**一 。首先创建个人虚拟环境：**

1.安装虚拟环境 virtualenvwrapper 【使用豆瓣源】

**maxinehehe@maxinehehe-PC:~$ pip install -i https://pypi.douban.com.simple/ virtualenvwrapper**

2.【windows忽略】修改bashrc文件

**maxinehehe@maxinehehe-PC:~$** gedit ~/.bashrc

添加：

export WORKON\_HOME=$HOME/.virtualenvs # .virtualenvs是虚拟环境文件夹 可自己创建

source /media/maxinehehe/5C584003583FDB0A/python35/Scripts/virtualenvwrapper.sh # 可通过 find / -name virtualenvwrapper.sh

**maxinehehe@maxinehehe-PC:~$** source ~/.bashrc

使配置立即生效

3.创建虚拟环境【python3】

**maxinehehe@maxinehehe-PC:~$** mkvirtualenv --python3=/usr/bin/python3 article\_spider

4.查看或进入虚拟环境

**maxinehehe@maxinehehe-PC:~$** workon

ArticelSpider

article\_spider

py2scrapy

py3scrapy

**maxinehehe@maxinehehe-PC:~$** workon article\_spider

**(article\_spider) maxinehehe@maxinehehe-PC:~$**

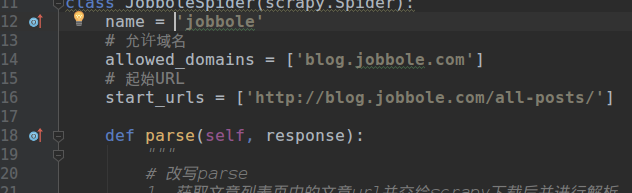
4.创建爬虫工程项目【pip install -i 豆瓣源 scrapy】

**maxinehehe@maxinehehe-PC:~$ scrapy startproject ArticleSpider**

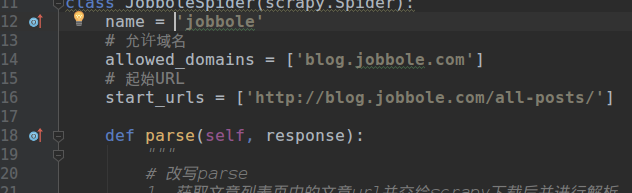
5.创建新的爬虫

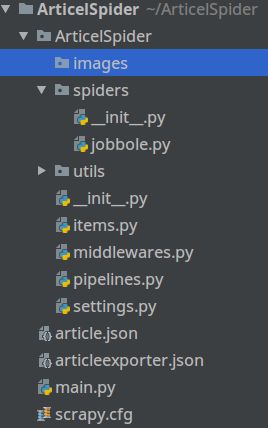
**maxinehehe@maxinehehe-PC:~$ scrapy genspider jobbole** <http://blog.jobbole.com/all-posts/>

根据模板生成一个新的爬虫



当前文件目录如下：





5.目录结构理解

**spiders**

**jobbole.py**

**文件夹下放的是实际的爬虫代码，处理爬虫的逻辑，比如使用正则表达式或xpath,css等选择器爬取内容。**

class JobboleSpider(scrapy.Spider): # 模板

name = **'jobbole'**

# 允许域名

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

# 起始URL

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

# 解析函数 解析网址

def parse(self, response):

# 解析列表页中的所有文章URL 拿到当前页文章的网址列表数组

post\_nodes = response.css(**"#archive .floated-thumb .post-thumb a"**) # 获取节点

# 该节点包含 图片地址 和 文件详情地址

# '<a target="\_blank" href="http://blog.jobbole.com/114194/" title="深入学习 Redis（3）：主从复制"><img #src="http://jbcdn2.b0.upaiyun.com/2016/04/49961db8952e63d98b519b76a2daa5e2.png" alt="" width="120" height="120"></a>'

for post\_node in post\_nodes:

image\_url = post\_node.css(**"img::attr(src)"**).extract\_first(**""**) # extract()[0] # 提取图片地址

post\_url = post\_node.css(**"::attr(href)"**).extract\_first(**""**) # 提取文章详情地址

yield Request(url=parse.urljoin(response.url, post\_url), meta={**"front\_image\_url"**:image\_url}, callback=self.parse\_detail)

