科大讯飞股份有限公司

IFLYTEK CO.,LTD.

科大讯飞 *MSC* 新手指南（*iOS*）

目录

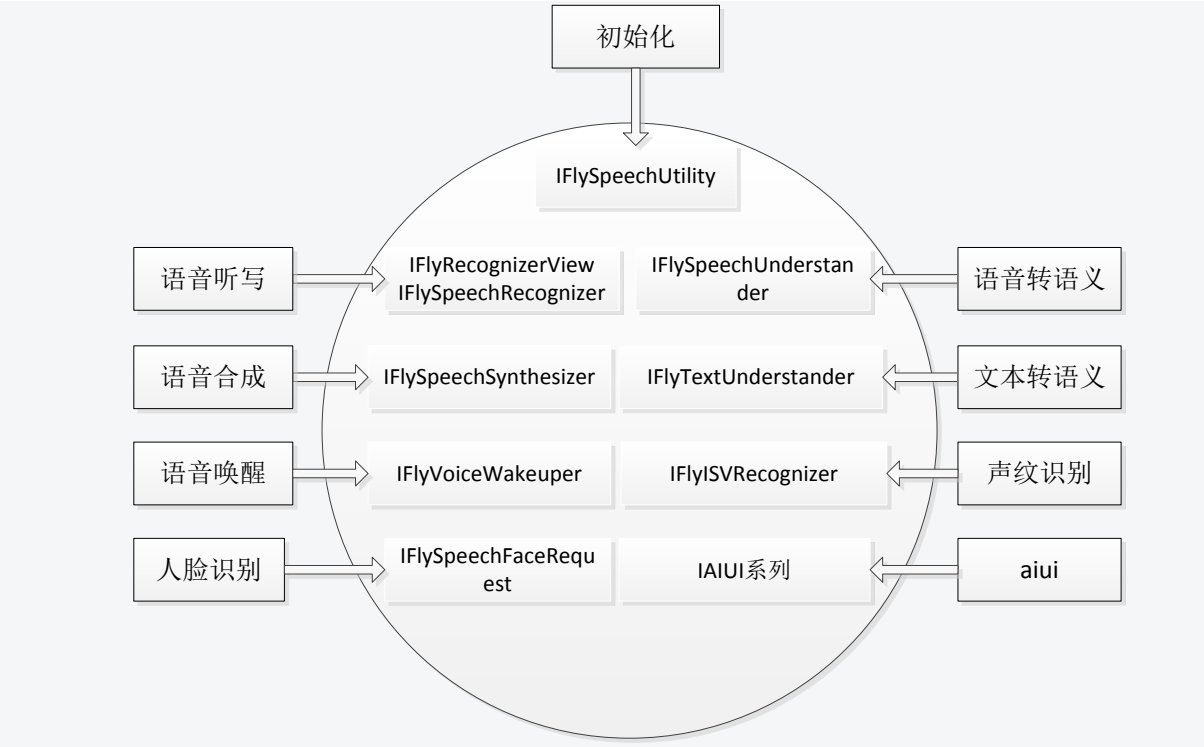
1. 概述

本文档是开发者集成科大讯飞 *MSC*（*Mobile Speech Client*，移动语音终端）*iOS* 版 *SDK* 的使用指南，定义了语音听写、语音识别、语音合成、语义理解，语音评测、语音唤醒，人脸识别等服务接口的使用。关于各服务接口更详细的说明，请参考《*com.iflytek.IFlyMSC.docset*》。在集成过程如有疑问，可登录语音云开发者论坛(*http://bbs.xfyun.cn/*)，查找答案或与其他开发者交流。

1.1. *iOS* 语音 *SDK* 概述

MSC SDK 的主要功能接口如下图所示：

图 1-1 *MSC SDK* 主要功能接口



1.2. 业务描述

为了更好地理解后续内容，这里对文档中出现的若干专有名词进行解释说明，更为详细的信息可查看官网文档(*http://www.xfyun.cn/doccenter/*)中的语音服务部分。

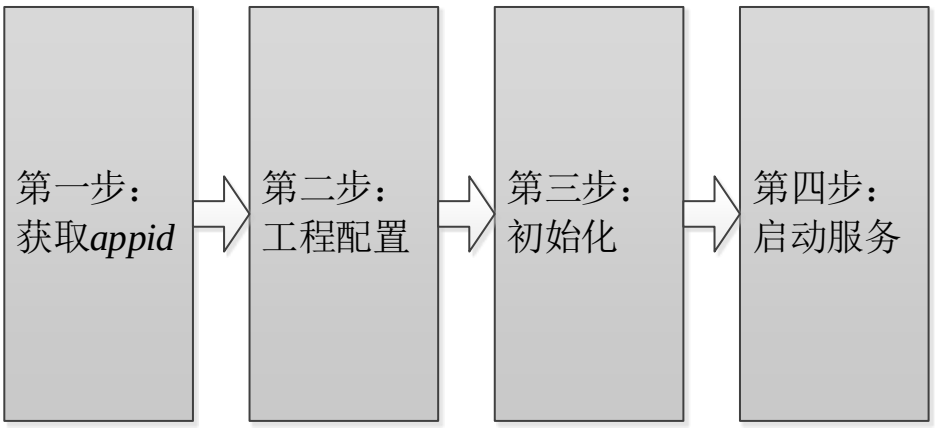
表 1-1 名词解释

名词	解释
语音合成	将一段文字转换成语音，可根据需要将文字合成出不同音色、语速和语调的声音，让机器像人一样开口说话。

语音听写	将一段语音转换成文字，把语音中包含的文字信息提取出来，并可以优先识别用户手机中特有的联系人和个性化数据。
语法识别	判断用户所说的内容是否与预定义的语法相符合，主要用于识别用户是否下达某项指令。使用语法识别前，需要先定义语法。
语义理解	在语音听写基础上，分析理解用户的说话意图，返回结构化的指令信息。开发者可在语义开放平台定义专属的问答格式。 语义理解目前有两个分支，一是开放平台的语义理解，二是 aiui 开放平台的语义理解，分别使用不同的接口实现。
语音评测	通过智能语音技术自动对发音水平进行评价，给出用户综合得分和发音信息。
声纹密码	根据语音波形反映说话人生理和行为特征的语音参数，自动识别说话人身份，声纹识别所提供的安全性可与其他生物识别技术（指纹、掌形和虹膜）相媲美。
人脸识别	基于人的脸部特征信息进行身份识别的一种生物识别技术，可以自动在图像中检测和跟踪人脸，进而对检测到的人脸进行检测和验证。系统同时支持人脸关键点检测、视频流人脸检测等功能，识别率高达 99%。
语音唤醒	即设备（手机、玩具、家电等）在休眠（或锁屏）状态下也能检测到用户的声音，并根据声音提示进行相应操作，开启全语音交互，同时支持唤醒+识别、唤醒+语义的 *OneShot* 方案。
身份验证	在本方案中，开发者可根据应用场景灵活的选择身份验证方式，如单人脸验证、单声纹验证以及人脸+声纹的融合验证方式。这样既解决了单生物特征识别暴露的局限性，也提供了更精准、更安全的识别和检测方案。身份验证方案还会持续增加更多的常用特征，达到更广泛的市场应用前景。

2. 集成流程

图 2-1 集成流程



2.1. 第一步：获取 *appid*

appid 是第三方应用集成语音云开放平台 SDK 的身份标识，SDK 静态库和 appid 是绑定的，每款应用必须保持唯一，否则会出现 10407 错误码。*appid* 在开放平台申请应用时可以获得，下载 *SDK* 后可从 *SDK* 中 *sample* 文件夹的 *Demo* 工程里找到（例如: */sample/MSCDemo/MSCDemo/Definition.h* 的 *APPID_VALUE*）。

2.2. 第二步：工程配置

2.2.1. 添加库

将开发工具包中 *lib* 目录下的 *iflyMSC.framework* 添加到工程中。同时请将 *Demo* 中依赖的其他库也添加到工程中。 按下图示例添加 *SDK* 所需要的 *iOS 系统*库:

图 2-2 Xcode 添加库示例（离线识别或者 aiui 接口请添加下表红色部分）

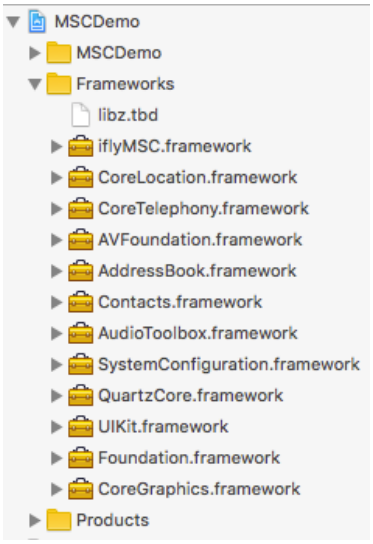


表 2-1 集成所需库及简要说明

库名称	添加范围	功能
iflyMSC.framework	必要	讯飞开放平台静态库
libz.tbd	必要	用于压缩、加密算法。
AVFoundation.framework	必要	用于系统录音和播放。
SystemConfiguration.framework	必要	用于系统设置。
Foundation.framework	必要	基本库。
CoreTelephoney.framework	必要	用于电话相关操作。
AudioToolbox.framework	必要	用于系统录音和播放。
UIKit.framework	必要	用于界面显示。
CoreLocation.framework	必要	用于定位。
Contacts.framework	必要	用于联系人
AddressBook.framework	必要	用于联系人。
QuartzCore.framework	必要	用于界面显示。

CoreGraphics.framework	必要	用于界面显示。
libc++.tbd	离线识别，Aiui 必要	用于支持 C++。
Libicucore.tbd	Aiui 必要	系统正则库。

注意：

1. 添加 iflyMSC.framework 时，请检查工程 BuildSetting 中的 framwork path 的设置，如果出现找不到 framework 的情况，可以将 path 清空，在 Xcode 中删除 framework，然后重新添加。

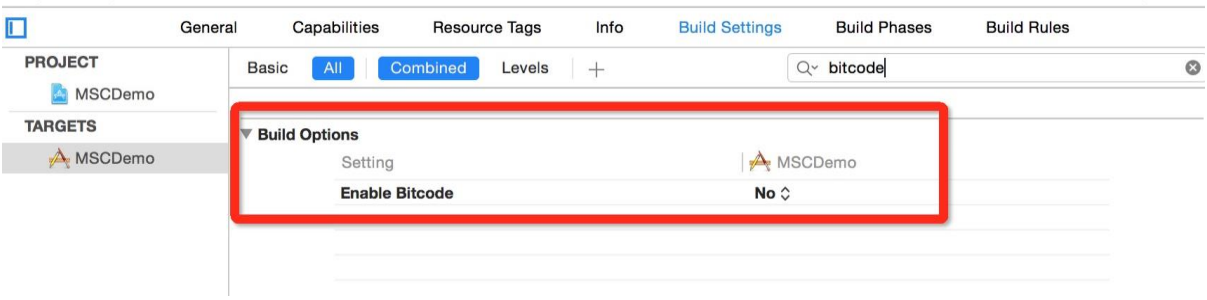
2. 如果使用了离线识别，需要增加 libc++.tbd。

3. 如果使用 aiui，需要添加 libc++.tbd，libicucord.tbd。

2.2.2. 设置 *Bitcode*

在 *Xcode* 7,8 默认开启了 *Bitcode*，而 *Bitcode* 需要工程依赖的所有类库同时支持。*MSC SDK* 暂时还不支持 *Bitcode*，可以先临时关闭。后续 *MSC SDK* 支持 *Bitcode* 时，会在讯飞开放平台上进行 *SDK* 版本更新，*QQ* 支持群中同时会有相关提醒，请关注。关闭此设置，只需在 *Targets - Build Settings* 中搜索 *Bitcode* 即可，找到相应选项，设置为 *NO*。

图 2-3 *Bitcode* 设置



2.2.3. 用户隐私权限配置

iOS 10 发布以来，苹果为了用户信息安全，加入隐私权限设置机制，让用户来选择是否允许。隐私权限配置可在 *info.plist* 新增相关 *privacy* 字段，*MSC SDK* 中需要用到的权限主要包括麦克风权限、联系人权限和地理位置权限：

```
<key>NSMicrophoneUsageDescription</key>
<string></string>
<key>NSLocationUsageDescription</key>
<string></string>
<key>NSLocationAlwaysUsageDescription</key>
<string></string>
<key>NSContactsUsageDescription</key>
<string></string>
```

即在 *Info.plist* 中增加下图设置：

图 2-4 privacy 配置

2.3. 第三步：初始化

初始化示例：

```
//Appid 是应用的身份信息，具有唯一性，初始化时必须要传入 Appid。
NSString *initString = [[NSString alloc] initWithFormat:@"appid=%@",
@"YourAppid"];
[IFlySpeechUtility createUtility:initString];
```

表 2-2 初始化参数说明

参数	说明	必填
appid	8 位 16 进制数字字符串，应用的唯一标识，与下载的 SDK 一一对应。	是
usr	开发者在云平台上注册的账号。	否
pwd	账号对应的密码，与账号同时存在。	否

