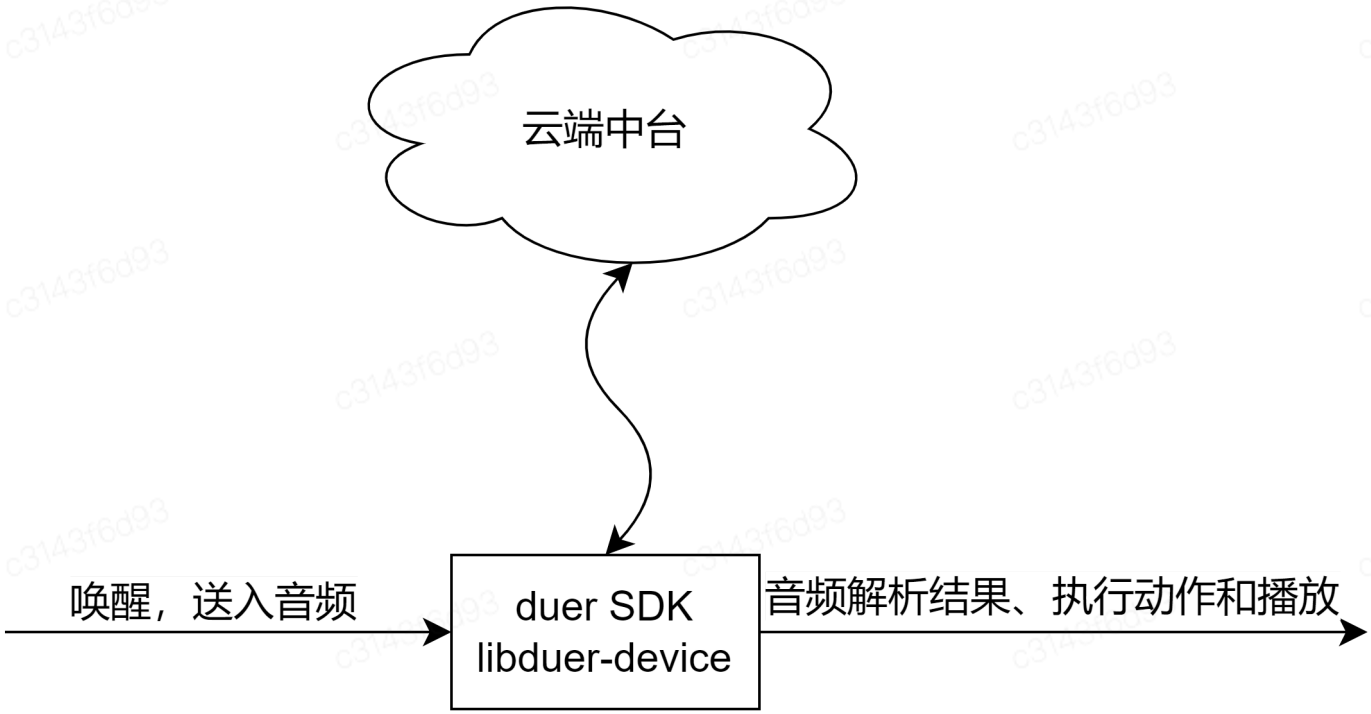
设备端SDK使用文档

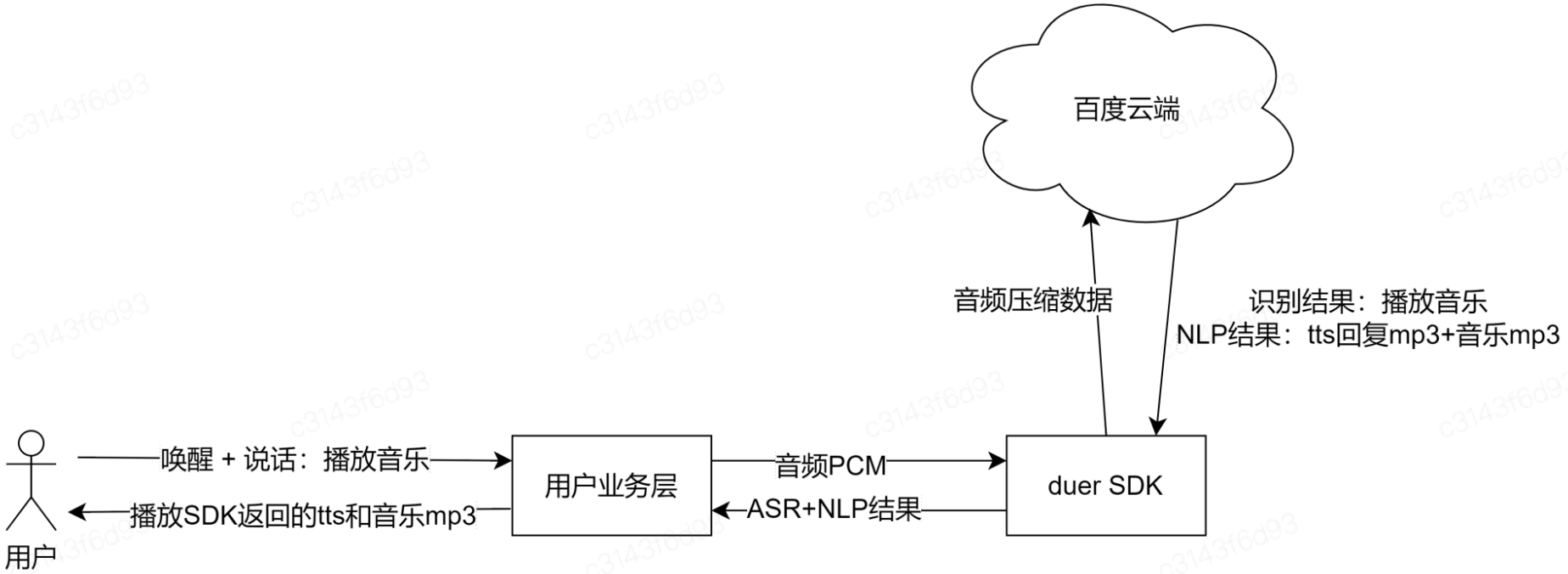
1. SDK说明

1.1. 基本框架

SDK(libduer-device)接收用户送入的音频，发送到云端中台后，由云端中台完成ASR和NLP处理，将相关结果返回给sdk，再由sdk进行解析返回相关意图给用户进行处理。



典型场景示意：



1.2. 重要接口说明

激活相关

四元组：

duer SDK激活时，需要传入四元组和激活地址，四元组可以从官网购买获取。只有激活后，SDK的一系列功能才能正常使用。

```
{
    "fc":"xxxx",    厂商名称ID
    "pk":"xxxx",    厂商下面产品的ID
    "ak":"xxxx",    每个设备的ID
    "sk":"xxxx",    跟AK一一对应的鉴权使用的
```

```
    "active_url": "https://smarthome-bdvs.baidubce.com/mini/v1/device/active"
}
```

激活通过dueros_bdvs_active_device传入四元组信息，返回dueros的profile用于启动dueros连接，使用参考demo的duerapp.c

```c
// 头文件: lightduer_bdvs_active_device.h
typedef struct {
    int type;               // 见 DUEROS_BDVS_ACTIVE_TYPE
    void * object;          // 以文件路径激活时，为文件路径，以字符串激活时，为json字符串
    char mac_addr[13];      // 设备mac地址 例如0016EAAE3C40，长度12 第13位为\0
    int retry_times;        // 激活重试次数 0:一直尝试激活
} dueros_bdvs_active_info_t;

char * dueros_bdvs_active_device(dueros_bdvs_active_info_t *active_info);
```

启动dueros连接

使用激活接口返回的profile进行启动，使用参考demo的duerapp.c

