Day07笔记

selenium - 鼠标操作

```
from selenium import webdriver
# 导入鼠标事件类
from selenium.webdriver import ActionChains

driver = webdriver.Chrome()
driver.get('http://www.baidu.com/')

# 移动到 设置，perform()是真正执行操作，必须有
element = driver.find_element_by_xpath('//*[@id="u1"]/a[8]')
ActionChains(driver).move_to_element(element).perform()

# 单击，弹出的Ajax元素，根据链接节点的文本内容查找
driver.find_element_by_link_text('高级搜索').click()
```

selenium - iframe

- **特点+方法**

```
【1】特点
    网页中嵌套了网页，先切换到iframe，然后再执行其他操作

【2】处理步骤
    2.1) 切换到要处理的Frame
    2.2) 在Frame中定位页面元素并进行操作
    2.3) 返回当前处理的Frame的上一级页面或主页面

【3】常用方法
    3.1) 切换到frame   -   driver.switch_to.frame(frame节点对象)
    3.2) 返回上一级    -   driver.switch_to.parent_frame()
    3.3) 返回主页面    -   driver.switch_to.default_content()

【4】使用说明
    4.1) 方法一：默认支持id和name属性值 ： switch_to.frame(id属性值|name属性值)
    4.2) 方法二：
        a> 先找到frame节点 ： frame_node = driver.find_element_by_xpath('xxxx')
        b> 在切换到frame    : driver.switch_to.frame(frame_node)
```

- **示例 - 登录豆瓣网**

```
"""
登录豆瓣网
"""
from selenium import webdriver
import time

# 打开豆瓣官网
driver = webdriver.Chrome()
driver.get('https://www.douban.com/')

# 切换到iframe子页面
login_frame = driver.find_element_by_xpath('//*[@id="anony-reg-new"]/div/div[1]/iframe')
driver.switch_to.frame(login_frame)

# 密码登录 + 用户名 + 密码 + 登录豆瓣
driver.find_element_by_xpath('/html/body/div[1]/div[1]/ul[1]/li[2]').click()
driver.find_element_by_xpath('//*[@id="username"]').send_keys('自己的用户名')
driver.find_element_by_xpath('//*[@id="password"]').send_keys('自己的密码')
driver.find_element_by_xpath('/html/body/div[1]/div[2]/div[1]/div[5]/a').click()
time.sleep(3)

# 点击我的豆瓣
driver.find_element_by_xpath('//*[@id="db-nav-sns"]/div/div/div[3]/ul/li[2]/a').click()
```

selenium+phantomjs|chrome|firefox总结

```
【1】特点
    1.1》简单，无需去详细抓取分析网络数据包，使用真实浏览器
    1.2》需要等待页面元素加载，需要时间，效率低

【2】设置无界面模式
    options = webdriver.ChromeOptions()
    options.add_argument('--headless')
    driver = webdriver.Chrome(excutable_path='/home/tarena/chromedriver',options=options)

【3】鼠标操作
    from selenium.webdriver import ActionChains
    ActionChains(driver).move_to_element('node').perform()

【4】切换句柄 - switch_to.frame(handle)
    all_handles = driver.window_handles
    driver.switch_to.window(all_handles[1])

【5】iframe子页面
    driver.switch_to.frame(frame_node)

【6】driver执行JS脚本
    driver.execute_script('window.scrollTo(0,document.body.scrollHeight)')

【7】lxml中的xpath 和 selenium中的xpath的区别
    7.1》lxml中的xpath用法 - 推荐自己手写
        div_list = p.xpath('//div[@class="abc"]/div')
```

```
        item = {}
        for div in div_list:
            item['name'] = div.xpath('.//a/@href')[0]
            item['likes'] = div.xpath('.//a/text()')[0]

    7.2》selenium中的xpath用法 - 推荐copy - copy xpath
        div_list = driver.find_elements_by_xpath('//div[@class="abc"]/div')
        item = {}
        for div in div_list:
            item['name'] = div.find_element_by_xpath('.//a').get_attribute('href')
            item['likes'] = div.find_element_by_xpath('.//a').text
```

