[伯乐在线文章爬虫 2](#_Toc524105597)

[新建项目和爬虫 2](#_Toc524105598)

[设置pycharm调试scrapy爬虫项目 3](#_Toc524105599)

[修改settings.py进行设置 4](#_Toc524105600)

[添加断点, 进行调试, 查看response响应的内容 5](#_Toc524105601)

[使用scrapy提取网页的信息 5](#_Toc524105602)

[使用xpath进行信息的提取. 5](#_Toc524105603)

[使用css选择器提取网页中的元素. 7](#_Toc524105604)

[使用extract\_first()代替extract() 9](#_Toc524105605)

[完善jobbole.py, 从帖子详情页中提取信息 10](#_Toc524105606)

[调试工具Stepping toolbar的使用 11](#_Toc524105607)

[从文章列表页爬取所有文章 13](#_Toc524105608)

[提取下一页的链接, 构造并发送请求 14](#_Toc524105609)

[使用item类 16](#_Toc524105610)

[列表页图片url的获取 20](#_Toc524105611)

[下载文章的封面图片 22](#_Toc524105612)

[自定义pipline获取下载的图片保存的路径. 24](#_Toc524105613)

[查看images.ImagesPipeline的源码 24](#_Toc524105614)

[修改pipelines.py, 自定义图片下载管道 27](#_Toc524105615)

[debug获取results的结构 27](#_Toc524105616)

[修改pipelines.py, 从results中取出图片保存的路径 29](#_Toc524105617)

[对article\_url进行md5处理 31](#_Toc524105618)

[保存item数据到json文件中 32](#_Toc524105619)

[使用自定义方法将数据保存到json文件中. 32](#_Toc524105620)

[使用scrapy的JsonItemExporter保存json文件 34](#_Toc524105621)

[通过pipeline保存数据到mysql 37](#_Toc524105622)

[安装python的mysql驱动程序 37](#_Toc524105623)

[建立数据库和数据表 39](#_Toc524105624)

[修改jobbole.py, 把发表时间修改为日期格式 41](#_Toc524105625)

[将数据保存到mysql中 45](#_Toc524105626)

[把django的ORM集成到scrapy中 48](#_Toc524105627)

[scrapy itemloader机制 49](#_Toc524105628)

[ItemLoader的使用方法 49](#_Toc524105629)

[修改jobbole\_it.py, 使用itemloader 51](#_Toc524105630)

[修改items.py, 对itemloader提取到的字段进行处理 53](#_Toc524105631)

[自定义itemloader对item中的信息进行处理 62](#_Toc524105632)

[通过downloadmiddleware随机更换user-agent 82](#_Toc524105633)

[实现user-agent的自动更换的方法 82](#_Toc524105634)

[查看scrapy自带的UserAgentMiddleware的源码 82](#_Toc524105635)

[使用Fake Useragent 84](#_Toc524105636)

[使用fake useragent来随机切换user-agent 87](#_Toc524105637)

[scrapy实现ip代理池 89](#_Toc524105638)

[爬虫代理哪家强？十大付费代理详细对比评测出炉！ 89](#_Toc524105639)

[使用2个代理获取代理地址 90](#_Toc524105640)

[修改middlewares.py, 同时切换代理和ua 92](#_Toc524105641)

[在settings.py中设置启动middleware 95](#_Toc524105642)

[解决图片下载出错引起的频繁更换代理的问题. 98](#_Toc524105643)

[解决mysql 4字节utf-8字符的问题 98](#_Toc524105644)

[把项目部署到 ubuntu server中 100](#_Toc524105645)

[环境配置 100](#_Toc524105646)

[把项目部署到ubuntu server中. 108](#_Toc524105647)

[运行爬虫 113](#_Toc524105648)

[修改为分布式爬虫 113](#_Toc524105649)

[使用docker部署爬虫 115](#_Toc524105650)

[安装Docker 115](#_Toc524105651)

[Docker的使用 122](#_Toc524105652)

[将容器封装为docker镜像. 138](#_Toc524105653)

[把保存的tar镜像文件复制到Yunzhuji的ubuntu物理机中. 139](#_Toc524105654)

[可选项: 使用bypy上传把文件上传到百度云中. 140](#_Toc524105655)

[在yunzhuji的容器中运行爬虫 141](#_Toc524105656)

[scrapy 日志处理 143](#_Toc524105657)

[分布式爬虫爬取结束时自动结束爬虫. 143](#_Toc524105658)

# 伯乐在线文章爬虫

## 新建项目和爬虫

进入到保存项目文件的目录中. 如 E:\scrapy\_projects

workon python3\_spider

scrapy startproject jobbole\_article

# scrapy在创建项目的时候可以自定义模板.

You can start your first spider with:

cd jobbole\_article

scrapy genspider example example.com

cd jobble\_article

scrapy genspider jobbole blog.jobbole.com

创建一个名为jobbole的爬虫. 会自动生成一个jobbole.py的爬虫文件

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

**import** **scrapy**

**class** **JobboleSpider**(scrapy.Spider):

name = 'jobbole'

allowed\_domains = ['blog.jobbole.com']

start\_urls = ['http://blog.jobbole.com/']

**def** parse(self, response):

**pass**

## 设置pycharm调试scrapy爬虫项目

1. 使用pycarm打开ArticleSpider

2. 设置项目的python解析器

pycharm settings > project interpreter > 导入之前创建的虚拟环境.

使用Pychram来导入本地的虚拟环境, Windows下要找到虚拟环境中\Scripts\python.exe. Linux下要找到bin下的python运行文件

3. pycharm中没有 scrapy 的调试模式, 需要自己写一个文件.

在与scrapy.cfg同目录下新建一个main.py

*#coding = utf-8*

**from** **scrapy** **import** cmdline

**import** **sys**, **os**

*# 把本文件所在的目录添加到python搜索路径列表中.*

*# os.path.abspath(\_\_file\_\_)得到的是当前的main文件所在的绝对目录, 而os.path.dirname得到的是某个文件的父级目录.*

*# print(os.path.dirname(os.path.abspath(\_\_file\_\_)))*

sys.path.append(os.path.dirname(os.path.abspath(\_\_file\_\_)))

*# cmdline.execute(['scrapy', 'crawl', 'jobbole'])*

cmdline.execute("scrapy crawl jobbole".split())

**python 的 sys.path.append()**

当我们导入一个模块时: import  xxx, 默认情况下python解析器会搜索当前目录、已安装的内置模块和第三方模块, 搜索路径存放在**sys模块的path**中:

>>> **import** sys

>>> sys.path

['', 'C:\\Python352\\Lib\\idlelib', 'C:\\Python352\\python35.zip', 'C:\\Python352\\DLLs', 'C:\\Python352\\lib', 'C:\\Python352', 'C:\\Python352\\lib\\site-packages', 'C:\\Python352\\lib\\site-packages\\setuptools-28.6.1-py3.5.egg', 'C:\\Python352\\lib\\site-packages\\pip-8.1.2-py3.5.egg', 'C:\\Python352\\lib\\site-packages\\requests-2.11.1-py3.5.egg', 'C:\\Python352\\lib\\site-packages\\xlutils-2.0.0-py3.5.egg', 'C:\\Python352\\lib\\site-packages\\xlwt-1.1.2-py3.5.egg', 'C:\\Python352\\lib\\site-packages\\pymongo-3.3.1-py3.5-win-amd64.egg', 'C:\\Python352\\lib\\site-packages\\pytz-2016.7-py3.5.egg', 'C:\\Python352\\lib\\site-packages\\zope.interface-4.3.3-py3.5-win-amd64.egg']

sys.path 返回的是一个列表！

该路径已经添加到系统的环境变量了, 当我们要添加自己的搜索目录时, 可以通过列表的append()方法；

对于模块和自己写的脚本不在同一个目录下, 在脚本开头加sys.path.append('xxx'):

**import** sys

sys.path.append(’引用模块的地址')

这种方法是运行时修改, 脚本运行后就会失效的.

另外一种方法是:

把路径添加到系统的环境变量, 或把该路径的文件夹放进已经添加到系统环境变量的路径内. 环境变量的内容会自动添加到模块搜索路径中.

4. 在pycharm右上角的run中点下拉列表, 选择run configurations出现Run/Debug Configurations.

添加Python,

Name任意,

Script为虚拟环境中\Lib\site-ackages\scrapy\cmdline.py, 如:

C:\Users\David\Envs\python3\_spider\Lib\site-packages\scrapy\cmdline.py

Script parameters为crawl spider\_name, 这里为crawl jobbole

Python interpreter为虚拟环境的python.exe.

Working directory为当前的项目目录. C:\Users\David\jobbole\_article

勾选 "Run with Python console", 否则只可能正常debug, 在Run时会出现 ModuleNotFoundError: No module named 'http.client' 的错误.

## 修改settings.py进行设置

ROBOTSTXT\_OBEY = False

DEFAULT\_REQUEST\_HEADERS = {

*# 'Accept': 'text/html,application/xhtml+xml,application/xml;q=0.9,\*/\*;q=0.8',*

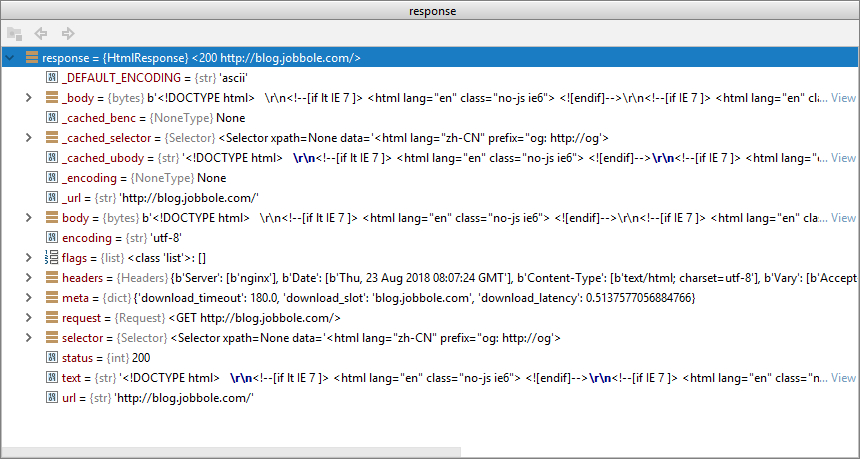
*# 'Accept-Language': 'en',*

'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/65.0.3325.181 Safari/537.36'

}

## 添加断点, 进行调试, 查看response响应的内容

在jobbole.py中的 pass 一行中点左边的空白位置设置一个断点. 然后在main.py或者jobbole.py中运行debug, 运行结束就可以看到debug的信息了. 查看jobbole.py中 def parse 中的response的内容.



## 使用scrapy提取网页的信息

### 使用xpath进行信息的提取.

在项目根目录中运行scrapy shell, 这样就可以使用项目的配置文件

scrapy shell http://blog.jobbole.com/113532/

title = response.xpath("//div[@class='entry-header']/h1/text()")

title

[<Selector xpath="//div[@class='entry-header']/h1/text()" data='一个技术大牛对程序员招聘的吐槽和建议'>]

# title是一个selector, 这样就可以直接在title的基础上进行二次的匹配.

# 使用title.extract()来获取选择器中的文本信息, 返回一个列表.

title = response.xpath("//div[@class='entry-header']/h1/text()").extract()

['一个技术大牛对程序员招聘的吐槽和建议']

# 取列表中的第1个值, 就得到了文本的信息

title = response.xpath("//div[@class='entry-header']/h1/text()").extract()[0]

title = response.xpath("//div[@class='entry-header']/h1/text()").extract\_first()

'一个技术大牛对程序员招聘的吐槽和建议'

# 获取发表时间

create\_date=response.xpath('//p[@class="entry-meta-hide-on-mobile"]')

[<Selector xpath='//p[@class="entry-meta-hide-on-mobile"]' data='<p class="entry-meta-hide-on-mobile">\r\n\r'>]

create\_date.extract()

create\_date=response.xpath('//p[@class="entry-meta-hide-on-mobile"]').extract()

['<p class="entry-meta-hide-on-mobile">\r\n\r\n 2018/01/26 · <a href="http://blog.jobbole.com/category/other/" rel="category tag">其他</a>\r\n \r\n · <a href="#article-comment"> 24 评论 </a>\r\n \r\n\r

\n \r\n · <a href="http://blog.jobbole.com/tag/%e6%8b%9b%e8%81%98/">招聘</a>, <a href="http://blog.jobbole.com/

tag/%e7%a8%8b%e5%ba%8f%e5%91%98/">程序员</a>, <a href="http://blog.jobbole.com/tag/%e8%81%8c%e5%9c%ba/">职场</a>\r\n \r\n</p

>']

create\_date=response.xpath('//p[@class="entry-meta-hide-on-mobile"]/text()')

[<Selector xpath='//p[@class="entry-meta-hide-on-mobile"]/text()' data='\r\n\r\n 2018/01/26 · '>,

<Selector xpath='//p[@class="entry-meta-hide-on-mobile"]/text()' data='\r\n \r\n '>,

<Selector xpath='//p[@class="entry-meta-hide-on-mobile"]/text()' data='\r\n \r\n\r\n \r\n '>,

<Selector xpath='//p[@class="entry-meta-hide-on-mobile"]/text()' data=', '>,

<Selector xpath='//p[@class="entry-meta-hide-on-mobile"]/text()' data=', '>,

<Selector xpath='//p[@class="entry-meta-hide-on-mobile"]/text()' data='\r\n \r\n'>]

# p标签的text只会获取p标签中的内容, 其子孙标签中的所有内容都会被忽略.

create\_date=response.xpath('//p[@class="entry-meta-hide-on-mobile"]/text()').extract()

['\r\n\r\n 2018/01/26 · ',

'\r\n \r\n · ',

'\r\n \r\n\r\n \r\n · ',

', ',

', ',

'\r\n \r\n']

# 这样就获取得发表时间. 对文本信息进行处理, 删除其中的空格.

create\_date=response.xpath('//p[@class="entry-meta-hide-on-mobile"]/text()').extract()[0].strip()

'2018/01/26 ·'

# 再使用replace

create\_date=response.xpath('//p[@class="entry-meta-hide-on-mobile"]/text()').extract()[0].strip().replace("·","").strip()

'2018/01/26'

# 要取得点赞数量

praise\_num = response.xpath("//span[contains(@class, 'vote-post-up')]/h10/text()").extract()[0]

'2'

# 获得收藏数量

fav\_num = response.xpath("//span[contains(@class, 'bookmark-btn')]/text()").extract()[0]

' 8 收藏'

import re

re.match(r".\*(\d+).\*", fav\_num).group(1)

'8'

re.match(r".\*?(\d+).\*?", fav\_num).group(1)

'8'

# 评论数

comment\_num = response.xpath("//a[@href='#article-comment']/span/text()").extract()[0]

re.match(r".\*(\d+).\*", comment\_num).group(1)

'4'

re.match(r".\*?(\d+).\*?", comment\_num).group(1)

re.match(r".\*?(\d+).\*", comment\_num).group(1)

'24'

# 文章正文, 不进行处理

content = response.xpath("//div[@class='entry']").extract()[0]

# 文章分类

tag\_list = response.xpath('//p[@class="entry-meta-hide-on-mobile"]/a/text()').extract()

# 去除掉评论数, 不能直接删除列表中的第2个元素, 因为没有评论的时候, 是不会产生1评论这个标签的.

['其他', ' 24 评论 ', '招聘', '程序员', '职场']

tag\_list = [element for element in tag\_list if not element.strip().endswith('评论')]

['其他', '招聘', '程序员', '职场']

# 把列表拼接成字符串

tags = ','.join(tag\_list)

'其他,招聘,程序员,职场'

### 使用css选择器提取网页中的元素.

scrapy shell http://blog.jobbole.com/106093/

#提取标题, 得到一个selector对象

response.css(".entry-header h1")

[<Selector xpath="descendant-or-self::\*[@class and contains(concat(' ', normalize-space(@class), ' '), ' entry-header ')]/descendant-or-self::\*/h1" data='<h1>如果有人让你推荐编程技术书，请叫他看这个列表</h1>'>]

#提取h1标签

response.css(".entry-header h1").extract()

['<h1>如果有人让你推荐编程技术书，请叫他看这个列表</h1>']

#使用::text来提取h1标签的值

response.css("p.entry-header h1::text").extract()

['顺丰菜鸟大战背后真实原因, 也许没有那么复杂']

title = response.css("div.entry-header h1::text").extract()

['如果有人让你推荐编程技术书，请叫他看这个列表']

**#发表时间**

response.css("p.entry-meta-hide-on-mobile::text").extract()

['\r\n\r\n 2016/10/30 · ',

'\r\n \r\n · ',

'\r\n \r\n\r\n \r\n · ',

'\r\n \r\n']

response.css("p.entry-meta-hide-on-mobile::text").extract()[0].strip()

'2016/10/30 ·'

response.css("p.entry-meta-hide-on-mobile::text").extract()[0].strip().replace("·","").strip()

'2016/10/30'

create\_date = response.css("p.entry-meta-hide-on-mobile::text").extract()[0].strip().replace("·", "").strip()

**#点赞数量**

praise\_num = response.css("span.vote-post-up h10::text").extract()[0]

'31'

**#收藏数量**

fav\_num = response.css("span.bookmark-btn::text").extract()[0]

' 420 收藏'

match\_re = re.match(".\*(\d+).\*", fav\_num)

if match\_re:

fav\_num = match\_re.group(1)

# 由于上面的.\*是使用的贪婪匹配的模式, 会尽量多的匹配内容, 对于收藏数量大于10的文章, 后面的(\d+)只匹配到最后一位数, 所以前面要使用非贪婪匹配的模式.

match\_re = re.match(".\*?(\d+).\*", fav\_num)

match\_re = re.match(".\*?(\d+).\*?", fav\_num)

if match\_re:

fav\_num = match\_re.group(1)

'420'

**#评论数量**

comment\_num = response.css("a[href='#article-comment'] span::text").extract()[0]

' 63 评论'

re.match(r".\*?(\d+).\*", comment\_num).group(1)

'63'

**#正文内容, 不进行处理**

content = response.css("div.entry").extract()[0]

**#标签, 取[0]得到职场, 不取的话得到所有的标签**

tag\_list = response.css("p.entry-meta-hide-on-mobile a::text").extract()

['书籍与教程', ' 63 评论 ', '书籍']

# 去掉评论的内容

tag\_list = [element for element in tag\_list if not element.strip().endswith("评论")]

tags = ",".join(tag\_list)

'书籍与教程,书籍'

### 使用extract\_first()代替extract()

使用extract()生成列表之后, 再取其中的元素, 但这样做会存在一定的风险, 如果列表为空, 就会抛出异常. 可以使用extract\_fisrt("")来直接取出其中的第1个元素.

查看extract\_first()的源码:

C:\Users\David\Envs\python3\_spider\Lib\site-packages\parsel\selector.py

def extract\_first(self, default=None):

"""

Return the result of ``.extract()`` for the first element in this list.

If the list is empty, return the default value.

"""

for x in self:

return x.extract()

else:

return default

在extract\_first()中有一个default的值. 如果提取不到, 就返回default中的值.

使用extract\_first("")就可以避免extract()中可以出现的问题了.

对于点赞数量, 如果为0的话, 使用extract()无法提取到内容, 所以要使用extract\_first(default='0')来提取

**#点赞数量**

praise\_num = response.css("span.vote-post-up h10::text").extract()

[]

praise\_num = response.css("span.vote-post-up h10::text").extract\_first(default='0')

'0'

对于评论数量和收藏数量, 如果为0的话, 使用re无法匹配到结果, 就会抛出异常, 所以要进行判断. 如 http://blog.jobbole.com/45/

**#收藏数量**

fav\_num = response.css("span.bookmark-btn::text").extract\_first('')

' 收藏'

fav\_num = re.match(".\*?(\d+).\*?", fav\_num).group(1) if re.match(".\*?(\d+).\*?", fav\_num) else '0'

**#评论数量**

comment\_num = response.css("a[href='#article-comment'] span::text").extract\_first('')

' 评论'

comment\_num = re.match(r".\*?(\d+).\*", comment\_num).group(1) if re.match(r".\*?(\d+).\*", comment\_num) else '0'

## 完善jobbole.py, 从帖子详情页中提取信息

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

**import** **scrapy**

**import** **re**

**class** **JobboleSpider**(scrapy.Spider):

name = 'jobbole'

allowed\_domains = ['blog.jobbole.com']

start\_urls = ['http://blog.jobbole.com/']

start\_urls = ['http://blog.jobbole.com/45/', 'http://blog.jobbole.com/106093/']

**def** parse(self, response):

*# 文章标题*

title = response.xpath("//div[@class='entry-header']/h1/text()").extract\_first()

title = response.css("div.entry-header h1::text").extract\_first()

*# 发表时间*

create\_date = response.xpath('//p[@class="entry-meta-hide-on-mobile"]/text()').extract\_first(default='1970/01/01').strip().replace("·", "").strip()

create\_date = response.css("p.entry-meta-hide-on-mobile::text").extract\_first(default='1970/01/01').strip().replace("·", "").strip()

*# 点赞数量*

praise\_num = response.xpath("//span[contains(@class, 'vote-post-up')]/h10/text()").extract\_first(default='0')

praise\_num = response.css("span.vote-post-up h10::text").extract\_first(default='0')

*# 收藏数量*

fav\_num\_str = response.xpath("//span[contains(@class, 'bookmark-btn')]/text()").extract\_first('')

fav\_num\_str = response.css("span.bookmark-btn::text").extract\_first('')

fav\_num = re.match(".\*?(\d+).\*?", fav\_num\_str).group(1) **if** re.match(".\*?(\d+).\*?", fav\_num\_str) **else** '0'

*# 评论数量*

comment\_num\_str = response.xpath("//a[@href='#article-comment']/span/text()").extract\_first('')

comment\_num\_str = response.css("a[href='#article-comment'] span::text").extract\_first('')

comment\_num = re.match(r".\*?(\d+).\*", comment\_num\_str).group(1) **if** re.match(r".\*?(\d+).\*", comment\_num\_str) **else** '0'

*# 文章正文, 只提取, 不做处理*

content\_str = response.xpath("//div[@class='entry']").extract\_first('')

content\_str = response.css("div.entry").extract\_first('')

content = re.sub(r"<.\*?>", "", content\_str).strip()

*# 文章标签*

tag\_list = response.xpath('//p[@class="entry-meta-hide-on-mobile"]/a/text()').extract()

tag\_list = response.css("p.entry-meta-hide-on-mobile a::text").extract()

*# 去除掉评论数, 不能直接删除列表中的第2个元素, 因为没有评论的时候, 是不会产生1评论这个标签的.*

tag\_list = [element **for** element **in** tag\_list **if** **not** element.strip().endswith('评论')]

*# 把列表拼接成字符串*

tags = ','.join(tag\_list)

article = {

"title": title,

"create\_date": create\_date,

"praise\_num": praise\_num,

"fav\_num": fav\_num,

"comment\_num": comment\_num,

"content": content,

"tags": tags

}

**print**(article)

**pass**

添加断点进行调试

在xpath的fav\_num和match\_re, 以及pass处添加断点进行调试.

## 调试工具Stepping toolbar的使用

|  |  |  |
| --- | --- | --- |
| **Item** | **Tooltip and Shortcut** | **Description** |
| icons debugger showCurrentFrame svg | Show Execution Point  Alt+F10 | Click this button to highlight the current execution point in the editor andshow the corresponding stack frame in the **Frames**pane.  高亮显示当前执点 |
| icons actions traceOver svg | Step Over  F8 | Click this button to execute the program until the next line in the current method or file, skipping the methods referenced at the current execution point (if any). If the current line is the last one in the method, execution steps to the line executed right after this method.  执行程序直到当前方法或文件的下一行 |
| icons actions traceInto svg | Step Into  F7 | Click this button to have the debugger step into the method called at the current execution point.  跳转到当前执行点调用的方法中 |
| step into my code | Step Into My Code  Shift+Alt+F7 | Click this button to skip stepping into library sources and keep focused on your own code.  跳过程序源代码的执行过程, 进入到自己的代码中 |
| icons actions stepOut svg | Step Out  Shift+F8 | Click this button to have the debugger step out of the current method, to the line executed right after it.  跳出到当前正在执行的方法, 进入到当前正在执行的下一行中 |
| icons actions popFrame svg | Drop frame | Interrupts execution and returns to the initial point of method execution. In the process, it drops the current method frames from the stack.  中断当前方法的执行, 跳回到正在执行的方法的起始点 |
| icons actions runToCursor svg | Run to Cursor Alt+F9 | Click this button to resume program execution and pause until the execution point reaches the line at the current cursor location in the editor. No breakpoint is required. Actually, there is a temporary breakpoint set for the current line at the caret, which is removed once program execution is paused. Thus, if the caret is positioned at the line which has already been executed, the program will be just resumed for further execution, because there is no way to roll back to previous breakpoints. This action is especially useful when you have stepped deep into the methods sequence and need to step out of several methods at once.  If there are breakpoints set for the lines that should be executed before bringing you to the specified line, the debugger will pause at the first encountered breakpoint.  继续程序的执行直到光标所在位置, 光标所在处不需要断点. 实际上, 在当前光标所在位置处会自动生成一个临时的断点, 当执行暂停时会自动删除这个断点. |
| icons debugger evaluateExpression svg | Evaluate Expression  Alt+F8 | Click this button to open the [Evaluate Expression](https://www.jetbrains.com/help/pycharm/evaluate-expression.html) dialog. |

|  |  |  |
| --- | --- | --- |
| F8 | step over | 程序在第1个断点处暂停, F8单行执行 |
| F9 | resume program | 执行到下一个断点处 |
| F7 | step into | 进入函数中执行, 如果没有函数, 单行执行 |
| Alt + shift + F9 | step into my code | 执行到自己的代码中 |
| shift + F8 | setp out | 跳出当前函数的执行, 进入到下一个函数中 |
| Alt + F9 | run to cursor | 执行到光标所在处 |

## 从文章列表页爬取所有文章

伯乐在线的所有文章可以从 <http://blog.jobbole.com/all-posts/> 获取到

start\_urls中网址的响应是由parse函数进行处理的, 想要爬取jobbole列表页的所有文章, 需要把start\_urls修改为 <http://blog.jobbole.com/all-posts/> , 同时, 把parse函数的功能修改为从列表页提取文章详情页的链接. 从文章详情页中提取文章详情的功能放在parse\_detail()函数中实现.

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

**import** **scrapy**

**import** **re**

**class** **JobboleSpider**(scrapy.Spider):

name = 'jobbole'

allowed\_domains = ['blog.jobbole.com']

*# start\_urls = ['http://blog.jobbole.com/45/', 'http://blog.jobbole.com/106093/']*

start\_urls = ['http://blog.jobbole.com/all-posts/']

**def** parse(self, response):

*# 从文章列表页获取详情页url*

*# 注意要添加#archive这个id, 不然的话就会取到很多不是文章列表中的url地址. id是唯一的, 就不需要再添加标签名了.*

post\_urls = response.css("a.archive-title::attr(href)").extract()

post\_urls = response.xpath("//a[@class='archive-title']/@href").extract()

**for** post\_url **in** post\_urls:

**print**(post\_url)

*# 构建请求对象, 把提取到的文章详情页的url交给parse\_detail这个解析函数进行处理. 因为所有文章都可以从all-posts这个列表页获取到, 这里就不需要对链接进行跟进了*

*# yield scrapy.http.Request(post\_url, callback=self.parse\_detail)*

**yield** scrapy.Request(post\_url, callback=self.parse\_detail)

*# 测试代码*

**break**

**def** parse\_detail(self, response):

*# 从文章详情页提取信息*

*# 文章标题*

title = response.xpath("//div[@class='entry-header']/h1/text()").extract\_first()

title = response.css("div.entry-header h1::text").extract\_first()

*# 发表时间*

create\_date = response.xpath('//p[@class="entry-meta-hide-on-mobile"]/text()').extract\_first(default='1970/01/01').strip().replace("·", "").strip()

create\_date = response.css("p.entry-meta-hide-on-mobile::text").extract\_first(default='1970/01/01').strip().replace("·", "").strip()

*# 点赞数量*

praise\_num = response.xpath("//span[contains(@class, 'vote-post-up')]/h10/text()").extract\_first(default='0')

praise\_num = response.css("span.vote-post-up h10::text").extract\_first(default='0')

*# 收藏数量*

fav\_num\_str = response.xpath("//span[contains(@class, 'bookmark-btn')]/text()").extract\_first('')

fav\_num\_str = response.css("span.bookmark-btn::text").extract\_first('')

fav\_num = re.match(".\*?(\d+).\*?", fav\_num\_str).group(1) **if** re.match(".\*?(\d+).\*?", fav\_num\_str) **else** '0'

*# 评论数量*

comment\_num\_str = response.xpath("//a[@href='#article-comment']/span/text()").extract\_first('')

comment\_num\_str = response.css("a[href='#article-comment'] span::text").extract\_first('')

comment\_num = re.match(r".\*?(\d+).\*", comment\_num\_str).group(1) **if** re.match(r".\*?(\d+).\*", comment\_num\_str) **else** '0'

*# 文章正文, 除掉所有的标签*

content\_str = response.xpath("//div[@class='entry']").extract\_first('')

content\_str = response.css("div.entry").extract\_first('')

content = re.sub(r"<.\*?>", "", content\_str).strip()

*# 文章标签*

tag\_list = response.xpath('//p[@class="entry-meta-hide-on-mobile"]/a/text()').extract()

tag\_list = response.css("p.entry-meta-hide-on-mobile a::text").extract()

*# 去除掉评论数, 不能直接删除列表中的第2个元素, 因为没有评论的时候, 是不会产生1评论这个标签的.*

tag\_list = [element **for** element **in** tag\_list **if** **not** element.strip().endswith('评论')]

*# 把列表拼接成字符串*

tags = ','.join(tag\_list)

article = {

"title": title,

"create\_date": create\_date,

"praise\_num": praise\_num,

"fav\_num": fav\_num,

"comment\_num": comment\_num,

"content": content,

"tags": tags

}

**print**(article)

**pass**

添加断点, 进行调试

## 提取下一页的链接, 构造并发送请求

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

**import** **scrapy**

**import** **re**

**class** **JobboleSpider**(scrapy.Spider):

name = 'jobbole'

allowed\_domains = ['blog.jobbole.com']

*# start\_urls = ['http://blog.jobbole.com/45/', 'http://blog.jobbole.com/106093/']*

start\_urls = ['http://blog.jobbole.com/all-posts/']

**def** parse(self, response):

*# 从文章列表页获取详情页url*

*# 注意要添加#archive这个id, 不然的话就会取到很多不是文章列表中的url地址. id是唯一的, 就不需要再添加标签名了.*

post\_urls = response.css("a.archive-title::attr(href)").extract()

post\_urls = response.xpath("//a[@class='archive-title']/@href").extract()

**for** post\_url **in** post\_urls:

**print**(post\_url)

*# 构建请求对象, 把提取到的文章详情页的url交给parse\_detail这个解析函数进行处理. 因为所有文章都可以从all-posts这个列表页获取到, 这里就不需要对链接进行跟进了*

*# yield scrapy.http.Request(post\_url, callback=self.parse\_detail)*

**yield** scrapy.Request(post\_url, callback=self.parse\_detail)

*# 测试代码*

**break**

*#提取出了包含下一页的a标签. 使用extract\_first就可以避免在最后一页时出错.*

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

*# .next和.page-numbers是属于同一个div的class.*

next\_url = response.css('a.next.page-numbers::attr(href)').extract\_first('')

*#如果存在下一页的这个值, 就构建Request对象. 在最后一页时这个值不存在.*

**if** next\_url:

*#使用回调函数把下一页的url传回到parse函数中重复进行提取. 不用写成self.parse(). scrapy是基于异步处理twisted完成的, 会自动根据函数名来调用函数.*

**yield** scrapy.Request(url=next\_url, callback=self.parse)

**def** parse\_detail(self, response):

*# 从文章详情页提取信息*

*# 文章标题*

title = response.xpath("//div[@class='entry-header']/h1/text()").extract\_first()

title = response.css("div.entry-header h1::text").extract\_first()

*# 发表时间*

create\_date = response.xpath('//p[@class="entry-meta-hide-on-mobile"]/text()').extract\_first(default='1970/01/01').strip().replace("·", "").strip()

create\_date = response.css("p.entry-meta-hide-on-mobile::text").extract\_first(default='1970/01/01').strip().replace("·", "").strip()

*# 点赞数量*

praise\_num = response.xpath("//span[contains(@class, 'vote-post-up')]/h10/text()").extract\_first(default='0')

praise\_num = response.css("span.vote-post-up h10::text").extract\_first(default='0')

*# 收藏数量*

fav\_num\_str = response.xpath("//span[contains(@class, 'bookmark-btn')]/text()").extract\_first('')

fav\_num\_str = response.css("span.bookmark-btn::text").extract\_first('')

fav\_num = re.match(r".\*?(\d+).\*?", fav\_num\_str).group(1) **if** re.match(r".\*?(\d+).\*?", fav\_num\_str) **else** '0'

*# 评论数量*

comment\_num\_str = response.xpath("//a[@href='#article-comment']/span/text()").extract\_first('')

comment\_num\_str = response.css("a[href='#article-comment'] span::text").extract\_first('')

comment\_num = re.match(r".\*?(\d+).\*", comment\_num\_str).group(1) **if** re.match(r".\*?(\d+).\*",

comment\_num\_str) **else** '0'

*# 文章正文, 除掉所有的标签*

content\_str = response.xpath("//div[@class='entry']").extract\_first('')

content\_str = response.css("div.entry").extract\_first('')

content = re.sub(r"<.\*?>", "", content\_str).strip()

*# 文章标签*

tag\_list = response.xpath('//p[@class="entry-meta-hide-on-mobile"]/a/text()').extract()

tag\_list = response.css("p.entry-meta-hide-on-mobile a::text").extract()

*# 去除掉评论数, 不能直接删除列表中的第2个元素, 因为没有评论的时候, 是不会产生1评论这个标签的.*

tag\_list = [element **for** element **in** tag\_list **if** **not** element.strip().endswith('评论')]

*# 把列表拼接成字符串*

tags = ','.join(tag\_list)

article = {

"title": title,

"create\_date": create\_date,

"praise\_num": praise\_num,

"fav\_num": fav\_num,

"comment\_num": comment\_num,

"content": content,

"tags": tags

}

**print**(article)

**pass**

## 使用item类

使用scrapy中的items把提取的内容保存到数据库中

数据爬取的目的就是从非结构化的数据中提取到结构化的数据. 最简单的是把所有字段放到字典中进行保存, 但字典中缺少结构化的数据, 必须要手动定义, 这样会比较容易出错. scrapy中定义了item类, 可以使用item类来指定字段.

