How to Crawl Infinite Scrolling Pages using Python

Last updated on Feb 25 2019 by Michael Yin

Introduction

Welcome to the article of my series about Web Scraping Using Python

In this tutorial, I will talk about **how to crawl infinite scrolling pages using Python**.

- You are going to learn about how to analyze HTTP request in web dev tools, and use the filter to help you quickly find the target request which gets the real data.
- This tutorial also includes two working code file based on `Scrapy` and `Beautifulsoup`. You can compare them to have a better understanding abut the top two web scraping framework in Python world.

Let's get started.

Background Context

Nowadays, more and more websites start to use `infinite scrolling` to replace the classic pagination. When user scroll to the bottom of the web pages, javascript will send HTTP request and load new items automatically. You can see `infinite scrolling` in most e-commerce website and blogs.

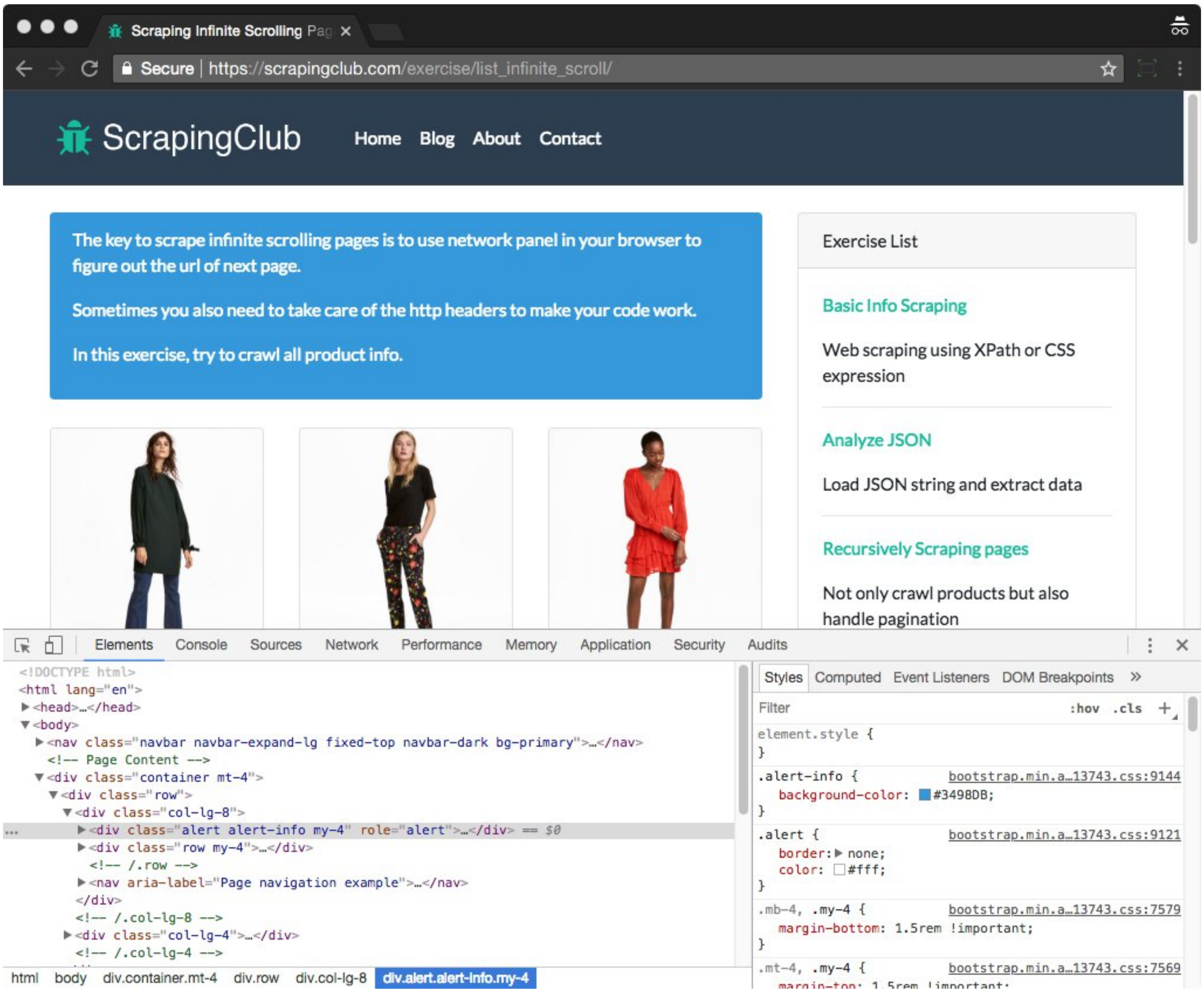
The biggest problem for people to scrape the data in `infinite scrolling pages` is to figure out the URL javascript used to get data of new items

