**《Python自动化运维》课程**

**期末学习报告**

班级：

姓名：\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

学号：\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. **任务背景与需求分析（宋体，加粗，小三号字）**

1．任务背景（宋体，四号字）

企业中存在很多重要的数据和文档，这些数据经常在日常工作中使用，因此需要定期备份和存档。手动备份和存档存在人为疏漏的风险，而且需要消耗很多人工时间。为了提高效率并确保数据安全性，需要为企业专门开发了一套定时备份工具，通过Python脚本自动化进行备份并记录信息。

2. 需求分析

**需要实现一个自动化备份任务，需求如下：**

**1，设计脚本：设计一个Python脚本来执行备份操作。在脚本中，您需要定义要备份的文件夹路径和备份文件夹路径。**

**2，定时备份：使用Python的os和shutil库，您可以编写代码来复制源文件夹中的文件和文件夹到目标备份文件夹中。可以使用操作系统的任务调度工具，例如crontab（适用于Linux系统）或Windows任务计划程序，来定时运行脚本。这样，可以设置脚本在特定的时间间隔内自动备份文件夹。**

**3，压缩备份文件：为了节省磁盘空间，可以使用Python的zipfile库将备份文件夹中的文件和文件夹压缩成一个单独的档案文件。这样可以减少备份文件的大小，并方便存储和传输。**

**4，更新备份日志：在备份完成后，您可以将相关信息记录到备份日志中，以便后续检查和管理。您可以使用Python的日志库来实现日志记录功能，例如logging模块。**

**5，自动发送备份文件：如果您希望自动将备份文件发送到其他位置，例如通过电子邮件或FTP协议，您可以使用Python的相应模块来实现。例如，您可以使用Python的smtplib模块来发送电子邮件**

1. **设计方案**

1,程序结构：

主程序：创建一个主程序，作为整个备份流程的入口点。该程序将负责调度和协调其他模块的工作。

模块划分：根据功能将程序划分为不同的模块，例如文件操作模块、备份处理模块、日志记录模块等，以提高代码的可维护性和可扩展性。

2,业务处理流程：

源文件夹扫描：通过文件操作模块，扫描源文件夹中的文件和子文件夹。可以使用递归的方式遍历源文件夹。

文件备份：备份处理模块将源文件夹中的文件复制到备份文件夹中。可以使用Python的shutil库来执行文件复制操作。

压缩备份文件：使用备份处理模块，将备份文件夹中的文件和文件夹压缩为一个单独的档案文件。可以使用Python的zipfile库来实现压缩功能。

日志记录：日志记录模块负责记录备份的相关信息，包括备份日期、时间、源文件夹路径、备份文件夹路径等。可以使用Python的logging模块来实现日志记录功能。

3,数据存储：

源文件夹信息：可以将源文件夹的路径和其他相关信息存储在配置文件或数据库中。这样可以方便地对源文件夹进行管理和配置。

备份文件存储：备份的压缩文件可以存储在本地磁盘中的特定目录下。可以定义一个文件命名规则，以便更好地组织和管理备份文件。

4,数据分析：

备份历史记录：将备份的相关信息记录在数据库中，以便进行数据分析。可以存储备份的日期、时间、文件大小、备份状态等信息。

1. **实现细节**

1,关键代码

1,使用python 爬取天气预报数据

需要使用f12控制台查找到需要获取的数据信息并且使用requests库提取查出来。

代码如下：

1. 扫描文件夹和子文件夹

file\_list = []  
for root, dirs, files in os.walk(source\_folder):  
 for file in files:  
 file\_path = os.path.join(root, file)  
 file\_list.append(file\_path)  
print("正在执行扫描文件以及子文件......")  
return file\_list

1. 备份文件到指定文件夹

def backup\_files(source\_folder, backup\_folder):  
 *"""  
 备份文件到指定文件夹  
 """* file\_list = scan\_folder(source\_folder) # 获取文件列表  
 for file\_path in file\_list:  
 target\_path = file\_path.replace(source\_folder, backup\_folder) # 构建目标文件路径  
 os.makedirs(os.path.dirname(target\_path), exist\_ok=True) # 创建目标文件夹（包括父文件夹）  
 shutil.copy2(file\_path, target\_path) # 复制文件到目标路径  
 print("正在执行备份文件夹到指定文件夹中......")

