**ASR流式识别使用手册**

**百度智能云团队提供**

**(版权所有,翻版必究)**

**目 录**

[1 概述 3](#_Toc54014715)

[1.1 支持的语音格式 3](#_Toc54014716)

[1.2 服务地址 3](#_Toc54014717)

[1.3 交互流程 3](#_Toc54014718)

[2 调用说明 3](#_Toc54014719)

[2.1 Python 3](#_Toc54014720)

[2.2 Java 7](#_Toc54014721)

[2.3 C++ 10](#_Toc54014722)

[2.4 Object-C 12](#_Toc54014723)

[3 返回结果(protobuf message) 14](#_Toc54014724)

[3.1 AudioFragmentResponse 字段表 15](#_Toc54014725)

[3.2 AudioFragmentResult 字段表 15](#_Toc54014726)

[3.2 错误码 15](#_Toc54014727)

## 1 概述

目前SDK支持C++、Java、Python等语言 使用GRPC实现了一个双向流的传输，即client传输音频流的同时，server会实时把识别的结果返回给client。

每种语言都有 demo code，涵盖了同步/异步等各种使用场景。用户如果想要多线程并发识别，需要在自己的项目 中自行实现多线程调度逻辑。

项目开源连接：https://github.com/baidubce/pie

### 1.1 支持的语音格式

* 支持的格式有:pcm(不压缩)、wav(不压缩，pcm编码)、amr(压缩格式)
* 原始 PCM 的录音参数必须符合 8k/16k 采样率、16bit 位深、单声道
* 仅支持 8000Hz和16000Hz 的采样率

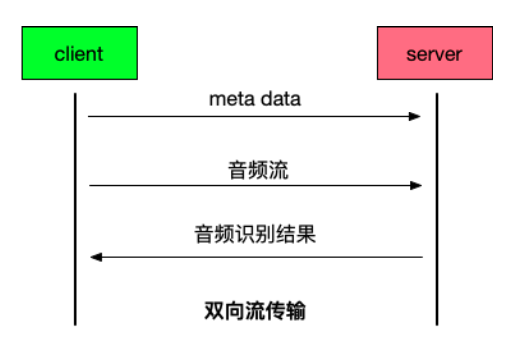
### 1.2 服务地址

asr.baiduai.cloud:8051

访问服务需要在 client 里配置 username 和 password，请联系百度人员获取。

### 1.3 交互流程

通过GRPC双向流的方式，能实时识别音频流，返回识别结果



## 2 调用说明

### 2.1 Python

#### 2.1.1使用要求

* python版本目前支持识别单音频文件、mac麦克风音频流、url音频流、管道音频流
* 支持python 2.7、python 3.x
* 依赖包:grpcio、protobuf、threadpool

代码可见：https://github.com/baidubce/pie/blob/master/audio-streaming-client- python/client\_demo.py

#### 2.1.2 client demo& metadata

* 执行pip install baidu-acu-asr安装sdk client初始化
* client初始化

url、port为服务的ip和端口，product id根据上述metadata设置，user\_name和password为用户验证信 息，product\_id和enalbe\_flush\_data为初始化必填项，其他metadata为选填项，可参照metadata项设置



1. client = AsrClient(url, port, product\_id, enable\_flush\_data,

2. log\_level=log\_level,

3. user\_name=user\_name,

4. password=password)

* metadata

|  |  |  |  |
| --- | --- | --- | --- |
| 参数 | 类型 | 默认值 | 描述 |
| server\_ip | string | / | 服务地址，直接填 asr.baiduai.cloud 即可 |
| server\_port | int | / | 服务端口，8051 |
| product | enum | / | asr 模型的类型，详见上文“ASR 模型列表”的表格 |
| app\_name | string | / | 可自己设定，描述自己的应用，设置规则为: 长度大于0小于255 不能包含 $ / \ \* ? < > | 等特殊字符 中间不能包含空格 |
| log\_level | int | 4 | log级别， 0:Trace，1:Debug， 2:Info，3:Warning， 4:Error，5:Fatal，6:关闭log |
| user\_name | string | / | 请联系百度人员获取 |
| password | string | / | 请联系百度人员获取 |
| enable\_long\_spe ech | bool | True | 是否允许长语音，本字段不需要手动设置 |
| enable\_flush\_da ta | bool | / | 是否逐字输出。若为False，则只会在一整句话识别完成后输 出 |
| enable\_chunk | bool | True | 是否允许服务端chunk发送 |
| sample\_point\_by tes | int | 2 | / |
| send\_per\_second s | double | 0.02 | 指定每次发送的音频数据包大小，数值越大，识别越快，但准 确率可能下降 |
| sleep\_ratio | double | 1 | 指定asr服务的识别间隔，数值越小，识别越快，但准确率可 能下降 |

