Day01笔记

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！！！

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

正则解析模块re

re模块使用流程

```
# 方法一
r_list=re.findall('正则表达式',html,re.S)
# re.S 让正则的.能够匹配\n换行符

# 方法二
pattern = re.compile('正则表达式',re.S)
r_list = pattern.findall(html)
```

- **思考 - 请写出匹配任意一个字符的正则表达式?**

```
import re
# 方法一
pattern = re.compile('[\s\S]')
result = pattern.findall(html)

# 方法二
pattern = re.compile('.',re.S)
result = pattern.findall(html)
```

- **代码示例**

```
import re

html = '''
<div><p>九霄龙吟惊天变</p></div>
<div><p>风云际会潜水游</p></div>
'''
# 贪婪匹配
p = re.compile('<div><p>.*</p></div>',re.S)
r_list = p.findall(html)
print(r_list)

# 非贪婪匹配
p = re.compile('<div><p>.*?</p></div>',re.S)
r_list = p.findall(html)
print(r_list)
```

正则表达式分组

- **作用**

```
在完整的模式中定义子模式，将每个圆括号中子模式匹配出来的结果提取出来
```

- **示例代码**

```
import re

s = 'A B C D'
p1 = re.compile('\w+\s+\w+')
print(p1.findall(s))
# 分析结果是什么？？？
# ['A B', 'C D']

p2 = re.compile('(\w+)\s+\w+')
print(p2.findall(s))
# 第一步: ['A B', 'C D']
# 第二步: ['A', 'C']

```

```
14
15 p3 = re.compile('(\w+)\s+(\w+)')
16 print(p3.findall(s))
17 # 第一步: ['A B', 'C D']
18 # 第二步: [('A','B'), ('C','D')]
```

- **分组总结**

```
1 1、在网页中,想要什么内容,就加()
2 2、先按整体正则匹配,然后再提取分组()中的内容
3    如果有2个及以上分组(),则结果中以元组形式显示 [(),(),()]
```

- **课堂练习**

```
1 # 从如下html代码结构中完成如下内容信息的提取:
2 问题1 : [('Tiger',' Two...'),('Rabbit','Small..')]
3 问题2 :
4     动物名称 : Tiger
5     动物描述 : Two tigers two tigers run fast
6     ***********************************************
7     动物名称 : Rabbit
8     动物描述 : Small white rabbit white and white
```

- **页面结构如下**

```
1  <div class="animal">
2      <p class="name">
3              <a title="Tiger"></a>
4      </p>
5      <p class="content">
6              Two tigers two tigers run fast
7      </p>
8  </div>
9
10 <div class="animal">
11     <p class="name">
12             <a title="Rabbit"></a>
13     </p>
14
15     <p class="content">
16             Small white rabbit white and white
17     </p>
18 </div>
```

- **练习答案**

```
1 import re
2
3 html = '''<div class="animal">
4     <p class="name">
5         <a title="Tiger"></a>
6     </p>
7
8     <p class="content">
9         Two tigers two tigers run fast
```

```
    </p>
</div>

<div class="animal">
    <p class="name">
        <a title="Rabbit"></a>
    </p>

    <p class="content">
        Small white rabbit white and white
    </p>
</div>'''

p = re.compile('<div class="animal">.*?title="(.*?)".*?content">(.*?)</p>.*?</div>',re.S)
r_list = p.findall(html)

for rt in r_list:
    print('动物名称:',rt[0].strip())   # strip() 去掉字符串两头的空白包括\n\t和空格
    print('动物描述:',rt[1].strip())
    print('*' * 50)


# 把想要提取的数据先复制出来，再按需删除，删除的地方加上.*?,需要提取的地方加(.*?)
```

猫眼电影top100抓取案例

- **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()
```

数据持久化 - MySQL

- **pymysql回顾**

```python
import pymysql

db = pymysql.connect('localhost','root','123456','maoyandb',charset='utf8')
cursor = db.cursor()

ins = 'insert into filmtab values(%s,%s,%s)'
cursor.execute(ins,['霸王别姬','张国荣','1993'])

db.commit()
cursor.close()
db.close()
```

- **练习 - 将电影信息存入MySQL数据库**

```
【1】提前建库建表
mysql -h127.0.0.1 -uroot -p123456
create database maoyandb charset utf8;
use maoyandb;
create table maoyantab(
name varchar(100),
star varchar(300),
time varchar(100)
)charset=utf8;

【2】 使用excute()方法将数据存入数据库思路
    2.1) 在 __init__() 中连接数据库并创建游标对象
    2.2) 在 save_html() 中将所抓取的数据处理成列表，使用execute()方法写入
    2.3) 在run() 中等数据抓取完成后关闭游标及断开数据库连接
```

- **汽车之家二手车信息抓取**

```
【1】URL地址
    进入汽车之家官网，点击 二手车
    即：https://www.che168.com/beijing/a0_0msdgscncgpi1lto1cspexx0/

【2】抓取目标
    每辆汽车的
    2.1) 汽车名称
    2.2) 行驶里程
    2.3) 城市
    2.4) 个人还是商家
    2.5) 价格

【3】抓取前5页
```

- **参考答案**

```python
import requests
import re
import time
import random

```

```python
class CarSpider:
    def __init__(self):
        self.url = 'https://www.che168.com/beijing/a0_0msdgscncgpi1lto1csp{}exx0/?pvareaid=102179#currengpostion'
        self.headers = {'User-Agent':'Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/81.0.4044.138 Safari/537.36'}

    def get_html(self, url):
        html = requests.get(url=url, headers=self.headers).content.decode('gb2312', 'ignore')
        self.parse_html(html)

    def parse_html(self, html):
        pattern = re.compile('<li class="cards-li list-photo-li".*?<div class="cards-bottom">.*?<h4 class="card-name">(.*?)</h4>.*?<p class="cards-unit">(.*?)</p>.*?<span class="pirce"><em>(.*?)</em>', re.S)
        car_list = pattern.findall(html)
        self.save_html(car_list)

    def save_html(self, car_list):
        for car in car_list:
            print(car)

    def run(self):
        for i in range(1,6):
            page_url = self.url.format(i)
            self.get_html(page_url)
            time.sleep(random.randint(1,2))

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

请求模块(requests)

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

with open('xxx.txt','w',encoding='utf-8') as f:
    f.write(html)
```

解析模块(re)

- **使用流程**

```python
p = re.compile('正则表达式',re.S)
r_list = p.findall(html)
```

- **贪婪匹配和非贪婪匹配**

```
贪婪匹配(默认) :  .*
非贪婪匹配      :  .*?
```

- **正则表达式分组**

```
【1】想要什么内容在正则表达式中加()
【2】多个分组,先按整体正则匹配,然后再提取()中数据。结果:[(),(),(),(),()]
```

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
```

```python
# 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))
```

day02笔记

数据持久化 - MySQL

- **pymysql回顾**

```python
import pymysql

db =
pymysql.connect('localhost','root','123456','maoyandb',charset='utf8')
cursor = db.cursor()

ins = 'insert into filmtab values(%s,%s,%s)'
cursor.execute(ins,['霸王别姬','张国荣','1993'])

db.commit()
cursor.close()
db.close()
```

数据持久化 - MongoDB

- **MongoDB特点**

```
【1】非关系型数据库,数据以键值对方式存储，端口27017
【2】MongoDB基于磁盘存储
【3】MongoDB数据类型单一,值为JSON文档,而Redis基于内存,
   3.1> MySQL数据类型：数值类型、字符类型、日期时间类型、枚举类型
   3.2> Redis数据类型：字符串、列表、哈希、集合、有序集合
   3.3> MongoDB数据类型：值为JSON文档
【4】MongoDB： 库 -> 集合 -> 文档
     MySQL  : 库 -> 表  ->  表记录
```

- **MongoDB常用命令**

```
Linux进入: mongo
>show dbs                    - 查看所有库
>use 库名                    - 切换库
>show collections            - 查看当前库中所有集合
>db.集合名.find().pretty()   - 查看集合中文档
>db.集合名.count()           - 统计文档条数
>db.集合名.drop()            - 删除集合
>db.dropDatabase()           - 删除当前库
```

- **pymongo模块使用**

```python
import pymongo

# 1.连接对象
conn = pymongo.MongoClient(host = 'localhost',port = 27017)
# 2.库对象
db = conn['maoyandb']
# 3.集合对象
myset = db['maoyanset']
# 4.插入数据库
myset.insert_one({'name':'赵敏'})
```

- **练习 - 将电影信息存入MongoDB数据库**

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

```
存入mongodb数据库中
"""
import requests
import re
import time
import random
import pymongo

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) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/81.0.4044.113 Safari/537.36'}
        # 三个对象：连接对象、库对象、集合对象
        self.conn = pymongo.MongoClient('127.0.0.1', 27017)
        self.db = self.conn['maoyandb']
        self.myset = self.db['maoyanset2']

    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"""
        for r in r_list:
            item = {}
            item['name'] = r[0].strip()
            item['star'] = r[1].strip()
            item['time'] = r[2].strip()
            print(item)
            # 存入到mongodb数据库
            self.myset.insert_one(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()
```

数据持久化 - csv

- **csv描述**

```
【1】作用
   将爬取的数据存放到本地的csv文件中

【2】使用流程
    2.1> 打开csv文件
    2.2> 初始化写入对象
    2.3> 写入数据(参数为列表)

【3】示例代码
    import csv
    with open('sky.csv','w') as f:
        writer = csv.writer(f)
        writer.writerow([])
```

- **示例**

```
【1】题目描述
    创建 test.csv 文件，在文件中写入数据

【2】数据写入 - writerow([])方法
    import csv
    with open('test.csv','w') as f:  # with open('test.csv','w',newline='') as f:----->windows里面的写法，因为再wiondows中每条数据会有一个空行
        writer = csv.writer(f)
        writer.writerow(['超哥哥','25'])

