

Python Project II 小组项目说明





小组成员：苏珊、杨若瑶、朱莹、徐林源

项目主题：多语种自动语音翻译系统开发 —— 语音识别模块、机器翻译模块










项目周期：5 weeks

项目文件列表：

项目管理文件：

-  Python Group Project.pptx
-  Python小组项目说明_201807.docx
-  gantt-chart_Python.xlsx
-  to-do-list_Python.xlsx

程序代码文件：

-  __init__.py
-  __main__.py
-  baidu.py
-  bing_lang.txt
-  machine_translation.py
-  speech_recognize.py
-  speech_synthesis.py
-  sr_mt_ss.py
-  youdao.py

__main__.py-- 参数设置、测试代码

sr_mt_ss.py -- 三个模块封装成一大类

speech_recognize.py -- 语音识别模块封装成类

machine_translation.py -- 机器翻译模块封装成类

















speech_synthesis.py -- 语音合成模块封装成类（假）

bing_lang.txt -- bing 语音识别支持语言

youdao.py -- 有道机器翻译封装

baidu.py -- 百度机器翻译封装

测试文件：

-  ch-long-non-stop.wav
-  ch-long-stop.wav
-  ch-short.wav
-  en-long-non-stop.wav
-  en-long-stop.wav
-  en-short.wav
-  fr_Le Papillon_part.wav
-  long_ch.wav
-  test.wav
-  test_ch.wav
-  test_en.wav
-  test_en_long.wav
-  zh.aiff
-  zh.flac
-  zh.raw
-  zh.wav

环境需求：需安装语音识别工具包、麦克风工具包。在 cmd 中运行如下两行代码：

```
pip install SpeechRecognition
```

```
pip install pyaudio
```

一、选题说明

1. 语音识别

(1) 语言识别工作原理概述

语音通过麦克风，从物理声音转换为电信号，然后通过模数转换器转换为数据。一旦被数字化，就可适用若干种模型，将音频转录为文本。

多数现代语音识别系统都依赖于隐马尔可夫模型（HMM）。其工作原理为：语音信号在非常短的时间尺度上（比如 10 毫秒）可被近似为静止过程，即一个其统计特性不随时间变化的过程。

许多现代语音识别系统会在 HMM 识别之前使用神经网络，通过特征变换和降维的技术来简化语音信号。也可以使用语音活动检测器（VAD）将音频信号减少到可能仅包含语音的部分。

(2) 选择 Python 语音识别包

PyPI 中有一些现成的语音识别软件包，其中包括：apiai/google-cloud-speech/pocketsphinx/SpeechRecognition/watson-developer-cloud/wit。

wit 和 apiai：提供了一些超出基本语音识别的内置功能，如识别讲话者意图的自然语言处理功能。

谷歌云语音：专注于语音向文本的转换。

SpeechRecognition：已构建完成访问麦克风和从头开始处理音频文件的脚本。

参考资料：<https://blog.csdn.net/dQCFKyQDXym3F8rB0/article/details/79832700>

2. 机器翻译

机器翻译又称为自动翻译，是利用计算机将一种自然语言（源语言）转换为另一种自然语言（目标语言）的过程。它是计算语言学的一个分支，是人工智能的终极目标之一，具有重要的科学研究价值。

本项目中机器翻译作为第二部分，用来接受语音识别结果并调用 api 进行翻译，最后返回结果。

二、小组分工与项目进度控制

详见 gantt-chart_Python.xlsx 和 to-do-list_Python.xlsx 文件

1	项目整体	
1.1	理解语音识别、机器翻译大致原理	小组合作
1.2	小组讨论了解到的知识，规划组内分工方式	小组合作
1.3	交流进度、代码调整	小组合作
1.4	交流进度、确定接口格式	小组合作
1.5	分别完成各模块代码基本功能	小组分工
1.5.1	完成语音识别代码	苏珊、杨若瑶
1.5.2	完成机器翻译代码	朱莹、徐林源
1.5.3	完善代码细节、扩展功能	小组
1.6	整合联通三个模块代码	小组合作
1.7	整理完善编码文档	小组合作
1.8	准备期末汇报文档	小组合作
1.9	整理整合项目管理文档	苏珊

