语音识别

SDK 文档

产品文档

腾讯云

【版权声明】

©2013-2019 腾讯云版权所有

本文档著作权归腾讯云单独所有，未经腾讯云事先书面许可，任何主体不得以任何形式复制、修改、抄袭、传播全部或部分本文档内容。

【商标声明】

及其它腾讯云服务相关的商标均为腾讯云计算（北京）有限责任公司及其关联公司所有。本文档涉及的第三方主体的商标，依法由权利人所有。

【服务声明】

本文档意在向客户介绍腾讯云全部或部分产品、服务的当时的整体概况，部分产品、服务的内容可能有所调整。您所购买的腾讯云产品、服务的种类、服务标准等应由您与腾讯云之间的商业合同约定，除非双方另有约定，否则，腾讯云对本文档内容不做任何明示或模式的承诺或保证。

文档目录

SDK 文档
- 一分钟跑通集成 SDK
  - iOS
    - 实时语音识别
  - Android
    - 实时语音识别

版权所有：腾讯云计算（北京）有限责任公司

SDK 文档
一分钟跑通集成 SDK
iOS
实时语音识别

最近更新时间：2022-07-25 10:00:09

接入准备

SDK 获取

实时语音识别的 iOS SDK 以及 Demo 的下载地址：iOS SDK。

接入须知

- 开发者在调用前请先查看实时语音识别的接口说明，了解接口的**使用要求**和**使用步骤**。
- 该接口需要手机能够连接网络（GPRS、3G 或 Wi-Fi 网络等），且系统为 **iOS 9.0** 及以上版本。

开发环境

在工程 `info.plist` 添加以下设置：

- **设置 NSAppTransportSecurity 策略，添加如下内容：**

```
<key>NSAppTransportSecurity</key>
<dict>
<key>NSExceptionDomains</key>
<dict>
<key>qcloud.com</key>
<dict>
<key>NSExceptionAllowsInsecureHTTPLoads</key>
<true/>
<key>NSExceptionMinimumTLSVersion</key>
<string>TLSv1.2</string>
<key>NSIncludesSubdomains</key>
<true/>
<key>NSRequiresCertificateTransparency</key>
<false/>
</dict>
```

