



量子文件舱管理系统

📋 目录

- [梳理题目要求](#)
- [思路](#)
- [学习笔记](#)
- [流程记录](#)
- [运行成功截图](#)

📋 梳理题目要求

- 1 文件分类：** 将给定文件夹 `incoming_data` 里的所有文件按照 **给定的规则** 进行分类并覆盖原文件夹内容
- 2 生成日志：** 将分类好的文件夹 `incoming_data` 里的文件目录按照 **规定的格式** 保存到 `hologram_log.txt`
- 3 文件名添加时间戳：** 对文件进行重命名，在前缀加上精确到毫秒的时间
- 4 增加日志格式：** 增加 **JSON 格式** 的日志格式

📋 思路

- 1 创建目标文件夹** ([如何通过python实现创建？](#))
- 2 遍历 `incoming_data` 文件夹** ([如何遍历一个文件夹？](#))，通过判断语句对 **文件后缀名** 进行分类并移动到目标文件夹，注意未知文件需要加上前缀 **ENCRYPTED_** ([如何单独提取出来文件的后缀名？](#)、[如何增加文件前缀？](#)、[如何移动文件？](#))
- 3 创建 `hologram_log.txt` 并写入 `incoming_data` 的目录结构** ([如何使用python对文件写入数据？](#)、[用于美化的 Unicode 如何处理？](#))

4 为文件在其前缀加上 **时间戳**([如何获取时间戳?](#))

5 增加 **JSON 格式** 的日志格式 ([什么是 JSON 格式的日志格式?](#)、[如何使用python进行实现?](#))

相关知识学习

1. 通过 python 创建新的文件夹

在Python中创建新文件夹并添加文件 [1](#) [2](#) [3](#)

在Python中，可以使用多种方法来创建新文件夹并在其中添加文件。以下是一些常用的方法：

使用os模块

os模块提供了创建文件夹和文件的基本功能。以下是一个示例代码，演示了如何使用os模块创建新文件夹并在其中创建文件：

```
import os

# 创建新文件夹
path = "new_folder"
os.makedirs(path)




# 在新文件夹中创建文件
file_path = os.path.join(path, "new_file.txt")
with open(file_path, "w") as file:
    file.write("Hello World!")
```

这段代码将创建一个名为“new_folder”的文件夹，并在其中创建一个名为“new_file.txt”的文件，并写入“Hello World!” [1](#)。

代码实现：

```
import os

#在 incoming_data 子目录下创建 quantum_core 文件夹
os.mkdir("incoming_data/quantum_core")
```

创建成功!   

考虑到要创建大量的新文件夹，可以将所有创建新文件夹的工作集中在一个函数中实现，保证代码模块化

```
4 def mkdir(): 1个用法
5     os.mkdir("incoming_data/quantum_core")
6     os.mkdir("incoming_data/quantum_core/SECTOR-7G")
7
8     os.mkdir("incoming_data/exobiology_lab")
9     os.mkdir("incoming_data/exobiology_lab/POD-09X")
10
11     os.mkdir("incoming_data/hologram_vault")
12     os.mkdir("incoming_data/hologram_vault/CHAMBER-12F")
13
14     os.mkdir("incoming_data/temporal_archive")
15     os.mkdir("incoming_data/temporal_archive/VAULT-00T")
16
17     os.mkdir("incoming_data/quantum_quarantine")
```

2. 通过 python 遍历文件夹中的文件

Python 遍历目录下文件 [1](#) [2](#) [3](#)

在 Python 中，有几种常用的方法可以遍历目录下的文件和文件夹。以下是三种主要的方法：

`os.walk()`、`os.listdir()` 和 `os.scandir()`。

使用 `os.walk()`

`os.walk()` 是 Python 中最常用的遍历文件夹的方法。它会递归地遍历目录中的所有子文件夹，并返回一个三元组 `(root, dirs, files)`，其中 `root` 表示当前遍历到的文件夹路径，`dirs` 表示当前文件夹下的子文件夹，`files` 表示当前文件夹下的所有文件。

```
import os

def traversal_folder(folder_path):
    for root, dirs, files in os.walk(folder_path):
        for file_name in files:
            file_path = os.path.join(root, file_name)
            print(file_path)

# 调用函数
folder_path = r"D:\data"
traversal_folder(folder_path)
```