```

- **练习 - 使用 writerow() 方法将猫眼电影数据存入本地 maoyan.csv 文件**

```
【1】在 __init__() 中打开csv文件，因为csv文件只需要打开和关闭1次即可
【2】在 save_html() 中将所抓取的数据处理成列表，使用writerow()方法写入
【3】在run() 中等数据抓取完成后关闭文件
```

- **代码实现**

```
"""
猫眼电影top100抓取（电影名称、主演、上映时间）
存入csv文件,使用writerow()方法
"""
import requests
import re
import time
import random
import csv

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)'}
```

```
        # 打开文件,初始化写入对象
        self.f = open('maoyan.csv', 'w', newline='', encoding='utf-8')
        self.writer = csv.writer(self.f)

    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"""
        for r in r_list:
            li = [ r[0].strip(), r[1].strip(), r[2].strip() ]
            self.writer.writerow(li)
            print(li)

    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(1,2))

        # 所有数据抓取并写入完成后关闭文件
        self.f.close()

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

useragent 池

```
sudo pip3 install fake_useragent        # 爬虫生成useragent的插件，大概有250个UserAgent
from fake_useragent import UserAgent

hearders = {'User-Agent': UserAgent().random}
```

汽车之家数据抓取 - 二级页面

- **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
```

```python
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)
```

```
70
71 if __name__ == '__main__':
72     spider = CarSpider()
73     spider.run()
```

- **练习 - 将数据存入MySQL数据库**

```
1  create database cardb charset utf8;
2  use cardb;
3  create table cartab(
4  name varchar(100),
5  km varchar(50),
6  years varchar(50),
7  type varchar(50),
8  displacement varchar(50),
9  city varchar(50),
10 price varchar(50)
11 )charset=utf8;
```

使用redis实现增量爬虫

```
1  """
2    提示: 使用redis中的集合,sadd()方法,添加成功返回1,否则返回0
3    请各位大佬忽略掉下面代码,自己独立实现
4  """
5
6  import requests
7  import re
8  import time
9  import random
10 import pymysql
11 from hashlib import md5
12 import sys
13 import redis
14
15
16 class CarSpider(object):
17     def __init__(self):
18         self.url =
'https://www.che168.com/beijing/a0_0msdgscncgpi1lto1csp{}exx0/'
19         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'}
20         self.db =
pymysql.connect('localhost','root','123456','cardb',charset='utf8')
21         self.cursor = self.db.cursor()
22         # 连接redis去重
23         self.r = redis.Redis(host='localhost',port=6379,db=0)
24
25     # 功能函数1 - 获取响应内容
26     def get_html(self,url):
27         html = requests.get(url=url,headers=self.headers).text
28
29         return html
30
31     # 功能函数2 - 正则解析
32     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()
```

```
86
87      def run(self):
88          for p in range(1,2):
89              url = self.url.format(p)
90              self.parse_html(url)
91
92          # 断开数据库链接
93          self.cursor.close()
94          self.db.close()
95
96  if __name__ == '__main__':
97      spider = CarSpider()
98      spider.run()
```

Chrome浏览器安装插件

- **安装方法**

```
【1】在线安装
    1.1> 下载插件 - google访问助手
    1.2> 安装插件 - google访问助手: Chrome浏览器-设置-更多工具-扩展程序-开发者模式-拖拽(解压后的插件)
    1.3> 在线安装其他插件 - 打开google访问助手 - google应用商店 - 搜索插件 - 添加即可

【2】离线安装
    2.1> 网上下载插件 - xxx.crx 重命名为 xxx.zip
    2.2> Chrome浏览器-设置-更多工具-扩展程序-开发者模式
    2.3> 拖拽 插件(或者解压后文件夹) 到浏览器中
    2.4> 重启浏览器，使插件生效
```

- **爬虫常用插件**

```
【1】google-access-helper : 谷歌访问助手,可访问 谷歌应用商店
【2】Xpath Helper: 轻松获取HTML元素的xPath路径
   打开/关闭: Ctrl + Shift + x
【3】JsonView: 格式化输出json格式数据
【4】Proxy SwitchyOmega: Chrome浏览器中的代理管理扩展程序
```

xpath解析

- **定义**

```
XPath即为XML路径语言，它是一种用来确定XML文档中某部分位置的语言，同样适用于HTML文档的检索
```

- **匹配演示 - 猫眼电影top100**

```
【1】查找所有的dd节点
    //dd
【2】获取所有电影的名称的a节点: 所有class属性值为name的a节点
    //p[@class="name"]/a
【3】获取dl节点下第2个dd节点的电影节点
    //dl[@class="board-wrapper"]/dd[2]
【4】获取所有电影详情页链接: 获取每个电影的a节点的href的属性值
    //p[@class="name"]/a/@href

【注意】
    1> 只要涉及到条件,加 [] : //dl[@class="xxx"]   //dl/dd[2]
    2> 只要获取属性值,加 @  : //dl[@class="xxx"]   //p/a/@href
```

- **选取节点**

```
【1】// : 从所有节点中查找（包括子节点和后代节点）
【2】@  : 获取属性值
  2.1> 使用场景1（属性值作为条件）
       //div[@class="movie-item-info"]
  2.2> 使用场景2（直接获取属性值）
       //div[@class="movie-item-info"]/a/img/@src

【3】练习 - 猫眼电影top100
  3.1> 匹配电影名称
      //div[@class="movie-item-info"]/p[1]/a/@title
  3.2> 匹配电影主演
      //div[@class="movie-item-info"]/p[2]/text()
  3.3> 匹配上映时间
      //div[@class="movie-item-info"]/p[3]/text()
  3.4> 匹配电影链接
      //div[@class="movie-item-info"]/p[1]/a/@href
```

- **匹配多路径（或）**

```
xpath表达式1 | xpath表达式2 | xpath表达式3
```

- **常用函数**

```
【1】text() : 获取节点的文本内容
    xpath表达式末尾不加 /text() :则得到的结果为节点对象
    xpath表达式末尾加 /text() 或者 /@href : 则得到结果为字符串

【2】contains() : 匹配属性值中包含某些字符串节点
    匹配class属性值中包含 'movie-item' 这个字符串的 div 节点
     //div[contains(@class,"movie-item")]
```

- **终极总结**

```
【1】字符串: xpath表达式的末尾为: /text() 、/@href  得到的列表中为'字符串'

【2】节点对象: 其他剩余所有情况得到的列表中均为'节点对象'
    [<element dd at xxxa>,<element dd at xxxb>,<element dd at xxxc>]
    [<element div at xxxa>,<element div at xxxb>]
    [<element p at xxxa>,<element p at xxxb>,<element p at xxxc>]
```

- **课堂练习**

```
【1】匹配汽车之家-二手车,所有汽车的链接 :
    //li[@class="cards-li list-photo-li"]/a[1]/@href
    //a[@class="carinfo"]/@href
【2】匹配汽车之家-汽车详情页中,汽车的
     2.1)名称:  //div[@class="car-box"]/h3/text()
     2.2)里程:  //ul/li[1]/h4/text()
     2.3)时间:  //ul/li[2]/h4/text()
     2.4)挡位+排量: //ul/li[3]/h4/text()
     2.5)所在地: //ul/li[4]/h4/text()
     2.6)价格:   //div[@class="brand-price-
item"]/span[@class="price"]/text()
```

lxml解析库

- **安装**

```
【1】Ubuntu:  sudo pip3 install lxml
【2】Windows: python -m pip install lxml
```

- **使用流程**

```
1、导模块
   from lxml import etree
2、创建解析对象
   parse_html = etree.HTML(html)
3、解析对象调用xpath
   r_list = parse_html.xpath('xpath表达式')
```

- **xpath最常用**

```
【1】基准xpath: 匹配所有电影信息的节点对象列表
   //dl[@class="board-wrapper"]/dd
   [<element dd at xxx>,<element dd at xxx>,...]

【2】遍历对象列表，依次获取每个电影信息
   item = {}
   for dd in dd_list:
        item['name'] = dd.xpath('.//p[@class="name"]/a/text()').strip()
        item['star'] = dd.xpath('.//p[@class="star"]/text()').strip()
        item['time'] =
dd.xpath('.//p[@class="releasetime"]/text()').strip()
```

- **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()
```

- **小作业**

```
汽车之家案例使用lxml+xpath实现
```

Day02回顾

数据抓取

- **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')
```

数据持久化

- **csv**

```
import csv
with open('xxx.csv','w',encoding='utf-8',newline='') as f:
   writer = csv.writer(f)
   writer.writerow([])
```

- **MySQL**

```
import pymysql

# __init__(self):
    self.db = pymysql.connect('IP',... ...)
    self.cursor = self.db.cursor()

# save_html(self,r_list):
    self.cursor.execute('sql',[data1])
    self.db.commit()

# run(self):
    self.cursor.close()
    self.db.close()
```

- **MongoDB**

```
import pymongo


# __init__(self):
    self.conn = pymongo.MongoClient('IP',27017)
    self.db = self.conn['cardb']
    self.myset = self.db['car_set']

# save_html(self,r_list):
    self.myset.insert_one(dict)

# MongoDB - Commmand - 库->集合->文档
mongo
>show dbs
>use db_name
```

```
>show collections
>db.集合名.find().pretty()
>db.集合名.count()
>db.集合名.drop()
>db.dropDatabase()
```

xpath表达式

- **匹配规则**

```
【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()
```

Day03笔记

- **xpath匹配汽车之家数据**

```
【1】匹配汽车之家-二手车,所有汽车的链接 ：
    //li[@class="cards-li list-photo-li"]/a[1]/@href
    //a[@class="carinfo"]/@href

【2】匹配汽车之家-汽车详情页中,汽车的
     2.1)名称：  //div[@class="car-box"]/h3/text()
     2.2)里程：  //ul/li[1]/h4/text()
     2.3)时间：  //ul/li[2]/h4/text()
     2.4)挡位+排量：//ul/li[3]/h4/text()
     2.5)所在地：//ul/li[4]/h4/text()
     2.6)价格：   //div[@class="brand-price-
item"]/span[@class="price"]/text()
```

lxml解析库

- **安装**

```
【1】Ubuntu：  sudo pip3 install lxml
【2】Windows：python -m pip install lxml
```