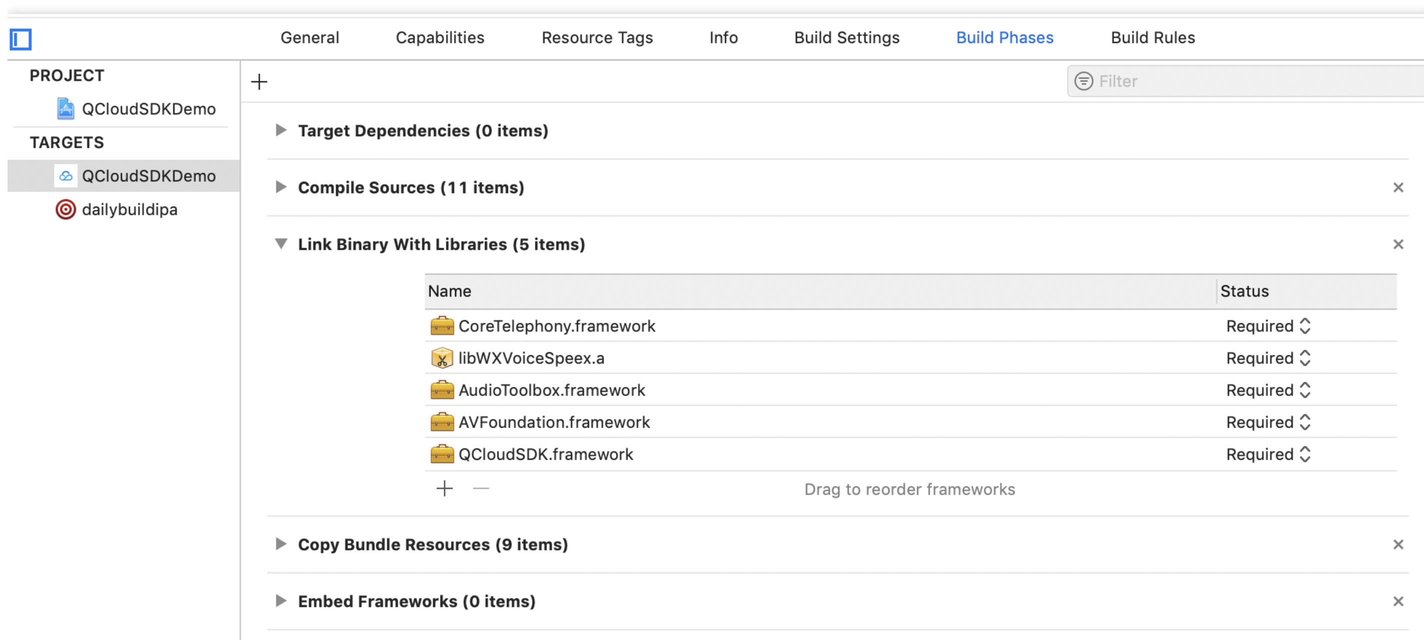
```
</dict>
</dict>
```

- **申请系统麦克风权限，添加如下内容：**

```
<key>NSMicrophoneUsageDescription</key>
<string>需要使用您的麦克风采集音频</string>
```

- **在工程中添加依赖库，在 build Phases Link Binary With Libraries 中添加以下库：**
  - AVFoundation.framework
  - AudioToolbox.framework
  - QCloudSDK.framework
  - CoreTelephony.framework
  - libWXVoiceSpeex.a

添加完后如下图所示：



快速接入

开发流程及接入示例

下面分别介绍**使用内置录音器采集语音识别**和**调用者提供语音数据**接入流程和示例。

版权所有：腾讯云计算（北京）有限责任公司

使用内置录音器采集语音识别示例

1. 引入 **QCloudSDK** 的头文件，将使用 **QCloudSDK** 的文件名后缀由 **.m->.mm**

```
#import<QCloudSDK/QCloudSDK.h>
```

2. 创建 **QCloudConfig** 实例

```
//1.创建 QCloudConfig 实例
QCloudConfig *config = [[QCloudConfig alloc] initWithAppId:kQDAppId
secretId:kQDSecretId
secretKey:kQDSecretKey
projectId:kQDProjectId];
config.sliceTime = 600; //语音分片时长600ms
config.enableDetectVolume = YES; //是否检测音量
config.endRecognizeWhenDetectSilence = YES; //是否检测到静音停止识别
```

3. 创建 **QCloudRealTimeRecognizer** 实例

```
QCloudRealTimeRecognizer *recognizer = [[QCloudRealTimeRecognizer alloc] initWithConfig:config];
```

4. 设置 **delegate**，实现 **QCloudRealTimeRecognizerDelegate** 方法

```
recognizer.delegate = self;
```

5. 开始识别

```
[recognizer start];
```

6. 结束识别

```
[recognizer stop];
```

调用者提供语音数据示例

1. 引入 **QCloudSDK** 的头文件，将使用 **QCloudSDK** 的文件名后缀由 **.m->.mm**

```
#import<QCloudSDK/QCloudSDK.h>
```

2. **创建 QCloudConfig 实例**

```
//1.创建 QCloudConfig 实例
QCloudConfig *config = [[QCloudConfig alloc] initWithAppId:kQDAppId
secretId:kQDSecretId
secretKey:kQDSecretKey
projectId:kQDProjectId];
config.sliceTime = 600; //语音分片时长600ms
config.enableDetectVolume = YES; //是否检测音量
config.endRecognizeWhenDetectSilence = YES; //是否检测到静音停止识别
```

