第 14 关、Scrapy 实操

1、项目：Scrapy 爬取“职友集”的招聘信息

1-1、明确目标

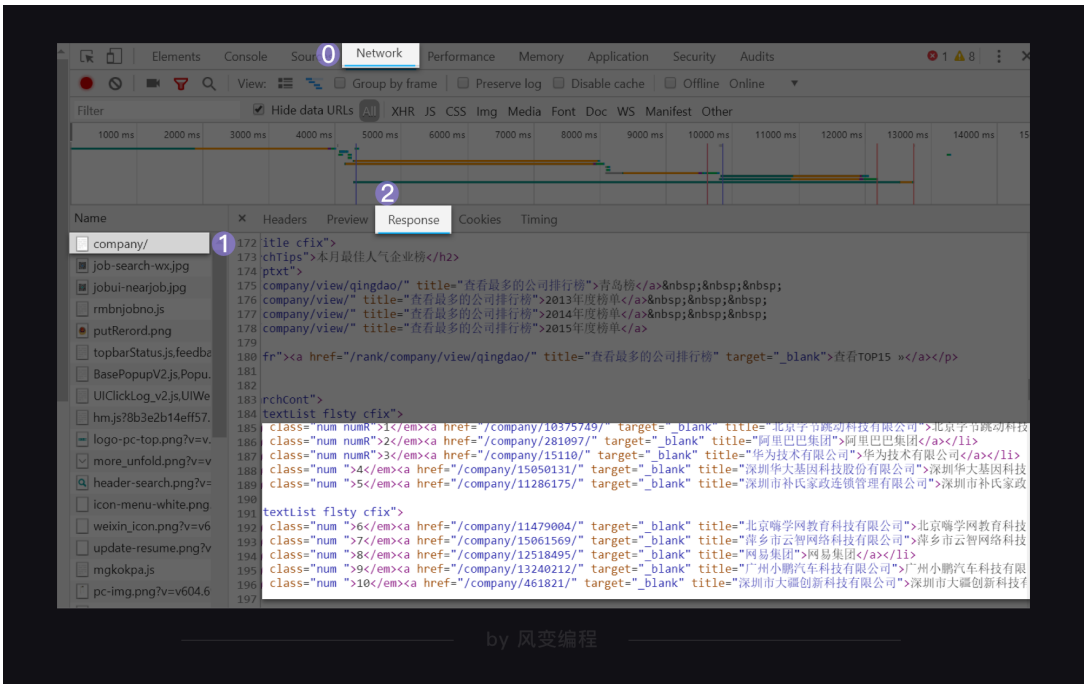
（1）目标网站：https://www.jobui.com/rank/company/；
（2）项目目标：爬取企业排行榜四个榜单里的公司，再跳转到这40家公司的招聘信息页面，爬取到公司名称、职位、工作地点和招聘要求。

1-2、过程分析

1-2-1、企业排行榜的公司信息

（1）确定数据所在页面

右击打开“检查”工具，点击 Network ，刷新页面。点开第 0 个请求 company/，看 Response，发现四个榜单的所有公司信息都在里面，说明企业排行榜的公司信息就藏在 html 里。



