**Scrapy一目了然**

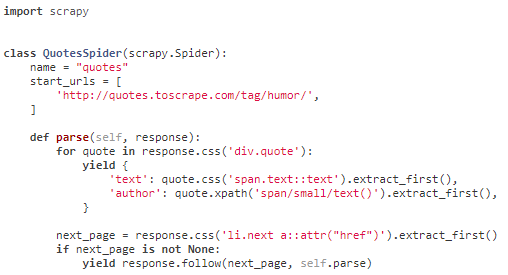
Scrapy是一个应用程序框架，用于抓取网站并提取可用于广泛的有用应用程序的结构化数据，如数据挖掘，信息处理或历史档案。

尽管Scrapy最初是为[网页抓取](https://en.wikipedia.org/wiki/Web_scraping)设计的，但它也可以用于使用API​​（如[Amazon Associates Web Services](https://affiliate-program.amazon.com/gp/advertising/api/detail/main.html)）或作为通用网络抓取工具提取数据。

**走过一个例子蜘蛛**

为了向您展示Scrapy带给您的东西，我们将使用最简单的方式来引导Scrapy Spider的一个示例来运行蜘蛛。

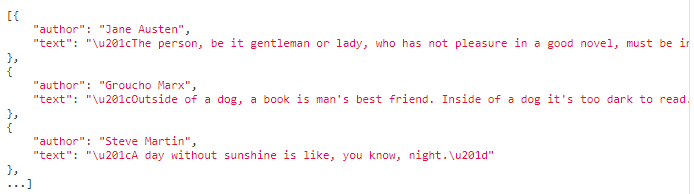
下面是一个蜘蛛的代码，它可以在分页之后从网站[http://quotes.toscrape.com](http://quotes.toscrape.com/)摘录出名言 ：



把它放在一个文本文件中，命名为类似的东西quotes\_spider.py ，然后使用以下[**runspider**](https://doc.scrapy.org/en/latest/topics/commands.html#std:command-runspider)命令运行蜘蛛：



完成此操作后，您将在quotes.json文件中包含JSON格式的引号列表，其中包含文本和作者，看起来像这样（为了更好的可读性，此处重新格式化）：



### 刚刚发生了什么？

当你运行这个命令时，Scrapy在里面寻找一个Spider定义，并通过它的履带引擎运行它。scrapy runspider quotes\_spider.py

通过向start\_urls 属性中定义的URL发送请求（在本例中，只有幽默类别中的引用URL ），并调用默认回调方法parse，将响应对象作为参数传递，从而开始爬网。在parse回调中，我们使用CSS Selector循环引用元素，产生一个带有提取的引用文本和作者的Python字典，查找指向下一页的链接，并使用与parse回调相同的方法安排另一个请求 。

这里您会注意到Scrapy的一个主要优势：请求被 [异步调度和处理](https://doc.scrapy.org/en/latest/topics/architecture.html#topics-architecture)。这意味着Scrapy不需要等待请求被完成和处理，它可以发送另一个请求或在此期间做其他事情。这也意味着即使某些请求失败或在处理错误时也可以继续执行其他请求。

虽然这可以使您快速执行爬网（同时以容错方式发送多个并发请求），但Scrapy还可以通过[几个设置](https://doc.scrapy.org/en/latest/topics/settings.html#topics-settings-ref)控制爬网的礼貌性。您可以执行诸如在每个请求之间设置下载延迟，限制每个域或每个IP的并发请求量，甚至[使用](https://doc.scrapy.org/en/latest/topics/autothrottle.html#topics-autothrottle)试图自动计算出这些值的自动[限制扩展](https://doc.scrapy.org/en/latest/topics/autothrottle.html#topics-autothrottle)。

**注意**

这是使用[Feed导出来](https://doc.scrapy.org/en/latest/topics/feed-exports.html#topics-feed-exports)生成JSON文件，您可以轻松更改导出格式（例如XML或CSV）或存储后端（例如FTP或[Amazon S3](https://aws.amazon.com/s3/)）。您还可以编写 [物料管道](https://doc.scrapy.org/en/latest/topics/item-pipeline.html#topics-item-pipeline)以将物料存储在数据库中。

## 还有什么？

您已经看过如何使用Scrapy从网站中提取和存储项目，但这只是表面。Scrapy提供了许多强大的功能，可以使抓取变得简单高效，例如：

* 内置支持，用于使用扩展CSS选择器和XPath表达式从HTML / XML源[选择和提取](https://doc.scrapy.org/en/latest/topics/selectors.html#topics-selectors)数据，并使用正则表达式提取帮助器方法。
* 一个[交互的shell控制台](https://doc.scrapy.org/en/latest/topics/shell.html#topics-shell)用于尝试的CSS和XPath表达式抽取数据，非常有用的写作时或调试蜘蛛（IPython都知道）。
* 内置支持以多种格式（JSON，CSV，XML）[生成Feed导出](https://doc.scrapy.org/en/latest/topics/feed-exports.html#topics-feed-exports)并将它们存储在多个后端（FTP，S3，本地文件系统）
* 强大的编码支持和自动检测，用于处理外部的，非标准的和破碎的编码声明。
* [强大的可扩展性支持](https://doc.scrapy.org/en/latest/index.html#extending-scrapy)，允许您使用[信号](https://doc.scrapy.org/en/latest/topics/signals.html#topics-signals)和定义良好的API（中间件，[扩展](https://doc.scrapy.org/en/latest/topics/extensions.html#topics-extensions)和 [管道](https://doc.scrapy.org/en/latest/topics/item-pipeline.html#topics-item-pipeline)）来插入自己的功能。
* 广泛的内置扩展和中间件处理：
  + cookies和会话处理
  + HTTP功能，如压缩，认证，缓存
  + 用户代理欺骗
  + 的robots.txt
  + 爬行深度限制
  + 和更多
* 一个[Telnet控制台，](https://doc.scrapy.org/en/latest/topics/telnetconsole.html#topics-telnetconsole)用于连接Scrapy进程中运行的Python控制台，以反编译和调试您的爬虫程序
* 再加上其他好东西，如可重复使用的蜘蛛，可以从[Sitemaps](https://www.sitemaps.org/index.html)和XML / CSV Feed中抓取网站，[自动下载](https://doc.scrapy.org/en/latest/topics/media-pipeline.html#topics-media-pipeline)与抓取的项目相关的[图像](https://doc.scrapy.org/en/latest/topics/media-pipeline.html#topics-media-pipeline)（或任何其他媒体）的介质管道，缓存DNS解析程序等等！

## 下一步是什么？

接下来的步骤是[安装Scrapy](https://doc.scrapy.org/en/latest/intro/install.html#intro-install)， [按照教程](https://doc.scrapy.org/en/latest/intro/tutorial.html#intro-tutorial)学习如何创建一个完整的Scrapy项目并[加入社区](https://scrapy.org/community/)。感谢您的关注！

# 安装指南

## 安装Scrapy

Scrapy在Python 2.7和Python 3.4以上运行，在CPython（默认Python实现）和PyPy（从PyPy 5.9开始）下运行。

如果您使用的是[Anaconda](https://docs.anaconda.com/anaconda/)或[Miniconda](https://conda.io/docs/user-guide/install/index.html)，则可以从[conda-](https://conda-forge.org/)[forge](https://conda.io/docs/user-guide/install/index.html)频道安装软件包，该频道具有适用于Linux，Windows和OS X的最新软件包。

要使用安装Scrapy conda，请运行：

conda install **-**c conda**-**forge scrapy

或者，如果您已经熟悉Python包的安装，则可以使用PyPI安装Scrapy及其依赖项：

pip install Scrapy

请注意，有时这可能需要根据您的操作系统解决某些Scrapy依赖项的编译问题，因此请务必查看 [平台特定的安装说明](https://doc.scrapy.org/en/latest/intro/install.html#intro-install-platform-notes)。

我们强烈建议您将Scrapy安装在[专用的virtualenv中](https://doc.scrapy.org/en/latest/intro/install.html#intro-using-virtualenv)，以避免与系统软件包发生冲突。

有关更详细的平台特定说明，请继续阅读。

### 事情很好知道

Scrapy是用纯Python编写的，并且依赖于几个关键的Python包（等等）：

* [lxml](http://lxml.de/)，一种高效的XML和HTML解析器
* [parsel](https://pypi.python.org/pypi/parsel)，一个写在lxml之上的HTML / XML数据提取库，
* [w3lib](https://pypi.python.org/pypi/w3lib)，一个用于处理URL和网页编码的多用途帮手
* [扭曲](https://twistedmatrix.com/)，一个异步网络框架
* [密码学](https://cryptography.io/)和[pyOpenSSL](https://pypi.python.org/pypi/pyOpenSSL)，来处理各种网络级别的安全需求

Scrapy测试的最低版本是：

* 扭曲14.0
* lxml 3.4
* pyOpenSSL 0.14

Scrapy可能会使用这些软件包的旧版本，但不能保证它会继续工作，因为它没有经过测试。

其中一些软件包本身依赖于非Python包，这可能需要额外的安装步骤，具体取决于您的平台。请[在下面](https://doc.scrapy.org/en/latest/intro/install.html#intro-install-platform-notes)查看[平台特定的指南](https://doc.scrapy.org/en/latest/intro/install.html#intro-install-platform-notes)。

如果与这些依赖关系有关的任何问题，请参阅其各自的安装说明：

* [lxml安装](http://lxml.de/installation.html)
* [密码学安装](https://cryptography.io/en/latest/installation/)

### 使用虚拟环境（推荐）

TL; DR：我们建议在所有平台的虚拟环境中安装Scrapy。

Python软件包既可以全局安装（又称系统范围），也可以安装在用户空间中。我们不建议安装scrapy系统。

相反，我们建议您在所谓的“虚拟环境”（[virtualenv](https://virtualenv.pypa.io/)）中安装scrapy 。Virtualenvs允许您不会与已经安装的Python系统软件包（这可能会破坏您的一些系统工具和脚本）发生冲突，并且仍然可以正常安装软件包pip（不sudo包括类似软件）。

要开始使用虚拟环境，请参阅[virtualenv安装说明](https://virtualenv.pypa.io/en/stable/installation/)。要全局安装它（全局安装在这里实际上有帮助），它应该是一个运行的问题：

$ [sudo] pip install virtualenv

查看本[用户指南](https://virtualenv.pypa.io/en/stable/userguide/)了解如何创建您的virtualenv。

**注意**

如果您使用Linux或OS X，[virtualenvwrapper](https://virtualenvwrapper.readthedocs.io/en/latest/install.html)是创建virtualenvs的方便工具。

一旦你创建了virtualenv，你就可以在其中安装scrapy pip，就像其他任何Python包一样。（有关您可能需要事先安装的非Python依赖关系，请参见下面的[平台特定指南](https://doc.scrapy.org/en/latest/intro/install.html#intro-install-platform-notes)）。

可以创建Python virtualenvs以默认使用Python 2，或默认使用Python 3。

* 如果你想用Python 3安装scrapy，请在Python 3 virtualenv中安装scrapy。
* 如果你想用Python 2安装scrapy，请在Python 2 virtualenv中安装scrapy。

## 平台特定的安装说明

### 视窗

尽管可以在Windows上使用pip安装Scrapy，但我们建议您安装[Anaconda](https://docs.anaconda.com/anaconda/)或[Miniconda，](https://conda.io/docs/user-guide/install/index.html)并使用[conda](https://conda.io/docs/user-guide/install/index.html)[-](https://conda-forge.org/)[forge](https://conda.io/docs/user-guide/install/index.html)频道中的软件包 ，这样可以避免大多数安装问题。

安装[Anaconda](https://docs.anaconda.com/anaconda/)或[Miniconda后](https://conda.io/docs/user-guide/install/index.html)，请使用以下[命令](https://conda.io/docs/user-guide/install/index.html)安装Scrapy：

conda install **-**c conda**-**forge scrapy

### Ubuntu 14.04或更高版本

目前Scrapy已经通过近期版本的lxml，twisted和pyOpenSSL测试，并且与最新的Ubuntu发行版兼容。但它也应该支持Ubuntu的早期版本，比如Ubuntu 14.04，尽管TLS连接可能存在问题。

**不要**使用python-scrapyUbuntu提供的软件包，它们通常太旧，速度太慢，无法赶上最新的Scrapy。

要在Ubuntu（或基于Ubuntu）的系统上安装scrapy，您需要安装这些依赖关系：

sudo apt **-** get install python **-** dev python **-** pip libxml2 **-** dev libxslt1 **-** dev zlib1g **-** dev libffi **-** dev libssl **–** dev

* python-dev，zlib1g-dev，libxml2-dev和libxslt1-dev 所需要的lxml
* libssl-dev并且libffi-dev是必需的cryptography

如果你想在Python 3上安装scrapy，你还需要Python 3开发头文件：

sudo apt **-** get install python3 python3 **-** dev

在[virtualenv中](https://doc.scrapy.org/en/latest/intro/install.html#intro-using-virtualenv)，你可以pip在那之后安装Scrapy ：

pip install scrapy

在Debian Jessie（8.0）及更高版本中，可以使用相同的非Python依赖项来安装Scrapy。

### Mac OS X

构建Scrapy的依赖需要存在C编译器和开发头文件。在OS X上，这通常由Apple的Xcode开发工具提供。要安装Xcode命令行工具，请打开一个终端窗口并运行：

xcode**-**select **–**install

有一个[已知问题](https://github.com/pypa/pip/issues/2468)阻止pip更新系统软件包。必须解决这个问题才能成功安装Scrapy及其依赖项。以下是一些建议的解决方案

* （推荐） **不要**使用系统python，安装一个不会与系统其余部分冲突的新的更新版本。以下是如何使用[自制](https://brew.sh/)软件包管理器执行此操作的方法：
  + 按照[https://brew.sh/中](https://brew.sh/)的说明安装[自制软件](https://brew.sh/)
  + 更新你的PATH变量来说明应该在系统包之前使用自制软件包（如果你使用[zsh](https://www.zsh.org/)作为默认shell，请更改.bashrc为[：）](https://www.zsh.org/).zshrc
* echo "export PATH=/usr/local/bin:/usr/local/sbin:$PATH" **>>** **~/.**bashrc

重新加载.bashrc以确保发生了变化： source **~/.**bashrc

Install python:

brew install python

Python的最新版本已pip与它们捆绑在一起，因此您不需要单独安装它。如果情况并非如此，请升级python：

brew update; brew upgrade python

* （可选）在独立的python环境中安装Scrapy。

此方法是解决上述OS X问题的一种解决方法，但它是管理依赖关系的一种总体良好实践，可以补充第一种方法。

[virtualenv](https://virtualenv.pypa.io/)是一个可以用来在python中创建虚拟环境的工具。我们建议您阅读[http://docs.python-guide.org/en/latest/dev/virtualenvs/等](http://docs.python-guide.org/en/latest/dev/virtualenvs/)教程 以开始使用。

在任何这些变通办法之后，您应该可以安装Scrapy：

pip install Scrapy

### PyPy

我们建议使用最新的PyPy版本。测试版本是5.9.0。对于PyPy3，仅测试了Linux安装。

大多数scrapy依赖现在都有用于CPython的二进制轮，但不适用于PyPy。这意味着这些依赖将在安装过程中建立。在OS X上，您可能会遇到构建密码依赖性的问题，[此处](https://github.com/pyca/cryptography/issues/2692#issuecomment-272773481)描述解决此问题的方法 ，即先导出此命令推荐的标志（仅在安装scrapy时需要）。除了安装构建依赖关系之外，在Linux上安装没有特殊问题。在Windows上使用PyPy安装scrapy未经测试。brew install openssl

您可以通过运行检查scrapy是否正确安装。如果这个命令给出了错误，例如 ，这意味着setuptools无法获取一个PyPy特定的依赖关系。要解决此问题，请运行。scrapy benchTypeError: ... got 2 unexpected keyword argumentspip install 'PyPyDispatcher>=2.1.0'

[下一个](https://doc.scrapy.org/en/latest/intro/tutorial.html)[以前](https://doc.scrapy.org/en/latest/intro/overview.html)

# Scrapy教程

在本教程中，我们假定Scrapy已经安装在您的系统上。如果不是这种情况，请参阅[安装指南](https://doc.scrapy.org/en/latest/intro/install.html#intro-install)。

我们将[去掉quotes.toscrape.com](http://quotes.toscrape.com/)，这是一个列出着名作家引用的网站。

本教程将引导您完成这些任务：

1. 创建一个新的Scrapy项目
2. 编写[蜘蛛](https://doc.scrapy.org/en/latest/topics/spiders.html#topics-spiders)抓取网站并提取数据
3. 使用命令行导出刮取的数据
4. 更改蜘蛛递归跟随链接
5. 使用蜘蛛参数

Scrapy是用[Python](https://www.python.org/)编写的。如果您对语言很陌生，您可能想先了解语言是什么样子，以充分利用Scrapy。

如果您已经熟悉其他语言，并希望快速学习Python，我们建议您阅读[Dive Into Python 3](http://www.diveintopython3.net/)。或者，您可以按照[Python教程](https://docs.python.org/3/tutorial)。

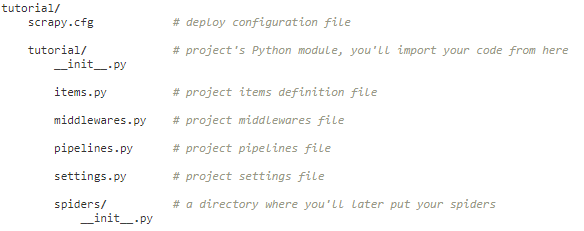
如果您是编程新手，想从Python开始，那么您可能会对联机丛书“ [Learn Python The Hard Way”感兴趣](https://learnpythonthehardway.org/book/)。您还可以查看[非程序员的Python资源列表](https://wiki.python.org/moin/BeginnersGuide/NonProgrammers)。

## 创建一个项目

在开始抓取之前，您将不得不建立一个新的Scrapy项目。输入您想要存储代码并运行的目录：

scrapy startproject tutorial

这将创建一个tutorial包含以下内容的目录：



## 我们的第一个蜘蛛

蜘蛛是您定义的类，并且Scrapy用于从网站（或一组网站）刮取信息。他们必须进行子类化**scrapy.Spider**和定义最初的请求，可选择如何关注页面中的链接，以及如何解析下载的页面内容以提取数据。

这是我们第一个蜘蛛的代码。将其保存在项目目录quotes\_spider.py下的一个文件 tutorial/spiders中：



正如您所看到的，我们的Spider子类[**scrapy.Spider**](https://doc.scrapy.org/en/latest/topics/spiders.html#scrapy.spiders.Spider) 定义了一些属性和方法：

* [**name**](https://doc.scrapy.org/en/latest/topics/spiders.html#scrapy.spiders.Spider.name)：标识蜘蛛。它在项目中必须是唯一的，也就是说，不能为不同的蜘蛛设置相同的名称。
* [**start\_requests()**](https://doc.scrapy.org/en/latest/topics/spiders.html#scrapy.spiders.Spider.start_requests)：必须返回Spider将开始抓取的请求的迭代（您可以返回请求列表或编写生成器函数）。随后的请求将从这些初始请求中连续生成。
* [**parse()**](https://doc.scrapy.org/en/latest/topics/spiders.html#scrapy.spiders.Spider.parse)：将被调用来处理为每个请求下载的响应的方法。响应参数是[**TextResponse**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.TextResponse)保存页面内容的一个实例，并有更多有用的方法来处理它。

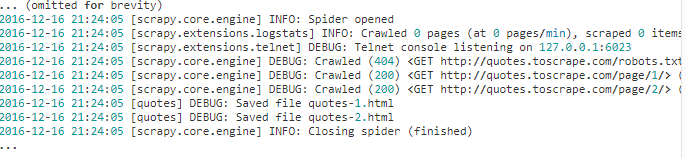
该[**parse()**](https://doc.scrapy.org/en/latest/topics/spiders.html#scrapy.spiders.Spider.parse)方法通常解析响应，将提取的数据提取为字符串，并查找新的URL并[**Request**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.Request)根据它们创建新的请求（）。

### 如何运行我们的蜘蛛

为了让我们的蜘蛛工作，请转到项目的顶层目录并运行：

scrapy crawl quotes

这个命令以quotes我们刚刚添加的名称运行蜘蛛，它将发送一些quotes.toscrape.com域的请求。你会得到类似于这样的输出：



现在，检查当前目录中的文件。按照我们的方法指示，您应该注意到已经创建了两个新文件：quotes-1.html和quotes-2.html，以及各个URL的内容parse。

**注意**

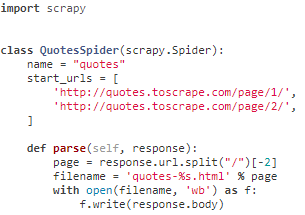
如果您想知道为什么我们还没有解析HTML，请坚持下去，我们会很快回覆。

#### 在引擎盖下发生了什么？

Scrapy安排[**scrapy.Request**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.Request)由start\_requestsSpider 的方法返回的对象。在收到每个响应后，它会实例化[**Response**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.Response)对象并调用与请求相关的回调方法（在本例中为 parse方法），将响应作为参数传递。

### start\_requests方法的快捷方式

不用实现一个从URL [**start\_requests()**](https://doc.scrapy.org/en/latest/topics/spiders.html#scrapy.spiders.Spider.start_requests)生成[**scrapy.Request**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.Request)对象的方法，你可以[**start\_urls**](https://doc.scrapy.org/en/latest/topics/spiders.html#scrapy.spiders.Spider.start_urls)用一系列URL 来定义一个类属性。这个列表将被默认实现[**start\_requests()**](https://doc.scrapy.org/en/latest/topics/spiders.html#scrapy.spiders.Spider.start_requests)用于为您的蜘蛛创建初始请求：



该[**parse()**](https://doc.scrapy.org/en/latest/topics/spiders.html#scrapy.spiders.Spider.parse)方法将被调用来处理这些URL的每个请求，即使我们没有明确告诉Scrapy这样做。发生这种情况是因为[**parse()**](https://doc.scrapy.org/en/latest/topics/spiders.html#scrapy.spiders.Spider.parse)Scrapy的默认回调方法，该方法在没有显式分配回调的情况下被调用。

### 提取数据

学习如何使用Scrapy提取数据的最佳方法是尝试使用Shell [Scrapy外壳的](https://doc.scrapy.org/en/latest/topics/shell.html#topics-shell)选择器。Run：

scrapy shell 'http://quotes.toscrape.com/page/1/'

**注意**

请记住，从命令行运行Scrapy shell时，请始终将引号括起来，否则包含参数（例如。&字符）的url 将不起作用。

在Windows上，请使用双引号：

scrapy shell <http://quotes.toscrape.com/page/1/>



使用shell，你可以尝试用响应对象使用[CSS](https://www.w3.org/TR/selectors)选择元素：

>>> response**.**css('title')

[<Selector xpath='descendant-or-self::title' data='<title>Quotes to Scrape</title>'>]

运行的结果response.css('title')是一个名为的类似列表的对象 [**SelectorList**](https://doc.scrapy.org/en/latest/topics/selectors.html#scrapy.selector.SelectorList)，它表示一系列[**Selector**](https://doc.scrapy.org/en/latest/topics/selectors.html#scrapy.selector.Selector)围绕XML / HTML元素的对象列表， 并允许您运行更多查询来细化选择或提取数据。

要从上述标题中提取文本，您可以执行以下操作：

>>> response**.**css('title::text')**.**extract()

['Quotes to Scrape']

这里需要注意两点：其一是我们已经添加::text到CSS查询中，意思是我们只想选择元素内部的文本元素 <title>。如果我们没有指定::text，我们会得到完整的标题元素，包括它的标签：

>>> response**.**css('title')**.**extract()

['<title>Quotes to Scrape</title>']

另一件事是调用的结果.extract()是一个列表，因为我们正在处理一个实例[**SelectorList**](https://doc.scrapy.org/en/latest/topics/selectors.html#scrapy.selector.SelectorList)。当你知道你只是想要第一个结果，就像在这种情况下，你可以这样做：

>>> response**.**css('title::text')**.**extract\_first()

'Quotes to Scrape'

或者，你可以写下：

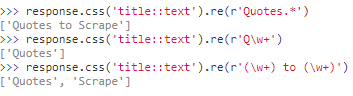
>>> response**.**css('title::text')[0]**.**extract()

'Quotes to Scrape'

但是，如果找不到与选择相匹配的元素，则使用.extract\_first()避免IndexError并返回 None。

这里有一个教训：对于大多数刮擦代码，您希望它能够灵活地处理由于在页面上未找到的东西而导致的错误，因此即使某些部分无法被刮取，您至少也可以获取**一些**数据。

除了[**extract()**](https://doc.scrapy.org/en/latest/topics/selectors.html#scrapy.selector.Selector.extract)和 **extract\_first()**方法之外，还可以使用该[**re()**](https://doc.scrapy.org/en/latest/topics/selectors.html#scrapy.selector.Selector.re)方法使用*正则表达式*进行提取：

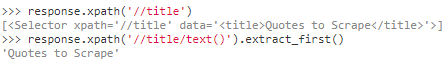


为了找到合适的CSS选择器来使用，你可能会发现在你的web浏览器中使用shell打开响应页面很有用view(response)。您可以使用浏览器开发工具或Firebug等扩展（请参阅关于[使用Firebug进行抓取](https://doc.scrapy.org/en/latest/topics/firebug.html#topics-firebug)和[使用Firefox进行抓取的部分](https://doc.scrapy.org/en/latest/topics/firefox.html#topics-firefox)）。

[Selector Gadget](http://selectorgadget.com/)也是一个很好的工具，可以快速找到可供选择的元素的CSS选择器，这可以在许多浏览器中使用。

#### XPath：一个简短的介绍

除[CSS外](https://www.w3.org/TR/selectors)，Scrapy选择器还支持使用[XPath](https://www.w3.org/TR/xpath)表达式：



XPath表达式非常强大，是Scrapy选择器的基础。实际上，CSS选择器在引擎盖下转换为XPath。您可以看到，如果仔细阅读shell中选择器对象的文本表示形式。

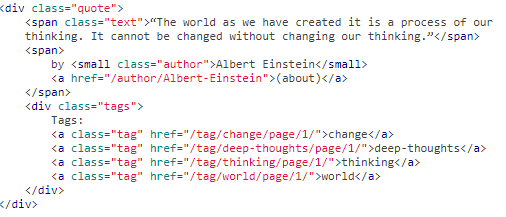
虽然可能不如CSS选择器那么受欢迎，但XPath表达式提供了更多的功能，因为除了浏览结构之外，它还可以查看内容。使用XPath，您可以选择如下内容：选择包含文本“下一页”的链接。这使得XPath非常适合抓取任务，并且即使您已经知道如何构建CSS选择器，我们也鼓励您学习XPath，这会使抓取更容易。

这里我们不会涉及到很多XPath，但您可以在这里阅读更多有关[使用Scrapy选择器的XPath的信息](https://doc.scrapy.org/en/latest/topics/selectors.html#topics-selectors)。要了解关于XPath的更多信息，我们推荐[本教程通过示例学习XPath](http://zvon.org/comp/r/tut-XPath_1.html)，[本教程将学习“如何在XPath中思考”](http://plasmasturm.org/log/xpath101/)。

#### 提取引号和作者

现在您已经了解了一些关于选择和提取的内容，让我们通过编写代码来从网页中提取引号来完成我们的蜘蛛。

[http://quotes.toscrape.com中的](http://quotes.toscrape.com/)每个引用都由HTML元素表示，如下所示：



让我们打开scrapy shell并玩一下，找出如何提取我们想要的数据：

$ scrapy shell'http://quotes.toscrape.com'

我们通过以下方式获得引用HTML元素的选择器列表：

>>> response**.**css("div.quote")

上述查询返回的每个选择器都允许我们对其子元素运行更多查询。让我们将第一个选择器分配给一个变量，以便我们可以直接在特定的引号上运行我们的CSS选择器：

>>> quote **=** response**.**css("div.quote")[0]

现在，让我们来提取title，author而tags从报价使用quote我们刚刚创建的对象：

>>> title **=** quote**.**css("span.text::text")**.**extract\_first()

>>> title

'“The world as we have created it is a process of our thinking. It cannot be changed without changing our thinking.”'

