概述

```
【1】定义
    1.1) 网络蜘蛛、网络机器人，抓取网络数据的程序
    1.2) 其实就是用Python程序模仿人点击浏览器并访问网站，而且模仿的越逼真越好

【2】爬取数据的目的
    2.1) 公司项目的测试数据，公司业务所需数据
    2.2) 获取大量数据，用来做数据分析

【3】企业获取数据方式
    3.1) 公司自有数据
    3.2) 第三方数据平台购买(数据堂、贵阳大数据交易所)
    3.3) 爬虫爬取数据

【4】Python做爬虫优势
    4.1) Python ：请求模块、解析模块丰富成熟,强大的Scrapy网络爬虫框架
    4.2) PHP ：对多线程、异步支持不太好
    4.3) JAVA：代码笨重,代码量大
    4.4) C/C++：虽然效率高,但是代码成型慢

【5】爬虫分类
    5.1) 通用网络爬虫(搜索引擎使用,遵守robots协议)
        robots协议： 网站通过robots协议告诉搜索引擎哪些页面可以抓取,哪些页面不能抓取，通用网络爬虫需要遵守robots协议（君子协议）
        示例： https://www.baidu.com/robots.txt
    5.2) 聚焦网络爬虫 ：自己写的爬虫程序

【6】爬取数据步骤
    6.1) 确定需要爬取的URL地址
    6.2) 由请求模块向URL地址发出请求,并得到网站的响应
    6.3) 从响应内容中提取所需数据
        a> 所需数据,保存
        b> 页面中有其他需要继续跟进的URL地址,继续第2步去发请求，如此循环
```

- 重大问题思考

网站如何来判定是人类正常访问还是爬虫程序访问？--检查请求头！！！

```
# 请求头（headers）中的 User-Agent
# 测试案例：向测试网站http://httpbin.org/get发请求，查看请求头(User-Agent)
import requests

url = 'http://httpbin.org/get'
res = requests.get(url=url)
html = res.text
print(html)
# 请求头中:User-Agent为-> python-requests/2.22.0 那第一个被网站干掉的是谁？？？我们是不是需要发送请求时重构一下User-Agent？？？添加 headers 参数！！！
```

- 重大问题解决

```
"""
包装好请求头后,向测试网站发请求,并验证
养成好习惯，发送请求携带请求头，重构User-Agent    User-Agent参数详解
"""
import requests

url = 'http://httpbin.org/get'
headers = {'User-Agent':'Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/535.1 (KHTML, like Gecko) Chrome/14.0.835.163 Safari/535.1'}
html = requests.get(url=url,headers=headers).content.decode('utf-8','ignore')  # 'ignore' 忽略无法转码的字符串 防止网页中带有无法识别字符串而报错
# Uni codeDecodeError: utf-8 xxx cannot decode char \xxx in.
    ignore 可解决
# UnicodeEncodeError: gbk code cannot encode char \xxx in,
    windows 写入文件时常报错误
    # with open('xxx. txt', 'w', encoding='gb18030') as f:
print(html)
```

- 小总结

```
【1】 什么是robots协议，爬虫分为通用网络爬虫和聚焦网络爬虫，只有通用爬虫需要遵守协议
【2】 requests模块使用
    res = requests.get(url=url,headers={'User-Agent':'xxx'})
    响应对象res属性:
        a> res.text      # 字符串文本
        b> res.content   # 二进制文本
        c> res.status_code   # 响应码
        d> res.url       # 真实url
【3】网站乱码解析
方法1:
    res = requests.get(url=url, headers=headers)
    res.encoding = 'utf-8'
    file.write(res.text)
方法2:   #  推荐使用方式

    获取bytes数据，手动转码
```

```
16      requests.get(url=url, headers=headers).content.decode('utf-8')
```

- 抓取步骤

```
【1】确定所抓取数据在响应中是否存在（右键 - 查看网页源码 - 搜索关键字）
【2】数据存在：查看URL地址规律
【3】写正则表达式，来匹配数据
【4】程序结构
    a>每爬取1个页面后随机休眠一段时间


```

```
# 程序结构
class xxxSpider(object):
    def __init__(self):
        # 定义常用变量,url,headers及计数等

    def get_html(self):
        # 获取响应内容函数,使用随机User-Agent

    def parse_html(self):
        # 使用正则表达式来解析页面，提取数据

    def save_html(self):
        # 将提取的数据按要求保存，csv、MySQL数据库等

    def run(self):
        # 程序入口函数，用来控制整体逻辑

if __name__ == '__main__':
    # 程序开始运行时间戳
    start = time.time()
    spider = xxxSpider()
    spider.run()
    # 程序运行结束时间戳
    end = time.time()
    print('执行时间:%.2f' % (end-start))
```

方法

- **思路步骤**

```
【1】先确定是否为动态加载网站
【2】找URL规律
【3】正则表达式 | xpath表达式
【4】定义程序框架，补全并测试代码
```

- **多级页面数据抓取思路**

```
【1】整体思路
    1.1> 爬取一级页面,提取 所需数据+链接,继续跟进
    1.2> 爬取二级页面,提取 所需数据+链接,继续跟进
    1.3> ... ...

【2】代码实现思路
    2.1> 避免重复代码 - 请求、解析需定义函数
```

