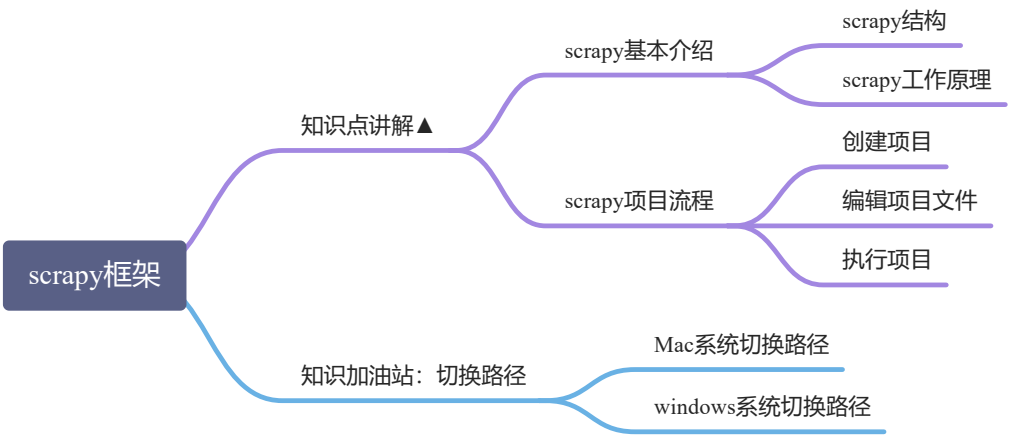
第13课 scrapy 框架

一、课程结构导图

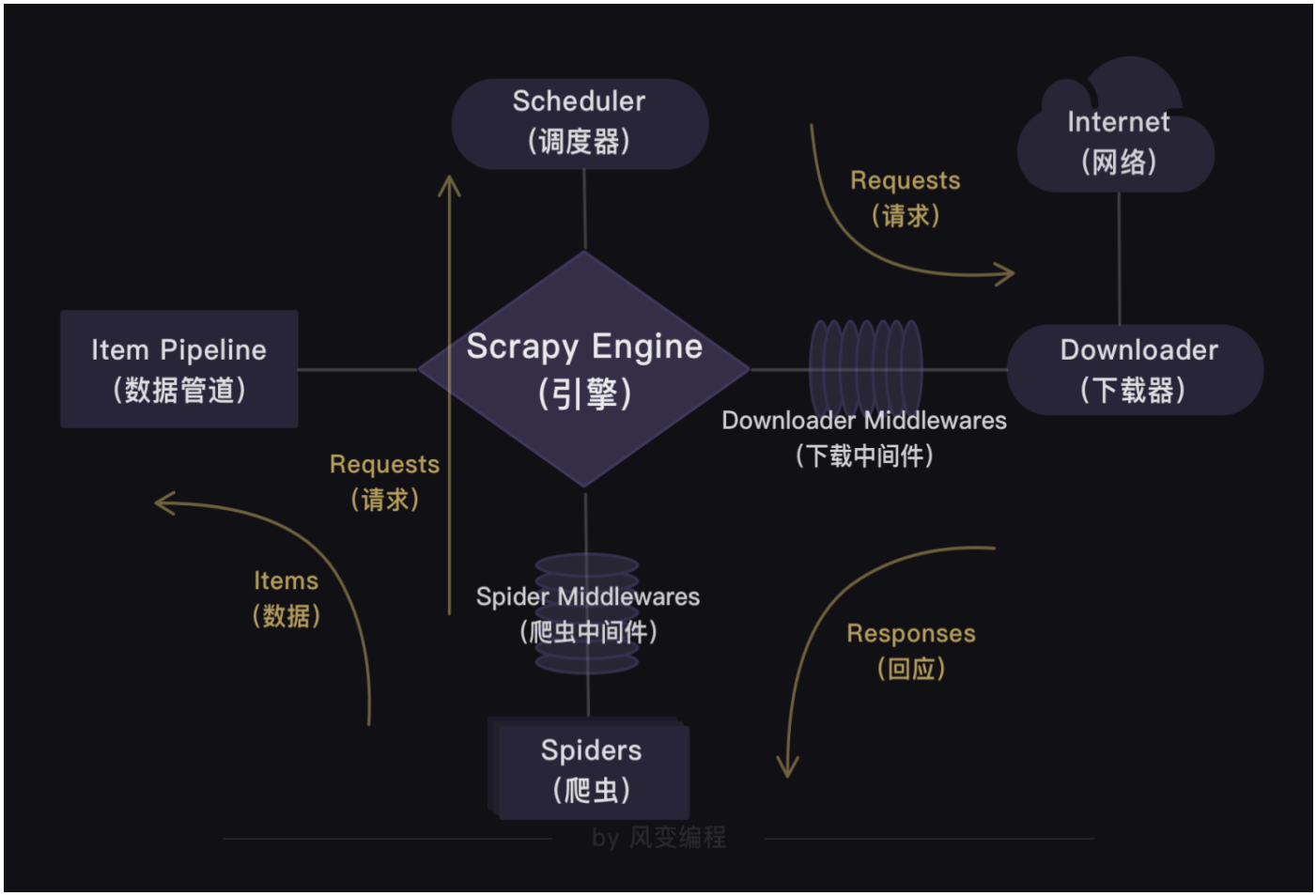


注：▲为重点知识点。

二、知识点讲解

2.1 scrapy基本介绍

2.1.1 scrapy结构



2.1.2 scrapy工作原理

Scrapy框架中，引擎是中心，其他组成部分由引擎调度。

在执行的过程中，先通过Spiders（爬虫）创建 requests 对象，传给引擎，然后引擎将其发给Scheduler（调度器）排好队列；

调度器排好顺序之后返回给引擎，引擎再将其发送到Downloader（下载器）进行网页爬取；

下载器爬取完将返回的 response（爬取到的内容）交给引擎，引擎再将其发送给Spiders（爬虫）进行解析并提取出有用的数据；

爬虫提取完数据返回给引擎，引擎再将其发送给Item Pipeline（数据管道），数据通道将数据下载下来并存储好，便完成一次爬取任务。

相关部件介绍：

- **Scheduler（调度器）：** 主要负责处理引擎发送过来的requests对象（即网页请求的相关信息集合，包括params，data，cookies，request headers...等），会把请求的url以有序的方式排列成队，并等待

引擎来提取（功能上类似于gevent库的queue模块）。

- **Downloader（下载器）：** 负责处理引擎发送过来的requests，进行网页爬取，并将爬取到的内容Response交给引擎。它对应【获取数据】的步骤。
- **Spiders（爬虫）：** 爬虫的核心部分，主要任务是创建requests对象和接受引擎发送过来的response（Downloader爬取到的内容），从中解析并提取出有用的数据。它对应【解析数据】和【提取数据】这两步。
- **Item Pipeline（数据管道）：** 只负责存储和处理Spiders提取到的有用数据。对应【存储数据】步骤。
- **Downloader Middlewares（下载中间件）：** 相当于下载器Downloader的秘书，比如会提前对引擎发送的诸多requests做出处理。
- **Spider Middlewares（爬虫中间件）：** 相当于爬虫Spiders的秘书，比如会提前接收并处理引擎发送来的response，过滤掉一些重复无用的东西。

在Scrapy里，整个爬虫程序的流程会在后台进行，所有的请求或返回的响应都由引擎自动分配去处理；
如果有某个请求出现异常，scrapy程序也会做异常处理，跳过报错的请求，继续往下运行程序；
Scrapy中的程序全部都是异步模式；

