第四章 Scrapy 框架爬虫程序（4.1小结）

福州大学 数学与计算机科学学院

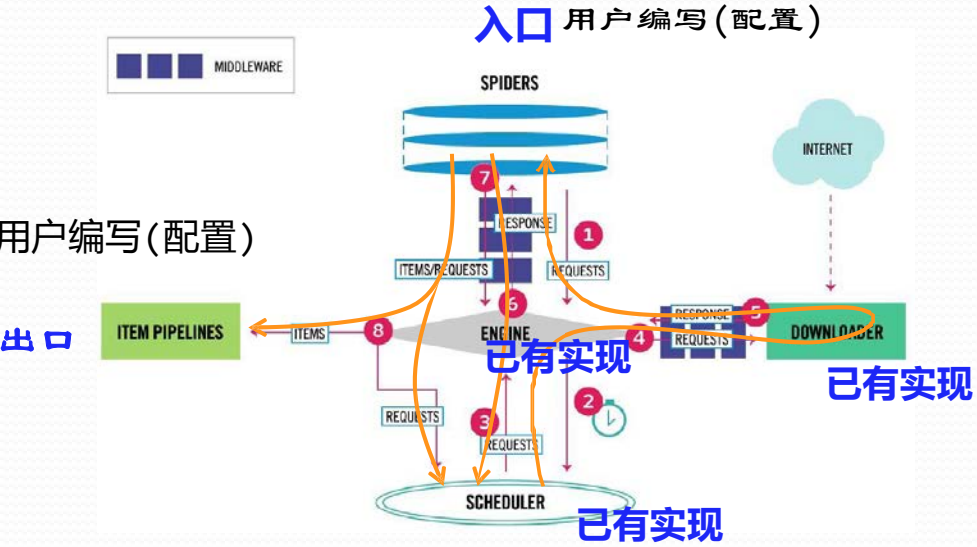
Fuzhou University College of Mathematics and Computer Science

教师：吴 伶

4.1Scrapy爬虫框架介绍

- 4.1.1 Scrapy框架介绍
  - 1.Scrapy的安装
  - 2. Scrapy爬虫框架结构：“5+2”结构
  - 3. Scrapy命令行的使用
  - 4.Scrapy与 requests的不同
- 4.1.2建立Scrapy项目
  - 1.Scrapy爬虫实例及目录结构
  - 2.Yield关键字和生成器
  - 3.Request、Response和Item类及方法

Scrapy 爬虫框架结构图



“5+2”结构

分布式

3条数据流路径

框架入口：Spider（初始爬取请求）

框架出口：Item Pipeline（爬取结果及处理）

Engine： 控制各模块数据流，不间断从Scheduler处 获得爬取请求，直至请求为空 。

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

Item Pipelines： 以流水线方式处理Spider产生的爬取项

Scheduler：对所有爬取请求进行调度管理。

Spider Middleware： 对请求和爬取项的再处理（修改、丢弃、新增请求或爬取项）

Downloader： 根据请求下载网页

Downloader Middleware： 实施Engine、Scheduler和Downloader 之间进行用户可配置的控制（修改、丢弃、新增请求或响应）

Scrapy常用命令

>scrapy <command> [options] [args]

Scrapy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	获得爬虫配置信息	scrapy settings [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]

requests vs. Scrapy

	requests	Scrapy
相同点	1. 实现Python爬虫重要技术路线 2. 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）	
不同点	页面级爬虫	网站级爬虫
	功能库	框架
	并发性考虑不足，性能较差	并发性好，性能较高
	重点在于页面下载	重点在于爬虫结构
	定制灵活	一般定制灵活，深度定制困难
	上手十分简单	入门稍难

2. yield关键字

yield ⇔ 生成器

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

生成器是一个不断产生值的函数

实例

```
>>> def gen(n):
        for i in range(n):
            yield i**2


>>> for i in gen(5):
        print(i, " ", end="")


0  1  4  9  16
>>>
```

生成器每调用一次在yield位置产生一个值，直到函数执行结束

为何要有生成器？

实例：求一组数的平方值

```
>>> def gen(n):
        for i in range(n):
            yield i**2


>>> for i in gen(5):
        print(i, " ", end="")


0  1  4  9  16
>>>
```

生成器写法

```
>>> def square(n):
        ls = [i**2 for i in range(n)]
        return ls


>>> for i in square(5):
        print(i, " ", end="")


0  1  4  9  16
>>>
```

普通写法

为何要有生成器？

生成器相比一次列出所有内容的优势：

1)更节省存储空间

2)响应更迅速

3)使用更灵活

```
>>> def square(n):
        ls = [i**2 for i in range(n)]
        return ls
```

```
>>> def gen(n):
        for i in range(n):
            yield i**2
```

如果n=1M、10M、100M或更大呢？

demo.py

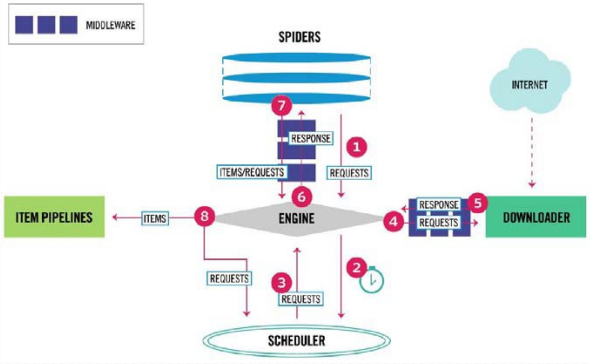
```
import scrapy

class DemoSpider(scrapy.Spider):
    name = "demo"

    def start_requests(self):
        urls = [
                'http://python123.io/ws/demo.html'
               ]
        for url in urls:
            yield scrapy.Request(url=url, callback=self.parse)

    def parse(self, response):
        fname = response.url.split('/')[-1]
        with open(fname, 'wb') as f:
            f.write(response.body)
        self.log('Saved file %s.' % fname)
```

3. Scrapy爬虫的数据类型



Request类

Response类

Item类

Request类

```
class scrapy.http.Request()
```

Request对象表示一个HTTP请求 由Spider生成，由Downloader执行

Request类

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	请求内容主体，字符串类型
.meta	用户添加的扩展信息，在Scrapy内部模块间传递信息使用
.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，由Spider处理

Response类型

属性或方法	说明
.url	Response对应的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