爬虫：从网上下载并提取项目感兴趣的海量数据

开发爬虫的方法很多：比如使用python自带的urllib和re；专业的爬虫开发框架(Scrapy)整合了大量工具包（比如Twisted，lxml，cssselect等）。

既然有跑车，为何还要步行去远方。

步骤：

step1发送url请求，获取相应

step2使用某种技术（如正则表达式，XPath）来提取页面中感兴趣的信息

step3识别响应中的链接信息，顺着链接递归执行step1-step3

step4 使用多线程有效管理网络通信交互

re识别响应页面的链接信息不如XPath，XPath的相关知识参见《疯狂XML讲义》

# 了解Scrapy

pip install scrapy

pyOpenSSL-22.0.0 python用于支持ssl（security socket layer）的包

cryptography: python用于加密的库

lxml-4.8.0 一个处理xml，html文档的库，比python内置的xml模块好用

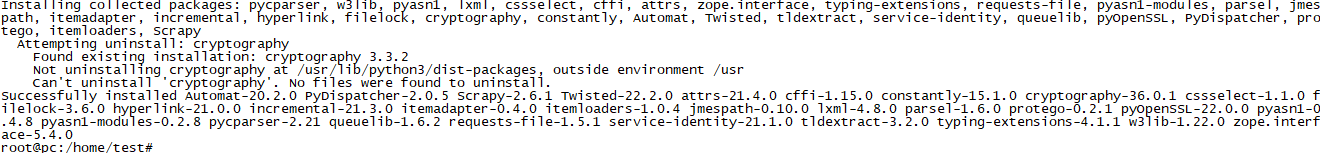
cssselect: python用于处理css选择器的扩展包

Twisted-22.2.0 为python提供基于事件驱动的网络引擎包

zope.interface-5.4.0 ：为python缺少接口而提供扩展的库

CFFI：python用于调用c的接口库





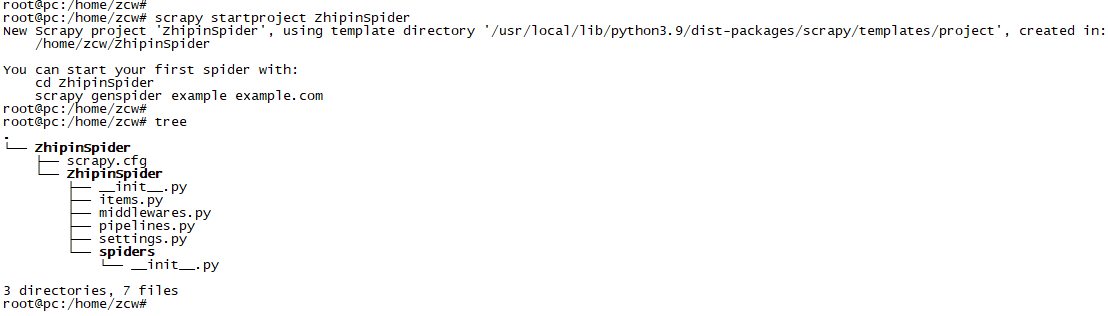
python3 -m pydoc -p 8899

http://localhost:8899/



# 使用爬虫爬取分和分析招聘信息

scrapy是命令，startproject是子命令，用于创建项目



scrapy.cfg: 项目总配置文件，通常无需修改

ZhipinSpider: 项目的python模块，程序将此处导入python代码

ZhipinSpider/settings.py: 项目配置文件，在该文减重进行项目相关的配置。

ZhipinSpider/items.py: 用于定义项目用到的Item类。Item类就是一个DTO（数据传输对象），通常就是定义N个属性，该类需要由开发者来定义。

