Automatic Speech Recognition

SDK Documentation

Product Documentation

Contents

SDK Documentation
Quick SDK Integration and Run
iOS
Real-Time Speech Recognition

Last updated：2022-07-25 10:03:59

Connection Preparations

SDK acquisition

The real-time speech recognition SDK and demo for iOS can be downloaded here.

Notes on connection

- You need to view the API description of real-time speech recognition to understand the **use requirements** and **directions** of the API before calling it.
- The API requires the phone to have an internet connection over GPRS, 3G, Wi-Fi, etc. and requires the system to be **iOS 9.0** or later.

Development environment

Add the following settings in the `info.plist` project:

- **Set the `NSAppTransportSecurity` policy by adding the following content:**

```
<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/>
```

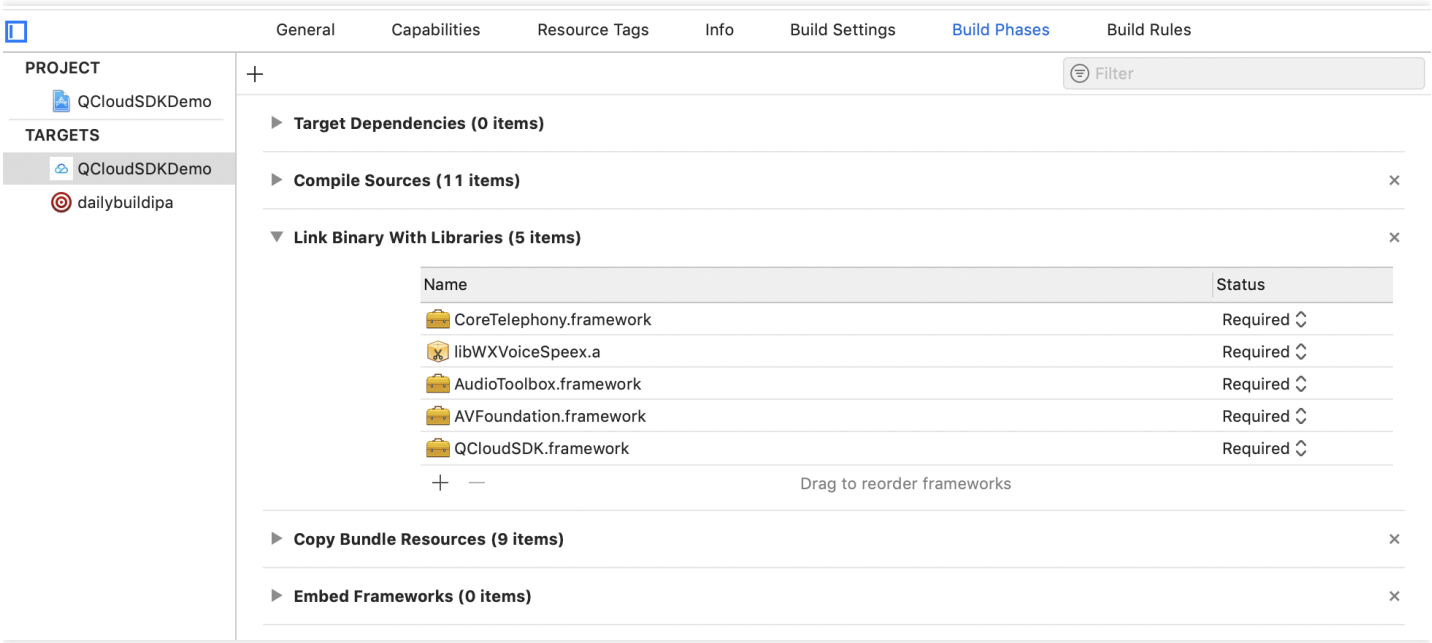
```
</dict>
</dict>
</dict>
```

- **Request the system's mic permission and add the following content:**

```
<key>NSMicrophoneUsageDescription</key>
<string>Your mic is required to capture audios</string>
```

- **Add dependent libraries in the project and add the following libraries in Build Phases' `Link Binary With Libraries` :**
  - AVFoundation.framework
  - AudioToolbox.framework
  - QCloudSDK.framework
  - CoreTelephony.framework
  - libWXVoiceSpeex.a

The libraries are added as shown below:



Quick Connection

Connection process and demo

The following describes the connection processes and demos for **capturing audio for recognition with the built-in recorder** and **providing audio data** respectively.

