DAY10

Day09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)
   import scrapy
   class TencentItem(scrapy.Item):
       job_name = scrapy.Field()
5、tencent.py (写爬虫文件)
   import scrapy
   class TencentSpider(scarpy.Spider):
      name = 'tencent'
      allowed_domains = ['tencent.com']
      start_urls = ['http://tencent.com/']
      def parse(self,response):
        xxx
        yield item
6、pipelines.py(数据处理)
   class TencentPipeline(object):
      def process_item(self,item,spider):
          return item
7、settings.py(全局配置)
  LOG_LEVEL = ''
  LOG_FILE = ''
  FEED_EXPORT_ENCODING = ''
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 - 字符串
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
  # <selector xpath='//article' data=''>
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
```

Day10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

# 一级页面(postId):
https://careers.tencent.com/tencentcareer/api/post/Query?timestamp=1566266592644&countryId=&cityId=&bgIds=&productId=&categoryId=&parentCategoryId=&attrId=&keyword=&pageIndex={}&pageSize=10&language=zh-cn&area=cn

# 二级页面
https://careers.tencent.com/tencentcareer/api/post/ByPostId?timestamp=1566266695175&postId={}&language=zh-cn
```

- **2、定义爬取的数据结构**

```
# 名称+类别+职责+要求+地址+时间
job_name = scrapy.Field()
job_type = scrapy.Field()
job_duty = scrapy.Field()
job_require = scrapy.Field()
job_address = scrapy.Field()
job_time = scrapy.Field()
```

- **3、爬虫文件**

```python
import scrapy
import json
from ..items import TencentItem
from urllib import parse
import requests

class TencentSpider(scrapy.Spider):
    name = 'tencent'
    allowed_domains = ['careers.tencent.com']
    one_url = 'https://careers.tencent.com/tencentcareer/api/post/Query?timestamp=1566266592644&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=1566266695175&postId={}&language=zh-cn'
    user_input = input('请输入工作类型:')

    # 重写start_requests()方法,把一级页面所有地址交给调度器
    def start_requests(self):
        # 给user_input进行编码
        user_input = parse.quote(self.user_input)
        # 获取到总页数:total
        total = self.get_total(user_input)
        for index in range(1,total):
            url = self.one_url.format(user_input,index)
            yield scrapy.Request(
                url = url,
                callback = self.parse_one_page
            )
    # 获取总页数
    def get_total(self,user_input):
        url = self.one_url.format(user_input,1)
        html = requests.get(url=url).json()
        total = html['Data']['Count'] // 10 + 1

        return total

    def parse_one_page(self, response):
        html = response.text
        html = json.loads(html)
        for job in html['Data']['Posts']:

            post_id = job['PostId']
            url = self.two_url.format(post_id)
            yield scrapy.Request(
                url = url,
                callback = self.parse_two_page
            )

    # 解析二级页面
    def parse_two_page(self,response):
        item = TencentItem()
        html = json.loads(response.text)['Data']
        item['job_name'] = html['RecruitPostName']
        item['job_type'] = html['CategoryName']
        item['job_duty'] = html['Responsibility']
        item['job_require'] = html['Requirement']
        item['job_address'] = html['LocationName']
        item['job_time'] = html['LastUpdateTime']

        yield item
```

- **4、管道文件**

```sql
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),
job_time varchar(100)
)charset=utf8;
```

管道文件pipelines实现

```python
class TencentPipeline(object):
    def process_item(self, item, spider):
        print(dict(item))
        return item

import pymysql

class TencentMysqlPipeline(object):
    def open_spider(self,spider):
        self.db = pymysql.connect(
          'localhost','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,%s)'
        L = [
            item['job_name'],
            item['job_type'],
            item['job_duty'],
            item['job_require'],
            item['job_address'],
            item['job_time']
        ]
        self.cursor.execute(ins,L)
        self.db.commit()

        return item

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

- **5、settings.py**

```python
# 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/zjl?ch=beauty&sn={}&listtype=new&temp=1'.format(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()
img_title = scrapy.Field()
```

3、爬虫文件实现图片链接+名字抓取

```python
import scrapy
import json
from ..items import SoItem

class SoSpider(scrapy.Spider):
    name = 'so'
    allowed_domains = ['image.so.com']
    url = 'http://image.so.com/zjl?ch=beauty&sn={}&listtype=new&temp=1'

    def start_requests(self):
        for sn in range(0,100,30):
            url = self.url.format(sn)
            yield scrapy.Request(
                url = url,
                callback = self.parse_page,
                dont_filter=False
            )

    def parse_page(self, response):
        html = json.loads(response.text)
        item = SoItem()
        for img in html['list']:
            item['img_url'] = img['qhimg_url']
            item['img_title'] = img['title']

            yield item
```

4、管道文件（pipelines.py）

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

class SoPipeline(ImagesPipeline):
    # 重写 get_media_requests()方法
    def get_media_requests(self,item,info):
        yield scrapy.Request(
          url = item['img_url'],
          meta = {'item':item['img_title']}
        )

    # 重写 file_path()方法
    def file_path(self,request,response=None,info=None):
        title = request.meta['item']
        filename = title+'.'+request.url.split('.')[-1]
        return filename
```

5、设置settings.py

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

6、创建run.py运行爬虫

字符串方法总结

```
1、strip()
2、split()
3、replace('','')
4、''.join()
5、字符串切片(正向切,反向切) : S[-10:]
```

scrapy shell的使用

- **基本使用**

```
# scrapy shell URL地址
*1、request.url     : 请求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()参数**

```
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)

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

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

Fiddler抓包工具

- **配置Fiddler**

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

- **配置浏览器代理**

```
1、安装Proxy SwitchyOmega插件
2、浏览器右上角：SwitchyOmega->选项->新建情景模式->AID1904(名字)->创建
   输入 ：HTTP://  127.0.0.1  8888
   点击 ：应用选项
3、点击右上角SwitchyOmega可切换代理
```

- **Fiddler常用菜单**

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