Finding broken external links on websites using Scrapy

Isaac Bythewood · 2022-07-23 · 3 min read

coding webdev

Finding broken external links on websites using Scrapy

Broken links are a problem for any content driven website as it ages, find them quickly and easily with Scrapy.

Scrapy is an open source scraping and web crawling tool written in Python. It's got a lot of good documentation on getting started and has a very simple to use CLI. I'm not going to go into the details of getting scrapy running since it would be redundant with their documentation, however I will note a quick way to find external broken links with a quick spider implementation that I've started using.

After you have a basic scrapy project running in your spiders folder add a new file named `broken_link_spider.py`.

```
from scrapy.spiders import CrawlSpider, Rule
from scrapy.linkextractors import LinkExtractor


class BrokenLinkSpider(CrawlSpider):
    name = 'broken_link_spider'
    start_urls = ['https://blog.bythewood.me/']
    handle_httpstatus_list = [200, 301, 302, 303, 307, 400, 401, 403, 404, 500]

    rules = (
        Rule(
            LinkExtractor(
                allow_domains=['blog.bythewood.me'],
            ),
            callback='parse_local',
            follow=True,
        ),
        Rule(
            LinkExtractor(),
            callback='parse_external',
        ),
    )

    def parse_local(self, response):
        if response.status != 200:
            return {
                'url': response.url,
                'status': response.status,
                'type': 'local',
            }

    def parse_external(self, response):
        if response.status != 200:
            return {
                'url': response.url,
                'status': response.status,
                'type': 'external',
            }
```

To explain the pieces of this spider:

- By default Scrapy ignores http responses that are not a 200 status code. You have to tell it to scrape all the status codes you want, including broken ones, with the `handle_httpstatus_list` variable.
- We then need two rules. The first is for internal links that we follow to make sure we scrape every page on our website, we also want to note non-200 status codes on our own site incase we have a broken link there.

- The second rule is for crawling the first page of an external site, we don't want to follow these since we don't inherently care about every page on the external websites, we just want to make sure the page we are landing on is not a dead link.
- We then have two functions to parse our external and local links and we only return links that are not a 200 status code. This is somewhat of a naïve implementation as there are other successful status codes so you may want to change this to suit your needs.

Running this crawler can be done easily with Scrapy's CLI tool using `scrapy crawl broken_link_spider -o output/broken_link_spider.json` and you should be left with a nice list of all the dead links on your site in the output file!

As always with scraping and crawling make sure you obey `robots.txt` files and have permission to crawl the website before you go wild with something like this. It's a quick way to get your IP address blacklisted if you aren't following the rules.