第四周：网络爬虫之框架

第一讲：Scrapy 爬虫框架

1.安装

执行 pip install scrapy 命令。

安装后小测：执行 scrapy -h

2.Scrapy 爬虫框架结构

爬虫框架是实现爬虫功能的一个软件结构和功能组件集合。爬虫框架是一个半成品，能够帮助用户实现专业网络爬虫。

Scrapy 爬虫包括 5+2 个结构，如图 1 所示。

图 1

它包括三条主要的数据流路径如图中的箭头所示：

（1）从 SPIDERS 发送 REQUESTS 到 ENGINE 模块，然后到 SCHEDULER，SCHEDULER 负责对请求进行调度。

（2）SCHEDULER 发送 REQUESTS 到 ENGINE，再将 REQUESTS 传送到 DOWNLOADER 模块。然后 DOWNLOADER 返回相应通过 ENGINE 到 SPIDERS。

（3）从 SPIDERS 获取到路径（2）的 RESPONSE，处理之后发送 ITEMS/REQUESTS 到 ENGINE，然后 ITEMS 传递给 ITEM PIPELINES，REQUESTS 传递给 SCHEDULER。

这个框架的入口是 SPIDERS, 出口是 ITEM PIPELINES。其他三个模块用户都不需要关心，用户需要编写的是 SPIDERS 和 ITEM PIPELINES 的配置。

3.Scrapy 爬虫框架解析

（1）ENGINE：控制所有模块之间的数据流，根据条件触发事件，不需要用户修改。

（2）DOWNLOADER：根据请求下载网页，也不需要用户修改。

（3）SCHEDULER，对所有爬取请求进行调度管理，不需要用户修改。

（4）Downloader Middleware，设置目的是实施 Engine、Scheduler 和 Downloader 之间进行用户可配置的控制，可以修改、丢弃、新增请求或响应。

（5）Spider：解析 Downloader 返回的响应（Response），产生爬取项（scaped item）和额外的爬取请求（Request）。

（6）Item Pipelines：以流水线方式处理 Spider 产生的爬取项。它是由一组操作顺序组成，类似流水线， 操作是一个 Item Pipeline 类型。可能操作包括：清理、检验和查重爬取项中的 HTML 数据，将数据存储到数据库。

（7）Spider Middleware：目的是对请求和爬取项进行再处理，功能包括修改、丢弃、新增请求或爬取项，用户可以配置代码。

4.requests 库和 Scarpy 爬虫比较

库名称	相同点	不同点
requests	1.两者都可以进行页面请求和爬取。 2.两者可用性都好，文档丰富、入门简单。 3.两者都没有处理 js、提交表单、应对验证码等功能（可扩展）。	页面级爬虫；功能库；并发性考虑不足，性能较差；重点在于页面下载；定制灵活；上手十分简单。

Scrapy		网站级爬虫；框架；并发性好、性能较高；重点在于爬虫结构；一般定制灵活，深度定制困难；入门稍难。

5.Scrapy 爬虫常用命令

Scrapy 是为持续运行设计的专业爬虫框架，提供操作的是 Scrapy 命令行。它的格式如下：

scrapy <command>[options][args]

Scrapy 常用命令如表 1.1 所示。

表 1.1 Scrapy 常用命令

命令	说明	格式
startproject	**创建一个新工程**	**scrapy startproject <name> [dir]**
genspider	**创建一个爬虫**	**scrapy genspider [options] <name><domain>**
settings	获取爬虫配置信息	scrapy setting [options]
crawl	**运行一个爬虫**	**scrapy crawl<spider>**
list	列出工程中所有爬虫	scrapy list
shell	启动 URL 调试命令行	scrapy shell [url]

第二讲：Scrapy 爬虫基本使用

1.Scrapy 爬虫的第一个实例

Scrapy 爬虫的步骤：

（1）第一步：建立一个工程；

在命令行输入 scrapy startproject python123demo

生成的工程目录包括以下内容：

```
python123demo/                外层目录
  scrapy.cfg                  部署 Scrapy 爬虫的配置文件
  python123demo/              Scrapy 框架的用户自定义 Python 代码
    __init__.py               初始化脚本
    items.py                  Items 代码模板（继承类）
    middlewares.py            Middlewares 代码模板（继承类）
    pipelines.py              Pipelines 代码模板（继承类）
    settings.py               爬虫的配置文件
    spiders/                  Spiders 代码模板目录（继承类）
      __init__.py             初始文件，无需修改
      __pycache__/            缓存目录，无需修改
```

（2）第二步：在工程中产生一个 Scrapy 爬虫。

scrapy genspider demo python123.io

在 demo.py 文件中，parse()用于处理响应，解析内容形成字典，发现新的 URL 爬取请求。

（3）第三步：配置产生的 spider 爬虫

```
# -*- coding: utf-8 -*-
import scrapy
class DemoSpider(scrapy.Spider):
    name = "demo"
    #allowed_domains = ["python123.io"]
    start_urls = ['https://python123.io/ws/demo.html']
    def parse(self, response):#对返回页面进行解析并且进行操作的相关步骤
        fname = response.url.split('/')[-1]
        with open(fname, 'wb') as f:
            f.write(response.body)
        self.log('Saved file %s.' % name)
```

（4）运行爬虫，获取网页。

```
scrapy crawl demo
```

2.yield 关键字的使用

yield<--->生成器

生成器是一个不断产生值的函数。

包含 yield 语句的函数是一个生成器。

生成器每次产生一个值（yield 语句），函数被冻结，被唤醒后再产生一个值。

实例：

```
def gen(n):
    for i in range(n):
        yield i**2
for i in gen(5):
    print(i," ",end="")
```

