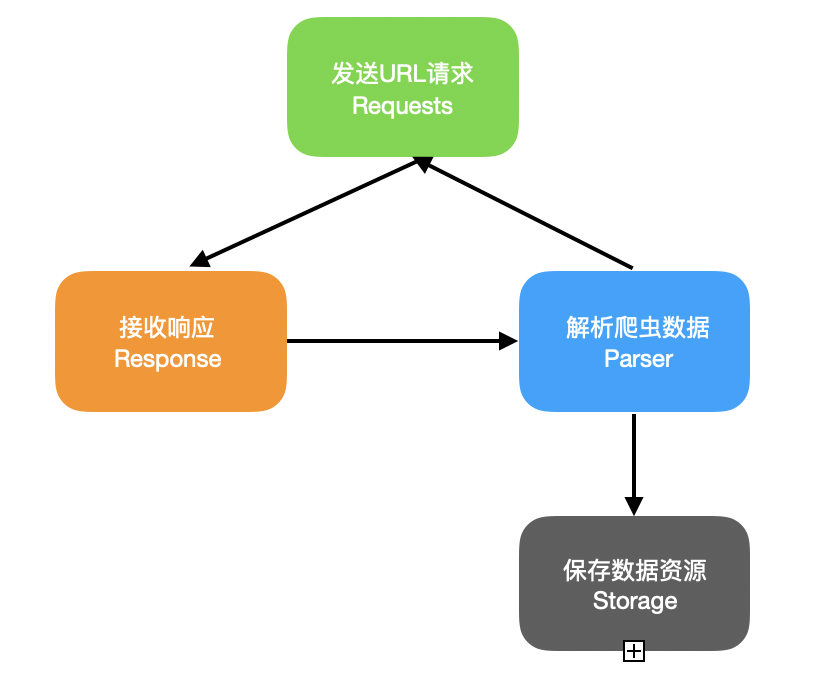
### Python网络爬虫分享会

当前是一个大数据时代，通过大数据分析，我们可以做到预测用户行为、预测市场发展方向等等。阿里巴巴创始人马云也曾在演讲中提到：“人类正从IT时代走向DT时代。”其中，DT即数据处理技术。可以说掌握了数据，就掌握了先机。那么，如何从互联网上获取更多的数据，就不得不说到本文要分享的主角——网络爬虫，其中基于Python的网络爬虫是当前业界比较火的技术。本次主要从网络爬虫的基础以及Python爬虫框架Scrapy两方面进行了分享。

**一、何谓网络爬虫**

它是一种按照一定的规则，自动地抓取万维网信息的程序或者脚本。

从当前主流的爬虫框架中大致可知，一个网络爬虫主要由控制器，解析器，资源库三部分组成。控制器主要负责管理爬取的url，为其分配并启动线程调用爬虫程序。解析器是爬虫的核心，主要负责下载网页，进行页面的处理，将一些JS脚本标签、CSS代码内容、空格字符、HTML标签等内容处理掉。资源库则是将爬取到的数据进行存储，如保存至数据库。总结下来，一个网络爬虫的基本过程大致如下图：



以下是一个基于Python开发的简易爬虫例子，实现了爬取特定链接，解析特定的数据内容，并保存至本地文件：

from urllib.request import urlopen  
from bs4 import BeautifulSoup  
  
html = urlopen('http://www.pythonscraping.com/pages/page1.html')  
bs = BeautifulSoup(html.read(), 'html.parser')  
print(bs.h1)  
  
with open('data.txt','w') as f:  
 f.write(bs.h1.get\_text())