2 语音识别			3 机器翻译		
2.1	理解原理、检索可参考代码	苏珊、杨若瑶	3.1	查找资料、理解原理	朱莹、徐林源
2.2	设计功能及实现方式	苏珊、杨若瑶	3.2	小组讨论、设计功能	朱莹、徐林源
2.3	例子代码分析讨论	苏珊、杨若瑶	3.3	阅读例子代码、更改设计构架	朱莹、徐林源
2.4	构建基础代码框架	苏珊、杨若瑶	3.4	设计实现的类和方法	朱莹、徐林源
2.5	每人实现1个API调用	苏珊、杨若瑶	3.5	每人实现一个api调用	朱莹、徐林源
2.6	完善整体功能并封装成类	苏珊、杨若瑶	3.6	根据语音识别部分设置传参接口	朱莹、徐林源
2.7	拆分细化代码	杨若瑶	3.7	API合并	朱莹、徐林源
2.8	代码异常处理	杨若瑶	3.8	百度翻译API调整	朱莹、徐林源
2.9	拓展代码功能	苏珊、杨若瑶	3.9	增加更多语言对	朱莹、徐林源
2.10	支持更多语言识别	苏珊、杨若瑶	3.10	完善代码注释	朱莹、徐林源
2.11	整理完善编码文档	苏珊、杨若瑶	3.11	优化代码（语言对）	朱莹、徐林源
2.12	代码整体测试与差错处理	苏珊、杨若瑶			

三、多语种自动语音翻译系统整体流程：

1. 自然语言输入：
 - 1) 麦克风直接输入或音频文件输入
2. 语音识别：
 - 1) 语音获取（自然输入实践）
 - 2) 语言设置选择
 - 3) 输入格式选择
 - 4) 调用语音识别 API
 - 5) 整理输出结果并选择存储位置。
3. 机器翻译：
 - 1) 文字获取
 - 2) 语言设置选择
 - 3) 调用机器翻译 API
 - 4) 整理输出结果并选择存储位置
4. 语音合成：
 - 1) 文字获取
 - 2) 选择语言
 - 3) 调用语音合成 API
 - 4) 整理输出结果并选择存储位置

（1）语音识别流程（步骤参考代码）

#1 从麦克风获取音频，将音频写入 RAW/WAV/AIFF/FLAC 文件

获取音频，将音频转换为所需格式

https://github.com/Uberi/speech_recognition/blob/master/examples/write_audio.py

#2 消除噪声对语音识别的影响，默认将文件流的第一秒识别为音频的噪声级别

https://github.com/Uberi/speech_recognition/blob/master/examples/calibrate_energy_threshold.py

#3 选择语言

#4 使用特定路径下的音频文件作为音频源，使用 Sphinx/Google 语音识别功能/Google Cloud Speech/Wit.ai/Microsoft Bing Voice Recognition...进行语音识别

https://github.com/Uberi/speech_recognition/blob/master/examples/audio_transcribe.py

https://github.com/Uberi/speech_recognition/blob/master/examples/extended_results.py (多个返回结果，第一个为最佳结果)

```
# recognize speech using Sphinx
try:
    print("Sphinx thinks you said " + r.recognize_sphinx(audio))
except sr.UnknownValueError:
    print("Sphinx could not understand audio")
except sr.RequestError as e:
    print("Sphinx error; {0}".format(e))
```

#5 将语音识别出的信息存储在相应位置，统一转换为机器翻译需要的格式

例子代码运行注意事项：环境搭建、库的官方文档（其他功能介绍）

(2) 机器翻译流程：

- #1 登录
- #2 读取文件
- #3 选择语言
- #4 加密用户名密码信息
- #5 获取 API 返回翻译结果
- #6 保存结果文件，转换成下一接口接受格式输出

四、项目注意事项记录：

代码包括：两个模块封装成类、测试资料与测试代码准备（测试结果展示）

编码文档：代码中的注释（文件级别、类级别、函数级别（参数、返回值、重点语句解释））

```
# @description: 构造函数
# @param string audio_file -- 读入的语音文件名称
# @param string in_lang --
# @param string out_lang --
# @return string body --
```

