scrapex Documentation

Release 0.1.2

Cung Nguyen

Dec 09, 2022

Contents

Scrapex is a simple web scraping framework. Built on top of requests and lxml, supports Python 2 and Python 3.

```
>>> from scrapex import Scraper
>>> s = Scraper(use_cache = True)
>>> doc = s.load('https://github.com/search?q=scraping')
>>>
>>> print(doc.extract("//h3[contains(text(),'results')]").strip())
59,256 repository results
>>>
>>> listings = doc.query("//ul[@class='repo-list']/li")
>>> print('number of listings on first page:', len(listings) )
number of listings on first page: 10
>>>
>>> for listing in listings[0:3]:
...     print('repo name: ',listing.extract(".//div[contains(@class,'text-normal')]/a
↪"))
...
repo name:  scrapinghub/portia
repo name:  scrapy/scrapy
repo name:  REMitchell/python-scraping
>>>
```

CHAPTER 1

Key Features

- Easy to parse data points from html documents using XPATH, regular expression, or string subtraction
- Easy to parse street address into components (street, city, state, zip code)
- Easy to parse person name into components (prefix, first name, middle name, last name, suffix)
- Easy to save results to CSV, Excel, JSON files
- **HTML cache:**
  - quickly response to any parsing mistakes/changes without having to send HTTP rquests again.
  - easy to resume a broken scrape without having to re-download the downloaded html files.
- Ramdom proxy rotation
- Random user agent rotation

1.1 Installation

To install stable version

```
pip install scrapex
```

To install development version

```
pip install https://github.com/cungnv/scrapex/archive/master.zip
```

1.2 Example scraping script

A complete scraping script to scrape Github search result pages. View the source file.

```
#encoding: utf8

import sys
import logging
from collections import OrderedDict

from scrapex import Scraper
from scrapex import common


logging.basicConfig(
    stream=sys.stdout,
    level=logging.INFO,
    format='%(levelname)s %(message)s')

logger = logging.getLogger(__name__)


#create the scraper object
#enable the cache system so that we can re-run the scrape without re-downloading the
↪html files

s = Scraper(use_cache=True, greeting=True)


def scrape():
    """
    - scrape repos from this search (first 5 result pages):
    https://github.com/search?q=scraping%20framework

    - results are save into an excel file: results.xlsx

    """

    url = 'https://github.com/search?q=scraping%20framework'

    page = 0

    while True:
        page += 1

        doc = s.load(url)

        if page==1:
            logger.info(doc.extract("//h3[contains(text(),'results')]").strip())


        listings = doc.query("//ul[@class='repo-list']/li")
        logger.info('page# %s >> %s listings', page, len(listings))

        for listing in listings:
            item = OrderedDict()

            item['name'] = listing.extract(".//div[contains(@class,'text-normal')]/a")

            item['description'] = listing.extract(".//div[@class='mt-n1']/p").rr('\s+
↪',' ').strip()
```

(continues on next page)

