g+1 2

Nov 18, 2014

By Editor

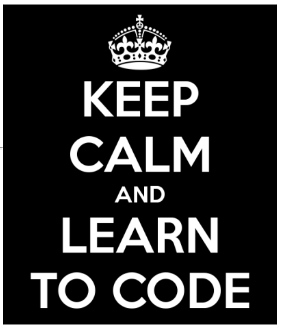
in DEVELOPMENT

1 COMMENT

Tags: PYTHON, SCRAPY, WEB SCRAPING

Web Scraping with Python + Scrapy (blog series)

This is part 1 of a series dedicated to getting novices started using a simple web scraping framework using python.



Introduction

In this post we will get up and running with simple web scraping using Python, specifically the Scrapy Framework.

What is Python?

Python is a clear and powerful a high-level general-purpose object-oriented programming language. This tutorial doesn't assume that you are an expert in Python, but if you've not used python before consider learning the basics of python over at Codecademy.

What is Scrapy?

Scrapy defines itself as A Fast and Powerful Scraping and Web Crawling Framework. It's an Open-Source framework written in Python and benifits from a vibrant and active community of contributors, maintainers and users.

Getting started:

So now that we know what Scrapy is, let's get started using it. From your console run the command:

```
pip install scrapy
```

pip will install the package, once it has completed you can test if everything is working by running:

```
scrapy version
```

at the time of writing scrapy is at v 0.24, so the output is simple prints

```
Scrapy 0.24.4.
```

Now that we have the scrapy package installed, lets use it to write a simple scraper. I would like to keep up with the latest news on technology, so I'm going to write a scraper to get technology news from one of my favourite news websites.

We can start a new project called news by running the startproject command, and passing it the name of your project, in our example i will call the project news

TAG CLOUD

ANALYTICS ANTI-SCRAPE BIG DATA CAPTCHA CRAWLING DATA MINING FREE GOOGLE HTTP IMPORT.IO JAVA JSON KIMONO PHP PROXY PYTHON REGEX SCRAPE-DETECTION SCRAPER SCRAPING TOOL SCRAPY SELENIUM SEO SERVICE SNIFFER STATISTICS STRUCTURED APIS VISUAL WEB RIPPER VISUALIZATION WEB SCRAPING XPATH

Your email:

Enter email address...

SUBSCRIBE UNSUBSCRIBE

FEATURED

OutWit Hub Review

Helium Scraper Review

Visual Web Ripper Review

BLOGROLL

SEO and Growth Hacking

SQL Backup Blog

Unbiased Software Reviews

```
scrapy startproject news

&gt; New Scrapy project 'news' created in:
&gt; /www/playground/scraping/news
&gt;
&gt; You can start your first spider with:
&gt; cd news
&gt; scrapy genspider example example.com
```

As you can see from the output, the scrapy tool has created a new project called news and added some boilerplate code which we can use to jump stright into.

The purpose of this scraper will be to grab the headlines, and a few other bits of data from the each story: We'll keep it fairly simple in this first article, but we will continue to extend the functionality in later tutorials.

Scrapy aims to extract what it calls Items, so we need to define a simple item, which is done by creating a class that extends the scrapy.Item class provided by the scrapy framework.

open up the items.py file and define a new item subclass called NewsItem:

```
class NewsItem(scrapy.Item):
headline = scrapy.Field()
intro = scrapy.Field()
url = scrapy.Field()
```

Here, we have defined simple item that just has 3 seperate data fields: headline, intro and url. Easy.

So, now that we have our Item subclass defined, we need to create a Spider. A spider 'crawls' domains (in accordance with some rules we will define) to collect all the pages that we wish to extract our NewsItem instances from. Most of this crawling logic is provided by Scrapy in the CrawlSpider class, so we can extend this class when writing our first spider.

```
from scrapy.contrib.spiders import CrawlSpider, Rule
from scrapy.contrib.linkextractors import LinkExtractor
from items import NewsItem

class BbcSpider(CrawlSpider):
name = "bbcnews"
allowed_domains = ["bbc.co.uk"]
start_urls = [
"http://www.bbc.co.uk/news/technology/",
]

rules = [Rule(LinkExtractor(allow=['/technology-\d+']), 'parse_story')]

def parse_story(self, response):

story = NewsItem()
story['url'] = response.url
story['headline'] = response.xpath("//title/text()").extract()
story['intro'] = response.css('p.introduction::text').extract()

return story
```

Can you understand what the code above achieves? I'll explain it below:

First we define our new BbcSpider class, which extends from the CrawlSpider class that Scrapy ships with. We then give it a name attribute, restrict it to crawling pages on a certain domain, and then provide it the initial url(s) to start our crawl with. The rules variable sets-up a few rules that we use to extract further links to crawl. The allow argument contains a regular expression restricts our crawls to urls that match the expression 'technology-' so we only scrape the technology articles, these URLs are crawled and the response object is passed to the parse-story method which we will explain below.

