Scrapy Tutorial #10: How To Build A Real Spider

Last updated on Feb 25 2019 by Michael Yin

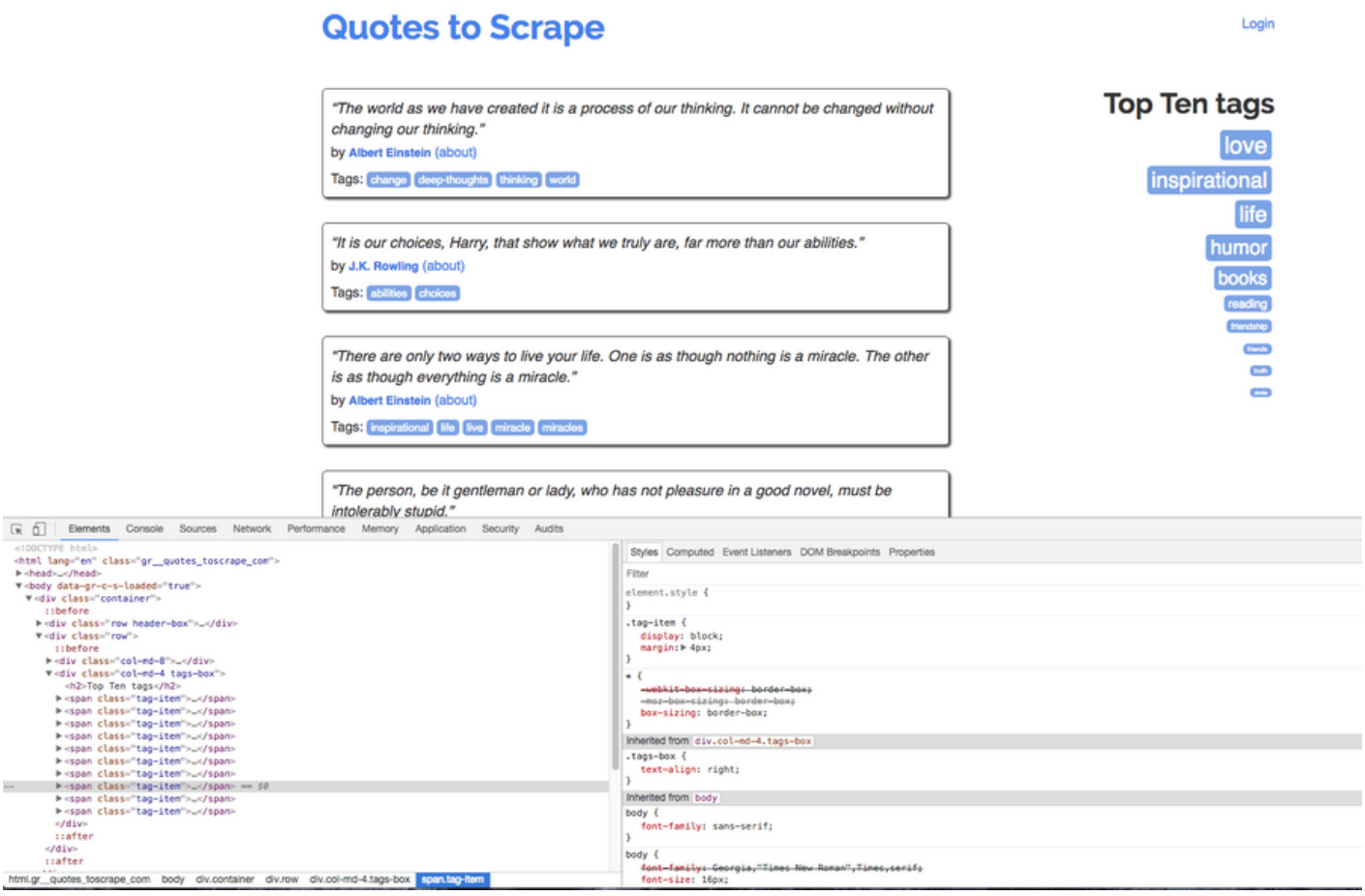
Introduction:

This is the #10 post of my Scrapy Tutorial Series, in this Scrapy tutorial, I will show you how to write a real spider so that we can reuse the commands, tools we talked about in previous posts and the new spider would iterate the container, iterate over the pages so in the end, we could get all the quotes from Quotes to Scrape.

You can get the source code of this project at the end of this tutorial.

Analyze Dom Element In Browser DevTools

Before starting to write code to the editor, we need to check the element in web browser to figure out the where the element is located, how the pagination work. This step is very important during the spider development.