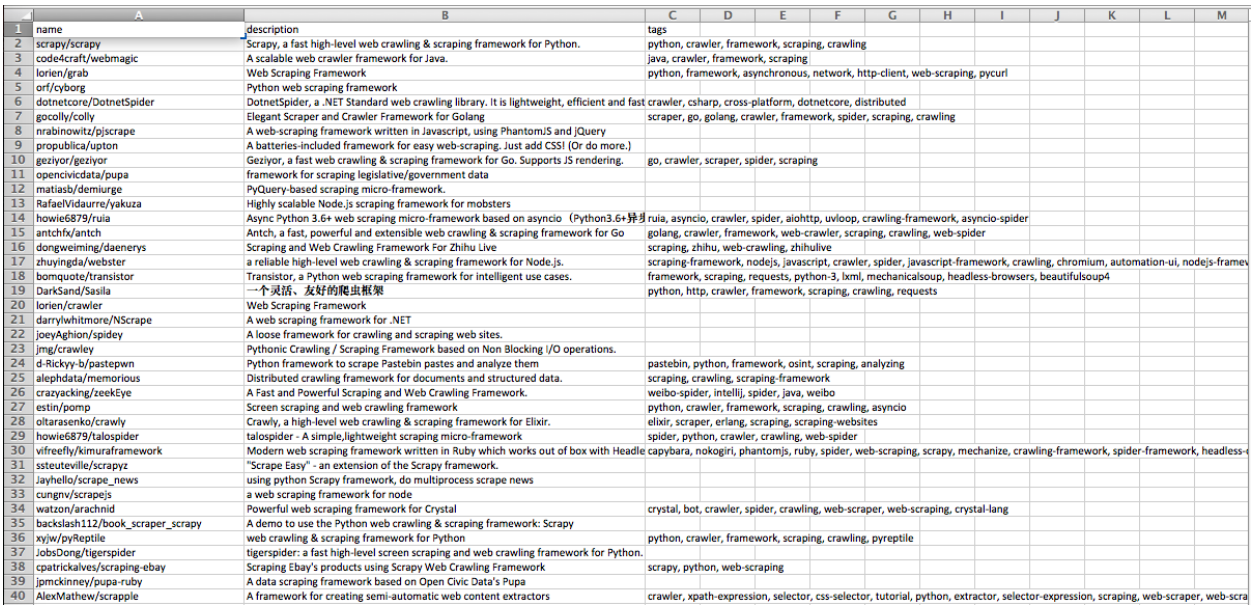
(continued from previous page)

```
            item['tags'] = listing.query(".//a[contains(@class,'topic-tag')]").join(',
↪ ')

            #save this item to excel file

            s.save(item,'results.xlsx')



        #find the url of the next result page

        url = doc.x("//a[.='Next']/@href")

        if url:
            logger.info('next page url: %s', url)
        else:
            logger.info('last page reached')

        if page == 5:
            break

if __name__ == '__main__':
    scrape()
```

Output screen:

```
CungMac:demo cung$ python example-script.py
scrape started
INFO 443 repository results
INFO page# 1 >> 10 listings
INFO next page url: https://github.com/search?p=2&q=scraping+framework&
↪type=Repositories
INFO page# 2 >> 10 listings
INFO next page url: https://github.com/search?p=3&q=scraping+framework&
↪type=Repositories
INFO page# 3 >> 10 listings
INFO next page url: https://github.com/search?p=4&q=scraping+framework&
↪type=Repositories
INFO page# 4 >> 10 listings
INFO next page url: https://github.com/search?p=5&q=scraping+framework&
↪type=Repositories
INFO page# 5 >> 10 listings
INFO next page url: https://github.com/search?p=6&q=scraping+framework&
↪type=Repositories
scrape finished
```

The results file screenshot:



	name	description	tags
2	scrapy/scrapy	Scrapy, a fast high-level web crawling & scraping framework for Python.	python, crawler, framework, scraping, crawling
3	code4craft/webmagic	A scalable web crawler framework for Java.	java, crawler, framework, scraping
4	lorien/grab	Web Scraping Framework	python, framework, asynchronous, network, http-client, web-scraping, pycurl
5	orf/cyborg	Python web scraping framework	
6	dotnetcore/DotnetSpider	DotnetSpider, a .NET Standard web crawling library. It is lightweight, efficient and fast	crawler, csharp, cross-platform, dotnetcore, distributed
7	gocolly/colly	Elegant Scraper and Crawler Framework for Golang	scraper, go, golang, crawler, framework, spider, scraping, crawling
8	nrabinowitz/pjscrape	A web-scraping framework written in Javascript, using PhantomJS and jQuery	
9	propublica/upton	A batteries-included framework for easy web-scraping. Just add CSS! (Or do more.)	
10	geziyor/geziyor	Geziyor, a fast web crawling & scraping framework for Go. Supports JS rendering.	go, crawler, scraper, spider, scraping
11	opencivicdata/pupa	framework for scraping legislative/government data	
12	matiasb/demiurge	PyQuery-based scraping micro-framework.	
13	RafaelVidaurre/yakuza	Highly scalable Node.js scraping framework for mobsters	
14	howie6879/ruia	Async Python 3.6+ web scraping micro-framework based on asyncio（Python3.6+异步	ruia, asyncio, crawler, spider, aiohttp, uvloop, crawling-framework, asyncio-spider
15	antchfx/antch	Antch, a fast, powerful and extensible web crawling & scraping framework for Go	golang, crawler, framework, web-crawler, scraping, crawling, web-spider
16	dongweiming/daenerys	Scraping and Web Crawling Framework For Zhihu Live	scraping, zhihu, web-crawling, zhihulive
17	zhuyingda/webster	a reliable high-level web crawling & scraping framework for Node.js.	scraping-framework, nodejs, javascript, crawler, spider, javascript-framework, crawling, chromium, automation-ui, nodejs-framev
18	bomquote/transistor	Transistor, a Python web scraping framework for intelligent use cases.	framework, scraping, requests, python-3, lxml, mechanicalsoup, headless-browsers, beautifulsoup4
19	DarkSand/Sasila	一个灵活、友好的爬虫框架	python, http, crawler, framework, scraping, crawling, requests
20	lorien/crawler	Web Scraping Framework	
21	darrylwhitmore/NScrape	A web scraping framework for .NET	
22	joeyAghion/spidey	A loose framework for crawling and scraping web sites.	
23	jmg/crawley	Pythonic Crawling / Scraping Framework based on Non Blocking I/O operations.	
24	d-Rickyy-b/pastepwn	Python framework to scrape Pastebin pastes and analyze them	pastebin, python, framework, osint, scraping, analyzing
25	alephdata/memorious	Distributed crawling framework for documents and structured data.	scraping, crawling, scraping-framework
26	crazyacking/zeekEye	A Fast and Powerful Scraping and Web Crawling Framework.	weibo-spider, intellij, spider, java, weibo
27	estin/pomp	Screen scraping and web crawling framework	python, crawler, framework, scraping, crawling, asyncio
28	oltarasenko/crawly	Crawly, a high-level web crawling & scraping framework for Elixir.	elixir, scraper, erlang, scraping, scraping-websites
29	howie6879/talospider	talospider - A simple,lightweight scraping micro-framework	spider, python, crawler, crawling, web-spider
30	vifreefly/kimuraframework	Modern web scraping framework written in Ruby which works out of box with Headle	capybara, nokogiri, phantomjs, ruby, spider, web-scraping, scrapy, mechanize, crawling-framework, spider-framework, headless-
31	ssteuteville/scrapyz	"Scrape Easy" - an extension of the Scrapy framework.	
32	Jayhello/scrape_news	using python Scrapy framework, do multiprocess scrape news	
33	cungnv/scrapejs	a web scraping framework for node	
34	watzon/arachnid	Powerful web scraping framework for Crystal	crystal, bot, crawler, spider, crawling, web-scraper, web-scraping, crystal-lang
35	backslash112/book_scraper_scrapy	A demo to use the Python web crawling & scraping framework: Scrapy	
36	xyjw/pyReptile	web crawling & scraping framework for Python	python, crawler, framework, scraping, crawling, pyreptile
37	JobsDong/tigerspider	tigerspider: a fast high-level screen scraping and web crawling framework for Python.	
38	cpatrickalves/scraping-ebay	Scraping Ebay's products using Scrapy Web Crawling Framework	scrapy, python, web-scraping
39	jpmckinney/pupa-ruby	A data scraping framework based on Open Civic Data's Pupa	
40	AlexMathew/scrapple	A framework for creating semi-automatic web content extractors	crawler, xpath-expression, selector, css-selector, tutorial, python, extractor, selector-expression, scraping, web-scraper, web-scra

