Scrapy Tutorial #6: Scrapy Shell Overview & Tips

Last updated on Feb 25 2019 by Michael Yin

Introduction:

This is the #6 post of my Scrapy Tutorial Series, in this Scrapy tutorial, I will talk about how to use Scrapy shell to help us extract data, and I will share with you some tips about how to make Scrapy shell more powerful.

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

Scrapy shell commands

Scrapy shell is a shell for us to extract data, it is strongly recommended to install `IPython` before using it. You can enter Scrapy shell using command `scrapy shell`, then you can see something like this.

```
$ scrapy shell
2017-08-25 10:18:44 [scrapy.utils.log] INFO: Scrapy 1.4.0 started (bot: scrapy_spider)
[s] Available Scrapy objects:
[s]   scrapy     scrapy module (contains scrapy.Request, scrapy.Selector, etc)
[s]   crawler    <scrapy.crawler.Crawler object at 0x10b5d84e0>
[s]   item       {}
[s]   settings   <scrapy.settings.Settings object at 0x10a0aff28>
[s] Useful shortcuts:
[s]   fetch(url[, redirect=True]) Fetch URL and update local objects (by default, redire
[s]   fetch(req)                  Fetch a scrapy.Request and update local objects
[s]   shelp()           Shell help (print this help)
[s]   view(response)    View response in a browser
```

In most cases, the output would be a little longer because other information such as middleware and extension would also be printed in the terminal, which you can ignore. In the output above, you can see, there are some Scrapy objects available in this IPython shell.

Here we use `fetch` command to make scrapy crawl some URL for us, for example, we use it to crawl page `http://quotes.toscrape.com/`

```
In [4]: fetch("http://quotes.toscrape.com/")
2017-08-25 10:28:11 [scrapy.core.engine] INFO: Spider opened
2017-08-25 10:28:12 [scrapy.core.engine] DEBUG: Crawled (404) <GET http://quotes.toscrap
2017-08-25 10:28:12 [scrapy.core.engine] DEBUG: Crawled (200) <GET http://quotes.toscrap
```

It seems scrapy try to crawl two URLs here, one is the robot.txt, the other is the URL we passed in the `fetch` method. If you have no idea what is robot.txt, take a look at this great post: What is robots.txt?. You can disable Scrapy crawl robot.txt every time by changing the `ROBOTSTXT_OBEY` to `False` in settings file.

After fetch finished, we can start to check the detail of the page by using `response` object. For example, we can get the HTTP headers and HTML source code by accessing `response.headers` and `response.body`.

Now we start to learn the most powerful feature of Scrapy shell. In Scrapy shell, we can enter our XPath expression or css expression to see if the code can extract the information as expected. If the code has some

bug, we can quickly test a new one in shell before copying the code into our spider source file. You can not image how many time can be saved by using this workflow.

```
In [17]: response.xpath("//div[@class='quote']/span[@class='text']").extract()
Out[17]:
['<span class="text" itemprop="text">“The world as we have created it is a process of ou

In [18]: response.xpath("//div[@class='quote']/span[@class='text']/text()").extract()
Out[18]:
['“The world as we have created it is a process of our thinking. It cannot be changed wi
```

In the code above, I try to modify XPath expression to make the code work as expected, when the output is right, I can copy the code directly to the spider source file, which is very convenient.

Many newbie developers like to change the code in spider source file, then run `scrapy crawl` to see if the spider work as expect, however, this workflow is very inefficient, if you want to become more productive, Scrapy shell is the tool you should master.

Make Your Scrapy Shell More Powerful

Very few people have talked about how to make you more efficient in Scrapy shell, that is why I decided to write this down to make spider developer DRY (Don't repeat yourself).

In Scrapy shell, some code snippet and function can be typed many times, is there any way to keep this DRY? Yes! We can create a python file to help us.

```
# create python file scrapy_env.py

from __future__ import print_function
import re, json , copy
import difflib

def compare(str1, str2):
    a = str1.splitlines()
    b = str2.splitlines()
    d = difflib.Differ()
    diff = d.compare(a, b)
    print('\n'.join(diff))
```

Then in Scrapy shell, you can `run scrapy_env.py` to run the python script, now you can start to use the command and function without manual import.

```
In [1]: run scrapy_env.py

In [2]: re
Out[2]: <module 're' from '/Users/michael_yin/.pyenv/versions/3.5.0/lib/python3.5/re.py'

In [3]: json
Out[3]: <module 'json' from '/Users/michael_yin/.pyenv/versions/3.5.0/lib/python3.5/jso
```

After you learned this way to make your Scrapy shell more powerful, you can start to extend the python script as you like, I believe this method can help you concentrate on the key point instead of typing.

Conclusion

In this Scrapy tutorial, I talked about how to use Scrapy shell, which is the most useful Scrapy command to help you quickly develop spider and test, what is more, I also shared a way to help people keep DRY in Scrapy shell. I hope people can start to use this great command after reading this tutorial.

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 e669614

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