

# Day07回顾

## selenium+phantomjs/chrome/firefox

### ■ 设置无界面模式 (chromedriver | firefox)

```
1 options = webdriver.ChromeOptions()
2 options.add_argument('--headless')
3
4 browser = webdriver.Chrome(options=options)
5 browser.get(url)
```

### ■ browser执行JS脚本

```
1 browser.execute_script(
2     'window.scrollTo(0,document.body.scrollHeight)'
3 )
4 time.sleep(1)
```

### ■ selenium常用操作

```
1 【1】 键盘操作
2     from selenium.webdriver.common.keys import Keys
3     node.send_keys(Keys.SPACE)
4     node.send_keys(Keys.CONTROL, 'a')
5     node.send_keys(Keys.CONTROL, 'c')
6     node.send_keys(Keys.CONTROL, 'v')
7     node.send_keys(Keys.ENTER)
8
9 【2】 鼠标操作
10    from selenium.webdriver import ActionChains
11    ActionChains(browser).move_to_element(node).perform()
12
13 【3】 切换句柄
14    all_handles = browser.window_handles
15    time.sleep(1)
16    browser.switch_to.window(all_handles[1])
17
18 【4】 iframe子框架
19    browser.switch_to.frame(iframe_element)
20    # 写法1 - 任何场景都可以:
21    iframe_node = browser.find_element_by_xpath('')
22    browser.switch_to.frame(iframe_node)
23
24    # 写法2 - 默认支持 id 和 name 两个属性值:
```

```
25 | browser.switch_to.frame('id属性值|name属性值')
```

## scrapy框架

### ■ 五大组件

```
1 | 【1】引擎 (Engine) ----- 整个框架核心
2 | 【2】爬虫程序 (Spider) ----- 数据解析提取
3 | 【3】调度器 (Scheduler) ----- 维护请求队列
4 | 【4】下载器 (Downloader) ----- 获取响应对象
5 | 【5】管道文件 (Pipeline) ----- 数据入库处理
6 |
7 |
8 | 【两个中间件】
9 |     下载器中间件 (Downloader Middlewares)
10 |    蜘蛛中间件 (Spider Middlewares)
```

### ■ 工作流程

```
1 | 【1】 Engine向Spider索要URL,交给Scheduler入队列
2 | 【2】 Scheduler处理后出队列,通过Downloader Middlewares交给Downloader去下载
3 | 【3】 Downloader得到响应后,通过Spider Middlewares交给Spider
4 | 【4】 Spider数据提取:
5 |     4.1) 数据交给Pipeline处理
6 |     4.2) 需要跟进URL,继续交给Scheduler入队列, 依次循环
```

### ■ 常用命令

```
1 | 【1】创建爬虫项目 : scrapy startproject 项目名
2 | 【2】创建爬虫文件
3 |     2.1) cd 项目文件夹
4 |     2.2) scrapy genspider 爬虫名 域名
5 | 【3】运行爬虫
6 |     scrapy crawl 爬虫名
```

### ■ scrapy项目目录结构

```
1 | Baidu                # 项目文件夹
2 | └─ Baidu              # 项目目录
3 |   └─ items.py         # 定义数据结构
4 |   └─ middlewares.py   # 中间件
5 |   └─ pipelines.py     # 数据处理
6 |   └─ settings.py      # 全局配置
7 |   └─ spiders
8 |       └─ baidu.py     # 爬虫文件
9 | └─ scrapy.cfg         # 项目基本配置文件
```

### ■ 全局配置文件settings.py

```
1 | 【1】定义User-Agent
```

```

2     USER_AGENT = 'Mozilla/5.0'
3
4     【2】是否遵循robots协议，一般设置为False
5     ROBOTSTXT_OBEY = False
6
7     【3】最大并发量，默认为16
8     CONCURRENT_REQUESTS = 32
9
10    【4】下载延迟时间
11    DOWNLOAD_DELAY = 1
12
13    【5】请求头，此处也可以添加User-Agent
14    DEFAULT_REQUEST_HEADERS = {}

