# 爬虫手册

# 爬虫应对方式Python

1. User-Agent
2. Cookie/toking
3. 代理
4. 图像识别验证码
5. selenium + PhantomJs模拟真实游览器加载js

验证码、动态加载（重构页面）、加密数据（js中）

# 请求报文

GET / HTTP/1.1 #协议

User-Agent: #必填，用户认证

Accept-Encoding: gzip, deflate, br #压缩方式，不能填写

Cookie: BAIDUID=AAAAAAAAA; #获取登陆后状态，

Host: www.baidu.com #选填

Connection: keep-alive #连接方式、选填

Accept:#返回数据

Accept-Language: zh-CN,zh;q=0.9 #语言

“User-Agent”:” Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/70.0.3538.110 Safari/537.36”

# 使用方法

**Get: //简单请求**

1、构造数据

2、4.1：data不需要数值

2、4.2

**POST：//简单请求**

1、构造数据

2、4.3：转换数据

3、4.1：将数据放入data

4、4.2

**携带COOKIE请求**

1、构造数据：header头数据需要填写COOKIE项

2、4.3：转换数据

3、4.1：将数据放入data

4、4.2

**COOKIE**

1、4.12

2、4.13

3、4.5

4、4.6

5、read

**HTTPHandler**

1、4.4

2、4.5

3、4.1：构造一个请求

5、4.6

6、4.2/opener.read

**登陆网站验证**

**--站点开启验证信息**

1、4.9

2、4.10

3、4.5

4、4.1

5、4.2/opener.read

# 底层方法

**Python3将库urllib2改名为urllib,并包请求方法包含在request对象**

urllib.request HTTP请求模块

-->Request 用于构造复杂的请求，通过urlopen发送

-->urlopen 可发送Request构建的请求，可直接发送请求，最简单的方法，这一个系统定义的opener模块

-->Handler http处理器，更灵活的请求方法  
urllib.error 异常处理模块

-->error.URLError 错误类型，使用try语法  
urllib.parse URL解析模块

-->urlencode 编码解析，常用  
urllib.robotparser robots.txt解析模块

## 1/ urllib.request.Request(Url,data,headers,method)

***构造一个url请求***

--URl ：请求地址

--data： base64数据，主要适用于POST请求，**需要parse编码**

--header：报文头，**字典类型**，可使用其他方法添加

--method：get、POST

--**Re：返回值**

**--导入模块：urllib.requset**

--------------------

支持方法：

Re.add\_header(“User-Agent”, header\_string) #添加header头

Re.get\_header(“User-agent”) #获取header指定信息

## 2/ urllib.request.urlopen(re,data,timeout)

***打开一个url/request对象,发送请求（可携带参数）***

--re：urllib.request.Request的返回对象/URl地址

--data：data=data请求数据，可不带（GET），携带（POST）

--timeout：设置超时时间

**--reponse：返回值**

**--导入模块：urllib.request**

**-----------------**

支持方法:

#对HTTPResponse类型数据进行操作

**reponse**.read() / .readline() / .readlines()/ .fileno()/ .close()

#返回HTTPMessage对象，表示远程服务器返回的头信息

**reponse**.info()

**reponse**.getcode()#返回Http状态码

**reponse**.geturl() #返回请求的url

**reponse**.status # 响应状态，属性

**reponse**.msg # 信息，属性

**reponse**.version #版本，属性

#转码，读取响应数据

**reponse**.read().decode(“utf-8”)

# 附录：转码

b：bytes编码 u：unicode编码 r：非转义字符

str.encode('utf-8')  
bytes.decode('utf-8')

## 3/ parse.urlencode(form\_data,encoding=’utf8’)

***将字符串转换为url编码***

--form\_data：数据，字典

--encoding=’utf8’：编码

**--导入模块：urllib**

--------------------

注意：

Post的数据必须是bytes或者iterable of bytes；

当urllib.request.Request()函数使用时还应使用

data = bytes(urllib.parse.urlencode({'word': 'hello'}), encoding='utf8')

response = urllib.request.urlopen('http://httpbin.org/post', data=data)

**bytes(from, encode=”utf8”)**

防止出现 'str' object has no attribute 'items'

