doc Documentation

Release 0.0.1

michaelyin

Jan 24, 2018

Warning: This project is deprecated and it has been merged into Scrapy Tutorial Series: Web Scraping Using Python

CHAPTER 1

Basic concepts

Intro

What is contained in this project.

1. A list of tasks which covers many basic points in spider development, each task is a short exercise. You will be able to solve real complex problem after you solve the simple tasks step by step. This idea derive from code kata
2. Some advanced tips and notes which help you improve the development productivity, and it will introduce you some great tools.

Supplement instead of alternative of scrapy doc

Scrapy doc is a good start for people who want to learn to write spider by using scrapy. Since scrapy doc mainly focus on the components and concepts in scrapy, some points which make sense in spider development with scrapy are missed in the doc. That is why I created this project.

I did not talk much in componetns of scrapy in this doc. **It is strongly recommend user to read scrapy** official doc **first to have a basic understanding such as how to create spider and how to run spider in scrapy. You might can not get some points here if you have no idea how the spider work in scrapy**. If you have question for scrapy, please check it in official doc first.

Doc

http://scrapy-guru.readthedocs.io/en/latest/index.html

Support Platform

OSX, Linux, python 2.7+, python 3.4+

Get started

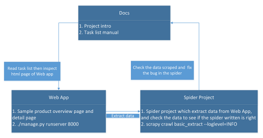
First, you should take a view of the workflow figure of this project to know how this project work and read basic concepts in doc.

Secondly user will choose one task in online doc of project and get started, it is recommended to solve the task in doc order considering the learning curve. User should create spider as doc asked and run the spider to get the data as expected. There is a sample spider callled `basic_extract` in the project, just follow it to create new one and troubleshoot If user can not make the spider to work, you can also check the working spider code in the solution repo which I will push later.

Thirdly user can get some advaned advise or tips in advanced topic , you can learn how to enhance your browser to make it more helpful in spider development or other stuff.

Workflow

Please click the image for better resolution.


Docs
1. Project intro
2. Task list manual
Read task list then inspect html page of Web app
Check the data scraped and fix the bug in the spider
Web App
1. Sample product overview page and detail page
2. ./manage.py runserver 8000
Extract data
Spider Project
1. Spider project which extract data from Web App, and check the data to see if the spider written is right
2. scrapy crawl basic_extract --loglevel=INFO

Project structure

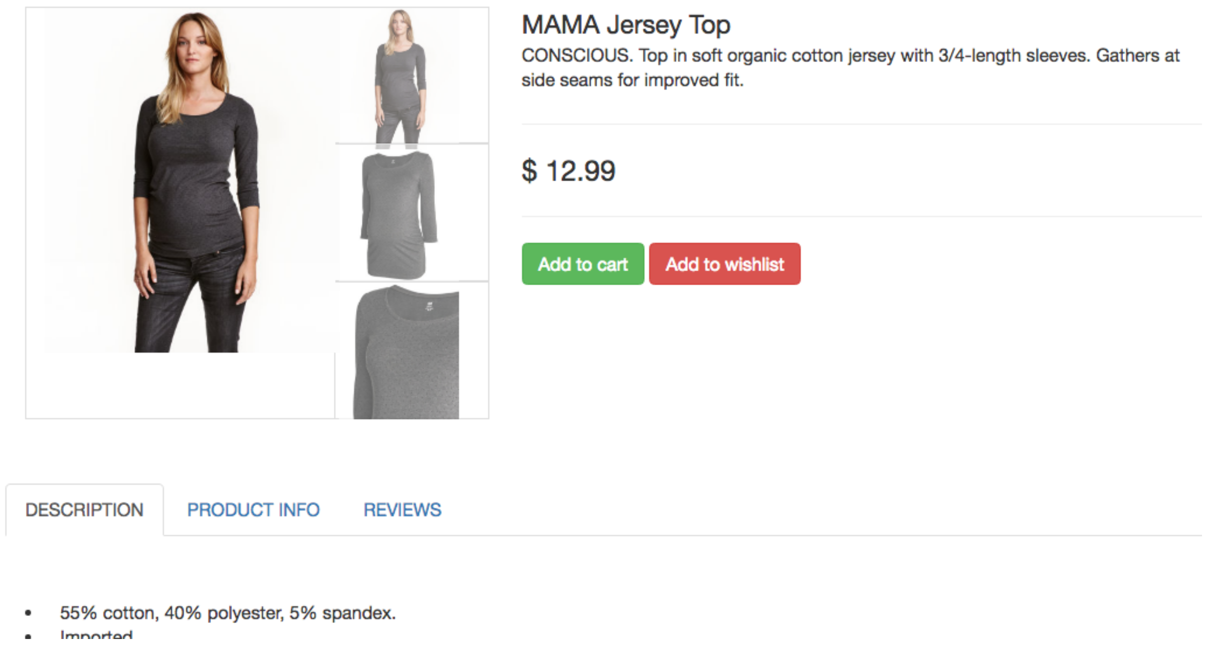
Here is the directory structure:

```
.
- docs
|    - Makefile
|    - build
|    - source
- requirements.txt
- spider_project
|    - release
|    - scrapy.cfg
|    - spider_project
- webapp
     - content
     - db.sqlite3
     - manage.py
     - staticfiles
     - templates
     - webapp
```

- `docs` contains the html documentation of this project
- `webapp` is a web application developed by `Django`, we can see this app as a website which show us product info and product links, and we need to write spider to extract the data from it.
- `spider_project` is a project based on `Scrapy` which we should write spider in it to extract data from `webapp`.

First glance

So here is an example product detail page, it is rendered by `webapp` mentioned above.


MAMA Jersey Top

CONSCIOUS. Top in soft organic cotton jersey with 3/4-length sleeves. Gathers at side seams for improved fit.

