

# Python 程序设计

## 第二次作业：简易 Web 服务器

2021211306 班 | 杜抒泽 2021211110

2023 年 11 月 5 日

### 目录

1 作业题目	2
2 作业内容	2
3 代码说明	7
3.1 简要需求分析	7
3.2 程序基本介绍与运行方式	8
3.3 程序组件介绍	8
3.3.1 工具函数	8
3.3.2 数据类与数据处理工具	9
3.3.3 应用配置与管理	9
3.3.4 应用初始化与运行	10
3.4 Sanic 路由配置暨 API 文档	10
3.4.1 /ping	10
3.4.2 /json	10
3.4.3 /csv	11
3.4.4 /xml	12
3.5 程序风格：PEP8 规范与类型注解	13
3.6 API 调用截图展示	13
3.6.1 /json	14
3.6.2 /csv	15
3.6.3 /xml	16
3.7 未来完善方向	17
3.8 总结	17

## 1 作业题目

数据文件 (graph.txt) 是一个全球温度年度异常历史数据。基于 Sanic 实现一个查询服务，服务包括：

- 按起始和结束年份查询历史数据，查询结果支持多种格式：JSON、XML、CSV（用逗号作为间隔符）。
- 按温度高低进行排序，支持升序和降序两种排序方式。

## 2 作业内容

程序源代码嵌入下方的 code block 中。由于代码较长，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。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。

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

from io import StringIO
from sanic import Sanic, response
from sanic.config import Config
from sanic.request import Request
from sanic.response import HTTPResponse
from typing import Any, Dict, List, Optional, Union

import csv
import os
import shutil

# ----- utilities -----


def dict_to_xml(data: Dict[str, Any]) → str:
    xml: str = ''
    for key, value in data.items():
        xml += f'<{key}>{value}</{key}>'
    return xml


def try_remove(path: str) → None:
    try:
```

```
if os.path.isdir(path):
    shutil.rmtree(path)
elif os.path.isfile(path):
    os.remove(path)
except FileNotFoundError:
    pass

# ----- data -----


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

    def to_dict(self) -> Dict[str, Union[int, float]]:
        return {
            'year': self.year,
            'temperature': self.temperature,
            'temperature_smoothed': self.temperature_smoothed
        }


class WeatherData:
    def __init__(self, data: List[WeatherDataEntry]) -> None:
        self.data: List[WeatherDataEntry] = data

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

class Params:
    def __init__(
        self, lower: int, upper: int, order: Optional[str] = None
    ) -> None:
        self.lower: int = lower
        self.upper: int = upper
        self.order: Optional[str] = order
        if self.order is not None and self.order not in ('asc', 'desc'):
            raise ValueError(
                f'Invalid order: {self.order}, '
            )
```

```
'must be either "asc" or "desc"
)

def query(self, params: Params) → List[WeatherDataEntry]:
    if params.order == None:
        return sorted(list(filter(
            lambda x: params.lower ≤ x.year ≤ params.upper, self.data
)), key=lambda x: x.year)

    return sorted(filter(
        lambda x: params.lower ≤ x.year ≤ params.upper, self.data
), key=lambda x: x.temperature, reverse=params.order == 'desc')

def query_dict(
    self, params: Params
) → List[Dict[str, Union[int, float]]]:
    return list(map(WeatherDataEntry.to_dict, self.query(params)))

def load_data_from_file(file_name: str) → WeatherData:
    data: WeatherData = WeatherData([])

    with open(file_name, 'r') as f:
        lines: List[str] = f.readlines()
        for line in lines:
            if line.startswith('#') or line == '\n':
                continue

            year, temperature, temperature_smoothed = line.split()
            data.append(WeatherDataEntry(
                int(year), float(temperature), float(temperature_smoothed)
            ))

    return data

# ----- app [config] -----


class AppConfig(Config):
    WEATHER_DATA_SRC_FILE: str
    WEATHER_DATA: WeatherData
    INSTANCE_DIR: str
    CACHE_DIR: str
```

