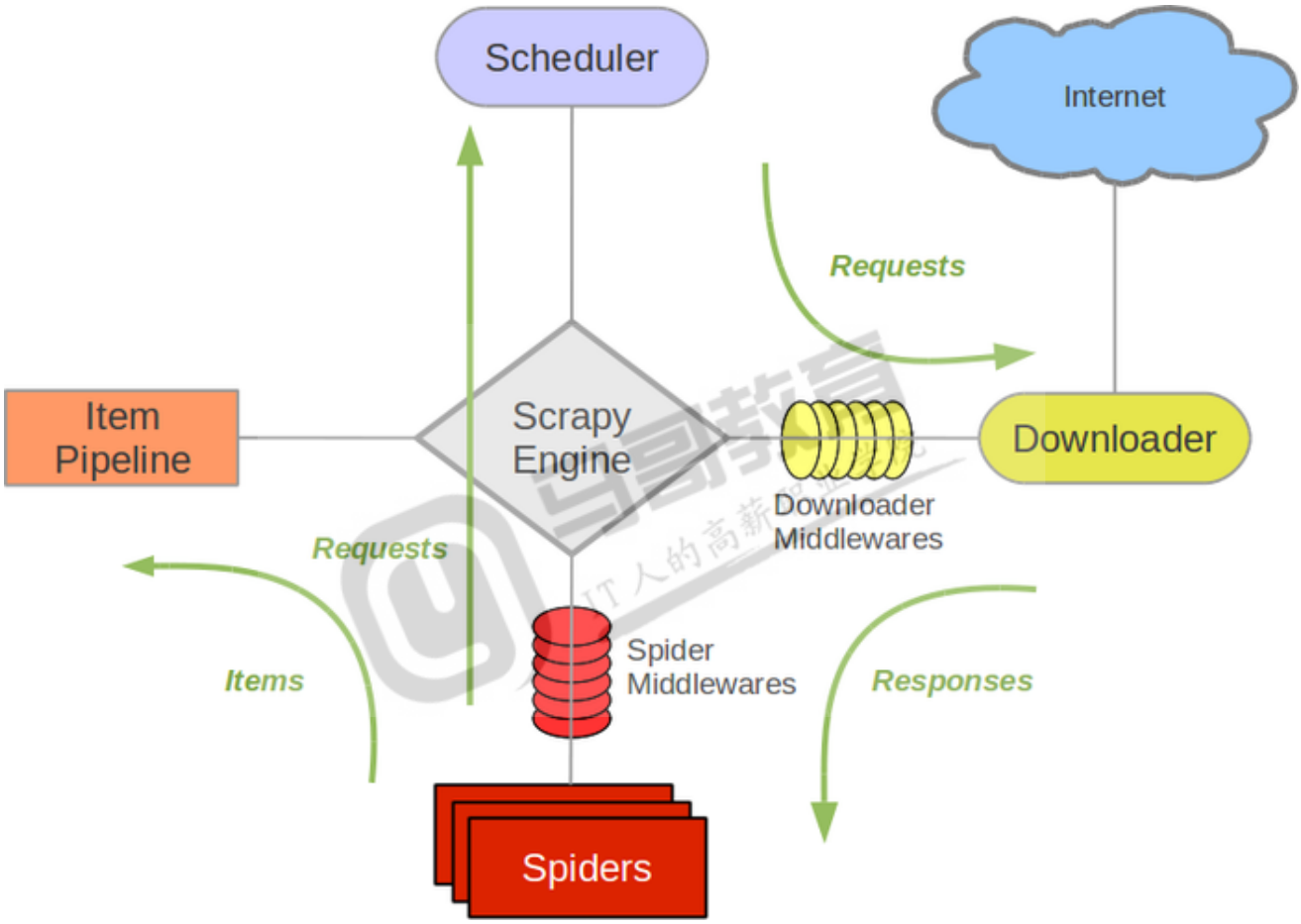
Scrapy框架

Scrapy是用Python实现的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。

Scrapy使用Twisted高效异步网络框架来处理网络通信，可以加快下载速度，不用自己去实现异步框架，并且包含了各种中间件接口，可以灵活的完成各种需求。

Scrapy架构



Scrapy Engine
引擎，负责控制数据流在系统中所有组件中流动，并在相应动作发生时触发事件。 此组件相当于爬虫的"大脑"，是整个爬虫的调度中心。

调度器(Scheduler)
调度器接收从引擎发送过来的request，并将他们入队，以便之后引擎请求他们时提供给引擎。
初始的爬取URL和后续在页面中获取的待爬取的URL将放入调度器中，等待爬取。同时调度器会自动**去除重复的URL**（如果特定的URL不需要去重也可以通过设置实现，如post请求的URL）

下载器(Downloader)
下载器负责获取页面数据并提供给引擎，而后提供给spider。

Spiders爬虫
Spider是编写的类，作用如下：
Scrapy用户编写用于分析response并提取item(即获取到的item)
额外跟进的URL，将额外跟进的URL提交给引擎，加入到Scheduler调度器中。将每个spider负责处理一个特定(或一些)网站。

Item Pipeline
Item Pipeline负责处理被spider提取出来的item。典型的处理有清理、 验证及持久化(例如存取到数据库中)。
当页面被爬虫解析所需的数据存入Item后，将被发送到项目管道(Pipeline)，并经过设置好次序的pipeline程序处理这些数据，最后将存入本地文件或存入数据库。

以下是item pipeline的一些典型应用：

- 清理HTML数据
- 验证爬取的数据(检查item包含某些字段)
- 查重(或丢弃)
- 将爬取结果保存到数据库中

下载器中间件(Downloader middlewares)

简单讲就是自定义扩展下载功能的组件

下载器中间件，是在引擎和下载器之间的特定钩子(specific hook)，处理它们之间的请求request和响应response。 它提供了一个简便的机制，通过插入自定义代码来扩展Scrapy功能。

通过设置下载器中间件可以实现爬虫自动更换user-agent、IP等功能。

Spider中间件(Spider middlewares)

Spider中间件，是在引擎和Spider之间的特定钩子(specific hook)，处理spider的输入(response)和输出(items或requests)。 也提供了同样的简便机制，通过插入自定义代码来扩展Scrapy功能。

数据流(Data flow)

