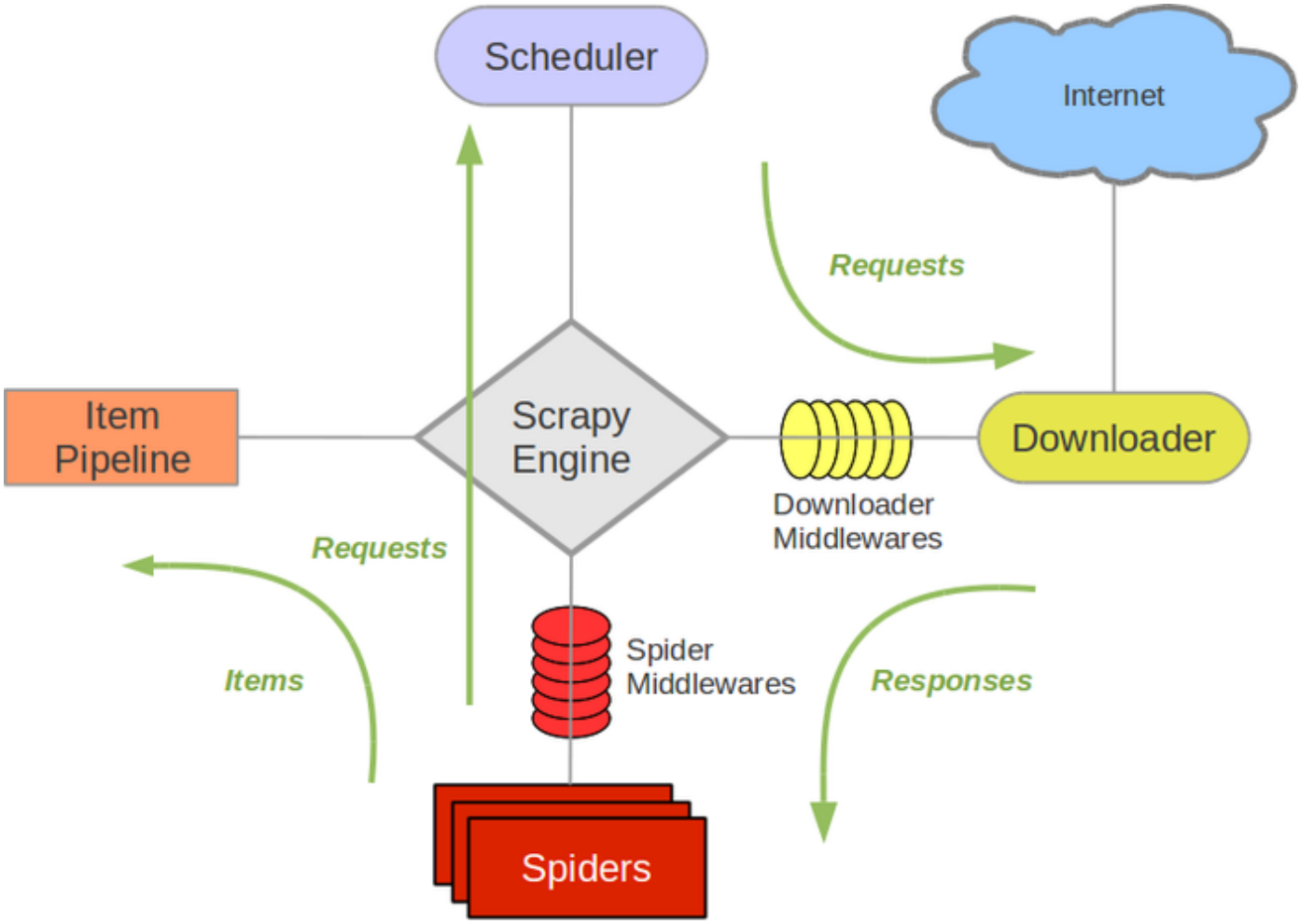
Scrapy框架

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基于事件的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。
类似管道`$ ls | grep test`。

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
```

一个简单的爬虫示例

创建爬虫

`scrapy startproject geektime`

在spiders目录下创建geektimeSpider.py并写入代码

```
import scrapy


class geektimeSpider(scrapy.Spider):
    name = "geektime"

    def start_requests(self):
        urls = [
            'https://quotes.toscrape.com/page/1/',
            'https://quotes.toscrape.com/page/2/',
        ]
        for url in urls:
            yield scrapy.Request(url=url, callback=self.parse)

    def parse(self, response):
        page = response.url.split("/")[-2]
        filename = f'quotes-{page}.html'
        with open(filename, 'wb') as f:
            f.write(response.body)
        self.log(f'Saved file {filename}')
```

还可以定义一个 start_urls 变量，内容为url列表，此时会默认使用此列表调用 start_requests 方法；

parse为默认回调方法，即使不指定也会自动回调。