# parse.urljoin(a,b) 比如 a = http://blog.jobbole.com b=11203 4 即是将二者连接成 http://blog.jobbole.com/112034

# 若 a=http://blog.jobbole.com b=http://blog.jobbole.com/112034 则结果为<http://blog.jobbole.com/112034>

# 所以用parse.urljoin()可以解决很多当前路径或者非当前路径问题

# meta 信息则会传递到request中 以便其它函数调用其中信息

# callback则是表示 回调函数 即调取哪一个函数来处理当前的url

# 下面是xpath和css两种选择器的取法

# next\_url = response.xpath('//\*[@id="archive"]/div[@class="navigation margin-20"]/a[@class="next page-numbers"]/@href').extract()[0]

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

print(**"下一页："**+next\_url+**"**\n**"**)

if next\_url:

# <http://blog.jobbole.com/all-posts/page/2/>

# 拿到下一页 交给scrapy下载器

# yield Request(url=parse.urljoin(response.url, post\_url), callback=self.parse) # 传函数名即可 由底层twisted调用

# 错误原因 由于传递的是post\_url始终转回爬取这一页 而next\_url并未传递过去

# 如果继续传入 post\_url【已爬取】scrapy会自动判断 为已经爬去 则终止爬虫运行

yield Request(url=parse.urljoin(response.url, next\_url), callback=self.parse) # 传函数名即可 由底层twisted调用

# 此处调取parse函数因为 这里需要回调处理文章列表页的函数

**为什么使用yield?**

**上面可以看出算是递归的形式的，如果网站待爬取条目趋近于无穷，那么这样的递归是无法想象的，还没等爬取，内存就已经溢出了。使用yield生成器则可以 使用的时候调取一次 不使用就不进行储存生成，极大地节省了空间，同时提高了程序运行效率。**

def parse\_detail(self, response):

*"""*

*提取文章的具体字段*

***:param*** *response:*

***:return****:*

*"""*

# 编写数据的解析和爬取

article\_item = JobboleArticleItem() # 实例化

front\_image\_url = response.meta.get(**"front\_image\_url"**, **""**) # 第二个“”是默认值 文章封面图

# 通过item\_loader加载item

# 传递 实例化对象

# ArticleItemLoader

item\_loader = ArticleItemLoader(item=JobboleArticleItem(), response=response)

# 统一使用css

item\_loader.add\_css(**"title"**, **".entry-header h1::text"**)

item\_loader.add\_value(**"url"**, response.url)

item\_loader.add\_value(**"url\_object\_id"**, get\_md5(response.url))

item\_loader.add\_css(**"create\_date"**, **"p.entry-meta-hide-on-mobile::text"**)

item\_loader.add\_value(**"front\_image\_url"**, [front\_image\_url])

item\_loader.add\_css(**"praise\_nums"**, **".vote-post-up h10::text"**)

item\_loader.add\_css(**"comment\_nums"**, **"a[href='#article-comment'] span::text"**)

item\_loader.add\_css(**"fav\_nums"**, **".bookmark-btn::text"**)

item\_loader.add\_css(**"tags"**, **"p.entry-meta-hide-on-mobile a::text"**)

item\_loader.add\_css(**"content"**, **"div.entry"**)

# 进行解析

article\_item = item\_loader.load\_item()

**# 只有调用load\_item()方法才会对item进行解析**

yield article\_item # 会传送到pipelines

# 此处需要说明的是为什么要使用ItemLoader呢？

**ItemLoader 简介**

通过之前的学习，已经知道网页的基本解析流程就是先通过 css/xpath 方法进行解析，然后再把值封装到 Item 中，如果有特殊需要的话还要对解析到的数据进行转换处理，这样当解析代码或者数据转换要求过多的时候，会导致代码量变得极为庞大，从而降低了可维护性。同时在 sipider 中编写过多的数据处理代码某种程度上也违背了单一职责的代码设计原则。我们需要使用一种更加简洁的方式来获取与处理网页数据，ItemLoader 就是用来完成这件事情的。

ItemLoader 类位于 scrapy.loader ，它可以接收一个 Item 实例来指定要加载的 Item, 然后指定 response 或者 selector 来确定要解析的内容，最后提供了 add\_css()、 add\_xpath() 方法来对通过 css 、 xpath 解析赋值，还有 add\_value() 方法来单独进行赋值。

