Python 程序设计

第三次作业：简易客户端应用

2021211306 班 | 杜抒泽 2021211110

2023 年 11 月 17 日

目录

1 作业题目

基于 aiohttp（https://docs.aiohttp.org/en/stable/）实现一个服务查询客户端，能够访问 #2 作业提供的服务。数据获取后进行格式转换：

- JSON 结果转换为 TEXT 格式（字段之间使用空格间隔、记录之间使用换行符间隔）
- XML 结果转换为 TEXT 格式（需求同上）。
- CSV 格式转换为 TEXT 格式（需求同上）。

要求客户端可以通过以上 3 种格式访问数据服务。

2 作业内容

由于作业要求，代码实现为了单文件，没有以文件的形式分离各个模块。本次作业有部分类型定义与第二次作业中重复，且半数代码用在了画 GUI 上，因此**代码十分冗长，建议直接点击目录中第三节的超链接跳转代码说明开始阅读。**

程序源代码嵌入下方的 code block 中。

```
#! /usr/bin/env python3
# -*- coding: utf-8 -*-


import json
import os
from tkinter.scrolledtext import ScrolledText
from tkinter.filedialog import asksaveasfilename
from tkinter import ttk
from typing import Dict, List, Literal, Optional, Tuple, Union

import aiohttp
import asyncio
import pyperclip # type: ignore
import tkinter as tk
import xml.dom.minidom


def unify_crlf(s: str) -> str:
    return os.linesep.join([s for s in s.splitlines() if s.strip()])


class WeatherDataEntry:
    def __init__(
        self, year: int, temperature: float, temperature_smoothed: float
```

```
    ) -> None:
        self.year: int = year
        self.temperature: float = temperature
        self.temperature_smoothed: float = temperature_smoothed

    @classmethod
    def try_from(
        cls, year: str, temperature: str, temperature_smoothed: str
    ) -> 'WeatherDataEntry':
        try:
            return cls(
                int(year), float(temperature), float(temperature_smoothed)
            )
        except ValueError as e:
            raise Exception(f'Invalid weather data entry {e}')

    def __str__(self) -> str:
        return f'{self.year} {self.temperature} {self.temperature_smoothed}'


class WeatherData:
    class Params:
        def __init__(
            self, lower: int, upper: int,
            order: Optional[Literal['asc', 'desc']] = None,
        ) -> None:
            self.lower: int = lower
            self.upper: int = upper
            self.order: Optional[str] = order

        @classmethod
        def try_from(
            cls, lower: str, upper: str, order: str
        ) -> 'WeatherData.Params':
            try:
                lower_: int = int(lower)
                upper_: int = int(upper)
                order_: Optional[Literal['asc', 'desc']] = None
                if order in ('asc', 'desc'):
                    order_ = order # type: ignore
                elif len(order) != 0:
                    raise ValueError(f'Order cannot be {order}')
            except ValueError as e:
                raise Exception(f'Invalid params: {e}')
```

```
            return cls(lower_, upper_, order_)

        def to_dict(self) -> Dict[str, Union[int, str]]:
            return {
                'lower': self.lower, 'upper': self.upper, 'order': self.order
            } if self.order else {'lower': self.lower, 'upper': self.upper}

    def __init__(self, data: Optional[List[WeatherDataEntry]] = None) -> None:
        self.data: List[WeatherDataEntry] = data if data else []

    def append(self, entry: WeatherDataEntry) -> None:
        self.data.append(entry)

    def __str__(self) -> str:
        return 'year temperature temperature_smoothed\n' + \
            '\n'.join(str(entry) for entry in self.data)

    @classmethod
    def from_json(cls, json_obj: str) -> 'WeatherData':
        ret: 'WeatherData' = cls()
        for entry in json.loads(json_obj):
            ret.append(WeatherDataEntry(
                entry['year'],
                entry['temperature'],
                entry['temperature_smoothed'])
            )
        return ret

    @classmethod
    def from_csv(cls, csv_obj: str) -> 'WeatherData':
        ret: 'WeatherData' = cls()
        for entry in csv_obj.splitlines():
            year, temperature, temperature_smoothed = entry.split(',')
            if year == 'year':
                continue
            ret.append(WeatherDataEntry.try_from(
                year, temperature, temperature_smoothed)
            )
        return ret

    @classmethod
    def from_xml(cls, xml_obj: str) -> 'WeatherData':
        ret: 'WeatherData' = cls()
```

```
        for entry in xml.dom.minidom.parseString(xml_obj) \
                .getElementsByTagName('entry'):
            ret.append(WeatherDataEntry.try_from(
                # autopep8: off
                entry.getElementsByTagName('year')[0] \
                    .firstChild.data, # type: ignore
                entry.getElementsByTagName('temperature')[0] \
                    .firstChild.data, # type: ignore
                entry.getElementsByTagName('temperature_smoothed')[0] \
