

Day09回顾

日志级别

```
1 | DEBUG < INFO < WARNING < ERROR < CRITICAL
```

数据持久化存储(MySQL、 MongoDB)

```
1 | 1、 settings.py : 定义相关变量
2 | 2、 pipelines.py : 新建类
3 | 3、 settings.py : 添加管道
4 |
5 | # 注意 : process_item() 函数中一定要 return item
```

保存为csv、 json文件

■ 命令格式

```
1 | scrapy crawl maoyan -o maoyan.csv
2 | scrapy crawl maoyan -o maoyan.json
3 | # settings.py FEED_EXPORT_ENCODING = 'utf-8'
```

settings.py常用变量

```
1 | # 1、设置日志级别
2 | LOG_LEVEL = ''
3 | # 2、保存到日志文件(不在终端输出)
4 | LOG_FILE = ''
5 | # 3、设置数据导出编码(主要针对于json文件)
6 | FEED_EXPORT_ENCODING = ''
7 | # 4、非结构化数据存储路径
8 | IMAGES_STORE = '路径'
9 | # 5、设置User-Agent
10 | USER_AGENT = ''
```

```

11 # 6、设置最大并发数(默认为16)
12 CONCURRENT_REQUESTS = 32
13 # 7、下载延迟时间(每隔多长时间请求一个网页)
14 # DOWNLOAD_DELAY 会影响 CONCURRENT_REQUESTS, 不能使并发显现
15 # 有CONCURRENT_REQUESTS, 没有DOWNLOAD_DELAY: 服务器会在同一时间收到大量的请求
16 # 有CONCURRENT_REQUESTS, 有DOWNLOAD_DELAY 时, 服务器不会在同一时间收到大量的请求
17 DOWNLOAD_DELAY = 3
18 # 8、请求头
19 DEFAULT_REQUEST_HEADERS = {}
20 # 9、添加项目管道
21 ITEM_PIPELINES = {}
22 # 10、添加下载器中间件
23 DOWNLOADER_MIDDLEWARES = {}

```

非结构化数据抓取

```

1 1、spider: yield item['链接']
2 2、pipelines.py
3 3、settings.py

```

scrapy.Request()参数

```

1 1、url
2 2、callback
3 3、headers
4 4、meta : 传递数据,定义代理
5 5、dont_filter : 是否忽略域组限制
6 默认False,检查allowed_domains['']

```

设置中间件

整体步骤

```

1 # middlewares.py
2 class xxx(object):
3     def process_request(self,request,spider):
4         pass

```

随机User-Agent

```

1 class RandomUaDownloaderMiddleware(object):
2     def process_request(self,request,spider):
3         request.header['User-Agent'] = xxx

```

随机代理

```
1 class RandomProxyDownloaderMiddleware(object):
2     def process_request(self, request, spider):
3         request.meta['proxy'] = xxx
4
5     def process_exception(self, request, exception, spider):
6         return request
```

Day10笔记

分布式爬虫

分布式爬虫介绍

■ 原理

```
1 多台主机共享1个爬取队列
```

■ 实现

```
1 重写scrapy调度器(scrapy_redis模块)
```

■ 为什么使用redis

```
1 1、Redis基于内存,速度快
2 2、Redis非关系型数据库,Redis中集合,存储每个request的指纹
3 3、scrapy_redis安装
4     sudo pip3 install scrapy_redis
```

Redis使用

■ windows安装客户端使用

```
1 1、服务端启动 : cmd命令行 -> redis-server.exe
2 客户端连接 : cmd命令行 -> redis-cli.exe
```

