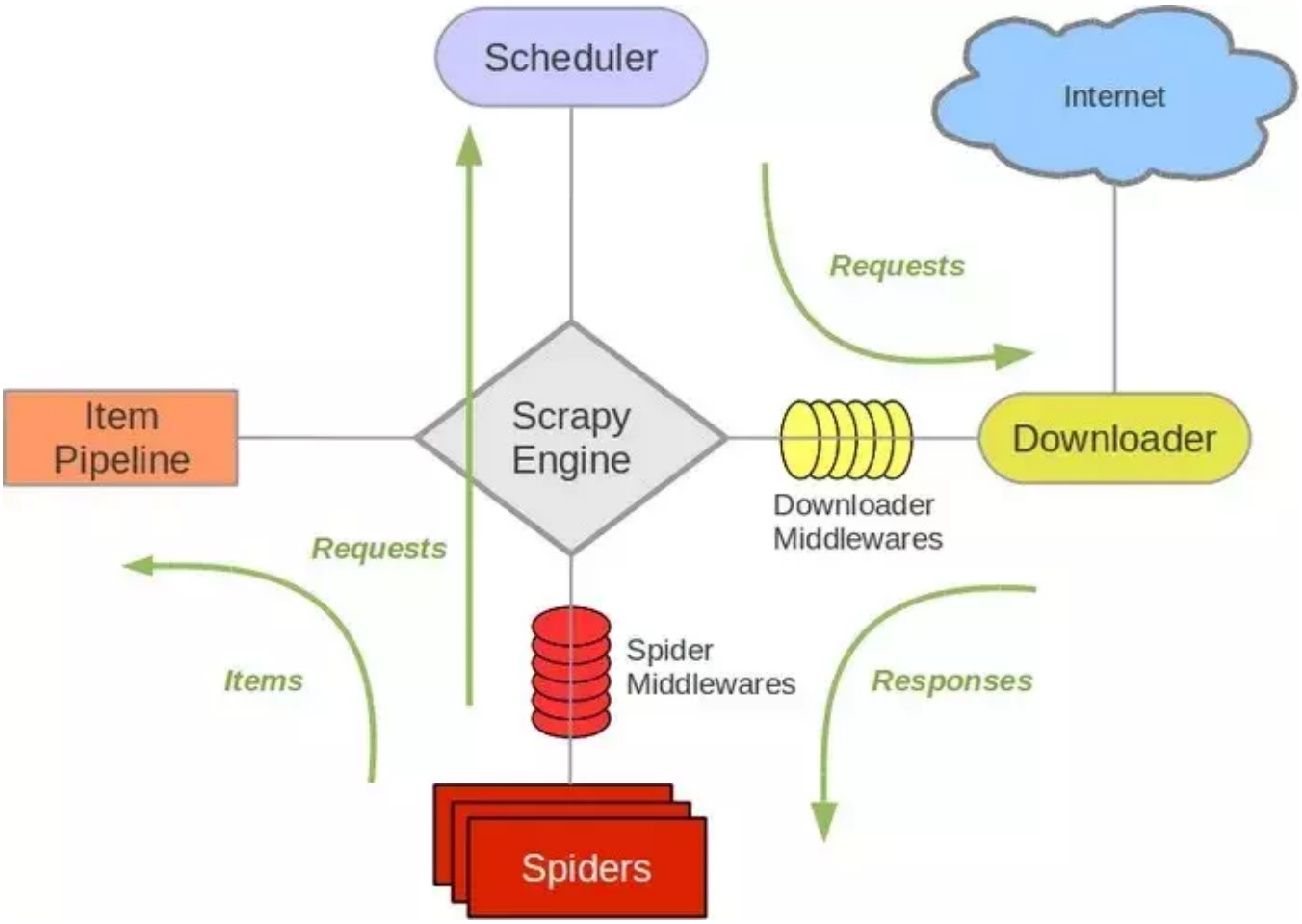
崔庆才丨静觅
2018年05月09日 阅读 581

Scrapy框架的使用之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。我们首先来看看它的架构，如下图所示。



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Doanloader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。

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。

一、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，数字小的Downloader Middleware会被优先调用。

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。

二、核心方法

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。

1. 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` ，是Request对象，即被处理的Request。
- `spider` ，是Spdier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。

- 当返回是None时，Scrapy将继续处理该Request，接着执行其他Downloader Middleware的 `process_request()` 方法，一直到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

- 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。

2. 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。

下面归纳一下不同的返回情况。

- 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。

3. 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。

- exception，是Exception对象，即抛出的异常。
- spdier，是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。

三、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'
```

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

接下来我们修改 `start_urls` 为： `[http://httpbin.org/](http://httpbin.org/)` 。随后将 `parse()` 方法添加一行日志输出，将 `response` 变量的 `text` 属性输出出来，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"
}
```

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 `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, li
```

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：

```
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
            '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` 变量的 `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"
}
```

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` 变量的 `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进行后处理，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。

四、本节代码

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

五、结语