4，压缩备份文件夹

def compress\_backup\_folder(backup\_folder, backup\_zip):  
 *"""  
 压缩备份文件夹为档案文件  
 """* with zipfile.ZipFile(backup\_zip, 'w', zipfile.ZIP\_DEFLATED, allowZip64=True) as zipf:  
 for root, dirs, files in os.walk(backup\_folder):  
 if root == backup\_zip:  
 continue  
 for file in files:  
 file\_path = os.path.join(root, file)  
 zipf.write(file\_path, os.path.relpath(file\_path, backup\_folder))  
 print("压缩备份文件")

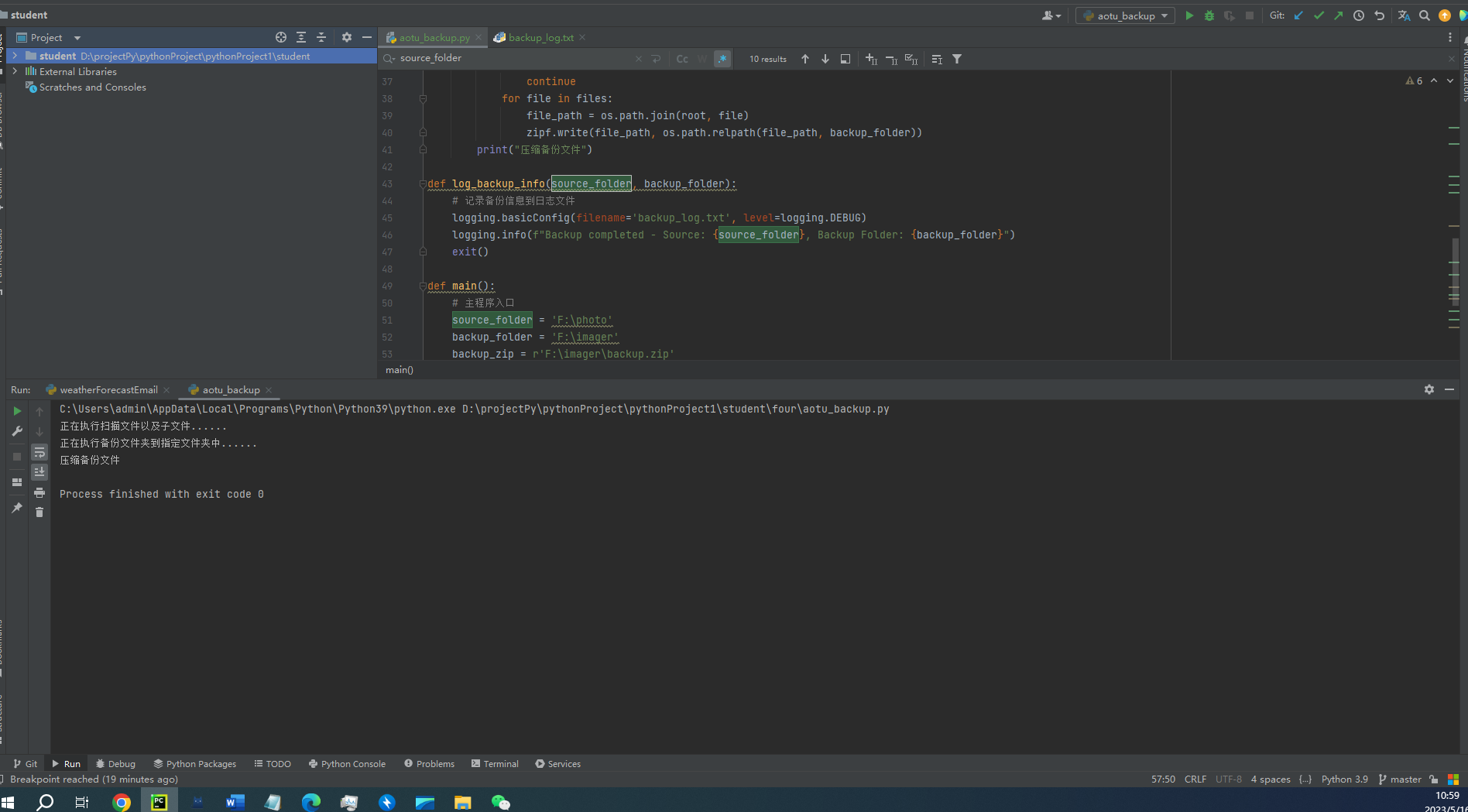
5，记录备份信息到日志文件

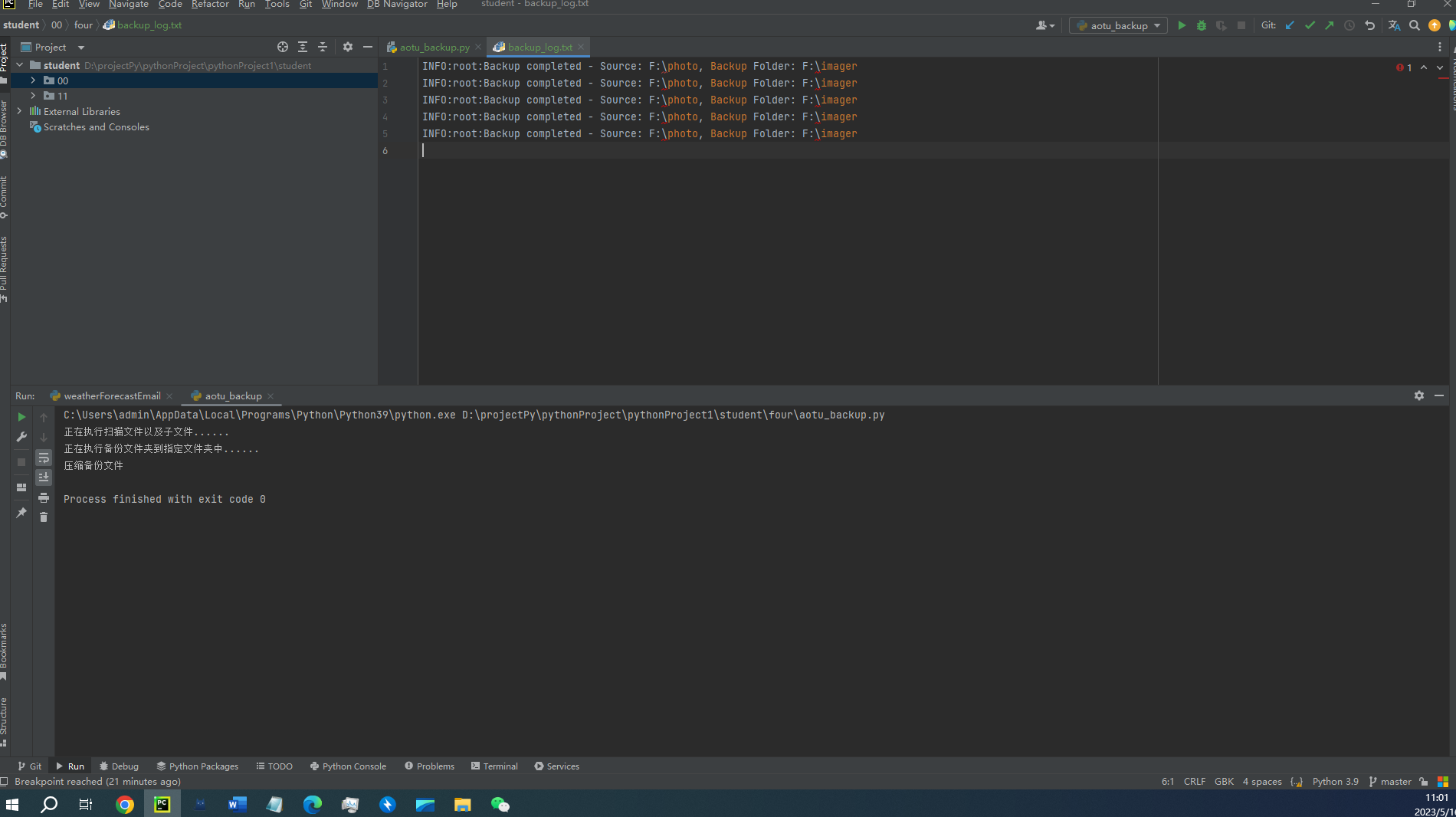
def log\_backup\_info(source\_folder, backup\_folder):  
 *"""  
 记录备份信息到日志文件  
 """* logging.basicConfig(filename='backup\_log.txt', level=logging.DEBUG)  
 logging.info(f"Backup completed - Source: {source\_folder}, Backup Folder: {backup\_folder}")  
 exit()