网络爬虫第一步是爬取，主要有两种形式，一种是爬取服务端渲染的数据，一种客户端渲染的方式。其中，服务端渲染指的是数据在服务器端就渲染至HTML页面，然后将页面结果返回。利用Python一些基本的HTTP请求库即可轻松爬取（如requests、urllib、urllib3等）服务端渲染的页面。而客户端渲染则是指页面的数据主要是通过Ajax接口等形式获取至客户端，然后在客户端通过JavaScript渲染成完整页面。针对此类客户端渲染的爬虫，一者可以通过Chrome/Fiddler等工具查看 Ajax 接口以及其具体的请求方式、参数等内容，然后用HTTP请求库模拟请求获取数据；再者可以模拟执行 JavaScript，然后再爬取所需数据，这里可以利用Python等类库Selenium轻松实现。

网络爬虫的第二步是解析，主要解析的数据类型是HTML、JSON、XML等。这方面Python也有丰富等类库支持，包括BeautifulSoup4、XPath、CSS Selector等等。同时，还需要学习掌握一定的正则表达式基础。

网络爬虫的第三部是数据存储，即根据需求选择合适的存储媒介保存爬取到的数据，媒介形式包括但不局限于文件（JSON/CSV/TXT/图片/音频/视频等）、数据库（MySQL/MongoDB/HBase等）。Python也有丰富等类库支持相应等操作，如csv、json、pandas、pymysql、pymongo。

总的来说，Python作为轻量级的脚本语言，其丰富的类库支持，使得其在网络爬虫方面具有得天独厚的优势。最后，针对这块内容，分享了一份较为完整的python代码供学习参考，该爬虫实现了爬取豆瓣网的书籍标签，并保存至csv文件：

#-\*- coding: UTF-8 -\*-  
import random  
import requests  
from bs4 import BeautifulSoup  
from openpyxl import Workbook  
  
userAgents=['Mozilla/5.0 (Macintosh; Intel Mac OS X 10\_14\_6) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/71.0.3578.98 Safari/537.36',  
 'Mozilla/5.0 (Macintosh; Intel Mac OS X 10\_12\_0) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/68.0.3440.106 Safari/537.36']  
  
# 模拟请求头  
def random\_create\_headers():  
 user\_agent = userAgents[random.randint(1,5) % len(userAgents)]  
 headers = {'User-Agent': user\_agent,  
 'Accept': 'text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,\*/\*;q=0.8'}  
 return headers  
  
# 豆瓣图书标签抓取  
def book\_tag\_spider():  
 tag\_lists = []  
 url = 'https://book.douban.com/tag/?view=type'  
 session = requests.Session()  
 req = session.get(url, headers=random\_create\_headers())  
 bs = BeautifulSoup(req.text, 'html.parser')  
 tables = bs.find\_all('table', {'class': 'tagCol'})  
 # 根据页面内容遍历豆瓣所有的书籍标签  
 for table in tables:  
 for book\_tag in table.find\_all('a'):  
 tag\_lists.append(book\_tag.get\_text())  
 return tag\_lists  
  
# 将爬取到到图书信息保存到csv文件，文件命名为'图书标签名.csv'  
def print\_book\_lists\_excel(book\_tags):  
 wb = Workbook()  
 ws = []  
 count = 1  
 ws.append(wb.create\_sheet(title='图书标签'))  
 ws[0].append(['序号', '图书标签'])  
 for i in range(len(book\_tags)):  
 ws[0].append([count, book\_tags[i]])  
 count += 1  
 wb.save('图书标签.xlsx')  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 print\_book\_lists\_excel(book\_tag\_spider())

