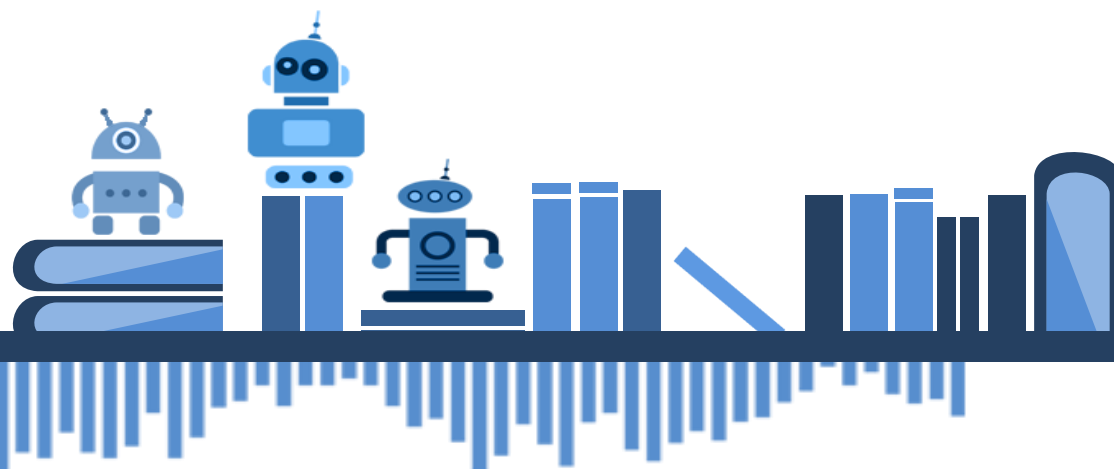




Scrapy框架

主讲老师：张涛



1

Scrapy介绍

2

第一个Scrapy网络爬虫

3

使用Spider实现数据的爬取

4

使用Item实现数据的封装

5

使用Pipeline实现数据的处理

6

案例-实现图片的爬取

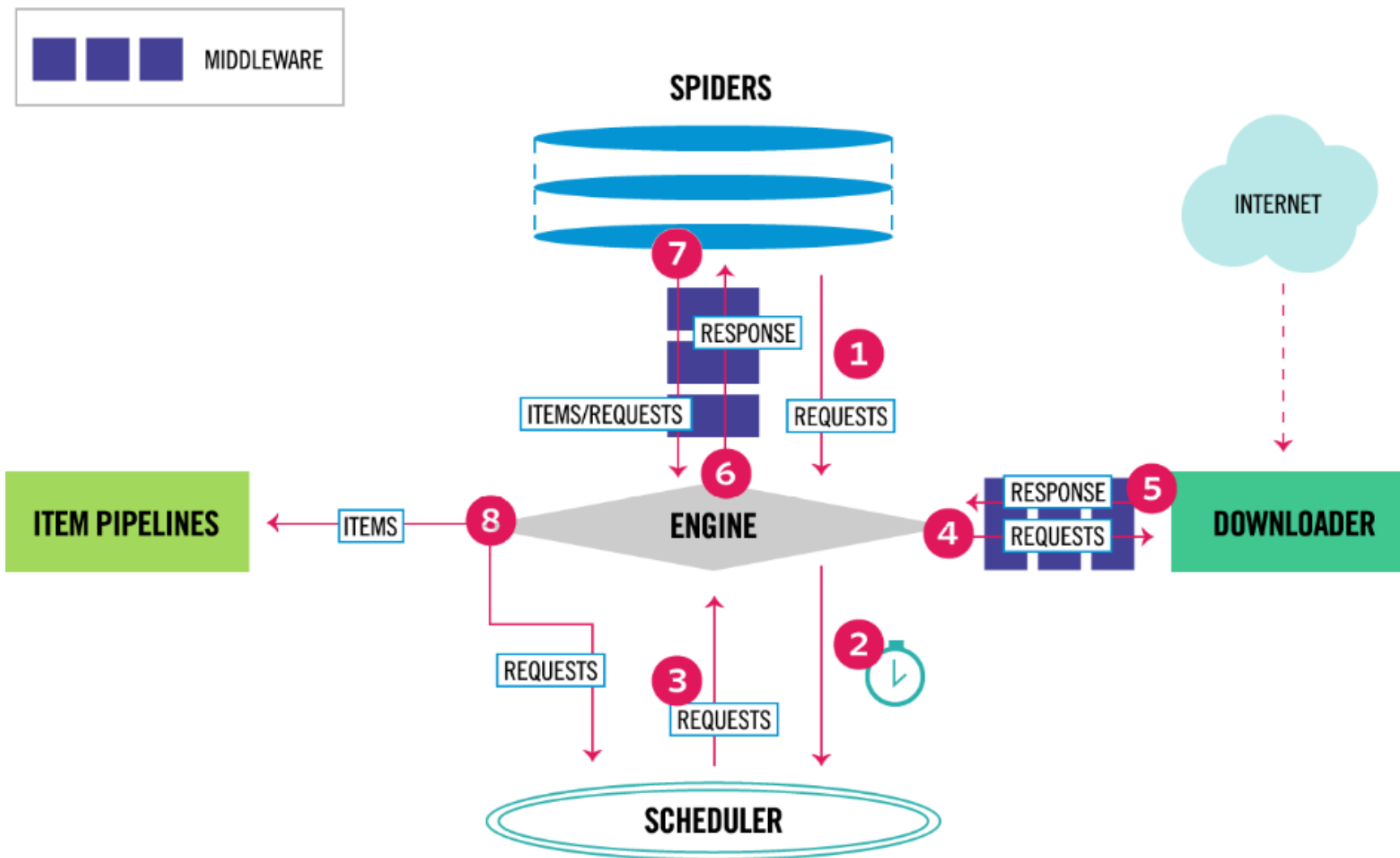
目录

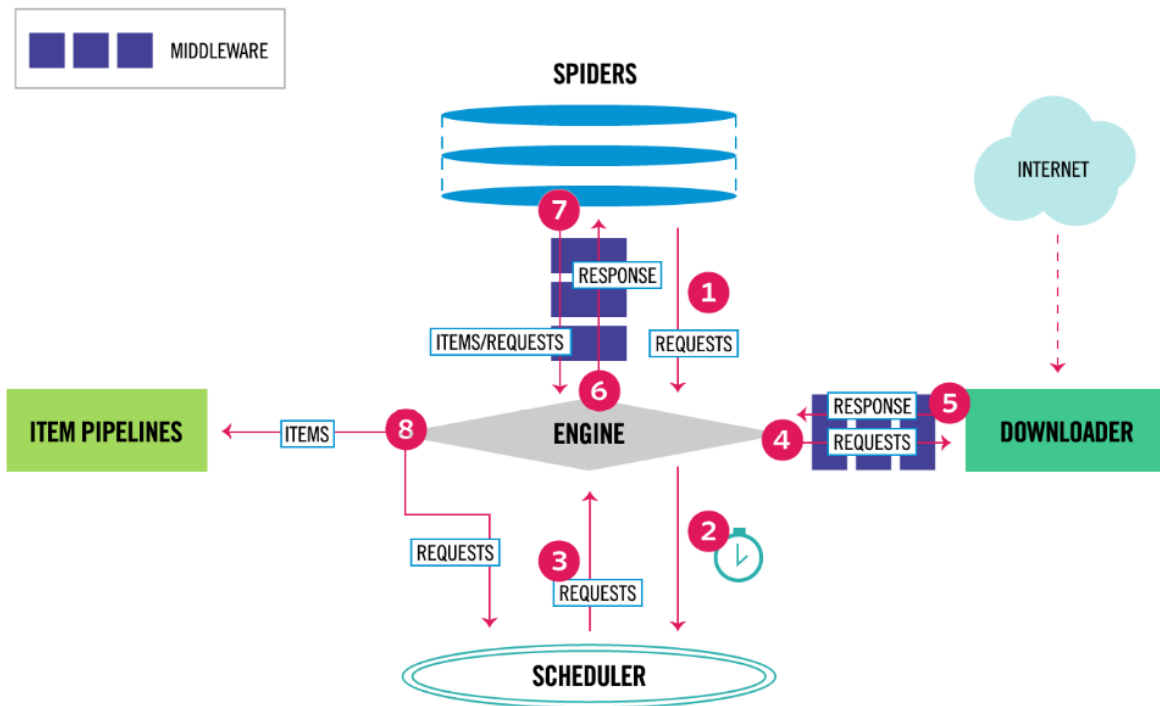


Scrapy = Scratch Python

- 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。可以应用在包括数据挖掘，信息处理或存储历史数据等一系列的程序中。
- 基于python的快速、高层次的屏幕抓取和Web抓取框架，用于抓取Web站点并从页面中提取结构化的数据
- Scrapy用途广泛，可以用于数据挖掘、监测和自动化测试。







◆ 引擎 (ENGINE)

整个系统的大脑，指挥其他组件协同工作。

◆ 调度器 (SCHEDULER)

调度器接收引擎发过来的请求，压入队列中。

◆ 爬虫 (SPIDERS)

最核心的组件，用于从特定的网页中提取自己需要的信息。

◆ 项目管道 (ITEM PIPELINES)

负责处理爬虫从网页中抽取的实体。

◆ 下载器中间件 (Downloader Middlewares)

介于引擎和下载器之间的组件。

◆ 爬虫中间件 (Spider Middlewares)

