# 1. 京东全网爬虫需求

目标: 明确要抓取那些信息

## 1.1 抓取首页的分类信息

抓取数据: 各级分类的名称 和 URL

* 大分类名称和URL
* 中分类名称和URL
* 小分类名称和URL



## 1.2 抓取商品信息

抓取: 商品名称, 商品价格, 商品评论数量, 商品店铺, 商品促销, 商品选项, 商品图片的URL



- 商品名称

- 商品价格

- 商品评论数量

- 商品店铺

- 商品促销

- 商品选项

- 商品图片的URL

# 2. 开发环境与技术选择

平台: Mac，可以运行Window和Linux上

开发语言: Python3

开发工具: PyCharm

爬虫技术: scrapy\_redis分布式爬虫:

由于全网爬虫, 抓取页面非常多, 为了提高抓的速度, 选择使用scrapy框架 + scrapy\_redis分布式组件

存储: 由于京东全网的数据量达到了亿级, 存储又是结构化数据, 数据库,

选择使用MongoDB;

# 3. 京东全网爬虫的实现步骤

## 3.1 总体设计

我们采用广度优先策略, 我们把类别和商品信息的抓取分开来做.

好处: 可以提高程序的稳定性

## 3.2 实现步骤:

1. 创建爬虫项目

2. 根据需求, 定义数据数据模型

3. 实现分类爬虫

4. 保存分类信息

5. 实现商品爬虫

6. 保存商品信息

7. 实现随机User-Agent和代理IP下载器中间件, 解决IP反爬.

## 3.3 创建爬虫项目

scrapy startproject mall\_spider

# 4. 数据模型类

## 4.1 明确要抓取的数据(定义数据模型)

爬虫数据模型, 我们只能根据需求, 定义一个大概, 随着对项目实现可能会对数据模型做相应的修改.

## 4.2. 类别数据模型

类别数据模型类: 用于存储类别信息(Category) - 字段:

b\_category\_name: 大类别名称

b\_category\_url: 大类别URL

m\_category\_name: 中分类名称

m\_category\_url: 中分类URL

s\_category\_name: 小分类名称

s\_category\_url: 小分类URL

代码

class Category(scrapy.Item):

"""商品类别"""

# 大分类名称

b\_category\_name = scrapy.Field()

# 大分类URL

b\_category\_url = scrapy.Field()

# 中分类名称

m\_category\_name = scrapy.Field()

# 中分类URL

m\_category\_url = scrapy.Field()

# 小分类名称

s\_category\_name = scrapy.Field()

# 小分类URL

s\_category\_url = scrapy.Field()

## 4.3. 商品数据模型

商品数据模型类: 用于存储商品信息(Product)

字段:

product\_category: 商品类别

product\_sku\_id: 商品ID

product\_name: 商品名称

product\_img\_url: 商品图片URL

product\_book\_info: 图书信息, 作者,出版社

product\_option: 商品选项

product\_shop: 商品店铺

product\_comments: 商品评论数量

product\_ad: 商品促销

product\_price: 商品价格

代码

class Product(scrapy.Item):

# 商品类别

product\_category = scrapy.Field()

# 商品ID, 唯一标志

product\_sku\_id = scrapy.Field()

# 商品名称

product\_name = scrapy.Field()

# 商品图片URL

product\_img\_url = scrapy.Field()

# 商品店铺

product\_shop = scrapy.Field()

# 图书信息, 作者, 出版社等, 只有图书才有的信息

product\_book\_info = scrapy.Field()

# 商品选项

product\_option = scrapy.Field()

# 商品评论数量

product\_comments = scrapy.Field()

# 商品促销

product\_ad = scrapy.Field()

# 商品价格

product\_price = scrapy.Field()

# 5. 商品分类爬虫

目标: 抓取各级分类信息

步骤:

分析页面, 确定分类信息的URL

创建类别爬虫, 抓取数据

## 5.1 分析, 分类信息的URL

目标: 确定分类信息的URL

步骤:

进入到京东首页

右键检查, 打开开发者工具, 搜索 家用电器

确定分类的URL

图解:



结论: - 分类URL: https://dc.3.cn/category/get

## 5.2 创建爬虫, 抓取数据

目标: 抓取分类数据, 交给引擎

步骤:

创建类别爬虫

指定起始URL

解析数据, 交给引擎

### 5.2.1 创建爬虫

进入项目目录: cd mall\_spider

创建爬虫: scrapy genspider category\_spider jd.com

### 5.2.2. 指定起始URL

修改起始URL: <https://dc.3.cn/category/get>

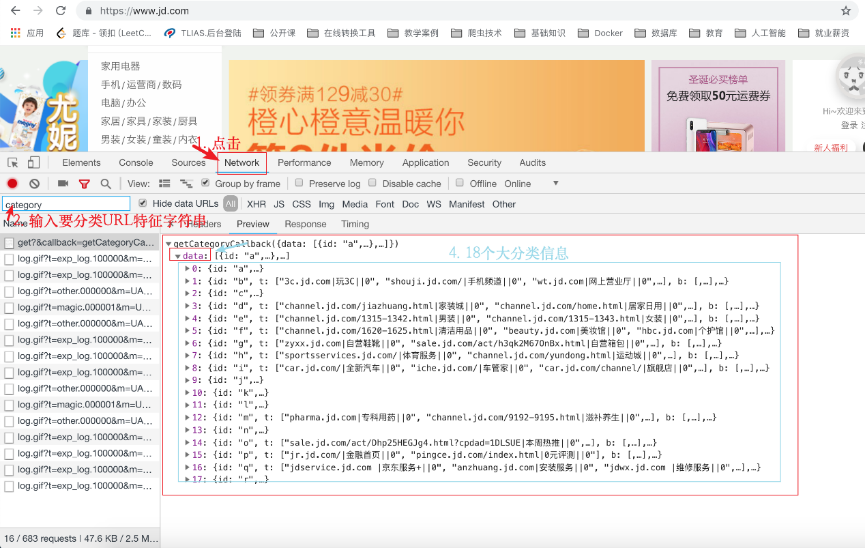
修改允许的域名

allowed\_domains = ['3.cn']

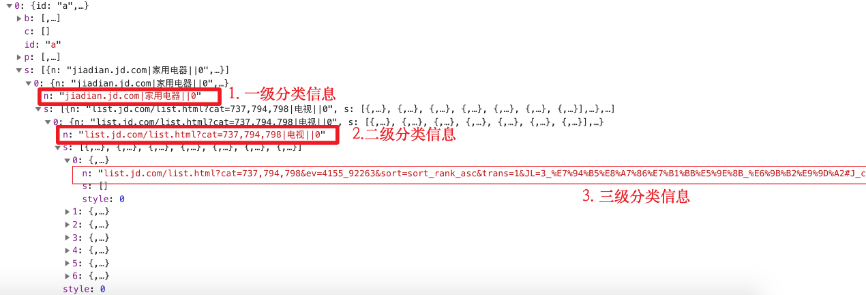
### 5.2.3. 解析数据, 交给引擎

分析数据格式:

整体数据



各级分类位置



分类信息格式

格式1:

jiadian.jd.com|家用电器||0

特点: 第一项分类URL, 第二项分类名称

格式2:

`652-654|摄影摄像||0

对应的URL: https://channel.jd.com/652-654.html

特点:第一项是频道ID, 包含一个 -

格式3:

1318-2628-12131|户外风衣||0

对应URL: https://list.jd.com/list.html?cat=1318,2628,12131

特点: 第一项为分类ID, 包含两个 -

格式化: 把 `-` 替换为逗号, 然后填充到占位的地方.

代码实现:

# -\*- coding: utf-8 -\*-

import scrapy

import json

from mall\_spider.items import Category

class JdCategorySpider(scrapy.Spider):

name = 'jd\_category'

allowed\_domains = ['dc.3.cn']

start\_urls = ['https://dc.3.cn/category/get']

# 频道URL模板

channel\_url\_pattern = 'https://channel.jd.com/{}.html'

# 列表URL模板

list\_url\_pattern = 'https://list.jd.com/list.html?cat={}'

def parse(self, response):

# 把传递过来的信息GBK进行解码, 因为京东的类别信息, 是使用GBK, 编码的

categorys = json.loads(response.body.decode('GBK'))

# 取出"data" 键中分类列表

categorys = categorys['data']

# 遍历分类列表

for category in categorys:

item = Category()

# 获取大分类,包含子分类; 注: 第一层的分类都在在0索引上;

b\_category = category['s'][0]

# 获取大分类信息(分类URL,名称)

b\_category\_info = b\_category['n']

# 解析大分类信息, 获取大分类名称和URL

item['b\_category\_name'], item['b\_category\_url'] = self.get\_category\_item(b\_category\_info)

# 获取中分类列表

m\_category\_s = b\_category['s']

# 遍历第二层分类列表

for m\_category in m\_category\_s:

# 获取中分类信息

m\_category\_info = m\_category['n']

item['m\_category\_name'], item['m\_category\_url'] = self.get\_category\_item(m\_category\_info)

# 获取小分类列表

s\_category\_s = m\_category['s']

# 遍历小分类分类列表

for s\_category in s\_category\_s:

# 获取第三层分类名称

s\_category\_info = s\_category['n']

# 获取三级分类信息

item['s\_category\_name'], item['s\_category\_url'] = self.get\_category\_item(s\_category\_info)

# print(item['s\_category\_name'])

# 把分类信息交给引擎

yield item

def get\_category\_item(self, category\_info):

# 使用 `|` 分割类型信息字符串

categorys = category\_info.split('|')

# 类别的名称

category\_name = categorys[1]

# 类别的URL

category\_url = categorys[0]

# 获取 category\_url 中 `-` 个数

count = category\_url.count('-')

if category\_url.count('jd.com') != 0:

# 其他就是本身就是URL, 前面补一个协议头

category\_url = 'https://' + category\_url

elif count == 1:

# 如果包含一个 '-' 是二级分类的频道

category\_url = self.channel\_url\_pattern.format(category\_url)

else:

# 如果包含2个 '-' 是三级分类的列表

# 1. 把 `-` 替换为 ','

category\_url = category\_url.replace('-', ',')