# 不使用的情况：

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

article\_item["title"] = title

# 使用的情况：

**item\_loader.add\_css("title", ".entry-header h1::text")**

**# 至于具体的提取逻辑则在 Item.py文件中进行处理**

**# yield article\_item则将 item传递给pipeline.py由pipeline内的类进行处理【保存至本地、数据库或者其他操作】**

item.py

**# 定义爬虫要爬取域的 item 模型，类似javabean，也可以使用django 的model。**

**import scrapy**

**from scrapy.loader import ItemLoader**

**from scrapy.loader.processors import MapCompose, TakeFirst, Join**

**import datetime**

**import re**

**def return\_value(value):**

**# 小技巧**

**return value**

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

**# 默认创建的item类**

**# define the fields for your item here like:**

**# name = scrapy.Field()**

**pass**

**def add\_jobbole(self, value):**

**return value + "-jobbole"**

**def date\_convert(value):**

**try:**

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

**except Exception as e:**

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

**return create\_date**

**def get\_nums(value):**

**pass**

**def add\_jobbole(self, value):**

**return value + "-jobbole"**

**def date\_convert(value):**

**try: # 可共用 提取数字**

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

**if match\_re:**

**nums = int(match\_re.group(1))**

**else:**

**nums = 0**

**return nums**

**# 如果每一个都去第一个值 是否都要自己去写每个TakeFirst() 当然不用scrapy.ItemLoader提供了对其进行重载**

**class ArticleItemLoader(ItemLoader):**

**default\_output\_processor = TakeFirst() # 默认取第一个**

**def remove\_comment\_tags(value):**

**# 去掉tag中提取的评论**

**if "评论" in value:**

**return ""**

**else:**

**return value**

**# 自定义item模型 用于统一处理对应类型**

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

**# item只有一种类型 统一为Field 可以接受任何类型 类似字典形式**

**# 在下面可以对输入进行预处理**

**“”“**

**需要说明的是：**

**input\_processor 是对该输入进行预处理的固定参数**

**output\_processor 是对该输出进行预处理的固定参数**

**”“”**

**title = scrapy.Field(**

**# 自定义对title预处理 在末尾添加标示**

**# MapCompose()可以从左到右调用函数 进行处理 title 注意：MapCompose可以调用多个函数**

**# input\_processor = MapCompose(add\_jobbole) # 调用add\_jobbole函数对title进行处理**

**input\_processor = MapCompose(lambda x:x + "-jobbole") # 一种方法**

**)**

**create\_date = scrapy.Field(**

**# 预处理时间 2017/2/23 成 2017-2-23**

**input\_processor = MapCompose(date\_convert), # 此处相当于参数**

**# output\_processor = TakeFirst()**

**# TakeFirst() 见名识意 即取【列表|...】第一个**

**)**

**url = scrapy.Field()**

**url\_object\_id = scrapy.Field()**

**# front\_image\_url必须是list 若用default\_input\_processor会导致下载出错**

**# 覆盖掉之前的方法**

**front\_image\_url = scrapy.Field(**

**output\_processor=MapCompose(return\_value) # 覆盖掉 default\_input\_processor**

**# 由于 default\_input\_processor对所有域都进行了默认输出处理 因此调用return\_value**

**# 覆盖调默认输出处理**

**)**

**front\_image\_path = scrapy.Field()**

**praise\_nums = scrapy.Field(**

**input\_processor=MapCompose(get\_nums), # 此处相当于参数**

**)**

**comment\_nums = scrapy.Field(**

**input\_processor=MapCompose(get\_nums) # 此处相当于参数**

**)**

**fav\_nums = scrapy.Field(**

**input\_processor=MapCompose(get\_nums) # 此处相当于参数**

**)**

**tags = scrapy.Field(**

**# tag\_list 转变**

**# 转变前：['职场 2 评论 程序员计算机专业']**

**# 转变后：“”**

**# 转变前：['职场 程序员计算机专业']**

**# 转变后：‘职场 程序员计算机专业'**

**input\_processor = MapCompose(remove\_comment\_tags),**

**output\_processor = Join(",")**

**)**

**content = scrapy.Field()**

**middlewares.py**

**scrapy中间件**

Spider中间件是介入到Scrapy的spider处理机制的钩子框架，可以添加代码来处理发送给 [Spiders](http://scrapy-chs.readthedocs.io/zh_CN/0.24/topics/spiders.html#topics-spiders)的response及spider产生的item和request。

