Day08回顾

scrapy框架

- 五大组件

```
引擎（Engine）
爬虫程序（Spider）
调度器（Scheduler）
下载器（Downloader）
管道文件（Pipeline）
# 两个中间件
下载器中间件（Downloader Middlewares）
蜘蛛中间件（Spider Middlewares）
```

- 工作流程

```
1、Engine向Spider索要URL,交给Scheduler入队列
2、Scheduler处理后出队列,通过Downloader Middlewares交给Downloader去下载
3、Downloader得到响应后,通过Spider Middlewares交给Spider
4、Spider数据提取：
    1、数据交给Pipeline处理
    2、需要跟进URL,继续交给Scheduler入队列，依次循环
```

- 常用命令

```
# 创建爬虫项目
scrapy startproject 项目名

# 创建爬虫文件
cd 项目文件夹
scrapy genspider 爬虫名 域名

# 运行爬虫
scrapy crawl 爬虫名
```

- scrapy项目目录结构

```
Baidu
├── Baidu               # 项目目录
│   ├── items.py        # 定义数据结构
│   ├── middlewares.py  # 中间件
│   ├── pipelines.py    # 数据处理
│   ├── settings.py     # 全局配置
│   └── spiders
│       ├── baidu.py    # 爬虫文件
└── scrapy.cfg          # 项目基本配置文件
```

- settings.py全局配置

```
1、USER_AGENT = 'Mozilla/5.0'
2、ROBOTSTXT_OBEY = False
3、CONCURRENT_REQUESTS = 32
4、DOWNLOAD_DELAY = 1
5、DEFAULT_REQUEST_HEADERS={}
6、ITEM_PIPELINES={'项目目录名.pipelines.类名':300}
```

创建项目流程

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

响应对象属性及方法

```
# 属性
1、response.text ：获取响应内容
2、response.body ：获取bytes数据类型
3、response.xpath('')

# response.xpath('')调用方法
1、结果 ：列表,元素为选择器对象
2、.extract() ：提取文本内容,将列表中所有元素序列化为Unicode字符串
3、.extract_first() ：提取列表中第1个文本内容
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**

```
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
# start_urls = ['http://www.baidu.com/']
```

- **方式二**

```
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

1、去掉start_urls变量
2、def start_requests(self):
      # 生成要爬取的URL地址，利用scrapy.Request()方法交给调度器 **
```

日志级别

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

数据持久化存储(MySQL、MongoDB)

```
1、在setting.py中定义相关变量
2、pipelines.py中新建管道类，并导入settings模块
    def open_spider(self,spider):
        # 爬虫开始执行1次,用于数据库连接
    def process_item(self,item,spider):
        # 用于处理抓取的item数据
    def close_spider(self,spider):
        # 爬虫结束时执行1次,用于断开数据库连接
3、settings.py中添加此管道
    ITEM_PIPELINES = {'':200}

# 注意 ： process_item() 函数中一定要 return item ***
```

保存为csv、json文件

- 命令格式

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

settings.py常用变量

```
# 1、设置日志级别
LOG_LEVEL = ''
# 2、保存到日志文件(不在终端输出)
LOG_FILE = ''
# 3、设置数据导出编码(主要针对于json文件)
FEED_EXPORT_ENCODING = ''
# 4、非结构化数据存储路径
IMAGES_STORE = '路径'
# 5、设置User-Agent
USER_AGENT = ''
# 6、设置最大并发数(默认为16)
CONCURRENT_REQUESTS = 32
# 7、下载延迟时间(每隔多长时间请求一个网页)
# DOWNLOAD_DELAY 会影响 CONCURRENT_REQUESTS, 不能使并发显现
# 有CONCURRENT_REQUESTS, 没有DOWNLOAD_DELAY:  服务器会在同一时间收到大量的请求
# 有CONCURRENT_REQUESTS, 有DOWNLOAD_DELAY 时, 服务器不会在同一时间收到大量的请求
DOWNLOAD_DELAY = 3
# 8、请求头
DEFAULT_REQUEST_HEADERS = {}
# 9、添加项目管道
ITEM_PIPELINES = {}
# 10、添加下载器中间件
DOWNLOADER_MIDDLEWARES = {}
```

scrapy.Request()参数

```
1、url
2、callback
3、meta : 传递数据,定义代理
```

Day09笔记

作业讲解 - 腾讯招聘

- **1、创建项目+爬虫文件**

```
scrapy startproject Tencent
cd Tencent
scrapy genspider tencent hr.tencent.com
```

- **2、定义爬取的数据结构**

```
# items.py
job_name = scrapy.Field()
# 类别
job_type = scrapy.Field()
# 职责
job_duty = scrapy.Field()
# 要求
job_require = scrapy.Field()
# 地址
job_address = scrapy.Field()
```

- **3、爬虫文件**

```
class TencentSpider(scrapy.Spider):
    name = 'tencent'
    allowed_domains = ['careers.tencent.com']
    one_url = 'https://careers.tencent.com/tencentcareer/api/post/Query?timestamp=1563912271089&countryId=&cityId=&bgIds=&productId=&categoryId=&parentCategoryId=&attrId=&keyword=&pageIndex={}&pageSize=10&language=zh-cn&area=cn'
    two_url = 'https://careers.tencent.com/tencentcareer/api/post/ByPostId?timestamp=1563912374645&postId={}&language=zh-cn'
    # 1. 去掉start_urls
    # 2. 重新start_requests()方法
    def start_requests(self):
        total_page = self.get_total_page()
        for page_index in range(1,total_page):
            url = self.one_url.format(page_index)
            yield scrapy.Request(
                url = url,
                callback = self.parse_one
            )