1.3 Make HTTP requests

Send GET requests

```
>>> from scrapex import Scraper
>>> s = Scraper()
>>> doc = s.load('https://github.com/search?q=scraping')
>>> print(doc.response.status_code)
200
>>> print(doc.response.request.headers)
{'Accept': 'text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8', 'User-
↪Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like␣
↪Gecko) Chrome/79.0.3945.79 Safari/537.36', 'Accept-Language': 'en-us,en;q=0.5',
↪'Accept-Encoding': 'gzip, deflate', 'Connection': 'keep-alive'}
>>>
```

Send POST requests

```
>>> doc = s.load('http://httpbin.org/post', data={'name1':'value1', 'name2':'value2'})
>>> doc.response.json()
{'args': {}, 'data': '', 'files': {}, 'form': {'name1': 'value1', 'name2': 'value2'},
↪'headers': {'Accept': 'text/html,application/xhtml+xml,application/xml;q=0.9,*/*;
↪q=0.8', 'Accept-Encoding': 'gzip, deflate', 'Accept-Language': 'en-us,en;q=0.5',
↪'Content-Length': '25', 'Content-Type': 'application/x-www-form-urlencoded', 'Host
↪': 'httpbin.org', 'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64)␣
↪AppleWebKit/537.36 (KHTML, like Gecko) Chrome/79.0.3945.88 Safari/537.36', 'X-Amzn-
↪Trace-Id': 'Root=1-5e42296f-72e27fb04c47860421601594'}, 'json': None, 'origin': '42.
↪114.13.13', 'url': 'http://httpbin.org/post'}
>>>
>>> doc = s.load('http://httpbin.org/post', data='name1=value1&name2=value2')
>>> doc.response.json()
{'args': {}, 'data': 'name1=value1&name2=value2', 'files': {}, 'form': {}, 'headers':
↪{'Accept': 'text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8',
↪'Accept-Encoding': 'gzip, deflate', 'Accept-Language': 'en-us,en;q=0.5', 'Content-
↪Length': '25', 'Host': 'httpbin.org', 'User-Agent': 'Mozilla/5.0 (Macintosh; Intel␣
↪Mac OS X 10_15_1) AppleWebKit/605.1.15 (KHTML, like Gecko) Version/13.0.3 Safari/
↪605.1.15', 'X-Amzn-Trace-Id': 'Root=1-5e4229f8-ce30254a99ad99202e50b612'}, 'json':␣
↪None, 'origin': '42.114.13.13', 'url': 'http://httpbin.org/post'}
```

(continues on next page)

(continued from previous page)

```
>>>
```

Send requests with custom headers

```
>>> headers = {'Referer': 'https://github.com/?q=scraping',
...     'User-Agent': 'Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_6) AppleWebKit/
↪537.36 (KHTML, like Gecko)',
...     }
>>> doc = s.load('https://github.com/cungnv/scrapex', headers=headers)
>>>
>>> doc.response.request.headers
{'Accept': 'text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8', 'User-
↪Agent': 'Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_6) AppleWebKit/537.36 (KHTML,␣
↪like Gecko)', 'Accept-Language': 'en-us,en;q=0.5', 'Accept-Encoding': 'gzip, deflate
↪', 'Connection': 'keep-alive', 'Referer': 'https://github.com/?q=scraping', 'Cookie
↪': '_octo=GH1.1.1947465757.1581393340; logged_in=no; _gh_
↪sess=WjhnT0tOV2Nqa2xZRHVqY1VwWkgwckRMT3FJck03UWtlSThMYkdUVFNTYzBMNW5jcTYvd3NKbzhGR0Y3bUJhOFlUZkt1VW
↪%3D%3D--85276db3b6db4db17aa5766f84e5251d8911f146'}
>>>
```

