

# 项目简介

## Contents

数据获取（ETL）

- [数据获取](#)
- [清理代码](#)

访问数据库

- [连接数据库SQLite](#)

提高代码性能（上千城市数据）

- [处理多城市](#)
- [采用异步Async](#)

展示结果

- [采用Plotly展示](#)

作者: **BingBlackBean**

微信: **BingBlackBean**

公众号: **数据如琥珀**

学习Python最重要的是练习，有的时候我们学习了pandas, numpy, requests等好多函数，却不知道何时使用。这里我分享给大家一个自己写的项目：天气预测信息。

通过这个项目，我们可以练习：

- 如何通过requests 获取数据
- 如何解析requests 结果
- 如何用Pandas 清洗和转换数据
- 如何存储数据到数据库SQLite
- 如何使用SQLAlchemy 管理数据库
- 如何处理采集上千个城市天气信息
- 如何优化requests 提高代码性能
- 如何使用plotly map
- 等等，还有很多我认为不错的写代码习惯和技巧

## 数据获取

为了能够获得实时数据，我们采用调用网上的开源API来获取。一贯的，我不推荐采用爬虫获取数据。

感谢国内大神的提供：<https://www.7timer.info/>.

通过这个网站，我们可以输入地理位置（纬度和经度）来获取未来的天气预测。比如，对于上海来说，它的入口地址为：

<https://www.7timer.info/bin/api.pl?lon=121.474&lat=31.23&product=civil&output=json>

需要说明的是：这个网站提供了多个产品，用于查询天气信息，最常用的就是民用 - **CIVIL**，如果你有需要就可以使用其他**API**获取更多信息

## 发生请求Request

对一个网站的API获取信息的方式就是发送请求，有超级多的模块可以帮你完成这个任务，最常用的就是requests。这个几乎是鼻祖级别的，其他的无非是打包一些新特色。

首先我们需要安装requests，使用pip即可。

同样需要注意的是：不能拼错，记得新闻里面提到过有黑客采用障眼法，弄了一个以假乱真的**requests** 包

```
! pip install requests
```

对于requests 包（任何一个包），查看他们的quickstart就可以实战了。

<https://docs.python-requests.org/en/latest/user/quickstart/>

```
import requests
```

```
url = 'https://www.7timer.info/bin/api.pl?lon=121.474&lat=31.23&product=civil&output=json'
```

```
r = requests.get(url)
print(r)
print(r.encoding)
print(type(r.text))
```

```
<Response [200]>
UTF-8
<class 'str'>
```

对于响应的数据，我们需要把它解析成比较方便处理的python对象，比如dict或者更高级的DataFrame。

默认情况下，requests 返回的是json格式。

Python自带有json模块，可以帮助我们通过loads 转换为其他dict。

```
import json
```

```
text_j= json.loads(r.text)
print(type(text_j))
```

```
<class 'dict'>
```

我们可以查看一下text\_j的内容

```
text_j.keys()
```

```
dict_keys(['product', 'init', 'dataseries'])
```

```
text_j['product']
```

```
'civil'
```

```
text_j['init']
```

```
'2022020200'
```

```
text_j['dataseries'][0]
```

```
{'timepoint': 3,
 'cloudcover': 9,
 'lifted_index': 15,
 'prec_type': 'rain',
 'prec_amount': 1,
 'temp2m': 4,
 'rh2m': '75%',
 'wind10m': {'direction': 'NE', 'speed': 3},
 'weather': 'lightrainday'}
```

## 解析数据

我们注意到最有用的信息包含在`dataseries`元素中，对于这种list of dict格式的数据，我们建议采用`DataFrame`来解析，应为它很像excel一个表格的内容。

关于每个字段的含义，可以参考 <https://github.com/Yeqzids/7timer-issues/wiki/Wiki>

```
import pandas as pd
```

```
weather_info = pd.DataFrame(text_j['dataseries'])
weather_info.head(5)
```

	timepoint	cloudcover	lifted_index	prec_type	prec_amount	temp2m	rh2m	wind1
0	3	9	15	rain	1	4	75%	{'direction': 'NE', 'speed': 3}
1	6	9	15	rain	2	4	70%	{'direction': 'NE', 'speed': 3}
2	9	9	15	rain	2	4	71%	{'direction': 'NE', 'speed': 3}
3	12	9	15	rain	2	3	86%	{'direction': 'NE', 'speed': 3}
4	15	9	15	rain	2	3	92%	{'direction': 'NE', 'speed': 3}