ZhipinSpider/pipelines.py: 项目的管道文件，它负责处理爬取到的信息（用户编写）

ZhipinSpider/spiders: 在该目录下存放项目所需的蜘蛛 —— 蜘蛛负责抓取项目感兴趣的信息。

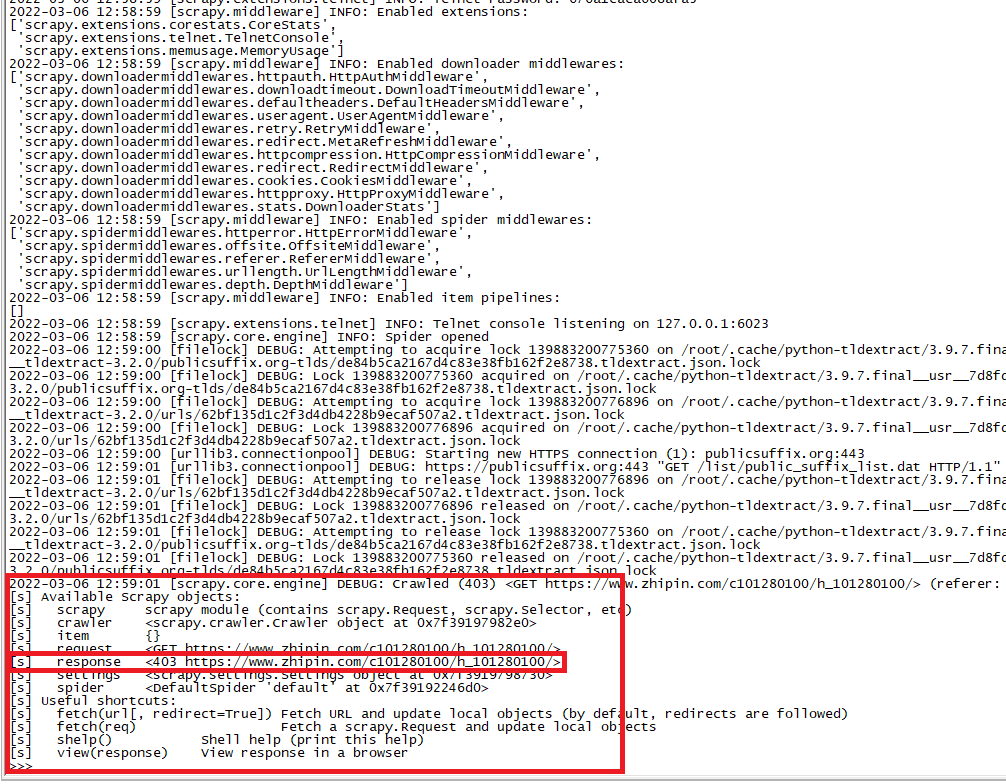
调用中间件 –>下载中间件 -> 蜘蛛中间件 ----Item----> pipeline