生成器相比一次列出所有内容的优势：更节省存储空间；响应更迅速；使用更加灵活。

3.Scrapy 爬虫的基本使用

Scrapy 爬虫的使用步骤：

步骤 1：创建一个工程和 Spider 模板

步骤 2：编写 Spider

步骤 3：编写 Item Pipeline

步骤 4：优化配置策略

Scrapy 爬虫的数据类型：

Request 类：class scrapy.http.Request()

Request 对象表示一个 HTTP 请求。由 Spider 生成，由 Donwnloader 执行。它包括 6 个属性或方法：

表 2.1 Request 类的属性或方法

属性或方法	说明
.url	Request 对应的请求 url 地址
.method	对应的请求方法，‘GET’’POST’等
.headers	字典类型风格的请求头
.body	请求内容主题，字符串类型
.meta	用户添加的扩展信息，在 Scrapy 内部模板间传递信息使用
.copy()	复制该请求

Response 类：class scrapy.http.Response()

Response 对象表示一个 HTTP 响应，由 Downloader 生成，Spider 处理。

表 2.2 Response 类的属性或方法

属性或方法	说明
.url	Response 对应的请求 url 地址
.status	HTTP 状态码，默认是 200
.headers	Response 对应的头部信息
.body	Response 对应的内容信息，字符串类型
.flags	一组标记
.request	产生 Response 类型对应的 Request 对象
.copy()	复制该响应

Item 类：class scrapy.item.Item()

Item 对象表示一个从 HTML 页面中提取的信息内容。由 Spider 生成，由 Item Pipeline 处理。Item 类似字典类型，可以按照字典类型操作。

Scrapy 爬虫支持多种 HTML 信息提取方法，包括 Beautiful Soup/lxml/re/XPath Selector/CSS Seletor 等。

下面简单介绍一下 CSS Selector。CSS Selector 的基本使用格式如下：

<HTML>.css('a::attr(href)').extract()

其中 a 是标签名称，href 是标签属性，这样就能获得对应的标签信息。CSS Selector 是由 W3C 组织维护并规范。

第三讲：实例 4：股票数据 Scrapy 爬虫

1.实例介绍

技术路线：scrapy

目标：获取上交所和深交所所有股票的名称和交易信息。

输出：保存在文件中。

2.实例编写

（1）配置 stocks.py 文件：修改对返回页面的处理；修改对新增 URL 爬取请求的处理。

stocks.py 代码

```
# -*- coding: utf-8 -*-
import scrapy
import re
class StocksSpider(scrapy.Spider):
    name = "stocks"
    start_urls = ['https://quote.eastmoney.com/stocklist.html']

    def parse(self, response):
    #对页面中所有的 a 链接进行提取，格式如下所示
        for href in response.css('a::attr(href)').extract():
            try:
                stock = re.findall(r"[s][hz]\d{6}", href)[0]
```

```
                    url = 'https://gupiao.baidu.com/stock/' + stock + '.html'
#上面是利用东方财富网中获取的股票代码在百度股票中获取其信息
                    yield scrapy.Request(url, callback=self.parse_stock)
                except:
                    continue
#如下在百度股票的单个页面中提取股票所需信息
    def parse_stock(self, response):
        infoDict = {}
        stockInfo = response.css('.stock-bets')
        name = stockInfo.css('.bets-name').extract()[0]
        keyList = stockInfo.css('dt').extract()
        valueList = stockInfo.css('dd').extract()
        for i in range(len(keyList)):
            key = re.findall(r'>.*</dt>', keyList[i])[0][1:-5]
            try:
                val = re.findall(r'\d+\.?.*</dd>', valueList[i])[0][0:-5]
            except:
                val = '--'
            infoDict[key]=val

        infoDict.update(
            {'股票名称': re.findall('\s.*\(',name)[0].split()[0] + \
              re.findall('\>.*\<', name)[0][1:-1]})
        yield infoDict
```

（2）编写 Pipelines，配置 pipelines.py 文件，定义对爬取项（Scraped Item）的处理类，配置 ITEM_PIPELINES 选项。

pipelines.py 源代码：

```
# -*- coding: utf-8 -*-
# Define your item pipelines here
# Don't forget to add your pipeline to the ITEM_PIPELINES setting
# See: https://doc.scrapy.org/en/latest/topics/item-pipeline.html
class BaidustocksPipeline(object):
    def process_item(self, item, spider):
        return item

class BaidustocksInfoPipeline(object):
    def open_spider(self, spider):
        self.f = open('BaiduStockInfo.txt', 'w')

    def close_spider(self, spider):
        self.f.close()

    def process_item(self, item, spider):#对每一个 item 进行处理
        try:
```

```
            line = str(dict(item)) + '\n'
            self.f.write(line)
        except:
            pass
        return item
```

setting.py 中被修改的部分的代码：

```
# Configure item pipelines
# See https://scrapy.readthedocs.org/en/latest/topics/item-pipeline.html
ITEM_PIPELINES = {
    'BaiduStocks.pipelines.BaidustocksInfoPipeline': 300,
}
```

3.实例优化

配置并发连接选项，在 settings.py 中，具体如下：

表 3.1 settings.py 文件

选项	说明
CONCURENT_REQUESTS	Downloader 最大并发请求下载数量，默认 32
CONCURENT_ITEMS	Item Pipeline 最大并发 ITEM 处理数量，默认 100
CONCURENT_REQUESTS_PER_DOMAIN	每个目标域名最大的并发请求数量，默认为 8
CONCURENT_REQUESTS_PER_IP	每个目标 IP 最大的并发请求数量，默认 0，非 0 有效