1. 引擎打开一个网站(open a domain)，找到处理该网站的Spider并向该spider请求第一个（批）要爬取的URL(s)
2. 引擎从Spider中获取到第一个要爬取的URL并加入到调度器(Scheduler)作为请求以备调度
3. 引擎向调度器请求下一个要爬取的URL
4. 调度器返回下一个要爬取的URL给引擎，引擎将URL通过下载中间件并转发给下载器(Downloader)
5. 一旦页面下载完毕，下载器生成一个该页面的Response，并将其通过下载中间件发送给引擎
6. 引擎从下载器中接收到Response，然后通过Spider中间件发送给Spider处理
7. Spider处理Response并返回提取到的Item及(跟进的)新的Request给引擎
8. 引擎将Spider返回的Item交给Item Pipeline，将Spider返回的Request交给调度器
9. (从第二步)重复执行，直到调度器中没有待处理的request，引擎关闭

注意：只有当调度器中没有任何request了，整个程序才会停止执行。如果有下载失败的URL，会重新下载

安装scrapy

```
安装wheel支持
$ pip install wheel

安装scrapy框架
$ pip install scrapy

window下，为了避免windows编译安装twisted依赖，安装下面的二进制包
$ pip install Twisted-18.4.0-cp35-cp35m-win_amd64.whl
```

```
windows下出现如下问题
copying src\twisted\words\xish\xpathparser.g -> build\lib.win-amd64-3.5\twisted\words\xish
    running build_ext
    building 'twisted.test.raiser' extension
    error: Microsoft Visual C++ 14.0 is required. Get it with "Microsoft Visual C++ Build
Tools": http://landinghub.visualstudio.com/visual-cpp-build-tools

解决方案是，下载编译好的twisted，https://www.lfd.uci.edu/~gohlke/pythonlibs/#twisted
python3.5 下载 Twisted-18.4.0-cp35-cp35m-win_amd64.whl
python3.6 下载 Twisted-18.4.0-cp36-cp36m-win_amd64.whl

安装twisted
$ pip install Twisted-18.4.0-cp35-cp35m-win_amd64.whl
之后在安装scrapy就没有什么问题了
```