- **增量爬虫实现思路**

```
【1】原理
    利用Redis集合特性，可将抓取过的指纹添加到redis集合中，根据返回值来判定是否需要抓取
    返回值为1 ：  代表之前未抓取过，需要进行抓取
    返回值为0 ：  代表已经抓取过，无须再次抓取

【2】代码实现模板
import redis
from hashlib import md5
import sys

class XxxIncrSpider:
  def __init__(self):
    self.r = redis.Redis(host='localhost',port=6379,db=0)

  def url_md5(self,url):
    """对URL进行md5加密函数"""
    s = md5()
    s.update(url.encode())
    return s.hexdigest()

  def run_spider(self):
    href_list = ['url1','url2','url3','url4']
    for href in href_list:
      href_md5 = self.url_md5(href)
      if self.r.sadd('spider:urls',href_md5) == 1:
        返回值为1表示添加成功，即之前未抓取过，则开始抓取
      else:
        sys.exit()
```

- **目前反爬处理**

```
【1】基于User-Agent反爬
    1.1) 发送请求携带请求头：headers={'User-Agent' : 'Mozilla/5.0 xxxxxx'}
    1.2) 多个请求时随机切换User-Agent
        a) 定义py文件存放大量User-Agent，导入后使用random.choice()每次随机选择
        b) 使用fake_useragent模块每次访问随机生成User-Agent
            from fake_useragent import UserAgent
            agent = UserAgent().random

【2】响应内容存在特殊字符
    解码时使用ignore参数
    html = requests.get(url=url, headers=headers).content.decode('', 'ignore')
```

```
【1】结果：节点对象列表
  1.1) xpath示例：//div、//div[@class="student"]、//div/a[@title="stu"]/span

【2】结果：字符串列表
  2.1) xpath表达式中末尾为：@src、@href、/text()
```

案例1-猫眼电影top100

- 爬虫需求

```
【1】确定URL地址
    百度搜索 - 猫眼电影 - 榜单 - top100榜

【2】 爬取目标
    所有电影的 电影名称、主演、上映时间
```

- 爬虫实现

```
【1】查看网页源码，确认数据来源
    响应内容中存在所需抓取数据 - 电影名称、主演、上映时间

【2】翻页寻找URL地址规律
    第1页：https://maoyan.com/board/4?offset=0
    第2页：https://maoyan.com/board/4?offset=10
    第n页：offset=(n-1)*10

【3】编写正则表达式
    <div class="movie-item-info">.*?title="(.*?)".*?class="star">(.*?)</p>.*?releasetime">(.*?)</p>

【4】开干吧兄弟
```

- 代码实现

```
"""
猫眼电影top100抓取（电影名称、主演、上映时间）
"""
import requests
import re
import time
import random

class MaoyanSpider:
    def __init__(self):
        self.url = 'https://maoyan.com/board/4?offset={}'
        self.headers = {'User-Agent':'Mozilla/5.0 (Windows NT 10.0; WOW64; Trident/7.0; rv:11.0) like Gecko'}

    def get_html(self, url):
        html = requests.get(url=url, headers=self.headers).text
        # 直接调用解析函数
        self.parse_html(html)

    def parse_html(self, html):
        """解析提取数据"""
        regex = '<div class="movie-item-info">.*?title="(.*?)".*?<p class="star">(.*?)</p>.*?<p class="releasetime">(.*?)</p>'
        pattern = re.compile(regex, re.S)
        r_list = pattern.findall(html)
        # r_list: [('活着','牛犇','2000-01-01'),(),(),...,()]
        self.save_html(r_list)

    def save_html(self, r_list):
        """数据处理函数"""
        item = {}
        for r in r_list:
            item['name'] = r[0].strip()
            item['star'] = r[1].strip()
            item['time'] = r[2].strip()
            print(item)

    def run(self):
        """程序入口函数"""
        for offset in range(0, 91, 10):
            url = self.url.format(offset)
            self.get_html(url=url)
            # 控制数据抓取频率:uniform()生成指定范围内的浮点数
            time.sleep(random.uniform(0,1))

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

案例2-汽车之家

- 领取任务

```
【1】爬取地址
    汽车之家 - 二手车 - 价格从低到高
    https://www.che168.com/beijing/a0_0msdgscncgpi1lto1csp1exx0/


【2】爬取目标
    所有汽车的 型号、行驶里程、上牌时间、档位、排量、车辆所在地、价格

【3】爬取分析
    **********一级页面需抓取***********
        1、车辆详情页的链接

    *********二级页面需抓取***********
        1、名称
        2、行驶里程
        3、上牌时间
        4、档位
        5、排量
        6、车辆所在地
        7、价格
```

- 实现步骤

```
【1】确定响应内容中是否存在所需抓取数据 - 存在

【2】找URL地址规律
    第1页: https://www.che168.com/beijing/a0_0msdgscncgpi1lto1csp1exx0/
    第2页: https://www.che168.com/beijing/a0_0msdgscncgpi1lto1csp2exx0/
    第n页: https://www.che168.com/beijing/a0_0msdgscncgpi1lto1csp{}exx0/

【3】 写正则表达式
    一级页面正则表达式:<li class="cards-li list-photo-li".*?<a href="(.*?)".*?</li>
    二级页面正则表达式:<div class="car-box">.*?<h3 class="car-brand-name">(.*?)
</h3>.*?<ul class="brand-unit-item fn-clear">.*?<li>.*?<h4>(.*?)</h4>.*?<h4>(.*?)
</h4>.*?<h4>(.*?)</h4>.*?<h4>(.*?)</h4>.*?<span class="price"
id="overlayPrice">¥(.*?)<b>