# 2. 生成具体列表的URL

category\_url = self.list\_url\_pattern.format(category\_url)

return category\_name, category\_url

# 6. 保存分类数据

* 目标: 把分类信息保存到MongoDBdb中
* 步骤:

实现保存分类的Pipeline类

在settings.py开启, 类别的Pipeline

## 6.1. 实现保存分类的Pipeline类

步骤:

open\_spider方法中, 链接MongoDB数据库, 获取要操作的集合

process\_item 方法中, 向MongoDB中插入类别数据

close\_spider 方法中, 关闭MongoDB的链接

代码

from mall\_spider.spiders.jd\_category import JdCategorySpider

from pymongo import MongoClient

from mall\_spider.settings import MONGO\_URL

class CategoryPipeline(object):

def open\_spider(self, spider):

if isinstance(spider, JdCategorySpider):

# 建立MongoDB数据库链接

self.client = MongoClient(MONGO\_URL)

# 获取要操作集合

self.category = self.client['jd']['category']

def process\_item(self, item, spider):

if isinstance(spider, JdCategorySpider):

# 把数据插入到mongo中

self.category.insert\_one(dict(item))

return item

def close\_spider(self, spider):

"""关闭"""

if isinstance(spider, JdCategorySpider):

self.client.close()

## 6.2. 在settings.py开启, 类别的Pipeline

# 在settings.py开启, 类别的Pipeline

ITEM\_PIPELINES = {

'mall\_spider.pipelines.CategoryPipeline': 300,

}

运行爬虫, 进行测试

mongo中进行操作

show dbs

use jd

show collections

db.category.find()

db.category.count()

# 7. 实现商品爬虫

总体设计:

把MongoDB中存储的分类信息, 放到redis\_key指定列表中

支持分布式爬虫, 当然也可以在一台电脑上运行多次, 以启动多个进程,充分使用CPU的多核.

所以这里的爬虫, 先从一个分类开始抓就可以了, 后面再改造为分布式

目标: 抓取商品数据

步骤:

* 分析, 确定数据所在的URL
* 代码实现:
* 商品爬虫实现分布式

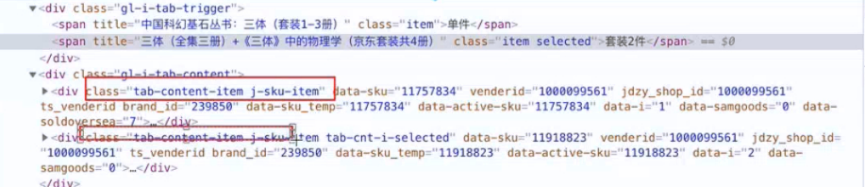
## 7.1. 分析, 确定数据所在的URL

### 列表页

提取商品 skuid



对于图书的列表页, 稍微有所不同, 对于有套装的图书, 有两个商品, 需要提取2个商品的sku-id

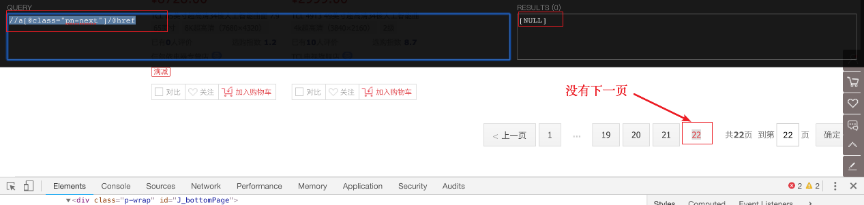


对比无套装的商品, 其sku-id的规则为 contains(@class, "j-sku-item")

j-sku-item

### 实现翻页

获取下一页URL



没有下一页的情况



### 详情页

由于PC和手机页面商品信息, 在js中, 且比较分散, 并且每次请求数量页比较大, 我们这里使用手机抓包, 抓到json数据.

charles配置:

https支持: Help > SSL Proxying > Install Charles Root Certificate

pc端charles能拦截到手机端的通信, 保持手机和pc在同一个网段中, 最好是手机连接pc端的wifi热点.

itools pro查看手机实时桌面

手机ip地址: 192.168.2.8

路由器(即pc的ip地址): 192.168.2.1

查看pc端ip地址: charles > help > Local IP Address: 192.168.2.1

手动配置手机的ip代理

服务器: 192.168.2.1

端口号: 8888

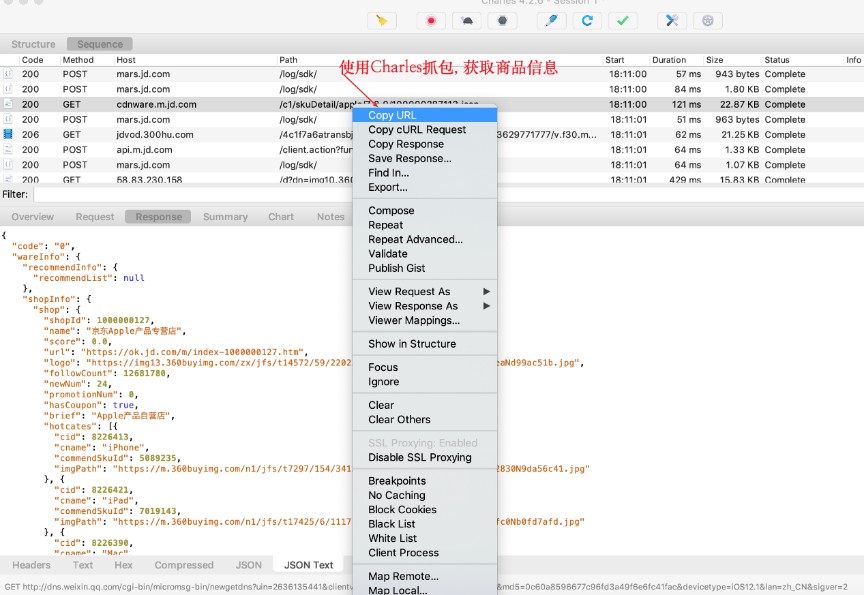
手机端安装charles的https证书:

Help > SSL Proxying > Install Charles Root Certificate on a Mobile Device or Remote Browser

进入手机京东详情页, 查看抓包的响应信息, 看是否有商品详情页的数据

商品基本信息

图:



URL: <https://cdnware.m.jd.com/c1/skuDetail/apple/7.3.0/32426231880.json>

最后一部分是商品skuid

可以获取到的信息: 商品名称, 商品店铺信息 , 商品类别id, 商品品牌id, 商品选项

{

"code": "0",

"wareInfo": {

"recommendInfo": {

"recommendList": null

},

// 商品店铺信息

"shopInfo": {

"shop": {

"shopId": 1000000127,

"name": "京东Apple产品专营店",

...

},

"basicInfo": {

"gift": false,

"bookInfo": {

// 如果是书, 这里是书的选项信息

"display": false

},

"colorSizeInfo": {

// 商品选项信息列表 有的没有

"colorSize": [{

"buttons": [{

"no": "1",

"skuList": ["100000177738", "100000287117", "100000287145", "100000309448", "100000309450", "100000375233", "100000435832", "100000458753", "100000458755", "100001860767", "100001860773"],

"text": "金色"

}, {

"no": "2",

"skuList": ["100000177764", "100000287113", "100000287135", "100000435780", "100000435816", "100000435818", "100000569049", "100000602206", "100000602208", "100001860765", "100002539302"],

"text": "深空灰色"

}, {

"no": "3",

"skuList": ["100000177740", "100000177784", "100000287147", "100000435834", "100000458737", "100000458739", "100000602174", "100000602176", "100000602204", "100001860789", "100002539304"],

"text": "银色"

}],

"title": "颜色"

}, {

"buttons": [{

"no": "1",

"skuList": ["100000177738", "100000177740", "100000177764", "100000177784", "100000287113", "100000287117", "100000287135", "100000287145", "100000287147"],

"text": "公开版"

},

...

],

"title": "版本"

}, {

"buttons": [{

"no": "1",

"skuList": ["100000177764", "100000287145", "100000287147", "100000375233", "100000435818", "100000458739", "100000458755", "100000602204", "100000602208", "100001860765", "100001860773", "100001860789"],

"text": "64GB"

},

...

],

"title": "内存"

}],

"colorSizeTips": "#与其他已选项无法组成可售商品，请重选"

},

...

// 品牌ID

"brandID": "14026",

...

// 商品图片

"wareImage": [{

"small": "https://m.360buyimg.com/mobilecms/s720x720\_jfs/t1/3/15/4536/138660/5b997bf8Ed72ebce7/819dcf182d743897.jpg!q70.jpg.webp",

...

}

...

],

...

// 商品名称

"name": "Apple iPhone XS Max (A2104) 256GB 深空灰色 移动联通电信4G手机 双卡双待",

// 商品类别id

"category": "9987;653;655"

}

}

}

### 商品促销信息(PC端):

图:



URL: https://cd.jd.com/promotion/v2?skuId=4749506&area=1\_72\_4137\_0&cat=737%2C794%2C798

参数

skuId=4749506: 商品sku\_id

area=1\_72\_4137\_0: 购买者区域, 固定的

cat=737%2C794%2C798: 类别

数据

{

...

// 商品促销信息

"ads": [{

"id": "AD\_4749506",

"ad": "【即刻预约，21号秒杀到手价2999】\n1、前100名晒单送腾讯企鹅影院季卡，联系客服领取！！\n2、曲面爆款，5.5万好评推荐！<a target=\"\_blank\" href=\"https://item.jd.com/7055876.html\">升级55Q1D超清全面屏电视</a>"

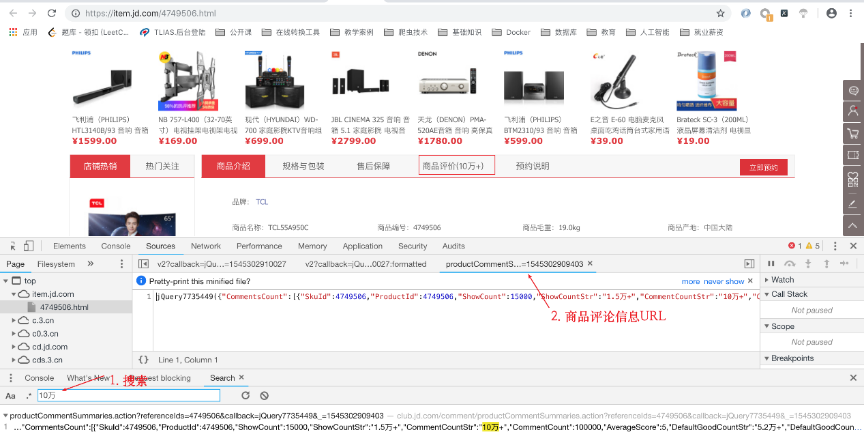
}],

...