$ 12.99

- 55% cotton, 40% polyester, 5% spandex.
- Imported

Now according to task in the doc, we need to extract product title and desc from the product detail page

Here is part of spider code:

```
class Basic_extractSpider(scrapy.Spider):
    taskid = "basic_extract"
    name = taskid
    entry = "content/detail_basic"

    def parse_entry_page(self, response):
        item = SpiderProjectItem()
        item["taskid"] = self.taskid
        data = {}
        title = response.xpath("//div[@class='product-title']/text()").extract()
        desc = response.xpath("//section[@class='container product-info']//li/text()
↪").extract()
        data["title"] = title
        data["desc"] = desc

        item["data"] = data
        yield item
```

We can run the spider now, the spider will start to crawl from the `self.entry` and it will check the data scraped automatically. if the data scraped have some mistake, it will give the detail of the error and help you get the spider work as expect.

Installation

Clone the project

```
git clone https://github.com/michael-yin/scrapy_guru.git
```

Virtual environment

If you have no idea what virtual environment is, please take look at https://virtualenv.pypa.io/en/stable/installation/

After you created virtual env and activated it, just `pip install -r requirements.txt` to install the packages needed

Config

Assign port

You should assign a port of your localhost to make webapp to run. By default, we recommend you run web app at `8000` port

```
cd webapp
python manage.py runserver 8000
```

Config spider_project

```
cd spider_project/spider_project
# edit settings.py , remember to change the port number if webapp is not 8000
WEB_APP_PREFIX = "http://127.0.0.1:8000/"
```

Done

Now you are done with installation, please read *Read before you start*

Read before you start

Entry

Every task have an entry point where spider start to crawl, this entry point may be overview page which contains many product page, or it might be product detail page. or something else.

Taskid

The taskid is unique, each task have unique taskid, and we need to remember to set it in item yield from spider.

Note: entry and taskid only make sense in this project and they are not neede in normal scrapy spider

Item

The data scraped by spider should be filed in `SpiderProjectItem` located in `spider_project/spider_project/items.py`:

```
class SpiderProjectItem(scrapy.Item):
    # define the fields for your item here like:
    taskid = scrapy.Field()
    data = scrapy.Field()
```

The `taskid` field is the taskid you can get in each task, and the `data` is the data scraped, in most cases, the data field is a `dict` python type.

How to know if the spider work fine in each task?

Since user should create spider on himself, so spider contract might not be suitable to check if the data scraped is right.

After spider yield the item, the item pipeline will check if the scraped data is right and the result can be found in log file. This work is done by `SpiderProjectPipeline` automatically.

Done

Now you are ready to start developing spider, please start here *Basic extract*

Intro Introduction of this project

Installation How to install and config this project

Read before you start Something you should know before you start

CHAPTER 2

Advanced topic

Enhance your browser

Incognito mode

Incognito mode is also called `Private Browsing` in some browser. In this mode the browser does not save cookie and history. This property make it very easy for spider development.

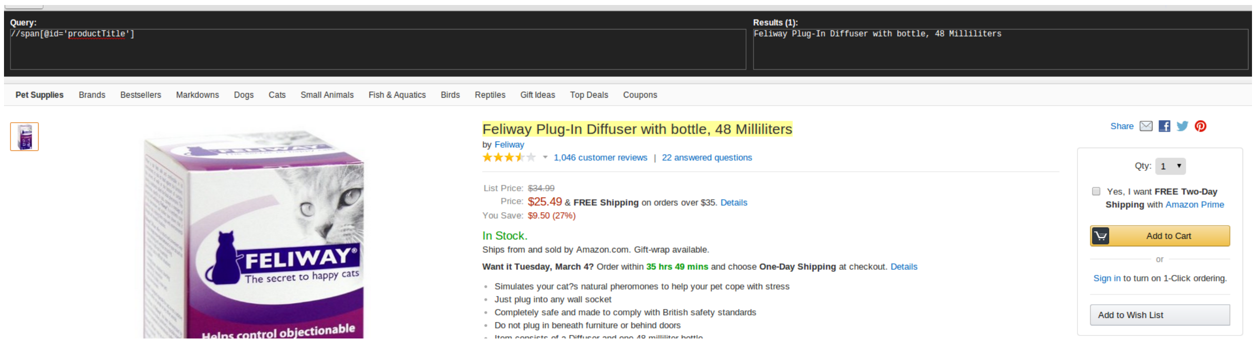
In many cases, we need to find out specific value in cookie to make spider to work, in incognito mode we can easily check the value and got to know how the website might work when spider crawling from a **fresh start**.

If you are using chrome, just follow the steps below

1. In the top-right corner of the browser window, select the Menu
2. Select New Incognito Window (computer)
3. A new window will open with the Incognito icon

Quickly test my xpath or css expression

There are several plugin in browser to support xpath extraction. You can try `XPath Helper` in google chrome, which will make it easy to evaluate xpath expression on webpage.



```
Query:
//span[@id='productTitle']
```

```
Results (1):
Feliway Plug-In Diffuser with bottle, 48 Milliliters
```

Feliway Plug-In Diffuser with bottle, 48 Milliliters
by Feliway
1,046 customer reviews | 22 answered questions

List Price: ~~$34.99~~
Price: $25.49 & **FREE Shipping** on orders over $35. Details
You Save: $9.50 (27%)

In Stock.
Ships from and sold by Amazon.com. Gift-wrap available.
Want it Tuesday, March 4? Order within **35 hrs 49 mins** and choose **One-Day Shipping** at checkout. Details

- Simulates your cat?s natural pheromones to help your pet cope with stress
- Just plug into any wall socket
- Completely safe and made to comply with British safety standards
- Do not plug in beneath furniture or behind doors

Yes, I want **FREE Two-Day Shipping** with Amazon Prime
Sign in to turn on 1-Click ordering.