```

## 还记得百度一下,你就知道吗

### ■ 步骤跟踪

```

1     【1】创建项目 'Baidu' 和爬虫文件 'baidu'
2         1.1) scrapy startproject Baidu
3         1.2) cd Baidu
4         1.3) scrapy genspider baidu www.baidu.com
5
6     【2】打开爬虫文件: baidu.py
7     import scrapy
8     class BaiduSpider(scrapy.Spider):
9         name = 'baidu'
10        allowed_domains = ['www.baidu.com']
11        start_urls = ['http://www.baidu.com/']
12        def parse(self, response):
13            r_list = response.xpath('')
14
15    【3】全局配置文件: settings.py
16    ROBOTSTXT_OBEY = False
17    DEFAULT_REQUEST_HEADERS = {'User-Agent': 'Mozilla/5.0'}
18
19    【4】创建文件(和项目目录同路径): run.py
20    from scrapy import cmdline
21    cmdline.execute('scrapy crawl baidu'.split())
22
23    【5】运行 run.py 启动爬虫

```

# Day08笔记

## 协程

## ▪ 定义+说明

- 1   **【1】定义**
- 2       又称为微线程，纤程，是比线程还要小的单元
- 3
- 4   **【2】作用**
- 5       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
- 6
- 7   **【3】优势**
- 8       3.1) 协程可以很完美的处理IO密集型的问题
- 9       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
- 10      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。
- 11
- 12   **【4】实现**
- 13      4.1) `yield`语句：实现协程的关键字
- 14      4.2) 第三方库实现：gevent

## scrapy框架

## ▪ 创建爬虫项目步骤

- 1   **【1】新建项目和爬虫文件**
- 2       `scrapy startproject 项目名`
- 3       `cd 项目文件夹`
- 4       新建爬虫文件：`scrapy genspider 文件名 域名`
- 5   **【2】明确目标(items.py)**
- 6   **【3】写爬虫程序(文件名.py)**
- 7   **【4】管道文件(pipelines.py)**
- 8   **【5】全局配置(settings.py)**
- 9   **【6】运行爬虫**
- 10      8.1) 终端：`scrapy crawl 爬虫名`
- 11      8.2) pycharm运行
- 12          a> 创建run.py(和scrapy.cfg文件同目录)
- 13              `from scrapy import cmdline`
- 14              `cmdline.execute('scrapy crawl maoyan'.split())`
- 15          b> 直接运行 run.py 即可

## 瓜子二手车直卖网 - 一级页面

## ▪ 目标

```
1 【1】抓取瓜子二手车官网二手车收据（我要买车）
2
3 【2】URL地址: https://www.guazi.com/langfang/buy/o{}/#bread
4     URL规律: o1 o2 o3 o4 o5 ... ...
5
6 【3】所抓数据
7     3.1) 汽车链接
8     3.2) 汽车名称
9     3.3) 汽车价格
```

## 实现步骤

### ■ 步骤1 - 创建项目和爬虫文件

```
1 scrapy startproject Car
2 cd Car
3 scrapy genspider car www.guazi.com
```

### ■ 步骤2 - 定义要爬取的数据结构

```
1 """items.py"""
2 import scrapy
3
4 class CarItem(scrapy.Item):
5     # 链接、名称、价格
6     url = scrapy.Field()
7     name = scrapy.Field()
8     price = scrapy.Field()
```

### ■ 步骤3 - 编写爬虫文件（代码实现1）

```
1 """
2 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
3
4 xpath表达式如下：
5 【1】基准xpath,匹配所有汽车节点对象列表
6     li_list = response.xpath('//ul[@class="carlist clearfix js-top"]/li')
7
8 【2】遍历后每辆车信息的xpath表达式
9     汽车链接: './a[1]/@href'
10    汽车名称: './h2[@class="t"]/text()'
11    汽车价格: './div[@class="t-price"]/p/text()'
12 """
13 # -*- coding: utf-8 -*-
14 import scrapy
15 from ..items import GuaziItem
16
17 class GuaziSpider(scrapy.Spider):
18     # 爬虫名
19     name = 'guazi'
20     # 允许爬取的域名
```

```

21     allowed_domains = ['www.guazi.com']
22     # 初始的URL地址
23     start_urls = ['https://www.guazi.com/dachang/buy/o1/#bread']
24     # 生成URL地址的变量
25     n = 1
26
27     def parse(self, response):
28         # 基准xpath: 匹配所有汽车的节点对象列表
29         li_list = response.xpath('//ul[@class="carlist clearfix js-top"]/li')
30         # 给items.py中的 GuaziItem类 实例化
31         item = GuaziItem()
32         for li in li_list:
33             item['url'] = li.xpath('./a[1]/@href').get()
34             item['name'] = li.xpath('./a[1]/@title').get()
35             item['price'] = li.xpath('./div[@class="t-price"]/p/text()').get()
36
37             # 把抓取的数据,传递给了管道文件 pipelines.py
38             yield item
39
40         # 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
41         if self.n < 5:
42             self.n += 1
43             url = 'https://www.guazi.com/dachang/buy/o{}/#bread'.format(self.n)
44             # 把url交给调度器入队列
45             yield scrapy.Request(url=url, callback=self.parse)