```c
// 头文件: lightduer_connagent.h
/**
 * Start the LightDuer egnine.
 *
 * @param data, the configuration data
 * @param size, the data size
 * @return int, the start result, success return DUER_OK, failed return DUER_ERR_FAILED.
```

```
 */
int duer_start(const void *data, size_t size);
```

唤醒后传入音频

唤醒由用户自行处理，唤醒后，通过duer_dcs_on_listen_started启动录音准备，再通过duer_voice_start设置采样率，随后可以通过duer_voice_send持续发送音频PCM数据

```c
// 头文件: lightduer_dcs.h
/**
 * DESC:
 * Notify DCS when recorder start to record.
 *
 * PARAM: none
 *
 * @RETURN: 0 when success, negative when fail.
 */
int duer_dcs_on_listen_started(void);

// 头文件: lightduer_voice.h
int duer_voice_start(int samplerate);
int duer_voice_send(const void *data, size_t size);
```

vad结束动作

云端收到音频后，会做vad检测，需要停止发送音频时，会通过duer_dcs_stop_listen_handler通知用户，停止send音频

注：云端vad涉及网络通信，存在不可靠问题，需要用户在设备端自行增加音频最长发送时间的控制，以兜底stop信号丢失情况。

```c
// 头文件lightduer_dcs.h
/**
 * DESC:
 * Developer needs to implement this interface to stop recording.
 *
 * PARAM: none
 *
 * @RETURN: none.
 */
void duer_dcs_stop_listen_handler(void);
```