Failed requests

```
>>> try:
...     doc = s.load('http://httpbin.org/status/404')
... except Exception as e:
...     print(type(e))
...     print('status code:', e.response.status_code)
...
<class 'requests.exceptions.HTTPError'>
status code: 404
>>>
```

1.4 How to parse data in scrapex

It's very easy to parse html document to data points in scrapex.

parse data points from the document using XPATH

```
>>> doc = s.load('https://github.com/search?q=scraping+framework')
>>> headline = doc.extract("//h3[contains(text(),'results')]").strip()
>>> print(headline)
445 repository results
>>>
>>> number_of_results = headline.subreg('([\d\,]+)').replace(',','')
>>> print(number_of_results)
445
>>>
>>> next_page_url = doc.extract("//a[.='Next']/@href")
>>> print(next_page_url)
https://github.com/search?p=2&q=scraping+framework&type=Repositories
>>>
```

parse data from a child node

```
>>> nodes = doc.query("//ul[@class='repo-list']/li")
>>> print('num of nodes:', len(nodes))
num of nodes: 10
>>>
>>> node = nodes[0] #play with first result
>>> repo_name = node.extract(".//div[contains(@class,'text-normal')]/a")
>>> print(repo_name)
scrapy/scrapy
>>>
>>> description = node.extract(".//div[@class='mt-n1']/p").strip()
>>> print(description)
Scrapy, a fast high-level web crawling & scraping framework for Python.
>>>
>>> tags = node.query(".//a[contains(@class,'topic-tag')]").join(', ')
>>> print(tags)
python, crawler, framework, scraping, crawling
>>>
```

parse street address components

```
>>> from scrapex import common
>>>
>>> doc = s.load('https://www.yellowpages.com/search?search_terms=restaurant&geo_
↪location_terms=New+York%2C+NY')
>>> first_result = doc.node("//div[@class='result']")
>>>
>>> name = first_result.extract(".//a[@class='business-name']").strip()
>>> print(name)
Mr. K's
>>>
>>> full_address = first_result.query(".//p[@class='adr']/following-sibling::div").
↪join(', ').replace(',,',',')
>>> print(full_address)
570 Lexington Ave, New York, NY 10022
>>>
>>> parsed_address = common.parse_address(full_address)
>>> print(parsed_address)
{'address': 570 Lexington Ave, 'city': New York, 'state': NY, 'zipcode': 10022}
>>>
```

1.5 Save results to csv/excel file

```
>>> from scrapex import Scraper
>>> from collections import OrderedDict
>>>
>>> s = Scraper(use_cache=True)
>>> doc = s.load('https://github.com/search?q=scraping+framework')
>>> nodes = doc.query("//ul[@class='repo-list']/li")
>>> print('num of nodes:', len(nodes))
num of nodes: 10
>>>
>>> for node in nodes:
...     item = OrderedDict()
...     item['name'] = node.extract(".//div[contains(@class,'text-normal')]/a")
...     item['description'] = node.extract(".//div[@class='mt-n1']/p").strip()
```

(continues on next page)

(continued from previous page)

```
...     item['tags'] = node.query(".//a[contains(@class,'topic-tag')]").join(', ')
...
...     s.save(item, 'results.csv')
...     s.save(item, 'other_file.xlsx') #save this same item to another file
...
>>>
```

1.6 Work with proxies

Enable proxies at scraper level, and choose a random one for each request

```
>>> from scrapex import Scraper
>>> s = Scraper(proxy_file="/path/to/a_proxy_file.txt")
>>>
>>> doc = s.load('https://httpbin.org/anything')
>>> proxy_used = doc.response.json()['origin']
>>> print(proxy_used)
193.31.72.120
>>>
```

Disable use of proxy at request level

```
>>> doc = s.load('https://httpbin.org/anything', use_proxy=False)
>>> client_real_ip = doc.response.json()['origin']
>>> print(client_real_ip)
42.114.13.13
>>>
```

The format of proxy file without authentication

```
ip:port
ip:port
ip:port
.......
```

The format of proxy file with authentication

```
user:password@ip:port
user:password@ip:port
user:password@ip:port
.......
```

1.7 Custom HTTP headers

By default, for each request, the scraper chooses a random User-Agent from a list of popular ones. It also uses a set of HTTP headers that a normal browser does.