【4】代码实现
```

- 代码实现

```
"""
```

```
汽车之家二手车信息抓取
思路
    1、一级页面：汽车的链接
    2、二级页面：具体汽车信息

建立User-Agent池：防止被网站检测到是爬虫
    使用fake_useragent模块
    安装：sudo pip3 install fake_useragent
    使用：
        from fake_useragent import UserAgent
        UserAgent().random
"""
import requests
import re
import time
import random
from fake_useragent import UserAgent

class CarSpider:
    def __init__(self):
        self.url = 'https://www.che168.com/beijing/a0_0msdgscncgpi1lto1csp{}exx0/'

    def get_html(self, url):
        """功能函数1 - 获取html"""
        headers = { 'User-Agent':UserAgent().random }
        html = requests.get(url=url, headers=headers).text

        return html

    def re_func(self, regex, html):
        """功能函数2 - 正则解析函数"""
        pattern = re.compile(regex, re.S)
        r_list = pattern.findall(html)

        return r_list

    def parse_html(self, one_url):
        """爬虫逻辑函数"""
        one_html = self.get_html(url=one_url)
        one_regex = '<li class="cards-li list-photo-li".*?<a href="(.*?)".*?</li>'
        href_list = self.re_func(regex=one_regex, html=one_html)
        for href in href_list:
            two_url = 'https://www.che168.com' + href
            # 获取1辆汽车的具体信息
            self.get_car_info(two_url)
            # 控制爬取频率
            time.sleep(random.randint(1,2))

    def get_car_info(self, two_url):
        """获取1辆汽车的具体信息"""

        two_html = self.get_html(url=two_url)
```

```
        two_regex = '<div class="car-box">.*?<h3 class="car-brand-name">(.*?)</h3>.*?<h4>(.*?)</h4>.*?<h4>(.*?)</h4>.*?<h4>(.*?)</h4>.*?<h4>(.*?)</h4>.*?<span class="price" id="overlayPrice">￥(.*?)<b>'
        # car_list: [('福睿斯','3万公里','2016年3月','手动 / 1.5L', '廊坊', '5.60'),]
        car_list = self.re_func(regex=two_regex, html=two_html)
        item = {}
        item['name'] = car_list[0][0].strip()
        item['km'] = car_list[0][1].strip()
        item['time'] = car_list[0][2].strip()
        item['type'] = car_list[0][3].split('/')[0].strip()
        item['displace'] = car_list[0][3].split('/')[1].strip()
        item['address'] = car_list[0][4].strip()
        item['price'] = car_list[0][5].strip()
        print(item)

    def run(self):
        for i in range(1,5):
            url = self.url.format(i)
            self.parse_html(url)

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

- 练习 - 将数据存入MySQL数据库

```
create database cardb charset utf8;
use cardb;
create table cartab(
name varchar(100),
km varchar(50),
years varchar(50),
type varchar(50),
displacement varchar(50),
city varchar(50),
price varchar(50)
)charset=utf8;
```

- 使用redis实现增量爬虫

```
"""
  提示：使用redis中的集合，sadd()方法，添加成功返回1，否则返回0
  请各位大佬忽略掉下面代码，自己独立实现
"""

import requests
import re
import time
import random
```

```python
import pymysql
from hashlib import md5
import sys
import redis


class CarSpider(object):
    def __init__(self):
        self.url = 'https://www.che168.com/beijing/a0_0msdgscncgpi1lto1csp{}exx0/'
        self.headers = {'User-Agent':'Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/535.1 (KHTML, like Gecko) Chrome/14.0.835.163 Safari/535.1'}
        self.db = pymysql.connect('localhost','root','123456','cardb',charset='utf8')
        self.cursor = self.db.cursor()
        # 连接redis去重
        self.r = redis.Redis(host='localhost',port=6379,db=0)

    # 功能函数1 - 获取响应内容
    def get_html(self,url):
        html = requests.get(url=url,headers=self.headers).text

        return html

    # 功能函数2 - 正则解析
    def re_func(self,regex,html):
        pattern = re.compile(regex,re.S)
        r_list = pattern.findall(html)

        return r_list

    # 爬虫函数开始
    def parse_html(self,one_url):
        one_html = self.get_html(one_url)
        one_regex = '<li class="cards-li list-photo-li".*?<a href="(.*?)".*?</li>'
        href_list = self.re_func(one_regex,one_html)
        for href in href_list:
            # 加密指纹
            s = md5()
            s.update(href.encode())
            finger = s.hexdigest()
            # 如果指纹表中不存在
            if self.r.sadd('car:urls',finger):
                # 每便利一个汽车信息，必须要把此辆汽车所有数据提取完成后再提取下一辆汽车信息
                url = 'https://www.che168.com' + href

                # 获取一辆汽车的信息
                self.get_data(url)
                time.sleep(random.randint(1,2))
            else:
                sys.exit('抓取结束')

    # 获取一辆汽车信息

    def get_data(self,url):
```

```
            two_html = self.get_html(url)
            two_regex = '<div class="car-box">.*?<h3 class="car-brand-name">(.*?)</h3>.*?<ul class="brand-unit-item fn-clear">.*?<li>.*?<h4>(.*?)</h4>.*?<h4>(.*?)</h4>.*?<h4>(.*?)</h4>.*?<h4>(.*?)</h4>.*?<span class="price" id="overlayPrice">¥(.*?)<b'
            item = {}
            car_info_list = self.re_func(two_regex,two_html)
            item['name'] = car_info_list[0][0]
            item['km'] = car_info_list[0][1]
            item['year'] = car_info_list[0][2]
            item['type'] = car_info_list[0][3].split('/')[0]
            item['displacement'] = car_info_list[0][3].split('/')[1]
            item['city'] = car_info_list[0][4]
            item['price'] = car_info_list[0][5]
            print(item)

            one_car_list = [
                item['name'],
                item['km'],
                item['year'],
                item['type'],
                item['displacement'],
                item['city'],
                item['price']
            ]
            ins = 'insert into cartab values(%s,%s,%s,%s,%s,%s,%s)'
            self.cursor.execute(ins,one_car_list)
            self.db.commit()

        def run(self):
            for p in range(1,2):
                url = self.url.format(p)
                self.parse_html(url)

            # 断开数据库链接
            self.cursor.close()
            self.db.close()

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

- **猫眼电影案例-xpath实现**