安装好，使用scrapy命令看看

```
> scrapy
Scrapy 1.5.0 - no active project

Usage:
  scrapy <command> [options] [args]

Available commands:
  bench         Run quick benchmark test
  check         Check spider contracts
  crawl         Run a spider
  edit          Edit spider
  fetch         Fetch a URL using the Scrapy downloader
  genspider     Generate new spider using pre-defined templates
  list          List available spiders
  parse         Parse URL (using its spider) and print the results
  runspider     Run a self-contained spider (without creating a project)
  settings      Get settings values
  shell         Interactive scraping console
  startproject  Create new project
  version       Print Scrapy version
  view          Open URL in browser, as seen by Scrapy
```

Scrapy开发

项目编写流程

1. 创建项目
   使用`scrapy startproject proname`创建一个scrapy项目
2. 编写item
   在items.py中编写Item类，明确从response中提取的item
3. 编写爬虫
   编写spiders/proname_spider.py，即爬取网站的spider并提取出item
4. 编写item pipeline
   item的处理，可以存储

1 创建项目

豆瓣书评爬取

```
标签为“编程”，第一页、第二页链接
https://book.douban.com/tag/%E7%BC%96%E7%A8%8B?start=0&type=T
https://book.douban.com/tag/%E7%BC%96%E7%A8%8B?start=20&type=T
```

随便找一个目录来创建项目，执行下面命令

```
$ scrapy startproject first
```

会产生如下目录和文件

```
first
 ├─  scrapy.cfg
 └─ first
    ├─  items.py
    ├─  middlewares.py
    ├─  pipelines.py
    ├─  settings.py
    ├─  __init__.py
    └─ spiders
        └─  __init__.py

first：外部的first目录是整个项目目录，内部的first目录是整个项目的全局目录

scrapy.cfg：必须有的重要的项目的配置文件

first 项目目录
    __init__.py 必须有，包文件
    item.py 定义Item类，从scrapy.Item继承，里面定义scrapy.Field类
    pipelines.py 重要的是process_item()方法
    settings.py：
        BOT_NAME                        爬虫名
        ROBOTSTXT_OBEY = True           是否遵从robots协议
```

```
        USER_AGENT = ''                 指定爬取时使用
        CONCURRENT_REQEUST = 16         默认16个并行
        DOWNLOAD_DELAY = 3              下载延时
        COOKIES_ENABLED = False         缺省是启用，一般需要登录时才需要开启cookie
        DEFAULT_REQUEST_HEADERS = {}    默认请求头，需要时填写
        SPIDER_MIDDLEWARES              爬虫中间件
        DOWNLOADER_MIDDLEWARES          下载中间件
            'first.middlewares.FirstDownloaderMiddleware': 543
            543优先级越小越高
        ITEM_PIPELINES                  管道配置
            'firstscrapy.pipelines.FirstscrapyPipeline': 300
            item交给哪一个管道处理，300优先级越小越高
    spiders目录
        __init__.py 必须有，可以在这里写爬虫类，也可以写爬虫子模块
```

```
# first/settings.py参考
BOT_NAME = 'first'

SPIDER_MODULES = ['first.spiders']
NEWSPIDER_MODULE = 'first.spiders'
USER_AGENT = "Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/55.0.2883.75 Safari/537.36"
ROBOTSTXT_OBEY = False

# Disable cookies (enabled by default)
COOKIES_ENABLED = False
```