You can easily customize the headers per request.

```
>>> from scrapex import Scraper
>>> s = Scraper()
>>> doc = s.load('https://httpbin.org/headers')
```

(continues on next page)

(continued from previous page)

```
>>> print(doc.response.text)
{
  "headers": {
    "Accept": "text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8",
    "Accept-Encoding": "gzip, deflate",
    "Accept-Language": "en-us,en;q=0.5",
    "Host": "httpbin.org",
    "User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:71.0) Gecko/20100101␣
↪Firefox/71.0",
    "X-Amzn-Trace-Id": "Root=1-5e42957e-d5aa6cbae8ef6e0c2a2946c4"
  }
}
>>>
>>> headers={'User-Agent':'my own user-agent', 'Referer': 'some referer url'}
>>> doc = s.load('https://httpbin.org/headers', headers=headers)
>>> print(doc.response.text)
{
  "headers": {
    "Accept": "text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8",
    "Accept-Encoding": "gzip, deflate",
    "Accept-Language": "en-us,en;q=0.5",
    "Host": "httpbin.org",
    "Referer": "some referer url",
    "User-Agent": "my own user-agent",
    "X-Amzn-Trace-Id": "Root=1-5e4295fe-d8454ea67c11ecf84e6c51be"
  }
}
>>>
```

1.8 Work with session/cookies

By default, session is turned off because it's a good practice for web scraping to avoid being tracked.

For scraping task that requires session to manage cookies, like scraping behind a loginwall, you can easily turn it on.

```
>>> from scrapex import Scraper
>>> s = Scraper(use_session=True)
>>>
>>> doc = s.load('http://httpbin.org/anything')
>>> print('cookies sent:',doc.response.request.headers.get('Cookie'))
cookies sent: None
>>>
>>> doc = s.load('http://httpbin.org/cookies/set?name1=value1&name2=value2')
>>> doc = s.load('http://httpbin.org/anything')
>>> print('cookies sent:',doc.response.request.headers.get('Cookie'))
cookies sent: name1=value1; name2=value2
>>>
```

1.9 Cache html content

By default, cache is turned off. In many scraping jobs, we need to make some tweaks to our parsing part and re-scrape the site again. In that situation, caching the html content from the first scrape is very helpful, especially for big scrapes.

Enable cache

```
>>> import os
>>> from scrapex import Scraper
>>> s = Scraper(use_cache=True)
>>> doc = s.load('http://httpbin.org/anything')
>>>
>>> print(os.listdir(s.cache.location))
['47a7ec08a34ed1fb8c78c931818dd082.htm']
>>>
```

Disable cache at request level

```
>>> doc = s.load('http://httpbin.org/anything', use_cache=False)
```

Disable cache at scraper level

```
>>> import os
>>> from scrapex import Scraper
>>> s = Scraper(use_cache=False)
>>> doc = s.load('http://httpbin.org/anything')
>>>
>>> print(os.listdir(s.cache.location))
[]
>>>
```

1.10 Multi threads

Sometimes multi-threads is required to speed up your scrape.

Example code

```
from scrapex import Scraper
from scrapex import common

s = Scraper()

def scrape_github_by_keyword(keyword):
        doc = s.load(url = 'https://github.com/search', params={'q': keyword})
        print(doc.extract("//h3[contains(text(),'results')]").strip())
        #....#

def scrape():
        keywords = ['scraping tool','python','nodejs','image processing','many more']
        print('start 3 threads')
        common.start_threads(keywords, scrape_github_by_keyword, cc=3)

if __name__ == '__main__':
        scrape()
```

CHAPTER 2

Indices and tables

- genindex
- modindex
- search