                    .firstChild.data, # type: ignore
                # autopep8: on
            ))
        return ret


class QueryType:
    VALUES = ['/json', '/csv', '/xml']

    def __init__(self, type: str) -> None:
        if type not in self.VALUES:
            raise Exception('Invalid query type')
        self.type = type


async def fetch_data(
    server: str, query_type: QueryType, params: WeatherData.Params
) -> Tuple[str, str]:
    url = server + query_type.type

    async with aiohttp.ClientSession() as session:
        async with session.get(url, params=params.to_dict()) as response:
            if response.status != 200:
                raise Exception(
                    'Failed to fetch data from '
                    f'{url} {response.status} {response.reason}'
                )

            raw: Optional[str] = None
            pretty: Optional[str] = None
            decoded: Optional[WeatherData] = None

            if query_type.type == '/json':
                raw = await response.text()
                pretty = json.dumps(json.loads(raw), indent=2)
```

```
                decoded = WeatherData.from_json(raw)
            elif query_type.type == '/csv':
                raw = await response.text()
                pretty = unify_crlf(raw)
                decoded = WeatherData.from_csv(raw)
            elif query_type.type == '/xml':
                raw = await response.text()
                pretty = unify_crlf(
                    xml.dom.minidom.parseString(raw).toprettyxml(indent='  ')
                )
                decoded = WeatherData.from_xml(raw)
            else:
                raise Exception(f'Failed to fetch data from {url}')

            return pretty, str(decoded)


class App(tk.Tk):
    def __init__(self) -> None:
        super().__init__()
        self.title('Simple Weather Data Client')
        self.create_widgets()

    def create_widgets(self) -> None:
        # 服务器地址输入框
        self.label_request_type = tk.Label(self, text='服务器: ')
        self.label_request_type.grid(
            row=0, column=1, padx=0, pady=10, sticky='E'
        )

        self.url_entry = tk.Entry(self, width=30)
        self.url_entry.insert(0, 'http://127.0.0.1:555')
        self.url_entry.grid(
            row=0, column=2, columnspan=4, padx=0, pady=10, sticky='EW'
        )

        # 查询按钮
        self.query_button = tk.Button(
            self, text='查询', width=5, command=self.perform_query
        )
        self.query_button.grid(row=0, column=7, padx=10, pady=0, sticky='E')

        # 下拉选择框
```

```
        self.label_request_type = tk.Label(self, text='查询类型: ')
        self.label_request_type.grid(
            row=1, column=0, padx=0, pady=0, sticky='E'
        )

        self.request_type_combobox = ttk.Combobox(
            self, width=7, state='readonly'
        )
        self.request_type_combobox['values'] = QueryType.VALUES
        self.request_type_combobox.current(0)
        self.request_type_combobox.grid(
            row=1, column=1, padx=0, pady=0, sticky='W'
        )

        # 排序方式
        self.label_order = tk.Label(self, text='排序方式: ')
        self.label_order.grid(row=1, column=2, padx=0, pady=0, sticky='E')

        self.order_combobox = ttk.Combobox(self, width=7, state='readonly')
        self.order_combobox['values'] = ['年份顺序', '温度升序', '温度降序']
        self.order_combobox.current(0)
        self.order_combobox.grid(row=1, column=3, padx=0, pady=0, sticky='W')

        # 起始年份
        self.label_lower = tk.Label(self, text='起始年份: ')
        self.label_lower.grid(
            row=1, column=4, padx=0, pady=0, sticky='E'
        )

        self.lower_entry = tk.Entry(self, width=7)
        self.lower_entry.insert(0, '1900')
        self.lower_entry.grid(row=1, column=5, padx=0, pady=0, sticky='W')

        # 结束年份
        self.label_upper = tk.Label(self, text='结束年份: ')
        self.label_upper.grid(row=1, column=6, padx=0, pady=0, sticky='E')

        self.upper_entry = tk.Entry(self, width=7)
        self.upper_entry.insert(0, '2020')
        self.upper_entry.grid(row=1, column=7, padx=0, pady=0, sticky='W')

        # 原始数据显示区域
        self.result_raw = ScrolledText(
```

```
            self, height=40, width=55, state='disabled')
        self.result_raw.grid(
            row=2, column=0, columnspan=4, padx=5, pady=0, sticky='NSEW'
        )

        # 文本显示区域
        self.result_text = ScrolledText(
            self, height=40, width=55, state='disabled')
        self.result_text.grid(
