第 5 章

调用百度 API 获取数据——小小翻译器

视频讲解

5.1 小小翻译器功能介绍

许多网站提供了 API，通过它可获取网站提供的信息，如天气预报、股票交易信息及火车售票信息等。小小翻译器使用百度翻译开放平台提供的 API，能够实现简单的翻译功能，用户输入自己需要翻译的单词或者句子，即可得到翻译的结果，运行界面如图 5-1 所示。该翻译器不仅能够将英文翻译成中文，也可以将中文翻译成英文或者其他语言。

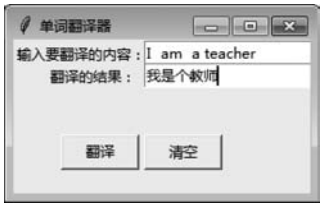

图 5-1 小小翻译器运行界面

5.2 程序设计的思路

百度翻译开放平台提供了 API，可以提供高质量的翻译服务。通过调用百度翻译 API 可以编写在线翻译程序。

百度翻译开放平台每月提供 200 万字符的免费翻译服务，只要拥有百度账号并申请成为开发者就可以获得所需要的账号和密码。下面是开发者的申请链接：

```
http://api.fanyi.baidu.com/api/trans/product/index
```

为方便使用，百度翻译开放平台提供了详细的接入文档，链接如下：

```
http://api.fanyi.baidu.com/api/trans/product/apidoc
```

在文档中列出了详细的使用方法。

按照百度翻译开放平台文档中的要求，生成 URL 请求网页，提交后可返回 JSON 数据格式的翻译结果，再将得到的 JSON 格式的翻译结果解析出来。

5.3 关键技术

5.3.1 urllib 库的高级使用

前面章节使用 urllib.request.urlopen()打开和读取 URL 信息，返回的对象 response 如同一个文本对象，可以调用 read()进行读取，再通过 print()将读到的信息打印出来。下面学习 urllib 库的其他一些功能。

1. 获取服务器响应信息

与浏览器的交互过程一样，request.urlopen()代表请求过程，它返回的 HTTPResponse 对象代表响应。返回内容作为一个对象更便于操作，HTTPResponse 对象 status 属性返回请求 HTTP 后的状态，在处理数据之前要先判断状态情况。如果请求未被响应，需要终止内容处理。reason 属性非常重要，可以得到未被响应的原因，url 属性是返回页面 URL。HTTPResponse.read()是获取请求的页面内容的二进制形式。

也可以使用 getheaders()返回 HTTP 响应的头信息，例如：

```
from urllib import request
f = request.urlopen('http://fanyi.baidu.com')
data = f.read()
print('Status:', f.status, f.reason)
for k, v in f.getheaders():
    print('%s: %s' % (k, v))
```

可以看到 HTTP 响应的头信息。

```
Status: 200 OK
Content-Type: text/html
Date: Sat, 15 Jul 2017 02:18:26 GMT
P3p: CP=" OTI DSP COR IVA OUR IND COM "
Server: Apache
Set-Cookie: locale=zh; expires=Fri, 11-May-2018 02:18:26 GMT; path=/; domain=
.baidu.com
Set-Cookie: BAIDUID=2335F4F896262887F5B2BCEAD460F5E9:FG=1; expires=Sun, 15-Jul-18
02:18:26 GMT; max-age=31536000; path=/; domain=.baidu.com; version=1
Vary: Accept-Encoding
Connection: close
Transfer-Encoding: chunked
```

同样也可以使用 response 对象的 geturl()方法、info()方法、getcode()方法获取相关的 URL、响应信息和响应 HTTP 状态码。

```
# - * - coding: UTF - 8 - * -
from urllib import request
if __name__ == "__main__":
    req = request.Request("http://fanyi.baidu.com/")
    response = request.urlopen(req)
    print("geturl 打印信息: %s" % (response.geturl()))
    print('*************************************************')
    print("info 打印信息: %s" % (response.info()))
    print('*************************************************')
    print("getcode 打印信息: %s" % (response.getcode()))
```