## 4/ HTTPHandler

### Handler处理器分类

1. HTTPHandler() ：没有任何特殊功能
2. ProxyHandler(普通代理)
3. ProxyBasicAuthHandler(密码管理器对象) ：私密代理

4、HTTPBasicAuthHandler(密码管理器对象) : web客户端认证

### 一般使用流程

1. 创建相关的Handler处理器对象

http\_handler = urllib.request.HTTPHandler()

2、创建自定义opener对象

opener = urllib.request.build\_opener(http\_handler)

1. 利用opener对象打开url

**req** = urllib.request.Request(url,headers=headers)

res = opener.open(**req**)

result = res.read().decode("utf-8")

### 一般使的函数

#### （1）urllib.HTTPHandler(debuglevel)

***构建一个HTTPHandler处理对象，支持HTTP请求***

--debuglevel：debug等级1，自动开启模式

**--导入模块：urllib**

**--返回值：http\_handler**

#### （2）urllib.build\_opener(http\_handler, proxy)

***构建一个处理对象，可添加多个处理器对象***

--http\_handler：HTTPHandler返回的对象

--proxy：代理处理器，proxyBasicAuthHandler对象

**--导入模块：urllib**

**--返回值：opener**

**------------**

opener**.**addheaders：添加head头，数组

#### （3）opener.open(re)

*打开一个url连接*

--re：urllib.request.Request()返回的构造对象

**--导入模块：urllib**

**--返回值：response**

**---------------**

response.read 读取数据

#### （5）ProxyHandle(disc)

***普通代理代理服务器***

--disc：字典{“http”:”127.0.0.1:80”}

使用有认证的代理{“http”:”root:密码@127.0.0.1:80”}

**--导入模块：urllib**

**--返回值：HTTP\_proxy**

#### （6）urllib.install\_opener(opener)

***创建一个全局opener对象，多个opener对象发送信息 使用***

--opener：urllib.build\_opener创建的对象

**--导入模块：urllib**

**--返回值：无**

### 密码/代理程序实现流程

***密码管理器对象作用***

**（1）私密代理（2）Web客户端认证**

***操作流程***

（1）创建密码管理器对象

pwdmg = urllib.request.HTTPPasswordMgrWithDefaultRealm()

（2）把认证信息添加到密码管理器对象

pwdmg.add\_password(None,webserver,user,passwd)

1. 创建Handler处理器对象

--->私密代理

proxy = urllib.request.ProxyAuthBasicHandler(pwdmg)

--->Web客户端

webbasic = urllib.request.HTTPBasicAuthHandler(pwdmg)

1. 使用openner
2. request构造

### 密码/代理使用的函数

#### （1）HTTPPasswordMgrWithDefaultRealm()

***密码管理对象***

**--导入模块：urllib**

**--返回值：passwordMgr**

-------------

方法：

“””

授权账户信息

@realm：域，没有天线none

@webserver：服务器地址

@root：账户

@password：密码

“””

passwordMgr.add\_password(realm, webserver, root, password)

#### （2）Urllib.HTTPBasicAuthHandler(passwordMgr)

***密码认证方法***

--passwordMgr：HTTPPasswordMgrWithDefaultRealm对象

**--导入模块：urllib**

**--返回值：http\_handler，可用于urllib.build\_opener()**

#### （2）urllib.ProxyBasicAuthHandler(passwordMgr)

***代理处理管理器类***

--passwordMgr：HTTPPasswordMgrWithDefaultRealm对象

**--导入模块：urllib**

**--返回值：proxyauth\_handler，可用于urllib.build\_opener()**

### cookie

#### （1）cookielib.cookiejar()

***构建一个cookiejar对象，用于保存cookie的值***

**--导入模块：cookie**

**--返回值：cookie对象**

#### （2）urllib.HTTPCookieProcessor(cookie)

***构建一个处理对象用于处理cookie***

--cookie：cookie.cookiejar返回的对象

**--导入模块：urllib**

**--返回值：cookie\_handler**

## 5/ error.URLError

from urllib import request, error

try:

response = request.urlopen('http://cuiqcoex.htm')

except **error.HTTPError** as e:

print(**e.reason, e.code, e.headers**, sep='\n')

except **error.URLError** as e:

print(**e.reason**)

else:

print('Request Successfully')

# 解析语法

## 正则

## beautifulsoup

1. from bs4 import BeautifulSoup
2. import requests #为了使用session
3. sess = requests.session() #创建一个session对象
4. 发送请求