（2）确定数据所在位置

①、数据规律

/company/+数字/ 为公司 id 的标识。

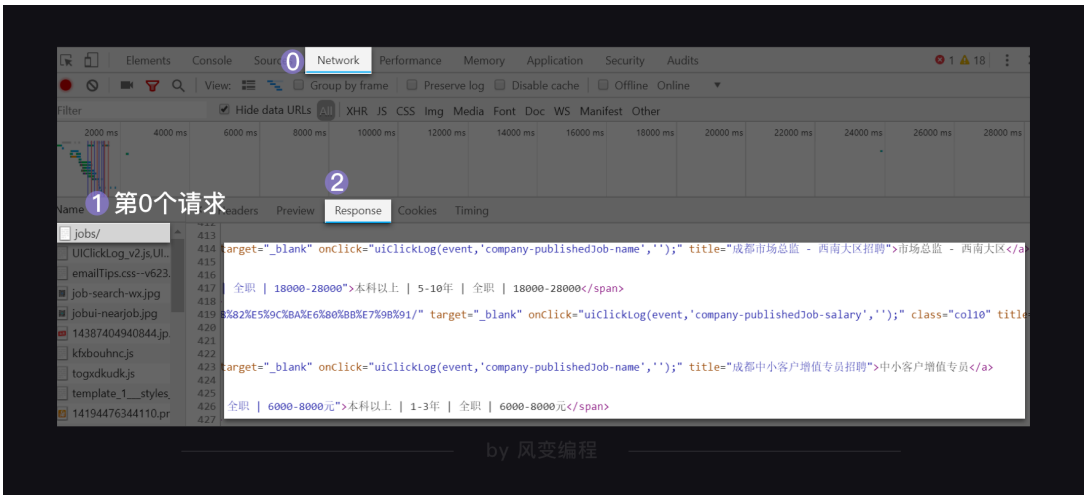
②、数据位置

先抓取最外层的 <ul> 标签，再抓取 <ul> 标签里的 <a> 元素，最后提取到 <a> 元素 href 属性的值。

1-2-2、公司详情页面的招聘信息

（1）确定数据所在页面

右击打开“检查”工具，点击 Network ，刷新页面。我们点击第 0 个请求 jobs/ ，查看 Response，里面有我们的招聘信息，说明企业排行榜的公司信息就藏在 html 里。



