淡蓝小点（直播or录屏待定）系列：OpenAI编程基础（Draft）

淡蓝小点Bluedotdot

微信：bluedotdot_cn

2024 年 4 月 16 日

目录

人人人人

设定API Key—方法之一

调用OpenAI的API需要首先正确设定API Key。指定API Key的方法有很多种。最常见的方法是在操作系统中创建一个名为"OPENAI_API_KEY"的环境变量，变量的值即为API Key。利用这种方法在代码中不用显示的加载API Key。

```
1  import os
2
3  api_key = os.environ.get("OPENAI_API_KEY")
4  print("My OpenAI API Key:", api_key)
```

这种方法根据所在环境（Windows、Linux、MacOS）不同，添加方法略有不同。这种方法虽然可以在代码中避免显示加载API Key，但它会暴露给所有使用计算机的用户。

设定API Key—方法之二

可以将Key只与单个项目绑定，这样可以避免Key暴露给所有项目或者所有使用该计算机的用户。在项目目录中创建一个".env"文件，在文件中添加如下内容

```
OPENAI_API_KEY=Your-API-Key-here
```

接着只需要从dotenv中导入load_dotenv，并利用load_dotenv()加载".env"中的环境变量即可

```
import os
from dotenv import load_dotenv

load_dotenv()
#The API Key can be found now
api_key = os.environ.get("OPENAI_API_KEY")
print("My OpenAI API Key:", api_key)
```

使用这种方法时需要提前安装python-dotenv包（可利用pip安装）。另外要特别注意，在使用版本管理工具时千万不要将".env"文件上传。

设定API Key—方法之三

最简单直接的方法是在代码中显示设定API Key，这种方法并不提倡，因为它的安全性很差。

```python
from openai import OpenAI

client = OpenAI(api_key="Your-API-Key-here")
```

网上也有资料说可以通过设定openai.api_key_path="xxx.txt"，将API Key存放在xxx.txt中，实现API Key的自动加载，但实际上这种做法是不可行的。不过将API Key存放在类似json的配置文件中，在程序开始时利用json.load读取出来是一种很好的方式。

文本生成-两类API端点

OpenAI中将文本类的回复称为“消息补全”(completions)，由于历史原因，有两类不同的API端点（ENDPOINT），它们分别适用于不同时期的模型

	MODEL FAMILIES	API ENDPOINT
2023年以后的模型	gpt-4, gpt-4-turbo-preview, gpt-3.5-turbo	chat.completions
2023年以前的模型	gpt-3.5-turbo-instruct, babbage-002, davinci-002	completions

文本生成-chat.completions

OpenAI用create函数发起一次对话，一个最基本的文本对话例子是

```python
from openai import OpenAI
client = OpenAI()

response = client.chat.completions.create(
  model="gpt-3.5-turbo",
  messages=[
    {"role": "system", "content": "You are a helpful assistant."},
    {"role": "user", "content": "Who won the world series in 2020?"},
    {"role": "assistant", "content": "The Los Angeles Dodgers won the World
        Series in 2020."},
    {"role": "user", "content": "Where was it played?"}
  ]
)
```

文本生成-chat.completions

create函数一共有十多个参数，其有只有两个是必须的：messages和model。model指定所要调用模型的名称（所有可用模型名称可点击查阅）；共有五种不同的message：system、user、assistant、tool、function。这里主要介绍前三种。

- system：指定模型的能力域（类似于在prompt最前面告诉模型“你是一个资深的心理医生”）
- user：发起会话、提出问题的用户
- assistant：系统的回答，当问题需要上下文时模型的回答用这一角色承载

create函数中的messages由多个message构成，每个message的类型由"role"指定，"content"则是相应的内容，这两个字段是必须的。消息中还有一个"name"字段，主要帮用户做消息管理。