scrapy_redis

■ GitHub地址

```
1 | https://github.com/rmax/scrapy-redis
```

■ settings.py说明

```
1 | # 重新指定调度器：启用Redis调度存储请求队列
2 | SCHEDULER = "scrapy_redis.scheduler.Scheduler"
3 |
4 | # 重新指定去重机制：确保所有的爬虫通过Redis去重
5 | DUPEFILTER_CLASS = "scrapy_redis.dupefilter.RFPDupeFilter"
6 |
7 | # 不清除Redis队列：暂停/恢复/断点续爬
8 | SCHEDULER_PERSIST = True
9 |
10 | # 优先级队列（默认）
11 | SCHEDULER_QUEUE_CLASS = 'scrapy_redis.queue.PriorityQueue'
12 | #可选用的其它队列
13 | # 先进先出队列
14 | SCHEDULER_QUEUE_CLASS = 'scrapy_redis.queue.FifoQueue'
15 | # 后进先出队列
16 | SCHEDULER_QUEUE_CLASS = 'scrapy_redis.queue.LifoQueue'
17 |
18 | # redis管道
19 | ITEM_PIPELINES = {
20 |     'scrapy_redis.pipelines.RedisPipeline': 300
21 | }
22 |
23 |
24 | #指定连接到redis时使用的端口和地址
25 | REDIS_HOST = 'localhost'
26 | REDIS_PORT = 6379
```

腾讯招聘笔记分布式案例

正常项目数据抓取（非分布式）

MySQL数据库--建库建表

```
1 | create database tencentdb charset utf8;
2 | use tencentdb;
3 | create table tencenttab(
4 | name varchar(100),
5 | type varchar(100),
6 | duty varchar(5000),
7 | requirement varchar(5000)
8 | )charset=utf8;
```

■ items.py

```

1 import scrapy
2
3
4 class TencentItem(scrapy.Item):
5     # define the fields for your item here like:
6     zh_name = scrapy.Field()
7     zh_type = scrapy.Field()
8     zh_duty = scrapy.Field()
9     zh_require = scrapy.Field()

```

■ tencent.py

```

1 # -*- coding: utf-8 -*-
2 import scrapy
3 import json
4 from ..items import TencentItem
5
6 class TencentSpider(scrapy.Spider):
7     name = 'tencent'
8     allowed_domains = ['tencent.com']
9
10
11     def start_requests(self):
12         for page_index in range(1,51):
13             url = 'https://careers.tencent.com/tencentcareer/api/post/Query?timestamp=1557114143837&countryId=&keyword=python&cityId=&bgIds=&productId=&categoryId=&parentCategoryId=&attrId=&keyword=&pageIndex=%s&pageSize=10&language=zh-cn&area=cn' % str(page_index)
14             yield scrapy.Request(
15                 url=url,
16                 callback=self.parse_one_page
17             )
18     # 一级页面解析
19     def parse_one_page(self, response):
20         html = json.loads(response.text)
21
22         for h in html['Data']['Posts']:
23             item = TencentItem()
24
25             item['zh_name'] = h['RecruitPostName']
26             item['zh_type'] = h['LocationName']
27             # 一级页面获取PostId,详情页URL需要此参数
28             post_id = h['PostId']
29             # 想办法获取到职位要求和职责,F12抓包,抓到地址
30             two_url = 'https://careers.tencent.com/tencentcareer/api/post/ByPostId?timestamp=1557122746678&postId=%s&language=zh-cn' % post_id
31
32             yield scrapy.Request(
33                 url=two_url,
34                 meta={'item':item},
35                 callback=self.parse_two_page
36             )
37     def parse_two_page(self, response):
38         item = response.meta['item']
39         html = json.loads(response.text)

```

```

40         # 职责
41         item['zh_duty'] = html['Data']['Responsibility']
42         # 要求
43         item['zh_require'] = html['Data']['Requirement']
44
45         yield item

```

■ pipelines.py

```

1  class TencentPipeline(object):
2      def process_item(self, item, spider):
3          print(dict(item))
4          return item
5
6
7  import pymysql
8  class TencentMysqlPipeline(object):
9      def open_spider(self, spider):
10         self.db =
pymysql.connect('192.168.153.134', 'tiger', '123456', 'tencentdb', charset='utf8')
11         self.cursor = self.db.cursor()
12
13     def process_item(self, item, spider):
14         ins = 'insert into tencenttab values(%s,%s,%s,%s)'
15         job_list = [
16             item['zh_name'], item['zh_type'], item['zh_duty'], item['zh_require']
17         ]
18         self.cursor.execute(ins, job_list)
19         self.db.commit()
20         return item
21
22     def close_spider(self, spider):
23         self.cursor.close()
24         self.db.close()