在对items类进行实例化之后, 在spider中对它进行yield之后, scrapy会把这个items传递到pipeline中, 就可以在pipeline中集中进行数据的去重, 保存等操作.

定义items.py

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

*# Define here the models for your scraped items*

*#*

*# See documentation in:*

*# https://doc.scrapy.org/en/latest/topics/items.html*

**import** **scrapy**

**class** **JobboleArticleItem**(scrapy.Item):

title = scrapy.Field()

create\_date = scrapy.Field()

*# 文章详情的url*

article\_url = scrapy.Field()

*#url是变长的, 通过md5把它变成固定长度的.*

url\_object\_id = scrapy.Field()

*#封面图的url地址*

front\_image\_url = scrapy.Field()

*#封面图在本地保存的路径.*

front\_image\_path = scrapy.Field()

praise\_num = scrapy.Field()

fav\_num = scrapy.Field()

comment\_num = scrapy.Field()

content = scrapy.Field()

tags = scrapy.Field()

修改jobbole.py, 实例化item对象, 并把提取的详情信息传递到item中.

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

**import** **scrapy**

**import** **re**

**from** **jobbole\_article.items** **import** JobboleArticleItem

**class** **JobboleSpider**(scrapy.Spider):

name = 'jobbole'

allowed\_domains = ['blog.jobbole.com']

*# start\_urls = ['http://blog.jobbole.com/45/', 'http://blog.jobbole.com/106093/']*

start\_urls = ['http://blog.jobbole.com/all-posts/']

**def** parse(self, response):

*# 从文章列表页获取详情页url*

article\_nodes = response.xpath('//div[@class="grid-8"]/div[not(contains(@class, "navigation"))]')

article\_nodes = response.xpath('//div[@class="post floated-thumb"]')

article\_nodes = response.css('div.post.floated-thumb')

**for** article\_node **in** article\_nodes:

*# 列表页文章图片*

front\_image\_url = article\_node.xpath('.//img/@src').extract\_first("")

front\_image\_url = article\_node.css('img::attr(src)').extract\_first("")

*# 列表页文章链接地址*

article\_url = response.css("a.archive-title::attr(href)").extract\_first("")

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

*# 构建请求对象, 把提取到的文章详情页的url交给parse\_detail这个解析函数进行处理. 因为所有文章都可以从all-posts这个列表页获取到, 这里就不需要对链接进行跟进了*

*# meta是字典dict类型的, 通过在Request请求对象中添加meta信息来在不同的解析函数之间传递数据.*

**yield** scrapy.Request(url=article\_url, meta={'front\_image\_url': front\_image\_url}, callback=self.parse\_detail)

*# 测试代码*

**break**

*#提取出了包含下一页的a标签. 使用extract\_first就可以避免在最后一页时出错.*

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

*# .next和.page-numbers是属于同一个div的class.*

next\_url = response.css('a.next.page-numbers::attr(href)').extract\_first('')

*#如果存在下一页的这个值, 就构建Request对象. 在最后一页时这个值不存在.*

**if** next\_url:

*#使用回调函数把下一页的url传回到parse函数中重复进行提取. 不用写成self.parse(). scrapy是基于异步处理twisted完成的, 会自动根据函数名来调用函数.*

**yield** scrapy.Request(url=next\_url, callback=self.parse)

**def** parse\_detail(self, response):

*# 从文章详情页提取信息*

*# 从response中的meta中取出文章封面图信息*

front\_image\_url = response.meta.get('front\_image\_url', '')

*# 文章标题*

title = response.xpath("//div[@class='entry-header']/h1/text()").extract\_first()

title = response.css("div.entry-header h1::text").extract\_first()

*# 发表时间*

create\_date = response.xpath('//p[@class="entry-meta-hide-on-mobile"]/text()').extract\_first(default='1970/01/01').strip().replace("·", "").strip()

create\_date = response.css("p.entry-meta-hide-on-mobile::text").extract\_first(default='1970/01/01').strip().replace("·", "").strip()

*# 点赞数量*

praise\_num = response.xpath("//span[contains(@class, 'vote-post-up')]/h10/text()").extract\_first(default='0')

praise\_num = response.css("span.vote-post-up h10::text").extract\_first(default='0')

*# 收藏数量*

fav\_num\_str = response.xpath("//span[contains(@class, 'bookmark-btn')]/text()").extract\_first('')

fav\_num\_str = response.css("span.bookmark-btn::text").extract\_first('')

fav\_num = re.match(r".\*?(\d+).\*?", fav\_num\_str).group(1) **if** re.match(r".\*?(\d+).\*?", fav\_num\_str) **else** '0'

*# 评论数量*

comment\_num\_str = response.xpath("//a[@href='#article-comment']/span/text()").extract\_first('')

comment\_num\_str = response.css("a[href='#article-comment'] span::text").extract\_first('')

comment\_num = re.match(r".\*?(\d+).\*", comment\_num\_str).group(1) **if** re.match(r".\*?(\d+).\*", comment\_num\_str) **else** '0'

*# 文章正文, 除掉所有的标签*

content\_str = response.xpath("//div[@class='entry']").extract\_first('')

content\_str = response.css("div.entry").extract\_first('')

content = re.sub(r"<.\*?>", "", content\_str).strip()

*# 文章标签*

tag\_list = response.xpath('//p[@class="entry-meta-hide-on-mobile"]/a/text()').extract()

tag\_list = response.css("p.entry-meta-hide-on-mobile a::text").extract()

*# 去除掉评论数, 不能直接删除列表中的第2个元素, 因为没有评论的时候, 是不会产生1评论这个标签的.*

tag\_list = [element **for** element **in** tag\_list **if** **not** element.strip().endswith('评论')]

*# 把列表拼接成字符串*

tags = ','.join(tag\_list)

*#实例化article对象*

article\_item = JobboleArticleItem(

title = title,

create\_date = create\_date,

article\_url = response.url,

front\_image\_url = front\_image\_url,

praise\_num = praise\_num,

fav\_num = fav\_num,

comment\_num = comment\_num,

content = content,

tags = tags

)

**yield** article\_item

在settings.py 中启动 ITEM\_PIPELINES

ITEM\_PIPELINES = {

'jobbole\_article.pipelines.JobboleArticlePipeline': 300,

}

piplines.py

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

**class** **JobboleArticlePipeline**(object):

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

**return** item

在pipelines.py中的def process\_item和reture item处添加断点, debug, 查看是否能传递过来item信息.

在item中有一个\_values的信息, 其中就包含了之前在jobbole爬虫中提取的信息.

可以在pipeline中进行信息的处理和保存.

## 列表页图片url的获取

获取列表页的图片. 在用户没有自定义列表页图片时, 网站会自动获取正文中的第一张图片来作为列表页的缩略图.

可以直接在列表页获取图片的url地址.

对于列表页的每一篇文章, 不再直接获取其详情的url地址了, 而是对每一个文章的div进行分组, 即获取详情url和封面图url的父节点, 再父节点的基础上再进行选取, 得到详情url和封面图的url, 这样在某个内容缺失的情况下就不会出现匹配错误的情况.

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

**import** **scrapy**

**import** **re**

**class** **JobboleSpider**(scrapy.Spider):

name = 'jobbole'

allowed\_domains = ['blog.jobbole.com']

*# start\_urls = ['http://blog.jobbole.com/45/', 'http://blog.jobbole.com/106093/']*

start\_urls = ['http://blog.jobbole.com/all-posts/']

**def** parse(self, response):

*# 从文章列表页获取详情页url*

article\_nodes = response.xpath('//div[@class="grid-8"]/div[not(contains(@class, "navigation"))]')

article\_nodes = response.xpath('//div[@class="post floated-thumb"]')

**for** article\_node **in** article\_nodes:

*# 列表页文章图片*

front\_image\_url = article\_node.xpath('.//img/@src').extract\_first("")

front\_image\_url = article\_node.css('img::attr(src)').extract\_first("")

*# 列表页文章链接地址*

article\_url = response.css("a.archive-title::attr(href)").extract\_first("")

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

*# 构建请求对象, 把提取到的文章详情页的url交给parse\_detail这个解析函数进行处理. 因为所有文章都可以从all-posts这个列表页获取到, 这里就不需要对链接进行跟进了*

*# meta是字典dict类型的, 通过在Request请求对象中添加meta信息来在不同的解析函数之间传递数据.*

**yield** scrapy.Request(url=article\_url, meta={'front\_image\_url': front\_image\_url}, callback=self.parse\_detail)

*# 测试代码*

**break**

*#提取出了包含下一页的a标签. 使用extract\_first就可以避免在最后一页时出错.*

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

*# .next和.page-numbers是属于同一个div的class.*

next\_url = response.css('a.next.page-numbers::attr(href)').extract\_first('')

*#如果存在下一页的这个值, 就构建Request对象. 在最后一页时这个值不存在.*

**if** next\_url:

*#使用回调函数把下一页的url传回到parse函数中重复进行提取. 不用写成self.parse(). scrapy是基于异步处理twisted完成的, 会自动根据函数名来调用函数.*

**yield** scrapy.Request(url=next\_url, callback=self.parse)

**def** parse\_detail(self, response):

*# 从文章详情页提取信息*

*# 从response中的meta中取出文章封面图信息*

front\_image\_url = response.meta.get('front\_image\_url', '')

*# 文章标题*

title = response.xpath("//div[@class='entry-header']/h1/text()").extract\_first()

title = response.css("div.entry-header h1::text").extract\_first()

*# 发表时间*

create\_date = response.xpath('//p[@class="entry-meta-hide-on-mobile"]/text()').extract\_first(default='1970/01/01').strip().replace("·", "").strip()

create\_date = response.css("p.entry-meta-hide-on-mobile::text").extract\_first(default='1970/01/01').strip().replace("·", "").strip()

*# 点赞数量*

praise\_num = response.xpath("//span[contains(@class, 'vote-post-up')]/h10/text()").extract\_first(default='0')

praise\_num = response.css("span.vote-post-up h10::text").extract\_first(default='0')

*# 收藏数量*

fav\_num\_str = response.xpath("//span[contains(@class, 'bookmark-btn')]/text()").extract\_first('')

fav\_num\_str = response.css("span.bookmark-btn::text").extract\_first('')

fav\_num = re.match(r".\*?(\d+).\*?", fav\_num\_str).group(1) **if** re.match(r".\*?(\d+).\*?", fav\_num\_str) **else** '0'

*# 评论数量*

comment\_num\_str = response.xpath("//a[@href='#article-comment']/span/text()").extract\_first('')

comment\_num\_str = response.css("a[href='#article-comment'] span::text").extract\_first('')

comment\_num = re.match(r".\*?(\d+).\*", comment\_num\_str).group(1) **if** re.match(r".\*?(\d+).\*", comment\_num\_str) **else** '0'

*# 文章正文, 除掉所有的标签*

content\_str = response.xpath("//div[@class='entry']").extract\_first('')

content\_str = response.css("div.entry").extract\_first('')

content = re.sub(r"<.\*?>", "", content\_str).strip()

*# 文章标签*

tag\_list = response.xpath('//p[@class="entry-meta-hide-on-mobile"]/a/text()').extract()

tag\_list = response.css("p.entry-meta-hide-on-mobile a::text").extract()

*# 去除掉评论数, 不能直接删除列表中的第2个元素, 因为没有评论的时候, 是不会产生1评论这个标签的.*

tag\_list = [element **for** element **in** tag\_list **if** **not** element.strip().endswith('评论')]

*# 把列表拼接成字符串*

tags = ','.join(tag\_list)

article = {

"title": title,

"article\_url": response.url,

"front\_image\_url": front\_image\_url,

"create\_date": create\_date,

"praise\_num": praise\_num,

"fav\_num": fav\_num,

"comment\_num": comment\_num,

"content": content,

"tags": tags

}

**print**(article)

**pass**

添加断点, 进行测试. 查看parse\_detail中是否能够取到front\_image\_url的值.

## 下载文章的封面图片

想要把文章的封面图片下载下来, 并保存到本地, scrapy提供了一种自动下载图片的方法, 只需要在settings中的ITEM\_PIPELINES中添加一个images的item\_pipline即可完成图片的下载.

C:\Users\David\Envs\python3\_spider\Lib\site-packages\scrapy\pipelines.images.py

# 从items.py中传递过来的数据都要经过ITEM\_PIPELINES这个管道的处理, 数字设置的越小越早经过这个pipeline.

在settings.py中启动ImagesPipeline, 并添加图片下载的信息

*# Configure item pipelines*

*# See https://doc.scrapy.org/en/latest/topics/item-pipeline.html*

ITEM\_PIPELINES = {

'jobbole\_article.pipelines.JobboleArticlePipeline': 300,

'scrapy.pipelines.images.ImagesPipeline' : 200

}

*################################################################*

*# 图片下载设置*

*# 设置图片url的字段, scrapy将从item中找出此字段进行图片下载*

IMAGES\_URLS\_FIELD = "front\_image\_url"

*# 设置图片下载保存的目录, 这里不直接使用本机的绝对路径, 而是通过程序获取路径. 这样在程序迁移后也能够正常运行.*

**import** **os**

*# os.path.dirname(\_\_file\_\_)获取当前文件所在的文件夹名称.*

*# os.path.abspath(os.path.dirname(\_\_file\_\_))获得当前文件所在的绝对路径.*

project\_path = os.path.dirname(os.path.abspath(\_\_file\_\_))

*# 想要把下载的图片保存在与settings同目录的 images 文件夹中, 要先在项目中新建此文件夹.*

*# IMAGES\_STORE中定义保存图片的路径*

IMAGES\_STORE = os.path.join(project\_path, "images")

*# 表示只下载大于100x100的图片, 查看images.py的源码, 程序会自动的从settings.py中读取设置的 IMAGES\_MIN\_HEIGHT 和 IMAGES\_MIN\_WIDTH 值*

*# IMAGES\_MIN\_HEIGHT = 100*

*# IMAGES\_MIN\_WIDTH = 100*

**if** **not** os.path.exists(IMAGES\_STORE):

os.mkdir(IMAGES\_STORE)

**else**:

**pass**

*################################################################*

之后运行项目可能包PIL未找到, 因此需要pip install pillow

运行爬虫, 出现错误

raise ValueError('Missing scheme in request url: %s' % self.\_url)

ValueError: Missing scheme in request url: h

scrapy的图片下载默认是接受一个数组/列表, 因此jobbole.py要修改为:

front\_image\_url = [front\_image\_url]

*#实例化article对象*

article\_item = JobboleArticleItem(

title = title,

create\_date = create\_date,

article\_url = response.url,

front\_image\_url = [front\_image\_url],

praise\_num = praise\_num,

fav\_num = fav\_num,

comment\_num = comment\_num,

content = content,

tags = tags

)

运行spider, 可能会出现以下两个错误,

ModuleNotFoundError: No module named 'win32api'

Pip install pywin32

ModuleNotFoundError: No module named 'PIL'

Pip install pillow

再次运行爬虫, 就能够下载到图片了. 现在图片已经保存在本地了.

## 自定义pipline获取下载的图片保存的路径.

要提取到图片保存的路径, 与对应的文章绑定在一起, 把它放在front\_image\_path这个字段中. 需要自定义pipeline, 继承系统的ImagePipeline.

### 查看images.ImagesPipeline的源码

"C:\Users\David\Envs\python3\_spider\Lib\site-packages\scrapy\pipelines\images.py"

**class** **ImagesPipeline**(FilesPipeline):

*"""Abstract pipeline that implement the image thumbnail generation logic*

*"""*

MEDIA\_NAME = 'image'

*# Uppercase attributes kept for backward compatibility with code that subclasses*

*# ImagesPipeline. They may be overridden by settings.*

MIN\_WIDTH = 0

MIN\_HEIGHT = 0

EXPIRES = 90

THUMBS = {}

DEFAULT\_IMAGES\_URLS\_FIELD = 'image\_urls'

DEFAULT\_IMAGES\_RESULT\_FIELD = 'images'

**def** \_\_init\_\_(self, store\_uri, download\_func=None, settings=None):

super(ImagesPipeline, self).\_\_init\_\_(store\_uri, settings=settings,

download\_func=download\_func)

**if** isinstance(settings, dict) **or** settings **is** None:

settings = Settings(settings)

resolve = functools.partial(self.\_key\_for\_pipe,

base\_class\_name="ImagesPipeline",

settings=settings)

self.expires = settings.getint(

resolve("IMAGES\_EXPIRES"), self.EXPIRES

)

**if** **not** hasattr(self, "IMAGES\_RESULT\_FIELD"):

self.IMAGES\_RESULT\_FIELD = self.DEFAULT\_IMAGES\_RESULT\_FIELD

**if** **not** hasattr(self, "IMAGES\_URLS\_FIELD"):

self.IMAGES\_URLS\_FIELD = self.DEFAULT\_IMAGES\_URLS\_FIELD

self.images\_urls\_field = settings.get(

resolve('IMAGES\_URLS\_FIELD'),

self.IMAGES\_URLS\_FIELD

)

self.images\_result\_field = settings.get(

resolve('IMAGES\_RESULT\_FIELD'),

self.IMAGES\_RESULT\_FIELD

)

self.min\_width = settings.getint(

resolve('IMAGES\_MIN\_WIDTH'), self.MIN\_WIDTH

)

self.min\_height = settings.getint(

resolve('IMAGES\_MIN\_HEIGHT'), self.MIN\_HEIGHT

)

self.thumbs = settings.get(

resolve('IMAGES\_THUMBS'), self.THUMBS

)

@classmethod

**def** from\_settings(cls, settings):

s3store = cls.STORE\_SCHEMES['s3']

s3store.AWS\_ACCESS\_KEY\_ID = settings['AWS\_ACCESS\_KEY\_ID']

s3store.AWS\_SECRET\_ACCESS\_KEY = settings['AWS\_SECRET\_ACCESS\_KEY']

s3store.POLICY = settings['IMAGES\_STORE\_S3\_ACL']

gcs\_store = cls.STORE\_SCHEMES['gs']

gcs\_store.GCS\_PROJECT\_ID = settings['GCS\_PROJECT\_ID']

store\_uri = settings['IMAGES\_STORE']

**return** cls(store\_uri, settings=settings)

**def** file\_downloaded(self, response, request, info):

**return** self.image\_downloaded(response, request, info)

**def** image\_downloaded(self, response, request, info):

checksum = None

**for** path, image, buf **in** self.get\_images(response, request, info):

**if** checksum **is** None:

buf.seek(0)

checksum = md5sum(buf)

width, height = image.size

self.store.persist\_file(

path, buf, info,

meta={'width': width, 'height': height},

headers={'Content-Type': 'image/jpeg'})

**return** checksum

**def** get\_images(self, response, request, info):

path = self.file\_path(request, response=response, info=info)

orig\_image = Image.open(BytesIO(response.body))

width, height = orig\_image.size

**if** width < self.min\_width **or** height < self.min\_height:

**raise** ImageException("Image too small (**%d**x**%d** < **%d**x**%d**)" %

(width, height, self.min\_width, self.min\_height))

image, buf = self.convert\_image(orig\_image)

**yield** path, image, buf

**for** thumb\_id, size **in** six.iteritems(self.thumbs):

thumb\_path = self.thumb\_path(request, thumb\_id, response=response, info=info)

thumb\_image, thumb\_buf = self.convert\_image(image, size)

**yield** thumb\_path, thumb\_image, thumb\_buf

**def** convert\_image(self, image, size=None):

**if** image.format == 'PNG' **and** image.mode == 'RGBA':

background = Image.new('RGBA', image.size, (255, 255, 255))

background.paste(image, image)

image = background.convert('RGB')

**elif** image.mode == 'P':

image = image.convert("RGBA")

background = Image.new('RGBA', image.size, (255, 255, 255))

background.paste(image, image)

image = background.convert('RGB')

**elif** image.mode != 'RGB':

image = image.convert('RGB')

**if** size:

image = image.copy()

image.thumbnail(size, Image.ANTIALIAS)

buf = BytesIO()

image.save(buf, 'JPEG')

**return** image, buf

*# 在scrapy.pipelines.images.ImagesPipeline中有一个get\_media\_requests函数, 这个函数是用for循环的方式处理的, 所以之前在对图片封面进行赋值的时候要使用一个数组, 从数组中取得地址, 使用Request发送这个地址的请求, 交给scrapy下载器进行下载.*

**def** get\_media\_requests(self, item, info):

**return** [Request(x) **for** x **in** item.get(self.images\_urls\_field, [])]

*# 使用item\_completed这个函数能够获取到图片在本地的保存路径, 图片的路径是保存在result这个参数中的, 需要对这个函数进行重写.*

**def** item\_completed(self, results, item, info):

**if** isinstance(item, dict) **or** self.images\_result\_field **in** item.fields:

item[self.images\_result\_field] = [x **for** ok, x **in** results **if** ok]

**return** item

**def** file\_path(self, request, response=None, info=None):

*## start of deprecation warning block (can be removed in the future)*

**def** \_warn():

**from** **scrapy.exceptions** **import** ScrapyDeprecationWarning

**import** **warnings**

warnings.warn('ImagesPipeline.image\_key(url) and file\_key(url) methods are deprecated, '

'please use file\_path(request, response=None, info=None) instead',

category=ScrapyDeprecationWarning, stacklevel=1)

*# check if called from image\_key or file\_key with url as first argument*

**if** **not** isinstance(request, Request):

\_warn()

url = request

**else**:

url = request.url

*# detect if file\_key() or image\_key() methods have been overridden*

**if** **not** hasattr(self.file\_key, '\_base'):

\_warn()

**return** self.file\_key(url)

**elif** **not** hasattr(self.image\_key, '\_base'):

\_warn()

**return** self.image\_key(url)

*## end of deprecation warning block*

image\_guid = hashlib.sha1(to\_bytes(url)).hexdigest() *# change to request.url after deprecation*

**return** 'full/**%s**.jpg' % (image\_guid)

### 修改pipelines.py, 自定义图片下载管道

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

**from** **scrapy.pipelines.images** **import** ImagesPipeline

**class** **JobboleArticlePipeline**(object):

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

**return** item

*# 自定义图片下载处理的中间件*

**class** **JobboleImagePipeline**(ImagesPipeline):

*# 重写函数, 改写item处理完成的函数*

**def** item\_completed(self, results, item, info):

**pass**

### debug获取results的结构

result的结构不清楚, 可以使用断点的方式进行调试. 把images下载的所有图片都删除.

修改settings中的ITEM\_PIPELINES, 使用我们自定义的图片下载的管道.

ITEM\_PIPELINES = {

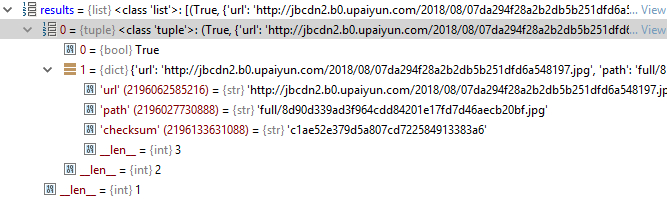
'jobbole\_article.pipelines.JobboleArticlePipeline': 300,

*# 'scrapy.pipelines.images.ImagesPipeline' : 200,*

'jobbole\_article.pipelines.JobboleImagePipeline': 200,

}

在pipelines.py中的pass添加断点, 进行调试, 查看result的结构, 可以看到它是一个list. list中的每一个元素都是一个tuple, tuple中第1个值是一个bool, 表示它有没有成功, 第2个值是一个字典, 字典中有一个path属性, 保存着下载的image文件的路径. 因为使用的是yield方法, 所以这里的result中只有一个值.



<class 'list'>: [(True, {'url': 'http://jbcdn2.b0.upaiyun.com/2017/12/a358ba4f51dd7f1dac9b1dfc1fef5343.png', 'path': 'full/bbf5b2d33074c754cd78cbcc77e858e9cd022815.jpg', 'checksum': '3957541cf02a7e430d69c8dd5d4e8367'})]

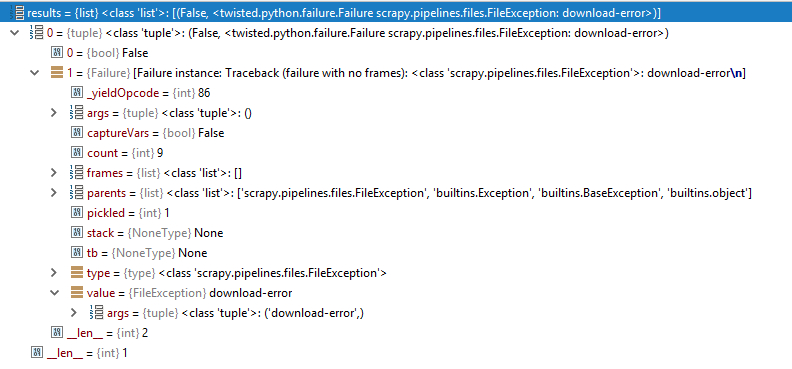
results = [(True, {'url': 'http://jbcdn2.b0.upaiyun.com/2017/12/a358ba4f51dd7f1dac9b1dfc1fef5343.png', 'path': 'full/bbf5b2d33074c754cd78cbcc77e858e9cd022815.jpg', 'checksum': '3957541cf02a7e430d69c8dd5d4e8367'})]

**for** key, value **in** results:

front\_image\_path = value.get('path','')

**print**(front\_image\_path)

但是当图片下载出错时, 如在解析 <http://blog.jobbole.com/113396/> 这个页面时, 图片无法下载, results中保存的就是出错的信息,



results = [(False, <twisted.python.failure.Failure scrapy.pipelines.files.FileException: download-error>)]

此时就无法从results中提取出path的值了, 所以在这里要进行判断.

**for** key, value **in** results:

**if** key == True:

front\_image\_path = value.get('path','')

**elif** key == False:

front\_image\_path = ''

**print**(front\_image\_path)

或者:

**for** key, value **in** results:

**try**:

front\_image\_path = value.get('path','')

**except** **Exception** **as** e:

front\_image\_path = ''

**print**(front\_image\_path)

### 修改pipelines.py, 从results中取出图片保存的路径

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

**from** **scrapy.pipelines.images** **import** ImagesPipeline

**class** **JobboleArticlePipeline**(object):

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

**return** item

*# 自定义图片下载处理的中间件*

**class** **JobboleImagePipeline**(ImagesPipeline):

*# 重写函数, 改写item处理完成的函数*

**def** item\_completed(self, results, item, info):

*# result是一个list的结构. 可以获取多个图片保存的信息. 但由于使用yield, 一次只传递过一个item, 所以这里的result中只有一个元素.*

**for** key, value **in** results:

**try**:

front\_image\_path = value.get('path','')

**except** **Exception** **as** e:

front\_image\_path = ''

item["front\_image\_path"] = front\_image\_path

*# 在完成处理后一定要返回item, 这样数据才能被下一个pipeline接收并处理.*

*# 在此处添加断点再次进行调试, 看item中是否保存了图片下载的路径.*

**return** item

因为ITEM\_PIPELINES中设置的JobboleImagePipeline的优先级高于JobboleArticlePipeline, 在JobboleImagePipeline处理完成后才开始处理JobboleArticlePipeline, 在处理JobboleArticlePipeline时item中已经保存有front\_image\_path的值了.

在pipelines.JobboleImagePipeline中的return item处设置断点, 进行调试, 查看item > values中有一个front\_image\_path的键值对, 就是图片在本地的保存地址.



更进一步改进,

1. 可以把每一年的文章的图片保存到一个文件夹中

2. 使用图片原始的文件名

3. 提取文章中的图片, 把图片保存到本地, 同时把原文中的图片地址修改为本地的相对引用地址.

3. 获取到front\_image\_path图片保存的路径之后, 就可以把content中对应的图片链接全部转换为本地的保存路径, 这样就可以把别人的网页按按照原样照搬过来.

## 对article\_url进行md5处理

**把url进行md5的处理. 就可以把url变成一个唯一且长度固定的值, 作为数据库的主键.**

与spiders文件夹同级目录下新建一个名为utils的python package来存放常用的函数.

新建utils/common.py,

**import** **hashlib**

**def** get\_md5(url):

m = hashlib.md5()

m.update(url)

**return** m.hexdigest()

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

*# 进行hash的字符串必须要先进行utf-8的编码, 才能对它进行hash操作*

**print**(get\_md5("http://jobbole.com".encode("utf-8")))

运行此文件, 可以看到把这个url地址转换为md5字符串了.

由于unicode格式的字符串不能进行md5的转换, 传递过来的url不确定是unccode格式的字符串还是经过utf-8编码的bytes类型, 所以要先进行判断. python3中所有的字符串都是使用的unicode编码, 已经没有类似于python2中的unicode这个关键词了.

**import** **hashlib**

**def** get\_md5(url):

*#如果url是以unicode字符串, 就把它进行utf-8的编码, 转换为bytes类型*

**if** isinstance(url, str):

url = url.encode("utf-8")

m = hashlib.md5()

m.update(url)

**return** m.hexdigest()

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

*# 进行hash的字符串必须要先进行utf-8的编码, 才能对它进行hash操作*

**print**(get\_md5("http://jobbole.com".encode("utf-8")))

修改jobbole.py, 引入utils.common.get\_md5这个函数, 对 url 地址进行处理.

**from** **jobbole\_article.utils.common** **import** get\_md5

# 在article\_item中添加处理后的url\_object\_id字段.

*#实例化article对象*

article\_item = JobboleArticleItem(

title = title,

create\_date = create\_date,

article\_url = response.url,

url\_object\_id = get\_md5(response.url),

front\_image\_url = [front\_image\_url],

praise\_num = praise\_num,

fav\_num = fav\_num,

comment\_num = comment\_num,

content = content,

tags = tags

)

至此, 数据都已经提取并处理完成, 只需要进行数据的保存即可.

## 保存item数据到json文件中

使用爬虫yield的item都会经过item pipeline的处理, 可以在item pipeline中把数据保存到mysql或json中.