作业概解

作业1 - 有道翻译实现

- **代码实现**

```
"""
selenium实现抓取有道翻译结果
思路:
    1、找到输入翻译单词节点,发送文字
    2、休眠一定时间,等待网站给出响应-翻译结果
    3、找到翻译结果节点,获取文本内容
"""
from selenium import webdriver
import time

class YdSpider:
    def __init__(self):
        self.url = 'http://fanyi.youdao.com/'
        # 设置无界面模式
        self.options = webdriver.ChromeOptions()
        self.options.add_argument('--headless')
        self.driver = webdriver.Chrome(options=self.options)
        # 打开有道翻译官网
        self.driver.get(self.url)

    def parse_html(self, word):
        # 发送翻译单词
        self.driver.find_element_by_id('inputOriginal').send_keys(word)
        time.sleep(1)
        # 获取翻译结果
        result = self.driver.find_element_by_xpath('//*
[@id="transTarget"]/p/span').text

        return result

    def run(self):
        word = input('请输入要翻译的单词:')
```

```
        print(self.parse_html(word))
        self.driver.quit()

if __name__ == '__main__':
    spider = YdSpider()
    spider.run()
```

作业2- 163邮箱登陆

- **代码实现**

```
"""
selenium模拟登录163邮箱
思路:
    1、密码登录在这里 - 此节点在主页面中,并非iframe内部
    2、切换iframe - 此处iframe节点中id的值每次都在变化,需要手写xpath,否则会出现无法定位iframe
    3、输入用户名和密码
    4、点击登录按钮
"""
from selenium import webdriver

driver = webdriver.Chrome()
driver.get('https://mail.163.com/')

# 1、切换iframe子页面 - 此处手写xpath,此处iframe中id的值每次都在变化
node = driver.find_element_by_xpath('//div[@id="loginDiv"]/iframe[1]')
driver.switch_to.frame(node)

# 2、输入用户名和密码
driver.find_element_by_name('email').send_keys('wangweichao_2020')
driver.find_element_by_name('password').send_keys('zhanshen001')
driver.find_element_by_id('dologin').click()
```

scrapy框架

- **定义**

```
异步处理框架,可配置和可扩展程度非常高,Python中使用最广泛的爬虫框架
```

- **安装**

```
【1】Ubuntu安装
    sudo pip3 install Scrapy

【2】Windows安装
    python -m pip install Scrapy

    如果安装过程中报如下错误 : 'Error: Microsoft Vistual C++ 14.0 is required xxx'
    则安装Windows下的Microsoft Vistual C++ 14.0 即可（笔记spiderfiles中有）
```

- **Scrapy框架五大组件**

```
【1】引擎（Engine）----------整个框架核心
【2】爬虫程序（Spider）------数据解析提取
【3】调度器（Scheduler）-----维护请求队列
【4】下载器（Downloader）----获取响应对象
【5】管道文件（Pipeline）-----数据入库处理


【两个中间件】
    下载器中间件（Downloader Middlewares）
        引擎->下载器,包装请求(随机代理等)
    蜘蛛中间件（Spider Middlewares）
        引擎->爬虫文件,可修改响应对象属性
```

- **scrapy爬虫工作流程**

```
【1】爬虫项目启动,由引擎向爬虫程序索要第一批要爬取的URL,交给调度器去入队列
【2】调度器处理请求后出队列,通过下载器中间件交给下载器去下载
【3】下载器得到响应对象后,通过蜘蛛中间件交给爬虫程序
【4】爬虫程序进行数据提取:
    4.1) 数据交给管道文件去入库处理
    4.2) 对于需要继续跟进的URL,再次交给调度器入队列，依次循环
```

- **scrapy常用命令**

```
【1】创建爬虫项目 : scrapy startproject 项目名
【2】创建爬虫文件
    2.1) cd 项目文件夹
    2.2) scrapy genspider 爬虫名 域名
【3】运行爬虫
    scrapy crawl 爬虫名
```

- **scrapy项目目录结构**

```
Baidu                   # 项目文件夹
├── 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常用变量**

```
【1】USER_AGENT = 'Mozilla/5.0'
【2】ROBOTSTXT_OBEY = False
    是否遵循robots协议,一般我们一定要设置为False
【3】CONCURRENT_REQUESTS = 32
    最大并发量,默认为16
【4】DOWNLOAD_DELAY = 0.5
    下载延迟时间：访问相邻页面的间隔时间,降低数据抓取的频率
【5】COOKIES_ENABLED = False | True
    Cookie默认是禁用的，取消注释则 启用Cookie，即：True和False都是启用Cookie
【6】DEFAULT_REQUEST_HEADERS = {}
    请求头,相当于requests.get(headers=headers)
```

- **创建爬虫项目步骤**

```
【1】新建项目和爬虫文件
    scrapy startproject 项目名
    cd 项目文件夹
    新建爬虫文件 ： scrapy genspider 文件名 域名
【2】明确目标(items.py)
【3】写爬虫程序(文件名.py)
【4】管道文件(pipelines.py)
【5】全局配置(settings.py)
【6】运行爬虫
    8.1) 终端：scrapy crawl 爬虫名
    8.2) pycharm运行
        a> 创建run.py(和scrapy.cfg文件同目录)
          from scrapy import cmdline
          cmdline.execute('scrapy crawl maoyan'.split())
        b> 直接运行 run.py 即可
```

瓜子二手车直卖网 - 一级页面

- **目标**

```
【1】抓取瓜子二手车官网二手车收据（我要买车）

【2】URL地址：https://www.guazi.com/bj/buy/o{}/#bread
   URL规律：o1  o2  o3  o4  o5  ... ...