安装方法：

- Windows系统：安装命令：pip install scrapy
- mac：安装命令：pip3 install scrapy

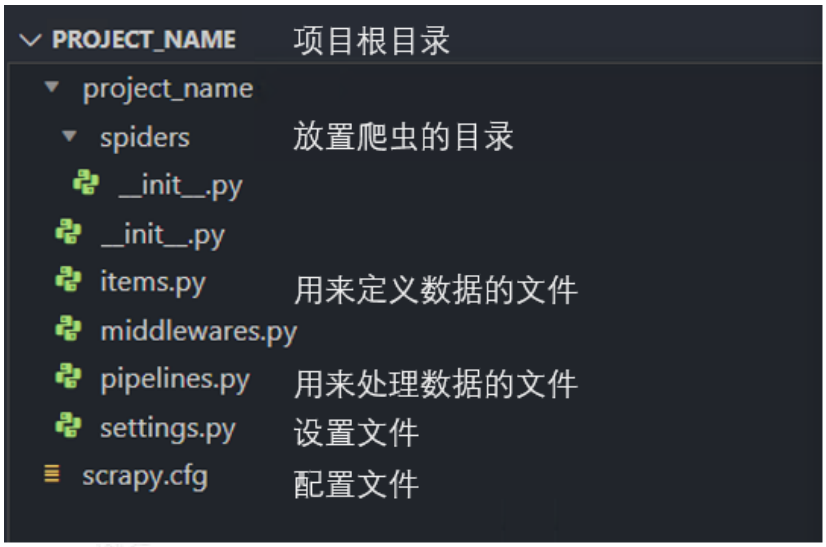
2.2 scrapy项目流程

2.2.1 创建项目

先打开命令符或终端，切换路径之后，输入以下命令，即可创建一个scrapy项目：

```
1 scrapy startproject [项目名称]      # [项目名称]是你的项目名称
```

创建完的项目有以下六个文件：（project_name是项目名称）。



2.2.2 编辑项目文件

- **爬虫文件**

爬虫文件也是python文件，需要自己新建，放在spider文件夹下即可。文件内容如下：

```python
import scrapy, bs4
from ..items import ProjectNameItem     # ProjectNameItem随项目名称变化

class ProjectNameSpider(scrapy.Spider): # ProjectNameSpider随项目名称变化
    name = 'projectname'                # projectname随项目名称变化
    allowed_domains = ['网站根目录']
    start_urls = ['目标网址']

    def parse(self, response):          # parse是默认处理response的方法，不能修改
        bs = bs4.BeautifulSoup(response.text,'html.parser')
        # 用BeautifulSoup解析response
        datas = bs.find_all('标签', 属性="属性值")
        for data in  datas:
            item = ProjectNameItem()    # 实例化ProjectNameItem这个类。
            item['variable_1'] = data.find('标签', 属性="属性值").text
            item['variable_2'] = data.find('标签', 属性="属性值")['属性']
            yield item                  # yield item是把获得的item传递给引擎。
```

注意：variable_1和variable_2是变量，要和items文件中的变量名保持一致。

- **items.py文件**

items文件内容比较简单，创建ProjectNameItem类，类中写入变量，这些变量用于爬虫文件的调用。

```
import scrapy
class ProjectNameItem(scrapy.Item): # ProjectNameName随项目名称变化
    variable_1 = scrapy.Field()
    variable_2 = scrapy.Field()
```

注意：variable_1和variable_2是变量，可以根据需要写入的数据自定义。

- **setting.py文件**

```
# Crawl responsibly by identifying yourself (and your website) on the user-agent
#USER_AGENT = 'project_name (+http://www.yourdomain.com)'

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

setting文件中需要传入请求头，可以直接新增，或者把USER_AGENT的取消注释，然后替换掉USER_AGENT的内容。可以改为：

```
USER_AGENT = 'Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/71.0.3578.98 Safari/537.36'
```

ROBOTSTXT_OBEY=True意思是遵守robots协议，对于robot协议禁止爬取的内容不进行爬取；如果改为False则是无需遵从robots协议，这样Scrapy就能不受限制地运行。

- **存储文件**

存储数据可以通过两个方法实现：一个是使用csv存储为csv文件，另一个是通过pipelines.py文件存储为Excel文件。

方法一：存储为csv文件；

在setting文件中，添加以下三行代码即可。