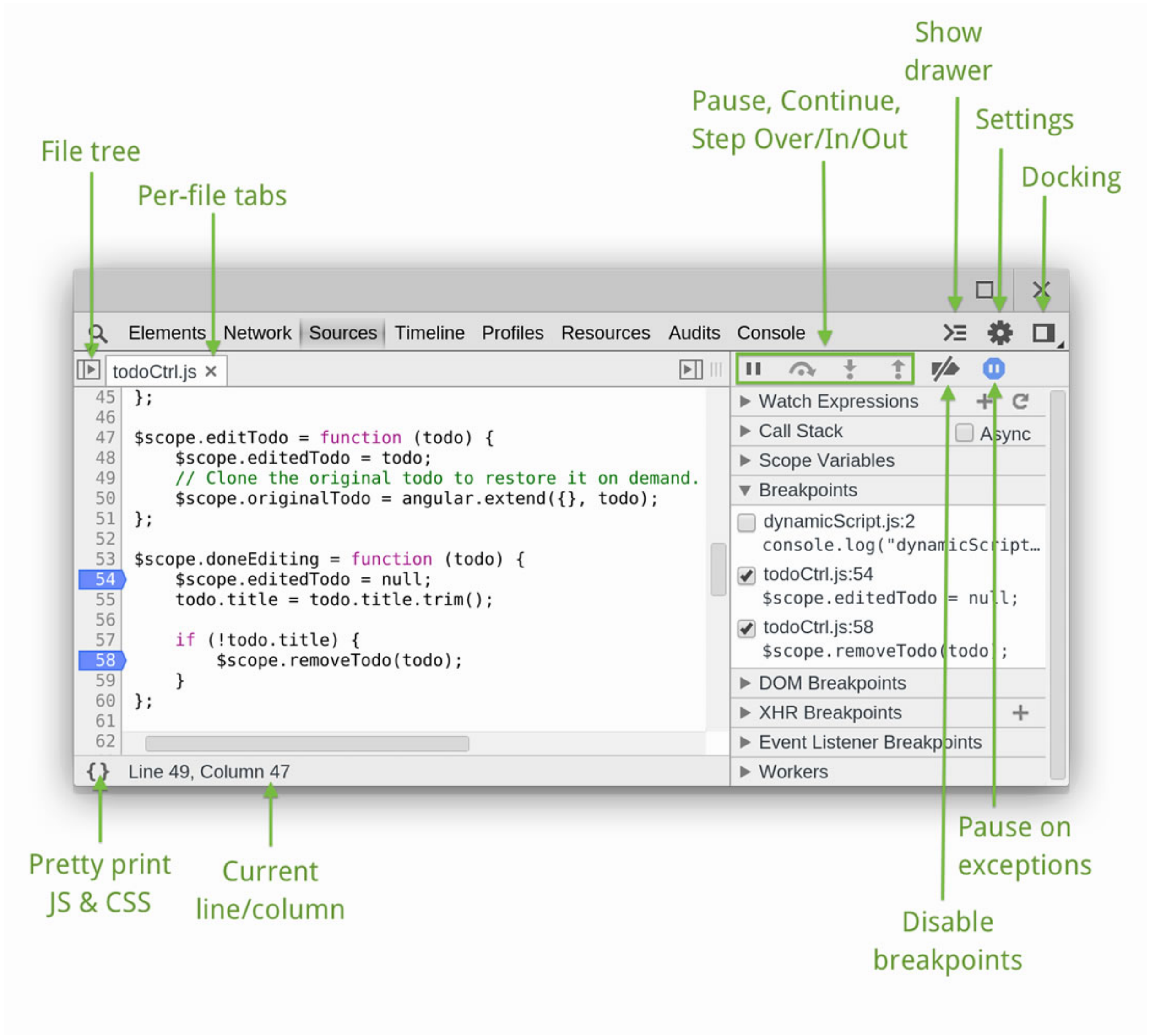
Here is how to use it. Press `ctrl+shift+x` to open XPath Helper, and you can see the input and output panel. You can type your xpath query string in the input panel, and the result of the xpath will show on the right side and in the web page the selected content will have a yellow backgroud, which is very easy to check if the xpath expression is right.

Note: What you should concern here is that in some cases the xpath espression which indeed work in browser can not work on raw html becuase some DOM element might been modified by js, so please test it in scrapy shell before write it in spider code.

If you do not want to install extention to make this done, google chrome has built-in support to query xpath and css expression. Take a look at `$()` and `$x()` in console, and follow this tutorials.

Use web dev tools

Here is overview screnshot about the web dev tools of google chrome, you can learn more here: https://developers.google.com/web/tools/chrome-devtools/



```
};

$scope.editTodo = function (todo) {
    $scope.editedTodo = todo;
    // Clone the original todo to restore it on demand.
    $scope.originalTodo = angular.extend({}, todo);
};

$scope.doneEditing = function (todo) {
    $scope.editedTodo = null;
    todo.title = todo.title.trim();

    if (!todo.title) {
        $scope.removeTodo(todo);
    }
};
```

Debug minified js file in chrome

chrome-debug-minified-js.

Enhance your terminal

Note: If you are newbie developer and have not much experience in terminal, it is not recommended for you to try content below, please focus on basic terminal concepts first. When you can master your terminal env, then you can use the tools below to improve your efficiency.

Tool

You should use good terminal tool before enter terminal world, `iterm2` in osx and `terminator` in linux are both good tools which worth trying.

Shell

You really need try zsh combined with `on-my-zsh`, which is a great project in github which have over 40000+ stars. Check it here

Terminal multiplexer

Terminal multiplexer can make you switch easily between several programs in one terminal. And this patten can make you focus on the work and make you more effieicent.

You can try `tmux` or `byobu`.

Incremental history search

When you develop spider, you need to run many commands and you will find out that most of them have common patten, and you might need to change some paras and rerun.

