# **[基于 Python 的 Scrapy 爬虫入门：代码详解](http://www.36dsj.com/archives/103881)**

## ****一、内容分析****

接下来创建一个爬虫项目，以 图虫网 为例抓取里面的图片。在顶部菜单“发现” “标签”里面是对各种图片的分类，点击一个标签，比如“美女”，网页的链接为：[https://tuchong.com/tags/美女/](https://tuchong.com/tags/%22%20%5Ct%20%22http%3A//www.36dsj.com/archives/_blank)，我们以此作为爬虫入口，分析一下该页面：

打开页面后出现一个个的图集，点击图集可全屏浏览图片，向下滚动页面会出现更多的图集，没有页码翻页的设置。Chrome右键“检查元素”打开开发者工具，检查页面源码，内容部分如下：

<div class="content">

 <div class="widget-gallery">

 <ul class="pagelist-wrapper">

 <li class="gallery-item...

可以判断每一个li.gallery-item是一个图集的入口，存放在ul.pagelist-wrapper下，div.widget-gallery是一个容器，如果使用 xpath 选取应该是：//div[@class=”widget-gallery”]/ul/li，按照一般页面的逻辑，在li.gallery-item下面找到对应的链接地址，再往下深入一层页面抓取图片。

但是如果用类似 Postman 的HTTP调试工具请求该页面，得到的内容是：

<div class="content">

 <div class="widget-gallery"></div>

</div>

也就是并没有实际的图集内容，因此可以断定页面使用了Ajax请求，只有在浏览器载入页面时才会请求图集内容并加入div.widget-gallery中，通过开发者工具查看XHR请求地址为：

https://tuchong.com/rest/tags/美女/posts?page=1&count=20&order=weekly&before\_timestamp=

参数很简单，page是页码，count是每页图集数量，order是排序，before\_timestamp为空，图虫因为是推送内容式的网站，因此before\_timestamp应该是一个时间值，不同的时间会显示不同的内容，这里我们把它丢弃，不考虑时间直接从最新的页面向前抓取。

请求结果为JSON格式内容，降低了抓取难度，结果如下：

{

 "postList": [

 {

 "post\_id": "15624611",

 "type": "multi-photo",

 "url": "https://weishexi.tuchong.com/15624611/",

 "site\_id": "443122",

 "author\_id": "443122",

 "published\_at": "2017-10-28 18:01:03",

 "excerpt": "10月18日",

 "favorites": 4052,

 "comments": 353,

 "rewardable": true,

 "parent\_comments": "165",

 "rewards": "2",

 "views": 52709,

 "title": "微风不燥 秋意正好",

 "image\_count": 15,

 "images": [

 {

 "img\_id": 11585752,

 "user\_id": 443122,

 "title": "",

 "excerpt": "",

 "width": 5016,

 "height": 3840

 },

 {

 "img\_id": 11585737,

 "user\_id": 443122,

 "title": "",

 "excerpt": "",

 "width": 3840,

 "height": 5760

 },

 ...

 ],

 "title\_image": null,

 "tags": [

 {

 "tag\_id": 131,

 "type": "subject",

 "tag\_name": "人像",

 "event\_type": "",

 "vote": ""

 },

 {

 "tag\_id": 564,

 "type": "subject",

 "tag\_name": "美女",

 "event\_type": "",

 "vote": ""

 }

 ],

 "favorite\_list\_prefix": [],

 "reward\_list\_prefix": [],

 "comment\_list\_prefix": [],

 "cover\_image\_src": "https://photo.tuchong.com/443122/g/11585752.webp",

 "is\_favorite": false

 }

 ],

 "siteList": {...},

 "following": false,

 "coverUrl": "https://photo.tuchong.com/443122/ft640/11585752.webp",

 "tag\_name": "美女",

 "tag\_id": "564",

 "url": "https://tuchong.com/tags/%E7%BE%8E%E5%A5%B3/",

 "more": true,

 "result": "SUCCESS"

}

根据属性名称很容易知道对应的内容含义，这里我们只需关心 postlist 这个属性，它对应的一个数组元素便是一个图集，图集元素中有几项属性我们需要用到：

* url：单个图集浏览的页面地址
* post\_id：图集编号，在网站中应该是唯一的，可以用来判断是否已经抓取过该内容
* site\_id：作者站点编号 ，构建图片来源链接要用到
* title：标题
* excerpt：摘要文字
* type：图集类型，目前发现两种，一种multi-photo是纯照片，一种text是文字与图片混合的文章式页面，两种内容结构不同，需要不同的抓取方式，本例中只抓取纯照片类型，text类型直接丢弃
* tags：图集标签，有多个
* image\_count：图片数量
* images：图片列表，它是一个对象数组，每个对象中包含一个img\_id属性需要用到

