2024-09-24

Обзор библиотек для сбора и анализа веб-данных средствами Python (завершение)

MechanicalSoup

Документация: Welcome to MechanicalSoup's documentation! — MechanicalSoup 1.3.0 documentation

Исходники: GitHub - MechanicalSoup/MechanicalSoup: A Python library for automating interaction with websites.

Появилась как переосмысление библиотеки Mechanize (GitHub - python-mechanize/mechanize: The official source code for the python-mechanize project) — в свое время эта библиотека была несовместима с 3-м Питоном + страдала от медленного темпа разработки (но в 2017-м году разработка продолжилась, поэтому обе библиотеки сейчас существуют параллельно)

- построена на базе `requests` и `bs4`
- основной целью библиотеки является симуляция пользовательского ввода на уровне взаимодействия с экранными формами
- !НО: имеет те же ограничения, что и `bs4` — сайты, построенные на клиентских фреймворках (React, Vue, Svelte, Angular и т.д.), нормально библиотекой обработаны не будут
- по сути скрещивает функции `requests` и `bs4`, поэтому в определенных сценариях может вовсе быть избыточной
- как следствие, сценарии применения (по субъективной оценке) в настоящее время весьма ограничены: использование автоматического взаимодействия с сайтами, которые все еще отдают полноценные HTML-документы и не зависят от исполнения JS-кода на клиенте

Пример: логин на гитхаб

```
"""Example app to login to GitHub, using the plain Browser class.

See example.py for an example using the more advanced
StatefulBrowser."""
import argparse

import mechanicalsoup

parser = argparse.ArgumentParser(description="Login to GitHub.")
parser.add_argument("username")
parser.add_argument("password")
args = parser.parse_args()

browser = mechanicalsoup.Browser(soup_config={'features': 'lxml'})

# request github login page. the result is a requests.Response object
# http://docs.python-requests.org/en/latest/user/quickstart/#response-
```

```
content
login_page = browser.get("https://github.com/login")

# similar to assert login_page.ok but with full status code in case of
# failure.
login_page.raise_for_status()

# login_page.soup is a BeautifulSoup object
# http://www.crummy.com/software/BeautifulSoup/bs4/doc/#beautifulsoup
# we grab the login form
login_form = mechanicalsoup.Form(login_page.soup.select_one('#login
form'))

# specify username and password
login_form.input({"login": args.username, "password": args.password})

# submit form
page2 = browser.submit(login_form, login_page.url)

# verify we are now logged in
messages = page2.soup.find("div", class_="flash-messages")
if messages:
    print(messages.text)
assert page2.soup.select(".logout-form")

print(page2.soup.title.text)

# verify we remain logged in (thanks to cookies) as we browse the rest
of
# the site
page3 = browser.get("https://github.com/MechanicalSoup/MechanicalSoup")
assert page3.soup.select(".logout-form")
```

Scrapy

Сайт: Scrapy | A Fast and Powerful Scraping and Web Crawling Framework

Документация: Scrapy 2.11 documentation — Scrapy 2.11.2 documentation

Исходный код: GitHub - scrapy/scrapy: Scrapy, a fast high-level web crawling & scraping framework for Python.

- высокоуровневый скраппер, не требующий специальных навыков веб-скраппинга/парсинга
- основная аудитория — дата-аналитики, которые используют его для быстрого и относительно простого сбора данных с интересующих веб-сайтов без необходимости погружения в Python
- использует концепт «пауков» (spiders) — название напрямую отсылает к роботам-краулерам поисковых движков
- использует механизмы современного Питона, поэтому много «синтаксического сахара», достаточно производителен и легко адаптировать к современным паттернам, например, асинхронным корутинам.
- заточен (сразу) под коллаборативную аналитику
- может работать как самостоятельное консольное приложение
- доступен в преднастроенных средах Питона, например, Conda (Anaconda/Miniconda и т.п.)
- !НО: все еще не умеет работать с реактивными фреймворками

```
from pathlib import Path

import scrapy

class QuotesSpider(scrapy.Spider):
```

```
    name = "quotes"

    def start_requests(self):
        urls = [
            "https://quotes.toscrape.com/page/1/",
            "https://quotes.toscrape.com/page/2/",
        ]
        for url in urls:
            yield scrapy.Request(url=url, callback=self.parse)

    def parse(self, response):
        page = response.url.split("/")[-2]
        filename = f"quotes-{page}.html"
        Path(filename).write_bytes(response.body)
        self.log(f"Saved file {filename}")

class BlogSpider(scrapy.Spider):
    name = 'blogspider'
    start_urls = ['https://www.zyte.com/blog/']

    def parse(self, response):
        for title in response.css('.oxy-post-title'):
            yield {'title': title.css('::text').get()}

        for next_page in response.css('a.next'):
            yield response.follow(next_page, self.parse)
```