可以得到如下运行结果：

```
geturl 打印信息: http://fanyi.baidu.com/
*************************************************
info 打印信息: Content - Type: text/html
Date: Sat, 15 Jul 2017 02:42:32 GMT
P3p: CP = " OTI DSP COR IVA OUR IND COM "
Server: Apache
Set - Cookie: locale = zh; expires = Fri, 11 - May - 2018 02: 42: 32 GMT; path = /; domain =
.baidu.com
Set - Cookie: BAIDUID = 976A41D6B0C3FD6CA816A09BEAC3A89A:FG = 1; expires = Sun, 15 - Jul - 18
02:42:32 GMT; max - age = 31536000; path = /; domain = .baidu.com; version = 1
Vary: Accept - Encoding
Connection: close
Transfer - Encoding: chunked
*************************************************
getcode 打印信息: 200
```

学会了使用简单的语句对网页进行抓取之后，接下来学习如何向服务器发送数据。

2. 向服务器发送数据

可以使用 urlopen()函数中的 data 参数向服务器发送数据。根据 HTTP 规范，GET 用于信息获取，POST 是向服务器提交数据的一种请求。

从客户端向服务器提交数据使用 POST；从服务器获得数据到客户端使用 GET。然而，GET 也可以提交，与 POST 的区别如下。

(1) GET 方式可以通过 URL 提交数据，待提交数据是 URL 的一部分；采用 POST 方式，待提交数据放置在 HTML HEADER 内。

(2) GET 方式提交的数据最多不超过 1024B，POST 没有对提交内容的长度限制。

下面通过具体的 GET 方式和 POST 方式提交例子来说明。

(1) 以 GET 方式提交 email 和 password 信息。

```
LOGIN_URL = "http://www.kiwisns.com/postLogin/"
values = {'email':'xmj@user.com','password':'123456'}
data = urllib.parse.urlencode(values).encode()
geturl = LOGIN_URL + "?" + data     # 直接以 URL 链接提交数据，链接中包含了所有的参数
req = urllib.request.Request(geturl)
response = request.urlopen(req)     # 传入创建好的 Request 对象，用 GET 方式提交
```

（2）以 POST 方式提交 email 和 password 信息。

```
LOGIN_URL = 'http:// www.kiwisns.com/postLogin/'
values = {'email':'xmj@user.com','password':'123456'}
data = urllib.parse.urlencode(values).encode()
req = urllib.request.Request(URL,data)          # 传送的数据就是这个参数 data
response = request.urlopen(req,data)            # 传入创建好的 Request 对象,用 POST 方式提交
```

如果没有设置 urlopen()函数的 data 参数，HTTP 请求采用 GET 方式，也就是从服务器获取信息。如果设置 data 参数，HTTP 请求采用 POST 方式，也就是向服务器传递数据。

data 参数有自己的格式，它是一个基于 application/x-www. form-urlencoded 的格式，具体格式不用了解，可以使用 urllib. parse. urlencode()函数将字符串自动转换成上面所说的格式。

5.3.2 使用 User Agent 隐藏身份

1. 为何要设置 User Agent

有一些网站不喜欢被爬虫程序访问，所以会检测连接对象，如果是爬虫程序，也就是非人点击访问，它就会阻止继续访问。所以，为了让程序正常运行，需要隐藏自己的爬虫程序的身份。此时，就可以通过设置 User Agent 来达到隐藏身份的目的。User Agent 的中文名为用户代理，简称 UA。

User Agent 存放于 Headers 中，服务器就是通过查看 Headers 中的 User Agent 来判断是谁在访问。在 Python 中，如果不设置 User Agent，程序将使用默认的参数，那么这个 User Agent 就会有 Python 的字样，如果服务器检查 User Agent，那么没有设置 User Agent 的 Python 程序将无法正常访问网站。

