

Day10回顾

settings.py常用变量

```
1 # 1、设置日志级别
2
3 # 2、保存到日志文件(不在终端输出)
4
5 # 3、设置数据导出编码(主要针对于json文件)
6
7 # 4、非结构化数据存储路径
8
9 # 5、设置User-Agent
10
11 # 6、设置最大并发数(默认为16)
12
13 # 7、下载延迟时间(每隔多长时间请求一个网页)
14
15 # 8、请求头
16
17 # 9、添加项目管道
18
19 # 10、添加下载器中间件
20
```

非结构化数据抓取

```
1 1、spider
2     yield item['链接']
3 2、pipelines.py
4     from scrapy.pipelines.images import ImagesPipeline
5     import scrapy
6     class TestPipeline(ImagesPipeline):
7         def get_media_requests(self,item,info):
8             yield scrapy.Request(url=item['url'],meta={'item':item['name']})
9         def file_path(self,request,response=None,info=None):
10             name = request.meta['item']
11             filename = name
12             return filename
13 3、settings.py
14     IMAGES_STORE = 'D:\\Spider\\images'
```

scrapy.Request()

```
1 # 参数
2 1、url
3 2、callback
4 3、headers
5 4、meta : 传递数据,定义代理
6 5、dont_filter : 是否忽略域组限制 - 默认False,检查allowed_domains['']
7 6、cookies
8 # request属性
9 1、request.url
10 2、request.headers
11 3、request.meta
12 4、request.method
13 # response属性
14 1、response.url
15 2、response.text
16 3、response.body
17 4、response.meta
18 5、response.encoding
```

设置中间件

随机User-Agent

```
1 # 1、middlewares.py - headers属性
2 class RandomUaDownloaderMiddleware(object):
3
4 # 2、settings.py
5 DOWNLOADER_MIDDLEWARES = {'xxx.middlewares.xxx':300}
```

随机代理

```
1 # 1、middlewares.py - meta属性
2 class RandomProxyDownloaderMiddleware(object):
3
4
5 # 2、settings.py
6 DOWNLOADER_MIDDLEWARES = {'xxx.middlewares.xxx':200}
```

Day11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
```

scrapy_redis详解

■ 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 | #指定连接到redis时使用的端口和地址
24 | REDIS_HOST = 'localhost'
25 | REDIS_PORT = 6379
```

腾讯招聘分布式改写

1、正常项目数据抓取（非分布式）

2、改写为分布式（同时存入redis）

1、settings.py

```
1 # 1、使用scrapy_redis的调度器
2 SCHEDULER = "scrapy_redis.scheduler.Scheduler"
3 # 2、使用scrapy_redis的去重机制
4 DUPEFILTER_CLASS = "scrapy_redis.dupefilter.RFPDupeFilter"
5 # 3、是否清除请求指纹,True:不清除 False:清除（默认）
6 SCHEDULER_PERSIST = True
7 # 4、（非必须）在ITEM_PIPELINES中添加redis管道
8 'scrapy_redis.pipelines.RedisPipeline': 200
9 # 5、定义redis主机地址和端口号
10 REDIS_HOST = '127.0.0.1'
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

腾讯招聘分布式改写- 方法二

■ 使用redis_key改写

```
1 # 第一步：settings.py无须改动
2 settings.py和上面分布式代码一致
3 # 第二步:tencent.py
```

```

4 from scrapy_redis.spiders import RedisSpider
5 class TencentSpider(RedisSpider):
6     # 1. 去掉start_urls
7     # 2. 定义redis_key
8     redis_key = 'tencent:spider'
9     def parse(self, response):
10         pass
11 # 第三步:把代码复制到所有爬虫服务器, 并启动项目
12 # 第四步
13 到redis命令行, 执行LPUSH命令压入第一个要爬取的URL地址
14 >LPUSH tencent:spider 第1页的URL地址
15
16 # 项目爬取结束后无法退出, 如何退出?
17 setting.py
18 CLOSESPIDER_TIMEOUT = 3600
19 # 到指定时间(3600秒)时, 会自动结束并退出

```

scrapy - post请求

▪ 方法+参数

```

1 scrapy.FormRequest(
2     url=posturl,
3     formdata=formdata,
4     callback=self.parse
5 )

```

▪ 有道翻译案例实现

1、创建项目+爬虫文件

```
1 |
```

2、items.py

```
1 |
```

3、youdao.py

```
1 |
```

4、settings.py

```
1 |
```

scrapy添加cookie的三种方式

```

1 # 1、修改 settings.py 文件
2 1、COOKIES_ENABLED = False # 取消注释，表示启用cookie,使用变量中Cookie值
3 2、DEFAULT_REQUEST_HEADERS = {} 添加Cookie
4
5 # 2、爬虫文件 - 利用cookies参数
6 COOKIES_ENABLED = True # 启用Cookie,使用Request()方法中cookies参数
7 def start_requests(self):
8     yield scrapy.Request(url=url,cookies={},callback=xxx)
9
10 # 3、DownloadMiddleware设置中间件
11 COOKIES_ENABLED = True # 启用Cookie,使用Request()方法中cookies参数
12 def process_request(self,request,spider):
13     request.cookies={}

```

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 |
```

▪ 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 |
```

2、访问云打码网站，获取验证码并在线识别

```
1 |
```

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

方法1 - 手机 + Fiddler

方法2 - F12浏览器工具

有道翻译手机版破解案例

爬虫总结

- 1 # 1、什么是爬虫
- 2 爬虫是请求网站并提取数据的自动化程序
- 3
- 4 # 2、robots协议是什么
- 5 爬虫协议或机器人协议,网站通过robots协议告诉搜索引擎哪些页面可以抓取, 哪些页面不能抓取
- 6
- 7 # 3、爬虫的基本流程
- 8 1、请求得到响应
- 9 2、解析
- 10 3、保存数据
- 11
- 12 # 4、请求
- 13 1、urllib
- 14 2、requests
- 15 3、scrapy
- 16
- 17 # 5、解析
- 18 1、re正则表达式
- 19 2、lxml+xpath解析
- 20 3、json解析模块
- 21
- 22 # 6、selenium+browser
- 23
- 24 # 7、常见反爬策略
- 25 1、Headers : 最基本的反爬手段, 一般被关注的变量是UserAgent和Referer, 可以考虑使用浏览器中
- 26 2、UA : 建立User-Agent池,每次访问页面随机切换
- 27 3、拉黑高频访问IP
- 28 数据量大用代理IP池伪装成多个访问者,也可控制爬取速度
- 29 4、Cookies
- 30 建立有效的cookie池, 每次访问随机切换
- 31 5、验证码
- 32 验证码数量较少可人工填写
- 33 图形验证码可使用tesseract识别
- 34 其他情况只能在线打码、人工打码和训练机器学习模型
- 35 6、动态生成
- 36 一般由js动态生成的数据都是向特定的地址发get请求得到的, 返回的一般是json
- 37 7、签名及js加密
- 38 一般为本地JS加密, 查找本地JS文件, 分析, 或者使用execjs模块执行JS
- 39 8、js调整页面结构
- 40 9、js在响应中指向新的地址
- 41
- 42 # 8、scrapy框架的运行机制

43

44 # 9、分布式爬虫的原理

45 多台主机共享一个爬取队列