文本生成-chat.completions

另外一些较常用的可选参数

- n：指定返回答案的数量。
- max_tokens：指定系统答复的最大长度，此长度受模型上下文总长度影响。
- seed：类似于随机数种子，指定相同的种子可以获得相同的回答
- temperature：取值介于0～ 2，默认值为1。取值越大回答更加确定、更加可预测，多用于对准确度、遵从性要求较高的情况；取值越小答案更加多样化、更加独特有趣、更加富有创新性。
- response_format：若设定为"{type:json_object}"消息会以json格式返回（仅适用gpt-4-turbo-preview和gpt-3.5-turbo-0125）
- frequency和presence：取值皆介于-2～2之间。对frequency，当取值靠近2时会尝试减少重复单词或短语的出现，避免冗余性。对presence，当取值靠近2时会尝试引入新的概念或词语，增强内容的丰富性。（这些参数有可能增加返回等待时长）

文本生成-chat.completions

```python
import openai

client = openai.OpenAI()
completion = client.chat.completions.create(
  model="gpt-4",
  messages=[
    {"role": "system", "content": "You are a very good poet"},
    {"role": "user", "content": "Write a short poem about Spring, not more
        than 100 words"}
  ],
  frequency_penalty= -1.8,
)
print(completion.choices[0].message.content)
```

由于指定了frequency_penalty值偏小，因此答案中会出现较多的重复token。一个可能的输出是

```
In the heart of the bloom, the springtime unfolds,
Awakening the buds, the stories of the old.
A gentle, rosy dawn, the the the the the.....
```

文本生成-chat.completions

由create函数返回的是chat completion对象，其中比较重要的字段包括

- choices：系统回复的答案，答案数与create函数中的n相对应，可通过角标提取第i个回答的内容（choice[i].message.content）
- system_fingerprint：字符串类型，代表模型运行时后端配置的唯一标识。只有当seed与system_fingerprint同时保持一致时，相同的问题才能得到相同的答案。
- usage：会话的token消耗情况，包括prompt的token数和回答的token数。

文本生成-completions

completions编程端点已稍显陈旧，不建议继续采用这类方式编码

```python
from openai import OpenAI
client = OpenAI()

response = client.completions.create(
  model="gpt-3.5-turbo-instruct",
  prompt="Write a tagline for an ice cream shop."
)
print(response.choices[0].text)
```

completions最大的特点是直接用"prompt"项传递查询内容，它返回的completion对象直接用choices[i].text获取模型回答。

文本生成-管理好tokens

管理好使用的token数量非常非常非常关键！token是一种介于character和word之间的编码方式（一般采用BPE，Byte Pair Encoding），例如"ChatGPT is great!"，它的token可能是["Chat","G","PT"," is", " great","!"]。将string分解成token的方法一般是从语料库中学习得到的，因而不同的学习策略得到不同的编码方法。OpenAI不同模型采用了不同的编码方法。

Encoding name	OpenAI models
cl100k_base	gpt-4, gpt-3.5-turbo, text-embedding-ada-002, text-embedding-3-small, text-embedding-3-large
p50k_base	Codex models, text-davinci-002, text-davinci-003
r50k_base(or gpt2)	GPT-3 models like davinci

可以通过导入OpenAI的tiktoken包实现对string的编码（conda只能在conda-forge中找到）

```
pip install tiktoken
```

文本生成-管理好tokens

可以通过两种不同的方式获取想要的encoding方式

```
import tiktoken

#it will take several minutes to download cl100k_base
encoding = tiktoken.get_encoding("cl100k_base")

#or we can get the encoding accroding to model name
encoding = tiktoken.encoding_for_model("gpt-4")
```

可以对string进行编码、解码

```
tokens = encoding.encode("tiktoken is great!")
#print(tokens): [83, 1609, 5963, 374, 2294, 0]
de_str = encoding.decode(tokens)
#print(de_str): tiktoken is great!
```

tiktoken is great!