Python 允许修改这个 User Agent 来模拟浏览器访问，它的强大毋庸置疑。

2. 常见的 User Agent

1）Android

- Mozilla/5.0 (Linux; Android 4.1.1; Nexus 7 Build/JRO03D) AppleWebKit/535.19 (KHTML, like Gecko) Chrome/18.0.1025.166 Safari/535.19
- Mozilla/5.0 (Linux; U; Android 4.0.4; en-gb; GT-I9300 Build/IMM76D) AppleWebKit/534.30 (KHTML, like Gecko) Version/4.0 Mobile Safari/534.30
- Mozilla/5.0 (Linux; U; Android 2.2; en-gb; GT-P1000 Build/FROYO) AppleWebKit/533.1 (KHTML, like Gecko) Version/4.0 Mobile Safari/533.1

2）Firefox

- Mozilla/5.0 (Windows NT 6.2; WOW64; rv:21.0) Gecko/20100101 Firefox/21.0
- Mozilla/5.0 (Android; Mobile; rv:14.0) Gecko/14.0 Firefox/14.0

3）Google Chrome

- Mozilla/5.0 (Windows NT 6.2; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/27.0.1453.94 Safari/537.36
- Mozilla/5.0 (Linux; Android 4.0.4; Galaxy Nexus Build/IMM76B) AppleWebKit/535.19 (KHTML, like Gecko) Chrome/18.0.1025.133 Mobile Safari/535.19

4）iOS

- Mozilla/5.0 (iPad; CPU OS 5_0 like Mac OS X) AppleWebKit/534.46 (KHTML, like Gecko) Version/5.1 Mobile/9A334 Safari/7534.48.3
- Mozilla/5.0 (iPod; U; CPU like Mac OS X; en) AppleWebKit/420.1 (KHTML, like Gecko) Version/3.0 Mobile/3A101a Safari/419.3

上面列举了 Andriod、Firefox、Google Chrome、iOS 的一些 User Agent。

3. 设置 User Agent 的方法

设置 User Agent 有两种方法。

（1）在创建 Request 对象时，填入 headers 参数（包含 User Agent 信息），这个 headers 参数要求为字典。

（2）在创建 Request 对象时不添加 headers 参数，在创建完成之后，使用 add_header() 的方法，添加 headers。

方法一：

使用上面提到的 Android 的第一个 User Agent，在创建 Request 对象时传入 headers 参数，编写代码如下：

```
# - * - coding: UTF - 8 - * -
from urllib import request
if __name__ == "__main__":
    #以 CSDN 为例，CSDN 不更改 User Agent 是无法访问的
    url = 'http://www.csdn.net/'
    head = {}
    #写入 User Agent 信息
    head['User - Agent'] = 'Mozilla/5.0 (Linux; Android 4.1.1; Nexus 7 Build/JRO03D) AppleWebKit/535.19 (KHTML, like Gecko) Chrome/18.0.1025.166 Safari/535.19'
    req = request.Request(url, headers = head)   #创建 Request 对象
    response = request.urlopen(req)              #传入创建好的 Request 对象
    html = response.read().decode('utf - 8')     #读取响应信息并解码
    print(html)                                  #打印信息
```

方法二：

使用上面提到的 Android 的第一个 User Agent，在创建 Request 对象时不传入 headers 参数，创建之后使用 add_header()方法，添加 headers，编写代码如下：

```
# - * - coding: UTF - 8 - * -
from urllib import request
```

```
if __name__ == "__main__":
    # 以 CSDN 为例,CSDN 不更改 User Agent 是无法访问的
    url = 'http://www.csdn.net/'
    req = request.Request(url)                    # 创建 Request 对象
    req.add_header('User-Agent', 'Mozilla/5.0 (Linux; Android 4.1.1; Nexus 7 Build/JRO03D)
AppleWebKit/535.19 (KHTML, like Gecko) Chrome/18.0.1025.166 Safari/535.19') # 传入 headers
    response = request.urlopen(req)               # 传入创建好的 Request 对象
    html = response.read().decode('utf-8')        # 读取响应信息并解码
    print(html)                                   # 打印信息
```