Take a look at the parse_story method, can you tell what it is doing with the HTML response of each URL that it has Extracted and crawled? It's actually very simple, It crates a new instance of NewsItem and assigns this instance to the story variable. It then populates the NewsItem by selecting the various parts of the response.

Scrapy provides various ways of extracting content from the data that we scrape. The framework provides functionaliy called 'Selectors' but in future tutorials we'll go through alternative tools such as the popular 'BeautifulSoup' library. Scrapy includes selector functionity for either XPATH or CSS style selectors. We've covered xpath here at scraping.pro before so you can learn more in these articles: XPATH review and XPath Cheat Sheets and Quick References. The CSS selector syntax is prefered by some people too. Both methods define a way to identify the specific data we want from the page by referencing the specific HTML tags we wish to extract data from.

One way of getting the headline from the HTML is extracting the text from thetag. This is simple to define in xpath: //title/text() We use the XPATH selector by calling response.xpath("//title/text()").extract() and assignin this to the 'headline' property of our NewsItem instance.

Some stories include an introduction that is included in a p tag, with a class of "introduction". We can can look for this tag by using the CSS selector p.introduction::text We then assign this to the "intro" property of the NewsItem instance

another method way of extracting data in scrapy is using CSS style selectors. Internally it uses the 'selector' library to convert these into xpath definitions, but either can be used to identify specific parts of the HTML responses.

the Url of the crawled page is available in the to the response object within the response.url attribute, so there's no need to use selectors to populate the url field of our NewsItem

Let's go!

To recap: we have the Item defined, and the Spider, so let's crawl! Before we do I recommend openning settings.py and adding a useragent to identify your crawler. However, if you don't want to broadcast your identity Scrapy does provide ways to help mask this. We will cover this in future tutorials. Stay tuned.

Now: start the crawl by running the scrapy crawl command

```
scrapy crawl bbcnews --output results.json
```

You will see Scrapy jump into action, by running your bbcnews spider. It will visit the start_urls we defined then start using the LinkExtractor to look for other pages it can crawl. Each page it crawls is passed into the parse_story method, and NewsItem Instances are created. These are written to the console along with other debugging information, but as we also passed the –output flag, we all of the NewsItems are saved to a file too.

```
[bbcnews] INFO: Dumping Scrapy stats:
{
'downloader/request_bytes': 20478,
'downloader/request_count': 51,
'downloader/request_method_count/GET': 51,
'downloader/response_bytes': 4768372,
'downloader/response_count': 51,
'downloader/response_status_count/200': 51,
'finish_reason': 'finished',
'finish_time': datetime.datetime(2014, 11, 3, 15, 43, 11, 913375),
'item_scraped_count': 50,
'log_count/DEBUG': 103,
'log_count/INFO': 8,
'request_depth_max': 1,
'response_received_count': 51,
'scheduler/dequeued': 51,
'scheduler/dequeued/memory': 51,
'scheduler/enqueued': 51,
'scheduler/enqueued/memory': 51,
'start_time': datetime.datetime(2014, 11, 3, 15, 43, 10, 408258)
}
```

Extra Credit:

If you can't wait until the next tutorial, and you're enjoying using scrapy. Try to extend the code we've written so far to do some more things:

Extend the functionality of our scraper to remove the prefix "BBC – News" at the start of each headline. Can you find were else the headline is in the HTML that would be more suitable, or perhaps you could strip out the prefix using another method?

Add another field to the NewsItem object, update the spider code to populate this new field. How about published date, or the full text body of the story?

How about writing another spider, to crawl technology stroies from a different source – How about HackerNews, or TechCrunch? Give it a go.

What about using the site's RSS feeds so you don't have to parse and select various parts of the HTML. Can you work out how to do this with scrapy? Read up on Scrapy Documentation for tips.

Remember to follow scraping.pro on twitter, and subscribe to our RSS feeds to be alerted about new stories about Web Data!

Have a great day!

« *Review: import.io's New Scraping Process and Features*

What is Crawlera? »

One Comment

KEVIN CHAE

NOV 18, 2014 @ 17:01

Thanks for the helpful tutorial.

REPLY

Leave a Reply

	Your Name
	Your Email
	Your Website

POST COMMENT