

简介

什么是框架？

所谓的框，其实说白了就是一个【项目的半成品】，该项目的半成品需要被集成了各种功能且具有较强的通用性。

Scrapy是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，非常出名，非常强悍。所谓的框架就是一个已经被集成了各种功能（高性能异步下载，队列，分布式，解析，持久化等）的具有很强通用性的项目模板。对于框架的学习，重点是要学习其框架的特性、各个功能的用法即可。

初期如何学习框架？

只需要学习框架集成好的各种功能的用法即可！前期切勿钻研框架的源码！

安装

Linux/mac系统：

```
pip install scrapy (任意目录下)
```

Windows系统：

```
a. pip install wheel (任意目录下)
```

b. 下载twisted文件，下载网址如下：

<http://www.lfd.uci.edu/~gohlke/pythonlibs/#twisted>

c. 终端进入下载目录，执行 `pip install Twisted-17.1.0-cp35-cp35m-win_amd64.whl`

注意：如果该步骤安装出错，则换一个版本的whl文件即可

d. `pip install pywin32`（任意目录下）

e. `pip install scrapy`（任意目录下）

如果安装好后，在终端中录入scrapy指令按下回车，如果没有提示找不到该指令，则表示安装成功

基本使用

- 创建项目
 - `scrapy startproject firstBlood`项目名称
 - 项目的目录结构：

firstBlood # 项目所在文件夹, 建议用pycharm
打开该文件夹

```
├── firstBlood      # 项目跟目录
│   ├── __init__.py
│   ├── items.py    # 封装数据的格式
│   ├── middlewares.py # 所有中间件
│   ├── pipelines.py # 所有的管道
│   ├── settings.py # 爬虫配置信息
│   └── spiders     # 爬虫文件夹, 稍后
                    # 里面会写入爬虫代码
└── scrapy.cfg      # scrapy项目配置信息, 不要删它, 别动它, 善待它.
```

- 创建爬虫爬虫文件:
 - cd project_name (进入项目目录)
 - scrapy genspider 爬虫文件的名称 (自定义一个名字即可) 起始url
 - (例如: scrapy genspider first www.xxx.com)
 - 创建成功后, 会在爬虫文件夹下生成一个py的爬虫文件
- 编写爬虫文件
 - 理解爬虫文件的不同组成部分

```
import scrapy
```

```

class FirstSpider(scrapy.Spider):
    #爬虫名称：爬虫文件唯一标识：可以使用该变量的值
    #来定位到唯一的一个爬虫文件
    name = 'first' #无需改动
    #允许的域名： scrapy只可以发起百度域名下的网络
    #请求
    # allowed_domains = ['www.baidu.com']
    #起始的url列表： 列表中存放的url可以被scrapy
    #发起get请求
    start_urls =
[ 'https://www.baidu.com/' , 'https://www.sogou.com' ]

    #专门用作于数据解析
    #参数response： 就是请求之后对应的响应对象
    #parse的调用次数， 取决于start_urls列表元素的
    #个数
    def parse(self, response):
        print( '响应对象为： ', response)

```

- 配置文件修改:settings.py
 - 不遵从robots协议： ROBOTSTXT_OBEY = False
 - 指定输出日志的类型： LOG_LEVEL = 'ERROR'
 - 指定UA： USER_AGENT = 'Mozilla/5.0 (Macintosh;

Intel Mac OS X 10_15_7) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/98.0.4758.109
Safari/537.36'

- 运行项目

`scrapy crawl` 爬虫名称 : 该种执行形式会显示执行的日志信息 (推荐)

数据解析

- 注意, 如果终端还在第一个项目的文件夹中, 则需要在终端中执行`cd ../`返回到上级目录, 在去新建另一个项目。
- 新建数据解析项目:
 - 创建工程: `scrapy startproject` 项目名称
 - `cd` 项目名称
 - 创建爬虫文件: `scrapy genspider` 爬虫文件名 www.xxx.com
- 配置文件的修改: `settings.py`
 - 不遵从robots协议: `ROBOTSTXT_OBEY = False`
 - 指定输出日志的类型: `LOG_LEVEL = 'ERROR'`
 - 指定UA: `USER_AGENT = 'Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/98.0.4758.109 Safari/537.36'`

- 编写爬虫文件：spiders/duanzi.py

```
import scrapy

class DuanziSpider(scrapy.Spider):
    name = 'duanzi'
    # allowed_domains = ['www.xxx.com']
    #对首页进行网络请求
    #scrapy会对列表中的url发起get请求
    start_urls =
[ 'https://ishuo.cn/duanzi' ]

    def parse(self, response):
        #如何获取响应数据
        #调用xpath方法对响应数据进行xpath形式的数据解析
        li_list = response.xpath('//*
[@id="list"]/ul/li')
        for li in li_list:
            # content =
li.xpath('./div[1]/text()')[0]
            # title =
li.xpath('./div[2]/a/text()')[0]
            # #<Selector
xpath='./div[2]/a/text()' data='一年奔波，尘
缘遇了谁'>
```

print(title)#selector的对象, 且我们想要的字符串内容存在于该对象的data参数里

#解析方案1:

```
# title =  
li.xpath('./div[2]/a/text()')[0]
```

```
# content =  
li.xpath('./div[1]/text()')[0]
```

#extract()可以将selector对象中data参数的值取出

```
# print(title.extract())  
# print(content.extract())
```

#解析方案2:

#title和content为列表, 列表只要一个列表元素

```
title =  
li.xpath('./div[2]/a/text()')
```

```
content =  
li.xpath('./div[1]/text()')
```

#extract_first()可以将列表中第0个列表元素表示的selector对象中data的参数值取出

```
print(title.extract_first())  
print(content.extract_first())
```

持久化存储

两种方案：

- 基于终端指令的持久化存储
- 基于管道的持久化存储（推荐）

基于终端指令的持久化存储

- 只可以将parse方法的返回值存储到指定后缀的文本文件中。
- 编码流程：
 - 在爬虫文件中，将爬取到的数据全部封装到parse方法的返回值中

```
import scrapy

class DemoSpider(scrapy.Spider):
    name = 'demo'
    # allowed_domains = ['www.xxx.com']
    start_urls =
[ 'https://ishuo.cn/duanzi' ]

    def parse(self, response):
        # 如何获取响应数据
```



```

        # 调用xpath方法对响应数据进行xpath形式的数据解析
        li_list = response.xpath('//*
[@id="list"]/ul/li')
        all_data = []#爬取到的数据全部都存储到了该列表中

        for li in li_list:
            title =
li.xpath(' ./div[2]/a/text()').extract_first()

            content =
li.xpath(' ./div[1]/text()').extract_first()

            #将段子标题和内容封装成parse方法的返回

            dic = {
                'title':title,
                'content':content
            }
            all_data.append(dic)

        return all_data

```

- 将parse方法的返回值存储到指定后缀的文本文件中:
 - scrapy crawl 爬虫文件名称 -o duanzi.csv

- 总结：
 - 优点：简单，便捷
 - 缺点：局限性强
 - 只可以将数据存储到文本文件无法写入数据库
 - 存储数据文件后缀是指定好的，通常使用.csv
 - 需要将存储的数据封装到parse方法的返回值中

基于管道实现持久化存储

优点：极大程度的提升数据存储的效率

缺点：编码流程较多

编码流程

1.在爬虫文件中进行数据解析

```
def parse(self, response):
    # 如何获取响应数据
    # 调用xpath方法对响应数据进行xpath形式的数据解析
    li_list = response.xpath('//*
[@id="list"]/ul/li')
    all_data = [] # 爬取到的数据全部都存储到了该列表中
    for li in li_list:
        title =
li.xpath('./div[2]/a/text()').extract_first()
        content =
li.xpath('./div[1]/text()').extract_first()
```

2.将解析到的数据封装到Item类型的对象中

- 2.1 在items.py文件中定义相关的字段

```
class SavedataproItem(scrapy.Item):
    # define the fields for your item here
    like:
    # name = scrapy.Field()
    #爬取的字段有哪些，这里就需要定义哪些变量存储爬
    取到的字段
    title = scrapy.Field()
    content = scrapy.Field()
```

- 2.2 在爬虫文件中引入Item类，实例化item对象，将解析到的数据存储到item对象中

```

def parse(self, response):
    from items import SavedataproItem #
    导入item类

    # 如何获取响应数据
    # 调用xpath方法对响应数据进行xpath形式的数据解析

    li_list = response.xpath('//*
[@id="list"]/ul/li')
    all_data = [] # 爬取到的数据全部都存储
    到了该列表中

    for li in li_list:
        title =
li.xpath('./div[2]/a/text()').extract_first
()

        content =
li.xpath('./div[1]/text()').extract_first()
        #实例化一个item类型的对象
        item = SavedataproItem()
        #通过中括号的方式访问item对象中的两
        个成员，且将解析到的两个字段赋值给item对象的两个成
        员即可

        item['title'] = title
        item['content'] = content

```

3.将item对象提交给管道

```
#将存储好数据的item对象提交给管道
yield item
```

4.在管道中接收item类型对象(pipelines.py就是管道文件)

- 管道只可以接收item类型的对象，不可以接收其他类型对象

```
class SavedataproPipeline:
    #process_item用来接收爬虫文件传递过来的item对象
    #item参数，就是管道接收到的item类型对象
    def process_item(self, item, spider):
        print(item)
        return item
```

5.在管道中对接收到的数据进行任意形式的持久化存储操作

- 可以存储到文件中也可以存储到数据库中

```
# Define your item pipelines here
#
# Don't forget to add your pipeline to the
ITEM_PIPELINES setting
# See:
https://docs.scrapy.org/en/latest/topics/item-pipeline.html
```

```
# useful for handling different item types
with a single interface
from itemadapter import ItemAdapter

class SavedataproPipeline:
    #重写父类的方法
    fp = None
    def open_spider(self, spider):
        print('我是open_spider方法，我在项目开始
运行环节，只会被执行一次！')
        self.fp =
open('duanzi.txt', 'w', encoding='utf-8')
        #process_item用来接收爬虫文件传递过来的item对
象
        #item参数，就是管道接收到的item类型对象
        #process_item方法调用的次数取决于爬虫文件给其提
交item的次数
    def process_item(self, item, spider):
        #item类型的对象其实就是一个字典
        # print(item)
        #将item字典中的标题和内容获取
        title = item['title']
        content = item['content']
        self.fp.write(title+':'+content+'\n')
        print(title, ':爬取保存成功！')
```

```
        return item

    def close_spider(self, spider):
        print('在爬虫结束的时候会被执行一次!')
        self.fp.close()
```

6.在配置文件中开启管道机制

- 注意：默认情况下，管道机制是没有被开启的，需要在配置文件中手动开启
- 在setting.py中把ITEM_PIPELINES解除注释就表示开启了管道机制