#使用GET发送请求

**html** = sess.get(“https://www.baidu.com”, headers).text

#使用POST请求

**html** = sess.POST(“https://www.baidu.com”, data , headers).text -----------------------------------------------------------------------

以上也可以使用urllib的cookiejer获取cookie

-----------------------------------------------------------------------

1. bs = BeautifulSoup(**html**, ‘lxml’) #使用beautifulSoup转换文档
2. 获取指定节点值

result = bs.find(“input”, attrs={“name”:”\_fxsrf}).get(“value”)

input：标签

attrs：属性{“value: value”}

get()：要获取的值

## lxml

1、from lxml import etree

2、通过请求获取HTML

3、content = etree.HTML(html) #转换html成dom树

4、result = content.xpath(“//div[@class]”) #通过节点查找字符

# 六、selenium/phantomJS

web自动化测试工具

phantomJS无界面游览器（需要从网站下载）

## 导入文件

#webdriver调用有连起和操作

from selenium import webdriver

#导入key，模仿操作

from selenium.webdriver.common.key import Keys

#测试模块

import unittest

driver.get(“www.baidu.com”) #获取页面

driver.find\_element\_by\_name(“name”).send\_keys(“value”)

driver.find\_element\_by\_id(“xxx”).click()

driver.save\_screenshot(“xxx.png”) #截屏

# 七：机器视觉/Tesseract

## 安装

## 使用

import pytesseract

from PIL import Image

image = Image.open(“”)

str = image\_to\_string(image) #将图片转换为文本

# scrapy

## 步骤

1、安装：**pip install Scrapy**

2、新建项目：**scrapy startproject** xxx

3、**配置文件**（根目录）

4、配置**item.py** （定义数据）

5、**scrapy genspider** 爬虫名 “域名”

6、spiders目录下的文件配置

7、修改parse方法

8、setting.py开启管道文件

9、修改管道文件pipeline

## 目录详解

lion

├── lion **#项目的python模块，将在这里引入代码**

│   ├── \_\_init\_\_.py

│   ├── items.py **#项目目标文件**

│   ├── middlewares.py

│   ├── pipelines.py **#项目的管道文件**

│   ├── \_\_pycache\_\_

│   ├── settings.py **#项目的设置文件**

│   └── spiders**存储爬虫代码的文件（主要）**

│   ├── \_\_init\_\_.py

│   └── \_\_pycache\_\_

└── scrapy.cfg #**项目配置文件**

## 命令

scrapy crawl mySpider #根目录运行，执行指定的爬虫

scrapy genspider 爬虫名 “域名” #爬虫名不能与项目名相同

scrapy crawl mySpider -o mydata.csv #保存为csv格式

## 配置文件

**查看手册scrapy框架->setting**

**DOWNLOAD\_DELAY**

#下载间隔

#一般设置2秒，减轻服务器压力

DOWNLOAD\_DELAY = 2

ITEM\_PIPELINES 管道文件

**DEFAULF\_QEQUEST\_HEADERS**

# 修改默认请求头文件，特别复杂的请求需要在**下载中间件**中修改:

DEFAULT\_REQUEST\_HEADERS = {

"User-Agent": "Mozilla/5.0 like Gecko) Version/5.0.2 Mobile/8J2",

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

# 'Accept-Language': 'en',

}

LOG\_FILE #日志

LOG\_LEVEL = ‘INFO’

**COOKIES\_ENABLED**

# Disable cookies (enabled by default)

# 关掉cookie，禁止服务器记录自己

COOKIES\_ENABLED = False

## 1、使用setting文件

form scrapy.utils.project import get\_porject\_settings

get\_porject\_setting().get(“xxx”)

## 1、定义数据类型

创建保存数据的类，类似于django中的model类

/items.py

import scrapy

class Myspider(scrapy.Item): #这个名字可以自定义

name = scrapy.Field() #要抓取的字段保存变量

level = scrapy.Field()