#### 2.1.3 单音频文件识别

单音频文件识别包括失败重传机制，重传次数为30次，间隔0.5秒



1. for i in range(30):
2. responses = client.get\_result("/audio/path.pcm")
3. try:
4. for response in responses:

5. if response.type == baidu\_acu\_asr.audio\_streaming\_pb2.FRAGMENT\_DATA:

|  |  |
| --- | --- |
| 6. | logging.info("%s\t%s\t%s\t%s", |
| 7. | response.audio\_fragment.start\_time, |
| 8. | response.audio\_fragment.end\_time, |
| 9. | response.audio\_fragment.result, |
| 10. | response.audio\_fragment.serial\_num) |
| 11. | else: |
| 12. | logging.warning("type is: %d", response.type) |
| 13. |  |



1. break
2. except Exception as ex:
3. # 如果出现异常，此处需要重试当前音频
4. logging.error("encounter an error: %s, will create a new channel and retry audio! times
   * %d",

|  |  |
| --- | --- |
| 18. | ex.message, i + 1) |

1. time.sleep(0.5)

#### 2.1.4 读取麦克风音频流识别

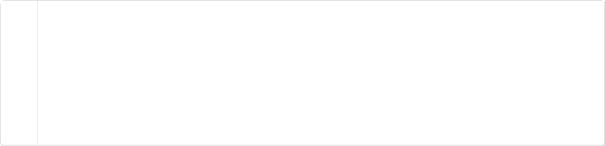


1. # 产生流（mac上麦克风读取音频流，需要先brew install portaudio）
2. def record\_micro():
3. NUM\_SAMPLES = 2560 # pyaudio内置缓冲大小
4. SAMPLING\_RATE = 8000 # 取样频率
5. pa = PyAudio()
6. stream = pa.open(format=paInt16, channels=1, rate=SAMPLING\_RATE, input=True, frames\_per\_buffer=NUM\_SAMPLES)
7. while True:
8. yield client.generate\_stream\_request(stream.read(NUM\_SAMPLES))

9.

1. def run\_file\_stream():
2. responses = client.get\_result\_by\_stream(generate\_file\_stream())
3. for response in responses:
4. logging.info("%s\t%s\t%s\t%s", response.start\_time, response.end\_time, response.result, response.serial\_num)

#### 2.1.5 读取url音频流识别



1. def read\_streaming\_from\_url():
2. data = urllib2.urlopen(audio\_url)
3. while True:
4. yield client.generate\_stream\_request(data.read(size=2560))
5. def run\_url\_streaming():
6. responses = client.get\_result\_by\_stream(read\_streaming\_from\_url())
7. for response in responses:
8. logging.info("%s\t%s\t%s\t%s", response.start\_time, response.end\_time, response.result, response.serial\_num)

#### 2.1.6 读取管道音频流识别



1. def general\_fifo\_stream():
2. """
3. 读取管道数据
4. 1.新建管道：mkfifo pipe.wav
5. 2.获取流存入管道：ffmpeg -i "http://path/of/video/stream" -vn -acodec pcm\_s16le -ac 1 -ar 8000 -f wav pipe:1 > pipe.wav
6. :return:
7. """
8. rf = os.open("/path/of/pipe.wav", os.O\_RDONLY)
9. while True:
10. stream = os.read(rf, 320)
11. yield client.generate\_stream\_request(stream)
12. def run\_fifo\_stream():
13. responses = client.get\_result\_by\_stream(general\_fifo\_stream())
14. for response in responses:
15. logging.info("%s\t%s\t%s\t%s", response.start\_time, response.end\_time, response.result, response.serial\_num)

