我们在 Scrapy 架构中，可以看到有一个叫作 Middleware 的概念，中文翻译过来就叫作中间件，在 Scrapy 中有两种 Middleware，一种是 Spider Middleware，另一种是 Downloader Middleware，本节课我们分别来介绍下。

Spider Middleware 的用法

Spider Middleware 是介入 Scrapy 的 Spider 处理机制的钩子框架。

当 Downloader 生成 Response 之后，Response 会被发送给 Spider，在发送给 Spider 之前，Response 会首先经过 Spider Middleware 处理，当 Spider 处理生成 Item 和 Request 之后，Item 和 Request 还会经过 Spider Middleware 的处理。

Spider Middleware 有如下三个作用。

- 我们可以在 Downloader 生成的 Response 发送给 Spider 之前，也就是在 Response 发送给 Spider 之前对 Response 进行处理。
- 我们可以在 Spider 生成的 Request 发送给 Scheduler 之前，也就是在 Request 发送给 Scheduler 之前对 Request 进行处理。
- 我们可以在 Spider 生成的 Item 发送给 Item Pipeline 之前，也就是在 Item 发送给 Item Pipeline 之前对 Item 进行处理。

使用说明

需要说明的是，Scrapy 其实已经提供了许多 Spider Middleware，它们被 SPIDER_MIDDLEWARES_BASE 这个变量所定义。

SPIDER_MIDDLEWARES_BASE 变量的内容如下：

```
{
    'scrapy.spidermiddlewares.httperror.HttpErrorMiddleware': 50,
    'scrapy.spidermiddlewares.offsite.OffsiteMiddleware': 500,
    'scrapy.spidermiddlewares.referer.RefererMiddleware': 700,
    'scrapy.spidermiddlewares.urllength.UrlLengthMiddleware': 800,
    'scrapy.spidermiddlewares.depth.DepthMiddleware': 900,
}
```

和 Downloader Middleware 一样，Spider Middleware 首先加入 SPIDER_MIDDLEWARES 的设置中，该设置会和 Scrapy 中 SPIDER_MIDDLEWARES_BASE 定义的 Spider Middleware 合并。然后根据键值的数字优先级排序，得到一个有序列表。第一个 Middleware 是最靠近引擎的，最后一个 Middleware 是最靠近 Spider 的。

核心方法

Scrapy 内置的 Spider Middleware 为 Scrapy 提供了基础的功能。如果我们想要扩展其功能，只需要实现某几个方法即可。

每个 Spider Middleware 都定义了以下一个或多个方法的类，核心方法有如下 4 个。

- process_spider_input(response, spider)
- process_spider_output(response, result, spider)
- process_spider_exception(response, exception, spider)
- process_start_requests(start_requests, spider)

只需要实现其中一个方法就可以定义一个 Spider Middleware。下面我们来看看这 4 个方法的详细用法。

process_spider_input(response, spider)

当 Response 通过 Spider Middleware 时，该方法被调用，处理该 Response。

方法的参数有两个：

- response，即 Response 对象，即被处理的 Response；
- spider，即 Spider 对象，即该 response 对应的 Spider。

process_spider_input() 应该返回 None 或者抛出一个异常。

- 如果其返回 None，Scrapy 将会继续处理该 Response，调用所有其他的 Spider Middleware 直到 Spider 处理该 Response。
- 如果其抛出一个异常，Scrapy 将不会调用任何其他 Spider Middleware 的 process_spider_input() 方法，并调用 Request 的 errback() 方法。 errback 的输出将会以另一个方向被重新输入到中间件中，使用 process_spider_output() 方法来处理，当其抛出异常时则调用 process_spider_exception() 来处理。

process_spider_output(response, result, spider)

当 Spider 处理 Response 返回结果时，该方法被调用。

方法的参数有三个：

- response，即 Response 对象，即生成该输出的 Response；
- result，包含 Request 或 Item 对象的可迭代对象，即 Spider 返回的结果；

- spider，即 Spider 对象，即其结果对应的 Spider。

process_spider_output() 必须返回包含 Request 或 Item 对象的可迭代对象。

process_spider_exception(response, exception, spider)