Demo for capturing audio for recognition with built-in recorder

1. **Import the header file of `QCloudSDK` and change the filename extension from `.m` to `.mm`.**

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

2. **Create a `QCloudConfig` instance.**

```
//1. Create a `QCloudConfig` instance
QCloudConfig *config = [[QCloudConfig alloc] initWithAppId:kQDAppId
secretId:kQDSecretId
secretKey:kQDSecretKey
projectId:kQDProjectId];
config.sliceTime = 600; // The length of the audio segment is 600 ms
config.enableDetectVolume = YES; // Specify whether to detect the volume
config.endRecognizeWhenDetectSilence = YES; // Specify whether to stop recognit
ion when silence is detected
```

3. **Create a `QCloudRealTimeRecognizer` instance.**

```
QCloudRealTimeRecognizer *recognizer = [[QCloudRealTimeRecognizer alloc] initWi
thConfig:config];
```

4. **Set the delegate and implement the QCloudRealTimeRecognizerDelegate method.**

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

5. **Start recognition.**

```
[recognizer start];
```

6. **End recognition.**

```
[recognizer stop];
```

Sample for providing audio data

1. **Import the header file of `QCloudSDK` and change the filename extension from `.m` to `.mm`.**

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

2. **Create a `QCloudConfig` instance.**

```
//1. Create a `QCloudConfig` instance
QCloudConfig *config = [[QCloudConfig alloc] initWithAppId:kQDAppId
secretId:kQDSecretId
secretKey:kQDSecretKey
projectId:kQDProjectId];
config.sliceTime = 600; // The length of the audio segment is 600 ms
config.enableDetectVolume = YES; // Specify whether to detect the volume
config.endRecognizeWhenDetectSilence = YES; // Specify whether to stop recognit
ion when silence is detected
```

3. **Customize `QCloudDemoAudioDataSource` and implement the QCloudAudioDataSource protocol.**

```
QCloudDemoAudioDataSource *dataSource = [[QCloudDemoAudioDataSource alloc] ini
t];
```

4. **Create a `QCloudRealTimeRecognizer` instance.**

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

5. **Set the delegate and implement the QCloudRealTimeRecognizerDelegate method.**

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

6. **Start recognition.**

```
[recognizer start];
```

7. **End recognition.**

```
[recognizer stop];
```

Descriptions of main API classes

QCloudRealTimeRecognizer initialization description

`QCloudRealTimeRecognizer` is the real-time speech recognition class, which provides two initialization methods.

```
/**
* Initialization method where the built-in recorder is used to capture audios
* @param config Configuration parameter. For more information, see the definition
of `QCloudConfig`.
*/
- (instancetype)initWithConfig:(QCloudConfig *)config;
/**
* Initialization method which will be called to pass in audio data
* @param config Configuration parameter. For more information, see the definition
of `QCloudConfig`.
* @param dataSource Data source of audio data. You must implement the `QCloudAudi
oDataSource` protocol.
*/
- (instancetype)initWithConfig:(QCloudConfig *)config dataSource:(id<QCloudAudioD
ataSource>)dataSource;
```