### 2.2 Java

#### 2.2.1版本及依赖

客户端会以 jar 包的形式发布到 maven 仓库中，

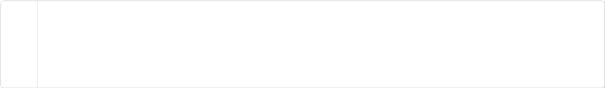
最新的发布版本在 <https://bintray.com/cynricshu/MavenRepo/audio-streaming-client-java>

* **使用要求**

- 支持 jdk1.7及以上版本  
- 使用 gradle 或 maven 构建项目

* **使用方式**

gradle项目

添加 jcenter 到 repository 中，并添加依赖:

1. repositories {
2. jcenter()
3. }
4. dependencies {
5. compile "com.baidu.acu.pie:audio-streaming-client-java:1.1.7"
6. }

maven项目

同样添加 jcenter 到 repository 中，并添加依赖:



1. <repositories>
2. <repository>
3. <id>jcenter</id>
4. <url>https://jcenter.bintray.com</url>
5. </repository>
6. </repositories>

7.

1. <dependencies>
2. <dependency>
3. <groupId>com.baidu.acu.pie</groupId>
4. <artifactId>audio-streaming-client-java</artifactId>
5. <version>1.1.9</version>
6. </dependency>
7. </dependencies>

#### 2.2.2demo code

请直接参考 https://github.com/baidubce/pie/tree/master/java-demo

#### 2.2.3 核心对象解释

在 demo code 中可以看到完整的使用代码。其中有几个 java class 比较重要，单独说明一下

* **AsrConfig**

在创建 AsrClient 的时候，需要传入一个 AsrConfig，顾名思义，就是一些核心的配置

|  |  |  |  |
| --- | --- | --- | --- |
| 参数 | 类型 | 默认值 | 描述 |
| serverIp | string | / | 服务地址，直接填 asr.baiduai.cloud 即可 |
| serverPort | int | / | 服务端口，8051 |
| product | enum | / | asr 模型的类型，详见上文“ASR 模型列表”的表格 |
| appName | string | / | 可自己设定，描述自己的应用 |
| log\_level | int | 4 | log级别，0:Trace，1:Debug， 2:Info，3:Warning，4:Error，5:Fatal，6:关闭log |
| userName | string | / | 请联系百度人员获取 |
| password | string | / | 请联系百度人员获取 |
| token | string | / | 如果填写了用户名和密码，就不需要填这个 |
| expireDateT ime | joda DateTi me | / | 如果填写了用户名和密码，就不需要填这个 |

* RequestMeta

在每次发起识别请求的时候，可以传入一个 RequestMeta 对象，用来控制识别速率，识别返回模式等等。

|  |  |  |  |
| --- | --- | --- | --- |
| 参数 | 类型 | 默认值 | 描述 |
| enableFlushData | bool | True | 是否逐字输出。若为False，则只会在一整句话识别完成后输 出 |
| sendPerSeconds | double | 0.02 | 每一帧发送0.02秒的音频数据，本参数 99.99% 情况下不用修 改。 |
| sendPackageRati o | double | 1.0 | 用来控制发包大小，数值越大，识别速度越快，但准确率可能 下降。 |
| sleepRatio | double | 1.0 | 用来控制发包间隔，数值越小，识别速度越快，但准确率可能 下降。 对接真人说话的实时场景下，请设置为 0。 离线文件识别场景下，不能小于 0.7 |
| timeoutMinutes | int | 10 | 识别单个文件的最大等待时间，默认10分，最长不能超过120 分 |

* RecognitionResult

是 sdk 用来保存底层识别结果的对象，在底层返回的 protobuf message的基础上进行了简单的封装

|  |  |  |  |
| --- | --- | --- | --- |
| 参数 | 类型 | 默认值 | 描述 |
| startTime | LocalTime | / | 句子在整个音频中的开始时间 |
| endTime | LocalTime | / | 句子在整个音频中的结束时间 |
| result | String | //1.0 | 识别结果文本 |
| serialNum | String | / | 本句话的唯一标识 |
| traceId | String | / | 本次 asr 请求的唯一标识，一次 asr 请求里面可以包含多句话 |
| isCompleted | bool | / | 该值为 true 的时候，表示是一整句的识别结果 |

