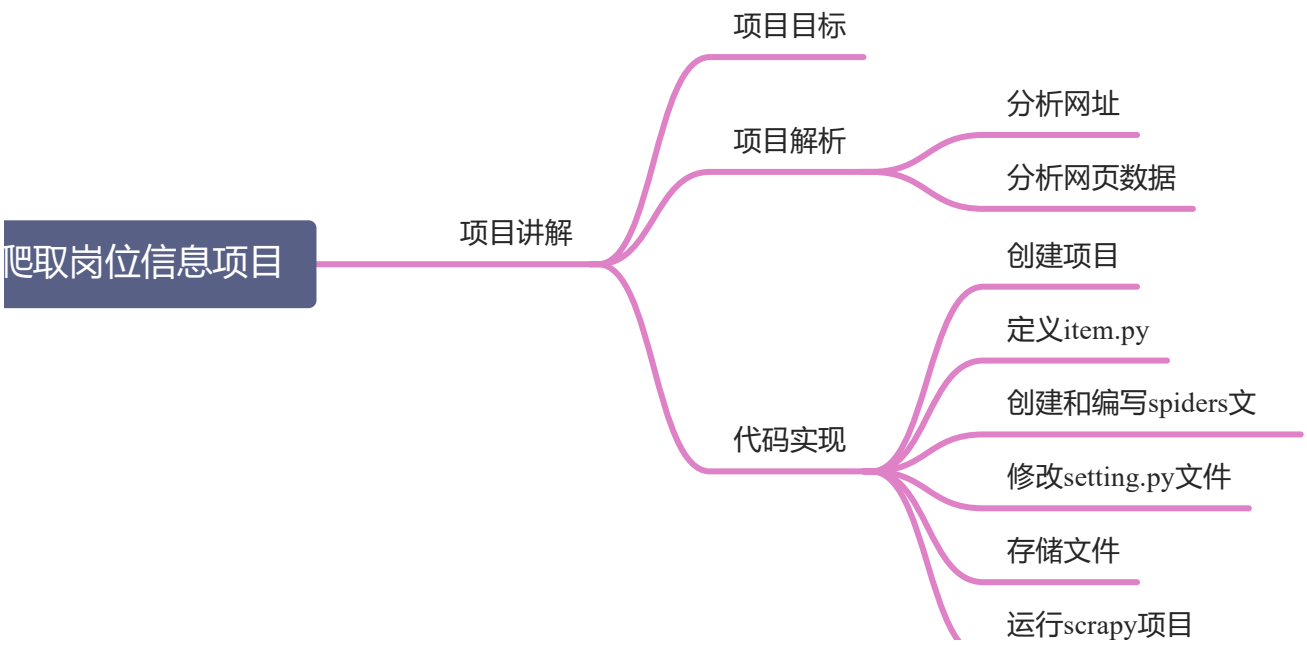
第14课 项目实操：爬取岗位信息

一、课程结构导图



二、项目讲解

2.1 项目目标

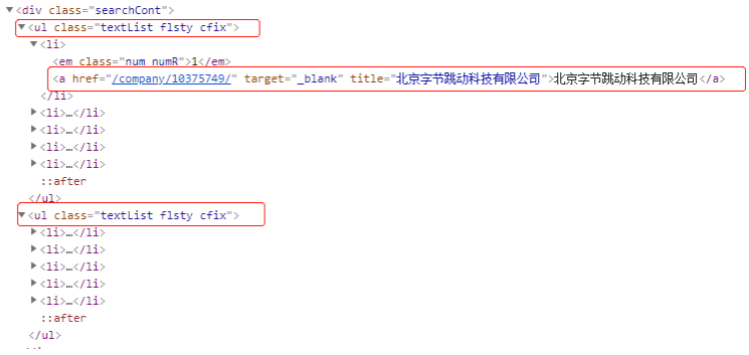
使用scrapy框架爬取人气榜上的公司的招聘首页的信息（包括公司名称、招聘岗位、工作地点、招聘要求）。

网址：https://www.jobui.com/rank/company/ <https://www.jobui.com/rank/company/>

2.2 项目解析

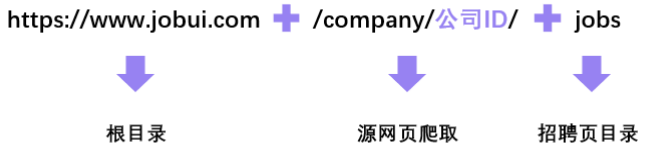
2.2.1 分析网址

查看第0个请求，确认公司详情页链接的关键部分在该页面，即在网页源代码。然后查找排行榜上的公司详情页的链接所在的标签或属性，经过查找，发现公司的详情页链接都在网页源代码中的两个<ul class="textList flsty cfix">标签里。