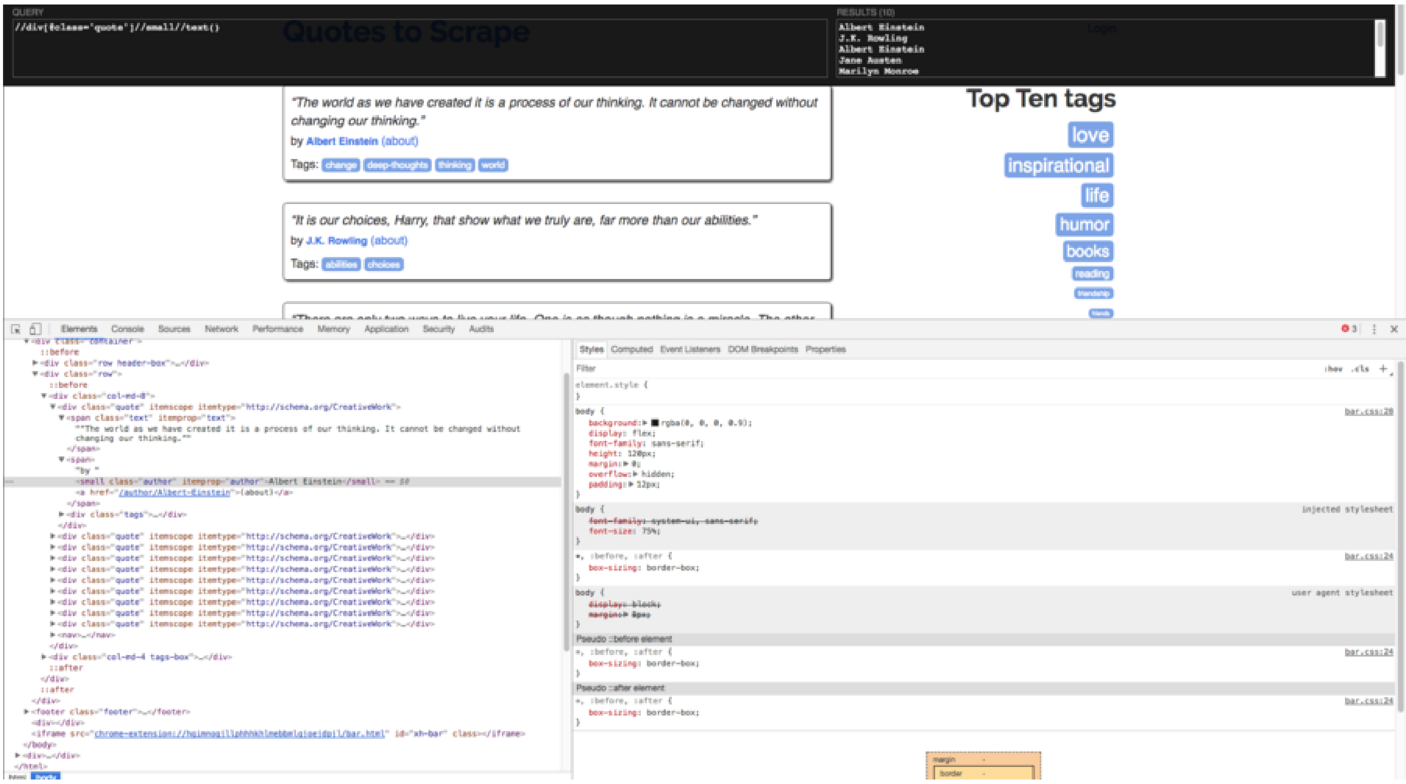


In Chrome, right click in the web page, then click the `inspect`, we can see a window popup in the browser, which is called `DevTools`. There are many tabs in DevTools, right now you can only focus on the `elements` tab, in this tab you can inspect all the dom element of the web page.

