goslate

Release 1.5.4

unknown

Jun 13, 2022

CONTENTS

Note: Google has updated its translation service recently with a ticket mechanism to prevent simple crawler programs like `goslate` from accessing. Though a more sophisticated crawler may still work technically, it would have crossed the fine line between using the service and breaking the service. `goslate` will not be updated to break google's ticket mechanism. Free lunch is over. Thanks for using.

`goslate` provides you *free* python API to google translation service by querying google translation website.

It is:

- **Free**: get translation through public google web site without fee
- **Fast**: batch, cache and concurrently fetch
- **Simple**: single file module, just `Goslate().translate('Hi!', 'zh')`

CHAPTER

ONE

SIMPLE USAGE

The basic usage is simple:

```
>>> import goslate
>>> gs = goslate.Goslate()
>>> print(gs.translate('hello world', 'de'))
hallo welt
```

CHAPTER

TWO

INSTALLATION

goslate support both Python2 and Python3. You could install it via:

```
$ pip install goslate
```

or just download latest goslate.py directly and use

`futures` package is optional but recommended to install for best performance in large text translation tasks.

CHAPTER

THREE

PROXY SUPPORT

Proxy support could be added as following:

```
import urllib2
import goslate

proxy_handler = urllib2.ProxyHandler({"http" : "http://proxy-domain.name:8080"})
proxy_opener = urllib2.build_opener(urllib2.HTTPHandler(proxy_handler),
                                    urllib2.HTTPSHandler(proxy_handler))

gs_with_proxy = goslate.Goslate(opener=proxy_opener)
translation = gs_with_proxy.translate("hello world", "de")
```

CHAPTER

FOUR

ROMANIZATION

Romanization or latinization (or romanisation, latinisation), in linguistics, is the conversion of writing from a different writing system to the Roman (Latin) script, or a system for doing so.

For example, pinyin is the default romanization method for Chinese language.

You could get translation in romanized writing as following:

```
>>> import goslate
>>> roman_gs = goslate.Goslate(writing=goslate.WRITING_ROMAN)
>>> print(roman_gs.translate('China', 'zh'))
Zhōngguó
```

You could also get translation in both native writing system and ramon writing system

```
>>> import goslate
>>> gs = goslate.Goslate(writing=goslate.WRITING_NATIVE_AND_ROMAN)
>>> gs.translate('China', 'zh')
('', 'Zhōngguó')
```

You could see the result will be a tuple in this case: `(Translation-in-Native-Writing, Translation-in-Roman-Writing)`

CHAPTER

FIVE

LANGUAGE DETECTION

Sometimes all you need is just find out which language the text is:

```
>>> import goslate
>>> gs = goslate.Goslate()
>>> language_id = gs.detect('hallo welt')
>>> language_id
'de'
>>> gs.get_languages()[language_id]
'German'
```

CHAPTER

SIX

CONCURRENT QUERYING

It is not necessary to roll your own multi-thread solution to speed up massive translation. Goslate has already done it for you. It utilizes `concurrent.futures` for concurrent querying. The max worker number is 120 by default.

The worker number could be changed as following:

```
>>> import goslate
>>> import concurrent.futures
>>> executor = concurrent.futures.ThreadPoolExecutor(max_workers=200)
>>> gs = goslate.Goslate(executor=executor)
>>> it = gs.translate(['text1', 'text2', 'text3'])
>>> list(it)
['translation1', 'translation2', 'translation3']
```

It is advised to install `concurrent.futures` backport lib in python2.7 (python3 has it by default) to enable concurrent querying.

The input could be list, tuple or any iterator, even the file object which iterate line by line

```
>>> translated_lines = gs.translate(open('readme.txt'))
>>> translation = '\n'.join(translated_lines)
```

Do not worry about short texts will increase the query time. Internally, goslate will join small text into one big text to reduce the unnecessary query round trips.

CHAPTER

SEVEN

BATCH TRANSLATION

Google translation does not support very long text, goslate bypasses this limitation by splitting the long text internally before sending it to Google and joining the multiple results into one translation text to the end user.

```
>>> import goslate
>>> with open('the game of thrones.txt', 'r') as f:
>>>     novel_text = f.read()
>>> gs = goslate.Goslate()
>>> gs.translate(novel_text)
```

CHAPTER

EIGHT

PERFORMANCE CONSIDERATION

Goslate uses batch and concurrent fetch aggressively to achieve maximized translation speed internally.

All you need to do is reduce API calling times by utilizing batch translation and concurrent querying.

For example, say if you want to translate 3 big text files. Instead of manually translate them one by one, line by line:

```
import goslate

big_files = ['a.txt', 'b.txt', 'c.txt']
gs = goslate.Goslate()

translation = []
for big_file in big_files:
    with open(big_file, 'r') as f:
        translated_lines = []
        for line in f:
            translated_line = gs.translate(line)
            translated_lines.append(translated_line)

        translation.append('\n'.join(translated_lines))
```

It is better to leave them to Goslate totally. The following code is not only simpler but also much faster (+100x) :

```
import goslate

big_files = ['a.txt', 'b.txt', 'c.txt']
gs = goslate.Goslate()

translation_iter = gs.translate(open(big_file, 'r').read() for big_file in big_files)
translation = list(translation_iter)
```

Internally, goslate will first adjust the text to make them not so big that do not fit Google query API, nor so small that increase the total HTTP querying times. Then it will use concurrent queries to speed things even further.