Selenium (for Python)

PyPi: https://pypi.org/project/selenium/

Документацию: The Selenium Browser Automation Project | Selenium (см. примеры на Питоне); Selenium Client Driver — Selenium 4.25.0 documentation

Исходный код: GitHub - SeleniumHQ/selenium: A browser automation framework and ecosystem.

- автоматизация полноценных браузеров
- требует для работы webdriver, который и управляет установленным в системе браузером (в системе браузер должен быть предварительно установлен)

    - для браузеров на основе Хромиума: Обзор ChromeDriver | Chrome for Developers
    - для Firefox: Releases · mozilla/geckodriver · GitHub
- единственный инструмент, позволяющий полноценно автоматически взаимодействовать с сайтами с загрузкой содержимого на клиенте, т.к. Selenium использует полноценный JavaScript-движок браузера при выполнении запроса

Для удобного управления вебдрайверами есть инструмент GitHub - SergeyPirogov/webdriver_manager (Python) / https://github.com/bonigarcia/webdrivermanager (Java)

```
# selenium 4 + Brave
from selenium import webdriver
from selenium.webdriver.chrome.service import Service as BraveService
from webdriver_manager.chrome import ChromeDriverManager
from webdriver_manager.core.os_manager import ChromeType

driver =
webdriver.Chrome(service=BraveService(ChromeDriverManager(chrome_type=Ch
romeType.BRAVE).install()))
```

```
# selenium 4 + Firefox
from selenium import webdriver
from selenium.webdriver.firefox.service import Service as FirefoxService
from webdriver_manager.firefox import GeckoDriverManager

driver =
webdriver.Firefox(service=FirefoxService(GeckoDriverManager().install())
)
```

PyFlink

Сам Flink (часть экосистемы Apache для построения ETL-пайплайнов и решения задач по сбору и анализу данных)

Apache Flink Documentation | Apache Flink

DataStream Overview | Apache Flink — ключевая технология

PyFlink — официальные связки для Flink и Python

Python Overview | Apache Flink

Примеры:

flink/flink-python/pyflink/examples at master · apache/flink · GitHub

информация дополнительная по запросу «Apache Flink Streaming in Python»

Лабораторная работа № 1

- API:Tutorial - MediaWiki
- Wikimedia API Portal

Нужно собрать данные о предметной области с ресурсов Wikimedia: Википедии, Викидаты, Wikimedia Commons и иных (если данные на них есть). Если берете с Википедий на других языках, то нужно автоматически перевести на русский.

Требуется написать автоматизированный сборщик данных на Python, на вход которому подается запрос о предметной области, на выход — JSON-структура, содержащая основной контент узла (текст статьи, записи; изображение и т.п.) и связанные с ним узлы (т.е. гиперссылки на другие статьи, записи, изображения и т.п.).

Из полученных данных на основе гиперссылок и иных явно формализованных связей (например, категории в движке Википедии) составить граф статей на языке DOT (каждый узел — статья; ребро — связь между статьями)

Итогом должен стать граф на языке DOT. Минимум 30 различных статей должно быть связано.

Example

Статья 1: Чёрная металлургия — Википедия

Статья 2: Ферросплавы — Википедия

`"Чёрная металлургия" -> "Ферросплавы"`, т.к. на странице «Черная металлургия» есть ссылка на статью «Ферросплавы»

`"Чёрная металлургия" - "Бурый дым"`, т.к. на странице «Черная металлургия» есть ссылка на категорию «Черная металлургия», в которую также входит статья «Бурый дым»

DOT (язык) — Википедия

Для этого понадобится:

- `requests`
- `bs4`
- `graphviz` (для просмотра и отладки графа)
- возможно, `pymorphy3` (для поиска ключевых слов по их нормальной форме)