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 www.tencent.com
【4】items.py(定义爬取数据结构)
    import scrapy
    class TencentItem(scrapy.Item):
        job_name = scrapy.Field()
        job_duty = scrapy.Field()

【5】tencent.py（写爬虫文件）
    import scrapy
    from ..items import TencentItem
    class TencentSpider(scrapy.Spider):
        name = 'tencent'
```

```
        allowed_domains =
['www.tencent.com']
        start_urls =
['http://www.tencent.com/']

        def parse(self, response):
            item = TencentItem()
            item['job_name'] =
response.xpath('').get()
            item['job_duty'] =
response.xpath('').get()
            yield item
```

【6】pipelines.py(数据处理)

```
    class TencentPipeline(object):
        def process_item(self, item,
spider):
            # MySQL、MongoDB
            return item
```

【7】settings.py(全局配置)

```
    ROBOTSTXT_OBEY = False
    CONCURRENT_REQUESTS = 32
    DOWNLOAD_DELAY = 0.5
    DEFAULT_REQUEST_HEADERS = {'User-
Agent':''}
    ITEM_PIPELINES = {
```

```
 'Tencent.pipelines.TencentPipeline':300,
    }

【8】run.py(和scrapy.cfg同目录)
    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

Day07笔记

Scrapy数据持久化

- **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'
```

腾讯招聘职位数据持久化

- **scrapy项目代码**

见day07笔记：Tencent 文件夹

- **建库建表SQL**

```
create database tencentdb charset utf8;
use tencentdb;
create table tencenttab(
job_name varchar(200),
job_type varchar(200),
job_duty varchar(2000),
job_require varchar(2000),
job_add varchar(100),
job_time varchar(100)
)charset=utf8;
```

- **MySQL数据持久化实现**

```
# 【1】settings.py添加
ITEM_PIPELINES = {
    # 在原来基础上添加MySQL的管道

'Tencent.pipelines.TencentMysqlPipeline
': 200,
}
MYSQL_HOST = '127.0.0.1'
MYSQL_USER = 'root'
MYSQL_PWD = '123456'
MYSQL_DB = 'tencentdb'
CHARSET = 'utf8'

# 【2】pipelines.py新建MySQL管道类
```

```
from .settings import *
import pymysql

class TencentMysqlPipeline:
    def open_spider(self, spider):
        self.db =
pymysql.connect(MYSQL_HOST, MYSQL_USER,
MYSQL_PWD, MYSQL_DB, charset=CHARSET)
        self.cur = self.db.cursor()
        self.ins = 'insert into
tencenttab values(%s,%s,%s,%s,%s,%s)'

    def process_item(self, item,
spider):
        li = [
            item['job_name'],
            item['job_type'],
            item['job_duty'],
            item['job_require'],
            item['job_add'],
            item['job_time'],
        ]
        self.cur.execute(self.ins, li)
        self.db.commit()

        return item

    def close_spider(self, item,
spider):
```

```
        self.cur.close()
        self.db.close()
```

- **MongoDB数据持久化实现**

```
# 【1】settings.py中添加
ITEM_PIPELINES = {
    # 添加MongoDB管道

'Tencent.pipelines.TencentMongoPipeline
': 400,
}
MONGO_HOST = '127.0.0.1'
MONGO_PORT = 27017
MONGO_DB = 'tencentdb'
MONGO_SET = 'tencentset'

# 【2】pipelines.py中新建MongoDB管道类
from .settings import *
import pymongo

class TencentMongoPipeline:
    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):

 self.myset.insert_one(dict(item))
```

- **csv及json数据持久化实现**

```
【1】csv
    scrapy crawl tencent -o tencent.csv

【2】json
    settings.py中添加变量:
FEED_EXPORT_ENCODING = 'utf-8'
    scrapy crawl tencent -o
tencent.json
```

盗墓笔记小说抓取 - 三级页面

- **目标**

```
【1】URL地址 ： http://www.daomubiji.com/
【2】要求 ： 抓取目标网站中盗墓笔记所有章节的所有小说的具体内容，保存到本地文件
    ./data/novel/盗墓笔记1:七星鲁王宫/七星鲁王_第一章_血尸.txt
    ./data/novel/盗墓笔记1:七星鲁王宫/七星鲁王_第二章_五十年后.txt
```

- **准备工作xpath**

```
【1】一级页面 - 大章节标题、链接：
    1.1) 基准xpath匹配a节点对象列表：
 '//li[contains(@id,"menu-item-20")]/a'
    1.2) 大章节标题：'./text()'
    1.3) 大章节链接：'./@href'

【2】二级页面 - 小章节标题、链接
    2.1) 基准xpath匹配article节点对象列表：
'//article'
    2.2) 小章节标题：'./a/text()'
    2.3) 小章节链接：'./a/@href'

【3】三级页面 - 小说内容
    3.1) p节点列表：
'//article[@class="article-
content"]/p/text()'
    3.2) 利用join()进行拼接：'
'.join(['p1','p2','p3',''])
```

项目实现

- **1、创建项目及爬虫文件**

```
scrapy startproject Daomu
cd Daomu
scrapy genspider daomu
www.daomubiji.com
```

- **2、定义要爬取的数据结构 - itemspy**

