控制平台。如果是首次使用讯飞开发平台，则控制台中应用是空的，点击创建新应用。

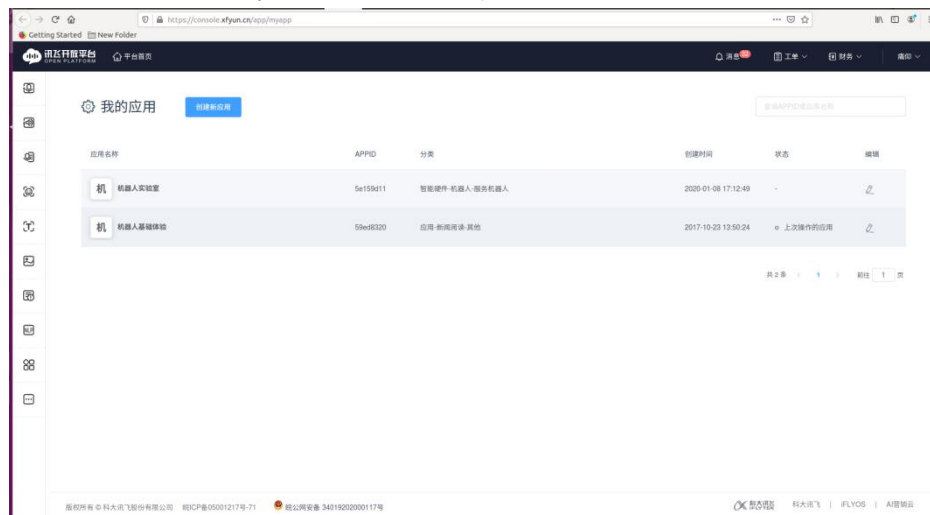


图 8 进入控制台

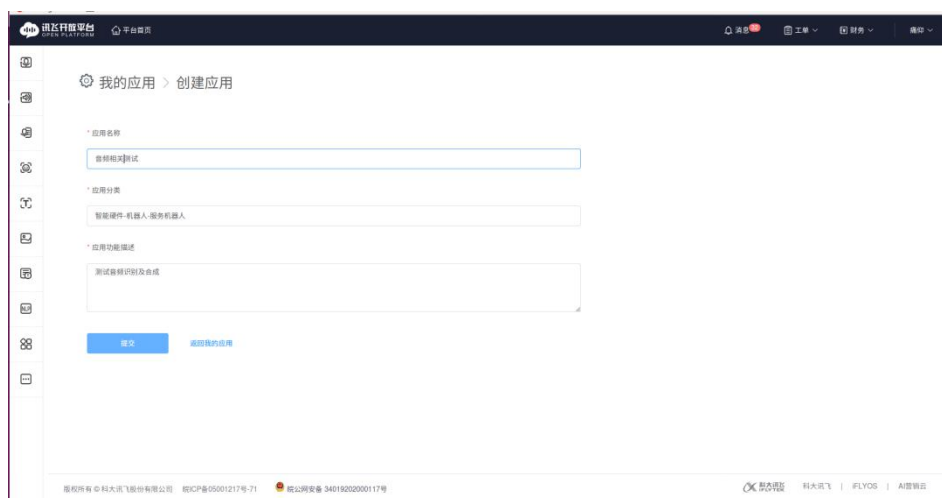


图 9 创建新应用

点击提交后，可看到用户的应用列表，选择刚刚创建的应用，则进行该应用配置界面，在左侧会有“语音识别”，“语音合成”，“语音扩展”，“人脸识别”，“图像识别”等菜单。创建完成之后就可以下载 sdk 使用了，两种方式：

- 1) 在开放平台主页，在菜单中选择 SDK 下载，然后选择应用名称，平台以及要使用的能力，此时下载的 SDK 会将所选的能力打包下载，其中的 lib 下的 libmsc.so 动态库也可看做是打包在一起的，其适用于你勾选的所有能力。如 4.5 所示。
- 2) 选择单个能力(如 4.1-4.4 介绍)，然后下载单个能力的 SDK 包，此时 SDK 里的动态库仅仅适用于该能力，若下载多个能力，则会有多个 SDK 包，对应多个 libmsc.so 动态库。