}

### 商品评论信息(PC端)

图:



URL: https://club.jd.com/comment/productCommentSummaries.action?referenceIds=4749506

参数

referenceIds=4749506: 商品sku\_id

数据:

{"CommentsCount":[

{

"CommentCountStr":"10万+",

"CommentCount":100000, //评论数量

"AverageScore":5,

"GoodRate":0.98, //好评率

"PoorCountStr":"600+",

"PoorCount":600, // 差评数量

...

}]}

### 商品价格信息:

图:



URL: https://p.3.cn/prices/mgets?skuIds=J\_4749506

参数:

skuIds=J\_4749506 商品的sku\_id

数据:

[

{

"op": "5499.00",

"m": "5999.00",

"id": "J\_4749506", //商品skuid

"p": "3299.00" // 商品价格

}

]

### 7.2 2. 代码实现

#### 步骤:

1. 重写start\_requests方法, 根据分类信息构建列表页的请求
2. 解析列表页, 提取商品的skuid, 构建商品基本的信息请求; 实现翻页
3. 解析商品基本信息, 构建商品促销信息的请求
4. 解析促销信息,构建商品评价信息的请求,
5. 解析商品评价信息, 构建价格信息的请求
6. 解析价格信息

#### 代码:

创建爬虫

scrapy genspider jd\_product jd.com

修改jd\_product.py爬虫文件, 注释掉start\_urls, 重写start\_requests方法, 从mongodb中读取分类信息, 发送请求.

# -\*- coding: utf-8 -\*-

import scrapy

import json

from jsonpath import jsonpath

class JdProductSpider(scrapy.Spider):

name = 'jd\_product'

allowed\_domains = ['jd.com', 'p.3.cn']

def start\_requests(self):

category = {"b\_category\_name" : "家用电器",

"b\_category\_url" : "https://jiadian.jd.com",

"m\_category\_name" : "洗衣机",

"m\_category\_url": "https://list.jd.com/list.html?cat=737,794,880",

"s\_category\_name" : "洗衣机配件",

"s\_category\_url" : "https://list.jd.com/list.html?cat=737,794,877" }

yield scrapy.Request(category['s\_category\_url'], self.parse, meta={'category': category})

def parse(self, response):

# 获取类别信息

category = response.meta['category']

# 获取类别的URL

category\_url = response.url.split('&')[0]

# 获取所有商品的sku\_ids

sku\_ids = response.xpath('//div[contains(@class, "j-sku-item")]/@data-sku').extract()

# 遍历sku\_ids, 构建基本详情信息的请求

for sku\_id in sku\_ids:

item = {

'product\_category': category,

'product\_sku\_id':sku\_id

}

product\_url = 'https://cdnware.m.jd.com/c1/skuDetail/apple/7.3.0/{}.json'.format(sku\_id)

yield scrapy.Request(product\_url, callback=self.parse\_product, meta={'item': item})

# 获取下一页的URL

next\_url = response.xpath('//a[@class="pn-next"]/@href').extract\_first()

if next\_url:

# 补全URL

next\_url = response.urljoin(next\_url)

# 构建下一页请求

yield scrapy.Request(next\_url, callback=self.parse, meta={'category': category})

def parse\_product(self, response):

# 取出传递过来的数据

item = response.meta['item']

# 把响应数据数据转为字典

product\_dic = json.loads(response.text)

# 获取商品名称

item['product\_name'] = product\_dic['wareInfo']['basicInfo']['name']

if item['product\_name']:

# 获取类别id, 把 `;` 替换为 ,

item['product\_category\_id'] = product\_dic['wareInfo']['basicInfo']['category'].replace(';', ',')

# 获取店铺信息

product\_shop = jsonpath(product\_dic, '$..shop')

if product\_shop:

product\_shop = product\_shop[0]

if product\_shop is None:

item['product\_shop'] = {'name':'京东自营'}

else:

item['product\_shop'] = {

"shopId": product\_shop['shopId'],

"name": product\_shop['name'],

"score": product\_shop['score'],

"url": product\_shop['url'],

}

# 如果是书, 记录书的信息

if product\_dic['wareInfo']['basicInfo']['bookInfo']['display']:

item['product\_book\_info'] = product\_dic['wareInfo']['basicInfo']['bookInfo']

# 删除display

del item['book\_info']['display']

# 获取商品选购信息

color\_sizes = jsonpath(product\_dic, '$..colorSize')

product\_option = {}

if color\_sizes:

for color\_size in color\_sizes[0]:

title = color\_size['title']

texts = jsonpath(color\_size, '$..text')

product\_option.update({title:texts})