### 使用自定义方法将数据保存到json文件中.

**修改pipelines.py**

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

**from** **scrapy.pipelines.images** **import** ImagesPipeline

**import** **codecs**, **json**

**class** **JobboleArticlePipeline**(object):

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

**return** item

*# 自定义图片下载处理的中间件*

**class** **JobboleImagePipeline**(ImagesPipeline):

*# 重写函数, 改写item处理完成的函数*

**def** item\_completed(self, results, item, info):

*# result是一个list的结构. 可以获取多个图片保存的信息. 但由于使用yield, 一次只传递过一个item, 所以这里的result中只有一个元素.*

**for** key, value **in** results:

**try**:

front\_image\_path = value.get('path','')

**except** **Exception** **as** e:

front\_image\_path = ''

item["front\_image\_path"] = front\_image\_path

*# 在完成处理后一定要返回item, 这样数据才能被下一个pipeline接收并处理.*

*# 在此处添加断点再次进行调试, 看item中是否保存了图片下载的路径.*

**return** item

*# 自定义管道将Item导出为Json文件*

**class** **JsonWithEncodingPipeline**(object):

*# 初始化时调用*

**def** \_\_init\_\_(self):

*# 打开json文件*

*# 使用codecs完成文件的打开和写入能够解决编码方面的问题*

self.file = codecs.open('article.json', 'w', encoding="utf-8")

*# 重写Item处理*

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

*# 先把item转换为dict格式, 再使用json.dump把它转换为字符串.*

*# 需要关闭ensure\_ascii, 使用utf-8编码写入数据, 否则会以ascii方式写入, 中文字符就无法正确显示.*

*# 用这种方法写入的每一行数据都是一个字典, 整体上其实并不是一个真正的json文件.*

lines = json.dumps(dict(item), ensure\_ascii=False, indent=2) + "**\n**"

*# 将一行数据写入*

self.file.write(lines)

*# 重写process\_item方法时必须要使用return把它返回去, 以供其它的pipeline使用.*

**return** item

*# 爬虫结束关闭spider时调用spider\_closed方法*

**def** spider\_closed(self, spider):

*# 关闭文件句柄*

self.file.close()

修改settings.py, 启动JsonWithEncodingPipeline

ITEM\_PIPELINES = {

'jobbole\_article.pipelines.JsonWithEncodingPipeline': 300,

*# 'scrapy.pipelines.images.ImagesPipeline' : 200,*

'jobbole\_article.pipelines.JobboleImagePipeline': 200,

}

执行爬虫, 就会在项目根目录下生成一个article.json文件.

### 使用scrapy的JsonItemExporter保存json文件

scrapy自身也提供了把数据导出为不同文件格式的方法, 以导出为json格式进行说明.

C:\Users\Olina\Envs\python3\_spider\Lib\site-packages\scrapy\exporters.py

#查看exporters.py的源码, 可以看到scrapy提供了把数据导出为多种格式的方法.

from scrapy import exporters

pdir(exporters)

module attribute:

\_\_cached\_\_, \_\_file\_\_, \_\_loader\_\_, \_\_name\_\_, \_\_package\_\_, \_\_spec\_\_

property:

\_\_all\_\_, \_\_builtins\_\_, csv, io, marshal, pickle, pprint, six, sys, warnings

special attribute:

\_\_doc\_\_

class:

BaseItem: Base class for all scraped items.

BaseItemExporter:

CsvItemExporter:

JsonItemExporter:

JsonLinesItemExporter:

MarshalItemExporter:

PickleItemExporter:

PprintItemExporter:

PythonItemExporter: The idea behind this exporter is to have a mechanism to serialize items

ScrapyJSONEncoder: Extensible JSON <http://json.org> encoder for Python data structures.

XMLGenerator: Interface for receiving logical document content events.

XmlItemExporter:

exception:

ScrapyDeprecationWarning: Warning category for deprecated features, since the default

function:

is\_listlike: >>> is\_listlike("foo")

to\_bytes: Return the binary representation of `text`. If `text`

to\_native\_str: Return str representation of `text`

to\_unicode: Return the unicode representation of a bytes object `text`. If `text`

查看JsonItemExporter的源码

**class** **JsonItemExporter**(BaseItemExporter):

**def** \_\_init\_\_(self, file, \*\*kwargs):

self.\_configure(kwargs, dont\_fail=True)

self.file = file

*# there is a small difference between the behaviour or JsonItemExporter.indent*

*# and ScrapyJSONEncoder.indent. ScrapyJSONEncoder.indent=None is needed to prevent*

*# the addition of newlines everywhere*

json\_indent = self.indent **if** self.indent **is** **not** None **and** self.indent > 0 **else** None

kwargs.setdefault('indent', json\_indent)

kwargs.setdefault('ensure\_ascii', **not** self.encoding)

self.encoder = ScrapyJSONEncoder(\*\*kwargs)

self.first\_item = True

**def** \_beautify\_newline(self):

**if** self.indent **is** **not** None:

self.file.write(b'**\n**')

**def** start\_exporting(self):

self.file.write(b"[")

self.\_beautify\_newline()

**def** finish\_exporting(self):

self.\_beautify\_newline()

self.file.write(b"]")

**def** export\_item(self, item):

**if** self.first\_item:

self.first\_item = False

**else**:

self.file.write(b',')

self.\_beautify\_newline()

itemdict = dict(self.\_get\_serialized\_fields(item))

data = self.encoder.encode(itemdict)

self.file.write(to\_bytes(data, self.encoding))

pdir(exporters.JsonItemExporter("file.json"))

property:

encoder, encoding, export\_empty\_fields, fields\_to\_export, file, first\_item, indent

function:

\_beautify\_newline:

\_configure: Configure the exporter by poping options from the ``options`` dict.

\_get\_serialized\_fields: Return the fields to export as an iterable of tuples

export\_item:

finish\_exporting:

serialize\_field:

start\_exporting:

修改pipelines.py, 使用JsonItemExporter导出json文件

**from** **scrapy.exporters** **import** JsonItemExporter

*#调用scrapy提供的json export导出json文件.*

**class** **JsonExporterPipeline**(object):

*# 调用scrapy提供的json exporter导出json文件*

**def** \_\_init\_\_(self):

*# 以二进制方法打开json文件*

self.file = open('json\_item\_exporter.json', 'wb')

*# 实例化一个JsonItemExporter对象exporter, 在实例化时需要传递几个参数.*

self.exporter = JsonItemExporter(self.file, encoding="utf-8", ensure\_ascii = False, indent=2)

*#使用start\_exporting方法开始导出*

self.exporter.start\_exporting()

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

self.exporter.export\_item(item)

**return** item

**def** close\_spider(self, spider):

*#使用finish\_exporting方法结束导出*

self.exporter.finish\_exporting()

self.file.close()

修改settings.py, 启动JsonExporterPipeline

ITEM\_PIPELINES = {

'jobbole\_article.pipelines.JsonWithEncodingPipeline': 300,

'jobbole\_article.pipelines.JsonExporterPipeline': 301,

*# 'scrapy.pipelines.images.ImagesPipeline' : 200,*

'jobbole\_article.pipelines.JobboleImagePipeline': 200,

}

运行spider, 就可以看到生成的json文件了, 所有的item整体是放在一个列表中的. 查看exporters.py的源码, 在start\_exporting(self)和finish\_exporting(self)中, 分别写入了 b"[\n" 和 b"\n]" 的符号. 这样就保证了保存的数据是标准的json格式.

自定义导出json, JsonItemExporter和JsonLinesItemExporter三种导出方式中只有JsonItemExporter导出的是真正的json格式.

在使用pycharm运行程序时, 点击stop按钮结束程序的运行是强制的结束, scrapy不会正常的结束, 所以JsonExporterPipeline的最后也不会正常写入 "]", 只有在命令行模式下执行时使用ctrl+C结束时scrapy才能接收到程序结束的信号, 才能正常结束程序的运行.

## 通过pipeline保存数据到mysql

### 安装python的mysql驱动程序

# 安装Mysql客户端的python驱动程序

pip install mysqlclient

在linux中安装可能会失败, 对于ubuntu, 使用以下命令安装支持包

sudo apt install libmysqlclient-dev

对于centos, 使用下面的命令安装支持包

sudo yum install python-devel mysql-devel

再执行安装mysqlclient的操作, 就会成功

pip install mysqlclient

import MySQLdb

pdir(MySQLdb)

**property:**

BINARY, DATE, DATETIME, FIELD\_TYPE, NULL, NUMBER, PY2, ROWID, STRING, TIME, TIMESTAMP, \_\_all\_\_, \_\_author\_\_, \_\_builtins\_\_, \_\_revision\_\_, \_\_version\_\_, \_mysql, apilevel, compat, constants, paramstyle, release, threadsafety, times, version\_info

special attribute:

\_\_doc\_\_

**class:**

DBAPISet: A special type of set for which A == x is true if A is a

Date: date(year, month, day) --> date object

Time: time([hour[, minute[, second[, microsecond[, tzinfo]]]]]) --> a time object

Timestamp: datetime(year, month, day[, hour[, minute[, second[, microsecond[,tzinfo]]]]])

connection: Returns a MYSQL connection object. Exclusive use of

result: result(connection, use=0, converter={}) -- Result set from a query.

**exception:**

DataError: Exception raised for errors that are due to problems with the

DatabaseError: Exception raised for errors that are related to the

Error: Exception that is the base class of all other error exceptions

IntegrityError: Exception raised when the relational integrity of the database

InterfaceError: Exception raised for errors that are related to the database

InternalError: Exception raised when the database encounters an internal

MySQLError: Exception related to operation with MySQL.

NotSupportedError: Exception raised in case a method or database API was used

OperationalError: Exception raised for errors that are related to the database's

ProgrammingError: Exception raised for programming errors, e.g. table not found

Warning: Exception raised for important warnings like data truncations

**function:**

Binary:

Connect: Factory function for connections.Connection.

Connection: Factory function for connections.Connection.

DateFromTicks: Convert UNIX ticks into a date instance.

TimeFromTicks: Convert UNIX ticks into a time instance.

TimestampFromTicks: Convert UNIX ticks into a datetime instance.

connect: Factory function for connections.Connection.

debug: Does a DBUG\_PUSH with the given string.

escape: escape(obj, dict) -- escape any special characters in object obj

escape\_dict: escape\_sequence(d, dict) -- escape any special characters in

escape\_sequence: escape\_sequence(seq, dict) -- escape any special characters in sequence

escape\_string: escape\_string(s) -- quote any SQL-interpreted characters in string s.

get\_client\_info: get\_client\_info() -- Returns a string that represents

server\_end: Shut down embedded server. If not using an embedded server, this

server\_init: Initialize embedded server. If this client is not linked against

string\_literal: string\_literal(obj) -- converts object obj into a SQL string literal.

test\_DBAPISet\_set\_equality:

test\_DBAPISet\_set\_equality\_membership:

test\_DBAPISet\_set\_inequality:

test\_DBAPISet\_set\_inequality\_membership:

thread\_safe: Indicates whether the client is compiled as thread-safe.

"C:\Users\David\Envs\python3\_spider\Lib\site-packages\MySQLdb\connections.py"

**class** **Connection**(\_mysql.connection):

*"""MySQL Database Connection Object"""*

default\_cursor = cursors.Cursor

waiter = None

**def** \_\_init\_\_(self, \*args, \*\*kwargs):

*"""*

*Create a connection to the database. It is strongly recommended*

*that you only use keyword parameters. Consult the MySQL C API*

*documentation for more information.*

*:param str host: host to connect*

*:param str user: user to connect as*

*:param str password: password to use*

*:param str passwd: alias of password, for backward compatibility*

*:param str database: database to use*

*:param str db: alias of database, for backward compatibility*

*:param int port: TCP/IP port to connect to*

*:param str unix\_socket: location of unix\_socket to use*

*:param dict conv: conversion dictionary, see MySQLdb.converters*

*:param int connect\_timeout:*

*number of seconds to wait before the connection attempt fails.*

*:param bool compress: if set, compression is enabled*

*:param str named\_pipe: if set, a named pipe is used to connect (Windows only)*

*:param str init\_command:*

*command which is run once the connection is created*

*:param str read\_default\_file:*

*file from which default client values are read*

*:param str read\_default\_group:*

*configuration group to use from the default file*

*:param type cursorclass:*

*class object, used to create cursors (keyword only)*

*:param bool use\_unicode:*

*If True, text-like columns are returned as unicode objects*

*using the connection's character set. Otherwise, text-like*

*columns are returned as strings. columns are returned as*

*normal strings. Unicode objects will always be encoded to*

*the connection's character set regardless of this setting.*

*Default to False on Python 2 and True on Python 3.*

*:param str charset:*

*If supplied, the connection character set will be changed*

*to this character set (MySQL-4.1 and newer). This implies*

*use\_unicode=True.*

*:param str sql\_mode:*

*If supplied, the session SQL mode will be changed to this*

*setting (MySQL-4.1 and newer). For more details and legal*

*values, see the MySQL documentation.*

*:param int client\_flag:*

*flags to use or 0 (see MySQL docs or constants/CLIENTS.py)*

*:param dict ssl:*

*dictionary or mapping contains SSL connection parameters;*

*see the MySQL documentation for more details*

*(mysql\_ssl\_set()). If this is set, and the client does not*

*support SSL, NotSupportedError will be raised.*

*:param bool local\_infile:*

*enables LOAD LOCAL INFILE; zero disables*

*:param bool autocommit:*

*If False (default), autocommit is disabled.*

*If True, autocommit is enabled.*

*If None, autocommit isn't set and server default is used.*

*:param bool binary\_prefix:*

*If set, the '\_binary' prefix will be used for raw byte query*

*arguments (e.g. Binary). This is disabled by default.*

*There are a number of undocumented, non-standard methods. See the*

*documentation for the MySQL C API for some hints on what they do.*

*"""*

### 建立数据库和数据表

需要先设计mysql数据表的结构, scrapy中items中字段与mysql数据库表中字段的关系类似于django中models与mysql的关系. 可以通过django来对数据表进行管理. 也可以使用navicat手动建立数据表.

这里使用navicat来手动建立数据表

新建一个article\_spider的数据库, 新建一个article的数据表.

在栏位中添加

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 名 | 类型 | 长度 | 小数点 | 不是null | default |
| Title | varchar | 200 | 0 | TRUE |  |
| create\_date | date | 0 | 0 | FALSE |  |
| article\_url | varchar | 300 | 0 | TRUE |  |
| url\_object\_id | varchar | 50 | 0 | TRUE | Empty string |
| front\_image\_url | varchar | 300 | 0 | FALSE |  |
| front\_image\_path | varchar | 200 | 0 | FALSE |  |
| comment\_num | int | 11 | 0 | TRUE | 0 |
| fav\_num | int | 11 | 0 | TRUE | 0 |
| praise\_num | int | 11 | 0 | TRUE | 0 |
| Tags | varchar | 200 | 0 | FALSE |  |
| Content | longtext | 0 | 0 | TRUE |  |

comment\_num, fav\_num和praise\_num设置为不能为null, 这时可以设置它们的默认值为0.

再把url\_object\_id设置为主键, 在后面就可以根据主键来进行更新.

**show** databases;

**create** **database** jobbole\_article charset=utf8;

**show** databases;

use jobbole\_article;

**create** **table** article(

title varchar(200) **not** **null**,

create\_date date,

article\_url varchar(300) **not** **null**,

url\_object\_id varchar(50) **not** **null** **default** '' **primary** **key**,

front\_image\_url varchar(300),

front\_image\_path varchar(200),

comment\_num int(11) **not** **null** **default** 0,

fav\_num int(11) **not** **null** **default** 0,

praise\_num int(11) **not** **null** **default** 0,

tags varchar(200),

content longtext **not** **null**

);

**desc** article;

**select** \* **from** article;

**select** **count**(\*) **from** article;

**truncate** **table** article;

### 修改jobbole.py, 把发表时间修改为日期格式

因为在提取数据时是把日期当成字符串处理的, 而在mysql中是把发表日期定义为date类型的, 所以要修改jobbole.py文件, 把发表日期转化为日期格式.

import datetime

# 时间转换处理

try:

create\_date = datetime.datetime.strptime(value, "%Y/%m/%d").date()

except Exception as e:

#如果出现异常, 就将它的时间设置为当前的时间

create\_date = datetime.datetime.now().date()

article\_item['create\_date'] = create\_date

**import** **datetime**

**class** **JobboleSpider**(scrapy.Spider):

**def** parse\_detail(self, response):

*# 发表时间*

create\_date\_str = response.xpath('//p[@class="entry-meta-hide-on-mobile"]/text()').extract\_first(default='1970/01/01').strip().replace("·", "").strip()

create\_date\_str = response.css("p.entry-meta-hide-on-mobile::text").extract\_first(default='1970/01/01').strip().replace("·", "").strip()

*# 把 create\_date 转换为日期格式*

create\_date = datetime.datetime.strptime(create\_date\_str, "%Y/%m/**%d**").date()

在此处添加断点, 进行调试, 可以正常转换为日期.

如果同时打开json相关的两个pipeline, 就会出现

TypeError: Object of type 'date' is not JSON serializable

解决方法参考

python datetime.datetime is not JSON serializable 报错问题解决

<https://blog.csdn.net/suibianshen2012/article/details/64444030>

1、问题描述

使用python自带的json，将数据转换为json数据时，datetime格式的数据报错：datetimeTypeError: datetime.datetime(2017, 3, 21, 2, 11, 21) is not JSON serializable。

使用python的json模块序列化时间或者其他不支持的类型时会抛异常，例如下面的代码：

# -\*- coding: cp936 -\*-

from datetime import datetime

import json

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

now = datetime.now()

json.dumps({'now':now})

运行会出现下面的错误信息：

Traceback (most recent call last):

File "C:\Users\xx\Desktop\t.py", line 8, in <module>

json.dumps({'now':now})

File "C:\Python27\lib\json\\_\_init\_\_.py", line 231, in dumps

return \_default\_encoder.encode(obj)

File "C:\Python27\lib\json\encoder.py", line 201, in encode

chunks = self.iterencode(o, \_one\_shot=True)

File "C:\Python27\lib\json\encoder.py", line 264, in iterencode

return \_iterencode(o, 0)

File "C:\Python27\lib\json\encoder.py", line 178, in default

raise TypeError(repr(o) + " is not JSON serializable")

TypeError: datetime.datetime(2012, 12, 26, 11, 51, 33, 409000) is not JSON serializable

意思是说datetime类不支持Json序列化

2、解决方法

就是重写构造json类，遇到日期特殊处理，其余的用内置的就行。

import json

import datetime

class DateEncoder(json.JSONEncoder):

def default(self, obj):

if isinstance(obj, datetime.datetime):

return obj.strftime('%Y-%m-%d %H:%M:%S')

elif isinstance(obj, date):

return obj.strftime("%Y-%m-%d")

else:

return json.JSONEncoder.default(self, obj)

使用时，调用上面定义的函数即可，如下：

print json.dumps(self\_data, cls=DateEncoder)

我们需要对json做下扩展，让它可以支持datetime类型。

class ComplexEncoder(json.JSONEncoder):

def default(self, obj):

if isinstance(obj, datetime):

return obj.strftime('%Y-%m-%d %H:%M:%S')

elif isinstance(obj, date):

return obj.strftime('%Y-%m-%d')

else:

return json.JSONEncoder.default(self, obj)

在调用json.dumps时需要指定cls参数为ComplexEncoder

例如：

json.dumps({'now':now}, cls=ComplexEncoder)

How to serialize a datetime object as JSON using Python?

<https://code-maven.com/serialize-datetime-object-as-json-in-python>

It is easy to serialize a Python data structure as JSON, we just need to call the json.dumps method, but if our data stucture contains a datetime object we'll get an exception:

TypeError: datetime.datetime(...) is not JSON serializable

How can we fix this?

In the first example we can see the problem:

examples/python/datetime\_json\_fails.py

import json

import datetime

d = {

'name' : 'Foo'

}

print(json.dumps(d)) # {"name": "Foo"}

d['date'] = datetime.datetime.now()

print(json.dumps(d)) # TypeError: datetime.datetime(2016, 4, 8, 11, 22, 3, 84913) is not JSON serializable

The first call to json.dumps works properly, but once we add a key with a value that is a datetime object, the call throws an exception.

The solution

The solution is quite simple. The json.dumps method can accept an optional parameter called default which is expected to be a function. Every time JSON tries to convert a value it does not know how to convert it will call the function we passed to it. The function will receive the object in question, and it is expected to return the JSON representation of the object.

In the function we just call the \_\_str\_\_ method of the datetime object that will return a string representation of the value. This is what we return.

While the condition we have in the function is not required, if we expect to have other types of objects in our data structure that need special treatment, we can make sure our function handles them too. As dealing with each object will probably be different we check if the current object is one we know to handle and do that inside the if statement.

examples/python/datetime\_json.py

import json

import datetime

d = {

'name' : 'Foo'

}

print(json.dumps(d)) # {"name": "Foo"}

d['date'] = datetime.datetime.now()

def myconverter(o):

if isinstance(o, datetime.datetime):

return o.\_\_str\_\_()

print(json.dumps(d, default = myconverter)) # {"date": "2016-04-08 11:43:36.309721", "name": "Foo"}

Other representation of datetime

The string representation that \_\_str\_\_ might match our needs, but if not we have other options. We can use the \_\_repr\_\_ method to return the following:

{"date": "datetime.datetime(2016, 4, 8, 11, 43, 54, 920632)", "name": "Foo"}

We can even hand-craft something like this:

return "{}-{}-{}".format(o.year, o.month, o.day)

That will return the following:

{"date": "2016-4-8", "name": "Foo"}

更好的解决方法是不对日期型字符串进行处理, 在pipelines.py中写入到mysql中时单独进行处理, 这样就绕过了上面的错误. 或者直接不做任何处理, 在写入到mysql数据库时mysql会自动把字符串类型的日期转换为date格式的.

### 将数据保存到mysql中

#### 同步化将Item保存入数据库

修改pipelines.py

**import** **MySQLdb**

*# 同步机制写入数据库*

**class** **MysqlPipeline**(object):

**def** \_\_init\_\_(self):

*#创建一个连接MySQLdb.connect('host', 'user', 'password', 'dbname', charset, use\_unicode), 可以打开connect函数查看其源码.*

*# self.conn = MySQLdb.connect('127.0.0.1', 'root', 'mysql', ' article\_spider', charset="utf8", use\_unicode=True)*

self.conn = MySQLdb.connect(

host = 'sh-cdb-pzoa3tqh.sql.tencentcdb.com',

user = 'root',

password = 'Xzq@8481',

db = 'jobbole\_article',

port = 63104,

charset = 'utf8',

)

*#执行数据库的具体操作是由cursor来完成的.*

self.cursor = self.conn.cursor()

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

*#这里insert插入的操作要与之前数据库中设置的字段的名称和顺序相同.*

insert\_sql = '''

insert ignore into article(title, create\_date, article\_url, url\_object\_id, front\_image\_url, front\_image\_path, comment\_num, praise\_num, fav\_num, tags, content)

VALUES(**%s**, **%s**, **%s**, **%s**, **%s**, **%s**, **%s**, **%s**, **%s**, **%s**, **%s**)

'''

*# 因为mysql中定义的create\_date为日期格式的, 所以要先把item中字符串格式的create\_date转换为日期格式*

*# item['create\_date'] = datetime.datetime.strptime(item['create\_date'], "%Y/%m/%d").date()*

self.cursor.execute(insert\_sql, (item["title"], item["create\_date"], item["article\_url"], item["url\_object\_id"], item["front\_image\_url"][0], item['front\_image\_path'], item["comment\_num"], item["praise\_num"], item["fav\_num"], item["tags"], item["content"]))

*# 注意使用的是conn.commit, 不是cursor.commit*

self.conn.commit()

**def** spider\_closed(self, spider):

self.conn.close()

在settings中配置启用MysqlPipeline

ITEM\_PIPELINES = {

'jobbole\_article.pipelines.JsonWithEncodingPipeline': 300,

'jobbole\_article.pipelines.JsonExporterPipeline': 301,

*# 'scrapy.pipelines.images.ImagesPipeline' : 200,*

'jobbole\_article.pipelines.JobboleImagePipeline': 200,

'jobbole\_article.pipelines.MysqlPipeline': 302,

}

在self.cursor.execute打上断点, 在jobbole.py中article\_item['create\_date']上打上断点, 进行debug. F6继续执行.

回到数据库中进行查看, 已经插入了一条数据. 按F8继续执行, 就可以看到数据会源源不断的写进数据库.

#### 异步化将Item保存入数据库

因为Scrapy的解析速度非常快, 加上文章的内容较大, 上面mysql数据库的写入是同步的操作, 可能会出现数据库的写入速度赶不上解析速度, 就会产生阻塞, 因此采用异步化的方式来进行数据的插入, twisted框架提供了一种连接池, 可以把mysql数据库的插入变成异步的操作.

在上面的操作中, self.cursor.execute和self.conn.commit()是同步的操作, 必须在完成上面的操作之后才能进行下一步的操作.

在settings中添加mysql的设置

*################################################################*

*# mysql配置*

MYSQL\_HOST = 'sh-cdb-pzoa3tqh.sql.tencentcdb.com'

*#这里设置的是数据库的名字, 不是数据表的名字*

MYSQL\_DBNAME = 'jobbole\_article'

MYSQL\_USERNAME = 'root'

MYSQL\_PASSWORD = 'Xzq@8481'

MYSQL\_PORT = "63104"

*################################################################*

修改pipelines.py文件

**import** **MySQLdb.cursors**

**from** **twisted.enterprise** **import** adbapi

*# 异步操作写入数据库*

**class** **MysqlTwistedPipline**(object):

*# from\_settings和\_\_init\_\_这两个方法就能实现在启动spider时, 就把dbpool传递进来*

**def** \_\_init\_\_(self, dbpool):

self.dbpool = dbpool

*# 在初始化时scrapy会调用from\_settings方法, 将setting文件中的配置读入, 成为一个settings对象, 这种写法是固定的, 其中的参数不能修改.*

@classmethod

**def** from\_settings(cls, settings):

dbparas = dict(

host = settings["MYSQL\_HOST"], *#可以在settings中设置此pipeline, 在此处放置断点, 进行debug, 查看能否导入. 在attribute中可以看到settings中定义的所有的值. F6执行, 就能看到取到了settings中设置的host的值了.*

port = settings["MYSQL\_PORT"],

db = settings["MYSQL\_DBNAME"],

user = settings["MYSQL\_USERNAME"],

passwd = settings["MYSQL\_PASSWORD"],

charset = "utf8",

*# pymysql模块中也有类似的模块pymysql.cursors.DictCursor*

cursorclass = MySQLdb.cursors.DictCursor,

use\_unicode = True

)

*# 创建twisted的mysql连接池, 使用twisted的连接池, 就能把mysql的操作转换为异步操作.*

*# twisted只是提供了一个异步的容器, 并没有提供连接mysql的方法, 所以还需要MySQLdb的连接方法.*

*# adbapi可以将mysql的操作变成异步化的操作. 查看ConectionPool, def \_\_init\_\_(self, dbapiName, \*connargs, \*\*connkw).*

*# 需要指定使用的连接模块的模块名, 第一个参数是dbapiName, 即mysql的模块名MySQLdb. 第二个参数是连接mysql的参数, 写为可变化的参数形式. 查看MySQLdb的源码, 在from MySQLdb.connections import Connection中查看Connection的源码, 在class Connection中就能看到MySQLdb模块在连接mysql数据库时需要传递的参数. param这个dict中参数的名称要与其中的参数名称保持一致. 即与connections.py中 class Connection中的def \_\_init\_\_中定义的参数保持一致.*

dbpool = adbapi.ConnectionPool("MySQLdb", \*\*dbparas)

*# 如果对上面的写法不太理解, 可以写成下面的形式*

*# dbpool = adbapi.ConnectionPool("MySQLdb", host = settings["MYSQL\_HOST"], db = settings["MYSQL\_DBNAME"], user = settings["MYSQL\_USERNAME"], passwd = settings["MYSQL\_PASSWORD"], charset = "utf8", cursorclass = MySQLdb.cursors.DictCursor, use\_unicode = True)*

*# 因为使用@classmethod把这个方法转换为类方法了, 所以cls就是指的MysqlTwistedPipline这个类, 所以cls(dbpool) 就相当于使用dbpool这个参数实例化MysqlTwistedPipline类的一个对象, 再把这个对象返回. 然后在init方法中接收这里创建的异步连接对象.*

**return** cls(dbpool)

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

*# 使用twisted将mysql数据库的插入变成异步插入, 第一个参数是自定义的函数, runInteraction可以把一个函数的操作变成异步的操作. 第二个参数是要插入的数据, 这里是item.*

query = self.dbpool.runInteraction(self.do\_insert, item)

*#处理异常, 这里也可以不传递item和spider*

query.addErrback(self.handle\_error, item, spider)

*# 自定义错误处理, 处理异步插入的异常, 这里也可以不传递item和spider, 只传递failure即可.*

**def** handle\_error(self, failure, item, spider):

**print**(failure)

**print**(item)

*# 执行具体的插入, 此时的cursor就是self.dbpool.runInteraction中传递过来的cursor, 使用这个cursor, 就可以把mysql的操作变成异步的操作. 并且此时也不用再手动执行commit的操作了.*

**def** do\_insert(self, cursor, item):

insert\_sql = '''

INSERT IGNORE INTO

article(title, create\_date, article\_url, url\_object\_id, front\_image\_url, front\_image\_path, comment\_num, praise\_num, fav\_num, tags, content)

VALUES

(**%s**, **%s**, **%s**, **%s**, **%s**, **%s**, **%s**, **%s**, **%s**, **%s**, **%s**)

'''

cursor.execute(insert\_sql, (item["title"], item["create\_date"], item["article\_url"], item["url\_object\_id"], item["front\_image\_url"][0], item["front\_image\_path"], item["comment\_num"], item["praise\_num"], item["fav\_num"], item["tags"], item["content"]))

在settings中启用MysqlTwistedPipline

ITEM\_PIPELINES = {

'jobbole\_article.pipelines.JsonWithEncodingPipeline': 300,

'jobbole\_article.pipelines.JsonExporterPipeline': 301,

*# 'scrapy.pipelines.images.ImagesPipeline' : 200,*

'jobbole\_article.pipelines.JobboleImagePipeline': 200,

*# 'jobbole\_article.pipelines.MysqlPipeline': 302,*

'jobbole\_article.pipelines.MysqlTwistedPipline': 302,

}

在执行操作前清空原有的所有数据库. twisted只支持关系型数据库的异步操作.

### 把django的ORM集成到scrapy中

https://github.com/scrapy-plugins/scrapy-djangoitem

实际上是scrapy的一个插件, 通过django model的形式将scrapy的item写入到数据库中.

在定义item的时候就定义为django item, 在保存item的时候就可以像django中的样, 直接使用save()来保存了. 会比上面使用sql语句写入数据库简单的多.

## scrapy itemloader机制

<https://doc.scrapy.org/en/latest/topics/loaders.html>

随着爬取的网站的增多和提取的信息的增多, 要写大量的css或xpath selector语法, 不同的爬虫之间的代码不能重用, 后期维护的成本也比较高. scrapy提供了一种itemloader的机制, 可以使后期的维护变得很简单.

itemloader提供了一个容器. 这个容器中可以配置item中的某个字段需要用到哪种css或xpath规则来解析, 配置完成后调用itemloader方法, 就可以直接生成itemloader.