## 数据变换

### 字符串转时间

“init”时间戳非常重要，但是它存储为str格式。所以我们需要还原为datetime格式。而且，我们决定采用`DataFrame`来转换原始数据，为了一致性，我们采用`pd.to_datetime`函数而不是`datetime`模块来进行变换。

```
start_time = pd.to_datetime(text_j['init'], format='%Y%m%d%H')
start_time
```

```
Timestamp('2022-02-02 00:00:00')
```

### int转timedelta

`timepoint`列表示未来N个小时，起始从init开始算起。同样的，我们需要将这个`timepoint`还原成真实的时间戳。

1. 将int转换为timedelta
2. 将timedelta变成时间戳

```
# convert timepoint to timestamp
weather_info['timestamp'] = pd.to_timedelta(weather_info['timepoint'],unit='h')
```

```
weather_info['timestamp'] = start_time+ weather_info['timepoint']
```

```
weather_info.head(5)
```

	timepoint	cloudcover	lifted_index	prec_type	prec_amount	temp2m	rh2m	wind1
0	0 days 03:00:00	9	15	rain	1	4	75%	{'directi '  'speed
1	0 days 06:00:00	9	15	rain	2	4	70%	{'directi '  'speed
2	0 days 09:00:00	9	15	rain	2	4	71%	{'directi '  'speed
3	0 days 12:00:00	9	15	rain	2	3	86%	{'directi '  'speed
4	0 days 15:00:00	9	15	rain	2	3	92%	{'directi '  'speed

## 从城市名获取地理位置

如果你想输入城市名，然后直接获取天气信息呢？那么我们就需要通过城市名先获取地理位置。幸运的是，已经有模块可以帮我们完成这个任务geopy

如果你需要的话，可以安装。

```
pip install geopy
```

关于geopy可以参考: <https://geopy.readthedocs.io/en/stable/>

```
from geopy.geocoders import Nominatim
```

```
geolocator = Nominatim(user_agent='baidu')
location = geolocator.geocode("shanghai")
print(location.address)
print((location.latitude, location.longitude))
```

```
上海市，黄浦区，上海市，200001，中国
(31.2322758, 121.4692071)
```

## 小结

到目前为止，我们一步一步的实现如何输入城市名，然后调用api获取天气预测信息。在进行其他操作之前，我们需要清理一下我们的代码，最好写成函数形式，因为未来需要重复调用它们。

## 清理代码

我们有必要清理一下我们的代码，让整个项目更整洁。

## 创建函数

为了更好的复用代码，我们将我们关键代码变为函数。

```
import requests
import json
import pandas as pd
from geopy.geocoders import Nominatim
```

```
# function to get url by inputing the city name

def get_geo_from_city(city):
    geolocator = Nominatim(user_agent='baidu')
    location = geolocator.geocode(city)
    return location.longitude, location.latitude

def generate_url(longitude, latitude):
    url = f'https://www.7timer.info/bin/api.pl?lon={longitude}&lat={latitude}&product=civil&output=json'
    return url

# function get weather raw info
def request_weather_info(url):
    r = requests.get(url)
    text_j = json.loads(r.text)
    return text_j
```

## 测试函数

对于创建好的函数，一定要先测试一下，否则后面bug套bug，比较累。

```
# test functions
city='shanghai'
lon,lat = get_geo_from_city(city)
url = generate_url(lon,lat)
text_j = request_weather_info(url)
text_j['dataseries'][0]
```

```
{'timepoint': 3,
 'cloudcover': 9,
 'lifted_index': 15,
 'prec_type': 'rain',
 'prec_amount': 1,
 'temp2m': 4,
 'rh2m': '70%',
 'wind10m': {'direction': 'NE', 'speed': 3},
 'weather': 'lightrainday'}
```

