**114.简述 requests模块的作用及基本使用？**

# 作用：

使用requests可以模拟浏览器的请求

# 常用参数：

url、headers、cookies、data、json、params、proxy

# 常用返回值：

content

iter\_content

text

encoding="utf-8"

cookie.get\_dict()

**115.简述 Beautifulsoup模块的作用及基本使用？**

#BeautifulSoup

用于从HTML或XML文件中提取、过滤想要的数据形式

#常用方法

解析：html.parser 或者 lxml（需要下载安装） 、find、find\_all、text、attrs、get   
<https://www.cnblogs.com/wcwnina/p/8093987.html>

**116.简述 seleninum模块的作用及基本使用?**

Selenium是一个用于Web应用程序测试的工具，他的测试直接运行在浏览器上，模拟真实用户，按照代码做出点击、输入、打开等操作，爬虫中使用他是为了解决requests无法解决javascript动态问题

**117.Scrapy框架中各组件的工作流程？**

Scrapy 使用了 Twisted 异步非阻塞网络库来处理网络通讯，整体架构大致如下（绿线是数据流向）：



Scrapy主要包括了以下组件：

* **引擎(Scrapy)**  
  *用来处理整个系统的数据流处理, 触发事务(框架核心)*
* **调度器(Scheduler)**  
  *用来接受引擎发过来的请求, 压入队列中, 并在引擎再次请求的时候返回. 可以想像成一个URL（抓取网页的网址或者说是链接）的优先队列, 由它来决定下一个要抓取的网址是什么, 同时去除重复的网址*
* **下载器(Downloader)**  
  *用于下载网页内容, 并将网页内容返回给蜘蛛(Scrapy下载器是建立在twisted这个高效的异步模型上的)*
* **爬虫(Spiders)**  
  *爬虫是主要干活的, 用于从特定的网页中提取自己需要的信息, 即所谓的实体(Item)。用户也可以从中提取出链接,让Scrapy继续抓取下一个页面*
* **项目管道(Pipeline)**  
  *负责处理爬虫从网页中抽取的实体，主要的功能是持久化实体、验证实体的有效性、清除不需要的信息。当页面被爬虫解析后，将被发送到项目管道，并经过几个特定的次序处理数据。*
* **下载器中间件(Downloader Middlewares)**  
  *介于Scrapy引擎和下载器之间的中间件，主要是处理Scrapy引擎与下载器之间的请求及响应。*
* **爬虫中间件(Spider Middlewares)**  
  *介于Scrapy引擎和爬虫之间的中间件，主要工作是处理蜘蛛的响应输入和请求输出。*
* **调度中间件(Scheduler Middewares)**  
  *介于Scrapy引擎和调度之间的中间件，从Scrapy引擎发送到调度的请求和响应。*

Scrapy运行流程大概如下：

1.引擎：Hi！Spider, 你要处理哪一个网站？

2.Spider：老大要我处理xxxx.com（初始URL）。

3.引擎：你把第一个需要处理的URL给我吧。

4.Spider：给你，第一个URL是xxxxxxx.com。

5.引擎：Hi！调度器，我这有request请求你帮我排序入队一下。

6.调度器：好的，正在处理你等一下。

7.引擎：Hi！调度器，把你处理好的request请求给我。

8.调度器：给你，这是我处理好的request

9.引擎：Hi！下载器，你按照老大的下载中间件的设置帮我下载一下这个request请求。

10.下载器：好的！给你，这是下载好的东西。（如果失败：sorry，这个request下载失败了。然后引擎告诉调度器，这个request下载失败了，你记录一下，我们待会儿再下载）

11.引擎：Hi！Spider，这是下载好的东西，并且已经按照老大的下载中间件处理过了，你自己处理一下（注意！这儿responses默认是交给def parse()这个函数处理的）

12.Spider：（处理完毕数据之后对于需要跟进的URL），Hi！引擎，我这里有两个结果，这个是我需要跟进的URL，还有这个是我获取到的Item数据。

13.引擎：Hi ！管道 我这儿有个item你帮我处理一下！调度器！这是需要跟进URL你帮我处理下。然后从第四步开始循环，直到获取完老大需要全部信息。

14.管道、调度器：好的，现在就做！

<https://blog.csdn.net/qq_37143745/article/details/80996707>

**118.在scrapy框架中如何设置代理（两种方法）？**

方式一：内置添加代理功能

# -\*- coding: utf-8 -\*-

import os

import scrapy

from scrapy.http import Request

class ChoutiSpider(scrapy.Spider):

name = 'chouti'

allowed\_domains = ['chouti.com']

start\_urls = ['https://dig.chouti.com/']

def start\_requests(self):

os.environ['HTTP\_PROXY'] = "http://192.168.11.11"

for url in self.start\_urls:

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

def parse(self, response):

print(response)

方式二：自定义下载中间件

import random

import base64

import six

def to\_bytes(text, encoding=None, errors='strict'):

"""Return the binary representation of `text`. If `text`

is already a bytes object, return it as-is."""

if isinstance(text, bytes):

return text

if not isinstance(text, six.string\_types):

raise TypeError('to\_bytes must receive a unicode, str or bytes '

'object, got %s' % type(text).\_\_name\_\_)

if encoding is None:

encoding = 'utf-8'

return text.encode(encoding, errors)

class MyProxyDownloaderMiddleware(object):

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

proxy\_list = [

{'ip\_port': '111.11.228.75:80', 'user\_pass': 'xxx:123'},

{'ip\_port': '120.198.243.22:80', 'user\_pass': ''},

{'ip\_port': '111.8.60.9:8123', 'user\_pass': ''},

{'ip\_port': '101.71.27.120:80', 'user\_pass': ''},

{'ip\_port': '122.96.59.104:80', 'user\_pass': ''},

{'ip\_port': '122.224.249.122:8088', 'user\_pass': ''},

]

proxy = random.choice(proxy\_list)

if proxy['user\_pass'] is not None:

request.meta['proxy'] = to\_bytes("http://%s" % proxy['ip\_port'])

encoded\_user\_pass = base64.encodestring(to\_bytes(proxy['user\_pass']))

request.headers['Proxy-Authorization'] = to\_bytes('Basic ' + encoded\_user\_pass)

else:

request.meta['proxy'] = to\_bytes("http://%s" % proxy['ip\_port'])

配置：

DOWNLOADER\_MIDDLEWARES = {

# 'xiaohan.middlewares.MyProxyDownloaderMiddleware': 543,

}

**119.scrapy框架中如何实现大文件的下载？**

from twisted.web.client import Agent, getPage, ResponseDone, PotentialDataLoss

from twisted.internet import defer, reactor, protocol

from twisted.web.\_newclient import Response

from io import BytesIO

class \_ResponseReader(protocol.Protocol):

def \_\_init\_\_(self, finished, txresponse, file\_name):

self.\_finished = finished

self.\_txresponse = txresponse

self.\_bytes\_received = 0

self.f = open(file\_name, mode='wb')

def dataReceived(self, bodyBytes):

self.\_bytes\_received += len(bodyBytes)

# 一点一点的下载

self.f.write(bodyBytes)

self.f.flush()

def connectionLost(self, reason):

if self.\_finished.called:

return

if reason.check(ResponseDone):

# 下载完成

self.\_finished.callback((self.\_txresponse, 'success'))

elif reason.check(PotentialDataLoss):

# 下载部分

self.\_finished.callback((self.\_txresponse, 'partial'))

else:

# 下载异常

self.\_finished.errback(reason)

self.f.close()

**120.scrapy中如何实现限速？**

http://scrapy-chs.readthedocs.io/zh\_CN/1.0/topics/autothrottle.html

**121.scrapy中如何实现暂停爬虫？**

# 有些情况下，例如爬取大的站点，我们希望能暂停爬取，之后再恢复运行。

# Scrapy通过如下工具支持这个功能:

一个把调度请求保存在磁盘的调度器；

一个把访问请求保存在磁盘的副本过滤器[duplicates filter]；

一个能持续保持爬虫状态(键/值对)的扩展；

Job 路径：

要启用持久化支持，你只需要通过 JOBDIR 设置 job directory 选项。

这个路径将会存储所有的请求数据来保持一个单独任务的状态(例如：一次spider爬取(a spider run))。

必须要注意的是，这个目录不允许被不同的spider共享，甚至是同一个spider的不同jobs/runs也不行。

也就是说，这个目录就是存储一个单独 job的状态信息。

**122.scrapy中如何进行自定制命令？**

在spiders同级创建任意目录，如：commands

在其中创建'crawlall.py'文件（此处文件名就是自定义的命令）

from scrapy.commands import ScrapyCommand

from scrapy.utils.project import get\_project\_settings

class Command(ScrapyCommand):

requires\_project = True

def syntax(self):

return '[options]'

def short\_desc(self):

return 'Runs all of the spiders'

def run(self, args, opts):

spider\_list = self.crawler\_process.spiders.list()

for name in spider\_list:

self.crawler\_process.crawl(name, \*\*opts.\_\_dict\_\_)

self.crawler\_process.start()

在'settings.py'中添加配置'COMMANDS\_MODULE = '项目名称.目录名称''

在项目目录执行命令：'scrapy crawlall'