```
def __init__(self, **kwargs: str) → None:
    super().__init__()

    self.INSTANCE_DIR = kwargs['INSTANCE_DIR']
    try_remove(self.INSTANCE_DIR)
    os.makedirs(self.INSTANCE_DIR, exist_ok=True)

    self.CACHE_DIR = kwargs['CACHE_DIR']
    try_remove(self.CACHE_DIR)
    os.makedirs(self.CACHE_DIR, exist_ok=True)

    self.WEATHER_DATA_SRC_FILE = kwargs['WEATHER_DATA_SRC_FILE']
    self.WEATHER_DATA = load_data_from_file(self.WEATHER_DATA_SRC_FILE)

WEATHER_DATA_SRC_FILE: str = './graph.txt'
INSTANCE_DIR: str = './instance'
CACHE_DIR = INSTANCE_DIR + '/cache'

app: Sanic = Sanic(
    'SimpleWeatherDataServer',
    config=AppConfig(
        WEATHER_DATA_SRC_FILE=WEATHER_DATA_SRC_FILE,
        INSTANCE_DIR=INSTANCE_DIR,
        CACHE_DIR=CACHE_DIR
    ),
)

# ----- app [api] -----


def extract_get_params(request: Request) → WeatherData.Params:
    lower_year: int = int(request.args['lower'][0])
    upper_year: int = int(request.args['upper'][0])
    order: Optional[str] = request.args.get('order')

    return WeatherData.Params(lower_year, upper_year, order)

@app.get('/ping')
async def ping_handler(request: Request) → HTTPResponse:
    arg: str = request.args.get('arg', 'ping')
```

```
return response.text(arg.replace('i', 'o').replace('I', 'O'))
```

```
@app.get('/json')
async def get_json_handler(request: Request) -> HTTPResponse:
    params: Optional[WeatherData.Params] = None

    try:
        params = extract_get_params(request)
    except Exception as e:
        return response.json({'message': f'Invalid query: {e}'}, status=400)

    try:
        return response.json(app.config.WEATHER_DATA.query_dict(params))
    except Exception as e:
        return response.json({'message': f'Internal error: {e}'}, status=500)
```

```
@app.get('/csv')
async def get_csv_handler(request: Request) -> HTTPResponse:
    params: Optional[WeatherData.Params] = None

    try:
        params = extract_get_params(request)
    except Exception as e:
        return response.json({'message': f'Invalid query: {e}'}, status=400)

    try:
        output_stream = StringIO()
        writer = csv.writer(output_stream)
        writer.writerow(['year', 'temperature', 'temperature_smoothed'])
        for entry in app.config.WEATHER_DATA.query(params):
            writer.writerow([
                entry.year, entry.temperature, entry.temperature_smoothed
            ])
        return response.text(
            output_stream.getvalue(), content_type='text/csv'
        )
    except Exception as e:
        return response.json({'message': f'Internal error: {e}'}, status=500)
```

```
@app.get('/xml')
```

```

async def get_xml_handler(request: Request) -> HTTPResponse:
    params: Optional[WeatherData.Params] = None

    try:
        params = extract_get_params(request)
    except Exception as e:
        return response.json({'message': f'Invalid query: {e}'}, status=400)

    try:
        data = app.config.WEATHER_DATA.query(params)

        xml = '<?xml version="1.0" encoding="UTF-8"?>'
        xml += '<data>'
        for entry in data:
            xml += '<entry>'
            xml += dict_to_xml(entry.to_dict())
            xml += '</entry>'
        xml += '</data>'

        return response.text(xml, content_type='text/xml')

    except Exception as e:
        return response.json({'message': f'Internal error: {e}'}, status=500)

if __name__ == '__main__':
    app.run(port=555)

```

## 3 代码说明

### 3.1 简要需求分析

题目给出了一段天气数据，包含 1880 到 2022 年的基于陆地和海洋数据的全球年平均气温估计<sup>1</sup>。数据文件包含三列，第一列是年份，第二列是年平均温度估计，第三列是对第二列进行平滑处理得到的数。第三列可能在分析整体数据时具有统计学用途，单独讨论其中某一年没有显著意义。

根据上述对数据的分析，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。

<sup>1</sup>根据 graph.txt 中给出的链接查阅了 NASA 官网仔细阅读数据来源得到，题目描述中提到的“全球温度年度异常历史数据”是翻译错误。

- 可以返回 JSON、XML 或 CSV 格式的数据，具体格式见下文介绍。

## 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后端服务器（暂定名为 Simple Weather Data Server），能够处理和响应用户对历年气温数据的查询，采用 Sanic 框架提供异步处理能力，能够响应多种数据格式的输出。

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
Sanic / sanic-routing / sanic-ext	23.6.0

除 Sanic 主动引用的依赖外，没有显式地用到更多的第三方库。理论上所有能够运行 Sanic 库的 Python 环境（3.7+）都可以运行本程序，但未经过测试。

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