## 添加更多变换

我们需要对原始的代码稍作升级，这样可以保存更多的信息。

1. 我们需要把wind列分成两列，风速和风向
2. 同时，我们需要把更多的信息保存到DataFrame中，比如城市信息。因为我们后面会添加多个城市的信息

```
# transform data

def transform_weather_raw(text_j):
    weather_info = pd.DataFrame(text_j['dataseries'])
    start_time = pd.to_datetime(text_j['init'],format='%Y%m%d%H')
    weather_info['timepoint'] = pd.to_timedelta(weather_info['timepoint'],unit='h')
    weather_info['timestamp'] = start_time+ weather_info['timepoint']
    weather_info.drop('timepoint',axis=1,inplace=True)
    # more clean data steps
    wind_df = pd.json_normalize(weather_info['wind10m'])
    wind_df.columns = ['wind_'+col for col in wind_df.columns]
    weather_info = pd.concat([weather_info,wind_df],axis=1)
    weather_info.drop('wind10m',axis=1,inplace=True)
    weather_info['rh2m'] = weather_info['rh2m'].str.rstrip('%')
    #['']
    return weather_info

def add_city_info(weather_info,longitude,latitude,city):
    weather_info['longitude'] = longitude
    weather_info['latitude'] = latitude
    weather_info['city'] = city
    return weather_info

weather_info_df = transform_weather_raw(text_j)
weather_info_df = add_city_info(weather_info_df,lon,lat,city)
weather_info_df
```

	cloudcover	lifted_index	prec_type	prec_amount	temp2m	rh2m	weather	tin
0	9	15	rain	1	4	70	lightrainday	02
1	9	15	rain	2	4	70	lightrainday	02
2	9	15	rain	2	4	70	lightrainday	02
3	9	15	rain	2	3	87	lightrainnight	02
4	9	15	rain	2	3	90	lightrainnight	02
...	...	...	...	...	...	...	...	...
59	9	15	rain	4	5	74	rainnight	09
60	9	15	none	4	5	73	cloudynight	09
61	7	15	none	4	4	78	mcloudynight	09
62	2	15	none	4	4	83	clearnight	09
63	2	15	none	4	4	75	clearday	10

64 rows × 13 columns

## 连接数据库SQLite

### 准备数据

之前的章节我们已经测试了Request API接口，并且把代码做了规范化处理。本节练习中，我们需要先把一些函数从notebook中导出到py文件中，这样就可以复用了。

需要注意的是: **Pycharm** 中软件帮我们设置了项目的根目录，并且把根目录包含在系统**path**中。所以当我们用**import** 导入时，**python**可以找到我们自己写的模块。这里我们采用的是原生的**jupyter notebook**，需要手动将我们的模块添加到系统**path**中。一种方法是手动添加到环境变量中，这是永久方法。还有一种方法是添加临时**path**。本文就是采用后一种方式。

```
import os
import sys

module_path = os.path.abspath(os.path.join('..'))
print(module_path)
if module_path not in sys.path:
    sys.path.append(module_path)
```

```
C:\Users\renb\PycharmProjects\weather_dashapp\weather_book
```

## 项目结构

```
weather_book
├──.ipynb_checkpoints
├──data
├──steps  <-- notebook 主要在这里
├──weather_app
└──models  <-- 函数写在这里
```

添加完path之后，就可以导入自己的模块了。注意如果是上面的路径中采用的“..”,说明只返回上一层，如果是“../..”,说明返回了两层。返回一层和两层对我的项目的区别是，以下代码是否需要包含weather\_book。如果只返回一层，系统可以直接找到weather\_app 模块，但是无法找到weather\_book（因为已经在weather\_book模块内了，不识模块真面目，只缘身在此山中）。

```
from weather_app.models.query_api import
get_geo_from_city,generate_url,request_weather_info,transform_weather_raw,add_city_info
```