>>> author **=** quote**.**css("small.author::text")**.**extract\_first()

>>> author

'Albert Einstein'

鉴于标签是一个字符串列表，我们可以使用该.extract()方法来获取所有的字符串：

>>> tags **=**  quote **。**CSS （“div.tags a.tag :: text” ）**。**extract （）

>>> tags

['change'，'deep-thoughts'，'thinking'，'world']

在弄清楚了如何提取每一位数据之后，我们现在可以遍历所有引号元素，并将它们放在一起形成一个Python字典：

>>> **for** quote **in** response**.**css("div.quote"):

... text **=** quote**.**css("span.text::text")**.**extract\_first()

... author **=** quote**.**css("small.author::text")**.**extract\_first()

... tags **=** quote**.**css("div.tags a.tag::text")**.**extract()

... print(dict(text**=**text, author**=**author, tags**=**tags))

{'tags': ['change', 'deep-thoughts', 'thinking', 'world'], 'author': 'Albert Einstein', 'text': '“The world as we have created it is a process of our thinking. It cannot be changed without changing our thinking.”'}

{'tags': ['abilities', 'choices'], 'author': 'J.K. Rowling', 'text': '“It is our choices, Harry, that show what we truly are, far more than our abilities.”'}

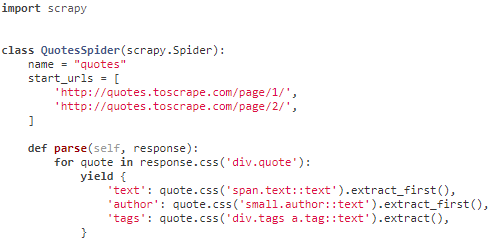
... a few more of these, omitted for brevity

>>>

### 在我们的蜘蛛中提取数据

让我们回到我们的蜘蛛。直到现在，它并没有特别提取任何数据，只是将整个HTML页面保存到本地文件中。让我们将上面的提取逻辑集成到我们的蜘蛛中。

Scrapy蜘蛛通常会生成许多包含从页面提取的数据的字典。为此，我们yield在回调中使用Python关键字，如下所示：



如果你运行这个蜘蛛，它会输出提取的数据和日志：

2016**-**09**-**19 18:57:19 [scrapy**.**core**.**scraper] DEBUG: Scraped **from** **<**200 http:**//**quotes**.**toscrape**.**com**/**page**/**1**/>**

{'tags': ['life', 'love'], 'author': 'André Gide', 'text': '“It is better to be hated for what you are than to be loved for what you are not.”'}

2016**-**09**-**19 18:57:19 [scrapy**.**core**.**scraper] DEBUG: Scraped **from** **<**200 http:**//**quotes**.**toscrape**.**com**/**page**/**1**/>**

{'tags': ['edison', 'failure', 'inspirational', 'paraphrased'], 'author': 'Thomas A. Edison', 'text': "“I have not failed. I've just found 10,000 ways that won't work.”"}

## 存储刮取的数据

存储刮取数据最简单的方法是使用[Feed输出](https://doc.scrapy.org/en/latest/topics/feed-exports.html#topics-feed-exports)，使用以下命令：

scrapy crawl quotes **-**o quotes**.**json

这将生成一个quotes.json包含所有抓取的项目的文件，并以[JSON](https://en.wikipedia.org/wiki/JSON)序列化。

由于历史原因，Scrapy附加到给定文件而不是覆盖其内容。如果您在第二次执行该命令两次而未删除该文件的情况下运行该命令，则最终会生成一个损坏的JSON文件。

您也可以使用其他格式，例如[JSON Lines](http://jsonlines.org/)：

scrapy crawl quotes **-**o quotes**.**jl

该[JSON行](http://jsonlines.org/)格式是有用的，因为它的流状，你可以很容易地新记录追加到它。当您运行两次时，它没有相同的JSON问题。另外，由于每条记录都是一条独立的行，因此您可以处理大文件而不必将所有内容都放在内存中，而像[JQ](https://stedolan.github.io/jq)这样的工具可以帮助在命令行执行该操作。

在小型项目中（如本教程中的），应该足够了。但是，如果您想使用刮取的物品执行更复杂的事情，您可以编写[物料管道](https://doc.scrapy.org/en/latest/topics/item-pipeline.html#topics-item-pipeline)。项目管道的占位符文件已经在创建项目时为您设置tutorial/pipelines.py。尽管如果你只是想存储刮取的物品，你不需要实现任何物品管线。

## 以下链接

比方说，我们不是从[http://quotes.toscrape.com](http://quotes.toscrape.com/)的前两页中提取内容，而是希望从网站中的所有页面引用报价。

现在您已经知道如何从页面提取数据，我们来看看如何跟踪它们的链接。

首先要提取我们想要关注的页面的链接。检查我们的页面，我们可以看到有一个到下一页的链接，并带有以下标记：

<ul class**=**"pager">

<li class**=**"next">

<a href**=**"/page/2/">Next <span aria-hidden**=**"true">&rarr ;</span></a>

</li>

</ul>

We can try extracting it in the shell:

>>> response**.**css('li.next a')**.**extract\_first()

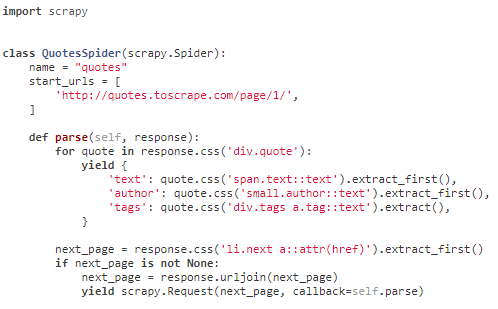
'<a href="/page/2/">Next <span aria-hidden="true">→</span></a>'

这会获取锚点元素，但我们需要该属性href。为此，Scrapy支持一个CSS扩展，让您选择属性内容，如下所示：

>>> response**.**css('li.next a::attr(href)')**.**extract\_first()

'/page/2/'

现在让我们看看我们的蜘蛛修改为递归地跟随链接到下一页，从中提取数据：



现在，在提取数据后，该parse()方法查找到下一页的链接，使用该[**urljoin()**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.Response.urljoin)方法构建完整的绝对URL （因为链接可以是相对的），并产生一个到下一页的新请求，将自己注册为回调来处理提取下一页的数据并保持所有页面的爬行。

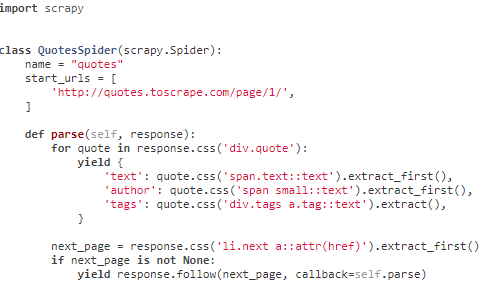
您在这里看到的是Scrapy的以下链接机制：当您在回调方法中产生请求时，Scrapy会安排发送请求并注册一个回调方法，以便在请求结束时执行。

使用这种方法，您可以根据您定义的规则构建复杂的抓取工具，并根据所访问的页面提取不同类型的数据。

在我们的例子中，它创建了一个循环，跟随到下一页的所有链接，直到它找不到用于抓取博客，论坛和其他分页的站点。

### 创建请求的快捷方式

作为创建Request对象的快捷方式，您可以使用 [**response.follow**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.TextResponse.follow)：



与scrapy.Request不同，它response.follow直接支持相对URL - 无需调用urljoin。请注意，response.follow只是返回一个Request实例; 你仍然必须产生这个请求。

你也可以传递一个选择器来response.follow代替一个字符串; 该选择器应该提取必要的属性：

**for** href **in** response**.**css('li.next a::attr(href)'):

**yield** response**.**follow(href, callback**=**self**.**parse)

对于<a>元素有一个快捷方式：response.follow自动使用他们的href属性。所以代码可以进一步缩短：

**for** a **in** response**.**css('li.next a'):

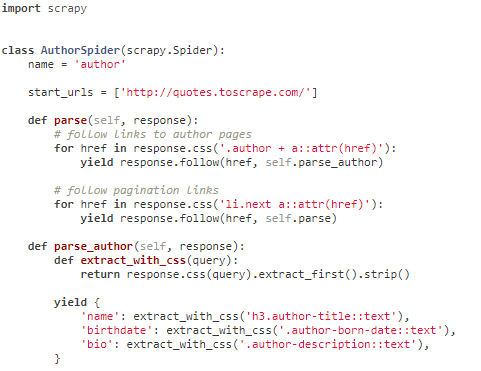
**yield** response**.**follow(a, callback**=**self**.**parse)

**注意**

response.follow(response.css('li.next a'))是无效的，因为它 response.css返回一个带有所有结果选择器的类列表对象，而不是单个选择器。甲for象在上面的例子中循环，或 是好的。response.follow(response.css('li.next a')[0])

### 更多的例子和模式

这是另一个蜘蛛，它演示了回调和跟随链接，这次是为了刮取作者信息：



这个蜘蛛将从主页面开始，它将跟随作者页面的所有链接，调用parse\_author每个页面的回调，并且还会跟parse我们之前看到的回调分页链接。

在这里，我们将回调传递给response.follow位置参数以缩短代码长度; 它也适用于scrapy.Request。

该parse\_author回调函数定义了一个辅助函数，用于从CSS查询中提取和清除数据，并生成带有作者数据的Python字典。

这个蜘蛛演示的另一个有趣的事情是，即使有来自同一作者的许多引用，我们也不必担心多次访问相同的作者页面。默认情况下，Scrapy会将重复的请求过滤到已经访问过的URL，避免因编程错误而导致服务器过多的问题。这可以通过设置进行配置 [**DUPEFILTER\_CLASS**](https://doc.scrapy.org/en/latest/topics/settings.html#std:setting-DUPEFILTER_CLASS)。

希望现在您已经对如何使用Scrapy的下列链接和回调机制有了很好的了解。

作为利用以下链接机制的又一个示例蜘蛛，请查看[**CrawlSpider**](https://doc.scrapy.org/en/latest/topics/spiders.html#scrapy.spiders.CrawlSpider)该类的通用蜘蛛，该蜘蛛实现了一个小规则引擎，您可以使用它来在其上编写爬网程序。

此外，一种常见模式是使用多个页面的数据构建项目，并使用[技巧将其他数据传递给回调](https://doc.scrapy.org/en/latest/topics/request-response.html#topics-request-response-ref-request-callback-arguments)。

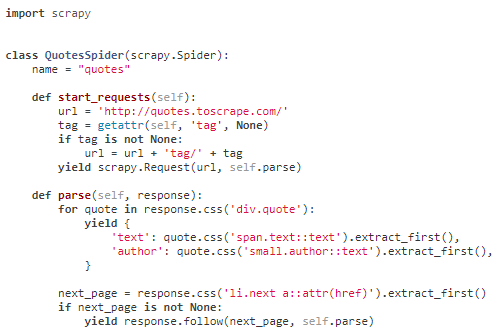
## 使用蜘蛛参数

-a 运行时，可以使用该选项向蜘蛛提供命令行参数：

scrapy crawl quotes **-**o quotes**-**humor**.**json **-**a tag**=**humor

这些参数被传递给蜘蛛的\_\_init\_\_方法，并默认成为蜘蛛属性。

在这个例子中，为参数提供的值tag将可以通过self.tag。你可以使用它来让你的蜘蛛只用特定的标签获取引号，根据参数构建URL：



如果你将tag=humor参数传递给这个蜘蛛，你会注意到它只会访问humor标签中的URL ，例如http://quotes.toscrape.com/tag/humor。

你可以[在这里了解更多关于处理蜘蛛参数的信息](https://doc.scrapy.org/en/latest/topics/spiders.html#spiderargs)。

## 下一步

本教程仅介绍了Scrapy的基础知识，但还有很多其他功能未在此处提及。检查[还有什么？](https://doc.scrapy.org/en/latest/intro/overview.html#topics-whatelse)在 [Scrapy中](https://doc.scrapy.org/en/latest/intro/overview.html#intro-overview)简要介绍最重要的章节。

您可以从“ [基本概念](https://doc.scrapy.org/en/latest/index.html#section-basics) ”部分继续了解更多关于命令行工具，蜘蛛，选择器和本教程未涉及的其他内容，如对抓取的数据进行建模。如果您喜欢玩示例项目，请查看[示例](https://doc.scrapy.org/en/latest/intro/examples.html#intro-examples)部分。

# 命令行工具

0.10版本中的新功能。

Scrapy通过scrapy命令行工具进行控制，在这里被称为“Scrapy工具”，以区别于我们称之为“命令”或“Scrapy命令”的子命令。

Scrapy工具提供了多种命令，用于多种目的，并且每个命令都接受一组不同的参数和选项。

（这个命令已经在1.0版本中被移除了，请参阅[部署您的项目](https://scrapyd.readthedocs.io/en/latest/deploy.html)。）scrapy deployscrapyd-deploy

## 配置设置

Scrapy将scrapy.cfg在标准位置的ini样式文件中查找配置参数：

1. /etc/scrapy.cfg或c:\scrapy\scrapy.cfg（系统范围），
2. ~/.config/scrapy.cfg（$XDG\_CONFIG\_HOME）和~/.scrapy.cfg（$HOME）用于全局（用户范围）设置，以及
3. scrapy.cfg 在scrapy项目的根目录中（见下一节）。

这些文件中的设置按所列出的优先顺序进行合并：用户定义的值具有比系统范围内的默认值更高的优先级，并且在定义时，项目范围的设置将覆盖所有其他文件。

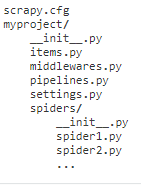
Scrapy也理解并可以通过一些环境变量进行配置。目前这些是：

* SCRAPY\_SETTINGS\_MODULE（请参阅[指定设置](https://doc.scrapy.org/en/latest/topics/settings.html#topics-settings-module-envvar)）
* SCRAPY\_PROJECT
* SCRAPY\_PYTHON\_SHELL（请参阅[Scrapy外壳](https://doc.scrapy.org/en/latest/topics/shell.html#topics-shell)）

## Scrapy项目的默认结构

在深入研究命令行工具及其子命令之前，我们先来了解Scrapy项目的目录结构。

虽然可以修改，但所有Scrapy项目默认具有相同的文件结构，与此类似：



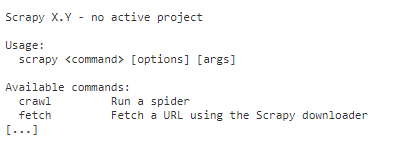
scrapy.cfg文件所在的目录称为项目根目录。该文件包含定义项目设置的python模块的名称。这里是一个例子：

[settings]

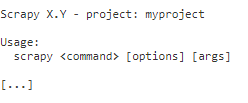
default **=** myproject**.**settings

## 使用该scrapy工具

您可以通过运行不带参数的Scrapy工具开始，它将打印一些使用帮助和可用的命令：



如果您在Scrapy项目中，则第一行将打印当前活动的项目。在这个例子中，它是从一个项目之外运行的。如果从一个项目中运行，它会打印出如下所示的内容：



### 创建项目

您通常使用该scrapy工具做的第一件事是创建您的Scrapy项目：

scrapy startproject myproject [ project\_dir ]

这将在project\_dir目录下创建一个Scrapy项目。如果project\_dir没有指定，project\_dir将会与之相同myproject。

接下来，你进入新的项目目录：

cd project\_dir

而且您已准备好使用该scrapy命令来管理和控制您的项目。

### 控制项目

您可以使用scrapy项目内部的工具来控制和管理它们。

例如，要创建一个新的蜘蛛：

scrapy genspider mydomain mydomain**.**com

一些Scrapy命令（如[**crawl**](https://doc.scrapy.org/en/latest/topics/commands.html#std:command-crawl)）必须从Scrapy项目中运行。有关哪些命令必须从项目内部运行的更多信息，请参阅下面的[命令参考](https://doc.scrapy.org/en/latest/topics/commands.html#topics-commands-ref)。

另外请记住，某些命令在从项目内部运行时可能会有稍微不同的行为。例如，如果抓取user\_agent的URL与某个特定的蜘蛛相关联，则抓取命令将使用蜘蛛覆盖的行为（例如该属性覆盖用户代理）。这是故意的，因为该fetch命令是用来检查蜘蛛是如何下载页面的。

## 可用的工具命令

本节包含可用内置命令的列表以及说明和一些使用示例。请记住，您可以通过运行始终获取有关每个命令的更多信息：

scrapy **<** command **>**  **-** h

你可以看到所有可用的命令：

scrapy **-** h

有两种类型的命令，那些只能在Scrapy项目中使用的命令（特定于项目的命令）以及没有活动的Scrapy项目（全局命令）的命令，尽管它们在项目中运行时可能略有不同因为他们会使用项目重写设置）。

全局命令：

* [**startproject**](https://doc.scrapy.org/en/latest/topics/commands.html#std:command-startproject)
* [**genspider**](https://doc.scrapy.org/en/latest/topics/commands.html#std:command-genspider)
* [**settings**](https://doc.scrapy.org/en/latest/topics/commands.html#std:command-settings)
* [**runspider**](https://doc.scrapy.org/en/latest/topics/commands.html#std:command-runspider)
* [**shell**](https://doc.scrapy.org/en/latest/topics/commands.html#std:command-shell)
* [**fetch**](https://doc.scrapy.org/en/latest/topics/commands.html#std:command-fetch)
* [**view**](https://doc.scrapy.org/en/latest/topics/commands.html#std:command-view)
* [**version**](https://doc.scrapy.org/en/latest/topics/commands.html#std:command-version)

仅限项目的命令：

* [**crawl**](https://doc.scrapy.org/en/latest/topics/commands.html#std:command-crawl)
* [**check**](https://doc.scrapy.org/en/latest/topics/commands.html#std:command-check)
* [**list**](https://doc.scrapy.org/en/latest/topics/commands.html#std:command-list)
* [**edit**](https://doc.scrapy.org/en/latest/topics/commands.html#std:command-edit)
* [**parse**](https://doc.scrapy.org/en/latest/topics/commands.html#std:command-parse)
* [**bench**](https://doc.scrapy.org/en/latest/topics/commands.html#std:command-bench)

### startproject命令

* 句法： scrapy startproject <project\_name> [project\_dir]
* 需要项目：没有

project\_name在该project\_dir 目录下创建一个名为Scrapy的新项目。如果project\_dir没有指定，project\_dir将会与之相同project\_name。

用法示例：

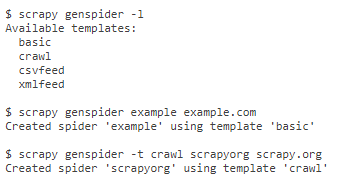
$ scrapy startproject myproject

### genspider

* 句法： scrapy genspider [-t template] <name> <domain>
* 需要项目：没有

spiders如果在项目中调用，则在当前文件夹或当前项目的文件夹中创建一个新的蜘蛛。该<name>参数设置为蜘蛛的name，而<domain>用于生成allowed\_domains和start\_urls蜘蛛的属性。

用法示例：



这只是一个便捷的快捷命令，用于根据预定义的模板创建蜘蛛，但当然不是创建蜘蛛的唯一方法。您可以自己创建蜘蛛源代码文件，而不是使用此命令。

### crawl

* 句法： scrapy crawl <spider>
* 需要项目：是的

开始使用蜘蛛爬行。

用法示例：

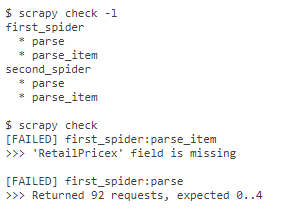


### check

* 句法： scrapy check [-l] <spider>
* 需要项目：*是的*

运行合同检查。

用法示例：



### list

* 句法： scrapy list
* 需要项目：*是的*

列出当前项目中所有可用的蜘蛛。输出是每行一个蜘蛛。

用法示例：

$ scrapy list

spider1

spider2

### edit

* 句法： scrapy edit <spider>
* 需要项目：*是的*

使用EDITOR环境变量中定义的编辑器编辑给定的蜘蛛，或者（如果未设置）编辑该[**EDITOR**](https://doc.scrapy.org/en/latest/topics/settings.html#std:setting-EDITOR)设置。

此命令仅作为最常见情况的便捷快捷方式提供，开发人员当然可以自由选择任何工具或IDE来编写和调试蜘蛛。

用法示例：



### fetch

* 句法： scrapy fetch <url>
* 需要项目：*没有*

使用Scrapy下载器下载给定的URL并将内容写入标准输出。

关于这个命令的有趣之处在于它抓取页面蜘蛛如何下载它。例如，如果蜘蛛有一个USER\_AGENT 覆盖用户代理的属性，它将使用该属性。

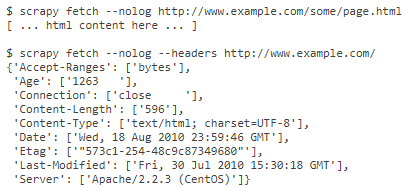
所以这个命令可以用来“看”你的蜘蛛如何获取某个页面。

如果在项目之外使用，则不会应用特定的每蜘蛛行为，它将仅使用默认的Scrapy下载器设置。

支持的选项：

* --spider=SPIDER：绕过蜘蛛自动检测并强制使用特定的蜘蛛
* --headers：打印响应的HTTP头，而不是响应的正文
* --no-redirect：不遵循HTTP 3xx重定向（默认是关注它们）

用法示例：



### view

* 句法： scrapy view <url>
* 需要项目：*没有*

在浏览器中打开给定的URL，因为Scrapy蜘蛛会“看到”它。有时候，蜘蛛看到的网页与普通用户不同，所以这可以用来检查蜘蛛“看到”什么，并确认它是你期望的。

支持的选项：

* --spider=SPIDER：绕过蜘蛛自动检测并强制使用特定的蜘蛛
* --no-redirect：不遵循HTTP 3xx重定向（默认是关注它们）

用法示例：



### shell

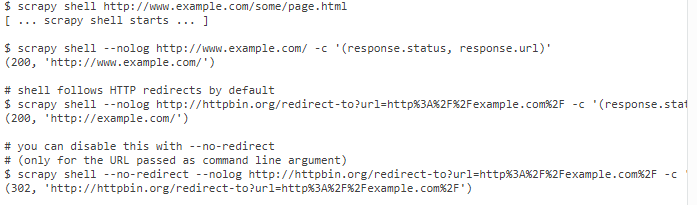
* 句法： scrapy shell [url]
* 需要项目：*没有*

为指定的URL（如果给定）启动Scrapy shell，如果没有给出URL，则为空。同时支持UNIX风格的本地文件路径，无论是相对 ./或../前缀或绝对文件路径。有关更多信息，请参阅[Scrapy外壳](https://doc.scrapy.org/en/latest/topics/shell.html#topics-shell)。

支持的选项：

* --spider=SPIDER：绕过蜘蛛自动检测并强制使用特定的蜘蛛
* -c code：评估shell中的代码，打印结果并退出
* --no-redirect：不遵循HTTP 3xx重定向（默认是关注它们）; 这只会影响您可能在命令行上作为参数传递的URL; 一旦你在shell中，fetch(url)默认情况下仍然会遵循HTTP重定向。

用法示例：



### parse

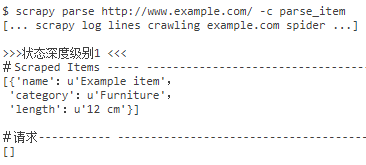
* 句法： scrapy parse <url> [options]
* 需要项目：*是的*

获取给定的URL并用处理它的蜘蛛解析它，使用随--callback选项传递的方法，或者parse如果没有给出。

支持的选项：

* --spider=SPIDER：绕过蜘蛛自动检测并强制使用特定的蜘蛛
* --a NAME=VALUE：设置蜘蛛参数（可能会重复）
* --callback或者-c：使用spider方法作为解析响应的回调
* --meta或者-m：将传递给回调请求的附加请求元。这必须是有效的json字符串。例如：-meta ='{“foo”：“bar”}'
* --pipelines：通过管道处理项目
* --rules或者-r：使用[**CrawlSpider**](https://doc.scrapy.org/en/latest/topics/spiders.html#scrapy.spiders.CrawlSpider) 规则来发现用于解析响应的回调（即蜘蛛方法）
* --noitems：不要显示被刮掉的物品
* --nolinks：不显示提取的链接
* --nocolour：避免使用pygments来着色输出
* --depth或-d：递归地跟踪请求的深度级别（默认值：1）
* --verbose或者-v：显示每个深度级别的信息

用法示例：



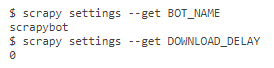
### settings

* 句法： scrapy settings [options]
* 需要项目：*没有*

获取Scrapy设置的价值。

如果在项目中使用它将显示项目设置值，否则它将显示该设置的默认Scrapy值。

用法示例：



### runspider

* 句法： scrapy runspider <spider\_file.py>
* 需要项目：*没有*

运行一个自包含在Python文件中的蜘蛛，而不必创建一个项目。

用法示例：



### Version

* 句法： scrapy version [-v]
* 需要项目：*没有*

打印Scrapy版本。如果与-v它一起使用，还会打印Python，Twisted和Platform信息，这对于错误报告很有用。

### bench

0.17版本中的新功能。

* 句法： scrapy bench
* 需要项目：*没有*

运行一个快速基准测试。[标杆管理](https://doc.scrapy.org/en/latest/topics/benchmarking.html#benchmarking)。

## 自定义项目命令

您也可以使用该[**COMMANDS\_MODULE**](https://doc.scrapy.org/en/latest/topics/commands.html#std:setting-COMMANDS_MODULE)设置添加自定义项目命令 。有关如何实现[命令](https://github.com/scrapy/scrapy/tree/master/scrapy/commands)的示例，请参阅[scrapy /](https://github.com/scrapy/scrapy/tree/master/scrapy/commands)命令中的 [Scrapy](https://github.com/scrapy/scrapy/tree/master/scrapy/commands)命令。

### COMMANDS\_MODULE

默认:( ''空字符串）

用于查找自定义Scrapy命令的模块。这用于为您的Scrapy项目添加自定义命令。

例：

COMMANDS\_MODULE **=**  'mybot.commands'

### 通过setup.py入口点注册命令

**注意**

这是一个实验性功能，请谨慎使用。

您还可以通过scrapy.commands在库setup.py 文件的入口点添加一个节来从外部库添加Scrapy命令 。

以下示例添加my\_command命令：

**from**  setuptools **import**  setup ， find\_packages

setup （name **=** 'scrapy-mymodule' ，

entry\_points **=** {

'scrapy.commands' ： [

'my\_command = my\_scrapy\_module.commands：MyCommand' ，

]，

}，

）

# 蜘蛛

蜘蛛是定义某个站点（或一组站点）如何被抓取的类，包括如何执行爬网（即跟踪链接）以及如何从其页面中提取结构化数据（即抓取项目）。换句话说，蜘蛛是您定义用于为特定网站（或者在某些情况下是一组网站）抓取和解析页面的自定义行为的地方。

对于蜘蛛来说，刮cycle周期会经历这样的事情：

1. 首先生成抓取第一个URL的初始请求，然后指定一个回调函数，使用从这些请求下载的响应来调用回调函数。

第一个执行请求是通过调用 [**start\_requests()**](https://doc.scrapy.org/en/latest/topics/spiders.html#scrapy.spiders.Spider.start_requests)方法（默认情况下）[**Request**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.Request)为方法中指定的URL[**start\_urls**](https://doc.scrapy.org/en/latest/topics/spiders.html#scrapy.spiders.Spider.start_urls)和 [**parse**](https://doc.scrapy.org/en/latest/topics/spiders.html#scrapy.spiders.Spider.parse)方法为请求的回调函数生成的。

1. 在回调函数中，您解析响应（网页），并返回提取的数据，[**Item**](https://doc.scrapy.org/en/latest/topics/items.html#scrapy.item.Item)对象， [**Request**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.Request)对象或这些对象的迭代中的字符串。这些请求也将包含一个回调（可能是相同的），然后由Scrapy下载，然后通过指定的回调处理它们的响应。
2. 在回调函数中，通常使用[选择器](https://doc.scrapy.org/en/latest/topics/selectors.html#topics-selectors)（但您也可以使用BeautifulSoup，lxml或您喜欢的任何机制）解析页面内容， 并使用解析的数据生成项目。
3. 最后，从蜘蛛返回的项目通常会持久化到数据库（在某些[项目管道中](https://doc.scrapy.org/en/latest/topics/item-pipeline.html#topics-item-pipeline)）或使用[Feed输出](https://doc.scrapy.org/en/latest/topics/feed-exports.html#topics-feed-exports)写入文件。

尽管这个周期适用于（或多或少）任何类型的蜘蛛，但为了不同的目的，有不同类型的默认蜘蛛捆绑到Scrapy中。我们将在这里讨论这些类型。

## scrapy.Spider

**class scrapy.spiders.Spider**

这是最简单的蜘蛛，也是其他蜘蛛必须继承的蜘蛛（包括与Scrapy捆绑在一起的蜘蛛，以及自己写的蜘蛛）。它不提供任何特殊功能。它只是提供了一个默认[**start\_requests()**](https://doc.scrapy.org/en/latest/topics/spiders.html#scrapy.spiders.Spider.start_requests)实现，它从[**start\_urls**](https://doc.scrapy.org/en/latest/topics/spiders.html#scrapy.spiders.Spider.start_urls)spider属性发送请求，并**parse** 为每个结果响应调用蜘蛛的方法。

**name**

定义此蜘蛛名称的字符串。蜘蛛名称是Scrapy如何定位（并实例化）蜘蛛，因此它必须是唯一的。然而，没有什么能够阻止你实例化同一个蜘蛛的多个实例。这是最重要的蜘蛛属性，它是必需的。

如果蜘蛛刮去单个域名，通常的做法是在域名之后命名蜘蛛，有或没有[TLD](https://en.wikipedia.org/wiki/Top-level_domain)。因此，例如，爬行的蜘蛛**mywebsite.com**通常会被调用 **mywebsite**。

**注意**

在Python 2中，这只能是ASCII。

**allowed\_domains**

包含允许此蜘蛛抓取的域的可选字符串列表。如果[**OffsiteMiddleware**](https://doc.scrapy.org/en/latest/topics/spider-middleware.html#scrapy.spidermiddlewares.offsite.OffsiteMiddleware)启用，则不会遵循不属于此列表中指定的域名（或其子域）的URL的请求 。

假设你的目标网址是**https://www.example.com/1.html**，然后添加**'example.com'**到列表中。

**start\_urls**

当没有指定特定网址时，蜘蛛将从哪个网址开始抓取的网址列表。所以，下载的第一个页面将在这里列出。随后的URL将从包含在起始URL中的数据中连续生成。

**custom\_settings**

运行此蜘蛛时将从项目范围配置覆盖的设置字典。它必须被定义为类属性，因为设置在实例化之前被更新。

有关可用内置设置的列表，请参阅： [内置设置参考](https://doc.scrapy.org/en/latest/topics/settings.html#topics-settings-ref)。

**crawler**

该属性[**from\_crawler()**](https://doc.scrapy.org/en/latest/topics/item-pipeline.html#from_crawler)在初始化该类后由class方法设置，并链接[**Crawler**](https://doc.scrapy.org/en/latest/topics/api.html#scrapy.crawler.Crawler)到此spider实例绑定的 对象。

爬虫在项目中封装了大量组件，以便进行单一入口访问（例如扩展，中间件，信号管理器等）。请参阅[Crawler API](https://doc.scrapy.org/en/latest/topics/api.html#topics-api-crawler)以了解更多关于它们的信息。

**settings**

运行这个蜘蛛的配置。这是一个 [**Settings**](https://doc.scrapy.org/en/latest/topics/api.html#scrapy.settings.Settings)实例，请参阅 [设置](https://doc.scrapy.org/en/latest/topics/settings.html#topics-settings)主题以获取关于此主题的详细介绍。

**logger**

Python记录器是用Spider创建的[**name**](https://doc.scrapy.org/en/latest/topics/spiders.html#scrapy.spiders.Spider.name)。您可以按照[从蜘蛛记录中](https://doc.scrapy.org/en/latest/topics/logging.html#topics-logging-from-spiders)所述，使用它来发送日志消息 。

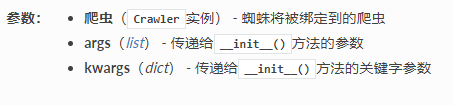
**from\_crawler(crawler, \*args, \*\*kwargs)**

