# 第一章 Python scrapy框架

本节所讲内容：

1.1 Scrapy 介绍

1.2 Scrapy 基本命令与结构

1.3 Scrapy 请求处理

实战：利用爬虫框架scrapy ，爬取JavaScript动态加载网页，将图片下载到本地。

## 1.1 Scrapy 介绍

1.1.1 Python scrapy 介绍 爬虫框架

爬虫（spider，网络蜘蛛）：模拟浏览器获取服务器资源的脚本。网络爬虫，是在网上进行数据抓取的程序，使用它能够抓取特定网页的HTML数据。

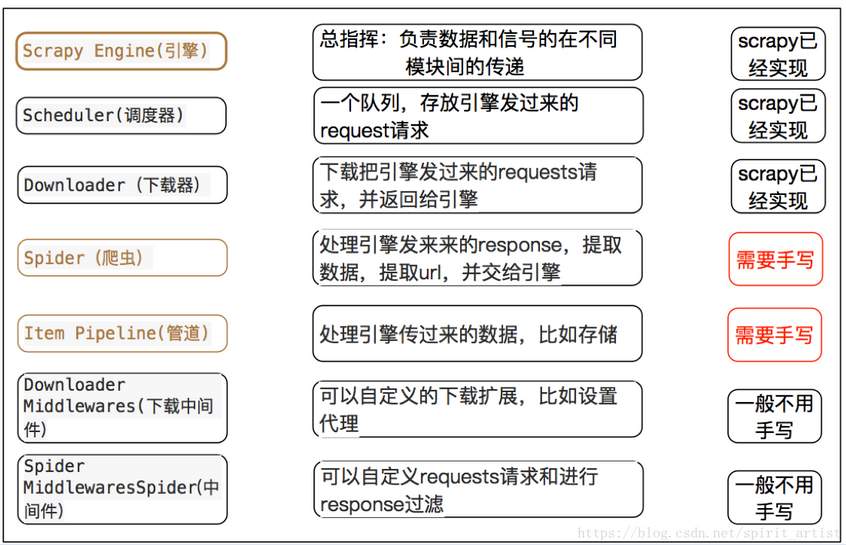
框架：对同一类型的项目共性代码的封装

Scrapy是用纯Python实现一个为了爬取网站数据、提取结构性数据而编写的应用框架，用途非常广泛。 框架的力量，用户只需要定制开发几个模块就可以轻松的实现一个爬虫，用来抓取网页内容以及各种图片，非常之方便。

让我们来看下它的基本结构：



Scrapy主要包括了以下组件：



1、引擎，用来处理整个系统的数据流处理，触发事务。

2、调度器，用来接受引擎发过来的请求，压入队列中，并在引擎再次请求的时候返回。

3、下载器，用于下载网页内容，并将网页内容返回给蜘蛛。

4、蜘蛛，蜘蛛是主要干活的，用它来制订特定域名或网页的解析规则。

5、项目管道，负责处理有蜘蛛从网页中抽取的项目，它的主要任务是清洗、验证和存储数据。当页面被蜘蛛解析后，将被发送到项目管道，并经过几个特定的次序处理数据。

6、下载器中间件，位于Scrapy引擎和下载器之间的钩子框架，主要是处理Scrapy引擎与下载器之间的请求及响应。

7、蜘蛛中间件，介于Scrapy引擎和蜘蛛之间的钩子框架，主要工作是处理蜘蛛的响应输入和请求输出。

8、调度中间件，介于Scrapy引擎和调度之间的中间件，从Scrapy引擎发送到调度的请求和响应。

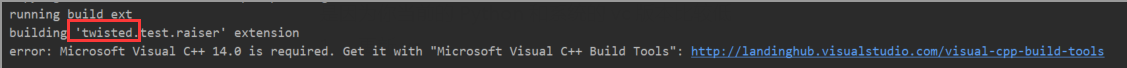
注：Scrapy 官方网站：<http://scrapy.org/>

1.1.2 Scrapy 安装

讲道理来说我们直接执行pip就OK了（pip install scrapy），但是几乎大部分的同学没有这个模块所用的依赖模块。如果大家不能直接安装或者安装出错，请按照以下安装步骤：