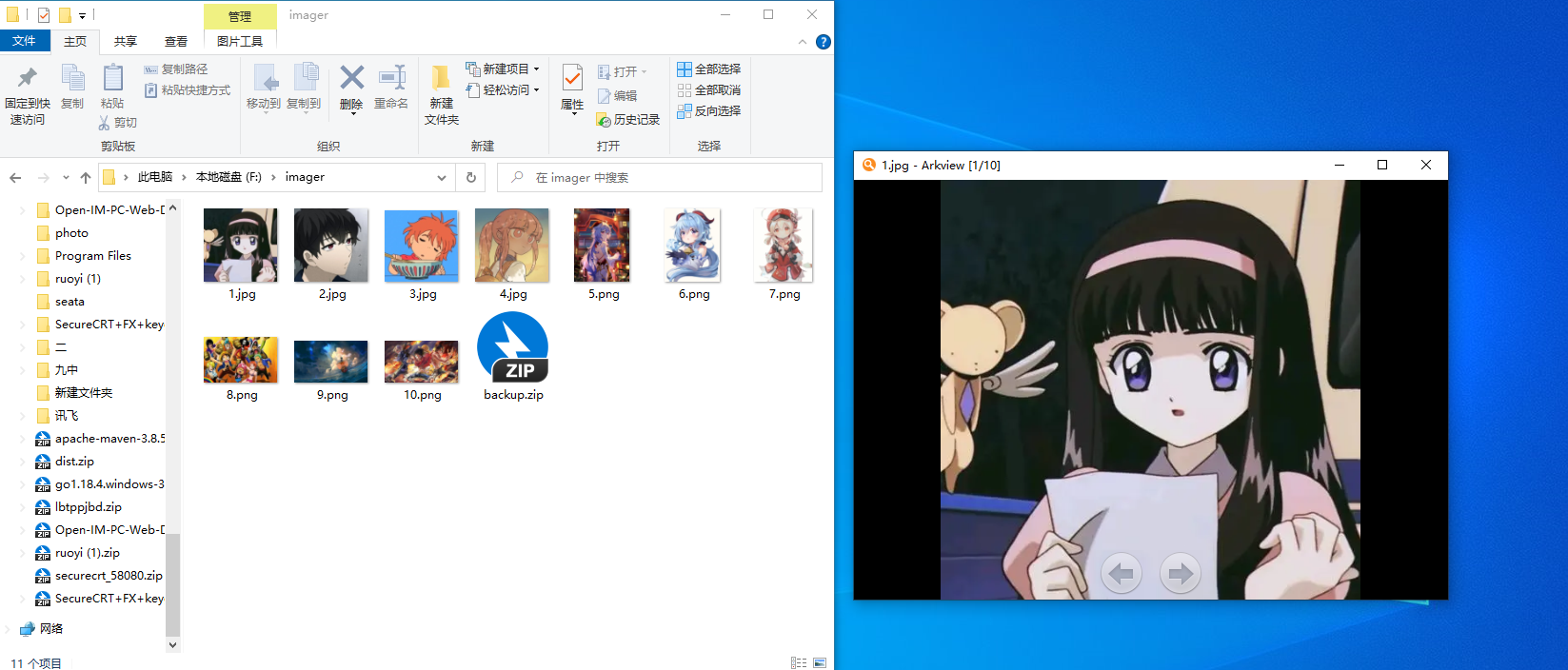
6，发送邮件 关键代码

# 构建邮件  
msg = MIMEMultipart()  
msg['Subject'] = '相册备份文件' # 邮件主题  
msg['From'] = sender\_email # 发件人  
msg['To'] = recipient\_email # 收件人  
  
# 邮件正文  
text = MIMEText('新备份的照片，请查收', 'plain')  
msg.attach(text)  
  
# 附件  
with open(r'F:\imager\backup.zip', 'rb') as f:  
 attachment = MIMEApplication(f.read())  
 attachment.add\_header('Content-Disposition', 'attachment', filename='backup.zip')  
 msg.attach(attachment)  
  
# 发送邮件  
with smtplib.SMTP('smtp.163.com', 25) as smtp:  
 smtp.login(sender\_email, password)  
 smtp.sendmail(sender\_email, recipient\_email, msg.as\_string())

