Start\_bat,bat:

@echo off

if "%1" == "h" goto begin

mshta vbscript:createobject("wscript.shell").run("""%~0"" h",0)(window.close)&&exit

:begin

D: & cd D:\Data\_upload\_tool &call .venv\Scripts\activate.bat

python BVPP\_Server.py

start.bat:

cmd.exe /K .\python-3.10.9-embed-amd64\pythonw Push\_Server.py

Push\_Server.py:

import sys

import os

BASE\_DIR = os.path.dirname(os.path.abspath(\_\_file\_\_))

rootPath = os.path.split(BASE\_DIR)[0]

sys.path.append(rootPath)

import numpy as np

import datetime

# from tools.get\_conn import get\_conn\_db

import struct

import time

import json

from tools.logging\_conf import logger

from tools.Create\_xml import Xml\_Producer

from tools.get\_data\_from\_api import Get\_datas

from Config.settings import config,config0

from tools.Breakpoint\_up import put\_data

import threading

# from tools.get\_conn import get\_conn\_db

from Config.template import template\_send,byte\_types

from tools.Modbus\_tcp\_con import get\_modbus\_tcp\_data

root\_path\_config = config.get('root\_path\_config')

chunk\_size\_config = config0.get('chunk\_size')

chunk\_number\_config = config0.get('chunk\_number')

chunk\_size = int(chunk\_size\_config)

num\_chunk = int(chunk\_number\_config)

hostAdress0 = "162.168.8.51"

hostAdress1 = "162.168.8.52"

with open(root\_path\_config + '/' + 'Branch\_code1.json') as f:

data = json.load(f)

# 对字典进行分割

def dict\_chunk(dicts, size):

new\_list = []

dict\_len = len(dicts)

# 获取分组数

while\_count = dict\_len // size + 1 if dict\_len % size != 0 else dict\_len / size

split\_start = 0

split\_end = size

while (while\_count > 0):

# 把字典的键放到列表中，然后根据偏移量拆分字典

new\_list.append({k: dicts[k] for k in list(dicts.keys())[split\_start:split\_end]})

split\_start += size

split\_end += size

while\_count -= 1

return new\_list

#获取sqlserver数据库数据

class Push\_data\_Sever(object):

\_Meterstate = [[] for \_ in range(num\_chunk)]

def \_\_init\_\_(self):

# 时间戳-年月日时分秒

nowtime1 = time.strftime('%Y-%m-%d %H:%M:%S', time.localtime(time.time()))

self.t1 = ''.join([x for x in nowtime1 if x.isdigit()])

# 建筑编号

self.topic0 = config0.get('user')

#modbus\_tcp获取数据

def get\_data\_from\_modbustcp(self):

start = datetime.datetime.now()

try:

lst = []

for Addr in [hostAdress0, hostAdress1]:

lst\_ = get\_modbus\_tcp\_data(Addr)

lst.extend(lst\_)

dict\_ = {i: lst[i] for i in range(len(lst))}

print(dict\_)

# 数据包数量

chunks = dict\_chunk(dicts=dict\_, size=chunk\_size)

for index, item in enumerate(chunks):

print(index, item)

# # 设备ID-01

Push\_data\_Sever.\_Meterstate[index].append(index + 1)

# # 获取当前时间(年-月-日 时:分:秒)

Push\_data\_Sever.\_Meterstate[index].extend(

[int(self.t1[2:4]), int(self.t1[4:6]), int(self.t1[6:8]), int(self.t1[8:10]),

int(self.t1[10:12]),

int(self.t1[12:14])])

# 电表数量

Push\_data\_Sever.\_Meterstate[index].append(len(item))

print(item)

# # 生成字节数据# 添加电表数据

payload = self.generate\_bytes(index\_=index, dict\_=item)

self.generate\_xml(payload)

time.sleep(3)

finally:

table\_name = 'modbus\_tcp'.format(hostAdress0,hostAdress1)

# 根据序列号判断是否执行续传检查

end = datetime.datetime.now()

s = (end - start).total\_seconds()

print('数据导出: %s, 耗时: %s 秒' % (table\_name, s))

logger.info('数据导出: %s, 耗时: %s 秒' % (table\_name, s))

#从数据库获取数据

def get\_data\_from\_sqlserver(self,table\_name):

"""

从sqlserver导出数据文件到本地

"""

start = datetime.datetime.now()

#存放键值对

lst = []

conn = None

try:

conn = get\_conn\_db('SOURCE')

if conn is None:

# 日志记录

logger.info("获取数据库连接失败")

raise Exception('获取数据库连接失败')

sql="""select a.PtId,round(a.Value,1) from {0} a,(select PtId,max(RecTime) create\_time from {0} where mod(PtId, 2)=0 group by PtId) b

where a.PtId = b.PtId and a.RecTime = b.create\_time""".format(table\_name)

# sql = "SELECT CODE,round(PRESENT\_VALUE,2) FROM {0} WHERE CODE LIKE '%Epd'".format(table\_name) # 数据库查询语句

with conn.cursor() as cur:

cur.execute(sql)

results = cur.fetchall()

for result in results:

# tuple\_list = tuple([str(i) for i in list(result)][0:])

# print(result)

result = list(result) # 元组转化为列表

for res in range(len(result)):

if isinstance(result[res], str):

result[res] = result[res].replace(' ', '')

# 解决空格问题

result = tuple(result) # 列表再转换为元组

lst.append(result)

# 将转换为字典

dict\_ = dict(lst)

print(dict\_)

# 数据包数量

chunks = dict\_chunk(dicts=dict\_, size=chunk\_size)

for index, item in enumerate(chunks):

print(index, item)

# # 设备ID-01

Push\_data\_Sever.\_Meterstate[index].append(index + 1)

# # 获取当前时间(年-月-日 时:分:秒)

Push\_data\_Sever.\_Meterstate[index].extend(

[int(self.t1[2:4]), int(self.t1[4:6]), int(self.t1[6:8]), int(self.t1[8:10]),

int(self.t1[10:12]),

int(self.t1[12:14])])

# 电表数量

Push\_data\_Sever.\_Meterstate[index].append(len(item))

print(item)

# # 生成字节数据# 添加电表数据

payload = self.generate\_bytes(index\_=index, dict\_=item)

self.generate\_xml(payload)

time.sleep(3)

finally:

table\_name = 'meter'

#根据序列号判断是否执行续传检查

end = datetime.datetime.now()

s = (end - start).total\_seconds()

print('数据导出: %s, 耗时: %s 秒' % (table\_name, s))

logger.info('数据导出: %s, 耗时: %s 秒' % (table\_name, s))

def get\_data\_from\_api(self):

try:

start = datetime.datetime.now()

# 转换为int类型的10位时间戳

timestam = int(time.time())

time\_new = str(timestam)

params = {"apiKey": '1555312928870420483',

"timestamp": time\_new}

res = Get\_datas().get\_data(params)

if res:

dict\_ = res.get('meters')

# 数据包数量

num\_chunks = len(dict\_) // chunk\_size + (len(dict\_) % chunk\_size > 0)

chunks = [dict\_[i \* chunk\_size:(i + 1) \* chunk\_size] for i in range(num\_chunks)]

for index, item in enumerate(chunks):

# # 设备ID-01

Push\_data\_Sever.\_Meterstate[index].append(index + 1)

# # 获取当前时间(年-月-日 时:分:秒)

Push\_data\_Sever.\_Meterstate[index].extend(

[int(self.t1[2:4]), int(self.t1[4:6]), int(self.t1[6:8]), int(self.t1[8:10]),

int(self.t1[10:12]),

int(self.t1[12:14])])

# 电表数量

Push\_data\_Sever.\_Meterstate[index].append(len(item))

# 生成字节数据# 添加电表数据

payload = self.generate\_bytes(index\_=index, dict\_=item)

self.generate\_xml(payload)

time.sleep(8)

finally:

table\_name = 'meter'

#根据序列号判断是否执行续传检查

end = datetime.datetime.now()

s = (end - start).total\_seconds()

print('数据导出: %s, 耗时: %s 秒' % (table\_name, s))

logger.info('数据导出: %s, 耗时: %s 秒' % (table\_name, s))

def generate\_xml(self,payload):

# 采集设备打包每次发送消息包含所有点表参数数据

# 生成xml包数据

xml\_str\_obj = Xml\_Producer(self.topic0, payload)

sequence = xml\_str\_obj.creat\_xml()[2]

print("更新完成,共写入%d条数据！" % sequence)

# 记录xml数据包序列数量[一天48个包，每天检查一次]

if sequence % 24 == 0:

thread = threading.Thread(target=put\_data, args=(f'{self.topic0}',))

thread.start()

# put\_data(buildid=self.topic0)

#api接口【优化发送模板】

def generate\_bytes(self,index\_,dict\_):

"""