注意一定要更改User-Agent，否则访问 `https://book.douban.com/` 会返回403

2 编写Item

在items.py中编写

```
import scrapy

class BookItem(scrapy.Item):
    title = scrapy.Field() # 书名
    rate = scrapy.Field() # 评分
```

3 编写爬虫

为爬取豆瓣书评编写爬虫类，在spiders目录下。
编写的爬虫类需要继承自scrapy.Spider，在这个类中定义爬虫名、爬取范围、其实地址等。
在scrapy.Spider中parse方法未实现，所以子类应该实现parse方法。该方法传入response对象。

```
# scrapy源码中
class Spider():
    def parse(self, response):
        raise NotImplementedError
```

爬取读书频道，tag为“编程”的书名和评分。
`https://book.douban.com/tag/%E7%BC%96%E7%A8%8B?start=20&type=T`

```
import scrapy

class BookSpider(scrapy.Spider): # BookSpider
    name = 'doubanbook' # 爬虫名
    allowed_domains = ['douban.com'] # 爬虫爬取范围
    url = 'https://book.douban.com/tag/%E7%BC%96%E7%A8%8B?start=0&type=T'
    start_urls = [url] # 起始URL

    # 下载器获取了WEB Server的response就行了，parse就是解析响应的内容
    def parse(self, response):
        print(type(response)) # scrapy.http.response.html.HtmlResponse
        print('-'*30)
        print(response)
```

以上模板代码可以使用`$ scrapy genspider -t basic book douban.com`创建

使用crawl爬取子命令

```
$ scrapy list
$ scrapy crawl -h
scrapy crawl [options] <spider>
# 指定爬虫名称开始爬取
$ scrapy crawl doubanbook
```

> 如果在windows下运行发生twisted的异常`ModuleNotFoundError: No module named 'win32api'`，请安装`$ pip install pywin32`

response是服务器端HTTP响应，它是scrapy.http.response.html.HtmlResponse类。由此，修改代码如下

```
import scrapy
from scrapy.http.response.html import HtmlResponse

class BookSpider(scrapy.Spider): # BookSpider
    name = 'doubanbook' # 爬虫名
    allowed_domains = ['douban.com'] # 爬虫爬取范围
    url = 'https://book.douban.com/tag/%E7%BC%96%E7%A8%8B?start=0&type=T'
    start_urls = [url] # 起始URL

    # 下载器获取了WEB Server的response就行了，parse就是解析响应的内容
    def parse(self, response:HtmlResponse):
        print(type(response)) # scrapy.http.response.html.HtmlResponse
        print('-'*30)
        print(type(response.text), type(response.body))
        print('-'*30)
        print(response.encoding)
        with open('o:/testbook.html', 'w', encoding='utf-8') as f:
            try:
                f.write(response.text)
```

```
                f.flush()
            except Exception as e:
                print(e)
```

3.1 解析HTML

爬虫获得的内容response对象，可以使用解析库来解析。
scrapy包装了lxml，父类TextResponse类也提供了xpath方法和css方法，可以混合使用这两套接口解析HTML。

```
import scrapy
from scrapy.http.response.html import HtmlResponse

response = HtmlResponse('file:///O:/testbook.html', encoding='utf-8')

with open('o:/testbook.html', encoding='utf8') as f:
    response._set_body(f.read())
    #print(response.text)

    # 获取所有标题及评分
    # xpath解析
    subjects = response.xpath('//li[@class="subject-item"]')
    for subject in subjects:
        title = subject.xpath('.//h2/a/text()').extract() # list
        print(title[0].strip())

        rate = subject.xpath('.//span[@class="rating_nums"]/text()').extract()
        print(rate[0].strip())

    print('-'*30)
    # css解析
    subjects = response.css('li.subject-item')
    for subject in subjects:
        title = subject.css('h2 a::text').extract()
        print(title[0].strip())

        rate = subject.css('span.rating_nums::text').extract()
        print(rate[0].strip())
    print('-'*30)

    # xpath和css混合使用、正则表达式匹配
    subjects = response.css('li.subject-item')
    for subject in subjects:
        # 提取链接
        href = subject.xpath('.//h2').css('a::attr(href)').extract()
        print(href[0])
        # 使用正则表达式
        id = subject.xpath('.//h2/a/@href').re(r'\d*99\d*')
        if id:
            print(id[0])

        # 要求显示9分以上数据
```

```
        rate = subject.xpath('.//span[@class="rating_nums"]/text()').re(r'^9.*')
        # rate = subject.css('span.rating_nums::text').re(r'^9\..*')
        if rate:
            print(rate)
```