1. 安装 pywin32
2. 安装Twisted
3. 安装 scrapy

在安装过程中，假如报关于Vc++ 14.0 。。。。。。的错误，如下图所示。



因为你当前的Python和系统的vc版本比较低

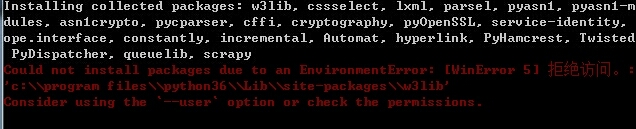
更新vc

安装twisted.whl包

pip install wheel

pip install twisted.whl

假如错误入学



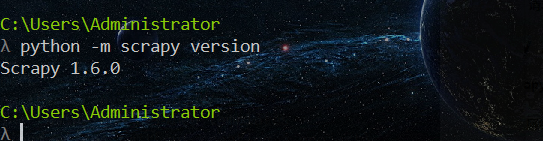
网络是不是完好，我们可以用pip install scrapy –i 国内源

阿里云  <http://mirrors.aliyun.com/pypi/simple/>  
中国科技大学  [https://pypi.mirrors.ustc.edu.cn/simple/](https://pypi.mirrors.ustc.edu.cn/simple/%20)  
豆瓣(douban)  <http://pypi.douban.com/simple/>   
清华大学  <https://pypi.tuna.tsinghua.edu.cn/simple/>  
中国科学技术大学  <http://pypi.mirrors.ustc.edu.cn/simple/>

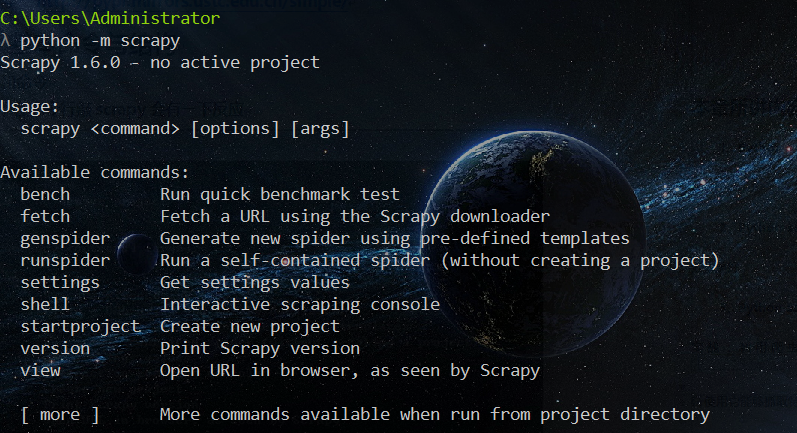
## 1.2 Scrapy 基本命令与结构

### 1.2.1 Scrapy 配置和命令

如果配置好之后，Version 查看版本，在cmder下敲入scrapy version 会有所反映，如下图所示：



然后我们可以输入scrapy 查看具体的命令，如下图：

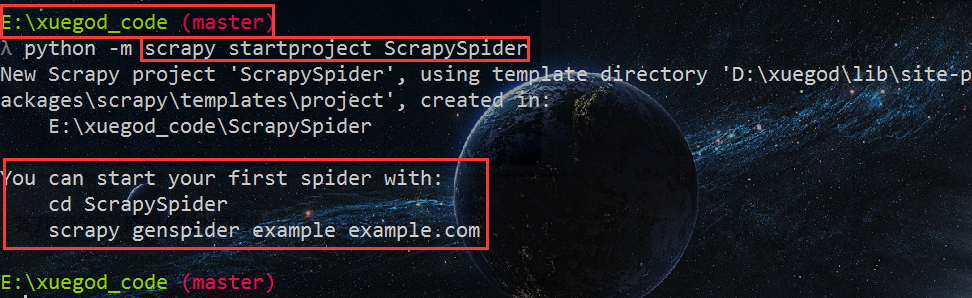


我们通常用到的命令

Scrapy命令在scrapy项目当中和项目外的参数有不同

### 1.2.2 Startproject 创建项目

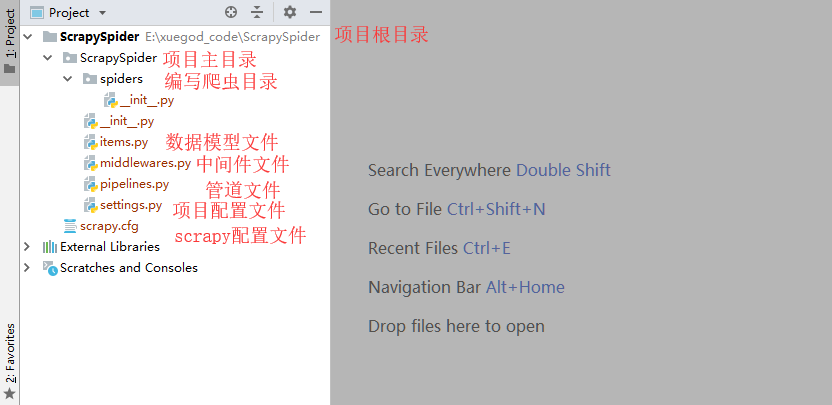
创建项目还是要注意，先在命令行当中切换到项目创建的目录，在目录下执行，就创建在那个目录下