# print(product\_option)

item['product\_option'] = product\_option

# 商品图片

item['product\_img\_url'] = jsonpath(product\_dic, '$..wareImage[0].small')[0]

# 构建促销信息的请求

ad\_url = 'https://cd.jd.com/promotion/v2?skuId={}&area=1\_72\_4137\_0&cat={}'.format(item['product\_sku\_id'], item['product\_category\_id'])

yield scrapy.Request(ad\_url, callback=self.parse\_ad, meta={'item': item})

def parse\_ad(self, response):

"""获取商品促销"""

item = response.meta['item']

ad\_dic = json.loads(response.body.decode('GB18030'))

ad = ad\_dic['ads'][0]['ad']

item['product\_ad'] = ad

# for key, value in item.items():

# print('{} = {}'.format(key, value))

# 构建平均信息请求

comments\_url = 'https://club.jd.com/comment/productCommentSummaries.action?referenceIds={}'.format(item['product\_sku\_id'])

yield scrapy.Request(comments\_url, callback=self.parse\_comments, meta={'item': item})

def parse\_comments(self, response):

"""解析商品评论信息"""

item = response.meta['item']

comments\_dic = json.loads(response.text)

comments = {

'comment\_count': jsonpath(comments\_dic, '$..CommentCount')[0],

'good\_rate': jsonpath(comments\_dic, '$..GoodRate')[0],

'poor\_count': jsonpath(comments\_dic, '$..PoorCount')[0],

}

item['product\_comments'] = comments

# print(item)

# 构建价格请求

price\_url = 'https://p.3.cn/prices/mgets?skuIds=J\_{}'.format(item['product\_sku\_id'])

yield scrapy.Request(price\_url, callback=self.parse\_price, meta={'item': item})

def parse\_price(self, response):

"""解析价格"""

item = response.meta['item']

item['product\_price'] = json.loads(response.text)[0]['p']

# print(item)

yield item

不显示日志信息

scrapy crawl jd\_product --nolog

## 7.3 商品爬虫实现分布式

步骤:

* 修改爬虫类
* 在settings文件中配置scrapy\_redis
* 写一个程序用于把MongoDB中分类信息, 放入到爬虫redis\_key指定的列表中

### 1. 修改爬虫类

步骤:

* 修改继承关系: 继承RedisSpider
* 指定redis\_key
* 把重写start\_requests 改为 重写 make\_request\_from\_data

代码

查看RedisSpider > RedisMixin, 在其中重写了start\_requests方法, 在其中调用了next\_requests()方法, 在next\_requests()方法中不断的从redis中读取起始url, 如果有数据, 就把数据交给make\_request\_from\_data, 通过make\_request\_from\_data构建并返回Request请求对象

from scrapy\_redis.spiders import RedisSpider

import pickle

# 1. 修改继承关系: 继承RedisSpider

class JdProductSpider(RedisSpider):

name = 'jd\_product'

allowed\_domains = ['jd.com', 'p.3.cn']

# 2. 指定redis\_key

redis\_key = 'jd\_product:start\_category'

# 3. 把重写start\_requests 改为 重写 make\_request\_from\_data

def make\_request\_from\_data(self, data):

# 把从Redis中读取到分类信息, 转换为字典

category = pickle.loads(data)

return scrapy.Request(category['s\_category\_url'], self.parse, meta={'category': category})

注意: 在make\_request\_from\_data不能使用 yield 必须使用 return

### 2. 在settings文件中配置scrapy\_redis

# MongoDB数据库的URL

MONGO\_URL = 'mongodb://127.0.0.1:27017'

# REDIS数据链接

REDIS\_URL = ' redis://127.0.0.1:6379/0'

# 去重容器类: 用于把已爬指纹存储到基于Redis的set集合中

DUPEFILTER\_CLASS = "scrapy\_redis.dupefilter.RFPDupeFilter"

# 调度器: 用于把待爬请求存储到基于Redis的队列

SCHEDULER = "scrapy\_redis.scheduler.Scheduler"

# 是不进行调度持久化:

# 如果是True, 当程序结束的时候, 会保留Redis中已爬指纹和待爬的请求

# 如果是False, 当程序结束的时候, 会清空Redis中已爬指纹和待爬的请求

SCHEDULER\_PERSIST = True

### 3. 写一个程序用于把MongoDB中分类信息, 放入到爬虫redis\_key指定的列表中

步骤:

在项目文件夹下创建 add\_category\_to\_redis.py

实现方法 add\_category\_to\_redis:

链接MongoDB

链接Redis

读取MongoDB中分类信息, 序列化后, 添加到商品爬虫redis\_key指定的list

关闭MongoDB

在if \_\_name\_\_ == '\_\_main\_\_':中调用add\_category\_to\_redis方法

代码

from redis import StrictRedis

from pymongo import MongoClient

import pickle

from mall\_spider.settings import MONGO\_URL, REDIS\_URL

from mall\_spider.spiders.jd\_product import JdProductSpider

# 把MongoDB中分类信息, 添加到Redis中

def add\_category\_to\_redis():

# 链接MongoDB

client = MongoClient(MONGO\_URL)

# 链接Redis

redis = StrictRedis.from\_url(REDIS\_URL)

cursor = client['jd']['category'].find()