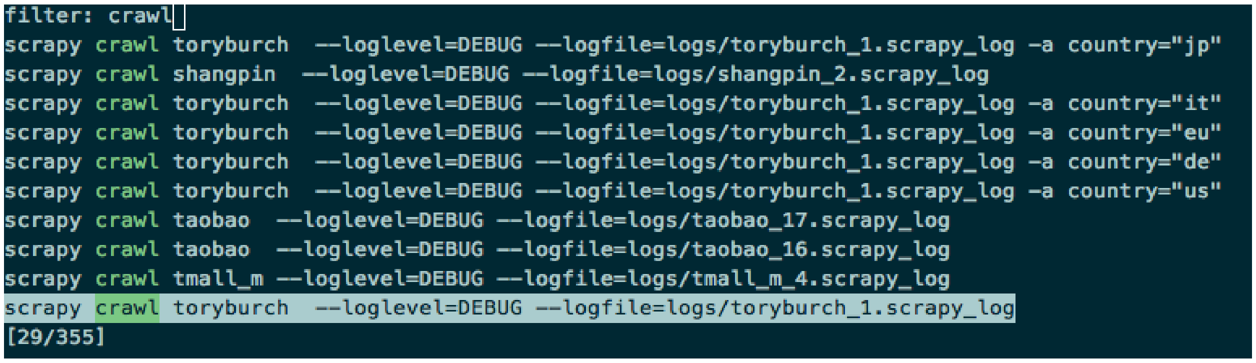
At first, you use `history` command and use `grep` to filter the command you want. The bad part of this approach is that you always need to enter number to select history command.

Here I want to introduce a tool which can make us handle history command more easily. This tool is `Zaw`, its homepage is https://github.com/zsh-users/zaw .

Its a tools help you select item from `source`. The source here can be something such as git log, hisotry, programs or others.

The only piece of Zaw that I introduce here is its excellent history search.

We can enter multiple keywords in Zaw and then flip through results until we fount what we want.



```
filter: crawl
scrapy crawl toryburch  --loglevel=DEBUG --logfile=logs/toryburch_1.scrapy_log -a country="jp"
scrapy crawl shangpin  --loglevel=DEBUG --logfile=logs/shangpin_2.scrapy_log
scrapy crawl toryburch  --loglevel=DEBUG --logfile=logs/toryburch_1.scrapy_log -a country="it"
scrapy crawl toryburch  --loglevel=DEBUG --logfile=logs/toryburch_1.scrapy_log -a country="eu"
scrapy crawl toryburch  --loglevel=DEBUG --logfile=logs/toryburch_1.scrapy_log -a country="de"
scrapy crawl toryburch  --loglevel=DEBUG --logfile=logs/toryburch_1.scrapy_log -a country="us"
scrapy crawl taobao  --loglevel=DEBUG --logfile=logs/taobao_17.scrapy_log
scrapy crawl taobao  --loglevel=DEBUG --logfile=logs/taobao_16.scrapy_log
scrapy crawl tmall_m --loglevel=DEBUG --logfile=logs/tmall_m_4.scrapy_log
scrapy crawl toryburch  --loglevel=DEBUG --logfile=logs/toryburch_1.scrapy_log
[29/355]
```

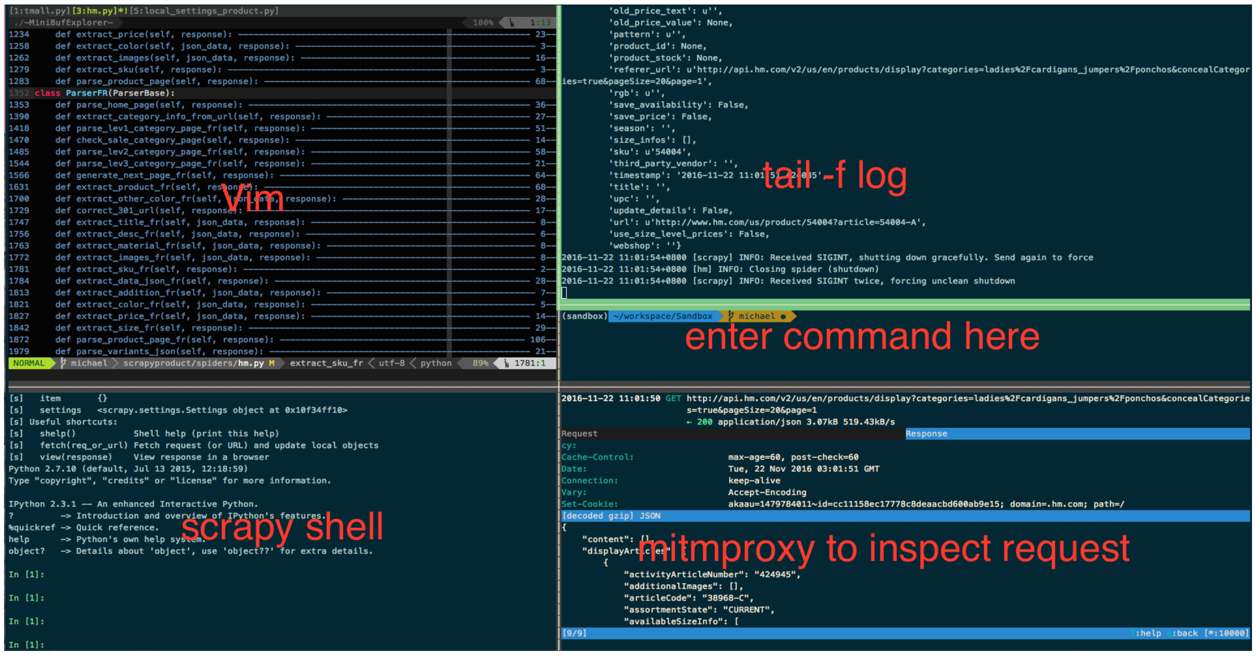
As you can see I enter `crawl` then the history will filtered and if I continue to enter `hm` then all the commands which have both `crawl` and `hm` will be filtered out, which is very handy.

