**Scrapy框架的使用**

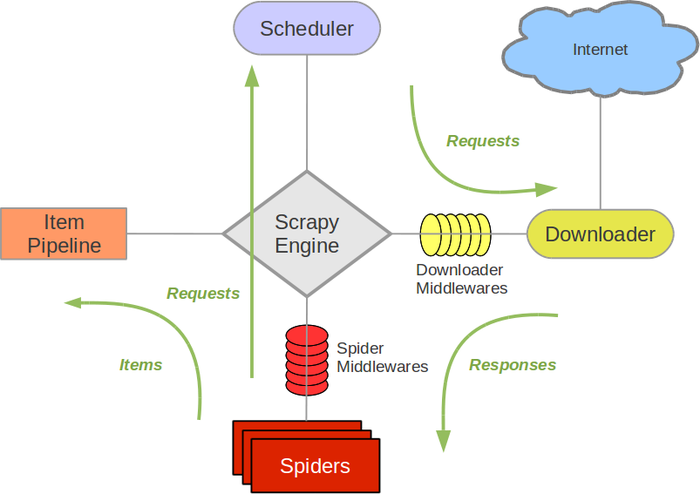
**一、 Scrapy框架的介绍**

Scrapy是一个基于 Twisted 异步处理的框架，是一个纯python的网络爬虫框架，是一个为了爬取网站数据，提取结构性数据而编写的应用框架。 其可以应用在数据挖掘，信息处理或存储历史数据等一系列的程序中。其最初是为了页面抓取 (更确切来说, 网络抓取 )所设计的， 也可以应用在获取API所返回的数据或者通用的网络爬虫。

Scrapy用途广泛，可以用于数据挖掘、监测和自动化测试。

**1.1 框架介绍**

Scrapy基于事件驱动网络框架 Twisted 编写。因此，Scrapy基于并发性考虑由非阻塞(即异步)的实现。我们先看看Scrapy的框架的架构图：



主要有以下组件：

* **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。 详细内容查看下面的数据流(Data Flow)部分。
* **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。
* **Downloader：**下载器负责获取页面数据并提供给引擎，而后提供给spider。
* **Spiders：**Spider是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。
* **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)。
* **Downloader middlewares：**下载器中间件是在引擎及下载器之间的特定钩子，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。

**1.2 数据流**

Scrapy中的数据流由执行引擎控制，其过程如下:

1. 引擎打开一个网站(open a domain)，找到处理该网站的Spider并向该spider请求第一个要爬取的URL(s)。
2. 引擎从Spider中获取到第一个要爬取的URL并在调度器(Scheduler)以Request调度。
3. 引擎向调度器请求下一个要爬取的URL。
4. 调度器返回下一个要爬取的URL给引擎，引擎将URL通过下载中间件(请求(request)方向)转发给下载器(Downloader)。
5. 一旦页面下载完毕，下载器生成一个该页面的Response，并将其通过下载中间件(返回(response)方向)发送给引擎。
6. 引擎从下载器中接收到Response并通过Spider中间件(输入方向)发送给Spider处理。
7. Spider处理Response并返回爬取到的Item及(跟进的)新的Request给引擎。
8. 引擎将(Spider返回的)爬取到的Item给Item Pipeline，将(Spider返回的)Request给调度器。
9. 重复第2到第8步，直到调度器(Scheduler)中没有更多地request，引擎关闭该网站。

**二、Scrapy的安装**

Scrapy是一个十分强大的爬虫框架，依赖的库比较多，至少需要依赖的库有Twisted 14.0、lxml 3.4和pyOpenSSL 0.14。在不同的平台环境下，它所依赖的库也各不相同，所以在安装之前，最好确保把一些基本库安装好。

**2.1 Windows环境**

如果你的Python不是使用Anaconda安装的，可以参考如下方式来一步步安装Scrapy。

**1、安装wheel**

pip install wheel

**2、安装lxml**

<https://pypi.python.org/pypi/lxml/4.1.0>

**3、安装pyopenssl**

<https://pypi.python.org/pypi/pyOpenSSL/17.5.0>

**4、安装Twisted**

<https://www.lfd.uci.edu/~gohlke/pythonlibs/>

**5、安装pywin32**

<https://sourceforge.net/projects/pywin32/files/>

**6、安装scrapy**

pip install scrapy

**2.2 Ubuntu环境**

在Ubuntu平台下，首先确保一些依赖库已经安装，运行如下命令：

sudo apt-get install build-essential python3-dev libssl-dev libffi-dev libxml2 libxml2-dev