I will use Scraping Infinite Scrolling Pages Exercise as an example to show you how to analyze the page and build spider to get the data.

Analyze web page

I would use `Google Chrome` as an example here.

First, we visit Scraping Infinite Scrolling Pages Exercise, then open web dev tools of our browser to help us inspect the web traffic of the website. If you are new to `web dev tools`, just `Right-click on any page element and select Inspect Element.`.


Scraping Infinite Scrolling Pag

Secure | https://scrapingclub.com/exercise/list_infinite_scroll/

ScrapingClub Home Blog About Contact

The key to scrape infinite scrolling pages is to use network panel in your browser to figure out the url of next page.

Sometimes you also need to take care of the http headers to make your code work.

In this exercise, try to crawl all product info.

Exercise List

Basic Info Scraping

Web scraping using XPath or CSS expression

Analyze JSON

Load JSON string and extract data

Recursively Scraping pages

Not only crawl products but also handle pagination

Elements Console Sources Network Performance Memory Application Security Audits

```
<!DOCTYPE html>
<html lang="en">
▶<head>…</head>
▼<body>
  ▶<nav class="navbar navbar-expand-lg fixed-top navbar-dark bg-primary">…</nav>
   <!-- Page Content -->
  ▼<div class="container mt-4">
    ▼<div class="row">
      ▼<div class="col-lg-8">
        ▶<div class="alert alert-info my-4" role="alert">…</div> == $0
        ▶<div class="row my-4">…</div>
         <!-- /.row -->
        ▶<nav aria-label="Page navigation example">…</nav>
       </div>
       <!-- /.col-lg-8 -->
      ▶<div class="col-lg-4">…</div>
       <!-- /.col-lg-4 -->
```

html body div.container.mt-4 div.row div.col-lg-8 div.alert.alert-info.my-4

Styles Computed Event Listeners DOM Breakpoints »

Filter :hov .cls +

```
element.style {
}
.alert-info {                     bootstrap.min.a…13743.css:9144
  background-color: #3498DB;
}
.alert {                          bootstrap.min.a…13743.css:9121
  border: ▶ none;
  color: #fff;
}
.mb-4, .my-4 {                    bootstrap.min.a…13743.css:7579
  margin-bottom: 1.5rem !important;
}
.mt-4, .my-4 {                    bootstrap.min.a…13743.css:7569
  margin-top: 1.5rem !important;
```

As you can see, a panel shows up for you to inspect the web page. You can use the web dev tool to help you inspect DOM element, debug js, etc.