- **使用流程**

```
1、导模块
   from lxml import etree
2、创建解析对象
   parse_html = etree.HTML(html)
3、解析对象调用xpath
   r_list = parse_html.xpath('xpath表达式')
    # 只要调用 xpath 得到的返回值一定是列表：[] 或者 [<element div at xxxx>,
<xxxxxxxxx>] 或者 ['字符串','xxx']
```

- **xpath最常用**

```
【1】基准xpath: 匹配所有电影信息的节点对象列表
   //dl[@class="board-wrapper"]/dd
   [<element dd at xxx>,<element dd at xxx>,...]

【2】遍历对象列表，依次获取每个电影信息
   item = {}
   for dd in dd_list:
        item['name'] = dd.xpath('.//p[@class="name"]/a/text()').strip()
        item['star'] = dd.xpath('.//p[@class="star"]/text()').strip()
        item['time'] =
dd.xpath('.//p[@class="releasetime"]/text()').strip()
```

豆瓣图书信息抓取 - xpath

- **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()
```

链家二手房案例（xpath）

- **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
### 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()
```

- **持久化到数据库中 - 自己实现**

```
【1】将数据存入MongoDB数据库
【2】将数据存入MySQL数据库
```

代理参数-proxies

- **定义及分类**

```
【1】定义 ： 代替你原来的IP地址去对接网络的IP地址

【2】作用 ： 隐藏自身真实IP,避免被封
```

- **普通代理**

```
【1】获取代理IP网站
   西刺代理、快代理、全网代理、代理精灵、... ...

【2】参数类型
   proxies = { '协议':'协议://IP:端口号' }
   proxies = {
        'http':'http://IP:端口号',
        'https':'https://IP:端口号',
   }
```

- **普通代理 - 示例**

```
# 使用免费普通代理IP访问测试网站: http://httpbin.org/get
import requests

url = 'http://httpbin.org/get'
headers = {'User-Agent':'Mozilla/5.0'}
# 定义代理,在代理IP网站中查找免费代理IP
proxies = {
    'http':'http://112.85.164.220:9999',
    'https':'https://112.85.164.220:9999'
}
html = requests.get(url,proxies=proxies,headers=headers,timeout=5).text
print(html)
```

- **私密代理+独享代理**

```
【1】语法结构
   proxies = { '协议':'协议://用户名:密码@IP:端口号' }

【2】示例
   proxies = {
      'http':'http://用户名:密码@IP:端口号',
      'https':'https://用户名:密码@IP:端口号',
   }
```

- **私密代理+独享代理 - 示例代码**

```
import requests
url = 'http://httpbin.org/get'
proxies = {
    'http': 'http://309435365:szayclhp@106.75.71.140:16816',
    'https':'https://309435365:szayclhp@106.75.71.140:16816',
}
headers = {
    'User-Agent' : 'Mozilla/5.0',
}

html = requests.get(url,proxies=proxies,headers=headers,timeout=5).text
print(html)
```

- **建立自己的代理IP池 - 开放代理 | 私密代理**

```
"""
```

```
收费代理:
    建立开放代理的代理IP池
思路:
    1、获取到开放代理
    2、依次对每个代理IP进行测试,能用的保存到文件中
"""
import requests

class ProxyPool:
    def __init__(self):
        self.url = '代理网站的API链接'
        self.headers = {'User-Agent':'Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/81.0.4044.122 Safari/537.36'}
        # 打开文件,用来存放可用的代理IP
        self.f = open('proxy.txt', 'w')

    def get_html(self):
        html = requests.get(url=self.url, headers=self.headers).text
        proxy_list = html.split('\r\n')
        for proxy in proxy_list:
            # 依次测试每个代理IP是否可用
            if self.check_proxy(proxy):
                self.f.write(proxy + '\n')

    def check_proxy(self, proxy):
        """测试1个代理IP是否可用,可用返回True,否则返回False"""
        test_url = 'http://httpbin.org/get'
        proxies = {
            'http' : 'http://{}'.format(proxy),
            'https': 'https://{}'.format(proxy)
        }
        try:
            res = requests.get(url=test_url, proxies=proxies, headers=self.headers, timeout=2)
            if res.status_code == 200:
                print(proxy,'\033[31m可用\033[0m')
                return True
            else:
                print(proxy,'无效')
                return False
        except:
            print(proxy,'无效')
            return False

    def run(self):
        self.get_html()
        # 关闭文件
        self.f.close()

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

requests.get()

get()方法参数

```
【1】url
【2】proxies : 字典(普通代理|私密代理+独享代理)
【3】headers : 字典(User-Agent,Cookie,Referer,... ...)
【4】timeout : 数字(超过指定时间网站未响应则抛出异常)
【5】cookies : 字典
```

响应对象属性

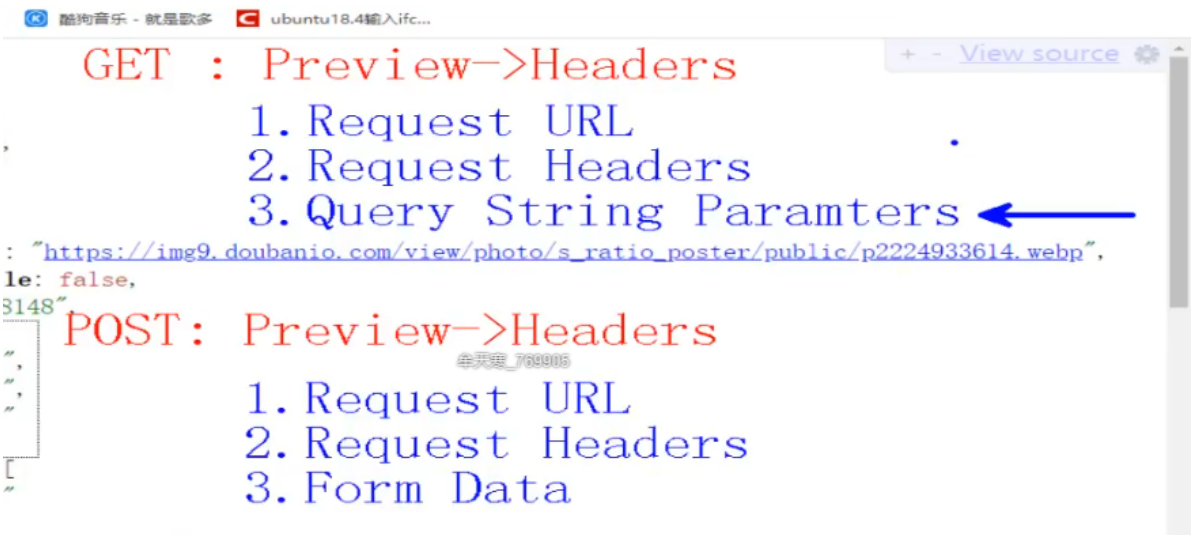
```
【1】text : 字符串
【2】content : 获取bytes数据类型
【3】status_code : HTTP响应码
【4】url : 返回实际数据的URL地址
```

requests.post()

- **适用场景**

```
【1】适用场景 : Post类型请求的网站

【2】参数 : data={}
   2.1) Form表单数据: 字典
   2.2) res = requests.post(url=url,data=data,headers=headers)

【3】POST请求特点 : Form表单提交数据


    # 请求头中最常见的需要检查的三个键
```

```
self.headers = {
    # 这个
    "Cookie": "OUTFOX_SEARCH_USER_ID=4
    "Referer": "http://fanyi.youdao.co
    "User-Agent": "Mozilla/5.0 (Windov
}
```



控制台抓包

- **打开方式及常用选项**

```
【1】打开浏览器，F12打开控制台，找到Network选项卡

post请求就找三个数据，url,headers,formdata

【2】控制台常用选项
   2.1) Network：抓取网络数据包
     a> ALL：抓取所有的网络数据包
     b> XHR：抓取异步加载的网络数据包
     c> JS ：抓取所有的JS文件
   2.2) Sources：格式化输出并打断点调试JavaScript代码，助于分析爬虫中一些参数
   2.3) Console：交互模式，可对JavaScript中的代码进行测试

【3】抓取具体网络数据包后
   3.1) 单击左侧网络数据包地址，进入数据包详情，查看右侧
   3.2) 右侧：
     a> Headers：整个请求信息
        General、Response Headers、Request Headers、Query String、Form Data
     b> Preview：对响应内容进行预览
     c> Response：响应内容
```

有道翻译破解案例(post)

- **目标**

```
破解有道翻译接口，抓取翻译结果
# 结果展示
请输入要翻译的词语：elephant
翻译结果：大象
**************************
请输入要翻译的词语：喵喵叫
翻译结果：mews
```

- **实现步骤**

```
【1】浏览器F12开启网络抓包,Network-All,页面翻译单词后找Form表单数据
【2】在页面中多翻译几个单词，观察Form表单数据变化（有数据是加密字符串）
【3】刷新有道翻译页面，抓取并分析JS代码（本地JS加密）
【4】找到JS加密算法，用Python按同样方式加密生成加密数据
【5】将Form表单数据处理为字典，通过requests.post()的data参数发送
```

- **具体实现**

1、开启F12抓包，找到Form表单数据如下：

```
i: 喵喵叫
from: AUTO
to: AUTO
smartresult: dict
client: fanyideskweb
salt: 15614112641250
sign: 94008208919faa19bd531acde36aac5d
ts: 1561411264125
```

```
bv: f4d62a2579ebb44874d7ef93ba47e822
doctype: json
version: 2.1
keyfrom: fanyi.web
action: FY_BY_REALTlME