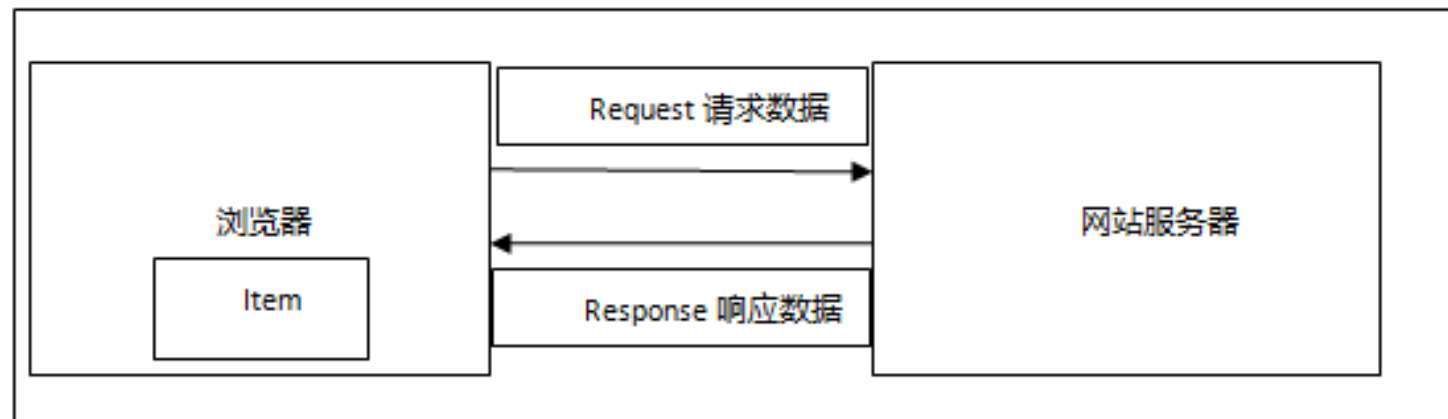
介于引擎和爬虫之间的组件。



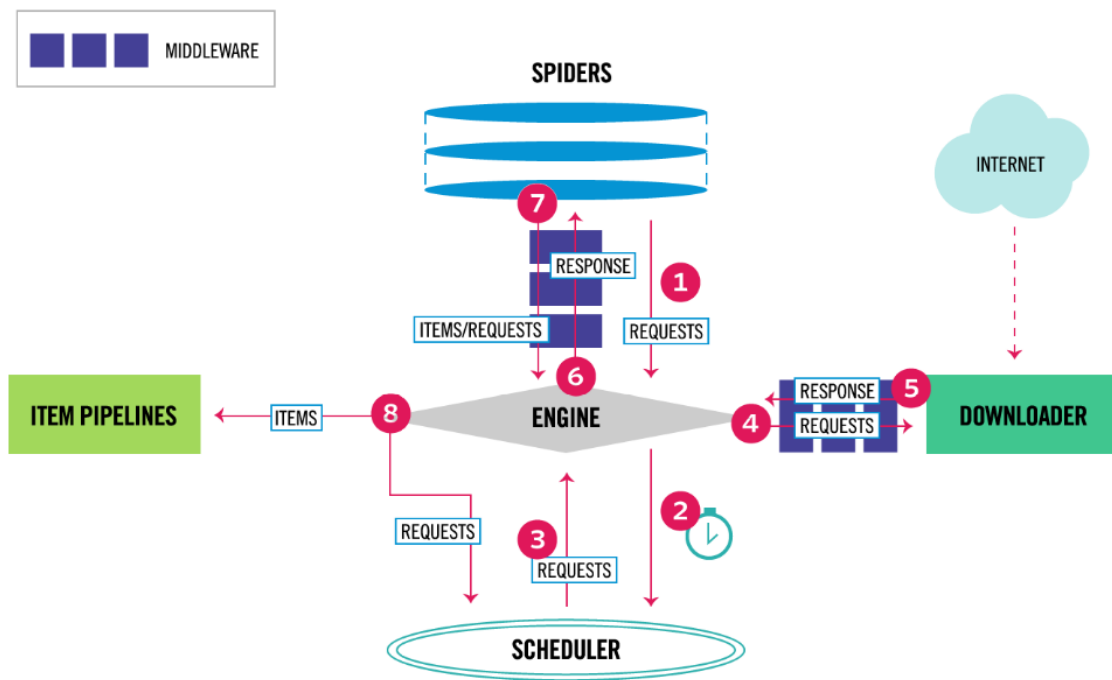
Scrapy框架结构中传递和处理的数据主要有：

- ◆ 向网站服务器发送的请求数据（请求的内容见第一章的HTTP请求）
- ◆ 网站服务器返回的响应数据（响应的内容见第一章的HTTP响应）
- ◆ 解析后的结构数据（类似于字典）

Scrapy中定义的Request和Response类用于保存请求和响应数据，
Item类保存解析后的结构数据，
如右图所示。



执行流程



第①步：爬虫（Spider）构造一个请求（Request）对象，提交给引擎（ENGINE）。

第②步：引擎将请求安排给调度器。

第③步：引擎从调度器获取即将要执行的请求。

第④步：将请求发送给下载器下载页面。

第⑤步：下载器生成一个响应（Response）对象并将其发送给引擎。

第⑥步：引擎将响应对象发送给爬虫（Spider）进行处理。

第⑦步：爬虫将抽取到的一条数据实体（Item）和新的请求（如下一页的链接）发送给引擎。

第⑧步：引擎将从爬虫获取到的Item发送给项目管道（ITEM PIPELINES），项目管道实现数据持久化等功能；将新的请求发送给调度器，再从第②步开始重复执行，直到调度器中没有更多的请求，引擎关闭该网站。