1. **Python爬虫框架Scrapy**

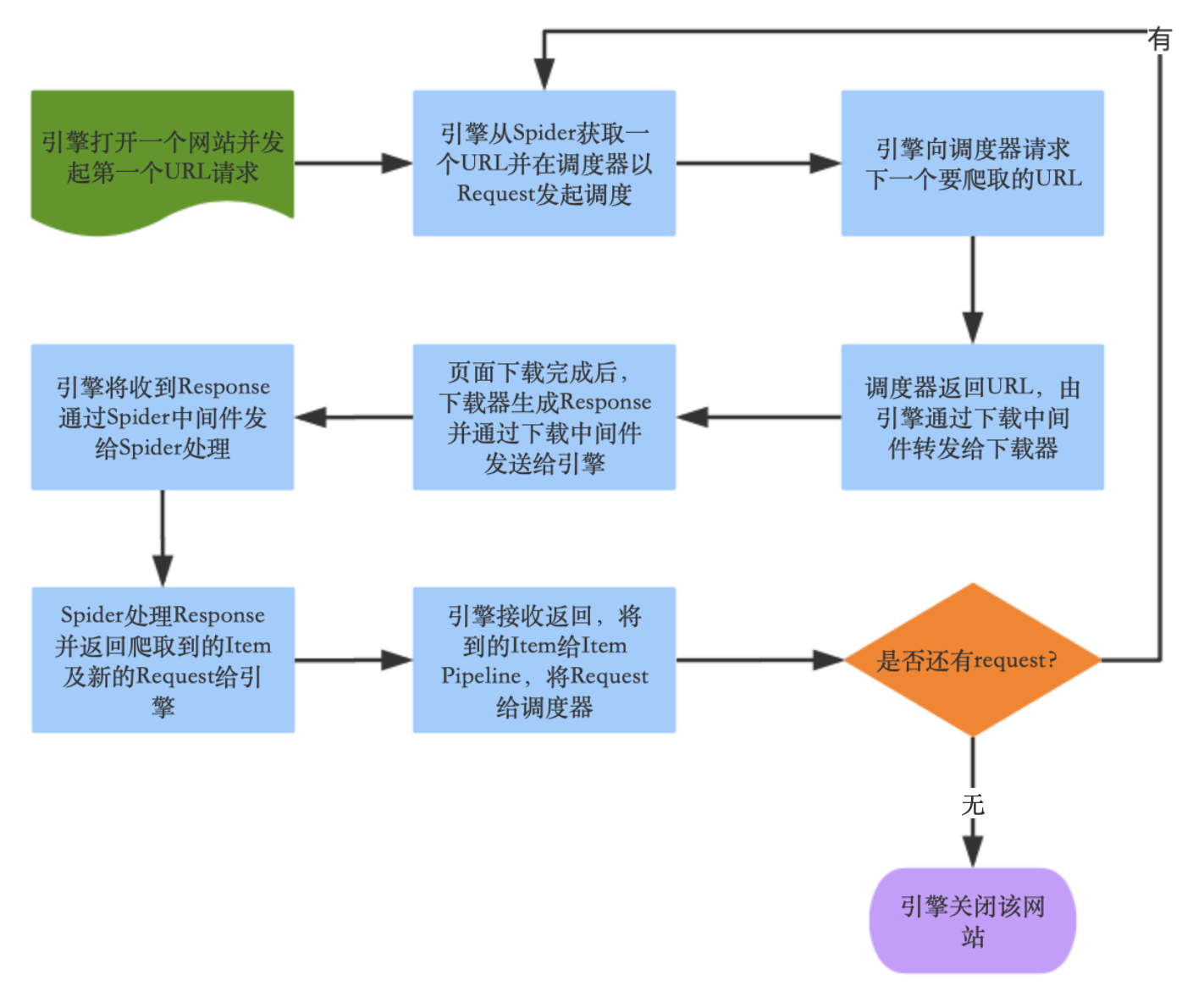
作为网络爬虫业界最为热门的编程语言之一，Python少不了爬虫框架的诞生，其中最热门的当属Scrapy框架。在介绍框架之前，先让我们看下官网提供的架构图，包括了组件及在系统中发生的数据流的概览(绿色箭头所示)：