For example, in a div which have class `col-md-8`, there are 10 divs which have class `quote`, then the author info and quote text are located in different span elements.

```
▼<div class="col-md-8">
  ▼<div class="quote" itemscope itemtype="http://schema.org/CreativeWork">
    ▼<span class="text" itemprop="text">
        "“The world as we have created it is a process of our thinking. It cannot be changed without
        changing our thinking.”" == $0
      </span>
    ▼<span>
        "by "
        <small class="author" itemprop="author">Albert Einstein</small>
        <a href="/author/Albert-Einstein">(about)</a>
      </span>
    ▶<div class="tags">…</div>
    </div>
  ▶<div class="quote" itemscope itemtype="http://schema.org/CreativeWork">…</div>
  ▶<div class="quote" itemscope itemtype="http://schema.org/CreativeWork">…</div>
  ▶<div class="quote" itemscope itemtype="http://schema.org/CreativeWork">…</div>
  ▶<div class="quote" itemscope itemtype="http://schema.org/CreativeWork">…</div>
  ▶<div class="quote" itemscope itemtype="http://schema.org/CreativeWork">…</div>
  ▶<div class="quote" itemscope itemtype="http://schema.org/CreativeWork">…</div>
  ▶<div class="quote" itemscope itemtype="http://schema.org/CreativeWork">…</div>
  ▶<div class="quote" itemscope itemtype="http://schema.org/CreativeWork">…</div>
  ▶<div class="quote" itemscope itemtype="http://schema.org/CreativeWork">…</div>
  ▶<nav>…</nav>
  </div>
```

After we have figured out the relationship and element attribute in DevTools, we can use the Browser Plugin help us get the XPath expression. If you want to know more about the XPath plugin, you can check [How to use XPath with Scrapy](#)



After analyzing and testing, now we can use XPath expression such as `//div[@class='quote']//span[@class='text']/text()` to get the quote text and `//div[@class='quote']//small//text()` to get the author info.

Testing Code in Scrapy Shell

After we get the XPath expression of the target information, now we can start to test code in Scrapy shell. We can enter `scrapy shell` to enter the Scrapy shell first. Then use `fetch("http://quotes.toscrape.com/")` to fetch the Html page.

The best way for newbie developer is to write code in your editor, and then paste the code block in Scrapy shell to see if the code can work as expected. For example, you can copy the code block below and paste in scrapy shell to check the output.

```
quotes = response.xpath("//div[@class='quote']")
for quote in quotes:
    text = quote.xpath(
        ".//span[@class='text']/text()").extract_first()
    author = quote.xpath(
        ".//small//text()").extract_first()
    print(text, author)
```

If you have installed IPython, then your Scrapy shell would have some magic commands to help you get job done effectively, one command you must know is `%paste`, this command would paste multi-line code from clipboard to scrapy shell, which is very convenient.

In the terminal, if the code can work as expected, then you will see the quote and author info printed in the terminal, which means the code is right.

Write Spider code

Next, you need to modify the spider code to make it return data to Scrapy to handle. The key point here is keyword `yield`

> yield is a keyword that is used like return, except the function will return a generator.

So if you want to return something for Scrapy to handle, you should use the keyword `yield`. Here is the spider code right now