```
"""
猫眼电影top100抓取（电影名称、主演、上映时间）
"""
import requests
import time
import random

from lxml import etree
```

```python

class MaoyanSpider:
    def __init__(self):
        self.url = 'https://maoyan.com/board/4?offset={}'
        self.headers = {'User-Agent':'Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; Win64; x64; Trident/5.0; .NET CLR 2.0.50727; SLCC2; .NET CLR 3.5.30729; .NET CLR 3.0.30729; Media Center PC 6.0; InfoPath.3; .NET4.0C; Tablet PC 2.0; .NET4.0E)'}

    def get_html(self, url):
        html = requests.get(url=url, headers=self.headers).text
        # 直接调用解析函数
        self.parse_html(html)

    def parse_html(self, html):
        """解析提取数据 - xpath"""
        p = etree.HTML(html)
        # 基准xpath：每个电影信息的节点对象dd列表 [<element dd at xxx>, <element dd at xxx>,...]
        dd_list = p.xpath('//dl[@class="board-wrapper"]/dd')
        print(dd_list)
        item = {}
        for dd in dd_list:
            item['name'] = dd.xpath('.//p[@class="name"]/a/@title')[0].strip()
            item['star'] = dd.xpath('.//p[@class="star"]/text()')[0].strip()
            item['time'] = dd.xpath('.//p[@class="releasetime"]/text()')[0].strip()
            print(item)

    def run(self):
        """程序入口函数"""
        for offset in range(0, 91, 10):
            url = self.url.format(offset)
            self.get_html(url=url)
            # 控制数据抓取频率:uniform()生成指定范围内的浮点数
            time.sleep(random.uniform(0,1))

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

案例3-豆瓣图书

- 需求分析

```
【1】抓取目标 - 豆瓣图书top250的图书信息
    https://book.douban.com/top250?start=0
    https://book.douban.com/top250?start=25
    https://book.douban.com/top250?start=50
    ... ...

【2】抓取数据
    2.1) 书籍名称 : 红楼梦
    2.2) 书籍描述 : [清] 曹雪芹 著 / 人民文学出版社 / 1996-12 / 59.70元
    2.3) 书籍评分 : 9.6
    2.4) 评价人数 : 286382人评价
    2.5) 书籍类型 : 都云作者痴，谁解其中味?
```

- **步骤分析**

```
【1】确认数据来源 - 响应内容存在
【2】分析URL地址规律 - start为0 25 50 75 ...
【3】xpath表达式
    3.1) 基准xpath,匹配每本书籍的节点对象列表
         //div[@class="indent"]/table

    3.2) 依次遍历每本书籍的节点对象，提取具体书籍数据
         书籍名称 :  .//div[@class="pl2"]/a/@title
         书籍描述 :  .//p[@class="pl"]/text()
         书籍评分 :  .//span[@class="rating_nums"]/text()
         评价人数 :  .//span[@class="pl"]/text()
         书籍类型 :  .//span[@class="inq"]/text()
```

- **代码实现**

```python
import requests
from lxml import etree
from fake_useragent import UserAgent
import time
import random

class DoubanBookSpider:
    def __init__(self):
        self.url = 'https://book.douban.com/top250?start={}'

    def get_html(self, url):
        """使用随机的User-Agent"""
        headers = {'User-Agent':UserAgent().random}
        html = requests.get(url=url, headers=headers).text
        self.parse_html(html)

    def parse_html(self, html):
        """lxml+xpath进行数据解析"""
        parse_obj = etree.HTML(html)
        # 1.基准xpath: 提取每本书的节点对象列表
        table_list = parse_obj.xpath('//div[@class="indent"]/table')
        for table in table_list:
```

```
            item = {}
            # 书名
            name_list = table.xpath('.//div[@class="pl2"]/a/@title')
            item['name'] = name_list[0].strip() if name_list else None
            # 描述
            content_list = table.xpath('.//p[@class="pl"]/text()')
            item['content'] = content_list[0].strip() if content_list else None
            # 评分
            score_list = table.xpath('.//span[@class="rating_nums"]/text()')
            item['score'] = score_list[0].strip() if score_list else None
            # 评价人数
            nums_list = table.xpath('.//span[@class="pl"]/text()')
            item['nums'] = nums_list[0][1:-1].strip() if nums_list else None
            # 类别
            type_list = table.xpath('.//span[@class="inq"]/text()')
            item['type'] = type_list[0].strip() if type_list else None

            print(item)

    def run(self):
        for i in range(5):
            start = (i - 1) * 25
            page_url = self.url.format(start)
            self.get_html(page_url)
            time.sleep(random.randint(1,2))

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

案例4-链家二手房

- **确定是否为静态**

```
打开二手房页面 -> 查看网页源码 -> 搜索关键字
```

- **xpath表达式**

```
【1】基准xpath表达式(匹配每个房源信息节点列表)
    '此处滚动鼠标滑轮时,li节点的class属性值会发生变化,通过查看网页源码确定xpath表达式'
    //ul[@class="sellListContent"]/li[@class="clear LOGVIEWDATA LOGCLICKDATA"]