新建jobbole\_it爬虫

scrapy genspider jobbole\_it blog.jobbole.com

### ItemLoader的使用方法

查看ItemLoader的源码

"C:\Users\David\Envs\python3\_spider\Lib\site-packages\scrapy\loader\\_\_init\_\_.py"

**class** **ItemLoader**(object):

default\_item\_class = Item

default\_input\_processor = Identity()

default\_output\_processor = Identity()

default\_selector\_class = Selector

**def** \_\_init\_\_(self, item=None, selector=None, response=None, parent=None, \*\*context):

**if** selector **is** None **and** response **is** **not** None:

selector = self.default\_selector\_class(response)

self.selector = selector

context.update(selector=selector, response=response)

**if** item **is** None:

item = self.default\_item\_class()

self.context = context

self.parent = parent

self.\_local\_item = context['item'] = item

self.\_local\_values = defaultdict(list)

from scrapy.loader import ItemLoader

pdir(ItemLoader())

**property:**

\_local\_item, \_local\_values, context, default\_input\_processor, default\_output\_processor, parent, selector

**descriptor:**

\_values: @property with getter

item: @property with getter

**class:**

default\_item\_class: Base class for all scraped items.

default\_selector\_class: :class:`Selector` allows you to select parts of an XML or HTML text using CSS

**function:**

\_add\_value:

\_check\_selector\_method:

\_get\_cssvalues:

\_get\_item\_field\_attr:

\_get\_values:

\_get\_xpathvalues:

\_process\_input\_value:

\_replace\_value:

add\_css:

add\_value:

add\_xpath:

get\_collected\_values:

get\_css:

get\_input\_processor:

get\_output\_processor:

get\_output\_value:

get\_value:

get\_xpath:

load\_item:

nested\_css:

nested\_xpath:

replace\_css:

replace\_value:

replace\_xpath:

**from** **scrapy.loader** **import** ItemLoader

*#通过itemloader加载item, 先要实例化ItemLoader, 查看ItemLoader源码, 它有两个重要的参数, 一个是item, 一个是response.*

*# item即之前在parse\_detail第一行中定义的JobBoleArticleItem(), 它实际上是一个实例化的对象.*

item\_loader = ItemLoader(item=JobBoleArticleItem(), response=response)

*# ItemLoader一共有3个重要的方法.*

*# item\_loader.add\_css()*

*# item\_loader.add\_xpath()*

*# item\_loader.add\_value()*

**def** add\_value(self, field\_name, value, \*processors, \*\*kw):

value = self.get\_value(value, \*processors, \*\*kw)

**if** value **is** None:

**return**

**if** **not** field\_name:

**for** k, v **in** six.iteritems(value):

self.\_add\_value(k, v)

**else**:

self.\_add\_value(field\_name, value)

**def** add\_xpath(self, field\_name, xpath, \*processors, \*\*kw):

values = self.\_get\_xpathvalues(xpath, \*\*kw)

self.add\_value(field\_name, values, \*processors, \*\*kw)

**def** add\_css(self, field\_name, css, \*processors, \*\*kw):

values = self.\_get\_cssvalues(css, \*\*kw)

self.add\_value(field\_name, values, \*processors, \*\*kw)

*# add\_css()中的第一个参数就是在items.py中定义的JoBoleArticleItem中的要提取的字段, 如title. 第二个参数就是它对应的css样式. 在item\_loader中实例化的JoBoleArticleItem()就会调用后面的css选择器, 从response中选择出内容放在title字段中. 如*

item\_loader.add\_css("title",".entry-header h1::text")

*# 实例化自定义item\_loader*

item\_loader = JobBoleArticleItemLoader(item=JobBoleArticleItem(),response=response)

### 修改jobbole\_it.py, 使用itemloader

from scrapy.loader **import** **ItemLoader**

**class** **JobboleSpider**(scrapy.Spider):

name = "jobbole"

allowed\_domains = ["blog.jobbole.com"]

start\_urls = ['http://blog.jobbole.com/all-posts/']

**def** parse\_detail(self, response):

*# 通过item loader加载item*

*# 文章封面图*

front\_image\_url = response.meta.get("front\_image\_url", "")

item\_loader = ItemLoader(item=JobBoleArticleItem(), response=response)

*# 通过item\_loader加载item*

item\_loader.add\_css('title', 'div.entry-header h1::text')

item\_loader.add\_css('create\_time', 'p.entry-meta-hide-on-mobile::text')

*# url的值并不是调用css选择器提取出来的, 而是通过response.url直接传递过来的, 要使用add\_value直接添加值的方法. 而add\_css则是调用css选择器把值取出来后再传递给前面的字段中.*

item\_loader.add\_value('url', response.url)

item\_loader.add\_value('url\_object\_id', get\_md5(response.url))

item\_loader.add\_value('front\_image\_url', [front\_image\_url])

item\_loader.add\_css('comment\_num', 'a[href="#article-comment"] span::text')

item\_loader.add\_css('content', 'div.entry')

item\_loader.add\_css('tags', 'p.entry-meta-hide-on-mobile a::text')

item\_loader.add\_css('praise\_num', '.vote-post-up h10::text')

item\_loader.add\_css('fav\_num', '.bookmark-btn::text')

*#写完上面的规则之后要调用item\_loader的load\_item()方法对以上的规则进行解析. 解析之后就生成了item对象.*

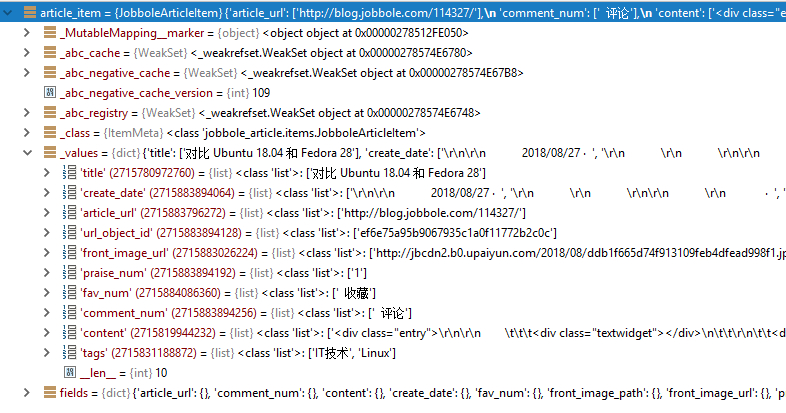
article\_item = item\_loader.load\_item()

**yield** article\_item

运用item\_loader的方法之后, 代码就变得非常简洁, 并且更具有可移植性和可配置性. 可以把css或xpath的匹配规则写入到数据库或文件中, 在解析时动态来获取这些匹配规则就可以了.

但还存在两个问题.

第一个问题. 在yield article\_item处添加断点, 修改 Run/Debug Configurations, 修改 Parameters为: crawl jobbole\_it, 运行debug. 看是否生成了item对象. 生成了一个JoboleArticleItem()的对象.



其中的values值就是在item\_loader中添加的所有的字段和add\_css或add\_value得到的值. 注意得到的所有的值都是列表的形式, 即使是使用add\_value方法传递了一个值, 也得到一个列表. 这也是符合逻辑的, 因为使用css选择器可能会选取到多个的值, 对于多值的css规则, 提取出来的值就会保存到列表中, 这里为了保持一致性, 就把提取出来的所有的元素都放在列表中了. 可以用提取列表中的第1个元素的方法解决.

第二个问题. 如果没有人评论, 只会提取到评论两个字, 不会取到评论数为0, 可以给comment\_num的提取规则加一个处理函数, 像前面一样使用正则表达式进行判断, 如果没有提取到评论数, 就把它设置为0.

为了解决以上两个问题, 需要修改items.py文件.

### 修改items.py, 对itemloader提取到的字段进行处理

#### input\_processor的使用

定义的scrpy.Field()实际上有两个参数, 一个是input\_processor, 当这个字段的值传递过来时, 可以通过input\_processor进行一些预处理.

from scrapy.loader.processors import MapCompose

# MapCompose这个方法可以传递任意多的函数, 能传递过来的值进行预处理. 如对于title字段, 想要加一个后缀-jobbole, 在这个类之前先定义一个函数add\_jobbole()

def add\_jobbole(value):

return value+'-jobbole'

title = scrapy.Field(

input\_processor = MapCompose(add\_jobbole)

)

在MapCompose调用add\_jobbole这个函数的时候, 会把title中获取的值传递到add\_jobbole这个函数中, add\_jobbole这个函数以value的形参接收这个title的值, 进行处理后返回到title中. 这样就完成了传递过来的值的预处理.

甚至可以在MapCompose中传递一个匿名函数

title = scrapy.Field(

input\_processor = MapCompose(lambda x:x+'-jobbole')

)

屏蔽上面的add\_jobbole函数, 在jobbole.py中yield article\_item一行添加断点进行debug, 查看jobbole.py中article\_item中title的值是否有所变化.

也可以调用2个函数

def add\_jobbole(value):

return value+'-bobby'

title = scrapy.Field(

input\_processor = MapCompose(lambda x:x+'-jobbole', add\_jobbole)

)

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

*# Define here the models for your scraped items*

*#*

*# See documentation in:*

*# https://doc.scrapy.org/en/latest/topics/items.html*

**import** **scrapy**

**from** **scrapy.loader.processors** **import** MapCompose

*# MapCompose这个方法可以传递任意多的函数, 能传递过来的值进行预处理. 如对于title字段, 想要加一个后缀-jobbole, 在这个类之前先定义一个函数add\_jobbole()*

**def** add\_jobbole(value):

**return** value+'-jobbole'

*# 为了使原来jobbole.py中的item依旧可用, 这里新建一个item, 并在jobbole\_it.py中使用*

**class** **JobboleArticleProcessItem**(scrapy.Item):

title = scrapy.Field(

*# 在MapCompose调用add\_jobbole这个函数的时候, 会把title中获取的值传递到add\_jobbole这个函数中, add\_jobbole这个函数以value的形参接收这个title的值, 进行处理后返回到title中. 这样就完成了传递过来的值的预处理.*

input\_processor = MapCompose(add\_jobbole)

)

create\_date = scrapy.Field()

*# 文章详情的url*

article\_url = scrapy.Field()

*#url是变长的, 通过md5把它变成固定长度的.*

url\_object\_id = scrapy.Field()

*#封面图的url地址*

front\_image\_url = scrapy.Field()

*#封面图在本地保存的路径.*

front\_image\_path = scrapy.Field()

praise\_num = scrapy.Field()

fav\_num = scrapy.Field()

comment\_num = scrapy.Field()

content = scrapy.Field()

tags = scrapy.Field()

修改jobbole\_it.py, 在其中使用JobboleArticleProcessItem

**from** **jobbole\_article.items** **import** JobboleArticleProcessItem

**def** parse\_detail(self, response):

*# 通过item loader加载item*

*# 文章封面图*

front\_image\_url = response.meta.get("front\_image\_url", "")

item\_loader = ItemLoader(item=JobboleArticleProcessItem(), response=response)

在jobbole.py 中 yield article\_item 一行添加断点进行debug, 查看 jobbole.py 中 article\_item 中 title 的值是否有所变化.



# 在MapCompose中传递匿名函数

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

*# Define here the models for your scraped items*

*#*

*# See documentation in:*

*# https://doc.scrapy.org/en/latest/topics/items.html*

**import** **scrapy**

**from** **scrapy.loader.processors** **import** MapCompose

*# MapCompose这个方法可以传递任意多的函数, 能传递过来的值进行预处理. 如对于title字段, 想要加一个后缀-jobbole, 在这个类之前先定义一个函数add\_jobbole()*

**def** add\_jobbole(value):

**return** value+'-jobbole'

*# 为了使原来jobbole.py中的item依旧可用, 这里新建一个item, 并在jobbole\_it.py中使用*

**class** **JobboleArticleProcessItem**(scrapy.Item):

*# title = scrapy.Field(*

*# # 在MapCompose调用add\_jobbole这个函数的时候, 会把title中获取的值传递到add\_jobbole这个函数中, add\_jobbole这个函数以value的形参接收这个title的值, 进行处理后返回到title中. 这样就完成了传递过来的值的预处理. 在jobbole.py 中 yield article\_item 一行添加断点进行debug, 查看 jobbole.py 中 article\_item 中 title 的值是否有所变化.*

*# input\_processor = MapCompose(add\_jobbole)*

*# )*

title = scrapy.Field(

*# 甚至可以在MapCompose中传递一个匿名函数. 屏蔽上面的add\_jobbole函数,*

input\_processor = MapCompose(**lambda** x:x+'-jobbole')

)

create\_date = scrapy.Field()

*# 文章详情的url*

article\_url = scrapy.Field()

*#url是变长的, 通过md5把它变成固定长度的.*

url\_object\_id = scrapy.Field()

*#封面图的url地址*

front\_image\_url = scrapy.Field()

*#封面图在本地保存的路径.*

front\_image\_path = scrapy.Field()

praise\_num = scrapy.Field()

fav\_num = scrapy.Field()

comment\_num = scrapy.Field()

content = scrapy.Field()

tags = scrapy.Field()

在jobbole.py 中 yield article\_item 一行添加断点, 再次进行debug, 查看 jobbole.py 中 article\_item 中 title 的值是否有所变化.

在MapCompose函数中传递多个函数

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

*# Define here the models for your scraped items*

*#*

*# See documentation in:*

*# https://doc.scrapy.org/en/latest/topics/items.html*

**import** **scrapy**

**from** **scrapy.loader.processors** **import** MapCompose

*# MapCompose这个方法可以传递任意多的函数, 能传递过来的值进行预处理. 如对于title字段, 想要加一个后缀-jobbole, 在这个类之前先定义一个函数add\_jobbole()*

**def** add\_jobbole(value):

**return** value+'-jobbole'

*# 为了使原来jobbole.py中的item依旧可用, 这里新建一个item, 并在jobbole\_it.py中使用*

**class** **JobboleArticleProcessItem**(scrapy.Item):

*# title = scrapy.Field(*

*# # 在MapCompose调用add\_jobbole这个函数的时候, 会把title中获取的值传递到add\_jobbole这个函数中, add\_jobbole这个函数以value的形参接收这个title的值, 进行处理后返回到title中. 这样就完成了传递过来的值的预处理. 在jobbole.py 中 yield article\_item 一行添加断点进行debug, 查看 jobbole.py 中 article\_item 中 title 的值是否有所变化.*

*# input\_processor = MapCompose(add\_jobbole)*

*# )*

*# title = scrapy.Field(*

*# # 甚至可以在MapCompose中传递一个匿名函数. 屏蔽上面的add\_jobbole函数, 在jobbole.py 中 yield article\_item 一行添加断点进行debug, 查看 jobbole.py 中 article\_item 中 title 的值是否有所变化.*

*# input\_processor = MapCompose(lambda x:x+'-jobbole')*

*# )*

title = scrapy.Field(

*# 也可以调用2个函数*

input\_processor = MapCompose(**lambda** x:x+'-jobbole', add\_jobbole)

)

create\_date = scrapy.Field()

*# 文章详情的url*

article\_url = scrapy.Field()

*#url是变长的, 通过md5把它变成固定长度的.*

url\_object\_id = scrapy.Field()

*#封面图的url地址*

front\_image\_url = scrapy.Field()

*#封面图在本地保存的路径.*

front\_image\_path = scrapy.Field()

praise\_num = scrapy.Field()

fav\_num = scrapy.Field()

comment\_num = scrapy.Field()

content = scrapy.Field()

tags = scrapy.Field()

再次debug, 查看title的值. 这样就说明了input\_process可以按顺序调用MapCompose中的多个函数.



#### 处理create\_date字段

可以用类似的方法处理时间, 把传递到create\_date中的的string时间转换为python的date对象. 只需要把之前对时间处理的语句封装为一个函数即可. 注意定义的date\_convert函数和add\_jobbole函数之间要间隔2行, 以满足PEP8的规范.

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

*# Define here the models for your scraped items*

*#*

*# See documentation in:*

*# https://doc.scrapy.org/en/latest/topics/items.html*

**import** **scrapy**

**from** **scrapy.loader.processors** **import** MapCompose

**import** **datetime**

**from** **scrapy.loader.processors** **import** TakeFirst

**class** **JobboleArticleItem**(scrapy.Item):

title = scrapy.Field()

create\_date = scrapy.Field()

*# 文章详情的url*

article\_url = scrapy.Field()

*#url是变长的, 通过md5把它变成固定长度的.*

url\_object\_id = scrapy.Field()

*#封面图的url地址*

front\_image\_url = scrapy.Field()

*#封面图在本地保存的路径.*

front\_image\_path = scrapy.Field()

praise\_num = scrapy.Field()

fav\_num = scrapy.Field()

comment\_num = scrapy.Field()

content = scrapy.Field()

tags = scrapy.Field()

*# MapCompose这个方法可以传递任意多的函数, 能传递过来的值进行预处理. 如对于title字段, 想要加一个后缀-jobbole, 在这个类之前先定义一个函数add\_jobbole()*

**def** add\_jobbole(value):

**return** value+'-jobbole'

*# 处理文章的发表时间*

**def** date\_convert(value):

*# return value.replace("*·*", "").strip()*

**try**:

create\_date = datetime.datetime.strptime(value.replace("·", "").strip(), "%Y/%m/**%d**").date()

**except** **Exception** **as** e:

*#如果出现异常, 就将它的时间设置为当前的时间*

create\_date = datetime.datetime.now().date()

**return** create\_date

*# 为了使原来jobbole.py中的item依旧可用, 这里新建一个item, 并在jobbole\_it.py中使用*

**class** **JobboleArticleProcessItem**(scrapy.Item):

title = scrapy.Field(

*# 在MapCompose调用add\_jobbole这个函数的时候, 会把title中获取的值传递到add\_jobbole这个函数中, add\_jobbole这个函数以value的形参接收这个title的值, 进行处理后返回到title中. 这样就完成了传递过来的值的预处理. 在jobbole.py 中 yield article\_item 一行添加断点进行debug, 查看 jobbole.py 中 article\_item 中 title 的值是否有所变化.*

input\_processor = MapCompose(add\_jobbole)

)

create\_date = scrapy.Field(

input\_processor = MapCompose(date\_convert)

)

*# 文章详情的url*

article\_url = scrapy.Field()

*#url是变长的, 通过md5把它变成固定长度的.*

url\_object\_id = scrapy.Field()

*#封面图的url地址*

front\_image\_url = scrapy.Field()

*#封面图在本地保存的路径.*

front\_image\_path = scrapy.Field()

praise\_num = scrapy.Field()

fav\_num = scrapy.Field()

comment\_num = scrapy.Field()

content = scrapy.Field()

tags = scrapy.Field()

再次debug, 查看create\_date的值. 它就变成了一个放在数组中的date对象, 而不是之前的string的对象了.



为什么在create\_date字段中会出现3个值呢, 这是因为原网页中是

<p class="entry-meta-hide-on-mobile">

2018/08/27 · <a href="http://blog.jobbole.com/category/it-tech/" rel="category tag">IT技术</a>

· <a href="http://blog.jobbole.com/tag/linux/">Linux</a>

</p>

使用entry-meta-hide-on-mobile提取到的是一个3个元素的列表, 由于没有使用extract\_first()提取其中的第1个元素, 所以会在结果中得到3个元素.

['\r\n\r\n 2018/08/27 · ', '\r\n \r\n \r\n\r\n \r\n · ', '\r\n \r\n']

但由于mysql会在保存时自动进行类型的转换, 这里还使用与之前相同的方法, 不进行日期的转换.

同样, 可以用类似的方法处理comment\_num, fav\_num, praise\_num 的问题.

#### TakeFirst()的使用

解决第二个问题, 即所有字段获取的值都是数组的形式. 需要用到scrapy中的另外一个函数TakeFirst(), 在output\_processor中调用TakeFirst()函数, 就能把列表中的第1个元素提取出来了.

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

*# Define here the models for your scraped items*

*#*

*# See documentation in:*

*# https://doc.scrapy.org/en/latest/topics/items.html*

**import** **scrapy**

**from** **scrapy.loader.processors** **import** MapCompose

**import** **datetime**

**from** **scrapy.loader.processors** **import** TakeFirst

*# MapCompose这个方法可以传递任意多的函数, 能传递过来的值进行预处理. 如对于title字段, 想要加一个后缀-jobbole, 在这个类之前先定义一个函数add\_jobbole()*

**def** add\_jobbole(value):

**return** value+'-jobbole'

*# 处理文章的发表时间*

**def** date\_convert(value):

*# return value.replace("*·*", "").strip()*

**try**:

create\_date = datetime.datetime.strptime(value.replace("·", "").strip(), "%Y/%m/**%d**").date()

**except** **Exception** **as** e:

*#如果出现异常, 就将它的时间设置为当前的时间*

create\_date = datetime.datetime.now().date()

**return** create\_date

*# 为了使原来jobbole.py中的item依旧可用, 这里新建一个item, 并在jobbole\_it.py中使用*

**class** **JobboleArticleProcessItem**(scrapy.Item):

title = scrapy.Field(

*# 在MapCompose调用add\_jobbole这个函数的时候, 会把title中获取的值传递到add\_jobbole这个函数中, add\_jobbole这个函数以value的形参接收这个title的值, 进行处理后返回到title中. 这样就完成了传递过来的值的预处理. 在jobbole.py 中 yield article\_item 一行添加断点进行debug, 查看 jobbole.py 中 article\_item 中 title 的值是否有所变化.*

input\_processor = MapCompose(add\_jobbole)

)

create\_date = scrapy.Field(

input\_processor = MapCompose(date\_convert),

output\_processor = TakeFirst()

)

*# 文章详情的url*

article\_url = scrapy.Field()

*#url是变长的, 通过md5把它变成固定长度的.*

url\_object\_id = scrapy.Field()

*#封面图的url地址*

front\_image\_url = scrapy.Field()

*#封面图在本地保存的路径.*

front\_image\_path = scrapy.Field()

praise\_num = scrapy.Field()

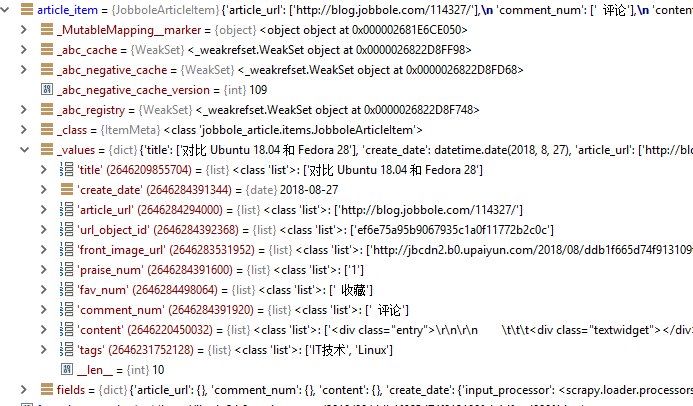
fav\_num = scrapy.Field()

comment\_num = scrapy.Field()

content = scrapy.Field()

tags = scrapy.Field()

再次debug, 查看create\_date的值. 它就变成了一个date的对象, 并且不再放在数组中了.



如果要对所有的字段都添加一个output\_processor = TakeFirst()的方法, 就会显得很麻烦. 为了对所有的字段都取其中的第一个值, 可以定义一个我们自己的item\_loader.

### 自定义itemloader对item中的信息进行处理

前面使用了最基本的方式来解析的文章详情页, 这样使得spider的代码十分长, 不容易维护, 因此可以采用自定义ItemLoder的方式方便对规则的管理

#### 修改item.py. 自定义ItemLoader

**from** **scrapy.loader** **import** ItemLoader

**from** **scrapy.loader.processors** **import** TakeFirst

*#自定义item\_loader, 继承于ItemLoader.*

**class** **JobboleArticleItemLoader**(ItemLoader):

*#查看ItemLoader的源码, 其中可以设置一个default\_output\_processor*

*"""*

*class ItemLoader(object):*

*default\_item\_class = Item*

*default\_input\_processor = Identity()*

*default\_output\_processor = Identity()*

*default\_selector\_class = Selector*

*"""*

default\_output\_processor = TakeFirst()

#### 修改jobbole\_it.py, 导入并使用自定义的JobboleArticleItemLoader

在jobbole.py中就不能再使用系统的ItemLoader了, 而是要使用我们这里自定义的JobboleArticleItemLoader.

**from** **jobbole\_article.items** **import** JobboleArticleItem

**from** **jobbole\_article.items** **import** JobboleArticleItemLoader

**class** **JobboleItSpider**(scrapy.Spider):

name = 'jobbole\_it'

allowed\_domains = ["blog.jobbole.com"]

start\_urls = ['http://blog.jobbole.com/all-posts/']

**def** parse\_detail(self, response):

*# 文章封面图*

front\_image\_url = response.meta.get("front\_image\_url", "")

*# 通过item loader加载item*

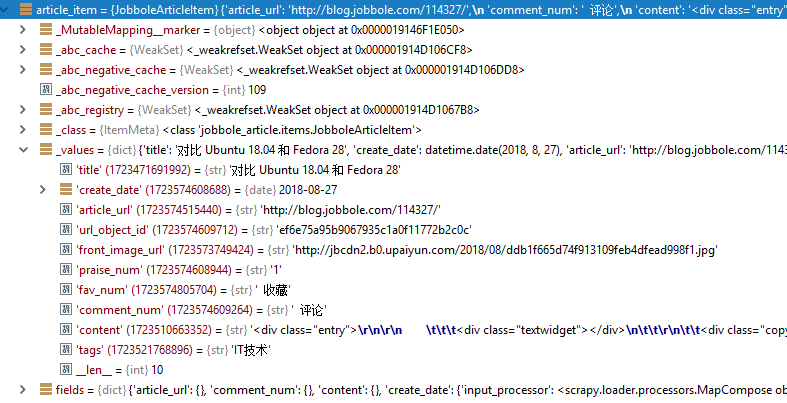
*# item\_loader = ItemLoader(item=JobboleArticleProcessItem(), response=response)*

*# 使用自定义的itemloader*

item\_loader = JobboleArticleItemLoader(item=JobboleArticleProcessItem(), response=response)

现在由于已经使用了添加过TakeFirst()的自定义的ArticleItemLoader, 在item.py中就不需要逐个的添加TakeFirst的语句了.

再次debug, 查看item中的value值, 所有的字段都不再放在一个列表中了, 都只取了原来列表中的第一个值了.



#### 修改items.py, 改进input\_processor.

可以把原来写入到item\_loader中的一些功能, 如get\_md5, 以及对几个\_num进行正则表达式的匹配, 转变create\_date的格式, 这些功能都可以在items.py中实现.

##### 对url\_object\_id进行md5处理

把对url\_object\_id进行md5处理的功能从jobbole\_it.py中转移到items.py中

修改items.py

**import** **hashlib**

**def** get\_md5(url):

*#如果url是以unicode字符串, 就把它进行utf-8的编码, 转换为bytes类型*

**if** isinstance(url, str):

url = url.encode("utf-8")

m = hashlib.md5()

m.update(url)

**return** m.hexdigest()

*# 为了使原来jobbole.py中的item依旧可用, 这里新建一个item, 并在jobbole\_it.py中使用*

**class** **JobboleArticleProcessItem**(scrapy.Item):

*#url是变长的, 通过md5把它变成固定长度的.*

url\_object\_id = scrapy.Field(

input\_processor = MapCompose(get\_md5)

)

修改jobbole\_it.py

**class** **JobboleItSpider**(scrapy.Spider):

name = 'jobbole\_it'

allowed\_domains = ["blog.jobbole.com"]

start\_urls = ['http://blog.jobbole.com/all-posts/']

**def** parse\_detail(self, response):

*# 把生成md5值的操作也放在items.py中进行处理*

item\_loader.add\_value('url\_object\_id', response.url)

item\_loader.add\_value('front\_image\_url', [front\_image\_url])

##### 删除content中的所有html标签

*# 删除文章正文的所有html标签*

**def** remove\_tags(value):

**return** re.sub(r"<.\*?>", "", value).strip()

*# 为了使原来jobbole.py中的item依旧可用, 这里新建一个item, 并在jobbole\_it.py中使用*

**class** **JobboleArticleProcessItem**(scrapy.Item):

content = scrapy.Field(

input\_processor = MapCompose(remove\_tags)

)

##### 对3个\_num进行处理

修改items.py, 完成\_num的正则表达式的匹配功能, 把之前的代码封装成一个函数即可. 这个函数可以用于praise\_num, fav\_num, comment\_num3个\_num.

*# 处理文章的发表时间, 这里不把字符串转换为时间格式, 而是使用mysql的自动转换功能. 也能避免时期无法使用josn序列化的问题.*

**def** date\_convert(value):

**return** value.strip().replace("·", "").strip()

**import** **re**

*# 提取数字*

**def** get\_num(value):

match\_re = re.match(r'.\*?(\d+).\*',value)

**if** match\_re:

x\_num = int(match\_re.group(1))

**else**:

x\_num = 0

**return** x\_num

*# 为了使原来jobbole.py中的item依旧可用, 这里新建一个item, 并在jobbole\_it.py中使用*

**class** **JobboleArticleProcessItem**(scrapy.Item):

title = scrapy.Field()

*# 文章的发表时间, 这里不把字符串转换为时间格式, 而是使用mysql的自动转换功能. 也能避免时期无法使用josn序列化的问题.*

create\_date = scrapy.Field(

input\_processor = MapCompose(date\_convert)

)

*# 文章详情的url*

article\_url = scrapy.Field()

*#url是变长的, 通过md5把它变成固定长度的.*

url\_object\_id = scrapy.Field()

*#封面图的url地址*

front\_image\_url = scrapy.Field()

*#封面图在本地保存的路径.*

front\_image\_path = scrapy.Field()

praise\_num = scrapy.Field(

input\_processor = MapCompose(get\_num)

)

fav\_num = scrapy.Field(

input\_processor = MapCompose(get\_num)

)

comment\_num = scrapy.Field(

input\_processor = MapCompose(get\_num)

)

content = scrapy.Field()

tags = scrapy.Field()

**from** **scrapy.loader** **import** ItemLoader

**from** **scrapy.loader.processors** **import** TakeFirst

*#自定义item\_loader, 继承于ItemLoader.*

**class** **JobboleArticleItemLoader**(ItemLoader):

*#查看ItemLoader的源码, 其中可以设置一个default\_output\_processor*

*"""*

*class ItemLoader(object):*

*default\_item\_class = Item*

*default\_input\_processor = Identity()*

*default\_output\_processor = Identity()*

*default\_selector\_class = Selector*

*"""*

default\_output\_processor = TakeFirst()

进行debug, 查看三者的值.



##### 对tags进行处理

还要对list格式的tags做一个join的操作. 因为提取出来的tags已经是一个list了, 对它进行take\_first的操作只能取到第一个值, 这样也就不合适了. 也要对它进行一个join的操作, 把list中的所有元素组合成一个字符串.

就不能再使用JobboleArticleItemLoader中的default\_output\_processor了, 而是要使用scrapy提供的Join的功能.

**from** **scrapy.loader.processors** **import** Join

tags = scrapy.Field(

*# 对tags单独设置output\_processor, 这样就能覆盖掉JobboleArticleItemLoader默认的default\_output\_processor*

*# 查看Join()的源码, 它的参数是seperator, 即连接符, 在这里把它设置成","*

output\_processor = Join(",")

)

debug查看tags的值. 把原来的列表变成了一个字符串.



但此时其中还包含着评论, 要把评论从中去除掉. 定义一个函数判断tags中的值是否是评论, 如果是评论, 把这个值返回为空即可.

**def** remove\_comment\_tag(value):

**if** "评论" **in** value:

**return** ""

**else**:

**return** value

*# 为了使原来jobbole.py中的item依旧可用, 这里新建一个item, 并在jobbole\_it.py中使用*

**class** **JobboleArticleProcessItem**(scrapy.Item):

tags = scrapy.Field(

input\_processor = MapCompose(remove\_comment\_tag),

*# 对tags单独设置output\_processor, 这样就能覆盖掉JobboleArticleItemLoader默认的default\_output\_processor*

*# 查看Join()的源码, 它的参数是seperator, 即连接符, 在这里把它设置成","*

output\_processor = Join(",")

)

##### 对 front\_image\_url进行处理

还需要注意的是front\_image\_url, 在配置了下载图片的pipeline之后, 这个字段传递回去的一定要是一个列表, 在处理时也是把它放在一个列表中传递过来的. 但现在使用了JobboleArticleItemLoader中的default\_output\_processor = TakeFirst()之后, items.py中的front\_image\_url就会变成字符串的形式. 把它交给image pipeline下载的时候就会抛出异常.

可以定义一个函数, 这个函数什么都不处理, 只返回一个value, 然后定义front\_image\_url中的output\_processor调用这个函数即可. 这样, 既没有修改其中的值又覆盖了默认的output\_processor.

修改 items.py

*# 处理front\_image\_url, 什么也不做, 只返回值*

**def** return\_value(value):

**return** value

*# 为了使原来jobbole.py中的item依旧可用, 这里新建一个item, 并在jobbole\_it.py中使用*

**class** **JobboleArticleProcessItem**(scrapy.Item):

*#封面图的url地址*

*# 由于在下载图片时必须要传递一个列表, 所以要对front\_image\_url单独使用output\_processor, 以覆盖JobboleArticleItemLoader中默认的output\_processor*

front\_image\_url = scrapy.Field(

output\_processor = MapCompose(return\_value)

)

之前写过一个插入jobbole文章的sql语句, 在这个sql语句中是直接取front\_image\_url这个字段, 这时它是一个list, 要改为取这个字段的list中的第一个值.