这是Scrapy用来创建蜘蛛的类方法。

你可能不需要直接覆盖它，因为默认的实现作为方法的代理**\_\_init\_\_()**，用给定的参数args和命名参数kwargs调用它。

尽管如此，这个方法 在新实例中设置[**crawler**](https://doc.scrapy.org/en/latest/topics/spiders.html#scrapy.spiders.Spider.crawler)和[**settings**](https://doc.scrapy.org/en/latest/topics/spiders.html#scrapy.spiders.Spider.settings)属性，以便在蜘蛛代码中稍后访问它们。

|  |  |
| --- | --- |
| **Parameters:** | * **crawler** ([**Crawler**](https://doc.scrapy.org/en/latest/topics/api.html#scrapy.crawler.Crawler) instance) – crawler to which the spider will be bound * **args** ([*list*](https://doc.scrapy.org/en/latest/topics/api.html#scrapy.loader.SpiderLoader.list)) – arguments passed to the **\_\_init\_\_()** method * **kwargs** (*dict*) – keyword arguments passed to the **\_\_init\_\_()** method |

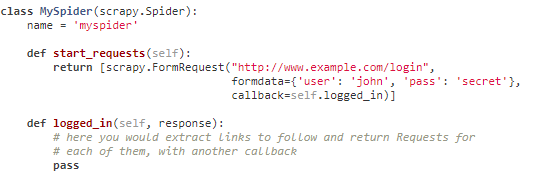


**start\_requests（）**

此方法必须返回一个可迭代的第一个请求以抓取此蜘蛛。Scrapy在蜘蛛被打开时被Scrapy调用。Scrapy只会调用它一次，因此[**start\_requests()**](https://doc.scrapy.org/en/latest/topics/spiders.html#scrapy.spiders.Spider.start_requests)作为生成器实现是安全 的。

默认实现会 为每个网址生成。**Request(url, dont\_filter=True)**[**start\_urls**](https://doc.scrapy.org/en/latest/topics/spiders.html#scrapy.spiders.Spider.start_urls)

如果您想更改用于开始抓取域的请求，则这是要覆盖的方法。例如，如果您需要使用POST请求登录，则可以执行以下操作：



**parse(response)**

这是Scrapy用来处理下载响应的默认回调，当它们的请求没有指定回调时。

该**parse**方法负责处理响应并返回所刮取的数据和/或更多的URL。其他请求回调与[**Spider**](https://doc.scrapy.org/en/latest/topics/spiders.html#scrapy.spiders.Spider)类有相同的要求。

这个方法以及任何其他的Request回调都必须返回可迭代的[**Request**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.Request)和/或字典或[**Item**](https://doc.scrapy.org/en/latest/topics/items.html#scrapy.item.Item)对象。

|  |  |
| --- | --- |
| **参数：** | **response**（[**Response**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.Response)） - 解析的响应 |

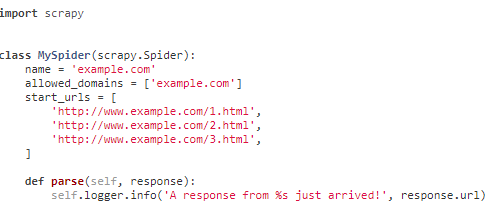
**log(message[, level, component])**

通过Spider发送日志消息的包装器[**logger**](https://doc.scrapy.org/en/latest/topics/spiders.html#scrapy.spiders.Spider.logger)，保持向后兼容性。有关更多信息，请参阅 [从蜘蛛记录](https://doc.scrapy.org/en/latest/topics/logging.html#topics-logging-from-spiders)。

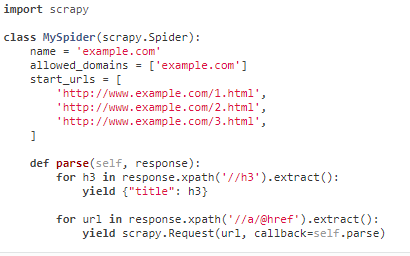
**closed(reason)**

当蜘蛛关闭时调用。此方法为signal.connect（）提供了[**spider\_closed**](https://doc.scrapy.org/en/latest/topics/signals.html#std:signal-spider_closed)信号的快捷方式。

我们来看一个例子：



从单个回调中返回多个请求和项目：



而不是[**start\_urls**](https://doc.scrapy.org/en/latest/topics/spiders.html#scrapy.spiders.Spider.start_urls)你可以[**start\_requests()**](https://doc.scrapy.org/en/latest/topics/spiders.html#scrapy.spiders.Spider.start_requests)直接使用; 给数据更多的结构你可以使用[Items](https://doc.scrapy.org/en/latest/topics/items.html#topics-items)：



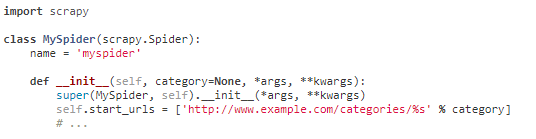
## 蜘蛛的论点

蜘蛛可以接收修改其行为的参数。蜘蛛参数的一些常见用途是定义起始URL或将爬网限制到站点的某些部分，但它们可用于配置蜘蛛的任何功能。

蜘蛛参数[**crawl**](https://doc.scrapy.org/en/latest/topics/commands.html#std:command-crawl)使用该-a选项通过命令 传递。例如：



蜘蛛可以在\_\_init\_\_方法中访问参数：



默认的\_\_init\_\_方法将采用任何蜘蛛参数并将其作为属性复制到蜘蛛中。上面的例子也可以写成如下：



请记住，蜘蛛参数只是字符串。蜘蛛不会自行解析。如果要从命令行设置start\_urls属性，则必须使用[ast.literal\_eval](https://docs.python.org/library/ast.html#ast.literal_eval) 或[json.loads之](https://docs.python.org/library/json.html#json.loads)类的东西将它自己解析为列表 ，然后将其设置为属性。否则，你会导致对一个start\_urls字符串进行迭代（一个非常常见的python陷阱），导致每个字符被视为一个单独的url。

有效的用例是设置由[**HttpAuthMiddleware**](https://doc.scrapy.org/en/latest/topics/downloader-middleware.html#scrapy.downloadermiddlewares.httpauth.HttpAuthMiddleware) 以下用户使用的http认证凭证或用户代理[**UserAgentMiddleware**](https://doc.scrapy.org/en/latest/topics/downloader-middleware.html#scrapy.downloadermiddlewares.useragent.UserAgentMiddleware)：

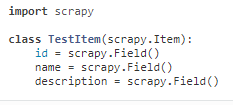


蜘蛛参数也可以通过Scrapyd schedule.jsonAPI 传递。请参阅[Scrapyd文档](https://scrapyd.readthedocs.io/en/latest/)。

## 通用蜘蛛

Scrapy附带了一些有用的通用蜘蛛，您可以使用它们对蜘蛛进行子类化。他们的目标是为一些常见的刮抢案例提供方便的功能，例如遵循特定规则的网站上的所有链接，从[Sitemaps](https://www.sitemaps.org/index.html)抓取或解析XML / CSV供稿。

对于下面的蜘蛛中使用的例子，我们假设你有一个TestItem在myproject.items模块中声明的项目：



### CrawlSpider

**Class scrapy.spiders.CrawlSpider**

这是抓取常规网站最常用的蜘蛛，因为它提供了一个通过定义一组规则来跟踪链接的便捷机制。它可能不是最适合您的特定网站或项目的，但它对于多种情况是足够通用的，所以您可以从它开始并根据需要覆盖它以获得更多自定义功能，或者只是实现您自己的蜘蛛。

除了从Spider继承的属性（必须指定）之外，该类还支持一个新的属性：

**rules**

哪个是一个（或多个）[**Rule**](https://doc.scrapy.org/en/latest/topics/spiders.html#scrapy.spiders.Rule)对象的列表。每个都[**Rule**](https://doc.scrapy.org/en/latest/topics/spiders.html#scrapy.spiders.Rule) 定义了用于爬网的特定行为。规则对象如下所述。如果多个规则匹配相同的链接，则会根据它在此属性中定义的顺序使用第一个规则。

这个蜘蛛也暴露了一个可覆盖的方法：

**parse\_start\_url*（*response*）***

*这个方法被称为start\_urls响应。*它允许解析初始响应，并且必须返回一个 [**Item**](https://doc.scrapy.org/en/latest/topics/items.html#scrapy.item.Item)对象，或者一个[**Request**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.Request) 对象或包含它们的迭代器。

#### 抓取规则

**Class scrapy.spiders.Rule（link\_extractor，callback = None，cb\_kwargs = None，follow = None，process\_links = None，process\_request = None ）**

**link\_extractor**是一个[链接提取器](https://doc.scrapy.org/en/latest/topics/link-extractors.html#topics-link-extractors)对象，它定义了如何从每个已爬网页中提取链接。

**callback**是一个可调用的或字符串（在这种情况下，将使用具有该名称的蜘蛛对象的方法）针对使用指定的link\_extractor提取的每个链接调用。这个回调接收到一个响应作为它的第一个参数，并且必须返回一个包含[**Item**](https://doc.scrapy.org/en/latest/topics/items.html#scrapy.item.Item)和/或 [**Request**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.Request)对象（或者它们的任何子类）的列表。

**警告**

编写爬网规则时，避免使用**parse**回调函数，因为它[**CrawlSpider**](https://doc.scrapy.org/en/latest/topics/spiders.html#scrapy.spiders.CrawlSpider)使用**parse**方法本身来实现其逻辑。因此，如果您重写该**parse**方法，抓取蜘蛛将不再起作用。

**cb\_kwargs** 是一个包含要传递给回调函数的关键字参数的字典。

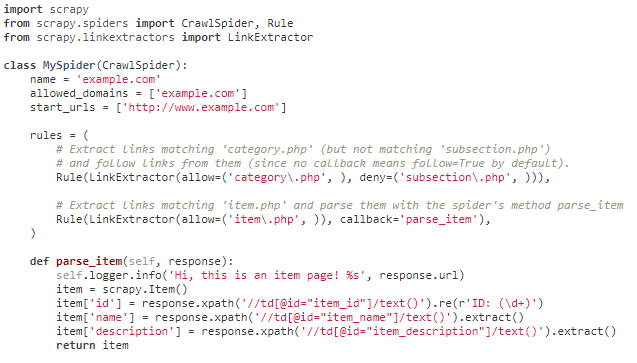
**follow**是一个布尔值，用于指定是否应该遵循通过此规则提取的每个响应之间的链接。如果**callback**是无**follow**默认值**True**，则默认为**False**。

**process\_links**是可调用的或字符串（在这种情况下，将使用具有该名称的蜘蛛对象的方法），将使用指定的对每个响应中提取的每个链接列表调用该方法**link\_extractor**。这主要用于过滤目的。

**process\_request** 是可调用的，或者是一个字符串（在这种情况下，将使用具有该名称的蜘蛛对象的方法），该方法将在此规则提取的每个请求中调用，并且必须返回请求或None（以过滤出请求） 。

#### CrawlSpider示例

现在我们来看一个带有规则的示例CrawlSpider：



这个蜘蛛将开始抓取example.com的主页，收集类别链接和项目链接，并用该parse\_item方法解析后者。对于每个项目响应，将使用XPath从HTML中提取一些数据，并将[**Item**](https://doc.scrapy.org/en/latest/topics/items.html#scrapy.item.Item)填充它。

### XMLFeedSpider

**Class scrapy.spiders.XMLFeedSpider**

XMLFeedSpider旨在通过以特定节点名称遍历它们来解析XML提要。迭代器可以选自：**iternodes**，**xml**，和**html**。我们推荐使用**iternodes**性能方面的原因迭代器，因为**xml**和**html**迭代器，以便分析它立刻生成整个DOM。但是，**html**在解析具有错误标记的XML时，用作迭代器可能会很有用。

要设置迭代器和标签名称，您必须定义以下类属性：

**iterator**

一个定义要使用的迭代器的字符串。它可以是：

* **'iternodes'** - 基于正则表达式的快速迭代器
* **'html'**- 一个使用的迭代器[**Selector**](https://doc.scrapy.org/en/latest/topics/selectors.html#scrapy.selector.Selector)。请记住，这使用DOM解析，并且必须加载内存中的所有DOM，这可能是大型提要的问题
* **'xml'**- 一个使用的迭代器[**Selector**](https://doc.scrapy.org/en/latest/topics/selectors.html#scrapy.selector.Selector)。请记住，这使用DOM解析，并且必须加载内存中的所有DOM，这可能是大型提要的问题

它默认为：**'iternodes'**。

**itertag**

一个字符串，其中包含要迭代的节点（或元素）的名称。示例：



**namespaces**

定义该文档中可用的名称空间的元组列表，该列表将与该蜘蛛一起处理。的 和将被用于自动注册使用的命名空间 的方法。**(prefix, uri) prefixuri** [**register\_namespace()**](https://doc.scrapy.org/en/latest/topics/selectors.html#scrapy.selector.Selector.register_namespace)

然后，您可以在[**itertag**](https://doc.scrapy.org/en/latest/topics/spiders.html#scrapy.spiders.XMLFeedSpider.itertag) 属性中指定具有名称空间的节点。

例：

**class**  **YourSpider** （XMLFeedSpider ）：

namespaces **=**  [（'n' ， 'http://www.sitemaps.org/schemas/sitemap/0.9' ）]

itertag **=**  'n：url'

*＃...*

除了这些新的属性，这个蜘蛛还有以下可覆盖的方法：

**adapt\_response(response)**

在蜘蛛开始分析它之前，一旦从蜘蛛中间件到达响应，它就会收到响应。在解析它之前，它可以用来修改响应主体。这个方法接收一个响应，并返回一个响应（它可能是相同的或另一个）

**parse\_node(response, selector)**

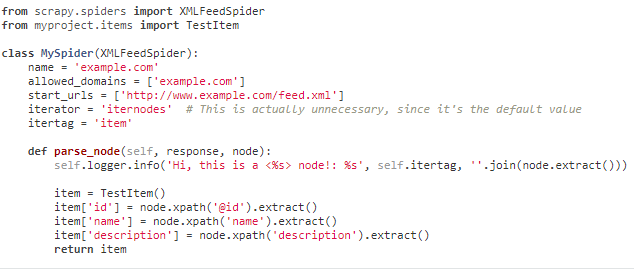
为与提供的标签名称（**itertag**）匹配的节点调用此方法。接收响应并[**Selector**](https://doc.scrapy.org/en/latest/topics/selectors.html#scrapy.selector.Selector)为每个节点分配一个响应 。覆盖此方法是强制性的。否则，你的蜘蛛将无法工作。此方法必须返回一个[**Item**](https://doc.scrapy.org/en/latest/topics/items.html#scrapy.item.Item)对象，一个 [**Request**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.Request)对象或包含它们的迭代器。

**process\_results(response, results)**

该方法针对蜘蛛所返回的每个结果（项目或请求）进行调用，并且它将在将结果返回给框架核心之前执行所需的最后一次处理，例如设置项目ID。它会收到一份结果清单和源自这些结果的回复。它必须返回结果列表（项目或请求）。

#### XMLFeedSpider示例

这些蜘蛛很容易使用，让我们看看一个例子：



基本上我们在那里做的是创建一个蜘蛛，从给定的下载饲料start\_urls，然后遍历每个item标签，打印出来，并存储一些随机数据[**Item**](https://doc.scrapy.org/en/latest/topics/items.html#scrapy.item.Item)。

### CSVFeedSpider

**Class scrapy.spiders.CSVFeedSpider**

这个蜘蛛与XMLFeedSpider非常相似，只是它遍历行而不是节点。在每次迭代中调用的方法是[**parse\_row()**](https://doc.scrapy.org/en/latest/topics/spiders.html#scrapy.spiders.CSVFeedSpider.parse_row)。

**delimiter**

包含CSV文件中每个字段的分隔符的字符串默认为**','**（逗号）。

**quotechar**

包含CSV文件中每个字段的外壳字符的字符串默认为**'"'**（引号）。

**headers**

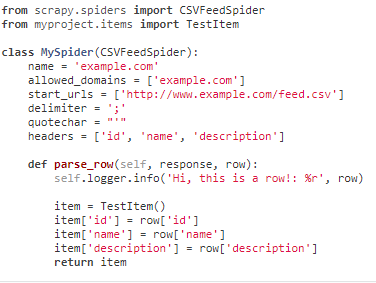
CSV文件中列名称的列表。

**parse\_row(response, row)**

接收每个提供（或检测）的CSV文件标题的响应和字典（代表每行）。这个蜘蛛也提供了覆盖的机会**adapt\_response**以及**process\_results**用于预处理和后处理的方法。

#### CSVFeedSpider示例

我们来看一个和上一个类似的例子，但是使用 [**CSVFeedSpider**](https://doc.scrapy.org/en/latest/topics/spiders.html#scrapy.spiders.CSVFeedSpider)：



### SitemapSpider

**Class scrapy.spiders.SitemapSpider**

SitemapSpider允许您通过使用[Sitemaps](https://www.sitemaps.org/index.html)发现网址来抓取 [网站](https://www.sitemaps.org/index.html)。

它支持嵌套站点地图并从[robots.txt中](http://www.robotstxt.org/)发现站点地图网址 。

**sitemap\_urls**

指向您要抓取的网址的站点地图的网址列表。

您也可以指向一个[robots.txt](http://www.robotstxt.org/)，并将其解析为从中提取网站地图网址。

**sitemap\_rules**

元组列表，其中：**(regex, callback)**

* **regex**是一个正则表达式，用于匹配从站点地图提取的网址。 **regex**可以是str或编译的正则表达式对象。
* 回调是用于处理匹配正则表达式的url的回调。**callback**可以是一个字符串（表示蜘蛛方法的名称）或可调用的字符串。

例如：

sitemap\_rules **=**  [（'/ product /' ， 'parse\_product' ）]

规则按顺序应用，只有匹配的第一个将被使用。

如果你忽略这个属性，所有在站点地图中找到的URL都将被**parse**回调处理。

**sitemap\_follow**

应遵循的站点地图正则表的列表。这仅适用于使用指向其他站点地图文件的[站点地图索引文件的](https://www.sitemaps.org/protocol.html#index)站点。

默认情况下，遵循所有站点地图。

**sitemap\_alternate\_links**

指定是否**url**应该遵循一个替代链接。这些是在同一个**url**区块内传递的另一种语言的同一网站的链接。

例如：

**<** url **>**

**<** loc **>** http ：**//** 示例**。**com **/ </** loc **>**

**<** xhtml ：link rel **=** “alternate” hreflang **=** “de” href **=** “http://example.com/de” **/>**

**</** url **>**

有了**sitemap\_alternate\_links**集，这将同时检索的网址。随着 **sitemap\_alternate\_links**禁用，只**http://example.com/**将被检索。

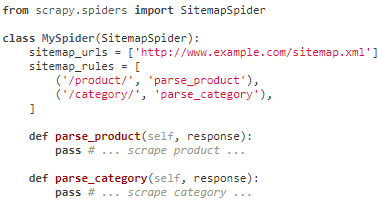
默认是**sitemap\_alternate\_links**禁用的。

#### SitemapSpider的例子

最简单的例子：使用parse回调处理通过站点地图发现的所有网址 ：



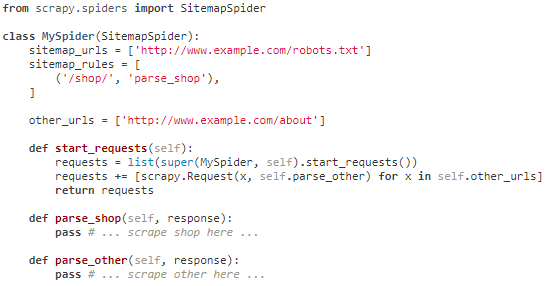
使用不同的回叫处理某些回叫和其他网址的某些网址：



遵循[robots.txt](http://www.robotstxt.org/)文件中定义的站点地图，并且只跟随其网址包含以下内容的站点地图/sitemap\_shop：



将SitemapSpider与其他网址来源相结合：



# Selectors

在抓取网页时，需要执行的最常见任务是从HTML源中提取数据。有几个库可以实现这一点：

* [BeautifulSoup](https://www.crummy.com/software/BeautifulSoup/)是Python程序员中非常流行的网页抓取库，它基于HTML代码的结构构建了一个Python对象，同时也很好地处理了糟糕的标记，但它有一个缺点：速度很慢。
* [lxml](http://lxml.de/)是一个基于[ElementTree](https://docs.python.org/2/library/xml.etree.elementtree.html)的pythonic API的XML解析库（它也解析HTML）。（lxml不是Python标准库的一部分。）

Scrapy具有自己的提取数据的机制。它们被称为选择器，因为它们“选择”由[XPath](https://www.w3.org/TR/xpath)或[CSS](https://www.w3.org/TR/selectors)表达式指定的HTML文档的某些部分。

[XPath](https://www.w3.org/TR/xpath)是一种用于选择XML文档中的节点的语言，它也可以用于HTML。[CSS](https://www.w3.org/TR/selectors)是一种将样式应用于HTML文档的语言。它定义选择器将这些样式与特定的HTML元素相关联。

Scrapy选择器是在[lxml](http://lxml.de/)库上构建的，这意味着它们在速度和分析准确性方面非常相似。

本页解释了选择器如何工作和描述其非常小和简单的API ，不像[lxml](http://lxml.de/) API更大，因为 [lxml](http://lxml.de/)库除了选择标记文档之外还可以用于许多其他任务。

有关选择器API的完整参考，请参阅 [选择器参考](https://doc.scrapy.org/en/latest/topics/selectors.html#topics-selectors-ref)

## 使用选择器

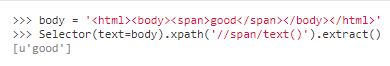
### 构建选择器

Scrapy选择器是[**Selector**](https://doc.scrapy.org/en/latest/topics/selectors.html#scrapy.selector.Selector)通过传递**文本**或[**TextResponse**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.TextResponse) 对象构造的类的实例。它会根据输入类型自动选择最佳的解析规则（XML vs HTML）：

>>> **from**  scrapy.selector **import**  Selector

>>> **from**  scrapy.http **import**  HtmlResponse

从文本构建：



从响应构建：



为方便起见，响应对象在.selector属性上公开选择器，在可能的情况下使用此快捷方式完全可以：



### 使用选择器

为了解释如何使用选择器，我们将使用Scrapy shell（提供交互式测试）以及位于Scrapy文档服务器中的示例页面：

<https://doc.scrapy.org/en/latest/_static/selectors-sample1.html>

这里是它的HTML代码：



首先，我们打开shell：



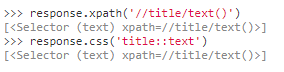
然后，在加载shell之后，您将有可用的响应作为response shell变量，并将其附加的选择器作为response.selector属性。

由于我们正在处理HTML，选择器将自动使用HTML解析器。

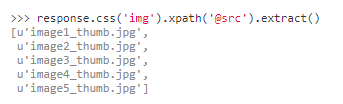
因此，通过查看该页面的[HTML代码](https://doc.scrapy.org/en/latest/topics/selectors.html#topics-selectors-htmlcode)，我们构建一个用于选择标题标签内的文本的XPath：



使用XPath和CSS查询响应非常常见，以至于响应包含两个便捷快捷键：response.xpath()和response.css()：



正如你所看到的，.xpath()并且.css()方法返回一个 [**SelectorList**](https://doc.scrapy.org/en/latest/topics/selectors.html#scrapy.selector.SelectorList)实例，这是新的选择列表。该API可用于快速选择嵌套数据：



要实际提取文本数据，您必须调用选择器.extract() 方法，如下所示：



如果你只想提取第一个匹配的元素，你可以调用选择器 .extract\_first()



None如果没有找到元素，它会返回：



默认返回值可以作为参数提供，用来代替None：



请注意，CSS选择器可以使用CSS3伪元素选择文本或属性节点：

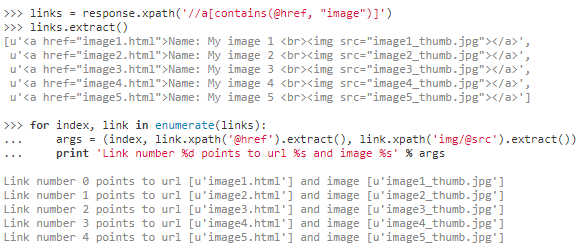


现在我们要获取基本URL和一些图像链接：



### 嵌套选择器

选择方法（.xpath()或.css()）返回相同类型的选择器列表，因此您也可以调用这些选择器的选择方法。这是一个例子：



### 使用正则表达式的选择器

[**Selector**](https://doc.scrapy.org/en/latest/topics/selectors.html#scrapy.selector.Selector)也有.re()使用正则表达式提取数据的方法。但是，与使用.xpath()或 .css()方法不同，.re()返回一个unicode字符串列表。所以你不能构建嵌套.re()调用。

以下是一个用于从上面的[HTML代码中](https://doc.scrapy.org/en/latest/topics/selectors.html#topics-selectors-htmlcode)提取图像名称的示例：



这里有一个额外的辅助往复.extract\_first()的.re()，命名.re\_first()。使用它只提取第一个匹配的字符串：



### 使用相对的XPath

请记住，如果您嵌套选择器并使用以XPath开头的XPath /，那么XPath对于文档来说绝对是绝对的，而不是相对于 Selector您从中调用它。

例如，假设你想提取<p>元素内的所有<div> 元素。首先，你会得到所有<div>元素：



起初，您可能会使用以下方法，这是错误的，因为它实际上会<p>从文档中提取所有元素，而不仅仅是<div>元素中的所有元素：



这是做到这一点的正确方法（注意.//pXPath 前缀的点）：



另一个常见的情况是提取所有直接的<p>孩子：



有关相关XPath的更多详细信息，请参阅XPath规范中的[位置路径](https://www.w3.org/TR/xpath#location-paths)部分。

### XPath表达式中的变量

XPath允许使用$somevariable语法在XPath表达式中引用变量。这与SQL世界中的参数化查询或预准备语句有些类似，您可以使用占位符替换查询中的某些参数，?然后用占位符传递的值替换。

下面是一个基于其“id”属性值匹配元素的示例，不用对其进行硬编码（以前显示过）：



下面是另一个例子，为了找到<div>包含五个<a>孩子的标签的“id”属性（这里我们将该值5作为整数传递）：



所有变量引用在调用时都必须有一个绑定值.xpath() （否则你会得到一个异常）。这是通过根据需要传递许多命名参数来完成的。ValueError: XPath error:

[parsel](https://parsel.readthedocs.io/)，为Scrapy选择器提供动力的库，有更多关于[XPath变量的](https://parsel.readthedocs.io/en/latest/usage.html#variables-in-xpath-expressions)细节和例子。

### 使用EXSLT扩展

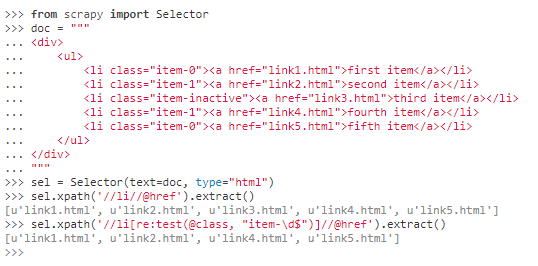
构建在[lxml](http://lxml.de/)之上，Scrapy选择器还支持一些[EXSLT](http://exslt.org/)扩展，并提供这些预先注册的名称空间以用于XPath表达式：

| **字首** | **命名空间** | **用法** |
| --- | --- | --- |
| 回覆 | http://exslt.org/regular-expressions | [常用表达](http://exslt.org/regexp/index.html) |
| 组 | http://exslt.org/sets | [设置操作](http://exslt.org/set/index.html) |

#### 常用表达

test()例如，当XPath starts-with()或者contains()不够用时，这个函数可以证明是非常有用的 。

使用以数字结尾的“class”属性选择列表项中链接的示例：



**警告**

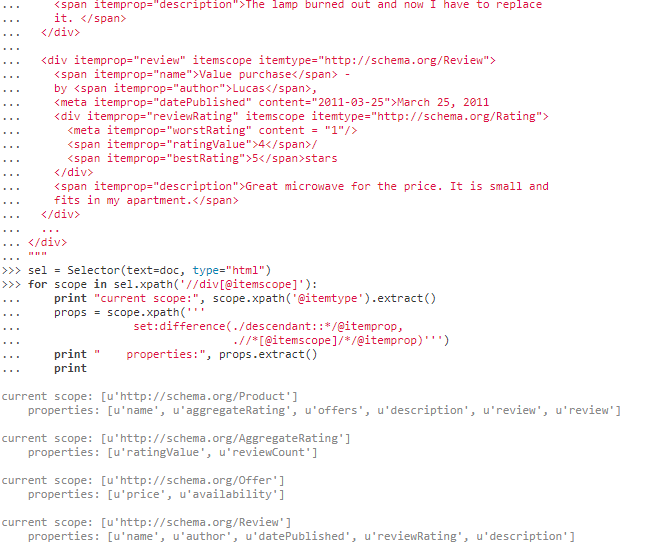
C库libxslt本身不支持EXSLT正则表达式，所以[lxml](http://lxml.de/)的实现在Python re模块中使用钩子。因此，在XPath表达式中使用正则表达式函数可能会增加一点性能损失。

#### 设置操作

例如，在提取文本元素之前，这些可以方便地排除文档树的部分内容。

使用itemscopes组和相应的itemprops 提取微数据（从<http://schema.org/Product>获取样本内容）的示例：





在这里，我们首先迭代itemscope元素，并为每个itemprops元素寻找所有元素并排除那些在另一个元素中的元素itemscope。

### 一些XPath技巧

以下是一些技巧，您可能会发现在使用Scrapy选择器的XPath时有用，基于[ScrapingHub博客的这篇文章](https://blog.scrapinghub.com/2014/07/17/xpath-tips-from-the-web-scraping-trenches/)。如果您还不太熟悉XPath，那么您可能需要先看看这个[XPath教程](http://www.zvon.org/comp/r/tut-XPath_1.html)。

#### 在条件中使用文本节点

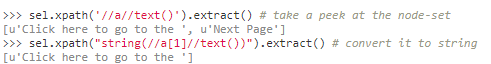
当您需要使用文本内容作为[XPath字符串函数的](https://www.w3.org/TR/xpath/#section-String-Functions)参数时，请避免使用.//text()和使用.。

这是因为表达式.//text()产生了一组文本元素 - 一个节点集。当一个节点集被转换为一个字符串时，当它作为参数传递给一个字符串函数（如contains()or）时starts-with()，会产生仅用于第一个元素的文本。