### 1.2.3 爬虫结构

了解爬虫结构，主要了解两部分，第一是爬虫项目结构

1、爬虫项目结构



2、爬虫结构

Scrapy是python编写的框架，类里面规则和python的一样

import scrapy

#scrapy 本身已经定义好了爬虫请求的结构，我们只需要继承和重写

class TestSpider(scrapy.spiders.Spider):

"""

TestSpider 我们定义的爬虫类

scrapy.spiders.Spider 是scrapy定义好的爬虫

"""

name = "baiduSpider" #定义爬虫的名称

allowed\_domains = [

"https://www.baidu.com"

] #允许爬取的范围

start\_urls = [

"https://www.baidu.com"

] #默认爬取的起始点

def parse(self,response):

"""

scrapy 会自动调用downloader的下载器，当请求成功scapy会自动调用parse，

将结果返回到parse的response参数上

:param response:

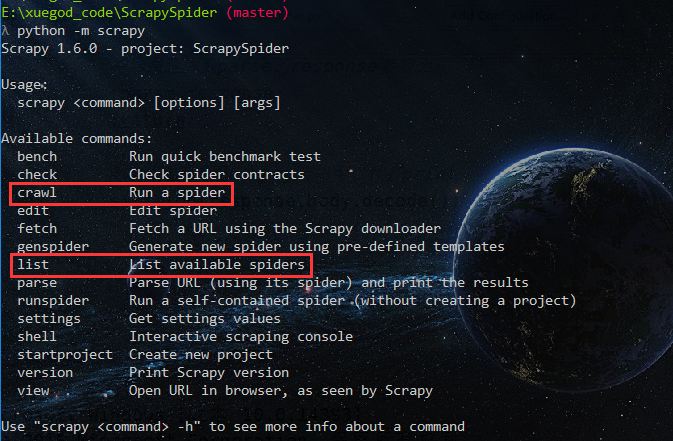
:return:

"""

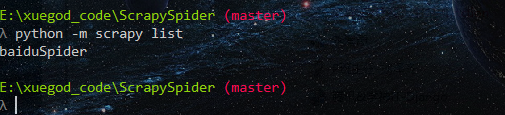
self.log(response.body.decode()) #调试方法在是scrapy当中可以代替print