修改pipelines.py文件

cursor.execute(insert\_sql, (item["title"], item["create\_time"], item["article\_url"], item["url\_object\_id"], item["front\_image\_url"][0], item["front\_image\_path"], item["comment\_num"], item["praise\_num"], item["fav\_num"], item["tags"], item["content"]))

之前在pipelines.py中写了一个JobboleImagePipeline的pipeline, 继承于ImagesPipeline

*# 自定义图片下载处理的中间件*

**class** **JobboleImagePipeline**(ImagesPipeline):

*# 重写函数, 改写item处理完成的函数*

**def** item\_completed(self, results, item, info):

*# result是一个list的结构. 可以获取多个图片保存的信息. 但由于使用yield, 一次只传递过一个item, 所以这里的result中只有一个元素.*

**for** key, value **in** results:

**try**:

front\_image\_path = value.get('path','')

**except** **Exception** **as** e:

front\_image\_path = ''

item["front\_image\_path"] = front\_image\_path

*# 在完成处理后一定要返回item, 这样数据才能被下一个pipeline接收并处理.*

*# 在此处添加断点再次进行调试, 看item中是否保存了图片下载的路径.*

**return** item

这个pipeline会对所有的item都会生效, 在后面爬取知乎的问答时, 文章没有封面图片, 再使用此pipeline进行提取时就可能会出错, 所以这里可以进行一个判断, 只处理有front\_image\_url的那些item. item是一个类字典的结构, 所以if "front\_image\_url" in item这种写法也是正确的.

*# 自定义图片下载处理的中间件*

**class** **JobboleImagePipeline**(ImagesPipeline):

*# 重写函数, 改写item处理完成的函数*

**def** item\_completed(self, results, item, info):

*# result是一个list的结构. 可以获取多个图片保存的信息. 但由于使用yield, 一次只传递过一个item, 所以这里的result中只有一个元素.*

**if** "front\_image\_url" **in** item:

**for** key, value **in** results:

**try**:

front\_image\_path = value.get('path','')

**except** **Exception** **as** e:

front\_image\_path = ''

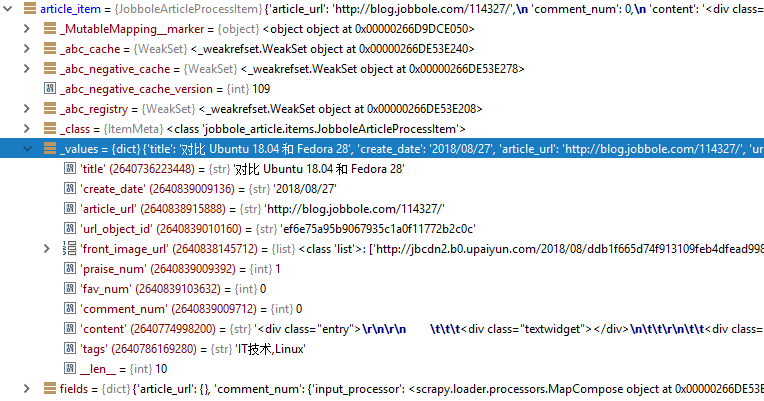
item["front\_image\_path"] = front\_image\_path

*# 在完成处理后一定要返回item, 这样数据才能被下一个pipeline接收并处理.*

*# 在此处添加断点再次进行调试, 看item中是否保存了图片下载的路径.*

**return** item

在jobbole.py的yield article\_item上加上断点, 通过main.py文件进行调试, 在articel\_item的value中就可以看到tags中已经没有了评论的内容. 同时front\_iamge\_url也变成了列表的形式.



把 jobbole\_it.py 中front\_image\_url提取的语句拿出来放在item\_loader = JobboleArticleItemLoader() 之前, 把之前使用 JobboleArticleItem 的方法写的提取数据的代码全部屏蔽即可, 以后就使用 JobboleArticleItemLoader 的方法来提取数据了. 这样代码更加精练, 并且在jobbole\_it.py爬虫文件中只进行数据的提取, 数据的处理都放在items.py中进行. 这也是和scrapy程序解耦的思想相一致的.

#### 最终完成的代码

##### jobbole\_it.py

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

**import** **scrapy**

*# import datetime*

**from** **scrapy.loader** **import** ItemLoader

*# from jobbole\_article.utils.common import get\_md5*

*# from jobbole\_article.items import JobboleArticleItem*

**from** **jobbole\_article.items** **import** JobboleArticleProcessItem

**from** **jobbole\_article.items** **import** JobboleArticleItemLoader

**class** **JobboleItSpider**(scrapy.Spider):

name = 'jobbole\_it'

allowed\_domains = ["blog.jobbole.com"]

start\_urls = ['http://blog.jobbole.com/all-posts/']

**def** parse(self, response):

*# 从文章列表页获取详情页url*

article\_nodes = response.xpath('//div[@class="grid-8"]/div[not(contains(@class, "navigation"))]')

article\_nodes = response.xpath('//div[@class="post floated-thumb"]')

article\_nodes = response.css('div.post.floated-thumb')

**for** article\_node **in** article\_nodes:

*# 列表页文章图片*

front\_image\_url = article\_node.xpath('.//img/@src').extract\_first("")

front\_image\_url = article\_node.css('img::attr(src)').extract\_first("")

*# 列表页文章链接地址*

article\_url = response.css("a.archive-title::attr(href)").extract\_first("")

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

*# 构建请求对象, 把提取到的文章详情页的url交给parse\_detail这个解析函数进行处理. 因为所有文章都可以从all-posts这个列表页获取到, 这里就不需要对链接进行跟进了*

*# meta是字典dict类型的, 通过在Request请求对象中添加meta信息来在不同的解析函数之间传递数据.*

**yield** scrapy.Request(url=article\_url, meta={'front\_image\_url': front\_image\_url}, callback=self.parse\_detail)

*# 测试代码*

*# break*

*# 提取出了包含下一页的a标签. 使用extract\_first就可以避免在最后一页时出错.*

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

*# .next和.page-numbers是属于同一个div的class.*

next\_url = response.css('a.next.page-numbers::attr(href)').extract\_first('')

*# 如果存在下一页的这个值, 就构建Request对象. 在最后一页时这个值不存在.*

**if** next\_url:

*# 使用回调函数把下一页的url传回到parse函数中重复进行提取. 不用写成self.parse(). scrapy是基于异步处理twisted完成的, 会自动根据函数名来调用函数.*

**yield** scrapy.Request(url=next\_url, callback=self.parse)

**def** parse\_detail(self, response):

*# 从response中获取文章封面图*

front\_image\_url = response.meta.get("front\_image\_url", "")

*# 通过item loader加载item*

*# item\_loader = ItemLoader(item=JobboleArticleProcessItem(), response=response)*

*# 使用自定义的itemloader*

item\_loader = JobboleArticleItemLoader(item=JobboleArticleProcessItem(), response=response)

*# 通过item\_loader的add\_css方法加载item*

item\_loader.add\_css('title', 'div.entry-header h1::text')

*# item\_loader.add\_xpath('title', '//div[@class="entry-header"]/h1/text()')*

item\_loader.add\_css('create\_date', 'p.entry-meta-hide-on-mobile::text')

*# item\_loader.add\_xpath('create\_date', '//p[@class="entry-meta-hide-on-mobile"]/text()')*

*# url的值并不是调用css选择器提取出来的, 而是通过response.url直接传递过来的, 要使用add\_value直接添加值的方法. 而add\_css则是调用css选择器把值取出来后再传递给前面的字段中.*

item\_loader.add\_value('article\_url', response.url)

*# 把生成md5值的操作也放在items.py中进行处理*

item\_loader.add\_value('url\_object\_id', response.url)

item\_loader.add\_value('front\_image\_url', [front\_image\_url])

item\_loader.add\_css('praise\_num', 'span.vote-post-up h10::text')

*# item\_loader.add\_xpath('praise\_num', '//span[contains(@class, "vote-post-up")]/h10/text()')*

item\_loader.add\_css('fav\_num', 'span.bookmark-btn::text')

*# item\_loader.add\_xpath('fav\_num', '//span[contains(@class, "bookmark-btn")]/text()')*

item\_loader.add\_css('comment\_num', 'a[href="#article-comment"] span::text')

*# item\_loader.add\_xpath('comment\_num', '//a[@href="#article-comment"]/span/text()"] span::text')*

item\_loader.add\_css('content', 'div.entry')

*# item\_loader.add\_xpath('content', '//div[@class="entry"]')*

item\_loader.add\_css('tags', 'p.entry-meta-hide-on-mobile a::text')

*# item\_loader.add\_xpath('tags', '//p[@class="entry-meta-hide-on-mobile"]/a/text()')*

*# 写完上面的规则之后要调用item\_loader的load\_item()方法对以上的规则进行解析. 解析之后就生成了item对象.*

article\_item = item\_loader.load\_item()

**yield** article\_item

##### items.py

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

*# Define here the models for your scraped items*

*#*

*# See documentation in:*

*# https://doc.scrapy.org/en/latest/topics/items.html*

**import** **re**

**import** **datetime**

**import** **hashlib**

**import** **scrapy**

**from** **scrapy.loader.processors** **import** MapCompose

**from** **scrapy.loader.processors** **import** TakeFirst

**from** **scrapy.loader.processors** **import** Join

**class** **JobboleArticleItem**(scrapy.Item):

title = scrapy.Field()

create\_date = scrapy.Field()

*# 文章详情的url*

article\_url = scrapy.Field()

*#url是变长的, 通过md5把它变成固定长度的.*

url\_object\_id = scrapy.Field()

*#封面图的url地址*

front\_image\_url = scrapy.Field()

*#封面图在本地保存的路径.*

front\_image\_path = scrapy.Field()

praise\_num = scrapy.Field()

fav\_num = scrapy.Field()

comment\_num = scrapy.Field()

content = scrapy.Field()

tags = scrapy.Field()

*# MapCompose这个方法可以传递任意多的函数, 能传递过来的值进行预处理. 如对于title字段, 想要加一个后缀-jobbole, 在这个类之前先定义一个函数add\_jobbole()*

**def** add\_jobbole(value):

**return** value+'-jobbole'

**def** remove\_dot(value):

**return** value.replace("·", "").strip()

*# 处理文章的发表时间, 这里不把字符串转换为时间格式, 而是使用mysql的自动转换功能. 也能避免时期无法使用josn序列化的问题.*

**def** date\_convert(value):

**try**:

create\_time = datetime.datetime.strptime(value, "%Y/%m/**%d**").date()

**except** **Exception** **as** e:

create\_time = datetime.datetime.now().date()

**return** create\_time

*# 提取数字*

**def** get\_num(value):

match\_re = re.match(r'.\*?(\d+).\*',value)

**if** match\_re:

x\_num = int(match\_re.group(1))

**else**:

x\_num = 0

**return** x\_num

*# 删除文章正文的所有html标签*

**def** remove\_tags(value):

**return** re.sub(r"<.\*?>", "", value).strip()

*# 去掉tag中的评论*

**def** remove\_comment\_tag(value):

**if** "评论" **in** value:

**return** ""

**else**:

**return** value

*# 处理front\_image\_url, 什么也不做, 只返回值*

**def** return\_value(value):

**return** value

**def** get\_md5(url):

*#如果url是以unicode字符串, 就把它进行utf-8的编码, 转换为bytes类型*

**if** isinstance(url, str):

url = url.encode("utf-8")

m = hashlib.md5()

m.update(url)

**return** m.hexdigest()

*# 为了使原来jobbole.py中的item依旧可用, 这里新建一个item, 并在jobbole\_it.py中使用*

**class** **JobboleArticleProcessItem**(scrapy.Item):

title = scrapy.Field()

*# 文章的发表时间, 这里不把字符串转换为时间格式, 而是使用mysql的自动转换功能. 也能避免时期无法使用josn序列化的问题.*

create\_date = scrapy.Field(

input\_processor = MapCompose(remove\_dot)

)

*# 文章详情的url*

article\_url = scrapy.Field()

*#url是变长的, 通过md5把它变成固定长度的.*

url\_object\_id = scrapy.Field(

input\_processor = MapCompose(get\_md5)

)

*#封面图的url地址*

*# 由于在下载图片时必须要传递一个列表, 所以要对front\_image\_url单独使用output\_processor, 以覆盖JobboleArticleItemLoader中默认的output\_processor*

front\_image\_url = scrapy.Field(

output\_processor = MapCompose(return\_value)

)

*# 封面图在本地保存的路径.*

front\_image\_path = scrapy.Field()

*# 对于没有点赞数量的帖子, 不能提取出数据, 也就不能进入到input\_processor中进行处理, 所以在数据库写入时使用get方法从item中取出数据..*

praise\_num = scrapy.Field(

input\_processor = MapCompose(get\_num)

)

fav\_num = scrapy.Field(

input\_processor = MapCompose(get\_num)

)

comment\_num = scrapy.Field(

input\_processor = MapCompose(get\_num)

)

content = scrapy.Field(

input\_processor = MapCompose(remove\_tags)

)

tags = scrapy.Field(

input\_processor = MapCompose(remove\_comment\_tag),

*# 对tags单独设置output\_processor, 这样就能覆盖掉JobboleArticleItemLoader默认的default\_output\_processor*

*# 查看Join()的源码, 它的参数是seperator, 即连接符, 在这里把它设置成","*

output\_processor = Join(",")

)

**from** **scrapy.loader** **import** ItemLoader

**from** **scrapy.loader.processors** **import** TakeFirst

*#自定义item\_loader, 继承于ItemLoader.*

**class** **JobboleArticleItemLoader**(ItemLoader):

*#查看ItemLoader的源码, 其中可以设置一个default\_output\_processor*

*"""*

*class ItemLoader(object):*

*default\_item\_class = Item*

*default\_input\_processor = Identity()*

*default\_output\_processor = Identity()*

*default\_selector\_class = Selector*

*"""*

default\_output\_processor = TakeFirst()

##### pipelines.py

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

**from** **scrapy.pipelines.images** **import** ImagesPipeline

**import** **codecs**, **json**

**import** **datetime**

**class** **JobboleArticlePipeline**(object):

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

**return** item

*# 自定义图片下载处理的中间件*

**class** **JobboleImagePipeline**(ImagesPipeline):

*# 重写函数, 改写item处理完成的函数*

**def** item\_completed(self, results, item, info):

*# result是一个list的结构. 可以获取多个图片保存的信息. 但由于使用yield, 一次只传递过一个item, 所以这里的result中只有一个元素.*

**if** "front\_image\_url" **in** item:

**for** key, value **in** results:

**try**:

front\_image\_path = value.get('path','')

**except** **Exception** **as** e:

front\_image\_path = ''

item["front\_image\_path"] = front\_image\_path

*# 在完成处理后一定要返回item, 这样数据才能被下一个pipeline接收并处理.*

*# 在此处添加断点再次进行调试, 看item中是否保存了图片下载的路径.*

**return** item

*# 自定义管道将Item导出为Json文件*

**class** **JsonWithEncodingPipeline**(object):

*# 初始化时调用*

**def** \_\_init\_\_(self):

*# 打开json文件*

*# 使用codecs完成文件的打开和写入能够解决编码方面的问题*

self.file = codecs.open('article.json', 'w', encoding="utf-8")

*# 重写Item处理*

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

*# 先把item转换为dict格式, 再使用json.dump把它转换为字符串.*

*# 需要关闭ensure\_ascii, 使用utf-8编码写入数据, 否则会以ascii方式写入, 中文字符就无法正确显示.*

*# 用这种方法写入的每一行数据都是一个字典, 整体上其实并不是一个真正的json文件.*

lines = json.dumps(dict(item), ensure\_ascii=False, indent=2) + "**\n**"

*# 将一行数据写入*

self.file.write(lines)

*# 重写process\_item方法时必须要使用return把它返回去, 以供其它的pipeline使用.*

**return** item

*# 爬虫结束关闭spider时调用spider\_closed方法*

**def** spider\_closed(self, spider):

*# 关闭文件句柄*

self.file.close()

**from** **scrapy.exporters** **import** JsonItemExporter

*# 调用scrapy提供的json export导出json文件.*

**class** **JsonExporterPipeline**(object):

*# 调用scrapy提供的json exporter导出json文件*

**def** \_\_init\_\_(self):

*# 以二进制方法打开json文件*

self.file = open('json\_item\_exporter.json', 'wb')

*# 实例化一个JsonItemExporter对象exporter, 在实例化时需要传递几个参数.*

self.exporter = JsonItemExporter(self.file, encoding="utf-8", ensure\_ascii=False, indent=2)

*# 使用start\_exporting方法开始导出*

self.exporter.start\_exporting()

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

self.exporter.export\_item(item)

**return** item

**def** close\_spider(self, spider):

*# 使用finish\_exporting方法结束导出*

self.exporter.finish\_exporting()

self.file.close()

**import** **MySQLdb**

*# 同步机制写入数据库*

**class** **MysqlPipeline**(object):

**def** \_\_init\_\_(self):

*#创建一个连接MySQLdb.connect('host', 'user', 'password', 'dbname', charset, use\_unicode), 可以打开connect函数查看其源码.*

*# self.conn = MySQLdb.connect('127.0.0.1', 'root', 'mysql', ' article\_spider', charset="utf8", use\_unicode=True)*

self.conn = MySQLdb.connect(

*# host = 'sh-cdb-pzoa3tqh.sql.tencentcdb.com',*

host = '127.0.0.1',

user = 'root',

password = 'Xzq@8481',

db = 'jobbole\_article',

*# port = 63104,*

port = 3306,

charset = 'utf8',

)

*#执行数据库的具体操作是由cursor来完成的.*

self.cursor = self.conn.cursor()

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

*#这里insert插入的操作要与之前数据库中设置的字段的名称和顺序相同.*

insert\_sql = '''

insert ignore into article(title, create\_date, article\_url, url\_object\_id, front\_image\_url, front\_image\_path, comment\_num, praise\_num, fav\_num, tags, content)

VALUES(**%s**, **%s**, **%s**, **%s**, **%s**, **%s**, **%s**, **%s**, **%s**, **%s**, **%s**)

'''

*# 因为mysql中定义的create\_date为日期格式的, 所以要先把item中字符串格式的create\_date转换为日期格式*

*# item['create\_date'] = datetime.datetime.strptime(item['create\_date'], "%Y/%m/%d").date()*

*# 注意, 当点赞数量为0时, 在jobbole.py中是无法取到值的, 也就无法进入到items.py中的input\_processor进行处理, item中就没有这个字段的值, 所以这里对praise\_num要使用get方法进行选择.*

self.cursor.execute(insert\_sql, (item["title"], item["create\_date"], item["article\_url"], item["url\_object\_id"], item["front\_image\_url"][0], item['front\_image\_path'], item["comment\_num"], item.get("praise\_num"), item["fav\_num"], item["tags"]), item["content"])

*# 注意使用的是conn.commit, 不是cursor.commit*

self.conn.commit()

**def** spider\_closed(self, spider):

self.conn.close()

**import** **MySQLdb.cursors**

**from** **twisted.enterprise** **import** adbapi

*# 异步操作写入数据库*

**class** **MysqlTwistedPipline**(object):

*# from\_settings和\_\_init\_\_这两个方法就能实现在启动spider时, 就把dbpool传递进来*

**def** \_\_init\_\_(self, dbpool):

self.dbpool = dbpool

*# 在初始化时scrapy会调用from\_settings方法, 将setting文件中的配置读入, 成为一个settings对象, 这种写法是固定的, 其中的参数不能修改.*

@classmethod

**def** from\_settings(cls, settings):

dbparas = dict(

host = settings["MYSQL\_HOST"], *#可以在settings中设置此pipeline, 在此处放置断点, 进行debug, 查看能否导入. 在attribute中可以看到settings中定义的所有的值. F6执行, 就能看到取到了settings中设置的host的值了.*

port = settings["MYSQL\_PORT"],

db = settings["MYSQL\_DBNAME"],

user = settings["MYSQL\_USERNAME"],

passwd = settings["MYSQL\_PASSWORD"],

charset = "utf8",

*# pymysql模块中也有类似的模块pymysql.cursors.DictCursor*

cursorclass = MySQLdb.cursors.DictCursor,

use\_unicode = True

)

*# 创建twisted的mysql连接池, 使用twisted的连接池, 就能把mysql的操作转换为异步操作.*

*# twisted只是提供了一个异步的容器, 并没有提供连接mysql的方法, 所以还需要MySQLdb的连接方法.*

*# adbapi可以将mysql的操作变成异步化的操作. 查看ConectionPool, def \_\_init\_\_(self, dbapiName, \*connargs, \*\*connkw).*

*# 需要指定使用的连接模块的模块名, 第一个参数是dbapiName, 即mysql的模块名MySQLdb. 第二个参数是连接mysql的参数, 写为可变化的参数形式. 查看MySQLdb的源码, 在from MySQLdb.connections import Connection中查看Connection的源码, 在class Connection中就能看到MySQLdb模块在连接mysql数据库时需要传递的参数. param这个dict中参数的名称要与其中的参数名称保持一致. 即与connections.py中 class Connection中的def \_\_init\_\_中定义的参数保持一致.*

dbpool = adbapi.ConnectionPool("MySQLdb", \*\*dbparas)

*# 如果对上面的写法不太理解, 可以写成下面的形式*

*# dbpool = adbapi.ConnectionPool("MySQLdb", host = settings["MYSQL\_HOST"], db = settings["MYSQL\_DBNAME"], user = settings["MYSQL\_USERNAME"], passwd = settings["MYSQL\_PASSWORD"], charset = "utf8", cursorclass = MySQLdb.cursors.DictCursor, use\_unicode = True)*

*# 因为使用@classmethod把这个方法转换为类方法了, 所以cls就是指的MysqlTwistedPipline这个类, 所以cls(dbpool) 就相当于使用dbpool这个参数实例化MysqlTwistedPipline类的一个对象, 再把这个对象返回. 然后在init方法中接收这里创建的异步连接对象.*

**return** cls(dbpool)

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

*# 使用twisted将mysql数据库的插入变成异步插入, 第一个参数是自定义的函数, runInteraction可以把这个函数的操作变成异步的操作. 第二个参数是要插入的数据, 这里是item.*

query = self.dbpool.runInteraction(self.do\_insert, item)

*#处理异常, 这里也可以不传递item和spider*

query.addErrback(self.handle\_error, item, spider)

*# 自定义错误处理, 处理异步插入的异常, 这里也可以不传递item和spider, 只传递failure即可.*

**def** handle\_error(self, failure, item, spider):

**print**(failure)

**print**(item)

*# 执行具体的插入, 此时的cursor就是self.dbpool.runInteraction中传递过来的cursor, 使用这个cursor, 就可以把mysql的操作变成异步的操作. 并且此时也不用再手动执行commit的操作了.*

**def** do\_insert(self, cursor, item):

insert\_sql = '''

INSERT IGNORE INTO

article(title, create\_date, article\_url, url\_object\_id, front\_image\_url, front\_image\_path, comment\_num, praise\_num, fav\_num, tags, content)

VALUES

(**%s**, **%s**, **%s**, **%s**, **%s**, **%s**, **%s**, **%s**, **%s**, **%s**, **%s**)

'''

*# 注意, 当点赞数量为0时, 在jobbole.py中是无法取到值的, 也就无法进入到items.py中的input\_processor进行处理, item中就没有这个字段的值, 所以这里对praise\_num要使用get方法进行选择.*

cursor.execute(insert\_sql, (item["title"], item["create\_date"], item["article\_url"], item["url\_object\_id"], item["front\_image\_url"][0], item["front\_image\_path"], item["comment\_num"], item.get("praise\_num","0"), item["fav\_num"], item["tags"], item["content"]))

##### settings.py

BOT\_NAME = 'jobbole\_article'

SPIDER\_MODULES = ['jobbole\_article.spiders']

NEWSPIDER\_MODULE = 'jobbole\_article.spiders'

*# Obey robots.txt rules*

ROBOTSTXT\_OBEY = False

*# Configure a delay for requests for the same website (default: 0)*

*# See https://doc.scrapy.org/en/latest/topics/settings.html#download-delay*

*# See also autothrottle settings and docs*

DOWNLOAD\_DELAY = 3

*# Disable cookies (enabled by default)*

COOKIES\_ENABLED = False

*# Override the default request headers:*

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; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/65.0.3325.181 Safari/537.36'

}

*# Configure item pipelines*

*# See https://doc.scrapy.org/en/latest/topics/item-pipeline.html*

ITEM\_PIPELINES = {

'jobbole\_article.pipelines.JsonWithEncodingPipeline': 300,

'jobbole\_article.pipelines.JsonExporterPipeline': 301,

*# 'scrapy.pipelines.images.ImagesPipeline' : 200,*

'jobbole\_article.pipelines.JobboleImagePipeline': 200,

*# 'jobbole\_article.pipelines.MysqlPipeline': 302,*

'jobbole\_article.pipelines.MysqlTwistedPipline': 302,

}

*################################################################*

*# 图片下载设置*

*# 设置图片url的字段, scrapy将从item中找出此字段进行图片下载*

IMAGES\_URLS\_FIELD = "front\_image\_url"

*# 设置图片下载保存的目录, 这里不直接使用本机的绝对路径, 而是通过程序获取路径. 这样在程序迁移后也能够正常运行.*

**import** **os**

*# os.path.dirname(\_\_file\_\_)获取当前文件所在的文件夹名称.*

*# os.path.abspath(os.path.dirname(\_\_file\_\_))获得当前文件所在的绝对路径.*

project\_path = os.path.dirname(os.path.abspath(\_\_file\_\_))

*# 想要把下载的图片保存在与settings同目录的images文件夹中, 要先在项目中新建此文件夹.*

*# IMAGES\_STORE中定义保存图片的路径*

IMAGES\_STORE = os.path.join(project\_path, "images")

*# 表示只下载大于100x100的图片, 查看images.py的源码, 程序会自动的从settings.py中读取设置的IMAGES\_MIN\_HEIGHT和IMAGES\_MIN\_WIDTH值*

*# IMAGES\_MIN\_HEIGHT = 100*

*# IMAGES\_MIN\_WIDTH = 100*

**if** **not** os.path.exists(IMAGES\_STORE):

os.mkdir(IMAGES\_STORE)

**else**:

**pass**

*################################################################*

*################################################################*

*# mysql配置*

MYSQL\_HOST = 'sh-cdb-pzoa3tqh.sql.tencentcdb.com'

*#这里设置的是数据库的名字, 不是数据表的名字*

MYSQL\_DBNAME = 'jobbole\_article'

MYSQL\_USERNAME = 'root'

MYSQL\_PASSWORD = 'Xzq@8481'

MYSQL\_PORT = 63104

*################################################################*

## 通过downloadmiddleware随机更换user-agent

### 实现user-agent的自动更换的方法

第一种方法, 在爬虫中设置一个全局性的参数headers, 每次yield发送Request时都使用headers=headers把这个请求头添上, 这也是之前spider中使用的方法, 使用这种方法所有的请求都只能使用同一个useragent, 无法进行随机选择.

第二种方法, 在settings中添加设置, 把所能想到的全部的user-agent都写在这个列表中, 如 user\_agent\_list. 然后修改爬虫文件, 从类中引入这个列表, 每次发送请求都从列表中随机选择一个user\_agent.

但必须要把生成随机user-agent的代码复制到每一个需要发送Request的类方法中. 并且如果爬虫太多, 一个个的进行修改就有些不现实.

第三种方法, 使用downloadermiddleware

在downloader和engine之间有一层middleware, 叫做download middleware, 是requests从engine发送到downloader和downloader下载数据返回到engine时都必须要经过的middleware. 这是一个全局的middleware, 只要在这个downoad midddleware中设置user-agent, 就不用把随机更换user-agent的代码写到spider类中的每一个方法中了.

只要定义一个downloadermiddleware的类, 来随机的更换user-agent, 并把它放在settings中的DOWNLOADER\_MIDDLEWARES中去, 就能实现对所有的请求都使用随机的user-agent.

### 查看scrapy自带的UserAgentMiddleware的源码

scrapy本身提供了一个user-agent的mdiddleware, scrapy.downloadermiddlewares.useragent.py,

"C:\Users\David\Envs\python3\_spider\Lib\site-packages\scrapy\downloadermiddlewares\useragent.py"

*"""Set User-Agent header per spider or use a default value from settings"""*

**from** **scrapy** **import** signals

**class** **UserAgentMiddleware**(object):

*"""This middleware allows spiders to override the user\_agent"""*

**def** \_\_init\_\_(self, user\_agent='Scrapy'):

self.user\_agent = user\_agent

@classmethod

**def** from\_crawler(cls, crawler):

o = cls(crawler.settings['USER\_AGENT'])

crawler.signals.connect(o.spider\_opened, signal=signals.spider\_opened)

**return** o

**def** spider\_opened(self, spider):

self.user\_agent = getattr(spider, 'user\_agent', self.user\_agent)

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

**if** self.user\_agent:

request.headers.setdefault(b'User-Agent', self.user\_agent)

在其中可以看到, user-agent实际上有一个Scrapy的默认值. 然后设置了一个静态的方法from\_crawler, 可以直接通过类来调用这个方法. 在这个from\_crawler的方法中, 会把当前的爬虫crawler传递进来, 然后从settings中读取USER\_AGENT的值. 如果读取不到, 就会使用默认的"Scrapy". 所以在settings中要单独设置一个USER\_AGENT的变量.

接下来process\_request的函数, 如果在自定义UserAgentMiddleware或downloader middleware的时候, 想要对request进行进一步的处理, 就要使用process\_request这个函数. 它对每一个request中的headers都设置了一个默认的self.user\_agent的值.

查看官方文档, Activating a downloader middleware中如何去自定义一个middleware. 这个downlaoder middleware既可以处理request又可以处理response.

重写了process\_request(request, spider)函数, 就会处理request

重写了process\_response(request, response, spider), 就会处理response

重写了process\_exception(request, exception, spider), 就会处理异常

#### 启用自定义的downloader middleware

如果要启用定义user-agent的download middleware, 就要把原来settings.py中scrapy的useragentmiddleware设置为None或注释掉或者把自定义的useragent downloadmiddleware的值设置的比系统的大, 这样自定义的后执行, 就覆盖掉了系统的useragent download midddleware.

如:

DOWNLOADER\_MIDDLEWARES = {

'myproject.middlewares.CustomDownloaderMiddleware': 543,

'scrapy.downloadermiddlewares.useragent.UserAgentMiddleware': None,

}

### 使用Fake Useragent

以上这样写是可以实现的, 但在后期还会需要手动去维护settings中的user\_agent\_list的列表. 是否还有更便携的方法.