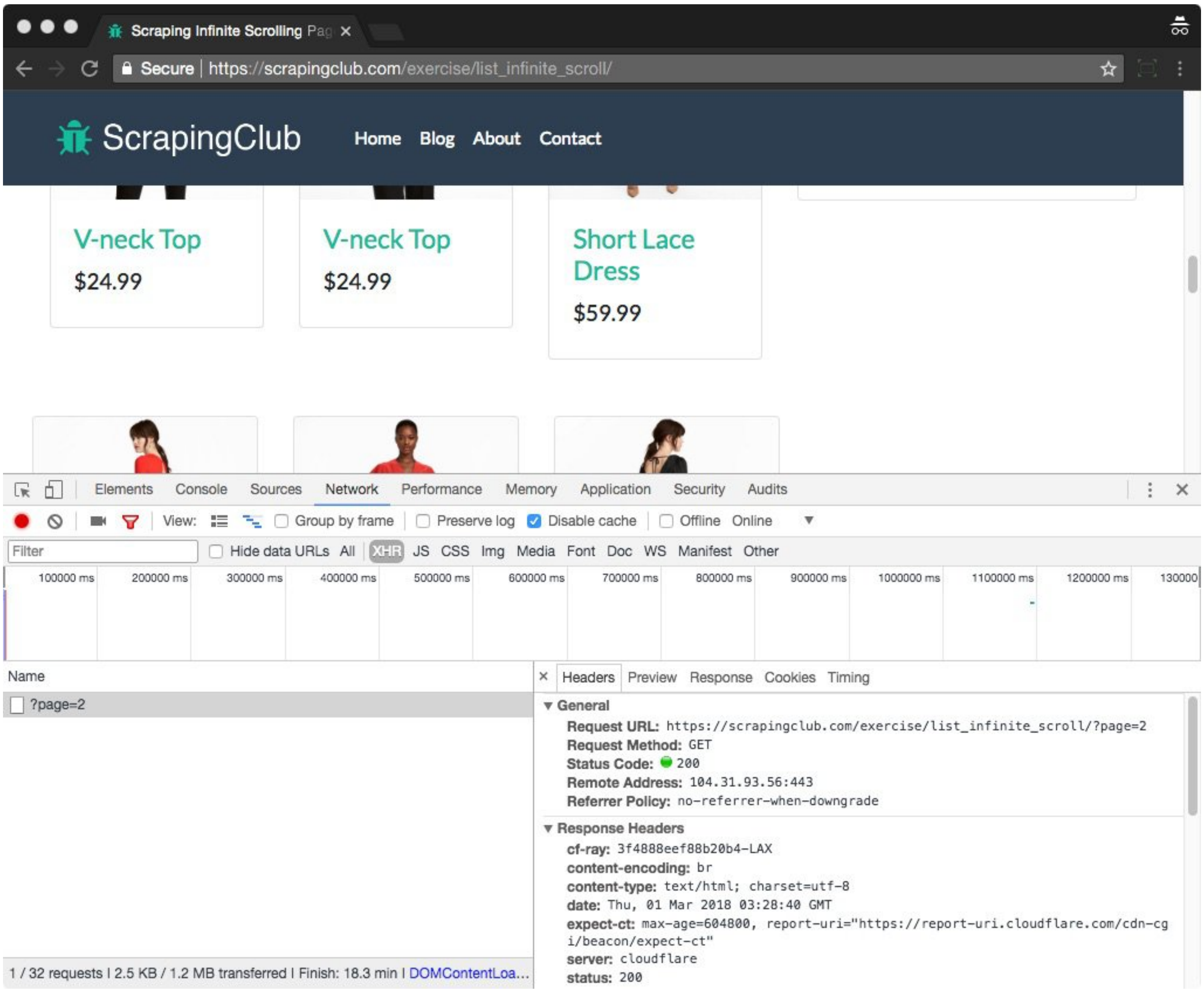
Now we need to find out the URL javascript use to get the following items, so we click the `Network` tab of the dev tool to check all HTTP requests when visiting the webpage.

Here are two basic points you should know about network tab.

1. You can input some keywords to filter requests
2. You can filter the requests based on the request types such as `image`, `XHR`

In most cases, the request we care about can be found in `XHR` (XMLHttpRequest), which means ajax request here.

So after you to set the filter to `XHR`, try to scroll to the bottom then you will see a new request is sent, and new products are loaded in the web page at the same time.