麦克风阵列	版本: 1.9
	日期: 2020-09-14
MIC_DOC	

4.1 集成离线命令词识别

4.1.1 添加离线命令词识别应用

选择左侧菜单中“语音识别”中的“离线命令词识别”，然后选择图中右侧的“Linux MSC”，点击“下载”来下载 sdk。

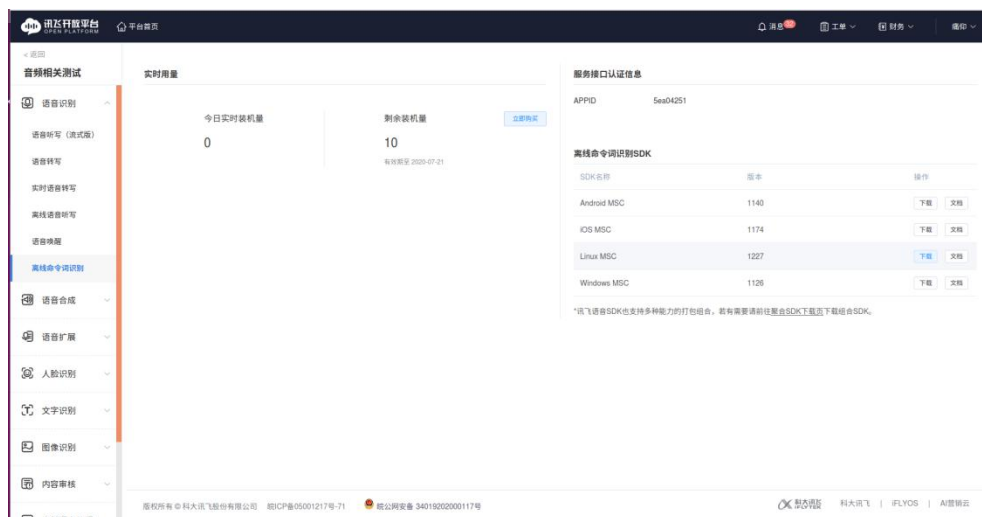


图 10 下载离线命令词识别 SDK

下载好的离线命令词识别的 sdk 包名称是以”标识+appid”形式命名的解压后的目录结构为:

```

├── bin
│   ├── call.bnf
│   ├── msc
│   └── wav
├── doc
│   ├── BNF Grammar Development Manual.pdf
│   └── readme.txt
├── include
│   ├── msp_cmh.h
│   ├── msp_errors.h
│   ├── msp_types.h
│   └── qsr.h
├── libs
│   ├── x64
│   └── x86
├── README
├── release.txt
├── samples
│   ├── asr_offline_record_sample
│   └── asr_offline_sample

```

麦克风阵列	版本: 1.9
	日期: 2020-09-14
MIC_DOC	

4.1.2 集成方法

若在使用离线命令词识别时出现 10102 等错误时，需要替换为自己的 APPID 和离线识别引擎，appid 我们在创建应用的时候就已得到，离线识别引擎 common.jet 在 bin/msc 中，找到 common.jet 后就可以进行替换了。

1) **替换 sdk:** 将用户自己的 common.jet 文件替换 vvui/config/msc/res/asr 路径下的 common.jet。将用户自己的 appid 替换 vvui/include 中 user_interface.h 文件中的“char *APPID”。然后再次编译。

2) **替换 ros 接口包:** 将用户自己的 common.jet 文件替换 vvui_ros-master/xf_mic_asr_offline/config/ms/res/asr 路径下的 common.jet，将 appid 替换 vvui_ros-master/xf_mic_asr_offline/config 中 mic_offline_params.yaml 文件中的“appid”。然后再次编译。