QCloudConfig initialization method description

```
/**
* Initialization method - direct authentication
* @param appid Tencent Cloud `appId`
* @param secretId Tencent Cloud `secretId`
* @param secretKey Tencent Cloud `secretKey`
* @param projectId Tencent Cloud `projectId`
*/
- (instancetype)initWithAppId:(NSString *)appid
secretId:(NSString *)secretId
secretKey:(NSString *)secretKey
projectId:(NSString *)projectId;
/**
* Initialization method - authentication through STS temporary credentials
* @param appid Tencent Cloud `appId`
* @param secretId Tencent Cloud temporary `secretId`
* @param secretKey Tencent Cloud temporary `secretKey`
* @param token Token
```

```
*/
- (instancetype)initWithAppId:(NSString *)appid
secretId:(NSString *)secretId
secretKey:(NSString *)secretKey
token:(NSString *)token;
```

QCloudRealTimeRecognizerDelegate method description

```
/**
* One real-time speech recognition is divided into multiple flows, each of which
can be understood as a sentence, and multiple sentences can be included in one re
cognition.
* Each flow contains multiple `seq` audio data packets, and the `seq` of each flo
w starts from 0.
*/
@protocol QCloudRealTimeRecognizerDelegate <NSObject>
@required
/**
* Recognition result of each audio package segment
* @param response Recognition result of the audio segment
*/
- (void)realTimeRecognizerOnSliceRecognize:(QCloudRealTimeRecognizer *)recognizer
response:(QCloudRealTimeResponse *)response;
@optional
/**
* Callback for recognition success
@param recognizer Real-time speech recognition instance
@param result Total text recognized at one time
*/
- (void)realTimeRecognizerDidFinish:(QCloudRealTimeRecognizer *)recognizer resul
t:(NSString *)result;
/**
* Callback for recognition failure
* @param recognizer Real-time speech recognition instance
* @param error Error message
* @param voiceId The `voiceId` attached to the error returned by the backend
*/
- (void)realTimeRecognizerDidError:(QCloudRealTimeRecognizer *)recognizer error:
(NSError *)error voiceId:(NSString * _Nullable) voiceId;
////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////
/**
* Callback for recording start
* @param recognizer Real-time speech recognition instance
* @param error Error message for recording start failure
```

```
*/
- (void)realTimeRecognizerDidStartRecord:(QCloudRealTimeRecognizer *)recognizer e
rror:(NSError *)error;
/**
* Callback for recording end
* @param recognizer Real-time speech recognition instance
*/
- (void)realTimeRecognizerDidStopRecord:(QCloudRealTimeRecognizer *)recognizer;
/**
* Real-time callback for recording volume
* @param recognizer Real-time speech recognition instance
* @param volume Audio volume level in the range of -40-0
*/
- (void)realTimeRecognizerDidUpdateVolume:(QCloudRealTimeRecognizer *)recognizer
volume:(float)volume;

////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////
/**
* Audio stream recognition start
* @param recognizer Real-time speech recognition instance
* @param voiceId `voiceId` of the audio stream, which is the unique identifier
* @param seq Sequence number of the flow
*/
- (void)realTimeRecognizerOnFlowRecognizeStart:(QCloudRealTimeRecognizer *)recogn
izer voiceId:(NSString *)voiceId seq:(NSInteger)seq;
/**
* Audio stream recognition end
* @param recognizer Real-time speech recognition instance
* @param voiceId `voiceId` of the audio stream, which is the unique identifier
* @param seq Sequence number of the flow
*/
- (void)realTimeRecognizerOnFlowRecognizeEnd:(QCloudRealTimeRecognizer *)recogniz
er voiceId:(NSString *)voiceId seq:(NSInteger)seq;
////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////
/**
* Audio stream recognition start
* @param recognizer Real-time speech recognition instance
* @param voiceId `voiceId` of the audio stream, which is the unique identifier
* @param seq Sequence number of the flow
*/
- (void)realTimeRecognizerOnFlowStart:(QCloudRealTimeRecognizer *)recognizer voic
eId:(NSString *)voiceId seq:(NSInteger)seq;
/**
* Audio stream recognition end
* @param recognizer Real-time speech recognition instance
```

```
* @param voiceId `voiceId` of the audio stream, which is the unique identifier
* @param seq Sequence number of the flow
*/
- (void)realTimeRecognizerOnFlowEnd:(QCloudRealTimeRecognizer *)recognizer voiceI
d:(NSString *)voiceId seq:(NSInteger)seq;
@end
```