## 2、创建爬取文件

### 1、scrapy genspider mySpider “www.baidu.com”

使用spider模板创建，创建一个名为mySpider的爬虫类，爬取域名为www.baidu.com

#当然可以手工创建，但代码都是重复的，省去麻烦

import scrapy

class MySpider(scrapy.Spider):

name = “mySpider” #爬虫类的名字, 唯一

allowd\_domains=[‘baidu.cn’] #限制爬取的域名

start\_url = (

“www.baidu.com”,

) #开始爬取的第一页

offset = 0 #设置偏移量，动态修改页面地址

def parse(self, response): #解析方法

#解析方法，初始url完成下载后将被调用，调用的时候每一个url回传的Response对象作为唯一的参数，作用如下

#解析网页传回的数据，提取数据生成item

#生成下一个url

# response.body = response.read()

# response.xpath()#scrapy自带

#request.text[] 获取文本数据

list = response.xpath(“//div[@class=”lit”])

for each in list:

name = each.xpath(“./h3/text()”).extract()# 将对象转换为文本

return #直接返回程序

### 2、scrapy genspider -t crawl 爬虫名 域名

使用CrawlSpider模板创建，更快捷方便

**（1）start\_url请求**

**（2）linkExtractor匹配返回页面中指定url**

**（3）发送匹配的url**

**（4）返回数据，调用callback**

**（5）process\_request发送新请求**

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

import scrapy

from scrapy.spiders import CrawlSpider, Rule

from scrapy.linkextractors import LinkExtractor

class YyspiderSpider(CrawlSpider):

name = 'yySpider'

allowed\_domains = ['www.yyetss.com']

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

# 匹配的正则规则

link\_home = LinkExtractor(allow=("detail-\d+\.html"))

rules = [

Rule(link\_home, callback="parse\_content", follow=False,

***process\_links***="deal\_links", ***process\_request***="deal\_request")

]

# CrawlSpider不能重写parse方法

# def parse(self, response):

def deal\_links(self, links):

def deal\_request(self, request):

def parse\_content(self, response):

print(response)

### 3、scrapy.spider.Rule()

**process\_links** 处理LinkExtractor中匹配的连接地址

LinkExtractor匹配a标签->deal\_link处理->callback指定的函数

def deal\_links(self, links):

for link in links:

print(link.url)

return links #必须将links返回

**process\_request过滤请求**

linkExtractor提取的链接会接着返回调度器，进入下载队列进行待下载，process\_request就是为了在这个时刻把请求过滤

def deal\_request(self, request):

print(request) #request是一个请求

#必须返回

return request

### xpath

#request自带xpath类，不用引入

def parse\_content(self, request):

# 名字 .extract将xpath对象转换为字符串

name = request.xpath("//ul[@class='pan']/li/a").extract()

name = name.replace(r"\xa0", "") # 将不可识别的字符替换

## 管道

### spider/myspider传数据到pipeline

/spiders/myspider.py

from mySpider.item import myItem

def parse(self, repose):

item = myItem() #实例化对象

......

item[‘aa’] = ‘value’

#如果是请求，引擎传给调度器下载数据

#请求放到最后

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

#如果是数据，引擎传给item，

yield item

# 基本不用,返回给程序

return item

/setting.py

ITEM\_PIPELINES = {

#项目路径，管道文件名，类名

‘mySpider.pipelines.myPipeline: 300,

}

/pippelines.py

class myPipeline（object）:

def \_\_init\_\_(self):#可选

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

#item就是数据

#默认编写的类

#中文不转ascii编码

jsontext = json.dumps(dict(item), ensure\_ascii = false)

return jsontext #必须要返回

def close\_spider(self, spider)

#关闭方法，可选

### pipeline-->pipline中传值

/setting.py

ITEM\_PIPELINES = {

#项目路径，管道文件名，类名

#设置多个管道文件，并填写调用的前后顺序

‘mySpider.pipelines.myPipeline: 300, #先

‘mySpider.pipelines.myTwoPipeline: 301, #其次

}

/pipeline.py

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

return jsontext #这里是将数据返回到item类中，方便其他管道文件调用

/其他

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

#通过item获取将上一个管道的数据

return jsontext #这里是将数据返回到item类中，方便其他管道文件调用

## 下载图片

python/site-packages/scrapy/piplines中有专门处理图片、多媒体的文件

必要条件：setting.py中设置IMAGES\_STORE

import time

import scrapy

import re

from scrapy.utils.project import get\_project\_settings # 导入setting文件from scrapy.pipelines.images import ImagesPipeline

class ImgPipeline(ImagesPipeline):