libxslt1-dev zlib1g-dev

然后利用pip安装Scrapy即可：

pip3 install Scrapy

**三、Scrapy入门**

这节我们主要介绍一个简单的项目，完整的看一遍项目流程。可以对Scrapy的基本用法和原理有大体了解。

在项目开始之前，需要保证已经安装好了Scrapy框架、MongoDB和pymongo库。

这里我们主要来爬取官方文档

**3.1 创建项目**

可以切换到相对应的文件路径下面(也可以通过路径来创建)，通过命令来创建：

scrapy startproject tutorial

通过这个命令，会创建一个tutorial文件夹，文件夹内包含以下内容：

tutorial**/**

scrapy**.**cfg *# Scrapy部署时的配置文件*

tutorial**/** *# 项目的模块，需要从这里引入*

\_\_init\_\_**.**py

items**.**py *# Items的定义，定义爬取的数据结构*

middlewares**.**py *# Middlewares的定义，定义爬取时的中间件*

pipelines**.**py *# Pipelines的定义，定义数据管道*

settings**.**py *# 配置文件*

spiders**/** *# 放置Spiders的文件夹*

\_\_init\_\_**.**py

**3.2 创建Spider**

Spider是一个自己定义的类，可以从一个网站（或者一组网站）上抓取信息。不过自定义的这个类必须继承Scrapy提供的scrapy.Spider，还要定义Spider的名称和起始请求，以及怎么获得响应的方法。

通过请求[http://quotes.toscrape.com](http://quotes.toscrape.com/)，来建立spider，通过执行以下命令：

cd tutorial

scrapy genspider quotes quotes.toscrape.com

通过scrapy genspider命令来执行，第一个参数为spider的名称，第二个参数是网站域名。这样在spiders文件会多了一个quotes.py文件，内容如下：

import scrapy

class QuotesSpider(scrapy.Spider):

name = 'quotes' #每个项目有唯一的名字，用来区分不同的spider

allowed\_domains = ['quotes.toscrape.com'] #允许爬取的域名，如果请求不在该域名下，会被过滤

start\_urls = ['http://quotes.toscrape.com/']#包含了Spider启动时的url列表，初始请求由他定义

#parse()在默认情况下调用start\_urls请求完成后返回的响应作为他的一个参数，主要负责解析和进一步请求

def parse(self, response):

pass

**3.3 创建Item**

Item时保存数据的容器，使用方法类似于字典，不过有额外的保护。在item.py文件中进行如下修改：

from scrapy import Item, Field

class QuoteItem(Item):

# define the fields for your item here like:

# name = scrapy.Field()

text = Field()

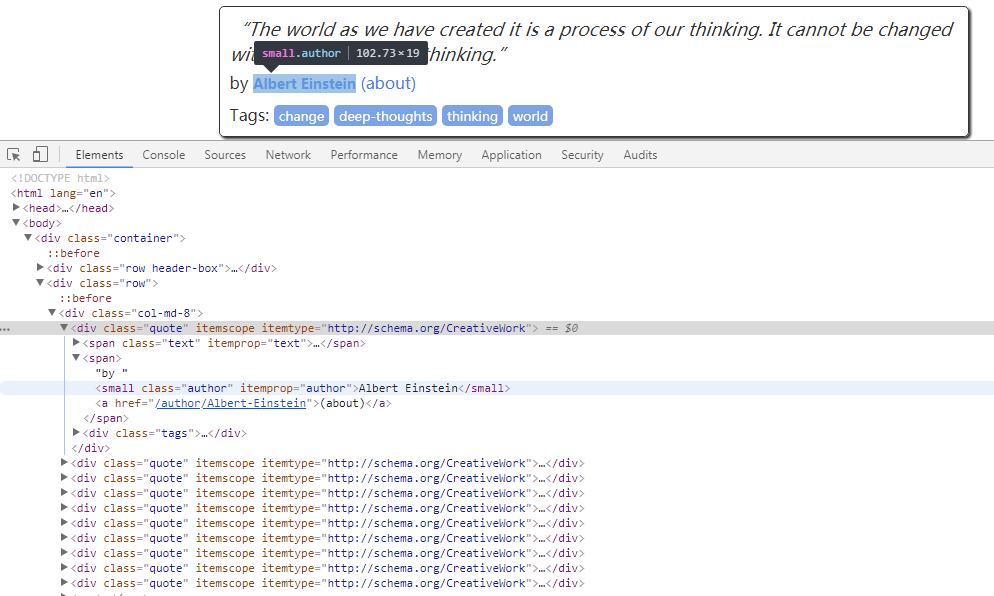
author = Field()

tags = Field()