【2】依次遍历后每个房源信息xpath表达式
  2.1)名称: .//div[@class="positionInfo"]/a[1]/text()
  2.2)地址: .//div[@class="positionInfo"]/a[2]/text()
  2.3)户型+面积+方位+是否精装+楼层+年代+类型
      info_list: './/div[@class="houseInfo"]/text()' ->  [0].strip().split('|')
      a)户型: info_list[0]
      b)面积: info_list[1]
      c)方位: info_list[2]
```

```
        d)精装：info_list[3]
        e)楼层：info_list[4]
        f)年代：info_list[5]
        g)类型：info_list[6]

    2.4)总价+单价
        a)总价：.//div[@class="totalPrice"]/span/text()
        b)单价：.//div[@class="unitPrice"]/span/text()

### 重要：页面中xpath不能全信，一切以响应内容为主
### 重要：页面中xpath不能全信，一切以响应内容为主
### 重要：页面中xpath不能全信，一切以响应内容为主
### 重要：页面中xpath不能全信，一切以响应内容为主
### 重要：页面中xpath不能全信，一切以响应内容为主
### 重要：页面中xpath不能全信，一切以响应内容为主
### 重要：页面中xpath不能全信，一切以响应内容为主
### 重要：页面中xpath不能全信，一切以响应内容为主
### 重要：页面中xpath不能全信，一切以响应内容为主
### 重要：页面中xpath不能全信，一切以响应内容为主
```

- **示意代码**

```python
import requests
from lxml import etree
from fake_useragent import UserAgent

# 1.定义变量
url = 'https://bj.lianjia.com/ershoufang/pg1/'
headers = {'User-Agent':UserAgent().random}
# 2.获取响应内容
html = requests.get(url=url,headers=headers).text
# 3.解析提取数据
parse_obj = etree.HTML(html)
# 3.1 基准xpath,得到每个房源信息的li节点对象列表，如果此处匹配出来空，则一定要查看响应内容
li_list = parse_obj.xpath('//ul[@class="sellListContent"]/li[@class="clear LOGVIEWDATA LOGCLICKDATA"]')
for li in li_list:
    item = {}
    # 名称
    name_list = li.xpath('.//div[@class="positionInfo"]/a[1]/text()')
    item['name'] = name_list[0].strip() if name_list else None
    # 地址
    add_list = li.xpath('.//div[@class="positionInfo"]/a[2]/text()')
    item['add'] = add_list[0].strip() if add_list else None
    # 户型 + 面积 + 方位 + 是否精装 + 楼层 + 年代 + 类型
    house_info_list = li.xpath('.//div[@class="houseInfo"]/text()')
    item['content'] = house_info_list[0].strip() if house_info_list else None
    # 总价
    total_list = li.xpath('.//div[@class="totalPrice"]/span/text()')
    item['total'] = total_list[0].strip() if total_list else None
    # 单价
    unit_list = li.xpath('.//div[@class="unitPrice"]/span/text()')
```

```
        item['unit'] = unit_list[0].strip() if unit_list else None

        print(item)
```

- **完整代码实现 - 自己实现**

```
import requests
from lxml import etree
import time
import random
from fake_useragent import UserAgent

class LianjiaSpider(object):
    def __init__(self):
        self.url = 'https://bj.lianjia.com/ershoufang/pg{}/'

    def parse_html(self,url):
        html = requests.get(url=url,headers=headers,timeout=3).content.decode('utf-8','ignore')
        self.get_data(html)
```

```
  def get_data(self,html):
      p = etree.HTML(html)
      # 基准xpath: [<element li at xxx>,<element li>]
      li_list = p.xpath('//ul[@class="sellListContent"]/li[@class="clear LOGVIEWDATA LOGCLICKDATA"]')
      # for遍历,依次提取每个房源信息,放到字典item中
      item = {}
      for li in li_list:
          # 名称+区域
          name_list = li.xpath('.//div[@class="positionInfo"]/a[1]/text()')
          item['name'] = name_list[0].strip() if name_list else None
          address_list = li.xpath('.//div[@class="positionInfo"]/a[2]/text()')
          item['address'] = address_list[0].strip() if address_list else None
          # 户型+面积+方位+是否精装+楼层+年代+类型
          # h_list: ['']
          h_list = li.xpath('.//div[@class="houseInfo"]/text()')
          if h_list:
              info_list = h_list[0].split('|')
              if len(info_list) == 7:
                  item['model'] = info_list[0].strip()
                  item['area'] = info_list[1].strip()
                  item['direct'] = info_list[2].strip()
                  item['perfect'] = info_list[3].strip()
                  item['floor'] = info_list[4].strip()
                  item['year'] = info_list[5].strip()[:-2]
                  item['type'] = info_list[6].strip()
              else:
                  item['model'] = item['area'] = item['direct'] = item['perfect'] = item['floor'] = item['year'] = item['type'] = None
          else:

              item['model'] = item['area'] = item['direct'] = item['perfect'] =
```

```
    item['floor'] = item['year'] = item['type'] = None

            # 总价+单价
            total_list = li.xpath('.//div[@class="totalPrice"]/span/text()')
            item['total'] = total_list[0].strip() if total_list else None
            unit_list = li.xpath('.//div[@class="unitPrice"]/span/text()')
            item['unit'] = unit_list[0].strip() if unit_list else None

            print(item)

    def run(self):
        for pg in range(1,101):
            url = self.url.format(pg)
            self.parse_html(url)
            time.sleep(random.randint(1,2))