蜘蛛：负责从下载数据中提取有效信息，提取到的信息会由Scrapy引擎以Item对象的形式转交给Pipeline。

Pipeline：收到Item对象(包含蜘蛛提取的信息)后，将信息写入文件或数据库。

# 使用shell抓取网页

scrapy shell https://www.zhipin.com/c101280100/h\_101280100/



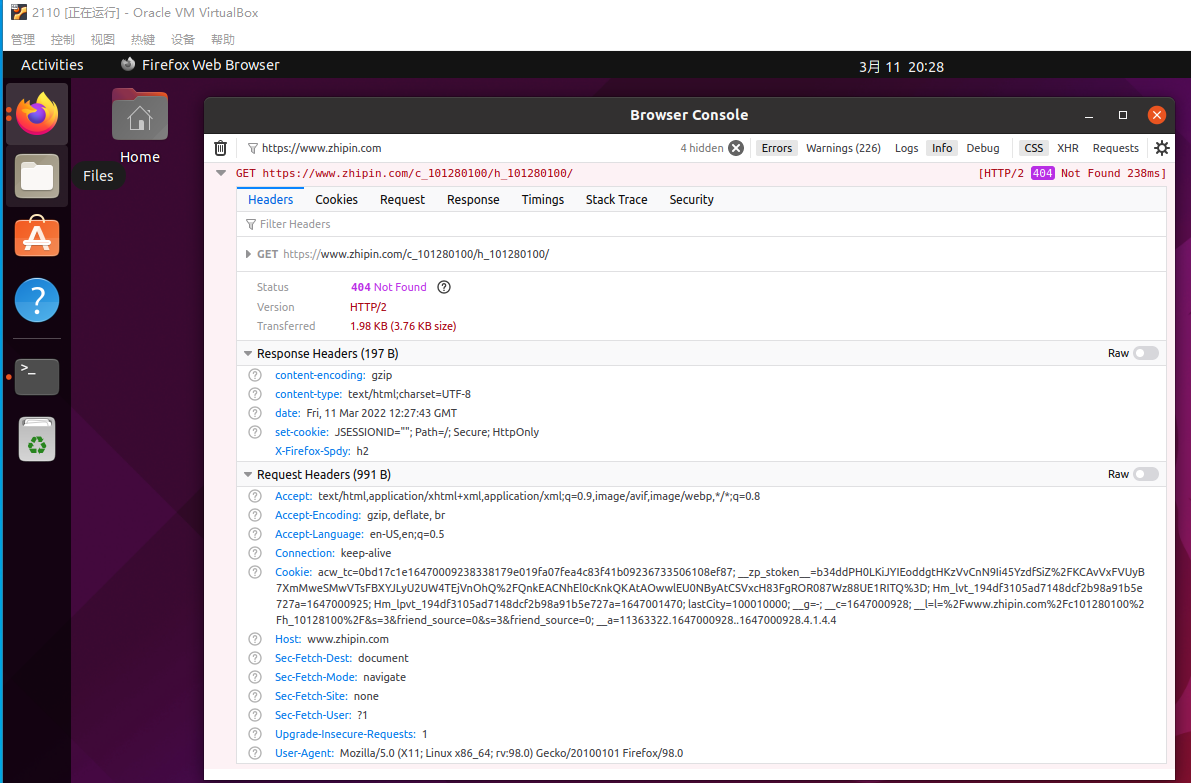
403说明目标网站开启了防爬虫

为了让Scrapy伪装成浏览器，需要在发送请求时设置User-Agent头，将User-Agent头的值设置为真实浏览器发送请求的User-Agent。

点击浏览器右上角 -> More tools -> Browser Console

在console上地址栏填写待过滤的url：<https://www.zhipin.com/>

然后在浏览器上输入url: http://www.zhipin.com/



进而得到User-Agent

User-Agent

Mozilla/5.0 (X11; Linux x86\_64; rv:98.0) Gecko/20100101 Firefox/98.0

# XPath和css选择器

https://www.runoob.com/xpath/xpath-tutorial.html

XPath最实用的简化写法

|  |  |
| --- | --- |
| 表达式 | 描述 |
| nodename | 选取此节点的所有内容 |
| / | 匹配根节点 |
| // | 匹配任意位置的节点 |
| . | 匹配当前节点 |
| .. | 匹配父节点 |
| @ | 匹配 |

XPath使用路径表达式在xml文档中进行导航

除了XPath，Scrapy也支持使用CSS选择器来匹配节点，但XPath匹配能力更强。

response.xpath('//div[@class="job-primary"]/div/h3/a/div/test()').extract()

response.css('div.job-primary>div.info-primary>h3.name span').extract()

# 普通爬虫

## 例1 boss直聘

step1: 创建项目

scrapy startproject ZhipinSpider

step2: 定义Item类

vi /home/test/ZhipinSpider/ZhipinSpider/items.py

step3: 开发Spider

cd /home/test/ZhipinSpider

#scrapy genspider job\_position "zhipin.com"

scrapy genspider job\_position "127.0.0.1"

vi /home/zcw/ ZhipinSpider/spiders/job\_position.py

step4: 配置User-Agent

vim /home/zcw/ ZhipinSpider/ZhipinSpider/settings.py

DEFAULT\_REQUEST\_HEADERS = {

'User-Agent' : 'Mozilla/5.0 (X11; Linux x86\_64; rv:98.0) Gecko/20100101 Firefox/98.0',

}

#获取User-Agent方法参见[使用shell抓取网页]

step5：创建测试资源

#python3直接访问<https://www.zhipin.com/c101280100/h_101280100/>

#得不到期望的html页面，所以通过查看源代码，将网页另存为test.html

apt install apache2 -y

systemctl status apache2

cp /home/test/test.html /var/www/html/

<http://127.0.0.1/test.html>

diff ZhipinSpider.bk ZhipinSpider -r > /home/test/new.patch

step6: 启动测试

cd /home/test/ZhipinSpider

scrapy crawl job\_position









## 例2 boss直聘: page-next超链接

<div class="page">

<a href="/c101280100/?page=0" ka="page-prev" class="prev"></a>

<a href="javascript:;" class="cur" ka="page-cur">1</a>

<a href="/c101280100/?page=2" ka="page-2">2</a>

<a href="/c101280100/?page=3" ka="page-3">3</a>

<span>...</span>

<a href="/c101280100/?page=2" ka="page-next" class="next"></a>

</div>

from lxml import etree

html=etree.parse('./test.html', etree.HTMLParser())

new\_links = html.xpath('//div[@class="page"]/a[@class="next"]/@href')

print(len(new\_links))

if new\_links and len(new\_links) > 0:

print(type(new\_links[0]))

