STAT243 Lecture 2.4 Working with Information on the Web

Logic

- 以编程方式与 Web 交互：以下载为主，也可上传
- 常见格式：HTML, XML, JSON, YAML。
- 核心工具：`requests`、`BeautifulSoup` / `lxml`、`json`、`yaml`、`pandas.read_html` / `read_json`。

1 Reading HTML（抓取 HTML 表格）

- 关键思路：获取 HTML 源码 → 解析 DOM → 抽取 `<table>` 或节点文本。
- 浏览器中可先查看 **View Source / Developer Tools** 了解结构与是否存在大量 Javascript。

1.1 使用 `find_all` 通过 HTML 标签或属性搜索

Python

```python
import io, requests, pandas as pd
from bs4 import BeautifulSoup as bs

URL = "https://en.wikipedia.org/wiki/List_of_countries_and_dependencies_by_population"
user_agent = "stat243_educational_bot/0.1 (paciorek@berkeley.edu)"
headers = {'User-Agent': user_agent}
response = requests.get(URL, headers=headers)
html = response.content

soup = bs(html, 'html.parser')
html_tables = soup.find_all('table')

# pandas.read_html 需要字符串/文件句柄，不直接接受 Tag 对象
pd_tables = [pd.read_html(io.StringIO(str(tbl)))[0] for tbl in html_tables]
[x.shape for x in pd_tables]
# 输出:
# [(242, 6), (13, 2), (1, 2)]

pd_tables[0].head()
# 输出:
#    Location  ...  Notes
# 0     World  ...    NaN
# 1     India  ...    [b]
# ...
```

- `BeautifulSoup` 构建树后可按标签或属性搜索，再交给 `pandas.read_html` 解析。

1.2 Extracting hyperlinks（提取超链接）

Python

```python
import requests
from bs4 import BeautifulSoup as bs

URL = "http://www1.ncdc.noaa.gov/pub/data/ghcn/daily/by_year"
response = requests.get(URL)
soup = bs(response.content, 'html.parser')

# 方法 1：所有 <a> 标签
```

```python
a_elements = soup.find_all('a')
links1 = [x.get('href') for x in a_elements]

# 方法 2: 具有 href 属性的 <a>
href_elements = soup.find_all('a', href=True)
links2 = [x.get('href') for x in href_elements]

links2[:9]
# 输出:
# ['?C=N;O=D', '?C=M;O=A', '?C=S;O=A', '?C=D;O=A', '/pub/data/ghcn/daily/',
#  '1750.csv.gz', '1763.csv.gz', '1764.csv.gz', '1765.csv.gz']
```

1.3 CSS selectors（用选择器抽取）

Python

```python
# 所有 <tr> 内部的 <th>
soup.select("tr th")
# 输出:
# [<th>...</th>, <th>...</th>, ...]

# 所有父元素为 <th> 的 <a>
soup.select("th > a")
# 输出:
# [<a href="?C=N;O=D">Name</a>, <a href="?C=M;O=A">Last modified</a>, ...]
```

1.4 XPath（用 lxml 进行 XPath 查询）

Python

```python
import lxml.html

# 将 BeautifulSoup object 转为 lxml object
lxml_doc = lxml.html.fromstring(str(soup))

# 所有带 href 的 <a>
a_elements = lxml_doc.xpath('//a[@href]')
links = [x.get('href') for x in a_elements]
links[:9]
# 输出:
# ['?C=N;O=D', '?C=M;O=A', '?C=S;O=A', '?C=D;O=A', '/pub/data/ghcn/daily/',
#  '1750.csv.gz', '1763.csv.gz', '1764.csv.gz', '1765.csv.gz']
```

2 XML, JSON, and YAML

- 三者均支持键值、数组与层级结构
- 读取需用对应库解析为 Python 结构（dict/list 等）。

2.1 XML（结构化自描述）

- XML 是一种以自描述格式储存数据的 markup language, 通常有 hierarchical structure, **不需要 metadata**
- XML 文档具有**树状结构**, 由元素（节点）组成
- 常见存档/办公文档/空间信息（如 KML）。