```
$ python3 -m pip install sanic[ext]  
$ python3 2nd.py # 若报错 Permission denied 则需要 sudo 或其他方式获取管理员权限
```

由于工期较短，程序并未实现命令行参数解析。程序的配置信息（如数据文件名、服务器实例与缓存目录）全部硬编码在代码中。如果需要修改这些配置信息，可以直接修改代码。

## 3.3 程序组件介绍

### 3.3.1 工具函数

这段代码中定义了一些工具函数，包含数据格式转换、文件系统操作等与具体业务无关的代码。

- `def dict_to_xml(data: Dict[str, Any]) → str`

接受一个 `Dict[str, Any]` 类型的参数，表示一个嵌套的字典。返回一个 `str` 类型的结果，表示将该字典转换为 XML 格式的字符串。由于 `data` 的类型较简单，因

此该函数并不涉及递归等复杂的逻辑。字典键值对中的 Any 实际上表示可以接受任何实现了魔术方法 `__repr__` 的类型。

- `def try_remove(path: str) → None`

接受一个 `str` 类型的参数，表示一个文件或文件夹的路径。若文件或文件夹不存在，忽略错误。若文件夹存在且不为空，递归删除文件夹中的所有文件和文件夹，然后删除该文件夹。否则直接删除该文件或文件夹。封装了文件系统管理中常见的操作，确保了代码的健壮性，提前捕获了部分可能抛出的异常。

### 3.3.2 数据类与数据处理工具

- `class WeatherDataEntry`

`WeatherDataEntry` 类为气温数据模型的定义，封装了年份、气温及平滑处理后的气温数据。`to_dict` 方法使得将此类的实例转换成字典变得简单，便于进行数据序列化或其他形式的处理。

- `class WeatherData`

`WeatherData` 类封装了 `List[WeatherDataEntry]`。承担了存储和操作 `WeatherDataEntry` 实例集合的职责。其内联的 `Params` 类专门用于查询请求的参数的封装与校验，确保了查询逻辑的稳定性和准确性。`query` 方法实现了查询的业务逻辑，`query_dict` 方法调用 `query` 方法并提供进一步封装，它们提供了数据查询和结果格式化的能力。

- `def load_data_from_file(file_name: str) → WeatherData`

接受一个 `str` 类型的参数，表示一个文件的路径。返回一个 `WeatherData` 类型的结果，表示从文件中读取的数据。这是数据加载暨程序读文件的唯一入口，通过读取文件并解析，将气温数据加载进 `WeatherData` 实例中。

### 3.3.3 应用配置与管理

- `class AppConfig(Config)`

`AppConfig` 类继承 `Sanic` 框架的 `Config` 类，又增添了一些专门针对本应用需求的配置项，包括对于气象数据源文件路径、实例目录和缓存目录的定义。类构造器中的逻辑确保在应用启动时这些目录被创建，同时对气象数据的加载（前文提到的 `load_data_from_file` 函数）也在此被调用。它可以被视为程序的启动配置文件，管理着程序运行所需的各项基础设施，展示了对应用生命周期管理的设计。

### 3.3.4 应用初始化与运行

- Sanic 框架实例化与配置

在代码的尾部，通过实例化一个 Sanic 对象 `app` 来定义了应用本身，通过 `AppConfig` 为应用配置了必要的参数，并绑定了一系列路由到 API。这一部分代码展示了 Sanic 框架的灵活性和强大功能，也展示了应用配置与框架功能如何无缝集成。使用 `AppConfig` 类对 Sanic 进行配置的方式体现了一个更高层次的抽象设计，允许开发者在单一的地方管理应用的配置，而无需散布在代码库的不同位置，简化了配置管理，使得开发集中精力于业务逻辑而非配置细节。

之后，通过调用 `app.run` 方法，应用启动，开始侦听网络请求。在这之前，所有的准备工作都已就绪，包括服务的配置、路由的设置以及其他任何必要的初始化过程。

## 3.4 Sanic 路由配置暨 API 文档

根据作业要求，所有接口均只响应 GET 请求。若后续接入前端，会全部修改为响应 POST 请求。

### 3.4.1 /ping

**接口功能** 检查应用服务的状态，通常用于监控应用是否正常运行。

**请求类型** GET

**请求参数** (可选) arg

**成功响应** 返回 200 OK 状态码，表示服务运行正常。若请求中包含参数 `arg`，则返回 `arg` 的值，其中的字符 `i` 和 `I` 分别被替换为 `o` 和 `O`；否则返回 `pong`，如同 `arg` 的值为 `ping`。

**失败响应** 一般不会出现失败响应，如果服务器内部出现问题，则可能返回 500 Internal Server Error。

### 3.4.2 /json

**接口功能** 此接口提供指定时间范围内的气象数据，数据以 JSON 格式返回，适合 web 前端收取并展示。若不指定排序方式则按照年份升序排序；若指定排序方式为 `asc` 或 `desc` 则按照温度升序或降序排序。

**请求类型** GET

## 请求参数

- `lower` (必须) : 指定查询数据的起始年份 (含)。
- `upper` (必须) : 指定查询数据的结束年份 (含)。
- `order` (可选) : 通过此参数指定结果的排序方式, `asc` 表示按年份升序, `desc` 表示按年份降序。

成功响应 返回 200 OK 状态码, 并且响应体中包含请求年份范围内的气象数据, 格式为 JSON 文本。例如 `/json?lower=2001&upper=2003&order=asc` 返回:

```
[  
  {  
    "year": 2001,  
    "temperature": 0.53,  
    "temperature_smoothed": 0.52  
  },  
  {  
    "year": 2003,  
    "temperature": 0.61,  
    "temperature_smoothed": 0.58  
  },  
  {  
    "year": 2002,  
    "temperature": 0.62,  
    "temperature_smoothed": 0.54  
  }  
]
```

## 失败响应

- 400 Bad Request: 必须参数缺失或参数不合法。
- 500 Internal Error: 如果服务器处理请求时发生了错误。

### 3.4.3 /csv

接口功能 此接口提供指定时间范围内的气象数据, 数据以 CSV 格式返回, 适合用于电子表格软件或数据分析。若不指定排序方式则按照年份升序排序; 若指定排序方式为 `asc` 或 `desc` 则按照温度升序或降序排序。

请求类型 GET

## 请求参数

- `lower` (必须) : 指定查询数据的起始年份 (含)。

- `upper` (必须) : 指定查询数据的结束年份 (含)。
- `order` (可选) : 通过此参数指定结果的排序方式, `asc` 表示按年份升序, `desc` 表示按年份降序。

成功响应 返回 `200 OK` 状态码, 并且响应体中包含请求年份范围内的气象数据, 格式为 CSV 文本。实践中, 浏览器可能直接将文件保存。已知 FireFox 和 Chrome 会采取此策略; 而 Safari 会将其当作纯文本显示。`/csv?lower=2001&upper=2003&order=asc` 返回:

```
year,temperature,temperature_smoothed
2001,0.53,0.52
2003,0.61,0.58
2002,0.62,0.54
```

## 失败响应

- `400 Bad Request`: 必须参数缺失或参数不合法。
- `500 Internal Error`: 如果服务器处理请求时发生了错误。

### 3.4.4 /xml

接口功能 此接口提供指定时间范围内的气象数据, 数据以 XML 格式返回。若不指定排序方式则按照年份升序排序; 若指定排序方式为 `asc` 或 `desc` 则按照温度升序或降序排序。

请求类型 GET

## 请求参数

- `lower` (必须) : 指定查询数据的起始年份 (含)。
- `upper` (必须) : 指定查询数据的结束年份 (含)。
- `order` (可选) : 通过此参数指定结果的排序方式, `asc` 表示按年份升序, `desc` 表示按年份降序。

成功响应 返回 `200 OK` 状态码, 并且响应体中包含请求年份范围内的气象数据, 格式为 XML 文本。实践中, 浏览器一般不会展示 XML 的元数据。`/xml?lower=2001&upper=2003&order=asc` 返回:

```
<?xml version="1.0" encoding="UTF-8"?>
<data>
```

```
<entry>
    <year>2001</year>
    <temperature>0.53</temperature>
    <temperature_smoothed>0.52</temperature_smoothed>
</entry>
<entry>
    <year>2003</year>
    <temperature>0.61</temperature>
    <temperature_smoothed>0.58</temperature_smoothed>
</entry>
<entry>
    <year>2002</year>
    <temperature>0.62</temperature>
    <temperature_smoothed>0.54</temperature_smoothed>
</entry>
</data>
```

## 失败响应

- 400 Bad Request: 必须参数缺失或参数不合法。
- 500 Internal Error: 如果服务器处理请求时发生了错误。

## 3.5 程序风格：PEP8 规范与类型注解

与第一次作业一样，这次作业仍然严格遵守 Python 的 PEP8 编码规范，在行过长时进行了手动格式化（官方的 `autopep8` 工具效果实在不太行）。这很大程度上是因为代码实现了冗长的类型注解。

利用 `mypy` 分析工具，开启所有不允许偷懒的选项，我们得到如下的类型检查结果：

```
$ mypy 2nd.py --check-untyped-defs --disallow-untyped-calls --disallow-untyped-defs --disallow-incomplete-defs --disallow-untyped-decorators
Success: no issues found in 1 source file
```

可以看到，这段代码完全通过了所有的类型标注检查。美观性的略微丧失并未导致可读性的降低，但是带来了类型安全，这种 trade-off 是完全值得的。

## 3.6 API 调用截图展示

实验要求在浏览器中进行测试，在此给出查询接口各自一个请求在浏览器和 postman 中的请求与响应截图。

### 3.6.1 /json

请求为 `/json?lower=2003&upper=2023&order=desc`, 响应如下 (FireFox 自带 json 响应美化)：

```

[{"year": 2016, "temperature": 1.01, "temperature_smoothed": 0.87}, {"year": 2020, "temperature": 1.01, "temperature_smoothed": 0.91}, {"year": 2019, "temperature": 0.97, "temperature_smoothed": 0.92}, {"year": 2017, "temperature": 0.92, "temperature_smoothed": 0.91}, {"year": 2015, "temperature": 0.89, "temperature_smoothed": 0.83}, {"year": 2022, "temperature": 0.89, "temperature_smoothed": 0.9}, {"year": 2018, "temperature": 0.85, "temperature_smoothed": 0.93}, {"year": 2021, "temperature": 0.84, "temperature_smoothed": 0.91}, {"year": 2014, "temperature": 0.74, "temperature_smoothed": 0.78}, {"year": 2010, "temperature": 0.72, "temperature_smoothed": 0.64}, {"year": 2005, "temperature": 0.67, "temperature_smoothed": 0.61}, {"year": 2013, "temperature": 0.67, "temperature_smoothed": 0.74}, {"year": 2007, "temperature": 0.66, "temperature_smoothed": 0.63}, {"year": 2009, "temperature": 0.65, "temperature_smoothed": 0.64}, {"year": 2012, "temperature": 0.65, "temperature_smoothed": 0.69}, {"year": 2006, "temperature": 0.63, "temperature_smoothed": 0.62}]

```

图 1：浏览器中的 json 响应