    # 32位是MD5加密     40位是SHA1加密
```

2、在页面中多翻译几个单词，观察Form表单数据变化

```
salt: 15614112641250
sign: 94008208919faa19bd531acde36aac5d
ts: 1561411264125
bv: f4d62a2579ebb44874d7ef93ba47e822
# 但是bv的值不变
```

3、一般为本地js文件加密，刷新页面，找到js文件并分析JS代码

```
【方法1】 : Network - JS选项 - 搜索关键词salt
【方法2】 : 控制台右上角 - Search - 搜索salt - 查看文件 - 格式化输出

【结果】 : 最终找到相关JS文件 : fanyi.min.js
```

4、打开JS文件，分析加密算法，用Python实现

```
【ts】经过分析为13位的时间戳，字符串类型
   js代码实现)  "" + (new Date).getTime()
   python实现) str(int(time.time()*1000))

【salt】
   js代码实现)  ts + parseInt(10 * Math.random(), 10);
   python实现)  ts + str(random.randint(0,9))

【sign】（'设置断点调试，来查看 e 的值，发现 e 为要翻译的单词'）
   js代码实现) n.md5("fanyideskweb" + e + salt + "n%A-rKaT5fb[Gy?;N5@Tj")
   python实现)
   from hashlib import md5
   string = "fanyideskweb" + e + salt + "n%A-rKaT5fb[Gy?;N5@Tj"
   s = md5()
   s.update(string.encode())
   sign = s.hexdigest()
```

4、pycharm中正则处理headers和formdata

```
【1】pycharm进入方法 ：Ctrl + r ，选中 Regex
【2】处理headers和formdata
    (.*): (.*)
    "$1": "$2",
【3】点击 Replace All
```

5、代码实现

```python
import requests
import time
import random
from hashlib import md5

class YdSpider(object):
  def __init__(self):
    # url一定为F12抓到的 headers -> General -> Request URL
    self.url = 'http://fanyi.youdao.com/translate_o?smartresult=dict&smartresult=rule'
    self.headers = {
      # 检查频率最高 - 3个
      "Cookie": "OUTFOX_SEARCH_USER_ID=970246104@10.169.0.83; OUTFOX_SEARCH_USER_ID_NCOO=570559528.1224236; _ntes_nnid=96bc13a2f5ce64962adfd6a278467214,1551873108952; JSESSIONID=aaae9i7plXPlKaJH_gkYw; td_cookie=18446744072941336803; SESSION_FROM_COOKIE=unknown; ___rl__test__cookies=1565689460872",
      "Referer": "http://fanyi.youdao.com/",
      "User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/76.0.3809.100 Safari/537.36",
    }

  # 获取salt,sign,ts
  def get_salt_sign_ts(self,word):
    # ts
    ts = str(int(time.time()*1000))
    # salt
    salt = ts + str(random.randint(0,9))
    # sign
    string = "fanyideskweb" + word + salt + "n%A-rKaT5fb[Gy?;N5@Tj"
    s = md5()
    s.update(string.encode())
    sign = s.hexdigest()

    return salt,sign,ts

  # 主函数
  def attack_yd(self,word):
    # 1. 先拿到salt,sign,ts
    salt,sign,ts = self.get_salt_sign_ts(word)
    # 2. 定义form表单数据为字典: data={}
    # 检查了salt sign
    data = {
      "i": word,
      "from": "AUTO",
      "to": "AUTO",
      "smartresult": "dict",
      "client": "fanyideskweb",
      "salt": salt,
      "sign": sign,
      "ts": ts,
      "bv": "7e3150ecbdf9de52dc355751b074cf60",
      "doctype": "json",
      "version": "2.1",
      "keyfrom": "fanyi.web",
      "action": "FY_BY_REALTlME",
    }
    # 3. 直接发请求:requests.post(url,data=data,headers=xxx)
```

```
        html = requests.post(
            url=self.url,
            data=data,
            headers=self.headers
        ).json()
        # res.json() 将json格式的字符串转为python数据类型
        result = html['translateResult'][0][0]['tgt']

        print(result)

    # 主函数
    def run(self):
        # 输入翻译单词
        word = input('请输入要翻译的单词:')
        self.attack_yd(word)

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

今日作业

```
【1】将猫眼电影使用 lxml + xpath 实现
【2】将汽车之家案例使用 lxml + xpath 实现
【3】完善链家二手房案例，使用 lxml + xpath
【4】将 周测题 - 电影天堂案例 使用 lxml + xpath 实现
【5】将 周测题 - 4567TV案例 使用 lxml + xpath 实现
【6】抓取西刺免费高匿代理，并测试是否可用来建立自己的代理IP池
    https://www.xicidaili.com/nn/
【7】仔细熟悉有道翻译案例抓包及流程分析
```

Day03回顾

目前反爬总结

- **反爬虫梳理**

```
【1】Headers反爬虫
   1.1) 检查: Cookie、Referer、User-Agent
   1.2) 解决方案: 通过F12获取headers,传给requests.get()方法

【2】IP限制
   2.1) 网站根据IP地址访问频率进行反爬,短时间内限制IP访问
   2.2) 解决方案:
        a) 构造自己IP代理池,每次访问随机选择代理,经常更新代理池
        b) 购买开放代理或私密代理IP
        c) 降低爬取的速度

【3】User-Agent限制
   3.1) 类似于IP限制，检测频率
```

```
    3.2) 解决方案: 构造自己的User-Agent池,每次访问随机选择
         a> fake_useragent模块
         b> 新建py文件,存放大量User-Agent

【4】对响应内容做处理
    4.1) 页面结构和响应内容不同
    4.2) 解决方案: 打印并查看响应内容,用xpath或正则做处理
```

Day04笔记

动态加载数据抓取-Ajax

- **特点**

```
【1】右键 -> 查看网页源码中没有具体数据
【2】滚动鼠标滑轮或其他动作时加载,或者页面局部刷新
```

- **抓取**

```
【1】F12打开控制台，页面动作抓取网络数据包
【2】抓取json文件URL地址
   2.1) 控制台中 XHR : 异步加载的数据包
   2.2) XHR -> QueryStringParameters(查询参数)
```

豆瓣电影数据抓取案例

- **目标**

```
【1】地址: 豆瓣电影 - 排行榜 - 剧情
【2】目标: 电影名称、电影评分
```

- **F12抓包（XHR）**

```
【1】Request URL(基准URL地址) : https://movie.douban.com/j/chart/top_list?
【2】Query String(查询参数)
    # 抓取的查询参数如下:
    type: 13 # 电影类型
    interval_id: 100:90
    action: ''
    start: 0  # 每次加载电影的起始索引值 0 20 40 60
    limit: 20 # 每次加载的电影数量
```

- **代码实现 - 全站抓取**

```
"""
豆瓣电影 - 全站抓取
"""
import requests
from fake_useragent import UserAgent
import time
import random
import re
import json
```

```python

class DoubanSpider:
    def __init__(self):
        self.url = 'https://movie.douban.com/j/chart/top_list?'
        self.i = 0
        # 存入json文件
        self.f = open('douban.json', 'w', encoding='utf-8')
        self.all_film_list = []

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

    def get_html(self, params):
        headers = {'User-Agent':self.get_agent()}
        html = requests.get(url=self.url, params=params, headers=headers).text
        # 把json格式的字符串转为python数据类型
        html = json.loads(html)

        self.parse_html(html)

    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)
            self.all_film_list.append(item)
            self.i += 1

    def run(self):
        # d: {'剧情':'11','爱情':'13','喜剧':'5',...,...}
        d = self.get_d()
        # 1、给用户提示,让用户选择
        menu = ''
        for key in d:
            menu += key + '|'
        print(menu)
        choice = input('请输入电影类别: ')
        if choice in d:
            code = d[choice]
            # 2、total: 电影总数
            total = self.get_total(code)
            for start in range(0,total,20):
                params = {
                    'type': code,
                    'interval_id': '100:90',
                    'action': '',
                    'start': str(start),
                    'limit': '20'
                }
                self.get_html(params=params)
                time.sleep(random.randint(1,2))

```

```
            # 把数据存入json文件
            json.dump(self.all_film_list, self.f, ensure_ascii=False)
            self.f.close()
            print('数量:',self.i)
        else:
            print('请做出正确的选择')

    def get_d(self):
        """{'剧情':'11','爱情':'13','喜剧':'5',...,...}"""
        url = 'https://movie.douban.com/chart'
        html = requests.get(url=url,headers={'User-Agent':self.get_agent()}).text
        regex = '<span><a href=".*?type_name=(.*?)&type=(.*?)&interval_id=100:90&action=">'
        pattern = re.compile(regex, re.S)
        # r_list: [('剧情','11'),('喜剧','5'),('爱情':'13')... ...]
        r_list = pattern.findall(html)
        # d: {'剧情': '11', '爱情': '13', '喜剧': '5', ..., ...}
        d = {}
        for r in r_list:
            d[r[0]] = r[1]

        return d

    def get_total(self, code):
        """获取某个类别下的电影总数"""
        url = 'https://movie.douban.com/j/chart/top_list_count?type={}&interval_id=100%3A90'.format(code)
        html = requests.get(url=url,headers={'User-Agent':self.get_agent()}).text
        html = json.loads(html)

        return html['total']

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

json解析模块

- **json.loads(json)**

```
【1】作用 : 把json格式的字符串转为Python数据类型

【2】示例 : html = json.loads(res.text)
```

- **json.dump(python,f,ensure_ascii=False)**

```
【1】作用
   把python数据类型 转为 json格式的字符串,一般让你把抓取的数据保存为json文件时使用
    # 不加 S 是保存到文件之中，dumps 是在程序中使用
【2】参数说明
   2.1) 第1个参数: python类型的数据(字典，列表等)
   2.2) 第2个参数: 文件对象
   2.3) 第3个参数: ensure_ascii=False 序列化时编码,可以显示中文

```

```
【3】示例代码
    # 示例1
    import json

    item = {'name':'QQ','app_id':1}
    with open('小米.json','a') as f:
      json.dump(item,f,ensure_ascii=False)