优点： `os.walk()` 能够自动遍历文件夹下的子文件夹，无需手动递归 [1](#)。**缺点：** 对于大型文件夹，`os.walk()` 会占用大量内存，因为它需要把所有的文件都存储在内存中 [2](#)。




代码实现：

```
import os

folder = "incoming_data"

for root, dirs, files in os.walk(folder):
    # root: 当前遍历的目录路径
    # dirs: 当前目录下的子目录名列表（字符串列表）
    # files: 当前目录下的文件名列表（字符串列表）
    # os.walk() 会从 folder 开始，逐层深入所有子目录，直到遍历完整个目录树。实质上是递归遍历

    # file_name 即文件名
    for file_name in files:
        # 获得文件路径 file_path
        file_path = os.path.join(root, file_name)
```

实现遍历文件目录   

3. 通过python提取文件的扩展名

获取文件扩展名

要获取文件扩展名，可以使用 `os.path.splitext()` 函数。这个函数接受一个文件路径作为参数，并返回一个元组，其中第一个元素是文件的基本名称（不包括扩展名），第二个元素是文件的扩展名（包括点号）。例如：

```
import os

file_path = "/home/user/documents/example.txt"
file_name, file_extension = os.path.splitext(file_path)
print("文件名:", file_name) # 输出: 文件名: /home/user/documents/example
print("扩展名:", file_extension) # 输出: 扩展名: .txt
```

如果你想要去掉点号，可以通过字符串切片操作来实现：


```
file_extension = file_extension[1:]
print("去掉点号的扩展名:", file_extension) # 输出: 去掉点号的扩展名: txt
```

那么就可以在遍历文件的基础上，依次检查文件的扩展名进行分类

```
import os

folder = 'incoming_data'

for root, dirs, files in os.walk(folder):
    for file in files:
        file_path = os.path.join(root, file_name)
        # 获得文件扩展名 file_extension
        file_name, file_extension = os.path.splitext(file_path)
        # 进行下一步的判断以及分类
```

成功获得文件信息   

4. 通过python增加文件前缀

使用os.path模块

`os.path` 模块提供了一些处理文件路径的函数。结合 `os` 模块，可以实现更复杂的文件名修改操作，例如添加前缀或后缀 [2](#)：

```
import os

# 指定目录
directory = '/path/to/directory'
prefix = 'new_'
suffix = '_backup'

# 遍历目录中的所有文件
for filename in os.listdir(directory):
    # 构造新的文件名
    new_filename = prefix + filename + suffix
    # 重命名文件
    os.rename(os.path.join(directory, filename), os.path.join(directory,
new_filename))
```

```
import os
```

```
folder = 'incoming_data'
```

```
def rename(filename):
    # 实现对未知文件增加前缀 ENCRYPTED_ 的操作
    prefix = 'ENCRYPTED_'
    new_file_name = prefix + file_name
    os.rename(os.path.join(folder, file_name), os.path.join(folder, new_file_name))
```

实现增加前缀操作   

5. 使用python移动文件

在 Python 中移动文件 [1](#) [2](#) [3](#)

在 Python 中，移动文件是一个常见的操作，可以通过多种方法实现。以下是几种常用的方法：

使用 `shutil.move()` 函数

`shutil` 模块提供了高级的文件操作功能，包括复制和移动文件。要使用 `shutil.move()` 函数移动文件，只需指定源文件路径和目标文件路径。例如：

```
import shutil

# 定义源文件路径和目标文件路径
file_source = "C:/myweb/chapter01/hello.txt"
file_destination = "C:/myweb/chapter02/hello_02.txt"

# 移动文件
shutil.move(file_source, file_destination)
```

在这个例子中，`shutil.move()` 函数将 `hello.txt` 文件从 `chapter01` 目录移动到 `chapter02` 目录，并将其重命名为 `hello_02.txt` [1](#) [2](#)。

由此可以实现对已经分好类的文件进行移动到目标文件夹的操作 [✓](#) [✓](#) [✓](#)