4.2 集成 AIUI

4.2.1 添加 AIUI 应用

选择左侧菜单中的“其他”，会看到右侧的列表，如图所示。

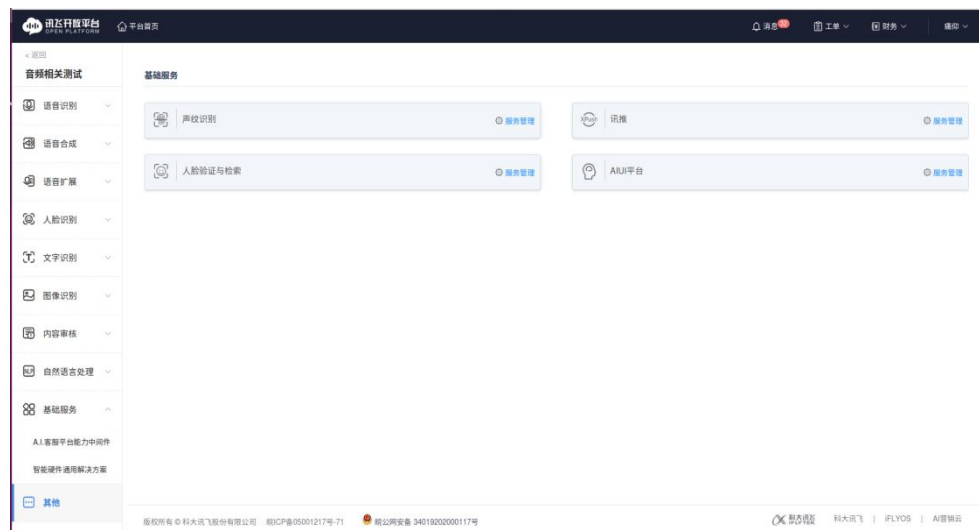


图 11 添加 AIUI 应用

选择“AIUI 平台”右侧的“服务管理”可以进入 AIUI 开放平台，该平台左侧有“应用信息”，“应用配置”，“开发工具”，“审核上线”，“服务统计”，“用户统计”菜单栏。若该 appid 是第一次添加 AIUI，系统会自动跳转到“应用信息”菜单下让用户设置应用平台，选择“Linux”即可。若已添加过则默认进入“应用配置”菜单中，如图 12 所示。

麦克风阵列	版本:	1.9
	日期:	2020-09-14
MIC_DOC		

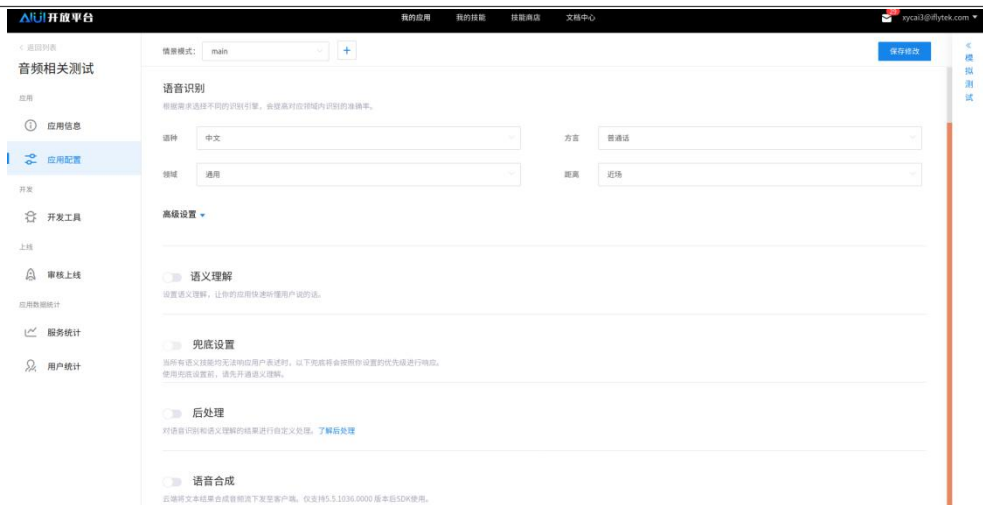


图 12 AIUI 应用配置

其中情景模式默认为“main”，可以结合用户场景需求根据使用说明来设置其他情景模式。在这里用户需要将“语义理解”，“兜底设置”，“语音合成”三个高级设置勾选，在语义理解中用户关注“语义技能”，首先选择“商店技能”，然后“添加商店技能”，你可以在弹出的技能商店中选择你需要的技能，如“天气”，“星座”等，如图 14 所示。点击确定即可。添加完后要选择页面右上角的“保存修改”才可生效。这个时候，就可以使用你刚添加的技能进行对话了。