根据图片浏览页面分析，基本上图片的地址都是这种格式： https://photo.tuchong.com/{site\_id}/f/{img\_id}.jpg ，很容易通过上面的信息合成。

## ****二、创建项目****

1. 进入cmder命令行工具，输入workon scrapy 进入之前建立的虚拟环境，此时命令行提示符前会出现(Scrapy) 标识，标识处于该虚拟环境中，相关的路径都会添加到PATH环境变量中便于开发及使用。
2. 输入 scrapy startproject tuchong 创建项目 tuchong
3. 进入项目主目录，输入 scrapy genspider photo tuchong.com 创建一个爬虫名称叫 photo (不能与项目同名)，爬取 tuchong.com 域名（这个需要修改，此处先输个大概地址），的一个项目内可以包含多个爬虫

经过以上步骤，项目自动建立了一些文件及设置，目录结构如下：

(PROJECT)

│ scrapy.cfg

│

└─tuchong

 │ items.py

 │ middlewares.py

 │ pipelines.py

 │ settings.py

 │ \_\_init\_\_.py

 │

 ├─spiders

 │ │ photo.py

 │ │ \_\_init\_\_.py

 │ │

 │ └─\_\_pycache\_\_

 │ \_\_init\_\_.cpython-36.pyc

 │

 └─\_\_pycache\_\_

 settings.cpython-36.pyc

 \_\_init\_\_.cpython-36.pyc

* scrapy.cfg：基础设置
* items.py：抓取条目的结构定义
* middlewares.py：中间件定义，此例中无需改动
* pipelines.py：管道定义，用于抓取数据后的处理
* settings.py：全局设置
* spiders\photo.py：爬虫主体，定义如何抓取需要的数据

## ****三、主要代码****

items.py 中创建一个TuchongItem类并定义需要的属性，属性继承自 scrapy.Field 值可以是字符、数字或者列表或字典等等：

import scrapy

class TuchongItem(scrapy.Item):

 post\_id = scrapy.Field()

 site\_id = scrapy.Field()

 title = scrapy.Field()

 type = scrapy.Field()

 url = scrapy.Field()

 image\_count = scrapy.Field()

 images = scrapy.Field()

 tags = scrapy.Field()

 excerpt = scrapy.Field()

 ...

这些属性的值将在爬虫主体中赋予。

spiders\photo.py 这个文件是通过命令 scrapy genspider photo tuchong.com 自动创建的，里面的初始内容如下：

import scrapy

class PhotoSpider(scrapy.Spider):

 name = 'photo'

 allowed\_domains = ['tuchong.com']

 start\_urls = ['http://tuchong.com/']

 def parse(self, response):

 pass

爬虫名 name，允许的域名 allowed\_domains（如果链接不属于此域名将丢弃，允许多个） ，起始地址 start\_urls 将从这里定义的地址抓取（允许多个）
函数 parse 是处理请求内容的默认回调函数，参数 response 为请求内容，页面内容文本保存在 response.body 中，我们需要对默认代码稍加修改，让其满足多页面循环发送请求，这需要重载 start\_requests 函数，通过循环语句构建多页的链接请求，修改后代码如下：

import scrapy, json

from ..items import TuchongItem

class PhotoSpider(scrapy.Spider):

 name = 'photo'

 # allowed\_domains = ['tuchong.com']

 # start\_urls = ['http://tuchong.com/']

 def start\_requests(self):

 url = 'https://tuchong.com/rest/tags/%s/posts?page=%d&count=20&order=weekly';

 # 抓取10个页面，每页20个图集

 # 指定 parse 作为回调函数并返回 Requests 请求对象

 for page in range(1, 11):

 yield scrapy.Request(url=url % ('美女', page), callback=self.parse)

 # 回调函数，处理抓取内容填充 TuchongItem 属性

 def parse(self, response):

 body = json.loads(response.body\_as\_unicode())

 items = []

 for post in body['postList']:

 item = TuchongItem()

 item['type'] = post['type']

 item['post\_id'] = post['post\_id']

 item['site\_id'] = post['site\_id']

 item['title'] = post['title']

 item['url'] = post['url']

 item['excerpt'] = post['excerpt']

 item['image\_count'] = int(post['image\_count'])

 item['images'] = {}

 # 将 images 处理成 {img\_id: img\_url} 对象数组

 for img in post.get('images', ''):

 img\_id = img['img\_id']

 url = 'https://photo.tuchong.com/%s/f/%s.jpg' % (item['site\_id'], img\_id)

 item['images'][img\_id] = url

 item['tags'] = []