6. 使用python对文件写入数据

基础语法介绍

在Python中，写入文件主要通过内置 [函数](#) `open()` 来完成。该函数可以以不同的模式打开一个文件，其中最常用的两种模式为只写模式 `'w'` 和追加模式 `'a'`。

- `'w'`：如果文件已存在，则覆盖原有内容；若不存在，则创建新文件。
- `'a'`：无论文件是否存在，都将在文件末尾添加内容。

一旦文件对象被成功打开，就可以调用 `write()` 或 `writelines()` 方法来进行写操作了。

示例代码：

```
1 with open('example.txt', 'w') as file:
2     file.write('Hello, World!\n')
3     file.write('这是第二行内容。')
4
5 print("写入完成！")
```

上述代码展示了如何使用 `with` 语句安全地打开一个文件，并向其中写入两行文本信息。注意，在使用完文件后，即使没有显式关闭，`with` 语句也会自动帮你完成这一步骤。

代码如下：

```

with open('hologram_log.txt', 'w') as file:
    file.write(' ')\n')
    file.write(' | 🚀 Xia-III 空间站数据分布全息图 | \n')
    file.write(' ')\n')
with open('hologram_log.txt', 'a') as file:
    file.write('这是一行增加的测试文本\n')

```

7. 使用python输出Unicode进行美化

- **Python:** 你可以使用 `ord()` 和 `chr()` 函数来在 Unicode 编码和字符之间进行转换。

python

复制

```

# 将字符转换为 Unicode 编码
print(hex(ord("😊")) # 输出: 0x1f600

# 将 Unicode 编码转换为字符
print(chr(0x1f600)) # 输出: 😊

```

```

print(hex(ord("🚀"))) # 输出: 0x1f680
print(chr(0x1f680)) # 输出: 🚀

```

但是 同时会发现，直接复制目标字符进行输出也可以实现 `print("🚀")`

The screenshot shows a code editor with two tabs: `main.py` and `test.py`. The `test.py` tab is active and contains the following code:

```

1 print(hex(ord("🚀"))) # 输出: 0x1f680
2
3 print(chr(0x1f680)) # 输出: 🚀
4 print("🚀")

```

Below the code editor, there is a terminal window titled "运行" (Run) with a Python icon and a close button. The terminal shows the command `C:\Users\zhang\miniconda3\python.exe D:\lianshidaiRecurit\test.py` and the output:

```

0x1f680
🚀
🚀

```

At the bottom of the terminal, it says "进程已结束, 退出代码为 0" (Process ended, exit code 0).

8. JSON 格式

知乎

首发于
IT行业

...

写文章

什么是json



IT小仙女

五年工作经验，专为设计而生

+ 关注她

58 人赞同了该文章

一、什么是json?

json的全称为：JavaScript Object Notation，是一种轻量级的数据交互格式。它基于ECMAScript (欧洲计算机协会⁺制定的js规范)的一个子集，采用完全独立于编程语言的文本格式来存储和表示数据。【以上来自于百度百科】

简单来说：json就是一种在各个编程语言中流通的数据格式，负责不同编程语言中的数据传递和交互。

类似于：

- 国际通用语言-**英语**
- 中国56个民族不同地区的通用语言-**普通话**。

但真要说json到底是什么，以及json的作用，我们总是难以描述，下面我将从各个方面来进行说明：

JSON 是一种简单、通用、可读性强的数据格式，广泛用于数据传输和存储。

JSON 由键值对和数组组成：

1. 对象（用大括号 {} 表示）

JSON 对象包含一个或多个 键值对，键必须是字符串，值可以是字符串、数字、布尔值、对象、数组或 null。

```
{
  "name": "小明",
  "age": 25,
  "isStudent": false
}
```

上面这个 JSON 代表一个人的信息，包括姓名、年龄和是否是学生。

2. 数组（用方括号 [] 表示）

JSON 数组可以存放多个值，比如一组对象：

```
{
  "students": [
    { "name": "小明", "age": 25 },
    { "name": "小红", "age": 23 }
  ]
}
```