**pipelines.py**

**import codecs**

**import json**

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

**from scrapy.exporters import JsonItemExporter**

**from twisted.enterprise import adbapi # 可以将MySQLdb的一些操作变成异步化操作**

**import MySQLdb**

**import MySQLdb.cursors**

**'''**

**利用pipeline保存到数据库中 pipelin用于处理数据域**

**'''**

**# 设置中打开ITEM\_PIPELINES方可生效**

**class ArticelspiderPipeline(object):**

**# 自动生成的Pipeline**

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

**return item # 要返回item [处理后的] 因为其他类可能要用**

**class JsonWithEncodingPipeline(object):**

**# 自定义json文件的导出**

**# 打开json文件 利用codecs处理文件 可以避免编码方面的麻烦**

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

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

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

**# 处理item的关键地方**

**lines = json.dumps(dict(item), ensure\_ascii=False)+"\n" # ensure\_ascii=False 防止写入中文出错**

**self.file.write(lines)**

**# 处理完之后 要返回去 下一个pipeline可能需要处理**

**return item**

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

**# 重载 当spider关闭时 保存文件**

**self.file.close()**

**class MysqlPipeline(object):**

***"""***

***自定义将数据写入数据库***

***"""***

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

**self.conn = MySQLdb.connect(host="127.0.0.1", user="root", password="123456", db="article\_spider", charset="utf8",**

**use\_unicode=True)**

**self.cursor = self.conn.cursor() # 实例化游标**

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

**insert\_sql = """insert into jobbole\_article(title, url, create\_date, fav\_nums)**

**values (%s, %s, %s, %s)"""**

**# 使用`占位符**

**self.cursor.execute(insert\_sql, (item["title"], item["url"], item["create\_date"], item["fav\_nums"]))**

**# 用commit（）才会提交到数据库**

**self.conn.commit()**

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

**self.cursor.close()**

**self.conn.close()**

**class MysqlTwistedPipeline(object):**

**# 采用异步机制写入MySQL**

**# 由于爬虫的爬取速度可能要快于数据写入数据库的速度 这样很容易造成阻塞 因此可以通过异步处理这种情况**

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

**self.dppool = dppool**

**# 创建连接池实例**

**# 用固定方法 【写法固定】 获取配置文件内信息**

**@classmethod**

**def from\_settings(cls, settings): # cls实际就是本类 MysqlTwistedPipeline**

**dpparms = dict(**

**# 从配置文件中获取参数**

**host = settings["MYSQL\_HOST"],**

**db = settings["MYSQL\_DBNAME"],**

**user = settings["MYSQL\_USER"],**

**passwd = settings["MYSQL\_PASSWORD"],**

**charset = "utf8",**

**cursorclass = MySQLdb.cursors.DictCursor, # 指定 curosr 类型 需要导入MySQLdb.cursors**

**use\_unicode = True**

**) # 由于要传递参数 所以参数名成要与connnect保持一致**

**# 用的仍是MySQLdb的库 twisted并不提供**

**# 异步操作**

**# adbapi # 可以将MySQLdb的一些操作变成异步化操作**

**dppool = adbapi.ConnectionPool("MySQLdb", \*\*dpparms)**

**# 告诉它使用的是哪个数据库模块 连接参数 pymysql 或者 MySQLdb**

**return cls(dppool) # 即实例化一个pipeline**

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