```

[{"year": 2016, "temperature": 1.01, "temperature_smoothed": 0.87}, {"year": 2020, "temperature": 1.01, "temperature_smoothed": 0.91}, {"year": 2019, "temperature": 0.97, "temperature_smoothed": 0.92}, {"year": 2017, "temperature": 0.92, "temperature_smoothed": 0.91}, {"year": 2015, "temperature": 0.89, "temperature_smoothed": 0.83}, {"year": 2022, "temperature": 0.89, "temperature_smoothed": 0.9}, {"year": 2018, "temperature": 0.85, "temperature_smoothed": 0.93}, {"year": 2021, "temperature": 0.84, "temperature_smoothed": 0.91}, {"year": 2014, "temperature": 0.74, "temperature_smoothed": 0.78}, {"year": 2010, "temperature": 0.72, "temperature_smoothed": 0.64}, {"year": 2005, "temperature": 0.67, "temperature_smoothed": 0.61}, {"year": 2013, "temperature": 0.67, "temperature_smoothed": 0.74}, {"year": 2007, "temperature": 0.66, "temperature_smoothed": 0.63}, {"year": 2009, "temperature": 0.65, "temperature_smoothed": 0.64}, {"year": 2012, "temperature": 0.65, "temperature_smoothed": 0.69}, {"year": 2006, "temperature": 0.63, "temperature_smoothed": 0.62}]

```

图 2：Postman 中的 json 响应

### 3.6.2 /csv

请求为 `/csv?lower=1976&upper=2023`, 响应如下 (Safari 直接显示文本而其他浏览器大多将其当作文件下载)：

```
year,temperature,temperature_smoothed
1976,-0.1,0.04
1977,0.18,0.07
1978,0.07,0.12
1979,0.16,0.16
1980,0.26,0.2
1981,0.32,0.21
1982,0.14,0.21
1983,0.31,0.21
1984,0.15,0.21
1985,0.12,0.22
1986,0.18,0.24
1987,0.32,0.27
1988,0.39,0.31
1989,0.27,0.33
1990,0.45,0.33
1991,0.4,0.33
1992,0.23,0.33
1993,0.22,0.33
1994,0.31,0.34
1995,0.44,0.36
1996,0.33,0.4
1997,0.46,0.42
1998,0.6,0.44
1999,0.38,0.47
2000,0.39,0.5
2001,0.53,0.52
2002,0.62,0.54
2003,0.61,0.58
2004,0.53,0.6
2005,0.67,0.61
2006,0.63,0.62
2007,0.64,0.63
2008,0.54,0.64
2009,0.65,0.64
2010,0.72,0.64
2011,0.61,0.66
2012,0.65,0.69
2013,0.67,0.74
2014,0.74,0.78
2015,0.89,0.83
2016,1.01,0.87
2017,0.92,0.91
2018,0.85,0.93
2019,0.97,0.92
2020,1.01,0.91
2021,0.84,0.91
2022,0.89,0.9
```

图 3：浏览器中的 csv 响应

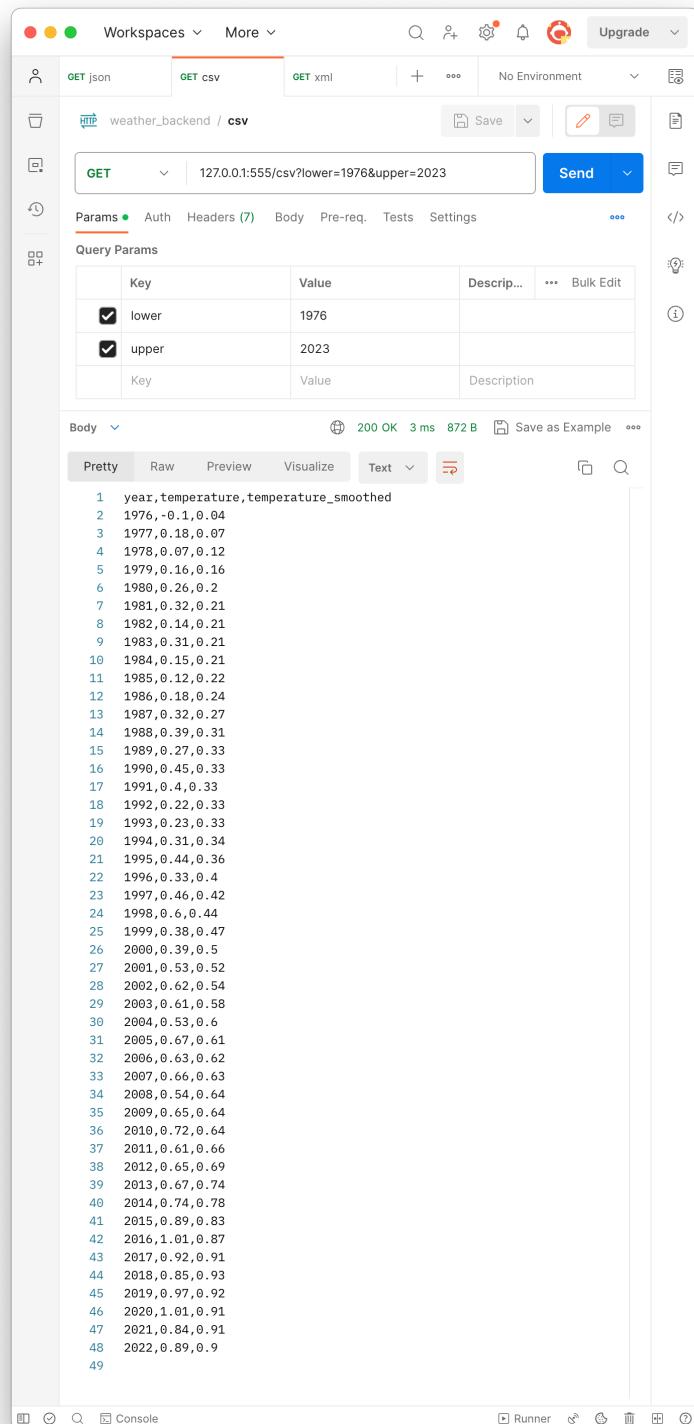


图 4：Postman 中的 csv 响应

### 3.6.3 /xml

请求为 /xml?lower=2003&upper=2023&order=asc，响应如下：