当 Spider 或 Spider Middleware 的 process_spider_input() 方法抛出异常时， 该方法被调用。

方法的参数有三个：

- response，即 Response 对象，即异常被抛出时被处理的 Response；
- exception，即 Exception 对象，被抛出的异常；
- spider，即 Spider 对象，即抛出该异常的 Spider。

process_spider_exception() 必须返回结果，要么返回 None ， 要么返回一个包含 Response 或 Item 对象的可迭代对象。

- 如果其返回 None ，Scrapy 将继续处理该异常，调用其他 Spider Middleware 中的 process_spider_exception() 方法，直到所有 Spider Middleware 都被调用。
- 如果其返回一个可迭代对象，则其他 Spider Middleware 的 process_spider_output() 方法被调用， 其他的 process_spider_exception() 将不会被调用。

process_start_requests(start_requests, spider)

该方法以 Spider 启动的 Request 为参数被调用，执行的过程类似于 process_spider_output() ，只不过其没有相关联的 Response 并且必须返回 Request。

方法的参数有两个：

- start_requests，即包含 Request 的可迭代对象，即 Start Requests；
- spider，即 Spider 对象，即 Start Requests 所属的 Spider。

其必须返回另一个包含 Request 对象的可迭代对象。

Downloader Middleware 的用法

Downloader Middleware 即下载中间件，它是处于 Scrapy 的 Request 和 Response 之间的处理模块。

Scheduler 从队列中拿出一个 Request 发送给 Downloader 执行下载，这个过程会经过 Downloader Middleware 的处理。另外，当 Downloader 将 Request 下载完成得到 Response 返回给 Spider 时会再次经过 Downloader Middleware 处理。

也就是说，Downloader Middleware 在整个架构中起作用的位置是以下两个。

- 在 Scheduler 调度出队列的 Request 发送给 Downloader 下载之前，也就是我们可以在 Request 执行下载之前对其进行修改。
- 在下载后生成的 Response 发送给 Spider 之前，也就是我们可以在生成 Resposne 被 Spider 解析之前对其进行修改。

Downloader Middleware 的功能十分强大，修改 User-Agent、处理重定向、设置代理、失败重试、设置 Cookies 等功能都需要借助它来实现。下面我们来了解一下 Downloader Middleware 的详细用法。

使用说明

需要说明的是，Scrapy 其实已经提供了许多 Downloader Middleware，比如负责失败重试、自动重定向等功能的 Middleware，它们被 DOWNLOADER_MIDDLEWARES_BASE 变量所定义。

DOWNLOADER_MIDDLEWARES_BASE 变量的内容如下所示：

```
{
    'scrapy.downloadermiddlewares.robotstxt.RobotsTxtMiddleware': 100,
    'scrapy.downloadermiddlewares.httpauth.HttpAuthMiddleware': 300,
    'scrapy.downloadermiddlewares.downloadtimeout.DownloadTimeoutMiddleware': 350,
    'scrapy.downloadermiddlewares.defaultheaders.DefaultHeadersMiddleware': 400,
    'scrapy.downloadermiddlewares.useragent.UserAgentMiddleware': 500,
    'scrapy.downloadermiddlewares.retry.RetryMiddleware': 550,
    'scrapy.downloadermiddlewares.ajaxcrawl.AjaxCrawlMiddleware': 560,
    'scrapy.downloadermiddlewares.redirect.MetaRefreshMiddleware': 580,
    'scrapy.downloadermiddlewares.httpcompression.HttpCompressionMiddleware': 590,
    'scrapy.downloadermiddlewares.redirect.RedirectMiddleware': 600,
    'scrapy.downloadermiddlewares.cookies.CookiesMiddleware': 700,
    'scrapy.downloadermiddlewares.httpproxy.HttpProxyMiddleware': 750,
    'scrapy.downloadermiddlewares.stats.DownloaderStats': 850,
    'scrapy.downloadermiddlewares.httpcache.HttpCacheMiddleware': 900,
}
```

这是一个字典格式，字典的键名是 Scrapy 内置的 Downloader Middleware 的名称，键值代表了调用的优先级，优先级是一个数字，数字越小代表越靠近 Scrapy 引擎，数字越大代表越靠近 Downloader。每个 Downloader Middleware 都可以定义 process_request() 和 request_response() 方法来分别处理请求和响应，对于 process_request() 方法来说，优先级数字越小越先被调用，对于 process_response() 方法来说，优先级数字越大越先被调用。