【3】所抓数据
    3.1) 汽车链接
    3.2) 汽车名称
    3.3) 汽车价格
```

实现步骤

- **步骤1 - 创建项目和爬虫文件**

```
scrapy startproject Car
cd Car
scrapy genspider car www.guazi.com
```

- **步骤2 - 定义要爬取的数据结构**

```
"""items.py"""
import scrapy

class CarItem(scrapy.Item):
    # 链接、名称、价格
    url = scrapy.Field()
    name = scrapy.Field()
    price = scrapy.Field()
```

- **步骤3 - 编写爬虫文件（代码实现1）**

```
"""
此方法其实还是一页一页抓取，效率并没有提升，和单线程一样

xpath表达式如下：
【1】基准xpath,匹配所有汽车节点对象列表
    li_list = response.xpath('//ul[@class="carlist clearfix js-top"]/li')

【2】遍历后每辆车信息的xpath表达式
    汽车链接：'./a[1]/@href'
    汽车名称：'.//h2[@class="t"]/text()'
    汽车价格：'.//div[@class="t-price"]/p/text()'
"""
# -*- coding: utf-8 -*-
import scrapy
from ..items import CarItem


class GuaziSpider(scrapy.Spider):
    # 爬虫名
    name = 'car'
    # 允许爬取的域名
    allowed_domains = ['www.guazi.com']
    # 初始的URL地址
    start_urls = ['https://www.guazi.com/bj/buy/o1/#bread']
    # 生成URL地址的变量
    n = 1

    def parse(self, response):
        # 基准xpath：匹配所有汽车的节点对象列表
        li_list = response.xpath('//ul[@class="carlist clearfix js-top"]/li')
        # 给items.py中的 GuaziItem类 实例化
        item = CarItem()
        for li in li_list:
            item['url'] = li.xpath('./a[1]/@href').get()
            item['name'] = li.xpath('./a[1]/@title').get()
            item['price'] = li.xpath('.//div[@class="t-price"]/p/text()').get()
```

```
            # 把抓取的数据,传递给了管道文件 pipelines.py
            yield item

        # 1页数据抓取完成,生成下一页的URL地址,交给调度器入队列
        if self.n < 5:
            self.n += 1
            url = 'https://www.guazi.com/bj/buy/o{}/#bread'.format(self.n)
            # 把url交给调度器入队列
            yield scrapy.Request(url=url, callback=self.parse)
```

- **步骤3 - 编写爬虫文件（代码实现2）**

```
"""
    重写start_requests()方法，效率极高
"""
# -*- coding: utf-8 -*-
import scrapy
from ..items import CarItem

class GuaziSpider(scrapy.Spider):
    # 爬虫名
    name = 'car2'
    # 允许爬取的域名
    allowed_domains = ['www.guazi.com']
    # 1、去掉start_urls变量
    # 2、重写 start_requests() 方法
    def start_requests(self):
        """生成所有要抓取的URL地址,一次性交给调度器入队列"""
        for i in range(1,6):
            url = 'https://www.guazi.com/bj/buy/o{}/#bread'.format(i)
            # scrapy.Request(): 把请求交给调度器入队列
            yield scrapy.Request(url=url,callback=self.parse)

    def parse(self, response):
        # 基准xpath: 匹配所有汽车的节点对象列表
        li_list = response.xpath('//ul[@class="carlist clearfix js-top"]/li')
        # 给items.py中的 GuaziItem类 实例化
        item = CarItem()
        for li in li_list:
            item['url'] = li.xpath('./a[1]/@href').get()
            item['name'] = li.xpath('./a[1]/@title').get()
            item['price'] = li.xpath('.//div[@class="t-price"]/p/text()').get()

            # 把抓取的数据,传递给了管道文件 pipelines.py
            yield item
```

- **步骤4 - 管道文件处理数据**

```
"""
pipelines.py处理数据
1、mysql数据库建库建表
create database cardb charset utf8;
use cardb;
create table cartab(
```

