# **Day07回顾**

## **selenium+phantomjs/chrome/firefox**

* **设置无界面模式（chromedriver | firefox）**
* options = webdriver.ChromeOptions()  
  options.add\_argument('--headless')  
    
  browser = webdriver.Chrome(options=options)  
  browser.get(url)
* **browser执行JS脚本**
* browser.execute\_script(  
   'window.scrollTo(0,document.body.scrollHeight)'  
  )  
  time.sleep(1)
* **selenium常用操作**
* 【1】键盘操作  
   from selenium.webdriver.common.keys import Keys  
   node.send\_keys(Keys.SPACE)  
   node.send\_keys(Keys.CONTROL, 'a')  
   node.send\_keys(Keys.CONTROL, 'c')  
   node.send\_keys(Keys.CONTROL, 'v')  
   node.send\_keys(Keys.ENTER)  
    
  【2】鼠标操作  
   from selenium.webdriver import ActionChains  
   ActionChains(browser).move\_to\_element(node).perform()  
    
  【3】切换句柄  
   all\_handles = browser.window\_handles  
   time.sleep(1)  
   browser.switch\_to.window(all\_handles[1])  
    
  【4】iframe子框架  
   browser.switch\_to.frame(iframe\_element)  
   # 写法1 - 任何场景都可以:   
   iframe\_node = browser.find\_element\_by\_xpath('')  
   browser.switch\_to.frame(iframe\_node)  
     
   # 写法2 - 默认支持 id 和 name 两个属性值:  
   browser.switch\_to.frame('id属性值|name属性值')

## **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）  
   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：  
   4.1) 数据交给Pipeline处理  
   4.2) 需要跟进URL,继续交给Scheduler入队列，依次循环
* **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
* **全局配置文件settings.py**
* 【1】定义User-Agent  
   USER\_AGENT = 'Mozilla/5.0'  
     
  【2】是否遵循robots协议，一般设置为False  
   ROBOTSTXT\_OBEY = False  
     
  【3】最大并发量，默认为16  
   CONCURRENT\_REQUESTS = 32  
     
  【4】下载延迟时间  
   DOWNLOAD\_DELAY = 1  
     
  【5】请求头，此处也可以添加User-Agent  
   DEFAULT\_REQUEST\_HEADERS = {}

## **还记得百度一下,你就知道吗**

* + **步骤跟踪**
  + 【1】创建项目 'Baidu' 和爬虫文件 'baidu'  
     1.1) scrapy startproject Baidu  
     1.2) cd Baidu  
     1.3) scrapy genspider baidu www.baidu.com  
       
    【2】打开爬虫文件: baidu.py  
     import scrapy  
     class BaiduSpider(scrapy.Spider):  
     name = 'baidu'  
     allowed\_domains = ['www.baidu.com']  
     start\_urls = ['http://www.baidu.com/']  
     def parse(self, response):  
     r\_list = respone.xpath('')  
       
    【3】全局配置文件: settings.py  
     ROBOTSTXT\_OBEY = False  
     DEFAULT\_REQUEST\_HEADERS = {'User-Agent':'Mozilla/5.0'}  
       
    【4】创建文件(和项目目录同路径): run.py  
     from scrapy import cmdline  
     cmdline.execute('scrapy crawl baidu'.split())  
       
    【5】运行 run.py 启动爬虫

# **Day08笔记**

## **协程**

* **定义+说明**
* 【1】定义  
   又称为微线程，纤程，是比线程还要小的单元  
     
  【2】作用  
   在执行A函数的时候，可以随时中断，去执行B函数，然后中断继续执行A函数（可以自动切换），单着一过程并不是函数调用（没有调用语句），过程很像多线程，然而协程只有一个线程在执行  
     
  【3】优势  
   3.1) 协程可以很完美的处理IO密集型的问题  
   3.2) 执行效率高，因为子程序切换函数，而不是线程，没有线程切换的开销，由程序自身控制切换。于多线程相比，线程数量越多，切换开销越大，协程的优势越明显  
   3.3) 不需要锁的机制，只有一个线程，也不存在同时写变量冲突，在控制共享资源时也不需要加锁。  
     