CHAPTER

NINE

LOOKUP DETAILS IN DICTIONARY

If you want detail dictionary explanation for a single word/phrase, you could

```
>>> import goslate
>>> gs = goslate.Goslate()
>>> gs.lookup_dictionary('sun', 'de')
[[['Sonne', 'sun', 0]],
 [['noun',
   ['Sonne'],
   [['Sonne', ['sun', 'Sun', 'Sol'], 0.44374731, 'die']],
   'sun',
   1],
  ['verb',
   ['der Sonne aussetzen'],
   [['der Sonne aussetzen', ['sun'], 1.1544633e-06]],
   'sun',
   2]],
 'en',
 0.9447732,
 [['en'], [0.9447732]]]
```

There are 2 limitations for this API:

- The result is a complex list structure which you have to parse for your own usage
- The input must be a single word/phase, batch translation and concurrent querying are not supported

CHAPTER

TEN

QUERY ERROR

If you get an HTTP 5xx error, it is probably because google has banned your client IP address from transaction querying.

You could verify it by accessing google translation service in the browser manually.

You could try the following to overcome this issue:

- query through a HTTP/SOCKS5 proxy, see *Proxy Support*
- using another google domain for translation: `gs = Goslate(service_urls=['http://translate.google.de'])`
- wait for 3 seconds before issue another querying

CHAPTER

ELEVEN

API REFERENCES

please check API reference

CHAPTER

TWELVE

COMMAND LINE INTERFACE

`goslate.py` is also a command line tool which you could use directly

- Translate `stdin` input into Chinese in GBK encoding

```
$ echo "hello world" | goslate.py -t zh-CN -o gbk
```

- Translate 2 text files into Chinese, output to UTF-8 file

```
$ goslate.py -t zh-CN -o utf-8 source/1.txt "source 2.txt" > output.txt
```

use `--help` for detail usage

```
$ goslate.py -h
```

CHAPTER

THIRTEEN

HOW TO CONTRIBUTE

- Report issues & suggestions
- Fork repository
- Donation

CHAPTER

FOURTEEN

WHAT'S NEW

14.1 1.5.4

- handle deprecated *threading.currentThread()* properly
- add *retry_wait_duration* param to fine control the retry behavior in case of connection error

14.2 1.5.2

- [fix bug] removes newlines from descriptions to avoid installation failure

14.3 1.5.0

- Add new API `Goslate.lookup_dictionary()` to get detail information for a single word/phrase, thanks for Adam's suggestion
- Improve document with more user scenario and performance consideration

14.4 1.4.0

- [fix bug] update to adapt latest google translation service changes

14.5 1.3.2

- [fix bug] fix compatible issue with latest google translation service json format changes
- [fix bug] unit test failure

14.6 1.3.0

- [new feature] Translation in roman writing system (romanization), thanks for Javier del Alamo's contribution.
- [new feature] Customizable service URL. you could provide multiple google translation service URLs for better concurrency performance
- [new option] roman writing translation option for CLI
- [fix bug] Google translation may change normal space to no-break space
- [fix bug] Google web API changed for getting supported language list

CHAPTER

FIFTEEN

REFERENCE

Goslate: Free Google Translate API

exception goslate.**Error**

Error type

class goslate.**Goslate**(*writing=('trans',)*, *opener=None*, *retry_times=4*, *executor=<concurrent.futures.thread.ThreadPoolExecutor object>*, *timeout=4*, *service_urls=('http://translate.google.com',)*, *debug=False*, *retry_wait_duration=0.0001*)

All goslate API lives in this class

You have to first create an instance of Goslate to use this API

Parameters

- **writing** – The translation writing system. Currently 3 values are valid
  - *WRITING_NATIVE* for native writing system
  - *WRITING_ROMAN* for roman writing system
  - *WRITING_NATIVE_AND_ROMAN* for both native and roman writing system. output will be a tuple in this case
- **opener** (urllib2.OpenerDirector) – The url opener to be used for HTTP/HTTPS query. If not provide, a default opener will be used. For proxy support you should provide an `opener` with `ProxyHandler`
- **retry_times** (*int*) – how many times to retry when connection reset error occured. Default to 4
- **retry_wait_duration** (*float*) – how many seconds to wait before retry when connection reset error occured. Default to 0.0001s
- **timeout** (*int/float*) – HTTP request timeout in seconds
- **debug** (*bool*) – Turn on/off the debug output
- **service_urls** (*single string or a sequence of strings*) – google translate url list. URLs will be used randomly for better concurrent performance. For example `['http://translate.google.com', 'http://translate.google.de']`
- **executor** (`futures.ThreadPoolExecutor`) – the multi thread executor for handling batch input, default to a global `futures.ThreadPoolExecutor` instance with 120 max thead workers if `futures` is avalible. Set to None to disable multi thread support

