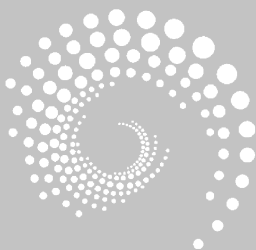


文件操作

CHAPTER 5



文件操作是程序与外部世界交互、实现数据持久化的核心桥梁。Python 提供了强大且标准化的工具集,用于高效处理各类文件(文本、二进制、CSV、JSON),涵盖从基础读写、格式解析到异常管理和资源释放的全流程。掌握 `open()` 函数的不同模式、`with` 语句的自动资源管理、`csv` 与 `json` 模块的专业化数据处理,以及 `try-except` 的健壮性保障,是构建可靠应用、实现数据存储交换与分析的必备能力。本章将系统阐述文件操作的原理、方法与最佳实践。

🔑 5.1 文件的读写

文件是存储在长期存储设备上的数据集合,可以包含文本、图像、音频、视频等多种类型的信息。文件根据其内容的组织形式主要分为文本文件和二进制文件两大类。文本文件由字符构成,可以使用文本编辑器阅读或编辑,如 `.txt`、`.py`、`.html` 等扩展名的文件;而二进制文件则不能用文本编辑器阅读或编辑,如 `.mp3`、`.mp4`、`.png` 等扩展名的文件,需要通过特定软件进行编辑。

5.1.1 文本文件的读取与写入

1. 文件的打开

`open()` 函数是 Python 中用于打开文件的内置函数,它返回一个文件对象,通过该对象可以进行文件的读取、写入或追加操作。`open()` 函数的基本语法如下。

```
open(file, mode='r', buffering=-1, encoding=None, errors=None, newline=None, closefd=True, opener=None)
```

file: 要打开的文件的路径。

mode: 打开文件的模式,默认为只读模式('r'),常见的文件打开模式如表 5.1 所示。

buffering: 设置文件的缓冲策略。

encoding: 指定文件的编码方式。

errors: 指定如何处理编码错误。

newline: 控制换行符的行为。

closefd: 如果为 True,则在关闭文件对象时关闭底层文件描述符。

opener: 自定义打开器。

表 5.1 文件打开模式

模式	说 明
'r'	只读模式(默认)。文件必须存在
'w'	写入模式。如果文件存在则清空文件,如果文件不存在则创建新文件
'a'	追加模式。如果文件存在则在文件末尾追加内容,如果文件不存在则创建新文件
'x'	独占创建模式。如果文件不存在则创建新文件,如果文件存在则引发 FileExistsError
'b'	二进制模式。与其他模式结合使用,如'rb','wb'等
't'	文本模式(默认)。与其他模式结合使用,如'rt','wt'等
'+'	读写模式。与其他模式结合使用,如'r+'、'w+'、'a+'等

2. 文件的读取

文件对象提供了多种读取方法。

read(size): 读取整个文件内容,如果指定 size 参数,则读取指定字节数的内容。

readline(): 逐行读取文件内容,每次读取一行。

readlines(): 读取所有行并返回一个列表,每行作为一个元素。

3. 文件的写入

文件对象提供了多种写入方法。

write(string): 将字符串写入文件。

writelines(lines): 将字符串列表写入文件,注意需要手动添加换行符。

4. 文件的关闭

使用 close()方法关闭文件,以释放操作系统资源。推荐使用 with 语句自动管理文件的打开和关闭,避免资源泄露。

5.1.2 二进制文件的处理

Python 提供强大的工具来处理二进制文件,这些文件存储的是字节序列,不包含格式化信息。与文本文件相比,二进制文件具有更紧凑的存储方式,无须进行十进制到二进制的转换,从而提高了 CPU 效率,并且允许随机跳转到特定位置读取数据。下面将详细介绍如何在 Python 中处理二进制文件。

1. 打开二进制文件

使用 open()函数以二进制模式打开文件。Python 中,打开二进制文件时需要指定模

式为“b”。常见的二进制文件模式如表 5.2 所示。

表 5.2 二进制文件模式

模式	说 明
‘rb’	以只读模式打开二进制文件
‘wb’	以写入模式打开二进制文件,如果文件存在则清空文件内容
‘ab’	以追加模式打开二进制文件,不会覆盖原有内容
‘r+b’	以读写模式打开二进制文件
‘w+b’	以读写模式打开二进制文件,如果文件存在则清空文件内容
‘a+b’	以读写模式打开二进制文件,不会覆盖原有内容

示例代码:

代码	<pre># 打开一个二进制文件进行读取 with open('example.bin', 'rb') as file: data = file.read() # 打开一个二进制文件进行写入 with open('example.bin', 'wb') as file: file.write(b'Hello, World!') # 打开一个二进制文件进行追加 with open('example.bin', 'ab') as file: file.write(b' Appended text.')</pre>
----	---

2. 读取二进制数据

使用 read()方法可以读取文件内容,返回的是字节形式的数据。可以指定读取的字节数,或不指定读取文件全部内容。

代码	<pre># 读取全部内容 with open('example.bin', 'rb') as file: data = file.read() # 读取指定字节数 with open('example.bin', 'rb') as file: data = file.read(1024) # 读取 1024B</pre>
----	---

3. 写入二进制数据

使用 write()方法可以将字节对象写入二进制文件。

代码	<pre># 写入字节对象 with open('example.bin', 'wb') as file: file.write(b'\x00\x01\x02\x03')</pre>
----	---

4. 解析字节数据

使用 struct.unpack()方法可以解析读取的字节数据,根据数据类型进行解析。

```

代码  import struct

# 解析字节数据
with open('example.bin', 'rb') as file:
    unpacked_data = struct.unpack('5i', file.read(20))
    print(unpacked_data)

```

5. 序列化和反序列化

使用 pickle 模块可以将 Python 对象序列化为字节流,并将其写入二进制文件。同样,可以将字节流反序列化为 Python 对象。

```

代码  import pickle

# 序列化对象
data = {'key': 'value'}
with open('data.pkl', 'wb') as file:
    pickle.dump(data, file)

# 反序列化对象
with open('data.pkl', 'rb') as file:
    data = pickle.load(file)
    print(data)

```

6. 错误处理

在处理文件时,错误处理是必不可少的。Python 提供了 try-except 语句来处理可能发生的异常。

```

代码  try:
    with open('example.bin', 'rb') as file:
        data = file.read()
except IOError as e:
    print(f"An IOError occurred: {e.strerror}")

```

7. 其他操作

二进制文件的其他操作类型如表 5.3 所示。

表 5.3 二进制文件的其他操作类型

操 作 类 型	方 法	描 述
移动指针	file.seek(offset)	移动文件指针到指定位置
获取指针位置	file.tell()	获取当前文件指针位置
保存数组	np.save(file, arr)	将 NumPy 数组保存到二进制文件
加载数组	np.load(file)	从二进制文件加载 NumPy 数组
读取到数组	data.fromfile(file, count)	将文件数据读取到数组中
写入数组	data.tofile(file)	将数组数据写入文件
直接读取	file.readinto(data)	将数据直接读取到数组中

5.1.3 with 语句的使用

with 语句是一种用于简化文件操作的高级语法,它能够自动管理文件的打开和关闭,确保文件资源被正确释放,即使在读写过程中发生异常也是如此。以下是关于 with 语句在文件读写中的详细说明。

1. 基本用法

with 语句的基本格式如下。

```
with open('文件名', '模式') as 文件对象:
    # 文件操作代码
```

其中,open()函数用于打开文件,模式参数指定文件的打开方式(如读取'r'、写入'w'等),文件对象是通过 as 关键字指定的变量,用于在 with 语句块中引用文件。

2. 自动管理文件的打开和关闭

使用 with 语句可以避免手动调用 close()方法,从而减少代码量并提高代码的可读性和安全性。例如:

代码	<pre>with open('example.txt', 'r') as file: content = file.read() print(content)</pre>
----	--

文件 example.txt 在 with 语句块结束后会自动关闭,无须显式调用 file.close()。

3. 异常处理

with 语句不仅简化了文件操作,还能自动处理异常。如果在 with 语句块中发生异常,文件也会被正确关闭。例如:

代码	<pre>try: with open('example.txt', 'r') as file: content = file.read() print(content) except FileNotFoundError: print("文件未找到")</pre>
----	--

如果文件 example.txt 不存在,程序会捕获 FileNotFoundError 异常并打印错误信息,同时文件会被自动关闭。

4. 处理多个文件

with 语句还可以同时处理多个文件,例如,读取一个文件并写入另一个文件。

代码	<pre>with open('input.txt', 'r') as infile, open('output.txt', 'w') as outfile: for line in infile: outfile.write(line)</pre>
----	---

input.txt 被读取,其内容被写入 output.txt 中,两个文件在 with 语句块结束后都会自动关闭。

5. 上下文管理器

with 语句的工作原理基于上下文管理器协议,即对象必须实现__enter__()和__exit__()方法。当进入 with 语句块时,调用对象的__enter__()方法获取资源;当离开 with 语句块时,调用对象的__exit__()方法释放资源。例如:

代码	<pre>class Sample: def __enter__(self): print("进入上下文") return self def __exit__(self, exc_type, exc_value, traceback): print("退出上下文") with Sample() as sample: print("在上下文中")</pre>
----	---

Sample 类实现了上下文管理器协议,当进入和离开 with 语句块时,会分别调用__enter__()和__exit__()方法。

6. 实际案例

以下是一个实际案例,展示了如何使用 with 语句将字典列表写入本地 txt 文件中,并以特定格式输出。

代码	<pre>a = [{'name': 'Alice', 'age': 25}, {'name': 'Bob', 'age': 30}] with open('a.txt', 'w+') as file: for item in a: file.write(f"Name: {item['name']}, Age: {item['age']}\n") with open('a.txt', 'r') as file: content = file.read() print(content)</pre>
输出	<pre>Name: Alice, Age: 25 Name: Bob, Age: 30</pre>

字典列表 a 被写入文件 a.txt 中,并以特定格式输出。然后,文件内容被读取并打印出来。

 5.2 处理 CSV 与 JSON 文件

CSV 与 JSON 作为数据交换的通用格式,在 Python 中拥有高效的处理工具。内置的 csv 模块提供了对 CSV 文件的逐行读写支持,而 json 模块则实现了 Python 对象与 JSON 格式的双向转换。结合 pandas 库,可进一步实现复杂数据的清洗、转换与批量处理。本节将系统介绍这些工具的使用方法,并通过电商订单数据处理案例展示从 CSV 到 JSON 的完整流程。

5.2.1 CSV 文件的读取与写入

1. 读取 CSV 文件

在 Python 中,可以使用内置的 csv 模块来读取 CSV 文件。以下是几种常见的读取方法。

(1) 使用 csv.reader 读取 CSV 文件。

代码	<pre>import csv with open('example.csv', 'r', newline='', encoding='utf-8') as csvfile: csvreader = csv.reader(csvfile) for row in csvreader: print(row) # 每一行数据作为一个列表输出</pre>
----	---

(2) 使用 csv.DictReader 读取带有表头的 CSV 文件。

代码	<pre>import csv with open('example_with_header.csv', 'r', newline='', encoding='utf-8') as csvfile: csvreader = csv.DictReader(csvfile) for row in csvreader: print(row) # 每一行数据作为一个字典输出,键为表头列名</pre>
----	--

(3) 使用 pandas 库读取 CSV 文件。

代码	<pre>import pandas as pd def load_data(): data = pd.read_csv("dataPajak.csv", encoding='iso-8859-1', parse_dates=True) return data db_tabel = load_data() db_tabel.head(5)</pre>
----	---

这个方法使用 pandas 库的 read_csv 函数来读取 CSV 文件,并返回一个 DataFrame 对象,方便后续的数据处理和分析。

2. 写入 CSV 文件

在 Python 中,同样可以使用 csv 模块来写入 CSV 文件。以下是几种常见的写入方法。

(1) 通过创建 writer 对象写入。

代码	<pre>import csv with open('output.csv', 'w', newline='', encoding='utf-8') as csvfile: csvwriter = csv.writer(csvfile) csvwriter.writerow(['Name', 'Age', 'City']) # 写入表头 csvwriter.writerow(['Alice', 30, 'New York']) # 写入一行数据 csvwriter.writerows([</pre>
----	---

	<pre> ['Bob', 25, 'Los Angeles'], ['Charlie', 35, 'Chicago']])</pre>	# 写入多行数据
--	--	----------

(2) 使用 DictWriter 写入。

代码	<pre>import csv with open('output_dict.csv', 'w', newline='', encoding='utf-8') as csvfile: fieldnames = ['Name', 'Age', 'City'] csvwriter = csv.DictWriter(csvfile, fieldnames=fieldnames) csvwriter.writeheader() # 写入表头 csvwriter.writerow({'Name': 'Alice', 'Age': 30, 'City': 'New York'}) csvwriter.writerows([{'Name': 'Bob', 'Age': 25, 'City': 'Los Angeles'}, {'Name': 'Charlie', 'Age': 35, 'City': 'Chicago'}])</pre>
----	---

(3) 使用 pandas 库写入 CSV 文件。

代码	<pre>import pandas as pd data = { 'Name': ['Alice', 'Bob', 'Charlie'], 'Age': [30, 25, 35], 'City': ['New York', 'Los Angeles', 'Chicago'] } df = pd.DataFrame(data) df.to_csv('output_pandas.csv', index=False)</pre>
----	--