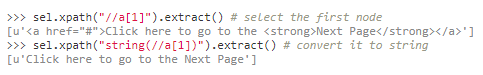
例：



将节点集转换为字符串：



一个节点转换为字符串，但是，拼文本的本身及其所有的后代：



所以，.//text()在这种情况下使用节点集将不会选择任何内容：



但使用.意味着节点，工作：

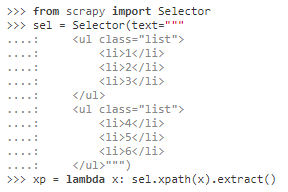


#### 当心//节点[1]和（//节点）[1]之间的区别

//node[1] 选择在其各自父母下首先发生的所有节点。

(//node)[1] 选择文档中的所有节点，然后只获取其中的第一个节点。

例：



<li> 无论它是它的父级，它都会获得所有的第一个元素：



这就得到<li> 了整个文档中的第一个元素：



这将获得 父项<li> 下的所有第一个元素<ul>：



这将获得 整个文档中父项<li> 下的第一个元素<ul>：



#### 按课程查询时，请考虑使用CSS

由于一个元素可以包含多个CSS类，因此按类选择元素的XPath方法相当冗长：



如果你使用了，@class='someclass'你最终可能会丢失具有其他类的元素，如果你只是用来弥补这一点，你可能会得到更多的元素，如果他们有不同的类名称共享字符串。contains(@class, 'someclass')someclass

事实证明，Scrapy选择器允许您链接选择器，所以大多数情况下，您可以使用CSS按类选择，然后在需要时切换到XPath：

>>> **from** scrapy **import** Selector

>>> sel **=** Selector(text**=**'<div class="hero shout"><time datetime="2014-07-23 19:00">Special date</time></div>')

>>> sel**.**css('.shout')**.**xpath('./time/@datetime')**.**extract()

[u'2014-07-23 19:00']

这比使用上面显示的详细XPath技巧更清晰。只要记住.在随后的XPath表达式中使用。

## 内置选择器参考

### 选择器对象

***Class* scrapy.selector.Selector（*response = None*，*text = None*，*type = None*）**

一个实例[**Selector**](https://doc.scrapy.org/en/latest/topics/selectors.html#scrapy.selector.Selector)是对选择其内容某些部分的响应封装。

**response**是将用于选择和提取数据的一个[**HtmlResponse**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.HtmlResponse)或一个 [**XmlResponse**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.XmlResponse)对象。

**text**当a **response**不可用时，它是一个unicode字符串或utf-8编码文本 。使用**text**和**response**一起是未定义的行为。

**type**定义选择器类型，它可以是**"html"**，**"xml"**或**None**（默认）。

如果**type**是**None**，选择器将根据**response**类型自动选择最佳类型（请参见下文），或者默认为**"html"**与其一起使用**text**。

如果**type**is **None**和a **response**被传递，则选择器类型从响应类型推断如下：

* **"html"**为[**HtmlResponse**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.HtmlResponse)类型
* **"xml"**为[**XmlResponse**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.XmlResponse)类型
* **"html"** 为其他任何事情

否则，如果**type**设置，选择器类型将被强制并且不会进行检测。

**xpath(query)**

查找与xpath匹配的节点**query**，并将结果作为[**SelectorList**](https://doc.scrapy.org/en/latest/topics/selectors.html#scrapy.selector.SelectorList)实例返回， 并将所有元素展平。列表元素也实现了[**Selector**](https://doc.scrapy.org/en/latest/topics/selectors.html#scrapy.selector.Selector)接口。

**query** 是一个包含要应用的XPATH查询的字符串。

**注意**

为了方便，这个方法可以被称为 **response.xpath()**

**css(query)**

应用给定的CSS选择器并返回一个[**SelectorList**](https://doc.scrapy.org/en/latest/topics/selectors.html#scrapy.selector.SelectorList)实例。

**query** 是一个包含要应用的CSS选择器的字符串。

在后台，使用[cssselect](https://pypi.python.org/pypi/cssselect/)库和run **.xpath()**方法将CSS查询转换为XPath查询 。

**注意**

为了方便起见，这个方法可以称为 **response.css()**

**extract()**

序列化并返回匹配的节点作为unicode字符串列表。编码百分比的内容未加引号。

应用给定的正则表达式并返回包含匹配的unicode字符串列表。

**re(regex)**

**regex** 可以是已编译的正则表达式，也可以是将使用正则表达式编译为正则表达式的字符串 **re.compile(regex)**

**注意**

请注意，**re()**并且**re\_first()**都解码HTML实体（除了**&lt;**和**&amp;**）。

**register\_namespace(prefix, uri)**

注册在此使用的给定名称空间[**Selector**](https://doc.scrapy.org/en/latest/topics/selectors.html#scrapy.selector.Selector)。如果不注册名称空间，则无法从非标准名称空间中选择或提取数据。看下面的例子。

**remove\_namespaces()**

删除所有名称空间，允许使用不含名称空间的xpaths来遍历文档。见下面的例子。

**\_\_nonzero\_\_()**

删除所有名称空间，允许使用不含名称空间的xpaths来遍历文档。见下面的例子。

### SelectorList对象

**Class scrapy.selector.SelectorList**

这个[**SelectorList**](https://doc.scrapy.org/en/latest/topics/selectors.html#scrapy.selector.SelectorList)类是内建**list** 类的一个子类，它提供了一些额外的方法。

**xpath(query)**

调用**.xpath()**此列表中每个元素的方法并将其结果展平为另一个[**SelectorList**](https://doc.scrapy.org/en/latest/topics/selectors.html#scrapy.selector.SelectorList)。

**query** 与在英国的论点是一样的 [**Selector.xpath()**](https://doc.scrapy.org/en/latest/topics/selectors.html#scrapy.selector.Selector.xpath)

**css(query)**

调用**.css()**此列表中每个元素的方法并将其结果展平为另一个[**SelectorList**](https://doc.scrapy.org/en/latest/topics/selectors.html#scrapy.selector.SelectorList)。

**query** 与在英国的论点是一样的 [**Selector.css()**](https://doc.scrapy.org/en/latest/topics/selectors.html#scrapy.selector.Selector.css)

**extract()**

调用**.extract()**此列表中每个元素的方法并将其结果展平，作为unicode字符串列表。

**re()**

调用**.re()**此列表中每个元素的方法并将其结果展平，作为unicode字符串列表。

### HTML响应的选择器示例

这里有几个[**Selector**](https://doc.scrapy.org/en/latest/topics/selectors.html#scrapy.selector.Selector)例子来说明几个概念。在所有情况下，我们都假设已经[**Selector**](https://doc.scrapy.org/en/latest/topics/selectors.html#scrapy.selector.Selector)有一个[**HtmlResponse**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.HtmlResponse)像这样的对象实例化了：



<h1>从HTML响应主体中选择所有元素，返回[**Selector**](https://doc.scrapy.org/en/latest/topics/selectors.html#scrapy.selector.Selector)对象列表 （即[**SelectorList**](https://doc.scrapy.org/en/latest/topics/selectors.html#scrapy.selector.SelectorList)对象）：



<h1>从HTML响应主体中提取所有元素的文本，返回一个unicode字符串列表：



遍历所有<p>标签并打印它们的类属性：



### XML响应的选择器示例

这里有几个例子来说明几个概念。在这两种情况下，我们都假设已经[**Selector**](https://doc.scrapy.org/en/latest/topics/selectors.html#scrapy.selector.Selector)有一个[**XmlResponse**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.XmlResponse)像这样的对象实例化了 ：



<product>从XML响应主体中选择所有元素，返回[**Selector**](https://doc.scrapy.org/en/latest/topics/selectors.html#scrapy.selector.Selector)对象列表（即[**SelectorList**](https://doc.scrapy.org/en/latest/topics/selectors.html#scrapy.selector.SelectorList)对象）：



从需要注册名称空间的[Google Base XML Feed中](https://support.google.com/merchants/answer/160589?hl=en&ref_topic=2473799)提取所有价格：



### 删除命名空间

在处理拼图项目时，通常完全摆脱名称空间并仅使用元素名称来编写更简单/方便的XPath是非常方便的。你可以使用该 [**Selector.remove\_namespaces()**](https://doc.scrapy.org/en/latest/topics/selectors.html#scrapy.selector.Selector.remove_namespaces)方法。

我们来展示一个用GitHub博客atom feed来说明这个问题的例子。

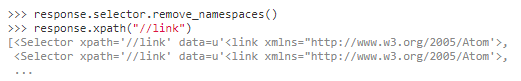
首先，我们用我们想要刮取的url打开shell：

$ scrapy shell https://github.com/blog.atom

一旦进入shell，我们可以尝试选择所有<link>对象并查看它不起作用（因为Atom XML名称空间正在模糊这些节点）：



但是一旦我们调用这个[**Selector.remove\_namespaces()**](https://doc.scrapy.org/en/latest/topics/selectors.html#scrapy.selector.Selector.remove_namespaces)方法，所有的节点都可以直接用它们的名字来访问：



如果您想知道为什么命名空间删除过程并不总是被默认调用，而不必手动调用它，这是因为两个原因，按照相关性顺序，这两个原因是：

1. 删除命名空间需要迭代和修改文档中的所有节点，这对于Scrapy搜索的所有文档来说是一个相当昂贵的操作
2. 在某些情况下，实际上需要使用名称空间，以防某些元素名称在名称空间之间发生冲突。这些案例虽然非常罕见。

# Items

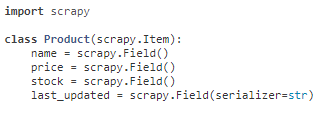
抓取的主要目标是从非结构化来源（通常是网页）中提取结构化数据。Scrapy蜘蛛可以将提取的数据作为Python字典返回。虽然方便且熟悉，但Python字典缺乏结构：在字段名称中输入拼写错误或返回不一致数据很容易，尤其是在包含许多蜘蛛的大型项目中。

定义公共输出数据格式Scrapy提供了这个[**Item**](https://doc.scrapy.org/en/latest/topics/items.html#scrapy.item.Item)类。 [**Item**](https://doc.scrapy.org/en/latest/topics/items.html#scrapy.item.Item)对象是用于收集刮取数据的简单容器。他们提供了一个[类似字典的](https://docs.python.org/2/library/stdtypes.html#dict) API，用于声明其可用字段的便捷语法。

各种Scrapy组件使用Items提供的额外信息：出口商查看已声明的字段以确定要导出的列，可以使用Item字段元数据定制序列化，**trackref** 跟踪Item实例以帮助查找内存泄漏（请参阅[使用trackref调试内存泄漏](https://doc.scrapy.org/en/latest/topics/leaks.html#topics-leaks-trackrefs)）等。

## 声明项目

项目使用简单的类定义语法和[**Field**](https://doc.scrapy.org/en/latest/topics/items.html#scrapy.item.Field) 对象来声明。这里是一个例子：



**注意**

熟悉[Django的人](https://www.djangoproject.com/)会注意到Scrapy项目被声明为类似于[Django Models](https://docs.djangoproject.com/en/dev/topics/db/models/)，除了Scrapy项目更简单，因为没有不同字段类型的概念。

## 项目字段（Item Fields）

[**Field**](https://doc.scrapy.org/en/latest/topics/items.html#scrapy.item.Field)对象用于为每个字段指定元数据。例如，last\_updated以上示例中所示字段的序列化器函数。

您可以为每个字段指定任何类型的元数据。[**Field**](https://doc.scrapy.org/en/latest/topics/items.html#scrapy.item.Field)对象接受的值没有限制。出于同样的原因，没有所有可用元数据密钥的参考列表。在[**Field**](https://doc.scrapy.org/en/latest/topics/items.html#scrapy.item.Field)对象中定义的每个键都可以被不同的组件使用，只有那些组件可以知道它。您也可以根据[**Field**](https://doc.scrapy.org/en/latest/topics/items.html#scrapy.item.Field)自己的需要定义和使用项目中的任何其他 键。[**Field**](https://doc.scrapy.org/en/latest/topics/items.html#scrapy.item.Field)对象的主要目标 是提供一种在一个地方定义所有字段元数据的方法。通常，那些行为依赖于每个字段的组件使用特定的字段键来配置该行为。您必须参考其文档以查看每个组件使用哪些元数据密钥。

请注意，[**Field**](https://doc.scrapy.org/en/latest/topics/items.html#scrapy.item.Field)用于声明项目的对象不会保留为类属性。相反，他们可以通过[**Item.fields**](https://doc.scrapy.org/en/latest/topics/items.html#scrapy.item.Item.fields)属性访问。

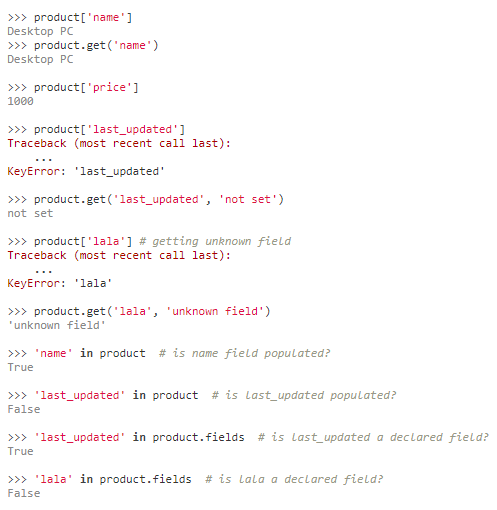
## 使用项目

以下是使用[上面声明](https://doc.scrapy.org/en/latest/topics/items.html#topics-items-declaring)的Product项目对项目执行的常见任务的一些示例 。你会注意到API与[dict API](https://docs.python.org/2/library/stdtypes.html#dict)非常相似。

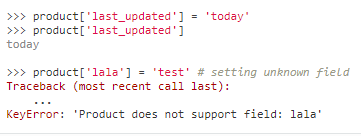
### 创建项目



### 获取字段值（Getting field values）

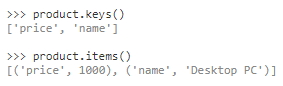


### 设置字段值（Setting field values）



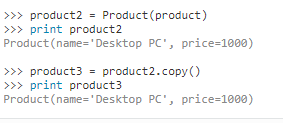
### 访问所有填充值

要访问所有填充值，只需使用典型的[dict API](https://docs.python.org/2/library/stdtypes.html#dict)：



### 其他常见任务

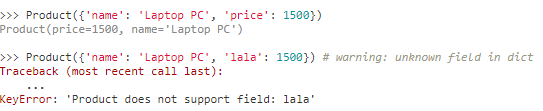
复制项目：



从项目创建字典：



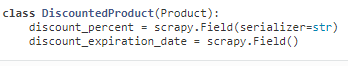
从字典创建项目：



## 扩展项目

您可以通过声明原始项目的子类来扩展项目（以添加更多字段或更改某些字段的某些元数据）。

例如：



您还可以使用以前的字段元数据并附加更多值或更改现有值来扩展字段元数据，如下所示：



这会添加（或替换）该字段的serializer元数据关键字name，并保留所有先前存在的元数据值。

## 项目对象（Item objects）

**Class scrapy.item.Item（[arg ]）**

返回一个新的Item，可以从给定的参数中初始化。

项目复制标准的[字典API](https://docs.python.org/2/library/stdtypes.html#dict)，包括其构造函数。Item提供的唯一附加属性是：

**fields**

一个包含这个Item的所有声明字段的字典，不仅包含那些填充的字段。键是字段名称，值是[**Field**](https://doc.scrapy.org/en/latest/topics/items.html#scrapy.item.Field)在[Item声明中](https://doc.scrapy.org/en/latest/topics/items.html#topics-items-declaring)使用的 对象。

## 字段对象（Field objects）

**Class scrapy.item.Field（[arg ]）**

该[**Field**](https://doc.scrapy.org/en/latest/topics/items.html#scrapy.item.Field)班只是一个别名内置的[字典](https://docs.python.org/2/library/stdtypes.html#dict)类，并没有提供任何额外功能或属性。换句话说， [**Field**](https://doc.scrapy.org/en/latest/topics/items.html#scrapy.item.Field)对象是普通的Python字典。一个单独的类用于支持 基于类属性的[项目声明语法](https://doc.scrapy.org/en/latest/topics/items.html#topics-items-declaring)。

# Item Loaders

项目加载器为填充刮取的[项目](https://doc.scrapy.org/en/latest/topics/items.html#topics-items)提供了一种便利的机制。尽管可以使用自己的字典式API填充项目，但项目加载器通过自动执行一些常见任务（例如在分配原始提取数据之前解析原始提取的数据），为从填充过程中填充它们提供了更方便的API。

换句话说，[Items](https://doc.scrapy.org/en/latest/topics/items.html#topics-items)提供了刮取数据的容器，而Item Loaders提供了填充该容器的机制。

项目加载器旨在提供一种灵活，高效且简单的机制来扩展和覆盖不同的字段解析规则，无论是通过蜘蛛还是通过源格式（HTML，XML等），而不会成为维护的噩梦。

## 使用项目加载器来填充项目

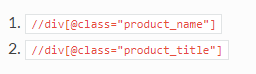
要使用Item Loader，你必须首先实例化它。您可以使用类似字典的对象（例如Item或dict）或没有它的实例化，在这种情况下，使用[**ItemLoader.default\_item\_class**](https://doc.scrapy.org/en/latest/topics/loaders.html#scrapy.loader.ItemLoader.default_item_class) 属性中指定的Item类在Item Loader构造函数中自动实例化Item 。

然后，你开始收集值到Item Loader中，通常使用 [选择器](https://doc.scrapy.org/en/latest/topics/selectors.html#topics-selectors)。您可以将多个值添加到相同的项目字段; Item Loader将知道如何使用适当的处理函数“加入”这些值。

这是一个典型的项目装载机使用的[蜘蛛](https://doc.scrapy.org/en/latest/topics/spiders.html#topics-spiders)，用[产品项目](https://doc.scrapy.org/en/latest/topics/items.html#topics-items-declaring)在申报[项目章节](https://doc.scrapy.org/en/latest/topics/items.html#topics-items)：



通过快速查看代码，我们可以看到该name字段是从页面中的两个不同XPath位置提取的：



换句话说，通过使用该[**add\_xpath()**](https://doc.scrapy.org/en/latest/topics/loaders.html#scrapy.loader.ItemLoader.add_xpath)方法从两个XPath位置提取数据来收集数据。这是稍后将分配给该name字段的数据。

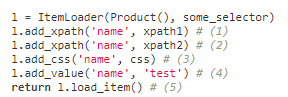
之后，使用类似的调用price和stock字段（后者在[**add\_css()**](https://doc.scrapy.org/en/latest/topics/loaders.html#scrapy.loader.ItemLoader.add_css)方法中使用CSS选择器），最后使用不同的方法last\_update直接使用字面值（today）填充字段：[**add\_value()**](https://doc.scrapy.org/en/latest/topics/loaders.html#scrapy.loader.ItemLoader.add_value)。

最后，收集到的所有数据时，该[**ItemLoader.load\_item()**](https://doc.scrapy.org/en/latest/topics/loaders.html#scrapy.loader.ItemLoader.load_item)方法被调用，实际上返回填充了先前提取并与收集到的数据的项目[**add\_xpath()**](https://doc.scrapy.org/en/latest/topics/loaders.html#scrapy.loader.ItemLoader.add_xpath)， [**add\_css()**](https://doc.scrapy.org/en/latest/topics/loaders.html#scrapy.loader.ItemLoader.add_css)和[**add\_value()**](https://doc.scrapy.org/en/latest/topics/loaders.html#scrapy.loader.ItemLoader.add_value)调用。

## 输入和输出处理器（Input and Output processors）

项目加载器为每个（项目）字段包含一个输入处理器和一个输出处理器。输入处理器只要它接收处理所提取的数据（通过[**add\_xpath()**](https://doc.scrapy.org/en/latest/topics/loaders.html#scrapy.loader.ItemLoader.add_xpath)，[**add\_css()**](https://doc.scrapy.org/en/latest/topics/loaders.html#scrapy.loader.ItemLoader.add_css)或 [**add\_value()**](https://doc.scrapy.org/en/latest/topics/loaders.html#scrapy.loader.ItemLoader.add_value)方法）和输入处理器的结果被收集和保持在ItemLoader内部。收集所有数据后，[**ItemLoader.load\_item()**](https://doc.scrapy.org/en/latest/topics/loaders.html#scrapy.loader.ItemLoader.load_item)调用该 方法来填充并获取已填充的 [**Item**](https://doc.scrapy.org/en/latest/topics/items.html#scrapy.item.Item)对象。这是在输出处理器被调用之前收集的数据（并使用输入处理器处理）的情况下。输出处理器的结果是分配给项目的最终值。

让我们来看一个例子来说明如何为特定字段调用输入和输出处理器（这同样适用于其他字段）：



所以会发生什么是：

1. 来自的数据xpath1被提取，并通过该字段的输入处理器name。输入处理器的结果被收集并保存在Item Loader中（但尚未分配给该项目）。
2. 来自的数据xpath2被提取，并通过（1）中使用的相同的输入处理器。输入处理器的结果附加到（1）中收集的数据（如果有的话）。
3. 除了数据是从cssCSS选择器中提取并通过（1）和（2）中使用的同一个输入处理器之外，这种情况与之前的类似。输入处理器的结果附加到（1）和（2）中收集的数据（如果有的话）。
4. 这种情况也类似于以前的情况，不同之处在于要收集的值是直接分配的，而不是从XPath表达式或CSS选择器中提取。但是，该值仍然通过输入处理器。在这种情况下，由于该值不可迭代，因此在将其传递给输入处理器之前将其转换为单个元素的迭代，因为输入处理器总是接收迭代。
5. 步骤（1），（2），（3）和（4）中收集的数据通过该字段的输出处理器name。输出处理器的结果是分配给name 项目中字段的值。

值得注意的是，处理器仅仅是可调用的对象，它们被调用以解析数据，并返回一个解析的值。所以你可以使用任何功能作为输入或输出处理器。唯一的要求是它们必须接受一个（且只有一个）位置参数，它将是一个迭代器。

**注意**

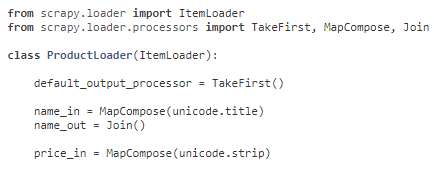
输入和输出处理器都必须接收迭代器作为其第一个参数。这些函数的输出可以是任何东西。输入处理器的结果将被附加到包含收集值（对于该字段）的内部列表（在加载程序中）。输出处理器的结果是最终将分配给该项目的值。

另外需要注意的是输入处理器返回的值在内部收集（以列表形式），然后传递给输出处理器以填充字段。

最后但并非最不重要的，Scrapy附带了一些内置的[常用处理器](https://doc.scrapy.org/en/latest/topics/loaders.html#topics-loaders-available-processors)以方便使用。

## 声明项目加载器（Declaring Item Loaders）

通过使用类定义语法，Item Loaders被声明为Items。这里是一个例子：



如您所见，输入处理器使用\_in后缀声明，而输出处理器使用\_out后缀声明。你也可以使用[**ItemLoader.default\_input\_processor**](https://doc.scrapy.org/en/latest/topics/loaders.html#scrapy.loader.ItemLoader.default_input_processor)和 [**ItemLoader.default\_output\_processor**](https://doc.scrapy.org/en/latest/topics/loaders.html#scrapy.loader.ItemLoader.default_output_processor)属性声明一个默认的输入/输出处理器 。

## 声明输入和输出处理器（Declaring Input and Output Processors）

如前一节所述，可以在Item Loader定义中声明输入和输出处理器，并且以这种方式声明输入处理器是很常见的。但是，还有一个地方可以指定要使用的输入和输出处理器：在“ [项目字段”](https://doc.scrapy.org/en/latest/topics/items.html#topics-items-fields) 元数据中。这里是一个例子：



输入和输出处理器的优先顺序如下：

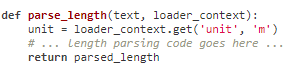
1. 项目加载程序字段特定的属性：field\_in和field\_out（最优先）
2. 字段元数据（input\_processor和output\_processor密钥）
3. 项目加载程序默认值：[**ItemLoader.default\_input\_processor()**](https://doc.scrapy.org/en/latest/topics/loaders.html#scrapy.loader.ItemLoader.default_input_processor)和[**ItemLoader.default\_output\_processor()**](https://doc.scrapy.org/en/latest/topics/loaders.html#scrapy.loader.ItemLoader.default_output_processor)（最低优先级）

另请参阅：[重用和扩展项目装入程序](https://doc.scrapy.org/en/latest/topics/loaders.html#topics-loaders-extending)。

## 项目加载器上下文（Item Loader Context）

Item Loader Context是Item Loader中所有输入和输出处理器共享的任意键/值的字典。它可以在声明，实例化或使用Item Loader时传递。它们用于修改输入/输出处理器的行为。

例如，假设您有一个parse\_length接收文本值并从中提取长度的函数：



通过接受loader\_context参数，函数明确地告诉Item Loader它能够接收Item Loader上下文，因此Item Loader在调用它时传递当前活动的上下文，并且处理函数（parse\_length在这种情况下）可以使用它们。

有几种方法可以修改Item Loader上下文值：

1. 通过修改当前活动的Item Loader上下文（[**context**](https://doc.scrapy.org/en/latest/topics/loaders.html#scrapy.loader.ItemLoader.context)属性）：



2. 在Item Loader实例化（Item Loader构造函数的关键字参数存储在Item Loader上下文中）中：



3. 在Item Loader声明中，对于那些支持使用Item Loader上下文实例化它们的输入/输出处理器。**MapCompose**是其中之一：



## ItemLoader对象

**Class scrapy.loader.ItemLoader([item, selector, response, ]\*\*kwargs)**

返回一个新的Item Loader来填充给定的Item。如果没有给出项目，则使用该类中的一个自动实例化 [**default\_item\_class**](https://doc.scrapy.org/en/latest/topics/loaders.html#scrapy.loader.ItemLoader.default_item_class)。

当使用*选择器*或*响应*参数实例化时，[**ItemLoader**](https://doc.scrapy.org/en/latest/topics/loaders.html#scrapy.loader.ItemLoader)该类提供了使用[选择器](https://doc.scrapy.org/en/latest/topics/selectors.html#topics-selectors)从网页提取数据的便捷机制。

|  |  |
| --- | --- |
| **参数：** | * **项**（[**Item**](https://doc.scrapy.org/en/latest/topics/items.html#scrapy.item.Item)对象） -项目实例来填充利用后续调用 [**add\_xpath()**](https://doc.scrapy.org/en/latest/topics/loaders.html#scrapy.loader.ItemLoader.add_xpath)，[**add\_css()**](https://doc.scrapy.org/en/latest/topics/loaders.html#scrapy.loader.ItemLoader.add_css)或[**add\_value()**](https://doc.scrapy.org/en/latest/topics/loaders.html#scrapy.loader.ItemLoader.add_value)。 * **选择器**（[**Selector**](https://doc.scrapy.org/en/latest/topics/selectors.html#scrapy.selector.Selector)对象） - 使用[**add\_xpath()**](https://doc.scrapy.org/en/latest/topics/loaders.html#scrapy.loader.ItemLoader.add_xpath)（resp。[**add\_css()**](https://doc.scrapy.org/en/latest/topics/loaders.html#scrapy.loader.ItemLoader.add_css)）或[**replace\_xpath()**](https://doc.scrapy.org/en/latest/topics/loaders.html#scrapy.loader.ItemLoader.replace_xpath)（resp。[**replace\_css()**](https://doc.scrapy.org/en/latest/topics/loaders.html#scrapy.loader.ItemLoader.replace_css)）方法时从中提取数据的选择器 。 * **response**（[**Response**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.Response)object） - 用于构造选择器的响应 [**default\_selector\_class**](https://doc.scrapy.org/en/latest/topics/loaders.html#scrapy.loader.ItemLoader.default_selector_class)，除非给出选择器参数，否则此参数将被忽略。 |

项目，选择器，响应和其余关键字参数分配给Loader上下文（可通过[**context**](https://doc.scrapy.org/en/latest/topics/loaders.html#scrapy.loader.ItemLoader.context)属性访问）。

[**ItemLoader**](https://doc.scrapy.org/en/latest/topics/loaders.html#scrapy.loader.ItemLoader) 实例具有以下方法：

**get\_value(value, \*processors, \*\*kwargs)**

**value**通过给定**processors**参数和关键字参数处理给定的参数。

可用关键字参数：

|  |  |
| --- | --- |
| **参数：** | **re**（*str 或编译正则表达式*） - 一种正则表达式**extract\_regex()**，用于在处理器之前应用使用方法从给定值中提取数据 |

例子：



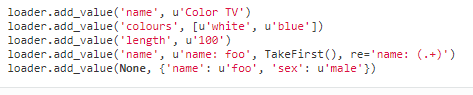
**add\_value(field\_name, value, \*processors, \*\*kwargs)**

处理，然后添加给**value**定字段的给定。

该值首先通过[**get\_value()**](https://doc.scrapy.org/en/latest/topics/loaders.html#scrapy.loader.ItemLoader.get_value)给出 **processors**and来**kwargs**传递，然后通过 [字段输入处理器](https://doc.scrapy.org/en/latest/topics/loaders.html#topics-loaders-processors)及其结果附加到为该字段收集的数据。如果该字段已包含收集的数据，则添加新数据。

给定**field\_name**可以是**None**，在这种情况下，可以添加多个字段的值。处理后的值应该是一个字段，并将field\_name映射到值。

例子：



**replace\_value（*field\_name*，*value*，*\* processors*，*\*\* kwargs*）**

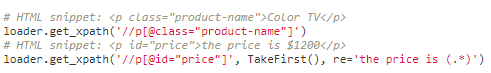
类似于[**add\_value()**](https://doc.scrapy.org/en/latest/topics/loaders.html#scrapy.loader.ItemLoader.add_value)使用新值替换收集的数据，而不是添加它。

**get\_xpath(xpath, \*processors, \*\*kwargs)**

与类似，[**ItemLoader.get\_value()**](https://doc.scrapy.org/en/latest/topics/loaders.html#scrapy.loader.ItemLoader.get_value)但接收XPath而不是值，该值用于从与此关联的选择器中提取unicode字符串列表[**ItemLoader**](https://doc.scrapy.org/en/latest/topics/loaders.html#scrapy.loader.ItemLoader)。

|  |  |
| --- | --- |
| **参数：** | * **xpath**（*str*） - 从中​​提取数据的XPath * **re**（*str 或编译正则表达式*） - 用于从所选XPath区域提取数据的正则表达式 |

例子：



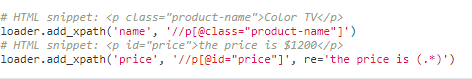
**add\_xpath（field\_name，xpath，\* processors，\*\* kwargs ）**

与类似，[**ItemLoader.add\_value()**](https://doc.scrapy.org/en/latest/topics/loaders.html#scrapy.loader.ItemLoader.add_value)但接收XPath而不是值，该值用于从与此关联的选择器中提取unicode字符串列表[**ItemLoader**](https://doc.scrapy.org/en/latest/topics/loaders.html#scrapy.loader.ItemLoader)。