    # 示例2
    import json

    item_list = []
    for i in range(3):
      item = {'name':'QQ','id':i}
      item_list.append(item)

    with open('xiaomi.json','a') as f:
        json.dump(item_list,f,ensure_ascii=False)
```

- **json模块总结**

```
# 爬虫最常用
【1】数据抓取 - json.loads(html)
    将响应内容由: json 转为 python
【2】数据保存 - json.dump(item_list,f,ensure_ascii=False)
    将抓取的数据保存到本地 json文件

# 抓取数据一般处理方式
【1】txt文件
【2】csv文件
【3】json文件
【4】MySQL数据库
【5】MongoDB数据库
【6】Redis数据库
```

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
```

知识点回顾

- **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))
```

selenium+PhantomJS/Chrome/Firefox

- **selenium**

```
【1】定义
    1.1) 开源的web自动化测试工具

【2】用途
    2.1) 对web系统进行功能性测试,版本迭代时避免重复劳动
    2.2) 兼容性测试(测试web程序在不同操作系统和不同浏览器中是否运行正常)
    2.3) 对web系统进行大数量测试

【3】特点
    3.1) 可根据指令操控浏览器
    3.2) 只是工具，必须与第三方浏览器结合使用

【4】安装
    4.1) Linux: sudo pip3 install selenium
    4.2) Windows: python -m pip install selenium
```

- **PhantomJS浏览器**

```
【1】定义
    phantomjs为无界面浏览器(又称无头浏览器)，在内存中进行页面加载,高效

【2】下载地址
    2.1) chromedriver ： 下载对应版本
       http://npm.taobao.org/mirrors/chromedriver/

    2.2) geckodriver
       https://github.com/mozilla/geckodriver/releases

    2.3) phantomjs
       https://phantomjs.org/download.html

【3】Ubuntu安装
    3.1) 下载后解压 ： tar -zxvf geckodriver.tar.gz

    3.2) 拷贝解压后文件到 /usr/bin/ （添加环境变量）
         sudo cp geckodriver /usr/bin/

    3.3) 添加可执行权限
         sudo chmod 777 /usr/bin/geckodriver

【4】Windows安装
    4.1) 下载对应版本的phantomjs、chromedriver、geckodriver
```

```
    4.2) 把chromedriver.exe拷贝到python安装目录的Scripts目录下(添加到系统环境变量)
         # 查看python安装路径: where python
    4.3) 验证
         cmd命令行: chromedriver

***************************总结***************************************
【1】解压 - 放到用户主目录(chromedriver、geckodriver、phantomjs)
【2】拷贝 - sudo cp /home/tarena/chromedriver /usr/bin/
【3】权限 - sudo chmod 777 /usr/bin/chromedriver

# 验证
【Ubuntu | Windows】
ipython3
from selenium import webdriver
webdriver.Chrome()
或者
webdriver.Firefox()

【mac】
ipython3
from selenium import webdriver
webdriver.Chrome(executable_path='/Users/xxx/chromedriver')
或者
webdriver.Firefox(executable_path='/User/xxx/geckodriver')
```

- **示例代码**

```
"""示例代码一：使用 selenium+浏览器 打开百度"""

# 导入seleinum的webdriver接口
from selenium import webdriver
import time

# 创建浏览器对象
browser = webdriver.Chrome()
browser.get('http://www.baidu.com/')
# 5秒钟后关闭浏览器
time.sleep(5)
browser.quit()
```

```
"""示例代码二：打开百度，搜索赵丽颖，点击搜索，查看"""

from selenium import webdriver
import time

# 1.创建浏览器对象 - 已经打开了浏览器
browser = webdriver.Chrome()
# 2.输入: http://www.baidu.com/
browser.get('http://www.baidu.com/')
# 3.找到搜索框,向这个节点发送文字: 赵丽颖
browser.find_element_by_xpath('//*[@id="kw"]').send_keys('赵丽颖')
# 4.找到 百度一下 按钮,点击一下
browser.find_element_by_xpath('//*[@id="su"]').click()
```

- **浏览器对象(browser)方法**

```
【1】browser.get(url=url)    - 地址栏输入url地址并确认
【2】browser.quit()          - 关闭浏览器
【3】browser.close()         - 关闭当前页
【4】browser.page_source     - HTML结构源码
【5】browser.page_source.find('字符串')
    从html源码中搜索指定字符串,没有找到返回: -1,经常用于判断是否为最后一页
【6】browser.maximize_window() - 浏览器窗口最大化
```

- **定位节点八种方法**

```
【1】单元素查找('结果为1个节点对象')
    1.1) 【最常用】browser.find_element_by_id('id属性值')
    1.2) 【最常用】browser.find_element_by_name('name属性值')
    1.3) 【最常用】browser.find_element_by_class_name('class属性值')
    1.4) 【最万能】browser.find_element_by_xpath('xpath表达式')
    1.5) 【匹配a节点时常用】browser.find_element_by_link_text('链接文本')
    1.6) 【匹配a节点时常用】browser.find_element_by_partical_link_text('部分链接文本')
    1.7) 【最没用】browser.find_element_by_tag_name('标记名称')
    1.8) 【较常用】browser.find_element_by_css_selector('css表达式')

【2】多元素查找('结果为[节点对象列表]')
    2.1) browser.find_elements_by_id('id属性值')
    2.2) browser.find_elements_by_name('name属性值')
    2.3) browser.find_elements_by_class_name('class属性值')
    2.4) browser.find_elements_by_xpath('xpath表达式')
    2.5) browser.find_elements_by_link_text('链接文本')
    2.6) browser.find_elements_by_partical_link_text('部分链接文本')
    2.7) browser.find_elements_by_tag_name('标记名称')
    2.8) browser.find_elements_by_css_selector('css表达式')
```

- **猫眼电影示例**

```python
from selenium import webdriver
import time

url = 'https://maoyan.com/board/4'
browser = webdriver.Chrome()
browser.get(url)

def get_data():
    # 基准xpath: [<selenium xxx li at xxx>,<selenium xxx li at>]
    li_list = browser.find_elements_by_xpath('//*[@id="app"]/div/div/div[1]/dl/dd')
    for li in li_list:
        item = {}
        # info_list: ['1', '霸王别姬', '主演：张国荣', '上映时间：1993-01-01', '9.5']
        info_list = li.text.split('\n')
        item['number'] = info_list[0]
        item['name'] = info_list[1]
        item['star'] = info_list[2]
        item['time'] = info_list[3]
        item['score'] = info_list[4]

        print(item)
```

```
22
23 while True:
24     get_data()
25     try:
26         browser.find_element_by_link_text('下一页').click()
27         time.sleep(2)
28     except Exception as e:
29         print('恭喜你!抓取结束')
30         browser.quit()
31         break
```

- **节点对象操作**

```
【1】文本框操作
    1.1) node.send_keys('')  - 向文本框发送内容
    1.2) node.clear()        - 清空文本
    1.3) node.get_attribute('value') - 获取文本内容

【2】按钮操作
    1.1) node.click()       - 点击
    1.2) node.is_enabled() - 判断按钮是否可用
    1.3) node.get_attribute('value') - 获取按钮文本
```

chromedriver设置无界面模式

```
from selenium import webdriver

options = webdriver.ChromeOptions()
# 添加无界面参数
options.add_argument('--headless')
browser = webdriver.Chrome(options=options)
```

selenium - 鼠标操作

```
from selenium import webdriver
# 导入鼠标事件类
from selenium.webdriver import ActionChains

driver = webdriver.Chrome()
driver.get('http://www.baidu.com/')

# 移动到 设置，perform()是真正执行操作，必须有
element = driver.find_element_by_xpath('//*[@id="u1"]/a[8]')
ActionChains(driver).move_to_element(element).perform()

# 单击，弹出的Ajax元素，根据链接节点的文本内容查找
driver.find_element_by_link_text('高级搜索').click()
```

今日作业

```
1
```

Day04回顾

- **requests.get()参数**

```
【1】url
【2】proxies -> {}
    proxies = {
       'http':'http://1.1.1.1:8888',
       'https':'https://1.1.1.1:8888'
    }
【3】timeout
【4】headers
```

- **requests.post()**

```
data ： 字典，Form表单数据
```

- **常见的反爬机制及处理方式**

```
【1】Headers反爬虫
   1.1) 检查：Cookie、Referer、User-Agent
   1.2) 解决方案：通过F12获取headers,传给requests.get()方法

【2】IP限制
   2.1) 网站根据IP地址访问频率进行反爬,短时间内限制IP访问
   2.2) 解决方案：
        a) 构造自己IP代理池,每次访问随机选择代理,经常更新代理池
        b) 购买开放代理或私密代理IP
        c) 降低爬取的速度

【3】User-Agent限制
   3.1) 类似于IP限制，检测频率
   3.2) 解决方案：构造自己的User-Agent池,每次访问随机选择
        a> fake_useragent模块
        b> 新建py文件,存放大量User-Agent
        c> 程序中定义列表,存放大量的User-Agent

【4】对响应内容做处理
   4.1) 页面结构和响应内容不同
   4.2) 解决方案：打印并查看响应内容,用xpath或正则做处理

【5】JS加密
   5.1) 抓取到对应的JS文件,寻找加密算法
   5.2) 用Python实现加密算法,生成指定的参数
```

- **有道翻译过程梳理**

```
【1】打开首页

【2】准备抓包：F12开启控制台

```

```
【3】寻找地址
   3.1) 页面中输入翻译单词，控制台中抓取到网络数据包，查找并分析返回翻译数据的地址
        F12-Network-XHR-Headers-Grneral-Request URL

【4】发现规律
   4.1) 找到返回具体数据的地址，在页面中多输入几个单词，找到对应URL地址
   4.2) 分析对比 Network - All(或者XHR) - Form Data，发现对应的规律