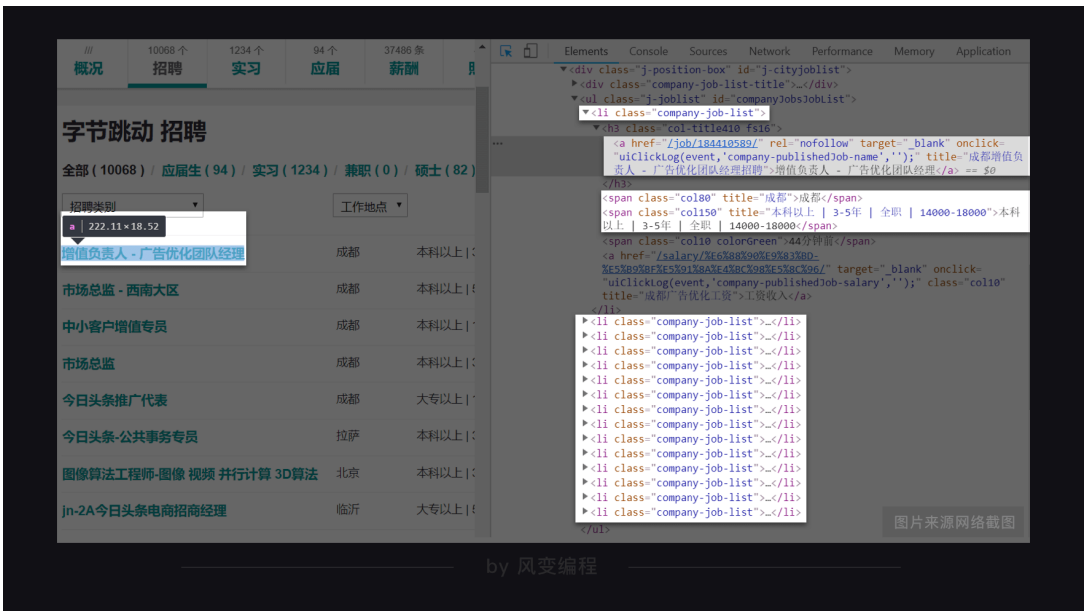
（2）确定数据所在位置

①、数据规律

/company/+数字/+jobs/ 为公司招聘详情信息。

②、数据位置

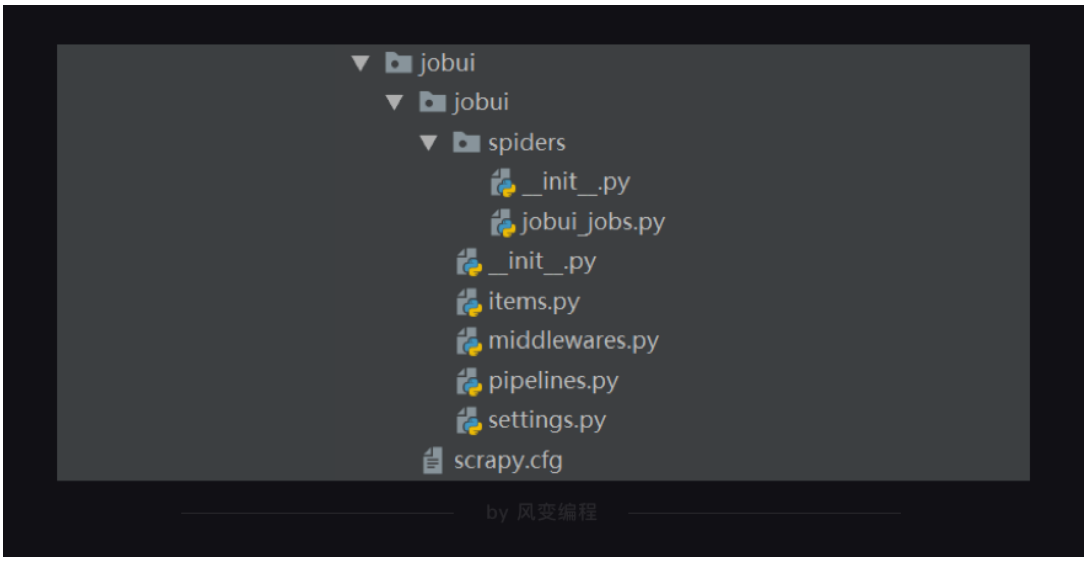
每个岗位的信息都藏在一个 <li> 标签下，职位名称在 <a> 元素的文本中，工作地点在 元素里，职位要求在 元素里。



1–3、代码实现

1-3-1、创建项目

创建 Scrapy 项目的命令：scrapy startproject jobui，jobui 就是 Scrapy 项目的名字。按下 enter 键，成功创建 Scrapy 项目。



1-3-2、定义 item

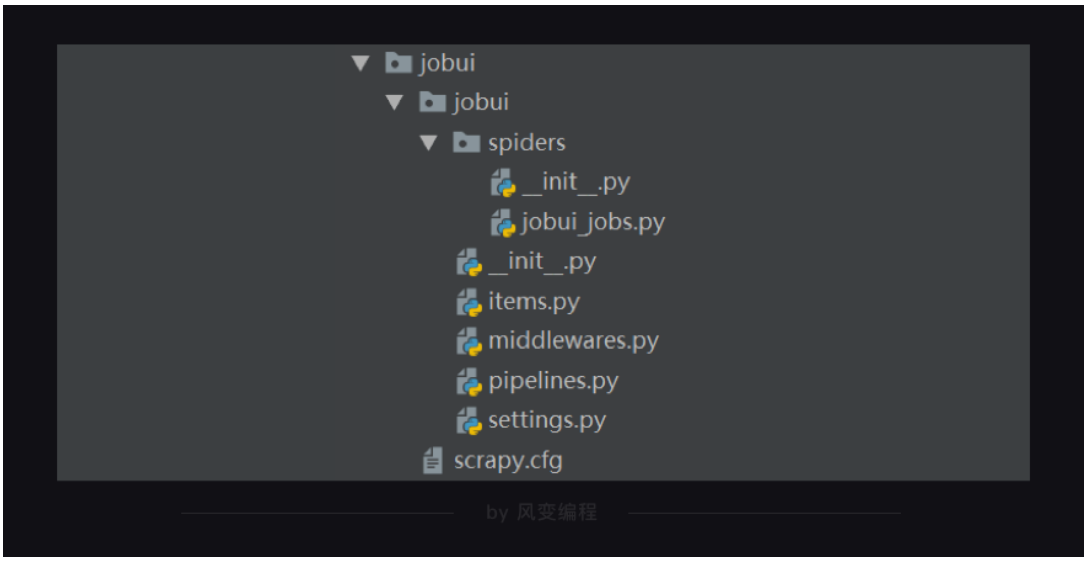
确定要爬取的数据是公司名称、职位名称、工作地点和招聘要求。

```
import scrapy

class JobuiItem(scrapy.Item):
#定义了一个继承自scrapy.Item的JobuiItem类
    company = scrapy.Field()
    #定义公司名称的数据属性
    position = scrapy.Field()
    #定义职位名称的数据属性
    address = scrapy.Field()
    #定义工作地点的数据属性
    detail = scrapy.Field()
    #定义招聘要求的数据属性
```