3. **自定义 QCloudDemoAudioDataSource，QCloudDemoAudioDataSource 实现 QCloudAudioDataSource 协议**

```
QCloudDemoAudioDataSource *dataSource = [[QCloudDemoAudioDataSource alloc] init];
```

4. **创建 QCloudRealTimeRecognizer 实例**

```
QCloudRealTimeRecognizer *recognizer = [[QCloudRealTimeRecognizer alloc] initWithConfig:config dataSource:dataSource];
```

5. **设置 delegate，实现 QCloudRealTimeRecognizerDelegate 方法**

```
recognizer.delegate = self;
```

6. **开始识别**

```
[recognizer start];
```

7. **结束识别**

```
[recognizer stop];
```

主要接口类说明

QCloudRealTimeRecognizer 初始化说明

QCloudRealTimeRecognizer 是实时语音识别类，提供两种初始化方法。

```
/**
* 初始化方法，调用者使用内置录音器采集音频
* @param config 配置参数，详见QCloudConfig定义
*/
- (instancetype)initWithConfig:(QCloudConfig *)config;
/**
* 初始化方法，调用者传递语音数据调用此初始化方法
* @param config 配置参数，详见QCloudConfig定义
* @param dataSource 语音数据数据源，必须实现QCloudAudioDataSource协议
*/
- (instancetype)initWithConfig:(QCloudConfig *)config dataSource:(id<QCloudAudioDataSource>)dataSource;
```

QCloudConfig 初始化方法说明

```
/**
* 初始化方法-直接鉴权
* @param appid 腾讯云 appId
* @param secretId 腾讯云 secretId
* @param secretKey 腾讯云 secretKey
* @param projectId 腾讯云 projectId
*/
- (instancetype)initWithAppId:(NSString *)appid
secretId:(NSString *)secretId
secretKey:(NSString *)secretKey
projectId:(NSString *)projectId;
/**
* 初始化方法-通过STS临时证书鉴权
* @param appid 腾讯云appId
* @param secretId 腾讯云临时secretId
* @param secretKey 腾讯云临时secretKey
* @param token 对应的token
*/
- (instancetype)initWithAppId:(NSString *)appid
secretId:(NSString *)secretId
secretKey:(NSString *)secretKey
token:(NSString *)token;
```