见[**get\_xpath()**](https://doc.scrapy.org/en/latest/topics/loaders.html#scrapy.loader.ItemLoader.get_xpath)的**kwargs**。

|  |  |
| --- | --- |
| **参数：** | **xpath**（*str*） - 从中​​提取数据的XPath |

例子：



**replace\_xpath(field\_name, xpath, \*processors, \*\*kwargs)**

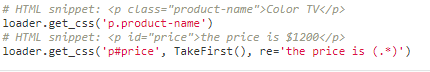
与[**add\_xpath()**](https://doc.scrapy.org/en/latest/topics/loaders.html#scrapy.loader.ItemLoader.add_xpath)收集的数据类似，但替代收集的数据。

**get\_css(css, \*processors, \*\*kwargs)**

与[**ItemLoader.get\_value()**](https://doc.scrapy.org/en/latest/topics/loaders.html#scrapy.loader.ItemLoader.get_value)CSS 类似，但接收CSS选择器而不是值，该值用于从与此关联的选择器中提取unicode字符串列表[**ItemLoader**](https://doc.scrapy.org/en/latest/topics/loaders.html#scrapy.loader.ItemLoader)。

|  |  |
| --- | --- |
| **参数：** | * **css**（*str*） - 从中​​提取数据的CSS选择器 * **re**（*str 或编译正则表达式*） - 用于从所选CSS区域提取数据的正则表达式 |

例子：



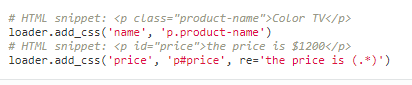
**add\_css（*field\_name*，*css*，*\* processors*，*\*\* kwargs*）**

与[**ItemLoader.add\_value()**](https://doc.scrapy.org/en/latest/topics/loaders.html#scrapy.loader.ItemLoader.add_value)CSS 类似，但接收CSS选择器而不是值，该值用于从与此关联的选择器中提取unicode字符串列表[**ItemLoader**](https://doc.scrapy.org/en/latest/topics/loaders.html#scrapy.loader.ItemLoader)。

见[**get\_css()**](https://doc.scrapy.org/en/latest/topics/loaders.html#scrapy.loader.ItemLoader.get_css)的**kwargs**。

|  |  |
| --- | --- |
| **参数：** | **css**（*str*） - 从中​​提取数据的CSS选择器 |

例子：



**replace\_css(field\_name, css, \*processors, \*\*kwargs)**

与[**add\_css()**](https://doc.scrapy.org/en/latest/topics/loaders.html#scrapy.loader.ItemLoader.add_css)收集的数据类似，但替代收集的数据。

**load\_item（）**

用目前收集的数据填充项目，然后返回。收集到的数据首先通过[输出处理器，](https://doc.scrapy.org/en/latest/topics/loaders.html#topics-loaders-processors)以获得分配给每个项目字段的最终值。

**nested\_xpath（*xpath*）**

用xpath选择器创建一个嵌套的加载器。提供的选择器是相对于与此相关的选择器应用的[**ItemLoader**](https://doc.scrapy.org/en/latest/topics/loaders.html#scrapy.loader.ItemLoader)。嵌套装载机股份**Item** 与母公司[**ItemLoader**](https://doc.scrapy.org/en/latest/topics/loaders.html#scrapy.loader.ItemLoader)如此呼吁[**add\_xpath()**](https://doc.scrapy.org/en/latest/topics/loaders.html#scrapy.loader.ItemLoader.add_xpath)，[**add\_value()**](https://doc.scrapy.org/en/latest/topics/loaders.html#scrapy.loader.ItemLoader.add_value)，[**replace\_value()**](https://doc.scrapy.org/en/latest/topics/loaders.html#scrapy.loader.ItemLoader.replace_value)等会像预期的那样。

**nested\_css（*css*）**

用css选择器创建一个嵌套的加载器。提供的选择器是相对于与此相关的选择器应用的[**ItemLoader**](https://doc.scrapy.org/en/latest/topics/loaders.html#scrapy.loader.ItemLoader)。嵌套装载机股份**Item** 与母公司[**ItemLoader**](https://doc.scrapy.org/en/latest/topics/loaders.html#scrapy.loader.ItemLoader)如此呼吁[**add\_xpath()**](https://doc.scrapy.org/en/latest/topics/loaders.html#scrapy.loader.ItemLoader.add_xpath)，[**add\_value()**](https://doc.scrapy.org/en/latest/topics/loaders.html#scrapy.loader.ItemLoader.add_value)，[**replace\_value()**](https://doc.scrapy.org/en/latest/topics/loaders.html#scrapy.loader.ItemLoader.replace_value)等会像预期的那样。

**get\_collected\_values（*field\_name*）**

返回给定字段的收集值。

**get\_output\_value（*field\_name*）**

对于给定的字段，返回使用输出处理器解析的收集值。此方法根本不填充或修改项目。

**get\_input\_processor（*field\_name*）**

返回给定字段的输入处理器。

**get\_output\_processor（*field\_name*）**

返回给定字段的输出处理器。

[**ItemLoader**](https://doc.scrapy.org/en/latest/topics/loaders.html#scrapy.loader.ItemLoader) 实例具有以下属性：

**item**

该[**Item**](https://doc.scrapy.org/en/latest/topics/items.html#scrapy.item.Item)对象通过这个项目装载机正在解析。

**context**

此项目加载器的当前活动[上下文](https://doc.scrapy.org/en/latest/topics/loaders.html#topics-loaders-context)。

**default\_item\_class**

Item类（或工厂），用于在构造函数中未给出时实例化项目。

**default\_input\_processor**

默认输入处理器用于那些没有指定一个的字段。

**default\_output\_processor**

默认输出处理器用于那些没有指定的字段。

**default\_selector\_class**

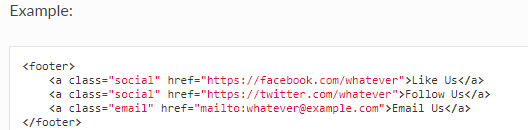
使用的类构造[**selector**](https://doc.scrapy.org/en/latest/topics/loaders.html#scrapy.loader.ItemLoader.selector)的此 [**ItemLoader**](https://doc.scrapy.org/en/latest/topics/loaders.html#scrapy.loader.ItemLoader)，如果只响应在构造函数中给出。如果在构造函数中给出选择器，则忽略该属性。该属性有时在子类中被覆盖。

**selector**

[**Selector**](https://doc.scrapy.org/en/latest/topics/selectors.html#scrapy.selector.Selector)从中提取数据的对象。它可以是构造函数中给出的选择器，也可以是使用构造函数中给出的响应创建的选择器 [**default\_selector\_class**](https://doc.scrapy.org/en/latest/topics/loaders.html#scrapy.loader.ItemLoader.default_selector_class)。该属性意味着是只读的。

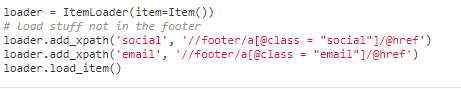
**嵌套装载机（Nested Loaders）**

从文档的子部分解析相关值时，创建嵌套的加载程序可能很有用。想象一下，您正在从页面的页脚中提取详细信息，如下所示：



没有嵌套的加载器，你需要为你想要提取的每个值指定完整的xpath（或css）。

例：



相反，您可以使用页脚选择器创建一个嵌套的加载器，并添加相对于页脚的值。功能相同，但您可以避免重复页脚选择器。

例：



您可以任意嵌套装载器，并且可以使用xpath或css选择器。作为一般指导原则，当您的代码变得更简单时，请使用嵌套加载器，但不要过度嵌套，否则解析器会变得难以阅读。

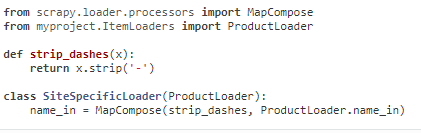
## 重用和扩展项目加载器

随着项目越来越大并且获得越来越多的蜘蛛，维护成为一个基本问题，尤其是当你必须处理每个蜘蛛的许多不同解析规则时，有很多例外情况，但也想重用常用处理器。

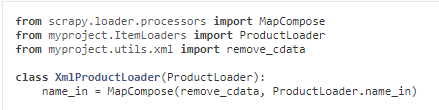
项目加载器旨在减轻解析规则的维护负担，同时又不失灵活性，同时还提供了扩展和覆盖它们的便利机制。出于这个原因，项目加载器支持传统的Python类继承来处理特定的蜘蛛（或蜘蛛组）的差异。

例如，假设某个特定网站用三条短横线（例如）包围它们的产品名称，并且您不希望最终产品名称中出现这些短划线。---Plasma TV---

您可以通过重新使用和扩展默认Product Item Loader（ProductLoader）来删除这些破折号：



另一种扩展项目加载器的情况会非常有用，那就是当你有多种源格式时，例如XML和HTML。在XML版本中，您可能希望删除CDATA事件。以下是如何执行此操作的示例：



这就是通常扩展输入处理器的方法。

至于输出处理器，在字段元数据中声明它们更为常见，因为它们通常仅取决于字段，而不取决于每个特定的站点解析规则（如输入处理器所做的那样）。另请参阅： [声明输入和输出处理器](https://doc.scrapy.org/en/latest/topics/loaders.html#topics-loaders-processors-declaring)。

还有很多其他可能的方式来扩展，继承和覆盖您的项目加载器，而不同的项目加载器层次结构可能更适合不同的项目。Scrapy只提供机制; 它不会对您的Loaders集合实施任何特定的组织 - 这取决于您和您的项目需求。

## 可用的内置处理器

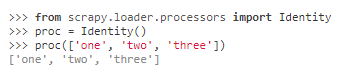
尽管您可以使用任何可调用的函数作为输入和输出处理器，但Scrapy提供了一些常用的处理器，下面将对其进行介绍。其中的一些[**MapCompose**](https://doc.scrapy.org/en/latest/topics/loaders.html#scrapy.loader.processors.MapCompose)（如通常用作输入处理器）组成按顺序执行的几个函数的输出，以产生最终解析值。

以下是所有内置处理器的列表：

**Class scrapy.loader.processors.Identity**

最简单的处理器，它什么都不做。它返回原始值不变。它不接收任何构造函数参数，也不接受Loader上下文。

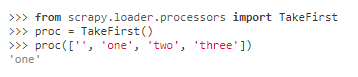
例：



**Class scrapy.loader.processors.TakeFirst**

从接收到的值中返回第一个非空/非空值，因此它通常用作单值字段的输出处理器。它不接收任何构造函数参数，也不接受Loader上下文。

例：

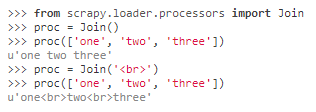


**Class scrapy.loader.processors.Join(separator=u' ')**

返回与构造函数中给定的分隔符连接的值，默认值为。它不接受Loader上下文。**u' '**

使用默认分隔符时，此处理器等同于以下功能： **u' '.join**

例子：

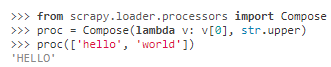


**Class scrapy.loader.processors.Compose(\*functions, \*\*default\_loader\_context)**

由给定函数的组成构成的处理器。这意味着该处理器的每个输入值都被传递给第一个函数，并且该函数的结果被传递给第二个函数，依此类推，直到最后一个函数返回该处理器的输出值。

默认情况下，停止处理**None**值。这个行为可以通过传递关键字参数来改变**stop\_on\_none=False**。

例：



每个功能都可以选择接收**loader\_context**参数。对于那些做的，这个处理器将通过该参数传递当前活动的[Loader上下文](https://doc.scrapy.org/en/latest/topics/loaders.html#topics-loaders-context)。

在构造函数中传递的关键字参数用作传递给每个函数调用的默认Loader上下文值。但是，传递给函数的最终Loader上下文值将通过该**ItemLoader.context()** 属性访问当前活动的Loader上下文。

**Class scrapy.loader.processors.MapCompose(\*functions, \*\*default\_loader\_context)**

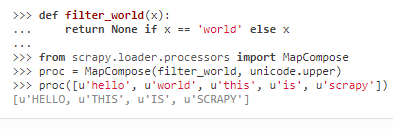
与处理器类似，由给定功能的组成构成的[**Compose**](https://doc.scrapy.org/en/latest/topics/loaders.html#scrapy.loader.processors.Compose)处理器。这个处理器的不同之处在于内部结果在各个函数之间传递的方式，如下所示：

这个处理器的输入值被迭代，并且第一个函数被应用于每个元素。这些函数调用的结果（每个元素一个）被连接起来构造一个新的迭代器，然后用于应用​​第二个函数，等等，直到最后一个函数被应用到所收集的值列表的每个值为止远。最后一个函数的输出值被连接在一起以产生该处理器的输出。

每个特定的函数都可以返回一个值或一个值列表，这些值将与应用于其他输入值的同一个函数返回的值列表一起展开。函数也可以返回，**None**在这种情况下，该函数的输出将被忽略，以便通过链进一步处理。

该处理器提供了一种便捷的方式来编写仅适用于单个值（而不是可迭代）的函数。由于这个原因， [**MapCompose**](https://doc.scrapy.org/en/latest/topics/loaders.html#scrapy.loader.processors.MapCompose)处理器通常用作输入处理器，因为通常使用[选择器](https://doc.scrapy.org/en/latest/topics/selectors.html#topics-selectors)的[**extract()**](https://doc.scrapy.org/en/latest/topics/selectors.html#scrapy.selector.Selector.extract)方法提取数据 ，[选择器](https://doc.scrapy.org/en/latest/topics/selectors.html#topics-selectors)返回一个unicode字符串列表。

下面的例子应该说明它的工作原理：

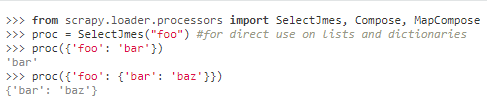


与Compose处理器一样，函数可以接收Loader上下文，并将构造函数关键字参数用作默认上下文值。查看 [**Compose**](https://doc.scrapy.org/en/latest/topics/loaders.html#scrapy.loader.processors.Compose)处理器了解更多信息。

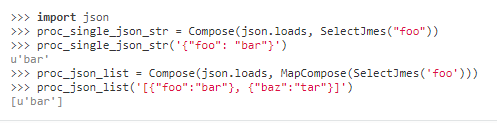
**Class scrapy.loader.processors.SelectJmes(json\_path)**

使用提供给构造函数的json路径查询该值并返回输出。需要运行jmespath（<https://github.com/jmespath/jmespath.py>）。该处理器一次只有一个输入。

例：



与Json合作：



# Scrapy shell

Scrapy shell是一个交互式shell，您可以非常快速地尝试并调试您的scraping代码，而无需运行蜘蛛。它旨在用于测试数据提取代码，但实际上它可以用于测试任何类型的代码，因为它也是一个常规的Python shell。

该shell用于测试XPath或CSS表达式，并查看它们的工作方式以及它们从您尝试抓取的网页中提取的数据。它可以让你在写蜘蛛的时候交互地测试你的表情，而不必运行蜘蛛来测试每一个变化。

一旦熟悉Scrapy shell，您会发现它是开发和调试您的蜘蛛的宝贵工具。

## 配置shell

如果你安装了[IPython](https://ipython.org/)，Scrapy shell将会使用它（而不是标准的Python控制台）。该[IPython的](https://ipython.org/)控制台功能更强大，并提供智能自动完成和彩色输出，等等。

我们强烈建议您安装[IPython](https://ipython.org/)，特别是如果您在Unix系统上工作（[IPython](https://ipython.org/)擅长）。有关 更多信息，请参阅[IPython安装指南](https://ipython.org/install.html)。

Scrapy也支持[bpython](https://www.bpython-interpreter.org/)，并会尝试在[IPython](https://ipython.org/) 不可用的情况下使用它。

通过scrapy的设置，您可以配置为使用中的任何一个 ipython，bpython或标准的python外壳，安装不管是哪个。这是通过设置SCRAPY\_PYTHON\_SHELL环境变量来完成的; 或者通过在[scrapy.cfg中](https://doc.scrapy.org/en/latest/topics/commands.html#topics-config-settings)定义它：



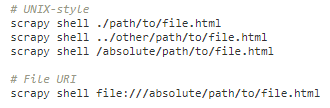
## 启动外壳

要启动Scrapy shell，您可以使用如下[**shell**](https://doc.scrapy.org/en/latest/topics/commands.html#std:command-shell)命令：



那里<url>是你想要刮的URL。

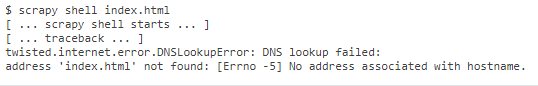
[**shell**](https://doc.scrapy.org/en/latest/topics/commands.html#std:command-shell)也适用于本地文件。如果你想玩一个网页的本地副本，这可以很方便。[**shell**](https://doc.scrapy.org/en/latest/topics/commands.html#std:command-shell)了解本地文件的以下语法：



**注意**

在使用相对文件路径时，应明确并在./（或../相关时）前加上它们。 将不会像人们所期望的那样工作（这是通过设计而非错误）。scrapy shell index.html

由于[**shell**](https://doc.scrapy.org/en/latest/topics/commands.html#std:command-shell)支持文件URI上的HTTP URL，并且index.html在语法上类似example.com， [**shell**](https://doc.scrapy.org/en/latest/topics/commands.html#std:command-shell)因此将index.html视为域名并触发DNS查找错误：



[**shell**](https://doc.scrapy.org/en/latest/topics/commands.html#std:command-shell)如果index.html 当前目录中存在调用的文件，则不会事先进行测试。再次，请明确。

## 使用shell

Scrapy shell只是一个普通的Python控制台（如果有的话，也可以是[IPython](https://ipython.org/)控制台），它提供了一些额外的便捷功能。

### 可用的快捷方式

* shelp() - 用可用对象和快捷方式的列表打印帮助
* fetch(url[, redirect=True]) - 从给定的URL获取新的响应并相应地更新所有相关的对象。您可以选择要求HTTP 3xx重定向，以免传递redirect=False
* fetch(request) - 从给定的请求中获取新的响应，并相应地更新所有相关的对象。
* view(response) - 在本地网络浏览器中打开给定的响应，以供检查。这会在响应主体中添加一个[<base>标记](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/base)，以便正确显示外部链接（如图像和样式表）。但请注意，这将在您的计算机中创建一个临时文件，该文件不会被自动删除。

### 可用的Scrapy对象

Scrapy外壳会自动从下载的页面创建一些便利的对象，如[**Response**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.Response)对象和 [**Selector**](https://doc.scrapy.org/en/latest/topics/selectors.html#scrapy.selector.Selector)对象（用于HTML和XML内容）。

这些对象是：

* crawler- 当前[**Crawler**](https://doc.scrapy.org/en/latest/topics/api.html#scrapy.crawler.Crawler)对象。
* spider- 已知处理URL的Spider，或者[**Spider**](https://doc.scrapy.org/en/latest/topics/spiders.html#scrapy.spiders.Spider)当前URL中没有发现蜘蛛的 对象
* request- [**Request**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.Request)最后一次抓取页面的对象。您可以使用[**replace()**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.Request.replace) 或通过使用fetch 快捷方式获取新请求（不离开shell）来修改此请求。
* response- [**Response**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.Response)包含上次获取的页面的对象
* settings- 当前的[Scrapy设置](https://doc.scrapy.org/en/latest/topics/settings.html#topics-settings)

## shell会话的例子

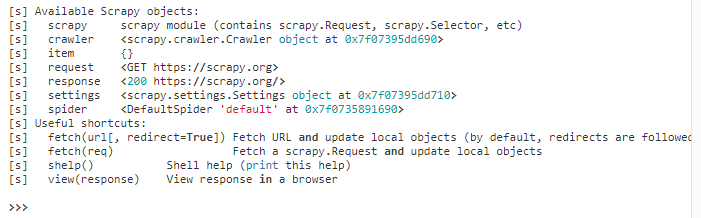
下面是一个典型的shell会话示例，我们首先抓取[https://scrapy.org](https://scrapy.org/)页面，然后继续刮取[https://reddit.com](https://reddit.com/) 页面。最后，我们将（Reddit）请求方法修改为POST并重新获取错误。我们通过在Windows中键入Ctrl-D（在Unix系统中）或Ctrl-Z来结束会话。

请记住，这里提取的数据在尝试时可能不尽相同，因为这些页面不是静态的，并且在测试时可能会发生变化。这个例子的唯一目的是让你熟悉Scrapy shell的工作原理。

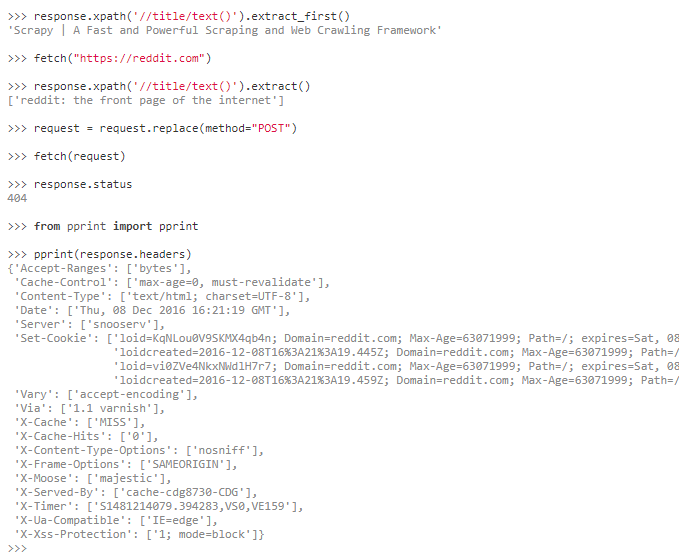
首先，我们启动shell：



然后，shell获取URL（使用Scrapy下载器）并打印可用对象列表和有用的快捷方式（您会注意到这些行都以[s]前缀开头）：



之后，我们可以开始玩对象：



## 从蜘蛛中调用外壳来检查响应

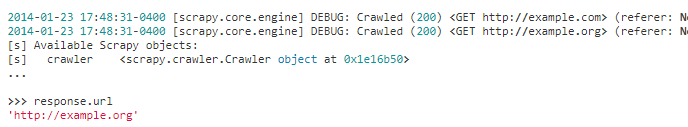
有时候你想检查一下蜘蛛某一点正在处理的反应，如果只是为了检查你期望的反应是否到达那里。

这可以通过使用该scrapy.shell.inspect\_response功能来实现。

以下是您如何从您的蜘蛛中调用它的示例：



当你运行蜘蛛时，你会得到类似于这样的东西：



然后，你可以检查提取代码是否工作：



不，它没有。因此，您可以在Web浏览器中打开响应，看看它是否是您期望的响应：



最后，您按Ctrl-D（或Windows中的Ctrl-Z）以退出shell并恢复爬网：



请注意，fetch由于Scrapy引擎被shell阻塞，因此您无法使用此快捷方式。但是，在离开壳后，蜘蛛将继续爬行，如上所示。

# Item Pipeline

物品被蜘蛛抓取后，它被发送到物品管线，该物品管线通过顺序执行的多个组件处理它。

每个项目管道组件（有时简称为“项目管道”）是一个实现简单方法的Python类。他们收到一个物品并对其执行操作，并决定该物品是否应该通过管道继续运行，或者被丢弃并不再处理。

项目管道的典型用途是：

* 清理HTML数据
* 验证刮取的数据（检查项目是否包含某些字段）
* 检查重复项（并放下它们）
* 将刮取的物品存储在数据库中

**编写自己的物品管道**

每个项目管道组件都是一个Python类，它必须实现以下方法：

**process\_item(self, item, spider)**

每个项目管道组件都会调用此方法。[**process\_item()**](https://doc.scrapy.org/en/latest/topics/item-pipeline.html#process_item) 必须：用数据返回一个字典，返回一个[**Item**](https://doc.scrapy.org/en/latest/topics/items.html#scrapy.item.Item) （或任何后代类）对象，返回[Twisted Deferred](https://twistedmatrix.com/documents/current/core/howto/defer.html)或者引发 [**DropItem**](https://doc.scrapy.org/en/latest/topics/exceptions.html#scrapy.exceptions.DropItem)异常。丢弃的物品不再被进一步的管道组件处理。

|  |  |
| --- | --- |
| **参数：** | * **项目**（[**Item**](https://doc.scrapy.org/en/latest/topics/items.html#scrapy.item.Item)对象或字典） - 项目被刮掉 * **蜘蛛**（[**Spider**](https://doc.scrapy.org/en/latest/topics/spiders.html#scrapy.spiders.Spider)物体） - 刮掉物品的蜘蛛 |

**open\_spider(self, spider)**

这个方法在蜘蛛打开时被调用。

|  |  |
| --- | --- |
| **参数：** | **蜘蛛**（[**Spider**](https://doc.scrapy.org/en/latest/topics/spiders.html#scrapy.spiders.Spider)物体） - 被打开的蜘蛛 |

**close\_spider(self, spider)**

这个方法在蜘蛛关闭时被调用。

|  |  |
| --- | --- |
| **参数：** | **蜘蛛**（[**Spider**](https://doc.scrapy.org/en/latest/topics/spiders.html#scrapy.spiders.Spider)物体） - 被关闭的蜘蛛 |

**from\_crawler(cls, crawler)**

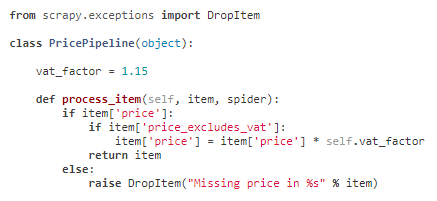
如果存在，就调用这个classmethod来从a创建一个管道实例[**Crawler**](https://doc.scrapy.org/en/latest/topics/api.html#scrapy.crawler.Crawler)。它必须返回一个新的管道实例。抓取工具对象提供对所有Scrapy核心组件的访问，如设置和信号; 它是管道访问它们并将其功能挂接到Scrapy的一种方式。

|  |  |
| --- | --- |
| **参数：** | **搜寻器**（[**Crawler**](https://doc.scrapy.org/en/latest/topics/api.html#scrapy.crawler.Crawler)对象） - 使用此管道的搜寻器 |

## 项目管道示例

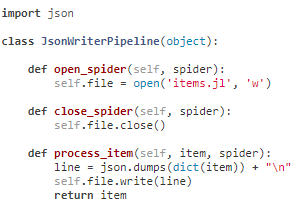
### 价格验证和丢弃物品没有价格

让我们来看看下面的假设管道，它调整 price那些不包含增值税（price\_excludes\_vat属性）的项目的属性，并删除那些不包含价格的项目：



### 将项目写入JSON文件

以下管道将所有抓取的项目（来自所有蜘蛛）存储到一个items.jl文件中，每行包含一个以JSON格式序列化的项目：



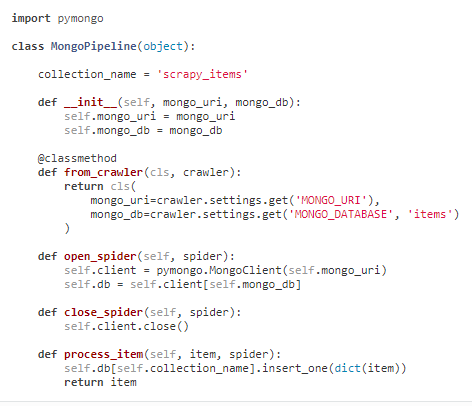
**注意**

JsonWriterPipeline的目的只是介绍如何编写项目管道。如果你真的想把所有被抓取的项目存储到一个JSON文件中，你应该使用[Feed输出](https://doc.scrapy.org/en/latest/topics/feed-exports.html#topics-feed-exports)。

### 将项目写入MongoDB

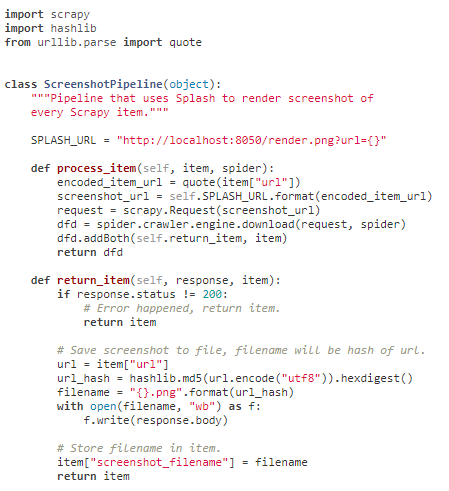
在这个例子中，我们将使用[pymongo](https://api.mongodb.org/python/current/)将项目写入[MongoDB](https://www.mongodb.org/)。MongoDB地址和数据库名称在Scrapy设置中指定; MongoDB集合以item类命名。

这个例子的要点是展示如何使用[**from\_crawler()**](https://doc.scrapy.org/en/latest/topics/item-pipeline.html#from_crawler) 方法以及如何正确清理资源。



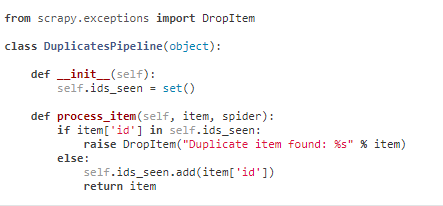
### 以项目的截图

这个例子演示了如何从方法返回[Deferred](https://twistedmatrix.com/documents/current/core/howto/defer.html)[**process\_item()**](https://doc.scrapy.org/en/latest/topics/item-pipeline.html#process_item)。它使用[Splash](https://splash.readthedocs.io/en/stable/)渲染项目url的屏幕截图。管道请求本地运行的[Splash](https://splash.readthedocs.io/en/stable/)实例。下载请求并延迟回调后，它将项目保存到一个文件并将文件名添加到项目。



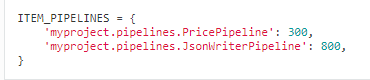
### 重复过滤器

过滤器查找重复的项目，并删除已处理的项目。假设我们的物品具有唯一的ID，但我们的蜘蛛会使用相同的ID返回多个物品：



## 激活项目管道组件

要激活Item Pipeline组件，您必须将其类添加到 [**ITEM\_PIPELINES**](https://doc.scrapy.org/en/latest/topics/settings.html#std:setting-ITEM_PIPELINES)设置中，如下例所示：



您在此设置中分配给类的整数值决定了它们的运行顺序：项目从较低值到较高值的类别。通常在0-1000范围内定义这些数字。

[下一个](https://doc.scrapy.org/en/latest/topics/feed-exports.html)[以前](https://doc.scrapy.org/en/latest/topics/shell.html)