注意：初始化是一个异步过程，可放在 App 启动时执行初始化，具体代码可以参照 Demo 的 MSCAppDelegate.m。

2.3.1. AIUI 初始化

如果使用的是 aiui 的业务，那么需要执行 aiui 的初始化模块。具体可以参照 demo 的 AiuiService。初始化的配置文件是 aiui.cfg，请关注 demo 的具体内容。

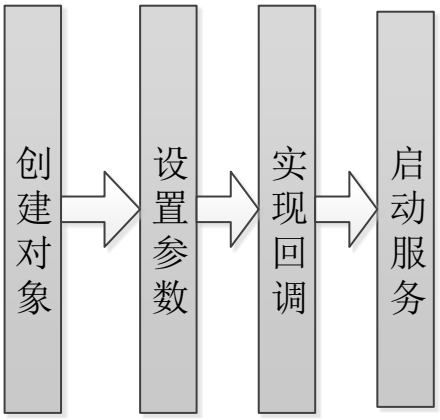
注意：调用 aiui 初始化的代码文件，请修改成.mm 的形式。

```
//包含头文件
#include "iflymsc/AIUI.h"
#include "iflymsc/AIUIConstant.h"

//aiui初始化,请注意红色的文件，需要从demo中获取。
void AiuiInitialize()
{
    //读取配置参数
    NSString *appPath = [[NSBundle mainBundle] resourcePath];
    NSString *cfgFilePath = [[NSString alloc]
initWithFormat:@"%@/aiui.cfg",appPath];
    NSString *cfg = [NSString stringWithContentsOfFile:cfgFilePath
encoding:NSUTF8StringEncoding error:nil];

    //设置模型vad的资源路径
    NSString *vadFilePath = [[NSString alloc]
initWithFormat:@"%@/meta_vad_16k.jet",appPath];
    //将cfg里的vad_res_path字符串替换成模型vad的资源路径
    NSString *cfgString  = [cfg
stringByReplacingOccurrencesOfString:@"vad_res_path"
withString:vadFilePath];

    const char *cfgBuffer = [cfgString UTF8String];

    m_angent = IAIUIAgent::createAgent(cfgBuffer, &m_listener);
    if (NULL != m_angent)
    {
        IAIUIMessage * wakeupMsg =
IAIUIMessage::create(AIUIConstant::CMD_WAKEUP);
        m_angent->sendMessage(wakeupMsg);

        wakeupMsg->destroy();
    }
}
```

2.4. 第四步：启动服务

所有的服务皆遵循如下的流程，如下图：

图 2-7 启动服务流程



3. 语音合成

合成支持在线和离线两种工作方式，默认使用在线方式。如果使用离线服务，有 2 种方式，一种是使用语记 *SDK*（原语音+ *SDK*）提供的免费服务，一种是付费购买后在应用内部集成。相关细节请关注语音云开放平台（*http://www.xfyun.cn/*）。

3.1. 在线合成

本示例对应 *Demo* 的 *TTSUIController* 文件，为在线合成的代码示例。

```
//包含头文件
#import "iflyMSC/IFlyMSC.h"
//需要实现 IFlySpeechSynthesizerDelegate 合成会话的服务代理
@interface TTSUIController :
UIViewController<IFlySpeechSynthesizerDelegate>
    @property (nonatomic, strong) IFlySpeechSynthesizer
*iFlySpeechSynthesizer;
@end
//获取语音合成单例
_iFlySpeechSynthesizer = [IFlySpeechSynthesizer sharedInstance];
//设置协议委托对象
_iFlySpeechSynthesizer.delegate = self;
//设置合成参数
//设置在线工作方式
[_iFlySpeechSynthesizer setParameter:[IFlySpeechConstant TYPE_CLOUD]
 forKey:[IFlySpeechConstant ENGINE_TYPE]];
//设置音量，取值范围 0~100
[_iFlySpeechSynthesizer setParameter:@"50"
```

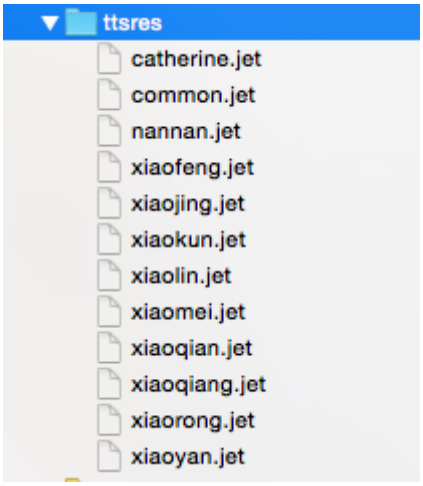
```
 forKey: [IFlySpeechConstant VOLUME]];
//发音人，默认为”xiaoyan”，可以设置的参数列表可参考“合成发音人列表”
[_iFlySpeechSynthesizer setParameter:@" xiaoyan "
 forKey: [IFlySpeechConstant VOICE_NAME]];
//保存合成文件名，如不再需要，设置为 nil 或者为空表示取消，默认目录位于 library/cache
下
[_iFlySpeechSynthesizer setParameter:@" tts.pcm"
 forKey: [IFlySpeechConstant TTS_AUDIO_PATH]];
//启动合成会话
[_iFlySpeechSynthesizer startSpeaking: @"你好，我是科大讯飞的小燕"];
//IFlySpeechSynthesizerDelegate 协议实现
//合成结束
- (void) onCompleted:(IFlySpeechError *) error {}
//合成开始
- (void) onSpeakBegin {}
//合成缓冲进度
- (void) onBufferProgress:(int) progress message:(NSString *)msg {}
//合成播放进度
- (void) onSpeakProgress:(int) progress beginPos:(int)beginPos
endPos:(int)endPos {}
```

3.2. 离线合成

离线合成指的是语音引擎和资源放置在应用内部，不需要连接到语音云时也可以使用的语音合成服务。因此使用时，需要在应用中添加离线引擎和发音人资源。离线引擎在提供离线服务的 *SDK* 中已内置。在使用时只需要根据选择的发音人来设置对应的离线发音人资源即可。

以 *Demo* 为例：

图 3-1 *Demo* 离线合成资源



ttsres 是离线合成的引擎资源集合，*common.jet* 是基础资源，其他文件是发音人各自对应的资源。在实际使用时，*common.jet* 和发音人资源需要同时设置。发音人可以根据需要自行选择。

SDK 大小：

	引擎大小
编译前静态库	18.2 MB（静态库大小）
编译后	4.3 MB（ipa 大小）

资源大小：

	资源大小
基础资源	4.3 MB
小燕	4.1 MB
小峰	1.3 MB
小梅	1.7 MB
凯瑟琳	3.0 MB

空间大小：（不同的编译器编译后大小会有不同，请以实际为准）

ipa 文件大小 = 引擎编译后（4.3MB） + 基础资源（4.3MB） + 所选择的发音人资源（如：小燕 4.1MB）。

下面代码是集成本地资源时需要添加的部分，其他代码与在线一致。

```
//设置语音合成的启动参数
[[IFlySpeechUtility getUtility] setParameter:@"tts"
forKey:[IFlyResourceUtil ENGINE_START]];
//获得语音合成的单例
_iFlySpeechSynthesizer = [IFlySpeechSynthesizer sharedInstance];
//设置协议委托对象
_iFlySpeechSynthesizer.delegate = self;
//设置本地引擎类型
[_iFlySpeechSynthesizer setParameter:[IFlySpeechConstant TYPE_LOCAL]
forKey:[IFlySpeechConstant ENGINE_TYPE]];
//设置发音人为小燕
[_iFlySpeechSynthesizer setParameter:@"xiaoyan"
forKey:[IFlySpeechConstant VOICE_NAME]];
//获取离线语音合成发音人资源文件路径。以发音人小燕为例，请确保资源文件的存在。
NSString *resPath = [[NSBundle mainBundle] resourcePath];
NSString *vcnResPath = [[NSString alloc]
initWithFormat:@"%@/ttsres/common.jet;%@/tts64res/xiaoyan.jet",resPath,
resPath];
//设置离线语音合成发音人资源文件路径
[_iFlySpeechSynthesizer setParameter:vcnResPath forKey:@"tts_res_path"];
```

4. 语音听写

IFlySpeechRecognizer 是不带界面的语音听写控件，*IFlyRecognizerView* 是带界面的控件，此处

仅介绍不带界面的语音听写控件。使用示例如下所示：

```
//需要实现 IFlyRecognizerViewDelegate 识别协议
@interface IATViewController :
UIViewController<IFlySpeechRecognizerDelegate>
//不带界面的识别对象
@property (nonatomic, strong) IFlySpeechRecognizer *iFlySpeechRecognizer;
@end

//创建语音识别对象
_iFlySpeechRecognizer = [IFlySpeechRecognizer sharedInstance];
//设置识别参数
//设置为听写模式
[_iFlySpeechRecognizer setParameter: @"iat" forKey: [IFlySpeechConstant
IFLY_DOMAIN]];
//asr_audio_path 是录音文件名，设置 value 为 nil 或者为空取消保存，默认保存目录在
Library/cache 下。
[_iFlySpeechRecognizer setParameter:@"iat.pcm" forKey:[IFlySpeechConstant
ASR_AUDIO_PATH]];
//启动识别服务
[_iFlySpeechRecognizer start];

//IFlySpeechRecognizerDelegate 协议实现
//识别结果返回代理
- (void) onResults:(NSArray *) results isLast:(BOOL)isLast{}
//识别会话结束返回代理
- (void)onError: (IFlySpeechError *) error{}
//停止录音回调
- (void) onEndOfSpeech{}
//开始录音回调
- (void) onBeginOfSpeech{}
//音量回调函数
- (void) onVolumeChanged: (int)volume{}
//会话取消回调
- (void) onCancel{}
```

5. 语义理解 - 开放平台接口

请注意开发平台语义接口与 AIUI 语义接口的区别，后台的接入地址有所不同，请详细阅读。

使用开放平台语义理解前需要首先确保对应的*appid*已经开通语义功能，开通地址：（*http://www.xfyun.cn/services/osp*）。语音识别结果请参照《语义开放平台API规范文档》。使用示例如下所示：

```
//需要实现 IFlySpeechRecognizerDelegate 协议
@interface UnderstandViewController :
UIViewController<IFlySpeechRecognizerDelegate>
//语义理解对象
@property (nonatomic, strong)IFlySpeechUnderstander
*_iFlySpeechUnderstander;
@end

//获得语义理解单例
_iFlySpeechUnderstander = [IFlySpeechUnderstander sharedInstance];
//设置委托
_iFlySpeechUnderstander.delegate = self;

//自定义场景设置，请参照 aiui 开放平台自创建的 scene，此参数默认为 main，如无自定义，
不用执行这句代码。
[_iFlySpeechUnderstander setParameter:@"自定义场景字符串" forKey:@"scene"];

//启动语义理解服务
[_iFlySpeechUnderstander startListening];

//语义理解实现 IFlySpeechRecognizerDelegate 协议
//结果返回代理
- (void) onResults:(NSArray *) results isLast:(BOOL) isLast{}
//会话结束回调
- (void) onError:(IFlySpeechError*) error{}
//音量回调
- (void) onVolumeChanged: (int)volume{}
//录音开始回调
```

6. 语义理解 - AIUI 接口

使用 AIUI 语义理解前需要首先确保对应的 *appid* 已经开通语义功能，开通地址：（*http://aiui.xfyun.cn/apps/index*）。

```
#pragma mark - aiui 发送数据
//aiui 发送数据
void AiuiSendBuffer(const void *buffer ,int size , bool isEnd)
{
    //是否最后一次数据
    if(isEnd){

        //发送结束标记
        IAIUIMessage * stopWrite =
IAIUIMessage::create(AIUIConstant::CMD_STOP_WRITE,
                                                    0, 0,
"data_type=audio,sample_rate=16000");
        m_angent->sendMessage(stopWrite);

        stopWrite->destroy();
    }
    else
    {
        //发送音频数据
        Buffer* pcmBuffer = Buffer::alloc(size);
        memcpy(pcmBuffer->data(), buffer, size);

        //msc.lng 和 msc.lat 的值为经纬度信息，发送数据时需要传入，请参照 demo 的获取示例
        IAIUIMessage * writeMsg =
IAIUIMessage::create(AIUIConstant::CMD_WRITE,0, 0,
"data_type=audio,sample_rate=16000,msc.lng=117.13825,msc.lat=31.83365
", pcmBuffer);
        m_angent->sendMessage(writeMsg);
        writeMsg->destroy();
    }

}
```