XML

```xml
<?xml version="1.0"?>
<catalog>
   <book id="bk101">
```

```xml
        <author>Gambardella, Matthew</author>
        <title>XML Developer's Guide</title>
        <genre>Computer</genre>
        <price>44.95</price>
        <publish_date>2000-10-01</publish_date>
        <description>An in-depth look at creating applications with XML.</description>
    </book>
    <book id="bk102">
        <author>Ralls, Kim</author>
        <title>Midnight Rain</title>
        <genre>Fantasy</genre>
        <price>5.95</price>
        <publish_date>2000-12-16</publish_date>
      <description>A former architect battles corporate zombies, an evil sorceress, and her own childhood to become queen of the world.</description>
    </book>
</catalog>
```

Example

示例：Kiva 最新贷款数据（注意：在线接口可能返回 403，需本地保存演示）。我们采用两种方法:

1. 暴力解法 (将数据视作列表而不是树结构)
2. 使用 XPath 来遍历树结构

Python

```python
import xmltodict
# 假设 'newest.xml' 已手动下载（或用浏览器另存）
with open('newest.xml', 'r') as file:
    content = file.read()

# 一些 XML 含有裸 '&'，需先替换以避免解析错误
content = content.replace("&", "and")
data = xmltodict.parse(content)

data.keys()
# 输出:
# dict_keys(['response'])

data['response'].keys()
# 输出:
# dict_keys(['paging', 'loans'])

len(data['response']['loans']['loan'])
# 输出:
# 20

type(data['response']['loans']['loan'][2])
# 输出:
# dictionary

data['response']['loans']['loan'][2]['activity']
# 输出:
# 'Retail'
```

Python

```python
# 同样可用 lxml + XPath 提取指定字段
```

```python
from lxml import etree

doc = etree.fromstring(content)
loans = doc.xpath("//loan")
[loan.xpath("activity/text()") for loan in loans][:3]
# 输出:
# [['Poultry'], ['Retail'], ['Retail']]

## 假设只想要 country locations of the loans (using XPath)
[loan.xpath("location/country/text()") for loan in loans]
# 输出:
# [['Uganda'], ['Ecuador'], ['Ecuador'] ...]

## or extract the geographic coordinates
[loan.xpath("location/geo/pairs/text()") for loan in loans]
# 输出:
# [['-0.352537 31.552699'], ['-1.054723 -80.45249'] ...]
```

2.2 JSON（层级键值，较 XML 简洁）

- JSON 文件以 **“attribute-value” pairs**（也称为“键-值”对）结构化，通常具有层次结构
- 可以使用 `json` 包将 JSON 读取到 Python 中
- JSON 的最外层可以是 **对象（object，用花括号 `{}`）** 或 **数组（array，用方括号 `[]`）**。如果最外层是数组，那么就没有键（key）去命名它的元素。

{ } JSON

```json
{
    "firstName": "John",
    "lastName": "Smith",
    "isAlive": true,
    "age": 25,
    "address": {
        "streetAddress": "21 2nd Street",
        "city": "New York",
        "state": "NY",
        "postalCode": "10021-3100"
    },
    "phoneNumbers": [
        { "type": "home", "number": "212 555-1234" },
        { "type": "office", "number": "646 555-4567" }
    ],
    "children": [],
    "spouse": null
}
```

Python

```python
import json
# 假设 'newest.json' 已手动下载
with open('newest.json', 'r') as file:
    content = file.read()

data = json.loads(content)
list(data.keys())
# 输出:
# ['loans']

type(data['loans']), data['loans'][0]['location']['country']
# 输出:
```

```
# (<class 'list'>, 'Uganda')

[c['location']['country'] for c in data['loans']][:5]
# 输出:
# ['Uganda', 'Ecuador', 'Ecuador', 'Tajikistan', 'Mali']
```

⚠ **Remark**

注意：JSON 不原生支持缺失、无穷大等特殊值。

2.3 YAML（常用于配置）

- 以缩进表达层级，人类可读；缩进易出错；部分关键字（如 `on`）在某些实现中会被当作布尔。

YAML

```yaml
name: deploy-book

# Only run this when the master branch changes
on:
  push:
    branches:
    - main

# This job installs dependencies, build the book, and pushes it to `gh-pages`
jobs:
  deploy-book:
    runs-on: ubuntu-latest
    steps:
    - uses: actions/checkout@v2

    # Install dependencies
    - name: Set up Python 3.9
      uses: actions/setup-python@v1
      with:
        python-version: 3.9

    - name: Install dependencies
      run: |
        pip install -r book-requirements.txt
```