这个 JSON 代表一组学生，每个学生都是一个对象。

9. 通过python写入 JSON 格式的日志

python中将数据转化为json格式并以文件的形式存储

原创 半瓶醋的历史 于 2020-06-08 16:45:40 发布 阅读量3k 收藏 13 点赞数 4

版权

```
1 import json
2
3 names = {
4     'name' : 'tianwnajie',
5     'age' : 100,
6 }
7
8 filename='tianwanjie.json'
9 with open(filename,'w',encoding='utf8') as file_obj:
10     #json.dump(names,file_obj,sort_keys=True, indent=4, ensure_ascii=False)
11     """参数1:需要转化的数据
12     参数2:文件对象
13     参数3:是否按字母排序
14     参数4:json内容左边的空格数
15     参数5:想输出真正的中文需要指定ensure_ascii=False, 如果为True则会输出这种类型: "name": "\u7530\u665a\u6770",
16     """
17     json.dump(names, file_obj, sort_keys=False, indent=8, ensure_ascii=True)
18     #json.dump(names, file_obj)
```

实现了将python的数据结构转化为 JSON 格式的文件,代码如下:

```
with open("incoming_data/hologram_log.json", 'w', encoding='utf-8') as json_file:
    json.dump(data, json_file, indent=4, ensure_ascii=False)
```

10. 通过python获取时间戳

datetime.datetime.now().strftime

原创

essenge

于 2020-06-10 20:57:41 发布


阅读量3.2w

收藏 138


点赞数 44

版权

分类专栏: [Jupyter Notebook](#) 文章标签: [pytorch](#)

 GitCode 开源社区 文章已被社区收录

加入社区

 Jupyter Notebook 专栏收录该内容

3 订阅 13 篇文章 订阅专栏

datetime.datetime.now().strftime('%Y-%m-%d-%H_%M_%S')

1.python datetime模块用strftime 格式化时间

```
1 import datetime
2 datetime.datetime.now()
```

这个会返回 microsecond。因此这个是我们不需要的。所以得做一下修改

```
1 datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S")
```

格式化之后，就得到了我们常见的格式了。

附：strftime参数

 **essenge** [关注](#)

44

138

0

分享

...

专栏目录

datetime.now() 获取当前日期时间等信息

datetime.now().strftime() 里的常见常用参数及其对应的意义如下：

参数	含义
%Y	年份数
%B	月份的英文全拼
%d	日数
%A	星期的英文全拼
%x	年月日
%X	时分秒
%c	年月日+时分秒
%H	24小时制当前小时

参数	含义
%l	12小时制当前小时

流程记录




为了模块化代码，应当把整个任务分为 **创建文件夹、文件名增加前缀、文件分类、日志记录** 四个子任务，由于互相之间彼此独立，因而决定用若干函数进行模块化编程

1 创建文件夹

`makedir()` 函数实现对于文件夹以及日志文件的创建

创建的copy文件夹为临时文件夹，用于将原incoming_data文件夹里的文件进行分类，然后删除 **原** incoming_data，最后将 copy 文件夹 **更名为** incoming_data

这成功避免了由于一边删除文件夹文件，一边遍历文件夹文件所产生的bug

`makedir()` 函数如下   

```
def makedir():
    """
    函数实现对于文件夹以及日志文件的创建
    """
    os.mkdir('copy')
    os.mkdir('copy/quantum_core/')
    os.mkdir('copy/quantum_core/SECTOR-7G/')

    os.mkdir('copy/hologram_vault/')
    os.mkdir('copy/hologram_vault/CHAMBER-12F/')

    os.mkdir('copy/exobiology_lab/')
    os.mkdir('copy/exobiology_lab/POD-09X/')




    os.mkdir('copy/temporal_archive/')
    os.mkdir('copy/temporal_archive/VAULT-00T/')

    os.mkdir('copy/quantum_quarantine/')

    open('copy/hologram_log.txt', 'w').close()
    open('copy/hologram_log.json', 'w').close()
```

2 文件名增加前缀

`rename(name, prefix)` 函数实现了接收 `name`, `prefix` 两个参数, 返回更名后文件所在位置, 方便下一步的移动操作

`rename(name, prefix)` 函数如下   

```
def rename(name, prefix):  
    """  
    :param name: 需要增加前缀的文件名  
    :param prefix: 需要增加的前缀名  
    :return: 更名之后文件所在的位置  
    """  
    folder = "incoming_data"  
    new_name = prefix + name  
    os.rename(os.path.join(folder, name), os.path.join(folder, new_name))  
    return os.path.join(folder, new_name)
```