版权所有：腾讯云计算（北京）有限责任公司

QCloudRealTimeRecognizerDelegate 方法说明

```
/**
* 一次实时录音识别，分为多个flow，每个 flow 可形象的理解为一句话，一次识别中可以包括多句话。
* 每个 flow 包含多个 seq 语音数据包，每个 flow 的 seq 从0开始
*/
@protocol QCloudRealTimeRecognizerDelegate <NSObject>
@required
/**
* 每个语音包分片识别结果
* @param response 语音分片的识别结果
*/
- (void)realTimeRecognizerOnSliceRecognize:(QCloudRealTimeRecognizer *)recognizer
response:(QCloudRealTimeResponse *)response;
@optional
/**
* 一次识别成功回调
@param recognizer 实时语音识别实例
@param result 一次识别出的总文本
*/
- (void)realTimeRecognizerDidFinish:(QCloudRealTimeRecognizer *)recognizer resul
t:(NSString *)result;
/**
* 一次识别失败回调
* @param recognizer 实时语音识别实例
* @param error 错误信息
* @param voiceId 如果错误是后端返回的，附带voiceId
*/
- (void)realTimeRecognizerDidError:(QCloudRealTimeRecognizer *)recognizer error:
(NSError *)error voiceId:(NSString * _Nullable) voiceId;
////////////////////////////////////////////////////////////////////////////////
//////////////////////////////////////////////
/**
* 开始录音回调
* @param recognizer 实时语音识别实例
* @param error 开启录音失败，错误信息
*/
- (void)realTimeRecognizerDidStartRecord:(QCloudRealTimeRecognizer *)recognizer e
rror:(NSError *)error;
/**
* 结束录音回调
* @param recognizer 实时语音识别实例
*/
- (void)realTimeRecognizerDidStopRecord:(QCloudRealTimeRecognizer *)recognizer;
/**
* 录音音量实时回调用
* @param recognizer 实时语音识别实例
```

```
* @param volume 声音音量，取值范围（-40-0）
*/
- (void)realTimeRecognizerDidUpdateVolume:(QCloudRealTimeRecognizer *)recognizer
volume:(float)volume;

////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////
/**
* 语音流的开始识别
* @param recognizer 实时语音识别实例
* @param voiceId 语音流对应的 voiceId，唯一标识
* @param seq flow 的序列号
*/
- (void)realTimeRecognizerOnFlowRecognizeStart:(QCloudRealTimeRecognizer *)recogn
izer voiceId:(NSString *)voiceId seq:(NSInteger)seq;
/**
* 语音流的结束识别
* @param recognizer 实时语音识别实例
* @param voiceId 语音流对应的 voiceId，唯一标识
* @param seq flow的序列号
*/
- (void)realTimeRecognizerOnFlowRecognizeEnd:(QCloudRealTimeRecognizer *)recogniz
er voiceId:(NSString *)voiceId seq:(NSInteger)seq;
////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////
/**
* 语音流开始识别
* @param recognizer 实时语音识别实例
* @param voiceId 语音流对应的 voiceId，唯一标识
* @param seq flow 的序列号
*/
- (void)realTimeRecognizerOnFlowStart:(QCloudRealTimeRecognizer *)recognizer voic
eId:(NSString *)voiceId seq:(NSInteger)seq;
/**
* 语音流结束识别
* @param recognizer 实时语音识别实例
* @param voiceId 语音流对应的 voiceId，唯一标识
* @param seq flow 的序列号
*/
- (void)realTimeRecognizerOnFlowEnd:(QCloudRealTimeRecognizer *)recognizer voiceI
d:(NSString *)voiceId seq:(NSInteger)seq;
@end
```

QCloudAudioDataSource 协议说明

调用者不适用 SDK 内置录音器进行语音数据采集，自己提供语音数据需要实现此协议所有方法，可见 Demo 工程中的 QDAudioDataSource 实现。

```
/**
* 语音数据数据源，如果调用者需要自己提供语音数据， 调用者实现此协议中所有方法
* 提供符合以下要求的语音数据：
* 采样率：16k
* 音频格式：pcm
* 编码：16bit位深的单声道
*/
@protocol QCloudAudioDataSource <NSObject>
@required
/**
* 标识 data source是否开始工作，执行完 start 后需要设置成 YES， 执行完 stop 后需要设置成 NO
*/
@property (nonatomic, assign) BOOL running;
/**
* SDK 会调用 start 方法，实现此协议的类需要初始化数据源。
*/
- (void)start:(void(^)(BOOL didStart, NSError *error))completion;
/**
* SDK 会调用 stop 方法，实现此协议的类需要停止提供数据
*/
- (void)stop;
/**
* SDK 会调用实现此协议的对象的此方法读取语音数据， 如果语音数据不足 expectLength，则直接返回nil。
* @param expectLength 期望读取的字节数，如果返回的 NSData 不足 expectLength个字节，SDK 会抛出异常。
*/
- (nullable NSData *)readData:(NSInteger)expectLength;
@end
```

Android 实时语音识别

最近更新时间：2022-07-28 19:21:45

接入准备

SDK 获取

实时语音识别 Android SDK 及 Demo 下载地址：Android SDK。

接入须知

- 开发者在调用前请先查看实时语音识别的接口说明，了解接口的**使用要求**和**使用步骤**。
- 该接口需要手机能够连接网络（GPRS、3G 或 Wi-Fi 等），且系统为 **Android 4.0** 及其以上版本。