            row=2, column=4, columnspan=4, padx=5, pady=0, sticky='NSEW'
        )

        # 保存原始数据按钮
        self.save_raw_button = tk.Button(
            self, text='保存原始数据', width=7, command=self.save_raw
        )
        self.save_raw_button.grid(
            row=3, column=0, padx=10, pady=10, sticky='W'
        )

        # 原始数据动作回显
        self.raw_action_label = tk.Label(self, text='')
        self.raw_action_label.grid(
            row=3, column=1, columnspan=2, padx=10, pady=10
        )

        # 复制原始数据到剪贴板按钮
        self.copy_raw_button = tk.Button(
           self, text='复制到剪贴板', width=7, command=self.copy_raw_to_clipboard
        )
        self.copy_raw_button.grid(
            row=3, column=3, padx=10, pady=10, sticky='E')

        # 保存解码后文本按钮
        self.save_text_button = tk.Button(
            self, text='保存解析文本', width=7, command=self.save_text)
        self.save_text_button.grid(
            row=3, column=4, padx=10, pady=10, sticky='W'
        )

        # 动作回显
        self.text_action_label = tk.Label(self, text='')
        self.text_action_label.grid(
```

```
            row=3, column=5, columnspan=2, padx=10, pady=10)

        # 复制文本到剪贴板按钮
        self.copy_button = tk.Button(
          self, text='复制到剪贴板', width=7, command=self.copy_text_to_clipboard
        )
        self.copy_button.grid(row=3, column=7, padx=10, pady=10, sticky='E')

    def empty_raw_action_label(self) -> None:
        self.raw_action_label.config(text='')

    def empty_text_action_label(self) -> None:
        self.text_action_label.config(text='')

    def perform_query(self) -> None:
        raw: Optional[str] = None
        text: Optional[str] = None

        try:
            server_url: str = self.url_entry.get().rstrip('/')
            query_type: QueryType = QueryType(self.request_type_combobox.get())
            params = WeatherData.Params.try_from(
                self.lower_entry.get(),
                self.upper_entry.get(),
                {'年份顺序': '', '温度升序': 'asc', '温度降序': 'desc'}.get(
                    self.order_combobox.get(), ''
                )
            )
            raw, text = asyncio.run(fetch_data(server_url, query_type, params))
        except Exception as e:
            raw = f'Failed to fetch data: {e}'
            text = ''

        self.result_raw.config(state='normal')
        self.result_raw.delete('1.0', tk.END)
        self.result_raw.insert(tk.END, raw)
        self.result_raw.config(state='disabled')

        self.result_text.config(state='normal')
        self.result_text.delete('1.0', tk.END)
        self.result_text.insert(tk.END, text)
        self.result_text.config(state='disabled')

    def save_raw(self) -> None:
```

```
        query_type: QueryType = QueryType(self.request_type_combobox.get())
        default_extension: str = {
            '/json': '.json', '/csv': '.csv', '/xml': '.xml'
        }.get(query_type.type, 'txt')
        file_type: Tuple[str, str] = {
            '/json': ('JSON Files', '*.json'),
            '/csv': ('CSV Files', '*.csv'),
            '/xml': ('XML Files', '*.xml'),
        }.get(query_type.type, ('Text Files', '*.txt'))
        file_path = asksaveasfilename(
            defaultextension=default_extension,
            filetypes=[file_type, ('All Files', '*.*')],
            initialdir=os.getcwd()
        )
        if file_path:
            with open(file_path, 'w') as f:
                f.write(self.result_raw.get('1.0', tk.END))
            self.raw_action_label.config(text=f'已保存到{file_path}')
            self.after(5000, self.empty_raw_action_label)

    def save_text(self) -> None:
        file_path = asksaveasfilename(
            defaultextension='.txt',
            filetypes=[('Text Files', '*.txt'), ('All Files', '*.*')],
            initialdir=os.getcwd()
        )
        if file_path:
            with open(file_path, 'w') as f:
                f.write(self.result_text.get('1.0', tk.END))
            self.text_action_label.config(text=f'已保存到{file_path}')
            self.after(5000, self.empty_text_action_label)

    def copy_raw_to_clipboard(self) -> None:
        pyperclip.copy(self.result_raw.get('1.0', tk.END))
        self.raw_action_label.config(text='已复制到剪贴板')
        self.after(5000, self.empty_raw_action_label)

    def copy_text_to_clipboard(self) -> None:
        pyperclip.copy(self.result_text.get('1.0', tk.END))
        self.text_action_label.config(text='已复制到剪贴板')
        self.after(5000, self.empty_text_action_label)


if __name__ == '__main__':
```

```
    app = App()
    app.mainloop()
```