注意：这里的链接只有【/company/公司ID/】，不是完整的长链接，需要再加上网址的根目录，拼接成完整链接，比如：https://www.jobui.com/company/10375749/ <https://www.jobui.com/company/10375749/> 。

通过详情页链接进入公司主页，查看招聘版块内容，查看此时网址的规律，得出结果如下图，招聘版块的链接由三部分组成：根目录、源网页爬取的公司ID和jobs。



2.2.2 分析网页数据

查看每个公司的招聘信息页面的源代码，分别定位各个目标数据所在的标签和属性，可以得出以下规律：

1. 公司名称在属性【id="companyH1"】所在的<H1>标签下的<a>元素中。

2. 每个岗位的信息都在最小父级标签 下（使用 和 都能够定位到），职位名称在<a>元素的文本中，工作地点和职位要求分别在 元素中的两个元素中。

2.3 代码实现

2.3.1 创建scrapy项目

打开命令符或终端，先切换路径到保存项目的目录下，然后输入以下命令，创建一个scrapy项目。

```
scrapy startproject jobui   # 将项目名称定义为jobui
```

2.3.2 定义item.py文件

定义四个变量，分别用来存放公司名称、职位名称、工作地点、招聘要求，用于传送数据。

```
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
```

2.3.3 创建和编写spiders文件

在spiders文件夹中，定义一个爬虫文件，用于爬取网页数据。

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

class JobuiSpider(scrapy.Spider):
    name = 'jobui'              # 项目名称
    allowed_domains = ['www.jobui.com']     # 域名
    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):
        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)
                # 把构造好的request对象传递给引擎。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
        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").find('a').text        # 提取出公司名称
        datas = bs.find_all('div',class_="job-simple-content")  # 里面含有招聘信息的数据
        for data in 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('a').find('h3').text
            # 提取出职位名称，并把数据返回JobuiItem类的position属性里
            item['address'] = data.find_all('span')[0]['title']
            # 提取出工作地点，并把数据返回JobuiItem类的address属性里
            item['detail'] = data.find_all('span')[1]['title']
            # 提取出招聘要求，并把数据返回JobuiItem类的detail属性里
            yield item                      # 用yield语句把item传递给引擎
```

2.3.4 修改setting.py文件

修改Scrapy中settings.py文件里的默认设置：添加请求头，把ROBOTSTXT_OBEY=True改为ROBOTSTXT_OBEY=False。

```
# Crawl responsibly by identifying yourself (and your website) on the user-agent
USER_AGENT = 'Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) C

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

为了控制爬取速度，可以修改【DOWNLOAD_DELAY = 0】来控制，即取消该行的注释，然后修改赋值。默认单位是秒（查找该关键字方法：ctrl+f或comand+f调出搜索框输入关键词搜索）。

2.3.5 存储文件

- **存储为csv格式**

存为csv，更改setting文件的配置即可，在setting中添加以下三行：

```
FEED_URI='./storage/data/%(name)s.csv'  # 导出文件的路径，可以自定义（会自动创建文件夹和文件）
FEED_FORMAT='CSV'               # 导出数据格式，写CSV就能得到CSV格式。
FEED_EXPORT_ENCODING='ansi'     # 导出文件编码，windows系统可以使用ansi，Mac系统使用utf-8
```

- **存储为Excel格式**

第一步：修改setting文件，取消以下三行的注释：

```
ITEM_PIPELINES = {
    'jobui.pipelines.jobuiPipeline': 300,
}
```

第二步：编写pipelines文件：

```
import openpyxl

class JobuiPipeline(object):
    def __init__(self):
        self.wb =openpyxl.Workbook()        # 创建工作薄
        self.ws = self.wb.active            # 获取活动表
        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
        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)    # 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返回给引擎

    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
```

2.3.6 运行scrapy项目

- **cmd或终端运行**

运行命令如下：

```
scrapy crawl jobui
```

注意：需要先切换路径到项目的根目录。

- **本地编译器运行**

创建main.py文件，代码如下：

```
from scrapy import cmdline
import os
dirpath=os.path.dirname(os.path.abspath(__file__))  # 获取当前路径
os.chdir(dirpath)          # 切换到当前目录

cmdline.execute(['scrapy','crawl','jobui']) # 和爬虫文件中的name的值保持一致
```