Scrapy Beginners Series Part 4: User Agents and Proxies

So far in this **Python Scrapy 5-Part Beginner Series** we learned how to build a basic Scrapy spider, get it to scrape some data from a website, clean up the data as it was being scraped and then save the data to a file or database.

In **Part 4** we will be exploring how to use User Agents and Proxies to bypass restrictions on sites who are trying to prevent any scraping taking place.

Python Scrapy 5-Part Beginner Series

- **Part 1: Basic Scrapy Spider -** We will go over the basics of Scrapy, and build our first Scrapy spider. (Part 1)
- **Part 2: Cleaning Dirty Data & Dealing With Edge Cases -** Web data can be messy, unstructured, and have lots of edge cases. In this tutorial we will make our spider robust to these edge cases, using Items, Itemloaders and Item Pipelines. (Part 2)
- **Part 3: Storing Our Data -** There are many different ways we can store the data that we scrape from databases, CSV files to JSON format, and to S3 buckets. We will explore several different ways we can store the data and talk about their Pro's, Con's and in which situations you would use them. (Part 3)
- **Part 4: User Agents & Proxies -** Make our spider production ready by managing our user agents & IPs so we don't get blocked. (This Tutorial)
- **Part 5: Deployment, Scheduling & Running Jobs -** Deploying our spider on a server, and monitoring and scheduling jobs via ScrapeOps. (Part 5)

The code for this project is available on Github here!

If you prefer video tutorials, then check out the video version of this article.

Getting Blocked & Banned Whilst Web Scraping

What you will quickly find out when you start scraping at any large scale volume, is that building and running your scrapers is the easy part. The true difficulty of web scraping is in being able to reliably retrieve HTML responses from the pages you want to scrape.

Whilst you can easily scrape a couple hundred pages with your local machine, when you need to scrape thousands or millions of pages websites will quickly start blocking your requests.

Large websites such as Amazon monitor who is visiting their website by tracking your IPs and user agents, and detecting any unusual behaviour using sophisticated anti-bot techniques. If they identify someone they think is a scraper then they will block your requests.

This isn't the end of the world however, as by properly managing our user agents, IP addresses and cookies we use when scraping we can bypass making of these anti-bot countermeasures.

For the purposes of our beginners project scraping Chocolate.co.uk we don't need to worry about it. However, in this guide we're still going to look at how we can dynamically rotate our User Agents and IPs so that you can apply these techniques if you ever need to scrape a more difficult website like Amazon.

In this tutorial, **Part 4: Beginners guide to Scrapy User Agents and Proxies** we're going to cover:

Using User Agents When Scraping

When scraping the web, oftentimes the site you want to scrape don't really want you scraping their data so you need to disguise who you are. To do so we need to manage the **User Agents** we send along with our HTTP requests.

User Agents are strings that let the website you are scraping identify the application, operating system (OSX/Windows/Linux), browser (Chrome/Firefox/Internet Explorer), etc. of the user sending a request to their website. They are sent to the server as part of the request headers.

You can think of a User Agent as a way for a browser to say “Hi, I’m Mozilla Firefox on Windows” or “Hi, I’m Safari on an iPhone” to a website.

Here is an example User agent sent when you visit a website with a Chrome browser:

```
user-agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/98.0.4758.82 Safari/537.36
```

Scrapy User Agent

Web scrapers and crawlers also need to set the user agents they use as otherwise the website may block your requests based on the user agent you send to their server.

For example, the default user agent Scrapy sends when making a request is:

```
Scrapy/VERSION (+https://scrapy.org)
```

This user agent will clearly identify your requests as comming from a web scraper so the website can easily block you from scraping the site. So if scraping most sites you will want to change this (will show you later).

Identifying Yourself (Googlebot)

Other times, you may want to clearly identify yourself as a specific web scraper/crawler as that website might want you to scrape their site and give you better treatment.

The most famous example is Google's Googlebot that crawls the web to index pages and rank them:

```
Googlebot/2.1 (+http://www.google.com/bot.html)
```

Web servers/websites can give bots special treatment, for example, by allowing them through mandatory registration screens, bypassing CAPTCHA etc.

Changing User Agents

If you are scraping any popular website, they will have some level of blocking mechanism in place based on the user agents it sees you using.

We can get around this by rotating through multiple user agents, that appear like real visitors to their site.

Luckily for us, the Scrapy community have created some great extensions that make rotating through user agents very easy.

In this case, we're going to use the `scrapy-user-agents` Scrapy download middleware.

To use the **scrapy-user-agents** download middleware, simply install it:

```
pip install scrapy-user-agents
```

Then in add it to your projects `settings.py` file, and disable Scrapy's default **UserAgentMiddleware** by setting its value to `None`:

```
DOWNLOADER_MIDDLEWARES = {
    'scrapy.downloadermiddlewares.useragent.UserAgentMiddleware': None,
    'scrapy_user_agents.middlewares.RandomUserAgentMiddleware': 400,
}
```

The **scrapy-user-agents** download middleware contains about 2,200 common user agent strings, and rotates through them as your scraper makes requests.

Okay, managing your user agents will improve your scrapers reliability, however, we also need to manage the IP addresses we use when scraping.

Using Proxies to Bypass Anti-bots and CAPTCHA's

Even when rotating your user agents, most websites will still refuse to serve your requests if you only specify User-Agent in the headers as they are also keeping track of the IP address.

When a site sees to many requests coming from one IP Address/User Agent combination they usually limit/throttle the requests coming from that IP address. Most of the time this is to prevent things such as DDOS attacks or just to limit the amout of traffic to their site so that their servers don't run out of capacity.

That is why we also will need to look at using proxies in combination with the random user agents to provide a much more reliable way of bypassing the restrictions placed on our spiders.