3 代码说明

3.1 程序需求分析

基于第二次作业实现的 web 服务器提供的接口，编写一个客户端程序，能够查询并展示这些数据。上一次作业中实现的 web 服务器包含以下功能：

- 根据年份范围查询数据。
- 默认按照年份升序排序，也可以指定按照年平均温度（第二列）升序或降序排序。
- 可以返回 JSON、XML 或 CSV 格式的数据，具体格式见下文介绍。

由此，确定本次的客户端实现以下功能：

- 采取 GUI 形式，提供用户界面。
- 指定服务器地址、端口、查询接口，向服务器发起请求。
- 根据用户输入的年份范围、排序方式，构造请求参数。
- 将服务器返回的数据按原始格式展示。
- 将服务器返回的数据转换为文本格式并展示。
- 将原始数据和转换后的文本保存到文件或剪贴板中。

3.2 程序基本介绍与运行方式

这段代码实现了一个简易的天气数据查询 GUI 客户端（暂定名为 Simple Weather Data Client），能够处理和响应用户对历年气温数据的查询，GUI 采用 tkinter 库实现，网络通信采用 aiohttp 库实现。

程序风格方面，由于作业要求，代码实现为了单文件，没有以文件的形式分离各个模块。与前两次作业一样，这次作业仍然严格遵守 Python 的 PEP8 编码规范，在行过长时仍然进行了手动格式化[1][2]。本次的代码中两次手动忽略类型检查后通过了所有的类型标注检查。

开发环境如下表：

软件环境或第三方库名	版本
操作系统 / macOS	13.0 22A380 arm64
Python	3.10.6

[1]官方的 autopep8 工具效果实在差。
[2]不保证不会有漏网之鱼。

aiohttp	3.8.6
pyperclip	1.8.2

没有显式地用到更多的第三方库。

在安装 Python 后，可以使用以下命令安装依赖与运行本程序（2nd.py）：

```
$ python3 -m pip install aiohttp pyperclip
$ python3 3rd.py
```

3.3 程序组件介绍

数据类与数据处理相关大部分都已在第二次作业中介绍，这里不再赘述。以下是本次作业中新增的组件。

3.3.1 网络通信

用到的所有网络通信需求都通过 aiohttp 库异步进行，封装在函数 `fetch_data` 中完成。函数接收三个参数，分别是服务器地址、查询接口 route、GET 请求的参数；返回一个元组，包含两个字符串，分别是原始数据和转换后的文本数据。格式的转换通过 `WeatherData` 类的 `to_json`、`to_csv` 和 `to_xml` 方法完成。