<https://github.com/hellysmile/fake-useragent>

http://useragentstring.com/

https://www.w3schools.com/browsers/default.asp

#### Installation

pip install fake-useragent

#### Usage

**from** **fake\_useragent** **import** UserAgent

*#使用UserAgent()的类生成实例.*

ua = UserAgent()

*# 可以通过以下命令来取到相应浏览器的随机的user-agent.*

ua.ie

*# Mozilla/5.0 (Windows; U; MSIE 9.0; Windows NT 9.0; en-US);*

ua.msie

*# Mozilla/5.0 (compatible; MSIE 10.0; Macintosh; Intel Mac OS X 10\_7\_3; Trident/6.0)'*

ua['Internet Explorer']

*# Mozilla/5.0 (compatible; MSIE 8.0; Windows NT 6.1; Trident/4.0; GTB7.4; InfoPath.2; SV1; .NET CLR 3.3.69573; WOW64; en-US)*

ua.opera

*# Opera/9.80 (X11; Linux i686; U; ru) Presto/2.8.131 Version/11.11*

ua.chrome

*# Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.2 (KHTML, like Gecko) Chrome/22.0.1216.0 Safari/537.2'*

ua.google

*# Mozilla/5.0 (Macintosh; Intel Mac OS X 10\_7\_4) AppleWebKit/537.13 (KHTML, like Gecko) Chrome/24.0.1290.1 Safari/537.13*

ua['google chrome']

*# Mozilla/5.0 (X11; CrOS i686 2268.111.0) AppleWebKit/536.11 (KHTML, like Gecko) Chrome/20.0.1132.57 Safari/536.11*

ua.firefox

*# Mozilla/5.0 (Windows NT 6.2; Win64; x64; rv:16.0.1) Gecko/20121011 Firefox/16.0.1*

ua.ff

*# Mozilla/5.0 (X11; Ubuntu; Linux i686; rv:15.0) Gecko/20100101 Firefox/15.0.1*

ua.safari

*# Mozilla/5.0 (iPad; CPU OS 6\_0 like Mac OS X) AppleWebKit/536.26 (KHTML, like Gecko) Version/6.0 Mobile/10A5355d Safari/8536.25*

*# and the best one, random via real world browser usage statistic*

*# 使用这个命令来取得随机的user-agent*

ua.random

在github的源码中并没有看到useragent的列表, 它实际上在下面的地址上维护了一些useragent的列表

<http://d2g6u4gh6d9rq0.cloudfront.net/browsers/fake_useragent_0.1.9.json>

<http://d2g6u4gh6d9rq0.cloudfront.net/browsers/fake_useragent_0.1.10.json>

#### Notes

fake-useragent store collected data at your os temp dir, like /tmp

If You want to update saved database just:

from fake\_useragent import UserAgent

ua = UserAgent()

ua.update()

If You don't want cache database or no writable file system:

from fake\_useragent import UserAgent

ua = UserAgent(cache=False)

Sometimes, [useragentstring.com](http://useragentstring.com/) or [w3schools.com](https://www.w3schools.com/browsers/browsers_stats.asp) changes their html, or down, in such case fake-useragent uses CDN [cloudfront](http://d2g6u4gh6d9rq0.cloudfront.net/browsers/fake_useragent_0.1.9.json) fallback

If You don't want to use hosted cache server (version 0.1.5 added)

from fake\_useragent import UserAgent

ua = UserAgent(use\_cache\_server=False)

In very rare case, if hosted cache server and sources will be unavailable fake-useragent wont be able to download data: (version 0.1.3 added)

from fake\_useragent import UserAgent

ua = UserAgent()

# Traceback (most recent call last):

# ...

# fake\_useragent.errors.FakeUserAgentError

# You can catch it via

from fake\_useragent import FakeUserAgentError

try:

ua = UserAgent()

except FakeUserAgentError:

pass

If You will try to get unknown browser: (version 0.1.3 changed)

from fake\_useragent import UserAgent

ua = UserAgent()

ua.best\_browser

# Traceback (most recent call last):

# ...

# fake\_useragent.errors.FakeUserAgentError

You can completely disable ANY annoying exception with adding fallback: (version 0.1.4 added)

import fake\_useragent

ua = fake\_useragent.UserAgent(fallback='Your favorite Browser')

# in case if something went wrong, one more time it is REALLY!!! rare case

ua.random == 'Your favorite Browser'

Want to control location of data file? (version 0.1.4 added)

import fake\_useragent

# I am STRONGLY!!! recommend to use version suffix

location = '/home/user/fake\_useragent%s.json' % fake\_useragent.VERSION

ua = fake\_useragent.UserAgent(path=location)

ua.random

If you need to safe some attributes from overriding them in UserAgent by \_\_getattr\_\_ method use safe\_attrs you can pass there attributes names. At least this will prevent you from raising FakeUserAgentError when attribute not found.

For example, when using fake\_useragent with [injections](https://github.com/tailhook/injections) you need to:

import fake\_useragent

ua = fake\_useragent.UserAgent(safe\_attrs=('\_\_injections\_\_',))

Please, do not use if you don't understand why you need this. This is magic for rarely extreme case.

#### Experiencing issues???

Make sure that You using latest version!!!

pip install -U fake-useragent

Check version via python console: (version 0.1.4 added)

import fake\_useragent

print(fake\_useragent.VERSION)

### 使用fake useragent来随机切换user-agent

#### 基本使用方法

修改middlewares.py

**import** **random**

**from** **fake\_useragent** **import** UserAgent

**class** **RandomFakeUserAgentMiddleware**(object):

**def** \_\_init\_\_(self, crawler):

*# 调用父类的初始化方法进行初始化.*

super(RandomFakeUserAgentMiddleware, self).\_\_init\_\_()

self.ua = UserAgent()

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

*# 处理所有的request, 把它的默认的headers中的user-agent设置为randowm\_agent*

request.headers.setdefault('User-Agent', self.ua.random)

在settings.py中启用RandomFakeUserAgentMiddleware 这个DOWNLOADER\_MIDDLEWARE

DOWNLOADER\_MIDDLEWARES = {

*# 'jobbole\_article.middlewares.JobboleArticleDownloaderMiddleware': 543,*

'jobbole\_article.middlewares.RandomFakeUserAgentMiddleware': 100,

}

#### 使用本地的fake\_useragent.json文件

在fake\_useragent服务器无法连接的时候, 会出现FakeUserAgentError的异常, 可以把最新的fake\_useragent的json文件下载到本地, 当出现错误时, 就使用本地的json文件, 从中读取user-agetn

<http://d2g6u4gh6d9rq0.cloudfront.net/browsers/fake_useragent_0.1.10.json>

把其中的所有user-agent都复制下来, 修改为列表的形式, 保存到文件中, 放在项目根目录下

import random

from fake\_useragent import UserAgent

from fake\_useragent import FakeUserAgentError

try:

ua = UserAgent()

user\_agent = ua.random

**except** FakeUserAgentError:

# 如果ua = UserAgent() 出现错误, 就读取本地的 fake\_useragent.json文件, 从中随机选择一个.

**with** open("fake\_useragent.json", "r") **as** f:

ua\_list = json.load(f)

user\_agent = random.choice(ua\_list)

**print**(user\_agent)

修改middlewares.py, 使用本地的fake\_useragent.json文件

**import** **random**

**import** **os**

**import** **json**

**class** **RandomLocalUserAgentMiddleware**(object):

**def** \_\_init\_\_(self, crawler):

super(RandomLocalUserAgentMiddleware, self).\_\_init\_\_()

self.project\_path = os.path.dirname(os.path.dirname(os.path.abspath(\_\_file\_\_)))

self.json\_file = os.path.join(project\_path, "fake\_useragent.json")

**with** open(self.json\_file, "r") **as** f:

self.ua\_list = json.load(f)

*# 把随机选择的功能放在\_\_init\_\_中, 只能在启动爬虫时随机选择一个user-agent*

*# self.user\_agent = random.choice(self.ua\_list)*

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

self.user\_agent = random.choice(self.ua\_list)

*# request.headers['User-Agent'] = user\_agent*

request.headers.setdefault('User-Agent', self.user\_agent)

**pass**

在settings.py中启动RandomLocalUserAgentMiddleware

DOWNLOADER\_MIDDLEWARES = {

*# 'jobbole\_article.middlewares.JobboleArticleDownloaderMiddleware': 543,*

*# 'jobbole\_article.middlewares.RandomFakeUserAgentMiddleware': 100,*

'jobbole\_article.middlewares.RandomLocalUserAgentMiddleware': 100,

}

在process\_request中的pass处打上断点, 在jobbole\_it.py中的response处打上断点, 查看是否随机取到了user\_agent的值, 查看response中的reqeust.headers中的user-agent.

也可以继承scrapy自带的UserAgentMiddleware

**import** **random**

**from** **scrapy.downloadermiddlewares.useragent** **import** UserAgentMiddleware

**class** **UADMiddleware**(UserAgentMiddleware):

**def** \_\_init\_\_(self, ua=''):

super(UADMiddleware, self).\_\_init\_\_()

self.project\_path = os.path.dirname(os.path.dirname(os.path.abspath(\_\_file\_\_)))

self.json\_file = os.path.join(self.project\_path, "fake\_useragent.json")

**with** open(self.json\_file, "r") **as** f:

self.ua\_list = json.load(f)

*# 把随机选择的功能放在\_\_init\_\_中, 只能在启动爬虫时随机选择一个user-agent*

*# self.user\_agent = random.choice(self.ua\_list)*

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

self.user\_agent = random.choice(self.ua\_list)

**print**("当前的用户代理是:**%s**" % self.user\_agent)

*# request.headers['User-Agent'] = user\_agent*

request.headers.setdefault('User-Agent', self.user\_agent)

在settings.py中启动UADMiddleware

DOWNLOADER\_MIDDLEWARES = {

*# 'jobbole\_article.middlewares.JobboleArticleDownloaderMiddleware': 543,*

*# 'jobbole\_article.middlewares.RandomFakeUserAgentMiddleware': 100,*

*# 'jobbole\_article.middlewares.RandomLocalUserAgentMiddleware': 100,*

'jobbole\_article.middlewares.UADMiddleware': 100,

}

## scrapy实现ip代理池

爬取时尽量不要让本机的ip被禁止掉.

关键是如何设置一个ip的代理池, 从代理池中随机的取出一些ip地址.

### 爬虫代理哪家强？十大付费代理详细对比评测出炉！

<https://cuiqingcai.com/5094.html>

|  |  |  |
| --- | --- | --- |
| 代理商家 | 套餐类型 | 官方网站 |
| 芝麻 HTTP 代理 | 默认版 | [http://h.zhimaruanjian.com](http://h.zhimaruanjian.com/) |
| 蘑菇代理 | 默认版 | [http://www.mogumiao.com](http://www.mogumiao.com/) |
| 讯代理 | 优质代理 | [http://www.xdaili.cn](http://www.xdaili.cn/) |
| 混播代理 |

注：其中蘑菇代理、太阳 HTTP 代理、芝麻 HTTP 代理的默认版表示此网站只有这一种代理，不同套餐仅是时长区别，代理质量没有差别。

经过测评，初步得到如下统计结果：

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 代理商家 | 套餐类型 | 测试次数 | 有效次数 | 可用率 | 响应时间均值 | 响应时间方差 |
| 芝麻 HTTP 代理 | 默认版 | 500 | 495 | 99.00% | 0.916853 | 1.331989 |
| 蘑菇代理 | 默认版 | 500 | 497 | 99.40% | 1.0985725 | 9.532586 |
| 讯代理 | 优质代理 | 500 | 495 | 99.00% | 1.0512681 | 6.4247565 |
| 混播代理 | 500 | 494 | 98.80% | 1.0664985 | 6.451699 |
| 独享代理 | 500 | 500 | 100% | 0.7056521 | 0.35416448 |
| 短效优质代理 | 500 | 488 | 97.60% | 1.5625348 | 8.121197 |

### 使用2个代理获取代理地址

讯代理 api

每天累计代理数>15000个

单个代理有效时长5~30分钟

代理有效率>95%

单次提取代理数最多20个

日提取代理上限1000\*N个

API调取频率5秒

支持类型HTTP/HTTPS

IP并发调用/多机器调用支持

匿名度高匿

<http://api.xdaili.cn/xdaili-api//greatRecharge/getGreatIp?spiderId=9b3446e17b004293976e09a081022d73&orderno=YZ20188178415lSPZWO&returnType=2&count=1>

# '{"ERRORCODE":"10055","RESULT":"提取太频繁,请按规定频率提取!"}'

# '{"ERRORCODE":"0","RESULT":[{"port":"48448","ip":"115.203.196.254"}]}'

蘑菇代理api

包天套餐

￥6元

每天使用上限 2000

一次最多提取数量最多100个

API最快调取频率1秒

有效时长1~5分钟

HTTP/HTTPS协议支持

订单到期预警支持

白名单绑定

<http://piping.mogumiao.com/proxy/api/get_ip_al?appKey=04756895ae5b498bb9b985798e990b9f&count=1&expiryDate=0&format=1&newLine=2>

# '{"code":"3001","msg":"提取频繁请按照规定频率提取!"}'

# '{"code":"0","msg":[{"port":"35379","ip":"117.60.2.113"}]}'

*# 从2个不同的api中获取代理*

**def** get\_random\_ip():

mogu\_api = 'http://piping.mogumiao.com/proxy/api/get\_ip\_al?appKey=04756895ae5b498bb9b985798e990b9f&count=1&expiryDate=0&format=1&newLine=2'

*# '{"code":"3001","msg":"提取频繁请按照规定频率提取!"}'*

*# '{"code":"0","msg":[{"port":"35379","ip":"117.60.2.113"}]}'*

xdaili\_api = 'http://api.xdaili.cn/xdaili-api//greatRecharge/getGreatIp?spiderId=9b3446e17b004293976e09a081022d73&orderno=YZ20188178415lSPZWO&returnType=2&count=1'

*# '{"ERRORCODE":"10055","RESULT":"提取太频繁,请按规定频率提取!"}'*

*# '{"ERRORCODE":"0","RESULT":[{"port":"48448","ip":"115.203.196.254"}]}'*

api\_list = [mogu\_api, xdaili\_api]

*# 打乱api\_list的顺序, 以免列表中第1个代理使用的次数过多*

random.shuffle(api\_list)

**for** api **in** api\_list:

response = requests.get(api)

js\_str = json.loads(response.text)

*# 如果正确提取到了ip地址*

**if** js\_str.get('code') == '0' **or** js\_str.get('ERRORCODE') == '0':

*# 从中取出ip*

**for** i, j **in** js\_str.items():

**if** j != '0':

*# proxies = {*

*# "http": "http://{}:{}".format(j[0].get('ip'), j[0].get('port')),*

*# "https": "https://{}:{}".format(j[0].get('ip'), j[0].get('port'))*

*# }*

proxies = "http://{}:{}".format(j[0].get('ip'), j[0].get('port'))

logger.info("从 {} 获取了一个代理 {}".format(re.split(r'.c', api)[0], proxies))

*# print("从{}获取了一个代理{}".format(re.split(r'.c', api)[0], proxies))*

**return** proxies

**break**

**else**:

*# print("提取太频繁, 等待中...")*

logger.info("api {} 提取太频繁, 等待中".format(api))

time.sleep(random.randint(5,10))

### 修改middlewares.py, 同时切换代理和ua

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

**from** **fake\_useragent** **import** UserAgent

**import** **time**

**import** **re**

**import** **os**

**import** **random**

**from** **twisted.internet** **import** defer

**from** **twisted.internet.error** **import** TimeoutError, ConnectionRefusedError, ConnectError, ConnectionLost, TCPTimedOutError, ConnectionDone

**import** **logging**

**import** **json**

**import** **requests**

logger = logging.getLogger(\_\_name\_\_)

**class** **RandomFakeUserAgentMiddleware**(object):

**def** \_\_init\_\_(self, crawler):

*# 调用父类的初始化方法进行初始化.*

super(RandomFakeUserAgentMiddleware, self).\_\_init\_\_()

self.ua = UserAgent()

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

*# 处理所有的request, 把它的默认的headers中的user-agent设置为randowm\_agent*

request.headers.setdefault('User-Agent', self.ua.random)

*# from scrapy.downloadermiddlewares.useragent import UserAgentMiddleware*

*# 也可以继承自UserAgentMiddleware*

*# class RandomLocalUserAgentMiddleware(UserAgentMiddleware):*

**class** **RandomLocalUserAgentMiddleware**(object):

**def** \_\_init\_\_(self, user\_agent=''):

super(RandomLocalUserAgentMiddleware, self).\_\_init\_\_()

self.project\_path = os.path.dirname(os.path.dirname(os.path.abspath(\_\_file\_\_)))

self.json\_file = os.path.join(self.project\_path, "fake\_useragent.json")

**with** open(self.json\_file, "r") **as** f:

self.ua\_list = json.load(f)

*# 把随机选择的功能放在\_\_init\_\_中, 只能在启动爬虫时随机选择一个user-agent*

*# self.user\_agent = random.choice(self.ua\_list)*

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

self.user\_agent = random.choice(self.ua\_list)

request.headers.setdefault('User-Agent', self.user\_agent)

**pass**

*# 从2个不同的api中获取代理*

**def** get\_random\_ip():

mogu\_api = 'http://piping.mogumiao.com/proxy/api/get\_ip\_al?appKey=04756895ae5b498bb9b985798e990b9f&count=1&expiryDate=0&format=1&newLine=2'

*# '{"code":"3001","msg":"提取频繁请按照规定频率提取!"}'*

*# '{"code":"0","msg":[{"port":"35379","ip":"117.60.2.113"}]}'*

xdaili\_api = 'http://api.xdaili.cn/xdaili-api//greatRecharge/getGreatIp?spiderId=9b3446e17b004293976e09a081022d73&orderno=YZ20188178415lSPZWO&returnType=2&count=1'

*# '{"ERRORCODE":"10055","RESULT":"提取太频繁,请按规定频率提取!"}'*

*# '{"ERRORCODE":"0","RESULT":[{"port":"48448","ip":"115.203.196.254"}]}'*

api\_list = [mogu\_api, xdaili\_api]

*# 打乱api\_list的顺序, 以免列表中第1个代理使用的次数过多*

random.shuffle(api\_list)

**for** api **in** api\_list:

response = requests.get(api)

js\_str = json.loads(response.text)

*# 如果正确提取到了ip地址*

**if** js\_str.get('code') == '0' **or** js\_str.get('ERRORCODE') == '0':

*# 从中取出ip*

**for** i, j **in** js\_str.items():

**if** j != '0':

*# proxies = {*

*# "http": "http://{}:{}".format(j[0].get('ip'), j[0].get('port')),*

*# "https": "https://{}:{}".format(j[0].get('ip'), j[0].get('port'))*

*# }*

proxies = "http://{}:{}".format(j[0].get('ip'), j[0].get('port'))

logger.info("从 {} 获取了一个代理 {}".format(re.split(r'.c', api)[0], proxies))

*# print("从{}获取了一个代理{}".format(re.split(r'.c', api)[0], proxies))*

**return** proxies

**break**

**else**:

*# print("提取太频繁, 等待中...")*

logger.info("api {} 提取太频繁, 等待中".format(api))

time.sleep(random.randint(5, 10))

**def** get\_random\_ua():

*# 从本地读取useragent并随机选择*

project\_path = os.path.dirname(os.path.dirname(os.path.abspath(\_\_file\_\_)))

json\_file = os.path.join(project\_path, "fake\_useragent.json")

**with** open(json\_file, "r") **as** f:

ua\_list = json.load(f)

user\_agent = random.choice(ua\_list)

*# print("当前的user-agent是:%s" % user\_agent)*

logger.info("随机获取了一个ua {}".format(user\_agent))

**return** user\_agent

**from** **twisted.internet.defer** **import** DeferredLock

**class** **RandomUAIPDownloaderMiddleware**(object):

**def** \_\_init\_\_(self, ua=''):

*# 初始时从api获取代理地址, 并给所有代理都设置为这个代理*

super(RandomUAIPDownloaderMiddleware, self).\_\_init\_\_()

self.user\_agent = get\_random\_ua()

self.proxy = get\_random\_ip()

self.exception\_list = (defer.TimeoutError, TimeoutError, ConnectionRefusedError, ConnectError, ConnectionLost, TCPTimedOutError, ConnectionDone)

*# 设置一个过期的代理集合*

self.blacked\_proxies = set()

self.lock = DeferredLock()

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

*# 把更新代理的操作都放在process\_request中进行. 这样, 不论是第一次的请求, 还是*

*# 判断request中使用的代理, 如果它不等于当前的代理, 就把它设置为当前的代理*

**if** request.meta.get('proxy') != self.proxy **and** self.proxy **not** **in** self.blacked\_proxies:

request.headers.setdefault('User-Agent', self.user\_agent)

request.meta["proxy"] = self.proxy

**pass**

**def** process\_response(self, request, response, spider):

*# 如果返回的response状态不是200，这里不再重新返回request对象, 因为很可能是因为无法请求对应的资源.*

*# 如http://images2015.cnblogs.com/blog/992994/201703/992994-20170302204433063-1243104447.png 这个图片无法下载, 如果返回request, 所有的线程都会去请求这个图片, 所以这里只记录错误即可.*

**if** response.status != 200:

logger.error("{} 响应出错, 状态码为 {}".format(request.url, response.status))

*# return request*

**return** response

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

*# 如果出现了上面列表中的异常, 就认为代理失效了. 由于scrapy使用的是异步框架, 所以代理失效时会有很多个请求同时出现上面列表中的异常, 同时进入到这里的代码中执行. 如果按照一般的思路, 把更新代理的操作放在这里, 那么所有异常的请求进入此代码后都要更新代理, 都要向api发送请求获取代理地址, 此时就会出现代理请求太频繁的提示.*

*# 这里使用的方法是, 只要出现了认为是代理失效的异常, 就把请求的proxy和user-agent设置为None, 同时设置另一个条件判断产生异常的代理是否等于self.proxy, 当异常发生时, 必定会有先后的顺序, 第1个异常的请求进入这里时, 满足此条件, 执行下面的代码, 更新self.user\_agent和self.proxy. 当以后发生异常的请求再次进入到这里的逻辑时, 因为第1个请求已经更新了self.proxy的值, 就不能满足第2个if判断中的条件, 就不会执行更新代理的操作了, 这样就避免了所有发生异常的请求同时请求api更新代理的情况.*

**if** isinstance(exception, self.exception\_list):

logger.info("Proxy {} 链接出错 {}".format(request.meta['proxy'], exception))

self.lock.acquire()

*# 如果失效的代理不在代理黑名单中, 表示这是这个代理地址第一次失效, 就执行更新代理的操作.*

**if** request.meta.get('proxy') **not** **in** self.blacked\_proxies:

*# 如果代理过期, 就把它添加到代理黑名单列表中*

self.blacked\_proxies.add(self.proxy)

**print**('**\n\n**')

**print**(self.blacked\_proxies)

**print**('**\n\n**')

self.user\_agent = get\_random\_ua()

self.proxy = get\_random\_ip()

self.lock.release()

request.meta["proxy"] = None

request.headers.setdefault('User-Agent', None)

**return** request.replace(dont\_filter=True)

### 在settings.py中设置启动middleware

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

*# Scrapy settings for jobbole\_article project*

*#*

*# For simplicity, this file contains only settings considered important or*

*# commonly used. You can find more settings consulting the documentation:*

*#*

*# https://doc.scrapy.org/en/latest/topics/settings.html*

*# https://doc.scrapy.org/en/latest/topics/downloader-middleware.html*

*# https://doc.scrapy.org/en/latest/topics/spider-middleware.html*

BOT\_NAME = 'jobbole\_article'

SPIDER\_MODULES = ['jobbole\_article.spiders']

NEWSPIDER\_MODULE = 'jobbole\_article.spiders'

*# 设置下载超时时间*

DOWNLOAD\_TIMEOUT = 10

*# 禁止请求失败时重试*

RETRY\_ENABLED = True

*# RETRY\_ENABLED = False*

*# 设置重试次数*

RETRY\_TIMES = 1

*###################################################*

*# 日志设置*

*# LOG\_LEVEL = 'INFO'*

*###################################################*

*# Crawl responsibly by identifying yourself (and your website) on the user-agent*

*# USER\_AGENT = 'jobbole\_article (+http://www.yourdomain.com)'*

*# Obey robots.txt rules*

ROBOTSTXT\_OBEY = False

*# Disable cookies (enabled by default)*

COOKIES\_ENABLED = False

*# Disable Telnet Console (enabled by default)*

*# TELNETCONSOLE\_ENABLED = False*

*# Override the default request headers:*

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; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/65.0.3325.181 Safari/537.36'

}

*# Enable or disable spider middlewares*

*# See https://doc.scrapy.org/en/latest/topics/spider-middleware.html*

*# SPIDER\_MIDDLEWARES = {*

*# 'jobbole\_article.middlewares.JobboleArticleSpiderMiddleware': 543,*

*# }*

*# Enable or disable downloader middlewares*

*# See https://doc.scrapy.org/en/latest/topics/downloader-middleware.html*

DOWNLOADER\_MIDDLEWARES = {

*# 'jobbole\_article.middlewares.JobboleArticleDownloaderMiddleware': 543,*

*# 'jobbole\_article.middlewares.RandomFakeUserAgentMiddleware': 100,*

*# 'jobbole\_article.middlewares.RandomLocalUserAgentMiddleware': 100,*

*# 'jobbole\_article.middlewares.UADMiddleware': 100,*

'jobbole\_article.middlewares.RandomUAIPDownloaderMiddleware': 100,

}

*# Configure item pipelines*

*# See https://doc.scrapy.org/en/latest/topics/item-pipeline.html*

ITEM\_PIPELINES = {

*# 'jobbole\_article.pipelines.JsonWithEncodingPipeline': 300,*

*# 'jobbole\_article.pipelines.JsonExporterPipeline': 301,*

*# 'scrapy.pipelines.images.ImagesPipeline' : 200,*

'jobbole\_article.pipelines.JobboleImagePipeline': 200,

*# 'jobbole\_article.pipelines.MysqlPipeline': 302,*

'jobbole\_article.pipelines.MysqlTwistedPipline': 302,

}

*################################################################*

*# 图片下载设置*

*# 设置图片url的字段, scrapy将从item中找出此字段进行图片下载*

IMAGES\_URLS\_FIELD = "front\_image\_url"

*# 设置图片下载保存的目录, 这里不直接使用本机的绝对路径, 而是通过程序获取路径. 这样在程序迁移后也能够正常运行.*

**import** **os**

*# os.path.dirname(\_\_file\_\_)获取当前文件所在的文件夹名称.*

*# os.path.abspath(os.path.dirname(\_\_file\_\_))获得当前文件所在的绝对路径.*

project\_path = os.path.dirname(os.path.abspath(\_\_file\_\_))

*# 想要把下载的图片保存在与settings同目录的images文件夹中, 要先在项目中新建此文件夹.*

*# IMAGES\_STORE中定义保存图片的路径*

IMAGES\_STORE = os.path.join(project\_path, "images")

*# 表示只下载大于100x100的图片, 查看images.py的源码, 程序会自动的从settings.py中读取设置的IMAGES\_MIN\_HEIGHT和IMAGES\_MIN\_WIDTH值*

*# IMAGES\_MIN\_HEIGHT = 100*

*# IMAGES\_MIN\_WIDTH = 100*

**if** **not** os.path.exists(IMAGES\_STORE):

os.mkdir(IMAGES\_STORE)

**else**:

**pass**

*################################################################*

*################################################################*

*# mysql配置*

MYSQL\_HOST = 'sh-cdb-pzoa3tqh.sql.tencentcdb.com'

MYSQL\_HOST = '127.0.0.1'

*# 这里设置的是数据库的名字, 不是数据表的名字*

MYSQL\_DBNAME = 'jobbole\_article'

MYSQL\_USERNAME = 'root'

MYSQL\_PASSWORD = 'Xzq@8481'

MYSQL\_PORT = 63104

MYSQL\_PORT = 3306

*################################################################*

### 解决图片下载出错引起的频繁更换代理的问题.

运行爬虫, 发现除了第1个ip地址之外, 从获取的第2个代理地址开始, 很快就会请求新的代理. 发现这是由于图片在设定的响应时间内没有下载完成导致的, 可能是因为图片太大引起的.

在settings.py中注释掉图片下载的pipeline即可.

ITEM\_PIPELINES = {

*# 'jobbole\_article.pipelines.JsonWithEncodingPipeline': 300,*

*# 'jobbole\_article.pipelines.JsonExporterPipeline': 301,*

*# 'scrapy.pipelines.images.ImagesPipeline' : 200,*

*# 'jobbole\_article.pipelines.JobboleImagePipeline': 200,*

*# 'jobbole\_article.pipelines.MysqlPipeline': 302,*

'jobbole\_article.pipelines.MysqlTwistedPipline': 302,

}

由于不下载图片了, 也无法获取 front\_image\_path了, 所以要在pipelines.py中修改为 item.get("front\_image\_path", "")

self.cursor.execute(insert\_sql, (item.get("title"), item.get("create\_date"), item.get("article\_url"), item.get("url\_object\_id"), item.get("front\_image\_url")[0], item['front\_image\_path'], item.get("comment\_num"), item.get("praise\_num", ""), item.get("fav\_num"), item.get("tags")), item.get("content"))

### 解决mysql 4字节utf-8字符的问题

2018-08-31 23:45:18 [py.warnings] WARNING: C:\Users\David\jobbole\_article\jobbole\_article\pipelines.py:179: Warning: (1300, "Invalid utf8 character string: 'F09F8F'")

item["content"]))

2018-08-31 23:45:18 [py.warnings] WARNING: C:\Users\David\jobbole\_article\jobbole\_article\pipelines.py:179: Warning: (1366, "Incorrect string value: '\\xF0\\x9F\\x8F\\xA0 \\xE7...' for column 'content' at row 1")

item["content"]))

# 修改mysql数据库的编码为uft8mb4

# 可以不执行此句

# ALTER SCHEMA ak47\_cms DEFAULT CHARACTER SET utf8mb4 DEFAULT COLLATE utf8mb4\_general\_ci ;

# 修改数据库:

ALTER DATABASE database\_name CHARACTER SET = utf8mb4 COLLATE = utf8mb4\_unicode\_ci;

alter database jobbole\_article character set = utf8mb4;

# 修改表:

ALTER TABLE table\_name CONVERT TO CHARACTER SET utf8mb4 COLLATE utf8mb4\_unicode\_ci;

alter table article character set = utf8mb4;

# 修改表字段:

ALTER TABLE table\_name CHANGE column\_name column\_name VARCHAR(191) CHARACTER SET utf8mb4 COLLATE utf8mb4\_unicode\_ci;

alter table article modify content longtext character set utf8mb4 not null;

# 修改连接数据库的连接代码

**class** **MysqlTwistedPipline**(object):

*# from\_settings和\_\_init\_\_这两个方法就能实现在启动spider时, 就把dbpool传递进来*

**def** \_\_init\_\_(self, dbpool):

self.dbpool = dbpool

*# 在初始化时scrapy会调用from\_settings方法, 将setting文件中的配置读入, 成为一个settings对象, 这种写法是固定的, 其中的参数不能修改.*

@classmethod

**def** from\_settings(cls, settings):

dbparas = dict(

host=settings["MYSQL\_HOST"],

*# 可以在settings中设置此pipeline, 在此处放置断点, 进行debug, 查看能否导入. 在attribute中可以看到settings中定义的所有的值. F6执行, 就能看到取到了settings中设置的host的值了.*

port=settings["MYSQL\_PORT"],

db=settings["MYSQL\_DBNAME"],

user=settings["MYSQL\_USERNAME"],

passwd=settings["MYSQL\_PASSWORD"],

*# charset="utf8",*

charset="utf8mb4",

*# pymysql模块中也有类似的模块pymysql.cursors.DictCursor*

cursorclass=MySQLdb.cursors.DictCursor,

use\_unicode=True

)

参考

<https://zhuanlan.zhihu.com/p/29287995>

<https://blog.csdn.net/poice00/article/details/52129351>

<https://blog.csdn.net/hzw19920329/article/details/55670782>

<https://my.oschina.net/xsh1208/blog/1052781>

## 把项目部署到 ubuntu server中

### 环境配置

#### 阿里云ubuntu基本配置

1. 阿里云在实例列表中停止实例.

2. 操作 > 更多 > 磁盘和镜像 > 更换系统盘 > 自定义登陆root账户的密码 > 等待系统重装完成

3. 远程连接, 输入6位远程连接密码 > 输入root登录 > 输入root密码.