```
FEED_URI='./storage/data/%(name)s.csv'  # 导出文件的路径，把存储的文件放到与scrapy.cfg文件同级的storage文件夹的data子文件夹里。（会自动创建文件夹和文件）
FEED_FORMAT='CSV'               # 导出数据格式，写CSV就能得到CSV格式。
FEED_EXPORT_ENCODING='ansi'     # 导出文件编码，windows系统可以使用ansi，Mac系统使用utf-8
```

注意：执行项目之后，如果没有报错，但没有写入文件，可以把FEED_FORMAT='CSV'中的CSV改为小写。

方法二：存储为Excel文件；

需要修改两个文件：setting.py和pipelines.py。修改setting文件，取消以下三行代码的注释：

```
ITEM_PIPELINES = {
   'project_name.pipelines.ProjectNamePipeline': 300,
}   # project_name和ProjectNamePipeline会随着项目名称改变
```

编写pipelines.py文件，新建三个方法：

```
import openpyxl

class ProjectNamePipeline(object):
#定义一个ProjectNamePipeline类，负责处理item，ProjectNamePipeline会随着项目名称改变
    def __init__(self):
        self.wb =openpyxl.Workbook()            #创建工作薄
        self.ws = self.wb.active    # 获取活动表
        self.ws.append(['列1名称', '列2名称'])   # 往表格添加表头，有多少列则添加多少个元素

    def process_item(self, item, spider):       #process_item是默认的处理item的方法，不可更改
        line = [item['variable_1'], item['variable_2']]# 把数据添加到列表中，赋值给line
        self.ws.append(line)        # 把数据写入表格中
        return item                 # 将item返回给引擎
    def close_spider(self, spider): # close_spider是当爬虫结束运行时，这个方法就会执行
        self.wb.save('./%(name)s.xlsx')# 保存文件
        self.wb.close()             # 关闭文件
```

2.2.3 执行项目

编写完爬虫文件之后，开始执行项目，执行的入口可以是cmd/终端，或者在编译器中新建main文件来执行。

- **cmd/终端运行**

打开cmd/终端，先切换到scrapy项目的根目录，然后输入一下命令，回车即可开始执行项目。

```
scrapy crawl [项目名称]     # [项目名称]是你的项目名称
```

- **编译器运行**

在编译器，如Visual Studio Code或PyCharm等执行项目，需要先添加**main.py文件**（与scrapy.cfg文件同级），然后**执行main文件。**

main文件内容如下：

```
from scrapy import cmdline      # 导入cmdline模块，可以实现控制终端命令行
cmdline.execute(['scrapy','crawl','你的项目名称'])    # 用execute（）方法，输入运行scrapy的命令
```

注意：如果运行报错：unknown command crawl，可以在main文件中修改路径，参考以下代码：

```
from scrapy import cmdline
import os
dirpath=os.path.dirname(os.path.abspath(__file__))  # 获取当前路径
os.chdir(dirpath)          # 切换到当前目录

cmdline.execute(['scrapy','crawl','你的项目名称'])    # 和爬虫文件中的name的值保持一致
```

三、知识加油站：切换路径

3.1 Mac系统切换路径

获取文件或文件夹的路径的方法：把文件或文件夹拖到终端，便会显示路径。
获取到文件所在的文件夹之后，复制文件夹的路径，然后使用以下命令，即可切换到对应的文件夹路径下：

```
cd [文件夹路径]       # [文件夹路径]是要切换的文件夹路径
```

注意：cd和[文件夹路径]中间必须加一个空格。

3.2 windows系统切换路径

windows系统切换路径有多种情况：

- 如果是在C盘，则直接切换路径（同Mac系统的操作）。
- 如果是在其他的盘符，需要先切换盘符，再切换路径。切换盘符的方法：输入盘符名称加冒号。例如：切换到D盘，输入**D:** 然后回车；切换到E盘，输入**E:** 然后回车。