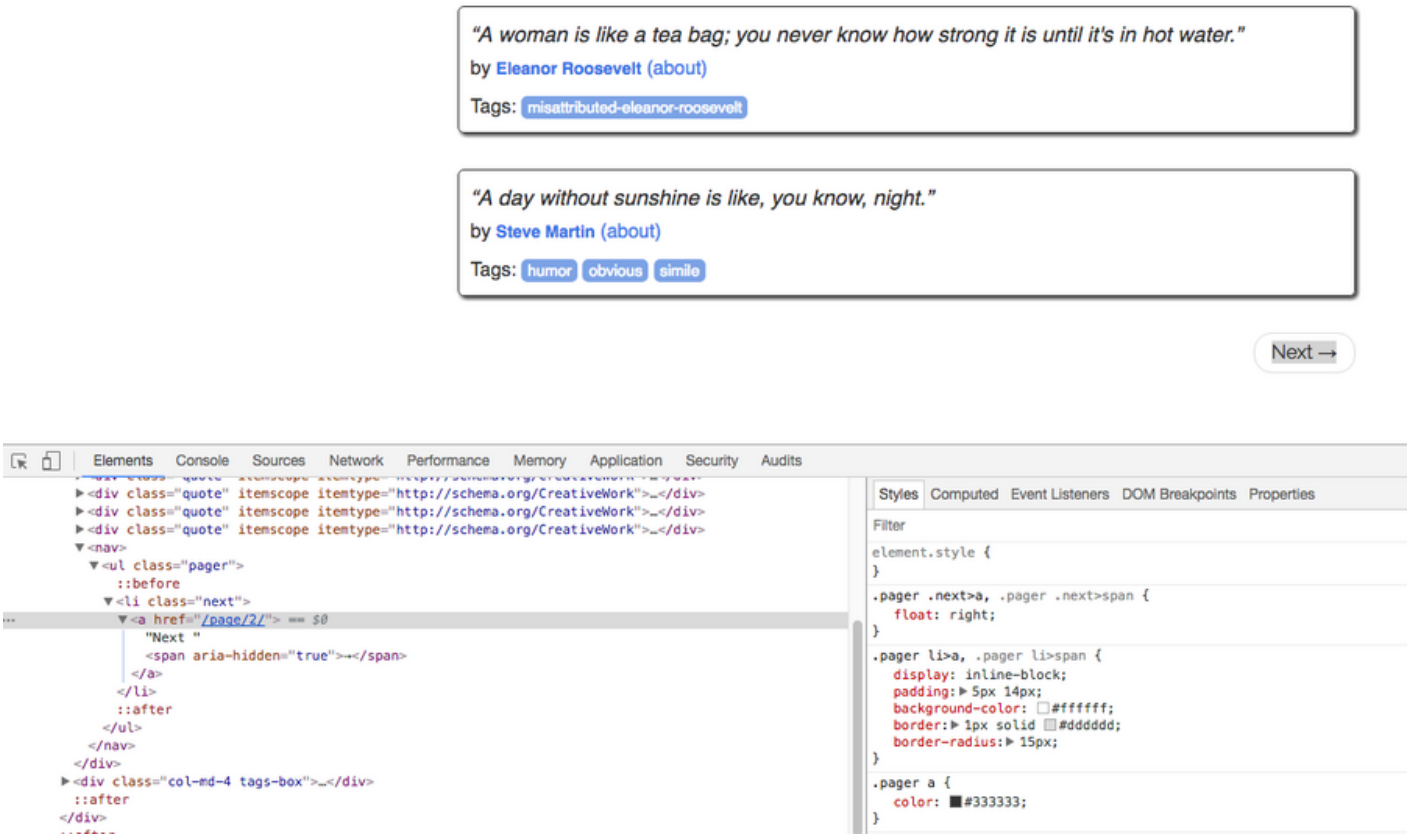
```
class QuotesSpiderSpider(scrapy.Spider):
    name = 'quotes_spider'
    allowed_domains = ['quotes.toscrape.com']
    start_urls = ['http://quotes.toscrape.com/']

    def parse(self, response):
        quotes = response.xpath("//div[@class='quote']")
        for quote in quotes:
            text = quote.xpath(
                ".//span[@class='text']/text()").extract_first()
            author = quote.xpath(
                ".//small//text()").extract_first()
            yield {'quote': text, "author": author}
```

We can run the spider `scrapy crawl quotes_spider` to check the output, The spider can extract the quotes and author info for us now!

Handle Pagination

In most cases, it is not enough to crawl the data from only one page, it makes sense to crawl data under one category, so you have to make the spider click the next page, extract the data, click the next page, over and over again. In this section, I will talk about how to make your spider click the next page automatically and extract all pages in Quotes to Scrape

To solve the pagination problem, first, we inspect the Dom element in DevTOols to figure out how the pagination work.



As we can see from the screenshot, the next page button is actually a HTML hyperlink, which means if we just follow the URL of the link and extract all data from the URLs, then we can get all quotes.

We can extract the URL of the next page at the end of `parse` method, send a new HTTP request to scrapy, so it will automatically help us get the response, when the response is available to handle, `parse` method would be called again to process the response. What you should know is that `parse` here is named callback function because this method is used to handle the response. If we do not set the callback function in the `Request`, then the current method would be used as callback function.

```
class QuotesSpiderSpider(scrapy.Spider):
    name = 'quotes_spider'
    allowed_domains = ['quotes.toscrape.com']
    start_urls = ['http://quotes.toscrape.com/']

    def parse(self, response):
        quotes = response.xpath("//div[@class='quote']")
        for quote in quotes:
            text = quote.xpath(
                ".//span[@class='text']/text()").extract_first()
            author = quote.xpath(
                ".//small//text()").extract_first()
            yield {'quote': text, "author": author}

        next_page_url = response.xpath("//li[@class='next']//a/@href").extract_first()
        if next_page_url:
            absolute_next_page_url = response.urljoin(next_page_url)
            yield scrapy.Request(absolute_next_page_url)
```

Here it the detailed scrapy cycle of this spider:

1. The first requests to perform are obtained from `start_urls`, here the spider start from `http://quotes.toscrape.com/`.

2. When `http://quotes.toscrape.com/` is downloaded by Scrapy, the default callback function `parse` is called, spider extract the data and yield back to Scrapy, and find the URL of the next page `http://quotes.toscrape.com/page/2/`, `yield` a new request which has the next page URL, the callback function of the new request is still the `parse` function.

