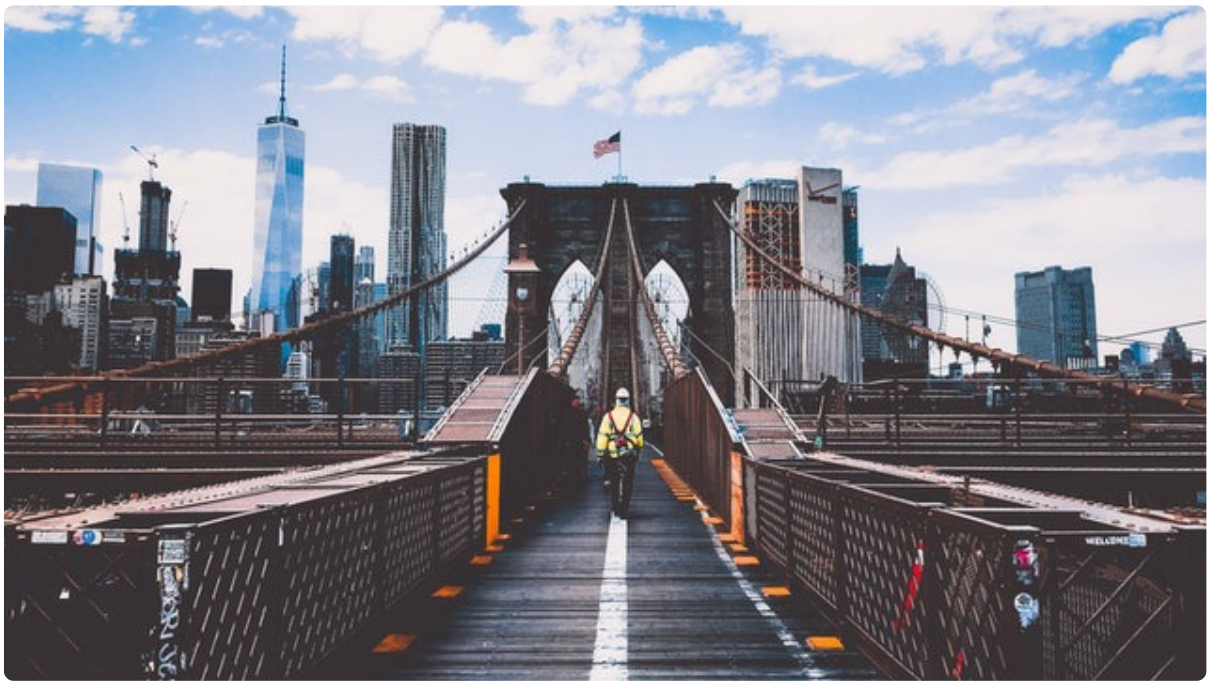
05 Python 世界最流行的网络爬虫框架 Scrapy

更新时间：2019-07-03 20:08:02



> 从不浪费时间的人，没有工夫抱怨时间不够。
>
> ——杰弗逊

本节内容会带你初步了解构建网络爬虫的优秀框架 Scrapy，迈出从原始人式的手写爬虫步入工程化、专业化爬虫的第一步。

- scrapy 简介 —— 不讲结构只讲作用（简单）重点讲讲并行计算能力
- scrapy 的安装
- scrapy 的最基本命令`startproject` 的使用，构建 scrapy 项目结构
- scrapy 项目结构说明，scrapy 文件与目录的作用
- scrapy 其它常用指令的介绍

scrapy 简介

Scrapy 是一个基于Twisted的异步处理框架，是纯 Python 实现的爬虫框架，其架构清晰，**模块之间的耦合程度低，可扩展性极强**，可以灵活完成各种需求。Scrapy 算得上是 Python 世界中最多人用的爬虫框架了，在时下流行的多种语言体系中属于最好的爬虫框架！但我认为它也是最难学习的框架之一。很多初学 Scarpy 的童鞋经常向我抱怨完全不清楚 Scrapy 该怎样入手，即使看的是中文的文档，也感到很难理解。我当初接触 Scrapy 时也有这样的感觉。之所以感到Scrapy难学，究其原因，是其官方文档实在太过凌乱，又缺少实用的代码例子，让人看得云里雾里，不知其所已然。