```

### ■ 步骤3 - 编写爬虫文件 (代码实现2)

```

1     """
2     重写start_requests()方法, 效率极高
3     """
4     # -*- coding: utf-8 -*-
5     import scrapy
6     from ..items import GuaziItem
7
8     class GuaziSpider(scrapy.Spider):
9         # 爬虫名
10        name = 'guazi2'
11        # 允许爬取的域名
12        allowed_domains = ['www.guazi.com']
13        # 1、去掉start_urls变量
14        # 2、重写 start_requests() 方法
15        def start_requests(self):
16            """生成所有要抓取的URL地址, 一次性交给调度器入队列"""
17            for i in range(1,6):
18                url = 'https://www.guazi.com/dachang/buy/o{}/#bread'.format(i)
19                # scrapy.Request(): 把请求交给调度器入队列
20                yield scrapy.Request(url=url, callback=self.parse)
21
22        def parse(self, response):
23            # 基准xpath: 匹配所有汽车的节点对象列表
24            li_list = response.xpath('//ul[@class="carlist clearfix js-top"]/li')
25            # 给items.py中的 GuaziItem类 实例化
26            item = GuaziItem()
27            for li in li_list:
28                item['url'] = li.xpath('./a[1]/@href').get()

```

```

29         item['name'] = li.xpath('./a[1]/@title').get()
30         item['price'] = li.xpath('.//div[@class="t-price"]/p/text()').get()
31
32         # 把抓取的数据,传递给了管道文件 pipelines.py
33         yield item

```

#### ■ 步骤4 - 管道文件处理数据

```

1  """
2  pipelines.py处理数据
3  1、mysql数据库建库建表
4  create database cardb charset utf8;
5  use cardb;
6  create table cartab(
7  name varchar(200),
8  price varchar(100),
9  url varchar(500)
10 )charset=utf8;
11 """
12 # -*- coding: utf-8 -*-
13
14 # 管道1 - 从终端打印输出
15 class CarPipeline(object):
16     def process_item(self, item, spider):
17         print(dict(item))
18         return item
19
20 # 管道2 - 存入MySQL数据库管道
21 import pymysql
22 from .settings import *
23
24 class CarMysqlPipeline(object):
25     def open_spider(self, spider):
26         """爬虫项目启动时只执行1次,一般用于数据库连接"""
27         self.db = pymysql.connect(MYSQL_HOST, MYSQL_USER, MYSQL_PWD, MYSQL_DB, charset=CHARSET)
28         self.cursor = self.db.cursor()
29
30     def process_item(self, item, spider):
31         """处理从爬虫文件传过来的item数据"""
32         ins = 'insert into guazitab values(%s,%s,%s)'
33         car_li = [item['name'], item['price'], item['url']]
34         self.cursor.execute(ins, car_li)
35         self.db.commit()
36
37         return item
38
39     def close_spider(self, spider):
40         """爬虫程序结束时只执行1次,一般用于数据库断开"""
41         self.cursor.close()
42         self.db.close()
43
44
45 # 管道3 - 存入MongoDB管道
46 import pymongo
47
48 class CarMongoPipeline(object):

```

```

49     def open_spider(self, spider):
50         self.conn = pymongo.MongoClient(MONGO_HOST, MONGO_PORT)
51         self.db = self.conn[MONGO_DB]
52         self.myset = self.db[MONGO_SET]
53
54     def process_item(self, item, spider):
55         car_dict = {
56             'name': item['name'],
57             'price': item['price'],
58             'url': item['url']
59         }
60         self.myset.insert_one(car_dict)