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

多线程爬虫

- 应用场景

```
【1】多进程 ：CPU密集程序
【2】多线程 ：爬虫(网络I/O)、本地磁盘I/O
```

- 队列

```
【1】导入模块
   from queue import Queue

【2】使用
    q = Queue()
    q.put(url)
    q.get()    # 当队列为空时，阻塞
    q.empty() # 判断队列是否为空，True/False

【3】q.get()解除阻塞方式
   3.1) q.get(block=False)
   3.2) q.get(block=True,timeout=3)
   3.3) if not q.empty():
            q.get()
```

- 线程模块

```
# 导入模块
from threading import Thread

# 使用流程
t = Thread(target=函数名) # 创建线程对象
t.start() # 创建并启动线程
t.join()  # 阻塞等待回收线程

# 如何创建多线程
t_list = []

for i in range(5):
    t = Thread(target=函数名)
    t_list.append(t)
    t.start()

for t in t_list:
    t.join()
```

- 线程锁

```
from threading import Lock

lock = Lock()
lock.acquire()
lock.release()

【注意】上锁成功后，再次上锁会阻塞
```

- 多线程爬虫示例代码

```
# 抓取豆瓣电影剧情类别下的电影信息
"""
豆瓣电影 - 剧情 - 抓取
"""
import requests
from fake_useragent import UserAgent
import time
import random
from threading import Thread,Lock
from queue import Queue

class DoubanSpider:
    def __init__(self):
        self.url = 'https://movie.douban.com/j/chart/top_list?type=13&interval_id=100%3A90&action=&start={}&limit=20'
        self.i = 0
```

```
        # 队列 + 锁
        self.q = Queue()
        self.lock = Lock()

    def get_agent(self):
        """获取随机的User-Agent"""
        return UserAgent().random

    def url_in(self):
        """把所有要抓取的URL地址入队列"""
        for start in range(0,684,20):
            url = self.url.format(start)
            # url入队列
            self.q.put(url)

    # 线程事件函数：请求+解析+数据处理
    def get_html(self):
        while True:
            # 从队列中获取URL地址
            # 一定要在判断队列是否为空 和 get() 地址 前后加锁,防止队列中只剩一个地址时出现重复判断
            self.lock.acquire()
            if not self.q.empty():
                headers = {'User-Agent': self.get_agent()}
                url = self.q.get()
                self.lock.release()

                html = requests.get(url=url, headers=headers).json()
                self.parse_html(html)
            else:
                # 如果队列为空,则最终必须释放锁
                self.lock.release()
                break

    def parse_html(self, html):
        """解析"""
        # html: [{},{},{},{}]
        item = {}
        for one_film in html:
            item['rank'] = one_film['rank']
            item['title'] = one_film['title']
            item['score'] = one_film['score']
            print(item)
            # 加锁 + 释放锁
            self.lock.acquire()
            self.i += 1
            self.lock.release()

    def run(self):
        # 先让URL地址入队列
        self.url_in()
        # 创建多个线程,开干吧

        t_list = []
```

```
        for i in range(1):
            t = Thread(target=self.get_html)
            t_list.append(t)
            t.start()

        for t in t_list:
            t.join()

        print('数量:',self.i)

if __name__ == '__main__':
    start_time = time.time()
    spider = DoubanSpider()
    spider.run()
    end_time = time.time()
    print('执行时间:%.2f' % (end_time-start_time))
```

- 【1】所用到的模块
  1.1) from threading import Thread
  1.2) from threading import Lock
  1.3) from queue import Queue

  【2】整体思路
  2.1) 创建URL队列: q = Queue()
  2.2) 产生URL地址,放入队列: q.put(url)
  2.3) 线程事件函数: 从队列中获取地址,开始抓取: url = q.get()
  2.4) 创建多线程,并运行

  【3】代码结构

```
    def __init__(self):
        """创建URL队列"""
        self.q = Queue()
        self.lock = Lock()

    def url_in(self):
        """生成待爬取的URL地址,入队列"""
        pass

    def parse_html(self):
        """线程事件函数,获取地址,进行数据抓取"""
        while True:
            self.lock.acquire()
            if not self.q.empty():
                url = self.q.get()
                self.lock.release()
            else:
                self.lock.release()
                break

    def run(self):
        self.url_in()
```

```
        t_list = []
        for i in range(3):
            t = Thread(target=self.parse_html)
            t_list.append(t)
            t.start()

        for th in t_list:
            th.join()

【4】队列要点：q.get()防止阻塞方式
    4.1) 方法1：q.get(block=False)
    4.2) 方法2：q.get(block=True,timeout=3)
    4.3) 方法3：
        if not q.empty():
           q.get()
```

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
    1.1) 安装依赖包
        a> sudo apt-get install libffi-dev
        b> sudo apt-get install libssl-dev
        c> sudo apt-get install libxml2-dev
        d> sudo apt-get install python3-dev
        e> sudo apt-get install libxslt1-dev
        f> sudo apt-get install zlib1g-dev
        g> sudo pip3 install -I -U service_identity

    1.2) 安装scrapy框架
        a> sudo pip3 install Scrapy

【2】Windows安装
    2.1) cmd命令行(管理员)：python -m pip install Scrapy
    【注意】：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】调度器(Scheduler)  : 维护请求队列
【3】下载器(Downloader): 获取响应对象
【4】爬虫文件(Spider)   : 数据解析提取
【5】项目管道(Pipeline): 数据入库处理
**********************************
【中间件1】: 下载器中间件(Downloader Middlewares) : 引擎->下载器,包装请求(随机代理等)
【中间件2】: 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
    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)
```

小试牛刀