Scrapy框架的安装非常简单，使用pip命令：

```
pip install scrapy
```

安装完后验证安装是否成功，输入如下代码，执行。

```
import scrapy  
print(scrapy.version_info)
```



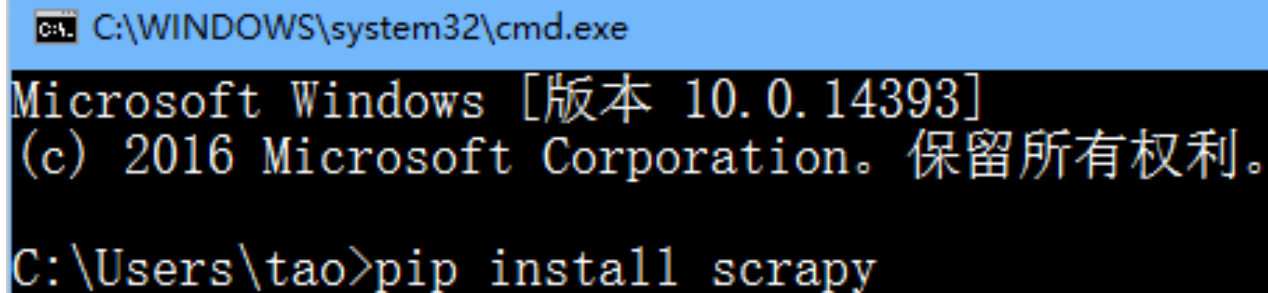


Scrapy框架的安装非常简单，使用pip命令：

```
pip install scrapy
```

安装完后验证安装是否成功，输入如下代码，执行。

```
import scrapy  
print(scrapy.version_info)
```



```
C:\WINDOWS\system32\cmd.exe  
Microsoft Windows [版本 10.0.14393]  
(c) 2016 Microsoft Corporation。保留所有权利。  
C:\Users\tao>pip install scrapy
```





1. 执行命令，生成一个Scrapy项目

scrapy startproject 项目名

```
C:\WINDOWS\system32\cmd.exe
Microsoft Windows [版本 10.0.17134.471]
(c) 2018 Microsoft Corporation。保留所有权利。

C:\Users\tao>scrapy startproject booksScrapy
New Scrapy project 'booksScrapy', using template directory
scrapy\templates\project', created in:
    C:\Users\tao\booksScrapy

You can start your first spider with:
    cd booksScrapy
    scrapy genspider example example.com

C:\Users\tao>_
```





2.运行PyCharm, 打开项目

- ▼ **spiderTest** D:\spiderTest
 - ▼ **booksScrapy** 该项目的python模块。之后将在此加入代码。
 - ▼ **booksScrapy** 放置spider代码的目录
 - > **spiders**
 - __init__.py**
 - items.py** 保存爬取到的数据的容器
 - middlewares.py**
 - pipelines.py** 持久化实体
 - settings.py** 项目的设置文件
 - scrapy.cfg** 项目配置文件



爬虫 (Spider) 组件

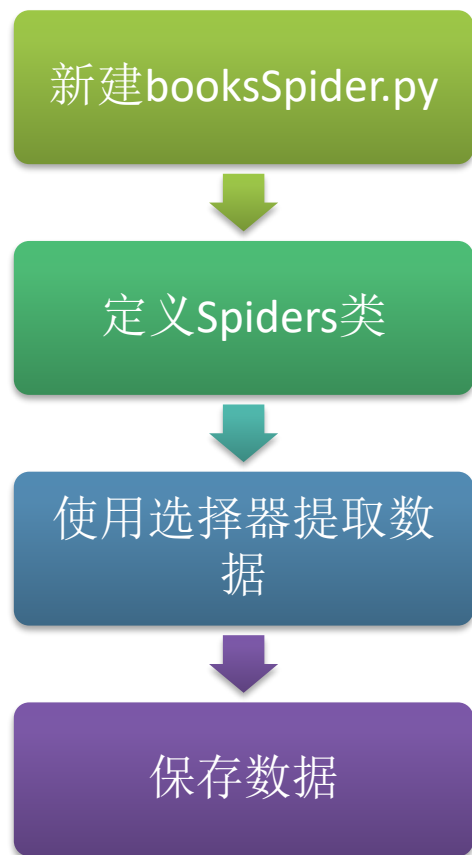
Scrapy核心组件。主要功能是封装一个发送给网站服务器的HTTP请求，以及解析网站返回的网页、提取数据。

Spider组件解决了以下问题：

- ✓ 爬虫从哪个或哪些页面开始爬取。
- ✓ 对于一个已下载的页面，提取其中的哪些数据。
- ✓ 爬取完当前页面后，接下来爬取哪个或哪些页面。



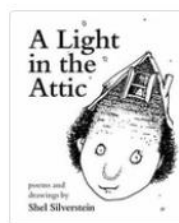
实现爬取网站的书籍信息，网址为：<http://books.toscrape.com/>



All products

1000 results - showing 1 to 20.

Warning! This is a demo website for web scraping purposes. Prices and ratings here were randomly assigned and have no real meaning.