虽然其文档不良，但却没有遮挡住它的光辉，它依然是 Python 世界中最好用的爬虫框架。其架构的设计思路、爬虫执行的效能，还有可扩展的能力都非常出众，再配以 Python 语言的简洁轻巧，使得爬虫的开发事半功倍。

Scrapy的优点

Scrapy 的最大特点是它属于一种全家桶式的框架，在架构上很特别，都是基于插件式的增量开发模式，而且它的并行运行能力非常出众，由于这些特点 Scrapy 在正儿八经的爬虫工程中被广泛应用。以下是一些Scrapy在技术上具体优点的汇总：

- 提供了内置的HTTP缓存，以加速本地开发。
- 提供了自动节流调节机制，而且具有遵守robots.txt的设置的能力。
- 可以定义爬行深度的限制，以避免爬虫进入死循环链接。
- 会自动保留会话。
- 执行自动HTTP基本认证。 不需要明确保存状态。
- 可以自动填写登录表单。
- Scrapy 有一个内置的中间件，可以自动设置请求中的引用（referer）头。
- 支持通过3xx响应重定向，也可以通过HTML元刷新。
- 避免被网站使用的`<noscript>` meta重定向困住，以检测没有JS支持的页面。
- 默认使用 CSS 选择器或XPath编写解析器。
- 可以通过Splash或任何其他技术（如Selenium）呈现JavaScript页面。
- 拥有强大的社区支持、丰富的插件和扩展来扩展其功能。
- 提供了通用的蜘蛛来抓取最常见的格式：站点地图、CSV和XML。
- 内置支持以多种格式（JSON、CSV、XML、JSON-lines）导出收集的数据，并将其存储在多个后端（FTP、S3、本地文件系统）中。

Scrapy 的缺点

- 由于官方文档很乱，学习曲线相对陡峭，对于初学者不容易上手
- 配置项目繁多，使用起来复杂

Scrapy与pySpider的对比

拿 Scrapy 的优点与缺点相比会发现它确实是一个不错的选择，尤其将其作为正式项目的框架选择。时下也出现了不少的与之争锋的小型框架，比较出风头的可数pySpider， 在撰写本专栏之前我还专门对它作了一番深入的研究。与 Scrapy 相比 pySpider 拥有更直观的架构和更为平坦的学习曲线，对于一些简单的爬虫项目或者完全没有爬虫理论基础的同学确实也是一个值得学习的框架，但对于数据量庞大，爬取逻辑相对复杂（反爬）的项目时我还是倾向于 Scrapy，它已经历了多年被实用性验证，而且也拥有极为庞大的社区资源，在面对各种爬虫项目所涉及到的问题都会有相应的解决办法。

本专栏我之所以全部基于 Scrapy 是因为考虑到读者从中可以学习到的内容能被真正地用在实际的工作中，由于Scrapy所覆盖的问题域较广，对于我们在掌握背后的理论也会有很大的帮助作用。

scrapy 的安装

Mac下安装Scrapy

Scrapy 的安装非常简单，在 Python2.x 的虚环境内直接执行以下的指令即可：

```
(venv) macOS $ pip install scrapy
```

Scrapy 被一直诟病不支持 Python3, 但最近Scrapy 也作出了更新，推出了 Python3 的版本，可以按以下的方法来安装：

```
$ virtualenv -p Python3 venv # 安装python3虚环境
$ . venv/bin/activate        # 激活虚环境
(venv) $ pip install scrapy3 # 安装scrapy for Python3版本
```

scrapy 安装成功后除了提供它的基础类库以外还有方便实用的命令行工具(cli)，接下来的内容将会再介绍一些常用的命令行工具。

Windows 下安装Scrapy

如果直接安装：

```
c:\pip install scrapy
```

很大可能会出现`Microsoft Visual C++ 14.0 is required. Get it with "Microsoft Visual C++ Build Tools": http://landinghub.visualstudio.com/visual-cpp-build-tools`的错误，下面是解决方法：

1. 安装Wheel

```
c:\ pip install wheel
```

2. 安装Twisted

你可以在下面的链接中选中与你操作系统及 Python 版本相符的Twisted安装包。下面链接引用至pythonlibs这个网址下载与你环境对应的Twisted的whl包。

Twisted, an event-driven networking engine.