完整定义如下：

```
async def fetch_data(
    server: str, query_type: QueryType, params: WeatherData.Params
) → Tuple[str, str]:
    url = server + query_type.type

    async with aiohttp.ClientSession() as session:
        async with session.get(url, params=params.to_dict()) as response:
            if response.status ≠ 200:
                raise Exception(
                    'Failed to fetch data from '
                    f'{url} {response.status} {response.reason}'
                )

            raw: Optional[str] = None
            pretty: Optional[str] = None
            decoded: Optional[WeatherData] = None

            if query_type.type == '/json':
                raw = await response.text()
                pretty = json.dumps(json.loads(raw), indent=2)
                decoded = WeatherData.from_json(raw)
```

```
            elif query_type.type == '/csv':
                raw = await response.text()
                pretty = unify_crlf(raw)
                decoded = WeatherData.from_csv(raw)
            elif query_type.type == '/xml':
                raw = await response.text()
                pretty = unify_crlf(
                    xml.dom.minidom.parseString(raw).toprettyxml(indent='  ')
                )
                decoded = WeatherData.from_xml(raw)
            else:
                raise Exception(f'Failed to fetch data from {url}')

            return pretty, str(decoded)
```

3.3.2 GUI 界面与应用

程序的 GUI 界面托管于 tkinter 库，由 App 类实现，继承自 tkinter.Tk。布局如下：

	服务器：	输入框					查询
查询类型：	下拉框	起始年份：	输入框	结束年份：	输入框	排序方式：	下拉框
原始数据				转换后的文本			
保存	动作提示信息		复制	保存	动作提示信息		复制

查询类型的可选值为 /json、/csv 和 /xml，排序方式的可选值为年份顺序、温度升序和温度降序。

点击查询按钮后，程序会获取前两行的控件中的值，构造请求参数，向服务器发起请求，将返回的数据展示在第三行的文本框控件中。文本框自带了滚动条避免内容过长无法显示的问题。

点击保存按钮后，程序调用操作系统的文件保存对话框 API。点击复制按钮后，程序通过 pyperclip 库将对对应的文本框中的内容复制到剪贴板中。动作提示信息会在用户点击保存或复制按钮后显示，提示用户操作成功，持续 5 秒后消失。

3.4 程序运行截图

以下给出各种功能的运行指引和截图。

程序启动后，会显示主界面，如下图所示。服务器的默认值 `http://127.0.0.1:555` 指向了部署在我本地的 web 服务器，其他选项也各自设定了默认值，用户可以直接点击查询按钮查看示例数据。

图 1：主界面

使用默认参数点击查询按钮后，会显示查询结果，如下图所示。原始数据和转换后的文本数据都会显示在对应的文本框中。可以看到，json 回显经过美化后展示在文本框中。

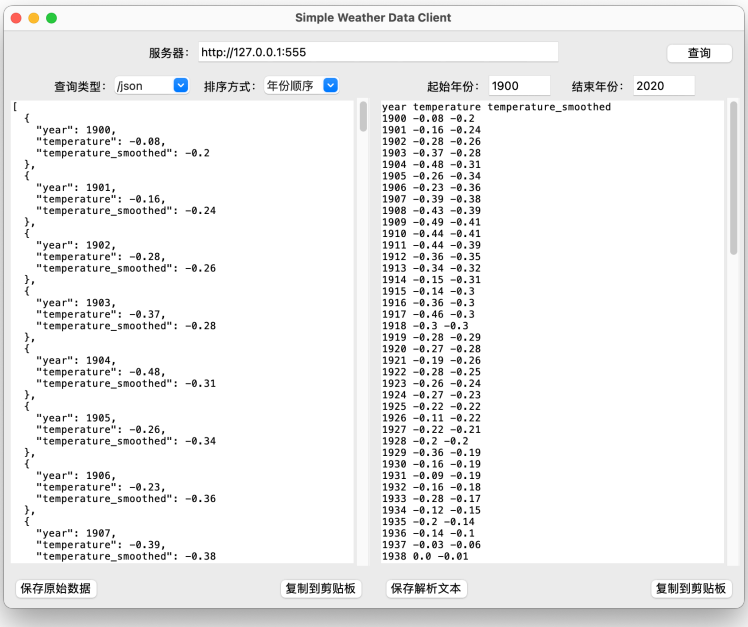

图 2：默认选项查询结果 - json/年份顺序/1900/2020

修改参数为 `/csv`、`温度升序`、`1976`、`2003` 后点击查询按钮，会显示查询结果，如下图所示。csv 格式的回显经过统一换行符后展示在文本框中。虽然 csv 格式的数据与纯文本数据格式极其相似，但其在本质上仍然是不同的。