```
import scrapy


class geektimeSpider(scrapy.Spider):
    name = "geektime"
    start_urls = [
        'https://quotes.toscrape.com/page/1/',
        'https://quotes.toscrape.com/page/2/',
    ]
```

```
    def parse(self, response):
        page = response.url.split("/")[-2]
        filename = f'quotes-{page}.html'
        with open(filename, 'wb') as f:
            f.write(response.body)
        self.log(f'Saved file {filename}')
```

- `name`：标识爬虫。 肯定是 在一个项目中是唯一的，即不能为不同的名称设置相同的爬虫名称。
- `start_requests()`：必须返回一个可迭代的请求。
- `parse()`：将被调用来处理的方法为每个请求下载的响应。 响应参数 是一个实例 `TextResponse`持有 页面内容，并有更多有用的方法来处理它。

  这 `parse()`方法通常解析响应，提取 将抓取的数据作为 dicts 并找到新的 URL 关注并创建新请求（`Request`） 从他们。

运行爬虫

在项目根目录下运行

```
scrapy crawl geektime
```

运行之后能够看到项目根目录下有两个下载好的文件。

提取数据

使用命令

`scrapy shell "https://quotes.toscrape.com/page/1/"`

之后会进入一个交互式shell，可以通过各种方法操纵返回数据

```
>>> response.css('title')      # 返回一个 Selector 对象
[<Selector xpath='descendant-or-self::title' data='<title>Quotes to Scrape</title>'>]

# 获取title的内容
>>> response.css('title::text').getall()
['Quotes to Scrape']

# 不指定 ::text 会获取整个标签
>>>response.css('title').getall()
['<title>Quotes to Scrape</title>']

# 使用正则表达式匹配
>>> response.css('title::text').re(r'Quotes.*')
```

```
['Quotes to Scrape']

>>> response.css('title::text').re(r'Q\w+')
['Quotes']

>>> response.css('title::text').re(r'(\w+) to (\w+)')
['Quotes', 'Scrape']
```

除了 CSS ， Scrapy 选择器还支持使用 XPath 表达式：

```
>>> response.xpath('//title')
[<Selector xpath='//title' data='<title>Quotes to Scrape</title>'>]

>>> response.xpath('//title/text()').get()
'Quotes to Scrape'
```

获取其它标签内容

`https://quotes.toscrape.com`页面代码

```
<div class="quote">
    <span class="text">“The world as we have created it is a process of our
    thinking. It cannot be changed without changing our thinking.”</span>
    <span>
        by <small class="author">Albert Einstein</small>
        <a href="/author/Albert-Einstein">(about)</a>
    </span>
    <div class="tags">
        Tags:
        <a class="tag" href="/tag/change/page/1/">change</a>
        <a class="tag" href="/tag/deep-thoughts/page/1/">deep-thoughts</a>
        <a class="tag" href="/tag/thinking/page/1/">thinking</a>
        <a class="tag" href="/tag/world/page/1/">world</a>
    </div>
</div>
```

运行命令 `$ scrapy shell 'https://quotes.toscrape.com'`

如果要获取`<div class="quote">`中的内容，可以这样写：

```
>>> quote = response.css("div.quote")[0]
>>> text = quote.css("span.text::text").get()
>>> text
'“The world as we have created it is a process of our thinking. It cannot be changed
without changing our thinking.”'

# 获取div标签class为tag中a标签class为tag的text
```

```
>>> tags = quote.css("div.tags a.tag::text").getall()
>>> tags
['change', 'deep-thoughts', 'thinking', 'world']


# 获取 a 标签的 href 属性值
>>> quote.css("div.tags a::attr(href)").getall()
['/tag/change/page/1/', '/tag/deep-thoughts/page/1/', '/tag/thinking/page/1/',
'/tag/world/page/1/']
```

在python代码中获取解析返回数据

```
import scrapy


class geektimeSpider(scrapy.Spider):
    name = "geektime2"

    def start_requests(self):
        urls = [
            'https://quotes.toscrape.com/page/1/',
            'https://quotes.toscrape.com/page/2/',
        ]
        for url in urls:
            yield scrapy.Request(url=url, callback=self.parse)

    def parse(self, response):
        for quote in response.css('div.quote'):
            yield {
                'text': quote.css('span.text::text').get(),
                'author': quote.css('small.author::text').get(),
                'tags': quote.css('div.tags a.tag::text').getall(),
            }
```

运行： `scrapy crawl geektime2`

存储数据：

- `scrapy crawl geektime2 -O quotes.json` **json列表形式 (所有json存储在一个列表中)**
- `scrapy crawl geektime2 -O quotes.jl` **json lines 格式 (一行一条json数据)**

提取下一页内容

通过HTML看到下一页的标签