- Twisted 19.2.0 for python2.7 & 32位 Windows
- Twisted 19.2.0 for python2.7 & 64位 Windows
- Twisted 19.2.0 for python3.5 & 32位 Windows
- Twisted 19.2.0 for python3.5 & 64位 Windows
- Twisted 19.2.0 for python3.6 & 32位 Windows
- Twisted 19.2.0 for python3.6 & 64位 Windows
- Twisted 19.2.0 for python3.7 & 32位 Windows
- Twisted 19.2.0 for python3.7 & 64位 Windows

下载后进入whl文件所在路径，执行以下的指令(以下为python3.6及windows64位版本的whl):

```
c:\pip install Twisted 19.2.0 cp36 cp36m win_amd64.whl
```

3. 安装scrapy

下载Scrapy 1.6.0 py2.py3 none any.whl 并进入whl包所在路径之后执行：

```
c:\pip install Scrapy-1.6.0-py2.py3-none-any.whl
```

完成安装后在命令行执行:

```
c:\scrapy -h
```

如果能正常输出scrapy的命令帮助信息则表明安装成功。

4. 运行scrapy项目

如果安装完成后在运行scrapy项目时报以下的错：

```
no module named win32api
```

则表示Windows系统中，python因为原生并不能访问windows的api，所以需要额外安装pypiwin32库。

```
pip install pypiwin32
```

如果不想这么折腾那就直接换掉操作系统，毕竟python在linux与unix环境才是天生强大的存在。

构建 scrapy 项目结构

scrapy 的项目对目录结构有严格要求，为了简化这个构建指定目录的重复性过程，我们可以使用scrapy内置的`startproject`指令来快速构建scrapy项目。

`startproject`的指令模式如下:

```
$ scrapy startproject <项目名称>
```

项目名称必须以英文开头，多个单词之间可以用下划线作(`_`)为连接符，切记不要采用`-`否则会报错，下面的输出就是使用了`-`作为项目名称：

```
Error: Project names must begin with a letter and contain only letters, numbers and underscores
```

首先，我们创建一个名为"my_crawler"的爬虫项目：

```
(venv) macOS$ scrapy startproject my_crawler
```

当`startproject`被成功执行，将会在控制台看到以下的输出信息:

```
(venv) macOS:scrapy-courses raymacbook$ scrapy startproject my_crawler
New Scrapy project 'my_crawler', using template directory '/Users/ray/Projects/scrapy-courses/venv/lib/python3.7/site-packa
ges/scrapy/templates/project', created in:
    /Users/ray/Projects/scrapy-courses/my_crawler

You can start your first spider with:
    cd my_crawler
    scrapy genspider example example.com
(venv) macOS:scrapy-courses macbook$
```

上述的输出提示的意思是我们可以进入`my_crawler`目录并执行`scrapy genspider 项目名 目标网站地址`创建scrapy项目中的爬虫类。在此之前先让我们来了解`startproject` 指令为我们创建的哪些文件、目录，以及它们的作用是什么。

此时我们就可以开始scrapy的编程了。

项目结构说明

展开"my_crawler"目录可以看到以下的目录结构：

```
my_crawler
├── my_crawler        # 爬虫项目包目录
│   ├── __init__.py
│   ├── items.py        # Item定义文件
│   ├── middlewares.py  # 自定义中间件
│   ├── pipelines.py    # 自定义管道
│   ├── settings.py     # 项目配置文件
│   └── spiders         # 蜘蛛类的存放目录
│       └── __init__.py
└── scrapy.cfg          # Scrapy 运行配置
```

一开始你可能会对 Scrapy 为什么建立一个与项目同名的源码目录感到奇怪，在最初接触 Scrapy时我也对此表示过疑惑。直到使用了一段时间以后才发现这种结构还是相当合理的。作者当时可能是出于以下几点的考虑：

- 打包python工程生成安装文件，相应的"setup.py"与编辑目录会与源码目录置于同层
- 打包后发布工程至pypi(python官方的安装包库)时项目名称可以与源码目录保持一至