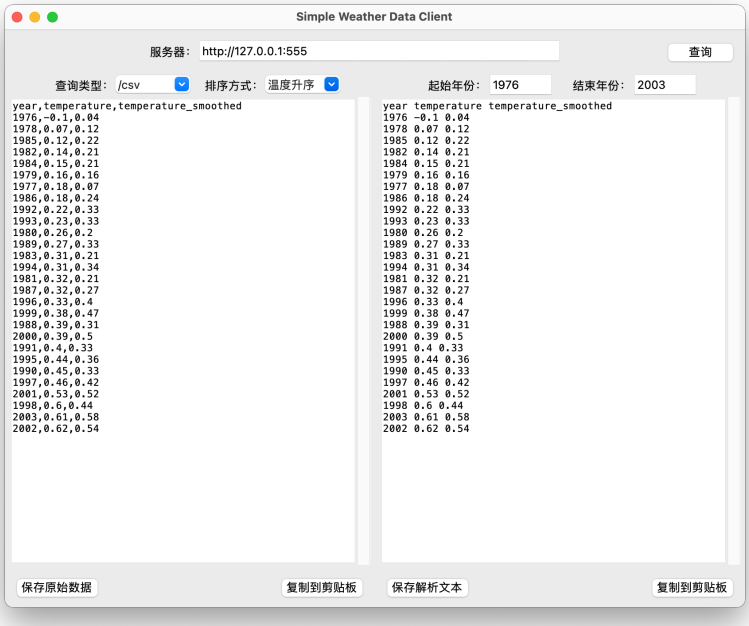

图 3：查询结果 - csv/温度升序/1976/2003

修改参数为 `/xml`、`温度降序`、`2003`、`2022` 后点击查询按钮，会显示查询结果，如下图所示。xml 格式的回显经过美化后展示在文本框中。

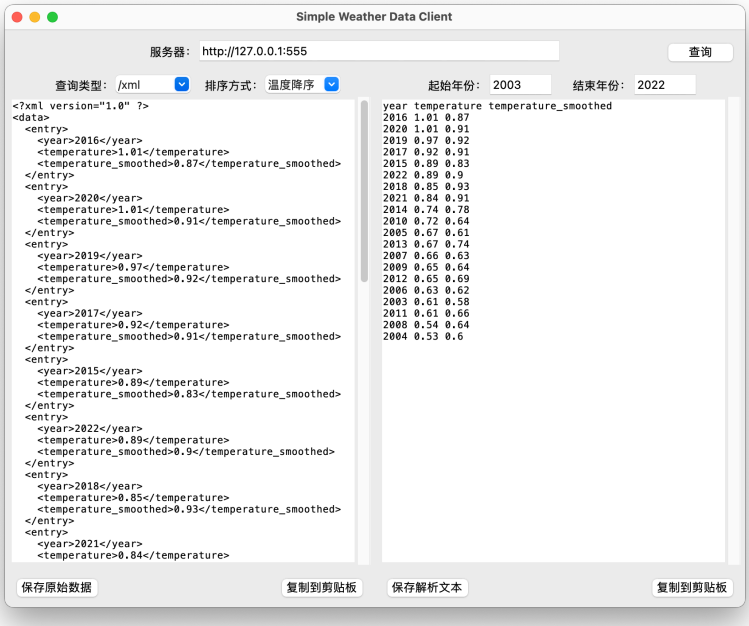

图 4：查询结果 - xml/温度降序/2003/2022

将默认参数下的原始数据保存到文件中，会弹出文件保存对话框，如下图所示。用户可以选择保存的文件名和保存的位置。

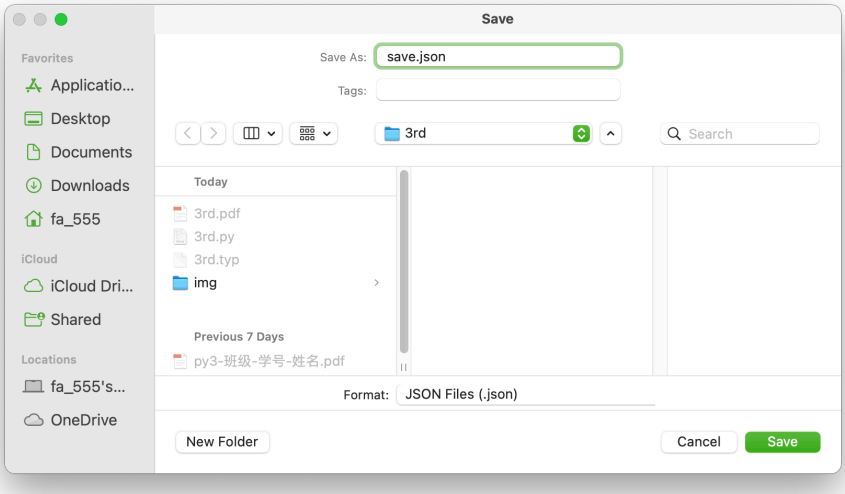

图 5：保存对话框

将默认参数下的解析文本复制到剪贴板中，出现动作提示信息，如左图所示。动作提示信息会在用户点击保存或复制按钮后显示，提示用户操作成功，持续 5 秒后消失。同时我们检查剪贴板中的内容，如右图所示，可以看到剪贴板中的内容确实是我们复制的文本。