从Scrapy的架构图看，其包含了各个功能不同的组件，根据官方文档描述，各组件主要功能如下：

1. Scrapy Engine：引擎负责控制数据流在系统中所有组件中流动，并在相应动作发生时触发事件。
2. Scheduler：调度器从引擎接受request并将他们入队，以便之后引擎请求他们时提供给引擎。
3. Downloader：下载器负责获取页面数据并提供给引擎，而后提供给spider。
4. Spiders：Scrapy用户编写用于分析response并提取item(即获取到的item)或额外跟进的URL的类。每个spider负责处理一个特定(或一些)网站。
5. Item Pipeline：Item Pipeline负责处理被spider提取出来的item。典型的处理有清理、 验证及持久化(例如存取到数据库中)。
6. Downloader middlewares：是在引擎及下载器之间的特定钩子(specific hook)，处理Downloader传递给引擎的response。 其提供了一个简便的机制，通过插入自定义代码来扩展Scrapy功能。
7. Spider middlewares：是在引擎及Spider之间的特定钩子(specific hook)，处理spider的输入(response)和输出(items及requests)。 其提供了一个简便的机制，通过插入自定义代码来扩展Scrapy功能。

Scrapy中的数据流由执行引擎控制，其处理过程大致如下：



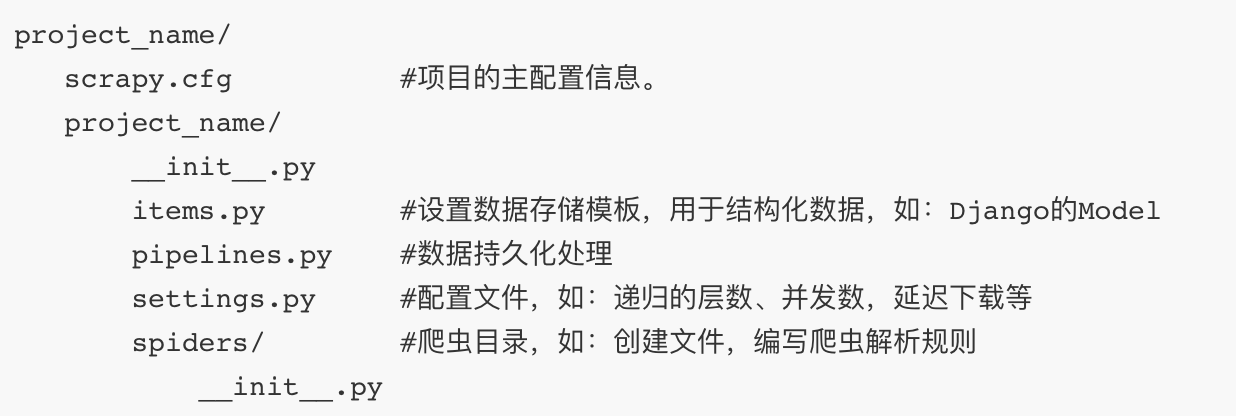
Scrapy的安装非常简便，支持以下两种方式安装：

1. Anaconda安装：conda install -c conda-forge scrapy
2. pip安装：pip3 install scrapy

安装完成后便可以进入项目实战，只需执行以下命令即可快速创建一个Scrapy项目，本次分享也皆有：

1、scrapy startproject simple\_scrapy  
2、cd simple\_scrapy  
3、scrapy genspider douban\_book\_spider https://book.douban.com/tag/%E7%BC%96%E7%A8%8B?start=0&type=T

创建项目后，我们可以发现Scrapy的项目结构如下：



创建项目后，主要需进行以下开发工作：

1. 编写items.py，用于存储字段的定义。即爬取的内容存与item类中：

import scrapy  
  
class SimpleScrapyItem(scrapy.Item):  
 sn\_id = scrapy.Field()  
 title = scrapy.Field()  
 author\_info = scrapy.Field()  
 pub\_info = scrapy.Field()  
 rating = scrapy.Field()  
 people\_num = scrapy.Field()

1. 编写douban\_book\_spider.py，实现爬虫的主体功能：

# -\*- coding: utf-8 -\*-  
import urllib  
import scrapy  
from simple\_scrapy.items import SimpleScrapyItem  
  
class DoubanBookSpiderSpider(scrapy.Spider):  
 name = 'douban\_book\_spider'  
 # 允许爬取的域名（如果遇到非该域名的url则爬取不到数据）  
 allowed\_domains = ['douban.com']  
 # 起始爬取的url  
 start\_urls = ['https://book.douban.com/tag/%E7%BC%96%E7%A8%8B?start=0&type=T']  
  