接收ASR结果

用户实现该函数，用于接收发送音频进行ASR之后，转成的文本信息。

```c
// 头文件: lightduer_dcs.h
/**
 * DESC:
 * Developer needs to implement this interface to get the ASR result.
 *
 * PARAM[in] text: the ASR text result.
 * PARAM[in] type: "INTERMEDIATE" or "FINAL".
 *
 * @RETURN: DUER_OK if success,
```

```
 *          DUER_MSG_RSP_BAD_REQUEST if the payload is invalid,
 *          DUER_ERR_FAILED if other error happened.
 */
duer_status_t duer_dcs_input_text_handler(const char *text, const char *type);
```

接收nlp结果

协议示例

```
{
    "directive": {
        "header": {
            "name": "SetResource",
            "namespace": "ai.dueros.device_interface.thirdparty.extensions",
            "messageId": "YmU5MDk2MmMtMTBmMy05ZjBmLTg3ZjEtYzMwNWMyMWFjZDVmKw==",
            "dialogRequestId": "95ab62e95040a48ae4cf3ab728605974000005"
        },
        "payload": {
            "extension": "{\"origin\":{\"query\":\"今天星期几\",\"encoding\":\"utf8\",\"results\":
[{\"domain\":\"ask_time\",\"intent\":\"ask_time\",\"slots\":{}}]},\"actionList\":[{\"name\":\"media.play\",\"arg\":
{\"domain\":\"other\",\"type\":\"common\",\"info\":{\"trackUrl\":\"http://smarthome-
test.baidubce.com/mini/v1/voice/du_fcdu_pkdu_ak_m_00141716358459501-001.mp3\"}}}]}"
        }
    },
    "iot_cloud_extra": {
        "timestamp": "1716358459911",
        "index": 2
    }
```

```
}
```

sdk内部已经包含协议的解析动作，用户只需要注册action和intent的接收函数完成相关动作即可。

```c
// 示例代码 duerapp_intent.c
// 头文件 lightduer_bdvs_data_parse.h

// 注册action的处理函数，对应协议actionList下的name:media.play
int add_new_action_handle(char * action_name, ACTION_HANDLE_FUNC in_func);
// 注册intent的处理函数，对应协议origin下的results，domain和intent
int add_new_intent_handle(char * domain, char * intent, INTENT_HANDLE_FUNC in_func);

// 注册完整的协议接收函数，可由用户自己解析，传入的为directive
void bdvs_set_directive_cb(directive_callback in_func);

// 相关动作执行完成后的回调函数，用户可注册接收对应类型列表的处理结束事件
void bdvs_set_class_done_cb(int type, class_done_callback in_func);
```

播放器相关：

播放器由用户自行维护，demo提供播放器示例，见duerapp_music.c和duerapp_player.c

注：播放器要求url能够顺序播放，nlp结果会存在多个url需要播放的情况，按照解析的顺序依次放入播放队列等待播放就可以。

大模型的播报会以多个url的形式返回，用户需要控制url之间的间隔，以免大模型结果播报的间隔明显问题，且由于大模型的返回无法预知，尾部几个url会存在没有数据问题，需要增加数据请求的超时时间，在超时后清除后续的url。

2. linux版本SDK使用

2.1. 富翰芯片SDK下载以及使用

sdk目录结构：

```Bash
├── doc                          # sdk相关文档
├── example                      # demo示例目录
|   ├── bdvs-demo                # bdvs linux标准demo工程
|   └── fullhanv3-demo           # 对应平台的demo工程
├── include                      # sdk提供的相关头文件
└── lib                          # sdk提供的库
    ├── arm
    |   └── fullhanv3linuxdemoconfig   # 指定平台库
    └── x86
        └── bdvslinuxdemoconfig        # linux标准平台通用库
```

使用方法

```Bash
cd example\fullhanv3-demo #进入demo目录
make    # 编译
# 再将生成的可执行重新和config.json拷贝到设备上运行
```

3. RTOS版本SDK使用详情

3.1. ESP32S3芯片下载以及使用

sdk目录结构

```Bash
├── doc                         # sdk相关文档
├── example                     # demo示例目录
|   ├── bdvs-demo               # bdvs linux标准demo工程
|   └── esp32-demo              # 对应平台的demo工程
|       ├── components          # esp32工程组件模块
|       |   ├── bdvs_player     # 播放器模块
|       |   ├── bdvs_wakeup     # 唤醒模块
|       |   ├── duer-device     # sdk的头文件和库组件，与最外层目录的include和lib下对应的库相同
|       |   |   ├── include
|       |   |   └── lib
|       |   └── profile         # 四元组配置profile，使用时烧录到对应分区
|       ├── docs                # esp32工程相关文档
|       ├── main                # esp32g工程主要业务代码
|       ├── output              # esp32编译的产物包
|       └── tools               # 一些工具
├── include                     # sdk头文件
└── lib                         # sdk库文件
    ├── esp32
    |   └── esp32config
    └── x86
        └── bdvslinuxdemoconfig
```

