实时语音识别

这段 Python 代码使用 Google Cloud Speech-to-Text API 和 PyAudio 库实现实时语音识别。它从麦克风捕获音频，将其流式传输到 Speech-to-Text API，并打印转录文本。`MicrophoneStream` 类处理音频输入，`main` 函数设置语音识别客户端并处理音频流。

```
import os
import argparse
import io
import sys
import time

from google.cloud import speech

import pyaudio
from six.moves import queue


# 音频录制参数
RATE = 16000
CHUNK = int(RATE / 10)  # 100ms


class MicrophoneStream(object):
    """ 打开一个录音流，作为生成器产生音频块。 """
    def __init__(self, rate, chunk):
        self._rate = rate
        self._chunk = chunk

        # 使用 PyAudio 创建一个音频接口
        self._audio_interface = pyaudio.PyAudio()
        self._audio_stream = self._audio_interface.open(
            format=pyaudio.paInt16,
            # API 目前仅支持单声道音频
            # https://goo.gl/z726ff
            channels=1, rate=self._rate,
            input=True, frames_per_buffer=self._chunk,
            # 异步运行音频流以填充缓冲区对象。
            # 这是必要的，以便在调用线程进行网络请求等操作时，输入设备的缓冲区不会溢出。
            stream_callback=self._fill_buffer,
```

```
        )
        self.closed = False
        self._buff = queue.Queue()

    def _fill_buffer(self, in_data, frame_count, time_info, status_flags):
        """ 持续从音频流收集数据到缓冲区。 """
        self._buff.put(in_data)
        return None, pyaudio.paContinue

    def generator(self, record_seconds):
        start_time = time.time()
        while not self.closed and time.time() - start_time < record_seconds:
            # 使用阻塞的 get() 确保至少有一块数据，如果块为 None 则停止迭代，表示音频流结束。
            chunk = self._buff.get()
            if chunk is None:
                return
            data = [chunk]

            # 现在使用任何其他缓冲数据。
            while True:
                try:
                    chunk = self._buff.get(block=False)
                    if chunk is None:
                        return
                    data.append(chunk)
                except queue.Empty:
                    break

            yield b''.join(data)

    def close(self):
        self.closed = True
        # 向生成器发出终止信号，以便客户端的流式识别方法不会阻塞进程终止。
        self._buff.put(None)
        self._audio_stream.close()
        self._audio_interface.terminate()

    def __enter__(self):
        return self
```

```
    def __exit__(self, type, value, traceback):
        self.close()


def main(record_seconds=10, language_code='en-US'):
    # 请参阅 http://g.co/cloud/speech/docs/languages
    # 获取支持的语言列表。
    # language_code = 'en-US'  # BCP-47 语言标签

    client = speech.SpeechClient()
    config = speech.RecognitionConfig(
        encoding=speech.RecognitionConfig.AudioEncoding.LINEAR16,
        sample_rate_hertz=RATE,
        language_code=language_code,
        model="latest_long",
    )

    streaming_config = speech.StreamingRecognitionConfig(
        config=config,
        interim_results=True)

    with MicrophoneStream(RATE, CHUNK) as stream:
        audio_generator = stream.generator(record_seconds)
        requests = (speech.StreamingRecognizeRequest(audio_content=content)
                    for content in audio_generator)

        responses = client.streaming_recognize(streaming_config, requests)

        # 现在，使用转录响应。
        transcript = ""
        for response in responses:
            print(response)
            # 转录完成后，打印结果。
            for result in response.results:
                if result.is_final:
                    alternative = result.alternatives[0]
                    transcript += alternative.transcript + " "
        print(u'Transcript: {}'.format(transcript))
```

```
if __name__ == '__main__':
    parser = argparse.ArgumentParser(description=" 可调节持续时间的实时语音识别。")
    parser.add_argument('--duration', type=int, default=10, help=" 录音时长（秒）。")
    parser.add_argument('--language_code', type=str, default='en-US', help=" 转录的语言代码。")
    args = parser.parse_args()
    print(" 请开始说话...")
    main(record_seconds=args.duration, language_code=args.language_code)
```