6.1. 语义结果解析

```
void TestListener::onEvent(IAIUIEvent& event)
{
    switch (event.getEventType()) {
        //解析结果
        case AIUIConstant::EVENT_RESULT:
        {
            using namespace VA;
            Json::Value bizParamJson;
            Json::Reader reader;
            if(!reader.parse(event.getInfo(), bizParamJson,false)){
                NSLog(@"parse error!,getinfo=%s",event.getInfo());
            }
            Json::Value data = (bizParamJson["data"])[0];
            Json::Value params = data["params"];
            Json::Value content = (data["content"])[0];
            std::string sub =  params["sub"].asString();
            if(sub == "nlp"){
                Json::Value empty;
                Json::Value contentId = content.get("cnt_id", empty);
                if(contentId.empty()){
                    NSLog(@"Content Id is empty");
                    break;
                }
                std::string cnt_id = contentId.asString();
                Buffer *buffer = event.getData()->getBinary(cnt_id.c_str());
                if(NULL != buffer){

                    const char * resultStr = (char *) buffer->data();
                    if(resultStr == NULL){
                        return;
                    }
                    NSLog(@"resultStr=%s",resultStr);
                }
            }
        } break;
            //错误处理
        case AIUIConstant::EVENT_ERROR:
        {
```

7. 个性化识别

个性化识别适用于语音听写，语义理解。支持上传联系人和上传词表功能，上传联系人可以增加通讯录联系人的识别率，上传词表可以增加词表的识别率。

7.1. 上传联系人

上传联系人，可以提升联系人的识别率，并在语义理解的打电话、发短信业务中亦生效。使用示例如下所示：

```
//创建上传对象
_uploader = [[IFlyDataUploader alloc] init];
//创建联系人对象
IFlyContact *iFlyContact = [[IFlyContact alloc] init];
NSString *contactList = [iFlyContact contact]; //获取联系人列表
//设置上传参数
[_uploader setParameter:@"uup" forKey:@"sub"];
[_uploader setParameter:@"contact" forKey:@"dtt"];
//启动上传
[_uploader uploadDataWithCompletionHandler:^(NSString *grammerID,
IFlySpeechError *error){
        //
    } name:@"contact" data: contactList];
```

7.2. 上传用户词表

上传的用户词表在语音听写中优先识别。使用示例如下所示：

```
//创建上传对象
_uploader = [[IFlyDataUploader alloc] init];
//用户词表
#define USERWORDS   @"{\"userword\":[{\"name\":\"iflytek\",\"words\":[\"德国盐猪手\",\"1912 酒吧街\",\"清蒸鲈鱼\",\"挪威三文鱼\",\"黄埔军校\",\"横沙牌坊\",\"科大讯飞\"]}]}"
IFlyUserWords *iFlyUserWords = [[IFlyUserWords alloc] initWithJson:USERWORDS ];
//设置上传参数
[_uploader setParameter:@"uup" forKey:@"sub"];
[_uploader setParameter:@"userword" forKey:@"dtt"];
//启动上传（请注意 name 参数的不同）
[_uploader uploadDataWithCompletionHandler:^(NSString * grammerID, IFlySpeechError *error){
      //
   }name: @"userwords" data:[iFlyUserWords toString]];
```

8. 语法识别

语法识别是基于语法文件的一种命令词识别技术。在线语法识别基于 *abnf* 语法文件；离线语法基于 *bnf* 语法文件。语法文件可以参照 *Demo* 的工程所示。

8. 1. 在线识别

8.1.1. 应用级命令词识别

使用浏览器访问语音云开放平台（http://www.xfyun.cn）。在打开的页面中，点击“产品服务”、“在线命令词识别”。如下图所示。

图 8-1 应用级命令词识别设置

在随后打开的页面中，点击“使用服务”，选择应用，点击“确定”，即可打开应用级在线语法文件上传页面，如下图所示。上传所需的语法文件，待页面提示“语法文件已生效”，则应用级在线语法文件启用成功。

图 8-2 应用级命令词识别设置

使用示例如下所示：

```
//获取识别对象单例
_iFlySpeechRecognizer = [IFlySpeechRecognizer sharedInstance];
//设置协议委托对象
_iFlySpeechRecognizer.delegate = self;
//设置在线识别参数
//设置引擎类型，cloud 或者 local
[_iflySpeechRecognizer setParameter:@"cloud" forKey:[IFlySpeechConstant
ENGINE_TYPE]];
//设置服务类型为 asr 识别
[_iflySpeechRecognizer setParameter:@"asr" forKey:[IFlySpeechConstant
IFLY_DOMAIN]];

//启动识别
[_iFlySpeechRecognizer startListening];

//识别 IFlySpeechRecognizerDelegate 协议
//本地和在线的识别返回代理是一致
//在切换在线和离线服务时还需要注意参数的重置,具体可以参照 demo 所示
//结果返回代理
- (void) onResults:(NSArray *) results isLast:(BOOL) isLast{}
//会话结束回调
- (void) onError:(IFlySpeechError*) error{}
//录音音量回调
- (void) onVolumeChanged: (int)volume{}
//录音开始回调
- (void) onBeginOfSpeech{}
//录音结束回调
- (void) onEndOfSpeech{}
//会话取消回调
- (void) onCancel{}
```

8.1.2. 终端级命令词识别

终端级在线命令词识别需要先在终端上构建语法文件，上传语法文件之后获得相应的 *Grammer ID*，以后每次使用识别功能前，设置该 *Grammar ID* 参数即可。其示例代码如下：

```
//获取识别对象单例
_iFlySpeechRecognizer = [IFlySpeechRecognizer sharedInstance];
//设置协议委托对象
_iFlySpeechRecognizer.delegate = self;
//设置在线识别参数
//开启候选结果
[_iflySpeechRecognizer setParameter:@"1" forKey:@"asr_wbest"];
//设置引擎类型，cloud 或者 local
[_iflySpeechRecognizer setParameter:@"cloud" forKey:[IFlySpeechConstant
ENGINE_TYPE]];
//设置字符编码为 utf-8
[_iflySpeechRecognizer setParameter:@"utf-8" forKey:[IFlySpeechConstant
TEXT_ENCODING]];
//语法类型，本地是 bnf，在线识别是 abnf
[_iflySpeechRecognizer setParameter:@"abnf" forKey:[IFlyResourceUtil
GRAMMARTYPE]];
//设置服务类型为 asr 识别
[_iflySpeechRecognizer setParameter:@"asr" forKey:[IFlySpeechConstant
IFLY_DOMAIN]];

//编译语法文件，注意 grammerType 参数的区别
//读取本地 abnf 语法文件内容
NSString* grammerContent = [self readFile:_abnfFilePath];
//调用构建语法接口
[_iflySpeechRecognizer buildGrammarCompletionHandler:^(NSString *
grammerID, IFlySpeechError *error){
      //设置 grammerID
      [_iFlySpeechRecognizer setParameter:grammerID
forKey:[IFlySpeechConstant CLOUD_GRAMMAR]];
   }grammarType:@”abnf” grammarContent:grammarContent];

//启动识别
[_iFlySpeechRecognizer startListening];

//识别 IFlySpeechRecognizerDelegate 协议
//本地和在线的识别返回代理是一致
//在切换在线和离线服务时还需要注意参数的重置,具体可以参照 demo 所示
//结果返回代理
- (void) onResults:(NSArray *) results isLast:(BOOL) isLast{}
//会话结束回调
- (void) onError:(IFlySpeechError*) error{}
//录音音量回调
- (void) onVolumeChanged: (int)volume{}
```

```
//录音开始回调
- (void) onBeginOfSpeech{}
//录音结束回调
- (void) onEndOfSpeech{}
//会话取消回调
- (void) onCancel{}
```

8.2. 离线命令词识别

引擎大小：

	引擎大小
编译前静态库	24.5 MB
编译后	8 MB

资源大小：

	资源大小
离线命令词识别资源	5.0 MB

空间大小：（不同的编译器编译后大小会有不同，请以实际为准）

ipa 文件大小 = 引擎编译后（8MB） +资源（5MB）= 13MB。

```
//设置本地识别参数，其他参数与在线方式一致
//设置引擎类型，cloud 或者 local
[_iflySpeechRecognizer setParameter:@ "local" forKey:[IFlySpeechConstant
ENGINE_TYPE]];
//语法类型，本地是 bnf，在线识别是 abnf
[_iflySpeechRecognizer setParameter:@ "bnf" forKey:[IFlyResourceUtil
GRAMMARTYPE]];
//设置引擎资源文件路径，如 demo 中的 aitalkResource 中的 common.mp3
NSString *aitalkResourcePath = [[NSString alloc]
initWithFormat:@"fo|%@/aitalkResource/common.mp3",appPath];
[_iflySpeechRecognizer setParameter:aitalkResourcePath
forKey:[IFlyResourceUtil ASR_RES_PATH]];

//启动 asr 识别引擎
[[IFlySpeechUtility getUtility] setParameter:@"asr"
forKey:[IFlyResourceUtil ENGINE_START]];

//编译语法文件（注意 grammarType 参数的区别）
//读取本地 bnf 语法文件内容
NSString* grammarContent = [self readFile:_bnfFilePath];
//调用语法编译接口
[_iflySpeechRecognizer buildGrammarCompletionHandler:^(NSString *
```

```
grammerID, IFlySpeechError *error){
        //设置 grammerID
        [_iFlySpeechRecognizer setParameter:grammerID
forKey:[IFlySpeechConstant LOCAL_GRAMMAR]];
   }grammarType:@”bnf” grammarContent:grammarContent];
//启动识别
[_iFlySpeechRecognizer startListening];

//识别 IFlySpeechRecognizerDelegate 协议
//本地和在线的识别返回代理是一致
//在切换在线和离线服务时还需要注意参数的重置,具体可以参照 demo 所示
//结果返回代理
- (void) onResults:(NSArray *) results isLast:(BOOL) isLast{}
//会话结束回调
- (void) onError:(IFlySpeechError*) error{}
//录音音量回调
- (void) onVolumeChanged: (int)volume{}
//录音开始回调
- (void) onBeginOfSpeech{}
//录音结束回调
- (void) onEndOfSpeech{}
//会话取消回调
- (void) onCancel{}
```

9. 声纹识别

声纹识别，主要是提供基于用户声纹特征的注册、验证服务，语音云平台支持 2 种类型的声纹密码类型，即文本密码和数字密码，在注册时需要指定声纹类型。

```
//创建声纹对象
isvRec=[IFlyISVRecognizer sharedInstance];
isvRec.delegate=self;

//设置声纹工作参数
//设置密码类型，pwdt 的取值为 1、3，分别表示文本密码和数字密码
[isvRec setParameter:[NSString stringWithFormat:@"%d",pwdt]
forKey:@"pwdt"];
```

pwdt 的取值说明如下表所示：

表 9-1 pwdt 取值与声纹服务类型

pwdt 取值	说明
1	文本密码。用户通过读出指定的文本内容来进行声纹注册和验证，现阶段支持的文本只有“芝麻开门”一种。
3	数字密码。从云端拉取一组特定的数字串（共分 5 组，每组 8 位数字），用户依次读出这 5 组数字进行注册，在验证过程中会生成一串特定的数字，用户通过朗读这串数字进行验证。

密码内容需调用接口从云端获取：

```
//通过调用 getPasswordList 方法来获取密码。
//获取密码的时候需指定声纹密码类型，pwdt 为 1 表示固定文本密码，pwdt 为 3 表示数字密码。
//getPasswordList 可以参照 demo 所示。
NSArray *tmpArray=[isvRec getPasswordList:ivppwdt];
```

获取到密码后，接下来进行声纹注册，即要求用户朗读若干次指定的内容，这一过程也称为声纹模型的训练。

```
// 设置业务类型为训练
[isvRec setParameter:@"train"  forKey:@"sst"];
// 设置密码类型
[isvRec setParameter:[NSString stringWithFormat:@"%d",pwdt]
forKey:@"pwdt"];
// 对于文本密码和数字密码，必须设置密码的文本内容.
// ptxt 的取值为"我的地盘我做主"、"移动改变生活"、"芝麻开门"或者是从云端拉取的数字密码(每 8 位用"-"隔开)。
[isvRec setParameter:ptxt forKey:@"ptxt"];
// 设置声纹对应的 auth_id，它用于标识声纹对应的用户
[isvRec setParameter:auth_id forKey:@"auth_id"];
// 设置有效录音时间
[isvRec setParameter:@"3000" forKey:@"vad_timeout"];
// 末端静音检测时间，用于检测到静音自动停止录音
[isvRec setParameter:@"700" forKey:@"vad_speech_tail"];

// 启动训练服务
// 开始注册,当得到注册结果时,SDK 会将其封装成 NSDictionary 对象,回调 onResult 方法进行处理，处理方法详见 Demo 示例
[isvRec startListening];

// 声纹协议 IFlyISVDelegate 实现
//会话结果返回回调
-(void) onResult:(NSDictionary *)dic;
```

```
//会话结束回调
-(void) onError:(IFlySpeechError *) errorCode;
//结果处理中回调
-(void) onRecognition;
//录音音量改变回调
-(void) onVolumeChanged: (int)volume;
```

推荐在注册声纹模型时每个用户都指定一个唯一的 *auth_id*。*auth_id* 的格式为：6-18 个字符，为字母、数字和下划线的组合且必须以字母开头，不支持中文字符，不能包含空格。

开发者通过重写 *onResult* 方法来处理注册和验证结果。在结果 *result* 中携带错误码，用来判别注册是否成功以及出错原因，部分错误码的含义如下表所示：