生成字节数流

:param arr:

:return:

uint8 = B

int8 = b

uint16 = H

int16 = h

uint32 = I

int32 = i

< 小端

> 大端

利用template\_send改写========

"""

arr = []

type = '>BBBBBBBB'

datain = list(data.keys())

print(datain)

for key,value in dict\_.items():

print("hhhh",key,value)

type += template\_send[0]['type']+template\_send[1]['type']

id = int(data.get(f'{key}'))

print(f"第{id}块表", id,key)

if str(key) in datain:

x = int(value \* template\_send[1]['mtp'])

print("数据",x)

if byte\_types[template\_send[1]['type']]['minvalue'] <= x <= byte\_types[template\_send[1]['type']]['maxvalue']:

arr.extend([id,x])

else:

arr.extend([id,byte\_types[template\_send[0]['type']]['default']])

else:

arr.extend([id,byte\_types[template\_send[0]['type']]['default']])

print(arr)

Push\_data\_Sever.\_Meterstate[index\_].extend(arr)

data\_list = Push\_data\_Sever.\_Meterstate[index\_]

to\_bytes= struct.pack(type, \*data\_list)

print(to\_bytes)

return to\_bytes

if \_\_name\_\_ == '\_\_main\_\_':

obj =Push\_data\_Sever()

# obj.get\_data\_from\_sqlserver(table\_name="IO\_Sta2AccHour202307")

# obj.get\_data\_from\_api()

obj.get\_data\_from\_modbustcp()

Modbus\_test.py:

import time

import modbus\_tk

import modbus\_tk.defines as cst

import modbus\_tk.modbus\_tcp as modbus\_tcp

import struct

import pandas as pd

logger = modbus\_tk.utils.create\_logger("console")

def ReadFloat(\*args, reverse=True):

"""

生成字节数流

:param arr:

:return:

uint8 = B

int8 = b

uint16 = H

int16 = h

uint32 = I

int32 = i

< 小端

> 大端

:param args:

:param reverse:

:return:

"""

# 解析浮点数

try:

for i, j in args:

i, j = '%04x' % i, '%04x' % j

if reverse:

v = i + j

else:

v = j + i

y\_bytes = bytes.fromhex(v)

y = struct.unpack('>i', y\_bytes)[0]

y = round(y, 6)

return y

except:

logger.warning("Error 2000: Failed When dec2float!" )

a = -88.8

return a

pass

# 32位有符号数据，低16位前，高16位后

# 配电房数据处理

def ModbusTcp\_server7(hostAdress,portName,slaveAdress):

try:

result = tuple()

master = modbus\_tcp.TcpMaster(host=hostAdress, port=portName)

master.set\_timeout(50.0)

logger.info("connected")

#指令格式：机号 功能代码 起始地址 结束地址【读取寄存器数据】

data0 = master.execute(slaveAdress,cst.READ\_HOLDING\_REGISTERS,1000,126)

data1 = master.execute(slaveAdress, cst.READ\_HOLDING\_REGISTERS,1126,126)

for i in [data0, data1]:

result = result.\_\_add\_\_(i)

except modbus\_tk.modbus.ModbusError as e:

logger.error("%s- Code=%d" % (e, e.get\_exception\_code()))

return result

# 配电房数据处理

def ModbusTcp\_server6(hostAdress, portName, slaveAdress):

try:

result = tuple()

master = modbus\_tcp.TcpMaster(host=hostAdress, port=portName)

master.set\_timeout(25.0)

logger.info("connected")

#指令格式：机号 功能代码 起始地址 结束地址【读取寄存器数据】

data0 = master.execute(slaveAdress,cst.READ\_HOLDING\_REGISTERS,1000,190)

data1 = master.execute(slaveAdress, cst.READ\_HOLDING\_REGISTERS,1191,190)

for i in [data0,data1]:

result = result.\_\_add\_\_(i)

except modbus\_tk.modbus.ModbusError as e:

logger.error("%s- Code=%d" % (e, e.get\_exception\_code()))

return result

if \_\_name\_\_ == '\_\_main\_\_':

# init 初始化信息

# 162.168.8.120

# 6号配电房 162.168.8.51

# 7号配电房 162.168.8.52

# 从站的IP地址

# hostAdress = "162.168.8.52"

while True:

hostAdress = "162.168.8.51"