QCloudAudioDataSource protocol description

If you provide audio data instead of capturing audio data with the recorder built in the SDK, you need to implement all methods in this protocol in the same way as used for the implementation of `QDAudioDataSource` in the demo project.

```
/**
* Data source of audio data. If you want to provide audio data on your own, you n
eed to implement all methods in this protocol.
* Provide audio data that meets the following requirements:
* Sample rate: 16 kHz
* Audio format: PCM
* Encoding: 16-bit mono-channel
*/
@protocol QCloudAudioDataSource <NSObject>
@required
/**
* It identifies whether the data source has started to work and needs to be set t
o `YES` after the `start` is executed and to `NO` after the `stop` is executed.
*/
@property (nonatomic, assign) BOOL running;
/**
* The SDK will call the `start` method. Implementing the class of this protocol r
equires initializing the data source.
*/
- (void)start:(void(^)(BOOL didStart, NSError *error))completion;
/**
* The SDK will call the `stop` method. Implementing the class of this protocol re
quires stopping supplying data.
*/
- (void)stop;
/**
* The SDK will call this method of the object that implements the protocol to rea
d audio data. If the audio data is less than `expectLength`, `nil` will be return
ed directly.
* @param expectLength The number of bytes expected to be read. If the returned `N
SData` is less than `expectLength` bytes, the SDK will report an exception.
*/
```

```
- (nullable NSData *)readData:(NSInteger)expectLength;
@end
```

Android
Real-Time Speech Recognition

Last updated：2022-08-30 11:17:56

Connection Preparations

SDK acquisition

The real-time speech recognition SDK and demo for Android can be downloaded here.

Notes on connection

- You need to view the API description of real-time speech recognition to understand the **use requirements** and **directions** of the API before calling it.
- The API requires the phone to have an internet connection over GPRS, 3G, Wi-Fi, etc. and requires the system to be **Android 4.0** or later.

Development environment

- Import the AAR package
  speech_release.aar: ASR SDK.

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

- Add dependencies
  Add the OkHttp3, Okio, GSON, and SLF4J dependencies in the `build.gradle` file:

```
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'
```

- Add the following permissions in `AndroidManifest.xml` :

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

Quick Connection

Starting real-time speech recognition

```
int appid = XXX;
int projectid = XXX;
String secretId = "XXX";
// The SDK provides a local signature for testing purposes. However, for the secu
rity of `secretKey`, we recommend you generate a signature on your own server in
the production environment.
AbsCredentialProvider credentialProvider = new LocalCredentialProvider("your secr
etKey");
final AAIClient aaiClient;
try {
// 1. Initialize the AAIClient object.
aaiClient = new AAIClient(this, appid, projectid, secretId, credentialProvider);
/**You can also use temporary credentials for authentication
* * 1. Get temporary credentials through STS. This step should be implemented on
your server.
* 2. Call the API through temporary credentials
* **/
// aaiClient = new AAIClient(MainActivity.this, appid, projectId,"temporary secre
tId", "temporary secretKey","corresponding token" ,credentialProvider);

// 2. Initialize the speech recognition request.
final AudioRecognizeRequest audioRecognizeRequest = new AudioRecognizeRequest.Bui
lder()
.pcmAudioDataSource(new AudioRecordDataSource()) // Set the audio source to mic i
nput
.build();
// 3. Initialize the speech recognition result listener.
final AudioRecognizeResultListener audioRecognizeResultListener = new AudioRecogn
izeResultListener() {
@Override
public void onSliceSuccess(AudioRecognizeRequest audioRecognizeRequest, AudioReco
gnizeResult audioRecognizeResult, int i) {
// Return the recognition result of the audio segment
}
@Override
public void onSegmentSuccess(AudioRecognizeRequest audioRecognizeRequest, AudioRe
cognizeResult audioRecognizeResult, int i) {
// Return the recognition result of the audio stream
}
@Override
public void onSuccess(AudioRecognizeRequest audioRecognizeRequest, String s) {
// Return all recognition results
```

```
}
@Override
public void onFailure(AudioRecognizeRequest audioRecognizeRequest, ClientExceptio
n e, ServerException e1) {
// Recognition failed
}
};
// 4. Start speech recognition.
new Thread(new Runnable() {
@Override
public void run() {
if (aaiClient!=null) {
aaiClient.startAudioRecognize(audioRecognizeRequest, audioRecognizeResultListener
);
}
}
}).start();
} catch (ClientException e) {
e.printStackTrace();
}
```