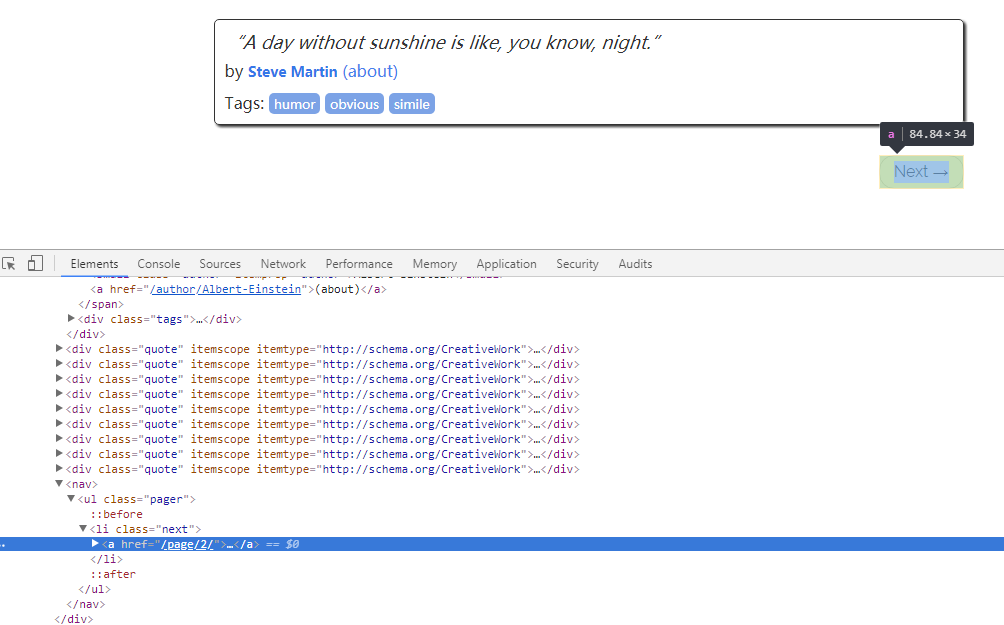
这里主要定义了三个字段，在接下来的解析Response中会用到这三个字段。

**3.4 解析Response**

parse()方法中，可以直接对response变量包含的内容进行解析。我们可以直接查看源代码：



同时，还需要考虑到翻页问题：



我们可以通过css选择器或者xpath选择，对我们想要的信息进行提取，parse()方法改写如下：

def parse(self, response):

quotes = response.css('.quote')

for quote in quotes:

item = QuoteItem()

item['text'] =quote.css('.text::text').extract\_first()

item['author'] = quote.css('.author::text').extract\_first()

item['tags'] = quote.css('.tags .tag::text').extract()

yield item

next\_page = response.css('.pager .next a::attr(href)').extract\_first()

url = response.urljoin(next\_page)

yield scrapy.Request(url=url,callback=self.parse)

可以通过在命令行，执行以下命令进行运行：

scrapy crawl quotes

**3.5 使用Item Pipeline**

如果想将数据保存到MongoDB，或者筛选某些有用的Item，就可以通过定义Item Pipeline来定义。

接下来我们主要想实现筛选text长度大于50的Item，并将结果保存到MongoDB：

import pymongo

from scrapy.exceptions import DropItem

class TextPipeline(object):

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

self.limit = 50

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

if item['text']:

if len(item['text']) > self.limit:

item['text'] = item['text'][0:self.limit].rstrip() + '...'

return item

else:

return DropItem('没有文本数据！')

class MongoPipeline(object):

def \_\_init\_\_(self,mongo\_url,mongo\_db):

self.mongo\_url = mongo\_url

self.mongo\_db = mongo\_db

@classmethod

def from\_crawler(cls,crawler):

return cls(

mongo\_url=crawler.settings.get('MONGO\_URL'),

mongo\_db = crawler.settings.get('MONGO\_DB')

)

def open\_spider(self,spider):

self.client = pymongo.MongoClient(self.mongo\_url)

self.db = self.client[self.mongo\_db]

def close\_spider(self,spider):

self.client.close()

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

name = item.\_\_class\_\_.\_\_name\_\_

self.db[name].insert(dict(item))

return item

我们需要将定义好的TextPipeline和MongoPipeline两个类在settings.py文件，进行配置：

ITEM\_PIPELINES = {

'toscrape.pipelines.MongoPipeline': 400,

'toscrape.pipelines.MongoPipeline': 300,

}

MONGO\_URL = 'localhost'

MONGO\_DB = 'ToScape'

再次运行就可以将获取和处理的数据保存到MongoDB里面。