开发文档、报告模板：模板与包含事项

项目实施：前端 UI 和各模块间的连接、注意留有接口（测试时可应用假数据）

注释规范参考：

www.techug.com/post/comments-in-python.html

<https://www.cnblogs.com/Rubick7/p/7755337.html>

五、例子代码阅读：

语音识别：

https://github.com/Uberi/speech_recognition/blob/master/speech_recognition/_init_.py

<https://blog.csdn.net/dQCFKyQDXym3F8rB0/article/details/79832700>

5-34 引入库、环境配置（Python 2.3）

Urllib 是 python 内置的 HTTP 请求库 <https://www.cnblogs.com/zhaof/p/6910871.html>

37-54 异常错误检查

46-206 麦克风输入

```
class AudioSource(object):
class Microphone(AudioSource):
```

209-322 读取音频文件

```
class AudioFile(AudioSource):
    def __init__(self, filename_or_fileobject):
    def __enter__(self): 尝试以各种格式读取音频文件
    def __exit__(self, exc_type, exc_value, traceback):
class AudioFileStream(object): 音频格式转换
# hasattr() 函数用于判断对象是否包含对应的属性。
```

325-495 写入音频文件

```
class AudioData(object):
```

499-1262 语音识别

```
class Recognizer(AudioSource):
    def __init__(self):
    def record(self, source, duration=None, offset=None): 语音识别、开始、结束
    def adjust_for_ambient_noise(self, source, duration=1): 环境噪声
    def snowboy_wait_for_hot_word(self, snowboy_location, snowboy_hot_word_files,
source, timeout=None): 等待热词
    def listen(self, source, timeout=None, phrase_time_limit=None,
snowboy_configuration=None): 热词检测（包括断句相关内容）
    def listen_in_background(self, source, callback, phrase_time_limit=None): 后台监听器
    746 def recognize_sphinx(self, audio_data, language="en-US", keyword_entries=None,
grammar=None, show_all=False):
```

在`audio_data`（AudioData实例）上使用 CMU Sphinx 执行语音识别。

```
# import the PocketSphinx speech recognition module
# create decoder object
# obtain audio data
# obtain recognition results (keyword_entries, grammar)
# return results
```

```
845 def recognize_google(self, audio_data, key=None, language="en-US",
show_all=False):
```

```
# obtain audio transcription results
# ignore any blank blocks
# return results
```

```
906 def recognize_google_cloud(self, audio_data, credentials_json=None,
language="en-US", preferred_phrases=None, show_all=False):
```

```
981 def recognize_wit(self, audio_data, key, show_all=False):
```

```
1018 def recognize_bing(self, audio_data, key, language="en-US", show_all=False):
```

```
1111 def recognize_houndify(self, audio_data, client_id, client_key, show_all=False):
```

```
1164 def recognize_ibm(self, audio_data, username, password, language="en-US",
show_all=False): 有中文
```

```
def recognize_tensorflow(self, audio_data, tensor_graph='tensorflow-  
data/conv_actions_frozen.pb', tensor_label='tensorflow-data/conv_actions_labels.txt'): 引入  
tensorflow 数据画图
```

1265-1295 FLAC 转换器 def get_flac_converter():

1298-1304 python2 兼容性 def shutil_which(pgm):

1307-1333 文件可同时重复打开 class PortableNamedTemporaryFile(object):

1343-1360 另一个 API 语音识别，目前不适用 python3，收费项目

```
def recognize_api(self, audio_data, client_access_token, language="en", session_id=None,  
show_all=False):
```

五、问题及解决方案

1. 项目整体——不同文件的互相调用

解决方案：包和模块的应用；相互调用

同一文件目录下

在b.py文件中用下面两条语句即可完成对a.py文件中func()函数的调用

```
import a #引用模块  
a.func( )
```

或者是

```
import a#应用模块  
from a import func #引用模块中的函数  
func ( ) #这里调用函数就不需要加上模块名的前缀了
```