5.3.3 JSON 使用

JSON(JavaScript Object Notation)是一种轻量级的数据交换格式,比 XML 更小、更快、更易解析,易于读写且占用带宽小,网络传输速度快,适用于数据量大且不要求保留原有类型的情况。它是 JavaScript 的子集,易于阅读和编写。

前端和后端进行数据交互,就是通过 JSON 进行的。因为 JSON 易于被识别的特性,常被作为网络请求的返回数据格式。在爬取动态网页时,经常会遇到 JSON 格式的数据,Python 中可以使用 JSON 模块来对 JSON 数据进行解析。

1. JSON 的结构

JSON 常见形式为“名称/值”对的集合。

例如下面这样:

```
{"firstName": "Brett", "lastName": "McLaughlin"}
```

JSON 允许使用数组,采用方括号[]实现。

例如:

```
{
     "people":[
         {"firstName": "Brett",
           "lastName":"McLaughlin"
         },
         {"firstName":"Jason",
           "lastName":"Hunter"
         }
      ]
}
```

在这个示例中,只有一个名为 people 的名称,值是包含两个元素的数组,每个元素是一个人的信息,其中包含名和姓。

XML 和 JSON 都使用结构化方法来标记数据,下面做一个简单的比较。

用 XML 表示中国部分省市数据如下:

```
<?xml version = "1.0" encoding = "utf - 8"?>
< country >
    < name >中国</name >
    < province >
        < name >黑龙江</name >
        < cities >
            < city >哈尔滨</city >
            < city >大庆</city >
        </cities >
    </province >
    < province >
        < name >广东</name >
        < cities >
            < city >广州</city >
            < city >深圳</city >
            < city >珠海</city >
        </cities >
    </province >
    < province >
        < name >新疆</name >
        < cities >
            < city >乌鲁木齐</city >
        </cities >
    </province >
</country >
```

用 JSON 表示如下，其中省份采用的是数组。

```
{
    "name": "中国",
    "province": [{
        "name": "黑龙江",
        "cities": {
            "city": ["哈尔滨", "大庆"]
        }
    }, {
        "name": "广东",
        "cities": {
            "city": ["广州", "深圳", "珠海"]
        }
    }, {
        "name": "新疆",
        "cities": {
            "city": ["乌鲁木齐"]
        }
    }]
}
```

由上例可以看出，JSON 简单的语法格式和清晰的层次结构明显比 XML 容易阅读，并且在数据交换方面，因为 JSON 所使用的字符要比 XML 少得多。所以，可以节约传输数据所占用的带宽。

JSON 中的数据类型和 Python 中的数据类型转化关系如表 5-1 所示。

表 5-1　JSON 中的数据类型和 Python 中的数据类型转化关系

JSON 的数据类型	Python 的数据类型	JSON 的数据类型	Python 的数据类型
object	dict	number (real)	float
array	list	true、false	True、False
string	str	null	None
number (int)	int		

2. JSON 模块中常用的方法

在使用 JSON 这个模块前，首先要导入 JSON 库：import json。

JSON 模块主要提供了 4 个方法：dumps、loads、dump 和 load，如表 5-2 所示。

表 5-2　JSON 模块中常用的方法

方　法	功 能 描 述
json.dumps()	将 Python 对象转换成 JSON 字符串
json.loads()	将 JSON 字符串转换成 Python 对象
json.dump()	将 Python 类型数据序列化为 JSON 对象后写入文件
json.load()	读取文件中 JSON 形式的字符串并转化为 Python 类型数据

下面通过例子说明这个方法的使用。

1) json.dumps()

json.dumps()的作用是将 Python 对象转换成 JSON 字符串。

```
import json
data = {'name':'nanbei','age':18}
s = json.dumps(data)            #将 Python 对象编码成 json 字符串
print(s)
```

运行结果如下：

```
{"name": "nanbei", "age": 18}
```

JSON 注意事项：

- 名称必须用双引号(即"name")来包括；
- 值可以是字符串、数字、true、false、null、JavaScript 数组或子对象。

从运行结果可见，原先的'name''age'单引号已经变成双引号"name" "age"。

2) json.loads()

json.loads()的作用是将 JSON 字符串转换成 Python 对象。

```
import json
data = {'name':'nanbei','age':18}
a = json.dumps(data)
print(json.loads(a))            #将 json 字符串编码成 Python 对象—dict 字典
```