# 读取MongoDB中分类信息, 序列化后, 添加到商品爬虫redis\_key指定的list

for category in cursor:

redis.rpush(JdProductSpider.redis\_key, pickle.dumps(category))

# 关闭MongoDB的链接

client.close()

if \_\_name\_\_ == '\_\_main\_\_':

add\_category\_to\_redis()

## 8. 保存商品数据

步骤:

实现存储商品Pipeline类

在settings.py中开启这个管道

### 8.1.实现存储商品Pipeline类

步骤

在 open\_spider方法, 建立MongoDB数据库连接, 获取要操作的集合

在 process\_item方法, 把数据插入到MongoDB中

在close\_spider方法, 关闭数据库连接

代码

class ProductPipeline(object):

def open\_spider(self, spider):

if isinstance(spider, JdProductSpider):

# 建立MongoDB数据库链接

self.client = MongoClient(MONGO\_URL)

# 获取要操作集合

self.category = self.client['jd']['product']

def process\_item(self, item, spider):

if isinstance(spider, JdProductSpider):

# 把数据插入到mongo中

self.category.insert\_one(dict(item))

return item

def close\_spider(self, spider):

"""关闭"""

if isinstance(spider, JdProductSpider):

self.client.close()

### 8.2. 在settings.py中开启这个管道

ITEM\_PIPELINES = {

'mall\_spider.pipelines.CategoryPipeline': 300,

# 开启商品管道

'mall\_spider.pipelines.ProductPipeline': 301,

}

## 9. 实现下载器中间件

为了避免IP反爬, 我们实现随机User-Agent和代理IP的中间件

步骤:

实现随机User-Agent的中间件

实现代理IP中间件

在settings.py 文件开启, 下载器中间件

### 9.1 实现随机User-Agent的中间件

步骤

准备User-Agent列表

在middlewares.py中, 实现RandomUserAgent类

实现process\_request方法

如果是请求是 https://cdnware.m.jd.com 开头的, 就是设置一个iPhone的user-agent

否则从User-Agent列表中随机取出一个

代码

import requests

import random

# 准备请求头

USER\_AGENTS = [

"Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1; AcooBrowser; .NET CLR 1.1.4322; .NET CLR 2.0.50727)",

"Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.0; Acoo Browser; SLCC1; .NET CLR 2.0.50727; Media Center PC 5.0; .NET CLR 3.0.04506)",

"Mozilla/4.0 (compatible; MSIE 7.0; AOL 9.5; AOLBuild 4337.35; Windows NT 5.1; .NET CLR 1.1.4322; .NET CLR 2.0.50727)",

"Mozilla/5.0 (Windows; U; MSIE 9.0; Windows NT 9.0; en-US)",

"Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; Win64; x64; Trident/5.0; .NET CLR 3.5.30729; .NET CLR 3.0.30729; .NET CLR 2.0.50727; Media Center PC 6.0)",

"Mozilla/5.0 (compatible; MSIE 8.0; Windows NT 6.0; Trident/4.0; WOW64; Trident/4.0; SLCC2; .NET CLR 2.0.50727; .NET CLR 3.5.30729; .NET CLR 3.0.30729; .NET CLR 1.0.3705; .NET CLR 1.1.4322)",

"Mozilla/4.0 (compatible; MSIE 7.0b; Windows NT 5.2; .NET CLR 1.1.4322; .NET CLR 2.0.50727; InfoPath.2; .NET CLR 3.0.04506.30)",

"Mozilla/5.0 (Windows; U; Windows NT 5.1; zh-CN) AppleWebKit/523.15 (KHTML, like Gecko, Safari/419.3) Arora/0.3 (Change: 287 c9dfb30)",

"Mozilla/5.0 (X11; U; Linux; en-US) AppleWebKit/527+ (KHTML, like Gecko, Safari/419.3) Arora/0.6",

"Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.8.1.2pre) Gecko/20070215 K-Ninja/2.1.1",

"Mozilla/5.0 (Windows; U; Windows NT 5.1; zh-CN; rv:1.9) Gecko/20080705 Firefox/3.0 Kapiko/3.0",

"Mozilla/5.0 (X11; Linux i686; U;) Gecko/20070322 Kazehakase/0.4.5",

"Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.9.0.8) Gecko Fedora/1.9.0.8-1.fc10 Kazehakase/0.5.6",

"Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/535.11 (KHTML, like Gecko) Chrome/17.0.963.56 Safari/535.11",

"Mozilla/5.0 (Macintosh; Intel Mac OS X 10\_7\_3) AppleWebKit/535.20 (KHTML, like Gecko) Chrome/19.0.1036.7 Safari/535.20",

"Opera/9.80 (Macintosh; Intel Mac OS X 10.6.8; U; fr) Presto/2.9.168 Version/11.52",

"Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/536.11 (KHTML, like Gecko) Chrome/20.0.1132.11 TaoBrowser/2.0 Safari/536.11",

"Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.1 (KHTML, like Gecko) Chrome/21.0.1180.71 Safari/537.1 LBBROWSER",

"Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; WOW64; Trident/5.0; SLCC2; .NET CLR 2.0.50727; .NET CLR 3.5.30729; .NET CLR 3.0.30729; Media Center PC 6.0; .NET4.0C; .NET4.0E; LBBROWSER)",

"Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1; QQDownload 732; .NET4.0C; .NET4.0E; LBBROWSER)",

"Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/535.11 (KHTML, like Gecko) Chrome/17.0.963.84 Safari/535.11 LBBROWSER",

"Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.1; WOW64; Trident/5.0; SLCC2; .NET CLR 2.0.50727; .NET CLR 3.5.30729; .NET CLR 3.0.30729; Media Center PC 6.0; .NET4.0C; .NET4.0E)",

"Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; WOW64; Trident/5.0; SLCC2; .NET CLR 2.0.50727; .NET CLR 3.5.30729; .NET CLR 3.0.30729; Media Center PC 6.0; .NET4.0C; .NET4.0E; QQBrowser/7.0.3698.400)",

"Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1; QQDownload 732; .NET4.0C; .NET4.0E)",

"Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; Trident/4.0; SV1; QQDownload 732; .NET4.0C; .NET4.0E; 360SE)",

"Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1; QQDownload 732; .NET4.0C; .NET4.0E)",

"Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.1; WOW64; Trident/5.0; SLCC2; .NET CLR 2.0.50727; .NET CLR 3.5.30729; .NET CLR 3.0.30729; Media Center PC 6.0; .NET4.0C; .NET4.0E)",

"Mozilla/5.0 (Windows NT 5.1) AppleWebKit/537.1 (KHTML, like Gecko) Chrome/21.0.1180.89 Safari/537.1",

"Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.1 (KHTML, like Gecko) Chrome/21.0.1180.89 Safari/537.1",

"Mozilla/5.0 (iPad; U; CPU OS 4\_2\_1 like Mac OS X; zh-cn) AppleWebKit/533.17.9 (KHTML, like Gecko) Version/5.0.2 Mobile/8C148 Safari/6533.18.5",

"Mozilla/5.0 (Windows NT 6.1; Win64; x64; rv:2.0b13pre) Gecko/20110307 Firefox/4.0b13pre",

"Mozilla/5.0 (X11; Ubuntu; Linux x86\_64; rv:16.0) Gecko/20100101 Firefox/16.0",

"Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.11 (KHTML, like Gecko) Chrome/23.0.1271.64 Safari/537.11",

"Mozilla/5.0 (X11; U; Linux x86\_64; zh-CN; rv:1.9.2.10) Gecko/20100922 Ubuntu/10.10 (maverick) Firefox/3.6.10"

]

class RandomUserAgent(object):

def process\_request(self, request, spider):

if request.url.startswith('https://cdnware.m.jd.com'):

# 如果使用手机抓包, 获取到商品信息; 生成请求请求头

request.headers['user-agent'] = 'JD4iPhone/164880 (iPhone; iOS 12.1.2; Scale/2.00)'

else:

# 随机获取一个请求头, 进行设置

request.headers['user-agent'] = random.choice(USER\_AGENTS)

### 9.2. 实现代理IP中间件

步骤:

在middlewares.py中, 实现ProxyMiddleware类

实现process\_request方法

从代理池中获取一个随机的代理IP, 需指定代理IP的协议, 和访问的域名

设置给request.meta['proxy']

实现process\_exception方法

当请求出现异常的时候, 代理池哪些代理IP在本域名下是不可以用的

查看scrapy的retry模块, 查看在哪些异常的情况下需要重新发起请求

from scrapy.downloadermiddlewares.retry import RetryMiddleware

代码

from twisted.internet import defer

from twisted.internet.error import TimeoutError, DNSLookupError, \

ConnectionRefusedError, ConnectionDone, ConnectError, \

ConnectionLost, TCPTimedOutError

from twisted.web.client import ResponseFailed

from scrapy.core.downloader.handlers.http11 import TunnelError

class ProxyMiddleware(object):

EXCEPTIONS\_TO\_RETRY = (defer.TimeoutError, TimeoutError, DNSLookupError,

ConnectionRefusedError, ConnectionDone, ConnectError,

ConnectionLost, TCPTimedOutError, ResponseFailed,

IOError, TunnelError)

def process\_request(self, request, spider):

"""

从代理池中获取一个随机的代理IP

设置给request.meta['proxy']

"""

response = requests.get('http://localhost:6868/random?protocol=https&domain=jd.com')

request.meta['proxy'] = response.content.decode()

request.meta['dont\_redirect'] = True

return None

def process\_exception(self, request, exception, spider):

if isinstance(exception, self.EXCEPTIONS\_TO\_RETRY):

# 获取代理IP

proxy = request.meta['proxy']

# 提取IP地址

ip = re.findall('https://(.+):\d+', proxy)[0]

params = {

'ip': ip,

'domain': 'jd.com'

}

requests.get('http://localhost:6868/disable\_domain', params=params)

# 构建请求返回

req = request.copy()

req.dont\_filter = True

return req

9.3. 在settings.py中开启上面的两个下载器中间件

# 配置下载器中间件

DOWNLOADER\_MIDDLEWARES = {

'mall\_spider.middlewares.RandomUserAgent': 500,

'mall\_spider.middlewares.ProxyMiddl eware': 543,

}