1-4、创建和编写爬虫文件

在 spiders 里创建爬虫文件，命名为 jobui_ jobs 。



（1）导入模块

```
import scrapy
import bs4
from ..items import JobuiItem
```

（2）核心代码

结合前面的数据规律分析和数据位置，可以得出以下代码：

```
#导入模块
import scrapy
import bs4
from ..items import JobuiItem

class JobuiSpider(scrapy.Spider):
#定义一个爬虫类JobuiSpider
    name = 'jobs'
    #定义爬虫的名字为jobs
    allowed_domains = ['www.jobui.com']
    #定义允许爬虫爬取网址的域名——职友集网站的域名
    start_urls = ['https://www.jobui.com/rank/company/']
    #定义起始网址——职友集企业排行榜的网址

    def parse(self, response):
    #parse是默认处理response的方法
        bs = bs4.BeautifulSoup(response.text, 'html.parser')
        #用BeautifulSoup解析response（企业排行榜的网页源代码）
        ul_list = bs.find_all('ul',class_="textList flsty cfix")
        #用find_all提取<ul class_="textList flsty cfix">标签
        for ul in ul_list:
        #遍历ul_list
            a_list = ul.find_all('a')
            #用find_all提取出<ul class_="textList flsty cfix">元素里的所有<a>元素
            for a in a_list:
            #再遍历a_list
                company_id = a['href']
                #提取出所有<a>元素的href属性的值，也就是公司id标识
                url = 'https://www.jobui.com{id}jobs'
```

```
                real_url = url.format(id=company_id)
                #构造出公司招聘信息的网址链接
```

- 第 6–13 行代码：定义了爬虫类 JobuiSpider 、爬虫的名字 jobs 、允许爬虫爬取的域名和起始网址。

（3）构造新的 requests 对象和定义新的方法处理 response

```
#导入模块：
import scrapy
import bs4
from ..items import JobuiItem

class JobuiSpider(scrapy.Spider):
    name = 'jobs'
    allowed_domains = ['www.jobui.com']
    start_urls = ['https://www.jobui.com/rank/company/']

#提取公司id标识和构造公司招聘信息的网址：
    def parse(self, response):
    #parse是默认处理response的方法
        bs = bs4.BeautifulSoup(response.text, 'html.parser')
        ul_list = bs.find_all('ul',class_="textList flsty cfix")
        for ul in ul_list:
            a_list = ul.find_all('a')
            for a in a_list:
                company_id = a['href']
                url = 'https://www.jobui.com{id}jobs'
                real_url = url.format(id=company_id)
                yield scrapy.Request(real_url, callback=self.parse_job)
#用yield语句把构造好的request对象传递给引擎。用scrapy.Request构造request对象。callback参数设置调用parsejob方法。

    def parse_job(self, response):
    #定义新的处理response的方法parse_job（方法的名字可以自己起）
        bs = bs4.BeautifulSoup(response.text, 'html.parser')
        #用BeautifulSoup解析response(公司招聘信息的网页源代码)
        company = bs.find(id="companyH1").text
```

```
        #用fin方法提取出公司名称
        datas = bs.find_all('li',class_="company-job-list")
        #用find_all提取<li class_="company-job-list">标签，里面含有招聘信息的数据
        for data in datas:
        #遍历datas
            item = JobuiItem()
            #实例化JobuiItem这个类
            item['company'] = company
            #把公司名称放回JobuiItem类的company属性里
            item['position']=data.find('h3').find('a').text
            #提取出职位名称，并把这个数据放回JobuiItem类的position属性里
            item['address'] = data.find('span',class_="col80").text
            #提取出工作地点，并把这个数据放回JobuiItem类的address属性里
            item['detail'] = data.find('span',class_="col150").text
            #提取出招聘要求，并把这个数据放回JobuiItem类的detail属性里
            yield item
            #用yield语句把item传递给引擎
```