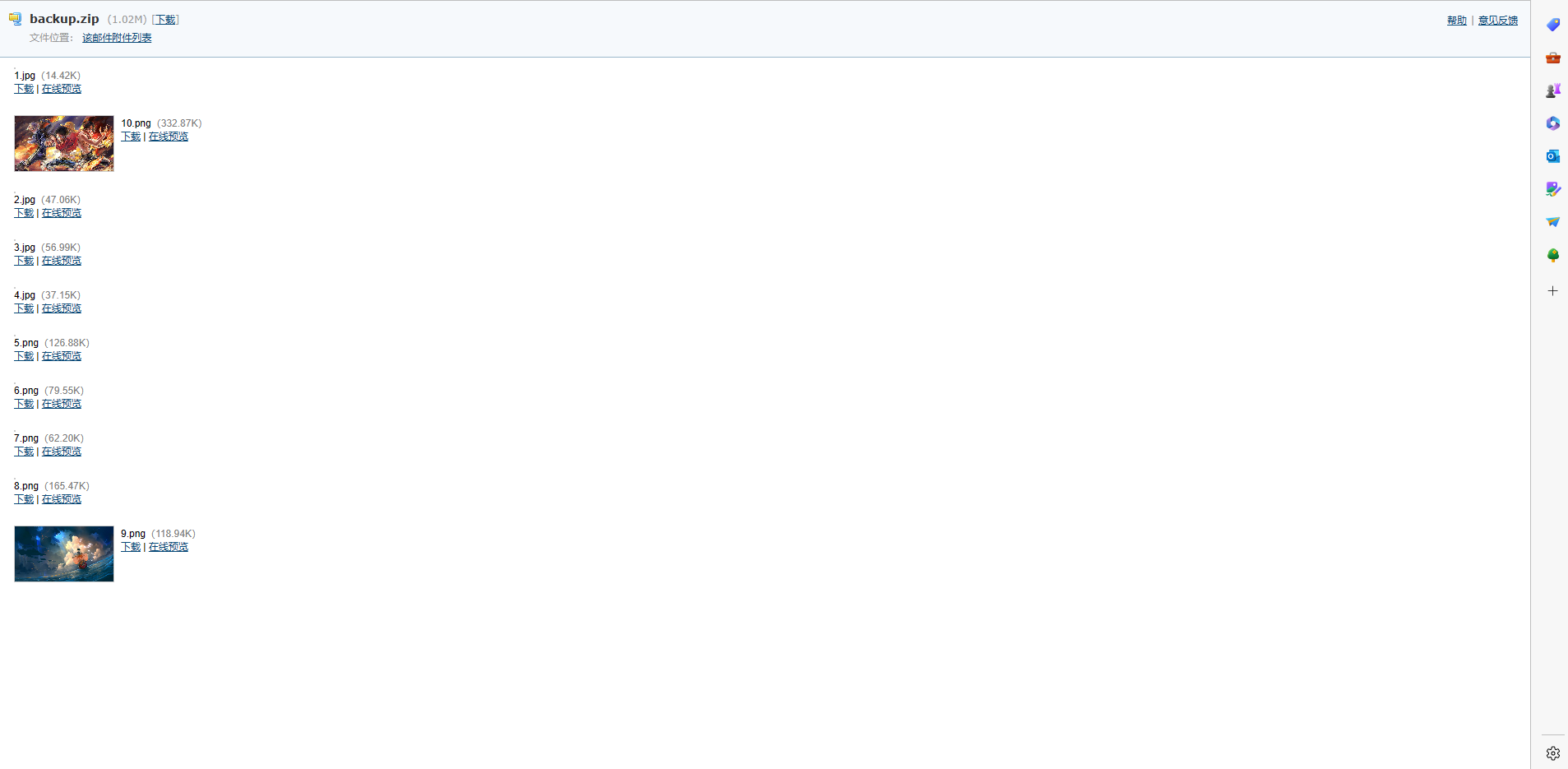
1. 运行截图

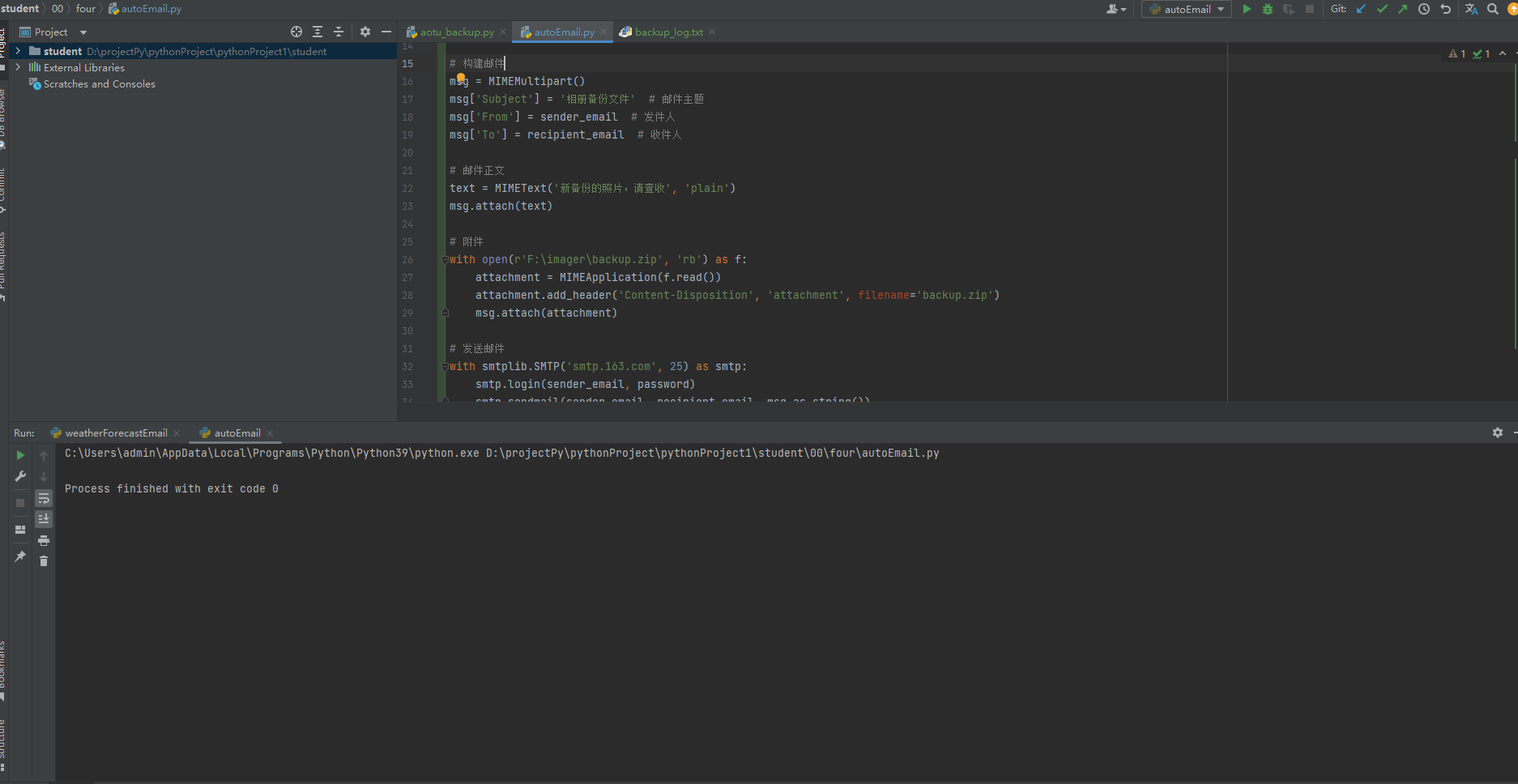
扫描文件以及子文件，进行备份，压缩

执行备份程序后生成日志

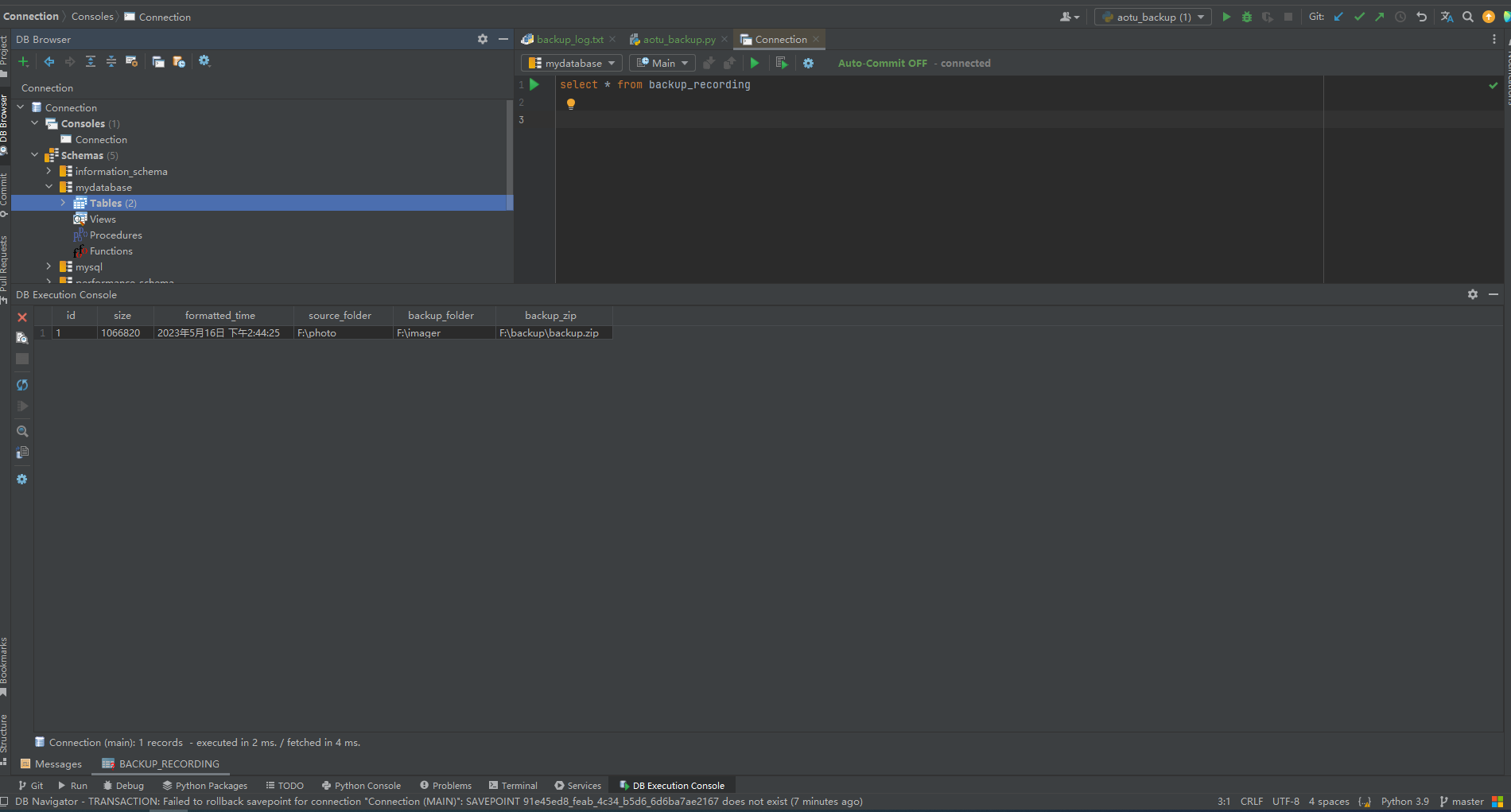
查看文件是否备份成功 目标文件截图

自动发送备份文件到邮箱截图

 执行邮件发送截图



备份信息存储数据库截图



1. **结果分析**

1，扫描源文件夹中的文件和子文件夹：

程序会递归地遍历源文件夹，并获取所有文件的路径。

2，备份文件到指定文件夹：

程序会将源文件夹中的文件备份到指定的备份文件夹中。

对于每个文件，程序会创建相应的目标路径，并使shutil.copy2函数进行文件复制。

3，压缩备份文件夹为档案文件：

程序会使用zipfile.ZipFile创建一个ZIP档案文件，并将 备份文件夹中的所有文件压缩到其中。输出"压缩备份文件"。

4，记录备份信息到日志文件：

程序会使用logging模块将备份完成的信息记录到名为"backup\_log.txt"的日志文件中。

日志中记录了备份的源文件夹路径和备份文件夹路径。

此外，程序还使用了schedule模块来调度任务，在每天的22:00执行main函数。

5,最后再通过SMTP服务器发送邮件。

总结：该程序会在指定时间定时执行备份操作，首先扫描源文件夹中的文件和子文件夹，然后将文件备份到指定的备份文件夹，接着将备份文件夹压缩为一个ZIP档案文件，最后将备份完成的信息记录到日志文件中和数据库中。

1. **系统涵盖的主要知识点及完整源代码**

系统涵盖的主要知识点

1，文件和文件夹操作：