【5】寻找JS加密文件
   5.1) 控制台右上角 ...->Search->搜索关键字->单击->跳转到Sources，左下角格式化符号{}

【6】查看JS代码
   6.1) 搜索关键字，找到相关加密方法，用python实现加密算法

【7】断点调试
   7.1) JS代码中部分参数不清楚可通过断点调试来分析查看

【8】完善程序
```

- **Ajax动态加载数据抓取流程**

```
【1】F12打开控制台，执行页面动作抓取网络数据包

【2】抓取json文件URL地址
   2.1) 控制台中 XHR ：找到异步加载的数据包
   2.2) GET请求: Network -> XHR -> URL 和 Query String Parameters(查询参数)
   2.3) POST请求:Network -> XHR -> URL 和 Form Data
```

- **json模块**

```
【1】抓取的json数据转为python数据类型
   1.1) html = json.loads('[{},{},{}]')
   1.2) html = requests.get(url=url,headers=headers).json()
   1.3) html = requests.post(url=url,data=data,headers=headers).json()

【2】抓取数据保存到json文件
   import json
   with open('xxx.json','w') as f:
       json.dump([{},{},{}],f,ensure_ascii=False)
```

- **数据抓取最终梳理**

```
【1】响应内容中存在
   1.1) 确认抓取数据在响应内容中是否存在

   1.2) 分析页面结构，观察URL地址规律
        a) 大体查看响应内容结构,查看是否有更改 -- (百度视频案例)
        b) 查看页面跳转时URL地址变化,查看是否新跳转 -- (民政部案例)

   1.3) 开始码代码进行数据抓取

【2】响应内容中不存在
   2.1) 确认抓取数据在响应内容中是否存在

   2.2) F12抓包,开始刷新页面或执行某些行为,主要查看XHR异步加载数据包
```

```
        a) GET请求: Request URL、Request Headers、Query String Paramters
        b) POST请求:Request URL、Request Headers、FormData

    2.3) 观察查询参数或者Form表单数据规律,如果需要进行进一步抓包分析处理
        a) 比如有道翻译的 salt+sign,抓取并分析JS做进一步处理
        b) 此处注意请求头中的Cookie和Referer以及User-Agent

    2.4) 使用res.json()获取数据,利用列表或者字典的方法获取所需数据
```

- **多线程爬虫梳理**

```
【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()
```

Day05笔记

selenium+PhantomJS/Chrome/Firefox

- **selenium**

```
【1】定义
    1.1) 开源的Web自动化测试工具

【2】用途
    2.1) 对Web系统进行功能性测试,版本迭代时避免重复劳动
    2.2) 兼容性测试(测试web程序在不同操作系统和不同浏览器中是否运行正常)
    2.3) 对web系统进行大数量测试

【3】特点
    3.1) 可根据指令操控浏览器
    3.2) 只是工具，必须与第三方浏览器结合使用

【4】安装
    4.1) Linux: sudo pip3 install selenium
    4.2) Windows: python -m pip install selenium
```

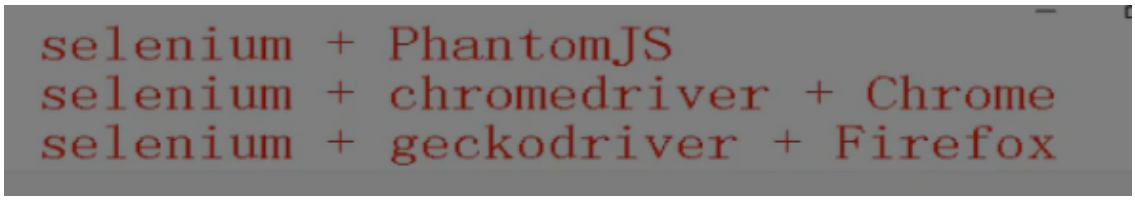
- **PhantomJS浏览器**

```
phantomjs为无界面浏览器(又称无头浏览器)，在内存中进行页面加载,高效
```

- **环境安装**

```
【1】下载驱动
    2.1) chromedriver : 下载对应版本
         http://npm.taobao.org/mirrors/chromedriver/
    2.2) geckodriver
         https://github.com/mozilla/geckodriver/releases
    2.3) phantomjs
         https://phantomjs.org/download.html

【2】添加到系统环境变量
    2.1) windows: 拷贝到Python安装目录的Scripts目录中
         windows查看python安装目录(cmd命令行): where python
    2.2) Linux :  拷贝到/usr/bin目录中 : sudo cp chromedriver /usr/bin/

【3】Linux中需要修改权限
    sudo chmod 777 /usr/bin/chromedriver

【4】验证
    4.1) Ubuntu | Windows
        from selenium import webdriver
        webdriver.Chrome()
        webdriver.Firefox()

    4.2) Mac
```

```
        from selenium import webdriver
        webdriver.Chrome(executable_path='/Users/xxx/chromedriver')
        webdriver.Firefox(executable_path='/User/xxx/geckodriver')
```



- **示例代码**

```
"""示例代码一：使用 selenium+浏览器 打开百度"""

# 导入seleinum的webdriver接口
from selenium import webdriver
import time

# 创建浏览器对象
browser = webdriver.Chrome()
browser.get('http://www.baidu.com/')
# 5秒钟后关闭浏览器
time.sleep(5)
browser.quit()
```

```
"""示例代码二：打开百度，搜索赵丽颖，点击搜索，查看"""

from selenium import webdriver
import time

# 1.创建浏览器对象 - 已经打开了浏览器
browser = webdriver.Chrome()
# 2.输入：http://www.baidu.com/
browser.get('http://www.baidu.com/')
# 3.找到搜索框,向这个节点发送文字：赵丽颖
browser.find_element_by_xpath('//*[@id="kw"]').send_keys('赵丽颖')
# 4.找到 百度一下 按钮,点击一下
browser.find_element_by_xpath('//*[@id="su"]').click()
```

- **浏览器对象(browser)方法**

```
【1】browser.get(url=url)    - 地址栏输入url地址并确认
【2】browser.quit()          - 关闭浏览器
【3】browser.close()         - 关闭当前页
【4】browser.page_source     - HTML结构源码
【5】browser.page_source.find('字符串')
    # 从html源码中搜索指定字符串,没有找到返回：-1,经常用于判断是否为最后一页
【6】browser.maximize_window() - 浏览器窗口最大化
```

- **定位节点八种方法**

```
【1】单元素查找('结果为1个节点对象')
    1.1) 【最常用】browser.find_element_by_id('id属性值')
    1.2) 【最常用】browser.find_element_by_name('name属性值')
    1.3) 【最常用】browser.find_element_by_class_name('class属性值')
```

```
    1.4) 【最万能】browser.find_element_by_xpath('xpath表达式')
    1.5) 【匹配a节点时常用】browser.find_element_by_link_text('链接文本')
    1.6) 【匹配a节点时常用】browser.find_element_by_partical_link_text('部分链接文本')
    1.7) 【最没用】browser.find_element_by_tag_name('标记名称')
    1.8) 【较常用】browser.find_element_by_css_selector('css表达式')

【2】多元素查找('结果为[节点对象列表]')
    2.1) browser.find_elements_by_id('id属性值')
    2.2) browser.find_elements_by_name('name属性值')
    2.3) browser.find_elements_by_class_name('class属性值')
    2.4) browser.find_elements_by_xpath('xpath表达式')
    2.5) browser.find_elements_by_link_text('链接文本')
    2.6) browser.find_elements_by_partical_link_text('部分链接文本')
    2.7) browser.find_elements_by_tag_name('标记名称')
    2.8) browser.find_elements_by_css_selector('css表达式')

【3】注意！！！
    当属性值中存在 空格 时,我们要使用 . 去代替空格
    页面中class属性值为: btn btn-account
    driver.find_element_by_class_name('btn.btn-account').click()

【4】如果找不到节点，会抛出异常
```

- **猫眼电影示例**

```python
from selenium import webdriver
import time

url = 'https://maoyan.com/board/4'
browser = webdriver.Chrome()
browser.get(url)

def get_data():
    # 基准xpath: [<selenium xxx li at xxx>,<selenium xxx li at>]
    li_list = browser.find_elements_by_xpath('//*[@id="app"]/div/div/div[1]/dl/dd')
    for li in li_list:
        item = {}
        # info_list: ['1', '霸王别姬', '主演：张国荣', '上映时间：1993-01-01', '9.5']
        info_list = li.text.split('\n')
        item['number'] = info_list[0]
        item['name'] = info_list[1]
        item['star'] = info_list[2]
        item['time'] = info_list[3]
        item['score'] = info_list[4]

        print(item)

while True:
    get_data()
    try:
        browser.find_element_by_link_text('下一页').click()
        time.sleep(2)
    except Exception as e:
```

```
        print('恭喜你!抓取结束')
        browser.quit()
        break
```

- **节点对象操作**

```
【1】文本框操作
    1.1) node.send_keys('')  - 向文本框发送内容
    1.2) node.clear()        - 清空文本
    1.3) node.get_attribute('value') - 获取文本内容

【2】按钮操作   只针对butten按钮
    1.1) node.click()       - 点击
    1.2) node.is_enabled() - 判断按钮是否可用
    1.3) node.get_attribute('value') - 获取按钮文本
```

chromedriver设置无头模式

```
from selenium import webdriver

options = webdriver.ChromeOptions()
# 添加无界面参数
options.add_argument('--headless')
browser = webdriver.Chrome(options=options)
```

selenium - 鼠标操作

```
from selenium import webdriver
# 导入鼠标事件类
from selenium.webdriver import ActionChains

driver = webdriver.Chrome()
driver.get('http://www.baidu.com/')

# 移动到 设置，perform()是真正执行操作，必须有
element = driver.find_element_by_xpath('//*[@id="u1"]/a[8]')
ActionChains(driver).move_to_element(element).perform()

# 单击，弹出的Ajax元素，根据链接节点的文本内容查找
driver.find_element_by_link_text('高级搜索').click()
```

selenium - 切换页面

- **适用网站+应对方案**

```
【1】适用网站类型
    页面中点开链接出现新的窗口，但是浏览器对象browser还是之前页面的对象，需要切换到不同的窗口进行操作

【2】应对方案 - browser.switch_to.window()