3 文件分类

`classify()` 函数实现了对 `incoming_data` 文件夹下的文件按照后缀名进行分类, 并保存在 `copy` 文件夹的正确目录下, 最后将 `incoming_data` 文件夹删除并更名 `copy` 文件夹为新的 `incoming_data`

`classify()` 函数如下   

```

def classify():
    """
    函数实现对incoming_data文件夹内的文件按照后缀名进行分类，
    分类结束之后的文件保存在copy文件夹正确的目录下，
    最后删除原incoming_data文件夹，将copy文件夹更名为incoming_data，
    从而实现目标
    """

    folder = "incoming_data"
    # 通过os.walk()进行对incoming_data的遍历
    for root, dirs, files in os.walk(folder):
        for file in files:
            # 获得incoming_data 文件夹里每一个文件的路径(file_path)、文件名(file_name)、后缀名(file_extension)
            file_path = os.path.join(folder, file)
            file_name, file_extension = os.path.splitext(file_path)

            # 获得当前时间信息（时 + 秒 + 微秒），以便后续的增加时间前缀
            dt = datetime.now()
            time_str = dt.strftime('%H%M%S%f') + '_'

            # 对文件后缀名进行分类，调用rename（）函数进行增加前缀操作，接着将其移动到copy文件夹正确位置
            if file_extension == '.quantum':
                new = rename(file, time_str)
                shutil.move(new, 'copy/quantum_core/SECTOR-7G/')
            elif file_extension == '.holo':
                new = rename(file, time_str)
                shutil.move(new, 'copy/hologram_vault/CHAMBER-12F/')
            elif file_extension == '.exo':
                new = rename(file, time_str)
                shutil.move(new, 'copy/exobiology_lab/POD-09X/')
            elif file_extension == '.chrono':
                new = rename(file, time_str)
                shutil.move(new, 'copy/temporal_archive/VAULT-00T/')
            else:
                # 未知文件需再增加一截前缀
                new = rename(file, time_str + 'ENCRYPTED_')
                shutil.move(new, 'copy/quantum_quarantine/')

    # 删除原 incoming_data 文件夹(此时应为空)，将已经分类好的文件夹 copy 更名为 incoming_data
    os.removedirs('incoming_data')
    os.rename('copy', 'incoming_data')

```

4 日志记录

为保存文件路径信息，日志记录函数需要 **递归查找** 文件夹目录

为保证代码的模块性和可读性，我将 **日志记录** 子任务分成

`write(start_path, file, prefix="", data=None)` 和 `save()` 两个函数进行实现

其中，

`save()` 实现 `hologram_log.txt` 文件中 **非递归写入数据** 和 `hologram_log.json` 文件的信息写入

`write(start_path, file, prefix="", data=None)` 实现 `hologram_log.txt` 文件的 **递归写入数据** 以及 **目录信息的保存**

`write()` 函数通过扫描当前文件夹下的全部文件/文件夹并记录下来，对于文件直接写入数据即可，对于文件夹则进行下一步的递归，直到查找完整个目标文件夹。

同时，为了保存文件夹文件信息，需要一个返回数据信息来保存，使用字典来保存目录结构，生成 **JSON** 格式的数据，其 **键** 保存文件夹的名称，**值** 保存另一个字典（表示子文件夹）或 `file` 键（保存文件名列表）