### 2.3 C++

#### 2.3.1版本及依赖

C++版本目前支持识别音频文件、音频流，代码可见:https://github.com/baidubce/pie/tree/master/audio-streaming-client-cpp

版本要求:

* 编译器支持c++11 及以上版本
* 使用 cmake , make 构建项目

依赖要求:

* 需要安装:grpc , protobuf3.0+ , gflags

#### 2.3.2demo code（单线程/多线程）

* 单线程调用demo

完整文件在demo文件夹下



1. // specify audio path : 假设有两个音频文件
2. std::string audio\_file[2];
3. audio\_file[0] = "../../data/10s.wav";
4. audio\_file[1] = "../../data/8k-0.pcm";
5. // create AsrClient
6. AsrClient client;
7. client.set\_enable\_flush\_data(true);
8. client.set\_product\_id(argv[1]);
9. client.init(argv[2]);
10. // 创建stream
11. AsrStream\* stream = client.get\_stream();
12. // 打开音频文件
13. FILE\* fp = fopen(audio\_file[i].c\_str(), "rb");
14. // write to stream continuously ：i 为第几个音频文件，如果不需要可以在write\_to\_stream中删除此参数
15. std::thread writer(write\_to\_stream, client, stream, fp, i);
16. // read from stream continuously :持续读取识别结果，
17. while (1) {
18. if (stream->read(default\_callback, tmp) != 0) {
19. break;
20. }
21. }
22. // 等待writer线程结束写入
23. if (writer.joinable()) {
24. writer.join();
25. }
26. // 销毁stream
27. client.destroy\_stream(stream);

* 多线程调用demo

完整文件在demo文件夹下



1. // create AsrClient
2. AsrClient client;
3. // create threads
4. std::thread recog\_threads[thread\_count];
5. for (int i = 0; i < thread\_count; i++) {
6. recog\_threads[i] = std::thread(do\_recog, &client, audio\_file[i], i);
7. }
8. // wait threads complete
9. for (int i = 0; i < thread\_count; i++) {
10. if (recog\_threads[i].joinable()) {
11. recog\_threads[i].join();
12. }
13. }
14. //读写stream的调用方式与单线程一样

### 2.4 Object-C

#### 2.4.1版本及依赖

Object-C版本目前支持麦克风录制音频文件识别、麦克风实时音频流识别 代码可见：<https://github.com/baidubce/pie/blob/master/ios-demo/BaiduARS>

版本要求:

支持iOS 8.0+

依赖包：详见工程代码中的Podfile文件

#### demo code

1、执行pod install 安装sdk运行所需要的依赖库文件

2、通过BaiduASR.xcworkspace文件打开工程，编译运行

3、Instance初始化 url、port为服务的ip和端口，product id根据上述metadata设置，username和password为用户验证信息，productid和sampleRate音频采样率为初始化必填项。设置delegate为self，并按需实现BDASRDelegate的几个代理方法。

self.asrInstance = [BDASRInstance shareInstance];

[self.asrInstance setUserName:user\_name passWord:password];

[self.asrInstance setHostAddress:url];

[self.asrInstance setServerPort:port];

[self.asrInstance setProductId:product\_id];

[self.asrInstance setSampleRate:sampleRate];

[self.asrInstance setDelegate:self];

#### 2.4.3 metadata

| **参数** | **类型** | **默认值** | **描述** |
| --- | --- | --- | --- |
| url | string | / | 服务地址，直接填 asr.baiduai.cloud 即可 |
| port | int | / | 服务端口，8051 |
| product\_id | enum | / | asr 模型的类型，详见上文“ASR 模型列表”的表格 |
| user\_name | string | / | 请联系百度人员获取 |
| password | string | / | 请联系百度人员获取 |
| sampleRate | string | / | 音频采样率，请填写与 product\_id相对应的正确的音频采样率，详见上文“ASR 模型列表”的表格 |