# 端口号码

portName = 502

# 从站的站号

slaveAdress = 1

if time.localtime().tm\_min % 2 == 0 and time.localtime().tm\_sec == 0: # 每2分钟存数据为Excel表格

data = ModbusTcp\_server6(hostAdress, portName, slaveAdress)

k\_V = {}

for i in range(round(len(data)/2)):

s = ReadFloat(data[i \* 2:i \* 2 + 2], reverse=False)

print(s)

k\_V[i] = round(s\*0.1, 1)

# #点表数据

print(k\_V)

print(data)

# value = list(k\_V.values())

# print("配电房数据", value)

key = list(k\_V.keys())

value = list(k\_V.values())

result\_excel = pd.DataFrame()

result\_excel["点号"] = key

result\_excel["遥测值"] = value

# 写入excel

result\_excel.to\_excel(r"C:\Users\Administrator\Desktop\TEST\配电房数据.xlsx")

M.py:

import serial

import sys

import modbus\_tk

import modbus\_tk.defines as cst

from modbus\_tk import modbus\_rtu

PORT = "502"

def main():

# Connect to the slave

master = modbus\_rtu.RtuMaster(

serial.Serial(port=PORT, baudrate=9600, bytesize=8, parity='N', stopbits=1, xonxoff=0)

)

master.set\_timeout(5.0)

master.set\_verbose(True)

'''

1、READ\_COILS H01 读线圈

2、READ\_DISCRETE\_INPUTS H02 读离散输入

3、READ\_HOLDING\_REGISTERS H03 读寄存器

4、READ\_INPUT\_REGISTERS H04 读输入寄存器

5、WRITE\_SINGLE\_COIL H05 写单一线圈

6、WRITE\_SINGLE\_REGISTER H06 写单一寄存器

7、WRITE\_MULTIPLE\_COILS H15 写多个线圈

8、WRITE\_MULTIPLE\_REGISTERS H16 写多寄存器

'''

# 指令格式：机号 功能代码 起始地址 结束地址

data = master.execute(1, cst.HOLDING\_REGISTERS,0, 8)

if \_\_name\_\_ == "\_\_main\_\_":

main()

BVPP\_Server.py:

from fastapi import FastAPI

from fastapi\_amis\_admin.admin.settings import Settings

from fastapi\_user\_auth.site import AuthAdminSite

from datetime import datetime

from fastapi\_scheduler import SchedulerAdmin

#在fastapi异步框架中，选择AsyncOScheduler调度程序，默认使用sqlite持久化定时任务，不至于重启就失效

from apscheduler.schedulers.asyncio import AsyncIOScheduler

from apscheduler.jobstores.sqlalchemy import SQLAlchemyJobStore

from sqlmodel import SQLModel

from tools.get\_data\_from\_api import Get\_datas

from tools.Put\_MQTT import Pub\_Server

import time

import json

import struct

import random

import sys

import os

import numpy as np

BASE\_DIR = os.path.dirname(os.path.abspath(\_\_file\_\_))

rootPath = os.path.split(BASE\_DIR)[0]

sys.path.append(rootPath)

from Config.settings import config, config0

from Config.template import template\_send,byte\_types

root\_path\_config = config.get('root\_path\_config')

topic\_set = config0.get('topic')

# 创建FastAPI应用

app = FastAPI()

# 创建AdminSite实例

site = AuthAdminSite(settings=Settings(database\_url\_async='sqlite+aiosqlite:///admisadmin.db'))

# site = AuthAdminSite(settings=Settings(database\_url\_async='mysql+aiomysql://root:root@localhost:3306/amisadmin'))

auth =site.auth

Scheduler = AsyncIOScheduler(jobstore={'default':SQLAlchemyJobStore(url='sqlite:///jobs.sqlite')})

# 创建定时任务调度器`SchedulerAdmin`实例

scheduler = SchedulerAdmin.bind(site)

#定时任务方式：1、date，固定时间执行一次 2、interval，固定间隔时间循环执行3、cron，crontab表达式，最为灵活

with open(root\_path\_config + '/' + 'Branch\_code1.json') as f:

data = json.load(f)

#寄存器数据

\_Meterstate = [[0]\*19]

def change\_format(parm,arr\_):

"""将数据按模板整理成十六进制数组"""

datain = dict(zip(parm, arr\_))

length = 0

for i in template\_send:

length += byte\_types[template\_send[i]['type']]['length']

arr = []

type = '>'

for i in range(len(template\_send)):

type += template\_send[i]['type']

if template\_send[i]['name'] in datain:

# print('template\_send',template\_send[i]['name'])

x = int(datain[template\_send[i]['name']] \* template\_send[i]['mtp'])

if byte\_types[template\_send[i]['type']]['minvalue'] <= x <= byte\_types[template\_send[i]['type']]['maxvalue']:

arr.append(x)

else:

arr.append(byte\_types[template\_send[i]['type']]['default'])

else:

arr.append(byte\_types[template\_send[i]['type']]['default'])

dataout = struct.pack(type, \*arr)

return dataout

def loadJsonData():

# 加载json

with open(r"D:\Data\_upload\_tool\_Kangtai\names1.json", encoding='utf-8') as data:

# content = data.read()

# jsonData = json.loads(content).get('data')['meters']

jsonData = json.load(data)

# .get('data')['meters']

print(jsonData)

dict\_ = jsonData.get('meters')

print('Meters', dict\_)

nested\_list=[]

for i in range(len(dict\_)):

# print(f"第{i}块表", dict\_[i])

dict\_s = dict\_[i]

# 固定参数为缺省值

# key = dict\_s.get('meterId')

# id = data.get(f'{key}')

if dict\_s.get('cabinetId') in [1643188726637314050,1643189909485568003,1643189909485568012,1643189909485568021,1643189909485568029,

1643189909485568038,1643189909485568047,1643189909485568053,1643179275125571586]:

arrs\_ = [1,int(dict\_s.get('powerTotalYg')), int(dict\_s.get('powerTotalWg')),

int(dict\_s.get('powerTotalSz')), int(dict\_s.get('powerYgA')), \

int(dict\_s.get('powerYgB')), int(dict\_s.get('powerYgC')), int(dict\_s.get('powerWgA')),

int(dict\_s.get('powerWgB')), int(dict\_s.get('powerWgC')), \

int(dict\_s.get('energyZxyg')), int(dict\_s.get('voltageA')), int(dict\_s.get('voltageB')),

int(dict\_s.get('voltageC')), int(dict\_s.get('currentA')), \

int(dict\_s.get('currentB')), int(dict\_s.get('currentC')), int(dict\_s.get('frequencyA')),

int(dict\_s.get('powerFactorTotal'))]

# print(arrs\_)

nested\_list.append(arrs\_)

#多台变压器数据合并计算

# 多维数组

w3 = np.array(nested\_list)

#掩码

temp = np.ma.masked\_array(w3, mask=w3 == 0)

V = temp.mean(axis=0, dtype=int)

Value = temp[:, 1].sum(axis=0, dtype=int)

V[1] = Value

RES = list(map(int, V.tolist()))

#

bytes = change\_format(parm=['powerTotalYg', 'powerTotalWg', 'powerTotalSz', 'powerYgA', 'powerYgB', 'powerYgC', 'powerWgA', 'powerWgB',

'powerWgC', 'energyZxyg', 'voltageA', 'voltageB', 'voltageC', 'currentA', 'currentB', 'currentC',

'frequencyA', 'powerFactorTotal'],arr\_=RES)

\_Meterstate[0]=RES

return bytes

def get\_data\_from\_api():

try:

start = datetime.datetime.now()

# 转换为int类型的10位时间戳

timestam = int(time.time())

time\_new = str(timestam)

params = {"apiKey": '1555312928870420483',

"timestamp": time\_new}

res = Get\_datas().get\_data(params)

if res:

dict\_ = res.get('meters')

print('Meters',dict\_)

finally:

table\_name = 'meter'

# 根据序列号判断是否执行续传检查

end = datetime.datetime.now()

s = (end - start).total\_seconds()

print('数据导出: %s, 耗时: %s 秒' % (table\_name, s))

# logger.info('数据导出: %s, 耗时: %s 秒' % (table\_name, s))

#定时上传数据到MQTT服务器【间隔1min】

# @scheduler.scheduled\_job('interval', minutes=1, start\_date='2023-06-15 21:00:00', end\_date='2023-12-31 23:00:00')

# @scheduler.scheduled\_job('interval', seconds=10, start\_date='2023-06-15 21:00:00', end\_date='2023-12-31 23:00:00')

# def interval\_task\_test1():

# print('interval task1 is run...')

# arrs\_ = \_Meterstate[0]

# msg\_pub = to\_bytes(arrs\_)

# topic = "U001/PUB"

# Pub\_object = Pub\_Server()

# Pub\_object.mqtt\_publish(topic, msg\_pub)

# print("=======0", \_Meterstate)

# print('时间', datetime.now().strftime('%Y-%m-%d %H:%M:%S'))

#定时获取数据库数据【间隔5min】

@scheduler.scheduled\_job('interval', minutes=1, start\_date='2023-06-15 21:00:00', end\_date='2023-12-31 23:00:00')

def cron\_task\_test1():

print('cron task1 is run...')