```
<ul class="pager">
    <li class="next">
        <a href="/page/2/">Next <span aria-hidden="true">&rarr;</span></a>
    </li>
</ul>
```

获取标签链接：

```
>>> response.css('li.next a::attr(href)').get()
'/page/2/'
```

使用爬虫代码解析

```
import scrapy


class geektimeSpider(scrapy.Spider):
    name = "geektimeSecond"

    start_urls = [
        'https://quotes.toscrape.com/page/1/',
    ]

    def parse(self, response):
        for quote in response.css('div.quote'):
            yield {
                'text': quote.css('span.text::text').get(),
                'author': quote.css('small.author::text').get(),
                'tags': quote.css('div.tags a.tag::text').getall(),
            }
        # 获取下一页链接
        next_page = response.css('li.next a::attr(href)').get()
        # 判断链接不是空
        if next_page is not None:
            # 接收下一页返回内容
            next_page = response.urljoin(next_page)
            # 继续调用 Scrapy 请求，并回调 parse 函数
            yield scrapy.Request(next_page, callback=self.parse)
```

更简便的方法是使用 `response.follow`

```
import scrapy
```

```
class geektimeSpider(scrapy.Spider):
    name = "geektimeSecond"

    start_urls = [
        'https://quotes.toscrape.com/page/1/',
    ]

    def parse(self, response):
        for quote in response.css('div.quote'):
            yield {
                'text': quote.css('span.text::text').get(),
                'author': quote.css('small.author::text').get(),
                'tags': quote.css('div.tags a.tag::text').getall(),
            }
        # ================================================================================
        next_page = response.css('li.next a::attr(href)').get()
        if next_page is not None:
            # 直接使用 response.follow 即可
            yield response.follow(next_page, callback=self.parse)

        # 当需要获取 a 标签的 href 属性时，支持自动获取
        # ================================================================================
        for a in response.css('ul.pager a'):
            yield response.follow(a, callback=self.parse)

        # 进一步缩短，使用 follow_all 方法
        # ================================================================================
        yield from response.follow_all(css='ul.pager a', callback=self.parse)
```

使用参数

`scrapy crawl geektimeSecond2 -O quotes-humor.json -a tag=humor`

```
import scrapy


class geektimeSpider(scrapy.Spider):
    name = "geektimeSecond2"

    def start_requests(self):
        url = 'https://quotes.toscrape.com/'
        tag = getattr(self, 'tag', None)
        if tag is not None:
            url = url + 'tag/' + tag
        yield scrapy.Request(url, self.parse)
```

```
    def parse(self, response):
        for quote in response.css('div.quote'):
            yield {
                'text': quote.css('span.text::text').get(),
                'author': quote.css('small.author::text').get(),
            }

        next_page = response.css('li.next a::attr(href)').get()
        if next_page is not None:
            yield response.follow(next_page, self.parse)
```

使用item

首先，在items.py 文件中定义item

```
# Define here the models for your scraped items
#
# See documentation in:
# https://docs.scrapy.org/en/latest/topics/items.html

import scrapy


class geektimeScrapyDemoItem(scrapy.Item):
    # 设置字段
    text = scrapy.Field()
    author = scrapy.Field()
    tags = scrapy.Field()
    last_updated = scrapy.Field(serializer=str)
```

```
import scrapy
from geektime_scrapy_demo.items import geektimeScrapyDemoItem


class geektimeSpider(scrapy.Spider):
    name = "MgeduItemTest"

    start_urls = [
        'https://quotes.toscrape.com/page/1/',
    ]

    def parse(self, response):
        for quote in response.css('div.quote'):
            item = geektimeScrapyDemoItem()
            item['text'] = quote.css('span.text::text').get()
```

```
            item['author'] = quote.css('small.author::text').get()
            item['tags'] = quote.css('div.tags a.tag::text').getall()
            yield item
```

使用 Item Pipeline

一个项目被爬虫抓取后，它被发送到 Item Pipeline 它通过按顺序执行的几个组件来处理。

每个 Item Pipeline 组件是一个 实现简单方法的 Python 类。

Item Pipeline 的典型用途是：

- 清理 HTML 数据
- 验证抓取的数据（检查项目是否包含某些字段）
- 检查重复项（并删除它们）
- 将抓取的项目存储在数据库中

在 pipelines.py 文件中写入，用来将爬取到的数据保存到 items.jl 文件中：