文本生成-管理好tokens

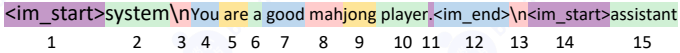
```python
import tiktoken

def tokens_from_messages(messages, model="gpt-3.5-turbo-0613"):
    encoding = tiktoken.get_encoding("cl100k_base")
    num_tokens = 0
    for message in messages:
        num_tokens += 4  # <im_start>{role/name}\n{content}<im_end>\n
        for key, value in message.items():
            num_tokens += len(encoding.encode(value))
            if key == "name":  # once there is name, role can omit
                num_tokens += -1
    num_tokens += 2  # reply begins with <im_start>assistant
    return num_tokens

messages = [{"role":"system","content":"You are a good mahjong player."},]
model = "gpt-3.5-turbo-0613"
print(f"{tokens_from_messages(messages, model)} prompt tokens counted.")
```

文本生成-管理好tokens

这个例子具体解析后为



或者利用completion对象中的usage对象获取消的token数（包括查询和回答）

```python
from openai import OpenAI
client = OpenAI()

response = client.chat.completions.create(
  model="gpt-3.5-turbo",
  messages=[{"role":"system","content":"You are a helpful assistant."},
          {"role":"user","content":"Who won the world series in 2020?"},])
print("completion_tokens:{}, prompt_tokens:{}, total_tokens:{}".
    format(response.usage.completion_tokens, response.usage.prompt_tokens,
        response.usage.total_tokens))
```

在真正的使用过程中，首先利用tokenizer将string转换成tokens，然后利用Emedding算法将token转换成维度相同的向量才能进行计算。但Embedding的应用范围不仅于此，几乎所有对文本处理的地方都需要它，如检索、聚类、推荐等。因此OpenAI也开放了Embedding API，但需按token数收费。3代模型只采用2021年9月之前的数据做训练。

Model	Description	Dimension
text-embedding-3-large	目前能力最强的模型，能适用于英文和非英文	3072
text-embedding-3-small	比第二代模型能力要强，但维度被压缩了	1536
text-embedding-ada-002	第二代模型	1536

Embedding

利用embedding的API端点编程，类似于文本生成的API端点。不过注意，这里的embedding模型是将一个string整体转换成一个向量。

```python
from openai import OpenAI
client = OpenAI()

response = client.embeddings.create(
    input="Your text string goes here",
    model="text-embedding-3-small"
)

print(len(response.data[0].embedding))
print(response.data[0].embedding)
```

embedding的请求中一共有五个参数，其中model和input是必须指定的，embedding指定模型名称，input是字符串或字符串数组。剩下三个可选参数分别是：encoding_format、dimensions、user。

encoding_format用于指定返回结果的表示形式，如float或base64。user用于指定请求者，它一般用于OpenAI对请求的管理。dimensions是指可以人为限定embedding输出的维度，这在很多时候可以节省存储空间和计算时间（目前仅对3代模型有用，需要将openai包升级到最新）。

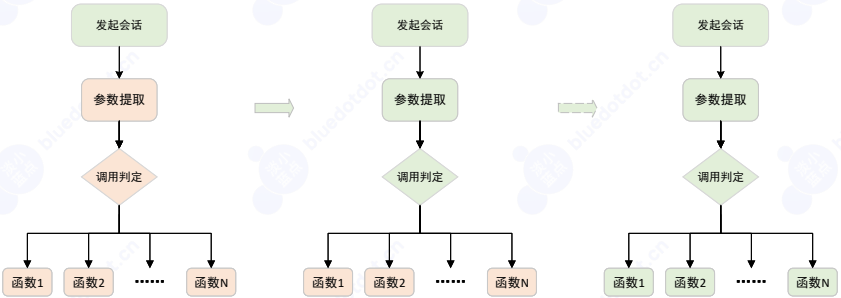
```python
from openai import OpenAI
client = OpenAI()

response = client.embeddings.create(
    input="Your text string goes here",
    model="text-embedding-3-small",
    dimensions=128,
)

print(len(response.data[0].embedding))
print(response.data[0].embedding)
```