一个好的习惯是，对你的函数进行快速测试。

```
city = 'shanghai'
lon, lat = get_geo_from_city(city)
url = generate_url(lon, lat)
text_j = request_weather_info(url)
weather_info_df = transform_weather_raw(text_j)
weather_info_df = add_city_info(weather_info_df, lon, lat, city)
weather_info_df
```

	cloudcover	lifted_index	prec_type	prec_amount	temp2m	rh2m	weather	tim
0	9	15	none	0	5	54	cloudyday	<sup>2</sup> 03
1	9	15	none	0	5	51	cloudyday	<sup>2</sup> 03
2	9	15	none	1	4	75	cloudyday	<sup>2</sup> 03
3	9	15	rain	1	4	82	lightrainnight	<sup>2</sup> 03
4	9	15	rain	1	4	68	lightrainnight	<sup>2</sup> 03
...	...	...	...	...	...	...	...	
59	3	15	none	4	5	71	pcloudynight	<sup>2</sup> 10
60	9	15	none	4	5	71	cloudynight	<sup>2</sup> 10
61	9	15	none	4	5	74	cloudynight	<sup>2</sup> 10
62	9	15	none	4	5	75	cloudynight	<sup>2</sup> 10
63	9	15	none	4	5	62	cloudyday	<sup>2</sup> 11

64 rows × 13 columns


## 引入SQLite

我们可以使用任意的关系数据库保存这些数据，但是这里我强烈推荐SQLite，因为它轻量，无服务器，免安装，开箱即用。对于新手来说，完全是无痛体验。但是它支持标准的SQL查询语言的功能，我们一样可以用它练习数据库的增删查操作。

如果想快速入门，可以参考中文网站：<https://www.runoob.com/sqlite/sqlite-intro.html#:~:text=SQLite是一个进程内,直接访问其存储文件。>

## 使用 SQLAlchemy

我们已经选择了SQLite 作为我们的数据库，作为“客户端”python和数据库之间的衔接，我们可以选择很多库，这里推荐的是SQLAlchemy。这里对SQLAlchemy做一个简单介绍，SQLAlchemy 是用于处理数据库和Python的工具。他的主要组成其实是两部分：core和ORM。core属于低阶API，ORM属于高阶API。也就是说，ORM建立在Core上。

 首先我们需要安装SQLAlchemy，采用pip安装。

```
pip install sqlalchemy
```

<https://docs.sqlalchemy.org/en/14/tutorial/index.html>

首先我们需要建立对话，调用create\_engine 函数。对于SQLite 来说，URL格式就是本地数据库（文件）的路径，这里我需要在当前文件夹生成这个数据库，所以不会添加其他相对路径，只有一个文件名而已。

```
from sqlalchemy import create_engine
```

```
engine = create_engine('sqlite:///weather.db')
```

```
engine
```

```
Engine(sqlite:///weather.db)
```



想要做增删查的操作，就需要建立一个Session会话。这样我们就有一个session为我们敞开大门，等着我们进行操作。

```
from sqlalchemy.orm import Session

session = Session(engine)

session

<sqlalchemy.orm.session.Session at 0x214dbafbf40>
```

## 使用ORM 管理表

使用ORM最常见的方式就是使用 declarative\_base() 函数构造一个基类，该函数会将声明性映射应用于从它派生的所有子类。所以我们需要建立class，然后从Base继承。 ([https://docs.sqlalchemy.org/en/20/orm/mapping\\_styles.html](https://docs.sqlalchemy.org/en/20/orm/mapping_styles.html))

参考 DataFrame的结构，我们创建一个 Python 类，并将 DataFrame 中的每一列作为属性包含在类中，每个类属性对应表中的特定列。

```
# create database table by defining python class
from sqlalchemy.orm import declarative_base
from sqlalchemy import Column,Integer,String,Float
Base = declarative_base()

class WeatherInfo(Base):
    __tablename__ = 'weather'
    __table_args__ = {'extend_existing': True}
    id = Column(Integer,primary_key=True,autoincrement=True) # use autoincrement
    timestamp = Column(String(55))
    cloudcover = Column(Integer)
    lifted_index = Column(Integer)
    prec_type = Column(String(10))
    prec_amount = Column(Integer)
    temp2m = Column(Integer)
    rh2m = Column(Integer)
    weather = Column(String(20))
    wind_direction = Column(String(4))
    wind_speed = Column(Integer)
    longitude = Column(Float(precision=10, decimal_return_scale=2))
    latitude = Column(Float(precision=10, decimal_return_scale=2))
    city=Column(String(50))
```