运行结果如下：

```
{'name': 'nanbei', 'age': 18}
```

对 JSON 文件要先读文件，然后才能转换成 Python 对象。

```
f = open('stus.json',encoding = 'utf - 8')          # 'stus.json'是一个 JSON 文件
content = f.read()                                  # 使用 loads()方法，需要先读文件成字符串
user_dic = json.loads(content)                      # 转换成 Python 的字典对象
print(user_dic)
```

3）json.load()

json.load()的作用是读取文件中 JSON 形式的字符串并转化为 Python 类型数据。

```
import json
f = open('stus.json',encoding = 'utf - 8')
user_dic = json.load(f)          # f 是文件对象
print(user_dic)
```

可见 loads()传入的是字符串，而 load()传入的是文件对象。使用 loads()时需要先读文件成字符串再使用，而 load()则不用先读文件成字符串，而是直接传入文件对象。

4）json.dump()

json.dump()的作用是将 Python 类型数据序列化为 JSON 对象后写入文件。

```
stus = { 'xiaojun':88, 'xiaohei':90, 'lrx':100}
f = open('stus2.json','w',encoding = 'utf - 8')          # 以写方式打开 stus2.json 文件
json.dump(stus,f)                                        # 写入 stus2.json 文件
f.close()                                                # 文件关闭
```

5.4 程序设计的步骤

5.4.1 设计界面

采用 Tkinter 的 place 几何布局管理器设计 GUI 图形界面，运行效果如图 5-2 所示。

图 5-2 place 几何布局管理器

新建文件 translate_test.py，编写如下代码：

```
from tkinter import *
if __name__ == "__main__":
    root = Tk()
    root.title("单词翻译器")
    root['width'] = 250;root['height'] = 130
    Label(root,text = '输入要翻译的内容：',width = 15).place(x = 1,y = 1)   #绝对坐标(1,1)
    Entry1 = Entry(root,width = 20)
    Entry1.place(x = 110,y = 1)                                              #绝对坐标(110,1)
    Label(root,text = '翻译的结果：',width = 18).place(x = 1,y = 20)  #绝对坐标(1,20)
    s = StringVar()                                                          #一个 StringVar()对象
    s.set("大家好,这是测试")
    Entry2 = Entry(root,width = 20,textvariable = s)
    Entry2.place(x = 110,y = 20)                                             #绝对坐标(110,20)
    Button1 = Button(root,text = '翻译',width = 8)
    Button1.place(x = 40,y = 80)                                             #绝对坐标(40,80)
    Button2 = Button(root,text = '清空',width = 8)
    Button2.place(x = 110,y = 80)                                            #绝对坐标(110,80)
    #给 Button 绑定鼠标监听事件
    Button1.bind("<Button-1>",leftClick)                                     #翻译按钮
    Button2.bind("<Button-1>",leftClick2)                                    #清空按钮
    root.mainloop()
```

5.4.2 使用百度翻译开放平台 API

使用百度翻译需要向 http://api.fanyi.baidu.com/api/trans/vip/translate 通过 POST 或 GET 方法发送表 5-3 中的请求参数来访问服务。

表 5-3 请求参数

参数名	类型	必填参数	描述	备注
q	TEXT	Y	请求翻译 query	UTF-8 编码
from	TEXT	Y	翻译源语言	语言列表(可设置为 auto)
to	TEXT	Y	译文语言	语言列表(不可设置为 auto)
appid	INT	Y	APP ID	可在管理控制台查看
salt	INT	Y	随机数	
sign	TEXT	Y	签名	appid+q+salt+密钥的 MD5 值

sign 签名是为了保证调用安全，使用 MD5 算法生成的一段字符串，生成的签名长度为 32 位，签名中的英文字符均为小写格式。为保证翻译质量，请将单次请求长度控制在 6000B 以内(汉字约为 2000 个)。

