

# Day07回顾

## 创建项目流程

```
1 1、 scrapy startproject Tencent
2 2、 cd Tencent
3 3、 scrapy genspider tencent tencent.com
4 4、 items.py(定义爬取数据结构)
5 5、 tencent.py (写爬虫文件)
6 6、 pipelines.py(数据处理)
7 7、 settings.py(全局配置)
8 8、 终端: scrapy crawl tencent
```

## 响应对象属性及方法

```
1 # 属性
2 1、 response.text : 获取响应内容
3 2、 response.body : 获取bytes数据类型
4 3、 response.xpath('')
5
6 # response.xpath('')调用方法
7 1、 结果 : 列表,元素为选择器对象
8 2、 .extract() : 提取文本内容,将列表中所有元素序列化为Unicode字符串
9 3、 .extract_first() : 提取列表中第1个文本内容
10 4、 .get() : 提取列表中第1个文本内容
```

## 爬虫项目启动方式

### ■ 方式一

```
1 从爬虫文件(spider)的start_urls变量中遍历URL地址,把下载器返回的响应对象(response)交给爬虫文件的
  parse()函数处理
2 # start_urls = ['http://www.baidu.com/', 'http://www.sina.com.cn']
```

### ■ 方式二

```
1 重写start_requests()方法，从此方法中获取URL，交给指定的callback解析函数处理
2
3 1、# 去掉start_urls变量
4 2、def start_requests(self):
5     # 生成要爬取的URL地址，利用scrapy.Request()方法交给调度器 **
```

## 数据传递

```
1 1、数据交给管道：yield item
2 2、URL交给调度器：yield scrapy.Request(url,callback=解析函数名)
3 3、爬虫文件传递item到下一个解析函数
4     yield scrapy.Request(
5         url,
6         meta={'item':item},
7         callback=解析函数名
8     )
```

## settings.py常用变量

```
1 1、LOG_LEVEL = ''
2 2、LOG_FILE = ''
3 3、FEED_EXPORT_ENCODING = ''
4 4、IMAGES_STORE = '路径'
```

## 日志级别

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

## 数据持久化存储

```
1 1、settings.py定义常用变量
2 2、pipelines.py自定义管道类
3     from .settings import *
4
5     class TencentMongoPipeline(object):
6         def open_spider(self,spider):
7             # 打开数据库连接
8
9         def process_item(self,item,spider):
10             # item数据处理，必须return item
11
12         def close_spider(self,spider):
```

```
13         # 收尾工作, 一般用于断开数据库连接
14
15 3、settings.py添加管道
16     ITEM_PIPELINES = {'Tencent.pipelines.TencentxxxPipeline':1}
```

## 保存为csv、json文件

```
1 1、 scrapy crawl tencnet -o xxx.json | xxx.csv
2 2、 设置到处编码
3     settings.py : FEED_EXPORT_ENCODING='utf-8'
```

# Day08笔记

## 图片管道(360图片抓取)

### ■ 目标

```
1 www.so.com -> 图片 -> 美女
2 # http://image.so.com/z?ch=beauty
```

### ■ F12抓包

```
1 # 数据1: json地址
2     url = 'http://image.so.com/zj?ch=beauty&sn={}&listtype=new&temp=1'
3
4 # 数据2: 查询参数 (QueryString)
5     ch: beauty
6     sn: 90
7     listtype: new
8     temp: 1
```

### ■ 项目实施

#### 1. items.py定义抓取的数据结构

```
1 # 图片链接
2 img_link = scrapy.Field()
```

## 2. so.py解析响应数据

```
1 class SoSpider(scrapy.Spider):
2     name = 'so'
3     allowed_domains = ['image.so.com']
4
5     # 重写Spider类中的start_requests方法
6     # 爬虫程序启动时执行此方法,不去找start_urls
7     def start_requests(self):
8         for page in range(5):
9             url = 'http://image.so.com/zj?ch=beauty&sn=
10 {}&listtype=new&temp=1'.format(str(page*30))
11             # 把url地址入队列
12             yield scrapy.Request(
13                 url = url,
14                 callback = self.parse_img
15             )
16
17     def parse_img(self, response):
18         html = json.loads(response.text)
19
20         for img in html['list']:
21             item = SoItem()
22             # 图片链接
23             item['img_link'] = img['qhimg_url']
24
25             yield item
```

## 3. pipelines.py数据处理

```
1 from scrapy.pipelines.images import ImagesPipeline
2 import scrapy
3
4 class SoPipeline(ImagesPipeline):
5     # 重写get_media_requests方法
6     def get_media_requests(self, item, info):
7         yield scrapy.Request(item['img_link'])
```