开发环境

- 引入 aar 包
  speech_release.aar：腾讯云语音识别 SDK。

```
implementation(name: 'speech_release', ext: 'aar')
```

- 添加相关依赖
  okhttp3、okio、gson 和 slf4j 依赖添加，在 build.gradle 文件中添加：

```
implementation 'com.squareup.okhttp3:okhttp:4.2.2'
implementation 'com.squareup.okio:okio:1.11.0'
implementation 'com.google.code.gson:gson:2.8.5'
implementation 'org.slf4j:slf4j-api:1.7.25'
```

- 在 AndroidManifest.xml 添加如下权限：

```
< uses-permission android:name="android.permission.RECORD_AUDIO"/>
< uses-permission android:name="android.permission.INTERNET"/>
< uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
```

快速接入

启动实时语音识别

```
int appid = XXX;
int projectid = XXX;
String secretId = "XXX";
// 为了方便用户测试，sdk提供了本地签名，但是为了secretKey的安全性，正式环境下请自行在第三方服务器上生成签名。
AbsCredentialProvider credentialProvider = new LocalCredentialProvider("your secretKey");
final AAIClient aaiClient;
try {
// 1、初始化AAIClient对象。
aaiClient = new AAIClient(this, appid, projectid, secretId, credentialProvider);
/**您也可以使用临时证书鉴权
* * 1.通过sts 获取到临时证书,此步骤应在您的服务器端实现
* 2.通过临时密钥调用接口
* **/
// aaiClient = new AAIClient(MainActivity.this, appid, projectId,"临时secretId",
"临时secretKey","对应的token" ,credentialProvider);

// 2、初始化语音识别请求。
final AudioRecognizeRequest audioRecognizeRequest = new AudioRecognizeRequest.Builder()
.pcmAudioDataSource(new AudioRecordDataSource()) // 设置语音源为麦克风输入
.build();
// 3、初始化语音识别结果监听器。
final AudioRecognizeResultListener audioRecognizeResultListener = new AudioRecognizeResultListener() {
@Override
public void onSliceSuccess(AudioRecognizeRequest audioRecognizeRequest, AudioRecognizeResult audioRecognizeResult, int i) {
// 返回语音分片的识别结果
}
@Override
public void onSegmentSuccess(AudioRecognizeRequest audioRecognizeRequest, AudioRecognizeResult audioRecognizeResult, int i) {
// 返回语音流的识别结果
}
@Override
public void onSuccess(AudioRecognizeRequest audioRecognizeRequest, String s) {
// 返回所有的识别结果
}
@Override
public void onFailure(AudioRecognizeRequest audioRecognizeRequest, ClientExceptio
```

```
n e, ServerException e1) {
// 识别失败
}
};
// 4、启动语音识别
new Thread(new Runnable() {
@Override
public void run() {
if (aaiClient!=null) {
aaiClient.startAudioRecognize(audioRecognizeRequest, audioRecognizeResultListener
);
}
}
}).start();
} catch (ClientException e) {
e.printStackTrace();
}
```

停止实时语音识别

```
// 1、获得请求的 ID
final int requestId = audioRecognizeRequest.getRequestId();
// 2、调用stop方法
new Thread(new Runnable() {
@Override
public void run() {
if (aaiClient!=null){
//停止语音识别，等待当前任务结束
aaiClient.stopAudioRecognize(requestId);
}
}
}).start();
```

取消实时语音识别

```
// 1、获得请求的id
final int requestId = audioRecognizeRequest.getRequestId();
// 2、调用cancel方法
new Thread(new Runnable() {
@Override
public void run() {
if (aaiClient!=null){
//取消语音识别，丢弃当前任务
aaiClient.cancelAudioRecognize(requestId);
}
```