为了不产生不必要的麻烦，我建议你在没有完全对Scrapy了解之前不要按照个人喜好来修改目录与文件的命名。毕竟我们是在别人的思路上进行开发，所以还是先按照官方所制定的规则来学习，这样的话会少走很多弯路。

scrapy 其它常用指令的介绍

Scrapy 给我们提供了两种类型的命令：项目命令（Project-specific）需要进入项目目录执行才可生效，全局命令则不需要进入项目。

所有的命令行工具都会以如下的调用格式提供：

```
$ scrapy <命令> [选项] [参数]
```

全局指令

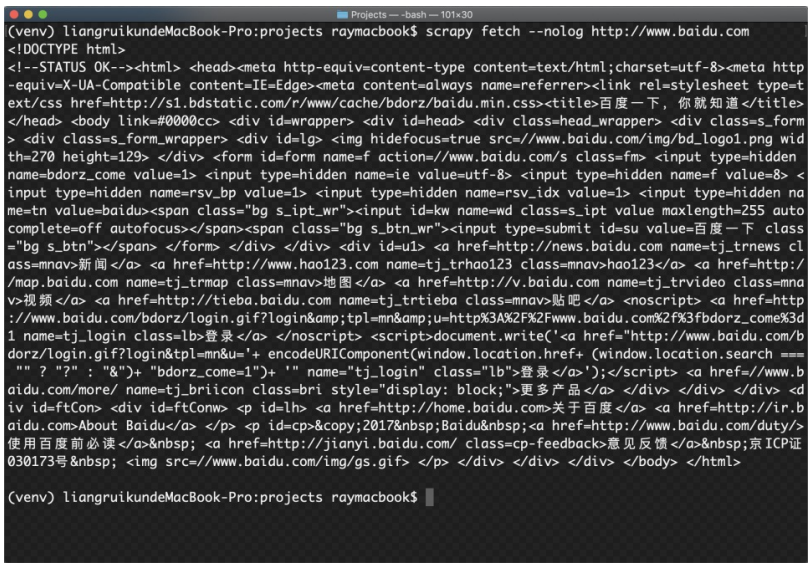
Scrapy 提供的全局命令(只能在 Scrapy 项目目录之外运行)主要包括以下这些:

- `scrapy startproject <项目名>` - 初始化爬虫项目，创建基本目录与必要的文件。
- `scrapy settings --get <配置项的名称>` - 在项目中运行时，该命令将会输出项目的设定值，否则输出Scrapy默认设定。
- `scrapy runspider <spider_file.py>` - 在未创建项目的情况下，运行一个编写在Python文件中的spider。
- `scrapy shell <url>` - 以给定的URL启动Scrapy指令外壳,可以在命令行中直接操作scrapy内置的python对象，我觉得并没有什么实用性，这些运行期变量我们都可以借助pyCharm的调试器看到,官方文档可参考Scrapy终端(Scrapy shell)。
- `scrapy fetch <url>` - 使用Scrapy下载器(downloader)下载给定的URL，并将获取到的内容打印到控制台上。
- `scrapy view <url>` - 在浏览器中打开给定的URL，并用一个Scrapy spider获取目标URL，并将结构显示到浏览器中。
- `scrapy version [-v]` - 输出Scrapy版本。配合 -v 运行时，该命令同时输出Python, Twisted以及平台的信息，方便bug提交。