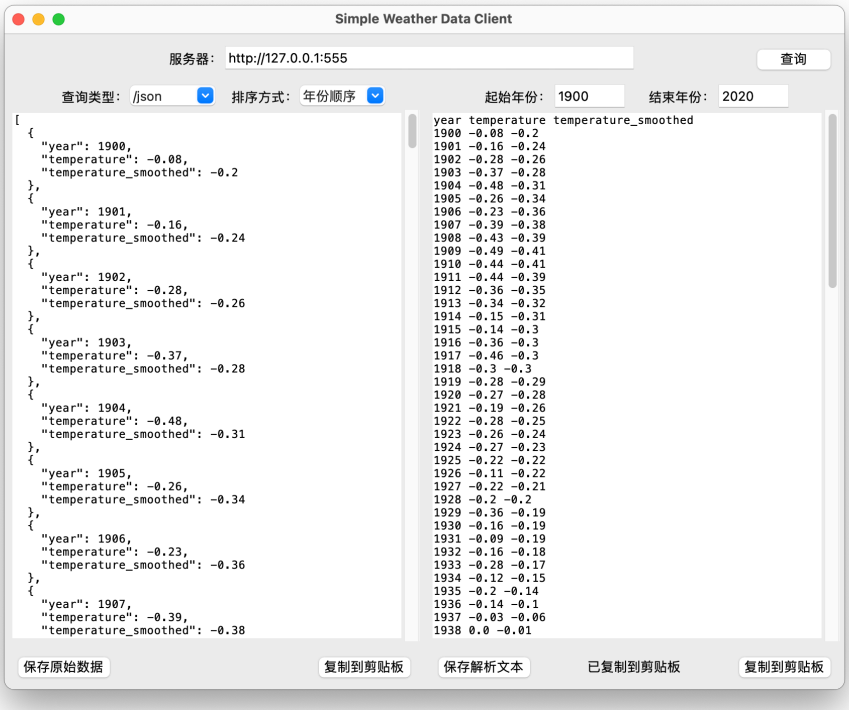

图 6：复制提示信息

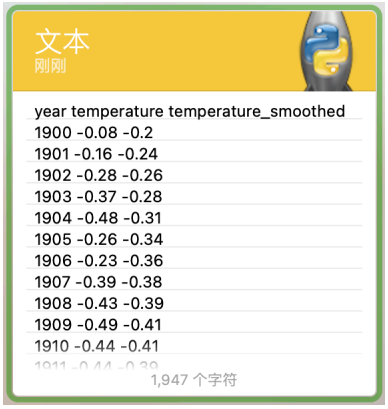

图 7：剪贴板内容

3.5 总结

在这次作业中，我实现了一个简易的天气数据查询客户端，能够处理和响应用户对历年气温数据的查询。这次作业的重点在于各种格式的数据到文本格式的转换，以及如何将数据展示在用户界面上。倘若时间充足，本次作业还有以下的改进空间：

- 与服务器同步继续增加接口的功能或提供更多接口；将接口改为 POST 请求。
- 重构代码，将源代码按照模块进行分离，方便后续维护。这样，第二次作业的服务器与本次作业的客户端可以使用同样的数据类型定义，方便扩展。
- 加强错误处理和日志记录，确保异常可以被适当捕获并返回有用的错误信息和操作记录。
- 规范用户界面，增加更多的交互功能，例如增加菜单栏、增加更多的快捷键、增加更多的提示信息等等。