  【4】实现  
   4.1) yield语句 : 实现协程的关键字  
   4.2) 第三方库实现: gevent

## **scrapy框架**

* **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/langfang/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 GuaziItem  
    
  class GuaziSpider(scrapy.Spider):  
   # 爬虫名  
   name = 'guazi'  
   # 允许爬取的域名  
   allowed\_domains = ['www.guazi.com']  
   # 初始的URL地址  
   start\_urls = ['https://www.guazi.com/dachang/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 = GuaziItem()  
   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/dachang/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 GuaziItem  
    
  class GuaziSpider(scrapy.Spider):  
   # 爬虫名  
   name = 'guazi2'  
   # 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/dachang/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 = GuaziItem()  
   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 = 2  
  【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 maoyan'.split())

## **数据持久化(MySQL)**

* **实现步骤**
* 【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}  
    
  【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

## **知识点汇总**

* **节点对象.xpath('')**
* 【1】列表,元素为选择器   
   [  
   <selector xpath='xxx' data='A'>,  
   <selector xpath='xxx' data='B'>  
   ]  
  【2】列表.extract() ：序列化列表中所有选择器为Unicode字符串 ['A','B']  
  【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'
* **日志变量及日志级别(settings.py)**
* # 日志相关变量 - settings.py  
  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
* **管道文件使用**
* 【1】在爬虫文件中为items.py中类做实例化，用爬下来的数据给对象赋值  
   from ..items import CarItem  
   item = CarItem()  
     
  【2】管道文件（pipelines.py）  
    
  【3】开启管道（settings.py）  
   ITEM\_PIPELINES = { '项目目录名.pipelines.类名':优先级 }

## **保存为csv、json文件**

* **命令格式**
* """run.py"""  
  【1】存入csv文件  
   scrapy crawl car -o car.csv  
     
  【2】存入json文件  
   scrapy crawl car -o car.json  
    
  【3】注意: settings.py中设置导出编码 - 主要针对json文件  
   FEED\_EXPORT\_ENCODING = 'utf-8'
* **课堂练习**
* 【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) 上牌时间: //ul[@class="assort clearfix"]/li[1]/span/text()  
   3.2) 行驶里程: //ul[@class="assort clearfix"]/li[2]/span/text()  
   3.3) 排量: //ul[@class="assort clearfix"]/li[3]/span/text()  
   3.4) 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**
* # 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 - guazi2.py**
* # -\*- coding: utf-8 -\*-  
  import scrapy  
  from ..items import GuaziItem  
    
  class GuaziSpider(scrapy.Spider):  
   # 爬虫名  
   name = 'guazi2'  
   # 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/langfang/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')  
   # 存放所有汽车详情页的Request对象  
   for li in li\_list:  
   # 每辆汽车的请求对象  
   item = GuaziItem()  
   item['url'] = 'https://www.guazi.com' + li.xpath('./a[1]/@href').extract()[0]  
   item['name'] = li.xpath('.//h2[@class="t"]/text()').extract()[0]  
   item['price'] = li.xpath('.//div[@class="t-price"]/p/text()').extract()[0]  
    
   # 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['time'] = response.xpath('//ul[@class="assort clearfix"]/li[1]/span/text()').get()  
   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'

## **腾讯招聘职位信息抓取 - 二级页面**

* **1、创建项目+爬虫文件**
* scrapy startproject Tencent  
  cd Tencent  
  scrapy genspider tencent careers.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(名称+类别+职责+要求+地址+时间)  
  https://careers.tencent.com/tencentcareer/api/post/ByPostId?timestamp=1566266695175&postId={}&language=zh-cn
* **2、定义爬取的数据结构**
* import scrapy  
    
  class TencentItem(scrapy.Item):  
   # 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()  
   # 具体职位链接  
   job\_url = scrapy.Field()  
   post\_id = scrapy.Field()
* **3、爬虫文件**
* # -\*- coding: utf-8 -\*-  
  import scrapy  
  from urllib import parse  
  import requests  
  import json  
  from ..items import TencentItem  
    
    
  class TencentSpider(scrapy.Spider):  
   name = 'tencent'  
   allowed\_domains = ['careers.tencent.com']  
   # 定义常用变量  
   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'  
   headers = {'User-Agent': 'Mozilla/5.0'}  
   keyword = input('请输入职位类别:')  
   keyword = parse.quote(keyword)  
    
   # 重写start\_requests()方法  
   def start\_requests(self):  
   total = self.get\_total()  
   # 生成一级页面所有页的URL地址,交给调度器  
   for index in range(1,total+1):  
   url = self.one\_url.format(self.keyword,index)  
   yield scrapy.Request(url=url,callback=self.parse\_one\_page)  
    