表 9-2 *onResult* 返回的部分错误码说明

错误码	说明
10106	缺少某个必要参数
10107	某个必要参数存在但无效
10110	引擎授权不足或者说授权客户端用户数达到上限
10114	操作超时
10116	数据库中模型不存在
10212	数据库中模型已经存在
10400	数据库中一般性错误，此时此 ssb 已经登录超过 3 次且均失败
10407	APPID 非法
10606	音频太短

1） 声纹验证

声纹验证过程与声纹注册类似，不同之处仅在于“*sst*”参数需要设置为“*verify*”，其他参数的设置、验证结果的处理过程可参考上一节。

另外，为了达到较好的效果，请在声纹注册与验证过程中尽量与麦克风保持同样的距离（建议的最佳距离是 15 厘米左右）。若距离较远，可能会对验证通过率产生较大影响。

2） 声纹模型操作

声纹注册成功后，在云端会生成一个对应的模型来存储声纹信息，声纹模型的操作即对模型进行查询和删除。

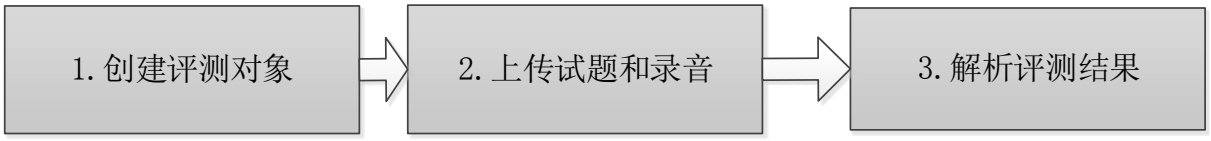
```
//开发者调用 sendRequest 方法查询或者删除模型，该函数的定义如下：
//cmd: @"query"表示查询，@"del"表示删除
//auth_id 表示用户名;
//pwdt 表示声纹类型;
//ptxt 表示查询或者删除的密码文本;
//vid 是用户注册成功后服务器返回的 32 位标识，查询和删除时，vid 可以设置位 nil;
//err 是查询的错误码。 通常查询或者删除成功，该函数会返回 YES,否则返回 NO;
-(BOOL) sendRequest:(NSString*)cmd authid:(NSString *)auth_id
pwdt:(int)pwdt ptxt:(NSString *)ptxt vid:(NSString *)vid err:(int
*)err;
```

3） 声纹业务返回结果格式和参数说明
参见附录 13.3 声纹业务

10. 语音评测

提供汉语、英语两种语言的评测，支持单字（汉语专有）、词语和句子朗读三种题型，通过简单地接口调用就可以集成到您的应用中。语音评测的使用主要有三个步骤：

图 10-1 评测流程




```
// 获取评测对象单例
_iFlySpeechEvaluator = [IFlySpeechEvaluator sharedInstance];
_iFlySpeechEvaluator.delegate = self;
// 设置训练参数
// 清空参数
[_iFlySpeechEvaluator setParameter:@"" forKey:[IFlySpeechConstant
PARAMS]];
// 设置评测采样率
[self.iFlySpeechEvaluator setParameter:@"16000" forKey:[IFlySpeechConstant
SAMPLE_RATE]];
// 设置评测题目编码，如果是 utf-8 格式，请添加 bom 头，添加方式可参考 demo。
[self.iFlySpeechEvaluator setParameter:@"utf-8" forKey:[IFlySpeechConstant
TEXT_ENCODING]];
// 设置评测题目结果格式，目前仅支持 xml
[self.iFlySpeechEvaluator setParameter:@"xml" forKey:[IFlySpeechConstant
ISE_RESULT_TYPE]];
// 设置评测前端点超时
[self.iFlySpeechEvaluator setParameter:self.iseParams.bos
forKey:[IFlySpeechConstant VAD_BOS]];
// 设置评测后端点超时
[self.iFlySpeechEvaluator setParameter:self.iseParams.eos
forKey:[IFlySpeechConstant VAD_EOS]];
// 设置评测题型
[self.iFlySpeechEvaluator setParameter:self.iseParams.category
forKey:[IFlySpeechConstant ISE_CATEGORY]];
// 设置评测语言
[self.iFlySpeechEvaluator setParameter:self.iseParams.language
```

```
forKey:[IFlySpeechConstant LANGUAGE]];
// 设置评测结果级别
[self.iFlySpeechEvaluator setParameter:self.iseParams.rstLevel
forKey:[IFlySpeechConstant ISE_RESULT_LEVEL]];
// 设置评测超时
[self.iFlySpeechEvaluator setParameter:self.iseParams.timeout
forKey:[IFlySpeechConstant SPEECH_TIMEOUT]];
```

可通过 setParameter 设置的评测相关参数说明如下：

表 10-1 评测相关参数说明

参数	说明	是否必需
language	评测语种，可选值：*en_us*（英语）、*zh_cn*（汉语）	是
category	评测题型，可选值：*read_syllable*（单字，汉语专有）、*read_word*（词语）、*read_sentence*（句子）	是
text_encoding	上传的试题编码格式，可选值：*gb2312*、*utf-8*。当进行汉语评测时，必须设置成 *utf-8*，建议所有试题都使用 *utf-8* 编码	是
vad_bos	前端点超时，默认 *5000ms*	否
vad_eos	后端点超时，默认 *1800ms*	否
speech_timeout	录音超时，当录音达到时限将自动触发 *VAD* 停止录音，默认-1（无超时）	否
result_level	评测结果等级，可选值：*plain*（仅英文）、*complete*，默认为 *complete*	否

实现协议

```
//语音评测实现 Delegate
// 音量和数据回调
- (void)onVolumeChanged:(int)volume buffer:(NSData *)buffer{}
// 开始录音回调
- (void)onBeginOfSpeech{}
// 停止录音回调
- (void)onEndOfSpeech{}
// 会话取消回调
- (void)onCancel{}
// 评测错误回调
- (void)onError:(IFlySpeechError *)errorCode{}
// 评测结果回调
- (void)onResults:(NSData *)results isLast:(BOOL)isLast{}
```

调用 *startListening* 即开始评测录音，读完试题内容后可以调用 *stopListening* 停止录音，也可以在一段时间后由 *SDK* 自动检测 *VAD* 并停止录音。当评测出错时，*SDK* 会回调 *onError* 方法抛出 *IFlySpeechError* 错误，通过 *IFlySpeechError* 的 *getErrorCode*()方法可获得错误码，常见的错误码详见附录： 错误码列表和下表：

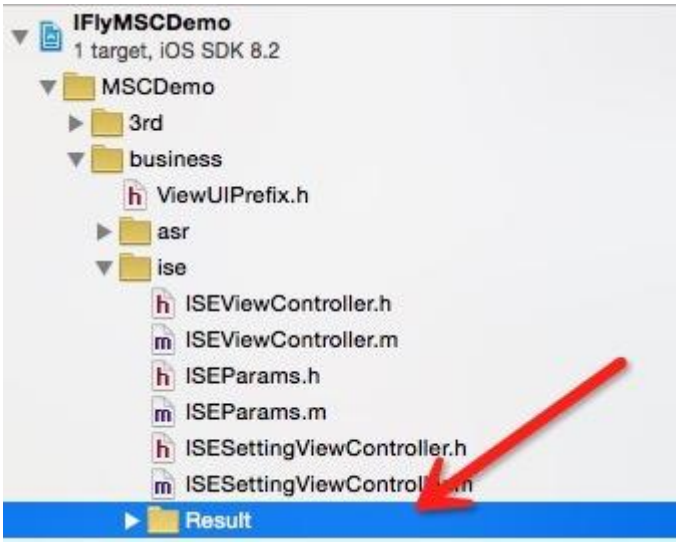
表 10-2 评测错误码

错误码	错误值	含义
MSP_ERROR_ASE_EXCEP_SILENCE	11401	无语音或音量太小
MSP_ERROR_ASE_EXCEP_SNRATIO	11402	信噪比低或有效语音过短
MSP_ERROR_ASE_EXCEP_PAPERDATA	11403	非试卷数据
MSP_ERROR_ASE_EXCEP_PAPERCONTENTS	11404	试卷内容有误
MSP_ERROR_ASE_EXCEP_NOTMONO	11405	录音格式有误
MSP_ERROR_ASE_EXCEP_OTHERS	11406	其他评测数据异常，包括错读、漏读、恶意录入、试卷内容等错误
MSP_ERROR_ASE_EXCEP_PAPERFMT	11407	试卷格式有误
MSP_ERROR_ASE_EXCEP_ULISTWORD	11408	存在未登录词，即引擎中没有该词语的信息

解析评测结果

SDK 通过 *onResult* 回调抛出 *XML* 格式的评测结果，结果格式及字段含义详见《语音评测参数、结果说明文档》文档，具体解析过程可参考 *Demo* 工程 *IFlyMscDemo* 中 *ISE* 目录下 *Result* 目录中的源代码。

图 10-2 语音评测示例代码位置

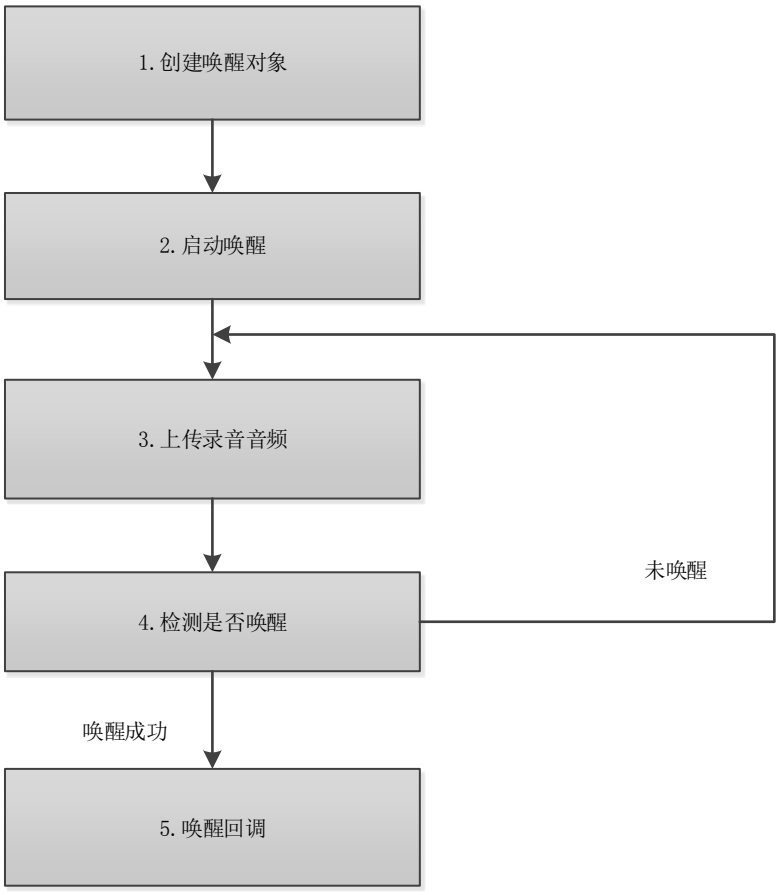


11. 语音唤醒

11.1. 语音唤醒

语音唤醒集成流程如下。

图 11-1 语音唤醒流程



使用示例如下所示：(*注意，启动唤醒需要关注系统录音权限，目前 iOS 唤醒只支持应用级唤醒。*)

```
#import "iflyMSC/IFlyMSC.h"
//获取唤醒单例对象
_iflyVoiceWakeuper = [IFlyVoiceWakeuper sharedInstance];
//设置唤醒协议委托
_iflyVoiceWakeuper.delegate = self;
//设置唤醒参数
//生成唤醒资源路径，唤醒资源需要定制，与唤醒词一一对应。
NSString *wordPath = [[NSBundle mainBundle]
pathForResource:@"ivwres/wakeresource" ofType:@"irf"];
NSString *ivwResourcePath = [[NSString alloc]
initWithFormat:@"fo|%@",wordPath];
//设置唤醒资源，并启动唤醒引擎
 [[IFlySpeechUtility getUtility] setParameter:[NSString stringWithFormat:
@"engine_start=ivw,ivw_res_path=%@",ivwResourcePath]
forKey:[IFlyResourceUtil ENGINE_START]];
//设置唤醒门限值
//门限设置要和资源中的唤醒词个数匹配，以;分割。
//例如：0:-20，0 代表第一个唤醒词 -20，代表第一个唤醒词门限
```

```
//根据下载的 SDK 中的说明来设置。
//0：表示第一个唤醒词，-20 表示唤醒词对应的门限值；
//1：表示第二个唤醒词，-20 表示唤醒词对应的门限值
[_iflyVoiceWakeuper setParameter:@"0:-20;1:-20;"
forKey:@"ivw_threshold"];

//设置唤醒的服务类型，目前仅支持 wakeup
[_iflyVoiceWakeuper setParameter:@"wakeup" forKey:@"ivw_sst"];
//设置唤醒的工作模式
//keep_alive 表示一次唤醒成功后是否继续录音等待唤醒。1：表示继续；0：表示唤醒终止
[_iflyVoiceWakeuper setParameter:@"1" forKey:@"keep_alive"];
//启动唤醒
int bRet = [self.iflyVoiceWakeuper startListening];
//唤醒实现 delegate
//录音开始
-(void) onBeginOfSpeech{}
//录音结束
-(void) onEndOfSpeech{}
//会话错误
-(void) onError:(IFlySpeechError *)error{}
//音量变化回调
- (void) onVolumeChanged: (int)volume{}
//唤醒结果回调
-(void) onResult:(NSMutableDictionary *)resultArray{}
```