    # 获取当前所有句柄（窗口）- [handle1,handle2]
    all_handles = browser.window_handles
    # 切换browser到新的窗口，获取新窗口的对象
    browser.switch_to.window(all_handles[1])
```

- **民政部网站案例-selenium**

```
"""
适用selenium+Chrome抓取民政部行政区划代码
http://www.mca.gov.cn/article/sj/xzqh/2019/
"""
from selenium import webdriver

class GovSpider(object):
    def __init__(self):
        # 设置无界面
        options = webdriver.ChromeOptions()
        options.add_argument('--headless')
        # 添加参数
        self.browser = webdriver.Chrome(options=options)
        self.one_url = 'http://www.mca.gov.cn/article/sj/xzqh/2019/'

    def get_incr_url(self):
        self.browser.get(self.one_url)
        # 提取最新链接节点对象并点击

 self.browser.find_element_by_xpath('//td[@class="arlisttd"]/a[contains(@title,"代码")]').click()
        # 切换句柄
        all_handlers = self.browser.window_handles
        self.browser.switch_to.window(all_handlers[1])
        self.get_data()

    def get_data(self):
        tr_list = self.browser.find_elements_by_xpath('//tr[@height="19"]')
        for tr in tr_list:
            code = tr.find_element_by_xpath('./td[2]').text.strip()
            name = tr.find_element_by_xpath('./td[3]').text.strip()
            print(name,code)

    def run(self):
        self.get_incr_url()
        self.browser.quit()

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

selenium - iframe

- **特点+方法**

```
【1】特点
    网页中嵌套了网页，先切换到iframe，然后再执行其他操作

【2】处理步骤
    2.1) 切换到要处理的Frame
    2.2) 在Frame中定位页面元素并进行操作
    2.3) 返回当前处理的Frame的上一级页面或主页面

【3】常用方法
    3.1) 切换到frame  -  browser.switch_to.frame(frame节点对象)
    3.2) 返回上一级   -  browser.switch_to.parent_frame()
    3.3) 返回主页面   -  browser.switch_to.default_content()

【4】使用说明
    4.1) 方法一: 默认支持id和name属性值 : switch_to.frame(id属性值|name属性值)
    4.2) 方法二:
        a> 先找到frame节点 : frame_node = browser.find_element_by_xpath('xxxx')
        b> 在切换到frame   : browser.switch_to.frame(frame_node)
```

- **示例1 - 登录豆瓣网**

```
"""
登录豆瓣网
"""
from selenium import webdriver
import time

# 打开豆瓣官网
browser = webdriver.Chrome()
browser.get('https://www.douban.com/')

# 切换到iframe子页面
login_frame = browser.find_element_by_xpath('//*[@id="anony-reg-new"]/div/div[1]/iframe')
browser.switch_to.frame(login_frame)

# 密码登录 + 用户名 + 密码 + 登录豆瓣
browser.find_element_by_xpath('/html/body/div[1]/div[1]/ul[1]/li[2]').click()
browser.find_element_by_xpath('//*[@id="username"]').send_keys('自己的用户名')
browser.find_element_by_xpath('//*[@id="password"]').send_keys('自己的密码')
browser.find_element_by_xpath('/html/body/div[1]/div[2]/div[1]/div[5]/a').click()
time.sleep(3)

# 点击我的豆瓣
browser.find_element_by_xpath('//*[@id="db-nav-sns"]/div/div/div[3]/ul/li[2]/a').click()
```

- **selenium+phantomjs|chrome|firefox小总结**

```
【1】设置无界面模式
    options = webdriver.ChromeOptions()
    options.add_argument('--headless')
    browser = webdriver.Chrome(excutable_path='/home/tarena/chromedriver',options=options)

【2】browser执行JS脚本

 browser.execute_script('window.scrollTo(0,document.body.scrollHeight)')

【3】键盘操作
    from selenium.webdriver.common.keys import Keys

【4】鼠标操作
    from selenium.webdriver import ActionChains
    ActionChains(browser).move_to_element('node').perform()

【5】切换句柄 - switch_to.frame(handle)
    all_handles = browser.window_handles
    browser.switch_to.window(all_handles[1])
    # 开始进行数据抓取
    browser.close()
    browser.switch_to.window(all_handles[0])

【6】iframe子页面
    browser.switch_to.frame(frame_node)
```

- **lxml中的xpath 和 selenium中的xpath的区别**

```
【1】lxml中的xpath用法 - 推荐自己手写
    div_list = p.xpath('//div[@class="abc"]/div')
    item = {}
    for div in div_list:
        item['name'] = div.xpath('.//a/@href')[0]
        item['likes'] = div.xpath('.//a/text()')[0]

【2】selenium中的xpath用法 - 推荐copy - copy xpath
    div_list = browser.find_elements_by_xpath('//div[@class="abc"]/div')
    item = {}
    for div in div_list:
        item['name'] = div.find_element_by_xpath('.//a').get_attribute('href')
        item['likes'] = div.find_element_by_xpath('.//a').text
```

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/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
```

今日作业

```
【1】使用selenium+浏览器 获取有道翻译结果
【2】使用selenium+浏览器 登录网易qq邮箱 : https://mail.qq.com/
【3】使用selenium+浏览器 登录网易163邮箱 : https://mail.163.com/
【4】熟记scrapy的五大组件,以及工作流程,能够描述的很清楚
```

Day05回顾

selenium+phantomjs/chrome/firefox

- **特点**

```
【1】简单，无需去详细抓取分析网络数据包，使用真实浏览器
【2】需要等待页面元素加载，需要时间，效率低
```

- **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)'
)
# 把下拉条从上拉到下
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')
```

Day06笔记

作业概解

作业1 - 有道翻译实现

- **代码实现**

```
"""
selenium实现抓取有道翻译结果
思路:
    1、找到输入翻译单词节点,发送文字
    2、休眠一定时间,等待网站给出响应-翻译结果
    3、找到翻译结果节点,获取文本内容
"""
from selenium import webdriver
import time

class YdSpider:
    def __init__(self):
        self.url = 'http://fanyi.youdao.com/'
```

```
        # 设置无界面模式
        self.options = webdriver.ChromeOptions()
        self.options.add_argument('--headless')
        self.driver = webdriver.Chrome(options=self.options)
        # 打开有道翻译官网
        self.driver.get(self.url)

    def parse_html(self, word):
        # 发送翻译单词
        self.driver.find_element_by_id('inputOriginal').send_keys(word)
        time.sleep(1)
        # 获取翻译结果
        result = self.driver.find_element_by_xpath('//*
[@id="transTarget"]/p/span').text

        return result

    def run(self):
        word = input('请输入要翻译的单词:')
        print(self.parse_html(word))
        self.driver.quit()

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

作业2 - 登录QQ邮箱

- **代码实现**

```
from selenium import webdriver
import time

driver = webdriver.Chrome()
driver.get('https://mail.qq.com/')

# 切换到iframe子框架
driver.switch_to.frame("login_frame")

# 用户名+密码+登录
driver.find_element_by_id('u').send_keys('2621470058')
driver.find_element_by_id('p').send_keys('zhanshen001')
driver.find_element_by_id('login_button').click()
```

作业3- 163邮箱登陆

- **代码实现**

```
"""
selenium模拟登录163邮箱
思路:
    1、密码登录在这里 - 此节点在主页面中,并非iframe内部
    2、切换iframe - 此处iframe节点中id的值每次都在变化,需要手写xpath,否则会出现
无法定位iframe
    3、输入用户名和密码
    4、点击登录按钮
```

```
"""
from selenium import webdriver

driver = webdriver.Chrome()
driver.get('https://mail.163.com/')

# 1、切换iframe子页面 - 此处手写xpath,此处iframe中id的值每次都在变化
node = driver.find_element_by_xpath('//div[@id="loginDiv"]/iframe[1]')
driver.switch_to.frame(node)

# 2、输入用户名和密码
driver.find_element_by_name('email').send_keys('wangweichao_2020')
driver.find_element_by_name('password').send_keys('zhanshen001')
driver.find_element_by_id('dologin').click()
```

作业4 - 京东爬虫

- **目标**

```
【1】目标网址 ：https://www.jd.com/
【2】抓取目标 ：商品名称、商品价格、评价数量、商品商家
```

- **思路提醒**

```
【1】打开京东，到商品搜索页
【2】匹配所有商品节点对象列表
【3】把节点对象的文本内容取出来，查看规律，是否有更好的处理办法？
【4】提取完1页后，判断如果不是最后1页，则点击下一页
    '问题：如何判断是否为最后1页？？？'
```

- **实现步骤**

```
【1】找节点
    1.1) 首页搜索框 ： //*[@id="key"]
    2.1) 首页搜索按钮    ://*[@id="search"]/div/div[2]/button
    2.3) 商品页的 商品信息节点对象列表 ://*[@id="J_goodsList"]/ul/li
    2.4) for循环遍历后
        a> 名称：.//div[@class="p-name"]/a/em
        b> 价格：.//div[@class="p-price"]
        c> 评论：.//div[@class="p-commit"]/strong
        d> 商家：.//div[@class="p-shopnum"]

【2】执行JS脚本，获取动态加载数据
    browser.execute_script(
      'window.scrollTo(0,document.body.scrollHeight)'
    )
```

- **代码实现**

```
from selenium import webdriver
import time

class JdSpider(object):
    def __init__(self):
        self.url = 'https://www.jd.com/'
```

```
        # 设置无界面模式
        self.options = webdriver.ChromeOptions()
        self.options.add_argument('--headless')
        self.browser = webdriver.Chrome(options=self.options)

    def get_html(self):
        # get():等页面所有元素加载完成后,才会执行后面的代码
        self.browser.get(self.url)
        # 搜索框 + 搜索按钮
        self.browser.find_element_by_xpath('//*[@id="key"]').send_keys('爬虫书')
        self.browser.find_element_by_xpath('//*[@id="search"]/div/div[2]/button').click()