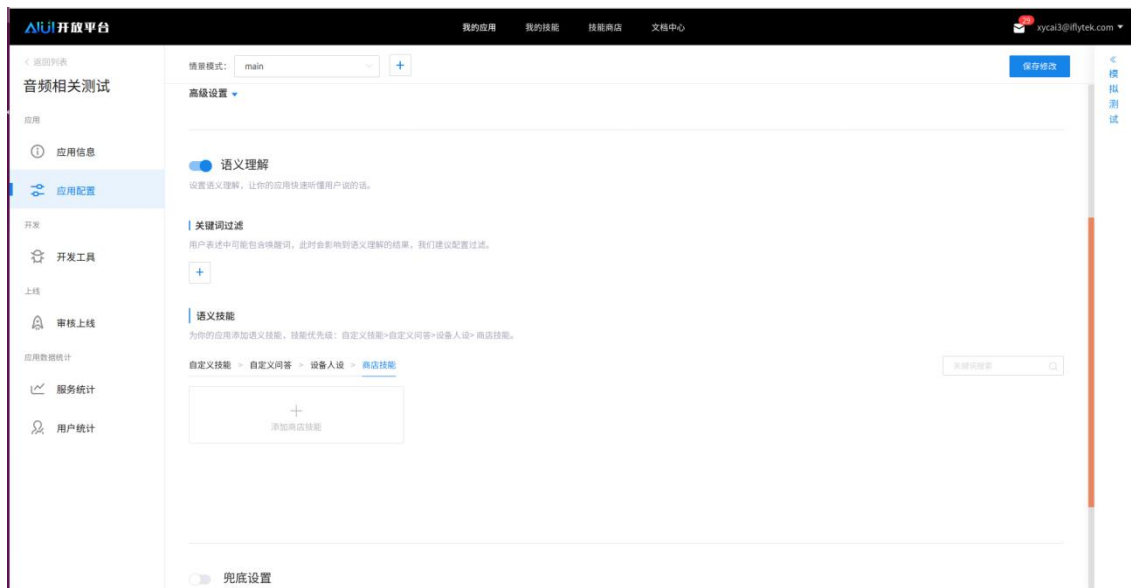


图 13 AIUI 配置语义理解

麦克风阵列	版本: 1.9
	日期: 2020-09-14
MIC_DOC	



图 14 AIUI 添加商店技能

在“兜底测试”中将“图灵机器人”，“讯飞闲聊”等开启后，就可以避免在一些语料未开通时，机器人依然可以答复诸如“这些我还没学会”等等的兜底回复。除此，aiui 还提供了一些自定义的技能便于用户使用，选择“我的技能”菜单，系统会跳转到[技能控制台](#)，然后根据[创建方法](#)在该平台创建完技能并发布后，再回到图中 AIUI 应用配置界面，在语音理解-语义技能中的自定义技能中添加刚创建的技能然后“保存修改”就可以了。之后就可以根据自己创建的技能以及技能标识进行交互判断了。当然，关于 AIUI 和其相关的技能应用还有很多，用户可根据官网的文档进行系统的学习，以充分使用科大讯飞开放平台的能力。