如果自己定义的 Downloader Middleware 要添加到项目里，DOWNLOADER_MIDDLEWARES_BASE 变量不能直接修改。Scrapy 提供了另外一个设置变量 DOWNLOADER_MIDDLEWARES，我们直接修改这个变量就可以添加自己定义的 Downloader Middleware，以及禁用 DOWNLOADER_MIDDLEWARES_BASE 里面定义的 Downloader Middleware。下面我们具体来看看 Downloader Middleware 的使用方法。

核心方法

Scrapy 内置的 Downloader Middleware 为 Scrapy 提供了基础的功能，但在项目实战中我们往往需要单独定义 Downloader Middleware。不用担心，这个过程非常简单，我们只需要实现某几个方法即可。

每个 Downloader Middleware 都定义了一个或多个方法的类，核心的方法有如下三个。

- process_request(request, spider)
- process_response(request, response, spider)
- process_exception(request, exception, spider)

我们只需要实现至少一个方法，就可以定义一个 Downloader Middleware。下面我们来看看这三个方法的详细用法。

process_request(request, spider)

Request 被 Scrapy 引擎调度给 Downloader 之前，process_request() 方法就会被调用，也就是在 Request 从队列里调度出来到 Downloader 下载执行之前，我们都可以用 process_request() 方法对 Request 进行处理。方法的返回值必须为 None、Response 对象、Request 对象之一，或者抛出 IgnoreRequest 异常。

process_request() 方法的参数有如下两个。

- request，即 Request 对象，即被处理的 Request；
- spider，即 Spider 对象，即此 Request 对应的 Spider。

返回类型不同，产生的效果也不同。下面归纳一下不同的返回情况。

- 当返回为 None 时，Scrapy 将继续处理该 Request，接着执行其他 Downloader Middleware 的 process_request() 方法，直到 Downloader 把 Request 执行后得到 Response 才结束。这个过程其实就是修改 Request 的过程，不同的 Downloader Middleware 按照设置的优先级顺序依次对 Request 进行修改，最后推送至 Downloader 执行。
- 当返回为 Response 对象时，更低优先级的 Downloader Middleware 的 process_request() 和 process_exception() 方法就不会被继续调用，每个 Downloader Middleware 的 process_response() 方法转而被依次调用。调用完毕之后，直接将 Response 对象发送给 Spider 来处理。
- 当返回为 Request 对象时，更低优先级的 Downloader Middleware 的 process_request() 方法会停止执行。这个 Request 会重新放到调度队列里，其实它就是一个全新的 Request，等待被调度。如果被 Scheduler 调度了，那么所有的 Downloader Middleware 的 process_request() 方法会被重新按照顺序执行。
- 如果 IgnoreRequest 异常抛出，则所有的 Downloader Middleware 的 process_exception() 方法会依次执行。如果没有一个方法处理这个异常，那么 Request 的 errorback() 方法就会回调。如果该异常还没有被处理，那么它便会被忽略。

process_response(request, response, spider)

Downloader 执行 Request 下载之后，会得到对应的 Response。Scrapy 引擎便会将 Response 发送给 Spider 进行解析。在发送之前，我们都可以用 process_response() 方法来对 Response 进行处理。方法的返回值必须为 Request 对象、Response 对象之一，或者抛出 IgnoreRequest 异常。

process_response() 方法的参数有如下三个。

- request，是 Request 对象，即此 Response 对应的 Request。
- response，是 Response 对象，即此被处理的 Response。
- spider，是 Spider 对象，即此 Response 对应的 Spider。

下面对不同的返回情况做一下归纳：

- 当返回为 Request 对象时，更低优先级的 Downloader Middleware 的 process_response() 方法不会继续调用。该 Request 对象会重新放到调度队列里等待被调度，它相当于一个全新的 Request。然后，该 Request 会被 process_request() 方法顺次处理。
- 当返回为 Response 对象时，更低优先级的 Downloader Middleware 的 process_response() 方法会继续调用，继续对该 Response 对象进行处理。
- 如果 IgnoreRequest 异常抛出，则 Request 的 errorback() 方法会回调。如果该异常还没有被处理，那么它便会被忽略。

process_exception(request, exception, spider)