Stopping real-time speech recognition

```
// 1. Get the request ID
final int requestId = audioRecognizeRequest.getRequestId();
// 2. Call the `stop` method
new Thread(new Runnable() {
@Override
public void run() {
if (aaiClient!=null){
// Stop speech recognition and wait for the current task to end
aaiClient.stopAudioRecognize(requestId);
}
}
}).start();
```

Canceling real-time speech recognition

```
// 1. Get the request ID
final int requestId = audioRecognizeRequest.getRequestId();
// 2. Call the `cancel` method
new Thread(new Runnable() {
@Override
public void run() {
if (aaiClient!=null){
```

```
// Cancel speech recognition to discard the current task
aaiClient.cancelAudioRecognize(requestId);
}
}
}).start();
```

Descriptions of Main API Classes and Methods

Calculating signature

You need to implement the `AbsCredentialProvider` API on your own to calculate the signature. This method is called inside the SDK, and the upper layer doesn't need to care about the `source` .

The signature calculation function is as follows:

```
/**
* Signature function: Encrypts the original string with the encryption algorithm
as described below.
* @param source Original string
* @return Ciphertext returned after encryption
*/
String getAudioRecognizeSign(String source);
```

Signature algorithm

`SecretKey` is used to encrypt the `source` with HMAC-SHA1 first, and then the ciphertext is Base64-encoded to get the final signature string, i.e., `sign=Base64Encode(HmacSha1(source,secretKey))` .

The SDK provides an implementation class **LocalCredentialProvider** for testing purposes, but we recommend you use it only in the test environment to guarantee the security of `SecretKey` and implement the method in the **AbsCredentialProvider** API in the upper layer in the production environment.

Initializing AAIClient

`AAIClient` is a core class of ASR, which you can call to start, stop, and cancel speech recognition.

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

Parameter	Type	Required	Description
context	Context	Yes	Context
appid	Int	Yes	`AppID` registered with Tencent Cloud

Parameter	Type	Required	Description
projectId	Int	No	Your `projectId`
secreteId	String	Yes	Your `SecreteId`
credentialProvider	AbsCredentialProvider	Yes	Authentication class

Sample:

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

If `AAIClient` is no longer needed, you need to call the `release()` method to release resources:

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

Configuring global parameters

You need to call the static methods of the `ClientConfiguration` class to modify the global configuration.

Method	Description	Default Value	Valid Range
setMaxAudioRecognizeConcurrentNumber	Maximum number of concurrent speech recognition requests	2	1–5
setMaxRecognizeSliceConcurrentNumber	Maximum number of concurrent segments for speech recognition	5	1–5
setAudioRecognizeSliceTimeout	HTTP read timeout period	5000 ms	500–10000 ms
setAudioRecognizeConnectTimeout	HTTP connection timeout period	5000 ms	500–10000 ms
setAudioRecognizeWriteTimeout	HTTP write timeout period	5000 ms	500–10000 ms

Sample:

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

Setting result listener

`AudioRecognizeResultListener` can be used to listen on speech recognition results. It has the following four APIs:

- Speech recognition result callback API for audio segment

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

Parameter	Type	Description
request	AudioRecognizeRequest	Speech recognition request
result	AudioRecognizeResult	Speech recognition result of the audio segment
order	Int	Sequence of the audio stream of the audio segment

- Speech recognition result callback API for audio stream

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

Parameter	Type	Description
request	AudioRecognizeRequest	Speech recognition request
result	AudioRecognizeResult	Speech recognition result of the audio segment
order	Int	Sequence of the audio stream

- Return of all recognition results

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

Parameter	Type	Description
request	AudioRecognizeRequest	Speech recognition request
result	String	All recognition results

- Callback function for speech recognition request failure

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

Parameter	Type	Description
request	AudioRecognizeRequest	Speech recognition request
clientException	ClientException	Client exception
serverException	ServerException	Server exception
response	String	JSON string returned by the server

For the sample code, see Demo.

Setting speech recognition parameters

By constructing the `AudioRecognizeConfiguration` class, you can set the speech recognition configuration:

Parameter	Type	Required	Description	Default Value
setSilentDetectTimeOut	Boolean	No	Specifies whether to enable silence detection. After it is enabled, the silence part before the actual speech starts will not be recognized	true
audioFlowSilenceTimeOut	Int	No	Specifies whether to enable speech start detection timeout. After it is enabled, recording will automatically stop after the timeout period elapses	5000 ms
minAudioFlowSilenceTime	Int	No	Minimum period for segmenting two audio streams	2000 ms
minVolumeCallbackTime	Int	No	Volume callback time	80 ms

Sample:

```
AudioRecognizeConfiguration audioRecognizeConfiguration = new AudioRecognizeConfi
guration.Builder()
.setSilentDetectTimeOut(true)// Specifies whether to enable silence detection. `f
alse` indicates not to check the silence part
.audioFlowSilenceTimeOut(5000) // Stop recording after the silence detection time
out period elapses
.minAudioFlowSilenceTime(2000) // Interval for audio stream segmentation
.minVolumeCallbackTime(80) // Volume callback time
.build();
// Start speech recognition
new Thread(new Runnable() {
@Override
public void run() {
if (aaiClient!=null) {
aaiClient.startAudioRecognize(audioRecognizeRequest, audioRecognizeResultListener
, audioRecognizeConfiguration);
}
}
}).start();
```

Setting status listener

`AudioRecognizeStateListener` can be used to listen on speech recognition status. It has the following APIs:

Method	Description
onStartRecord	Start of recording
onStopRecord	Stop of recording
onVoiceFlowStart	Start of audio stream
onVoiceFlowStartRecognize	Start of audio stream recognition
onVoiceFlowFinishRecognize	End of audio stream recognition
onVoiceVolume	Volume
onNextAudioData	Return of the audio stream to the host layer for recording caching. It will take effect when `true` is passed in for `new AudioRecordDataSource(true)`

Setting timeout listener

`AudioRecognizeTimeoutListener` can be used to listen on speech recognition timeout. It has the following two APIs:

Method	Description
onFirstVoiceFlowTimeout	Detects the timeout of the first audio stream
onNextVoiceFlowTimeout	Detects the timeout of the next audio stream