4. 设置允许使用密码通过ssh远程登录

vim /etc/ssh/sshd\_config

PasswordAuthentication yes

5. 使用ssh远程登录软件登录

6. 更新系统软件

apt update

apt upgrade -y

7. reboot

#### 安装mysql-server并配置

16. 安装mysql-server

使用MySQL APT Repositoryubuntu安装 mysql

<https://dev.mysql.com/doc/mysql-apt-repo-quick-guide/en/#apt-repo-fresh-install>

16.1. 下载deb包

https://dev.mysql.com/downloads/repo/apt/

wget <https://repo.mysql.com//mysql-apt-config_0.8.10-1_all.deb>

16.2. 安装deb包

sudo dpkg -i mysql-apt-config\_0.8.10-1\_all.deb

选择要安装的mysql的版本

3. 更新系统

sudo apt update

sudo apt upgrade

4. 安装mysql server

apt install mysql-server -y

5. Starting and Stopping the MySQL Server

The MySQL server is started automatically after installation. You can check the status of the MySQL server with the following command:

sudo service mysql status

Stop the MySQL server with the following command:

sudo service mysql stop

To restart the MySQL server, use the following command:

sudo service mysql start

6. Installing Additional MySQL Products and Components

You can use APT to install individual components of MySQL from the MySQL APT repository. Assuming you already have the MySQL APT repository on your system's repository list (see Adding the MySQL APT Repository for instructions), first, use the following command to get the latest package information from the MySQL APT repository:

sudo apt-get update

Install any packages of your choice with the following command, replacing package-name with name of the package (here is a list of available packages):

For example, to install the MySQL Workbench:

sudo apt-get install mysql-workbench-community

To install the shared client libraries:

sudo apt-get install libmysqlclient18

apt search libmysqlclient

libcrypt-mysql-perl/xenial 0.04-6build1 amd64

Perl module to emulate the MySQL PASSWORD() function

libglpk36/xenial 4.57-1build3 amd64

linear programming kit with integer (MIP) support

libmysqlclient-dev/unknown 5.7.23-1ubuntu16.04 amd64

MySQL development headers

libmysqlclient20/unknown 5.7.23-1ubuntu16.04 amd64

MySQL shared client libraries

#### 安装redis并配置

安装 redis

wget http://download.redis.io/redis-stable.tar.gz

# tar xvzf redis-stable.tar.gz

tar xf redis-stable.tar.gz

cd redis-stable

make

make install

make之后可以使用make test进行测试. 然后再make install

apt install tcl

make test

启动 redis

redis-server

查看redis是否启动, 重新打开一个终端执行redis-cli连接到redis-server

$ redis-cli

redis 127.0.0.1:6379> ping

PONG

redis 127.0.0.1:6379> set mykey somevalue

OK

redis 127.0.0.1:6379> get mykey

"somevalue"

使用redis\_init\_script脚本启动redis

* Create a directory where to store your Redis config files and your data:

sudo mkdir /etc/redis

sudo mkdir /var/redis

* Copy the init script that you'll find in the Redis distribution under the **utils** directory into /etc/init.d. # 将redis安装目录的utils下的redis\_init\_script拷贝到/etc/init.d目录下, 放在/etc/init.d下，主要是将redis作为一个系统的daemon进程去运行的，每次系统启动，redis进程一起启动

We suggest calling it with the name of the port where you are running this instance of Redis. For example:

17839

sudo cp ./redis-stable/utils/redis\_init\_script /etc/init.d/redis\_6379

sudo cp ./redis-stable/utils/redis\_init\_script /etc/init.d/redis\_17839

* (optional) 赋予脚本执行权限

chmod a+x /etc/init.d/redis\_6379

chmod a+x /etc/init.d/redis\_17839

* Edit the init script.

sudo vim /etc/init.d/redis\_6379

sudo vim /etc/init.d/redis\_17839

Make sure to modify **REDISPORT** accordingly to the port you are using. Both the pid file path and the configuration file name depend on the port number.

# 根据需要修改端口号

REDISPORT=6379

# REDISPORT=17839

# 根据需要修改安装的redis目录

EXEC=/usr/local/bin/redis-server

CLIEXEC=/usr/local/bin/redis-cli

PIDFILE=/var/run/redis\_${REDISPORT}.pid

# 可以修改配置文件目录，也可以按照这个目录在linux上创建

CONF="/etc/redis/${REDISPORT}.conf"

* Copy the template configuration file you'll find in the root directory of the Redis distribution into /etc/redis/ using the port number as name, for instance:

sudo cp redis.conf /etc/redis/6379.conf

sudo cp ./redis-stable/redis.conf /etc/redis/17839.conf

* Create a directory inside /var/redis that will work as data and working directory for this Redis instance:

sudo mkdir /var/redis/6379

sudo mkdir /var/redis

sudo mkdir /var/redis/17839

* Edit the configuration file, making sure to perform the following changes:

sudo vim /etc/redis/6379.conf

sudo vim /etc/redis/17839.conf

1. 注释掉 bind 127.0.0.1 允许远程登录 redis
2. Change the port accordingly. In our example it is not needed as the default port is already 6379. 设置redis监听的端口号
3. Set daemonize to yes (by default it is set to no). 让redis以daemon进程启动
4. Set the pidfile to /var/run/redis\_6379.pid (modify the port if needed). 设置redis的pid文件.
5. Set your preferred loglevel. 设置日志等级
6. Set the logfile to /var/log/redis\_6379.log
7. Set the dir to /var/redis/6379 (very important step!) 设置持久化文件的存储位置（该目录需要有足够的磁盘空间）
8. requirepass 设置登录密码
9. Finally add the new Redis init script to all the default runlevels using the following command:

sudo update-rc.d redis\_6379 defaults

sudo update-rc.d redis\_17839 defaults

# 出错信息, 但不影响正常使用

root@ubuntu:~# sudo update-rc.d redis\_6379 defaults

insserv: Script redis\_6379 is broken: incomplete LSB comment.

insserv: missing `Required-Start:' entry: please add even if empty.

insserv: missing `Required-Stop:' entry: please add even if empty.

You are done! Now you can try running your instance with: 启动redis

sudo /etc/init.d/redis\_6379 start

sudo /etc/init.d/redis\_17839 start

验证redis是否启动成功

ps -ef | grep redis

ps -aux | grep redis

没有设置登录密码的redis的关闭

/etc/init.d/redis\_6379 stop

登录带有密码验证的redis-server

redis-cli -p 17839 -a Xzz@8481

或者

redis-cli -p 17839

auth password

# 设置有密码验证的redis的关闭

方法一:

redis-cli -p 17839 -a password shutdown

方法二:

redis-cli -p 17839

auth password

shut down

quit

Make sure that everything is working as expected:

1. Try pinging your instance with redis-cli.
2. Do a test save with redis-cli save and check that the dump file is correctly stored into /var/redis/6379/ (you should find a file called dump.rdb).
3. Check that your Redis instance is correctly logging in the log file.
4. If it's a new machine where you can try it without problems make sure that after a reboot everything is still working.

Note: In the above instructions we skipped many Redis configuration parameters that you would like to change, for instance in order to use AOF persistence instead of RDB persistence, or to setup replication, and so forth. Make sure to read the example redis.conf file (that is heavily commented) and the other documentation you can find in this web site for more information.

#### 安装python3.6并创建虚拟环境

8. 添加第三方软件源

sudo apt-get install software-properties-common -y

sudo add-apt-repository ppa:jonathonf/python-3.6 -y

sudo apt-get update -y

9. 安装python3.6

sudo apt-get install python3.6 -y

检查已安装的 Python 3.6 版本

python3.6 -V

# Python 3.6.5

10. 安装pip, 由于ubuntu 16.04 默认使用的是python3.5, 所以安装的是python3.5对应的pip

sudo apt install python3-pip -y

# 查看安装的pip3的版本

pip3 -V

# pip 8.1.1 from /usr/lib/python3/dist-packages (python 3.5)

11. 升级pip版本, 会使用pip3 替换系统默认的pip

sudo pip3 install --upgrade pip

pip3 -V

# pip 18.0 from /usr/local/lib/python3.5/dist-packages/pip (python 3.5)

pip -V #(出错)

reboot # 重启后pip3 -V和pip -V都显示为pip 18.0

12. 使用pip 国内镜像

# 如果目录不存在, 要先创建目录, 否则无法保存文件.

mkdir ~/.pip

vim ~/.pip/pip.conf

[global]

index-url = http://pypi.douban.com/simple

[install]

trusted-host=pypi.douban.com

# 阿里云ECS服务器默认配置

## Note, this file is written by cloud-init on first boot of an instance

## modifications made here will not survive a re-bundle.

###

[global]

index-url=http://mirrors.cloud.aliyuncs.com/pypi/simple/

[install]

trusted-host=mirrors.cloud.aliyuncs.com

[global]

index-url = https://mirrors.aliyun.com/pypi/simple/

[install]

trusted-host=mirrors.aliyun.com

12. 安装虚拟环境, 如果pip和pip3的版本不同, 要执行pip3 install virtualenv virtualenvwrapper

pip install virtualenv

pip install virtualenvwrapper

13. 配置虚拟环境

# 查找python3和virtualenvwrapper.sh的路径

which python3

which virtualenvwrapper.sh

# 配置virtualwrapper环境变量

sudo vim ~/.bashrc

在最后一行添加如下内容

# WORKON\_HOME设置虚拟环境保存的目录, $HOME为用户的主目录, 即设置虚拟环境保存在用户主目录下的.virtualenvs目录下.

export WORKON\_HOME=$HOME/.virtualenvs # 虚拟环境创建的地方

export VIRTUALENVWRAPPER\_PYTHON=/usr/bin/python3 # 指定虚拟使用的python解释器路径

source /usr/local/bin/virtualenvwrapper.sh # 每次登陆用户自动执行下脚本

14. 使用python3.6新建虚拟环境

which python3.6

/usr/bin/python3.6

# 指定使用python3.6创建虚拟环境

mkvirtualenv --python=/usr/bin/python3.6 python3\_spider

15. 安装python开发包

一键安装

sudo apt install python3.6-dev libmysqlclient-dev

pip install ipython pdir2 retrying requests lxml beautifulsoup4 pillow fake-useragent mysqlclient pymysql pymongo redis selenium openvr scrapy scrapy-redis scrapyd

分立功能

pip install ipython pdir2

pip install retrying

pip install requests lxml

pip install beautifulsoup4

pip install pillow fake-useragent

sudo apt install python3.6-dev libmysqlclient-dev

pip install mysqlclient

pip install pymysql pymongo redis

pip install selenium openvr

pip install scrapy scrapy-redis scrapyd

mysqlclient 如果安装失败

ubuntu安装

sudo apt install python3.6-dev libmysqlclient-dev

centos安装

sudo yum install python-devel mysql-devel

### 把项目部署到ubuntu server中.

进入到开发环境中, 进入到项目根目录中, 保存开发环境中的所有安装包

pip freeze > requirements.txt

如果是在linux中部署, 需要删除掉其中的pywin32 和 pypiwin32

把项目打包, 上传到ubuntu server中.

使用cmder 来输入命令

# 使用tar命令来打包

tar -cvf jobbole\_article.tar jobbole\_article

tar -zcvf jobbole\_article.gz jobbole\_article

# 使用tar命令来解包

tar -xvf jobbole\_article.tar

tar -zcvf jobbole\_article.gz

# linux中使用zip来压缩和解压

zip jobbole\_article jobbole\_article.zip

unzip jobbole\_article.zip

# cmder中使用scp命令把文件或文件夹上传到远程服务器中. 注意scp复制的是文件夹下的所有文件, 并不复制本文件夹

# Yunzhuji

scp -r jobbole\_article root@219.235.1.146:/root/test

scp jobbole\_article.zip root@219.235.1.146:/root/test

# 直接复制文件夹

scp -r jobbole\_article root@219.235.1.146:/root/jobbole\_article

密码

0v0oSZ,Ns5

# Aliyun

scp -r jobbole\_article root@47.100.193.219:/root/jobbole\_article

# 在ubuntu server中进入到虚拟环境中

workon python3\_spider

安装需要的包.

pip install -r requirements.txt

pip install ipython beautifulsoup4 chardet fake-useragent lxml mysqlclient PyMySQL

openvr pdir2 pillow pymongo redis requests retrying scrapy scrapy-redis scrapyd selenium

#### pip install mysqlclient: mysql\_config not found错误

<https://blog.csdn.net/ping523/article/details/54289398>

(python3\_spider) root@ubuntu:~# pip install mysqlclient

Looking in indexes: http://pypi.douban.com/simple

Collecting mysqlclient

Downloading http://pypi.doubanio.com/packages/ec/fd/83329b9d3e14f7344d1cb31f1275c9e57896815dbb1988ad/mysqlclient-1.3.13.tar.gz (90kB)

100% |████████████████████████████████| 92kB 2.5MB/s

Complete output from command python setup.py egg\_info:

/bin/sh: 1: mysql\_config: not found

Traceback (most recent call last):

File "<string>", line 1, in <module>

File "/tmp/pip-install-1nj63j0t/mysqlclient/setup.py", line 18, in <module

metadata, options = get\_config()

File "/tmp/pip-install-1nj63j0t/mysqlclient/setup\_posix.py", line 53, in g

libs = mysql\_config("libs\_r")

File "/tmp/pip-install-1nj63j0t/mysqlclient/setup\_posix.py", line 28, in m

raise EnvironmentError("%s not found" % (mysql\_config.path,))

OSError: mysql\_config not found

----------------------------------------

Command "python setup.py egg\_info" failed with error code 1 in /tmp/pip-install-qlclient/

sudo apt-get install libmysqlclient-dev

然后进入mysql\_config的路径（/usr/bin/mysql\_config）

sudo updatedb

locate mysql\_config

#### 错误2: Could not run curl-config: [Errno 2] No such file or directory: 'curl-config': 'curl-config'

Collecting pycurl==7.43.0.2 (from -r requirements.txt (line 47))

Downloading http://pypi.doubanio.com/packages/e8/e4/0dbb8735407189f00b33d84122b9be52c790c7c3b25286826f4e1bdb7bde/pycurl-7.43.0.2.tar.gz (214kB)

100% |████████████████████████████████| 215kB 1.3MB/s

Complete output from command python setup.py egg\_info:

Traceback (most recent call last):

File "/tmp/pip-install-urbu06t2/pycurl/setup.py", line 223, in configure\_unix

stdout=subprocess.PIPE, stderr=subprocess.PIPE)

File "/usr/lib/python3.6/subprocess.py", line 709, in \_\_init\_\_

restore\_signals, start\_new\_session)

File "/usr/lib/python3.6/subprocess.py", line 1344, in \_execute\_child

raise child\_exception\_type(errno\_num, err\_msg, err\_filename)

FileNotFoundError: [Errno 2] No such file or directory: 'curl-config': 'curl-config'

During handling of the above exception, another exception occurred:

Traceback (most recent call last):

File "<string>", line 1, in <module>

File "/tmp/pip-install-urbu06t2/pycurl/setup.py", line 913, in <module>

ext = get\_extension(sys.argv, split\_extension\_source=split\_extension\_source)

File "/tmp/pip-install-urbu06t2/pycurl/setup.py", line 582, in get\_extension

ext\_config = ExtensionConfiguration(argv)

File "/tmp/pip-install-urbu06t2/pycurl/setup.py", line 99, in \_\_init\_\_

self.configure()

File "/tmp/pip-install-urbu06t2/pycurl/setup.py", line 227, in configure\_unix

raise ConfigurationError(msg)

\_\_main\_\_.ConfigurationError: Could not run curl-config: [Errno 2] No such file or directory: 'curl-config': 'curl-config'

----------------------------------------

Command "python setup.py egg\_info" failed with error code 1 in /tmp/pip-install-urbu06t2/pycurl/

(python3\_spider) root@ubuntu:~# ^C

(python3\_spider) root@ubuntu:~# apt-get install libcurl4-gnutls-dev

在安装 pyspider 的时候我就遇到了这个问题， pyspider 依赖 pycurl 这个库，而 pycurl 要求系统中存在相对应的库。

经过我的测试， curl 是正确安装了的，但是却依然提示了找不到 curl-config 的问题

解决方案：

apt-get install libcurl4-gnutls-dev

#### Failed building wheel for mysqlclient

building '\_mysql' extension

creating build/temp.linux-x86\_64-3.6

x86\_64-linux-gnu-gcc -pthread -DNDEBUG -g -fwrapv -O2 -Wall -Wstrict-prototypes -g -fstack-protector-strong -Wformat -Werror=format-security -Wdate-time -D\_FORTIFY\_SOURCE=2 -fPIC -Dversion\_info=(1,3,13,'final',0) -D\_\_version\_\_=1.3.13 -I/usr/include/mysql -I/usr/include/python3.6m -I/root/.virtualenvs/python3\_spider/include/python3.6m -c \_mysql.c -o build/temp.linux-x86\_64-3.6/\_mysql.o

\_mysql.c:37:20: fatal error: Python.h: No such file or directory

compilation terminated.

error: command 'x86\_64-linux-gnu-gcc' failed with exit status 1

----------------------------------------

Failed building wheel for mysqlclient

Running setup.py clean for mysqlclient

Failed to build mysqlclient

Installing collected packages: mysqlclient

Running setup.py install for mysqlclient ... error

sudo apt-get install python3.6-dev

yum install python36-devel

#### 错误3: Failed to build pycurl

In file included from src/docstrings.c:4:0:

src/pycurl.h:168:30: fatal error: gnutls/gnutls.h: No such file or directory

compilation terminated.

error: command 'x86\_64-linux-gnu-gcc' failed with exit status 1

缺少libgnutls28-dev依赖。

sudo apt-get install libgnutls28-dev

再次pip3 install pycurl

#### mysql\_config not found

(python3\_spider) root@aliyun:~# pip install mysqlclient

Looking in indexes: http://mirrors.cloud.aliyuncs.com/pypi/simple/

Collecting mysqlclient

Downloading http://mirrors.cloud.aliyuncs.com/pypi/packages/ec/fd/83329b9d3e14f7344d1cb31f128e6dbba70c5975c9e57896815dbb1988ad/mysqlclient-1.3.13.tar.gz (90kB)

100% |████████████████████████████████| 92kB 13.2MB/s

Complete output from command python setup.py egg\_info:

/bin/sh: 1: mysql\_config: not found

Traceback (most recent call last):

File "<string>", line 1, in <module>

File "/tmp/pip-install-9oxnh6cu/mysqlclient/setup.py", line 18, in <module>

metadata, options = get\_config()

File "/tmp/pip-install-9oxnh6cu/mysqlclient/setup\_posix.py", line 53, in get\_config

libs = mysql\_config("libs\_r")

File "/tmp/pip-install-9oxnh6cu/mysqlclient/setup\_posix.py", line 28, in mysql\_config

raise EnvironmentError("%s not found" % (mysql\_config.path,))

OSError: mysql\_config not found

----------------------------------------

Command "python setup.py egg\_info" failed with error code 1 in /tmp/pip-install-9oxnh6cu/mysqlclient/

### 运行爬虫

cd jobbole\_article

## 修改为分布式爬虫

cd jobbole\_article

scrapy genspider jobbole\_dis

把jobbole\_it中的所有内容全部复制到jobbole\_dis中. 修改爬虫名称, 继承自RedisSpider, 并且在爬虫中使用自定义设置.

jobbole\_dis.py

redis\_host和redis\_port可以写到一起, 使用redis\_url来代替.

'REDIS\_HOST': '219.235.1.146',

'REDIS\_PORT': 6379,

REDIS\_URL = 'redis://127.0.0.1:6379/11'

对每个爬虫使用自定义的设置.

<https://zhuanlan.zhihu.com/p/34035463>

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

**import** **scrapy**

**from** **scrapy.loader** **import** ItemLoader

**from** **jobbole\_article.items** **import** JobboleArticleProcessItem

**from** **jobbole\_article.items** **import** JobboleArticleItemLoader

**from** **scrapy\_redis.spiders** **import** RedisSpider

**class** **JobboleItSpider**(RedisSpider):

name = 'jobbole\_dis'

allowed\_domains = ["blog.jobbole.com"]

*# start\_urls = ['http://blog.jobbole.com/all-posts/']*

redis\_key = 'jobbole:start\_urls'

*# 自定义设置*

custom\_settings = {

'LOG\_LEVEL': 'DEBUG',

'DOWNLOAD\_DELAY': 0,

*# 修改为分布式的爬虫*

*# 使用redis的调度器, 确保request存储到redis中*

"SCHEDULER": "scrapy\_redis.scheduler.Scheduler",

*# 使用scrapy\_redis进行去重, 确保所有爬虫共享相同的去重指纹*

"DUPEFILTER\_CLASS": "scrapy\_redis.dupefilter.RFPDupeFilter",

*# 在redis中保持scrapy-redis用到的队列, 不会清理redis中的队列, 从而可以实现暂停和恢复的功能.*

"SCHEDULER\_PERSIST": True,

*# 指定redis数据库的连接参数*

'REDIS\_HOST': '219.235.1.146',

'REDIS\_PORT': 6379,

*# 指定 redis链接密码，和使用哪一个数据库*

'REDIS\_PARAMS' : {

'password': 'Xzz@8481',

'db': 2

},

*# 启动redis的pipeline, 把提取到的数据都传输并保存到redis服务器中.*

"ITEM\_PIPELINES": {

*# 'jobbole\_article.pipelines.JsonWithEncodingPipeline': 300,*

*# 'jobbole\_article.pipelines.JsonExporterPipeline': 301,*

*# 'scrapy.pipelines.images.ImagesPipeline' : 200,*

*# 'jobbole\_article.pipelines.JobboleImagePipeline': 200,*

*# 'jobbole\_article.pipelines.MysqlPipeline': 302,*

'jobbole\_article.pipelines.MysqlTwistedPipline': 302,

'scrapy\_redis.pipelines.RedisPipeline': 300

}

}

**def** parse(self, response):

*# 从文章列表页获取详情页url*

在芝麻代理官网中添加ip白名单, 使所有的节点都能使用芝麻代理.

运行爬虫

在每个爬虫节点中运行爬虫

cd /root/jobbole\_article/jobbole\_article/spiders

scrapy runspider jobbole\_dis.py

在运行redis-server的服务器中push入start\_urls

lpush jobbole:start\_urls <http://blog.jobbole.com/all-posts/>

## 使用docker部署爬虫

### 安装Docker

#### Get Docker CE for Ubuntu

##### Uninstall old versions

Older versions of Docker were called docker or docker-engine. If these are installed, uninstall them:

sudo apt-get remove docker docker-engine docker.io

It’s OK if apt-get reports that none of these packages are installed.

The contents of /var/lib/docker/, including images, containers, volumes, and networks, are preserved. The Docker CE package is now called docker-ce.

##### Install using the repository

Before you install Docker CE for the first time on a new host machine, you need to set up the Docker repository. Afterward, you can install and update Docker from the repository.

###### SET UP THE REPOSITORY

Update the apt package index:

sudo apt-get update

Install packages to allow apt to use a repository over HTTPS:

sudo apt-get install \

apt-transport-https \

ca-certificates \

curl \

software-properties-common

首先安装依赖：

sudo apt-get install apt-transport-https ca-certificates curl gnupg2 software-properties-common

Add Docker’s official GPG key:

信任Docker的GPG公钥：

curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -

Verify that you now have the key with the fingerprint 9DC8 5822 9FC7 DD38 854A E2D8 8D81 803C 0EBF CD88, by searching for the last 8 characters of the fingerprint.

sudo apt-key fingerprint 0EBFCD88

pub 4096R/0EBFCD88 2017-02-22

Key fingerprint = 9DC8 5822 9FC7 DD38 854A E2D8 8D81 803C 0EBF CD88

uid Docker Release (CE deb) <docker@docker.com>

sub 4096R/F273FCD8 2017-02-22

Use the following command to set up the stable repository. You always need the stable repository, even if you want to install builds from the edge or test repositories as well. To add the edge or test repository, add the word edge or test (or both) after the word stable in the commands below.

对于amd64架构的计算机，添加软件仓库：

sudo add-apt-repository \

"deb [arch=amd64] https://download.docker.com/linux/ubuntu \

$(lsb\_release -cs) \

stable"

sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb\_release -cs) stable"

###### INSTALL DOCKER CE

Update the apt package index.

sudo apt-get update

Install the latest version of Docker CE, or go to the next step to install a specific version:

sudo apt-get install docker-ce

###### ubuntu 16.04 server 安装docker时出错

(Reading database ... 103698 files and directories currently installed.)

Preparing to unpack .../docker-ce\_18.06.1~ce~3-0~ubuntu\_amd64.deb ...

Unpacking docker-ce (18.06.1~ce~3-0~ubuntu) ...

Processing triggers for man-db (2.7.5-1) ...

Processing triggers for systemd (229-4ubuntu21.4) ...

Processing triggers for ureadahead (0.100.0-19) ...

Setting up docker-ce (18.06.1~ce~3-0~ubuntu) ...

insserv: Script redis\_17839 is broken: incomplete LSB comment.

insserv: missing `Required-Start:' entry: please add even if empty.

insserv: missing `Required-Stop:' entry: please add even if empty.

insserv: Script redis\_17839 is broken: incomplete LSB comment.

insserv: missing `Required-Start:' entry: please add even if empty.

insserv: missing `Required-Stop:' entry: please add even if empty.

insserv: Script redis\_17839 is broken: incomplete LSB comment.

insserv: missing `Required-Start:' entry: please add even if empty.

insserv: missing `Required-Stop:' entry: please add even if empty.

insserv: Script redis\_17839 is broken: incomplete LSB comment.

insserv: missing `Required-Start:' entry: please add even if empty.

insserv: missing `Required-Stop:' entry: please add even if empty.

insserv: Script redis\_17839 is broken: incomplete LSB comment.

insserv: missing `Required-Start:' entry: please add even if empty.

insserv: missing `Required-Stop:' entry: please add even if empty.

insserv: Script redis\_17839 is broken: incomplete LSB comment.

insserv: missing `Required-Start:' entry: please add even if empty.

insserv: missing `Required-Stop:' entry: please add even if empty.

insserv: Script redis\_17839 is broken: incomplete LSB comment.

insserv: missing `Required-Start:' entry: please add even if empty.

insserv: missing `Required-Stop:' entry: please add even if empty.

insserv: Script redis\_17839 is broken: incomplete LSB comment.

insserv: missing `Required-Start:' entry: please add even if empty.

insserv: missing `Required-Stop:' entry: please add even if empty.

root@aliyun:~# vim /etc/init.d/redis\_17839

root@aliyun:~# vim /etc/init.d/redis\_17839

root@aliyun:~# vim /etc/init.d/redis\_17839

root@aliyun:~# apt remove docker-ce

Reading package lists... Done

参考: <https://gist.github.com/lsbardel/257298>

解决方法: 修改redis启动文件, 在Begin init info中增加required-start和required-stop字段.

vim /etc/init.d/redis\_17839

### BEGIN INIT INFO

# Provides: redis\_17839

# Required-Start: $syslog

# Required-Stop: $syslog

# Should-Start: $local\_fs

# Should-Stop: $local\_fs

# Default-Start: 2 3 4 5

# Default-Stop: 0 1 6

# Short-Description: Redis data structure server

# Description: Redis data structure server. See https://redis.io

### END INIT INFO

卸载docer-ce, 再次安装

apt remove docker-ce

apt install docker-ce

##### Uninstall Docker CE

Uninstall the Docker CE package:

sudo apt-get purge docker-ce

Images, containers, volumes, or customized configuration files on your host are not automatically removed. To delete all images, containers, and volumes:

sudo rm -rf /var/lib/docker

sudo rm -rf /usr/bin/dockerd-ce

You must delete any edited configuration files manually.

#### 添加到用户组（可选项）

添加到用户组（so easy）

sudo groupadd docker

sudo usermod -aG docker david

注销系统重新进入系统，就可以直接使用docker开头了。

如果不添加到用户组会发生什么呢？

如果直接运行:

docker run hello-world

你会发现下面的错误:

Got permission denied while trying to connect to the Docker daemon socket at unix:///var/run/docker.sock: Get http://%2Fvar%2Frun%2Fdocker.sock/v1.30/containers/json: dial unix /var/run/docker.sock: connect: permission denied

这是因为：

docker守护程序绑定到Unix套接字而不是TCP端口。默认情况下，Unix套接字由用户root拥有，其他用户只能使用sudo访问它。 docker守护程序始终以root用户身份运行。 如果您不想在使用docker命令时使用sudo，请创建名为docker的Unix组，并将用户添加到该组。当docker守护进程启动时，它会使Docker组的Unix套接字的所有权读/写。

linux下安装一条命令即可

curl -sSL https://get.daocloud.io/docker | sh

这条命令在ubuntu 14.04和ubuntu 16.04都可以成功安装docker。

安装完成功后，可能会提示你这样的信息:

If you would like to use Docker as a non-root user, you should now consider

adding your user to the "docker" group with something like:

sudo usermod -aG docker vagrant

Remember that you will have to log out and back in for this to take effect!

vagrant是你的用户名，可能你的用户名跟我的不一样。

意思就是说，你可以把当前用户加入到docker组，以后要管理docker就方便多了，不然你以后有可能要使用docker命令前，要在前面加sudo。

如果没加sudo就是类似这样的提示:

Got permission denied while trying to connect to the Docker daemon socket at unix:///var/run/docker.sock: Get http://%2Fvar%2Frun%2Fdocker.sock/v1.26/containers/json: dial unix /var/run/docker.sock: connect: permission denied

不过执行了sudo usermod -aG docker vagrant之后，你再重新登录(ssh)，就可以免去加sudo。

安装成功，需要把docker这个服务启动起来：

如果是ubuntu 14.04的系统，它会自动启动，你也可以使用下面的命令来启动。

$ sudo /etc/init.d/docker start

如果是ubuntu 16.04的系统，就用下面的命令：

$ sudo systemctl status docker.service

#### 使用阿里云docker加速器

使用阿里云加速器提升获取Docker官方镜像的速度

国内建议可以使用一个加速器!