# Feed exports

0.10版本中的新功能。

在实现刮板时最常用的功能之一是能够正确存储刮取的数据，而且通常这意味着生成一个带有刮取数据的“导出文件”（通常称为“导出Feed”），以供其他系统使用。

Scrapy通过Feed Export提供了这种功能，它允许您使用多个序列化格式和存储后端生成包含已删除项目的提要。

## 序列化格式

为了序列化抓取的数据，Feed输出使用[Item输出](https://doc.scrapy.org/en/latest/topics/exporters.html#topics-exporters)。这些格式支持开箱即用：

* [JSON](https://doc.scrapy.org/en/latest/topics/feed-exports.html#topics-feed-format-json)
* [JSON行](https://doc.scrapy.org/en/latest/topics/feed-exports.html#topics-feed-format-jsonlines)
* [CSV](https://doc.scrapy.org/en/latest/topics/feed-exports.html#topics-feed-format-csv)
* [XML](https://doc.scrapy.org/en/latest/topics/feed-exports.html#topics-feed-format-xml)

但是你也可以通过[**FEED\_EXPORTERS**](https://doc.scrapy.org/en/latest/topics/feed-exports.html#std:setting-FEED_EXPORTERS)设置来扩展支持的格式 。

### JSON

* [**FEED\_FORMAT**](https://doc.scrapy.org/en/latest/topics/feed-exports.html#std:setting-FEED_FORMAT)： json
* 出口商使用： [**JsonItemExporter**](https://doc.scrapy.org/en/latest/topics/exporters.html#scrapy.exporters.JsonItemExporter)
* 如果您在使用大型供稿时使用JSON，请参阅[此警告](https://doc.scrapy.org/en/latest/topics/exporters.html#json-with-large-data)。

### JSON行

* [**FEED\_FORMAT**](https://doc.scrapy.org/en/latest/topics/feed-exports.html#std:setting-FEED_FORMAT)： jsonlines
* 出口商使用： [**JsonLinesItemExporter**](https://doc.scrapy.org/en/latest/topics/exporters.html#scrapy.exporters.JsonLinesItemExporter)

### CSV

* [**FEED\_FORMAT**](https://doc.scrapy.org/en/latest/topics/feed-exports.html#std:setting-FEED_FORMAT)： csv
* 出口商使用： [**CsvItemExporter**](https://doc.scrapy.org/en/latest/topics/exporters.html#scrapy.exporters.CsvItemExporter)
* 指定要导出的列及其顺序使用 [**FEED\_EXPORT\_FIELDS**](https://doc.scrapy.org/en/latest/topics/feed-exports.html#std:setting-FEED_EXPORT_FIELDS)。其他Feed出口商也可以使用此选项，但它对CSV很重要，因为与许多其他导出格式不同，CSV使用固定标题。

### XML

* [**FEED\_FORMAT**](https://doc.scrapy.org/en/latest/topics/feed-exports.html#std:setting-FEED_FORMAT)： xml
* 出口商使用： [**XmlItemExporter**](https://doc.scrapy.org/en/latest/topics/exporters.html#scrapy.exporters.XmlItemExporter)

### Pickle

* [**FEED\_FORMAT**](https://doc.scrapy.org/en/latest/topics/feed-exports.html#std:setting-FEED_FORMAT)： pickle
* 出口商使用： [**PickleItemExporter**](https://doc.scrapy.org/en/latest/topics/exporters.html#scrapy.exporters.PickleItemExporter)

### Marshal

* [**FEED\_FORMAT**](https://doc.scrapy.org/en/latest/topics/feed-exports.html#std:setting-FEED_FORMAT)： marshal
* 出口商使用： **MarshalItemExporter**

## 储量（Storages）

在使用Feed输出时，您可以使用[URI](https://en.wikipedia.org/wiki/Uniform_Resource_Identifier)定义存储Feed的位置 （通过[**FEED\_URI**](https://doc.scrapy.org/en/latest/topics/feed-exports.html#std:setting-FEED_URI)设置）。Feed输出支持由URI方案定义的多个存储后端类型。

后端支持的存储后端是：

* [本地文件系统](https://doc.scrapy.org/en/latest/topics/feed-exports.html#topics-feed-storage-fs)
* [FTP](https://doc.scrapy.org/en/latest/topics/feed-exports.html#topics-feed-storage-ftp)
* [S3](https://doc.scrapy.org/en/latest/topics/feed-exports.html#topics-feed-storage-s3)（需要[botocore](https://github.com/boto/botocore)或[boto](https://github.com/boto/boto)）
* [标准输出](https://doc.scrapy.org/en/latest/topics/feed-exports.html#topics-feed-storage-stdout)

如果所需的外部库不可用，某些存储后端可能不可用。例如，S3后端仅在安装[botocore](https://github.com/boto/botocore) 或[boto](https://github.com/boto/boto)库时才可用（Scrapy 仅在Python 2上支持[boto](https://github.com/boto/boto)）。

## 存储URI参数

存储URI还可以包含在创建订阅源时被替换的参数。这些参数是：

* %(time)s - 当饲料正在被创建时，被时间戳取代
* %(name)s - 被蜘蛛的名字取代

任何其他命名参数将被相同名称的spider属性替换。例如， 在创建订阅源时将被属性%(site\_id)s替换spider.site\_id。

以下是一些例子来说明：

* 每个蜘蛛使用一个目录存储在FTP中：
  + ftp://user:password@ftp.example.com/scraping/feeds/%(name)s/%(time)s.json
* 每个蜘蛛使用一个目录在S3中存储：
  + s3://mybucket/scraping/feeds/%(name)s/%(time)s.json

## 存储后端

### 本地文件系统

这些提要存储在本地文件系统中。

* URI方案： file
* 示例URI： file:///tmp/export.csv
* 所需的外部库：无

请注意，对于本地文件系统存储（仅限于），如果指定绝对路径，则可以省略该方案/tmp/export.csv。但这只适用于Unix系统。

### FTP

这些提要存储在FTP服务器中。

* URI方案： ftp
* 示例URI： ftp://user:pass@ftp.example.com/path/to/export.csv
* 所需的外部库：无

### S3

供稿存储在[Amazon S3上](https://aws.amazon.com/s3/)。

* URI方案： s3
* 示例URI：
  + s3://mybucket/path/to/export.csv
  + s3://aws\_key:aws\_secret@mybucket/path/to/export.csv
* 所需的外部库：[botocore](https://github.com/boto/botocore)或[boto](https://github.com/boto/boto)

AWS凭证可以作为URI中的用户/密码传递，也可以通过以下设置传递：

* [**AWS\_ACCESS\_KEY\_ID**](https://doc.scrapy.org/en/latest/topics/settings.html#std:setting-AWS_ACCESS_KEY_ID)
* [**AWS\_SECRET\_ACCESS\_KEY**](https://doc.scrapy.org/en/latest/topics/settings.html#std:setting-AWS_SECRET_ACCESS_KEY)

### 标准输出

Feed被写入Scrapy进程的标准输出。

* URI方案： stdout
* 示例URI： stdout:
* 所需的外部库：无

## 设置

这些是用于配置Feed输出的设置：

* [**FEED\_URI**](https://doc.scrapy.org/en/latest/topics/feed-exports.html#std:setting-FEED_URI) （强制）
* [**FEED\_FORMAT**](https://doc.scrapy.org/en/latest/topics/feed-exports.html#std:setting-FEED_FORMAT)
* [**FEED\_STORAGES**](https://doc.scrapy.org/en/latest/topics/feed-exports.html#std:setting-FEED_STORAGES)
* [**FEED\_EXPORTERS**](https://doc.scrapy.org/en/latest/topics/feed-exports.html#std:setting-FEED_EXPORTERS)
* [**FEED\_STORE\_EMPTY**](https://doc.scrapy.org/en/latest/topics/feed-exports.html#std:setting-FEED_STORE_EMPTY)
* [**FEED\_EXPORT\_ENCODING**](https://doc.scrapy.org/en/latest/topics/feed-exports.html#std:setting-FEED_EXPORT_ENCODING)
* [**FEED\_EXPORT\_FIELDS**](https://doc.scrapy.org/en/latest/topics/feed-exports.html#std:setting-FEED_EXPORT_FIELDS)
* [**FEED\_EXPORT\_INDENT**](https://doc.scrapy.org/en/latest/topics/feed-exports.html#std:setting-FEED_EXPORT_INDENT)

### FEED\_URI

默认： None

导出Feed的URI。请参阅支持的URI方案的[存储后端](https://doc.scrapy.org/en/latest/topics/feed-exports.html#topics-feed-storage-backends)。

此设置对于启用Feed输出是必需的。

### FEED\_FORMAT

要用于Feed的序列化格式。请参阅 [序列化格式](https://doc.scrapy.org/en/latest/topics/feed-exports.html#topics-feed-format)以获取可能的值。

### FEED\_EXPORT\_ENCODING

默认： None

要用于Feed的编码。

如果未设置或设置为None（默认），则对于除JSON输出（\uXXXX因历史原因使用安全数字编码（序列））以外的所有内容都使用UTF-8 。

utf-8如果您还想为JSON 使用UTF-8，请使用它。

### FEED\_EXPORT\_FIELDS

默认： None

要导出的字段列表，可选。例如：。FEED\_EXPORT\_FIELDS = ["foo", "bar", "baz"]

使用FEED\_EXPORT\_FIELDS选项来定义要导出的字段及其顺序。

当FEED\_EXPORT\_FIELDS为空或无（默认值）时，Scrapy使用在字典或[**Item**](https://doc.scrapy.org/en/latest/topics/items.html#scrapy.item.Item)蜘蛛正在产生的子类中定义的字段。

如果导出程序需要一组固定的字段（这是[CSV](https://doc.scrapy.org/en/latest/topics/feed-exports.html#topics-feed-format-csv)导出格式的情况 ），并且FEED\_EXPORT\_FIELDS为空或无，则Scrapy会尝试从导出的​​数据中推断字段名称 - 目前它使用第一项中的字段名称。

### FEED\_EXPORT\_INDENT

默认： 0

用于缩进每个级别的输出的空间量。如果FEED\_EXPORT\_INDENT 是非负整数，则数组元素和对象成员将与该缩进级别相匹配。0（缺省值）或负值的缩进级别会将每个项目放在一个新行中。None选择最紧凑的表示。

目前仅通过[**JsonItemExporter**](https://doc.scrapy.org/en/latest/topics/exporters.html#scrapy.exporters.JsonItemExporter) 和实现[**XmlItemExporter**](https://doc.scrapy.org/en/latest/topics/exporters.html#scrapy.exporters.XmlItemExporter)，即当您输出到.json或.xml。

### FEED\_STORE\_EMPTY

默认： False

是否导出空的Feed（即没有项目的Feed）。

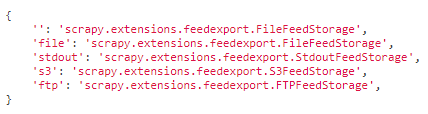
### FEED\_STORAGES

默认： {}

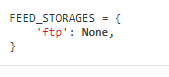
包含您的项目支持的额外的后端存储后端的字典。密钥是URI方案，值是存储类的路径。

### FEED\_STORAGES\_BASE

默认：



包含Scrapy支持的内置Feed存储后端的字典。您可以通过分配其中None的URI方案 来禁用这些后端中的任何一个[**FEED\_STORAGES**](https://doc.scrapy.org/en/latest/topics/feed-exports.html#std:setting-FEED_STORAGES)。例如，要禁用内置FTP存储后端（无需替换），请将其放置在settings.py：



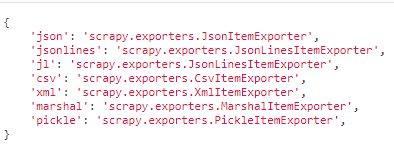
### FEED\_EXPORTERS

默认： {}

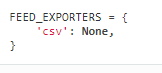
包含您的项目支持的其他出口商的字典。这些键是序列化格式，值是[Item导出器](https://doc.scrapy.org/en/latest/topics/exporters.html#topics-exporters)类的路径。

### FEED\_EXPORTERS\_BASE

默认：



包含Scrapy支持的内置Feed出口商的字典。您可以通过指定其中None的序列化格式来禁用这些导出器中的任何一个[**FEED\_EXPORTERS**](https://doc.scrapy.org/en/latest/topics/feed-exports.html#std:setting-FEED_EXPORTERS)。例如，要禁用内置CSV导出器（无需替换），请将其放置在settings.py：



# Requests and Responses

Scrapy使用[**Request**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.Request)和[**Response**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.Response)对象来爬取网站。

通常情况下，[**Request**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.Request)对象在蜘蛛中生成并传递到系统，直到它们到达下载器，该下载器执行请求并返回一个[**Response**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.Response)对象，该对象返回发出请求的蜘蛛。

两者[**Request**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.Request)和[**Response**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.Response)类都具有添加基类中不需要的功能的子类。这些在[请求子类](https://doc.scrapy.org/en/latest/topics/request-response.html#topics-request-response-ref-request-subclasses)和 [响应子类中](https://doc.scrapy.org/en/latest/topics/request-response.html#topics-request-response-ref-response-subclasses)进行了描述。

**请求对象**

***Class* scrapy.http.Request（*url*[，*callback*，*method ='GET'*，*headers*，*body*，*cookies*，*meta*，*encoding ='utf-8'*，*priority = 0*，*dont\_filter = False*，*errback*，*flags*]）**

一个[**Request**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.Request)对象表示一个HTTP请求，它通常在Spider中生成并由Downloader执行，从而生成一个[**Response**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.Response)。

|  |  |
| --- | --- |
| **参数：** | * **url** (string) - 此请求的URL * **callback** (callable)（*可调用的*） - 将这个请求的响应（一旦下载完成）作为第一个参数调用的函数。有关更多信息，请参阅下面将[附加数据传递给回调函数](https://doc.scrapy.org/en/latest/topics/request-response.html#topics-request-response-ref-request-callback-arguments)。如果请求没有指定回调，则将使用蜘蛛的[**parse()**](https://doc.scrapy.org/en/latest/topics/spiders.html#scrapy.spiders.Spider.parse)方法。请注意，如果在处理期间引发异常，则会调用errback。 * **method** (string) - 此请求的HTTP方法。默认为**'GET'**。 * **meta** (dict)  - [**Request.meta**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.Request.meta)属性的初始值。如果给出，在此参数中传递的字典将被浅拷贝。 * **body** (str or unicode) - 请求正文。如果a **unicode**被传递，那么它被编码为 **str**使用传递的*编码*（默认为**utf-8**）。如果 **body**没有给出，则存储空字符串。无论此参数的类型如何，存储的最终值都将是**str**（从不 **unicode**或**None**）。 * **headers** (dict)  - 这个请求的标题。字典值可以是字符串（对于单值标题）或列表（对于多值标题）。如果 **None**作为值传递，HTTP标头根本不会被发送。 * **cookies** (dict or [list](https://doc.scrapy.org/en/latest/topics/api.html#scrapy.loader.SpiderLoader.list))  -   请求cookies。这些可以以两种形式发送。   * 1. 使用字典： |



后一种形式允许自定义**domain**和**path** Cookie的属性。这仅在cookie被保存用于以后的请求时才有用。

当某个站点返回cookie（作为响应）时，这些cookie将存储在该域的cookie中，并将在未来的请求中再次发送。这是任何常规Web浏览器的典型行为。但是，如果出于某种原因想要避免与现有Cookie合并，可以通过将**dont\_merge\_cookies**关键字设置为True 来指示Scrapy执行此操作 [**Request.meta**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.Request.meta)。

不合并Cookie的请求示例：



|  |
| --- |
| * 欲了解更多信息，请参阅[CookiesMiddleware](https://doc.scrapy.org/en/latest/topics/downloader-middleware.html#cookies-mw)。 * **encoding** (string) - 此请求的编码（默认为**'utf-8'**）。此编码将用于对URL进行百分比编码并将其转换为**str**（如果给定为**unicode**）。 * **priority**（int） - 此请求的优先级（默认为**0**）。调度程序使用优先级来定义用于处理请求的顺序。具有较高优先级值的请求将在较早时间执行。为了表示相对低的优先级，允许负值。 * **dont\_filter**（boolean） - 表示这个请求不应该被调度器过滤。当您想多次执行相同的请求时使用此选项，以忽略重复过滤器。小心使用它，否则你将进入爬行循环。默认为**False**。 * **errback**（callable） - 如果在处理请求时引发任何异常，将会调用该函数。这包括404 HTTP错误等失败的页面。它收到[Twisted Failure](https://twistedmatrix.com/documents/current/api/twisted.python.failure.Failure.html)实例作为第一个参数。有关更多信息，请参阅下面的[请求处理](https://doc.scrapy.org/en/latest/topics/request-response.html#topics-request-response-ref-errbacks)中的[使用errbacks来捕获异常](https://doc.scrapy.org/en/latest/topics/request-response.html#topics-request-response-ref-errbacks)。 * **flags** ([list](https://doc.scrapy.org/en/latest/topics/api.html#scrapy.loader.SpiderLoader.list))  - 发送到请求的标志可用于日志记录或类似目的。 |

**url**

包含此请求的URL的字符串。请记住，此属性包含转义的URL，因此它可能与构造函数中传递的URL不同。

该属性是只读的。更改请求使用的URL [**replace()**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.Request.replace)。

**method**

表示请求中的HTTP方法的字符串。这保证是大写的。例如：**"GET"**，**"POST"**，**"PUT"**，等

**headers**

一个包含请求头文件的类似字典的对象。

**body**

包含请求主体的str。

该属性是只读的。改变请求使用的主体 [**replace()**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.Request.replace)。

**meta**

包含此请求的任意元数据的字典。这个词典对于新的请求是空的，并且通常由不同的Scrapy组件（扩展，中间件等）填充。因此，此字典中包含的数据取决于您启用的扩展。

有关由Scrapy识别的特殊元键列表，请参阅[Request.meta特殊键](https://doc.scrapy.org/en/latest/topics/request-response.html#topics-request-meta)。

当使用or 方法克隆请求时， 该字典被[复制](https://docs.python.org/2/library/copy.html)，并且也可以在属性中的蜘蛛中访问。**copy()replace()response.meta**

**copy（）**

返回一个新请求，它是此请求的副本。另请参阅：将 [附加数据传递给回调函数](https://doc.scrapy.org/en/latest/topics/request-response.html#topics-request-response-ref-request-callback-arguments)。

**replace（[url，method，headers，body，cookies，meta，encoding，dont\_filter，callback，errback ]）**

使用相同的成员返回一个Request对象，除了那些由指定的关键字参数赋予新值的成员。该属性[**Request.meta**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.Request.meta)被默认复制（除非在**meta**参数中给出一个新值）。另请参见将 [附加数据传递给回调函数](https://doc.scrapy.org/en/latest/topics/request-response.html#topics-request-response-ref-request-callback-arguments)。

### 将附加数据传递给回调函数

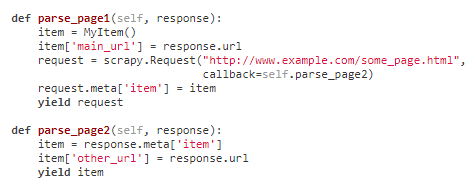
请求的回调函数将在下载该请求的响应时调用。回调函数将以下载的[**Response**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.Response)对象作为第一个参数被调用。

例：



在某些情况下，您可能有兴趣将参数传递给这些回调函数，以便稍后在第二个回调函数中接收参数。您可以使用该[**Request.meta**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.Request.meta)属性。

以下是如何使用此机制传递项目以填充不同页面的不同字段的示例：



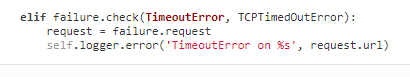
### 使用errbacks在请求处理中捕获异常

请求的错误是一个函数，当处理异常时会调用它。

它接收[Twisted Failure](https://twistedmatrix.com/documents/current/api/twisted.python.failure.Failure.html)实例作为第一个参数，可用于跟踪连接建立超时，DNS错误等。

以下是一个蜘蛛日志记录所有错误并在需要时捕获一些特定错误的示例：





## Request.meta特殊键

该[**Request.meta**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.Request.meta)属性可以包含任何任意数据，但Scrapy及其内置扩展可识别一些特殊键。

那些是：

* [**dont\_redirect**](https://doc.scrapy.org/en/latest/topics/downloader-middleware.html#std:reqmeta-dont_redirect)
* [**dont\_retry**](https://doc.scrapy.org/en/latest/topics/downloader-middleware.html#std:reqmeta-dont_retry)
* [**handle\_httpstatus\_list**](https://doc.scrapy.org/en/latest/topics/spider-middleware.html#std:reqmeta-handle_httpstatus_list)
* [**handle\_httpstatus\_all**](https://doc.scrapy.org/en/latest/topics/spider-middleware.html#std:reqmeta-handle_httpstatus_all)
* dont\_merge\_cookies（请参阅构造函数的cookies参数[**Request**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.Request)）
* [**cookiejar**](https://doc.scrapy.org/en/latest/topics/downloader-middleware.html#std:reqmeta-cookiejar)
* [**dont\_cache**](https://doc.scrapy.org/en/latest/topics/downloader-middleware.html#std:reqmeta-dont_cache)
* [**redirect\_urls**](https://doc.scrapy.org/en/latest/topics/downloader-middleware.html#std:reqmeta-redirect_urls)
* [**bindaddress**](https://doc.scrapy.org/en/latest/topics/request-response.html#std:reqmeta-bindaddress)
* [**dont\_obey\_robotstxt**](https://doc.scrapy.org/en/latest/topics/downloader-middleware.html#std:reqmeta-dont_obey_robotstxt)
* [**download\_timeout**](https://doc.scrapy.org/en/latest/topics/request-response.html#std:reqmeta-download_timeout)
* [**download\_maxsize**](https://doc.scrapy.org/en/latest/topics/settings.html#std:reqmeta-download_maxsize)
* [**download\_latency**](https://doc.scrapy.org/en/latest/topics/request-response.html#std:reqmeta-download_latency)
* [**download\_fail\_on\_dataloss**](https://doc.scrapy.org/en/latest/topics/request-response.html#std:reqmeta-download_fail_on_dataloss)
* [**proxy**](https://doc.scrapy.org/en/latest/topics/downloader-middleware.html#std:reqmeta-proxy)
* ftp\_user（查看[**FTP\_USER**](https://doc.scrapy.org/en/latest/topics/settings.html#std:setting-FTP_USER)更多信息）
* ftp\_password（查看[**FTP\_PASSWORD**](https://doc.scrapy.org/en/latest/topics/settings.html#std:setting-FTP_PASSWORD)更多信息）
* [**referrer\_policy**](https://doc.scrapy.org/en/latest/topics/spider-middleware.html#std:reqmeta-referrer_policy)
* [**max\_retry\_times**](https://doc.scrapy.org/en/latest/topics/request-response.html#std:reqmeta-max_retry_times)

### bindaddress

用于执行请求的传出IP地址的IP。

### download\_timeout

下载器在超时之前等待的时间（以秒为单位）。另见：[**DOWNLOAD\_TIMEOUT**](https://doc.scrapy.org/en/latest/topics/settings.html#std:setting-DOWNLOAD_TIMEOUT)。

### download\_latency

从请求开始以来（即通过网络发送的HTTP消息）获取响应所花费的时间量。这个元键只有在响应被下载后才可用。虽然大多数其他元键用于控制Scrapy行为，但它应该是只读的。

### download\_fail\_on\_dataloss

是否对失败的反应失败。参见： [**DOWNLOAD\_FAIL\_ON\_DATALOSS**](https://doc.scrapy.org/en/latest/topics/settings.html#std:setting-DOWNLOAD_FAIL_ON_DATALOSS)。

### max\_retry\_times

元密钥用于设置每个请求的重试次数。初始化时， [**max\_retry\_times**](https://doc.scrapy.org/en/latest/topics/request-response.html#std:reqmeta-max_retry_times)元键优先于 [**RETRY\_TIMES**](https://doc.scrapy.org/en/latest/topics/downloader-middleware.html#std:setting-RETRY_TIMES)设置。

## 请求子类

这里是内置[**Request**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.Request)子类的列表。您也可以将其子类化以实现您自己的自定义功能。

### FormRequest对象

FormRequest类[**Request**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.Request)通过用于处理HTML表单的功能扩展了基础。它使用[lxml.html表单](http://lxml.de/lxmlhtml.html#forms) 预先填充来自[**Response**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.Response)对象的表单数据的表单字段。

**Class scrapy.http.FormRequest(url[, formdata, ...])**

本[**FormRequest**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.FormRequest)类增加了新的参数构造函数。其余的参数与[**Request**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.Request)该类相同，这里没有记录。

|  |  |
| --- | --- |
| **参数：** | **formdata**（字典或元组的迭代） - 是包含HTML表单数据的字典（或（键，值）元组的迭代器），这些数据将被URL编码并分配给请求的主体。 |

[**FormRequest**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.FormRequest)除了标准[**Request**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.Request)方法外，这些对象还支持以下类方法：

**classmethodfrom\_response（response [，formname = None，formid = None，formnumber = 0，formdata = None，formxpath = None，formcss = None，clickdata = None，dont\_click = False，... ]）**

返回一个新的[**FormRequest**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.FormRequest)对象，其表单字段值预先填充**<form>**到给定响应中包含的HTML 元素中。有关示例，请参阅 [使用FormRequest.from\_response（）模拟用户登录](https://doc.scrapy.org/en/latest/topics/request-response.html#topics-request-response-ref-request-userlogin)。

该政策默认情况下自动模拟点击任何可点击的表单控件，如 **<input type="submit">**。尽管这很方便，而且通常是所需的行为，但有时它可能会导致难以调试的问题。例如，在处理使用javascript填充和/或提交的表单时，[**from\_response()**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.FormRequest.from_response)默认行为可能不是最合适的。要禁用此行为，可以将参数**dont\_click**设置 为**True**。另外，如果你想改变点击的控件（而不是禁用它），你也可以使用 参数**clickdata 。**

**警告**

由于[lxml中存在](https://bugs.launchpad.net/lxml/+bug/1665241)一个[错误，在](https://bugs.launchpad.net/lxml/+bug/1665241)选项值中使用具有前导空白或尾随空白的select元素使用此方法将不起作用 ，这应该在lxml 3.8及更高版本中修复。

|  |  |
| --- | --- |
| **参数：** | * **响应**（[Response](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.Response)对象） - 包含将用于预填充表单域的HTML表单的响应 * **formname**（string） - 如果给定，将使用名称属性设置为此值的表单。 * **formid**（string） - 如果给定，将使用id属性设置为该值的表单。 * **formxpath**（string） - 如果给定，将使用匹配xpath的第一个表单。 * **formcss**（string） - 如果给出，将使用匹配CSS选择器的第一个表单。 * **formnumber**（整数） - 响应包含多个表单时要使用的表单数。第一个（也是默认）是0。 * **formdata**（dict） - 要在表单数据中覆盖的字段。如果某个字段已经存在于响应<form>元素中，则其值会被此参数中传递的值覆盖。如果此参数中传递的值是None，则该字段将不会包含在请求中，即使它存在于响应<form>元素中。 * **clickdata**（dict） - 用于查找点击的控件的属性。如果没有给出，表单数据将被提交模拟点击第一个可点击的元素。除了html属性之外，控件还可以通过相对于表单内其他可提交输入的从零开始的索引，通过nr属性进行标识。 * **dont\_click**（boolean） - 如果为True，表单数据将被提交而不需要单击任何元素。 |

这个类方法的其他参数直接传递给 [**FormRequest**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.FormRequest)构造函数。

在新版本0.10.3：该**formname**参数。

在新版本0.17：该**formxpath**参数。

新的版本1.1.0：该**formcss**参数。

新的版本1.1.0：该**formid**参数。

### 请求使用示例

#### 使用FormRequest通过HTTP POST发送数据

如果你想在你的蜘蛛模拟一个HTML表单POST并发送一些键值字段，你可以[**FormRequest**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.FormRequest)像这样返回一个对象（来自你的蜘蛛）：



#### 使用FormRequest.from\_response（）模拟用户登录

网站通常通过元素提供预先填充的表单字段，例如会话相关数据或身份验证令牌（用于登录页面）。在抓取时，您需要自动预填这些字段，并仅覆盖其中的几个字段，例如用户名和密码。你可以使用 这个工作的方法。这是一个使用它的蜘蛛示例：<input type="hidden">[**FormRequest.from\_response()**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.FormRequest.from_response)



## Response objects

**Class scrapy.http.Response(url[, status=200, headers=None, body=b'', flags=None, request=None])**

一个[**Response**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.Response)对象代表一个HTTP响应，通常会下载（通过下载器）并传送给蜘蛛进行处理。

|  |  |
| --- | --- |
| **参数：** | * **url**（*字符串*） - 此响应的URL * **status** (integer) - 响应的HTTP状态。默认为**200**。 * **headers** (dict) - 这个响应的标题。字典值可以是字符串（对于单值标题）或列表（对于多值标题）。 * **body** (bytes)  - 响应主体。要以str（Python 2中的unicode）访问解码后的文本，您可以使用**response.text**编码感知的 [Response子类](https://doc.scrapy.org/en/latest/topics/request-response.html#topics-request-response-ref-response-subclasses)，例如[**TextResponse**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.TextResponse)。 * **flags** ([list](https://doc.scrapy.org/en/latest/topics/api.html#scrapy.loader.SpiderLoader.list))  - 是包含[**Response.flags**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.Response.flags)属性初始值的列表 。如果给出，列表将被浅拷贝。 * **request** ([**Request**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.Request) object) - [**Response.request**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.Response.request)属性的初始值。这代表了[**Request**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.Request)产生这个回应的那个。   **url**  一个包含响应URL的字符串。  该属性是只读的。更改Response使用的URL [**replace()**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.Response.replace)。  **status**  表示响应的HTTP状态的整数。例如：**200**， **404**。  **headers**  一个包含响应头文件的类似字典的对象。值可以通过使用**get()**指定名称返回第一个标头值或**getlist()**返回具有指定名称的所有标头值来访问。例如，这个调用会给你头上的所有cookie：    **body**  这个响应的主体。请记住，Response.body始终是一个字节对象。如果你想使用unicode版本[**TextResponse.text**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.TextResponse.text)（只能在[**TextResponse**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.TextResponse) 和子类中使用）。  该属性是只读的。改变Response的使用体 [**replace()**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.Response.replace)。  **request**  [**Request**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.Request)生成此响应的对象。在响应和请求已经通过所有[Downloader中间件](https://doc.scrapy.org/en/latest/topics/downloader-middleware.html#topics-downloader-middleware)之后，在Scrapy引擎中分配此属性。特别是，这意味着：   * HTTP重定向会将原始请求（重定向前的URL）分配给重定向的响应（重定向后使用最终的URL）。 * Response.request.url并不总是等于Response.url * 该属性仅在蜘蛛代码和[Spider Middleware中](https://doc.scrapy.org/en/latest/topics/spider-middleware.html#topics-spider-middleware)可用 ，但在Downloader Middleware中不可用（尽管您可以通过其他方式获得请求）以及[**response\_downloaded**](https://doc.scrapy.org/en/latest/topics/signals.html#std:signal-response_downloaded)信号的处理程序。   **meta**  对象[**Request.meta**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.Request.meta)属性的 快捷方式[**Response.request**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.Response.request)（即。**self.request.meta**）。  与[**Response.request**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.Response.request)属性不同，该[**Response.meta**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.Response.meta) 属性沿着重定向和重试传播，因此您将获得[**Request.meta**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.Request.meta)从蜘蛛发送的原始内容。  **也可以看看**  [**Request.meta**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.Request.meta) 属性  **flags**  包含此响应标志的列表。标志是用于标记响应的标签。例如：“缓存”，“重定向 ”等。它们显示在 引擎用于记录的Response（\_\_str\_\_方法）的字符串表示中。  **copy（）**  返回一个新的Response，它是此Response的副本。  **replace（[url，status，headers，body，request，flags，cls ]）**  使用相同的成员返回一个Response对象，除了那些由指定的关键字参数赋予新值的成员。该属性[**Response.meta**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.Response.meta)默认被复制。  **urljoin（url ）**  通过将Response [**url**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.Response.url)与可能的相对URL 结合来构造绝对URL。  这是一个通过[urlparse.urljoin](https://docs.python.org/2/library/urlparse.html#urlparse.urljoin)的封装，它仅仅是进行这个调用的别名： |