使用os.walk函数递归遍历文件夹中的文件和子文件夹。

使用shutil.copy2函数进行文件复制。

2，ZIP压缩：

使用zipfile.ZipFile创建ZIP档案文件，并使用ZIP\_DEFLATED参数指定压缩算法。

使用zipf.write将文件添加到ZIP档案文件中。

使用allowZip64=True允许压缩大于4GB的文件。

3，日志记录：

使用logging.basicConfig进行基本的日志配置，将日志信息写入文件。

使用logging.info记录备份完成的信息到日志文件。

4，定时任务调度：

使用schedule模块实现定时任务调度，定时执行备份操作。

5，异常处理：

处理可能出现的错误情况。

6，字符编码和路径处理：

在程序开头指定了文件的字符编码为UTF-8。

使用os.path.join和os.path.dirname处理文件路径。

7，程序入口点：

使用if \_\_name\_\_ == "\_\_main\_\_":来确保主程序在直接运行脚本时执行，而不是作为模块被导入时执行。

8,数据库存储

使用pymysql 连接数据库 获取游标 创建sql语句 将备份信息存储到数据库中

总结：该程序涉及了文件和文件夹操作、ZIP压缩、日志记录、定时任务调度，数据库等多个主要知识点。这些知识点在实现自动化备份任务时非常有用

1. 源代码

Auto\_backup.py

#!/usr/bin/env python  
# -\*- coding: utf-8 -\*-  
  
import os  
import shutil  
import zipfile  
import logging  
from datetime import datetime  
  
import pymysql  
import schedule  
  
def scan\_folder(source\_folder):  
 *"""  
 扫描源文件夹中的文件和子文件夹，并返回文件列表  
 """* file\_list = []  
 for root, dirs, files in os.walk(source\_folder):  
 for file in files:  
 file\_path = os.path.join(root, file)  
 file\_list.append(file\_path)  
 print("正在执行扫描文件以及子文件......")  
 return file\_list  
  
  
def backup\_files(source\_folder, backup\_folder):  
 *"""  
 备份文件到指定文件夹  
 """* file\_list = scan\_folder(source\_folder) # 获取文件列表  
 for file\_path in file\_list:  
 target\_path = file\_path.replace(source\_folder, backup\_folder) # 构建目标文件路径  
 os.makedirs(os.path.dirname(target\_path), exist\_ok=True) # 创建目标文件夹（包括父文件夹）  
 shutil.copy2(file\_path, target\_path) # 复制文件到目标路径  
 print("正在执行备份文件夹到指定文件夹中......")  
  