    # 获取总页数
    def get_total_page(self):
        url = self.one_url.format(1)
        html = requests.get(url=url).json()
        total_page = int(html['Data']['Count']) // 10 + 1

        return total_page

    # 解析一级页面函数
    def parse_one(self,response):
        html = json.loads(response.text)
        for job in html['Data']['Posts']:
            item = TencentItem()
            # postId: 拼接二级页面的地址
            post_id = job['PostId']
            two_url = self.two_url.format(post_id)
            # 交给调度器
            yield scrapy.Request(
                url = two_url,
                meta = {'item':item},
                callback = self.parse_two_page
```

```
            )

    def parse_two_page(self,response):
        item = response.meta['item']
        html = json.loads(response.text)
        item['job_name'] = html['Data']['RecruitPostName']
        item['job_type'] = html['Data']['CategoryName']
        item['job_duty'] = html['Data']['Responsibility']
        item['job_require'] = html['Data']['Responsibility']
        item['job_address'] = html['Data']['LocationName']


        yield item
```

- **4、管道文件**

```
create database tencentdb charset utf8;
use tencentdb;
create table tencenttab(
job_name varchar(500),
job_type varchar(100),
job_duty varchar(1000),
job_require varchar(1000),
job_address varchar(100)
)charset=utf8;
```

管道文件pipelines实现

```
import pymysql
class TencentMysqlPipeline(object):
    def open_spider(self,spider):
        self.db = pymysql.connect(
            '127.0.0.1','root','123456','tencentdb',
            charset='utf8'
        )
        self.cursor = self.db.cursor()

    def process_item(self,item,spider):
        ins = 'insert into tencenttab values(%s,%s,%s,%s,%s)'
        job_list = [
            item['job_name'],item['job_type'],item['job_duty'],
            item['job_require'],item['job_address']
        ]
        self.cursor.execute(ins,job_list)
        self.db.commit()
        return item

    def close_spider(self,spider):
        self.cursor.close()
        self.db.close()
```

- **5、settings.py**

```
定义常用变量，添加管道即可
```

图片管道(360图片抓取案例)

- **目标**

```
www.so.com -> 图片 -> 美女
```

- **抓取网络数据包**

```
2、F12抓包,抓取到json地址 和 查询参数(QueryString)
      url = 'http://image.so.com/zj?ch=beauty&sn={}&listtype=new&temp=1'.format(str(sn))
      ch: beauty
      sn: 90
      listtype: new
      temp: 1
```

- **项目实现**

1、创建爬虫项目和爬虫文件

```
scrapy startproject So
cd So
scrapy genspider so image.so.com
```

2、定义要爬取的数据结构(items.py)

```
img_link = scrapy.Field()
```

3、爬虫文件实现图片链接抓取

```
# -*- coding: utf-8 -*-
import scrapy
import json
from ..items import SoItem

class SoSpider(scrapy.Spider):
    name = 'so'
    allowed_domains = ['image.so.com']

    # 重写Spider类中的start_requests方法
    # 爬虫程序启动时执行此方法,不去找start_urls
    def start_requests(self):
        for page in range(5):
            url = 'http://image.so.com/zj?ch=beauty&sn=
{}&listtype=new&temp=1'.format(str(page*30))
            # 把url地址入队列
            yield scrapy.Request(
                url = url,
                callback = self.parse_img
            )

    def parse_img(self, response):
        html = json.loads(response.text)
```

```
        for img in html['list']:
            item = SoItem()
            # 图片链接
            item['img_link'] = img['qhimg_url']

            yield item
```

4、管道文件（pipelines.py）

```
from scrapy.pipelines.images import ImagesPipeline
import scrapy

class SoPipeline(ImagesPipeline):
    # 重写get_media_requests方法
    def get_media_requests(self, item, info):
        yield scrapy.Request(item['img_link'])
```

5、设置settings.py

```
IMAGES_STORE = '/home/tarena/images/'
```

6、创建run.py运行爬虫

scrapy shell的使用

- **基本使用**

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

- **scrapy.Request()**

```
1、url
2、callback
3、headers
4、meta ：传递数据,定义代理
5、dont_filter ：是否忽略域组限制
   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**

```
# settings.py
USER_AGENT = ''
DEFAULT_REQUEST_HEADERS = {}
```

- **方法二**

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

设置中间件(随机代理)

```
rclass RandomProxyDownloaderMiddleware(object):
    def process_request(self,request,spider):
        request.meta['proxy'] = xxx

    def process_exception(self,request,exception,spider):
        return request
```

分布式爬虫

分布式爬虫介绍

- **原理**

```
多台主机共享1个爬取队列
```

- **实现**

```
重写scrapy调度器(scrapy_redis模块)
```

- **为什么使用redis**

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

Redis使用

- **windows安装客户端使用**

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

scrapy_redis

- **GitHub地址**

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

- **settings.py说明**

```
# 重新指定调度器: 启用Redis调度存储请求队列
SCHEDULER = "scrapy_redis.scheduler.Scheduler"

# 重新指定去重机制: 确保所有的爬虫通过Redis去重
DUPEFILTER_CLASS = "scrapy_redis.dupefilter.RFPDupeFilter"

# 不清除Redis队列: 暂停/恢复/断点续爬
SCHEDULER_PERSIST = True

```

```
# 优先级队列 （默认）
SCHEDULER_QUEUE_CLASS = 'scrapy_redis.queue.PriorityQueue'
#可选用的其它队列
# 先进先出队列
SCHEDULER_QUEUE_CLASS = 'scrapy_redis.queue.FifoQueue'
# 后进先出队列
SCHEDULER_QUEUE_CLASS = 'scrapy_redis.queue.LifoQueue'

# redis管道
ITEM_PIPELINES = {
    'scrapy_redis.pipelines.RedisPipeline': 300
}


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

腾讯招聘笔记分布式案例