当 Downloader 或 process_request() 方法抛出异常时，例如抛出 IgnoreRequest 异常，process_exception() 方法就会被调用。方法的返回值必须为 None、Response 对象、Request 对象之一。

process_exception() 方法的参数有如下三个。

- request，即 Request 对象，即产生异常的 Request。
- exception，即 Exception 对象，即抛出的异常。

- spdier，即 Spider 对象，即 Request 对应的 Spider。

下面归纳一下不同的返回值。

- 当返回为 None 时，更低优先级的 Downloader Middleware 的 process_exception() 会被继续顺次调用，直到所有的方法都被调度完毕。
- 当返回为 Response 对象时，更低优先级的 Downloader Middleware 的 process_exception() 方法不再被继续调用，每个 Downloader Middleware 的 process_response() 方法转而被依次调用。
- 当返回为 Request 对象时，更低优先级的 Downloader Middleware 的 process_exception() 也不再被继续调用，该 Request 对象会重新放到调度队列里面等待被调度，它相当于一个全新的 Request。然后，该 Request 又会被 process_request() 方法顺次处理。

以上内容便是这三个方法的详细使用逻辑。在使用它们之前，请先对这三个方法的返回值的处理情况有一个清晰的认识。在自定义 Downloader Middleware 的时候，也一定要注意每个方法的返回类型。

下面我们用一个案例实战来加深一下对 Downloader Middleware 用法的理解。

项目实战

新建一个项目，命令如下所示：

```
scrapy startproject scrapydownloadertest
```

新建了一个 Scrapy 项目，名为 scrapydownloadertest。进入项目，新建一个 Spider，命令如下所示：

```
scrapy genspider httpbin httpbin.org
```

新建了一个 Spider，名为 httpbin，源代码如下所示：

```
import scrapy
class HttpbinSpider(scrapy.Spider):
    name = 'httpbin'
    allowed_domains = ['httpbin.org']
    start_urls = ['http://httpbin.org/']

    def parse(self, response):
        pass
```

接下来我们修改 start_urls 为：`['http://httpbin.org/']`。随后将 parse() 方法添加一行日志输出，将 response 变量的 text 属性输出，这样我们便可以看到 Scrapy 发送的 Request 信息了。
修改 Spider 内容如下所示：

```
import scrapy

class HttpbinSpider(scrapy.Spider):
    name = 'httpbin'
    allowed_domains = ['httpbin.org']
    start_urls = ['http://httpbin.org/get']

    def parse(self, response):
        self.logger.debug(response.text)
```

接下来运行此 Spider，执行如下命令：

```
scrapy crawl httpbin
```

Scrapy 运行结果包含 Scrapy 发送的 Request 信息，内容如下所示：

```
{"args": {},
  "headers": {
    "Accept": "text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8",
    "Accept-Encoding": "gzip,deflate,br",
    "Accept-Language": "en",
    "Connection": "close",
    "Host": "httpbin.org",
    "User-Agent": "Scrapy/1.4.0 (+http://scrapy.org)"
  },
  "origin": "60.207.237.85",
  "url": "http://httpbin.org/get"
}
```

我们观察一下 Headers，Scrapy 发送的 Request 使用的 User-Agent 是 Scrapy/1.4.0(+http://scrapy.org)，这其实是由 Scrapy 内置的 UserAgentMiddleware 设置的，UserAgentMiddleware 的源码如下所示：

```
from scrapy import signals

class UserAgentMiddleware(object):
    def __init__(self, user_agent='Scrapy'):
        self.user_agent = user_agent

    @classmethod
    def from_crawler(cls, crawler):
        o = cls(crawler.settings['USER_AGENT'])
        crawler.signals.connect(o.spider_opened, signal=signals.spider_opened)
```

```
        return o

    def spider_opened(self, spider):
        self.user_agent = getattr(spider, 'user_agent', self.user_agent)

    def process_request(self, request, spider):
        if self.user_agent:
            request.headers.setdefault(b'User-Agent', self.user_agent)
```

在 from_crawler() 方法中，首先尝试获取 settings 里面的 USER_AGENT，然后把 USER_AGENT 传递给 **init**() 方法进行初始化，其参数就是 user_agent。如果没有传递 USER_AGENT 参数就默认设置为 Scrapy 字符串。我们新建的项目没有设置 USER_AGENT，所以这里的 user_agent 变量就是 Scrapy。接下来，在 process_request() 方法中，将 user-agent 变量设置为 headers 变量的一个属性，这样就成功设置了 User-Agent。因此，User-Agent 就是通过此 Downloader Middleware 的 process_request() 方法设置的。
修改请求时的 User-Agent 可以有两种方式：一是修改 settings 里面的 USER_AGENT 变量；二是通过 Downloader Middleware 的 process_request() 方法来修改。