**123.scrapy中如何实现的记录爬虫的深度？**

'DepthMiddleware'是一个用于追踪每个Request在被爬取的网站的深度的中间件。其可以用来限制爬取深度的最大深度或类似的事情。

'DepthMiddleware'可以通过下列设置进行配置(更多内容请参考设置文档):

'DEPTH\_LIMIT':爬取所允许的最大深度，如果为0，则没有限制。

'DEPTH\_STATS':是否收集爬取状态。

'DEPTH\_PRIORITY':是否根据其深度对requet安排优先

**124.scrapy中的pipelines工作原理？**

Scrapy 提供了 pipeline 模块来执行保存数据的操作。

在创建的 Scrapy 项目中自动创建了一个 pipeline.py 文件，同时创建了一个默认的 Pipeline 类。

我们可以根据需要自定义 Pipeline 类，然后在 settings.py 文件中进行配置即可

**125.scrapy的pipelines如何丢弃一个item对象？**

通过raise DropItem**()**方法

**126.简述scrapy中爬虫中间件和下载中间件的作用？**

http://www.cnblogs.com/wupeiqi/articles/6229292.html

**127.scrapy-redis组件的作用？**

实现了分布式爬虫，url去重、调度器、数据持久化；

'scheduler'调度器

'dupefilter'URL去重规则（被调度器使用）

'pipeline'数据持久化

**128.scrapy-redis组件中如何实现的任务的去重？**

a. 内部进行配置，连接Redis；  
b.去重规则通过redis的集合完成，集合的Key为：  
   key **=**defaults.DUPEFILTER\_KEY **% {'timestamp':**int**(**time.time**())}**默认配置：DUPEFILTER\_KEY **= 'dupefilter:%(timestamp)s'**c.去重规则中将url转换成唯一标示，然后在redis中检查是否已经在集合中存在  
   **from**scrapy.utils **import**request  
   **from**scrapy.http **import**Request  
   req **=**Request**(**url**='http://www.cnblogs.com/wupeiqi.html')**result **=**request.request\_fingerprint**(**req**)**print**(**result**)**# 8ea4fd67887449313ccc12e5b6b92510cc53675c

scrapy和scrapy-redis的去重规则（源码）

1. scrapy中去重规则是如何实现？

class RFPDupeFilter(BaseDupeFilter):

"""Request Fingerprint duplicates filter"""

def \_\_init\_\_(self, path=None, debug=False):

self.fingerprints = set()

@classmethod

def from\_settings(cls, settings):

debug = settings.getbool('DUPEFILTER\_DEBUG')

return cls(job\_dir(settings), debug)

def request\_seen(self, request):

# 将request对象转换成唯一标识。

fp = self.request\_fingerprint(request)

# 判断在集合中是否存在，如果存在则返回True，表示已经访问过。

if fp in self.fingerprints:

return True

# 之前未访问过，将url添加到访问记录中。

self.fingerprints.add(fp)

def request\_fingerprint(self, request):

return request\_fingerprint(request)

2. scrapy-redis中去重规则是如何实现？

class RFPDupeFilter(BaseDupeFilter):

"""Redis-based request duplicates filter.

This class can also be used with default Scrapy's scheduler.

"""

logger = logger

def \_\_init\_\_(self, server, key, debug=False):

# self.server = redis连接

self.server = server

# self.key = dupefilter:123912873234

self.key = key

@classmethod

def from\_settings(cls, settings):

# 读取配置，连接redis

server = get\_redis\_from\_settings(settings)

# key = dupefilter:123912873234

key = defaults.DUPEFILTER\_KEY % {'timestamp': int(time.time())}

debug = settings.getbool('DUPEFILTER\_DEBUG')

return cls(server, key=key, debug=debug)

@classmethod

def from\_crawler(cls, crawler):

return cls.from\_settings(crawler.settings)

def request\_seen(self, request):

fp = self.request\_fingerprint(request)

# This returns the number of values added, zero if already exists.

# self.server=redis连接

# 添加到redis集合中：1，添加工程；0，已经存在

added = self.server.sadd(self.key, fp)

return added == 0

def request\_fingerprint(self, request):

return request\_fingerprint(request)

def close(self, reason=''):

self.clear()

def clear(self):

"""Clears fingerprints data."""

self.server.delete(self.key)