不同文件目录下

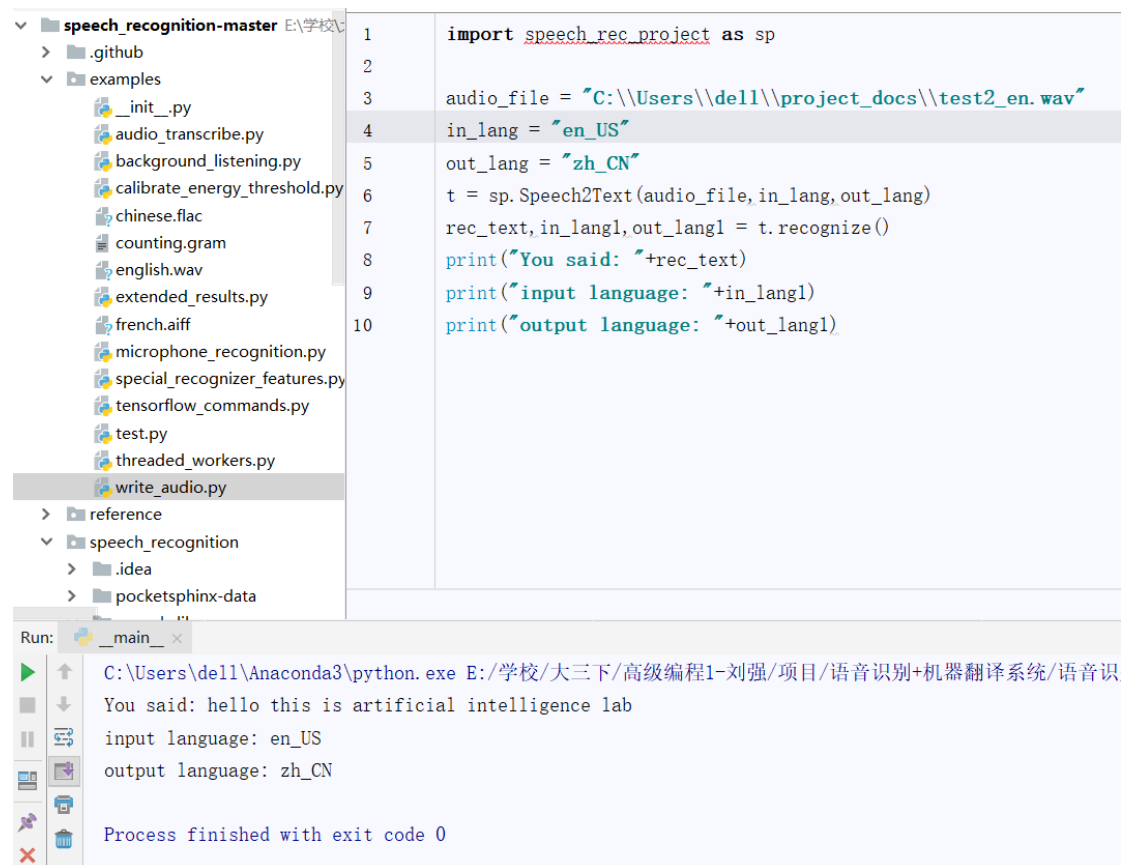
若不在同一目录，python查找不到，必须进行查找路径的设置，将模块所在的文件夹加入系统查找路径

```
import sys  
sys.path.append('a.py所在的路径')  
import a  
a.func()
```

2. 项目整体——接口统一问题

前端 > 语音识别： pcm 或 wav 格式音频，具体传送形式（文件名称？路径？）

语音识别 > 机器翻译：返回三个值，均为字符串形式



1) rec_text 语音识别结果

机器翻译用的时候需要判断一下，如果是空字符串（rec_text=""）就不要进行机器翻译，等待用户重新说话。

```

str=""
if str.strip()=="":
    print("str is null")
if not str.strip():
    print("str is null")