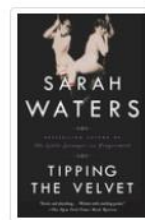
★★★★★

A Light in the ...

£51.77

✓ In stock

Add to basket



★★★★★

Tipping the Velvet

£53.74

✓ In stock

Add to basket



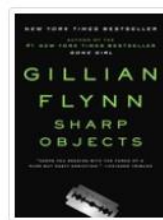
★★★★★

Soumission

£50.10

✓ In stock

Add to basket



★★★★★

Sharp Objects

£47.82

✓ In stock

Add to basket





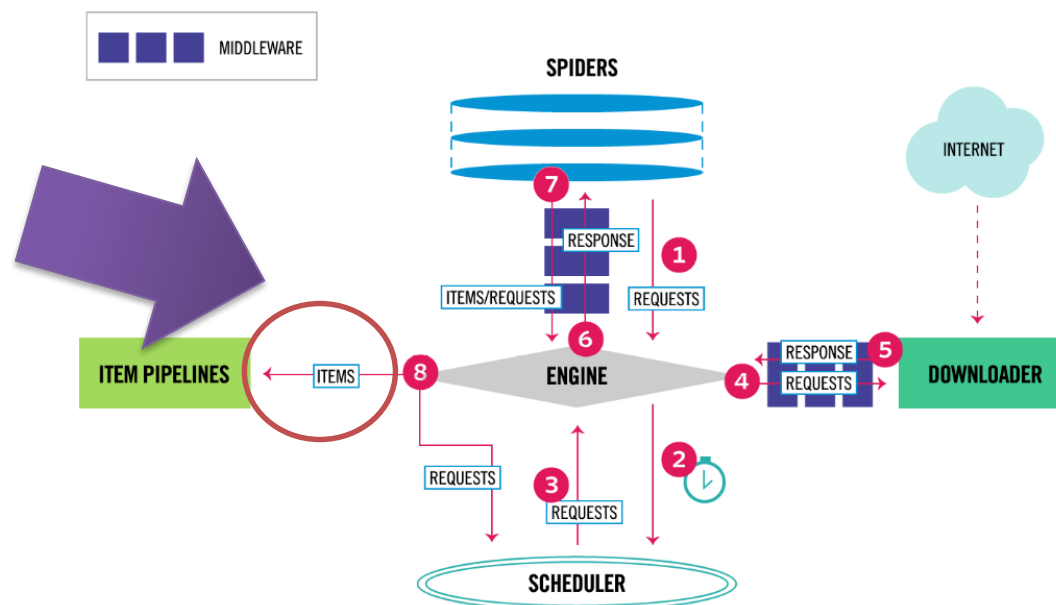
运行爬虫：

```
scrapy crawl book
```



为了收集抓取到的这些数据，Scrapy提供了一个简单的容器Item。Item对象是一个简单的容器，用于收集抓取到的数据，其提供了类似于字典（dictionary-like）的API并具有用于声明可用字段的简单语法。

Scrapy还定义了一个类Field，用于描述自定义数据类包含的字段。



继续完成网站的书籍信息爬取功能，网址为：<http://books.toscrape.com/>，使用Item封装数据

1、items.py源文件中，增加两个项目：

```
import scrapy
```

```
class BookscrapyItem(scrapy.Item):
```

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

```
    #书名
```

```
    name = scrapy.Field()
```

```
    #价格
```

```
    price = scrapy.Field()
```



继续完成网站的书籍信息爬取功能，网址为：<http://books.toscrape.com/>，使用Item封装数据

2、修改类booksSpider，将书名和价格保存于Item中：

```
from booksScrapy.items import BookscrapyItem
#解析数据的函数
def parse(self, response):
    item = BookscrapyItem() #定义Item类的对象，用于保存一条数据
    li_selector = response.xpath("//ol[@class='row']/li")
    for one_selector in li_selector:
        #获取书名
        name = one_selector.xpath("article/h3/a/@title").extract()[0]
        #价格
        price = one_selector.xpath("article/div[@class='product_price']/p[1]/text()").extract()[0]
        item["name"] = name
        item["price"] = price
    yield item
```