Python

```python
import yaml
with open("book.yml") as stream:
    config = yaml.safe_load(stream)
print(config)
# 输出:
# {'name': 'deploy-book', True: {'push': {'branches': ['main']}}, 'jobs': {'deploy-book':
...}}

print(config.get('name'))
# 输出:
# deploy-book

len(config['jobs']['deploy-book']['steps'])
# 输出:
# 3
```

Remark

注意 `on` 会被视作 boolean value

3 Web APIs and webscraping

- 目标：获取 Web 数据时，优先使用正式 API；当 API 不可用时，再考虑 webscraping，并遵守网站条款与伦理。
- 常用库：`requests`, `json`, `pandas`, `BeautifulSoup`, `lxml`, `yaml`。
- 核心操作路径：理解 HTTP → REST 风格 API → 参数拼接与分页 → 处理压缩与归档 → POST 与认证 → 第三方封装 → 动态页面。

3.1 What is HTTP?

- **请求方法**：常用 GET、POST、PUT、DELETE，实际数据提取以 GET 为主。
- **状态码**：200 成功，4xx 客户端错误（如 403/404），5xx 服务器错误。
- **URL 查询字符串**：`?` 之后为参数，`&` 分隔键值对，空格常编码为 `+` 或 `%20`。

Example

1. `www.somewebsite.com?param1=arg1&param2=arg2`
2. `https://www.yelp.com/search?find_desc=plumbers&find_loc=Berkeley+CA&ns=1`

- **响应内容**: 通常包含文本形式的内容（例如，HTML、XML、JSON）或原始字节

Python

```python
import requests

url = "https://httpbin.org/status/200"
r = requests.get(url)
print(r.status_code)
# 输出:
# 200
```

3.2 APIs: REST-based web services

Logic

- 理想情况下，一个网络服务会用提供其 API (Application Programming Interface) 文档，该接口用于提供数据或允许其他交互
- REST 是一种流行的 API 标准/风格

- **资源导向**：以 URL (也叫 endpoint) 作为资源，通过 query string 过滤，通过 `GET` 实现 request, 常返回 JSON。
- **分页与每页条数**：注意 `page`, `per_page`, `limit`, `offset` 等字段。
- **两种构造 request 的方式**：
  1. 直接拼接查询字符串。
  2. 使用 `params` 以 dict 传参。

Python

```python
import json, requests

# 方法一: 直接拼接查询字符串（World Bank 示例）
url = "https://api.worldbank.org/V2/country?incomeLevel=MIC&format=json"
resp = requests.get(url)
data = json.loads(resp.content)
```

```python

## 注意 data truncation/pagination
if False:
    url = "https://api.worldbank.org/V2/country?incomeLevel=MIC&format=json&per_page=1000"
    response = requests.get(url)
    data = json.loads(response.content)

# 方法二: Programmatic control
baseURL = "https://api.worldbank.org/V2/country"
group = 'MIC'
format = 'json'
args = {'incomeLevel': group, 'format': format, 'per_page': 1000}
url = baseURL + '?' +
    '&'.join(['='.join([key, str(args[key])])
            for key in args])
response = requests.get(url)
data = json.loads(response.content)

# 方法三: params 传参 (更稳妥)
baseURL = "https://api.worldbank.org/V2/country"
params = {"incomeLevel": "MIC", "format": "json", "per_page": 1000}
resp = requests.get(baseURL, params=params)
data = resp.json()

print(type(data), len(data))
# 输出:
# <class 'list'> 2

print(len(data[1]), isinstance(data[1][5], dict), data[1][5]['name'])
# 输出:
# 104 True Benin
```

Python

```python
# 简单的分页遍历 (若 API 需要翻页)
all_rows = []
page = 1
while True:
    params = {"incomeLevel": "MIC", "format": "json", "per_page": 100, "page": page}
    resp = requests.get(baseURL, params=params)
    data = resp.json()
    rows = data[1]
    if not rows:
        break
    all_rows.extend(rows)
    page += 1

print(len(all_rows) >= 100)
# 输出:
# True
```

3.3 HTTP requests by deconstructing an (undocumented) API

- **思路**: 用浏览器 DevTools 的 Network 面板观察实际请求 (URL、headers、查询参数、cookies), 在代码中复现。
- **典型场景**: 下载链接返回 zip 压缩, 需内存解压后再读取 CSV。