```

#### ■ 步骤5 - 全局配置文件 (settings.py)

```

1  【1】ROBOTSTXT_OBEY = False
2  【2】DOWNLOAD_DELAY = 2
3  【3】COOKIES_ENABLED = False
4  【4】DEFAULT_REQUEST_HEADERS = {
5      "Cookie": "此处填写抓包抓取到的Cookie",
6      "User-Agent": "此处填写自己的User-Agent",
7  }
8
9  【5】ITEM_PIPELINES = {
10     'Car.pipelines.CarPipeline': 300,
11     #'Car.pipelines.CarMysqlPipeline': 400,
12     #'Car.pipelines.CarMongoPipeline': 500,
13 }
14
15 【6】定义MySQL相关变量
16 MYSQL_HOST = 'localhost'
17 MYSQL_USER = 'root'
18 MYSQL_PWD = '123456'
19 MYSQL_DB = 'guazidb'
20 CHARSET = 'utf8'
21
22 【7】定义MongoDB相关变量
23 MONGO_HOST = 'localhost'
24 MONGO_PORT = 27017
25 MONGO_DB = 'guazidb'
26 MONGO_SET = 'guaziset'

```

#### ■ 步骤6 - 运行爬虫 (run.py)

```

1  """run.py"""
2  from scrapy import cmdline
3  cmdline.execute('scrapy crawl maoyan'.split())

```

## 数据持久化(MySQL)

#### ■ 实现步骤



```

1  【1】 在setting.py中定义相关变量
2
3  【2】 pipelines.py中导入settings模块
4      def open_spider(self, spider):
5          """爬虫开始执行1次,用于数据库连接"""
6
7          def process_item(self, item, spider):
8              """具体处理数据"""
9              return item
10
11         def close_spider(self, spider):
12             """爬虫结束时执行1次,用于断开数据库连接"""
13
14     【3】 settings.py中添加此管道
15         ITEM_PIPELINES = {'':200}
16
17     【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  【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'

```

### ■ 日志变量及日志级别(settings.py)

```

1  # 日志相关变量 - settings.py
2  LOG_LEVEL = ''
3  LOG_FILE = '文件名.log'
4
5  # 日志级别
6  5 CRITICAL : 严重错误
7  4 ERROR    : 普通错误
8  3 WARNING  : 警告
9  2 INFO     : 一般信息
10 1 DEBUG    : 调试信息
11 # 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  【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
2      from ..items import CarItem
3      item = CarItem()
4
5  【2】管道文件 (pipelines.py)
6
7  【3】开启管道 (settings.py)
8      ITEM_PIPELINES = { '项目目录名.pipelines.类名':优先级 }

```

## 保存为csv、json文件

### 命令格式

```

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

```

### 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  【1】在抓取一级页面的代码基础上升级
2  【2】一级页面所抓取数据（和之前一样）：
3      2.1) 汽车链接
4      2.2) 汽车名称
5      2.3) 汽车价格
6  【3】二级页面所抓取数据
7      3.1) 上牌时间：//ul[@class="assort clearfix"]/li[1]/span/text()
8      3.2) 行驶里程：//ul[@class="assort clearfix"]/li[2]/span/text()
9      3.3) 排量：//ul[@class="assort clearfix"]/li[3]/span/text()
10     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

```

1 # 添加二级页面所需抓取的数据结构
2
3 import scrapy
4
5 class GuaziItem(scrapy.Item):
6     # define the fields for your item here like:
7     # 一级页面：链接、名称、价格
8     url = scrapy.Field()
9     name = scrapy.Field()
10    price = scrapy.Field()
11    # 二级页面：时间、里程、排量、变速箱
12    time = scrapy.Field()
13    km = scrapy.Field()
14    disp = scrapy.Field()
15    trans = scrapy.Field()

```

## ■ 步骤2 - guazi2.py

```

1 # -*- coding: utf-8 -*-
2 import scrapy
3 from ..items import GuaziItem
4
5 class GuaziSpider(scrapy.Spider):
6     # 爬虫名
7     name = 'guazi2'
8     # 允许爬取的域名
9     allowed_domains = ['www.guazi.com']
10    # 1、去掉start_urls变量
11    # 2、重写 start_requests() 方法
12    def start_requests(self):
13        """生成所有要抓取的URL地址,一次性交给调度器入队列"""
14        for i in range(1,6):
15            url = 'https://www.guazi.com/langfang/buy/o{}/#bread'.format(i)
16            # scrapy.Request(): 把请求交给调度器入队列
17            yield scrapy.Request(url=url,callback=self.parse)
18
19    def parse(self, response):
20        # 基准xpath: 匹配所有汽车的节点对象列表
21        li_list = response.xpath('//ul[@class="carlist clearfix js-top"]/li')
22        # 存放所有汽车详情页的Request对象
23        for li in li_list:
24            # 每辆汽车的请求对象
25            item = GuaziItem()
26            item['url'] = 'https://www.guazi.com' + li.xpath('./a[1]/@href').extract()[0]
27            item['name'] = li.xpath('./h2[@class="t"]/text()').extract()[0]
28            item['price'] = li.xpath('./div[@class="t-price"]/p/text()').extract()[0]
29
30            # 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
31            yield scrapy.Request(url=item['url'],meta={
                'meta_1':item},callback=self.detail_parse)
32
33    def detail_parse(self,response):
34        """汽车详情页的解析函数"""
35        # 获取上个解析函数传递过来的 meta 数据
36        item = response.meta['meta_1']