print("https://www.zhipin.com"+new\_links[0])

root@pc:/home/zcw# python3 test.py

1

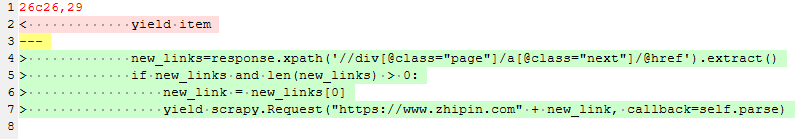
<class 'lxml.etree.\_ElementUnicodeResult'>

https://www.zhipin.com/c101280100/?page=2

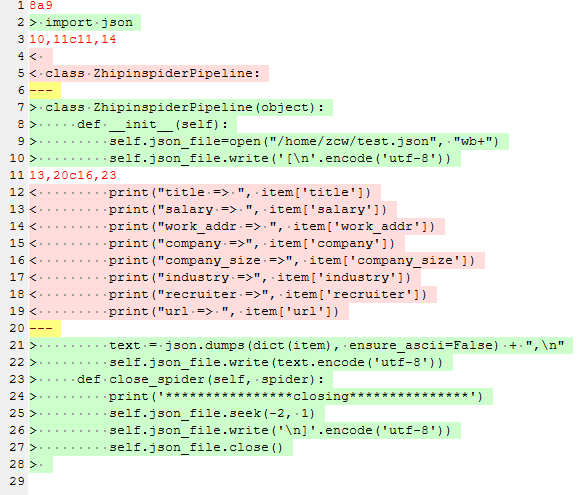
root@pc:/home/zcw#

item = ZhipinspiderItem()

...



## 例3 boss直聘: 使用json导出信息



root@pc:/home/zcw# cat /home/zcw/ZhipinSpider/ZhipinSpider/pipelines.py

from itemadapter import ItemAdapter

import json

class ZhipinspiderPipeline(object):

def \_\_init\_\_(self):

self.json\_file=open("/home/zcw/test.json", "wb+")

self.json\_file.write('[\n'.encode('utf-8'))

def process\_item(self, item, spider):

text = json.dumps(dict(item), ensure\_ascii=False) + ",\n"

self.json\_file.write(text.encode('utf-8'))

def close\_spider(self, spider): #覆写方法

print('\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*closing\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*')

self.json\_file.seek(-2, 1)

self.json\_file.write('\n]'.encode('utf-8'))

self.json\_file.close()

root@pc:/home/zcw/ZhipinSpider# scrapy crawl job\_position



## 例4 boss直聘: 写数据库



root@pc:/home/zcw# cat ZhipinSpider/ZhipinSpider/pipelines.py

from itemadapter import ItemAdapter

import mysql.connector

class ZhipinspiderPipeline(object):

def \_\_init\_\_(self):

self.conn = mysql.connector.connect(user='root', password='123456', host='localhost', port='3306', database='python', use\_unicode=True)

self.cur = self.conn.cursor()

def process\_item(self, item, spider):

self.cur.execute("insert into job\_inf values(null, %s, %s, %s, %s, %s, %s, %s, %s)",

(item['title'], item['salary'], item['work\_addr'], item['url'], item['company'], item['industry'], item['company\_size'], item['recruiter']))

self.conn.commit()

def close\_spider(self, spider):

print('\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*CLOSE MYSQL\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*')

self.cur.close()

self.conn.close()

root@pc:/home/zcw#

**测试**

apt install python3-pip -y

pip install mysql-connector-python

apt install mysql-server -y

systemctl status mysql

#要使用mysql-connector-python，mysql必须配置密码

mysql> alter user 'root'@'localhost' identified with mysql\_native\_password by '123456';

mysql> flush privileges;

mysql> create database python;

mysql> use python

mysql> create table job\_inf(id int(11) not null auto\_increment primary key, title varchar(255), salary varchar(255), work\_addr varchar(255), url varchar(255), company varchar(255), industry varchar(255), company\_size varchar(255), recruiter varchar(255));