完成“应用配置”后，用户就可以下载 AIUI 的 sdk 了，选择“开发工具”，可以看到如图 15 所示的界面：

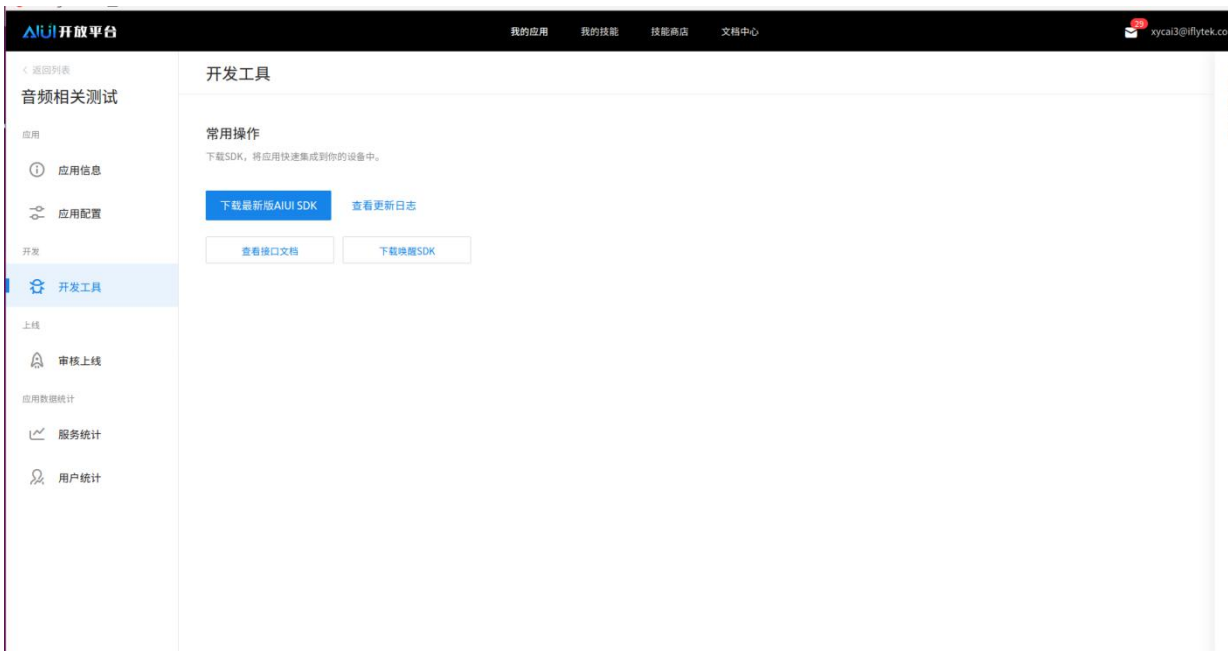


图 15 下载 AIUI SDK

麦克风阵列	版本: 1.9
	日期: 2020-09-14
MIC_DOC	

点击“下载最新版 AIUISDK”即可。下载好的 sdk 包名称是以“标识+appid”形式命名的解压后的目录结构为

```

bin
├── gm_continuous_digit.abnf
├── ise_cn
├── ise_en
├── msc
├── output.txt
├── source.txt
├── tts_sample.wav
├── userwords.txt
├── wav
├── include
│   ├── aiui
│   ├── msp_cmn.h
│   ├── msp_errors.h
│   ├── msp_types.h
│   ├── qise.h
│   ├── qisr.h
│   └── qtts.h
├── libs
│   ├── x64
│   └── x86
├── README
└── samples
    ├── aiui_sample
    ├── asr_sample
    ├── iat_record_sample
    ├── iat_sample
    ├── ise_sample
    └── tts_sample

```

4.2.2 集成方法

若在使用 2.5 节中在线人机交互案例时出现”11201”的错误，是因为 appid 选择的 AIUI 应用是免费的，每天交互次数受限，你可根据如下方法进行修改，来使用你创建的“aiui”应用，方法是在“AIUI 开放平台”上找到“APPID”和“APPKEY”，在进行替换的时候只用这两个参数就可以了，替换方法将这两个参数分别替换 AIUITester.cpp 文件中的 AIUITester::createAgent 函数里面的字符串 appid 以及 key。