  
def compress\_backup\_folder(backup\_folder, backup\_zip):  
 *"""  
 压缩备份文件夹为档案文件  
 """* with zipfile.ZipFile(backup\_zip, 'w', zipfile.ZIP\_DEFLATED, allowZip64=True) as zipf:  
 for root, dirs, files in os.walk(backup\_folder):  
 if root == backup\_zip:  
 continue  
 for file in files:  
 file\_path = os.path.join(root, file)  
 zipf.write(file\_path, os.path.relpath(file\_path, backup\_folder))  
 print("压缩备份文件")  
  
  
def get\_folder\_size(folder\_path):  
 total\_size = 0  
 for dirpath, dirnames, filenames in os.walk(folder\_path):  
 for f in filenames:  
 fp = os.path.join(dirpath, f)  
 total\_size += os.path.getsize(fp)  
 return total\_size  
  
  
def log\_backup\_info(source\_folder, backup\_folder,backup\_zip):  
 *"""  
 数据备份信息导入到数据库  
 """* # 连接MySQL数据库，需要提供主机名（host）、用户名（user）、密码（password）、数据库名称（database）和字符集（charset  
 mydb = pymysql.connect(  
 host="localhost",  
 user="root",  
 password="root123",  
 database="mydatabase",  
 charset='utf8'  
 )# 获取数据库操作游标  
 mycursor = mydb.cursor()  
  
 folder\_path = r"F:\backup"  
 size = get\_folder\_size(folder\_path)  
 now = datetime.now()  
 formatted\_time = now.strftime("%Y-%m-%d %H:%M:%S")  
  
 # 插入邮件正文到MySQL中 定义要执行的SQL语句，将邮件正文和发件人地址插入到名为emails的表中的body和address字段中  
 sql = "INSERT INTO backup\_recording(size,formatted\_time,source\_folder,backup\_folder,backup\_zip) VALUES (%s,%s,%s,%s,%s)"  
 # 定义要插入的数据，即文件大小,源文件地址，目标文件地址，压缩文件地址，备份时间等信息，使用元组（tuple）来存储  
 val = (size,formatted\_time,source\_folder,backup\_folder,backup\_zip)  
 # 执行SQL语句，将数据插入到数据库中  
 mycursor.execute(sql, val)  
 # 提交事务，将数据插入到数据库中  
 mydb.commit()  
 # 打印插入的记录数  
 print(mycursor.rowcount, "record inserted.")  
 """  
 记录备份信息到日志文件  
 """  
 logging.basicConfig(filename='backup\_log.txt', level=logging.DEBUG)  
 logging.info(f"文件大小为：{size}kb,备份时间为：{formatted\_time},源文件地址为: {source\_folder}, 备份文件地址为: {backup\_folder},压缩文件地址为：{backup\_zip}")  
 print(f"文件大小为：{size}kb,备份时间为：{formatted\_time},源文件地址为: {source\_folder}, 备份文件地址为: {backup\_folder},压缩文件地址为：{backup\_zip}")  
 exit()  
  
  
def main():  
 # 主程序入口  
 source\_folder = 'F:\photo' # 源文件夹路径  
 backup\_folder = 'F:\imager' # 备份文件夹路径  
 backup\_zip = r'F:\backup\backup.zip' # 备份档案文件路径  
 try:  
 backup\_files(source\_folder, backup\_folder) # 备份文件  
 compress\_backup\_folder(backup\_folder, backup\_zip) # 压缩备份文件夹  
 log\_backup\_info(source\_folder, backup\_folder,backup\_zip) # 记录备份信息  
  
 except Exception as e:  
 logging.error(f"An error occurred during backup: {str(e)}")  
 exit()

AutoEmail.py

#!/usr/bin/env python  
# -\*- coding: utf-8 -\*-  
import smtplib  
from email.mime.text import MIMEText  
from email.mime.multipart import MIMEMultipart  
from email.mime.application import MIMEApplication  
from aotu\_backup import main  
# 发送方邮箱地址和密码  
sender\_email = "学习@163.com"  
password = "IPNXYLLGGCSBPTKH"  
  
# 收件方邮箱地址  
recipient\_email = "学习@163.com"  
  
# 构建邮件  
msg = MIMEMultipart()  
msg['Subject'] = '相册备份文件' # 邮件主题  
msg['From'] = sender\_email # 发件人  
msg['To'] = recipient\_email # 收件人  
  
# 执行备份操作  
main()  
  
# 邮件正文  
text = MIMEText('新备份的照片，请查收', 'plain')  
msg.attach(text)  
# 附件  
with open(r'F:\backup\backup.zip', 'rb') as f:  
 attachment = MIMEApplication(f.read())  
 attachment.add\_header('Content-Disposition', 'attachment', filename='backup.zip')  
 msg.attach(attachment)  
  
# 发送邮件  
with smtplib.SMTP('smtp.163.com', 25) as smtp:  
 smtp.login(sender\_email, password)  
 smtp.sendmail(sender\_email, recipient\_email, msg.as\_string())  
 print("已发送完成")  
  
# schedule.every().day.at("22:00").do(main) # 定时执行主程序