cd /home/zcw/ZhipinSpider

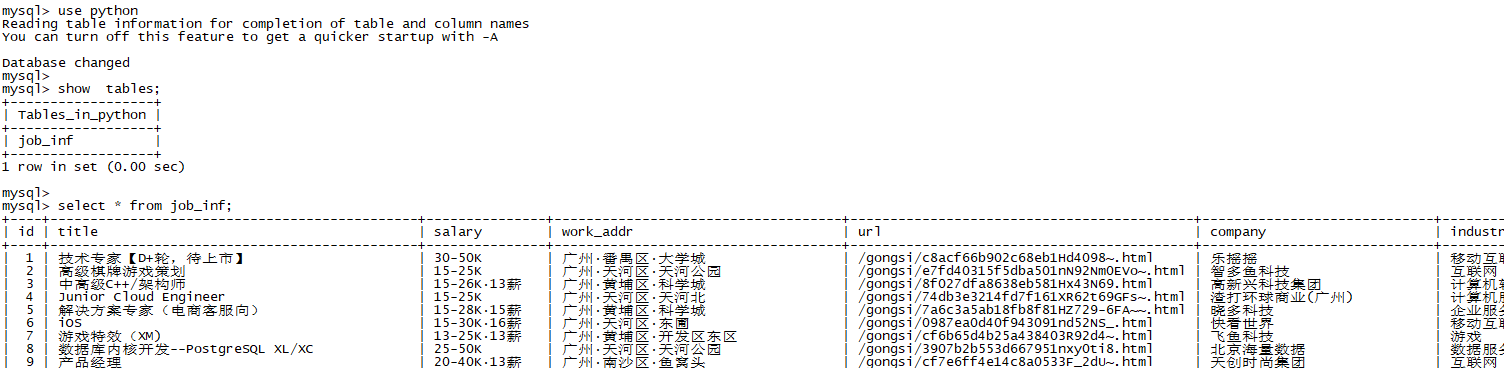
scrapy crawl job\_position

mysql -u root -p

use python

show tables;

select \* from job\_inf;



## 例5 boss直聘: 使用Pygal展示招聘信息



root@pc:/home/zcw# cat ZhipinSpider/ZhipinSpider/pipelines.py

from itemadapter import ItemAdapter

import json

import pygal, codecs

class ZhipinspiderPipeline(object):

def \_\_init\_\_(self):

self.json\_file=open("/home/zcw/test.json", "wb+")

self.json\_file.write('[\n'.encode('utf-8'))

def process\_item(self, item, spider):

print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*process\_item\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

text = json.dumps(dict(item), ensure\_ascii=False) + ",\n"

self.json\_file.write(text.encode('utf-8'))

def close\_spider(self, spider):

print('\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*closing\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*')

self.json\_file.seek(-2, 1)

self.json\_file.write('\n]'.encode('utf-8'))

self.json\_file.close()

print('\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*use pygal\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*')

with codecs.open("/home/zcw/test.json", 'r', 'utf-8', buffering=True) as f:

job\_list = json.load(f)

job\_dict = {}

for li in job\_list:

if li['industry'] in job\_dict:

job\_dict[li['industry']] += 1

else:

job\_dict[li['industry']] = 1

print(job\_dict)

pie = pygal.Pie()

#for kk in job\_dict.keys():

# if job\_dict[kk] < 5:

# other\_num += job\_dict[kk]

# else:

# pie.add(kk, job\_dict[kk])

#pie.add('other', other\_num)

for kk in job\_dict.keys():

pie.add(kk, job\_dict[kk])

pie.title = 'Statistical chart of popular recruitment in Guangzhou'

pie.legend\_at\_bottom = True

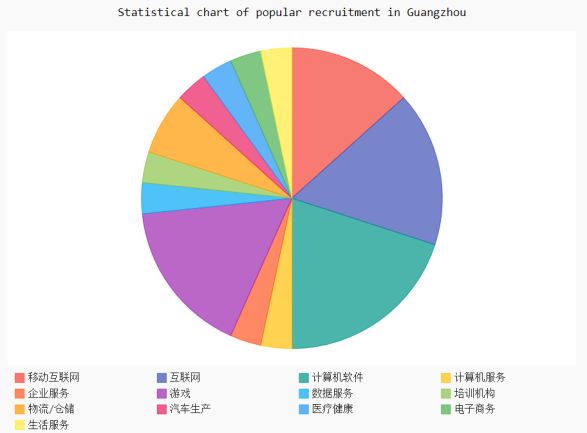
pie.render\_to\_file('/home/zcw/job\_position.svg')