print('时间', datetime.now().strftime('%Y-%m-%d %H:%M:%S'))

msg\_pub = loadJsonData()

print(msg\_pub)

# topic = "U001/PUB"

topic = topic\_set

Pub\_object = Pub\_Server()

Pub\_object.mqtt\_publish(topic, msg\_pub)

print("=======1",\_Meterstate)

#每10s从寄存器取一次数据

@scheduler.scheduled\_job('cron', hour='0-23',second='10,20,30,40,50')

def cron\_task\_test2():

print('cron task2 is run...')

print('时间', datetime.now().strftime('%Y-%m-%d %H:%M:%S'))

arrs\_ = \_Meterstate[0]

msg\_pub = change\_format(parm=['powerTotalYg', 'powerTotalWg', 'powerTotalSz', 'powerYgA', 'powerYgB', 'powerYgC', 'powerWgA', 'powerWgB',

'powerWgC', 'energyZxyg', 'voltageA', 'voltageB', 'voltageC', 'currentA', 'currentB', 'currentC',

'frequencyA', 'powerFactorTotal'],arr\_=arrs\_)

topic = topic\_set

# topic = "U001/PUB"

Pub\_object = Pub\_Server()

Pub\_object.mqtt\_publish(topic, msg\_pub)

print("=======2",\_Meterstate)

# #定时获取数据库数据【间隔5min】

# @scheduler.scheduled\_job('cron', hour='0-23',minute='0,15,30,45')

# def cron\_task\_test1():

# print('cron task1 is run...')

# msg\_pub = loadJsonData()

# print(msg\_pub)

# topic = "U001/PUB"

# Pub\_object = Pub\_Server()

# Pub\_object.mqtt\_publish(topic, msg\_pub)

# print("=======1",\_Meterstate)

# print('时间', datetime.now().strftime('%Y-%m-%d %H:%M:%S'))

# 挂载后台管理系统

site.mount\_app(app)

# 创建初始化数据库表

@app.on\_event("startup")

async def startup():

await site.db.async\_run\_sync(SQLModel.metadata.create\_all,is\_session=False)

await auth.create\_role\_user(role\_key='admin')

@app.on\_event("startup")

async def startup():

# 启动定时任务调度器

scheduler.start()

Scheduler.start()

if \_\_name\_\_ == '\_\_main\_\_':

# import uvicorn

# uvicorn.run(app, debug=True)

msg\_pub = loadJsonData()

print(msg\_pub)

# topic = "U001/PUB"

# Pub\_object = Pub\_Server()

# Pub\_object.mqtt\_publish(topic, msg\_pub)

# print('时间', datetime.now().strftime('%Y-%m-%d %H:%M:%S'))

#

# print(\_Meterstate)

# uvicorn BVPP\_Server:app --host=0.0.0.0 --port=9999

# supervisord -c E:\Build\_monitor\_AI\supervisord.conf

其中tools 文件夹下有以下文件：

1. **init**.py
2. AES\_MODE.py
3. Breakpoint\_up.py
4. Create\_xml.py
5. get\_conn.py
6. get\_data\_from\_api.py
7. logging\_conf.py
8. Modbus\_tcp\_con.py
9. Put\_MQTT.py
10. Reminder\_service.py
11. subscription\_model.py
12. webservice\_client.py

这个项目包含以下几个主要的文件夹和文件：

* **Config**：配置文件夹，可能包含各种配置文件。
* **Database**：数据库文件夹，用于存储数据库相关文件。
* **dos**：DOS 相关文件夹，可能包含 DOS 命令脚本或工具。
* **log**：日志文件夹，用于存放程序运行时生成的日志文件。

在 tools 文件夹中：

* **upload**：上传相关的子文件夹。
  + **admisadmin.db**：一个名为 admisadmin.db 的数据库文件。
  + **BVPP\_Server.py**：一个 Python 脚本文件。
  + **M.py**：另一个 Python 脚本文件。
  + **Modbus\_test.py**：一个测试 Modbus 协议的 Python 脚本。
  + **Push\_Server.py**：一个推送服务器相关的 Python 脚本。
* **start.bat** 和 **start\_bat.bat**：两个批处理文件（.bat），通常用于启动某些服务或执行一系列命令。