签名生成方法如下。

(1) 将请求参数中的 appid、query(q，注意为 UTF-8 编码)、随机数 salt 以及平台分配的密钥(可在管理控制台查看)按照 appid+q+salt+密钥的顺序拼接得到字符串 1。

(2) 对字符串 1 做 md5，得到 32 位小写的 sign。

注意

- 先将需要翻译的文本转换为 UTF-8 编码。
- 在发送 HTTP 请求之前需要对各字段做 URL encode。
- 在生成签名拼接 appid＋q＋salt＋密钥字符串时，q 不需要做 URL encode，在生成签名之后，发送 HTTP 请求之前才需要对要发送的待翻译文本字段 q 做 URL encode。

例如，将 apple 从英文翻译成中文，请求参数如下：

```
q = apple
from = en
to = zh
appid = 2015063000000001
salt = 1435660288
平台分配的密钥: 12345678
```

生成签名参数 sign。

(1) 拼接字符串 1。

```
拼接 appid = 2015063000000001 + q = apple + salt = 1435660288 + 密钥 = 12345678
得到字符串 1 = 2015063000000001apple143566028812345678
```

(2) 计算签名 sign(对字符串 1 做 md5 加密，注意计算 md5 之前，串 1 必须为 UTF-8 编码)。

```
sign = md5(2015063000000001apple143566028812345678)
sign = f89f9594663708c1605f3d736d01d2d4
```

通过 Python 提供的 hashlib 模块中的 hashlib. md5()可以实现签名计算。例如：

```
import hashlib
m = '2015063000000001apple143566028812345678'
m_MD5 = hashlib.md5(m)
sign = m_MD5.hexdigest()
print( 'm = ',m)
print ('sign = ',sign)
```

得到签名之后，按照百度文档中的要求生成 URL 请求，提交后可返回翻译结果。

完整请求如下：

```
http://api. fanyi. baidu. com/api/trans/vip/translate? q = apple&from = en&to = zh&appid =
2015063000000001&salt = 1435660288&sign = f89f9594663708c1605f3d736d01d2d4
```

也可以使用 POST 方法传送需要的参数。

本案例采用 urllib. request. urlopen()函数中的 data 参数向服务器发送数据。

下面是发送 data 实例，向“百度翻译”发送要翻译数据，得到翻译结果。

```
# - * - coding: UTF-8 - * -
from tkinter import *
from urllib import request
from urllib import parse
import json
import hashlib
def translate_Word(en_str):
    #simulation browse load host url,get cookie
    URL = 'http://api.fanyi.baidu.com/api/trans/vip/translate'
    #en_str = input("请输入要翻译的内容:")
    #创建 Form_Data 字典,存储向服务器发送的 Data
    #Form_Data = {'from':'en','to':'zh','q':en_str,''appid'':'2015063000000001', 'salt':
'1435660288'}
    Form_Data = {}
    Form_Data['from'] = 'en'
    Form_Data['to'] = 'zh'
    Form_Data['q'] = en_str                              #要翻译数据
    Form_Data['appid'] = '2015063000000001'              #申请的 APP ID
    Form_Data['salt'] = '1435660288'
    Key = "12345678"                                     #平台分配的密钥
    m = Form_Data['appid'] + en_str + Form_Data['salt'] + Key
    m_MD5 = hashlib.md5(m.encode('utf8'))
    Form_Data['sign'] = m_MD5.hexdigest()

    data = parse.urlencode(Form_Data).encode('utf-8') #使用 urlencode 方法转换标准格式
    response = request.urlopen(URL,data)          #传递 Request 对象和转换完格式的数据
    html = response.read().decode('utf-8')               #读取信息并解码
    translate_results = json.loads(html)                 #使用 JSON
    print(translate_results)                             #打印出 JSON 数据
    translate_results = translate_results['trans_result'][0]['dst']     #找到翻译结果
    print("翻译的结果是: %s" % translate_results)     #打印翻译信息
    return translate_results
def leftClick(event):                                    #翻译按钮事件函数
   en_str = Entry1.get()                                 #获取要翻译的内容
   print(en_str)
   vText = translate_Word(en_str)
   Entry2.config(Entry2,text = vText)                    #修改翻译结果框文字
   s.set("")
   Entry2.insert(0,vText)
def leftClick2(event):                                   #清空按钮事件函数
   s.set("")
   Entry2.insert(0,"")
```