## 从数据库查询和插入

如下文所示，由于数据表是全新创建的，因此查询得到0结果。

我们将 DataFrame 插入数据库后，重新查询会获取更多数据。

请记住在所有事务之后关闭会话。

```
Base.metadata.create_all(engine)
result = session.query(WeatherInfo).all()
len(result)

83393

weather_info_df.to_sql('weather',engine,if_exists='append',index=False) # without index

64

result = session.query(WeatherInfo).all()
len(result)

83393
```

```
session.close()
```

## 处理多城市

### 获取全球城市

我们已经知道如何获取一个城市的天气信息，如果有多个城市，操作方式也是类似的，无非就是添加一个循环而已。

但是如何获取城市列表呢？这里推荐一个网站：<https://simplemaps.com/data/world-cities>

这里面包含全球城市的完整信息，不仅包含城市名称，还包括国家，经纬度，是否是首都，省会等信息。

下载后，解压，就可以通过pandas 读取了。

```
import pandas as pd
```

```
city_file = '../data/worldcities.csv'  
city_df = pd.read_csv(city_file,encoding='utf-8')
```

```
city_df
```

	city	city_ascii	lat	lng	country	iso2	iso3	admin_name
0	Tokyo	Tokyo	35.6897	139.6922	Japan	JP	JPN	Tōkyō
1	Jakarta	Jakarta	-6.2146	106.8451	Indonesia	ID	IDN	Jakarta
2	Delhi	Delhi	28.6600	77.2300	India	IN	IND	Delhi
3	Mumbai	Mumbai	18.9667	72.8333	India	IN	IND	Mahārāshtra
4	Manila	Manila	14.6000	120.9833	Philippines	PH	PHL	Manila
...	...	...	...	...	...	...	...	...
40996	Tukchi	Tukchi	57.3670	139.5000	Russia	RU	RUS	Khabarovskiy Kray
40997	Numto	Numto	63.6667	71.3333	Russia	RU	RUS	Khanty-Mansiyskiy Avtonomnyy Okrug-Yugra
40998	Nord	Nord	81.7166	-17.8000	Greenland	GL	GRL	Sermersooq
40999	Timmiarmiut	Timmiarmiut	62.5333	-42.2167	Greenland	GL	GRL	Kujalleq
41000	Nordvik	Nordvik	74.0165	111.5100	Russia	RU	RUS	Krasnoyarskiy Kray

41001 rows × 11 columns

### 数据筛选

我们仅仅对国内的主要城市感兴趣，所以，我们需要对DataFrame进行筛选。

```
capital_china = city_df[(city_df['country']=='China') & (city_df['capital'] is not None)]  
capital_china
```

	city	city_ascii	lat	lng	country	iso2	iso3	admin_name	capital
5	Shanghai	Shanghai	31.1667	121.4667	China	CN	CHN	Shanghai	admin
9	Guangzhou	Guangzhou	23.1288	113.2590	China	CN	CHN	Guangdong	admin
10	Beijing	Beijing	39.9050	116.3914	China	CN	CHN	Beijing	primary
17	Shenzhen	Shenzhen	22.5350	114.0540	China	CN	CHN	Guangdong	regional
29	Nanyang	Nanyang	32.9987	112.5292	China	CN	CHN	Henan	
...	...	...	...	...	...	...	...	...	...
40725	Taoyan	Taoyan	34.7706	103.7903	China	CN	CHN	Gansu	
40744	Jingping	Jingping	33.7844	104.3652	China	CN	CHN	Gansu	
40776	Dayi	Dayi	33.8312	104.0362	China	CN	CHN	Gansu	
40782	Biancang	Biancang	33.9007	104.0321	China	CN	CHN	Gansu	
40938	Nichicun	Nichicun	29.5333	94.4167	China	CN	CHN	Tibet	

1498 rows × 11 columns

## 小技巧

一个好的习惯是，先对单个数据进行处理，确保无误后，再对所有数据进行循环。

所以我们这里依然需要测一下我们的函数能否获取单个城市的天气信息。

```
import os
import sys

module_path = os.path.abspath(os.path.join '..', '..'))
print(module_path)
if module_path not in sys.path:
    sys.path.append(module_path)
```

