Day08回顾

selenium切换句柄

1、适用网站

```
页面中点开链接出现新的页面，但是浏览器对象browser还是之前页面的对象
```

2、应对方案

```
# 获取当前所有句柄（窗口）
all_handles = browser.window_handles
# 切换到新的窗口
browser.switch_to_window(all_handles[1])
```

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/','http://www.sina.com.cn']
```

- 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 **
```

Day09笔记

日志变量及日志级别(settings.py)

```
# 日志相关变量
LOG_LEVEL = ''
LOG_FILE = '文件名.log'

# 日志级别
5 CRITICAL ： 严重错误
4 ERROR    ： 普通错误
3 WARNING  ： 警告
2 INFO     ： 一般信息
1 DEBUG    ： 调试信息
# 注意：只显示当前级别的日志和比当前级别日志更严重的
```

数据持久化存储(MySQL)

实现步骤

```
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 ***
```

练习

把猫眼电影数据存储到MySQL数据库中

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'
```

盗墓笔记小说抓取案例（三级页面）

- 目标

```
# 抓取目标网站中盗墓笔记1-8中所有章节的所有小说的具体内容，保存到本地文件
1、网址 ：http://www.daomubiji.com/
```

- 准备工作xpath

```
1、一级页面xpath（此处响应做了处理）：//ul[@class="sub-menu"]/li/a/@href
2、二级页面xpath：/html/body/section/div[2]/div/article
  基准xpath ：//article
3、三级页面xpath：response.xpath('//article[@class="article-content"]//p/text()').extract()
```

- 项目实现

1. **创建项目及爬虫文件**

```
创建项目 ： Daomu
创建爬虫 ： daomu  www.daomubiji.com
```

2. 定义要爬取的数据结构（把数据交给管道）

```python
import scrapy

class DaomuItem(scrapy.Item):
    # 卷名
    volume_name = scrapy.Field()
    # 章节数
    zh_num = scrapy.Field()
    # 章节名
    zh_name = scrapy.Field()
    # 章节链接
    zh_link = scrapy.Field()
    # 小说内容
    zh_content = scrapy.Field()
```

3. **爬虫文件实现数据抓取**

```python
# -*- coding: utf-8 -*-
import scrapy
from ..items import DaomuItem

class DaomuSpider(scrapy.Spider):
    name = 'daomu'
    allowed_domains = ['www.daomubiji.com']
    start_urls = ['http://www.daomubiji.com/']

    # 解析一级页面,提取 盗墓笔记1 2 3 ... 链接
    def parse(self, response):
        one_link_list = response.xpath('//ul[@class="sub-menu"]/li/a/@href').extract()
        print(one_link_list)
        # 把链接交给调度器入队列
        for one_link in one_link_list:
            yield
scrapy.Request(url=one_link,callback=self.parse_two_link,dont_filter=True)

    # 解析二级页面
    def parse_two_link(self,response):
        # 基准xpath,匹配所有章节对象列表
        article_list = response.xpath('/html/body/section/div[2]/div/article')
        # 依次获取每个章节信息
        for article in article_list:
            # 创建item对象
            item = DaomuItem()
            info = article.xpath('./a/text()').extract_first().split()
            # info : ['七星鲁王','第一章','血尸']
            item['volume_name'] = info[0]
            item['zh_num'] = info[1]
```

```
            item['zh_name'] = info[2]
            item['zh_link'] = article.xpath('./a/@href').extract_first()
            # 把章节链接交给调度器
            yield scrapy.Request(
                url=item['zh_link'],
                # 把item传递到下一个解析函数
                meta={'item':item},
                callback=self.parse_three_link,
                dont_filter=True
            )

    # 解析三级页面
    def parse_three_link(self,response):
        item = response.meta['item']
        # 获取小说内容
        item['zh_content'] = '\n'.join(response.xpath(
          '//article[@class="article-content"]//p/text()'
        ).extract())

        yield item

        # '\n'.join(['第一段','第二段','第三段'])
```

4. **管道文件实现数据处理**

```
# -*- coding: utf-8 -*-

# Define your item pipelines here
#
# Don't forget to add your pipeline to the ITEM_PIPELINES setting
# See: https://doc.scrapy.org/en/latest/topics/item-pipeline.html


class DaomuPipeline(object):
    def process_item(self, item, spider):
        filename = '/home/tarena/aid1902/{}-{}-{}.txt'.format(
            item['volume_name'],
            item['zh_num'],
            item['zh_name']
        )

        f = open(filename,'w')
        f.write(item['zh_content'])
        f.close()
        return item
```

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']
```

```
28
29              yield item
```

4. **管道文件（pipelines.py）**

```python
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**

```
1
```

6. **创建run.py运行爬虫**

```
1
```

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)

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