继续完成网站的书籍信息爬取功能，网址为：<http://books.toscrape.com/>，使用Item封装数据

3、运行爬虫：

```
scrapy crawl book -o books.csv
```

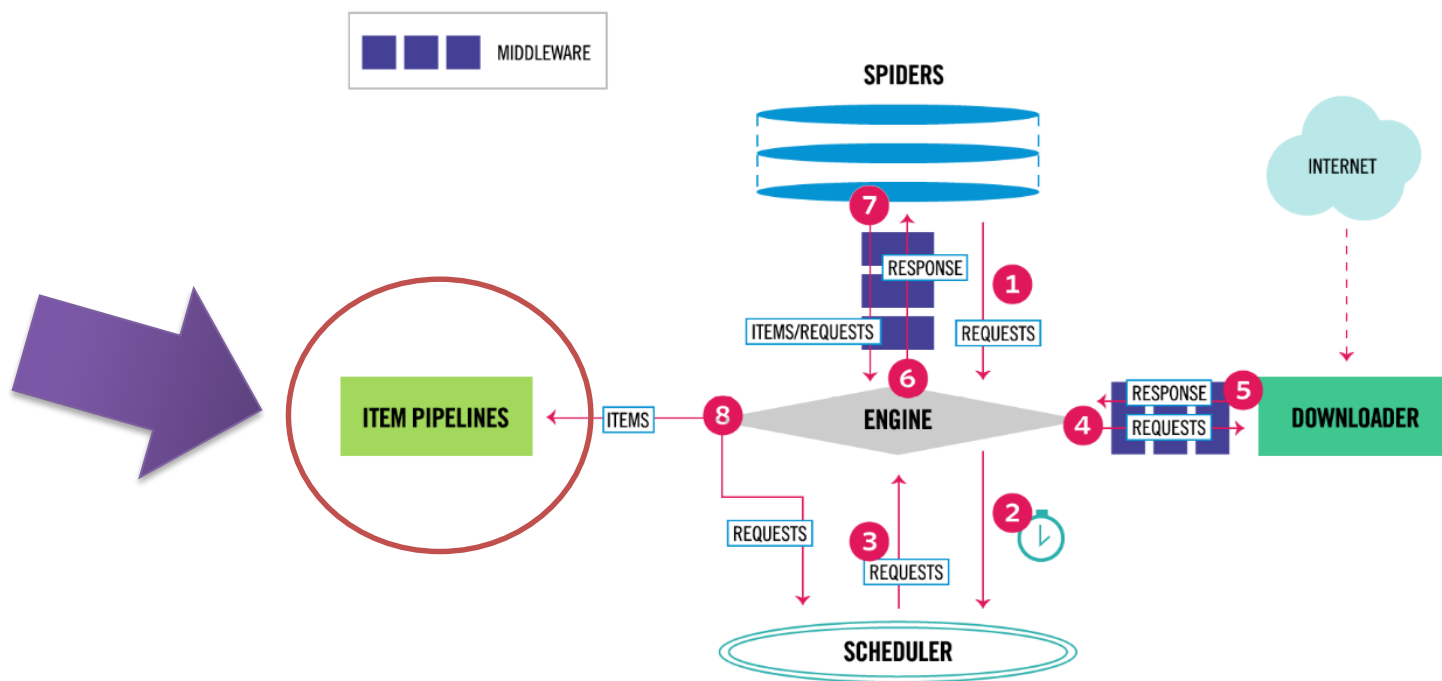
books.csv x	
1	name, price
2	A Light in the Attic, £51.77
3	Tipping the Velvet, £53.74
4	Soumission, £50.10
5	Sharp Objects, £47.82
6	Sapiens: A Brief History of Humankind, £54.23
7	The Requiem Red, £22.65
8	The Dirty Little Secrets of Getting Your Dream Job, £33.34
9	The Coming Woman: A Novel Based on the Life of the Infamous Feminist, Vi
10	The Boys in the Boat: Nine Americans and Their Epic Quest for Gold at the
11	The Black Maria, £52.15
12	Starving Hearts (Triangular Trade Trilogy, #1), £13.99
13	Shakespeare's Sonnets, £20.66
14	Set Me Free, £17.46



当我们通过Spider爬取数据，通过Item收集数据后，就需要对数据进行一些处理了，因为我们爬取到的数据并不一定是我们想要的最终数据，可能还需要进行数据的清洗以及验证数据的有效性。Scrapy中的Pipeline组件就用于数据的处理，一个Pipeline组件就是一个包含特定接口的类，通常只负责一种功能的数据处理，在一个项目中可以同时启用多个Pipeline。

以下是Pipeline的几种典型应用：

- ✓ 清洗数据
- ✓ 验证数据的有效性
- ✓ 过滤掉重复的数据
- ✓ 将数据存入数据库



继续完成网站的书籍信息爬取功能，网址为：<http://books.toscrape.com/>，使用Pipeline将数据持久化存储。

1、pipelines.py源文件中BooksscrapyPipeline类中，增加数据持久化功能代码：

```
class BooksscrapyPipeline(object):  
    def process_item(self, item, spider):  
        #对数据进行处理-写入到txt文件中  
        with open("mybooks.txt", "a", encoding="utf-8") as f:  
            oneStr = item["name"] + ";" + item["price"] + ";" + "\n"  
            f.write(oneStr)  
        return item
```





继续完成网站的书籍信息爬取功能，网址为：<http://books.toscrape.com/>，使用Pipeline将数据持久化存储。

2、在配置文件settings.py中，启用Pipeline（去掉注释）：

```
ITEM_PIPELINES = {  
    'booksScrapy.pipelines.BooksscrapyPipeline': 300,  
}
```