C:\Users\renb\PycharmProjects\weather\_dashapp\weather\_book

```
from weather_app.models.query_api import
get_geo_from_city, generate_url, request_weather_info, transform_weather_raw, add_city_info
```

```
city_info = capital_china.iloc[0,:].to_dict()
city_info
```

```
{'city': 'Shanghai',
 'city_ascii': 'Shanghai',
 'lat': 31.1667,
 'lng': 121.4667,
 'country': 'China',
 'iso2': 'CN',
 'iso3': 'CHN',
 'admin_name': 'Shanghai',
 'capital': 'admin',
 'population': 22120000.0,
 'id': 1156073548}
```

```
city = city_info['city']
lon = city_info['lng']
lat = city_info['lat']
url = generate_url(longitude=lon, latitude=lat)
text_j = request_weather_info(url)
weather_info_df = transform_weather_raw(text_j)
weather_info_df = add_city_info(weather_info_df, lon, lat, city)
weather_info_df
```

	cloudcover	lifted_index	prec_type	prec_amount	temp2m	rh2m	weather	time
0	9	15	none	0	5	53	cloudyday	203
1	9	15	none	0	5	51	cloudyday	203
2	9	15	none	1	4	72	cloudyday	203
3	9	15	rain	1	4	81	lightrainnight	203
4	9	15	rain	1	4	66	lightrainnight	203
...	...	...	...	...	...	...	...	...
59	3	15	none	4	5	72	pcloudynight	210
60	9	15	none	4	5	70	cloudynight	210
61	9	15	none	4	5	71	cloudynight	210
62	9	15	none	4	5	74	cloudynight	210
63	9	15	none	4	5	63	cloudyday	211

64 rows × 13 columns

## 获取多个城市信息

确保单个城市可以准确获取信息后，我们建立循环。通过iterrows 逐行进行处理。

这里我们先用10个城市测测性能。

可以看到每个循环需要等网站回复很慢，2-3秒一个循环，如果是2000个城市的话，需要80分钟

```
from tqdm.notebook import tqdm
all_cities_df = pd.DataFrame()
for index,city_info in tqdm(capital_china.iloc[0:10,:].iterrows()):
    city = city_info['city']
    print(city)
    lon = city_info['lng']
    lat = city_info['lat']
    url = generate_url(longitude=lon,latitude=lat)
    text_j = request_weather_info(url)
    weather_info_df = transform_weather_raw(text_j)
    weather_info_df = add_city_info(weather_info_df,lon,lat,city)
    all_cities_df = pd.concat([all_cities_df,weather_info_df],axis=0)
```

Shanghai
Guangzhou
Beijing
Shenzhen
Nanyang
Baoding
Chengdu
Linyi
Tianjin
Shijiazhuang

```
all_cities_df.head()
```

	cloudcover	lifted_index	prec_type	prec_amount	temp2m	rh2m	weather	time
0	9	15	none	0	5	50	cloudyday	2003 0
1	9	15	none	0	5	55	cloudyday	2003 0
2	9	15	none	1	4	73	cloudyday	2003 0
3	9	15	rain	1	4	83	lightrainnight	2003 1
4	9	15	rain	1	4	69	lightrainnight	2003 1

小结

我们可以通过简单的循环来获取多个城市的天气信息，但是性能不佳。

采用异步Async

问题分析

我们上文中获取天气信息的性能不佳，主要在给网站发送请求的时候，需要等很久才能获得回复。

这节我们需要想办法提高性能。

我们先把关键代码复制到本节。

```
import pandas as pd

city_file = '../data/worldcities.csv'
city_df = pd.read_csv(city_file,encoding='utf-8')
city_df
```