 # 爬取多页  
 pageNum = 0 # 起始页码  
 url = 'https://book.douban.com/tag/%s?start=%s&type=T'  
 count = 0  
  
 # 访问起始URL并获取结果后的回调函数，该函数的response参数就是向起始的url发送请求后，获取的响应对象.该函数返回值必须为可迭代对象或者NUll  
 def parse(self, response):  
 subject\_lists = response.xpath('//\*[@id="subject\_list"]/ul')  
 book\_items=subject\_lists.xpath('.//li[@class="subject-item"]')  
  
 for book\_item in book\_items:  
 self.count += 1  
 book\_info = book\_item.xpath('.//div[@class="info"]')  
 # 书名  
 title = book\_info.xpath('.//a[1]/@title').extract\_first()  
 book\_detail\_info = book\_info.xpath('.//div[@class="pub"]/text()').extract\_first().strip().split('/')  
  
 # 作者/译者  
 try:  
 author\_info = ''.join(book\_detail\_info[0:-3])  
 except:  
 author\_info = '暂无'  
 # 出版信息  
 try:  
 pub\_info = ''.join(book\_detail\_info[-3:])  
 except:  
 pub\_info = '暂无'  
 # 评分  
 try:  
 rating = book\_info.xpath('.//span[@class="rating\_nums"]/text()').extract\_first().strip()  
 except:  
 rating = '0.0'  
 # 评价人数  
 try:  
 people\_num = book\_info.xpath('.//span[@class="pl"]/text()').extract\_first().strip().strip('()人评价')  
 except:  
 people\_num = '0'  
  
 item = SimpleScrapyItem()  
 item['sn\_id'] = str(self.count)  
 item['title'] = title  
 item['author\_info'] = author\_info  
 item['pub\_info'] = pub\_info  
 item['rating'] = rating  
 item['people\_num'] = people\_num  
  
 yield item  
  
 if self.pageNum <= 10:  
 self.pageNum += 1  
 next\_url = format(self.url % (urllib.parse.quote('编程'), self.pageNum\*20))  
  
 # 递归爬取数据：callback参数的值为回调函数（将url请求后，得到的相应数据继续进行parse解析），递归调用parse函数  
 yield scrapy.Request(url=next\_url, callback=self.parse,dont\_filter=False)

1. 编写pipelines.py，用于存储爬虫数据到文件中：

class SimpleScrapyPipeline(object):  
  
 #构造方法  
 def \_\_init\_\_(self):  
 self.fp = None #定义一个文件描述符属性  
  
 # 下列都是在重写父类的方法：开始爬虫时，执行一次  
 def open\_spider(self,spider):  
 print('爬虫开始')  
 self.fp = open('./book\_list.txt', 'w')  
  
 # 因为该方法会被执行调用多次，所以文件的开启和关闭操作写在了另外两个只会各自执行一次的方法中。  
 def process\_item(self, item, spider):  
 #将爬虫程序提交的item进行持久化存储  
 self.fp.write(item['sn\_id'] + ':' + item['title'] + ':' + item['author\_info'] + ':' + item['pub\_info'] + ':' + item['rating'] + ':' + item['people\_num'] + '\n')  
 return item  
  
 #结束爬虫时，执行一次  
 def close\_spider(self,spider):  
 self.fp.close()  
 print('爬虫结束')

本次分享主要介绍了何谓网络爬虫，以及Python在网络爬虫的应用，同时介绍了一个主流的Python爬虫框架Scrapy。分享由浅及深，逐步给技术爱好者们揭开了网络爬虫的面纱，并进行相关的代码实践。未来是数据时代，学好网络爬虫，能够在大数据时代占据先机，从数据中挖掘出无限的发展潜能。