上面我们完成了scrapy的编写，然后执行项目。

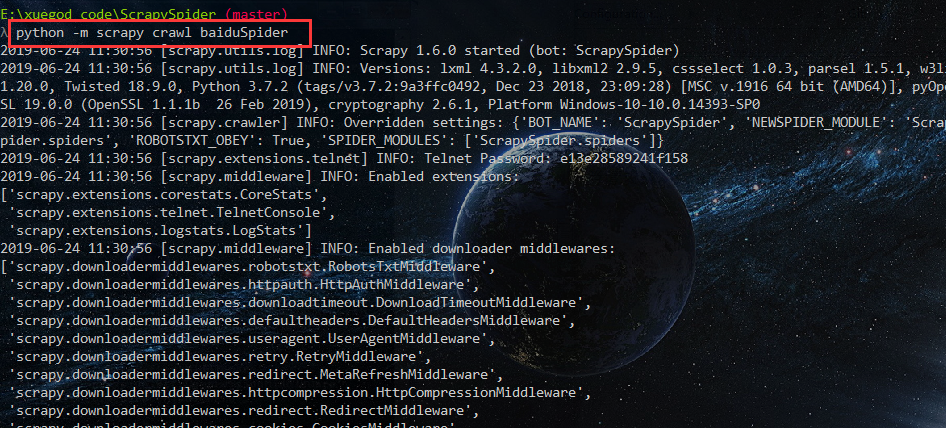
Scrapy命令在scrapy项目当中和项目外的参数有不同



list list 代表列出所有爬虫



crawl代表执行爬虫

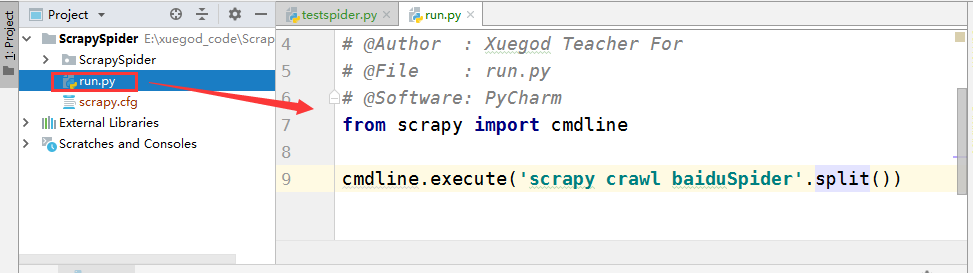


### 1.3.4 编译器中执行项目

在我们工作当中，我们需要对项目进行断点，这样执行很显然是不行的。断点需要在编译器如： pycharm当中执行项目

1. 创建启动文件

在爬虫的根目录下创建run.py



1. 编写执行代码

#coding:utf-8

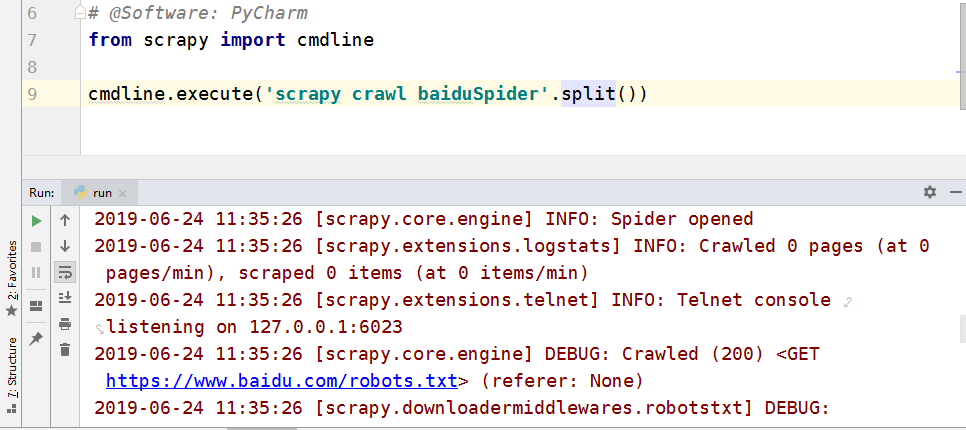
#导入scrapy执行命令的接口

from scrapy import cmdline

#调动接口执行命令

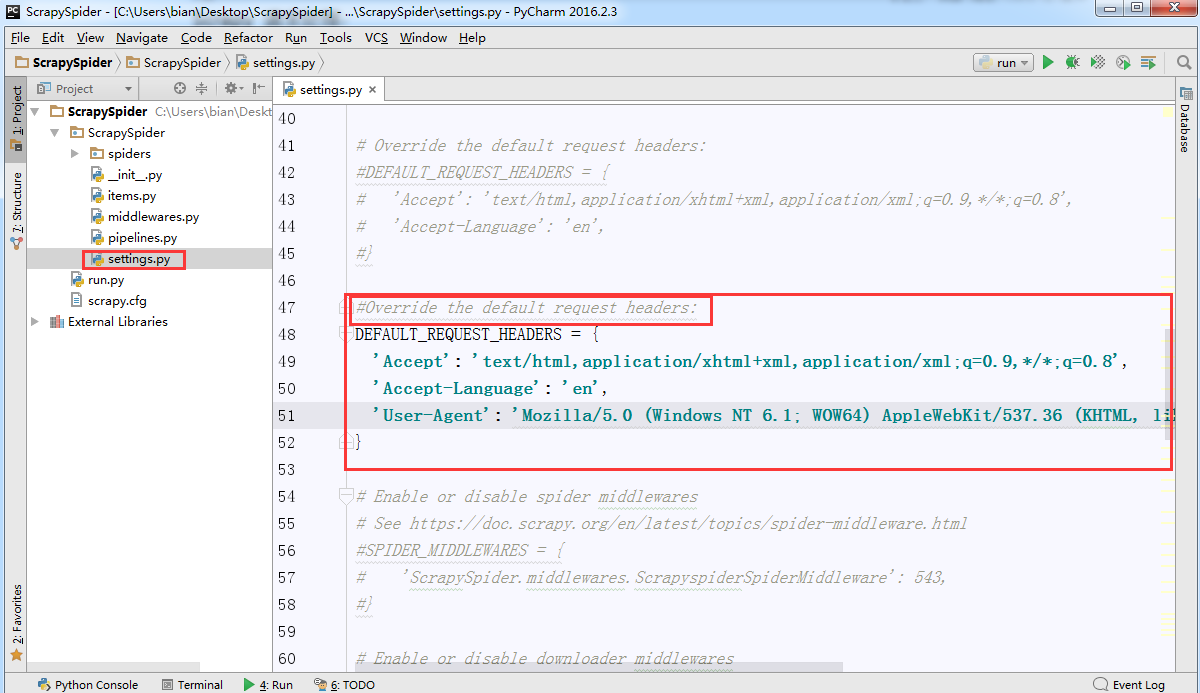
cmdline.execute("scrapy crawl baiduSpider".split())