	city	city_ascii	lat	lng	country	iso2	iso3	admin_name
0	Tokyo	Tokyo	35.6897	139.6922	Japan	JP	JPN	Tōkyō
1	Jakarta	Jakarta	-6.2146	106.8451	Indonesia	ID	IDN	Jakarta
2	Delhi	Delhi	28.6600	77.2300	India	IN	IND	Delhi
3	Mumbai	Mumbai	18.9667	72.8333	India	IN	IND	Mahārāshtra
4	Manila	Manila	14.6000	120.9833	Philippines	PH	PHL	Manila
...	...	...	...	...	...	...	...	...
40996	Tukchi	Tukchi	57.3670	139.5000	Russia	RU	RUS	Khabarovskiy Kray
40997	Numto	Numto	63.6667	71.3333	Russia	RU	RUS	Khanty-Mansiyskiy Avtonomnyy Okrug-Yugra
40998	Nord	Nord	81.7166	-17.8000	Greenland	GL	GRL	Sermersooq
40999	Timmiarmiut	Timmiarmiut	62.5333	-42.2167	Greenland	GL	GRL	Kujalleq
41000	Nordvik	Nordvik	74.0165	111.5100	Russia	RU	RUS	Krasnoyarskiy Kray

41001 rows × 11 columns

```
capital_china = city_df[(city_df['country']=='China') & (city_df['capital'] is not None)]
capital_china
```

	city	city_ascii	lat	lng	country	iso2	iso3	admin_name	capital
5	Shanghai	Shanghai	31.1667	121.4667	China	CN	CHN	Shanghai	capital
9	Guangzhou	Guangzhou	23.1288	113.2590	China	CN	CHN	Guangdong	capital
10	Beijing	Beijing	39.9050	116.3914	China	CN	CHN	Beijing	primary
17	Shenzhen	Shenzhen	22.5350	114.0540	China	CN	CHN	Guangdong	regional
29	Nanyang	Nanyang	32.9987	112.5292	China	CN	CHN	Henan	
...	...	...	...	...	...	...	...	...	...
40725	Taoyan	Taoyan	34.7706	103.7903	China	CN	CHN	Gansu	
40744	Jingping	Jingping	33.7844	104.3652	China	CN	CHN	Gansu	
40776	Dayi	Dayi	33.8312	104.0362	China	CN	CHN	Gansu	
40782	Biancang	Biancang	33.9007	104.0321	China	CN	CHN	Gansu	
40938	Nichicun	Nichicun	29.5333	94.4167	China	CN	CHN	Tibet	

1498 rows × 11 columns

```
def generate_url(longitude,latitude):
    url = f'https://www.7timer.info/bin/api.pl?lon={longitude}&lat={latitude}&product=civil&output=json'
    return url

def transform_weather_raw(text_j):
    weather_info = pd.DataFrame(text_j['dataseries'])
    start_time = pd.to_datetime(text_j['init'],format='%Y%m%d%H')
    weather_info['timepoint'] = pd.to_timedelta(weather_info['timepoint'],unit='h')
    weather_info['timestamp'] = start_time+ weather_info['timepoint']
    weather_info.drop('timepoint',axis=1,inplace=True)
    # more clean data steps
    wind_df = pd.json_normalize(weather_info['wind10m'])
    wind_df.columns = ['wind_'+col for col in wind_df.columns]
    weather_info = pd.concat([weather_info,wind_df],axis=1)
    weather_info.drop('wind10m',axis=1,inplace=True)
    weather_info['rh2m'] = weather_info['rh2m'].str.rstrip('%')
    #['']
    return weather_info

def add_city_info(weather_info,longitude,latitude,city):
    weather_info['longitude'] = longitude
    weather_info['latitude'] = latitude
    weather_info['city'] = city
    return weather_info
```

## 采用异步请求

我们已经简单分析过，发送一个请求，我们需要等网站很久（2秒）才能得到回复。对于上千个城市，我们等不起啊。其实很多网站API都是接受多个客户端请求的，也就是同时可以接收多个get request。

所以我们不用等第一个request返回结果，就把第二个，第三个，第N个request发送出去，这样等第一个request返回结果时，我们再处理就可以了。这样就不用傻等了，这就是异步的思想。

这里我们就采用requests的异步升级版本grequests来进行异步操作。

安装依然很容易： `pip install grequests`

入门稍微有点难，主要涉及到一些异步的思想和map的思想。不过对于我们入门来说，只需要调用一个函数即可： `grequests.imap(rs, size=50)`