```

```

37         item['time'] = response.xpath('//ul[@class="assort
clearfix"]/li[1]/span/text()').get()
38         item['km'] = response.xpath('//ul[@class="assort
clearfix"]/li[2]/span/text()').get()
39         item['disp'] = response.xpath('//ul[@class="assort
clearfix"]/li[3]/span/text()').get()
40         item['trans'] = response.xpath('//ul[@class="assort
clearfix"]/li[4]/span/text()').get()
41
42         # 1条数据最终提取全部完成,交给管道文件处理
43         yield item

```

### ■ 步骤3 - pipelines.py

```

1  # 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
2  # MongoDB管道
3  import pymongo
4
5  class GuaziMongoPipeline(object):
6      def open_spider(self, spider):
7          """爬虫项目启动时只执行1次,用于连接MongoDB数据库"""
8          self.conn = pymongo.MongoClient(MONGO_HOST, MONGO_PORT)
9          self.db = self.conn[MONGO_DB]
10         self.myset = self.db[MONGO_SET]
11
12         def process_item(self, item, spider):
13             car_dict = dict(item)
14             self.myset.insert_one(car_dict)
15             return item

```

### ■ 步骤4 - settings.py

```

1  # 定义MongoDB相关变量
2  MONGO_HOST = 'localhost'
3  MONGO_PORT = 27017
4  MONGO_DB = 'guazidb'
5  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

```

1 scrapy startproject Tencent
2 cd Tencent
3 scrapy genspider tencent careers.tencent.com
4
5 # 一级页面(postId):
6 https://careers.tencent.com/tencentcareer/api/post/Query?
timestamp=1566266592644&countryId=&cityId=&bgIds=&productId=&categoryId=&parentCategoryId=&at
trId=&keyword={}&pageIndex={}&pageSize=10&language=zh-cn&area=cn
7
8 # 二级页面(名称+类别+职责+要求+地址+时间)
9 https://careers.tencent.com/tencentcareer/api/post/ByPostId?timestamp=156626695175&postId=
{}&language=zh-cn

```

## ■ 2、定义爬取的数据结构

```

1 import scrapy
2
3 class TencentItem(scrapy.Item):
4     # 名称+类别+职责+要求+地址+时间
5     job_name = scrapy.Field()
6     job_type = scrapy.Field()
7     job_duty = scrapy.Field()
8     job_require = scrapy.Field()
9     job_address = scrapy.Field()
10    job_time = scrapy.Field()
11    # 具体职位链接
12    job_url = scrapy.Field()
13    post_id = scrapy.Field()

```

## ■ 3、爬虫文件

```

1 # -*- coding: utf-8 -*-
2 import scrapy
3 from urllib import parse
4 import requests
5 import json
6 from ..items import TencentItem
7
8
9 class TencentSpider(scrapy.Spider):
10     name = 'tencent'
11     allowed_domains = ['careers.tencent.com']
12     # 定义常用变量
13     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'
14     two_url = 'https://careers.tencent.com/tencentcareer/api/post/ByPostId?
timestamp=156626695175&postId={}&language=zh-cn'
15     headers = {'User-Agent': 'Mozilla/5.0'}
16     keyword = input('请输入职位类别:')
17     keyword = parse.quote(keyword)
18
19     # 重写start_requests()方法
20     def start_requests(self):
21         total = self.get_total()

```