准备工作：

请确保开发者已具备以下基本能力，再进行本例程的二次开发工作。

```Bash
A: c/c++开发能力，基本的python阅读和使用
B: Makefile、CMake、Shell等相关编译工具的阅读和使用能力
C: git拉取代码及相关命令的操作，git子模块的更新和切换版本
D: 熟悉乐鑫官方IDF、ADF框架，能够完成乐鑫官方提供的例程编译、烧录和使用，详细信息见adf、idf的github文档
```

本例程使用环境

```Bash
系统：Ubuntu 22.04
ADF版本：v2.6
IDF版本：v2.6子模块配套的版本，v4.4.4
components/esp-sr模块：切换到master分支，再切换到commit ed2311821412f
其他子模块：adf配套的版本
开发板：ESP32-S3-Korvo-2
```

支持范围

```Bash
百度：
    libduer-device库的框架和使用问题
    设备四元组购买及激活的相关问题
    在线语音识别及nlp技能问题

以下问题请寻求乐鑫技术支持或其他支持：
    ADF、IDF的编译环境问题
```

```
    唤醒技术问题
    回声消除问题
    播放器问题
    生产问题

示例:
1.申请的四元组烧录后，激活失败无法使用：百度
2.设备激活后，一些常见的交互没有url返回：确保音频输入正确的情况下再联系百度支持
3.github拉取adf仓库时，一直拉不下来：自行解决
4.唤醒不了、唤醒效果差：联系乐鑫支持
5.播放器卡顿：检查网络速率，或联系乐鑫支持
6.生产时四元组烧录：百度提供的是四元组内容，生产烧录方式自行解决
7.在demo里面增加自己的模块：参考已有模块，自行添加
8.需要做app配网功能：联系乐鑫支持
```

拉取github上的esp32 adf官方源码，并更新子模块

```bash
git clone git@github.com:espressif/esp-adf.git -b v2.6
git submodule update --init
```

安装配置esp32环境

```bash
#设置ADF IDF环境变量
ADF_PATH="your path/esp-adf"
```

```
IDF_PATH="your path/esp-adf/esp-idf"

#安装相关环境，参考idf官方手册
$IDF_PATH/install.sh
. $IDF_PATH/export.sh

#再单独更新一下components/esp-sr/目录到master分支，可以降低乐鑫的唤醒模型体积，demo使用版本为commit ed2311821412f52457fd75a9eee5c99da8a587dd

#应用idf对应的patch文件，esp-adf/idf_patches
```

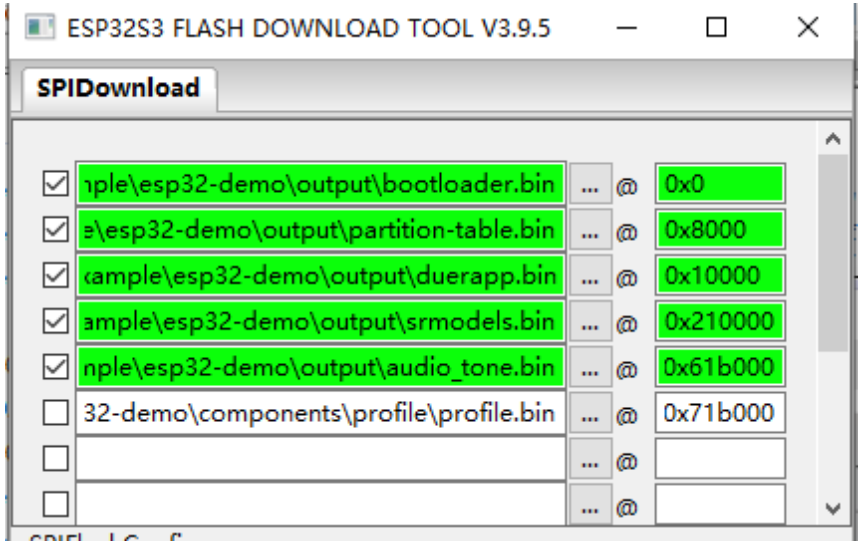
在demo目录进行编译

```bash
# 进入到demo目录，修改build.sh里面配置的adf路径，再运行
./build.sh

#编译完后会在output目录下生成相关bin文件
```

进行烧录

按照分区表烧录个分区，profile是四元组配置文件，每个机器不一样，注意烧录时区分并勾选，output\example\esp32-demo\tools下有烧录工具



使用：

```Bash
1 烧录后启动通过串口命令【wifi_set ssid passwd】设置wifi联网，等待联网成功，设备激活后（大概20+s）就可以通过“hi 乐鑫”进行唤醒和交互
```