root@pc:/home/zcw#

cd /home/zcw/ZhipinSpider

scrapy crawl job\_position





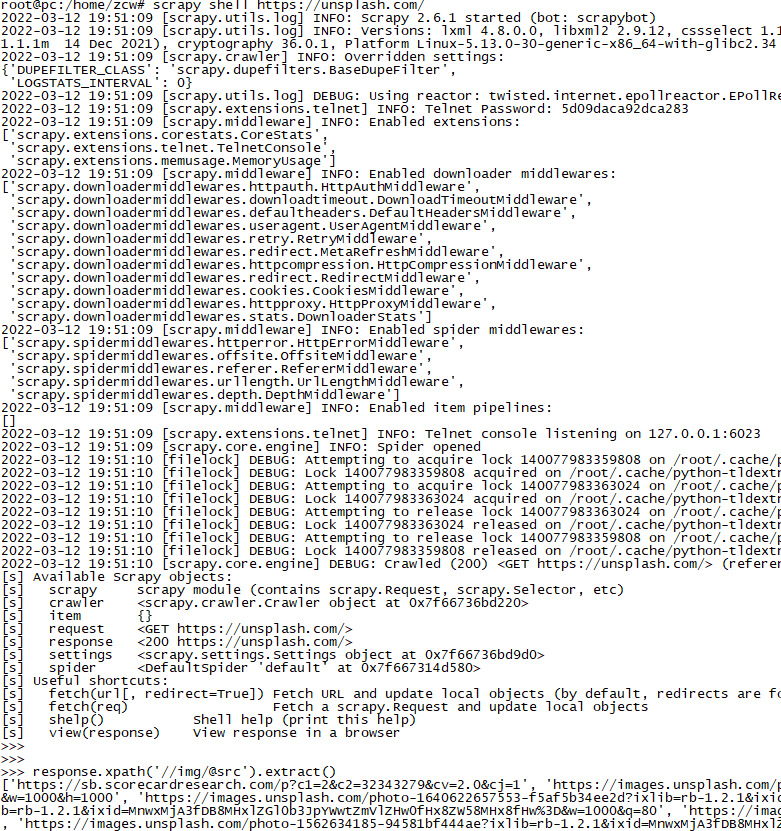
# 处理反爬虫

网站的反爬虫处理：网页内容不是静态的，而是javascript动态加载的，此时的爬虫程序也需要做相应的改进。

## 使用shell调试工具分析目标站点

root@pc:/home/zcw# scrapy shell https://unsplash.com/

>>> response.xpath('//img/@src').extract()



执行上面命令，将看到一系列图片的url，但都不是高清图片的url。

Ctrl+Shift+I 或者点击浏览器右上角 -> More tools -> Browser Console

在console上配置过滤url: <https://unsplash.com/>

在浏览器地址栏输入url: <https://unsplash.com/>

从console发现一批url，形如:

https://unsplash.com/napi/photos?page=N&per\_page=M&order\_by=latest

最后再从控制台回到scrapy shell

scrapy shell https://unsplash.com/napi/photos?page=1&per\_page=10&order\_by=latest

>>> import json

>> len(json.loads(response.txt))

>> len(json.loads(response.txt)[0][‘links’][‘download’])

一番调试分析下来，图片请求的整个过程 及 每张图片的属性 一目了然。

(1)每张图片的属性包括：id，created\_at, updated\_at, width,height等基本信息和links属性（该属性值转换后，对应python的dict），links属性包含self，html，download，download\_location等属性。

(2)图片请求整个过程

不停的拉动垂直滚动条，会不停的发出分页请求

要点：网络爬虫毕竟是针对别人的网站爬取数据的，而目标网站的结构随时可能变化，读者应该学习（浏览器控制台+scrapy shell）分析网页的方法，而非生搬硬套的照抄经典。

## 分页爬虫

根据shell调试工具分析得出的结果，创建动态网页的爬虫项目 ——分页爬虫

scrapy startproject UnsplashImageSpider

scrapy genspider unsplash\_image 'unsplash.com'

scrapy crawl unsplash\_image

root@pc:/home/zcw# scrapy startproject UnsplashImageSpider

New Scrapy project 'UnsplashImageSpider', using template directory '/usr/local/lib/python3.9/dist-packages/scrapy/templates/project', created in:

/home/zcw/UnsplashImageSpider

You can start your first spider with:

cd UnsplashImageSpider

scrapy genspider example example.com

root@pc:/home/zcw#

root@pc:/home/zcw# cd UnsplashImageSpider/

root@pc:/home/zcw/UnsplashImageSpider# scrapy genspider unsplash\_image 'unsplash.com'

Created spider 'unsplash\_image' using template 'basic' in module:

UnsplashImageSpider.spiders.unsplash\_image

root@pc:/home/zcw/UnsplashImageSpider# tree

.