`write()` 函数如下 

```

def write(start_path, file, prefix="", data=None):
    """
    递归遍历目录结构，并写入日志文件，同时生成 JSON 结构化数据。
    :param start_path: 要遍历的目录路径。
    :param file: 目标日志文件对象。
    :param prefix: 用于格式化输出的前缀（默认值为空）。
    :param data: 存储目录结构的 JSON 数据（默认为 None，函数内部初始化）。
    :return: 包含目录结构的 JSON 数据。
    """
    # 如果 data 为空，则初始化为字典
    if data is None:
        data = {}

    # 获取 start_path 目录下的所有文件和文件夹
    items = os.listdir(start_path)

    # 遍历当前目录中的所有文件和文件夹
    for item in items:
        path = os.path.join(start_path, item) # 构造完整的路径

        # 在日志文件中写入当前项的前缀
        file.write(prefix + "├─ ")

        # 处理特殊文件 hologram_log.txt，直接写入名称
        if item == "hologram_log.txt":
            file.write(item + '\n')

        # 处理文件夹
        elif os.path.isdir(path):
            file.write('🚀') # 目录前加上火箭符号表示
            data[item] = {} # 在 JSON 结构中创建该目录的字典
            file.write(item + '\n')
            new_prefix = prefix + "|   " # 生成新的前缀，用于子项缩进
            write(path, file, new_prefix, data[item]) # 递归处理子目录

        # 处理文件
        elif os.path.isfile(path):
            # 判断文件类型，并在日志中加上对应的图标
            if item.endswith(".quantum") or item.endswith(".holo") or item.endswith(".exo") or :
                file.write('💡')
            else:
                file.write('⚠️')

```



```

file.write(item + '\n')

if "files" not in data: # 如果 data 字典中没有 "files" 这个键
    data["files"] = [] # 初始化 "files" 为一个空列表
data.setdefault("files", []).append(item)

return data # 返回 JSON 结构化数据

```

save() 函数由于承担着写入非递归数据的任务，因而结构较为简单，只需要使用文件写入语句，同时调用 write() 函数写入递归数据即可

save() 函数如下   

```

def save():
    """
    函数实现 hologram_log.txt 和 hologram_log.json 两个日志文件的数据写入
    """

    # 写入 hologram_log.txt
    with open("incoming_data/hologram_log.txt", "w", encoding="utf-8") as f:
        f.write("┌───────────────────────────────────┐\n")
        f.write("│ 🚀 Xia-III 空间站数据分布全息图 │\n")
        f.write("└───────────────────────────────────┘\n\n")
        f.write("└─🚀 incoming_data" + '\n')

    # 此处调用 write () 函数，获得返回值 data 储存了文件夹中数据的目录信息
    data = {"incoming_data" : write("incoming_data", f, "│ ")}

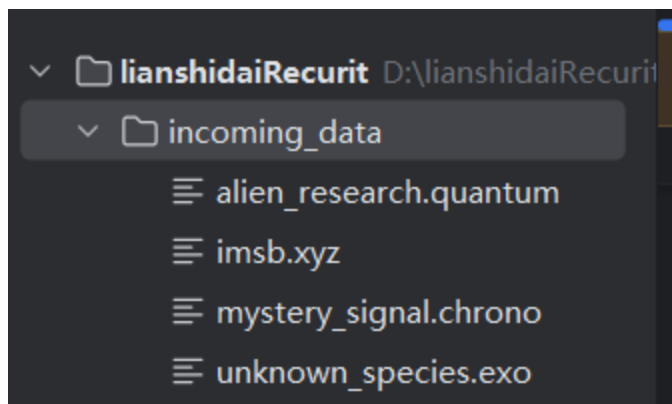
    f.write('\n\n')
    f.write('🕒 SuperNova · 地球标准时 2142-10-25T12:03:47\n')
    f.write('⚠️ 警告：请勿直视量子文件核心')

    # 写入 hologram_log.json
    with open("incoming_data/hologram_log.json", 'w', encoding='utf-8') as json_file:
        json.dump(data, json_file, indent=4, ensure_ascii=False)