Note: multi thread worker relys on futures, if it is not avalible, `goslate` will work under single thread mode

Example

```
>>> import goslate
>>>
>>> # Create a Goslate instance first
>>> gs = goslate.Goslate()
>>>
>>> # You could get all supported language list through get_languages
>>> languages = gs.get_languages()
>>> print(languages['en'])
English
>>>
>>> # Tranlate English into German
>>> print(gs.translate('Hello', 'de'))
Hallo
>>> # Detect the language of the text
>>> print(gs.detect('some English words'))
en
>>> # Get goslate object dedicated for romanlized translation␣
↪(romanlization)
>>> gs_roman = goslate.Goslate(WRITING_ROMAN)
>>> print(gs_roman.translate('hello', 'zh'))
Nín hǎo
```

`detect`(*text*)

Detect language of the input text

Note:

- Input all source strings at once. Goslate will detect concurrently for maximize speed.
- futures is required for best performance.
- It returns generator on batch input in order to better fit pipeline architecture.

Parameters

text (*UTF-8 str; unicode; sequence of string*) – The source text(s) whose language you want to identify. Batch detection is supported via sequence input

Returns

the language code(s)

- unicode: on single string input
- generator of unicode: on batch input of string sequence

Raises

Error if parameter type or value is not valid

Example:

```
>>> gs = Goslate()
>>> print(gs.detect('hello world'))
en
```

(continues on next page)

(continued from previous page)

```
>>> for i in gs.detect([u'apple', 'apfel']):
...     print(i)
...
en
de
```

get_languages()

Discover supported languages

It returns iso639-1 language codes for supported languages for translation. Some language codes also include a country code, like zh-CN or zh-TW.

Note: It only queries Google once for the first time and use cached result afterwards

---

Returns

a dict of all supported language code and language name mapping `{'language-code', 'Language name'}`

Example

```
>>> languages = Goslate().get_languages()
>>> assert 'zh' in languages
>>> print(languages['zh'])
Chinese
```

lookup_dictionary(*text*, *target_language*, *source_language='auto'*, *examples=False*, *etymology=False*, *pronunciation=False*, *related_words=False*, *synonyms=False*, *antonyms=False*, *output_language=None*)

Lookup detail meaning for single word/phrase

Note:

- Do not input sequence of texts

Parameters

- **text** (*UTF-8 str*) – The source word/phrase(s) you want to lookup.
- **target_language** (*str; unicode*) – The language to translate the source text into. The value should be one of the language codes listed in *get_languages()*
- **source_language** (*str; unicode*) – The language of the source text. The value should be one of the language codes listed in *get_languages()*. If a language is not specified, the system will attempt to identify the source language automatically.
- **examples** – include example sentences or not
- **pronunciation** – include pronunciation in roman writing or not
- **related_words** – include related words or not
- **output_language** – the dictionary's own language, default to English.

Returns

a complex list structure contains multiple translation meanings for this word/phrase and detail explaination.

`translate`(*text*, *target_language*, *source_language='auto'*)

Translate text from source language to target language

Note:

- Input all source strings at once. Goslate will batch and fetch concurrently for maximize speed.
- futures is required for best performance.
- It returns generator on batch input in order to better fit pipeline architecture

---

Parameters

- **text** (*UTF-8 str; unicode; string sequence (list, tuple, iterator, generator)*) – The source text(s) to be translated. Batch translation is supported via sequence input
- **target_language** (*str; unicode*) – The language to translate the source text into. The value should be one of the language codes listed in *get_languages()*
- **source_language** (*str; unicode*) – The language of the source text. The value should be one of the language codes listed in *get_languages()*. If a language is not specified, the system will attempt to identify the source language automatically.

Returns

the translated text(s)

- unicode: on single string input
- generator of unicode: on batch input of string sequence
- tuple: if WRITING_NATIVE_AND_ROMAN is specified, it will return tuple/generator for tuple (u"native", u"roman format")

Raises

- *Error* ('invalid target language') if target language is not set
- *Error* ('input too large') if input a single large word without any punctuation or space in between

Example

```
>>> gs = Goslate()
>>> print(gs.translate('Hello World', 'de'))
Hallo Welt
>>>
>>> for i in gs.translate(['good', u'morning'], 'de'):
...     print(i)
...
gut
Morgen
```

To output romanlized translation

Example

```
>>> gs_roman = Goslate(WRITING_ROMAN)
>>> print(gs_roman.translate('Hello', 'zh'))
Nín hǎo
```

goslate.**WRITING_NATIVE = ('trans',)**

native target language writing system

goslate.**WRITING_NATIVE_AND_ROMAN = ('trans', 'translit')**

both native and roman writing. The output will be a tuple

goslate.**WRITING_ROMAN = ('translit',)**

romanlized writing system. only valid for some langauges, otherwise it outputs empty string

CHAPTER

SIXTEEN

DONATE

Donate now to help Goslate up-to-date and get better!

- PayPal: zhuo.qiang@gmail.com
- Alipay:

PYTHON MODULE INDEX

INDEX