继续完成网站的书籍信息爬取功能，网址为：<http://books.toscrape.com/>，使用Pipeline将数据持久化存储。

3、运行爬虫：

scrapy crawl book

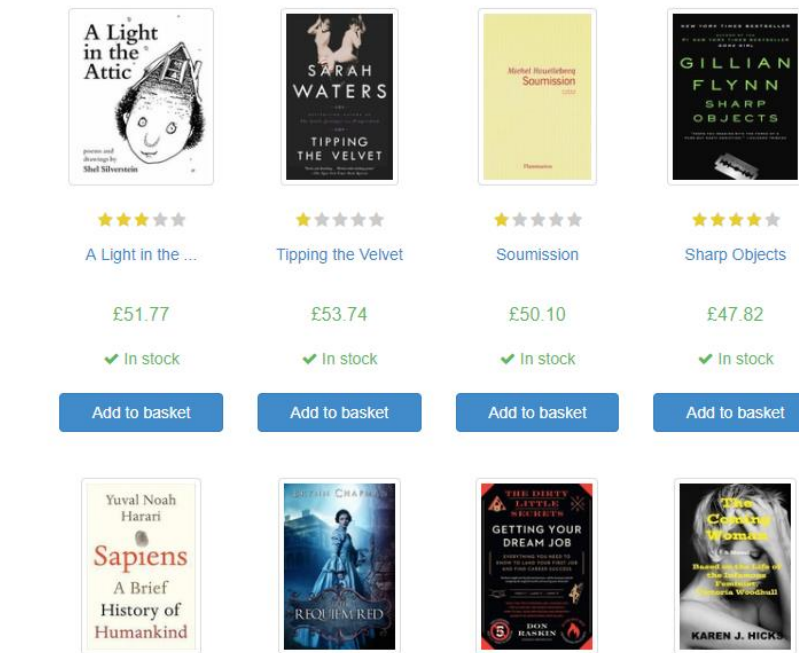
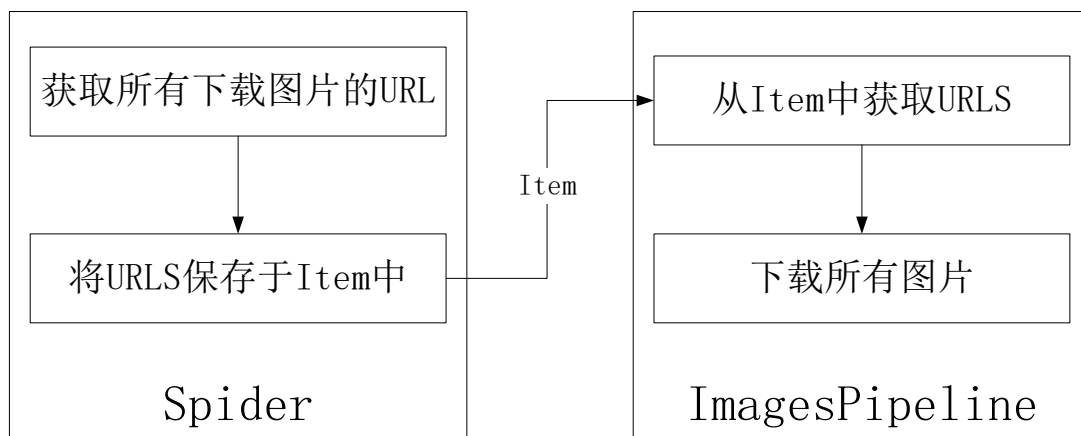
mybooks.txt ×	
1	A Light in the Attic;£51.77
2	Tipping the Velvet;£53.74
3	Soumission;£50.10
4	Sharp Objects;£47.82
5	Sapiens: A Brief History of Humankind;£54.23
6	The Requiem Red;£22.65
7	The Dirty Little Secrets of Getting Your Dream Job;£33.34
8	The Coming Woman: A Novel Based on the Life of the Infamous Feminist, Vi
9	The Boys in the Boat: Nine Americans and Their Epic Quest for Gold at th
10	The Black Maria;£52.15
11	Starving Hearts (Triangular Trade Trilogy, #1);£13.99
12	Shakespeare' s Sonnets;£20.66



如何将网站中展示书籍的图片下载到本地呢？

Scrapy提供了图片管道ImagesPipeline用于实现图片的下载。

执行流程为：



1、Item中增加保存图片url的字段:

```
import scrapy
```

```
class BookscrapyItem(scrapy.Item):
```

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

```
    #书名
```

```
    name = scrapy.Field()
```

```
    #价格
```

```
    price = scrapy.Field()
```

```
    #图片url地址
```

```
    img_url = scrapy.Field()
```



2、修改Spider组件booksSpider:

- ✓ 爬取图片url地址存入item中

```
class booksSpider(Spider):  
    name = 'book'  
    .....  
    #解析数据的函数  
    def parse(self,response):  
        .....  
        #图片url地址  
        url = one_selector.xpath("article/div[@class='image_container']/a/img/@src").extract()[0]  
        url = url.split("../")[-1]  
        url = "http://books.toscrape.com"+url  
        item["img_url"] = url  
        yield item
```



3、新建继承于ImagesPipeline的管道组件:

```
from scrapy.pipelines.images import ImagesPipeline # 下载图片的管道
from scrapy import Request
from scrapy.exceptions import DropItem # 异常
import logging
logger = logging.getLogger("SaveImagePipeline")
# 图片管道, 继承于ImagesPipeline
class SaveImagePipeline(ImagesPipeline):
    def get_media_requests(self, item, info): # 下载图片的请求
        yield Request(url = item["img_url"])

    def item_completed(self, results, item, info): # 判断是否正确下载
        if not results[0][0]:
            raise DropItem("下载失败")
        # 打印日志
        logger.debug("下载图片成功")

        return item

    def file_path(self, request, response=None, info=None):
        # 返回图片名称
        return request.url.split("/")[-1]
```