除了 `startproject` 是常用命令之外，只有 `fetch` 和 `view` 指命比较有用，其它的指令作用不大。

`fetch` 指令

这个指令可以理解为`curl`指令的最简化版本，就是直接从指定的 url 上下载网页内容，然后就输出到控制台,效果如下：



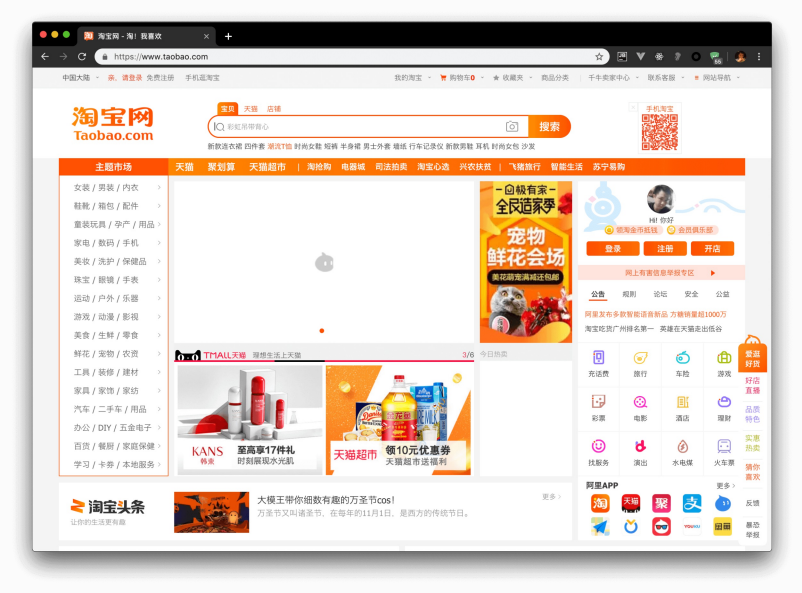
可见，它的输出只不过是杂乱无章的一大堆内容，实用性很差。但如果加入另一个参数`--headers`这个指令就变得有用得多：

```
$ scrapy fetch --nolog --headers http://www.baidu.com
> Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
> Accept-Language: en
> User-Agent: Scrapy/1.0.1 (+https://scrapy.org)
> Accept-Encoding: gzip,deflate
>
< Cache-Control: private, no-cache, no-store, proxy-revalidate, no-transform
< Content-Type: text/html
< Date: Sat, 20 Apr 2019 08:19:34 GMT
< Last-Modified: Mon, 23 Jan 2017 13:27:57 GMT
< Pragma: no-cache
< Server: bfe/1.0.8.18
< Set-Cookie: BDORZ=27315; max-age=86400; domain=.baidu.com; path=/
```

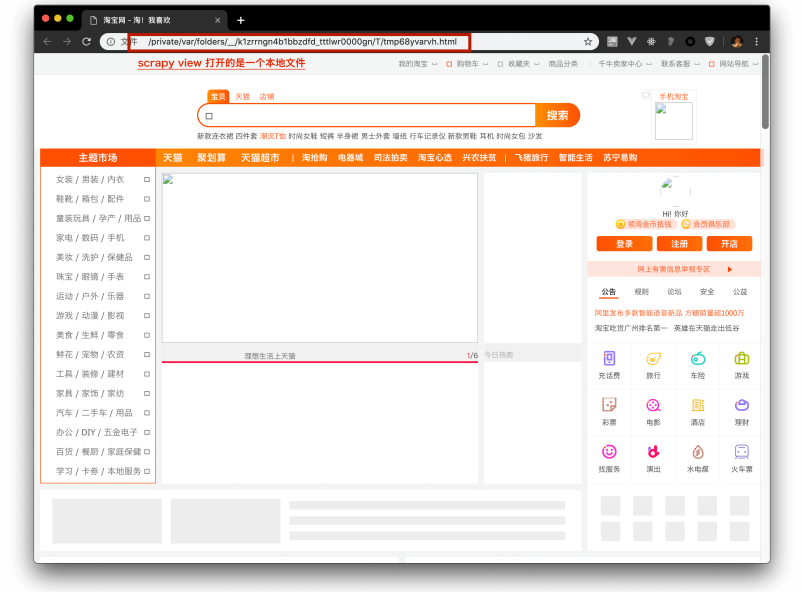
使用`--headers`之后我们可以很快速地看到 Scrapy 向目标网页发出的请求头与响应头的详细内容，这对于我们进行爬网分析与设计会起到很大的作用。本专栏后续的章节中也会采用此命令来进行分析，此时你也可以动手试试对不同网址的请求与返回的结果是什么样子的。

`view` 指令

用对象向目标网站发出请求返回的内容，很多时候与我们直接打开浏览器所看到的并不一样，尤其是那些重度运用javascript技术的网站，`view`指令就是直接用 Scrapy 内置的`Request`对象向目标网站发出请求，然后将返回内容存到本地的临时文件中再用浏览器打开。这个指令的意义在于让我们清楚地知道，用`Request`对象发出请求后到底会得到些什么，就以淘宝为例，以下第一张图是用浏览器直接打开的网页，第二张图是用`view`指令打开淘宝的效果:




```
$ scrapy view http://www.taobao.com
```



对比之后可以发现很多内容在用`Request`发起请求的时候是获取不到的。

项目（Project-only）命令:

以下这些指令是必须先进入项目目录后才能使用的命令：

- `scrapy crawl <蜘蛛名>` - 使用指定的蜘蛛执行爬取。
- `scrapy genspider` - 在当前项目中创建spider。这仅仅是创建spider的一种快捷方法。该方法可以使用提前定义好的模板来生成spider。
- `scrapy list` - 列出当前项目中有多少个可用的蜘蛛（不实用）
- `scrapy check [-l] <蜘蛛名>` - 运行contract检查。（没有什么实质作用，contract是一种非常简陋的测试机制，不推荐使用）

- `scrapy edit <蜘蛛名>` - 启用一个IDE编辑蜘蛛
- `scrapy deploy [ <target:project> | -l <target> | -L ]` - 将项目部署到Scrapyd服务
- `scrapy bench` - 运行benchmark测试

`crawl` 指令

`crawl`指令是 Scrapy 众多内置指令中使用频次最高的一个，每次执行爬取你都需要运行它，例如:

```
$ scrapy crawl news_spider
```

此命令一旦执行，就会将所有的输出信息打印到当前终端窗口内，直至爬虫完成所有的爬取任务为止。在此不再作过多的赘述，在后面的内容里我们将会经常与它见面，当开发了具体的蜘蛛代码时再回过头来仔细学习这个命令。

`genspider` 指令

`genspider`指令是一个对于Scrapy初学者非常有用的指令，顾名思义它就是用来帮助我们生成蜘蛛代码框架构的，它的指令结构如下:

```
$ scrapy genspider [-t 模板名] <蜘蛛名> <域>
```

模板名参数提供以下4种常用的蜘蛛模板：

- basic - 默认，创建基本的蜘蛛
- crawl - 创建基于`CrawlSpider`的蜘蛛
- csvfeed - 创建基于`CsvFeedSpider`的蜘蛛
- xmlfeed - 创建基于`XmlFeedSpider`的蜘蛛

此处请容我卖个关子，先不展开对具体蜘蛛的作用的解释，后面的内容中会一一有针对性地对它们进行实践与解释。

创建一个爬取新浪新闻的普通蜘蛛:

```
$ scrapy genspider news http://www.sina.com
```

当指令执行成功后就会在 `[项目名]/[项目名]/spiders/` 目录下找到一个`news.py`的蜘蛛源码文件:

```
news_crawler
├── __init__.py
├── items.py
├── middlewares.py
├── pipelines.py
├── settings.py
└── spiders
    └── news.py
```

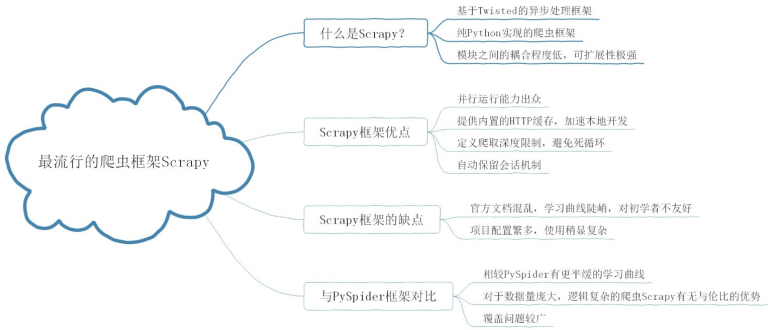
打开 `news.py`:

```
# -*- coding: utf-8 -*-
import scrapy

class NewsSpider(scrapy.Spider): # 生成蜘蛛后Scrapy会将蜘蛛名首字母大写并在其后加Spider作为蜘蛛的类名
    name = 'news'                                # 蜘蛛名
    allowed_domains = ['http://www.sina.com']   # 在genspider指定的域
    start_urls = ['http://http://www.sina.com/']

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

用`startproject`和`genspider`两个指令就已经可以创建一个最基本可以运行的爬虫项目了。

小结

至此，我们已经对 Scrapy 的安装和基本使用建立了一个基本的认识。也从 Scrapy 那纷繁复杂的指令中集中了解了哪些指令是必须学的，哪些不过是摆设。如果想在爬虫的开发路上走得更加顺畅，试试使用 Ubuntu 或者其它的 Linux 来开发，这样的话，你会更加理解为什么我用如此之长的篇幅来讲述“指令”这一强大的工具。



04 动手开发最简单的单文件爬虫

06 新闻供稿专用爬虫开发实践