```

22     # 生成一级页面所有页的URL地址,交给调度器
23     for index in range(1,total+1):
24         url = self.one_url.format(self.keyword,index)
25         yield scrapy.Request(url=url,callback=self.parse_one_page)
26
27     # 获取总页数
28     def get_total(self):
29         url = self.one_url.format(self.keyword, 1)
30         html = requests.get(url=url, headers=self.headers).json()
31         count = html['Data']['Count']
32         total = count//10 if count%10==0 else count//10 + 1
33
34         return total
35
36     def parse_one_page(self, response):
37         html = json.loads(response.text)
38         for one in html['Data']['Posts']:
39             # 此处是不是有URL需要交给调度器去入队列了? - 创建item对象!
40             item = TencentItem()
41             item['post_id'] = one['PostId']
42             item['job_url'] = self.two_url.format(item['post_id'])
43             # 创建1个item对象,请将其交给调度器入队列
44             yield scrapy.Request(url=item['job_url'],meta=
{'item':item},callback=self.detail_page)
45
46     def detail_page(self,response):
47         """二级页面: 详情页数据解析"""
48         item = response.meta['item']
49         # 将响应内容转为python数据类型
50         html = json.loads(response.text)
51         # 名称+类别+职责+要求+地址+时间
52         item['job_name'] = html['Data']['RecruitPostName']
53         item['job_type'] = html['Data']['CategoryName']
54         item['job_duty'] = html['Data']['Responsibility']
55         item['job_require'] = html['Data']['Requirement']
56         item['job_address'] = html['Data']['LocationName']
57         item['job_time'] = html['Data']['LastUpdateTime']
58
59         # 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
60         yield item

```

#### ■ 4、提前建库建表 - MySQL

```

1  create database tencentdb charset utf8;
2  use tencentdb;
3  create table tencenttab(
4  job_name varchar(500),
5  job_type varchar(200),
6  job_duty varchar(5000),
7  job_require varchar(5000),
8  job_address varchar(100),
9  job_time varchar(100)
10 )charset=utf8;

```

#### ■ 5、管道文件

```

1 class TencentPipeline(object):
2     def process_item(self, item, spider):
3         return item
4
5 import pymysql
6 from .settings import *
7
8 class TencentMysqlPipeline(object):
9     def open_spider(self, spider):
10         """爬虫项目启动时,连接数据库1次"""
11         self.db = pymysql.connect(MYSQL_HOST, MYSQL_USER, MYSQL_PWD, MYSQL_DB, charset=CHARSET)
12         self.cursor = self.db.cursor()
13
14     def process_item(self, item, spider):
15         ins = 'insert into tencenttab values(%s,%s,%s,%s,%s,%s,%s)'
16         job_li = [
17             item['job_name'],
18             item['job_type'],
19             item['job_duty'],
20             item['job_require'],
21             item['job_address'],
22             item['job_time']
23         ]
24         self.cursor.execute(ins, job_li)
25         self.db.commit()
26
27         return item
28
29     def close_spider(self, spider):
30         """爬虫项目结束时,断开数据库1次"""
31         self.cursor.close()
32         self.db.close()

```

## ■ 6、settings.py

```

1 ROBOTS_TXT = False
2 DOWNLOAD_DELAY = 0.5
3 DEFAULT_REQUEST_HEADERS = {
4     'Accept': 'text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8',
5     'Accept-Language': 'en',
6     'User-Agent': 'Mozilla/5.0',
7 }
8 ITEM_PIPELINES = {
9     'Tencent.pipelines.TencentPipeline': 300,
10    'Tencent.pipelines.TencentMysqlPipeline': 500,
11 }
12 # MySQL相关变量
13 MYSQL_HOST = 'localhost'
14 MYSQL_USER = 'root'
15 MYSQL_PWD = '123456'
16 MYSQL_DB = 'tencentdb'
17 CHARSET = 'utf8'

```

# 盗墓笔记小说抓取 - 三级页面

## ■ 目标

```
1 【1】 URL地址 : http://www.daomubiji.com/
2 【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
3     ./data/novel/盗墓笔记1:七星鲁王宫/七星鲁王_第一章_血尸.txt
4     ./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 【1】 一级页面 - 大章节标题、链接:
2     1.1) 基准xpath匹配a节点对象列表: '//li[contains(@id,"menu-item-20")]/a'
3     1.2) 大章节标题: './text()'
4     1.3) 大章节链接: './@href'
5
6 【2】 二级页面 - 小章节标题、链接
7     2.1) 基准xpath匹配article节点对象列表: '//article'
8     2.2) 小章节标题: './a/text()'
9     2.3) 小章节链接: './a/@href'
10
11 【3】 三级页面 - 小说内容
12     3.1) p节点列表: '//article[@class="article-content"]/p/text()'
13     3.2) 利用join()进行拼接: ' '.join(['p1', 'p2', 'p3', ''])
```

## 项目实施

### ■ 1、创建项目及爬虫文件