版权所有：腾讯云计算（北京）有限责任公司

```
}
}).start();
```

主要接口类和方法说明

计算签名

调用者需要自己实现 AbsCredentialProvider 接口来计算签名，此方法为 SDK 内部调用，上层不用关心 source 来源。

计算签名函数如下：

```
/**
* 签名函数：将原始字符串进行加密，具体的加密算法见以下说明。
* @param source 原文字符串
* @return 加密后返回的密文
*/
String getAudioRecognizeSign(String source);
```

计算签名算法

先以 SecretKey 对 source 进行 HMAC-SHA1 加密，然后对密文进行 Base64 编码，获得最终的签名串。即：sign=Base64Encode(HmacSha1(source，secretKey))。

为方便用户测试，SDK 已提供一个实现类 **LocalCredentialProvider**，但为保证 SecretKey 的安全性，请仅在测试环境下使用，正式版本建议上层实现接口 **AbsCredentialProvider** 中的方法。

初始化 AAIClient

AAIClient 是语音服务的核心类，用户可以调用该类来开始、停止以及取消语音识别。

```
public AAIClient(Context context, int appid, int projectId, String secreteId, AbsCredentialProvider credentialProvider) throws ClientException
```

参数名称	类型	是否必填	参数描述
context	Context	是	上下文
appid	Int	是	腾讯云注册的 AppID
projectId	Int	否	用户的 projectId
secreteId	String	是	用户的 SecreteId
credentialProvider	AbsCredentialProvider	是	鉴权类

示例：

```
try {
AaiClient aaiClient = new AAIClient(context, appid, projectId, secretId, credenti
alProvider);
} catch (ClientException e) {
e.printStackTrace();
}
```

如果 aaiClient 不再需要使用，请调用 release() 方法释放资源：

```
aaiClient.release();
```

配置全局参数

用户调用 ClientConfiguration 类的静态方法来修改全局配置。

方法	方法描述	默认值	有效范围
setMaxAudioRecognizeConcurrentNumber	语音识别最大并发请求数	2	1 - 5
setMaxRecognizeSliceConcurrentNumber	语音识别分片最大并发数	5	1 - 5
setAudioRecognizeSliceTimeout	HTTP 读超时时间	5000ms	500 - 10000ms
setAudioRecognizeConnectTimeout	HTTP 连接超时时间	5000ms	500 - 10000ms
setAudioRecognizeWriteTimeout	HTTP 写超时时间	5000ms	500 - 10000ms

示例：

```
ClientConfiguration.setMaxAudioRecognizeConcurrentNumber(2)
ClientConfiguration.setMaxRecognizeSliceConcurrentNumber(5)
ClientConfiguration.setAudioRecognizeSliceTimeout(2000)
ClientConfiguration.setAudioRecognizeConnectTimeout(2000)
ClientConfiguration.setAudioRecognizeWriteTimeout(2000)
```

设置结果监听器

AudioRecognizeResultListener 可以用来监听语音识别的结果，共有如下四个接口：

- 语音分片的语音识别结果回调接口

```
void onSliceSuccess(AudioRecognizeRequest request, AudioRecognizeResult result,
int order);
```

参数	参数类型	参数描述
request	AudioRecognizeRequest	语音识别请求
result	AudioRecognizeResult	语音分片的语音识别结果
order	Int	该语音分片所在语音流的次序

- 语音流的语音识别结果回调接口

```
void onSegmentSuccess(AudioRecognizeRequest request, AudioRecognizeResult result,
int order);
```

参数	参数类型	参数描述
request	AudioRecognizeRequest	语音识别请求
result	AudioRecognizeResult	语音分片的语音识别结果
order	Int	该语音流的次序

- 返回所有的识别结果

```
void onSuccess(AudioRecognizeRequest request, String result);
```

参数	参数类型	参数描述
request	AudioRecognizeRequest	语音识别请求
result	String	所有的识别结果

- 语音识别请求失败回调函数

```
void onFailure(AudioRecognizeRequest request, final ClientException clientExcepti
on, final ServerException serverException,String response);
```