```
【1】执行3条命令,创建项目基本结构
    scrapy startproject Baidu
    cd Baidu
    scrapy genspider baidu www.baidu.com

【2】完成爬虫文件: spiders/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 = response.xpath('/html/head/title/text()').extract()[0]
            print(r_list)

【3】完成settings.py配置
    3.1) ROBOTSTXT_OBEY = False
    3.2) DEFAULT_REQUEST_HEADERS = {
        'User-Agent' : 'Mozilla/5.0'
    }

【4】运行爬虫
    4.1) 创建run.py(和scrapy.cfg同路径)
    4.2) run.py
         from scrapy import cmdline
         cmdline.execute('scrapy crawl baidu'.split())

【5】执行 run.py 运行爬虫
```

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）**

    ```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'] = 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()
```

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**

```
# 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**

```
"""
    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'
```

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**

```
class DaomuItem(scrapy.Item):
    # 拷问：你的pipelines.py中需要处理哪些数据？ 文件名、路径
    # 文件名：小标题名称  son_title：七星鲁王 第一章 血尸
    son_title = scrapy.Field()
    directory = scrapy.Field()
    content = 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):
        """一级页面解析函数：提取大标题+大链接，并把大链接交给调度器入队列"""
        a_list = response.xpath('//li[contains(@id,"menu-item-20")]/a')
        for a in a_list:
            item = DaomuItem()
            parent_title = a.xpath('./text()').get()
            parent_url = a.xpath('./@href').get()
            item['directory'] = './novel/{}/'.format(parent_title)
            # 创建对应文件夹
            if not os.path.exists(item['directory']):
                os.makedirs(item['directory'])
            # 交给调度器入队列
            yield scrapy.Request(url=parent_url, meta={'meta_1':item},
callback=self.detail_page)

    # 返回了11个response，调用了这个函数
    def detail_page(self, response):
        """二级页面解析函数：提取小标题、小链接"""
        # 把item接收
        meta_1 = response.meta['meta_1']
        art_list = response.xpath('//article')
        for art in art_list:
            # 只要有继续交往调度器的请求，就必须新建item对象
            item = DaomuItem()
            item['son_title'] = art.xpath('./a/text()').get()
            son_url = art.xpath('./a/@href').get()
            item['directory'] = meta_1['directory']
```

```
            # 再次交给调度器入队列
            yield scrapy.Request(url=son_url, meta={'item':item},
callback=self.get_content)

    # 盗墓笔记1：传过来了75个response
    # 盗墓笔记2：传过来了 n 个response
    # ... ...
    def get_content(self, response):
        """三级页面解析函数：提取具体小说内容"""
        item = response.meta['item']
        # content_list: ['段落1','段落2','段落3',...]
        content_list = response.xpath('//article[@class="article-
content"]/p/text()').extract()
        item['content'] = '\n'.join(content_list)

        # 至此，一条item数据全部提取完成
        yield item
```

- **4、管道文件实现数据处理 - pipelines.py**

```
class DaomuPipeline(object):
    def process_item(self, item, spider):
        # filename: ./novel/盗墓笔记1:七星鲁王宫/七星鲁王_第一章_血尸.txt
        filename = '{}{}.txt'.format(item['directory'],
item['son_title'].replace(' ', '_'))
        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,
}
```

- **完整流程**

```
【1】scrapy startproject Tencent
【2】cd Tencent
【3】scrapy genspider tencent tencent.com
【4】items.py(定义爬取数据结构)
```

```
    import scrapy
    class TencentItem(scrapy.Item):
        name = scrapy.Field()
        address = scrapy.Field()

【5】tencent.py（写爬虫文件）
    import scrapy
    from ..items import TencentItem
    class TencentSpider(scrapy.Spider):
        name = 'tencent'
        allowed_domains = ['tencent.com']
        start_urls = ['']
        def parse(self, response):
            item = TencentItem()
            item['name'] = xxxx
            yield item

【6】pipelines.py(数据处理)
    class TencentPipeline(object):
        def process_item(self, item, spider):
            return item

【7】settings.py(全局配置)

【8】run.py
    from scrapy import cmdline
    cmdline.execute('scrapy crawl tencent'.split())
```

我们必须记住

- **熟练记住**

```
【1】响应对象response属性及方法
    1.1) response.text ：获取响应内容 - 字符串
    1.2) response.body ：获取bytes数据类型
    1.3) response.xpath('')
    1.4) response.xpath('').extract() ：提取文本内容，将列表中所有元素序列化为Unicode字符串
    1.5) response.xpath('').extract_first() ：序列化提取列表中第1个文本内容
    1.6) response.xpath('').get() ： 提取列表中第1个文本内容(等同于extract_first())

【2】settings.py中常用变量
    2.1) 设置数据导出编码(主要针对于json文件)
         FEED_EXPORT_ENCODING = 'utf-8'
    2.2) 设置User-Agent
         USER_AGENT = ''
    2.3) 设置最大并发数(默认为16)
         CONCURRENT_REQUESTS = 32
    2.4) 下载延迟时间(每隔多长时间请求一个网页)
         DOWNLOAD_DELAY = 0.5
    2.5) 请求头
         DEFAULT_REQUEST_HEADERS = {'Cookie' : 'xxx'}
    2.6) 添加项目管道
```

```
        ITEM_PIPELINES = {'目录名.pipelines.类名' : 优先级}
    2.7) cookie(默认禁用,取消注释-True|False都为开启)
        COOKIES_ENABLED = False
```

爬虫项目启动方式

- **启动方式**

```
【1】方式一:基于start_urls
    1.1) 从爬虫文件(spider)的start_urls变量中遍历URL地址交给调度器入队列,
    1.2) 把下载器返回的响应对象(response)交给爬虫文件的parse(self,response)函数处理