项目发布的过程（上传-发布），上线，发布，测试，谁发布测试

pandas库有哪些函数，介绍pandas函数的作用：

前言：本博文摘抄自中国慕课大学上的课程《Python数据分析与展示》，推荐刚入门的同学去学习，这是非常好的入门视频。

继续一个新的库，Pandas库。Pandas库围绕Series类型和DataFrame类型这两种数据结构，提供了一种高效便捷的数据处理方式。

一、常用功能及函数简介

包导入

一般我们需要做如下导入，numpy和pandas一般需要联合使用：

import pandas as pd

import numpy as np

本文采用如下缩写：

df：Pandas DataFrame对象

s：  Pandas Series对象

数据导入

pd.read\_csv(filename)：从CSV文件导入数据

pd.read\_table(filename)：从限定分隔符的文本文件导入数据

pd.read\_excel(filename)：从Excel文件导入数据

pd.read\_sql(query, connection\_object)：从SQL表/库导入数据

pd.read\_json(json\_string)：从JSON格式的字符串导入数据

pd.read\_html(url)：解析URL、字符串或者HTML文件

pd.read\_clipboard()：从粘贴板获取内容

pd.DataFrame(dict)：从字典对象导入数据

数据导出

df.to\_csv(filename)：导出数据到CSV文件

df.to\_excel(filename)：导出数据到Excel文件

df.to\_sql(table\_name, connection\_object)：导出数据到SQL表

df.to\_json(filename)：以Json格式导出数据到文本文件

创建对象

pd.DataFrame(np.random.rand(20,5))：创建20行5列的随机数组成的DataFrame对象

pd.Series(my\_list)：从可迭代对象my\_list创建一个Series对象

df.index = pd.date\_range('1900/1/30', periods=df.shape[0])：增加一个日期索引

index和reindex联合使用很有用处，index可作为索引并且元素乱排序之后，所以跟着元素保持不变，因此，当重拍元素时，只需要对index进行才重排即可:reindex。

另外， reindex时，还可以增加新的标为NaN的元素。

数据查看

df.head(n)：查看DataFrame对象的前n行

df.tail(n)：查看DataFrame对象的最后n行

df.shape()：查看行数和列数

df.info()：查看索引、数据类型和内存信息

df.describe()：查看数值型列的汇总统计

s.value\_counts(dropna=False)：查看Series对象的唯一值和计数

df.apply(pd.Series.value\_counts)：查看DataFrame对象中每一列的唯一值和计数

apply的用处很多，比如可以通过跟lambda函数联合，完成很多功能：将包含某个部分的元素挑出来等等。

cities['Is wide and has saint name'] = (cities['Area square miles'] > 50) & cities['City name'].apply(lambda name: name.startswith('San'))

数据选取

df[col]：根据列名，并以Series的形式返回列

df[[col1, col2]]：以DataFrame形式返回多列

s.iloc[0]：按位置选取数据

s.loc['index\_one']：按索引选取数据

df.iloc[0,:]：返回第一行

数据清洗

df.columns = ['a','b','c']：重命名列名

pd.isnull()：检查DataFrame对象中的空值，并返回一个Boolean数组

pd.notnull()：检查DataFrame对象中的非空值，并返回一个Boolean数组

df.dropna()：删除所有包含空值的行

df.fillna(x)：用x替换DataFrame对象中所有的空值

s.astype(float)：将Series中的数据类型更改为float类型

s.replace(1,'one')：用‘one’代替所有等于1的值

df.rename(columns=lambda x: x + 1)：批量更改列名

df.set\_index('column\_one')：更改索引列

数据处理：Filter, Sort, GroupBy

df[df[col] > 0.5]：选择col列的值大于0.5的行

df.sort\_values(col1)：按照列col1排序数据，默认升序排列

df.groupby(col)：返回一个按列col进行分组的Groupby对象

df.groupby(col1).agg(np.mean)：返回按列col1分组的所有列的均值

df.pivot\_table(index=col1, values=[col2,col3], aggfunc=max)：创建一个按列col1进行分组，并计算col2和col3的最大值的数据透视表

data.apply(np.mean)：对DataFrame中的每一列应用函数np.mean

数据合并

df1.append(df2)：将df2中的行添加到df1的尾部

df.concat([df1, df2],axis=1)：将df2中的列添加到df1的尾部

df1.join(df2,on=col1,how='inner')：对df1的列和df2的列执行SQL形式的join

数据统计

df.describe()：查看数据值列的汇总统计

df.mean()：返回所有列的均值

df.corr()：返回列与列之间的相关系数

df.count()：返回每一列中的非空值的个数

df.max()：返回每一列的最大值

df.min()：返回每一列的最小值

df.median()：返回每一列的中位数

df.std()：返回每一列的标准差

Pandas支持的数据类型

int 整型

float 浮点型

bool 布尔类型

object 字符串类型

category 种类

datetime 时间类型

补充：

df.astypes: 数据格式转换

df.value\_counts:相同数值的个数统计

df.hist(): 画直方图

df.get\_dummies: one-hot编码，将类型格式的属性转换成矩阵型的属性。比如：三种颜色RGB，红色编码为[1 0 0]