**# 使用twisted将mysql插入编程异步操作**

**# 指定操作方法和操作的数据 [下面会将方法异步处理]**

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

**# AttributeError: 'Deferred' object has no attribute 'addErrorback'**

**# query.addErrorback(self.handle\_error) # 处理异常**

**query.addErrback(self.handle\_error) # 处理异常**

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

**# 定义错误 处理异步插入的异常**

**print(failure)**

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

***"""***

***此类内其他都可以看作是通用 针对不同的sql操作只需要改写这里即可了***

***:param cursor:***

***:param item:***

***:return:***

***"""***

**insert\_sql = """insert into jobbole\_article(title, url, create\_date, fav\_nums, url\_object\_id,**

**front\_image\_url, front\_image\_path, comment\_nums, praise\_nums, tags, content)**

**values (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s )"""**

**# 使用`占位符**

**cursor.execute(insert\_sql, (item["title"], item["url"], item["create\_date"], item["fav\_nums"],**

**item["url\_object\_id"], item["front\_image\_url"],**

**item["front\_image\_path"], item["comment\_nums"], item["praise\_nums"],**

**item["tags"],item["content"]))**

**class JsonExporterPipeline(object):**

**# 调用scrapy提供的json文件 exporter导出json文件**

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

**self.file = open("articleexporter.json", "wb")**

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

**self.exporter.start\_exporting()**

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

**self.exporter.finish\_exporting()**

**self.file.close()**

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

**self.exporter.export\_item(item)**

**return item**

**# 定义自己的pipeline用于自定义存储图片**

**class ArticleImagePipeline(ImagesPipeline):**

***"""***

***设置文件保存路径***

***"""***

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

**# 重载该函数**

**if "front\_image\_url" in item:**

**for ok, value in results:**

**image\_file\_path = value["path"]**

**item["front\_image\_path"] = image\_file\_path**

**return item**

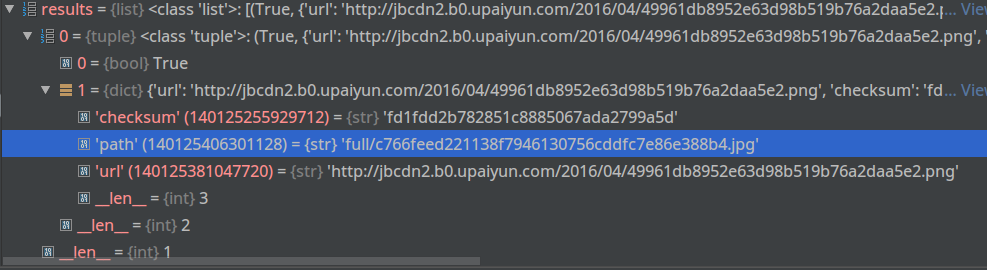
**"""**

需要在自定义的ImagePipeline类中重载的方法：get\_media\_requests(item, info)和item\_completed(results, items, info)。

正如工作流程所示，Pipeline将从item中获取图片的URLs并下载它们，所以必须重载get\_media\_requests，并返回一个Request对象，这些请求对象将被Pipeline处理，当完成下载后，结果将发送到item\_completed方法，这些结果为一个二元组的list，每个元祖的包含(success, image\_info\_or\_failure)。 \* success: boolean值，true表示成功下载 \* image\_info\_or\_error：如果success=true，image\_info\_or\_error词典包含以下键值对。失败则包含一些出错信息。 \* url：原始URL \* path：本地存储路径 \* checksum：校验码。

**"""**

results : **<class 'list'>: [(True, {'url': 'http://jbcdn2.b0.upaiyun.com/2016/04/49961db8952e63d98b519b76a2daa5e2.png', 'checksum': 'fd1fdd2b782851c8885067ada2799a5d', 'path': 'full/c766feed221138f7946130756cddfc7e86e388b4.jpg'})]**