这样就可以查看翻译的结果了,如下所示:

输入要翻译的内容:I am a teacher

翻译的结果是:我是个教师。

得到的 JSON 数据如下:

```
{'from': 'en', 'to': 'zh', 'trans_result': [{'dst': '我是个教师.', 'src': 'I am a teacher'}]}
```

返回结果是 JSON 格式的，包含表 5-4 中的字段。

表 5-4 翻译结果的 JSON 字段

字 段 名	类 型	描 述
from	TEXT	翻译源语言
to	TEXT	译文语言
trans_result	MIXED LIST	翻译结果
src	TEXT	原文
dst	TEXT	译文

其中，trans_result 包含 src 和 dst 字段。

JSON 是一种轻量级的数据交换格式，其中保存着想要的翻译结果，我们需要从爬取到的内容中找到 JSON 格式的数据，再将得到的 JSON 格式的翻译结果解析出来。

这里向服务器发送数据 Form_Data 也可以直接按如下方式写：

```
Form_Data = {'from':'en', 'to':'zh', 'q':en_str,''appid'':'2015063000000001', 'salt': '1435660288'}
```

现在只做了将英文翻译成中文，稍加改动就可以将中文翻译英文了：

```
Form _Data = { 'from':'zh', 'to':'en', 'q':en_str,''appid'':'2015063000000001', 'salt':
'1435660288'}
```

这一行中的 from 和 to 的取值，应该可以用于其他语言之间的翻译。如果源语言语种不确定可设置为 auto，目标语言语种不可设置为 auto。百度翻译支持的语言简写如表 5-5 所示。

表 5-5 百度翻译支持的语言简写

语言简写	名 称	语言简写	名 称
auto	自动检测	bul	保加利亚语
zh	中文	est	爱沙尼亚语
en	英语	dan	丹麦语
yue	粤语	fin	芬兰语
wyw	文言文	cs	捷克语
jp	日语	rom	罗马尼亚语
kor	韩语	slo	斯洛文尼亚语
fra	法语	swe	瑞典语
spa	西班牙语	hu	匈牙利语
th	泰语	cht	繁体中文
ara	阿拉伯语	vie	越南语
ru	俄语	el	希腊语
pt	葡萄牙语	nl	荷兰语
de	德语	pl	波兰语

读者可查阅资料编程向“有道翻译”http://fanyi.youdao.com/translate? smartresult=dict 发送要翻译的数据 data，得到翻译结果。

5.5 API 调用拓展——爬取天气预报信息

目前绝大多数网站以动态网页的形式发布信息。所谓动态网页，就是用相同的格式呈现不同的内容。例如，每天访问中国天气网，看到的信息呈现格式是不变的，但天气信息数据是变化的。如果网站没有提供 API 调用的功能，则可以先获取网页数据，然后将网页数据转换为字符串后利用正则表达式提取所需的内容，即所谓的爬虫方式。利用爬虫经常获取的是网页中动态变化的数据。因此，爬虫程序是自动获取网页中动态变化数据的工具。

中国天气网(http://www.weather.com.cn)向用户提供国内各城市的天气信息，并提供 API 供程序获取所需的天气数据，返回数据格式为 JSON。

中国天气网提供 API 网址类似 http://www.weather.com.cn/data/cityinfo/101180101.html，其中，101180101 为郑州的城市编码。各城市编码可通过网络搜索取得。

例如：荥阳 101180103 新郑 101180106 新密 101180105 登封 101180104 中牟 101180107 巩义 101180102 上街 101180108 郑州 101180101 卢氏 101181704 灵宝 101181702 三门峡 101181701 义马 101181705 渑池 101181703 陕县 101181706 南阳 101180701 新野 101180709 邓州 101180711 南召 101180702 方城 101180703