 # 将 tags 处理成 tag\_name 数组

 for tag in post.get('tags', ''):

 item['tags'].append(tag['tag\_name'])

 items.append(item)

 return items

经过这些步骤，抓取的数据将被保存在 TuchongItem 类中，作为结构化的数据便于处理及保存。

前面说过，并不是所有抓取的条目都需要，例如本例中我们只需要 type=”multi\_photo 类型的图集，并且图片太少的也不需要，这些抓取条目的筛选操作以及如何保存需要在pipelines.py中处理，该文件中默认已创建类 TuchongPipeline 并重载了 process\_item函数，通过修改该函数只返回那些符合条件的 item，代码如下：

...

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

 # 不符合条件触发 scrapy.exceptions.DropItem 异常，符合条件的输出地址

 if int(item['image\_count']) < 3:

 raise DropItem("美女太少: " + item['url'])

 elif item['type'] != 'multi-photo':

 raise DropItem("格式不对: " + + item['url'])

 else:

 print(item['url'])

 return item

...

当然如果不用管道直接在 parse 中处理也是一样的，只不过这样结构更清晰一些，而且还有功能更多的FilePipelines和ImagePipelines可供使用，process\_item将在每一个条目抓取后触发，同时还有 open\_spider 及 close\_spider 函数可以重载，用于处理爬虫打开及关闭时的动作。

注意：管道需要在项目中注册才能使用，在 settings.py 中添加：

ITEM\_PIPELINES = {

 'tuchong.pipelines.TuchongPipeline': 300, # 管道名称: 运行优先级(数字小优先)

}

另外，大多数网站都有反爬虫的 Robots.txt 排除协议，设置 ROBOTSTXT\_OBEY = True 可以忽略这些协议，是的，这好像只是个君子协定。如果网站设置了浏览器User Agent或者IP地址检测来反爬虫，那就需要更高级的Scrapy功能，本文不做讲解。

## ****四、运行****

返回 cmder 命令行进入项目目录，输入命令：

scrapy crawl photo

终端会输出所有的爬行结果及调试信息，并在最后列出爬虫运行的统计信息，例如：

[scrapy.statscollectors] INFO: Dumping Scrapy stats:

{'downloader/request\_bytes': 491,

 'downloader/request\_count': 2,

 'downloader/request\_method\_count/GET': 2,

 'downloader/response\_bytes': 10224,

 'downloader/response\_count': 2,

 'downloader/response\_status\_count/200': 2,

 'finish\_reason': 'finished',

 'finish\_time': datetime.datetime(2017, 11, 27, 7, 20, 24, 414201),

 'item\_dropped\_count': 5,

 'item\_dropped\_reasons\_count/DropItem': 5,

 'item\_scraped\_count': 15,

 'log\_count/DEBUG': 18,

 'log\_count/INFO': 8,

 'log\_count/WARNING': 5,

 'response\_received\_count': 2,

 'scheduler/dequeued': 1,

 'scheduler/dequeued/memory': 1,

 'scheduler/enqueued': 1,

 'scheduler/enqueued/memory': 1,

 'start\_time': datetime.datetime(2017, 11, 27, 7, 20, 23, 867300)}

主要关注ERROR及WARNING两项，这里的 Warning 其实是不符合条件而触发的 DropItem 异常。

## ****五、保存结果****

大多数情况下都需要对抓取的结果进行保存，默认情况下 item.py 中定义的属性可以保存到文件中，只需要命令行加参数 -o {filename} 即可：

scrapy crawl photo -o output.json # 输出为JSON文件

scrapy crawl photo -o output.csv # 输出为CSV文件

注意：输出至文件中的项目是未经过 TuchongPipeline 筛选的项目，只要在 parse 函数中返回的 Item 都会输出，因此也可以在 parse 中过滤只返回需要的项目

如果需要保存至数据库，则需要添加额外代码处理，比如可以在 pipelines.py 中 process\_item 后添加:

...

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

 ...

 else:

 print(item['url'])

 self.myblog.add\_post(item) # myblog 是一个数据库类，用于处理数据库操作

 return item

...

为了在插入数据库操作中排除重复的内容，可以使用 item[‘post\_id’] 进行判断，如果存在则跳过。