4、配置文件settings.py中设置配置项:

- ✓ 不遵守robots协议
- ✓ 设置用户代理USER_AGENT
- ✓ 设置图片下载路径
- ✓ 启用文件管道

```
# Obey robots.txt rules
ROBOTSTXT_OBEY = False
```

```
USER_AGENT = "Mozilla/5.0 (Windows NT 10.0;Win64; x64) " \
              "AppleWebKit/537.36 (KHTML, like Gecko) " \
              "Chrome/68.0.3440.106 Safari/537.36"
```

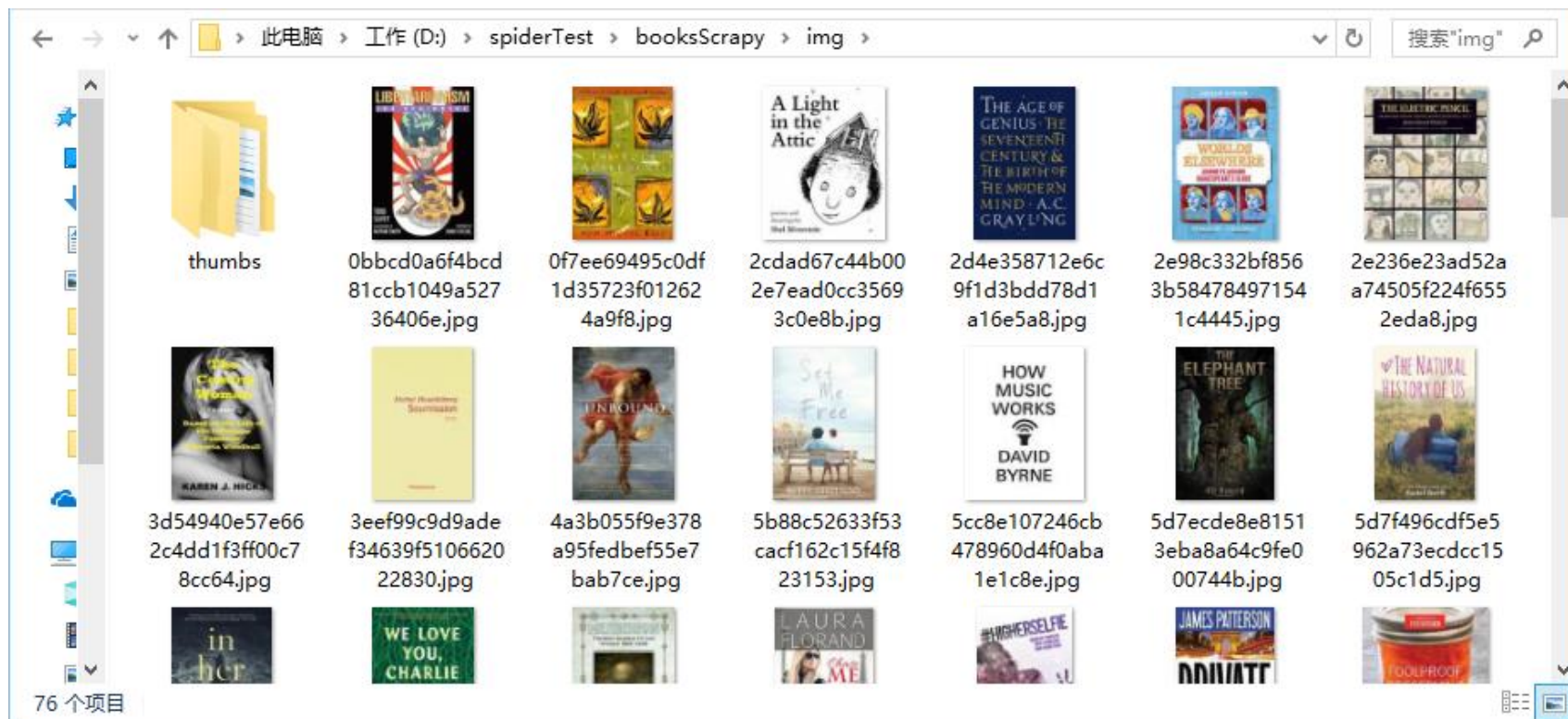
```
IMAGES_STORE = './img' #文件路径
IMAGES_THUMBS = {#缩略图
                  'small':(10,10),
                  'big':(50,50)
                }
#过滤掉尺寸过小的图片
IMAGES_MIN_WIDTH = 5
IMAGES_MIN_HEIGHT = 5
```

```
ITEM_PIPELINES = {
    'booksScrapy.pipelines.BookscrapyPipeline': 300,
    'booksScrapy.pipelines.SaveImagePipeline': 400,
}
```



5、运行爬虫:

scrapy crawl book





结束 谢谢收看