运行结果如下



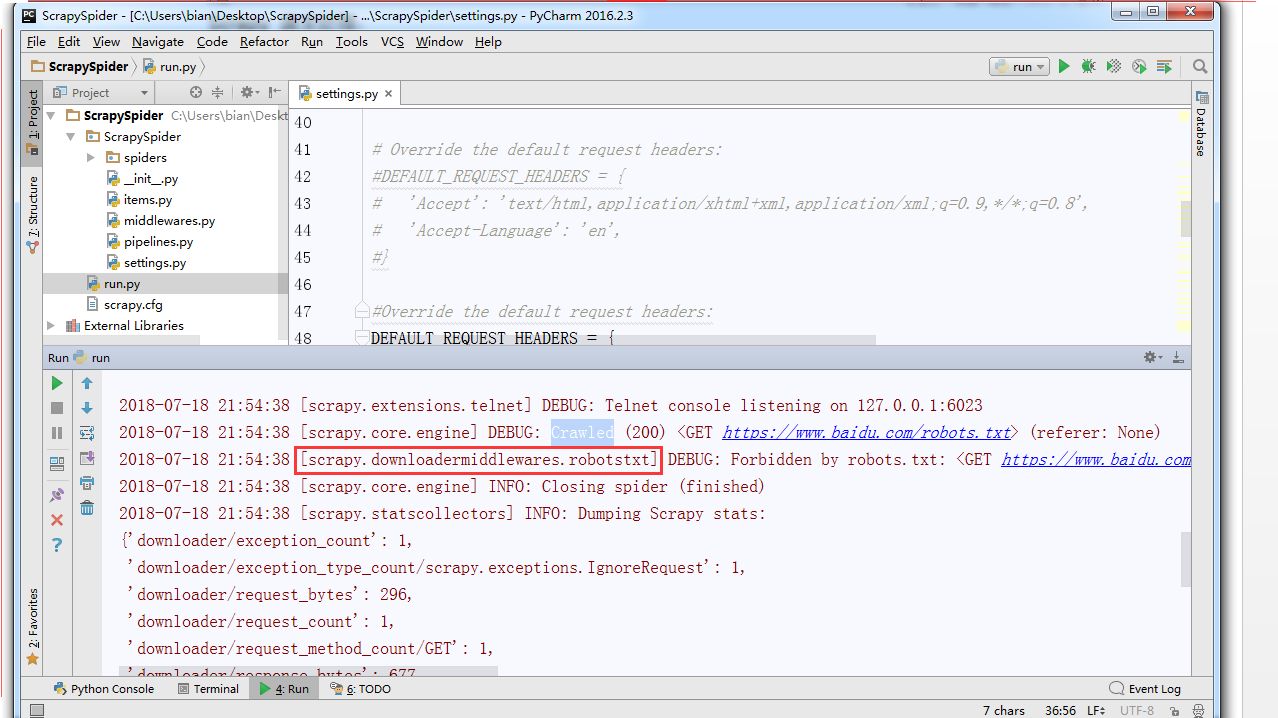
## 1.3 Scrapy 请求处理

### 1.3.1 Scrapy默认请求头配置

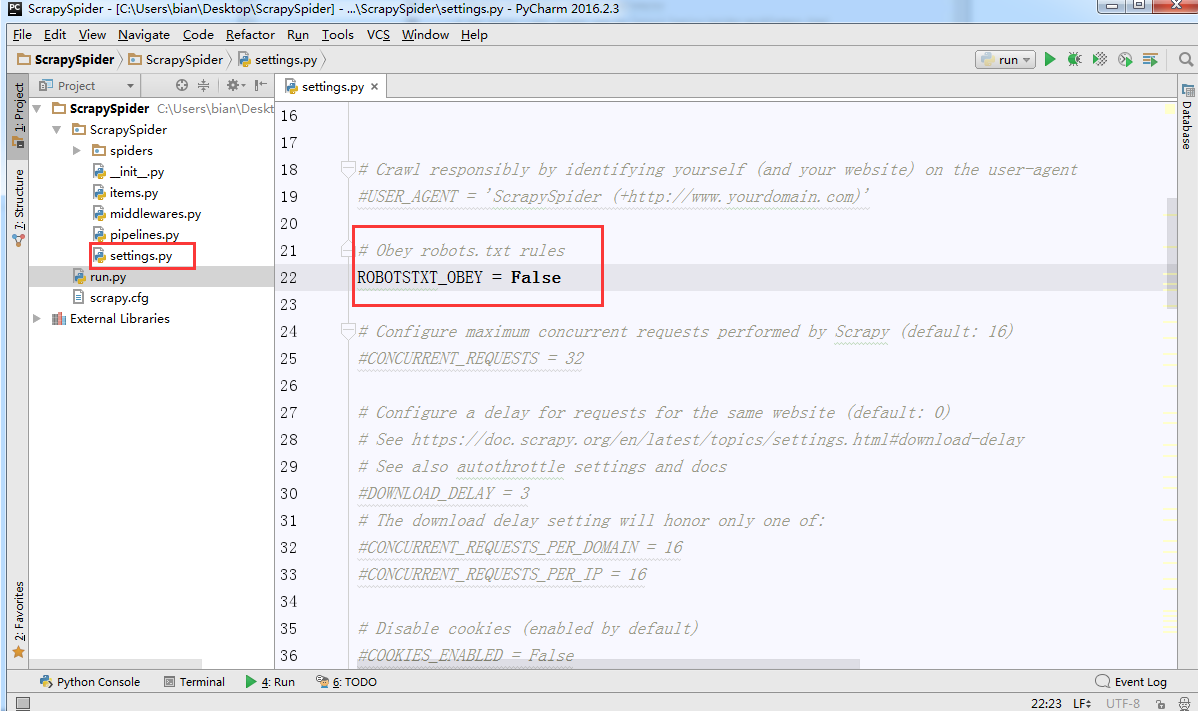


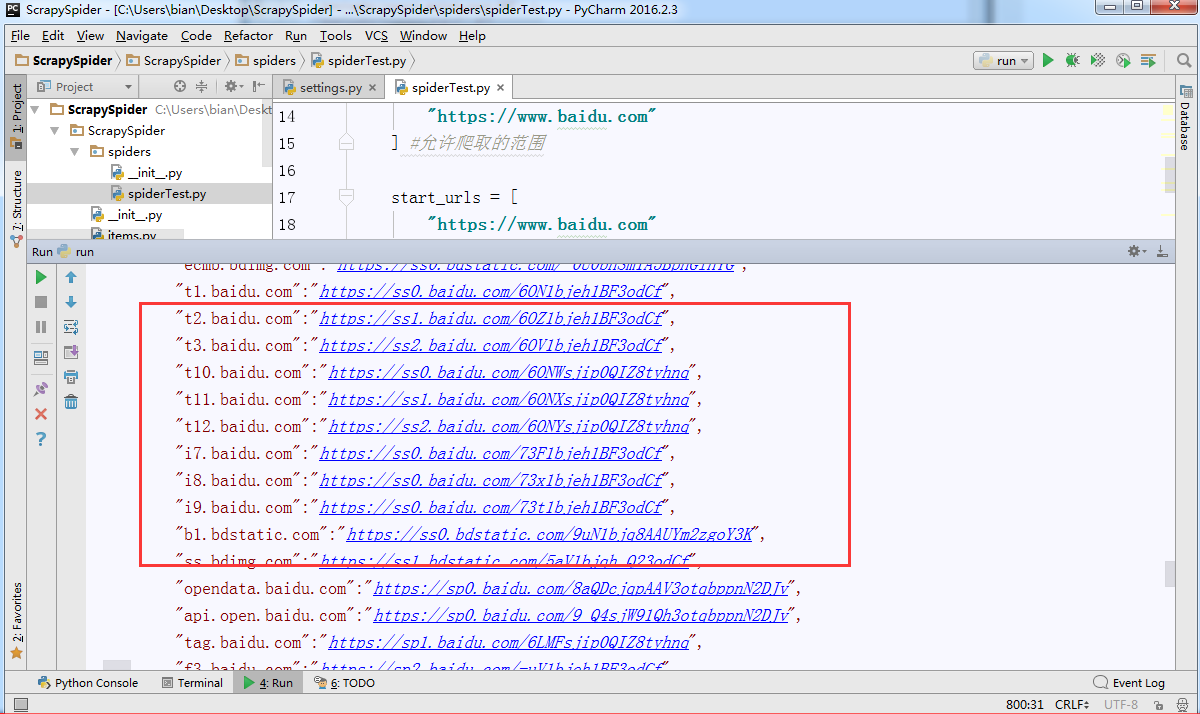
运行结果如下：

注意：有反爬虫协议



不启动机器人协议





加入我们只是爬一个百度这样写是没有问题的，但是如果爬小说网站，请求头有变化，那么我们刚才配置默认请求头就不够用了，所以我们需要配置自己的请求

### 1.3.2 Request对象

在这里我们了解一个scrapy的request对象

首先request对象是由scrapy.Request方法创建

我们来认识一下request当中的参数

# url 请求的url

# callback=None 请求的回调函数，默认是parse

# method='GET', 请求的方式

# headers=None, 请求的头部

# body=None, 请求的正文，可以是bytes或者str

#cookies=None, 请求携带的cookie