参数	参数类型	参数描述
request	AudioRecognizeRequest	语音识别请求
clientException	ClientException	客户端异常
serverException	ServerException	服务端异常

版权所有：腾讯云计算（北京）有限责任公司

参数	参数类型	参数描述
response	String	服务端返回的 json 字符串

示例代码详见 入门示例。

设置语音识别参数

通过构建 AudioRecognizeConfiguration 类，可以设置语音识别时的配置：

参数名称	类型	是否必填	参数描述	默认值
setSilentDetectTimeOut	Boolean	否	是否开启静音检测，开启后说话前的静音部分不进行识别	true
audioFlowSilenceTimeOut	Int	否	开启检测说话启始超时，开启后超时会自动停止录音	5000ms
minAudioFlowSilenceTime	Int	否	两个语音流最短分割时间	2000ms
minVolumeCallbackTime	Int	否	音量回调时间	80ms

示例：

```
AudioRecognizeConfiguration audioRecognizeConfiguration = new AudioRecognizeConfiguration.Builder()
.setSilentDetectTimeOut(true)// 是否使能静音检测, false 表示不检查静音部分
.audioFlowSilenceTimeOut(5000) // 静音检测超时停止录音
.minAudioFlowSilenceTime(2000) // 语音流识别时的间隔时间
.minVolumeCallbackTime(80) // 音量回调时间
.build();
// 启动语音识别
new Thread(new Runnable() {
@Override
public void run() {
if (aaiClient!=null) {
aaiClient.startAudioRecognize(audioRecognizeRequest, audioRecognizeResultListener, audioRecognizeConfiguration);
}
}
}).start();
```

设置状态监听器

AudioRecognizeStateListener 可以用来监听语音识别的状态，一共有如下八个接口：

方法	方法描述
onStartRecord	开始录音
onStopRecord	结束录音
onVoiceFlowStart	检测到语音流的起点
onVoiceFlowStartRecognize	语音流开始识别
onVoiceFlowFinishRecognize	语音流结束识别
onVoiceVolume	音量
onNextAudioData	返回音频流，用于返回宿主层做录音缓存业务。new AudioRecordDataSource(true) 传递 true 时生效

设置超时监听器

AudioRecognizeTimeoutListener 可以用来监听语音识别的超时，一共有如下两个接口：

方法	方法描述
onFirstVoiceFlowTimeout	检测第一个语音流超时
onNextVoiceFlowTimeout	检测下一个语音流超时

示例：

```
AudioRecognizeStateListener audioRecognizeStateListener = new AudioRecognizeState
Listener() {
@Override
public void onStartRecord(AudioRecognizeRequest audioRecognizeRequest) {
// 开始录音
}
@Override
public void onStopRecord(AudioRecognizeRequest audioRecognizeRequest) {
// 结束录音
}
@Override
public void onVoiceFlowStart(AudioRecognizeRequest audioRecognizeRequest, int i)
{
// 语音流开始
}
@Override
public void onVoiceFlowFinish(AudioRecognizeRequest audioRecognizeRequest, int i)
```

```
{
// 语音流结束
}
@Override
public void onVoiceFlowStartRecognize(AudioRecognizeRequest audioRecognizeReques
t, int i) {
// 语音流开始识别
}
@Override
public void onVoiceFlowFinishRecognize(AudioRecognizeRequest audioRecognizeReques
t, int i) {
// 语音流结束识别
}
@Override
public void onVoiceVolume(AudioRecognizeRequest audioRecognizeRequest, int i) {
// 音量回调
}
};
/**
* 返回音频流,
* 用于返回宿主层做录音缓存业务。
* 由于方法跑在sdk线程上，这里多用于文件操作，宿主需要新开一条线程专门用于实现业务逻辑
* new AudioRecordDataSource(true) 有效，否则不会回调该函数
* @param audioDatas
*/
@Override
public void onNextAudioData(final short[] audioDatas, final int readBufferLength)
{
}
```