11.2. 语音唤醒 *Oneshot*

oneshot 是唤醒的一种扩展方式，支持唤醒+识别，唤醒+听写，唤醒+语义的组合解决方案。

此处介绍唤醒+识别模式，可以适用于唤醒词数量有限，而命令词又无法常驻运行的场景。在游戏场景下，可采用声控的方式来操控游戏。例如：定义唤醒词为阿里巴巴，定义命令词序列：“打开城门”，“发起攻击”，这样就形成若干命令序列。用户可以说：

- 阿里巴巴打开城门；
- 阿里巴巴发起攻击。

除了游戏外，用户可以很容易的将唤醒+识别的应用扩展到其他领域，比如家电，车载等比较适合语音的场所。

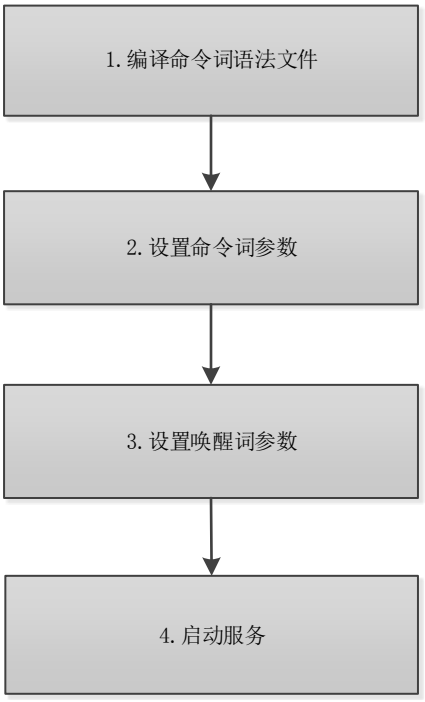
1) 详细 demo

唤醒+识别的代码比较复杂，限于篇幅不在本文档描述，详细过程可以参考 *Demo* 中的 *OneshotViewController* 示例代码。

Demo 的示例，用户可以说 讯飞语音/讯飞语点 + 张三/李四/张海洋 的组合。

2) 使用唤醒+识别步骤

图 11-2 唤醒+识别使用流程



3) 唤醒+识别回调

```
//录音开始
- (void) onBeginOfSpeech{}
//录音结束
- (void) onEndOfSpeech{}
//音量回调
- (void) onVolumeChanged: (int)volume{}
//服务结束回调
- (void) onError:(IFlySpeechError *) error{}
//唤醒结果回调
- (void) onResult:(NSMutableDictionary *)resultArray {}
//会话事件回调
- (void) onEvent:(int)eventType isLast:(BOOL)isLast arg1:(int)arg1
data:(NSMutableDictionary *)eventData(){}
```

12. 人脸识别

人脸识别可以检测出照片中的人脸和关键点。相关概念的说明如下：

表 12-1 人脸识别概念说明

名称	含义	说明
detect	检测	上传一张图片，返回该图片中人脸的位置（支持多张人脸）。
align	聚焦	上传一张图片，返回该图片中人脸的关键点坐标（支持多张人脸）。

为了获得较高的准确率，请确保输入的图片满足以下要求：

表 12-2 上传图片规格要求

项目	要求
色彩、格式	彩色，PNG、JPG、BMP 格式的图片。
人脸大小、角度	大小应超过 100*100 像素，可以容忍一定程度的侧脸，为保证识别准确率，最好使用正脸图片。
光照	均匀光照，可容忍部分阴影。
遮挡物	脸部尽量无遮挡，眼镜等物品会一定程度上影响准确率。

12.1. 人脸检测

人脸检测返回人脸所在的矩形框位置,示例代码如下：

```
// 设置人脸检测参数
[self.iFlySpFaceRequest setParameter:[IFlyFaceConstant DETECT]
forKey:[IFlyFaceConstant SST]];
[self.iFlySpFaceRequest setParameter:USER_APPID forKey:[IFlyFaceConstant
APPID]];
[self.iFlySpFaceRequest sendRequest:imgData];

// 实现 IFlyFaceRequestDelegate 协议
// 消息回调
- (void) onEvent:(int) eventType WithBundle:(NSString*) params{}
// 数据回调，可能调用多次，也可能一次不调用
- (void) onData:(NSData* )data{}
// 结束回调，没有错误时，error 为 nil
- (void) onCompleted:(IFlySpeechError*) error{}
```

12.2. 人脸聚焦

人脸聚焦返回人脸关键点的位置,示例代码如下：

```
// 设置人脸聚焦参数
[self.iFlySpFaceRequest setParameter:[IFlyFaceConstant ALIGN]
forKey:[IFlyFaceConstant SST]];
[self.iFlySpFaceRequest setParameter:USER_APPID forKey:[IFlyFaceConstant
APPID]];
[self.iFlySpFaceRequest sendRequest:imgData];

// 实现 IFlyFaceRequestDelegate 协议
// 消息回调
- (void) onEvent:(int) eventType WithBundle:(NSString*) params{}
// 数据回调，可能调用多次，也可能一次不调用
- (void) onData:(NSData* )data{}
// 结束回调，没有错误时，error 为 nil
- (void) onCompleted:(IFlySpeechError*) error{}
```

13. 身份验证（声纹+人脸）

多生物特征融合验证（*Multi-biometrics Fusion Verification*，简称 *MFV*）平台是科大讯飞推出的新一代个人身份验证平台。功能上支持 **1：1 单一验证，1：1 融合验证，1：N 鉴别**。生物特征上支持**人脸、声纹**，并将在未来支持指纹、虹膜等其他特征信息。

13.1. 基本介绍

MFV 目前提供的功能组合如表 12-1 所示。

表 13-1 MFV 功能组合说明

功能 / 生物特征	1：1 单一验证	1：1 融合验证	1：N 鉴别
人脸	人脸验证	人脸声纹融合验证	人脸鉴别
声纹	声纹验证		声纹鉴别

MFV 相关的概念如表 12-2 所示。

表 13-2 MFV 相关概念说明

分类	概念	英文标识	说明
id 相关	用户 *id*	*auth_id*	用户身份的唯一标识。

	组 *id*	*group_id*	组的唯一标识。 组被用来限定 1：N 鉴别的用户范围。
功能相关	特征注册	*enroll*	上传用户特征数据，在云端生成特征模型。 其中，人脸图像数据的大小应控制在 *200K* 以下。
	特征验证	*verify*	上传用户特征数据，云端将其与已注册的特征模型进行比对，返回结果（相似度、是否通过验证等）。
	融合验证	*mixed verify*	上传用户多项特征数据，云端将其与已注册的多项特征模型进行比对，返回结果（综合相似度、是否通过验证等）。
	特征鉴别	*identify*	上传用户特征数据，并指定鉴别组 *id*，云端将上传数据与组内用户对应的已注册的特征模型进行比对，返回结果（相似度排行、用户名称）。
会话相关	业务场景	*scenes*	会话的场景。 包括：人脸（*ifr*），声纹（*ivp*），人脸声纹融合（*ifr*|*ivp*），组管理（*ipt*）。
	业务类型	*sst*	会话的业务类型。 在不同的会话场景（*scenes*）下有不同的业务类型，详情见表 12-3。
	子业务类型	*ssub*	子业务类型。 包括：人脸（*ifr*），声纹（*ivp*），组管理（*ipt*）。

表 13-3 MFV 业务场景与业务类型组合

业务场景 / 业务类型	人脸（*ifr*）	声纹（*ivp*）	人脸声纹融合（*ifr*|*ivp*）	组管理（*ipt*）
注册	√	√		
验证	√	√	√	
鉴别	√	√		

表 13-4 MFV 子业务操作组合

子业务类型 / 操作类型	人脸（*ifr*）	声纹（*ivp*）	组管理（*ipt*）
创建			√
加入			√
查询		√	√
删除	√	√	√

密码下载		√	

13.2. 功能使用

1：1 验证：

1) 首次使用需先进行特征注册。
2) 开始验证。设定各项参数，接着开启会话，上传待验证的生物特征数据，最后获取验证结果并解析。

1：N 鉴别：

1) 首次使用需先进行特征注册。
2) 创建组。
3) 将相关的用户 *id* 加入到组中。
4) 开始鉴别。指定鉴别组 *id* 及各项参数，接着开启会话，上传待鉴别的生物特征数据，最后获取鉴别结果并解析。

注意：身份验证鉴别功能支持创建 5000 个鉴别组及每个鉴别组支持添加 100 个成员。开发者若有更高需求，请邮件联系：msp_support@iflytek.com

13.2.1. 特征注册

人脸注册流程：

1) 设置参数：业务场景“人脸（*ifr*）”，业务类型“注册（*enroll*）”，用户 *id*（*auth_id*）。
2) 设置监听，开启会话。
3) 指定子业务类型“人脸（*ifr*）”、人脸数据内容及数据长度，然后上传数据。
4) 待会话监听器返回结果，此次注册操作结束。

人脸注册示例代码：

```
// 取消上次会话、清空参数
[self.identityVerifier cancel];
[self.identityVerifier setParameter:nil forKey:[IFlySpeechConstant
PARAMS]];
// 人脸参数
// 设置 sub 参数请求业务类型
[self.identityVerifier setParameter:@"mfv" forKey:[IFlySpeechConstant
MFV_SUB]];
// 设置 scenes，
// 有“ifr（人脸）”，“ivp（声纹）”，“ifr|ivp（人脸和声纹）”三种取值，
// 指明注册的特征种类
[self.identityVerifier setParameter:@"ifr" forKey:[IFlySpeechConstant
MFV_SCENES]];
```

```
NSString* auth_id=self.authIdLabel.text;
[self.identityVerifier setParameter:@"enroll" forKey:[IFlySpeechConstant
MFV_SST]];
// 设置 delegate ,auth_id，开始会话
[self.identityVerifier setParameter:auth_id forKey:[IFlySpeechConstant
MFV_AUTH_ID]];
[self.identityVerifier startWorking];
// 人脸数据参数
NSString* dwtParams=[NSString
stringWithFormat:@"%@=%@,",[IFlySpeechConstant MFV_SST],@"enroll"];
dwtParams=[dwtParams
stringByAppendingFormat:@"%@=%@,",[IFlySpeechConstant
MFV_AUTH_ID],auth_id];
// 压缩人脸数据并写入
NSData* data=[self.face compressedData];
[self.identityVerifier write:@"ifr" data:data offset:0 length:(int)[data
length] withParams:dwtParams];
[self.identityVerifier stopWrite:@"ifr"];

// 回调接口定义如下，使用方法参考 demo 源码;
// 错误回调
- (void)onError:(IFlySpeechError *)error{}
// 结果回调，results 中包含有 json 格式结果，字段说明参考附录身份验证结果说明。
- (void)onResults:(IFlyIdentityResult *)results isLast:(BOOL)isLast{}
// 扩展接口，用于抛出音量和 vad_eos 消息
- (void)onEvent:(int)eventType arg1:(int)arg1 arg2:(int)arg2
extra:(id)obj{}
```

声纹注册：

1) 从云端下载声纹注册所用的数字密码文本。
2) 设置参数：业务场景“声纹（*ivp*）”，业务类型“注册（*enroll*）”，用户 *id*（*auth_id*）。
3) 设置监听，开启会话。
4) 指定子业务类型“声纹（*ifr*）”，设定声纹注册相关参数“训练次数（*rgn*）、密码内容（*ptxt*）、密码类型（*pwdt*）”，并指定声纹数据内容及长度，然后上传数据。
5) 待会话监听器返回结果，此次注册操作结束。