Sample:

```
AudioRecognizeStateListener audioRecognizeStateListener = new AudioRecognizeState
Listener() {
@Override
public void onStartRecord(AudioRecognizeRequest audioRecognizeRequest) {
// Start recording
}
@Override
public void onStopRecord(AudioRecognizeRequest audioRecognizeRequest) {
// End recording
}
@Override
public void onVoiceFlowStart(AudioRecognizeRequest audioRecognizeRequest, int i)
{
// Start the audio stream
}
@Override
public void onVoiceFlowFinish(AudioRecognizeRequest audioRecognizeRequest, int i)
{
// End the audio stream
}
@Override
public void onVoiceFlowStartRecognize(AudioRecognizeRequest audioRecognizeReques
t, int i) {
// Start recognizing the audio stream
}
@Override
public void onVoiceFlowFinishRecognize(AudioRecognizeRequest audioRecognizeReques
t, int i) {
// End recognizing the audio stream
}
@Override
public void onVoiceVolume(AudioRecognizeRequest audioRecognizeRequest, int i) {
// Call back the volume
}
};
```

```
/**
* Return the audio stream
* to the host layer for recording caching.
* As the method runs on the SDK thread, it is generally used for file operations
here, and the host needs to open a new thread for implementing the business logic
* `new AudioRecordDataSource(true)` must be valid; otherwise, the function will n
ot be called back
* @param audioDatas
*/
@Override
public void onNextAudioData(final short[] audioDatas, final int readBufferLength)
{
}
```

Descriptions of other important classes

AudioRecognizeRequest

If both `templateName` and `customTemplate` are set, `templateName` will be used preferably.

Parameter	Type	Required	Description	Default Value
pcmAudioDataSource	PcmAudioDataSource	Yes	Audio data source	None
templateName	String	No	Template name set in the console	None
customTemplate	AudioRecognizeTemplate	No	Custom template	("16k_zh", 1)

AudioRecognizeResult

Speech recognition result object, which corresponds to the `AudioRecognizeRequest` object and is used to return the speech recognition result.

Parameter	Type	Description
code	Int	Recognition status code
message	String	Recognition prompt message
text	String	Recognition result
seq	Int	Sequence number of the audio segment
voiceId	String	ID of the audio stream of the audio segment

Parameter	Type	Description
cookie	String	Cookie value

AudioRecognizeTemplate

Custom audio template, for which you need to set the following parameters:

Parameter	Type	Required	Description
engineModelType	String	Yes	Engine model type
resType	Int	Yes	Result return method

Sample:

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

PcmAudioDataSource

This API class can be implemented to recognize mono-channel PCM audio data with a sample rate of 16 kHz. It mainly includes the following APIs:

- Add data to the speech recognizer: copy the data with the length of `length` starting from subscript 0 to the `audioPcmData` array, and the actual length of the copied data will be returned.

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

- Callback function when recognition is started, where you can perform initialization.

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

- Callback function when recognition is ended, where you can perform clearing.

```
void stop();
```

- Get the path of the SDK recording source file in PCM format.

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

- Get the path of the SDK recording source file in WAV format.

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

- Set the maximum amount of data read by the speech recognizer each time.

```
int maxLengthOnceRead();
```

AudioRecordDataSource

Implementation class of the `PcmAudioDataSource` API, which can directly read the audio data input by the mic for real-time recognition.

AudioFileDataSource

Implementation class of the `PcmAudioDataSource` API, which can directly read mono-channel PCM audio data files with a sample rate of 16 kHz.

> Note：
> Data in other formats cannot be recognized accurately.

AAILogger

You can use `AAILogger` to choose to output logs at the DEBUG, INFO, WARN, or ERROR level.

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

Guide for Local Audio Data Caching

You can choose to save audios in the host layer locally by following the steps below:

1. Set `isSaveAudioRecordFiles` to `true` during the initialization of `new AudioRecordDataSource(isSaveAudioRecordFiles)`.
2. Add the file logic for creating the recording in the `AudioRecognizeStateListener.onStartRecord` callback function. You can customize the path and filename.
3. Add the stream closing logic in the `AudioRecognizeStateListener.onStopRecord` callback function and optionally save PCM files as WAV files.
4. Add the logic for writing audio streams to local files in the `AudioRecognizeStateListener.onNextAudioData` callback function.
5. As the callback functions all run on the SDK thread, to avoid slow writes that may affect the internal running smoothness of the SDK, we recommend you complete the above steps in a single thread pool. For more information, see the sample code in the `MainActivity` class in the demo project.