   # 获取总页数  
   def get\_total(self):  
   url = self.one\_url.format(self.keyword, 1)  
   html = requests.get(url=url, headers=self.headers).json()  
   count = html['Data']['Count']  
   total = count//10 if count%10==0 else count//10 + 1  
    
   return total  
    
   def parse\_one\_page(self, response):  
   html = json.loads(response.text)  
   for one in html['Data']['Posts']:  
   # 此处是不是有URL需要交给调度器去入队列了？ - 创建item对象！  
   item = TencentItem()  
   item['post\_id'] = one['PostId']  
   item['job\_url'] = self.two\_url.format(item['post\_id'])  
   # 创建1个item对象,请将其交给调度器入队列  
   yield scrapy.Request(url=item['job\_url'],meta={'item':item},callback=self.detail\_page)  
    
   def detail\_page(self,response):  
   """二级页面: 详情页数据解析"""  
   item = response.meta['item']  
   # 将响应内容转为python数据类型  
   html = json.loads(response.text)  
   # 名称+类别+职责+要求+地址+时间  
   item['job\_name'] = html['Data']['RecruitPostName']  
   item['job\_type'] = html['Data']['CategoryName']  
   item['job\_duty'] = html['Data']['Responsibility']  
   item['job\_require'] = html['Data']['Requirement']  
   item['job\_address'] = html['Data']['LocationName']  
   item['job\_time'] = html['Data']['LastUpdateTime']  
    
   # 至此: 1条完整数据提取完成,没有继续送往调度器的请求了,交给管道文件  
   yield item
* **4、提前建库建表 - MySQL**
* create database tencentdb charset utf8;  
  use tencentdb;  
  create table tencenttab(  
  job\_name varchar(500),  
  job\_type varchar(200),  
  job\_duty varchar(5000),  
  job\_require varchar(5000),  
  job\_address varchar(100),  
  job\_time varchar(100)  
  )charset=utf8;
* **5、管道文件**
* class TencentPipeline(object):  
   def process\_item(self, item, spider):  
   return item  
    
  import pymysql  
  from .settings import \*  
    
  class TencentMysqlPipeline(object):  
   def open\_spider(self,spider):  
   """爬虫项目启动时,连接数据库1次"""  
   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):  
   ins='insert into tencenttab values(%s,%s,%s,%s,%s,%s)'  
   job\_li = [  
   item['job\_name'],  
   item['job\_type'],  
   item['job\_duty'],  
   item['job\_require'],  
   item['job\_address'],  
   item['job\_time']  
   ]  
   self.cursor.execute(ins,job\_li)  
   self.db.commit()  
    
   return item  
    
   def close\_spider(self,spider):  
   """爬虫项目结束时,断开数据库1次"""  
   self.cursor.close()  
   self.db.close()
* **6、settings.py**
* ROBOTS\_TXT = False  
  DOWNLOAD\_DELAY = 0.5  
  DEFAULT\_REQUEST\_HEADERS = {  
   'Accept': 'text/html,application/xhtml+xml,application/xml;q=0.9,\*/\*;q=0.8',  
   'Accept-Language': 'en',  
   'User-Agent': 'Mozilla/5.0',  
  }  
  ITEM\_PIPELINES = {  
   'Tencent.pipelines.TencentPipeline': 300,  
   'Tencent.pipelines.TencentMysqlPipeline': 500,  
  }  
  # MySQL相关变量  
  MYSQL\_HOST = 'localhost'  
  MYSQL\_USER = 'root'  
  MYSQL\_PWD = '123456'  
  MYSQL\_DB = 'tencentdb'  
  CHARSET = 'utf8'

## **盗墓笔记小说抓取 - 三级页面**

* **目标**
* 【1】URL地址 ：http://www.daomubiji.com/  
  【2】要求 : 抓取目标网站中盗墓笔记所有章节的所有小说的具体内容，保存到本地文件  
   ./data/novel/盗墓笔记1:七星鲁王宫/七星鲁王\_第一章\_血尸.txt  
   ./data/novel/盗墓笔记1:七星鲁王宫/七星鲁王\_第二章\_五十年后.txt
* **准备工作xpath**
* 【1】一级页面 - 大章节标题、链接：  
   1.1) 基准xpath匹配a节点对象列表: '//li[contains(@id,"menu-item-20")]/a'  
   1.2) 大章节标题: './text()'  
   1.3) 大章节链接: './@href'  
     
  【2】二级页面 - 小章节标题、链接  
   2.1) 基准xpath匹配article节点对象列表: '//article'  
   2.2) 小章节标题: './a/text()'  
   2.3) 小章节链接: './a/@href'  
     