声纹注册示例代码：

```
// 下载声纹密码
// 取消上次会话、清空参数
[self.identityVerifier cancel];
[self.identityVerifier setParameter:nil forKey:[IFlySpeechConstant
```

```
PARAMS]];

// 设置 scenes 为“ivp（声纹）”
[self.identityVerifier setParameter:@"ivp" forKey:[IFlySpeechConstant
MFV_SCENES]];
// 设置密码参数
NSString* params=[NSString
stringWithFormat:@"%@=%d,%@=%d,",[IFlySpeechConstant
MFV_PWDT],3,[IFlySpeechConstant MFV_RGN],TOTAL_TIMES];
// 执行获取密码操作
[self.identityVerifier execute:@"ivp" cmd:@"download" params:params];

// 声纹注册
// 取消上次会话、清空参数
[self.identityVerifier cancel];
[self.identityVerifier setParameter:nil forKey:[IFlySpeechConstant
PARAMS]];
// 设置 sub 参数请求业务类型，可选值：
// mfv（默认，融合验证），ivp（声纹），ifr（人脸）
[self.identityVerifier setParameter:@"mfv" forKey:[IFlySpeechConstant
MFV_SUB]];
// 设置 scenes，
// 有“ifr（人脸）”，“ivp（声纹）”，“ifr|ivp（人脸和声纹）”三种取值，
// 指明注册的特征种类
[self.identityVerifier setParameter:@"ivp" forKey:[IFlySpeechConstant
MFV_SCENES]];
// 设置 delegate ,auth_id，开始会话
self.identityVerifier.delegate=self;
[self.identityVerifier setParameter:auth_id forKey:[IFlySpeechConstant
MFV_AUTH_ID]];
[self.identityVerifier startWorking];
// 准备声纹注册相关参数，如训练次数、密码内容和密码类型等
// 注意：注册时使用的源码需要先从语音云下载再使用，详情参见 demo 源码；
NSString* dwtParams=nil;
dwtParams=[NSString stringWithFormat:@"%@=%@,",[IFlySpeechConstant
MFV_SST],@"enroll"];
dwtParams=[dwtParams
stringByAppendingFormat:@"%@=%@,",[IFlySpeechConstant MFV_RGN],@"5"];
NSString* dlPtxt=[numberPasswords componentsJoinedByString:@"-"];
dwtParams=[dwtParams
stringByAppendingFormat:@"%@=%@,",[IFlySpeechConstant MFV_PTXT],dlPtxt];
dwtParams=[dwtParams
stringByAppendingFormat:@"%@=%@,",[IFlySpeechConstant MFV_PWDT],@"3"];
```

```
// 开启 VAD 功能 设置 VAD_BOS(前端点) VAD_EOS(后端点) 设置采样率
dwtParams=[dwtParams
stringByAppendingFormat:@"%@=%@,",[IFlySpeechConstant VAD_ENABLE],@"1"];
dwtParams=[dwtParams
stringByAppendingFormat:@"%@=%@,",[IFlySpeechConstant VAD_BOS],@"10000"];
dwtParams=[dwtParams
stringByAppendingFormat:@"%@=%@,",[IFlySpeechConstant VAD_EOS],@"2000"];
dwtParams=[dwtParams
stringByAppendingFormat:@"%@=%@,",[IFlySpeechConstant
MFV_DATA_FORMAT],@"16000"];
// 写入音频数据，data 为音频数据
// 需要开发者自行从外部录音机 IFlyPcmRecorder 获取，获取方法详
// 见 demo 源码。
// writeData 可以连续多次调用，调用的时候要指定子业务类型、子业务参数、写入
// 数据及长度
NSData *audioBuffer = [NSData dataWithBytes:buffer length:size];
[self.identityVerifier write:@"ivp" data:audioBuffer offset:0
length:(int)[audioBuffer length] withParams:dwtParams];
// 写入完成后，需要调用 stopWrite 停止写入，在停止的时候同样需要指定子业务类型。
// 只有 stopWrite 之后，才会触发 delegate 的回调接口，返回结果或者错误。
[self.identityVerifier stopWrite:@"ivp"];
// 回调接口定义如下，使用方法参考 demo 源码；
// 错误回调
- (void)onError:(IFlySpeechError *)error{}
// 结果回调，results 中包含有 json 格式结果，字段说明参考附录身份验证结果说明。
- (void)onResults:(IFlyIdentityResult *)results isLast:(BOOL)isLast{}
// 扩展接口，用于抛出音量和 vad_eos 消息
- (void)onEvent:(int)eventType arg1:(int)arg1 arg2:(int)arg2
extra:(id)obj{}
```

13.2.2. 特征验证

根据场景不同，特征验证又分为**人脸验证**、**声纹验证**以及**人脸声纹融合验证**。验证的过程和注册的过程很相似，以下是**融合验证**的示例代码。

```
// 取消上次会话、清空参数
[self.identityVerifier cancel];
[self.identityVerifier setParameter:nil forKey:[IFlySpeechConstant
PARAMS]];
// 设置 sst 为 verify
[self.identityVerifier setParameter:@"verify" forKey:[IFlySpeechConstant
MFV_SST]];
// 设置 scenes，有"ifr（人脸）"，"ivp（声纹）"，"ifr|ivp（人脸和声纹）"三种取值，指
明验证的// 特征种类，这里设置为"人脸+声纹"融合验证
[self.identityVerifier setParameter:@"ivp|ifr" forKey:[IFlySpeechConstant
MFV_SCENES]];
[self.identityVerifier setParameter:@"mix" forKey:[IFlySpeechConstant
MFV_VCM]];
// 设置 delegate 、auth_id，开始会话
self.identityVerifier.delegate=self;
[self.identityVerifier setParameter:auth_id forKey:[IFlySpeechConstant
MFV_AUTH_ID]];
[self.identityVerifier startWorking];
// 准备人脸参数
NSString* dwtParams=[NSString
stringWithFormat:@"%@=%@,",[IFlySpeechConstant MFV_AUTH_ID],auth_id];
dwtParams=[dwtParams
stringByAppendingFormat:@"%@=%@,",[IFlySpeechConstant
MFV_SST],@"verify"];
// 写入人脸照片数据，imageData 为 bitmap 格式
NSData* data= UIImageJPEGRepresentation(self.face, 1.0);
[self.identityVerifier write:@"ifr" data:data offset:0 length:(int)[data
length] withParams:dwtParams];
[self.identityVerifier stopWrite:@"ifr"];
// 准备声纹验证相关参数，如密码内容和密码类型等
NSString* dwtParams=nil;
dwtParams=[NSString stringWithFormat:@"%@=%@,",[IFlySpeechConstant
MFV_SST],@"verify"];
NSString* ptxt=self.numberLabel.text;
dwtParams=[dwtParams
stringByAppendingFormat:@"%@=%@,",[IFlySpeechConstant MFV_PTXT],ptxt];
dwtParams=[dwtParams
stringByAppendingFormat:@"%@=%@,",[IFlySpeechConstant MFV_PWDT],@"3"];
dwtParams=[dwtParams
stringByAppendingFormat:@"%@=%@,",[IFlySpeechConstant VAD_ENABLE],@"1"];
dwtParams=[dwtParams
stringByAppendingFormat:@"%@=%@,",[IFlySpeechConstant VAD_BOS],@"10000"];
```

```
dwtParams=[dwtParams
stringByAppendingFormat:@"%@=%@,",[IFlySpeechConstant VAD_EOS],@"2000"];
dwtParams=[dwtParams
stringByAppendingFormat:@"%@=%@,",[IFlySpeechConstant
MFV_DATA_FORMAT],@"16000"];
// 写入音频数据，data 为音频数据，需要开发者自行从外部录音机 PcmRecorder 获取，
// 获取方法详见 demo 源码。writeData 可以连续多次调用。
// 调用的时候要指定子业务类型、子业务参数、写入数据及长度
NSData *audioBuffer = [NSData dataWithBytes:buffer length:size];
[self.identityVerifier write:@"ivp" data:audioBuffer offset:0
length:(int)[audioBuffer length] withParams:dwtParams];
// 音频数据写入完成
[self.identityVerifier stopWrite:@"ivp"];

// 回调接口定义如下，使用方法参考 demo 源码；
// 错误回调
- (void)onError:(IFlySpeechError *)error{}
// 结果回调，results 中包含有 json 格式结果，字段说明参考附录身份验证结果说明。
- (void)onResults:(IFlyIdentityResult *)results isLast:(BOOL)isLast{}
// 扩展接口，用于抛出音量和 vad_eos 消息
- (void)onEvent:(int)eventType arg1:(int)arg1 arg2:(int)arg2
extra:(id)obj{}
```

13.2.3. 特征鉴别

根据场景不同，特征鉴别分为**人脸鉴别**和**声纹鉴别**。两种鉴别流程相同，以**人脸鉴别**为例。

```
// 取消上次会话、清空参数
[self.identityVerifier cancel];
[self.identityVerifier setParameter:nil forKey:[IFlySpeechConstant
PARAMS]];
// 设置业务场景
[self.identityVerifier setParameter:@"ifr" forKey:[IFlySpeechConstant
MFV_SCENES]];
// 设置业务类型
[self.identityVerifier setParameter:@"identify" forKey:[IFlySpeechConstant
MFV_SST]];
// 设置写入数据参数
NSString* dwtParams=[NSString
stringWithFormat:@"%@=%@,group_id=%@,topc=3",[IFlySpeechConstant
MFV_SST],@"identify",self.groupID];
```

```
// 开始会话
[self.identityVerifier startWorking];

// 向子业务写入数据，人脸数据可以一次写入
NSData* data=[self.face compressedData];
[self.identityVerifier write:@"ifr" data:data offset:0 length:(int)[data
length] withParams:dwtParams];
// 写入完毕
[self.identityVerifier stopWrite:@"ifr"];

// 回调接口定义如下，使用方法参考 demo 源码；
// 错误回调
- (void)onError:(IFlySpeechError *)error{}

// 结果回调，results 中包含有 json 格式结果，字段说明参考附录身份验证结果说明。
- (void)onResults:(IFlyIdentityResult *)results isLast:(BOOL)isLast{}

// 扩展接口，用于抛出音量和 vad_eos 消息
- (void)onEvent:(int)eventType arg1:(int)arg1 arg2:(int)arg2
extra:(id)obj{}
```

13.2.4. 模型操作

声纹模型目前支持的操作有**查询（query）**、**删除（delete）**、**密码下载（download）**三种。人脸模型目前支持**删除（delete）**操作。

声纹密码下载（download），已经在声纹模型注册的过程中展示，此处不再赘述。

以下以**声纹模型查询**为例：

```
// 取消上次会话、清空参数
[self.identityVerifier cancel];
[self.identityVerifier setParameter:nil forKey:[IFlySpeechConstant
PARAMS]];
// 设置场景为声纹
[self.identityVerifier setParameter:@"ivp" forKey:[IFlySpeechConstant
MFV_SCENES]];
// 准备声纹验证相关参数，即密码内容和密码类型
[self.identityVerifier setParameter:[NSString
stringWithFormat:@"%d",TOTAL_TIMES] forKey:[IFlySpeechConstant MFV_RGN]];
NSString* params=[NSString stringWithFormat:@"%@=%d,",[IFlySpeechConstant
MFV_PWDT],3];
```

```
[self.identityVerifier setParameter:auth_id forKey:[IFlySpeechConstant
MFV_AUTH_ID]];
// 调用 execute 方法执行操作。第一个参数为子业务类型，取值为“ivp”、“ifr”，// 第二个参数为操作类型，支持“query”（查询）、“delete”（删除）和
//“download”（密码下载）三种。
// 声纹密码下载也是通过调用 execute 方法，具体实现见 IFlyMFVDemo 源码。
[self.identityVerifier execute:@"ivp" cmd:@"query" params:params];

// 回调接口定义如下，使用方法参考 demo 源码；
// 错误回调
- (void)onError:(IFlySpeechError *)error{}
// 结果回调，results 中包含有 json 格式结果，字段说明参考附录身份验证结果说明。
- (void)onResults:(IFlyIdentityResult *)results isLast:(BOOL)isLast{}
// 扩展接口，用于抛出音量和 vad_eos 消息
- (void)onEvent:(int)eventType arg1:(int)arg1 arg2:(int)arg2
extra:(id)obj{}
```

13.2.5. 组管理

组管理目前支持的操作有：**创建组、删除组、查询组用户、用户加入组**以及**用户退出组**。**创建组**示例代码：

```
// 取消上次会话、清空参数
[self.identityVerifier cancel];
[self.identityVerifier setParameter:nil forKey:[IFlySpeechConstant
PARAMS]];
// 设置会话场景
[self.identityVerifier setParameter:@"ipt" forKey:[IFlySpeechConstant
MFV_SCENES]];
// 用户 id
NSString* authId=self.authIdLabel.text;
[self.identityVerifier setParameter:authId forKey:[IFlySpeechConstant
MFV_AUTH_ID]];

// 设置模型参数，若无可以传空字符串
NSString* params =[NSString
stringWithFormat:@"auth_id=%@,scope=group,group_name=%@",authId,self.grou
pNameText.text];
// 执行模型操作
[self.identityVerifier execute:@"ipt" cmd:@"add" params:params];

// 回调接口定义如下，使用方法参考 demo 源码；
```

```
// 错误回调
- (void)onError:(IFlySpeechError *)error;
// 结果回调，results 中包含有 json 格式结果，字段说明参考附录身份验证结果说明。
- (void)onResults:(IFlyIdentityResult *)results isLast:(BOOL)isLast{}
// 扩展接口，用于抛出音量和 vad_eos 消息
- (void)onEvent:(int)eventType arg1:(int)arg1 arg2:(int)arg2
extra:(id)obj{}
```