```

2) in_lang1 源语言

3) out_lang1 目标语言

language-Country	language-Country	language-Country	language-Country
de-DE	zh-TW	zh-HK	ru-RU
es-ES	ja-JP	ar-EG*	da-DK
en-GB	en-IN	fi-FI	nl-NL
en-US	pt-BR	pt-PT	ca-ES
fr-FR	ko-KR	en-NZ	nb-NO
it-IT	fr-CA	pl-PL	es-MX
zh-CN	en-AU	en-CA	sv-SE

（语音识别支持语言）

1	zh	中文
2	en	英语
3	yue	粤语
4	wyw	文言文
5	jp	日语
6	kor	韩语
7	fra	法语
8	spa	西班牙语
9	th	泰语
10	ara	阿拉伯语
11	ru	俄语
12	pt	葡萄牙语
13	de	德语
14	it	意大利语
15	el	希腊语
16	nl	荷兰语
17	pl	波兰语
18	bul	保加利亚语
19	est	爱沙尼亚语
20	dan	丹麦语
21	fin	芬兰语
22	cs	捷克语
23	rom	罗马尼亚语
◀ ▶		Sheet1

（百度机器翻译支持语言）

语言	代码
中文	zh-CHS
日文	ja
英文	EN
韩文	ko
法文	fr
阿拉伯文	ar
波兰文	pl
丹麦文	da
德文	de
俄文	ru
芬兰文	fi
荷兰文	nl
捷克文	cs
罗马尼亚文	ro
挪威文	no
葡萄牙文	pt
瑞典文	sv
斯洛伐克文	sk
西班牙文	es
印地文	hi
印度尼西亚文	id
意大利文	it
泰文	th
土耳其文	tr
希腊文	el
匈牙利文	hu

（有道机器翻译支持语言）

3. 机器翻译——语言对处理问题

解决方案：由于各个 api 支持的语言对格式不统一，所以在机器翻译进行之前需要对接收的语言对进行转换，采用了词典的方法，将语音识别支持的语言对作为 **key**，将机器翻译支持的语言对作为 **value**，转换后再进行翻译。如下图示例：

```

langlist_Youdao={
    'zh-CN':'zh-CHS',
    'en-US':'EN',
    'ja-JP':'ja',
    'ko-KR':'ko',
    'fr-FR':'fr',
    'es-ES':'es',
    'ru-RU':'ru',
    'de-DE':'de',
    'pt-PT':'pt',
    'pl-PL':'pl',
    'sv-SE':'sv',
    'ar-EG*':'ar',
    'nl-NL':'nl',
    'nb-NO':'no',
    'fi-FI':'fi',
    'da-DK':'da'
}

key_list=[]
value_list=[]
for key,value in langlist_Youdao.items():
    key_list.append(key)
    value_list.append(value)

class Youdao():
    """Description:有道翻译类"""
    def __init__(self,in_Lang,out_Lang):
        """
        :description:构造函数，初始化有道翻译所需参数
        :param str langFrom:源语言
        :param str langTo:目标语言
        :param str url:有道翻译api的连接
        :param str appKey:有道翻译API用户ID
        :param str appSecret:有道翻译API用户密钥
        """
        if in_Lang in key_list:
            key_index=key_list.index(in_Lang)
            langFrom=value_list[key_index]
        if out_Lang in key_list:
            key_index=key_list.index(out_Lang)
            langTo=value_list[key_index]

```

4. 语音识别——Bing 语音识别只识别到第一个停顿处，后面的语音输入被忽略；Houndify 语音识别能识别整个句子，但即使有意停顿，识别结果中也无标点。

解决方案:关于停顿相关问题目前可以不涉及。现采用在识别前向前端用户提示该信息，如此处与前端有交互，用户可选择暂停此次语音识别过程，断句后重新分别输入。

构想思路:连续语音识别（每一次调用不同的接口，后期的识别可以修正前期识别的结果）。

5. 机器翻译——调用 API 结构不同，无法整合在一起

解决方案:独自作为一个类，在外部进行封装

6. 语音识别——语音识别支持语言可以提前写在 txt 里，读取文件即可。

7. 项目整体——错误码单独定义为类或模块:大写字母名称 = 数字（便于理解）