```
import json
# Define your item pipelines here
#
# Don't forget to add your pipeline to the ITEM_PIPELINES setting
# See: https://docs.scrapy.org/en/latest/topics/item-pipeline.html


# useful for handling different item types with a single interface
from itemadapter import ItemAdapter


class geektimeScrapyDemoPipeline:
    # 爬虫开启时自动调用此方法
    def open_spider(self, spider):
        self.file = open('items.jl', 'w')

    # 爬虫关闭时自动调用此方法
    def close_spider(self, spider):
        self.file.close()

    def process_item(self, item, spider):
        line = json.dumps(ItemAdapter(item).asdict()) + "\n"
        self.file.write(line)
        return item
```

在 settings.py 文件中将以下代码注释取消：

```
ITEM_PIPELINES = {
   # pipelines 中类的名称，可以加入多个
   'geektime_scrapy_demo.pipelines.geektimeScrapyDemoPipeline': 300,
}
```

使用中间件

- `process_spider_input(response, spider)`
  - 对于通过 spider 的每个响应都会调用此方法 中间件并进入 spider，进行处理。
    - `process_spider_input()`应该返回 None
    - 如果它返回 `None`, Scrapy 会继续处理这个响应， 执行所有其他中间件，直到最终收到响应 交给spider处理
- `process_spider_output(response, result, spider)`
  - 使用从 Spider 返回的结果调用此方法，之后 它已经处理了响应。
    - 必须返回一个 result 的可迭代对象
- `process_start_requests(start_requests, spider)`
  - 这个方法是在spider的启动请求中调用的，必须只返回requests

示例

在middlewares.py 中写入中间件代码

```
import random

# 自动替换UA
class UAMiddleware(object):

    def __init__(self):
        self.USER_AGENT_LIST = [
            "Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/45.0.2454.101 Safari/537.36",
            "Dalvik/1.6.0 (Linux; U; Android 4.2.1; 2013022 MIUI/JHACNBL30.0)",
            "Mozilla/5.0 (Linux; U; Android 4.4.2; zh-cn; HUAWEI MT7-TL00
Build/HuaweiMT7-TL00) AppleWebKit/533.1 (KHTML, like Gecko) Version/4.0 Mobile
Safari/533.1",
            "AndroidDownloadManager",
            "Apache-HttpClient/UNAVAILABLE (java 1.4)",
            "Dalvik/1.6.0 (Linux; U; Android 4.3; SM-N7508V Build/JLS36C)",
            "Android50-AndroidPhone-8000-76-0-Statistics-wifi",
            "Dalvik/1.6.0 (Linux; U; Android 4.4.4; MI 3 MIUI/V7.2.1.0.KXCCNDA)",
            "Dalvik/1.6.0 (Linux; U; Android 4.4.2; Lenovo A3800-d Build/LenovoA3800-
d)",
            "Lite 1.0 ( http://litesuits.com )",
            "Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 5.1; Trident/4.0; .NET4.0C;
.NET4.0E; .NET CLR 2.0.50727)",
```

```
            "Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/38.0.2125.122 Safari/537.36 SE 2.X MetaSr 1.0",
            "Mozilla/5.0 (Linux; U; Android 4.1.1; zh-cn; HTC T528t Build/JRO03H)
AppleWebKit/534.30 (KHTML, like Gecko) Version/4.0 Mobile Safari/534.30;
360browser(securitypay,securityinstalled); 360(android,uppayplugin); 360 Aphone Browser
(2.0.4)",
        ]
    # 使用 process_spider_input 方法
    def process_spider_input(self, response, spider):
        ua = random.choice(self.USER_AGENT_LIST)
        response.request.headers['User-Agent'] = ua
        return None
```

在settings.py 中激活中间件

```
SPIDER_MIDDLEWARES = {
   'geektime_scrapy_demo.middlewares.UAMiddleware': 544,
}
```

在spider中输出请求的UA信息

```
import scrapy
from geektime_scrapy_demo.items import geektimeScrapyDemoItem


class geektimeSpider(scrapy.Spider):
    name = "MgeduItemTest"

    start_urls = [
        'https://quotes.toscrape.com/page/1/',
    ]

    def parse(self, response):
        for quote in response.css('div.quote'):
            ..............
            print(response.request.headers['User-Agent'])
            yield item
```

作业：

1.爬取代理网站`https://free.kuaidaili.com`

2.使用item设置类，其中包含 IP、PORT、类型、位置、响应速度、最后验证时间

3.爬取内容写入文件，按最后验证时间倒序排列

4.设置爬虫参数：

1.设置地理位置，例如：根据参数设置位置为 香港，则只爬取香港的代理地址

2.设置响应时间，例如：设置响应时间为 0.3秒，则爬取响应速度小于等于0.3秒的代理IP地址