IMAGE\_SOURCE = get\_project\_settings().get("IMAGES\_STORE")

# def \_\_init\_\_(self, \*args, \*\*kwargs):

# super().\_\_init\_\_(\*args, \*\*kwargs)

def get\_media\_requests(self, item, info): # 发送请求

# item就是items.py中定义的变量集合

# 这里可以修改申请下载的文件

yield scrapy.Request(item['image'])

def item\_completed(self, result, item, info): # 处理图片

image\_path = [x['path'] for ok, x in result if ok]

a = os.path.splitext(image\_path[0])

time\_name = str(round(time.time() \* 1000)) + '.' + a[1]

# 重保存位置

os.rename(self.IMAGE\_SOURCE + '/' +

image\_path[0], self.IMAGE\_SOURCE + '/' + time\_name)

# ['full/c9579622aeb4e01f62c74bd57bbde4485647331c.jpg']

# scrapy会自动下载图片并给一个嘻哈值 image\_path中保持

#[(True, {'url': 'http://aikanmeiju.b0.upaiyun.com/images\_all/p2536907410.jpg!thumb', 'path': 'full/0e0b75df307ec0049c238c5068b6e065e4f61612.jpg', 'checksum': '7a2de211906c62a1a60623b7e7ebbf37'})]

return item

## 数据返回

yield

return

## 登陆三种策略

1. 拿cookie登陆
2. 只提供登陆数据
3. 正规模拟登陆

## 抓取数据保存到数据库

使用正常连接数据库方式

pymongo--pymysql

## 下载中间件

（1）setting中添加代理、user-agent

/setting.py

#将相关信息写入setting，或设置常数，

#设置PROXIES项

PROXIEX =[{“ip”:“127.0.0.1”,”user”:’name:pass’}]

USER\_AGENTS = [

"Mozilla/5.0 (compatible; MSIE 9.0; Windows N”,

]

#设置下载中间件

DOWNLOADER\_MIDDLEWARES = {

'mySpider.middlewares.RandomUserAgent': 1,

'mySpider.middlewares.ProxyMiddleware': 100

}

1. 编写下载中间件/middlewares.py

设置请求头

from setting import USER

from scrapy.utils.project import get\_project\_settings # 导入setting文件

class RandomUserAgent(object):

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

pass

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

# 必须实现，所有下载请求都通过我

# 设置请求头

User = get\_project\_setting().get(‘User’)

request.headers.setdefault("User-Agent", User)

设置代理

from setting import PROXY

class RandomProxy(object):

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

pass

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

proxy = random.choice(PROXY)

# 经过base64转换的密码，HTTP要求

request.headers["Proxy-Authorization"] = "Basic " + base\_userpasswd

#设置密码转换，必须用base64编码

base64\_userpasswd = base64.b64encode(proxy['user\_passwd'])

#设置请求IP与端口

request.meta['proxy'] = "http://" + proxy[‘ipAndPort’]

# scrapy-redis

redis注释掉所有bind daemonize yes 后台

#去重组件

DUPEFILTER\_CLASS “scrapy-redis.dupefilter.RFPDupeFilter”

#使用scrapy-redis组件

SCHEDULER = ‘scrapy\_redis.scheduler.Scheduler’

SCHEDULER\_PERSIST = True #允许暂停

#默认请求集合

SCHEDULER\_QUEUE\_CLASS = “scrapy\_redis.queue.SpiderPriorityQueue”

ITEM\_PIPELINES中要使用scrapy\_redis.pipelines.Redispipeline

REDIS\_HOST = “” #指定数据库的ip位置

REDIS\_PORT = “”#指定数据库的port

# 错误：

## 1、使用代理服务器乱码

**原因:**

代理服务器请求数据时使用了其他的编码格式，本机程序编码不同，

解决方法：使用.encode(“UTF8”)转码

## Could not find ...

Could not find a version that satisfies the requirement **Twisted**>=13.1.0 (from Scrapy) (from versions: )

No matching distribution found for Twisted>=13.1.0 (from Scrapy)

**原因**：没有安装Twisted

解决方法：

（1）下载Twisted官网下载tar.bz2包

（2）解压，没有bzip要先安装yum -y install bzip2

（3）python setup.py install（进入Twisted目录）