```

■ settings.py

```

1  USER_AGENT = 'Mozilla/5.0'
2  ROBOTSTXT_OBEY = False
3  ITEM_PIPELINES = {
4      'Tencent.pipelines.TencentPipeline': 300,
5      'Tencent.pipelines.TencentMysqlPipeline': 200,
6  }

```

改写为分布式（同时存入redis）

1. settings.py

```

1  # 使用scrapy_redis的调度器
2  SCHEDULER = "scrapy_redis.scheduler.Scheduler"
3  # 使用scrapy_redis的去重机制
4  DUPEFILTER_CLASS = "scrapy_redis.dupefilter.RFPDupeFilter"
5  # 爬取完成后是否清除请求指纹,True:不清除 False:清除
6  SCHEDULER_PERSIST = False
7  # 在ITEM_PIPELINES中添加redis管道
8  'scrapy_redis.pipelines.RedisPipeline': 200
9  # 定义redis主机地址和端口号
10 REDIS_HOST = '111.111.111.111'
11 REDIS_PORT = 6379

```

改写为分布式（同时存入mysql）

- 修改管道

```

1  ITEM_PIPELINES = {
2      'Tencent.pipelines.TencentPipeline': 300,
3      # 'scrapy_redis.pipelines.RedisPipeline': 200
4      'Tencent.pipelines.TencentMysqlPipeline': 200,
5  }

```

- 清除redis数据库

```

1  flushdb

```

- 代码拷贝一份到分布式中其他机器，两台或多台机器同时执行此代码

机器视觉与tesseract

作用

```

1  处理图形验证码

```

三个重要概念

- OCR

```

1  # 定义
2  OCR: 光学字符识别(Optical Character Recognition)
3  # 原理
4  通过扫描等光学输入方式将各种票据、报刊、书籍、文稿及其它印刷品的文字转化为图像信息，再利用文字识别技术将图像信息转化为电子文本

```

- tesseract-ocr

```
1 | OCR的一个底层识别库（不是模块，不能导入）
2 | # Google维护的开源OCR识别库
```

■ pytesseract

```
1 | Python模块,可调用底层识别库
2 | # 对tesseract-ocr做的一层Python API封装
```

安装tesseract-ocr

■ Ubuntu

```
1 | sudo apt-get install tesseract-ocr
```

■ Windows

```
1 | 1、下载安装包
2 | 2、添加到环境变量(Path)
```

■ 测试

```
1 | # 终端 | cmd命令行
2 | tesseract xxx.jpg 文件名
```

安装pytesseract

■ 安装

```
1 | sudo pip3 install pytesseract
```

■ 使用

```
1 | import pytesseract
2 | # Python图片处理标准库
3 | from PIL import Image
4 |
5 | # 创建图片对象
6 | img = Image.open('test1.jpg')
7 | # 图片转字符串
8 | result = pytesseract.image_to_string(img)
9 | print(result)
```

■ 爬取网站思路（验证码）

- 1 1、获取验证码图片
- 2 2、使用PIL库打开图片
- 3 3、使用pytesseract将图片中验证码识别并转为字符串
- 4 4、将字符串发送到验证码框中或者某个URL地址

