2025-09-22

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

Фреймворки для веб-скрапинга

MechanicalSoup

Документация: MechanicalSoup 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 — server-side rendering на фреймворках а-ля Laravel

Пример логина на GitHub

```
"""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

Документация: Scrapy 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 JS и client-side rendering**

```
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)
```

Crawlee

crawlee

Документация: Quick start | Crawlee for Python · Fast, reliable Python web crawlers.

Исходный код: GitHub - apify/crawlee: Crawlee—A web scraping and browser automation library for Node.js to build reliable crawlers. In JavaScript and TypeScript. Extract data for AI, LLMs, RAG, or GPTs. Download HTML, PDF, JPG, PNG, and other files from websites. Works with Puppeteer, Playwright, Cheerio, JSDOM, and raw HTTP. Both headful and headless mode. With proxy rotation.

- современная **async-first** библиотека для веб-скрапинга и автоматизации
- единый интерфейс для HTTP-запросов и браузерной автоматизации
- автоматический роутинг прокси и управление сессиями

```
import asyncio

from crawlee.crawlers import BeautifulSoupCrawler,
BeautifulSoupCrawlingContext
```

```
async def main() -> None:
    # BeautifulSoupCrawler crawls the web using HTTP requests
    # and parses HTML using the BeautifulSoup library.
    crawler = BeautifulSoupCrawler(max_requests_per_crawl=10)

    # Define a request handler to process each crawled page
    # and attach it to the crawler using a decorator.
    @crawler.router.default_handler
    async def request_handler(context: BeautifulSoupCrawlingContext) ->
None:
        context.log.info(f'Processing {context.request.url} ...')
        # Extract relevant data from the page context.
        data = {
            'url': context.request.url,
            'title': context.soup.title.string if context.soup.title else
None,
        }
        # Store the extracted data.
        await context.push_data(data)
        # Extract links from the current page and add them to the crawling
queue.
        await context.enqueue_links()

    # Add first URL to the queue and start the crawl.
    await crawler.run(['https://crawlee.dev'])


if __name__ == '__main__':
    asyncio.run(main())
```

```
import asyncio

from crawlee.crawlers import PlaywrightCrawler, PlaywrightCrawlingContext


async def main() -> None:
    # PlaywrightCrawler crawls the web using a headless browser
    # controlled by the Playwright library.
    crawler = PlaywrightCrawler()

    # Define a request handler to process each crawled page
    # and attach it to the crawler using a decorator.
    @crawler.router.default_handler
    async def request_handler(context: PlaywrightCrawlingContext) -> None:
        context.log.info(f'Processing {context.request.url} ...')
        # Extract relevant data from the page context.
        data = {
```

```
            'url': context.request.url,
            'title': await context.page.title(),
        }
        # Store the extracted data.
        await context.push_data(data)
        # Extract links from the current page and add them to the crawling
queue.
        await context.enqueue_links()

    # Add first URL to the queue and start the crawl.
    await crawler.run(['https://crawlee.dev'])


if __name__ == '__main__':
    asyncio.run(main())
```

Средства браузерной автоматизации

Документация: Selenium Documentation

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

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
- инструмент, позволяющий полноценно автоматически взаимодействовать с сайтами с загрузкой содержимого на клиенте, т.к. Selenium использует

полноценный JavaScript-движок браузера при выполнении запроса

```
# Для 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=ChromeType.BRAVE).install()
))

# Для Firefox
from selenium.webdriver.firefox.service import Service as FirefoxService
from webdriver_manager.firefox import GeckoDriverManager

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

Для удобного управления вебдрайверами есть инструмент GitHub - SergeyPirogov/webdriver_manager (Python) / https://github.com/bonigarcia/webdrivermanager (Java) — **пользуйтесь с осторожностью, может тянуть старые версии, ставить WebDriver лучше вручную**

Playwright

Документация: Playwright Python

Исходный код: GitHub - microsoft/playwright-python: Python version of the Playwright testing and automation library.

- Современная кроссплатформенная автоматизация браузеров
- Поддержка Chromium, Firefox и WebKit
- Мощные возможности автоматизации и эмуляции устройств

```
from playwright.sync_api import sync_playwright

with sync_playwright() as p:
    for browser_type in [p.chromium, p.firefox, p.webkit]:
        browser = browser_type.launch()
        page = browser.new_page()
        page.goto('http://playwright.dev')
        page.screenshot(path=f'example-{browser_type.name}.png')
        browser.close()
```

```python
import asyncio
from playwright.async_api import async_playwright

async def main():
    async with async_playwright() as p:
        for browser_type in [p.chromium, p.firefox, p.webkit]:
            browser = await browser_type.launch()
            page = await browser.new_page()
            await page.goto('http://playwright.dev')
            await page.screenshot(path=f'example-{browser_type.name}.png')
            await browser.close()

asyncio.run(main())
```

Вне конкурса: 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»

Выбор инструментов

Задача	Инструмент
Парсинг статических сайтов (SSR)	httpx + BeautifulSoup + lxml
Динамические сайты с client-side rendering и кучей JS	**Playwright** / Selenium
Параллельный / масштабный скрапинг	**Scrapy** / Crawlee
Статические сайты с традиционными формами	MechanicalSoup
ETL-процедуры и/или экосисистема Apache	PyFlink
Продвинутый скрапинг с обходами защит (анти-бот и т.п.) с перспективой коммерциализации	Zenrows + роутинг прокси

Практика 2

Разработать систему сбора данных с современных веб-сайтов, которые требуют различных подходов к извлечению информации:

1. создать Scrapy-проект для сбора структурированных данных с сайтов с классической архитектурой
2. интегрировать Playwright в Scrapy для обработки динамического контента
3. реализовать минимум 2 различных паука: для новостного портала с SSR, для динамического сайта, например, e-commerce
4. каждый паук должен собрать не менее 100 записей
5. необходимо реализовать также обработку ошибок и повторные попытки

Должна быть поддержка:

- статических сайтов (через классические Scrapy Spider)
- динамические сайты и SPA-приложения (через Playwright)

Попробовать еще добавить сайты с анти-бот защитой (через эмуляцию пользовательского поведения), например, через сервисы типа Anti-Captcha

Что использовать:

- **Scrapy** — фреймворк для масштабируемого веб-скрапинга
- **Playwright** — инструмент для автоматизации современных браузеров
- **Scrapy-Playwright** — интеграция двух технологий

Опционально **Redis** для распределенной очереди задач и **Docker** для управления зависимостями

Выгрузку сделать в JSON и CSV

Пример запуска:

```
python ./run_spider.py --spider news --keywords "ПНИПУ" --days 7 --output
news_pstu.json
python ./run_spider.py --spider ecommerce --category "ноутбуки" --stores
"wildberries,ozon" --output prices.csv
```