├── scrapy.cfg

└── UnsplashImageSpider

├── \_\_init\_\_.py

├── items.py

├── middlewares.py

├── pipelines.py

├── \_\_pycache\_\_

│   ├── \_\_init\_\_.cpython-39.pyc

│   └── settings.cpython-39.pyc

├── settings.py

└── spiders

├── \_\_init\_\_.py

├── \_\_pycache\_\_

│   └── \_\_init\_\_.cpython-39.pyc

└── unsplash\_image.py

4 directories, 11 files

root@pc:/home/zcw/UnsplashImageSpider#

vi /home/zcw/UnsplashImageSpider/UnsplashImageSpider/items.py

import scrapy

class ImageItem(scrapy.Item):

image\_id = scrapy.Field()

download = scrapy.Field()

vi /home/zcw/UnsplashImageSpider/UnsplashImageSpider/spiders/unsplash\_image.py

import scrapy, json

from UnsplashImageSpider.items import ImageItem

class UnsplashImageSpider(scrapy.Spider):

name = 'unsplash\_image'

allowed\_domains = ['unsplash.com']

start\_urls = ['http://unsplash.com/napi/photos?page=1&per\_page=10&order\_by=latest']

def \_\_init\_\_(self):

self.page\_index = 1

def parse(self, response):

photo\_list = json.loads(response.text)

for photo in photo\_list:

item = ImageItem()

item['image\_id'] = photo['id']

item['download'] = photo['links']['download']

yield item

self.page\_index += 1

next\_link = 'http://unsplash.com/napi/photos?page=' + str(self.page\_index) + '&per\_page=10&order\_by=latest'

yield scrapy.Request(next\_link, callback=self.parse)

vi /home/zcw/UnsplashImageSpider/UnsplashImageSpider/pipelines.py

from itemadapter import ItemAdapter

from urllib.request import \*

class UnsplashimagespiderPipeline(object):

def process\_item(self, item, spider):

print('------------------' + item['image\_id'])

real\_url = item['download'] + "?force=true"

try:

pass

with urlopen(real\_url) as result:

data = result.read()

with open("images/" + item['image\_id'] + '.jpg', 'wb+') as f:

f.write(data)

except:

print('download photo error', item['image\_id'])

vi /home/zcw/UnsplashImageSpider/UnsplashImageSpider/settings.py

DEFAULT\_REQUEST\_HEADERS = {

'User-Agent': 'Mozilla/5.0 (X11; Linux x86\_64; rv:98.0) Gecko/20100101 Firefox/98.0',

'Accept': 'text/html,application/xhtml+xml,application/xml;q=0.9,\*/\*;q=0.8',

}

ITEM\_PIPELINES = {

'UnsplashImageSpider.pipelines.UnsplashimagespiderPipeline': 300,

}

scrapy crawl unsplash\_image

## 应对反爬虫的常见方法

网站的反爬虫策略：

(1)网站会检测客户端的ip地址，若发现某ip客户端瞬时频繁请求数据，则该ip客户端会被判定为爬虫。

(2)有些网站会跟踪Cookie，假设某ip客户端开了cookie，且单位时间内的请求过于频繁，则website会根据cookie判定ip客户端为爬虫。

(3) 很多web站点会提供一个robots.txt文件，在该文件中制定了一系列爬虫规则。例如，weibo.com网站下的robots.txt文件内容如下：

... User-Agent: Baiduspider Disallow: User-agent:360Spider ...

该规则只接收baidu的爬虫，不接受其他爬虫程序。

(4) 有些网站为了防止机器程序访问，会做一些特殊设计。

会记录同一个ip客户端的访问次数，只要达到阈值，会强制弹出图形验证码。

爬虫策略：

1. 修改settings.py

#开启访问频率限制

AUTOTHROTTLE\_ENABLED = True

#设置访问开始的延迟

AUTOTHROTTLE\_START\_DELAY = 5

#设置访问最大延迟

AUTOTHROTTLE\_MAX\_DELAY = 60

#设置scrapy并行发给每台远程服务器的请求数量

AUTOTHROTTLE\_TARGET\_CONCURRENCY = 1.0

#设置下载之后的自动延迟

AUTOTHROTTLE\_DEBUG = False

或者

修改下载中间件，不断改变爬虫客户端的IP

vi middlewares.py

class RandomProxyMiddleware(object):

def process\_request(self, requtest, spider):