While it's easy to scrape a few pages from a website using your local IP address and random User Agents if you want to scrape thousands/millions of pages you will need a proxy.

Note: For those of you who don't know, your IP address is your computers unique identifier on the internet.

Rotating IP Addresses With Proxies

To bypass this rate limiting/throttling the easiest thing we can do is to change the IP address from which we are sending our scraping requests - just like the randomising of the User Agents which we have already looked at. This is done using **proxies**.

Proxies are a gateway through which you route your scraping requests/traffic. As part of this routing process the IP address is updated to be the IP address of the gateway through which your scraping requests went through.

Several companies provide this service of rotating proxies and their costs vary depending on their level of service and reliability.

Proxy prices range from Free to thousands of dollars per month, and price often always isn't correlated to the performance.

Using Paid Proxies

There are many professional proxy services available that provide much higher quality of proxies that ensure almost all the requests you send via their proxies will reach the site you intend to scrape.

Here are some of the best proxy providers:

- ScrapeOps
- Bright Data
- Oxylabs
- ScraperAPI
- Zyte
- Geonode

All of these proxy providers are slightly different with different proxy products and integration methods so we can't cover all of them in detail within this guide.

However, you can use our **Free Proxy Comparison Tool** that allows to compare the pricing, features and limits of every proxy provider on the market so you can find the one that best suits your needs. Including finding the proxy providers who offer the most generous Free Plans.

For this Beginner Series we're going to use ScrapeOps, as it has a great free plan and it's the most reliable solution(Being an "All-In-One Proxy" - It uses the best proxy provider from a pool of over 20+ proxy providers). However, you can use a different proxy if you wish.

Integrating ScrapeOps

The first thing we need to do would is to register for a free ScrapeOps account.

Once you have your account setup and have found the API key from your account you can then start to interact with the API endpoint they provide using a very simple method which we will code into our `chocolatespider`.

To do so we can simply add the following code to the top of our `chocolatespider.py` file.

```
API_KEY = 'YOUR_API_KEY'

def get_proxy_url(url):
    payload = {'api_key': API_KEY, 'url': url}
    proxy_url = 'https://proxy.scrapeops.io/v1/?' + urlencode(payload)
    return proxy_url
```

And switch the use this function in our Scrapy request:

```
yield scrapy.Request(url=get_proxy_url(url), callback=self.parse)
```

This is how your final code should look.

```
import scrapy
from chocolatescraper.itemloaders import ChocolateProductLoader
from chocolatescraper.items import ChocolateProduct
from urllib.parse import urlencode

API_KEY = 'YOUR_API_KEY'

def get_proxy_url(url):
    payload = {'api_key': API_KEY, 'url': url}
    proxy_url = 'https://proxy.scrapeops.io/v1/?' + urlencode(payload)
    return proxy_url

class ChocolateSpider(scrapy.Spider):

   # The name of the spider
   name = 'chocolatespider'

   # These are the urls that we will start scraping
   def start_requests(self):
        start_url = 'https://www.chocolate.co.uk/collections/all'
        yield scrapy.Request(url=get_proxy_url(start_url), callback=self.parse)


   def parse(self, response):
       products = response.css('product-item')

       for product in products:
            chocolate = ChocolateProductLoader(item=ChocolateProduct(), selector=product)
            chocolate.add_css('name', "a.product-item-meta__title::text")
            chocolate.add_css('price', 'span.price', re='<span class="price">\n              <span class="visually-
hidden">Sale price</span>(.*)</span>')
            chocolate.add_css('url', 'div.product-item-meta a::attr(href)')
            yield chocolate.load_item()

       next_page = response.css('[rel="next"] ::attr(href)').get()

       if next_page is not None:
           next_page_url = 'https://www.chocolate.co.uk' + next_page
           yield response.follow(get_proxy_url(next_page_url), callback=self.parse)
```

Concurrent Requests

When running more than a small scale crawl we might want to have several requests happen at the same time. We can make this happen with the `CONCURRENT_REQUESTS` setting inside our `settings.py` file. However, when using paid proxies we need to make sure that the plan we are using has a high enough concurrent requests limit - otherwise they will block some of our requests and only allow the amount we have paid for through.

So a plan might be cheaper, but the concurrency limit might be very low - this can slow down our crawl as we'll only be able to send out one request at a time. You'll need to decide if you want your crawl to be completed quicker and then pay for higher concurrency limits or take longer if you only want to pay for a plan with a lower concurrency limit.

Before running the spider we need to update our `settings.py` file to set the number of `CONCURRENT_REQUESTS` to **1** and make sure the previous user-agent and proxy middlewares we added above are disabled.

```
## settings.py

CONCURRENT_REQUESTS = 1

DOWNLOADER_MIDDLEWARES = {

    ## Rotating User Agents
    # 'scrapy.downloadermiddlewares.useragent.UserAgentMiddleware': None,
    # 'scrapy_user_agents.middlewares.RandomUserAgentMiddleware': 400,

    ## Rotating Free Proxies
    # 'scrapy_proxy_pool.middlewares.ProxyPoolMiddleware': 610,
    # 'scrapy_proxy_pool.middlewares.BanDetectionMiddleware': 620,
}
```

???

Now when we run our spider, Scrapy will route our requests through ScrapeOps and prevent them from being blocked.

Next Steps

We hope you now have a good understanding of how to use User Agents and Proxies/Proxy Providers to bypass any sites which are blocking/limiting you from scraping the data you need! If you have any questions leave them in the comments below and we'll do our best to help out!

If you would like the code from this example please check it out on Github.

The next tutorial covers how to make our spider production ready by deploying our spiders to an external server, setting up monitoring, alerting and automated scheduling.

We'll cover these in Part 5 - Deployment, Monitoring, Alerting and Scheduling with Scrapy .