```
name varchar(200),
price varchar(100),
url varchar(500)
)charset=utf8;
"""
# -*- coding: utf-8 -*-

# 管道1 - 从终端打印输出
class CarPipeline(object):
    def process_item(self, item, spider):
        print(dict(item))
        return item

# 管道2 - 存入MySQL数据库管道
import pymysql
from .settings import *

class CarMysqlPipeline(object):
    def open_spider(self,spider):
        """爬虫项目启动时只执行1次,一般用于数据库连接"""
        self.db = pymysql.connect(MYSQL_HOST,MYSQL_USER,MYSQL_PWD,MYSQL_DB,charset=CHARSET)
        self.cursor = self.db.cursor()

    def process_item(self,item,spider):
        """处理从爬虫文件传过来的item数据"""
        ins = 'insert into guazitab values(%s,%s,%s)'
        car_li = [item['name'],item['price'],item['url']]
        self.cursor.execute(ins,car_li)
        self.db.commit()

        return item

    def close_spider(self,spider):
        """爬虫程序结束时只执行1次,一般用于数据库断开"""
        self.cursor.close()
        self.db.close()


# 管道3 - 存入MongoDB管道
import pymongo

class CarMongoPipeline(object):
    def open_spider(self,spider):
        self.conn = pymongo.MongoClient(MONGO_HOST,MONGO_PORT)
        self.db = self.conn[MONGO_DB]
        self.myset = self.db[MONGO_SET]

    def process_item(self,item,spider):
        car_dict = {
            'name' : item['name'],
            'price': item['price'],
            'url'  : item['url']
        }
        self.myset.insert_one(car_dict)
```

- **步骤5 - 全局配置文件（settings.py）**

```
【1】ROBOTSTXT_OBEY = False
【2】DOWNLOAD_DELAY = 1
【3】COOKIES_ENABLED = False
【4】DEFAULT_REQUEST_HEADERS = {
    "Cookie": "此处填写抓包抓取到的Cookie",
    "User-Agent": "此处填写自己的User-Agent",
  }

【5】ITEM_PIPELINES = {
     'Car.pipelines.CarPipeline': 300,
     'Car.pipelines.CarMysqlPipeline': 400,
     'Car.pipelines.CarMongoPipeline': 500,
  }

【6】定义MySQL相关变量
MYSQL_HOST = 'localhost'
MYSQL_USER = 'root'
MYSQL_PWD = '123456'
MYSQL_DB = 'guazidb'
CHARSET = 'utf8'

【7】定义MongoDB相关变量
MONGO_HOST = 'localhost'
MONGO_PORT = 27017
MONGO_DB = 'guazidb'
MONGO_SET = 'guaziset'
```

- **步骤6 - 运行爬虫（run.py）**

```
"""run.py"""
from scrapy import cmdline
cmdline.execute('scrapy crawl car'.split())
```

知识点汇总

- **数据持久化 - 数据库**

```
【1】在setting.py中定义相关变量
【2】pipelines.py中导入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):
        """具体处理数据"""
        return item

    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}
```

```
14
15 【注意】 : process_item() 函数中一定要 return item ,当前管道的process_item()的返回值会作为下一个管道 process_item()的参数
16
17 【4】日志级别
18     4.1》5个级别 : DEBUG < INFO < WARNING < ERROR < CRITICAL
19         DEBUG : 调试信息
20         INFO  : 一般信息
21         WARNING:警告信息
22         ERROR : 错误信息
23         CRITICAL:严重错误
24     4.2》settings.py中提供了2个变量
25         LOG_LEVEL = 'WARNING'  # 只显示WARNING和比WARNING严重信息(WARNING ERROR CRITECAL)
26         LOG_FILE = 'guazi.log' # 把日志存放到日志文件中(guazi.log)
```

- **数据持久化 - csv、json文件**

```
1 【1】存入csv文件
2     scrapy crawl car -o car.csv
3
4 【2】存入json文件
5     scrapy crawl car -o car.json
6
7 【3】注意: settings.py中设置导出编码 - 主要针对json文件
8     FEED_EXPORT_ENCODING = 'utf-8'
```

- **节点对象.xpath('')**

```
1 【1】列表,元素为选择器 @
2     [
3         <selector xpath='xxx' data='A'>,
4         <selector xpath='xxx' data='B'>
5     ]
6 【2】列表.extract() : 序列化列表中所有选择器为Unicode字符串 ['A','B']
7 【3】列表.extract_first() 或者 get() :获取列表中第1个序列化的元素(字符串) 'A'
```

- **课堂练习**