其他重要类说明

AudioRecognizeRequest

templateName 和 customTemplate 都设置时，优先使用 templateName 的设置。

参数名称	类型	是否必填	参数描述	默认值
pcmAudioDataSource	PcmAudioDataSource	是	音频数据源	无
templateName	String	否	用户控制台设置的模板名称	无
customTemplate	AudioRecognizeTemplate	否	用户自定义的模板	("16k_zh", 1)

AudioRecognizeResult

语音识别结果对象，和 AudioRecognizeRequest 对象相对应，用于返回语音识别的结果。

参数名称	类型	参数描述
code	Int	识别状态码
message	String	识别提示信息
text	String	识别结果
seq	Int	该语音分片的序号
voiceId	String	该语音分片所在语音流的 ID
cookie	String	cookie 值

AudioRecognizeTemplate

自定义的语音模板，需要设置的参数包括：

参数名称	类型	是否必填	参数描述
engineModelType	String	是	引擎模型类型
resType	Int	是	结果返回方式

示例：

```
AudioRecognizeTemplate audioRecognizeTemplate = new AudioRecognizeTemplate("16k_zh",1);
```

PcmAudioDataSource

用户可以实现这个接口来识别单通道、采样率16k的 PCM 音频数据。主要包括如下几个接口：

- 向语音识别器添加数据，将长度为 length 的数据从下标0开始复制到 audioPcmData 数组中，并返回实际的复制的数据量的长度。

```
int read(short[] audioPcmData, int length);
```

- 启动识别时回调函数，用户可以在这里做些初始化的工作。

```
void start() throws AudioRecognizerException;
```

- 结束识别时回调函数，用户可以在这里进行一些清理工作。

```
void stop();
```

- 获取 sdk Pcm 格式录音源文件路径。

```
void savePcmFileCallBack(String filePath);
```

- 获取 sdk wav 格式录音源文件路径。

```
void saveWaveFileCallBack(String filePath);
```

- 设置语音识别器每次最大读取数据量。

```
int maxLengthOnceRead();
```

AudioRecordDataSource

PcmAudioDataSource 接口的实现类，可以直接读取麦克风输入的音频数据，用于实时识别。

AudioFileDataSource

PcmAudioDataSource 接口的实现类，可以直接读取单通道、采样率16k的 PCM 音频数据的文件。

> 注意：
> 其他格式的数据无法正确识别。

AAILogger

用户可以利用 AAILogger 来控制日志的输出，可以选择性的输出 debug、info、warn 以及 error 级别的日志信息。

```
public static void disableDebug();
public static void disableInfo();
public static void disableWarn();
public static void disableError();
public static void enableDebug();
```

版权所有：腾讯云计算（北京）有限责任公司

```
public static void enableInfo();
public static void enableWarn();
public static void enableError();
```

音频数据本地缓存指引

宿主层可根据自身业务需求选择将音频保存到本地或者不保存。若需要保存到本地可按照如下步骤进行操作：

1. `new AudioRecordDataSource(isSaveAudioRecordFiles)` 初始化时，`isSaveAudioRecordFiles` 设置为 true。
2. `AudioRecognizeStateListener.onStartRecord` 回调函数内添加创建本次录音的文件逻辑。路径、文件名可支持自定义。
3. `AudioRecognizeStateListener.onStopRecord` 回调函数内添加关流逻辑。（可选）将 PCM 文件转存为 WAV 文件。
4. `AudioRecognizeStateListener.onNextAudioData` 回调函数内添加将音频流写入本地文件的逻辑。
5. 由于回调函数均跑在 sdk 线程中。为了避免写入业务耗时问题影响 sdk 内部运行流畅度，建议将上述步骤放在单独线程池里完成，详情见 Demo 工程中的 `MainActivity` 类中的示例代码。

版权所有：腾讯云计算（北京）有限责任公司