3.2 item封装数据

```
# spiders/bookspider.py
import scrapy
from scrapy.http.response.html import HtmlResponse
from ..items import BookItem

class BookSpider(scrapy.Spider): # BookSpider
    name = 'doubanbook' # 爬虫名
    allowed_domains = ['douban.com'] # 爬虫爬取范围
    url = 'https://book.douban.com/tag/%E7%BC%96%E7%A8%8B?start=0&type=T'
    start_urls = [url] # 起始URL

    # 下载器获取了WEB Server的response就行了，parse就是解析响应的内容
    def parse(self, response:HtmlResponse):
        items = []
        # xpath解析
        subjects = response.xpath('//li[@class="subject-item"]')
        for subject in subjects:
            title = subject.xpath('.//h2/a/text()').extract()
            rate = subject.xpath('.//span[@class="rating_nums"]/text()').extract_first()
            item = BookItem()
            item['title'] = title[0].strip()
            item['rate'] = rate.strip()
            items.append(item)

        print(items)

        # 下面文件测试用，以后不这么用
        with open('o:/test.json', 'w', encoding='utf8') as f:
            for item in items:
                f.write('{} {}\n'.format(item['title'], item['rate']))

        return items

# 使用命令保存return的数据
# scrapy crawl -h
# --output=FILE, -o FILE  dump scraped items into FILE (use - for stdout)
# 文件扩展名支持'json', 'jsonlines', 'jl', 'csv', 'xml', 'marshal', 'pickle'
# scrapy crawl doubanbook -o dbbooks.json
```

得到下图数据

```
[
{"rate": "9.5", "name": "\u8ba1\u7b97\u673a\u7a0b\u5e8f\u7684\u6784\u9020\u548c\u89e3\u91ca"},
{"rate": "9.2", "name": "\u7f16\u7801"},
{"rate": "9.3", "name": "\u4ee3\u7801\u5927\u5168\uff08\u7b2c2\u7248\uff09"},
{"rate": "9.5", "name": "\u6df1\u5165\u7406\u89e3\u8ba1\u7b97\u673a\u7cfb\u7edf"},
{"rate": "9.4", "name": "C\u7a0b\u5e8f\u8bbe\u8ba1\u8bed\u8a00"},
{"rate": "9.3", "name": "\u7b97\u6cd5\u5bfc\u8bba\uff08\u539f\u4e66\u7b2c2\u7248\uff09"},
{"rate": "9.4", "name": "\u7b97\u6cd5\uff08\u7b2c4\u7248\uff09"},
{"rate": "9.3", "name": "JavaScript\u9ad8\u7ea7\u7a0b\u5e8f\u8bbe\u8ba1\uff08\u7b2c3\u7248\uff09"},
{"rate": "8.8", "name": "\u9ed1\u5ba2\u4e0e\u753b\u5bb6"},
{"rate": "9.0", "name": "\u96c6\u4f53\u667a\u6167\u7f16\u7a0b"},
{"rate": "9.1", "name": "\u7f16\u7a0b\u73e0\u7391"},
{"rate": "9.1", "name": "Java\u7f16\u7a0b\u601d\u60f3 \uff08\u7b2c4\u7248\uff09"},
{"rate": "9.2", "name": "Python\u7f16\u7a0b\uff1a\u4ece\u5165\u95e8\u5230\u5b9e\u8df5"},
{"rate": "9.2", "name": "C++ Primer \u4e2d\u6587\u7248\uff08\u7b2c 4 \u7248\uff09"},
{"rate": "8.8", "name": "\u7a0b\u5e8f\u5458\u7684\u81ea\u6211\u4fee\u517b"},
{"rate": "9.6", "name": "Fluent Python"},
{"rate": "9.4", "name": "UNIX\u73af\u5883\u9ad8\u7ea7\u7f16\u7a0b"},
{"rate": "9.0", "name": "Python\u7f16\u7a0b\u5feb\u901f\u4e0a\u624b"},
{"rate": "8.6", "name": "\u7a0b\u5e8f\u5458\u4fee\u70bc\u4e4b\u9053"},
{"rate": "9.0", "name": "\u91cd\u6784"}
]
```