下面的代码为调用 API 在中国天气网获取郑州市当天天气预报数据的实例。

```
import urllib.request                    #引入 urllib 包中的模块 request
import json                              #引入 json 模块
code = '101180101'                       #郑州市城市编码
#用字符串变量 url 保存合成的网址
url = 'http://www.weather.com.cn/data/cityinfo/%s.html' % code
print('url = ',url)
obj = urllib.request.urlopen(url)  #调用函数 urlopen()打开给定的网址，结果返回到对象 obj 中
print('type(obj) = ',type(obj))    #输出 obj 的类型
data_b = obj.read()                      #read()从对象 obj 中读取内容，内容为 bytes 字节流数据
print('字节流数据 = ',data_b)
data_s = data_b.decode('utf-8')    #bytes 字节流数据转换为字符串类型
print('字符串数据 = ',data_s)
#调用 JSON 的函数 loads()将 data_s 中保存的字符串数据转换为字典型数据
data_dict = json.loads(data_s)
print('data_dict = ',data_dict)    #输出字典 data_dict 的内容
rt  = data_dict['weatherinfo']     #取得键为"weatherinfo"的内容
print('rt = ',rt)                        #rt 仍然为字典型变量
#获取城市名称、天气状况、最高温和最低温
my_rt = ('%s,%s,%s～%s') % (rt['city'],rt['weather'],rt['temp1'],rt['temp2'])
print(my_rt)
```

代码中，用字符串变量 url 保存合成的网址，该网址为给定编码城市的当天天气预报。调用函数 urlopen()打开给定的网址，结果返回到对象 obj 中。调用函数 read()从对象 obj 中读取天气预报内容，最后调用 JSON 的函数 loads()将天气预报内容转换为字典型数据，保存到字典型变量 data_dict 中。从字典型变量 data_dict 中取得键为“weatherinfo”的内

容，保存到变量 rt 中，rt 仍然为字典型变量。

城市名称、天气状况、最高温和最低温均是从字典型变量 rt 中取得的，键分别为“city”“weather”“temp1”“temp2”。

运行结果如下：

```
url = http://www.weather.com.cn/data/cityinfo/101180101.html
type(obj) = <class 'http.client.HTTPResponse'>
字节流数据 = b'{"weatherinfo":{"city":"\xe9\x83\x91\xe5\xb7\x9e","cityid":"101180101",
"temp1":"17\xe2\x84\x83","temp2":"27\xe2\x84\x83","weather":"\xe9\x98\xb4\xe8\xbd\x99\
xb4","img1":"n2.gif","img2":"d0.gif","ptime":"18:00"}}'
字符串数据 = {"weatherinfo":{"city":"郑州","cityid":"101180101","temp1":"17℃","temp2":
"27℃","weather":"阴转晴","img1":"n2.gif","img2":"d0.gif","ptime":"18:00"}}
data_dict = {'weatherinfo': {'ptime': '18:00', 'img2': 'd0.gif', 'weather': '阴转晴', 'img1':
'n2.gif', 'temp1': '17℃', 'cityid': '101180101', 'temp2': '27℃', 'city': '郑州'}}
rt = {'ptime': '18:00', 'img2': 'd0.gif', 'weather': '阴转晴', 'img1': 'n2.gif', 'temp1': '17℃',
'cityid': '101180101', 'temp2': '27℃', 'city': '郑州'}
郑州,阴转晴,17℃ ～27℃
```

从结果可知，函数 urlopen()的返回值为来自服务器的回应对象，调用其 read()函数可得 bytes 字节流类型的数据，将 bytes 字节流类型的数据转换为字符串类型，即为 JSON 数据。调用 JSON 函数 loads()可将 JSON 数据转换为字典型数据，而中国天气网返回的数据为嵌套 1 层的字典型数据。因此，首先通过 rt＝data_dict['weatherinfo']取得城市天气预报信息，再通过 rt['city']、rt['weather']、rt['temp1']和 rt['temp2']取得具体的数据。