删除组示例代码：

```
// 取消上次会话、清空参数
[self.identityVerifier cancel];
[self.identityVerifier setParameter:nil forKey:[IFlySpeechConstant
PARAMS]];
// 设置会话场景
[self.identityVerifier setParameter:@"ipt" forKey:[IFlySpeechConstant
MFV_SCENES]];
// 用户 id
NSString* authId=self.authIdLabel.text;
[self.identityVerifier setParameter:authId forKey:[IFlySpeechConstant
MFV_AUTH_ID]];

// 设置模型参数，若无可以传空字符传
NSString* params =[NSString
stringWithFormat:@"scope=group,group_id=%@",self.groupIdText.text];
// 执行模型操作
[self.identityVerifier execute:@"ipt" cmd:@"delete" params:params];

// 回调接口定义如下，使用方法参考 demo 源码；
// 错误回调
- (void)onError:(IFlySpeechError *)error{}
// 结果回调，results 中包含有 json 格式结果，字段说明参考附录身份验证结果说明。
- (void)onResults:(IFlyIdentityResult *)results isLast:(BOOL)isLast{}

// 扩展接口，用于抛出音量和 vad_eos 消息
- (void)onEvent:(int)eventType arg1:(int)arg1 arg2:(int)arg2
extra:(id)obj{}
```

查询组用户示例代码：

```
// 取消上次会话、清空参数
[self.identityVerifier cancel];
[self.identityVerifier setParameter:nil forKey:[IFlySpeechConstant
PARAMS]];
// 设置会话场景
[self.identityVerifier setParameter:@"ipt" forKey:[IFlySpeechConstant
MFV_SCENES]];
// 用户 id
NSString* authId=self.authIdLabel.text;
[self.identityVerifier setParameter:authId forKey:[IFlySpeechConstant
MFV_AUTH_ID]];
// 设置模型参数，若无可以传空字符传
NSString* params =[NSString
stringWithFormat:@"auth_id=%@,scope=group,group_id=%@",authId,self.groupI
dText.text];
// 执行模型操作
[self.identityVerifier execute:@"ipt" cmd:@"query" params:params];

// 回调接口定义如下，使用方法参考 demo 源码；
// 错误回调
- (void)onError:(IFlySpeechError *)error{}
// 结果回调，results 中包含有 json 格式结果，字段说明参考附录身份验证结果说明。
- (void)onResults:(IFlyIdentityResult *)results isLast:(BOOL)isLast{}
// 扩展接口，用于抛出音量和 vad_eos 消息
- (void)onEvent:(int)eventType arg1:(int)arg1 arg2:(int)arg2
extra:(id)obj{}
```

用户加入组示例代码：

```
// 取消上次会话、清空参数
[self.identityVerifier cancel];
[self.identityVerifier setParameter:nil forKey:[IFlySpeechConstant
PARAMS]];
// 设置会话场景
[self.identityVerifier setParameter:@"ipt" forKey:[IFlySpeechConstant
MFV_SCENES]];
// 用户 id
NSString* authId=self.authIdLabel.text;
[self.identityVerifier setParameter:authId forKey:[IFlySpeechConstant
MFV_AUTH_ID]];

// 设置模型参数，若无可以传空字符传
NSString* params =[NSString
stringWithFormat:@"auth_id=%@,scope=person,group_id=%@",authId,self.group
IdText.text];
// 执行模型操作
[self.identityVerifier execute:@"ipt" cmd:@"add" params:params];

// 回调接口定义如下，使用方法参考 demo 源码；
// 错误回调
- (void)onError:(IFlySpeechError *)error;
// 结果回调，results 中包含有 json 格式结果，字段说明参考附录身份验证结果说明。
- (void)onResults:(IFlyIdentityResult *)results isLast:(BOOL)isLast;
// 扩展接口，用于抛出音量和 vad_eos 消息
- (void)onEvent:(int)eventType arg1:(int)arg1 arg2:(int)arg2 extra:(id)obj;
```

用户退出组示例代码：

```
// 取消上次会话、清空参数
[self.identityVerifier cancel];
[self.identityVerifier setParameter:nil forKey:[IFlySpeechConstant
PARAMS]];
// 设置会话场景
[self.identityVerifier setParameter:@"ipt" forKey:[IFlySpeechConstant
MFV_SCENES]];
// 用户 id
NSString* authId=self.authIdLabel.text;
[self.identityVerifier setParameter:authId forKey:[IFlySpeechConstant
MFV_AUTH_ID]];

// 设置模型参数，若无可以传空字符传
NSString* params =[NSString
stringWithFormat:@"auth_id=%@,scope=person,group_id=%@",authId,self.group
IdText.text];
// 执行模型操作
[self.identityVerifier execute:@"ipt" cmd:@"delete" params:params];

// 回调接口定义如下，使用方法参考 demo 源码；
// 错误回调
- (void)onError:(IFlySpeechError *)error[]
// 结果回调，results 中包含有 json 格式结果，字段说明参考附录身份验证结果说明。
- (void)onResults:(IFlyIdentityResult *)results isLast:(BOOL)isLast{}
// 扩展接口，用于抛出音量和 vad_eos 消息
- (void)onEvent:(int)eventType arg1:(int)arg1 arg2:(int)arg2
extra:(id)obj{}
```

13.3. 参数设置

多生物特征融合验证平台的参数分为两种，分别为 *MFV* 主参数和子业务参数。

MFV 主参数通过 *IFlyIdentityVerifier* 的- *(BOOL)setParameter:(NSString *)value forKey:(NSString *)key* 方法进行设置，参数如表 12-5 所示。

表 13-5 MFV 主参数

名称	说明	取值范围	默认值
auth_id	用户 *id*,用户身份的唯一标识	自拟，长度 6-18 位，仅包括英文、数字	无
group_id	通过组管理功能创建的鉴别组的唯一标识	长度 20 以内的字符串，由组管理功能创建得到，或由他人告知	无，在鉴别场景下必须指定

scenes	会话场景	*ifr*（人脸），*ivp*（声纹），*ifr|ivp*（人脸+声纹），组管理（*ipt*）	无，必须指定
sst	会话的业务类型	*enroll*（注册），*verify*（验证），*identify*（鉴别）	无，必须指定
vcm	验证模式，仅在验证场景下使用	*sin*（单一特征），*mix*（融合），*agi*（灵活）	无，在验证场景下必须指定
afc	灵活验证保留结果时间	0-43200s	无，只在 *vcm* 设置成 *agi* 时生效
prot_type	联网协议	*ssl*	非 *ssl* 协议
sslcert	证书内容	证书内容	赛门铁克安全证书（仅在 *prot_type=ssl* 时生效）

注意：

1.scenes 和 vcm 必须组合使用：例如指定 vcm 为 sin 则 scenes 只可以选择 ifr 或者 ivp 单独业务，vcm 选择 mix 则 scenes 只可以选择 ifr|ivp。另：agi(灵活)模式可以设置 sences=ifr|ivp，在当次会话只发送一种业务数据，服务端保留验证结果，如在设置的时间间隔内再传入另外的业务数据即可做到混合验证。

2. prot_type、sslcert 参数也可在 IFlySpeechUtility. createUtility 时传入，之后每次会话都会生效。

3.支持服务端回调通知业务。当用户进行了注册、验证操作后，讯飞服务端发送结果消息给开发者的业务服务器，如需此服务请联系：msp_support@iflytek.com

子业务参数以 *String* 格式的键-值对在调用 *IflyIdentityVerifier* 的*-(void)write:(NSString*)ssub data:(NSData*)data offset:(int)offset length:(int)length withParams:(NSString*)params;*写入子业务特征数据或者调用*-(void)execute:(NSString*)ssub cmd:(NSString*)cmd params:(NSString*)params;*执行模型操作时传入，对应于 *params* 参数。详见 *IFlyMFVDemo*。

表 13-6 MFV 子业务参数

子业务	名称	说明	取值	默认值
ivp（声纹）	*rgn*	注册次数。	2-9。	5(建议使用默认值，效果最好)
	ptxt	密码文本，指定声纹密码注册时使用的声纹密码内容。	由服务端下发，比如数字密码所需的数字串。	无，必须指定
	pwdt	密码类型，指定声纹密码注册时使用的声纹密码类型。	3（数字密码） *注：其他类型暂不支持。*	无，必须指定
ifr（人脸）	暂无			
ipt（组管理）	*auth_id*	用户 *id*		设备 *id*，必须指定
	scope	操作对象	*group*（组） *person*（成员）	无，必须指定
	group_name	创建的组名称		无，必须指定
	group_id	加入的组 *id*		无，必须指定

结果格式说明详见附录 13.5

14. 附录

14.1. 常见问题整理

参见论坛帖子：*iOS MSC SDK* 常见问题整理。

14.2. 语音识别结果说明

*JSON*字段	英文全称	类型	说明
sn	*sentence*	*int*	第几句
ls	*last sentence*	*boolean*	是否最后一句
bg	*begin*	*int*	开始
ed	*end*	*int*	结束
ws	*words*	*array*	词
cw	*chinese word*	*array*	中文分词
w	*word*	*string*	单字
sc	*socre*	*int*	分数

语音听写结果示例：

```
{"sn":1,"ls":true,"bg":0,"ed":0,"ws":[{"bg":0,"cw":[{"w":"　今
天　","sc":0}]},{"bg":0,"cw":[{"w":"　的","sc":0}]},{"bg":0,"cw":[{"w":"　天
气　","sc":0}]},{"bg":0,"cw":[{"w":"　怎么样　","sc":0}]},{"bg":0,"cw":[{"w":"　。","sc":0}]}]}
```

多候选结果示例：

```
{"sn":1,"ls":false,"bg":0,"ed":0,"ws":[
{"bg":0,"cw":[{"w":"我想听","sc":0}]},
{"bg":0,"cw":[{"w":"拉德斯基进行曲","sc":0},{"w":"拉得斯进行曲","sc":0}]}]}
```

语法识别结果示例：

```
{"sn":1,"ls":true,"bg":0,"ed":0,"ws":[
        {"bg":0,"cw":[{"sc":"70","gm":"0","w":"北京到上海"},
                {"sc":"69","gm":"0","w":"天京到上海"},
                {"sc":"58","gm":"0","w":"东京到上海"}
               ]
        }
```

```
]}
```

14.3. 声纹业务

文本密码 *JSON* 示例

```
{"txt_pwd":["我的地盘我做主","移动改变生活","芝麻开门"]}
```

数字密码 *JSON* 示例

```
{"num_pwd":["03285469","09734658","53894276","57392804","68294073"]}
```

声纹业务结果（*VerifierResult*）成员说明

成员	说明
sst	业务类型，取值为 train 或 verify
ret	返回值，0 为成功，-1 为失败
vid	注册成功的声纹模型 id
score	当前声纹相似度
suc	本次注册已成功的训练次数
rgn	本次注册需要的训练次数
trs	注册完成描述信息
err	注册/验证返回的错误码
dcs	描述信息

14.4. 人脸识别结果说明

*JSON*字段	类型	说明
sst	*string*	指定本路会话是属于何种性质
rst	*bool*	结果
sid	*string*	会话 *id*
ret	*int*	错误码
uid	*string*	用户 *id*
score	*float*	打分
pose	*dictionary*	面部朝向
confidence	*float*	置信度
position	*dictionary*	面部的矩形区域
landmark	*dictionary*	关键点数组

人脸检测时响应结果返回 json 格式串：


```
{
    "face": [
        {
            "attribute": {
                "pose": {
                    "pitch": 1
                }
            },
            "confidence": "10.412",
            "position": {
                "bottom": 447,
                "left": 140,
                "right": 419,
                "top": 168
            },
            "tag": ""
        }
    ],
    "ret": "0",
    "rst": "success",
    "sid": "wfr278422e9@ch47fc0817c22e477000",
    "sst": "detect",
    "uid": "a1644276827"
}
```


关键点检测时响应结果返回 *json* 格式串：


```
{
    "result": [
        {
            "landmark": {
                "left_eye_center": {
                    "x": "209.739",
                    "y": "229.428"
                },
                "left_eye_left_corner": {
                    "x": "177.219",
                    "y": "230.914"
                },
                "left_eye_right_corner": {
                    "x": "235.839",
                    "y": "236.793"
```

```
},
"left_eyebrow_left_corner": {
    "x": "155.253",
    "y": "187.392"
},
"left_eyebrow_middle": {
    "x": "199.240",
    "y": "182.701"
},
"left_eyebrow_right_corner": {
    "x": "246.582",
    "y": "192.358"
},
"mouth_left_corner": {
    "x": "204.203",
    "y": "386.777"
},
"mouth_lower_lip_bottom": {
    "x": "262.768",
    "y": "416.832"
},
"mouth_middle": {
    "x": "263.705",
    "y": "390.507"
},
"mouth_right_corner": {
    "x": "317.841",
    "y": "390.864"
},
"mouth_upper_lip_top": {
    "x": "264.736",
    "y": "367.996"
},
"nose_bottom": {
    "x": "267.811",
    "y": "339.358"
},
"nose_left": {
    "x": "225.449",
    "y": "319.586"
},
"nose_right": {
    "x": "308.086",
    "y": "323.936"
```

```
                    },
                    "nose_top": {
                        "x": "271.755",
                        "y": "310.934"
                    },
                    "right_eye_center": {
                        "x": "335.608",
                        "y": "234.335"
                    },
                    "right_eye_left_corner": {
                        "x": "306.995",
                        "y": "238.703"
                    },
                    "right_eye_right_corner": {
                        "x": "364.231",
                        "y": "240.307"
                    },
                    "right_eyebrow_left_corner": {
                        "x": "300.652",
                        "y": "194.243"
                    },
                    "right_eyebrow_middle": {
                        "x": "347.711",
                        "y": "188.787"
                    },
                    "right_eyebrow_right_corner": {
                        "x": "391.572",
                        "y": "197.455"
                    }
                }
            }
        ],
        "ret": "0",
        "rst": "success",
        "sid": "wfr278522e9@ch47fc0817c255477600",
        "sst": "align",
        "uid": "a1644276827"
}
```