【2】方式二
    重写start_requests()方法,从此方法中获取URL,交给指定的callback解析函数处理
    2.1) 去掉start_urls变量
    2.2) def start_requests(self):
            # 生成要爬取的URL地址,利用scrapy.Request()方法交给调度器
```

数据持久化存储

- **MySQL-MongoDB-Json-csv**

```
**************************存入MySQL、MongoDB*****************************

【1】在setting.py中定义相关变量
【2】pipelines.py中新建管道类,并导入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):
        # 用于处理抓取的item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

*******************************存入JSON、CSV文件**********************
scrapy crawl maoyan -o maoyan.csv
scrapy crawl maoyan -o maoyan.json
【注意】
    存入json文件时候需要添加变量(settings.py) : FEED_EXPORT_ENCODING = 'utf-8'
```

分布式爬虫

- **分布式爬虫介绍**
- 多台主机共享一个爬取队列

```
【1】原理
    多台主机共享1个爬取队列
    scrapy的调度器本身不支持分布式

【2】实现
    2.1) 重写scrapy调度器(scrapy_redis模块)
    2.2) sudo pip3 install scrapy_redis
```

- **为什么使用redis**

```
【1】Redis基于内存,速度快
【2】Redis非关系型数据库,Redis中集合,存储每个request的指纹
```

scrapy_redis详解

- **GitHub地址**

```
https://github.com/rmax/scrapy-redis
```

- **settings.py说明**

```
# 重新指定调度器: 启用Redis调度存储请求队列
SCHEDULER = "scrapy_redis.scheduler.Scheduler"

# 重新指定去重机制: 确保所有的爬虫通过Redis去重
DUPEFILTER_CLASS = "scrapy_redis.dupefilter.RFPDupeFilter"

# 不清除Redis队列: 暂停/恢复/断点续爬(默认清除为False,设置为True不清除)
SCHEDULER_PERSIST = True

# 优先级队列 （默认）
SCHEDULER_QUEUE_CLASS = 'scrapy_redis.queue.PriorityQueue'
#可选用的其它队列
# 先进先出
SCHEDULER_QUEUE_CLASS = 'scrapy_redis.queue.FifoQueue'
# 后进先出
SCHEDULER_QUEUE_CLASS = 'scrapy_redis.queue.LifoQueue'

# redis管道
ITEM_PIPELINES = {
    'scrapy_redis.pipelines.RedisPipeline': 300
}

#指定连接到redis时使用的端口和地址
REDIS_HOST = 'localhost'
REDIS_PORT = 6379
```

腾讯招聘分布式改写

- **分布式爬虫完成步骤**

```
【1】首先完成非分布式scrapy爬虫 : 正常scrapy爬虫项目抓取
【2】设置,部署成为分布式爬虫
```

- **分布式环境说明**

```
【1】分布式爬虫服务器数量: 2 (其中1台Windows,1台Ubuntu虚拟机)
【2】服务器分工:
    2.1) Windows : 负责数据抓取
    2.2) Ubuntu  : 负责URL地址统一管理,同时负责数据抓取
```

- **腾讯招聘分布式爬虫 - 数据同时存入1个Redis数据库**

```
【1】完成正常scrapy项目数据抓取 (非分布式 - 拷贝之前的Tencent)

【2】设置settings.py, 完成分布式设置
    2.1-必须) 使用scrapy_redis的调度器
        SCHEDULER = "scrapy_redis.scheduler.Scheduler"

    2.2-必须) 使用scrapy_redis的去重机制
        DUPEFILTER_CLASS = "scrapy_redis.dupefilter.RFPDupeFilter"

    2.3-必须) 定义redis主机地址和端口号
        REDIS_HOST = '192.168.1.107'
        REDIS_PORT = 6379

    2.4-非必须) 是否清除请求指纹,True:不清除 False:清除 (默认)
        SCHEDULER_PERSIST = True

    2.5-非必须) 在ITEM_PIPELINES中添加redis管道,数据将会存入redis数据库
        'scrapy_redis.pipelines.RedisPipeline': 200

【3】把代码原封不动的拷贝到分布式中的其他爬虫服务器,同时开始运行爬虫

【结果】: 多台机器同时抓取,数据会统一存到Ubuntu的redis中, 而且所抓数据不重复
```

- **腾讯招聘分布式爬虫 - 数据存入MySQL数据库**

```
"""和数据存入redis步骤基本一样,只是变更一下管道和MySQL数据库服务器的IP地址"""
【1】settings.py
    1.1) SCHEDULER = 'scrapy_redis.scheduler.Scheduler'
    1.2) DUPEFILTER_CLASS = 'scrapy_redis.dupefilter.RFPDupeFilter'
    1.3) SCHEDULER_PERSIST = True
    1.4) REDIS_HOST = '192.168.1.105'
    1.5) REDIS_PORT = 6379
    1.6) ITEM_PIPELINES = {'Tencent.pipelines.TencentMysqlPipeline' : 300}
    1.7) MYSQL_HOST = '192.168.1.105'

```

```
【2】将代码拷贝到分布式中所有爬虫服务器

【3】多台爬虫服务器同时运行scrapy爬虫

# 赠送腾讯MySQL数据库建库建表语句
"""
create database tencentdb charset utf8;
use tencentdb;
create table tencenttab(
job_name varchar(1000),
job_type varchar(200),
job_duty varchar(5000),
job_require varchar(5000),
job_address varchar(200),
job_time varchar(200)
)charset=utf8;
"""
```