setting.py

**配置文件**

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

**import os**

**# Scrapy settings for ArticelSpider 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 = 'ArticelSpider'**

**SPIDER\_MODULES = ['ArticelSpider.spiders']**

**NEWSPIDER\_MODULE = 'ArticelSpider.spiders'**

**# Crawl responsibly by identifying yourself (and your website) on the user-agent**

**#USER\_AGENT = 'ArticelSpider (+http://www.yourdomain.com)'**

**# Obey robots.txt rules**

**ROBOTSTXT\_OBEY = True # False 防止过滤掉不符合机器爬取规则的URL**

**# 爬取被阻止 则更改为False**

**# Configure maximum concurrent requests performed by Scrapy (default: 16)**

**#CONCURRENT\_REQUESTS = 32**

**# 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**

**# The download delay setting will honor only one of:**

**# 设置下载时延**

**# CONCURRENT\_REQUESTS\_PER\_DOMAIN = 1**

**# DOWNLOAD\_DELAY=2**

**#CONCURRENT\_REQUESTS\_PER\_IP = 16**

**# 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',**

**#}**

**# Enable or disable spider middlewares**

**# See https://doc.scrapy.org/en/latest/topics/spider-middleware.html**

**#SPIDER\_MIDDLEWARES = {**

**# 'ArticelSpider.middlewares.ArticelspiderSpiderMiddleware': 543,**

**#}**

**# Enable or disable downloader middlewares**

**# See https://doc.scrapy.org/en/latest/topics/downloader-middleware.html**

**#DOWNLOADER\_MIDDLEWARES = {**

**# 'ArticelSpider.middlewares.ArticelspiderDownloaderMiddleware': 543,**

**#}**

**# Enable or disable extensions**

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

**#EXTENSIONS = {**

**# 'scrapy.extensions.telnet.TelnetConsole': None,**

**#}**

**# Configure item pipelines**

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

**ITEM\_PIPELINES = {**

**# 'ArticelSpider.pipelines.JsonExporterPipeline': 2,**

**'scrapy.pipelines.images.ImagesPipeline':2, # scrapy自带图片下载 数字表示处理顺序【在该列表中的处理顺序】**

**'ArticelSpider.pipelines.ArticleImagePipeline': 1, # 重载方法 获取图片保存路径**

**# 'ArticelSpider.pipelines.MysqlPipeline': 1,**

**'ArticelSpider.pipelines.MysqlTwistedPipeline': 3, # 异步处理数据保存至数据库**

**}**

**# 配置item爬去字段**

**IMAGES\_URLS\_FIELD = "front\_image\_url" # 需要注意的是 scrapy会将front\_image\_url当成数组处理 所以格式要正确**

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

**# 讲两个路径名链接 找到images文件**

**IMAGES\_STORE = os.path.join(project\_dir, 'images') # 尽量配置相对路径**

**# 过滤掉小图片**

**# IMAGES\_MIN\_HEIGHT=100**

**# IMAGES\_MIN\_WIDTH=100**

**# Enable and configure the AutoThrottle extension (disabled by default)**

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

**#AUTOTHROTTLE\_ENABLED = True**

**# The initial download delay**

**#AUTOTHROTTLE\_START\_DELAY = 5**

**# The maximum download delay to be set in case of high latencies**

**#AUTOTHROTTLE\_MAX\_DELAY = 60**

**# The average number of requests Scrapy should be sending in parallel to**

**# each remote server**

**#AUTOTHROTTLE\_TARGET\_CONCURRENCY = 1.0**

**# Enable showing throttling stats for every response received:**

**#AUTOTHROTTLE\_DEBUG = False**

**# Enable and configure HTTP caching (disabled by default)**

**# See https://doc.scrapy.org/en/latest/topics/downloader-middleware.html#httpcache-middleware-settings**

**#HTTPCACHE\_ENABLED = True**

**#HTTPCACHE\_EXPIRATION\_SECS = 0**

**#HTTPCACHE\_DIR = 'httpcache'**

**#HTTPCACHE\_IGNORE\_HTTP\_CODES = []**

**#HTTPCACHE\_STORAGE = 'scrapy.extensions.httpcache.FilesystemCacheStorage'**

**MYSQL\_HOST = "127.0.0.1"**

**MYSQL\_DBNAME = "article\_spider" # 数据库名称 非表名称 表名称会在sql语句中指明**

**MYSQL\_USER = "root"**

**MYSQL\_PASSWORD = "123456"**

**最后**main.py**文件 运行文件**

**#!/usr/bin/env python**

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

**# @Time : 18-7-1 下午5:49**

**# @Author : maxinehehe**

**# @Site :**

**# @File : main.py**

**# @Software: PyCharm**

**from scrapy.cmdline import execute**

**import sys**

**import os**

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

**# os.path.abspath(\_\_file\_\_) 获取当前文件的绝对路径**

**# os.path.dirname() 获取括号内 文件的上一级路径 即是所在目录**

**execute(["scrapy", "crawl", "jobbole"]) # 【命令终端】启动爬虫 scrapy crawl 爬虫名称**

**终端运行：**

**maxinehehe@maxinehehe-PC:~$ scrapy crawl jobbole**

**jobbole为爬虫名称**

**至此项目完成。**