3. When `http://quotes.toscrape.com/page/2/` is downloaded, `parse` is called again, quote data are extracted, next page URL found, new request are yield back to Scrapy.

4. The spider follows this pattern over and over again until it visits page 10 of this website. In that page, there is no next page button, which means spider will not find next_page_url, the value of `next_page_url` is `None`, so spider would not yield new request to Scrapy. The whole crawl process would end here.

Now we can run `scrapy crawl quotes_spider` to make the spider crawl all the quotes and author for us, below is part of the log.

```
2017-09-20 16:33:42 [scrapy.core.engine] DEBUG: Crawled (200) <GET http://quotes.toscrap
2017-09-20 16:33:42 [scrapy.core.scraper] DEBUG: Scraped from <200 http://quotes.toscrap
{'author': 'Albert Einstein', 'quote': '“The world as we have created it is a process of
2017-09-20 16:33:42 [scrapy.core.scraper] DEBUG: Scraped from <200 http://quotes.toscrap
{'author': 'J.K. Rowling', 'quote': '“It is our choices, Harry, that show what we truly
2017-09-20 16:33:42 [scrapy.core.scraper] DEBUG: Scraped from <200 http://quotes.toscrap
{'author': 'Albert Einstein', 'quote': '“There are only two ways to live your life. One
2017-09-20 16:33:42 [scrapy.core.scraper] DEBUG: Scraped from <200 http://quotes.toscrap
{'author': 'Jane Austen', 'quote': '“The person, be it gentleman or lady, who has not pl
2017-09-20 16:33:42 [scrapy.core.scraper] DEBUG: Scraped from <200 http://quotes.toscrap
{'author': 'Marilyn Monroe', 'quote': "“Imperfection is beauty, madness is genius and it
2017-09-20 16:33:42 [scrapy.core.scraper] DEBUG: Scraped from <200 http://quotes.toscrap
{'author': 'Albert Einstein', 'quote': '“Try not to become a man of success. Rather becc
2017-09-20 16:33:42 [scrapy.core.scraper] DEBUG: Scraped from <200 http://quotes.toscrap
{'author': 'André Gide', 'quote': '“It is better to be hated for what you are than to be
2017-09-20 16:33:42 [scrapy.core.scraper] DEBUG: Scraped from <200 http://quotes.toscrap
{'author': 'Thomas A. Edison', 'quote': "“I have not failed. I've just found 10,000 ways
2017-09-20 16:33:42 [scrapy.core.scraper] DEBUG: Scraped from <200 http://quotes.toscrap
{'author': 'Eleanor Roosevelt', 'quote': "“A woman is like a tea bag; you never know how
2017-09-20 16:33:42 [scrapy.core.scraper] DEBUG: Scraped from <200 http://quotes.toscrap
{'author': 'Steve Martin', 'quote': '“A day without sunshine is like, you know, night.”'
2017-09-20 16:33:42 [scrapy.core.engine] DEBUG: Crawled (200) <GET http://quotes.toscrap
2017-09-20 16:33:42 [scrapy.core.scraper] DEBUG: Scraped from <200 http://quotes.toscrap
{'author': 'Marilyn Monroe', 'quote': "“This life is what you make it. No matter what, y
2017-09-20 16:33:42 [scrapy.core.scraper] DEBUG: Scraped from <200 http://quotes.toscrap
{'author': 'J.K. Rowling', 'quote': '“It takes a great deal of bravery to stand up to ou
2
```

From the log of the terminal, you can notice that the data of page 1 is crawled first, then page 2, page 3, the spider has been closed after visiting page 10.

Conclusion

In this Scrapy tutorial, I have shown you how to develop a Scrapy spider from scratch. First, you should analyze the Dom element in DevTools, actually, DevTools is very powerful tool since it can help us inspect network such as ajax request or other network behavior. Second, you should test the code in Scrapy Shell, **Since Dom element can be modified by javascript, we some XPath work in browser plugin can fail in Scrapy shell and Scrapy, you need keep that in mind**.

To make the spider can handle pagination automatically, you must have a good understanding of Scrapy workflow, try to figure out the running cycle before writing code, which can save you a lot of time. **If you have any questions, just leave me to message here and I am happy to help people to master Scrapy**

To help user focus on the key part, I only paste part of the source code instead of the whole file in this tutorial, If you want source code which can run in your local env directly, just

```
git clone git@github.com:michael-yin/scrapy-spider-example.git

cd scrapy-spider-example

git checkout 25564c2

# If you can star my project, I would appreciate that
```