函数调用

函数调用（Function calling）是OpenAI一个非常强大的编程接口，它允许用户设计自己的功能函数并最大化利用GPT模型的能力做函数调用，这对于基于GPT的应用开发有巨大意义。



函数调用

要正确使用OpenAI的函数调用功能有以下几个关键点：

- 定义函数
- 配置函数：{"type":"function", "function":{"name": xx_func, "description":xxxx, "parameters":{...}}}
- 发起会话

```
response = client.chat.completions.create(
  model=...,
  messages=[...],
  tools = tools,
  tool_choice="auto")
```

- 获取函数调用信息（函数名和函数参数）

```
for func in response.choices[0].message.tool_calls:
  func_name = func.function.name
  args = func.function.arguments
```

函数调用

```python
import os
from openai import OpenAI

client = OpenAI(api_key=os.environ['OPENAI_API_KEY'])

sys_content='''You are an accurate context format reader, please extract
    the key information from the context as format below:
            "City": the name of the city,
            "Abbr.": abbreviation of the city,
            "Nickname": nickname of the city,
            "Area": total area of the city,
            "Population": number of population living in the city,'''

usr_content='''
            "Beijing, where is short for Jing..."
            "Shanghai, also named Hu, or called as Shen, ..."
            "Shenzhen, short for Shen, or named as Peng,..."'''
```

函数调用

```
tools = [
  { "type": "function",
    "function": {
      "name": "context_format_extract",
      "description": "Get the current weather in a given location",
      "parameters":{
        "type": "object",
        "properties":{
          "City":{
            "type": "string",
            "description": "The name of the city"
          },
          ......
        }
      }
    }
  }
]
```

函数调用

```python
def context_format_extract(name, abbr, nickname, area, population):
  '''In context format extract'''
  print("In context_format_extract")
  print("City:{}\n"
        "Abbr:{}\n"
        "Nickname:{}\n"
        "Area:{}\n"
        "Population:{}\n".format(name, abbr, nickname, area, population))
```

函数调用

```python
response = client.chat.completions.create(
  model="gpt-3.5-turbo",
  messages=[
    {"role": "user", "content": usr_content1},
    {"role": "user", "content": usr_content2},
    {"role": "user", "content": usr_content3},
  ],
  tools = tools,
  tool_choice="auto"
)

for func in response.choices[0].message.tool_calls:
  func_name = func.function.name
  if func_name == "context_format_extract":
    args = func.function.arguments
    print("args:{}".format(args))
    args = json.loads(args)
    context_format_extract(*list(args.values()))
```

函数调用

也可以根据问答情况，发起不同的函数调用。例如我们在前面的例子中引入一个用高德API查询天气的函数。GPT会从回答中自动提取出函数调用的参数并选择恰当的函数。

```
1  def get_location_weather(location="Beijing", extensions="base", output="
       json"):
2    '''GAODE weather API'''
3    city_code = { "Beijing": 110000, "Shanghai": 310000, "Guangzhou": 440100,
         "Shenzhen": 440300}
4    citycode = city_code.get(location)
5
6    API_URL = "https://restapi.amap.com/v3/weather/weatherInfo"
7    params = {"key": os.environ['GAODE_API_KEY'], "city": citycode, "
       extensions": extensions, "output": output}
8    response = requests.get(API_URL, params=params)
9    # print(response.url)
10   weather_info = response.json()
11   print("{} realtime weather information: ".format(location))
12   ......
13   #more messages print here
```

函数调用

函数调用是OpenAI目前最主要开发的功能之一，它有很多特性目前尚不稳定，使用它时有如下关键点要注意：

- GPT目前判定该调用哪个函数并根据会话内容准备好发起调用的参数，但不会主动发起调用（feature???）
- 发起函数调用或者返回会话消息，目前这二者不可兼得，只能取其一
- 从会话内容中提取参数并不能确保百分百准确，同样的问题有可能成功也有可能失败
- 目前在配置函数、获取参数等步骤中需要用到JSON
- 对于函数的配置、描述需要支付token费用

The end