```
1 【熟悉整个流程】 : 将猫眼电影案例数据抓取，存入MySQL数据库
```

瓜子二手车直卖网 - 二级页面

- **目标说明**

```
【1】在抓取一级页面的代码基础上升级
【2】一级页面所抓取数据（和之前一样）：
    2.1) 汽车链接
    2.2) 汽车名称
    2.3) 汽车价格
【3】二级页面所抓取数据
    3.1) 行驶里程: //ul[@class="assort clearfix"]/li[2]/span/text()
    3.2) 排量:    //ul[@class="assort clearfix"]/li[3]/span/text()
    3.3) 变速箱:  //ul[@class="assort clearfix"]/li[4]/span/text()
```

在原有项目基础上实现

- **步骤1 - items.py**

```python
# 添加二级页面所需抓取的数据结构

import scrapy

class GuaziItem(scrapy.Item):
    # define the fields for your item here like:
    # 一级页面：链接、名称、价格
    url = scrapy.Field()
    name = scrapy.Field()
    price = scrapy.Field()
    # 二级页面：时间、里程、排量、变速箱
    time = scrapy.Field()
    km = scrapy.Field()
    disp = scrapy.Field()
    trans = scrapy.Field()
```

- **步骤2 - car2.py**

```python
"""
    重写start_requests()方法，效率极高
"""
# -*- coding: utf-8 -*-
import scrapy
from ..items import CarItem

class GuaziSpider(scrapy.Spider):
    # 爬虫名
    name = 'car2'
    # 允许爬取的域名
    allowed_domains = ['www.guazi.com']
    # 1、去掉start_urls变量
    # 2、重写 start_requests() 方法
    def start_requests(self):
        """生成所有要抓取的URL地址,一次性交给调度器入队列"""
        for i in range(1,6):
            url = 'https://www.guazi.com/bj/buy/o{}/#bread'.format(i)
            # scrapy.Request(): 把请求交给调度器入队列
```

```
            yield scrapy.Request(url=url,callback=self.parse)

    def parse(self, response):
        # 基准xpath: 匹配所有汽车的节点对象列表
        li_list = response.xpath('//ul[@class="carlist clearfix js-top"]/li')
        # 给items.py中的 GuaziItem类 实例化
        item = CarItem()
        for li in li_list:
            item['url'] = 'https://www.guazi.com' + li.xpath('./a[1]/@href').get()
            item['name'] = li.xpath('./a[1]/@title').get()
            item['price'] = li.xpath('.//div[@class="t-price"]/p/text()').get()
            # Request()中meta参数: 在不同解析函数之间传递数据,item数据会随着response一起返回
            yield scrapy.Request(url=item['url'], meta={'meta_1': item},
callback=self.detail_parse)

    def detail_parse(self, response):
        """汽车详情页的解析函数"""
        # 获取上个解析函数传递过来的 meta 数据
        item = response.meta['meta_1']
        item['km'] = response.xpath('//ul[@class="assort
clearfix"]/li[2]/span/text()').get()
        item['disp'] = response.xpath('//ul[@class="assort
clearfix"]/li[3]/span/text()').get()
        item['trans'] = response.xpath('//ul[@class="assort
clearfix"]/li[4]/span/text()').get()

        # 1条数据最终提取全部完成,交给管道文件处理
        yield item
```

- **步骤3 - pipelines.py**

```
# 将数据存入mongodb数据库,此处我们就不对MySQL表字段进行操作了,如有兴趣可自行完善
# MongoDB管道
import pymongo

class GuaziMongoPipeline(object):
    def open_spider(self,spider):
        """爬虫项目启动时只执行1次,用于连接MongoDB数据库"""
        self.conn = pymongo.MongoClient(MONGO_HOST,MONGO_PORT)
        self.db = self.conn[MONGO_DB]
        self.myset = self.db[MONGO_SET]

    def process_item(self,item,spider):
        car_dict = dict(item)
        self.myset.insert_one(car_dict)
        return item
```

- **步骤4 - settings.py**

```
# 定义MongoDB相关变量
MONGO_HOST = 'localhost'
MONGO_PORT = 27017
MONGO_DB = 'guazidb'
MONGO_SET = 'guaziset'
```

今日作业

【1】腾讯招聘职位信息抓取（二级页面）
    要求：输入职位关键字，抓取该类别下所有职位信息（到职位详情页抓取）
    具体数据如下：
    1.1）职位名称
    1.2）职位地点
    1.3）职位类别
    1.4）发布时间
    1.5）工作职责
    1.6）工作要求