```
1 scrapy startproject Daomu
2 cd Daomu
3 scrapy genspider daomu www.daomubiji.com
```

### ■ 2、定义要爬取的数据结构 - itemspy

```
1 import scrapy
2
3 class DaomuItem(scrapy.Item):
4     # define the fields for your item here like:
5     # 1、一级页面: 大标题+链接
6     parent_title = scrapy.Field()
7     parent_url = scrapy.Field()
8     # 2、二级页面: 小标题+链接
9     son_title = scrapy.Field()
10    son_url = scrapy.Field()
11    # 3、三级页面: 小说内容
12    content = scrapy.Field()
13    # 4、目录
14    directory = scrapy.Field()
```



### ■ 3、爬虫文件实现数据抓取 - daomu.py

```
1  # -*- coding: utf-8 -*-
2  import scrapy
3  from ..items import DaomuItem
4  import os
5
6  class DaomuSpider(scrapy.Spider):
7      name = 'daomu'
8      allowed_domains = ['www.daomubiji.com']
9      start_urls = ['http://www.daomubiji.com/']
10
11     def parse(self, response):
12         """一级页面解析：提取大标题和链接"""
13         a_list = response.xpath('//li[contains(@id,"menu-item-20")]/a')
14         for a in a_list:
15             # 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
16             item = DaomuItem()
17             item['parent_title'] = a.xpath('./text()').get()
18             item['parent_url'] = a.xpath('./@href').get()
19             directory = './novel/{}/'.format(item['parent_title'])
20             item['directory'] = directory
21             # 创建对应目录
22             if not os.path.exists(directory):
23                 os.makedirs(directory)
24             # 继续交给调度器入队列
25             yield scrapy.Request(
26                 url=item['parent_url'],meta={'meta_1':item},callback=self.detail_page)
27
28     def detail_page(self,response):
29         """二级页面解析：提取小标题名字、链接"""
30         meta1_item = response.meta['meta_1']
31         article_list = response.xpath('//article')
32         for article in article_list:
33             # 又有继续交给调度器入队列的请求了
34             item = DaomuItem()
35             item['son_title'] = article.xpath('./a/text()').get()
36             item['son_url'] = article.xpath('./a/@href').get()
37             item['parent_title'] = meta1_item['parent_title']
38             item['parent_url'] = meta1_item['parent_url']
39             item['directory'] = meta1_item['directory']
40             # 把每一个章节的item对象交给调度器入队列
41             yield scrapy.Request(
42                 url=item['son_url'],meta={'meta_2':item},callback=self.get_content)
43
44     def get_content(self,response):
45         """三级页面：提取小说具体内容"""
46         # 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
47         item = response.meta['meta_2']
48         # content_list: ['段落1','段落2','段落3','段落4']
49         content_list = response.xpath('//article[@class="article-
content"]/p/text()).extract()
50         item['content'] = '\n'.join(content_list)
51
52         # 1条数据彻底搞完，交给管道文件处理
53         yield item
```

#### ■ 4、管道文件实现数据处理 - pipelines.py

```
1  # -*- coding: utf-8 -*-
2
3  class DaomuPipeline(object):
4      def process_item(self, item, spider):
5          # 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
6          # directory: ./novel/盗墓笔记1:七星鲁王宫/
7          filename = item['directory'] + item['son_title'].replace(' ', '_') + '.txt'
8          with open(filename, 'w') as f:
9              f.write(item['content'])
10
11         return item
12
```

#### ■ 5、全局配置 - setting.py

```
1  ROBOTSTXT_OBEY = False
2  DOWNLOAD_DELAY = 0.5
3  DEFAULT_REQUEST_HEADERS = {
4      'Accept': 'text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8',
5      'Accept-Language': 'en',
6      'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like
7      Gecko) Chrome/80.0.3987.149 Safari/537.36'
8  }
9  ITEM_PIPELINES = {
10     'Daomu.pipelines.DaomuPipeline': 300,
11 }
```