**follow（url，callback = None，method ='GET'，headers = None，body = None，cookies = None，meta = None，encoding ='utf-8'，priority = 0，dont\_filter = False，errback = None ）**

返回一个[**Request**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.Request)实例来跟踪链接**url**。它接受与**Request.\_\_init\_\_**方法相同的参数，但**url**可以是相对URL或**scrapy.link.Link**对象，不仅是绝对URL。

[**TextResponse**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.TextResponse)提供了[**follow()**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.TextResponse.follow) 一种除绝对/相对URL和链接对象之外还支持选择器的方法。

## 响应子类

以下是可用的内置Response子类的列表。您也可以继承Response类来实现您自己的功能。

### TextResponse对象

**Class scrapy.http.TextResponse(url[, encoding[, ...]])**

[**TextResponse**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.TextResponse)对象将编码功能添加到基[**Response**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.Response)类，该基 类仅用于二进制数据，如图像，声音或任何媒体文件。

[**TextResponse**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.TextResponse)除了基础[**Response**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.Response)对象之外，对象还支持一个新的构造函数参数。其余的功能与[**Response**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.Response)该类相同，不在此处记载。

|  |  |
| --- | --- |
| **参数：** | **encoding** (string)（字符串） - 是一个包含用于此响应的编码的字符串。如果[**TextResponse**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.TextResponse)使用unicode主体创建对象，则会使用此编码进行编码（请记住，body属性始终是一个字符串）。如果**encoding**是**None**（默认值），则将在响应标题和正文中查找编码。 |

[**TextResponse**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.TextResponse)除了标准对象之外，对象还支持以下属性[**Response**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.Response)：

**text**

响应主体，如unicode。

与**response.body.decode(response.encoding)**第一次调用后缓存的结果相同，因此可以**response.text**多次访问 而无需额外开销。

**注意**

**unicode(response.body)**不是将响应主体转换为unicode的正确方法：您将使用系统默认编码（通常为ascii）而不是响应编码。

**encoding**

一个包含此响应编码的字符串。通过按以下顺序尝试以下机制来解决编码问题：

1. 在构造函数编码参数中传递的编码
2. 在Content-Type HTTP头中声明的编码。如果这种编码无效（即未知），它将被忽略，并尝试下一个解析机制。
3. 在响应正文中声明的编码。TextResponse类没有为此提供任何特殊功能。但是，[**HtmlResponse**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.HtmlResponse)和[**XmlResponse**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.XmlResponse)类一样。
4. 通过查看响应主体来推断编码。这是更脆弱的方法，但也是最后一个尝试。

**selector**

甲[**Selector**](https://doc.scrapy.org/en/latest/topics/selectors.html#scrapy.selector.Selector)使用响应作为目标实例。第一次访问时，选择器是懒洋洋地实例化的。

[**TextResponse**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.TextResponse)除了标准对象外，对象还支持以下方法[**Response**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.Response)：

**xpath（查询）**

快捷方式**TextResponse.selector.xpath(query)**：

response**.**xpath('//p')

**css（查询）**

快捷方式**TextResponse.selector.css(query)**：

response**.**css('p')

**follow（url，callback = None，method ='GET'，headers = None，body = None，cookies = None，meta = None，encoding = None，priority = 0，dont\_filter = False，errback = None ）**

返回一个[**Request**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.Request)实例来跟踪链接**url**。它接受与**Request.\_\_init\_\_**方法相同的参数，但**url**不仅可以是绝对URL，还可以

* 相对URL;
* scrapy.link.Link对象（例如链接提取器结果）;
* 属性选择器（不是SelectorList） - 例如 **response.css('a::attr(href)')[0]**或**response.xpath('//img/@src')[0]**。
* 选择器**<a>**或**<link>**元素，例如 **response.css('a.my\_link')[0]**。

请参阅[创建](https://doc.scrapy.org/en/latest/intro/tutorial.html#response-follow-example)使用示例[请求的快捷方式](https://doc.scrapy.org/en/latest/intro/tutorial.html#response-follow-example)。

**body\_as\_unicode（）**

与之相同[**text**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.TextResponse.text)，但作为一种方法可用。这种方法保持向后兼容; 请选择**response.text**。

### HtmlResponse对象

**classscrapy.http.HtmlResponse（url [，... ]）**

该[**HtmlResponse**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.HtmlResponse)班是的子类[**TextResponse**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.TextResponse) ，其增加了通过查看HTML编码自动发现支持[META HTTP-EQUIV](https://www.w3schools.com/TAGS/att_meta_http_equiv.asp)属性。看[**TextResponse.encoding**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.TextResponse.encoding)。

### XmlResponse对象

**classscrapy.http.XmlResponse（url [，... ]）**

该[**XmlResponse**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.XmlResponse)班是的子类[**TextResponse**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.TextResponse)，其增加了通过查看XML声明行编码自动发现支持。看[**TextResponse.encoding**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.TextResponse.encoding)。

# 链接提取器

链接提取器是唯一目的是从网页（[**scrapy.http.Response**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.Response)对象）中提取链接的对象，这些链接最终将被跟踪。

scrapy.linkextractors.LinkExtractor在Scrapy 中可用，但您可以通过实现简单的界面来创建自己的自定义链接提取器，以满足您的需求。

每个链接提取器所具有的唯一公共方法是extract\_links接收一个[**Response**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.Response)对象并返回一个**scrapy.link.Link**对象列表。链接提取器意味着被实例化一次，并且他们的extract\_links方法被多次调用以提取不同的响应来提取要遵循的链接。

链接提取器[**CrawlSpider**](https://doc.scrapy.org/en/latest/topics/spiders.html#scrapy.spiders.CrawlSpider) 通过一系列规则在类中使用（在Scrapy中可用），但即使不从子类继承，也可以在蜘蛛 中使用链接提取器[**CrawlSpider**](https://doc.scrapy.org/en/latest/topics/spiders.html#scrapy.spiders.CrawlSpider)，因为它的目的非常简单：提取链接。

## 内置链接提取器参考

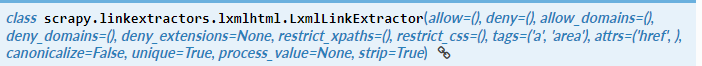
[**scrapy.linkextractors**](https://doc.scrapy.org/en/latest/topics/link-extractors.html#module-scrapy.linkextractors)模块中提供了与Scrapy捆绑在一起的链接提取器类 。

默认链接提取器是LinkExtractor，它与以下内容相同 [**LxmlLinkExtractor**](https://doc.scrapy.org/en/latest/topics/link-extractors.html#scrapy.linkextractors.lxmlhtml.LxmlLinkExtractor)：



以前的Scrapy版本中曾经有其他链接提取器类，但现在已弃用。

### LxmlLinkExtractor



LxmlLinkExtractor是推荐的链接提取器，具有方便的过滤选项。它使用lxml的健壮HTMLParser实现。

|  |  |
| --- | --- |
| **参数：** | * **allow**（正则表达式（或列表）） - 单个正则表达式（或正则表达式列表），以便（（绝对））网址必须匹配才能被提取。如果没有给出（或空），它将匹配所有链接。 * **deny**（一个正则表达式（或列表）） - 一个正则表达式（或正则表达式列表），为了被排除（即不被提取），（绝对）URL必须匹配。它优先于**allow**参数。如果没有给出（或空），它不会排除任何链接。 * **allow\_domains**（str 或[list](https://doc.scrapy.org/en/latest/topics/api.html#scrapy.loader.SpiderLoader.list)） - 包含将被考虑用于提取链接的域的单个值或字符串列表 * **deny\_domains**（str 或[list](https://doc.scrapy.org/en/latest/topics/api.html#scrapy.loader.SpiderLoader.list)） - 一个单独的值或包含域的字符串列表，这些字段不会被视为提取链接 * **deny\_extensions**（[list](https://doc.scrapy.org/en/latest/topics/api.html#scrapy.loader.SpiderLoader.list)） - 包含扩展的单个值或字符串列表，在提取链接时应被忽略。如果没有给出，它将默认为[scrapy.linkextractors](https://github.com/scrapy/scrapy/blob/master/scrapy/linkextractors/__init__.py)包**IGNORED\_EXTENSIONS**中定义的 列表 。 * **restrict\_xpaths**（str 或[list](https://doc.scrapy.org/en/latest/topics/api.html#scrapy.loader.SpiderLoader.list)） - 是一个XPath（或XPath的列表），它定义了应该从中提取链接的响应内的区域。如果给定，只有那些XPath选择的文本才会被扫描以查找链接。看下面的例子。 * **restrict\_css**（str 或[list](https://doc.scrapy.org/en/latest/topics/api.html#scrapy.loader.SpiderLoader.list)） - 一个CSS选择器（或选择器列表），它定义响应中应从中提取链接的区域。具有与......相同的行为**restrict\_xpaths**。 * **tags**（str 或[list](https://doc.scrapy.org/en/latest/topics/api.html#scrapy.loader.SpiderLoader.list)） - 提取链接时要考虑的标记或标记列表。默认为。**('a', 'area')** * **attrs**（[列表](https://doc.scrapy.org/en/latest/topics/api.html#scrapy.loader.SpiderLoader.list)） - 查找提取链接时应考虑的属性或属性列表（仅适用于**tags** 参数中指定的那些标签）。默认为**('href',)** * **canonicalize**（布尔） - 规范每个提取的URL（使用w3lib.url.canonicalize\_url）。默认为**False**。请注意，canonicalize\_url用于重复检查; 它可以更改服务器端可见的URL，因此对于使用规范化和原始URL的请求，响应可能会有所不同。如果您使用LinkExtractor来跟踪链接，那么保留默认值会更加健壮**canonicalize=False**。 * **unique** （布尔） - 是否应对抽取的链接应用重复筛选。 * **process\_value**（可调用） -   接收从标签提取的每个值和被扫描属性的函数，并且可以修改该值并返回一个新值，或者返回**None**忽略该链接。如果没有给出，则**process\_value**默认为。**lambda x: x**  例如，要从此代码中提取链接： |



* **strip**（*布尔*） - 是否从提取的属性中去除空白。根据HTML5标准，前导和尾部空格必须从被剥离**href**的属性**<a>**，**<area>** 以及许多其他的元素，**src**属性**<img>**，**<iframe>** 元件等，所以LinkExtractor默认条空间字符。设置**strip=False**关闭它（例如，如果你从元素或属性，允许前导/尾随空格提取网址）。

# Settings

Scrapy设置允许您自定义所有Scrapy组件的行为，包括核心，扩展，管道和蜘蛛本身。

设置的基础结构提供了代码可用于从中提取配置值的键值映射的全局名称空间。这些设置可以通过不同的机制进行填充，下面将对此进行介绍。

这些设置也是选择当前活动的Scrapy项目的机制（以防万一）。

有关可用内置设置的列表，请参阅：[内置设置参考](https://doc.scrapy.org/en/latest/topics/settings.html#topics-settings-ref)。

## 指定设置

当你使用Scrapy时，你必须告诉它你正在使用哪些设置。你可以通过使用环境变量来做到这一点SCRAPY\_SETTINGS\_MODULE。

值SCRAPY\_SETTINGS\_MODULE应该在Python路径语法中，例如 myproject.settings。请注意，设置模块应该位于Python [导入搜索路径中](https://docs.python.org/2/tutorial/modules.html#the-module-search-path)。

## 填充设置

可以使用不同的机制来填充设置，每种机制都有不同的优先级。以下是按优先级降序排列的列表：

1. 命令行选项（最优先）
2. 每个蜘蛛的设置
3. 项目设置模块
4. 每个命令的默认设置
5. 默认的全局设置（优先级较低）

这些设置源的人口在内部得到了处理，但使用API​​调用可以手动处理。请参阅 [设置API](https://doc.scrapy.org/en/latest/topics/api.html#topics-api-settings)主题以供参考。

这些机制在下面更详细地描述。

### 1.命令行选项

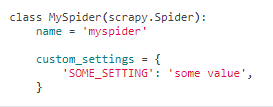
命令行提供的参数是最优先的参数，覆盖任何其他选项。您可以使用-s（或--set）命令行选项明确地覆盖一个（或多个）设置。

例：



### 2.设置每个蜘蛛

蜘蛛（请参阅[蜘蛛](https://doc.scrapy.org/en/latest/topics/spiders.html#topics-spiders)章节以供参考）可以定义它们自己的设置，这些设置将优先考虑并覆盖项目。他们可以通过设置[**custom\_settings**](https://doc.scrapy.org/en/latest/topics/spiders.html#scrapy.spiders.Spider.custom_settings)属性来实现：



### 3.项目设置模块

项目设置模块是Scrapy项目的标准配置文件，它是大多数自定义设置将被填充的地方。对于标准Scrapy项目，这意味着您将添加或更改settings.py为项目创建的文件中的设置。

### 4.每个命令的默认设置

每个[Scrapy工具](https://doc.scrapy.org/en/latest/topics/commands.html)命令都可以有自己的默认设置，它们覆盖全局默认设置。这些自定义命令设置是在default\_settings命令类的属性中指定的。

### 5.默认全局设置

全局默认值位于scrapy.settings.default\_settings 模块中，并记录在“ [内置设置参考”](https://doc.scrapy.org/en/latest/topics/settings.html#topics-settings-ref)部分。

## 如何访问设置

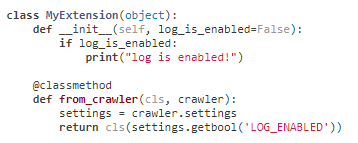
蜘蛛中的设置可以通过self.settings以下方式获得：



**注意**

该settings属性在蜘蛛初始化后设置在基础Spider类中。如果你想在初始化之前使用这些设置（例如，在你的蜘蛛的\_\_init\_\_()方法中），你需要重写该 [**from\_crawler()**](https://doc.scrapy.org/en/latest/topics/spiders.html#scrapy.spiders.Spider.from_crawler)方法。

可以通过[**scrapy.crawler.Crawler.settings**](https://doc.scrapy.org/en/latest/topics/api.html#scrapy.crawler.Crawler.settings) 传递给from\_crawler扩展，中间件和项目管道中的方法的Crawler 的属性来访问设置：



设置对象可以像字典（例如， settings['LOG\_ENABLED']）一样使用，但通常最好使用[**Settings**](https://doc.scrapy.org/en/latest/topics/api.html#scrapy.settings.Settings)API 提供的方法之一以您需要的格式提取设置以避免类型错误。

## 设置名称的理由

设置名称通常以它们配置的组件为前缀。例如，对于一个虚构的robots.txt扩展正确的设置名称会ROBOTSTXT\_ENABLED，ROBOTSTXT\_OBEY，ROBOTSTXT\_CACHEDIR，等。

## 内置设置参考

以下列出了所有可用的Scrapy设置，按字母顺序排列，以及它们的默认值和应用范围。

范围（如果可用）显示设置的使用位置，如果与任何特定组件绑定。在这种情况下，将显示该组件的模块，通常是扩展，中间件或管道。这也意味着必须启用该组件才能使设置发挥作用。

### AWS\_ACCESS\_KEY\_ID

默认： None

需要访问[Amazon Web服务](https://aws.amazon.com/)的代码使用的AWS访问密钥，例如[S3提要存储后端](https://doc.scrapy.org/en/latest/topics/feed-exports.html#topics-feed-storage-s3)。

### AWS\_SECRET\_ACCESS\_KEY

默认： None

需要访问[Amazon Web服务](https://aws.amazon.com/)的代码使用的AWS密钥，例如[S3提要存储后端](https://doc.scrapy.org/en/latest/topics/feed-exports.html#topics-feed-storage-s3)。

### BOT\_NAME

默认： 'scrapybot'

此Scrapy项目实现的bot的名称（也称为项目名称）。这将用于默认构建User-Agent，也用于记录。

当您使用该[**startproject**](https://doc.scrapy.org/en/latest/topics/commands.html#std:command-startproject)命令创建项目时，它会自动填充您的项目名称。

### CONCURRENT\_ITEMS

默认： 100

在项目处理器（也称为[项目管道](https://doc.scrapy.org/en/latest/topics/item-pipeline.html#topics-item-pipeline)）中并行处理的最大并行项目数（每个响应）。

### CONCURRENT\_REQUESTS

默认： 16

Scrapy下载器将执行的并发（即同时）请求的最大数量。

### CONCURRENT\_REQUESTS\_PER\_DOMAIN

默认： 8

将对任何单个域执行的并发（即同时）请求的最大数量。

另请参阅：[AutoThrottle扩展](https://doc.scrapy.org/en/latest/topics/autothrottle.html#topics-autothrottle)及其 [**AUTOTHROTTLE\_TARGET\_CONCURRENCY**](https://doc.scrapy.org/en/latest/topics/autothrottle.html#std:setting-AUTOTHROTTLE_TARGET_CONCURRENCY)选项。

### CONCURRENT\_REQUESTS\_PER\_IP

默认： 0

将对任何单个IP执行的并发（即同时）请求的最大数目。如果非零，该[**CONCURRENT\_REQUESTS\_PER\_DOMAIN**](https://doc.scrapy.org/en/latest/topics/settings.html#std:setting-CONCURRENT_REQUESTS_PER_DOMAIN)设置将被忽略，并使用该 设置。换言之，并发限制将应用于每个IP，而不是每个域。

此设置还会影响[**DOWNLOAD\_DELAY**](https://doc.scrapy.org/en/latest/topics/settings.html#std:setting-DOWNLOAD_DELAY)和 [油门延伸](https://doc.scrapy.org/en/latest/topics/autothrottle.html#topics-autothrottle)：如果[**CONCURRENT\_REQUESTS\_PER\_IP**](https://doc.scrapy.org/en/latest/topics/settings.html#std:setting-CONCURRENT_REQUESTS_PER_IP) 不为零，下载延迟每个IP强制执行，而不是每个域。

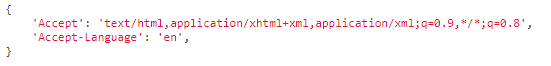
### DEFAULT\_ITEM\_CLASS

默认： 'scrapy.item.Item'

将用于实例化[Scrapy shell](https://doc.scrapy.org/en/latest/topics/shell.html#topics-shell)中的项目的默认类。

### DEFAULT\_REQUEST\_HEADERS

默认：



用于Scrapy HTTP请求的默认标题。他们在人群中居住 [**DefaultHeadersMiddleware**](https://doc.scrapy.org/en/latest/topics/downloader-middleware.html#scrapy.downloadermiddlewares.defaultheaders.DefaultHeadersMiddleware)。

### DEPTH\_LIMIT

默认： 0

范围： scrapy.spidermiddlewares.depth.DepthMiddleware

将允许为任何网站抓取的最大深度。如果为零，则不会施加限制。

### DEPTH\_PRIORITY

默认： 0

范围： scrapy.spidermiddlewares.depth.DepthMiddleware

根据深度调整请求优先级的整数：

* 如果为零（默认），则不会从深度进行优先级调整
* **正值会降低优先级，即较高深度的请求将在稍后处理** ; 这是广泛进行抓取时常用的（BFO）
* 负值会增加优先级，即更高深度的请求将被更快地处理（DFO）

另请参见：[Scrapy是否以广度优先或深度优先的顺序进行爬网？](https://doc.scrapy.org/en/latest/faq.html#faq-bfo-dfo)关于调整Scrapy为BFO或DFO。

**注意**

此设置调整优先级**以相反的方式**相对于其他优先级设置[**REDIRECT\_PRIORITY\_ADJUST**](https://doc.scrapy.org/en/latest/topics/settings.html#std:setting-REDIRECT_PRIORITY_ADJUST)和[**RETRY\_PRIORITY\_ADJUST**](https://doc.scrapy.org/en/latest/topics/settings.html#std:setting-RETRY_PRIORITY_ADJUST)。

### DEPTH\_STATS

默认： True

范围： scrapy.spidermiddlewares.depth.DepthMiddleware

是否收集最大深度统计。

### DEPTH\_STATS\_VERBOSE

默认： False

范围： scrapy.spidermiddlewares.depth.DepthMiddleware

是否收集详细深度统计信息。如果启用此功能，则会在统计信息中收集每个深度的请求数量。

### DNSCACHE\_ENABLED

默认： True

是否启用DNS内存中缓存。

### DNSCACHE\_SIZE

默认： 10000

DNS内存中缓存大小。

### DNS\_TIMEOUT

默认： 60

以秒为单位处理DNS查询超时。浮点支持。

### DOWNLOADER

默认： 'scrapy.core.downloader.Downloader'

用于抓取的下载程序。

### DOWNLOADER\_HTTPCLIENTFACTORY

默认： 'scrapy.core.downloader.webclient.ScrapyHTTPClientFactory'

定义protocol.ClientFactory 用于HTTP / 1.0连接（for HTTP10DownloadHandler）的Twisted 类。

**注意**

HTTP / 1.0现在很少或使用，因此您可以放心地忽略此设置，除非你使用双绞线<11.1，如果你真的想使用HTTP / 1.0，并覆盖[**DOWNLOAD\_HANDLERS\_BASE**](https://doc.scrapy.org/en/latest/topics/settings.html#std:setting-DOWNLOAD_HANDLERS_BASE)了http(s)相应的方案，即'scrapy.core.downloader.handlers.http.HTTP10DownloadHandler'。

### DOWNLOADER\_CLIENTCONTEXTFACTORY

默认： 'scrapy.core.downloader.contextfactory.ScrapyClientContextFactory'

表示要使用的ContextFactory的类路径。

在这里，“ContextFactory”是SSL / TLS上下文的Twisted术语，定义要使用的TLS / SSL协议版本，是否执行证书验证，甚至启用客户端身份验证（以及其他各种各样的事情）。

**注意**

Scrapy默认上下文工厂**不执行远程服务器证书验证**。这通常适用于网页抓取。

如果您确实需要启用远程服务器证书验证，则Scrapy还有另一个可设置的上下文工厂类'scrapy.core.downloader.contextfactory.BrowserLikeContextFactory'，它使用平台的证书来验证远程端点。 **这仅在使用Twisted> = 14.0时可用。**

如果您使用自定义的ContextFactory，请确保它method 在init处接受参数（这是OpenSSL.SSL方法映射 [**DOWNLOADER\_CLIENT\_TLS\_METHOD**](https://doc.scrapy.org/en/latest/topics/settings.html#std:setting-DOWNLOADER_CLIENT_TLS_METHOD)）。

### DOWNLOADER\_CLIENT\_TLS\_METHOD

默认： 'TLS'

使用此设置可以自定义默认HTTP / 1.1下载器使用的TLS / SSL方法。

该设置必须是以下字符串值之一：

* 'TLS'：映射到OpenSSL TLS\_method()（aka SSLv23\_method()），它允许从平台支持的最高级开始进行协议协商; **默认，推荐**
* 'TLSv1.0'：此值强制HTTPS连接使用TLS版本1.0; 如果您希望Scrapy <1.1的行为，请设置此值
* 'TLSv1.1'：强制TLS 1.1版
* 'TLSv1.2'：强制TLS 1.2版
* 'SSLv3'：强制SSL版本3（**不推荐**）

**注意**

我们建议您使用PyOpenSSL> = 0.13和Twisted> = 0.13或更高（如果可以，Twisted> = 14.0）。

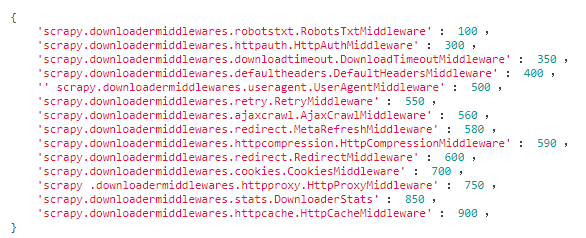
### DOWNLOADER\_MIDDLEWARES

默认：： {}

包含您项目中启用的下载器中间件的字典及其订单。有关更多信息，请参阅[激活下载中间件](https://doc.scrapy.org/en/latest/topics/downloader-middleware.html#topics-downloader-middleware-setting)。

### DOWNLOADER\_MIDDLEWARES\_BASE

默认：



包含Scrapy中默认启用的下载器中间件的字典。低订单靠近引擎，高订单靠近下载器。你不应该在你的项目中修改这个设置，[**DOWNLOADER\_MIDDLEWARES**](https://doc.scrapy.org/en/latest/topics/settings.html#std:setting-DOWNLOADER_MIDDLEWARES)而应该修改 。有关更多信息，请参阅 [激活下载中间件](https://doc.scrapy.org/en/latest/topics/downloader-middleware.html#topics-downloader-middleware-setting)。

### DOWNLOADER\_STATS

默认： True

是否启用下载器统计信息收集。

### DOWNLOAD\_DELAY

默认： 0

下载器在从同一网站下载连续页面之前应等待的时间（以秒为单位）。这可以用来限制爬网速度，以避免太大的打击服务器。支持十进制数字。例：



此设置也受[**RANDOMIZE\_DOWNLOAD\_DELAY**](https://doc.scrapy.org/en/latest/topics/settings.html#std:setting-RANDOMIZE_DOWNLOAD_DELAY) 设置（默认情况下启用）的影响。默认情况下，Scrapy不会在请求之间等待一段固定的时间，但会使用0.5 \* [**DOWNLOAD\_DELAY**](https://doc.scrapy.org/en/latest/topics/settings.html#std:setting-DOWNLOAD_DELAY)和1.5 \* 之间的随机时间间隔[**DOWNLOAD\_DELAY**](https://doc.scrapy.org/en/latest/topics/settings.html#std:setting-DOWNLOAD_DELAY)。

何时[**CONCURRENT\_REQUESTS\_PER\_IP**](https://doc.scrapy.org/en/latest/topics/settings.html#std:setting-CONCURRENT_REQUESTS_PER_IP)非零，每个IP地址而不是每个域强制执行延迟。

您也可以通过设置download\_delay 蜘蛛属性来更改每个蜘蛛的这个设置。

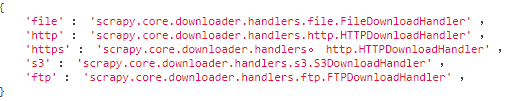
### DOWNLOAD\_HANDLERS

默认： {}

包含在您的项目中启用的请求下载程序处理程序的字典。[**DOWNLOAD\_HANDLERS\_BASE**](https://doc.scrapy.org/en/latest/topics/settings.html#std:setting-DOWNLOAD_HANDLERS_BASE)例如参见格式。

### DOWNLOAD\_HANDLERS\_BASE

默认：



包含Scrapy中默认启用的请求下载处理程序的字典。你不应该在你的项目中修改这个设置，[**DOWNLOAD\_HANDLERS**](https://doc.scrapy.org/en/latest/topics/settings.html#std:setting-DOWNLOAD_HANDLERS)而应该修改 。

您可以通过在其中分配NoneURI模式来禁用任何这些下载处理程序[**DOWNLOAD\_HANDLERS**](https://doc.scrapy.org/en/latest/topics/settings.html#std:setting-DOWNLOAD_HANDLERS)。例如，要禁用内置的FTP处理程序（无需替换），请将其放置在settings.py：



### DOWNLOAD\_TIMEOUT

默认： 180

下载器在超时之前等待的时间（以秒为单位）。

**注意**

可以使用**download\_timeout** spider属性和每个请求使用[**download\_timeout**](https://doc.scrapy.org/en/latest/topics/request-response.html#std:reqmeta-download_timeout) Request.meta键为每个蜘蛛设置此超时。

### DOWNLOAD\_MAXSIZE

默认：1073741824（1024MB）

下载器将下载的最大响应大小（以字节为单位）。

如果你想禁用它，设置为0。

**注意**

这个大小可以使用**download\_maxsize** spider属性和每个请求使用[**download\_maxsize**](https://doc.scrapy.org/en/latest/topics/settings.html#std:reqmeta-download_maxsize) Request.meta键为每个蜘蛛设置。

此功能需要Twisted> = 11.1。

### DOWNLOAD\_WARNSIZE

默认：33554432（32MB）

下载器将开始发出警告的响应大小（以字节为单位）。

如果你想禁用它，设置为0。

**注意**

这个大小可以使用**download\_warnsize** spider属性和每个请求使用**download\_warnsize** Request.meta键为每个蜘蛛设置。

此功能需要Twisted> = 11.1。

### DOWNLOAD\_FAIL\_ON\_DATALOSS

默认： True

是否在失败响应中失败，即声明 Content-Length与服务器发送的内容不匹配，或分块响应未正确完成。如果True这些回应产生 ResponseFailed([\_DataLoss])错误。如果False这些响应通过并且该标志dataloss被添加到响应中，即： 是。'dataloss' in response.flagsTrue