Scraping Infinite Scrolling Pag
Secure | https://scrapingclub.com/exercise/list_infinite_scroll/
ScrapingClub
Home Blog About Contact
V-neck Top
$24.99
V-neck Top
$24.99
Short Lace Dress
$59.99
Elements Console Sources Network Performance Memory Application Security Audits
View: Group by frame Preserve log Disable cache Offline Online
Filter Hide data URLs All XHR JS CSS Img Media Font Doc WS Manifest Other
Name
?page=2
Headers Preview Response Cookies Timing

```
▼ General
   Request URL: https://scrapingclub.com/exercise/list_infinite_scroll/?page=2
   Request Method: GET
   Status Code: ● 200
   Remote Address: 104.31.93.56:443
   Referrer Policy: no-referrer-when-downgrade
▼ Response Headers
   cf-ray: 3f4888eef88b20b4-LAX
   content-encoding: br
   content-type: text/html; charset=utf-8
   date: Thu, 01 Mar 2018 03:28:40 GMT
   expect-ct: max-age=604800, report-uri="https://report-uri.cloudflare.com/cdn-cg
   i/beacon/expect-ct"
   server: cloudflare
   status: 200
```

1 / 32 requests | 2.5 KB / 1.2 MB transferred | Finish: 18.3 min | DOMContentLoa...

You can check the URL, request headers and cookies values of the target request

Here we can see the next page URL is `https://scrapingclub.com/exercise/list_infinite_scroll/?page=2`, and HTTP headers are listed below

```
Referer:https://scrapingclub.com/exercise/list_infinite_scroll/
User-Agent:Mozilla/5.0 (Macintosh; Intel Mac OS X 10_10_4) AppleWebKit/537.36 (KHTML, li
X-Requested-With:XMLHttpRequest
```

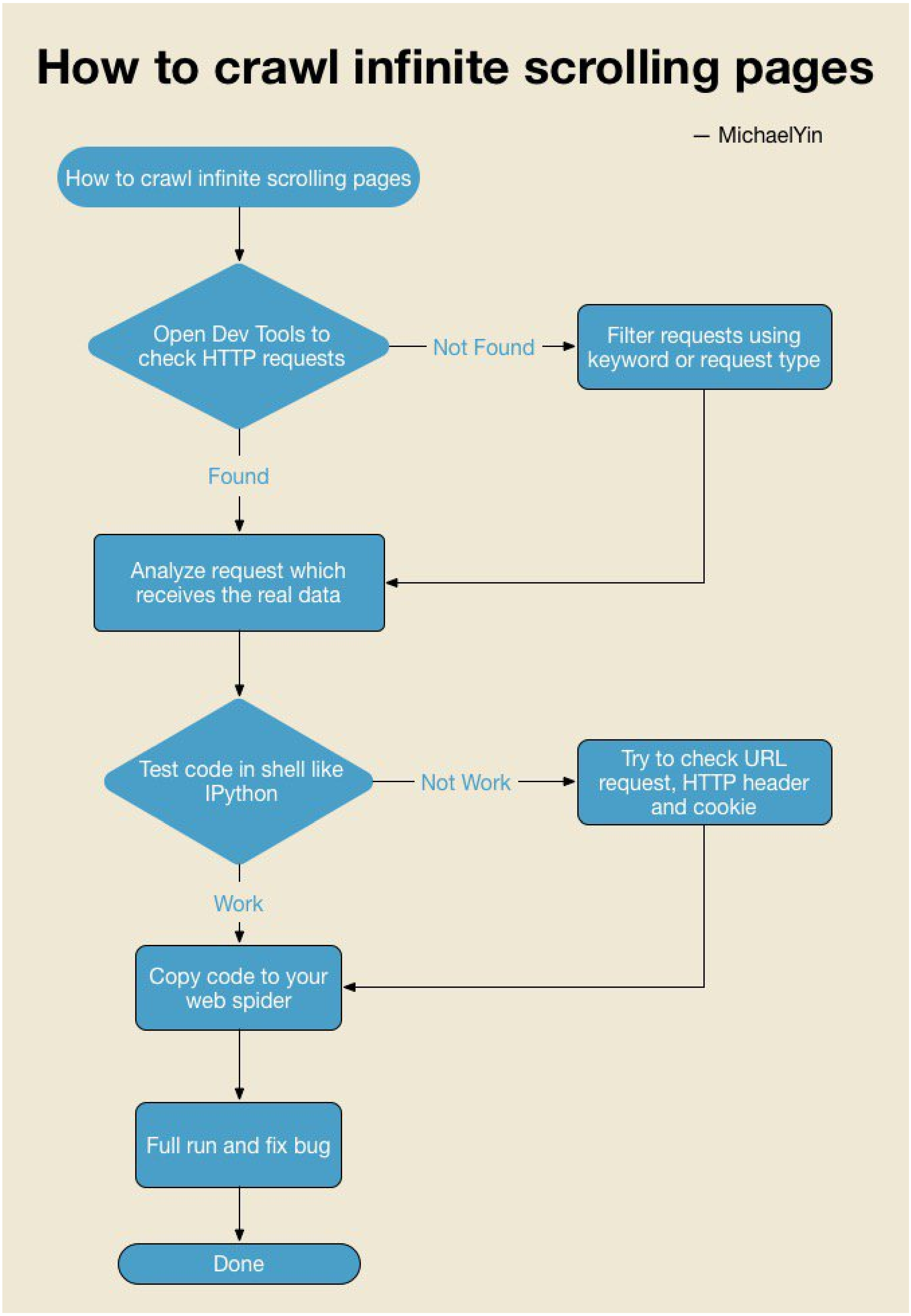
Let me do a brief analyze here, there are three values in HTTP headers, `User-Agent` means which browser you use to visit the page. We can only focus on `X-Requested-With` and `Referer` here.