```

<data>
<entry><year>2004</year>
<temperature>0.53</temperature>
<temperature_smoothed>0.6</temperature_smoothed>
</entry>
<entry>
<year>2008</year>
<temperature>0.54</temperature>
<temperature_smoothed>0.64</temperature_smoothed>
</entry>
<entry>
<year>2003</year>
<temperature>0.61</temperature>
<temperature_smoothed>0.58</temperature_smoothed>
</entry>
<entry>
<year>2011</year>
<temperature>0.61</temperature>
<temperature_smoothed>0.66</temperature_smoothed>
</entry>
<entry>
<year>2006</year>
<temperature>0.63</temperature>
<temperature_smoothed>0.62</temperature_smoothed>
</entry>
<entry>
<year>2009</year>
<temperature>0.65</temperature>
<temperature_smoothed>0.64</temperature_smoothed>
</entry>
<entry>
<year>2012</year>
<temperature>0.65</temperature>
<temperature_smoothed>0.69</temperature_smoothed>
</entry>
<entry>
<year>2007</year>
<temperature>0.66</temperature>
<temperature_smoothed>0.63</temperature_smoothed>
</entry>
<entry>
<year>2005</year>
<temperature>0.67</temperature>
<temperature_smoothed>0.61</temperature_smoothed>
</entry>
<entry>
<year>2013</year>
<temperature>0.67</temperature>