获得加速器的方法步骤:

进入网址

[https://account.aliyun.com/login/login.htm?oauth\_callback=https%3A%2F%2Fcr.console.aliyun.com%2F&lang=zh#/accelerator](https://link.zhihu.com/?target=https%3A//account.aliyun.com/login/login.htm%3Foauth_callback%3Dhttps%3A//cr.console.aliyun.com/%26lang%3Dzh%23/accelerator)

用自己的淘宝帐号登录进去，新用户跳过所有的步骤，进入到docker镜像仓库，点击下面的加速器，自动获得加速器，

https://cr.console.aliyun.com/cn-shanghai/mirrors

加速器地址

https://dlfa9xic.mirror.aliyuncs.com

##### 操作文档

###### Ubuntu, CentOS

1. 安装／升级Docker客户端

推荐安装1.10.0以上版本的Docker客户端，参考文档 docker-ce

<https://yq.aliyun.com/articles/110806?spm=5176.8351553.0.0.44e41991l15Vnz>

2. 配置镜像加速器

针对Docker客户端版本大于 1.10.0 的用户

您可以通过修改daemon配置文件/etc/docker/daemon.json来使用加速器

sudo mkdir -p /etc/docker

sudo tee /etc/docker/daemon.json <<-'EOF'

{

"registry-mirrors": ["https://dlfa9xic.mirror.aliyuncs.com"]

}

EOF

sudo systemctl daemon-reload

sudo systemctl restart docker

或者:

sudo vim /etc/docker/daemon.json

添加如下内容

{

"registry-mirrors": ["https://dlfa9xic.mirror.aliyuncs.com"]

}

###### Windows

1. 安装／升级Docker客户端

对于Windows 10以下的用户，推荐使用Docker Toolbox

Windows安装文件：http://mirrors.aliyun.com/docker-toolbox/windows/docker-toolbox/

对于Windows 10以上的用户 推荐使用Docker for Windows

Windows安装文件：http://mirrors.aliyun.com/docker-toolbox/windows/docker-for-windows/

2. 配置镜像加速器

针对安装了Docker Toolbox的用户，您可以参考以下配置步骤：

创建一台安装有Docker环境的Linux虚拟机，指定机器名称为default，同时配置Docker加速器地址。

docker-machine create --engine-registry-mirror=https://dlfa9xic.mirror.aliyuncs.com -d virtualbox default

查看机器的环境配置，并配置到本地，并通过Docker客户端访问Docker服务。

docker-machine env default

eval "$(docker-machine env default)"

docker info

针对安装了Docker for Windows的用户，您可以参考以下配置步骤：

在系统右下角托盘图标内右键菜单选择 Settings，打开配置窗口后左侧导航菜单选择 Docker Daemon。编辑窗口内的JSON串，填写下方加速器地址：

{

"registry-mirrors": ["https://dlfa9xic.mirror.aliyuncs.com"]

}

编辑完成后点击 Apply 保存按钮，等待Docker重启并应用配置的镜像加速器。

注意

Docker for Windows 和 Docker Toolbox互不兼容，如果同时安装两者的话，需要使用hyperv的参数启动。

docker-machine create --engine-registry-mirror=https://dlfa9xic.mirror.aliyuncs.com -d hyperv default

Docker for Windows 有两种运行模式，一种运行Windows相关容器，一种运行传统的Linux容器。同一时间只能选择一种模式运行。

3. 相关文档

Docker 命令参考文档

<https://docs.docker.com/engine/reference/commandline/cli/?spm=5176.8351553.0.0.44e41991l15Vnz>

Dockerfile 镜像构建参考文档

<https://docs.docker.com/engine/reference/builder/?spm=5176.8351553.0.0.44e41991l15Vnz>

### Docker的使用

#### 在aliyun ubuntu中创建容器并进行配置

#### docker基本操作

# docker的启动

systemctl start docker

# 输入docker查看是否成功

docker

# 运行hello-world镜像.

sudo docker run hello-world

# docker镜像的搜索. 镜像可以看成是母版, 是他人已经开发好的各种环境, 使用镜像就可以很方便的开发出自己的容器级虚拟机

# 搜索ubuntu相关的镜像, 并按starts排名

docker search ubuntu

# 搜索与python相关的镜像

docker search python

# 下载镜像并指定版本

docker pull ubuntu:14.04

docker pull ubuntu:16.04

# 使用国内的镜像下载

http://get.daocloud.io

http://hub.daocloud.io

docker pull daocloud.io/library/ubuntu:latest

# 查看本机的docker镜像, REPOSITORY, TAG, IMAGE ID, CREATED, SIZE.

docker images

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| REPOSITORY | TAG | IMAGE ID | CREATED | SIZE |
| ubuntu | 16.04 | 52b10959e8aa | 11 days ago | 115MB |
| hello-world | latest | 2cb0d9787c4d | 7 weeks ago | 1.85kB |

# 基于镜像创建容器, 容器实际上就是虚拟机, 把一台linux服务器虚拟成多台机器 , 在这些机器上运行爬虫, 就实现了分布式的功能

# tid即上面显示的IMAGE ID 52b10959e8aa的前4位, 创建的容器以e186开头

docker run -tid 52b1

# e186ec893ea6d4a85b629957e994253ceabff69a77a40e7ec2d1838605c19287

# 进入容器/虚拟机

docker attach e186

# 显示如下开头的提示即表示成功了, 有时候会长时间不显示, 再输入回车就显示出来了.

root@e186ec893ea6:/ #

# 退出容器. 如果使用exit退出容器, 就会退出的同时中止容器的运行. 一般使用快捷键Ctrl+P+Q来退出容器但不停止运行, 按下回车确定, 就退出到物理机.

# 查看所有容器的状态, 其中的status为Up 3 minutes, 就表示是处于运行的状态.

docker ps -a

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| CONTAINER ID | IMAGE | COMMAND | CREATED | STATUS | PORTS | NAMES |
| e186ec893ea6 | 52b1 | /bin/bash | 4 minutes ago | Up 4 minutes |  | stupefied\_jang |
| ec4788cef542 | hello-world | /hello | 11 minutes ago | Exited (0) 11 minutes ago |  | fervent\_panini |
| d3e000965856 | hello-world | /hello | 11 minutes ago | Exited (0) 11 minutes ago |  | hardcore\_montalcini |

# 容器的命名, 在创建容器时没有给容器命名, 就会默认使用随机的名称.

docker run --name scrapy00 -tid 52b1

# 再次查看所有容器的状态. 就可以在NAMES中看到它的名称了

docker ps -a

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| CONTAINER ID | IMAGE | COMMAND | CREATED | STATUS | PORTS | NAMES |
| b671f532a842 | 52b1 | /bin/bash | 7 seconds ago | Up 6 seconds |  | scrapy00 |
| e186ec893ea6 | 52b1 | /bin/bash | 7 minutes ago | Up 7 minutes |  | stupefied\_jang |
| ec4788cef542 | hello-world | /hello | 13 minutes ago | Exited (0) 13 minutes ago |  | fervent\_panini |
| d3e000965856 | hello-world | /hello | 14 minutes ago | Exited (0) 14 minutes ago |  | hardcore\_montalcini |

# 进入到容器中.

docker attach scrapy01

# 使用attach方法进入到容器中后使用exit方式退出时是关闭容器的运行并退出到物理机中.

exit

# 开始容器的运行

docker start scrapy00

# 停止容器的运行

docker stop scrapy00

docker start scrapy00

# 使用exec这种方式进入容器, 再使用exit退出时就是切换到后台运行了.

docker exec -it scrapy00 /bin/bash

# 切换到后台运行

exit

#### 安装mysql-server

##### 下载deb包

root@b671f532a842:/# wget https://repo.mysql.com//mysql-apt-config\_0.8.10-1\_all.deb

bash: wget: command not found

root@b671f532a842:/# apt install wget

apt install wget -y

wget <https://repo.mysql.com//mysql-apt-config_0.8.10-1_all.deb>

##### 安装deb包

root@6119cae9dac1:/# dpkg -i mysql-apt-config\_0.8.10-1\_all.deb

Selecting previously unselected package mysql-apt-config.

dpkg: regarding mysql-apt-config\_0.8.10-1\_all.deb containing mysql-apt-config, pre-dependency problem:

mysql-apt-config pre-depends on lsb-release

lsb-release is not installed.

dpkg: error processing archive mysql-apt-config\_0.8.10-1\_all.deb (--install):

pre-dependency problem - not installing mysql-apt-config

Errors were encountered while processing:

mysql-apt-config\_0.8.10-1\_all.deb

dpkg: error processing archive mysql-apt-config\_0.8.10-1\_all.deb (--install):

pre-dependency problem - not installing mysql-apt-config

Errors were encountered while processing:

mysql-apt-config\_0.8.10-1\_all.deb

apt-get install lsb-release -y

dpkg -i mysql-apt-config\_0.8.10-1\_all.deb

##### 选择要安装的mysql版本

##### 更新系统

apt update

apt upgrade

##### 安装mysql server

apt install mysql-server -y

##### 报错invoke-rc.d

update-alternatives: using /etc/mysql/mysql.cnf to provide /etc/mysql/my.cnf (my.cnf) in auto mode

invoke-rc.d: could not determine current runlevel

invoke-rc.d: policy-rc.d denied execution of start.

错误处理1:

invoke-rc.d: could not determine current runlevel

<https://zhangxinde.com/2018/06/04/66.html>

错误处理2:

invoke-rc.d: policy-rc.d denied execution of start.

<https://blog.csdn.net/hankai945/article/details/63343447>

大多数发行版不会包含 /usr/sbin/policy-rc.d 文件，因为这个文件是用于帮助管理员控制哪些包可以在安装、卸载、更新时执行脚本。

而在 docker 容器中，大多数 apt-get install 发生在 docker build 阶段，这个阶段如果去启动/停止服务，可能因为一些意外而无法成功并且也没有必要。大多数服务都是在 docker run 或 docker start 时启动/停止。

如果你一定要启动/停止服务，可以将 /usr/sbin/policy-rc.d 文件中的返回值改为0。

0 - action allowed

1 - unknown action (therefore, undefined policy)

100 - unknown initscript id

101 - action forbidden by policy

102 - subsystem error

103 - syntax error

104 - [reserved]

105 - behaviour uncertain, policy undefined.

106 - action not allowed. Use the returned fallback actions

(which are implied to be “allowed”) instead.

##### Starting and Stopping the MySQL Server

The MySQL server is started automatically after installation. You can check the status of the MySQL server with the following command:

sudo service mysql status

Stop the MySQL server with the following command:

sudo service mysql stop

To restart the MySQL server, use the following command:

sudo service mysql start

##### 报错: /lib/apparmor/profile-load: No such file or directory

root@abe27b06a17a:/# /etc/init.d/mysql start

/etc/init.d/mysql: line 63: /lib/apparmor/profile-load: No such file or directory

root@abe27b06a17a:/# service mysql start

/etc/init.d/mysql: line 63: /lib/apparmor/profile-load: No such file or directory

解决方法, 进入到ubuntu物理机中搜索此文件, 然后复制到docker中.

find / -name profile-load

# /lib/apparmor/profile-load

docker cp /lib/apparmor/profile-load scrapy00:/lib/apparmor

docker cp /lib/apparmor/functions scrapy00:/lib/apparmor/

docker attach scrapy00

service mysql stop

service mysql start

apt install vim

#### 安装redis并配置

##### 安装 redis

wget http://download.redis.io/redis-stable.tar.gz

tar xvzf redis-stable.tar.gz

cd redis-stable

apt install make gcc

##### make 错误

root@b671f532a842:/redis-stable# make

cd src && make all

make[1]: Entering directory '/redis-stable/src'

CC Makefile.dep

CC adlist.o

In file included from adlist.c:34:0:

zmalloc.h:50:31: fatal error: jemalloc/jemalloc.h: No such file or directory

compilation terminated.

Makefile:228: recipe for target 'adlist.o' failed

make[1]: \*\*\* [adlist.o] Error 1

make[1]: Leaving directory '/redis-stable/src'

Makefile:6: recipe for target 'all' failed

make: \*\*\* [all] Error 2

解决方法:

make MALLOC=libc

# optional

make之后可以使用make test进行测试. 然后再make install

apt install tcl

make test

make install

##### 启动 redis

redis-server

查看redis是否启动, 重新打开一个终端执行redis-cli连接到redis-server

$ redis-cli

redis 127.0.0.1:6379> ping

PONG

redis 127.0.0.1:6379> set mykey somevalue

OK

redis 127.0.0.1:6379> get mykey

"somevalue"

##### 使用redis\_init\_script脚本启动redis

* Create a directory where to store your Redis config files and your data:

sudo mkdir /etc/redis

sudo mkdir /var/redis

* Copy the init script that you'll find in the Redis distribution under the **utils** directory into /etc/init.d. # 将redis安装目录的utils下的redis\_init\_script拷贝到/etc/init.d目录下, 放在/etc/init.d下，主要是将redis作为一个系统的daemon进程去运行的，每次系统启动，redis进程一起启动

We suggest calling it with the name of the port where you are running this instance of Redis. For example:

17839

sudo cp ./utils/redis\_init\_script /etc/init.d/redis\_6379

sudo cp ./utils/redis\_init\_script /etc/init.d/redis\_17839

* (optional) 赋予脚本执行权限

chmod a+x /etc/init.d/redis\_6379

chmod a+x /etc/init.d/redis\_17839

* Edit the init script.

sudo vim /etc/init.d/redis\_6379

sudo vim /etc/init.d/redis\_17839

Make sure to modify **REDISPORT** accordingly to the port you are using. Both the pid file path and the configuration file name depend on the port number.

# 根据需要修改端口号

REDISPORT=6379

# REDISPORT=17839

# 根据需要修改安装的redis目录

EXEC=/usr/local/bin/redis-server

CLIEXEC=/usr/local/bin/redis-cli

PIDFILE=/var/run/redis\_${REDISPORT}.pid

# 可以修改配置文件目录，也可以按照这个目录在linux上创建

CONF="/etc/redis/${REDISPORT}.conf"

* Copy the template configuration file you'll find in the root directory of the Redis distribution into /etc/redis/ using the port number as name, for instance:

sudo cp redis.conf /etc/redis/6379.conf

sudo cp ./redis-stable/redis.conf /etc/redis/17839.conf

* Create a directory inside /var/redis that will work as data and working directory for this Redis instance:

sudo mkdir /var/redis/6379 -p

sudo mkdir /var/redis

sudo mkdir /var/redis/17839

* Edit the configuration file, making sure to perform the following changes:

sudo vim /etc/redis/6379.conf

sudo vim /etc/redis/17839.conf

1. 注释掉 bind 127.0.0.1 允许远程登录 redis
2. Change the port accordingly. In our example it is not needed as the default port is already 6379. 设置redis监听的端口号
3. Set daemonize to yes (by default it is set to no). 让redis以daemon进程启动
4. Set the pidfile to /var/run/redis\_6379.pid (modify the port if needed). 设置redis的pid文件.
5. Set your preferred loglevel. 设置日志等级
6. Set the logfile to /var/log/redis\_6379.log
7. Set the dir to /var/redis/6379 (very important step!) 设置持久化文件的存储位置（该目录需要有足够的磁盘空间）
8. requirepass 设置登录密码

* Finally add the new Redis init script to all the default runlevels using the following command:

sudo update-rc.d redis\_6379 defaults

sudo update-rc.d redis\_17839 defaults

* 出错信息 incomplete LSB comment

root@6119cae9dac1:/redis-stable# update-rc.d redis\_6379 defaults

insserv: Script redis\_6379 is broken: incomplete LSB comment.

insserv: missing `Required-Start:' entry: please add even if empty.

insserv: missing `Required-Stop:' entry: please add even if empty.

解决方法

解决方法: 修改redis启动文件, 在Begin init info中增加required-start和required-stop字段.

vim /etc/init.d/redis\_6379

### BEGIN INIT INFO

# Provides: redis\_6379

# Required-Start: $syslog

# Required-Stop: $syslog

# Should-Start: $local\_fs

# Should-Stop: $local\_fs

# Default-Start: 2 3 4 5

# Default-Stop: 0 1 6

# Short-Description: Redis data structure server

# Description: Redis data structure server. See https://redis.io

### END INIT INFO

再次执行

sudo update-rc.d redis\_6379 defaults

* You are done! Now you can try running your instance with: 启动redis

sudo /etc/init.d/redis\_6379 start

sudo /etc/init.d/redis\_17839 start

* 验证redis是否启动成功

ps -ef | grep redis

ps -aux | grep redis

* 没有设置登录密码的redis的关闭

/etc/init.d/redis\_6379 stop

* 登录带有密码验证的redis-server

redis-cli -p 17839 -a Xzz@8481

或者

redis-cli -p 17839

auth password

* 设置有密码验证的redis的关闭

方法一:

redis-cli -p 17839 -a password shutdown

方法二:

redis-cli -p 17839

auth password

shut down

quit

Make sure that everything is working as expected:

1. Try pinging your instance with redis-cli.
2. Do a test save with redis-cli save and check that the dump file is correctly stored into /var/redis/6379/ (you should find a file called dump.rdb).
3. Check that your Redis instance is correctly logging in the log file.
4. If it's a new machine where you can try it without problems make sure that after a reboot everything is still working.

Note: In the above instructions we skipped many Redis configuration parameters that you would like to change, for instance in order to use AOF persistence instead of RDB persistence, or to setup replication, and so forth. Make sure to read the example redis.conf file (that is heavily commented) and the other documentation you can find in this web site for more information.

#### 安装python3.6并创建虚拟环境

##### 添加第三方软件源

sudo apt-get install software-properties-common -y

sudo add-apt-repository ppa:jonathonf/python-3.6 -y

sudo apt-get update -y

##### 安装python3.6

sudo apt-get install python3.6 -y

检查已安装的 Python 3.6 版本

python3.6 -V

# Python 3.6.5

##### 安装pip3

由于ubuntu 16.04 默认使用的是python3.5, 所以安装的是python3.5对应的pip

sudo apt install python3-pip -y

# 查看安装的pip3的版本

pip3 -V

# pip 8.1.1 from /usr/lib/python3/dist-packages (python 3.5)

##### 升级pip版本

会使用pip3 替换系统默认的pip

sudo pip3 install --upgrade pip

pip3 -V

# pip 18.0 from /usr/local/lib/python3.5/dist-packages/pip (python 3.5)

pip -V #(出错)

docker中使用exit关闭容器, 回到ubuntu中.

docker start scrapy00

docker attach scrapy00

reboot # 重启容器后pip3 -V和pip -V都显示为pip 18.0

##### 使用pip 国内镜像

# 如果目录不存在, 要先创建目录, 否则无法保存文件.

mkdir ~/.pip

vim ~/.pip/pip.conf

[global]

index-url = http://pypi.douban.com/simple

[install]

trusted-host=pypi.douban.com

# 阿里云ECS服务器默认配置

## Note, this file is written by cloud-init on first boot of an instance

## modifications made here will not survive a re-bundle.

###

[global]

index-url=http://mirrors.cloud.aliyuncs.com/pypi/simple/

[install]

trusted-host=mirrors.cloud.aliyuncs.com

[global]

index-url = https://mirrors.aliyun.com/pypi/simple/

[install]

trusted-host=mirrors.aliyun.com

##### 安装虚拟环境

如果pip和pip3的版本不同, 要执行pip3 install virtualenv virtualenvwrapper

pip install virtualenv

pip install virtualenvwrapper

##### 配置虚拟环境

# 查找python3和virtualenvwrapper.sh的路径

which python3

# /usr/bin/python3

which virtualenvwrapper.sh

# /usr/local/bin/virtualenvwrapper.sh

# 配置virtualwrapper环境变量

sudo vim ~/.bashrc

# 在最后一行添加如下内容

export WORKON\_HOME=$HOME/.virtualenvs

export VIRTUALENVWRAPPER\_PYTHON=/usr/bin/python3

source /usr/local/bin/virtualenvwrapper.sh

# 使配置失效

source ~/.bashrc

##### 使用python3.6新建虚拟环境

which python3.6

/usr/bin/python3.6

# 指定使用python3.6创建虚拟环境

mkvirtualenv --python=/usr/bin/python3.6 python3\_spider

##### 安装python开发包

一键安装

sudo apt install python3.6-dev libmysqlclient-dev

pip install ipython pdir2 retrying requests lxml beautifulsoup4 pillow fake-useragent mysqlclient pymysql pymongo redis selenium openvr scrapy scrapy-redis scrapyd

分立功能

pip install ipython pdir2

pip install retrying

pip install requests lxml

pip install beautifulsoup4

pip install pillow fake-useragent

sudo apt install python3.6-dev libmysqlclient-dev

pip install mysqlclient

pip install pymysql pymongo redis

pip install selenium openvr

pip install scrapy scrapy-redis scrapyd

mysqlclient 如果安装失败

ubuntu安装

sudo apt install python3.6-dev libmysqlclient-dev

centos安装

sudo yum install python-devel mysql-devel

#### 配置物理机中的mysql和redis-server

爬虫在容器中运行时, 使用物理机的redis-server来去重, 同时把数据保存到物理机的mysql中.

##### 测试容器和物理机间能否正常通信

# 升级apt-get命令

apt-get update

# 安装ifconfig命令包

apt-get install net-tools

# 安装ping命令包

apt-get install iputils-ping

aliyun物理机 docker0网卡的ip地址为: 172.17.0.1

外网地址为: 47.100.193.219

进入到scrapy00 容器中, 使用ifconfig 查看自身的ip地址

scrapy00: 172.17.0.2

在scrapy00容器中测试能否ping通物理机

ping 172.17.0.1

##### aliyun修改mysql服务器的配置

修改mysql的配置文件

vim /etc/mysql/mysql.conf.d/mysqld.cnf

注释掉 bind-address = 127.0.0.1

# 重启mysql服务器

service mysql restart

使用root账号登录

mysql -uroot -p

grant all privileges on \*.\* to root@"%" identified by "Xzz@8481" with grant option;

新建阿里云6603端口的入站规则

也可以在mysql中新建用户并赋予权限

create user "david"@"%" inentified by "s6u9p5e3r8"

grant all privileges on \*.\* to david;

grant create,select,update,delete,insert on \*.\* to david;

在docker中测试aliyun的mysql和redis连接

mysql -h 172.17.0.1 -p

redis-cli 172.17.0.1 -p 17839

##### 在docker中测试爬虫

在windows中新建爬虫 jobbole\_docker

cd jobbole\_article

scrapy genspider jobbole\_docker blog.jobbole.com

把原来jobbole\_dis.py中的所有内容复制到jobole\_docker.py中, 修改爬虫名, 修改mysql和redis配置, 修改item pipeline, 把数据保存到aliyun物理机的mysql中.

aliyun中ubuntu物理机的redis服务器端口号是17839, 所以先要修改jobole\_docker.py中redis\_port的值为17839.

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

**import** **scrapy**

**from** **scrapy.loader** **import** ItemLoader

**from** **jobbole\_article.items** **import** JobboleArticleProcessItem

**from** **jobbole\_article.items** **import** JobboleArticleItemLoader

**from** **scrapy\_redis.spiders** **import** RedisSpider

**class** **JobboleItSpider**(RedisSpider):

name = 'jobbole\_dokcer'

allowed\_domains = ["blog.jobbole.com"]

*# start\_urls = ['http://blog.jobbole.com/all-posts/']*

redis\_key = 'jobbole:start\_urls'

*# 自定义设置*

custom\_settings = {

'LOG\_LEVEL': 'DEBUG',

'DOWNLOAD\_DELAY': 0,

*# 修改为分布式的爬虫*

*# 使用redis的调度器, 确保request存储到redis中*

"SCHEDULER": "scrapy\_redis.scheduler.Scheduler",

*# 使用scrapy\_redis进行去重, 确保所有爬虫共享相同的去重指纹*

"DUPEFILTER\_CLASS": "scrapy\_redis.dupefilter.RFPDupeFilter",

*# 在redis中保持scrapy-redis用到的队列, 不会清理redis中的队列, 从而可以实现暂停和恢复的功能.*

"SCHEDULER\_PERSIST": True,

*# 指定redis数据库的连接参数, 这里使用aliyun物理机的redis数据库*

'REDIS\_HOST': '172.17.0.1',

'REDIS\_PORT': 17839,

*# 指定 redis链接密码，和使用哪一个数据库*

'REDIS\_PARAMS' : {

'password': 'Xzz@8481',

'db': 2

},

*# 把数据保存到ubuntu物理机中的mysql数据库中.*

'MYSQL\_HOST': '172.17.0.1',

*# 这里设置的是数据库的名字, 不是数据表的名字*

'MYSQL\_DBNAME': 'jobbole\_article',

'MYSQL\_USERNAME': 'root',

'MYSQL\_PASSWORD': 'Xzz@8481',

'MYSQL\_PORT': 3306,

*# 启动redis的pipeline, 把提取到的数据都传输并保存到redis服务器中.*

"ITEM\_PIPELINES": {

*# 'jobbole\_article.pipelines.JsonWithEncodingPipeline': 300,*

*# 'jobbole\_article.pipelines.JsonExporterPipeline': 301,*

*# 'scrapy.pipelines.images.ImagesPipeline' : 200,*

*# 'jobbole\_article.pipelines.JobboleImagePipeline': 200,*

*# 'jobbole\_article.pipelines.MysqlPipeline': 302,*

'jobbole\_article.pipelines.MysqlTwistedPipline': 302,

*# 'scrapy\_redis.pipelines.RedisPipeline': 300*

}

}

在aliyun ubuntu 中删除掉原来的jobbole\_article项目文件夹, 在windows中把 jobbole\_article 文件夹上传到aliyun的服务器中

scp -r jobbole\_article root@47.100.193.219:/root/jobbole\_article

把jobbole\_article 复制到docker中.

docker cp jobbole\_article scrapy00:/root/

# 进入到docker中

docker attach scrapy00

# 进入到虚拟环境中

workon python3\_spider

打开另一个ssh, 连接到aliyun

# 连接到redis数据库

redis-cli -p 17839 -a password

# 选择数据库2

select 2

# 向redis-server数据库2中推送start\_urls地址.

lpush jobbole:start\_urls <http://blog.jobbole.com/all-posts/>

# 回到docker中, 运行爬虫

cd /root/jobbole\_article/jobbole\_article/spiders

scrapy runspider jobbole\_docker.py

# 查看aliyun物理机中的mysql数据库中是否写入了数据.

### 将容器封装为docker镜像.

镜像是母版, 不能够直接修改, 可以基于镜像创建一个容器, 在容器中部署相关的环境, 想要在其它地方使用相同环境的容器, 就可以把已经部署好生产环境的容器封装为一个镜像. 就可以通过封装的镜像来创建新的容器或虚拟机.

docker commit scrapy00 ubuntu\_scrapy\_base:20180904

# 查看所有的镜像

root@aliyun:~# docker images

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| REPOSITORY | TAG | IMAGE ID | CREATED | SIZE |
| ubuntu\_scrapy\_base | 20180904 | 651ec10c83c7 | 4 minutes ago | 1.19GB |
| ubuntu | 16.04 | 52b10959e8aa | 12 days ago | 115MB |
| ubuntu | 18.04 | 16508e5c265d | 12 days ago | 84.1MB |
| hello-world | latest | 2cb0d9787c4d | 7 weeks ago | 1.85kB |

删除docker image

docker rmi 651e

把部署好开发环境的docker容器保存为镜像.

参考:

<https://www.jianshu.com/p/8408e06b7273>

<https://blog.csdn.net/a906998248/article/details/46236687>

# 把scrapy00这个容器封装为一个镜像, 镜像名为ubuntu\_base, 版本号为20180904.

docker commit scrapy00 ubuntu\_base:20180904

# 把镜像保存为tar文件.

docker save -o /root/ubuntu\_base.tar ubuntu\_base

docker save -output /root/ubuntu\_base.tar ubuntu\_base

或:

docker save ubuntu\_base > /root/ubuntu\_base.tar

### 把保存的tar镜像文件复制到Yunzhuji的ubuntu物理机中.

scp /root/ubuntu\_base.tar [toot@219.235.1.146:/root/](mailto:toot@219.235.1.146:/root/)

进入yunzhuji中保存ubuntu\_base.tar的目录中, 导入镜像

docker load --input /root/ubuntu\_base.tar

docker load -i /root/ubuntu\_base.tar

或者:

docker load < /root/ubuntu\_base.tar

也可以直接把容器导出为镜像.

docker export scrpy00 > ubuntu\_base\_export.tar

docker export --output="ubuntu\_base\_export.tar" scrpy00

把镜像导入为容器

docker import /root/ubuntu\_base.tar

修改容器的 Repository 和 tag

docker tag 880f ubuntu\_base:20180904

### 可选项: 使用bypy上传把文件上传到百度云中.

Python client for Baidu Yun (Personal Cloud Storage) 百度云/百度网盘Python客户端

<https://github.com/houtianze/bypy>

pip install bypy

显示在云盘（程序的）根目录下文件列表：

bypy list

把当前目录同步到云盘：

bypy syncup

or

bypy upload

把云盘内容同步到本地来：

bypy syncdown

or

bypy downdir /

比较本地当前目录和云盘（程序的）根目录（个人认为非常有用）：

bypy compare

更多命令和详细解释请见运行bypy的输出。

调试

运行时添加-v参数，会显示进度详情。

运行时添加-d，会显示一些调试信息。

运行时添加-ddd，还会会显示HTTP通讯信息（警告：非常多）

经验分享

请移步至wiki，方便分享/交流。

直接在Python程序中调用

from bypy import ByPy

bp=ByPy()

bp.list() # or whatever instance methods of ByPy class

在aliyun中把ubuntu\_base.tar保存到百度云中.

cd ~

mkdir baiduyunsync

mv ubuntu\_base.tar baiduyunsync

cd baiduyunsync

bypy upload

在yunzhuji中同步文件夹

bypy syncdown

### 在yunzhuji的容器中运行爬虫

# 加载ubuntu\_base.tar为镜像文件

docker load -i /root/ubuntu\_base.tar

docker images

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| REPOSITORY | TAG | IMAGE ID | CREATED | SIZE |
| ubuntu\_base | v20180904 | e3074586a4e5 | 11 hours ago | 1.19GB |
| ubuntu | 16.04 | 52b10959e8aa | 13 days ago | 115MB |

使用ubuntu\_base创建5个容器, --name参数分别为scrapy01~scrapy05

docker run --name scrapy01 -tid e3074

# 显示正在运行的容器

docker ps

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| CONTAINER ID | IMAGE | COMMAND | CREATED | STATUS | PORTS | NAMES |
| b04d8ea63302 | e3074 | /bin/bash | About a minute ago | Up About a minute | | scrapy05 |
| 7c2a1adc80f0 | e3074 | /bin/bash | About a minute ago | Up About a minute | | scrapy04 |
| 093dbc99d2fc | e3074 | /bin/bash | About a minute ago | Up About a minute | | scrapy03 |
| 06ae29cca356 | e3074 | /bin/bash | About a minute ago | Up About a minute | | scrapy02 |
| 2d98ec3fd767 | e3074 | /bin/bash | About a minute ago | Up About a minute | | scrapy01 |

# 在yunzhuji ubuntu物理机中安装如下2个包.

# 安装ifconfig命令包

apt-get install net-tools

# 安装ping命令包

apt-get install iputils-ping

# 查看docker0网卡的地址

ifconfig docker0

# 172.17.0.1

yunzhuji ubuntu中修改mysql服务器的配置

vim /etc/mysql/mysql.conf.d/mysqld.cnf

注释掉 bind-address = 127.0.0.1

# 重启mysql服务器

service mysql restart

使用root账号登录

mysql -uroot -p

grant all privileges on \*.\* to root@"%" identified by "Xzz@8481" with grant option;

查看yunzhuji的物理机中mysql和redis服务是否已经启动并能正常运行.

进入到每个容器中, 测试其与物理机的连接状态.

docker exec -it scrapy01 /bin/bash

ping 172.17.0.1

测试能否连接到物理机的mysql数据库中.

mysql -h 172.17.0.1 -uroot -p

测试能否连接到物理机的redis数据库中.

redis-cli -h 172.17.0.1 -a password

注意, yunzhuji中redis-server的端口号为6379, 需要修改 jobbole\_docker.py中redis的端口号.

可以先在windows中修改, 然后使用scp复制到yunzhuji的物理机中

scp -r jobbole\_article root@219.235.1.146:/root/jobbole\_article

密码

0v0oSZ,Ns5

再从yunzhuji的物理机中复制到 每一个容器中. 注意需要进入每一个容器, 删除原来的jobbole\_article项目文件夹及其中的所有文件.

docker exec -it scrapy01 /bin/bash

cd

rm -r jobbole\_article

docker cp jobbole\_article scrapy01:/root/

进入到每个容器中, 进入虚拟环境, 运行 jobbole\_docker.py

cd jobbole\_article/jobbole\_article/spiders/

workon python3\_spider

scrapy runspider jobbole\_docker.py

向物理机的redis数据库的数据库2 push入start\_urls

lpush jobbole:start\_urls <http://blog.jobbole.com/all-posts/>

### scrapy 日志处理

<https://blog.csdn.net/hopyGreat/article/details/78771500>

### 分布式爬虫爬取结束时自动结束爬虫.

# 通过自定义扩展实现, 通过空跑的次数决定

<https://acefei.github.io/how-to-stop-scrapy-redis-spider-when-its-idle.html>

通过修改源码实现

<https://blog.csdn.net/zwq912318834/article/details/78873172>

# 通过自定义扩展实现, 通过时间关闭爬虫

<https://blog.csdn.net/xc_zhou/article/details/80907047>

这里使用第一种方法, 下载源码

<https://github.com/acefei/ace-crawler>

把extensions.py放在settings.py同目录下, 修改分布式爬虫文件 jobbole\_dis.py和jobbole\_docker.py, 在custom\_settings中启动extensions, 如下

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

**import** **scrapy**

**from** **scrapy.loader** **import** ItemLoader

**from** **jobbole\_article.items** **import** JobboleArticleProcessItem

**from** **jobbole\_article.items** **import** JobboleArticleItemLoader

**from** **scrapy\_redis.spiders** **import** RedisSpider

**class** **JobboleItSpider**(RedisSpider):

name = 'jobbole\_docker'

allowed\_domains = ["blog.jobbole.com"]

*# start\_urls = ['http://blog.jobbole.com/all-posts/']*

redis\_key = 'jobbole:start\_urls'

*# 自定义设置*

custom\_settings = {

'LOG\_LEVEL': 'DEBUG',

'DOWNLOAD\_DELAY': 0,

*# 修改为分布式的爬虫*

*# 使用redis的调度器, 确保request存储到redis中*

"SCHEDULER": "scrapy\_redis.scheduler.Scheduler",

*# 使用scrapy\_redis进行去重, 确保所有爬虫共享共同的去重指纹*

"DUPEFILTER\_CLASS": "scrapy\_redis.dupefilter.RFPDupeFilter",

*# 在redis中保持scrapy-redis用到的队列, 不会清理redis中的队列, 从而可以实现暂停和恢复的功能.*

"SCHEDULER\_PERSIST": True,

*# 指定redis数据库的连接参数*

'REDIS\_HOST': '172.17.0.1',

'REDIS\_PORT': 6379,

*# 指定redis数据库的连接参数, 这里使用ubuntu物理机的redis数据库*

'REDIS\_PARAMS' : {

'password': 'Xzz@8481',

'db': 2

},

*# 把数据保存到ubuntu物理机中的mysql数据库中.*

'MYSQL\_HOST': '172.17.0.1',

*# 这里设置的是数据库的名字, 不是数据表的名字*

'MYSQL\_DBNAME': 'jobbole\_article',

'MYSQL\_USERNAME': 'root',

'MYSQL\_PASSWORD': 'Xzz@8481',

'MYSQL\_PORT': 3306,

*# 启动redis的pipeline, 把提取到的数据都传输并保存到redis服务器中.*

"ITEM\_PIPELINES": {

*# 'jobbole\_article.pipelines.JsonWithEncodingPipeline': 300,*

*# 'jobbole\_article.pipelines.JsonExporterPipeline': 301,*

*# 'scrapy.pipelines.images.ImagesPipeline' : 200,*

*# 'jobbole\_article.pipelines.JobboleImagePipeline': 200,*

*# 'jobbole\_article.pipelines.MysqlPipeline': 302,*

'jobbole\_article.pipelines.MysqlTwistedPipline': 302,

*# 'scrapy\_redis.pipelines.RedisPipeline': 300*

},

*# 启用扩展功能, 当爬虫爬取结束时, 空跑5次时自动结束爬虫.*

'EXTENSIONS': {

'jobbole\_article.extensions.CloseSpiderRedis': 100,

},

'CLOSE\_SPIDER\_AFTER\_IDLE\_TIMES' = 5

}