Here is a great post talking about the Zaw hisotry search and config.

http://blog.patshead.com/2013/04/more-powerful-zsh-history-search-using-zaw.html

Workspace

Here is the screenshot of my workspace.



```
[1:tmall.py][3:hm.py]*![5:local_settings_product.py]
./-MiniBufExplorer-                                          100%    1:13
1234    def extract_price(self, response): ---------------------------- 23--
1258    def extract_color(self, json_data, response): ----------------- 3--
1262    def extract_images(self, json_data, response): ---------------- 16--
1279    def extract_sku(self, response): ------------------------------ 3--
1283    def parse_product_page(self, response): ----------------------- 68--
1352 class ParserFR(ParserBase):
1353    def parse_home_page(self, response): -------------------------- 36--
1390    def extract_category_info_from_url(self, response): ----------- 27--
1418    def parse_lev1_category_page_fr(self, response): -------------- 51--
1470    def check_sale_category_page(self, response): ----------------- 14--
1485    def parse_lev2_category_page_fr(self, response): -------------- 58--
1544    def parse_lev3_category_page_fr(self, response): -------------- 21--
1566    def generate_next_page_fr(self, response): -------------------- 64--
1631    def extract_product_fr(self, response): ----------------------- 68--
1700    def extract_other_color_fr(self, json_data, response): -------- 28--
1729    def correct_301_url(self, response): -------------------------- 17--
1747    def extract_title_fr(self, json_data, response): -------------- 8--
1756    def extract_desc_fr(self, json_data, response): --------------- 6--
1763    def extract_material_fr(self, json_data, response): ----------- 8--
1772    def extract_images_fr(self, json_data, response): ------------- 8--
1781    def extract_sku_fr(self, response): --------------------------- 2--
1784    def extract_data_json_fr(self, response): --------------------- 28--
1813    def extract_addition_fr(self, json_data, response): ----------- 7--
1821    def extract_color_fr(self, json_data, response): -------------- 5--
1827    def extract_price_fr(self, json_data, response): -------------- 14--
1842    def extract_size_fr(self, response): -------------------------- 29--
1872    def parse_product_page_fr(self, response): -------------------- 106--
1979    def parse_variants_json(self, response): ---------------------- 21--
NORMAL  michael  scrapyproduct/spiders/hm.py M  extract_sku_fr < utf-8 < python  89%  1781:1
```

```
          'old_price_text': u'',
          'old_price_value': None,
          'pattern': u'',
          'product_id': None,
          'product_stock': None,
          'referer_url': u'http://api.hm.com/v2/us/en/products/display?categories=ladies%2Fcardigans_jumpers%2Fponchos&concealCategor
ies=true&pageSize=20&page=1',
          'rgb': u'',
          'save_availability': False,
          'save_price': False,
          'season': '',
          'size_infos': [],
          'sku': u'54004',
          'third_party_vendor': '',
          'timestamp': '2016-11-22 11:01:51.224035',
          'title': '',
          'upc': '',
          'update_details': False,
          'url': u'http://www.hm.com/us/product/54004?article=54004-A',
          'use_size_level_prices': False,
          'webshop': ''}
2016-11-22 11:01:54+0800 [scrapy] INFO: Received SIGINT, shutting down gracefully. Send again to force
2016-11-22 11:01:54+0800 [hm] INFO: Closing spider (shutdown)
2016-11-22 11:01:54+0800 [scrapy] INFO: Received SIGINT twice, forcing unclean shutdown

(sandbox) ~/workspace/Sandbox  michael ●
```

```
[s]   item       {}
[s]   settings   <scrapy.settings.Settings object at 0x10f34ff10>
[s] Useful shortcuts:
[s]   shelp()           Shell help (print this help)
[s]   fetch(req_or_url) Fetch request (or URL) and update local objects
[s]   view(response)    View response in a browser
Python 2.7.10 (default, Jul 13 2015, 12:18:59)
Type "copyright", "credits" or "license" for more information.

IPython 2.3.1 -- An enhanced Interactive Python.
?         -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object', use 'object??' for extra details.

In [1]:

In [1]:

In [1]:

In [1]:
```

```
2016-11-22 11:01:50 GET http://api.hm.com/v2/us/en/products/display?categories=ladies%2Fcardigans_jumpers%2Fponchos&concealCategorie
                        s=true&pageSize=20&page=1
                        ← 200 application/json 3.07kB 519.43kB/s
Request                                                 Response
cy:
Cache-Control:                max-age=60, post-check=60
Date:                         Tue, 22 Nov 2016 03:01:51 GMT
Connection:                   keep-alive
Vary:                         Accept-Encoding
Set-Cookie:                   akaau=1479784011~id=cc11158ec17778c8deaacbd600ab9e15; domain=.hm.com; path=/
[decoded gzip] JSON
{
    "content": [],
    "displayArticles": [
        {
            "activityArticleNumber": "424945",
            "additionalImages": [],
            "articleCode": "38968-C",
            "assortmentState": "CURRENT",
            "availableSizeInfo": [
[9/9]                                                           ?:help q:back [*:10000]
```

You can see I have opened a lot of panels in single one tmux window, I can quickly switch between them and do not need to jump out my favorate terminal env.

Troubleshoot spider

Scrapy shell

scrapy shell is a scrapy command which provides us a interactive shell where we can test our code and check the output.

It is stronglly recommended to install ipython with scrapy. ipython offers introspection, rich media, shell syntax, tab completion, and history.

For example, when you try to solve *Basic extract* , you can use scrapy shell quickly test your code.

- enter `scrapy shell` to open scrapy shell
- use `fetch http://127.0.0.1:8000/content/detail_basic` to get web page we want to analyze
- you can use `response.xpath` and `response.css` to test your expression in this web page, this can quickly find out the error, in this task, we can test `response.xpath("//div[@class='product-title']/text()").extract()`
- if the output is right, just copy the code in spider.

Scrapy parse

`scrapy parse` can help you test your method to make sure it work fine. Here is a example

```
scrapy parse --spider=basic_extract --loglevel=DEBUG -c parse_entry_page "http://127.
↪0.0.1:8000/content/detail_basic"
```

Make sure to use this to test your methods and it will save your a lot of time later, trust me!

Print log

Log is the only way to figure out what really happend when scrawl working. So I will give you some suggestion about the log.