Python

```python
import io, zipfile, requests, pandas as pd

```

```python
itemCode = 526
baseURL = "https://data.un.org/Handlers/DownloadHandler.ashx"
yrs = ",".join(str(yr) for yr in range(2012, 2018))
filter_ = f"?DataFilter=itemCode:{itemCode};year:{yrs}"
args1 = "&DataMartId=FAO&Format=csv&c=2,3,4,5,6,7&"
args2 = "s=countryName:asc,elementCode:asc,year:desc"
url = baseURL + filter_ + args1 + args2

resp = requests.get(url)
# 把 zip 文件放在内存中，而不是保存成 .zip 文件
with io.BytesIO(resp.content) as stream:
    # 在内存中打开这个 zip 文件
    with zipfile.ZipFile(stream, "r") as archive:
        # 从压缩包中读取第一个文件，并用 pandas 加载
        name = archive.filelist[0].filename
        with archive.open(name, "r") as f:
            dat = pd.read_csv(f)

print(dat.head(2))
# 输出:
#   Country or Area  Element Code  ...   Value  Value Footnotes
# 0     Afghanistan           432  ...  202.19              NaN
# 1     Afghanistan           432  ...   27.45              NaN
```

3.4 Webscraping ethics and best practices

- **是否应该抓**：优先使用公开下载文件或正式 API；抓取是下策。
- **是否允许抓**：遵守网站条款与 `robots.txt`；尊重速率限制与版权、隐私。
- **实践建议**：请求加 `User-Agent`，缓存响应避免重复请求；对高频请求使用 `time.sleep`；谨慎处理认证信息。

Python

```python
import time, requests

headers = {"User-Agent": "stat243_educational_bot/0.1 (contact@example.com)"}
for page in range(1, 4):
    r = requests.get("https://httpbin.org/get", params={"page": page}, headers=headers, timeout=10)
    print(r.status_code, r.json()["args"]["page"])
    # 输出:
    # 200 1
    # 200 2
    # 200 3
    time.sleep(1)  # 友好限速
```

3.5 More details on HTTP requests

- **结构化参数**：`params` 适合 GET，避免手写拼接错误。
- **复杂下载**：大文件/二进制内容可用 `stream=True` 分块下载。
- **错误处理**：`response.raise_for_status()`，或根据 `status_code` 分支处理。

Python

```python
# 分块下载（示意）
import requests

url = "https://speed.hetzner.de/100MB.bin"
with requests.get(url, stream=True) as r:
    r.raise_for_status()
    total = 0
    for chunk in r.iter_content(chunk_size=8192):
        if chunk:
            total += len(chunk)
print(total > 0)
# 输出:
# True
```

3.6 POST example（创建 GitHub issue）

- **说明**：需个人 access token；最小权限原则；切勿把 token 提交到仓库。
- **两种方式**：裸 `requests.post` 与封装库 `PyGitHub`。

Python

```python
import requests

with open(".github-access-token.txt", "r") as file:
    ghtoken = file.read().strip()

owner, repo = "paciorek", "test"
url = f"https://api.github.com/repos/{owner}/{repo}/issues"
issue = {
    "title": "This is an example issue",
    "body": "This is the body of the issue created via API."
}
headers = {
    "Authorization": f"token {ghtoken}",
    "Accept": "application/vnd.github+json"
}
resp = requests.post(url, json=issue, headers=headers)
print(resp.status_code in (200, 201))
# 输出:
# True
```

Python

```python
# 使用 PyGitHub 封装
from github import Github

with open(".github-access-token.txt", "r") as file:
    ghtoken = file.read().strip()

g = Github(ghtoken)
repo = g.get_repo("paciorek/test")
issue = repo.create_issue(
    title="Test Issue Created Programmatically",
    body="This is an issue filed programmatically using PyGitHub."
)
print(f"#{issue.number}", issue.html_url)
# 输出:
# #18 https://github.com/paciorek/test/issues/18
g.close()
```

3.7 Accessing dynamic pages

- **适用场景**：内容需 Javascript 渲染或需要模拟用户交互。
- **方案**：`selenium` 驱动浏览器；或 `scrapy` 框架配合 `splash` 渲染。

- **注意**：渲染成本高、速率慢，更应重视限速、重试与缓存；若站点有公开接口，优先 API。