注意上图的数据已经是unicode字符，汉字的unicode表达

4 pipeline处理

将bookspider.py中BookSpider改成生成器，只需要把 `return items` 改造成 `yield item` ，即由产生一个列表变成yield一个个item。

脚手架帮我们创建了一个pipelines.py文件和一个类。

4.1 开启pipeline

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

整数300表示优先级，越小越高。取值范围为0-1000。

4.2 常用方法

名称	参数	
process_item(self, item, spider)	item爬取的一个个数据 spider表示item的爬取者 每一个item处理都调用 返回一个Item对象，或抛出DropItem异常 被丢弃的Item对象将不会被之后的pipeline组件处理	必须
open_spider(self, spider)	spider表示被开启的spider 调用一次	可选
close_spider(self, spider)	spider表示被关闭的spider 调用一次	可选
__init__(self)	spider实例创建时 调用一次	可选

```
# 常用方法
class FirstPipeline(object):
    def __init__(self): # 全局设置
        print('~~~~~~~~~~ init ~~~~~~~~~~~~')

    def open_spider(self, spider): # 当某spider开启时调用
        print(spider, '~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~')

    def process_item(self, item, spider):
        # item 获取的item；spider 获取该item的spider
        return item

    def close_spider(self, spider): # 当某spider关闭时调用
        print(spider, '========================================')
```

需求

通过pipeline将爬取的数据存入json文件中

```
# spider/bookspider.py
import scrapy
from scrapy.http.response.html import HtmlResponse
from ..items import BookItem

class BookSpider(scrapy.Spider): # BookSpider
    name = 'doubanbook' # 爬虫名
    allowed_domains = ['douban.com'] # 爬虫爬取范围
    url = 'https://book.douban.com/tag/%E7%BC%96%E7%A8%8B?start=0&type=T'
    start_urls = [url] # 起始URL

    # spider上自定义配置信息
    custom_settings = {
        'filename' : 'o:/books.json'
    }
```

```
    # 下载器获取了WEB Server的response就行了，parse就是解析响应的内容
    def parse(self, response:HtmlResponse):
        #items = []
        # xpath解析
        subjects = response.xpath('//li[@class="subject-item"]')
        for subject in subjects:
            title = subject.xpath('.//h2/a/text()').extract()
            rate = subject.xpath('.//span[@class="rating_nums"]/text()').extract_first()
            item = BookItem()
            item['title'] = title[0].strip()
            item['rate'] = rate.strip()
            #items.append(item)

            yield item


# pipelines.py
import json

class FirstPipeline(object):
    def __init__(self): # 全局设置
        print('~~~~~~~~~~ init ~~~~~~~~~~~~')

    def open_spider(self, spider): # 当某spider开启时调用
        print('{} ~~~~~~~~~~~~~~~~~~~~'.format(spider))
        print(spider.settings.get('filename'))
        self.file = open(spider.settings['filename'], 'w', encoding='utf-8')
        self.file.write('[\n')

    def process_item(self, item, spider):
        # item 获取的item；spider 获取该item的spider
        self.file.write(json.dumps(dict(item)) + ',\n')
        return item

    def close_spider(self, spider): # 当某spider关闭时调用
        self.file.write(']')
        self.file.close()
        print('{} ======================='.format(spider))
        print('-'*30)
```