# 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 startproject xxx #新建scrapy项目

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

scrapy crawl mySpider #根目录运行，执行指定的爬虫

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

# 启动爬虫

scrapy crawl 爬虫名称 #根目录运行，执行指定的爬虫

# 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

## 使用setting文件

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

get\_porject\_setting().get(“xxx”)

#通过配置pipline读取setting

class MysqlTispipline(object):

# scrapy会自动调用from\_setting并把配置文件传入

@classmethod

def from\_setting(cls, setting):

host = setting[‘host’]

return cls

# 开始使用

## 创建默认模板

**命令：**scrapy genspider mySpider “www.baidu.com”

**说明：**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

1. **导入库**

import scrapy

from scrapy.http import Request

1. **配置类参数**

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, request):

pass

1. **填写parse方法**

def parse(self, request):

**“””**

**1\获取文章中的url地址**

**2\获取下一页的url并交给scrapy进行下载，下载完成后交给parse**

**“””**

# response.css(“.entry-header h1::text”).extract() css选择器

# TODO：解析文本，获取文章url

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

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

for post\_url in list:

# TODO： 将解析的下一页url交给scrapy进行下载

# yield 会自动将url交给scrapy进行下载

**yield Request(url=post\_url, callback=self.parse\_detail)**

**#解析下一页的url**

next\_url= response.xpath(“//div[@class=”lit”])

**yield Request(url=next\_url, callback=self.parse)**

**# 将数据传给pipline**

**# 如果是一个request将发送到指定方法**

**yield item**

**def parse\_detail(self, request):**

# 解析文章内容

....

## 使用CrawlSpider模板创建

### 1、基本使用方法

命令：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/'] （1）

# 匹配的正则规则

link\_home = LinkExtractor(allow=("detail-\d+\.html")) （2）

rules = [

Rule(

link\_home, #解析正则

callback="parse\_content", #符合当前规则要调用的解析方法

follow=False,

***process\_links***="deal\_links", #过滤匹配到的数据通过deal\_links方法

***process\_request***="deal\_request" # 过滤发送的url，通过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)

### 2、Rules方法-Crawl模板

**scrapy.spider.Rule(process\_links, process\_request)**

**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

return request #必须返回

# 使用xpath解析

#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

# 保存数据

## 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 myItem(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()

## 使用Item类型

***/spider/myspider.py***

from mySpider.item import myItem # 从item导入相关类

def parse(self, reponse):

item = myItem() #实例化对象

item[‘aa’] = ‘value’

yield item # 会将参数传递到pipeline中

return item

## 4、通过pymysql进行保存

***/pipline.py***

import pymysql

class MysqlPipline(object):

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

self.conn = pymysql.connect(...)

self.curser = self.conn.curser

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

inser\_sql = “inser into user value (‘1’)”

## 5、设置

/setting.py

ITEM\_PIPELINES = {

#项目路径，管道文件名，类名

‘mySpider.pipelines.myPipeline: 300,

}

1. 根据

def do\_insert(self, cursor, item):

        #执行具体的插入

        #根据不同的item 构建不同的sql语句并插入到mysql中

        insert\_sql, params = item.get\_insert\_sql()

        print (insert\_sql, params)

        cursor.execute(insert\_sql, params)

# pipline接收参数

## 爬虫spder.py文件向pipline文件中传值

***/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

图片文件中会用到itme\_completed方法

***/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

# 模拟登陆

当一个网站需要登录时，应重写start\_request(self)方法

class Zhihu(scrapy.Spider):

def pares():pass

# 主要完成登录事件

def start\_requests(self):

# TODO：使用request.get获取登录页信息，获取\_xsrf数据

return [scrapy.FormRequest(

url = “http://baidu.com”,

formadata = {

“\_xsrf”: \_xsrf

},

headers = headers #设置请求头

callback=self.check\_login

)]

#验证是否登录成功

def check\_login(self, request):

text.json = json.loads(request.text)

if “msg” in text and text\_json[‘msg’] == “登录成功”:

# 发送信息

for url in self.start\_urls:

yied scrapy.Request(url, dont\_filter=Tree, headers=headers)

# 下载图片

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

**pip install pillow 必须安装这个库**

## 设置

/setting.py

ITEM\_PIPELINES = {

#项目路径，管道文件名，类名

‘scrapy.pipelines.images.ImagesPipeline’ : 1,

}

IMAGES\_URLS\_FILED = “itme中定义图片url的字段名” # parse中字段需要写成数组

import os

images\_path = os.path.join(os.path.abspath(os.path.dirname(\_\_file\_\_), “images”)

IMAGES\_STORE = “images\_path” #图片保存的路径

## 自定义图片下载功能

1. **设置**

ITEM\_PIPELINES = {

#项目路径，管道文件名，类名

‘scrapy.pipelines.MyImagesPipeline’ : 1,

}

1. **新建文件MyImagesPipline**

**与pipline.py同级目录**

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, results, item, info):

#当一个单独项目中的所有图片请求完成时（要么完成下载，要么因为某种原因下载失败）， [FilesPipeline.item\_completed()](https://scrapy-chs.readthedocs.io/zh_CN/1.0/topics/media-pipeline.html" \l "scrapy.pipeline.files.FilesPipeline.item_completed" \o "scrapy.pipeline.files.FilesPipeline.item_completed) 方法将被调用。

# 方法一------------------

for ok, value in results:

image\_file\_path = value[“path”]

item[‘front\_image\_path’] = image\_file\_path

# 方法二-----------------

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

# 重命名

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

# 使用ItemLoader

from scrapy.loader import ItemLoader

def parse(self, reponse):

item\_load = ItemLoader(item=JobBoleArt(),response=)

# 使用下载中间件

## 配置

再setting.py中填写相关配置项，关键**开启中间件**

#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’}] #设置代理

USER\_AGENTS = [ #配置请求头

"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

}

## 设置请求头

编写下载中间件/middlewares.py

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

编写下载中间件/middlewares.py

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使用selenium

**from scrapy.http import HtmlResponse**

**# /sprder/xx.py**

class JobSpider(scrapy.Spider):

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

#防止每次都启动chrome

self.browser = webdriver.Chrome(....)

super(JobSpider, self).\_\_init\_\_()

**# 配置信号,调用spider\_closed方法**

dispatcher.connect(self.spider\_closed, signals.spider\_closed)

**def spider\_closed(self, spider):**

self.browser.quit()

**# /middleware.py**

class JsPageMiddleware(object):

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

if spider.name == “jobblie”: #指定哪些需要使用

# TODO:使用无界面的selenium方法

spider.browser.get(request.url) #代替系统发送http请求

# scrapy会自动将HtmlResponse对象返回sprider

return **HtmlResponse**(

url=spider.browser.current\_url,

body=spider.browser.page\_source,

encoding=”utf-8”, # 编码需要统一

request=request)

# 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

# pipline生成json文件

**配置通道文件**

## 自定义pipline

import json

import codecs # 解决编码问题的一个库

class JsonWithEncodingPipeline(object):

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

# art.json文件名，位置在爬虫根目录

self.file = codecs.open(“art.json”, “w”, encode=”utf-8”)

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

line=json.dump(item) # 转json文件

self.file.write(line)

return item

def spider\_closed(self, spider):

self.file.close() #关闭codecs

## 使用scrapy中类生成json

class JsonExporterPipleline(object):

    #调用scrapy提供的json export导出json文件

    def \_\_init\_\_(self):

        self.file = open('articleexport.json', 'wb')

        self.exporter = JsonItemExporter(

self.file, encoding="utf-8", ensure\_ascii=False

)

        self.exporter.start\_exporting()

    def close\_spider(self, spider):

        self.exporter.finish\_exporting()

        self.file.close()

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

        self.exporter.export\_item(item)

        return item

# 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）