这个方法使用 pandas 库的 DataFrame 对象,并通过 to_csv()方法将数据写入 CSV 文件。

注意事项:

(1) 编码问题:在读取和写入 CSV 文件时,需要指定正确的编码格式,如 utf-8 或 iso-8859-1,以避免乱码问题。

(2) 换行符问题:在写入 CSV 文件时,需要设置 newline=' '参数,以防止出现多余的空行。

(3) 文件模式:在打开文件时,需要指定正确的文件模式,如‘r’表示只读模式,‘w’表示写入模式,‘a’表示追加模式。

通过上述方法,可以有效地在 Python 中处理和操作 CSV 文件,实现数据的读取和写入。这些方法适用于不同的场景和需求,可以根据具体情况进行选择和调整。

5.2.2 JSON 格式的解析与生成

1. 基本概念

JSON(JavaScript Object Notation)是一种轻量级的数据交换格式,易于人阅读和编

写,同时也易于机器解析和生成。JSON 基于 JavaScript 编程语言,具有以下特点。

- (1) 键值对: JSON 对象由键值对组成,键为字符串,值可以是字符串、数字、布尔值、数组或另一个 JSON 对象。
- (2) 数组: 数组是有序的值列表,元素间用逗号分隔,值类型与键值对相同。
- (3) 数字类型: 支持十进制整数、小数、零和浮点数。
- (4) 布尔类型: 只有 true 和 false 两个值。

2. 解析方法

Python 提供了内置的 json 模块来处理 JSON 数据。主要的解析方法如下。

json.loads(): 将 JSON 字符串转换为 Python 数据结构。

json.load(): 从文件读取 JSON 数据并转换为 Python 数据结构。

3. 生成方法

Python 中生成 JSON 数据的方法如下。

json.dumps(): 将 Python 数据结构转换为 JSON 字符串。

json.dump(): 将 Python 数据结构写入文件并转换为 JSON 格式。

以下是一些具体的示例代码,展示了如何在 Python 中解析和生成 JSON 数据。

解析 JSON 字符串:

代码	<pre>import json #JSON 数据以字符串形式给出 json_str = '{"name": "张三", "age": 28, "city": "北京"}' #使用 json.loads()将 JSON 字符串解析为 Python 字典 data = json.loads(json_str) #输出解析后的数据 print(data) print(data['name'])</pre>
输出	<pre>{'name': '张三', 'age': 28, 'city': '北京'} 张三</pre>

将 Python 字典转换为 JSON 字符串:

代码	<pre>import json #创建一个 Python 字典对象 data = { "name": "小明", "age": 20, "city": "上海" } #使用 json.dumps()将 Python 字典转换为 JSON 字符串 json_str = json.dumps(data, ensure_ascii=False)</pre>
----	---

	<pre>#输出生成的 JSON 字符串 print(json_str)</pre>
输出	<pre>{"name": "小明", "age": 20, "city": "上海"}</pre>

从文件读取 JSON 数据：

代码	<pre>import json #假设有一个名为 data.json 的文件,内容如下 #{ "name": "李四", "age": 25, "city": "广州" } #使用 json.load() 从文件读取 JSON 数据并转换为 Python 字典 with open('data.json', 'r', encoding='utf-8') as file: data = json.load(file) #输出解析后的数据 print(data)</pre>
输出	<pre>{'name': '李四', 'age': 25, 'city': '广州'}</pre>

将 Python 字典写入 JSON 文件：

代码	<pre>import json #创建一个 Python 字典对象 data = { "name": "王五", "age": 30, "city": "深圳" } #使用 json.dump() 将 Python 字典写入文件并转换为 JSON 格式 with open('output.json', 'w', encoding='utf-8') as file: json.dump(data, file, ensure_ascii=False, indent=4)</pre>
----	--

4. 使用 pandas 库解析 JSON

pandas 库提供了更高级的解析功能,如按行或列解析 JSON 数据,将 DataFrame 转换为 JSON 格式,以及将 DataFrame 写入 JSON 文件。主要方法如下。

pandas.read_json(): 读取 JSON 文件。

DataFrame.to_json(): 将 DataFrame 转换为 JSON 格式。

假设有一个名为 data.json 的文件,内容如下: [{"name": "赵六", "age": 22, "city": "南京"}, {"name": "孙七", "age": 27, "city": "杭州"}]。

代码	<pre>import pandas as pd #使用 pandas.read_json() 读取 JSON 文件 df = pd.read_json('data.json') print(df)</pre>
----	--