```
import scrapy

class DaomuItem(scrapy.Item):
    # 思考一下: 管道文件中你最终需要什么数据?
    # 思考完毕:
    # 1.小说路径 : ./novel/盗墓笔记1:xxx/
    # 2.文件名: 七星鲁王_第一章_血尸.txt
    # 3.小说内容
    directory = scrapy.Field()
    son_title = 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"""
        # 提取数据: 大名称 + 大链接 + 目录
```

```
        a_list =
response.xpath('//li[contains(@id,"menu
-item-20")]/a')
        for a in a_list:
            # 有继续交给调度器的请求,则创建
item对象
            item = DaomuItem()
            parent_title =
a.xpath('./text()').get()
            parent_url =
a.xpath('./@href').get()
            # ./novel/盗墓笔记1:七星鲁王
宫/
            item['directory'] =
'./novel/{}/'.format(parent_title)
            # 把对应文件夹创建了
            if not
os.path.exists(item['directory']):

 os.makedirs(item['directory'])
            # 继续交给调度器入队列
            yield scrapy.Request(
                url=parent_url,
                meta={'meta1':item},

 callback=self.parse_two_page
            )

    # 10个response由下载器返回
```

```
    def parse_two_page(self, response):
        """二级页面解析函数"""
        # 提取数据:章节名称 + 链接
        meta1 = response.meta['meta1']
        # meta1:盗墓笔记1
        # 基准xpath
        article_list =
response.xpath('//article')
        for article in article_list:
            # 又有继续交给调度器的请求,则单
独创建item对象
            item = DaomuItem()
            item['son_title'] =
article.xpath('./a/text()').get()
            son_url =
article.xpath('./a/@href').get()
            item['directory'] =
meta1['directory']

            # 交给调度器入队列
            yield scrapy.Request(
                url=son_url,
                meta={'item':item},

 callback=self.parse_three_page
            )

    # 盗墓笔记1 传过来了75个response
    # 盗墓笔记2 传过来了70个response
```

```
    # ... ...
    def parse_three_page(self,
response):
        """三级页面解析函数"""
        # 提取数据: 具体小说内容
        item = response.meta['item']
        # content_list: [<>,<>,<>]
        content_list =
response.xpath('//article[@class="artic
le-content"]/p/text()').extract()
        item['content'] =
'\n'.join(content_list)

        # 至此,1条完整数据提取完成,交给管道文
件处理
        yield item
```

- **4、管道文件实现数据处理 - pipelines.py**

```
class DaomuPipeline:
    def process_item(self, item, spider):
        # filename: ./novel/盗墓笔记1:七星/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,applic
ation/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,
}
```

分布式爬虫

- **分布式爬虫介绍**

```
【1】原理
    多台主机共享1个爬取队列

【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
```

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) 到离终点位置不远的地方开始减速
  【4】详细看代码注释
  总共：200个像素
  第一步：快速移动160个像素，停住！此时速度为0
  第二步：开始移动剩下的40个像素
          把这40个像素分成两部分(一部分4/5，另一部分1/5)
          第一部分： 32个像素，匀加速
          第二部分： 8个像素，匀减速

- **验证码面试问题**

【1】图形验证码

1.1》tesseract处理图形验证码（tessdata）

1.2》在线打码（调用在线打码接口，返回图片识别的结果-字符串）

【2】滑块验证码

2.1》默认人滑动的轨迹，先快速移动一段距离，剩下的距离先匀加速，再匀减速

2.2》如何计算滑块到缺口的距离

方案一：利用图像识别计算缺口的位置

方案二：利用Chrome浏览器的插件（page ruler），大概计算距离

- **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/bo
dy/div[1]/div[1]/ul[1]/li[2]').click()
browser.find_element_by_xpath('//*
[@id="username"]').send_keys('151102257
26')
```

```
browser.find_element_by_xpath('//*
[@id="password"]').send_keys('zhanshen0
01')
browser.find_element_by_xpath('/html/bo
dy/div[1]/div[2]/div[1]/div[5]/a').clic
k()
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_w
ith_offset(to_element=element,xoffset=1
80,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

```
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)
```

scrapy中间件使用

- **使用流程**

```
【1】middlewares.py中定义自己的中间件
【2】settings.py中开启中间件
    DOWNLOADER_MIDDLEWARES = {'':优先级}
```

- **随机User-Agent中间件（request.headers属性）**

```
# 中间件1 - 包装随机的User-Agent
from fake_useragent import UserAgent

class BaiduRandomUaDownloaderMiddleware(object):
    def process_request(self, request, spider):
        agent = UserAgent().random
        # scrapy.Request()当中所有参数都可以作为 request 的属性
        # 利用 request.headers 属性赋值
        request.headers['User-Agent'] = agent
        print(agent)
```

- **随机代理IP中间件（利用request.meta属性）**

```python
# 中间2 - 包装随机的代理
import random
from .proxy_list import proxy_li

class
BaiduProxyDownloaderMiddleware(object):
    def process_request(self, request,
spider):
        # 随机代理
        proxy = random.choice(proxy_li)
        # 利用  request.meta 属性
        # meta: 作用一:不同解析函数之间传递
数据 作用2:定义代理 meta={'proxy':''}
        request.meta['proxy'] = proxy
        print(proxy)

    def process_exception(self,
request, exception, spider):
        # 因为代理ip很可能不能用，scrapy会自
动尝试三次就抛出异常
        # 一直找到可用的代理为止
        return request
```