麦克风阵列	版本: 1.9
	日期: 2020-09-14
MIC_DOC	

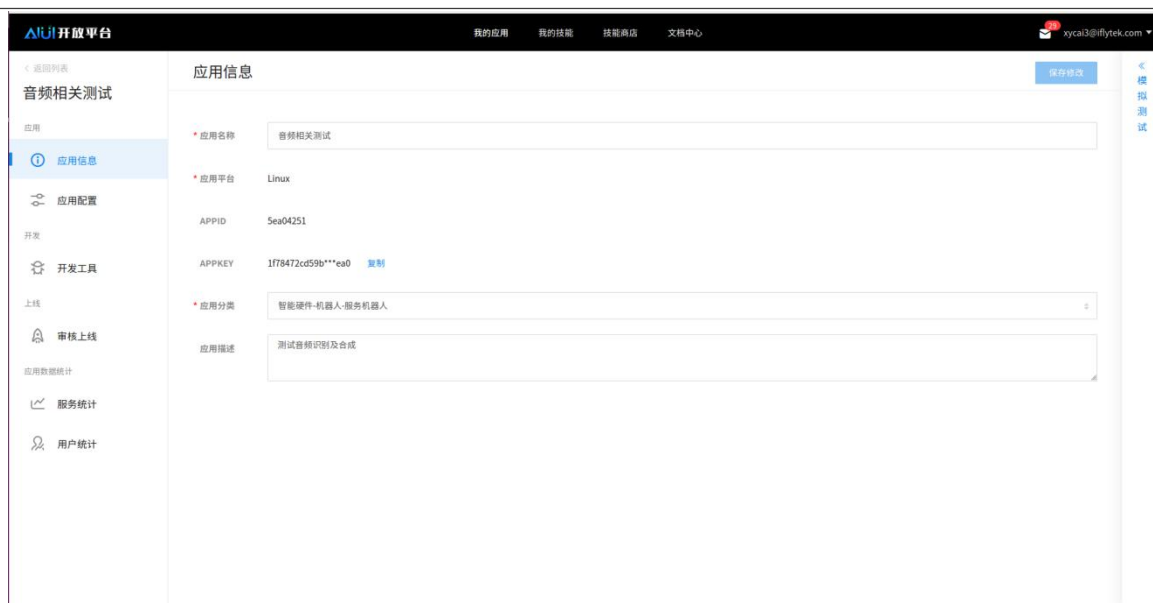


图 16 查看 APIID 与 APPKEY

4.3 合并下载 SDK

进入讯飞开发平台主页，上部菜单栏里找到“资料库”，然后选择“SDK”下载。

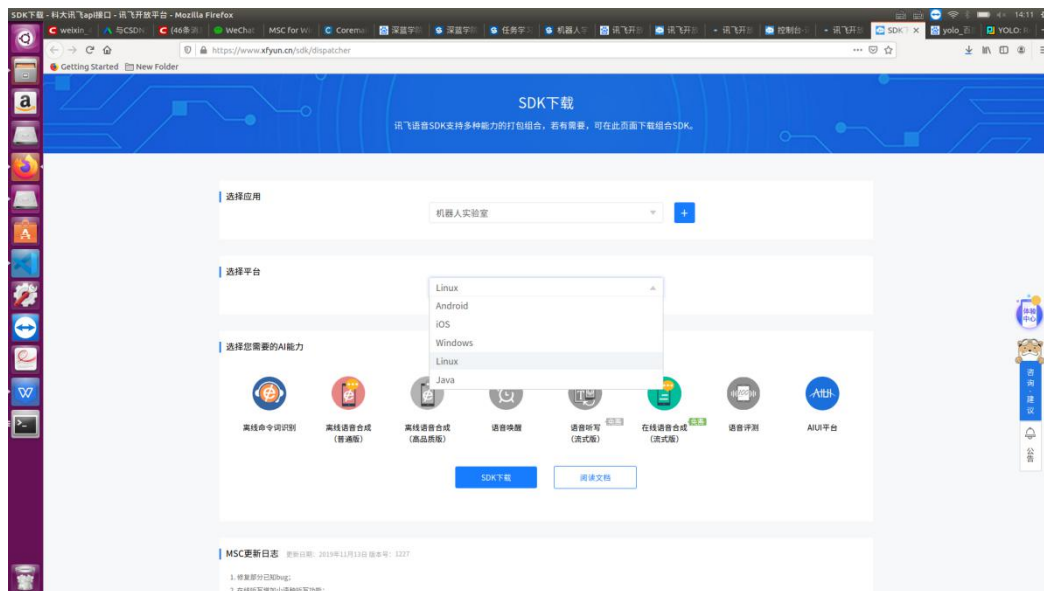



图 17 合并下载 SDK

麦克风阵列	版本: 1.9
	日期: 2020-09-14
MIC_DOC	

5. 镜像烧录

在麦克风阵列板上已带有固件，若用户想对本麦克风阵列进行二次开发，或系统升级，可再刷新的固件，固件烧录软件为 IH 固件烧录工具，首先安装该烧录软件，软件和固件获取见[网盘](#)(访问密码：LXLN)烧录过程如下：