The spider may raise exception when working due to the different html structure or something else, you might need to log the entire html souce code to analyze later. Here is an example, we print the response.body in log to troubleshoot.

```
self.logger.debug('error occur at ' + response.url)
self.logger.debug(response.body)
```

PDB

I do not understand why scrapy not recommend pdb over scrapy shell, in my opinion pdb is the best debuging tool when developing spider.

You can set breakpoint, conditional breakpoint in spider, inspect variable in pdb shell, and print traceback, which make debug work easier. Somebody might not know `ipdb`. I must say ipdb add some more usefull feature to `pdb` and it is worthile to give it a try.

Take a look at this great post

Mitmproxy

Intro

mitmproxy is an interactive, SSL-capable intercepting proxy with a console interface.

Pros and cons

Here is list of the popluar network tools user use to inspect http traffic.

wireshark	Fiddler	Charlesproxy	Mitmproxy
Win, OSX, Linux	Win	OSX	Win, OSX, Linux
GUI(Qt, GTK)	GUI(Native)	GUI(Native)	Console

As we can see, mitmproxy has no gui interface for newbie user to inspect http request, but in my eyes this is the great advantage because we can launch mitmproxy in terminal and quickly detect http request

How to use mitmproxy to speed the development of spider

Terminal

Mitmproxy work fine in my terminal env and I can quickly switch between tools which used in spider deveopment. You can read *Enhance your terminal*.

In mitmproxy I can quickly check content of http requests by entering some key.

Filter

Sometime you know the website might use some ajax request to get the data you want to scrape, so you go to the network panel of your spider try to check the detail of the request. After you click 10+ links, your are tired and hope some tool can save your life here.

mitmproxy can really help you here.

You can write filter expression to make mitmproxy filter the request based on the expression. For example, if you want to filter http request which have content `MAMA Jersey Top`, you can use the expression `~b "MAMA Jersey Top"`, or you can filter the http reqeusrt based on url, response.body, response.header

You can give it a try and I promise you will be surprised.

If you want to analyze https

When you start to use mitmproxy, it is stronglly recommened to install the CA certificate before you start because if you did not install the CA certificate you can not make mitmproxy inspect https request.

Take a look at this after install.

http://docs.mitmproxy.org/en/stable/certinstall.html

Enhance your browser How to enhance your browser to make it help you develope spider

Enhance your terminal How to enhance your terminal shell.

Troubleshoot spider How to troubleshoot your scrapy spider.

Mitmproxy How to inspect your http request.

CHAPTER 3

Task List

Basic extract

Goal

There are mainly two ways in web crawling package such as scrapy, beautifulsoup to extract info from html, one is `css` and the other is `xpath`, you can learn css here and xpath here

I must say there is not much difference between them, you can pick the one you prefer in spider developing.

You might need quickly test your xpath or css expression in your browser, check it here *Quickly test my xpath or css expression*

I have created basic_extract spider to show you how to use it in this project. You are free to delete it and create your own or modify it.

Entry

If you have no idea what entry and taskid is, check *Read before you start*

Remember to config `WEB_APP_PREFIX` which located in spider_project/spider_project/settings.py

Entry:

```
content/detail_basic
```

If your webapp is working on 8000, click the link below

http://127.0.0.1:8000/content/detail_basic

Taskid

Taskid:

```
basic_extract
```

Detail of task

Once you finish the coding just run `scrapy crawl basic_extract --loglevel=INFO` to check the output, this command is a scrapy command which run spider which have name basic_extract and set the logging level to INFO. This command will run the spider, crawl the data and check the data. Results will show up in terminal

In this task we extract the title, description from the entry page (above), the final data should be:

```
[{
    "data": {
        "desc": ["55% cotton, 40% polyester, 5% spandex.", "Imported", "Art.No. 85-
↪8023"],
        "title": ["MAMA Jersey Top"]
    },
    "taskid": "basic_extract"
}]
```

Advanded

Note: What you should concern in this task is that in some cases the xpath espression which indeed work in your browser can not work on raw html becuase some DOM element might been modified by js, so please test it in scrapy shell before write it in spider code.

Json extract

Goal

Recently many websites start to use json format to save data. So we need to learn how to handle this situation.

Entry

If you have no idea what entry and taskid is, check *Read before you start*

Remember to config `WEB_APP_PREFIX` which located in spider_project/spider_project/settings.py

Entry:

```
content/detail_json
```

If your webapp is working on 8000, click the link below

http://127.0.0.1:8000/content/detail_json

Taskid

Taskid:

```
json_extract
```

Detail of task

In this task we try to crawl product title and price info. You should find out that the value returned by xpath is not the one you see in your brower. Because javascript have change that.

Once you finish the coding just run `scrapy crawl json_extract --loglevel=INFO` to check the output

The final data should be:

```
[{
    "data": {
        "price": "$ 13.99",
        "title": "MAMA Jersey Top"
    },
    "taskid": "json_extract"
}]
```

Advanded

Note: Sometime there are some unicode char in the raw json string which might cause program raise UnicodeDecodeError. You should remember before runing json.loads, make the the json_data is decoded as unicode string type. If there are some syntax error in json string, you can use json lint to help you figure out where the error is.

Ajax extract

Goal

Recently many websites get product info through ajax request so it make sense for us to quickly figure out how it works and find a way to get the real data.