第一种方法非常简单，我们只需要在 setting.py 里面加一行 USER_AGENT 的定义即可：

```
USER_AGENT = 'Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_6) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/59.0.3071.115 Safari/537.36'
```

一般推荐使用此方法来设置。但是如果想设置得更灵活，比如设置随机的 User-Agent，那就需要借助 Downloader Middleware 了。所以接下来我们用 Downloader Middleware 实现一个随机 User-Agent 的设置。

在 middlewares.py 里面添加一个 RandomUserAgentMiddleware 的类，如下所示：

```
import random

class RandomUserAgentMiddleware():
    def __init__(self):
        self.user_agents = ['Mozilla/5.0 (Windows; U; MSIE 9.0; Windows NT 9.0; en-US)',
            'Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.2 (KHTML, like Gecko) Chrome/22.0.1216.0 Safari/537.2',
            'Mozilla/5.0 (X11; Ubuntu; Linux i686; rv:15.0) Gecko/20100101 Firefox/15.0.1'
        ]

    def process_request(self, request, spider):
        request.headers['User-Agent'] = random.choice(self.user_agents)
```

我们首先在类的 __init__() 方法中定义了三个不同的 User-Agent，并用一个列表来表示。接下来实现了 process_request() 方法，它有一个参数 request，我们直接修改 request 的属性即可。在这里我们直接设置了 request 对象的 headers 属性的 User-Agent，设置内容是随机选择的 User-Agent，这样一个 Downloader Middleware 就写好了。

不过，要使之生效我们还需要再去调用这个 Downloader Middleware。在 settings.py 中，将 DOWNLOADER_MIDDLEWARES 取消注释，并设置成如下内容：

```
DOWNLOADER_MIDDLEWARES = {'scrapydownloadertest.middlewares.RandomUserAgentMiddleware': 543,}
```

接下来我们重新运行 Spider，就可以看到 User-Agent 被成功修改为列表中所定义的随机的一个 User-Agent 了：

```
{"args": {},
  "headers": {
    "Accept": "text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8",
    "Accept-Encoding": "gzip,deflate,br",
    "Accept-Language": "en",
    "Connection": "close",
    "Host": "httpbin.org",
    "User-Agent": "Mozilla/5.0 (Windows; U; MSIE 9.0; Windows NT 9.0; en-US)"
  },
  "origin": "60.207.237.85",
  "url": "http://httpbin.org/get"
}
```

我们就通过实现 Downloader Middleware 并利用 process_request() 方法成功设置了随机的 User-Agent。

另外，Downloader Middleware 还有 process_response() 方法。Downloader 对 Request 执行下载之后会得到 Response，随后 Scrapy 引擎会将 Response 发送回 Spider 进行处理。但是在 Response 被发送给 Spider 之前，我们同样可以使用 process_response() 方法对 Response 进行处理。比如这里修改一下 Response 的状态码，在 RandomUserAgentMiddleware 添加如下代码：

```
def process_response(self, request, response, spider):
    response.status = 201
    return response
```

我们将 response 对象的 status 属性修改为 201，随后将 response 返回，这个被修改后的 Response 就会被发送到 Spider。

我们再在 Spider 里面输出修改后的状态码，在 parse() 方法中添加如下的输出语句：

```
self.logger.debug('Status Code: ' + str(response.status))
```

重新运行之后，控制台输出了如下内容：

```
[httpbin] DEBUG: Status Code: 201
```

可以发现，Response 的状态码成功修改了。因此要想对 Response 进行处理，就可以借助于 process_response() 方法。

另外还有一个 process_exception() 方法，它是用来处理异常的方法。如果需要异常处理的话，我们可以调用此方法。不过这个方法的使用频率相对低一些，在此不用实例演示。

本节代码

本节源代码为：
https://github.com/Python3WebSpider/ScrapyDownloaderTest。

结语

本节讲解了 Spider Middleware 和 Downloader Middleware 的基本用法。利用它们我们可以方便地实现爬虫逻辑的灵活处理，需要好好掌握。