After we are clear about the detail of the request, the next thing is to implement it in code.

Workflow Chart

Most web scraping tutorial talks more about code and talks less about how to analyze the web page, however, I believe teaching people how to analyze website is much more important than directly giving them lines of code.

Here is a workflow chart helping you to solve similar problem. Feel free to download it and check it when necessary.


How to crawl infinite scrolling pages
— MichaelYin
How to crawl infinite scrolling pages
Open Dev Tools to check HTTP requests
Not Found
Filter requests using keyword or request type
Found
Analyze request which receives the real data
Test code in shell like IPython
Not Work
Try to check URL request, HTTP header and cookie
Work
Copy code to your web spider
Full run and fix bug
Done

If you saw chart above you might be a little confused about `Test code in shell`, let me explain.

Some people like to debug and test spider after it is done, and this make is hard and time-consuming to fix bug. Testing code in Python shell can make sure code work as expect and save a lot of time.

Scrapy solution

Next, I will try to show you how to crawl infinite scrolling pages using `Scrapy`, which is the NO.1 option for people to develop spider in Python.

First, we use the commands below to create a scrapy project, if you have trouble installing `scrapy` on your machine, you can check the detailed install guide for mac, linux and win

```
$ scrapy startproject scrapy_spider
$ cd scrapy_spider
```

Now we enter `scrapy shell` and test our code in it.

```
$ scrapy shell https://scrapingclub.com/exercise/list_infinite_scroll/
```

If you have not installed `IPython shell`, then `scrapy` will use the default python shell, but I recommend you to install `IPython` to bring more powerful feature to your python shell.

```
>>> from scrapy.http.request import Request
>>> url = 'https://scrapingclub.com/exercise/list_infinite_scroll/?page=2'
>>> req = Request(url=url)
>>> fetch(req)
Crawled (200) <GET https://scrapingclub.com/exercise/list_infinite_scroll/?page=2> (refe
```

I built request with only the next URL, and it works!, the website did not check the `useragent`, `X-Requested-With`, I was feeling lucky! if you still fail in this step, you need to add headers as I mentioned above to make sure the request sent by our spider is exactly the same as browser sent, that is the key!