# meta=None, request post请求，数据

# encoding='utf-8', 字符集 url和body的字符集

# priority=0, 优先级，优先级越高，越先下载

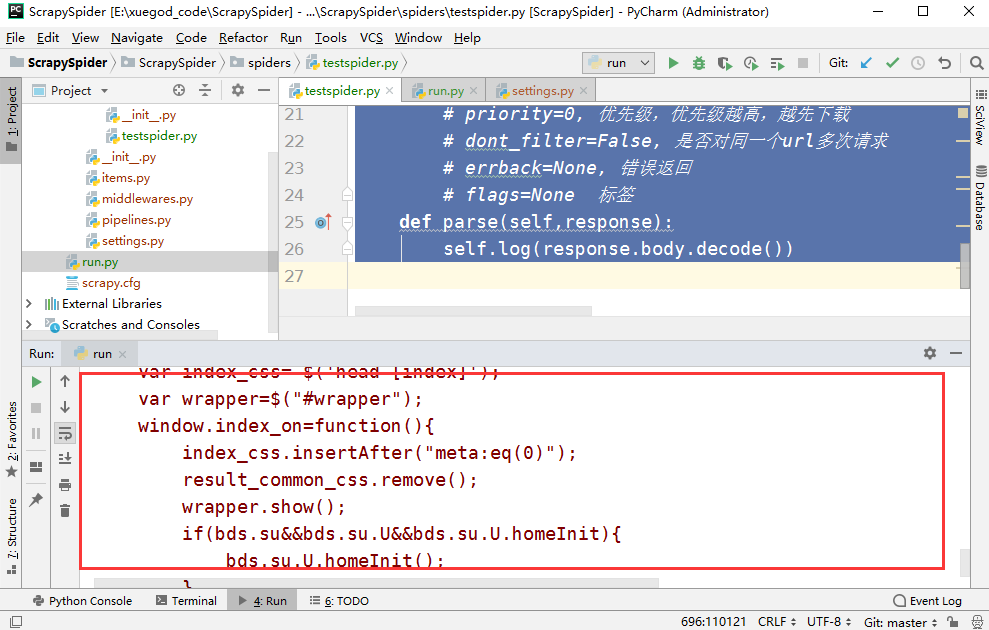
# dont\_filter=False, 是否对同一个url多次请求

# errback=None, 错误返回

# flags=None 标签

我们重构的代码  
import scrapy  
from scrapy import Request  
class TestSpider(scrapy.spiders.Spider):  
 name = "baiduSpider"  
  
 def start\_requests(self):  
 url = "https://www.baidu.com"  
 headers = {  
 'User-Agent': 'Mozilla/1.0 (Windows NT 6.1; WOW64) AppleWebKit/137.36 (KHTML, like Gecko) Chrome/67.0.3396.99 Safari/137.36'  
 }  
 yield Request(url,headers = headers)  
 *# url 请求的url  
 # callback=None 请求的回调函数，默认是parse  
 # method='GET', 请求的方式  
 # headers=None, 请求的头部  
 # body=None, 请求的正文，可以是bytes或者str  
 #cookies=None, 请求携带的cookie  
 # meta=None, request post请求，数据  
 # encoding='utf-8', 字符集 url和body的字符集  
 # priority=0, 优先级，优先级越高，越先下载  
 # dont\_filter=False, 是否对同一个url多次请求  
 # errback=None, 错误返回  
 # flags=None 标签* def parse(self,response):  
 self.log(response.body.decode())