- 1) 运行 IH 固件烧录工具；
- 2) 选择/替换固件点击按钮可以替换所需要烧录的固件。
- 3) 将 HUB 连接到 PC，并将 HUB 的电源和所有 USB 数据线连接好
- 4) 连接一个 ADFU 设备或多个 ADFU 设备(可参考[标识 USB 端口号方法](#))
- 5) 点击开始下载固件：

具体步骤如下：

☑ 1 准备固件烧录

点击“1”处选择固件，然后按住麦克风 ADFU 键，并将麦克风 usb 口插到 P C。若显示 usb 设备被检测到，其 USB 标识显示为绿色，表示设备连接成功，此时“开始下载”按钮呈有效可用状态。界面显示如下图(仅供参考)：



图 18 烧录软件界面

☑ 2 固件烧录

注意事项：设备正在烧录固件时，请不要随意移动设备或拔除 USB 线，以免造成 USB 连接松动或连接断开，导致固件烧录失败。

点击“下载”按钮后，工具开始给设备烧录固件。进度条会显示下载固件百分比，进度条右边显示烧录时间，且工具各个按钮呈灰色状态(不可用)。界面显示如下图(仅供参考)：

麦克风阵列	版本: 1.9
	日期: 2020-09-14
MIC_DOC	



图 19 烧录状态界面

☑ 3 固件烧录完成

固件烧录进度显示 100%，累计通过和当前通过数量已更新，所连接的设备在固件烧录完成后自动弹出，工具显示当前 0 USB 设备被检测到。界面显示如下图(仅供参考)：

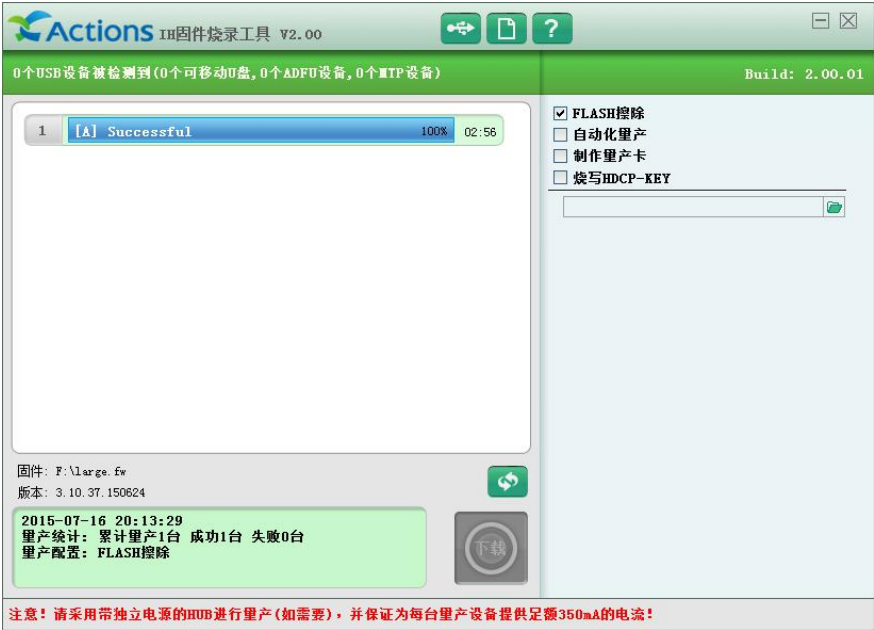


图 20 烧录结束状态界面

注意：请在设备完全烧录固件完成后再拔除 USB 线，以免设备开机之后因固件下载未完成而出现异常情况。

麦克风阵列	版本: 1.9
	日期: 2020-09-14
MIC_DOC	

6. 版本升级

本麦克风版本号涉及到四个部分：

☑ 1 麦克风板载固件版本(通常不需要升级)

☑ 2 SDK 与协议版本(升级使用)

☑ 3 文档版本(配套升级)