Testing code in Python shell first is the most efficient way and you really should learn how to do it

Congratulations! You have got the skill to analyze web page and test code in Python shell. Below I've added the entire Scrapy spider code so you can learn if you are interested. You can put the file at `scrapy_spider/spiders/infinite_scroll.py` and then run command `scrapy crawl infinite_scroll` to run the Scrapy spider.

```
# -*- coding: utf-8 -*-

from __future__ import print_function
import json
import re
import logging

import scrapy
from scrapy.http.request import Request
from spider_project.items import SpiderProjectItem

from six.moves.urllib import parse
```

```
class List_infinite_scroll_Spider(scrapy.Spider):
    name = "infinite_scroll"

    def start_requests(self):
        yield Request(
            url="https://scrapingclub.com/exercise/list_infinite_scroll/",
            callback=self.parse_list_page
        )

    def parse_list_page(self, response):
        """
        The url of next page is like
        https://scrapingclub.com/exercise/list_infinite_scroll/?page=2

        It can be found in a.next-page
        """
        #First, check if next page available, if found, yield request
        next_link = response.xpath(
            "//a[@class='page-link next-page']/@href").extract_first()
        if next_link:
            # If the website has strict policy, you should do more work here
            # Such as modifying HTTP headers

            # concatenate url
            url = response.url
            next_link = url[:url.find('?')] + next_link
            yield Request(
                url=next_link,
                callback=self.parse_list_page
            )

        # find product link and yield request back
        for req in self.extract_product(response):
            yield req

    def extract_product(self, response):
        links = response.xpath("//div[@class='col-lg-8']//div[@class='card']/a/@href").e
        for url in links:
            result = parse.urlparse(response.url)
            base_url = parse.urlunparse(
                (result.scheme, result.netloc, "", "", "", "")
            )
            url = parse.urljoin(base_url, url)
            yield Request(
                url=url,
                callback=self.parse_product_page
            )

    def parse_product_page(self, response):
        """
        The product page use ajax to get the data, try to analyze it and finish it
        by yourself.
        """
        logging.info("processing " + response.url)
        yield None
```

BeautifulSoup solution

Since BeautifulSoup is so popular in Python world, so here I also add code using BeautifulSoup for you to compare and learn. The most interesting part is that you can find out that you can easily migrate your code to Scrapy if your pattern is in this way. You can save this file as `infinite_scroll.py` and `python infinite_scroll.py`

```python
#! /usr/bin/env python
# -*- coding: utf-8 -*-
# vim:fenc=utf-8

"""

BeautifulSoup does not support XPath expression by default, so we use CSS
the expression here, but you can use https://github.com/scrapy/parsel to write
XPath to extract data as you like

"""
from __future__ import print_function
from bs4 import BeautifulSoup
import requests
from six.moves.urllib import parse

START_PAGE = "https://scrapingclub.com/exercise/list_infinite_scroll/"

QUEUE = []

def parse_list_page(url):
    r = requests.get(url)
    soup = BeautifulSoup(r.text, "lxml")

    links = soup.select('a[class="page-link next-page"]')
    if links:
        next_link = links[0].attrs['href']
        next_link = url[:url.find('?')] + next_link
        QUEUE.append(
            (parse_list_page, next_link)
        )

    links = soup.select('div.col-lg-8 div.card a')
    for link in links:
        product_url = link.attrs['href']
        result = parse.urlparse(url)
        base_url = parse.urlunparse(
            (result.scheme, result.netloc, "", "", "", "")
        )
        product_url = parse.urljoin(base_url, product_url)
        QUEUE.append(
            (parse_detail_page, product_url)
        )

def parse_detail_page(url):
    # r = requests.get(url)
    # soup = BeautifulSoup(r.text, "lxml")
    print("processing " + url)

def main():
    """
    Push callback method and url to queue
    """
    QUEUE.append(
        (parse_list_page, START_PAGE)
    )

    while len(QUEUE):
        call_back, url = QUEUE.pop(0)
        call_back(url)

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

What’s Next and What Have You Learned?

In this article, we build a spider using Python to crawl infinite scrolling pages. We learned how to use `web dev tools` to help us analyze web traffic, and how to test code in `Scrapy shell` which is an efficient way for us to develop spiders.

So where should you go next?

- You can implement `parse_product_page` of spider above, to make it can crawl product detail page as well, all the skills needed has been discussed in this article, treat it like an exercise.

- If you want to improve your web scraping skills, just check other interesting web scraping exercises on ScrapingClub

Thanks a lot for reading! Looking forward to your questions.