这里的size就是每次发送请求的数量。

```

import requests
import json
from tqdm.notebook import tqdm

city_list = []
lon_list = []
lat_list = []
url_list = []
for index,city_info in capital_china.iterrows():
    city = city_info['city']
    city_list.append(city)
    lon = city_info['lng']
    lon_list.append(lon)
    lat = city_info['lat']
    lat_list.append(lat)
    url = generate_url(longitude=lon,latitude=lat)
    url_list.append(url)

rs = (requests.get(u) for u in url_list)
all_cities_df = pd.DataFrame()
for i,r in tqdm(enumerate(requests.imap(rs, size=50))):
    text_j= json.loads(r.text)
    weather_info_df = transform_weather_raw(text_j)
    weather_info_df = add_city_info(weather_info_df,lon_list[i],lat_list[i],city_list[i])
    all_cities_df = pd.concat([all_cities_df,weather_info_df],axis=0)

```

```

c:\users\renb\pycharmprojects\weather_dashapp\dash\lib\site-packages\gevent\hub.py:161:
UserWarning: libuv only supports millisecond timer resolution; all times less will be set to 1 ms
with loop.timer(seconds, ref=ref) as t:

```

获取的数据，我们需要保存到数据库里。

```

import os
import sys

module_path = os.path.abspath(os.path.join('../..'))
print(module_path)
if module_path not in sys.path:
    sys.path.append(module_path)
from weather_book.weather_app.models.db_models import engine,WeatherInfo
all_cities_df['id'] = [i for i in range(all_cities_df.shape[0])]
all_cities_df.to_sql('weather',engine,if_exists='append',index=False) # without index

```

```

C:\Users\renb\PycharmProjects\weather_dashapp

```

```

83392

```

```

c:\users\renb\pycharmprojects\weather_dashapp\dash\lib\site-packages\gevent\hub.py:161:
UserWarning: libuv only supports millisecond timer resolution; all times less will be set to 1 ms
with loop.timer(seconds, ref=ref) as t:

```

## 采用Plotly展示

### 从数据库查询

我们将之前保存的全国主要城市的天气信息从数据库中读取。

```

import os
import sys

module_path = os.path.abspath(os.path.join('../..'))
print(module_path)
if module_path not in sys.path:
    sys.path.append(module_path)

```

```

C:\Users\renb\PycharmProjects\weather_dashapp

```

```

from weather_book.weather_app.models.db_models import engine,WeatherInfo
import pandas as pd

```

```
df = pd.read_sql_table(WeatherInfo.__tablename__,engine)
```

```
df.shape
```

```
(89664, 14)
```

这里我们仅仅展示一个时刻的信息，所以我们需要对每个城市的数据“去重”，仅仅保留最后一个数据。

需要强调的是：如果数据库比较乱，每个城市最后一个数据对应的时间戳并不相同。

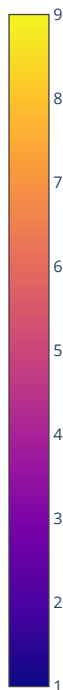
```
df.drop_duplicates(subset=['city'], keep='last',inplace=True)  
df.shape
```

```
(1237, 14)
```

```
df.head()
```

	cloudcover	lifted_index	prec_type	prec_amount	temp2m	rh2m	weather	time
5823	1	15	none	0	9	12	clearday	2010-01-01T00:00:00
5887	2	15	rain	4	4	31	rainday	2010-01-01T00:00:00
5951	1	15	none	0	5	10	clearday	2010-01-01T00:00:00
6015	1	15	none	0	7	11	clearday	2010-01-01T00:00:00
6079	9	15	none	3	11	47	cloudyday	2010-01-01T00:00:00

```
import plotly.graph_objects as go  
import plotly.offline as pyo  
pyo.init_notebook_mode()  
fig = go.Figure(go.Densitymapbox(lat=df.latitude, lon=df.longitude, z=df.cloudcover,  
                                radius=20))  
fig.update_layout(mapbox_style="stamen-terrain",  
mapbox_center_lon=112,mapbox_center_lat=28,mapbox_zoom=3)  
fig.update_layout(margin={"r":0,"t":0,"l":0,"b":0})  
fig.show()
```





## 总结

至此我们就完成了该项目。当然了由于是入门级教程，很多内容只涉及到相关的点，没有进一步深入。

不过我个人认为，这是最快的学习方式，就是实践中不断学习，然后总结，再学习。

---

By BingBlackBean

© Copyright 2021.