  【3】三级页面 - 小说内容  
   3.1) p节点列表: '//article[@class="article-content"]/p/text()'  
   3.2) 利用join()进行拼接: ' '.join(['p1','p2','p3',''])

### **项目实现**

* **1、创建项目及爬虫文件**
* scrapy startproject Daomu  
  cd Daomu  
  scrapy genspider daomu www.daomubiji.com
* **2、定义要爬取的数据结构 - itemspy**
* import scrapy  
    
  class DaomuItem(scrapy.Item):  
   # define the fields for your item here like:  
   # 1、一级页面: 大标题+链接  
   parent\_title = scrapy.Field()  
   parent\_url = scrapy.Field()  
   # 2、二级页面：小标题+链接  
   son\_title = scrapy.Field()  
   son\_url = scrapy.Field()  
   # 3、三级页面：小说内容  
   content = scrapy.Field()  
   # 4、目录  
   directory = scrapy.Field()
* **3、爬虫文件实现数据抓取 - daomu.py**
* # -\*- coding: utf-8 -\*-  
  import scrapy  
  from ..items import DaomuItem  
  import os  
    
  class DaomuSpider(scrapy.Spider):  
   name = 'daomu'  
   allowed\_domains = ['www.daomubiji.com']  
   start\_urls = ['http://www.daomubiji.com/']  
    
   def parse(self, response):  
   """一级页面解析：提取大标题和链接"""  
   a\_list = response.xpath('//li[contains(@id,"menu-item-20")]/a')  
   for a in a\_list:  
   # 思考: 此处是否需要继续交给调度器入队列？-需要！创建item对象  
   item = DaomuItem()  
   item['parent\_title'] = a.xpath('./text()').get()  
   item['parent\_url'] = a.xpath('./@href').get()  
   directory = './novel/{}/'.format(item['parent\_title'])  
   item['directory'] = directory  
   # 创建对应目录  
   if not os.path.exists(directory):  
   os.makedirs(directory)  
   # 继续交给调度器入队列  
   yield scrapy.Request(  
   url=item['parent\_url'],meta={'meta\_1':item},callback=self.detail\_page)  
    
   def detail\_page(self,response):  
   """二级页面解析：提取小标题名字、链接"""  
   meta1\_item = response.meta['meta\_1']  
   article\_list = response.xpath('//article')  
   for article in article\_list:  
   # 又有继续交给调度器入队列的请求了  
   item = DaomuItem()  
   item['son\_title'] = article.xpath('./a/text()').get()  
   item['son\_url'] = article.xpath('./a/@href').get()  
   item['parent\_title'] = meta1\_item['parent\_title']  
   item['parent\_url'] = meta1\_item['parent\_url']  
   item['directory'] = meta1\_item['directory']  
   # 把每一个章节的item对象交给调度器入队列  
   yield scrapy.Request(  
   url=item['son\_url'],meta={'meta\_2':item},callback=self.get\_content)  
    
   def get\_content(self,response):  
   """三级页面：提取小说具体内容"""  
   # 最后一级页面,没有继续交给调度器入队列的请求了,所以不需要创建item对象了  
   item = response.meta['meta\_2']  
   # content\_list: ['段落1','段落2','段落3','段落4']  
   content\_list = response.xpath('//article[@class="article-content"]/p/text()').extract()  
   item['content'] = '\n'.join(content\_list)  
    
   # 1条数据彻底搞完,交给管道文件处理  
   yield item
* **4、管道文件实现数据处理 - pipelines.py**
* # -\*- coding: utf-8 -\*-  
    
  class DaomuPipeline(object):  
   def process\_item(self, item, spider):  
   # 最终目标: ./novel/盗墓笔记1:七星鲁王宫/七星鲁王\_第一章\_血尸.txt  
   # directory: ./novel/盗墓笔记1:七星鲁王宫/  
   filename = item['directory'] + item['son\_title'].replace(' ','\_') + '.txt'  
   with open(filename,'w') as f:  
   f.write(item['content'])  
    
   return item
* **5、全局配置 - setting.py**
* ROBOTSTXT\_OBEY = False  
  DOWNLOAD\_DELAY = 0.5  
  DEFAULT\_REQUEST\_HEADERS = {  
   'Accept': 'text/html,application/xhtml+xml,application/xml;q=0.9,\*/\*;q=0.8',  
   'Accept-Language': 'en',  
   'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/80.0.3987.149 Safari/537.36'  
  }  
  ITEM\_PIPELINES = {  
   'Daomu.pipelines.DaomuPipeline': 300,  
  }