<temperature_smoothed>0.74</temperature_smoothed>
</entry>
<entry>
<year>2010</year>
<temperature>0.72</temperature>
<temperature_smoothed>0.64</temperature_smoothed>
</entry>
<entry>
<year>2014</year>
<temperature>0.74</temperature>
<temperature_smoothed>0.78</temperature_smoothed>
</entry>
<entry>
<year>2021</year>
<temperature>0.84</temperature>
<temperature_smoothed>0.91</temperature_smoothed>
</entry>
<entry>

```

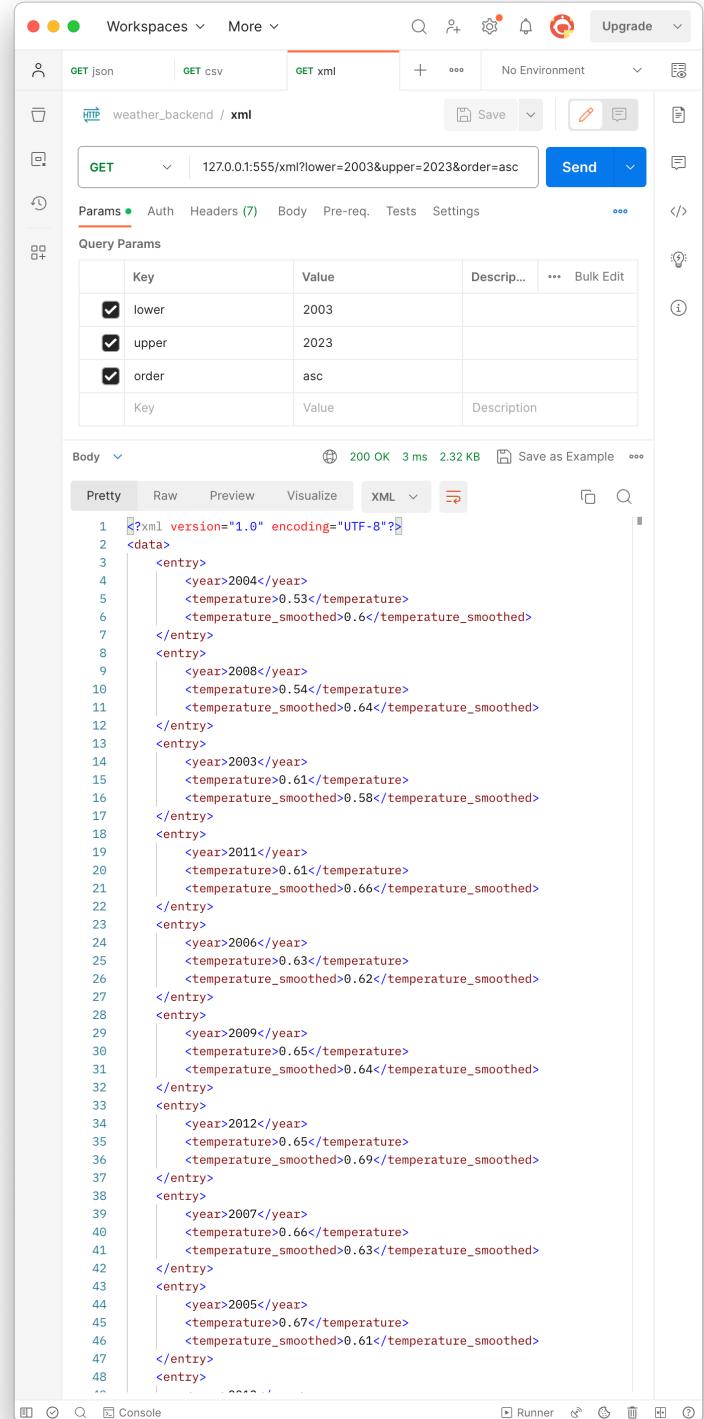


图 5：浏览器中的 xml 响应

图 6：Postman 中的 xml 响应

### 3.7 未来完善方向

由于时间有限，本次作业并未实现较多功能。若后续作业跟进用户界面，以下是在该 web 服务器的一些可以进一步完善的地方：

- 为 /json、/csv 和 /xml 接口增加 POST 请求支持以便接入前端。
- 将源代码进行分离，将各个模块分别放置在不同的文件中，方便后续维护，同时构建起一个真正的项目。
- 继续增加接口的功能或提供更多接口。
- 加强错误处理机制，确保异常可以被适当捕获并返回有用错误信息给维护人员；同时引入详细的日志记录系统，对关键操作进行记录，方便问题追踪。
- 将服务端部署到公网服务器上。

### 3.8 总结

在这次作业中，我搭建起了一个气象数据查询服务器，提供了能返回多种响应格式的 RESTful API 接口。

此前在使用 Python 完成后端开发时，我向来使用 Flask 框架。本次使用的 Sanic 框架与 Flask 颇有相似之处，但 Sanic 框架有更强的异步处理能力。在这次作业中，我对 Sanic 框架进行了深入的了解，也对 Python 异步编程有了更深刻的认识。除了 web 后端开发和异步编程，此次作业中涉及的数据处理、多种文本格式的实现以及其在 HTTP 协议中 Content-Type 与 MIME Type 的实现也使我收获良多。

总之，这次作业加深了我对后端开发和数据处理的理解和经验，是一次有意义的实践。我期待后续作业对本次项目的扩展，也期待将这次作业中获得的知识应用到未来的项目中。