在线打码平台

■ 为什么使用在线打码

- 1 `tesseract-ocr`识别率很低,文字变形、干扰,导致无法识别验证码

■ 云打码平台使用步骤

- 1 1、下载并查看接口文档
- 2 2、调整接口文档,调整代码并接入程序测试
- 3 3、真正接入程序,在线识别后获取结果并使用

■ 破解云打码网站验证码

1. 下载并调整接口文档,封装成函数,打码获取结果

```
1 def get_result(filename):
2     # 用户名
3     username = 'yibeizi001'
4
5     # 密码
6     password = 'zhanshen002'
7
8     # 软件ID, 开发者分成必要参数。登录开发者后台【我的软件】获得!
9     appid = 1
10
11    # 软件密钥, 开发者分成必要参数。登录开发者后台【我的软件】获得!
12    appkey = '22cc5376925e9387a23cf797cb9ba745'
13
14    # 图片文件
15    # filename = 'getimage.jpg'
16
17    # 验证码类型, # 例: 1004表示4位字母数字, 不同类型收费不同。请准确填写, 否则影响识别率。在此查询所有类型 http://www.yundama.com/price.html
18    codetype = 5000
19
20    # 超时时间, 秒
21    timeout = 60
22
23    # 初始化
24    yundama = YDMHttp(username, password, appid, appkey)
25
26    # 登陆云打码
27    uid = yundama.login();
28
29    # 查询余额
```

```

30     balance = yundama.balance();
31
32     # 开始识别, 图片路径, 验证码类型ID, 超时时间 (秒), 识别结果
33     cid, result = yundama.decode(filename, codetype, timeout);
34
35     return result
36
37     #####

```

2. 访问云打码网站, 获取验证码并在线识别

```

1  from selenium import webdriver
2  from ydmapi import *
3  # 处理图片
4  from PIL import Image
5
6  # options = webdriver.ChromeOptions()
7  # options.add_argument('windows-size=1900x3000')
8  browser = webdriver.Chrome()
9
10
11 # 获取网站首页截图
12 def get_screen_shot():
13     browser.get('http://www.yundama.com')
14     browser.save_screenshot('index.png')
15
16 # 从首页截图中截取验证码图片
17 def get_caphe():
18     # 定位验证码元素的位置(x y坐标)
19     location = browser.find_element_by_xpath(
20         '//*[@id="verifyImg"]'
21     ).location
22     # 大小(宽度和高度)
23     size = browser.find_element_by_xpath(
24         '//*[@id="verifyImg"]'
25     ).size
26     # 左上角x坐标
27     left = location['x']
28     # 左上角y坐标
29     top = location['y']
30     # 右下角x坐标
31     right = location['x'] + size['width']
32     # 右下角y坐标
33     bottom = location['y'] + size['height']
34
35     # 截图验证码图片(crop()):对图片进行剪切,参数为元组
36     img = Image.open('index.png').crop((left,top,right,bottom))
37     # 保存截取后的图片
38     img.save('yzm.png')
39
40     # 调用在线打码平台进行识别
41     result = get_result('yzm.png')
42
43     return result
44
45 if __name__ == '__main__':

```

```
46 | get_screen_shot()
47 | result = get_caphe()
48 | print('识别结果为:',result)
```

Fiddler抓包工具

■ 配置Fiddler

```
1 | # 添加证书信任
2 | 1、Tools - Options - HTTPS
3 |   勾选 Decrypt Https Traffic 后弹出窗口，一路确认
4 | # 设置只抓取浏览器的数据包
5 | 2、...from browsers only
6 | # 设置监听端口（默认为8888）
7 | 3、Tools - Options - Connections
8 | # 配置完成后重启Fiddler（重要）
9 | 4、关闭Fiddler,再打开Fiddler
```

■ 配置浏览器代理

```
1 | 1、安装Proxy SwitchyOmega插件
2 | 2、浏览器右上角：SwitchyOmega->选项->新建情景模式->AID1901(名字)->创建
3 |   输入：HTTP:// 127.0.0.1 8888
4 |   点击：应用选项
5 | 3、点击右上角SwitchyOmega可切换代理
```

■ Fiddler常用菜单

```
1 | 1、Inspector：查看数据包详细内容
2 |   整体分为请求和响应两部分
3 | 2、常用菜单
4 |   Headers：请求头信息
5 |   WebForms：POST请求Form表单数据：<body>
6 |   GET请求查询参数：<QueryString>
7 |   Raw
8 |   将整个请求显示为纯文本
```

移动端app数据抓取

-----让我来告诉你-----