## 4. settings.py全局配置

```
1 # 定义存储图片的路径
2 IMAGES_STORE = 'D:\\\\王伟超'
```

## 5. begin.py运行爬虫

```
1 from scrapy import cmdline
2
3 cmdline.execute('scrapy crawl so'.split())
```

# scrapy shell

- 基本使用

```
1 1、 scrapy shell URL地址
2 *2、 request.headers : 请求头(字典)
3 *3、 request.meta      : item数据传递, 定义代理(字典)
4 4、 response.text      : 字符串
5 5、 response.body      : bytes
6 6、 response.xpath('')
```

- scrapy.Request()

```
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(随机User-Agent)

### *少量User-Agent切换*

- 方法一

```
1 # settings.py
2 USER_AGENT = ''
3 DEFAULT_REQUEST_HEADERS = {}
```

- 方法二

```
1 # spider
2 yield scrapy.Request(url,callback=函数名,headers={})
```

### *大量User-Agent切换 (中间件)*

- middlewares.py设置中间件

```

1 1、获取User-Agent
2   # 方法1：新建useragents.py,存放大量User-Agent, random模块随机切换
3   # 方法2：安装fake_useragent模块(sudo pip3 install fake_useragent)
4       from fake_useragent import UserAgent
5       ua_obj = UserAgent()
6       ua = ua_obj.random
7 2、middlewares.py新建中间件类
8       class RandomUseragentMiddleware(object):
9           def process_request(self, request, spider):
10               ua = UserAgent()
11               request.headers['User-Agent'] = ua.random
12 3、settings.py添加此下载器中间件
13     DOWNLOADER_MIDDLEWARES = {'': 优先级}

```

## 设置中间件(随机代理)

```

1 request.meta['proxy'] = 'http://127.0.0.1:8888'
2 ** 使用代理尝试 **

```

## 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
5

```

### ■ 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 #####
2 def get_ydm(filename):
3     # 用户名
4     username = 'yibeizi001'
5     # 密码
6     password = 'zhanshen001'
7     # 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!
8     appid = 1
9     # 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!
10    appkey = '22cc5376925e9387a23cf797cb9ba745'
11    # 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
12    codetype = 5000
13    # 超时时间, 秒
14    timeout = 60
15    # 初始化
16    yundama = YDMHttp(username, password, appid, appkey)
17    # 登陆云打码
18    uid = yundama.login();
19    # print('uid: %s' % uid)
20    # 查询余额
21    balance = yundama.balance();
22    # print('balance: %s' % balance)
23    # 开始识别, 图片路径, 验证码类型ID, 超时时间(秒), 识别结果
24    cid, result = yundama.decode(filename, codetype, timeout)
25    return result
26 #####
```

#### 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 PIL import Image
3 from ydmapi import *
4
5 browser = webdriver.Chrome()
6
7 # 获取网站首页截图(包含验证码)
8 def get_screen_shot():
9     browser.get('http://www.yundama.com/')
10    browser.save_screenshot('screenshot.png')
11
12 # 从首页截图中截取验证码图片
13 def get_image():
14     # 定位验证码元素的位置及大小
15     location = browser.find_element_by_xpath('//*[@id="verifyImg"]').location
16     size = browser.find_element_by_xpath('//*[@id="verifyImg"]').size
17     # 获取图片具体位置(左上角和右下角的坐标)
18     # 左上角x坐标
19     left = location['x']
20     # 左上角y坐标
21     top = location['y']
22     # 右下角x坐标
23     right = location['x'] + size['width']
24     # 右下角的y坐标
25     bottom = location['y'] + size['height']
26
27     # crop函数: 对图片进行剪切,参数为元组
28     img = Image.open('screenshot.png').crop((left,top,right,bottom))
29     # 保存剪切后的图片
30     img.save('capche.png')
31     result = get_capche('capche.png')
32
33     return result
34
35 if __name__ == '__main__':
36     get_screen_shot()
37     result = get_image()
38     print('结果为:',result)

```

## 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

```
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
```

- Ubuntu安装redis

```
1 # 安装
2 sudo apt-get install redis-server
3 # 启动
4 redis-server
5 # 连接
6 redis-cli -h IP地址
```

## *腾讯招聘笔记分布式案例*

### 正常项目数据抓取（非分布式）

```
1 首先将项目以非分布式方式完成
```

### 改写为分布式（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 # 在ITEM_PIPELINES中添加redis管道
6 'scrapy_redis.pipelines.RedisPipeline': 200
7 # 定义redis主机地址和端口号
8 REDIS_HOST = '172.40.91.129'
9 REDIS_PORT = 6379
```

## 改写为分布式（mongodb）

- 修改管道

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

- 清除redis数据库

```
1 flushall
```

- 代码拷贝一份到Ubuntu（分布式中其他机器）,两台机器同时执行此代码

## 移动端数据抓取

见笔记 文件夹资料