- 第 22 行代码：scrapy.Request 是构造 requests 对象的类。real_url 是我们往 requests 对象里传入的每家公司招聘信息网址的参数。callback 的中文意思是回调。self.parse_job 是我们新定义的 parse_job 方法。往 requests 对象里传入 callback=self.parse_job 这个参数后，引擎就能知道 response 要前往的下一站，是 parse_job() 方法；
- 第 26 行代码，是我们定义的新的 parse_job 方法。这个方法是用来解析和提取公司招聘信息的数据；
- 第 26-42 行代码：提取出公司名称、职位名称、工作地点和招聘要求这些数据，并把这些数据放进我们定义好的 Jobuiltem 类里。

1-4-1、存储文件

（1）setting.py 文件

①、存储成 csv 文件只需在 settings.py 文件里，添加如下代码：

```
FEED_URI='./storage/data/%(name)s.csv'
FEED_FORMAT='CSV'
FEED_EXPORT_ENCODING='ansi'
```

- FEED_URI 是导出文件的路径。'./storage/data/%(name)s.csv'，就是把存储的文件放到与 settings.py 文件同级的 storage 文件夹的 data 子文件夹里；
- FEED_FORMAT 是导出数据格式，写 CSV 就能得到 CSV 格式；
- FEED_EXPORT_ENCODING 是导出文件编码，ansi 是一种在 windows 上的编码格式（也可以把它变成 utf-8 用在mac电脑上）。

②、存储成 Excel 文件的方法：

→ 先在 setting.py 里设置启用 ITEM_PIPELINES：

```
# Configure item pipelines
# See https://doc.scrapy.org/en/latest/topics/item-pipeline.html
ITEM_PIPELINES = {
     'jobuitest.pipelines.JobuitestPipeline': 300,
}
```

- 取消 ITEM_PIPELINES 的注释（删掉#）即可。

→ 再编辑 pipelines.py 文件（编辑 pipelines.py 文件用 openpyxl 模块来实现）：

```
import openpyxl

class JobuiPipeline(object):
#定义一个JobuiPipeline类，负责处理item
    def __init__(self):
    #初始化函数 当类实例化时这个方法会自启动
        self.wb =openpyxl.Workbook()
        #创建工作薄
        self.ws = self.wb.active
        #定位活动表
        self.ws.append(['公司', '职位', '地址', '招聘信息'])
        #用append函数往表格添加表头

    def process_item(self, item, spider):
    #process_item是默认的处理item的方法，就像parse是默认处理response的方法
        line = [item['company'], item['position'], item['address'], item['detail']]
        #把公司名称、职位名称、工作地点和招聘要求都写成列表的形式，赋值给line
        self.ws.append(line)
        #用append函数把公司名称、职位名称、工作地点和招聘要求的数据都添加进表格
        return item
        #将item丢回给引擎，如果后面还有这个item需要经过的itempipeline，引擎会自己调度

    def close_spider(self, spider):
    #close_spider是当爬虫结束运行时，这个方法就会执行
        self.wb.save('./jobui.xlsx')
        #保存文件
        self.wb.close()
        #关闭文件
```

1-4-2、修改设置

（1）setting.py 文件

修改默认设置：添加请求头，以及把 ROBOTSTXT_OBEY=True 改成 ROBOTSTXT_OBEY=False 。

```
#需要修改的默认设置:

# Crawl responsibly by identifying yourself (and your website) on the user-agent
#USER_AGENT = 'douban (+http://www.yourdomain.com)'

# Obey robots.txt rules
ROBOTSTXT_OBEY = False
```

```
# Configure a delay for requests for the same website (default: 0)
```

```
# See https://doc.scrapy.org/en/latest/topics/settings.html#download-delay
# See also autothrottle settings and docs
DOWNLOAD_DELAY = 0.5
```

- 需要取消 DOWNLOAD_DELAY = 0 这行的注释（删掉#）。DOWNLOAD_DELAY 翻译成中文是下载延迟的意思，这行代码可以控制爬虫的速度。因为这个项目的爬取速度不宜过快，我们要把下载延迟的时间改成 0.5 秒。

1-4-3、运行项目