运行结果如下：



实战：利用爬虫框架scrapy ，爬取JavaScript动态加载网页，将图片下载到本地。

第一步： 创建项目

第二步： 配置文件settings.py，关闭robots协议

第三步： 编写Item.py文件

第四步： spider的编写

第五步： 执行scrapy

第一步：

Scrapy startproject so\_image

cd so\_image

Scrapy genspider images image.so.com

第二步：

ROBOTSTXT\_OBEY = False

ITEM\_PIPELINES = {  
 *# 'so\_images.pipelines.SoImagesPipeline': 300,* 'scrapy.pipelines.images.ImagesPipeline':1,  
}  
IMAGES\_STORE='downlaod\_iamges'

第三步：

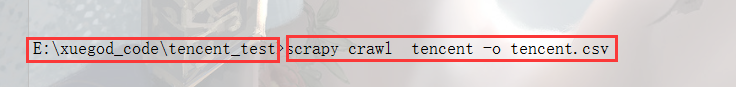
class SoImagesItem(scrapy.Item):  
 *# define the fields for your item here like:* image\_urls = scrapy.Field()  
 pass

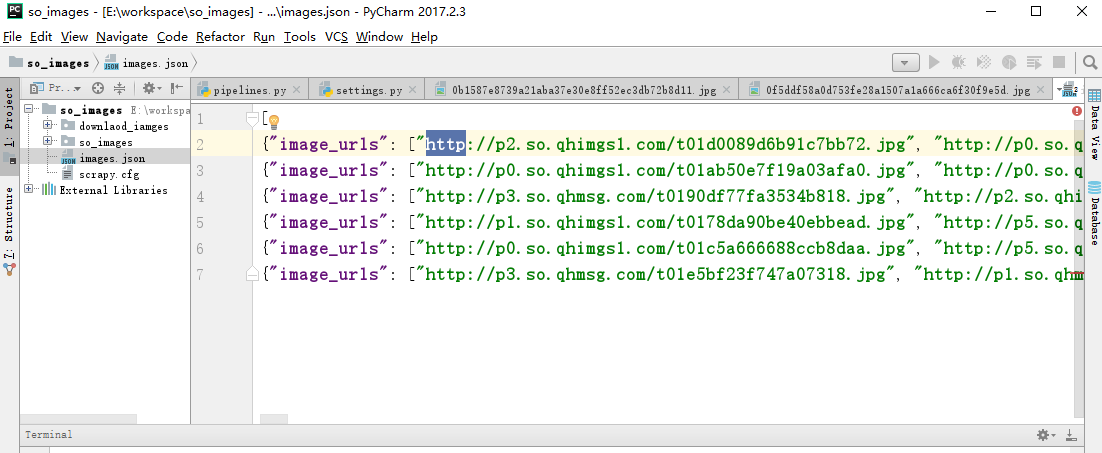
第四步：

spiders.py中编写

import scrapy  
import json  
from ScrapySpider.items import ScrapyspiderItem  
class ImagesSpider(scrapy.Spider):  
 BASE\_URL = 'http://image.so.com/zj?ch=art&sn=%s&listtype=new&temp=1'  
 name = 'images'  
 start\_index = 0  
 allowed\_domains = ['image.so.com']  
 start\_urls = ['http://image.so.com/zj?ch=art&sn=0&listtype=new&temp=1']  
 MAX\_DOWNLOAD\_NUM = 100  
 def parse(self, response):  
 infos = json.loads(response.body.decode())  
 *# images = ScrapyspiderItem()  
 # images['image\_urls']=[info['qhimg\_url']for info in infos['list']]  
 # yield images* yield {'image\_urls':[info['qhimg\_url']for info in infos['list']]}  
 self.start\_index += infos['count']  
 if infos['count'] > 0 and self.start\_index < self.MAX\_DOWNLOAD\_NUM:  
 yield scrapy.Request(self.BASE\_URL%self.start\_index)

第五步：





总结：

1.1 Scrapy 介绍

1.2 Scrapy 基本命令与结构

1.3 Scrapy 请求处理