```

运行成功截图

运行程序之前 incoming_data   

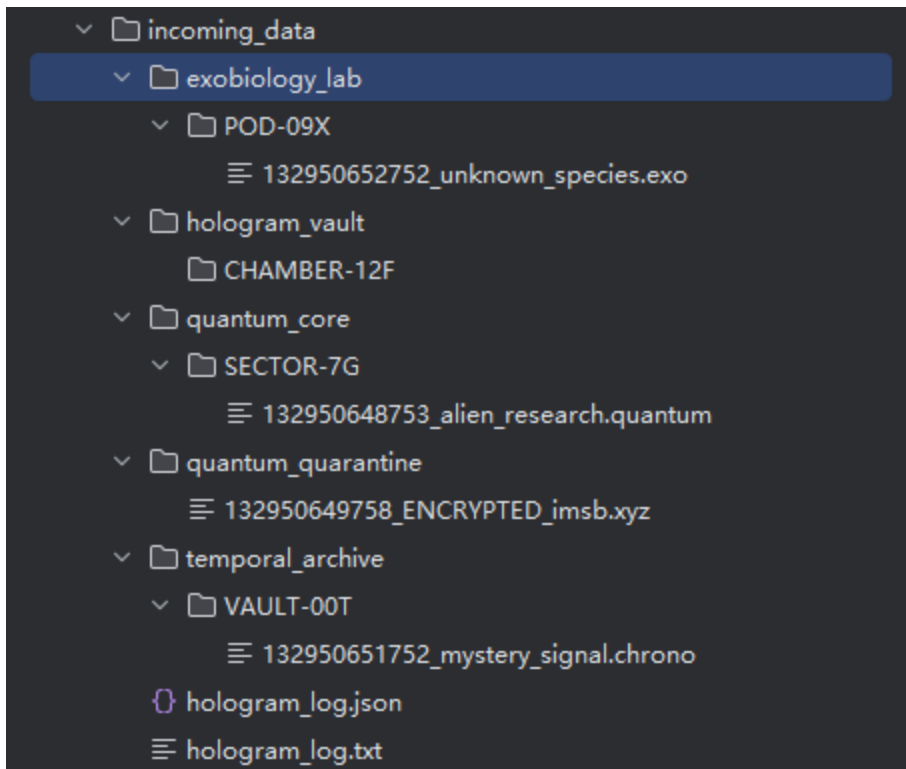


运行程序：

```
C:\Users\zhang\miniconda3\python.exe D:\lianshidaiRecurit\main.py
🚀 量子文件舱管理系统开始工作 🚀 🚀 🚀
-----
已创建好目标舱室 🏆 🏆 🏆
-----
已将外星数据文件归档分类 🌟 🌟 🌟
-----
已保存3D全息日志到'hologram_log.txt 和 hologram_log.json'，请查看 📄 📄 📄
-----
已完成量子数据流处理 🚀 🚀 🚀

进程已结束，退出代码为 0
```

运行程序后的 incoming_data [↓](#) [↓](#) [↓](#)



hologram_log.json 文件   

```
main.py  hologram_log.txt  hologram_log.json ×
1  {
2      "incoming_data": {
3          "exobiology_lab": {
4              "POD-09X": {
5                  "files": [
6                      "132950652752_unknown_species.exo"
7                  ]
8              }
9          },
10         "files": [
11             "hologram_log.json",
12             "hologram_log.txt"
13         ],
14         "hologram_vault": {
15             "CHAMBER-12F": {}
16         },
17         "quantum_core": {
18             "SECTOR-76": {
19                 "files": [
20                     "132950648753_alien_research.quantum"
21                 ]
22             }
23         },
24         "quantum_quarantine": {
25             "files": [
26                 "132950649758_ENCRYPTED_imsb.xyz"
27             ]
28         },
29         "temporal_archive": {
30             "VAULT-00T": {
31                 "files": [
32                     "132950651752_mystery_signal.chrono"
33                 ]
34             }
35         }
36     }
37 }
```

hologram_log.txt 文件 [↓](#) [↓](#) [↓](#)

```
main.py  hologram_log.txt  hologram_log.json
1
2  Xia-III 空间站数据分布全息图
3
4
5  incoming_data
6    exobiology_lab
7      POD-09X
8        132950652752_unknown_species.exo
9    hologram_log.json
10   hologram_log.txt
11   hologram_vault
12     CHAMBER-12F
13   quantum_core
14     SECTOR-76
15       132950648753_alien_research.quantum
16   quantum_quarantine
17     132950649758_ENCRYPTED_imsb.xyz
18   temporal_archive
19     VAULT-00T
20       132950651752_mystery_signal.chrono
21
22
23  SuperNova · 地球标准时 2142-10-25T12:03:47
24  ⚠ 警告：请勿直视量子文件核心
```

顺利完成任务🚀🚀🚀