    # 循环体中的函数: 拉进度条,提取数据
    def parse_html(self):
        # 执行js脚本,将进度条拉到最底部
        self.browser.execute_script(
            'window.scrollTo(0,document.body.scrollHeight)'
        )
        # 给页面元素加载预留时间
        time.sleep(3)
        li_list = self.browser.find_elements_by_xpath('//*[@id="J_goodsList"]/ul/li')

        for li in li_list:
            item = {}
            item['price'] = li.find_element_by_xpath('.//div[@class="p-price"]').text
            item['mame'] = li.find_element_by_xpath('.//div[@class="p-name"]/a/em').text
            item['commit'] = li.find_element_by_xpath('.//div[@class="p-commit"]/strong').text
            item['shop'] = li.find_element_by_xpath('.//div[@class="p-shopnum"]').text
            print(item)

    def run(self):
        self.get_html()
        while True:
            self.parse_html()
            if self.browser.page_source.find('pn-next disabled') == -1:
                self.browser.find_element_by_xpath('//*[@id="J_bottomPage"]/span[1]/a[9]').click()
            else:
                self.browser.quit()
                break

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

scrapy框架

- **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）
        引擎->下载器,包装请求(随机代理等)
    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
    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
```

- **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)
```

- **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/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）**

```
"""
    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()


# 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 = 1
【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",
  }

# 优先级，1--1000，数字越小越高
【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 car'.split())
```

知识点汇总

- **数据持久化 - 数据库**

```
【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
```

- **数据持久化 - csv、json文件**

```
【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'
```

- **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'
```

- **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) 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,
}
```

今日作业

```
【1】腾讯招聘职位信息抓取（二级页面）
    要求：输入职位关键字，抓取该类别下所有职位信息（到职位详情页抓取）
    具体数据如下：
    1.1) 职位名称
    1.2) 职位地点
    1.3) 职位类别
    1.4) 发布时间
    1.5) 工作职责
    1.6) 工作要求
```

Day06回顾

scrapy框架

- **五大组件+工作流程+常用命令**

```
【1】五大组件
    1.1) 引擎（Engine）
    1.2) 爬虫程序（Spider）
    1.3) 调度器（Scheduler）
    1.4) 下载器（Downloader）
    1.5) 管道文件（Pipeline）
    1.6) 下载器中间件（Downloader Middlewares）
    1.7) 蜘蛛中间件（Spider Middlewares）

【2】工作流程
    2.1) 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
    2.3) Downloader得到响应后,通过Spider Middlewares交给Spider
    2.4) Spider数据提取：
        a) 数据交给Pipeline处理
        b) 需要跟进URL,继续交给Scheduler入队列，依次循环

【3】常用命令
    3.1) scrapy startproject 项目名
    3.2) scrapy genspider 爬虫名 域名
    3.3) scrapy crawl 爬虫名
```

完成scrapy项目完整流程

- **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'
```

Day07笔记

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;
"""
```

机器视觉与tesseract

- **概述**

```
【1】作用
    处理图形验证码

【2】三个重要概念 - OCR、tesseract-ocr、pytesseract
    2.1) OCR
        光学字符识别(Optical Character Recognition),通过扫描等光学输入方式将各种票据、报刊、书籍、文稿及其它印刷品的文字转化为图像信息，再利用文字识别技术将图像信息转化为电子文本

    2.2) tesseract-ocr
        OCR的一个底层识别库（不是模块，不能导入），由Google维护的开源OCR识别库

    2.3) pytesseract
        Python模块,可调用底层识别库，是对tesseract-ocr做的一层Python API封装
```

- **安装tesseract-ocr**

```
【1】Ubuntu安装
    sudo apt-get install tesseract-ocr

【2】Windows安装
    2.1) 下载安装包
    2.2) 添加到环境变量(Path)

【3】测试（终端 | cmd命令行）
    tesseract xxx.jpg 文件名
```

- **安装pytesseract**

```
【1】安装
    sudo pip3 install pytesseract

【2】使用示例
    import pytesseract
    # Python图片处理库
    from PIL import Image

    # 创建图片对象
    img = Image.open('test1.jpg')
    # 图片转字符串
    result = pytesseract.image_to_string(img)
    print(result)
```

补充 - 滑块缺口验证码案例

豆瓣网登录

- **案例说明**

```
【1】URL地址: https://www.douban.com/
【2】先输入几次错误的密码，让登录出现滑块缺口验证，以便于我们破解
【3】模拟人的行为
    3.1) 先快速滑动
    3.2) 到离重点位置不远的地方开始减速
【4】详细看代码注释
```

- **代码实现**

```
"""
说明：先输入几次错误的密码，出现滑块缺口验证码
"""
from selenium import webdriver
# 导入鼠标事件类
from selenium.webdriver import ActionChains
import time

# 加速度函数
def get_tracks(distance):
    """
    拿到移动轨迹，模仿人的滑动行为，先匀加速后匀减速
    匀变速运动基本公式:
    ①v=v0+at
    ②s=v0t+½at²
    """
    # 初速度
    v = 0
    # 单位时间为0.3s来统计轨迹，轨迹即0.3内的位移
    t = 0.3
    # 位置/轨迹列表,列表内的一个元素代表0.3s的位移
    tracks = []
    # 当前的位移
    current = 0
    # 到达mid值开始减速
    mid = distance*4/5
    while current < distance:
        if current < mid:
            # 加速度越小,单位时间内的位移越小,模拟的轨迹就越多越详细
            a = 2
        else:
            a = -3

        # 初速度
        v0 = v
        # 0.3秒内的位移
        s = v0*t+0.5*a*(t**2)
        # 当前的位置
        current += s
        # 添加到轨迹列表
        tracks.append(round(s))
        # 速度已经达到v，该速度作为下次的初速度
        v = v0 + a*t
```

```
    return tracks
    # tracks: [第一个0.3秒的移动距离,第二个0.3秒的移动距离,...]


# 1、打开豆瓣官网 - 并将窗口最大化
browser = webdriver.Chrome()
browser.maximize_window()
browser.get('https://www.douban.com/')

# 2、切换到iframe子页面
login_frame = browser.find_element_by_xpath('//*[@id="anony-reg-new"]/div/div[1]/iframe')
browser.switch_to.frame(login_frame)

# 3、密码登录 + 用户名 + 密码 + 登录豆瓣
browser.find_element_by_xpath('/html/body/div[1]/div[1]/ul[1]/li[2]').click()
browser.find_element_by_xpath('//*[@id="username"]').send_keys('15110225726')
browser.find_element_by_xpath('//*[@id="password"]').send_keys('zhanshen001')
browser.find_element_by_xpath('/html/body/div[1]/div[2]/div[1]/div[5]/a').click()
time.sleep(4)

# 4、切换到新的iframe子页面 - 滑块验证
auth_frame = browser.find_element_by_xpath('//*[@id="TCaptcha"]/iframe')
browser.switch_to.frame(auth_frame)

# 5、按住开始滑动位置按钮 - 先移动180个像素
element = browser.find_element_by_xpath('//*[@id="tcaptcha_drag_button"]')
# click_and_hold(): 按住某个节点并保持
ActionChains(browser).click_and_hold(on_element=element).perform()
# move_to_element_with_offset(): 移动到距离某个元素(左上角坐标)多少距离的位置
ActionChains(browser).move_to_element_with_offset(to_element=element,xoffset=180,yoffset=0).perform()

# 6、使用加速度函数移动剩下的距离
tracks = get_tracks(28)
for track in tracks:
    # move_by_offset() : 鼠标从当前位置移动到某个坐标

 ActionChains(browser).move_by_offset(xoffset=track,yoffset=0).perform()

# 7、延迟释放鼠标: release()
time.sleep(0.5)
ActionChains(browser).release().perform()
```

Fiddler抓包工具

- **配置Fiddler**

```
【1】Tools -> Options -> HTTPS
    1.1) 添加证书信任:  勾选 Decrypt Https Traffic 后弹出窗口，一路确认
    1.2) 设置之抓浏览器的包:  ...from browsers only

【2】Tools -> Options -> Connections
    2.1) 设置监听端口（默认为8888）

【3】配置完成后重启Fiddler（'重要'）
    3.1) 关闭Fiddler,再打开Fiddler
```

- **配置浏览器代理**

```
【1】安装Proxy SwitchyOmega谷歌浏览器插件

【2】配置代理
    2.1) 点击浏览器右上角插件SwitchyOmega -> 选项 -> 新建情景模式 -> myproxy(名字) -> 创建
    2.2) 输入  HTTP://  127.0.0.1  8888
    2.3) 点击 ：应用选项

【3】点击右上角SwitchyOmega可切换代理

【注意】：一旦切换了自己创建的代理,则必须要打开Fiddler才可以上网
```

- **Fiddler常用菜单**

```
【1】Inspector ：查看数据包详细内容
    1.1) 整体分为请求和响应两部分

【2】Inspector常用菜单
    2.1) Headers ：请求头信息
    2.2) WebForms: POST请求Form表单数据 ：<body>
                   GET请求查询参数: <QueryString>
    2.3) Raw ： 将整个请求显示为纯文本
```

移动端app数据抓取

- **方法1 - 手机 + Fiddler**

```
设置方法见文件夹 - 移动端抓包配置
```

- **方法2 - F12浏览器工具**

有道翻译手机版破解案例

```python
import requests
from lxml import etree

word = input('请输入要翻译的单词:')

post_url = 'http://m.youdao.com/translate'
post_data = {
  'inputtext':word,
  'type':'AUTO'
```

```
}

html = requests.post(url=post_url,data=post_data).text
parse_html = etree.HTML(html)
xpath_bds = '//ul[@id="translateResult"]/li/text()'
result = parse_html.xpath(xpath_bds)[0]

print(result)
```