#get\_random\_proxy()函数随机返回代理服务器的ip地址和端口

request.meta["proxy"] = get\_random\_proxy()

vi settings.py

DOWNLOADER\_MIDDLEWARES = {

'UnsplashImageSpider.middlewares.UnsplashimagespiderDownloaderMiddleware': 543,

}

=>

DOWNLOADER\_MIDDLEWARES = {

'UnsplashImageSpider.middlewares. RandomProxyMiddleware': 543,

}

1. 如果Scrapy不需要登录，那么干脆就让Scrapy禁用Cookie好了

COOKIES\_ENABLED = False

1. 让爬虫取消robots限制，主动强制违反爬虫规则文件的限制，强行爬取站点信息

ROBOTSTXT\_OBEY = True => ROBOTSTXT\_OBEY = FALSE

(4)

方法1：使用PIL，Libsvm等库开发程序来识别图形验证码，虽然灵活，但需自行开发代码

方法2：是固体不过第三方识别。有不少图形验证码的在线识别网站。识别率高的收费高，不收费的识别率低。

# Selenium库

整合Selenium模拟浏览器的行为：

方法1：

有些网站需要username和password，所以普通的爬虫不太行，Selenium库可以帮忙请求。

编码复杂，参见Selenium爬虫项目

登录成功后，Selenium会记录cookie，方便后续访问。

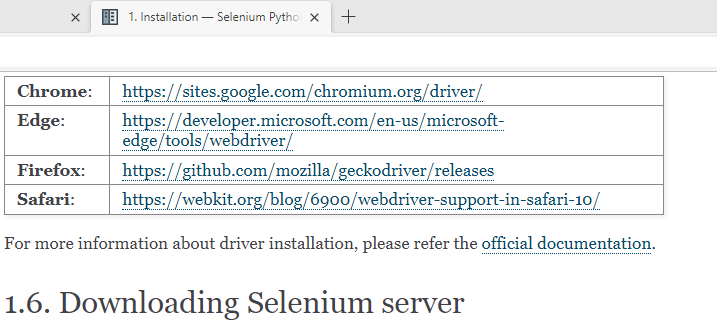
方法2： Selenium可以启动真正的浏览器来访问网站。

效率太低，参见简单selenium程序

为了python3使用Selenium，需要以下3步：

(1)pip install selenium

(2)下载浏览驱动(登录https://selenium-python.readthedocs.io/installation.html#drivers)



以geckodriver.exe驱动为例，下载后直接放到项目目录下

(3)安装目标浏览器，比如本项目需要启动firefox，就需要在目标机上安装firefox浏览器。

## Selenium简单程序

root@pc:/home/zcw# ls

geckodriver.exe selenium\_test.py

root@pc:/home/zcw# cat selenium\_test.py

from selenium import webdriver

import time

browser = webdriver.Firefox(executable\_path="./geckodriver.exe")

time.sleep(3) #等待浏览器启动

browser.get("http://www.crazyit.org/")

time.sleep(5) #暂停5秒

## Selenium爬虫项目 - username，password，cookie

scrapy startproject Weibopider

cd Weibopider/

scrapy genspider weibo\_post 'weibo.com'

cd Weibopider/

scrapy crawl weibo\_post

root@pc:/home/zcw# tree

.

└── Weibopider

├── geckodriver.exe #浏览器驱动

├── scrapy.cfg

└── Weibopider

├── \_\_init\_\_.py

├── items.py

├── middlewares.py

├── pipelines.py

├── \_\_pycache\_\_

│   ├── \_\_init\_\_.cpython-39.pyc

│   └── settings.cpython-39.pyc

├── settings.py

└── spiders

├── \_\_init\_\_.py

├── \_\_pycache\_\_

│   └── \_\_init\_\_.cpython-39.pyc

└── weibo\_post.py

5 directories, 12 files

root@pc:/home/zcw#

# 本章小结

Scrapy是python领域专业的爬虫开发框架，scrapy框架已经完成爬虫的大部分通用工作，因此使用Scrapy开发爬虫项目简单方便。

使用Scrapy开发爬虫的核心步骤：

(1) 定义Item类

(2) 开发Spider，从网页上提取数据，也提取翻页连接

(3) 开发Pipeline，负责将提取的数据写入文件或数据库。

要区别对待静态网页和动态网页，两种网页的爬取方式略有差异。