If you have no idea what ajax is, read it

Entry

If you have no idea what entry and taskid is, check *Read before you start*

Remember to config `WEB_APP_PREFIX` which located in spider_project/spider_project/settings.py

Entry:

```
content/detail_ajax
```

If your webapp is working on 8000, click the link below

http://127.0.0.1:8000/content/detail_ajax

Taskid

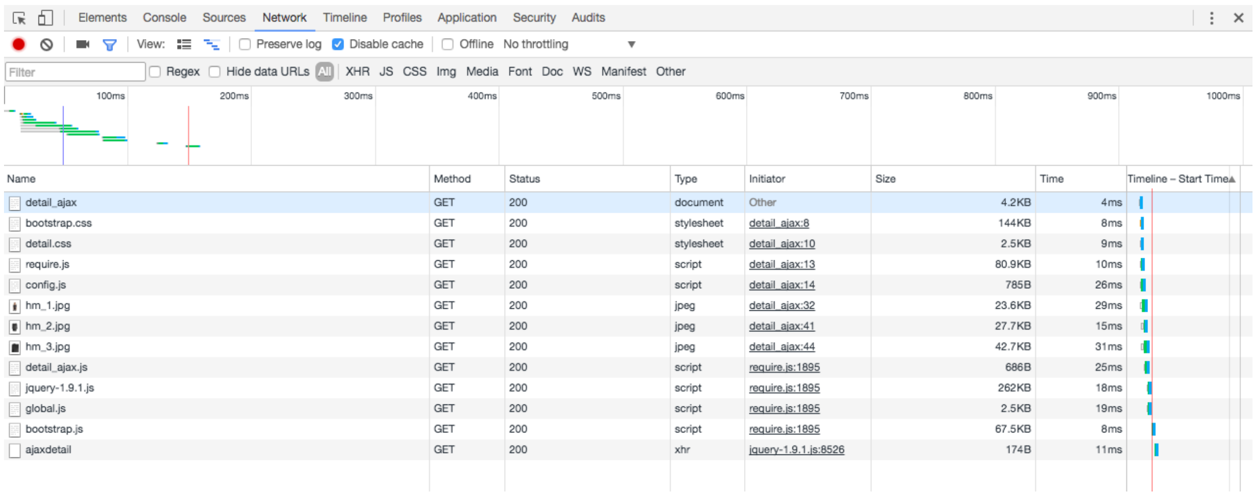
Taskid:

```
ajax_extract
```

Detail of task

In this task we try to crawl product title and price info. You should find out that the value in html is not the one you see in your brower.

You can check the network panel of your brower to filter out ajax url the browser used and try to implement it in your spider. You should yield a request in parse_entry_page method to minic ajax request.



Name	Method	Status	Type	Initiator	Size	Time
detail_ajax	GET	200	document	Other	4.2KB	4ms
bootstrap.css	GET	200	stylesheet	detail_ajax:8	144KB	8ms
detail.css	GET	200	stylesheet	detail_ajax:10	2.5KB	9ms
require.js	GET	200	script	detail_ajax:13	80.9KB	10ms
config.js	GET	200	script	detail_ajax:14	785B	26ms
hm_1.jpg	GET	200	jpeg	detail_ajax:32	23.6KB	29ms
hm_2.jpg	GET	200	jpeg	detail_ajax:41	27.7KB	15ms
hm_3.jpg	GET	200	jpeg	detail_ajax:44	42.7KB	31ms
detail_ajax.js	GET	200	script	require.js:1895	686B	25ms
jquery-1.9.1.js	GET	200	script	require.js:1895	262KB	18ms
global.js	GET	200	script	require.js:1895	2.5KB	19ms
bootstrap.js	GET	200	script	require.js:1895	67.5KB	8ms
ajaxdetail	GET	200	xhr	jquery-1.9.1.js:8526	174B	11ms

Once you finish the coding just run `scrapy crawl ajax_extract --loglevel=INFO` to check the output

The final data should be:

```
[{
    "data": {
        "price": "$ 12.99",
        "title": "MAMA Jersey Top"
    },
    "taskid": "ajax_extract"
}]
```

Advanded

Note: You must be able to use tools of browser to analyze http request. see *Use web dev tools*.

Ajax Header

Goal

Some backend systems would check http header to block some abnormal request. In this case we need to make sure the request from our spider will hsave the same http header as we see in the browser.

You should check the http header in the browser first and then implement it in your spider.

If you have no idea what http header is , check here

Entry

If you have no idea what entry and taskid is, check *Read before you start*

Remember to config `WEB_APP_PREFIX` which located in spider_project/spider_project/settings.py

Entry:

```
content/detail_header
```

If your webapp is working on 8000, click the link below

http://127.0.0.1:8000/content/detail_header

Taskid

Taskid:

```
ajax_header
```

Detail of task

In this task we try to crawl product title and price info. You should find out that the value in html is not the one you see in your brower.

Once you finish the coding just run `scrapy crawl ajax_header --loglevel=INFO` to check the output

The final data should be:

```
[{
    "data": {
        "price": "$ 12.99",
        "title": "MAMA Jersey Top"
    },
    "taskid": "ajax_header"
}]
```

Advanded

Note: Actually you can use some proxy tools to help you analyze http request easier, visit *Mitmproxy*.

Meta StoreInfo

Goal

Sometime if you want to pass value between more than one http pages, then you will need response.meta as a tmp datatable.

You can learn more here

Entry

If you have no idea what entry and taskid is, check *Read before you start*

Remember to config `WEB_APP_PREFIX` which located in spider_project/spider_project/settings.py

Entry:

```
content/detail_header
```

If your webapp is working on 8000, click the link below

http://127.0.0.1:8000/content/detail_header

Taskid

Taskid:

```
meta_storeinfo
```

Detail of task

In this task we try to crawl product title, product description, price info.

You should be concern that the description is in the raw html, but the title and price info should be given by ajax. To deal with this situation, you should save the description in response.meta and pass it in request.

The final data should be:

```
[{
    "data": {
        "price": "$ 12.99",
        "description": ["55% cotton, 40% polyester, 5% spandex.", "Imported", "Art.No.
↪ 85-8023"],
        "title": "MAMA Jersey Top"
    },
    "taskid": "meta_storeinfo"
}]
```

Ajax Cookie

Goal

It is importtant to analyze cookies of http request in many cases

If you have no idea what cookie is , read it

If you are using chrome, try visiting chrome://settings/cookies , then you can inspect all cookies in your browser.

Entry

If you have no idea what entry and taskid is, check *Read before you start*

Remember to config `WEB_APP_PREFIX` which located in spider_project/spider_project/settings.py

Entry:

```
content/detail_cookie
```

If your webapp is working on 8000, click the link below

http://127.0.0.1:8000/content/detail_cookie

Taskid

Taskid:

```
ajax_cookie
```

Detail of task

In this task we try to crawl product title, product description, price info.