在程序开机的时候，会显示当前的软件版本与协议版本。

麦克风版本会根据需要进行升级，通常仅升级协议。用户可直接使用给出的新版本 SDK。也可以仅改变协议文件即“system.tar”及“config.txt”文件，用户需要将这两个文件替换现有文件即可。对于 sdk 来说，用户需要用这两个文件替换 vvui/tmp 文件夹中的对应文件；对于 ros 功能包集来说，用户需要用这两个文件替换 vvui_ros-master/xf_mic_asr_offline/tmp 文件夹中的对应文件。

麦克风阵列	版本: 1.9
	日期: 2020-09-14
MIC_DOC	

7. 常见错误

☑ 找不到麦克风设备，请重新插拔

该现象表示系统未检测到麦克风。

☑ 在启动后出现报红或 segment fault 字样

未设置麦克风 udev。可尝试使用 sudo 启动或设置 udev。

检查一下是否未配置过本麦克风的 udev 规则，执行如下指令进行配置：

```
$ sudo cp xf_mic.rules /etc/udev/rules.d/
```

然后执行如下指令重启 udev 服务。

```
$ sudo service udev restart
```

再次重启设备或插拔麦克风设备即可。

☑ “在启动后出现 libcjson.so.1 不存在”

需要安装 cJSON，且需要将/usr/local/lib 中的 libcjson.so.1 移动到/usr/lib 下。

☑ “23300”或 “bnf_recognise_error”

离线命令词识别时语法 bnf 有误，请检查标点符号，槽定义等是否有误，可参考 bnf 编写说明书。

☑ “10407”或“unfit_appid_and_lib_error”

appid 与动态库 libmsec.so 不匹配，可参考 4.2.2 进行修改，在修改后一定要编译后再运行；

☑ “10102”或“build_grammar_error”

离线命令词识别过程中，common.jet 资源文件路径有误，为避免出错，建议修改为绝对路径；

☑ “11212”或“low_confidence error or 11212_license_expired_error”

离线命令词识别过程中，离线授权体验期过期，需重新购买，也可重新注册讯飞开放平台获取新 appid 及资源文件，按照 4.1.2 节指引进行替换。

☑ “11210”

appid 与离线资源即 common.jet 不匹配。

☑ “11201”

在线业务超次数或离线资源装机量超限。如离线命令词识别中，一个试用版的 appid 和资源在超过 10 台机器上使用。

☑ “文件无法打开...”

在执行案例中，若在录音时出现文件无法打开的问题，需确认音频文件是否有读写写的权限，可删除该录音文件后再运行，或者使用超级用户权限。

☑ “在启动例程时一直显示麦克风未启动”

请检查是否有开启录音后，未停止录音的操作。若有，则需要手动停止录音或者断开与麦克风的连接来结束通信。

☑ 其他能力应用错误

可参照 <https://shimo.im/sheet/w3yUy39uNKs0J7DT> 进行修改。

麦克风阵列	版本: 1.9
	日期: 2020-09-14
MIC_DOC	

8. 版本记录

版本号	描述	操作者
V1.0	1.麦克风阵列功能介绍; 2.麦克风阵列硬件介绍; 3.sdk 软件使用说明; 4.镜像烧录方法;	xinnie
V1.1	1.修改了 sdk 案例执行步骤;	xinnie
V1.2	1.对离线命令词识别功能进一步封装,将用户可调的参数;	xinnie
V1.3	1.添加 ros 接口; 2.添加开放平台应用创建过程; 3.添加版本升级方法;	xinnie
V1.4	1.修改少量图片;	xycai3
V1.5	1.更正部分 bug;	xinnie
V1.6	1. 增加麦克风接口描述; 2. 添加自定义唤醒词接口; 3. 更改 ros 接口中录音为服务; 4. 增加唤醒角度实时获取服务;	xinnie
V1.7	1. 修复文档说与程序执行中的不一致问题; 2. 将灯光点亮和主麦设置分为两个接口;	xinnie
V1.8	1. 关闭录音时, 不设置灯光与主麦; 2. 离线命令词识别动态库开放; 3. 增加软件和协议版本获取接口;	xinnie
V1.9	1. 修复主麦设置问题; 2. 修复离线词识别无音源仍识别出结果的问题.	xinnie