**麦克风录制音频文件识别**

麦克风录制音频文件识别逻辑已经封装到SDK，其中调用麦克风录制文件，以及音频文件的存储和传输逻辑可在BaiduASR模块中找到，无需接入方自行实现，如果有特殊的音频处理需求可自行修改代码逻辑。 调用startRecord 开始音频录制，录音结束后调用stopRecord，此时会对音频文件处理并传输给server进行识别，识别完成后会回调- (void)bdasrAnalizeDone:(BOOL)done result:(id \_Nullable )result error:(NSError\* \_Nullable)error;方法，具体结果解析可参照demo中的代码逻辑。

// 开始录制

[self.asrInstance startRecord];

// 停止录制

[self.asrInstance stopRecord];

// 回调

- (void)bdasrAnalizeDone:(BOOL)done result:(id \_Nullable )result error:(NSError\* \_Nullable)error;

**麦克风实时音频流识别**

与麦克风录制音频文件识别类似，实时音频流识别逻辑也完整封装到SDK。逻辑可在BaiduASR模块中找到，无需接入方自行实现，如果有特殊的音频处理需求可自行修改代码逻辑。 调用startRealTimeRecord 开始设备实时收音识别，调用stopRealTimeRecord 结束实时收音识别。调用startRealTimeRecord 后，麦克风会实时采集音频数据，同时实时发送给server端。识别结果也会实时地通过 (void)bdasrRealTimeAnalizeResult:(id \_Nullable )result error:(NSError\* \_Nullable)error;回调给客户端，具体结果解析可参照demo中的代码逻辑。

// 开始实时识别

[self.asrInstance startRealTimeRecord];

// 停止实时识别

[self.asrInstance stopRealTimeRecord];

// 回调

- (void)bdasrRealTimeAnalizeResult:(id \_Nullable )result error:(NSError\* \_Nullable)error;

## 3 返回结果(protobuf message)

返回结果是一个两层嵌套的对象。 最外层是AudioFragmentResponse，里面是一些通用的字段，如 error\_code，error\_message 等。

error\_message 里面会夹带一个32位的 uuid，是这次请求的唯一标识，用于 trace，当发生错误时，可以将这个提供给开发人员，帮助定位问题。

如果请求成功，那么 error\_code为 0，此时可以从 audio\_fragment 字段中取出识别结果。



1. message AudioFragmentResponse {
2. int32 error\_code = 1;
3. string error\_message = 2;
4. ResponseType type = 3;
5. AudioFragmentResult audio\_fragment = 4;
6. }

7.

1. message AudioFragmentResult {
2. string start\_time = 1;
3. string end\_time = 2;
4. string result = 3;
5. bool completed = 4;
6. string serial\_num = 5;
7. }

### 3.1 AudioFragmentResponse 字段表

|  |  |  |
| --- | --- | --- |
| 参数 | 类型 | 描述 |
| error\_code | int | 错误码，0表示请求成功，否则请求失败，具体错误码详情见下 |
| error\_message | string | 错误详情，请求成功该值为空，请求失败会有详情说明 |
| type | enum | UNKNOWN\_TYPE, FRAGMENT\_DATA 二选一，暂时没有其他取值 |
| audio\_fragment | AudioFragmentResult对象 | 识别结果详情，见下表 |

### 3.2 AudioFragmentResult 字段表

|  |  |  |
| --- | --- | --- |
| 参数 | 类型 | 描述 |
| start\_time | string | 识别出来的文字对应音频开始时间 |
| end\_time | string | 识别出来的文字对应音频结束时间 |
| result | string | 识别出来的文字 |
| completed | bool | true 表示是一整句识别结果，false 表示是单字识别结果 |
| serial\_num | string | 本句话的唯一标识，一次asr识别请求可以包含多句话 |

### 3.2 错误码

|  |  |  |
| --- | --- | --- |
|  |  |  |
| error\_code | error\_message | 描述 |
| -1000 | Cannot find audio meta | 在创建 client 的时候，没有设置 config |
| -1000 | Failed to parse audio met a | 在创建 client 的时候，config 设置错误 |
| -1000 | Failed to check user xxx | 用户名密码不正确 |
| -1000 | Invalid product id | 使用了不合法的 asr product 类型，请看上文“ASR 模型列表 ” |
| -1000 | Internal server error | 内部错误，请联系百度人员同时提供 logid 等信息便于 debug |