After some tests, you might find out it is hard to make the spider get the data through ajax, so you need to dive into the detail of the ajax request.

You need to make sure the url, http header, cookie values are all reasonable.

Once you finish the coding just run `scrapy crawl ajax_cookie --loglevel=INFO` to check the output

The final data should be:

```
[{
    "data": {
        "price": "$ 20.00",
        "description": ["55% cotton, 40% polyester, 5% spandex.", "Imported", "Art.No.
↪ 85-8023"],
        "title": "Congratulations"
    },
    "taskid": "ajax_cookie"
}]
```

Advanded

Note: When dealing with cookies in browser, it seems a fresh start without any cookie is a good start. see *Incognito mode*.

Ajax Sign

Goal

Many websites now minified js file when deployment, so we should learn how to analyze the minmized code in browser and try to debug it in some cases to figure out the workflow. This process is like disassemble in reverse engineering.

Entry

If you have no idea what entry and taskid is, check *Read before you start*

Remember to config `WEB_APP_PREFIX` which located in spider_project/spider_project/settings.py

Entry:

```
content/detail_sign
```

If your webapp is working on 8000, click the link below

http://127.0.0.1:8000/content/detail_sign

Taskid

Taskid:

```
ajax_sign
```

Detail of task

In this task we try to crawl product title, product description, price info.

You found out that the ajax url used `sign` in the url but you have no idea where it is from, and it seems the js file `detail_sign.js` is minified.

Once you finish the coding just run `scrapy crawl ajax_sign --loglevel=INFO` to check the output

The final data should be:

```
[{
    "data": {
        "price": "$ 20.00",
        "description": ["55% cotton, 40% polyester, 5% spandex.", "Imported", "Art.No.
↪ 85-8023"],
        "title": "Congratulations"
    },
    "taskid": "ajax_sign"
}]
```

Advanded

Note: Learn how to pretty print minified js and debug the minified js, chrome-debug-minified-js

Regex extract

Goal

Regex is a very powerful tool when dealing with text, you have no reason to ignore it. A regulare expression is a string describing a certain amount of texts. If you have no knowledge of regex, you should learn it before you begin this task.

You can read this great tutorials here , once you have learned regex, you can try this online regex tool to quickly test your regex written in python.

Entry

If you have no idea what entry and taskid is, check *Read before you start*

Remember to config `WEB_APP_PREFIX` which located in spider_project/spider_project/settings.py

Entry:

```
content/detail_regex
```

If your webapp is working on 8000, click the link below

http://127.0.0.1:8000/content/detail_regex

Taskid

Taskid:

```
regex_extract
```

Detail of task

In this task we try to crawl product title and price info. Since the data in js is not very easy to extract, regex is a good tool to handle this situation.

Once you finish the coding just run `scrapy crawl regex_extract --loglevel=INFO` to check the output

The final data should be:

```
[{
    "data": {
        "title": "Regex is important",
        "price": "$ 13.99"
    },
    "taskid": "regex_extract"
}]
```

List page and products extract

Goal

In most cases, your spider should start from a list index page and crawl all the product links in the page, so in this task you will learn how to write spider to work in this case.

Entry

If you have no idea what entry and taskid is, check *Read before you start*

Remember to config `WEB_APP_PREFIX` which located in spider_project/spider_project/settings.py

Entry:

```
content/list_basic/1
```

If your webapp is working on 8000, click the link below

http://127.0.0.1:8000/content/list_basic/1

Taskid

Taskid:

```
list_extract
```

Detail of task

There are **10** products in list page 1, you should extract all product links first, and for each product, you should crawl title, price, and sku. Sku can be extracted from product url

Once you finish the coding just run `scrapy crawl list_extract --loglevel=INFO` to check the output

The final data is too long, this is part of it:

```
[{
    "data": {
        "sku": "0184140017",
        "price": ["$14.99"],
        "title": ["Washed linen table runner-Anthracite grey"]
    },
    "taskid": "list_extract"
}, {
    "data": {
        "sku": "0184140016",
        "price": ["$14.99"],
        "title": ["Washed linen table runner-Grey"]
    },
    "taskid": "list_extract"
}, {
    "data": {
        "sku": "0184124001",
        "price": ["$19.99"],
        "title": ["Lace table runner-White"]
    },
    "taskid": "list_extract"
}]
```

List page and pagination extract

Goal

The only difference between this task and *List page and products extract* is that thie task also needs deal with pagination

Entry

If you have no idea what entry and taskid is, check *Read before you start*

Remember to config `WEB_APP_PREFIX` which located in spider_project/spider_project/settings.py

Entry:

```
content/list_basic/1
```

If your webapp is working on 8000, click the link below

http://127.0.0.1:8000/content/list_basic/1

Taskid

Taskid:

```
list_extract_pagination
```

Detail of task

There are about 100+ products in all list pages, you should crawl them all, for each product, you should crawl title, price, and sku. Sku can be extracted from product url

The final data is too long, this is part of it:

```
[{
    "data": {
        "sku": "0447183001",
        "price": ["$14.99"],
        "title": ["Textured trinket box-White"]
    },
    "taskid": "list_extract_pagination"
}, {
    "data": {
        "sku": "0463014001",
        "price": ["$39.99"],
        "title": ["Cotton terry dressing gown-Light grey"]
    },
    "taskid": "list_extract_pagination"
}]
```

Basic extract Understand the spider workflow and basic xpath syntax.

Json extract Learn to use json module to extract json data.

Ajax extract Learn to inspect ajax request.

Ajax Header Learn to inspect http header of ajax request.

Meta StoreInfo Learn to pass additional data to callback functions

Ajax Cookie Learn to analyze cookie of http request.

Ajax Sign Learn to analyze minified js and debug code in browser.

Regex extract Learn to use regex expression to extract info.

List page and products extract Learn to extract products from list pages.

List page and pagination extract Learn to extract info from list page and handle pagination.