14.5. 身份验证结果说明

14.5.1. 人脸注册字段：

JSON 字段	类型	说明
ssub	*String*	业务类型，人脸业务为 *ifr*：人脸验证
sst	*String*	子业务类型，注册业务为 *enroll*；
ret	*int*	返回值，0 为请求成功，其他为请求失败
suc	*int*	本次注册已成功的训练次数
rgn	*int*	本次注册需要的训练次数
fid	*String*	人脸模型 *id* (当前无需关注)

人脸注册结果示例：

```
{"ret":0,"suc":1,"rgn":1,"sst":"enroll","ssub":"ifr","fid":"90f821fa7381ee297a80ed9570dea635"}
```

14.5.2. 声纹注册字段：

JSON 字段	类型	说明
ssub	String	业务类型，声纹业务为 ivp
sst	String	子业务类型，注册业务为 enroll；
ret	int	返回值，0 为请求成功，其他为请求失败
rgn	int	本次注册需要的训练次数
suc	int	本次注册已成功的训练次数
vid	String	声纹模型 id (当前无需关注)

声纹注册结果示例：

```
{"vid":"16eb6a9f24c96405647347f8458e4cea","suc":5,"rgn":5,"sst":"enroll","ssub":"ivp","ret":0}
```

14.5.3. 人脸、声纹和融合验证字段

JSON 字段	类型	说明

ssub	*String*	业务类型，取值： *ivp*：声纹验证； *ifr*：人脸验证； *ivp*| *ifr*：融合验证；
sst	*String*	子业务类型，验证业务为 *verify*；
ret	*int*	返回值，0 为请求成功，其他为请求失败
decision	*String*	*accepted*：验证成功，*rejected*：验证失败
fusion _score	*double*	相似度得分，仅验证业务返回
face _score	*double*	人脸验证得分，仅验证业务返回
voice _score	*double*	声纹验证得分，仅验证业务返回

验证结果示例：

{"ret":0,"face_score":99.732,"voice_score":86.874,"ssub":"ivp|ifr","decision":"accepted","fusion_score":99.823,"sst":"verify"}

14.5.4. 人脸、声纹鉴别字段

JSON 字段	类型	说明
sst	*String*	业务类型，鉴别业务为 *identify*
ssub	*String*	子业务类型，取值： *ivp*：声纹； *ifr*：人脸。
ret	*int*	返回值，0 为请求成功，其他为请求失败
group_id	*String*	本次鉴别的成员组 *id*
group_name	*String*	本次鉴别的成员组 *id* 对应的组名称
topc	*int*	本次鉴别返回的结果数
model_id	*String*	模型 *id*
decision	*String*	*accepted*：匹配成功，*rejected*：匹配失败
score	*double*	匹配相似度
user_name	*String*	该模型对应用户名

鉴别结果示例：

{"ret":0,"group_id":"xxxxxx","group_name":"xxxxxx","ifv_result":{"candidates":[{"model_id":"xxxxxxxx","decision":"accepted","score":88.888888,"user":"user_name"}]},"sst":"identify","ssub":"ivp","topc":1}

14.5.5. 查询/删除模型字段：

JSON 字段	类型	说明
ssub	*String*	业务类型，取值： *ivp*：声纹业务； *ifr*：人脸业务；（暂无查询业务）
ret	*int*	返回值，0 为请求成功，其他为请求失败
sst	*String*	子业务类型，取值： *query*：查询模型； *delete*：删除模型；

查询结果示例：

```
{"ssub":"ivp","sst":"query","ret":0}
```

删除结果示例：

```
{"ssub":"ivp","sst":"delete","ret":0}
```

14.5.6. 组管理字段：

JSON 字段	类型	说明
ssub	*String*	*ipt*：组管理
ret	*int*	返回值，0 为请求成功，其他为请求失败
group_name	*String*	组名称
group_id	*String*	组 *id*
person	*array*	组内成员集
user	*String*	用户名

创建组结果示例：

```
{"ssub":"ipt","group_name":" xxxxxxxx ","sst":"add","ret":0,"group_id":"xxxxxxxx"}
```

删除组结果示例：

```
{"ssub":"ipt","group_name":" xxxxxxxx ","sst":"delete","ret":0,"group_id":" xxxxxxxx "}
```

查询组中人员结果示例：

```
{"ssub":"ipt","person":[{"user":" xxxxxxxx "}],"group_name":" xxxxxxxx
```

","sst":"query","ret":0,"group_id":" xxxxxxxx "}

用户加入组结果示例：

{"ssub":"ipt","group_name":" xxxxxxxx ","ret":0,"sst":"add","user":" xxxxxxxx ","group_id":" xxxxxxxx "}

用户退出组结果示例：

{"ssub":"ipt","group_name":" xxxxxxxx ","ret":0,"sst":"delete","user":" xxxxxxxx ","group_id":" xxxxxxxx "}

14.6. 合成发音人列表

语言：

1、 语言为中英文的发音人可以支持中英文的混合朗读。

2、 英文发音人只能朗读英文，中文无法朗读。

3、 汉语发音人只能朗读中文，遇到英文会以单个字母的方式进行朗读。

4、 使用**新引擎参数**会获得更好的合成效果。

发音人名称	属性	语言	参数名称	新引擎参数	备注
小燕	青年女声	中英文（普通话）	*xiaoyan*		默认
小宇	青年男声	中英文（普通话）	*xiaoyu*		
凯瑟琳	青年女声	英文	*catherine*		
亨利	青年男声	英文	*henry*		
玛丽	青年女声	英文	*vimary*		
小研	青年女声	中英文（普通话）	*vixy*		
小琪	青年女声	中英文（普通话）	*vixq*	*xiaoqi*	
小峰	青年男声	中英文（普通话）	*vixf*		
小梅	青年女声	中英文（粤语）	*vixm*	*xiaomei*	
小莉	青年女声	中英文(台湾普通话)	*vixl*	*xiaolin*	
小蓉	青年女声	汉语（四川话）	*vixr*	*xiaorong*	
小芸	青年女声	汉语（东北话）	*vixyun*	*xiaoqian*	
小坤	青年男声	汉语（河南话）	*vixk*	*xiaokun*	
小强	青年男声	汉语（湖南话）	*vixqa*	*xiaoqiang*	
小莹	青年女声	汉语（陕西话）	*vixying*		
小新	童年男声	汉语（普通话）	*vixx*	*xiaoxin*	
楠楠	童年女声	汉语（普通话）	*vinn*	*nannan*	
老孙	老年男声	汉语（普通话）	*vils*		
Mariane		法语	*Mariane*		
Guli		维语	*Guli*		
Allabent		俄语	*Allabent*		
Gabriela		西班牙语	*Gabriela*		
Abha		印地语	*Abha*		

XiaoYun		越南语	*XiaoYun*		

14.7. 错误码列表

1、10000~19999 的错误码参见 *MSC* 错误码链接:
http://www.xfyun.cn/default/doccenter/doccenterInner?itemTitle=ZmFx

2、其它错误码参见下表

错误码	错误值	意义
SPEECH_ERROR_NO_NETWORK	20001	无有效的网络连接
SPEECH_ERROR_NETWORK_TIMEOUT	20002	网络连接超时
SPEECH_ERROR_NET_EXPECTION	20003	网络异常
SPEECH_ERROR_INVALID_RESULT	20004	无有效的结果
SPEECH_ERROR_NO_MATCH	20005	无匹配结果
SPEECH_ERROR_AUDIO_RECORD	20006	录音失败
SPEECH_ERROR_NO_SPPECH	20007	未检测到语音
SPEECH_ERROR_SPEECH_TIMEOUT	20008	音频输入超时
SPEECH_ERROR_EMPTY_UTTERANCE	20009	无效的文本输入
SPEECH_ERROR_FILE_ACCESS	20010	文件读写失败
SPEECH_ERROR_PLAY_MEDIA	20011	音频播放失败
SPEECH_ERROR_INVALID_PARAM	20012	无效的参数
SPEECH_ERROR_TEXT_OVERFLOW	20013	文本溢出
SPEECH_ERROR_INVALID_DATA	20014	无效数据
SPEECH_ERROR_LOGIN	20015	用户未登陆
SPEECH_ERROR_PERMISSION_DENIED	20016	无效授权
SPEECH_ERROR_INTERRUPT	20017	被异常打断
SPEECH_ERROR_VERSION_LOWER	20018	版本过低
ERROR_IN_USE	20019	未知错误

14.8. 集成帮助文档

打开终端（*termainl* 或 *iterm*），*cd* 到压缩包的 *doc* 目录，执行以下命令：

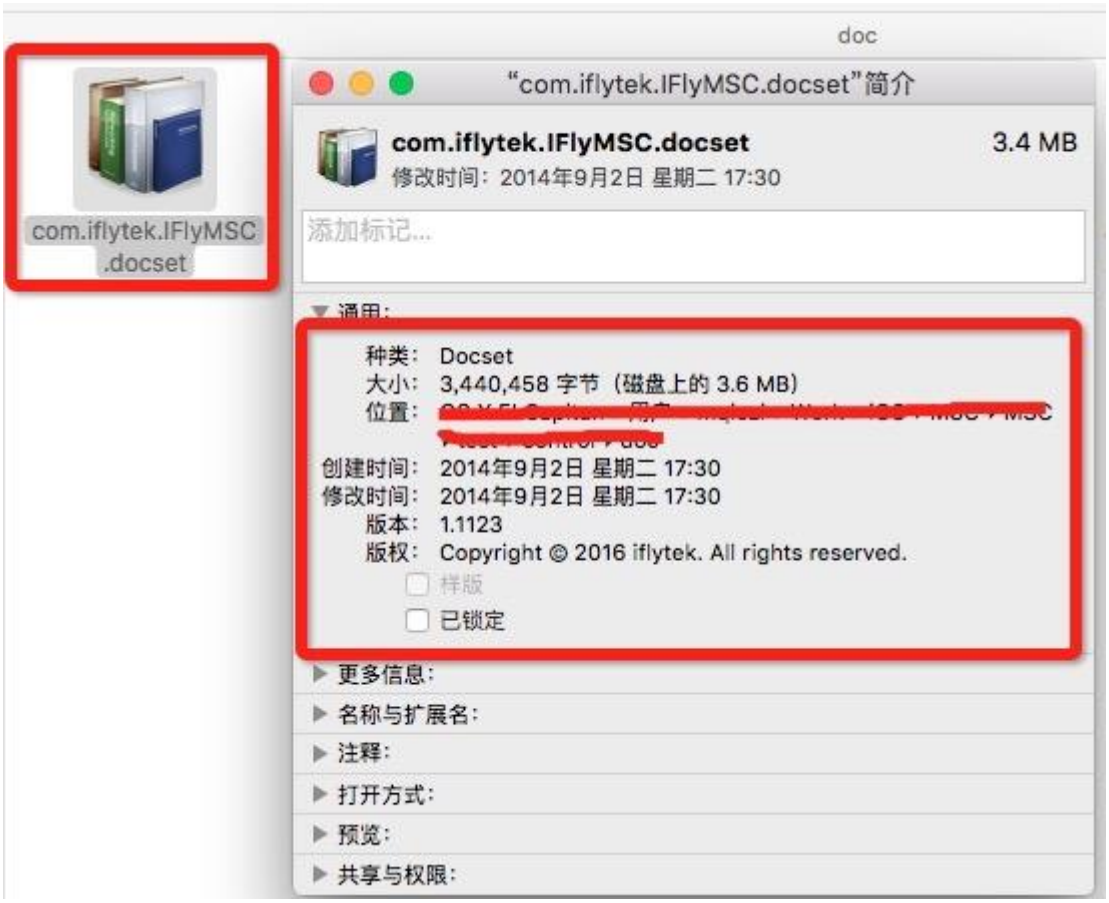
```
cp -R -f -a com.iflytek.IFlyMSC.docset ~/Library/Developer/Shared/Documentation/DocSets/
```

然后执行命令

```
open ~/Library/Developer/Shared/Documentation/DocSets/
```

注意：不同的 Xcode 版本，对应的 docset 路径可能有变化，需要根据实际路径来操作。（例如：/Applications/Xcode.app/Contents/Developer/Documentation/DocSets/）

请核对文档的版本为最新下载的版本



打开 *Xcode* 的帮助文档就可以看到已经集成的文档。