或者，可以通过使用[**download\_fail\_on\_dataloss**](https://doc.scrapy.org/en/latest/topics/request-response.html#std:reqmeta-download_fail_on_dataloss)Request.meta键来设置每个请求的基础 False。

**注意**

在几种情况下（从服务器错误配置到网络错误到数据损坏），可能发生错误的响应或数据丢失错误。用户可以决定是否有理由处理破坏的响应，因为它们可能包含部分或不完整的内容。如果[**RETRY\_ENABLED**](https://doc.scrapy.org/en/latest/topics/downloader-middleware.html#std:setting-RETRY_ENABLED)是，True并且此设置设置为True，则ResponseFailed([\_DataLoss])失败将照常重试。

### DUPEFILTER\_CLASS

默认： 'scrapy.dupefilters.RFPDupeFilter'

该类用于检测和过滤重复的请求。

RFPDupeFilter基于请求指纹的默认（）过滤器使用该scrapy.utils.request.request\_fingerprint函数。为了改变检查重复的方式，你可以继承RFPDupeFilter和重写它的request\_fingerprint方法。该方法应该接受scrapy [**Request**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.Request)对象并返回它的指纹（一个字符串）。

您可以通过设置[**DUPEFILTER\_CLASS**](https://doc.scrapy.org/en/latest/topics/settings.html#std:setting-DUPEFILTER_CLASS)为禁用重复请求的过滤 'scrapy.dupefilters.BaseDupeFilter'。不过要小心，因为你可以进入爬行循环。将dont\_filter参数设置为不应被过滤True的特定参数 通常是更好的办法[**Request**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.Request)。

### DUPEFILTER\_DEBUG

默认： False

默认情况下，RFPDupeFilter只记录第一个重复请求。设置[**DUPEFILTER\_DEBUG**](https://doc.scrapy.org/en/latest/topics/settings.html#std:setting-DUPEFILTER_DEBUG)为True将使其记录所有重复的请求。

### EDITOR

默认:( vi在Unix系统上）或IDLE编辑器（在Windows上）

编辑器用于使用该[**edit**](https://doc.scrapy.org/en/latest/topics/commands.html#std:command-edit)命令编辑蜘蛛。此外，如果EDITOR设置了环境变量，则该[**edit**](https://doc.scrapy.org/en/latest/topics/commands.html#std:command-edit) 命令将优先于默认设置。

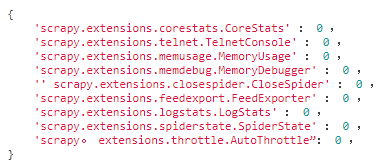
### EXTENSIONS

默认：： {}

一个包含在您的项目中启用的扩展的字典，以及它们的命令。

### EXTENSIONS\_BASE

默认：



包含Scrapy中默认可用扩展名的字典及其订单。该设置包含所有稳定的内置扩展。请记住，其中一些需要通过设置启用。

有关更多信息，请参阅[附加用户指南](https://doc.scrapy.org/en/latest/topics/extensions.html#topics-extensions) 和[可用扩展](https://doc.scrapy.org/en/latest/topics/extensions.html#topics-extensions-ref)的[列表](https://doc.scrapy.org/en/latest/topics/extensions.html#topics-extensions-ref)。

### FEED\_TEMPDIR

Feed Temp目录允许您设置一个自定义文件夹来保存抓取工具临时文件，然后使用[FTP Feed存储](https://doc.scrapy.org/en/latest/topics/feed-exports.html#topics-feed-storage-ftp)和[Amazon S3](https://doc.scrapy.org/en/latest/topics/feed-exports.html#topics-feed-storage-s3)进行上传。

### FTP\_PASSIVE\_MODE

默认： True

启动FTP传输时是否使用被动模式。

### FTP\_PASSWORD

默认： "guest"

该密码才能使用FTP连接时，有没有"ftp\_password" 在Request元。

**注意**

解释[RFC 1635](https://tools.ietf.org/html/rfc1635)，虽然通常使用匿名FTP的密码“guest”或电子邮件地址，但某些FTP服务器明确要求用户的电子邮件地址，并且不允许使用“guest”密码登录。

### FTP\_USER

默认： "anonymous"

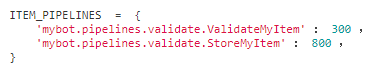
用户名使用的FTP连接时，有没有"ftp\_user" 在Request元。

### ITEM\_PIPELINES

默认： {}

包含要使用的物品管道的字典及其订单。顺序值是任意的，但习惯上将它们定义在0-1000范围内。在下单之前处理下单。

例：



### ITEM\_PIPELINES\_BASE

默认： {}

包含Scrapy中默认启用的管道的字典。你不应该在你的项目中修改这个设置，[**ITEM\_PIPELINES**](https://doc.scrapy.org/en/latest/topics/settings.html#std:setting-ITEM_PIPELINES)而应该修改。

### LOG\_ENABLED

默认： True

是否启用日志记录。

### LOG\_ENCODING

默认： 'utf-8'

用于记录的编码。

### LOG\_FILE

默认： None

用于记录输出的文件名。如果None将使用标准错误。

### LOG\_FORMAT

默认： '%(asctime)s [%(name)s] %(levelname)s: %(message)s'

用于格式化日志消息的字符串。有关可用占位符的整个列表，请参阅[Python日志记录文档](https://docs.python.org/2/library/logging.html#logrecord-attributes)。

### LOG\_DATEFORMAT

默认： '%Y-%m-%d %H:%M:%S'

用于格式化日期/时间的字符串，%(asctime)s占位符的扩展[**LOG\_FORMAT**](https://doc.scrapy.org/en/latest/topics/settings.html#std:setting-LOG_FORMAT)。有关可用指令的整个列表，请参阅[Python日期时间文档](https://docs.python.org/2/library/datetime.html#strftime-and-strptime-behavior)。

### LOG\_LEVEL

默认： 'DEBUG'

记录的最低级别。可用的级别是：关键，错误，警告，信息，调试。有关更多信息，请参阅[记录](https://doc.scrapy.org/en/latest/topics/logging.html#topics-logging)。

### LOG\_STDOUT

默认： False

如果True，你的进程的所有标准输出（和错误）将被重定向到日志。例如，如果它将出现在Scrapy日志中。print 'hello'

### LOG\_SHORT\_NAMES

默认： False

如果True日志只包含根路径。如果设置为，False 则显示负责日志输出的组件

### MEMDEBUG\_ENABLED

默认： False

是否启用内存调试。

### MEMDEBUG\_NOTIFY

默认： []

当启用内存调试时，如果此设置不为空，内存报告将发送到指定的地址，否则报告将写入日志。

例：

MEMDEBUG\_NOTIFY **=**  [ 'user@example.com' ]

### MEMUSAGE\_ENABLED

默认： True

范围： scrapy.extensions.memusage

是否启用内存使用扩展。该扩展跟踪进程使用的峰值内存（将其写入统计数据）。它还可以选择性地在Scrapy进程超过内存限制时关闭（请参阅参考资料[**MEMUSAGE\_LIMIT\_MB**](https://doc.scrapy.org/en/latest/topics/settings.html#std:setting-MEMUSAGE_LIMIT_MB)），并在发生这种情况时通过电子邮件通知（请参阅参考资料[**MEMUSAGE\_NOTIFY\_MAIL**](https://doc.scrapy.org/en/latest/topics/settings.html#std:setting-MEMUSAGE_NOTIFY_MAIL)）。

请参阅[内存使用扩展](https://doc.scrapy.org/en/latest/topics/extensions.html#topics-extensions-ref-memusage)。

### MEMUSAGE\_LIMIT\_MB

默认： 0

范围： scrapy.extensions.memusage

在关闭Scrapy之前允许的最大内存容量（以兆字节为单位）（如果MEMUSAGE\_ENABLED为True）。如果为零，则不执行检查。

请参阅[内存使用扩展](https://doc.scrapy.org/en/latest/topics/extensions.html#topics-extensions-ref-memusage)。

### MEMUSAGE\_CHECK\_INTERVAL\_SECONDS

版本1.1中的新功能

默认： 60.0

范围： scrapy.extensions.memusage

的[内存使用扩展](https://doc.scrapy.org/en/latest/topics/extensions.html#topics-extensions-ref-memusage) 会检查当前存储器使用，相对于限制由设置 [**MEMUSAGE\_LIMIT\_MB**](https://doc.scrapy.org/en/latest/topics/settings.html#std:setting-MEMUSAGE_LIMIT_MB)和[**MEMUSAGE\_WARNING\_MB**](https://doc.scrapy.org/en/latest/topics/settings.html#std:setting-MEMUSAGE_WARNING_MB)，以固定时间间隔。

这设置这些间隔的长度，以秒为单位。

请参阅[内存使用扩展](https://doc.scrapy.org/en/latest/topics/extensions.html#topics-extensions-ref-memusage)。

### MEMUSAGE\_NOTIFY\_MAIL

默认： False

范围： scrapy.extensions.memusage

电子邮件列表，通知是否已达到内存限制。

例：

MEMUSAGE\_NOTIFY\_MAIL **=**  [ 'user@example.com' ]

请参阅[内存使用扩展](https://doc.scrapy.org/en/latest/topics/extensions.html#topics-extensions-ref-memusage)。

### MEMUSAGE\_WARNING\_MB

默认： 0

范围： scrapy.extensions.memusage

在发送警告电子邮件通知它之前，允许的最大内存量（以兆字节为单位）。如果为零，则不会产生警告。

### NEWSPIDER\_MODULE

默认： ''

模块在哪里使用[**genspider**](https://doc.scrapy.org/en/latest/topics/commands.html#std:command-genspider)命令创建新的蜘蛛。

例：

NEWSPIDER\_MODULE **=**  'mybot.spiders\_dev'

### RANDOMIZE\_DOWNLOAD\_DELAY

默认： True

如果启用，Scrapy将在从同一网站获取请求时等待一段随机时间（介于0.5 \* [**DOWNLOAD\_DELAY**](https://doc.scrapy.org/en/latest/topics/settings.html#std:setting-DOWNLOAD_DELAY)和1.5 \*之间[**DOWNLOAD\_DELAY**](https://doc.scrapy.org/en/latest/topics/settings.html#std:setting-DOWNLOAD_DELAY)）。

这种随机化降低了爬虫被分析请求的站点检测到（并随后被阻止）的可能性，这些站点在请求之间的时间内寻找统计上显着的相似性。

随机策略与[wget](https://www.gnu.org/software/wget/manual/wget.html) --random-wait选项使用的策略相同。

如果[**DOWNLOAD\_DELAY**](https://doc.scrapy.org/en/latest/topics/settings.html#std:setting-DOWNLOAD_DELAY)为零（默认），则此选项无效。

### REACTOR\_THREADPOOL\_MAXSIZE

默认： 10

Twisted Reactor线程池大小的最大限制。这是各种Scrapy组件使用的常见多用途线程池。Threaded DNS Resolver，BlockingFeedStorage，S3FilesStore等等。如果您遇到阻塞IO不足的问题，请增加此值。

### REDIRECT\_MAX\_TIMES

默认： 20

定义请求可以重定向的最大次数。在此最大值之后，请求的响应按原样返回。我们将Firefox的默认值用于相同的任务。

### REDIRECT\_PRIORITY\_ADJUST

默认： +2

范围： scrapy.downloadermiddlewares.redirect.RedirectMiddleware

调整重定向请求相对于原始请求的优先级：

* **积极的优先级调整（默认）意味着更高的优先级。**
* 否定优先级调整意味着优先级较低。

### RETRY\_PRIORITY\_ADJUST

默认： -1

范围： scrapy.downloadermiddlewares.retry.RetryMiddleware

调整相对于原始请求的重试请求优先级：

* 积极的优先调整意味着更高的优先。
* **负优先级调整（默认）意味着较低的优先级。**

### ROBOTSTXT\_OBEY

默认： False

范围： scrapy.downloadermiddlewares.robotstxt

如果启用，Scrapy将尊重robots.txt策略。欲了解更多信息，请参阅 [RobotsTxtMiddleware](https://doc.scrapy.org/en/latest/topics/downloader-middleware.html#topics-dlmw-robots)。

**注意**

虽然默认值是False出于历史原因，但此选项在命令生成的settings.py文件中默认启用。scrapy startproject

### SCHEDULER

默认： 'scrapy.core.scheduler.Scheduler'

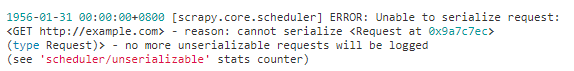
调度程序用于抓取。

### SCHEDULER\_DEBUG

默认： False

设置为True将记录有关请求调度程序的调试信息。如果请求无法序列化到磁盘，则此操作当前会记录（仅一次）。统计计数器（scheduler/unserializable）跟踪发生这种情况的次数。

日志中的示例条目：



### SCHEDULER\_DISK\_QUEUE

默认： 'scrapy.squeues.PickleLifoDiskQueue'

调度程序将使用的磁盘队列的类型。其他可用的类型有scrapy.squeues.PickleFifoDiskQueue，scrapy.squeues.MarshalFifoDiskQueue，scrapy.squeues.MarshalLifoDiskQueue。

### SCHEDULER\_MEMORY\_QUEUE

默认： 'scrapy.squeues.LifoMemoryQueue'

调度程序使用的内存中队列的类型。其他可用的类型是： scrapy.squeues.FifoMemoryQueue。

### SCHEDULER\_PRIORITY\_QUEUE

默认： 'queuelib.PriorityQueue'

调度程序使用的优先级队列的类型。

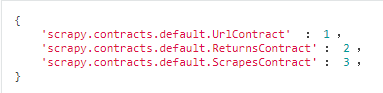
### SPIDER\_CONTRACTS

默认：： {}

一个包含您的项目中启用的蜘蛛协议的字典，用于测试蜘蛛。欲了解更多信息，请参阅[蜘蛛合同](https://doc.scrapy.org/en/latest/topics/contracts.html#topics-contracts)。

### SPIDER\_CONTRACTS\_BASE

默认：



包含Scrapy默认启用的scrapy合约的字典。你不应该在你的项目中修改这个设置，[**SPIDER\_CONTRACTS**](https://doc.scrapy.org/en/latest/topics/settings.html#std:setting-SPIDER_CONTRACTS)而应该修改。欲了解更多信息，请参阅[蜘蛛合同](https://doc.scrapy.org/en/latest/topics/contracts.html#topics-contracts)。

您可以通过分配其中None的类路径来禁用任何这些合约[**SPIDER\_CONTRACTS**](https://doc.scrapy.org/en/latest/topics/settings.html#std:setting-SPIDER_CONTRACTS)。例如，要禁用内置功能ScrapesContract，请将其置于以下位置settings.py：



### SPIDER\_LOADER\_CLASS

默认： 'scrapy.spiderloader.SpiderLoader'

将用于加载蜘蛛的类，它必须实现 [SpiderLoader API](https://doc.scrapy.org/en/latest/topics/api.html#topics-api-spiderloader)。

### SPIDER\_LOADER\_WARN\_ONLY

1.3.3版本中的新功能。

默认： False

默认情况下，当scrapy试图从中导入蜘蛛类时[**SPIDER\_MODULES**](https://doc.scrapy.org/en/latest/topics/settings.html#std:setting-SPIDER_MODULES)，如果有任何ImportError异常，它将会大声地失败。但是，您可以选择将此异常消除并通过设置将其变为简单警告。SPIDER\_LOADER\_WARN\_ONLY = True

**注意**

有些[scrapy命令](https://doc.scrapy.org/en/latest/topics/commands.html#topics-commands)使用此设置运行True 已经（即他们只会发出警告并不会失败），因为他们实际上并不需要加载蜘蛛类的工作：

[**scrapy runspider**](https://doc.scrapy.org/en/latest/topics/commands.html#std:command-runspider), [**scrapy settings**](https://doc.scrapy.org/en/latest/topics/commands.html#std:command-settings), [**scrapy startproject**](https://doc.scrapy.org/en/latest/topics/commands.html#std:command-startproject), [**scrapy version**](https://doc.scrapy.org/en/latest/topics/commands.html#std:command-version).

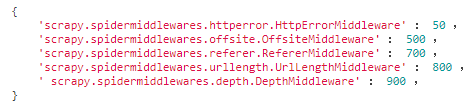
### SPIDER\_MIDDLEWARES

默认：： {}

一个包含您的项目中启用的蜘蛛中间件的字典，以及他们的订单。有关更多信息，请参阅[激活蜘蛛中间件](https://doc.scrapy.org/en/latest/topics/spider-middleware.html#topics-spider-middleware-setting)。

### SPIDER\_MIDDLEWARES\_BASE

默认：



包含Scrapy中默认启用的蜘蛛中间件的字典及其订单。低订单靠近引擎，高订单靠近蜘蛛。有关更多信息，请参阅[激活蜘蛛中间件](https://doc.scrapy.org/en/latest/topics/spider-middleware.html#topics-spider-middleware-setting)。

### SPIDER\_MODULES

默认： []

Scrapy将寻找蜘蛛的模块列表。

例：



### STATS\_CLASS

默认： 'scrapy.statscollectors.MemoryStatsCollector'

用于收集统计信息的类，他们必须实现 [Stats Collector API](https://doc.scrapy.org/en/latest/topics/api.html#topics-api-stats)。

### STATS\_DUMP

默认： True

一旦蜘蛛结束，将[Scrapy统计信息](https://doc.scrapy.org/en/latest/topics/stats.html#topics-stats)（Scrapy日志）转储。

欲了解更多信息，请参阅：[Stats Collection](https://doc.scrapy.org/en/latest/topics/stats.html#topics-stats)。

### STATSMAILER\_RCPTS

默认:( []空列表）

在蜘蛛完成刮取后发送Scrapy数据。查看 **StatsMailer**更多信息。

### TELNETCONSOLE\_ENABLED

默认： True

一个布尔值，指定是否 启用[telnet控制台](https://doc.scrapy.org/en/latest/topics/telnetconsole.html#topics-telnetconsole)（只要其分机也启用）。

### TELNETCONSOLE\_PORT

默认： [6023, 6073]

用于telnet控制台的端口范围。如果设置为None或0，则使用动态分配的端口。有关更多信息，请参阅 [Telnet控制台](https://doc.scrapy.org/en/latest/topics/telnetconsole.html#topics-telnetconsole)。

### TEMPLATES\_DIR

默认值：templatesdir在scrapy模块中

使用[**startproject**](https://doc.scrapy.org/en/latest/topics/commands.html#std:command-startproject)命令创建新项目时使用命令和新蜘蛛创建新模板的目录 [**genspider**](https://doc.scrapy.org/en/latest/topics/commands.html#std:command-genspider) 。

项目名称不得与project子目录中的自定义文件或目录的名称冲突。

### URLLENGTH\_LIMIT

默认： 2083

范围： spidermiddlewares.urllength

允许抓取的网址的最大网址长度。有关此设置的默认值的更多信息，请参阅：[https](https://boutell.com/newfaq/misc/urllength.html)：[//boutell.com/newfaq/misc/urllength.html](https://boutell.com/newfaq/misc/urllength.html)

### USER\_AGENT

默认： "Scrapy/VERSION (+https://scrapy.org)"

抓取时使用的默认User-Agent，除非被覆盖。

### 设置记录在别处：

以下设置在其他地方有记录，请检查每个特定情况以了解如何启用和使用它们。

* [AJAXCRAWL\_ENABLED](https://doc.scrapy.org/en/latest/topics/downloader-middleware.html#std:setting-AJAXCRAWL_ENABLED)
* [AUTOTHROTTLE\_DEBUG](https://doc.scrapy.org/en/latest/topics/autothrottle.html#std:setting-AUTOTHROTTLE_DEBUG)
* [AUTOTHROTTLE\_ENABLED](https://doc.scrapy.org/en/latest/topics/autothrottle.html#std:setting-AUTOTHROTTLE_ENABLED)
* [AUTOTHROTTLE\_MAX\_DELAY](https://doc.scrapy.org/en/latest/topics/autothrottle.html#std:setting-AUTOTHROTTLE_MAX_DELAY)
* [AUTOTHROTTLE\_START\_DELAY](https://doc.scrapy.org/en/latest/topics/autothrottle.html#std:setting-AUTOTHROTTLE_START_DELAY)
* [AUTOTHROTTLE\_TARGET\_CONCURRENCY](https://doc.scrapy.org/en/latest/topics/autothrottle.html#std:setting-AUTOTHROTTLE_TARGET_CONCURRENCY)
* [CLOSESPIDER\_ERRORCOUNT](https://doc.scrapy.org/en/latest/topics/extensions.html#std:setting-CLOSESPIDER_ERRORCOUNT)
* [CLOSESPIDER\_ITEMCOUNT](https://doc.scrapy.org/en/latest/topics/extensions.html#std:setting-CLOSESPIDER_ITEMCOUNT)
* [CLOSESPIDER\_PAGECOUNT](https://doc.scrapy.org/en/latest/topics/extensions.html#std:setting-CLOSESPIDER_PAGECOUNT)
* [CLOSESPIDER\_TIMEOUT](https://doc.scrapy.org/en/latest/topics/extensions.html#std:setting-CLOSESPIDER_TIMEOUT)
* [COMMANDS\_MODULE](https://doc.scrapy.org/en/latest/topics/commands.html#std:setting-COMMANDS_MODULE)
* [COMPRESSION\_ENABLED](https://doc.scrapy.org/en/latest/topics/downloader-middleware.html#std:setting-COMPRESSION_ENABLED)
* [COOKIES\_DEBUG](https://doc.scrapy.org/en/latest/topics/downloader-middleware.html#std:setting-COOKIES_DEBUG)
* [COOKIES\_ENABLED](https://doc.scrapy.org/en/latest/topics/downloader-middleware.html#std:setting-COOKIES_ENABLED)
* [FEED\_EXPORTERS](https://doc.scrapy.org/en/latest/topics/feed-exports.html#std:setting-FEED_EXPORTERS)
* [FEED\_EXPORTERS\_BASE](https://doc.scrapy.org/en/latest/topics/feed-exports.html#std:setting-FEED_EXPORTERS_BASE)
* [FEED\_EXPORT\_ENCODING](https://doc.scrapy.org/en/latest/topics/feed-exports.html#std:setting-FEED_EXPORT_ENCODING)
* [FEED\_EXPORT\_FIELDS](https://doc.scrapy.org/en/latest/topics/feed-exports.html#std:setting-FEED_EXPORT_FIELDS)
* [FEED\_EXPORT\_INDENT](https://doc.scrapy.org/en/latest/topics/feed-exports.html#std:setting-FEED_EXPORT_INDENT)
* [FEED\_FORMAT](https://doc.scrapy.org/en/latest/topics/feed-exports.html#std:setting-FEED_FORMAT)
* [FEED\_STORAGES](https://doc.scrapy.org/en/latest/topics/feed-exports.html#std:setting-FEED_STORAGES)
* [FEED\_STORAGES\_BASE](https://doc.scrapy.org/en/latest/topics/feed-exports.html#std:setting-FEED_STORAGES_BASE)
* [FEED\_STORE\_EMPTY](https://doc.scrapy.org/en/latest/topics/feed-exports.html#std:setting-FEED_STORE_EMPTY)
* [FEED\_URI](https://doc.scrapy.org/en/latest/topics/feed-exports.html#std:setting-FEED_URI)
* [FILES\_EXPIRES](https://doc.scrapy.org/en/latest/topics/media-pipeline.html#std:setting-FILES_EXPIRES)
* [FILES\_RESULT\_FIELD](https://doc.scrapy.org/en/latest/topics/media-pipeline.html#std:setting-FILES_RESULT_FIELD)
* [FILES\_STORE](https://doc.scrapy.org/en/latest/topics/media-pipeline.html#std:setting-FILES_STORE)
* [FILES\_STORE\_S3\_ACL](https://doc.scrapy.org/en/latest/topics/media-pipeline.html#std:setting-FILES_STORE_S3_ACL)
* [FILES\_URLS\_FIELD](https://doc.scrapy.org/en/latest/topics/media-pipeline.html#std:setting-FILES_URLS_FIELD)
* [GCS\_PROJECT\_ID](https://doc.scrapy.org/en/latest/topics/media-pipeline.html#std:setting-GCS_PROJECT_ID)
* [HTTPCACHE\_ALWAYS\_STORE](https://doc.scrapy.org/en/latest/topics/downloader-middleware.html#std:setting-HTTPCACHE_ALWAYS_STORE)
* [HTTPCACHE\_DBM\_MODULE](https://doc.scrapy.org/en/latest/topics/downloader-middleware.html#std:setting-HTTPCACHE_DBM_MODULE)
* [HTTPCACHE\_DIR](https://doc.scrapy.org/en/latest/topics/downloader-middleware.html#std:setting-HTTPCACHE_DIR)
* [HTTPCACHE\_ENABLED](https://doc.scrapy.org/en/latest/topics/downloader-middleware.html#std:setting-HTTPCACHE_ENABLED)
* [HTTPCACHE\_EXPIRATION\_SECS](https://doc.scrapy.org/en/latest/topics/downloader-middleware.html#std:setting-HTTPCACHE_EXPIRATION_SECS)
* [HTTPCACHE\_GZIP](https://doc.scrapy.org/en/latest/topics/downloader-middleware.html#std:setting-HTTPCACHE_GZIP)
* [HTTPCACHE\_IGNORE\_HTTP\_CODES](https://doc.scrapy.org/en/latest/topics/downloader-middleware.html#std:setting-HTTPCACHE_IGNORE_HTTP_CODES)
* [HTTPCACHE\_IGNORE\_MISSING](https://doc.scrapy.org/en/latest/topics/downloader-middleware.html#std:setting-HTTPCACHE_IGNORE_MISSING)
* [HTTPCACHE\_IGNORE\_RESPONSE\_CACHE\_CONTROLS](https://doc.scrapy.org/en/latest/topics/downloader-middleware.html#std:setting-HTTPCACHE_IGNORE_RESPONSE_CACHE_CONTROLS)
* [HTTPCACHE\_IGNORE\_SCHEMES](https://doc.scrapy.org/en/latest/topics/downloader-middleware.html#std:setting-HTTPCACHE_IGNORE_SCHEMES)
* [HTTPCACHE\_POLICY](https://doc.scrapy.org/en/latest/topics/downloader-middleware.html#std:setting-HTTPCACHE_POLICY)
* [HTTPCACHE\_STORAGE](https://doc.scrapy.org/en/latest/topics/downloader-middleware.html#std:setting-HTTPCACHE_STORAGE)
* [HTTPERROR\_ALLOWED\_CODES](https://doc.scrapy.org/en/latest/topics/spider-middleware.html#std:setting-HTTPERROR_ALLOWED_CODES)
* [HTTPERROR\_ALLOW\_ALL](https://doc.scrapy.org/en/latest/topics/spider-middleware.html#std:setting-HTTPERROR_ALLOW_ALL)
* [HTTPPROXY\_AUTH\_ENCODING](https://doc.scrapy.org/en/latest/topics/downloader-middleware.html#std:setting-HTTPPROXY_AUTH_ENCODING)
* [HTTPPROXY\_ENABLED](https://doc.scrapy.org/en/latest/topics/downloader-middleware.html#std:setting-HTTPPROXY_ENABLED)
* [IMAGES\_EXPIRES](https://doc.scrapy.org/en/latest/topics/media-pipeline.html#std:setting-IMAGES_EXPIRES)
* [IMAGES\_MIN\_HEIGHT](https://doc.scrapy.org/en/latest/topics/media-pipeline.html#std:setting-IMAGES_MIN_HEIGHT)
* [IMAGES\_MIN\_WIDTH](https://doc.scrapy.org/en/latest/topics/media-pipeline.html#std:setting-IMAGES_MIN_WIDTH)
* [IMAGES\_RESULT\_FIELD](https://doc.scrapy.org/en/latest/topics/media-pipeline.html#std:setting-IMAGES_RESULT_FIELD)
* [IMAGES\_STORE](https://doc.scrapy.org/en/latest/topics/media-pipeline.html#std:setting-IMAGES_STORE)
* [IMAGES\_STORE\_S3\_ACL](https://doc.scrapy.org/en/latest/topics/media-pipeline.html#std:setting-IMAGES_STORE_S3_ACL)
* [IMAGES\_THUMBS](https://doc.scrapy.org/en/latest/topics/media-pipeline.html#std:setting-IMAGES_THUMBS)
* [IMAGES\_URLS\_FIELD](https://doc.scrapy.org/en/latest/topics/media-pipeline.html#std:setting-IMAGES_URLS_FIELD)
* [MAIL\_FROM](https://doc.scrapy.org/en/latest/topics/email.html#std:setting-MAIL_FROM)
* [MAIL\_HOST](https://doc.scrapy.org/en/latest/topics/email.html#std:setting-MAIL_HOST)
* [MAIL\_PASS](https://doc.scrapy.org/en/latest/topics/email.html#std:setting-MAIL_PASS)
* [MAIL\_PORT](https://doc.scrapy.org/en/latest/topics/email.html#std:setting-MAIL_PORT)
* [MAIL\_SSL](https://doc.scrapy.org/en/latest/topics/email.html#std:setting-MAIL_SSL)
* [MAIL\_TLS](https://doc.scrapy.org/en/latest/topics/email.html#std:setting-MAIL_TLS)
* [MAIL\_USER](https://doc.scrapy.org/en/latest/topics/email.html#std:setting-MAIL_USER)
* [MEDIA\_ALLOW\_REDIRECTS](https://doc.scrapy.org/en/latest/topics/media-pipeline.html#std:setting-MEDIA_ALLOW_REDIRECTS)
* [METAREFRESH\_ENABLED](https://doc.scrapy.org/en/latest/topics/downloader-middleware.html#std:setting-METAREFRESH_ENABLED)
* [METAREFRESH\_MAXDELAY](https://doc.scrapy.org/en/latest/topics/downloader-middleware.html#std:setting-METAREFRESH_MAXDELAY)
* [REDIRECT\_ENABLED](https://doc.scrapy.org/en/latest/topics/downloader-middleware.html#std:setting-REDIRECT_ENABLED)
* [REDIRECT\_MAX\_TIMES](https://doc.scrapy.org/en/latest/topics/downloader-middleware.html#std:setting-REDIRECT_MAX_TIMES)
* [REFERER\_ENABLED](https://doc.scrapy.org/en/latest/topics/spider-middleware.html#std:setting-REFERER_ENABLED)
* [REFERRER\_POLICY](https://doc.scrapy.org/en/latest/topics/spider-middleware.html#std:setting-REFERRER_POLICY)
* [RETRY\_ENABLED](https://doc.scrapy.org/en/latest/topics/downloader-middleware.html#std:setting-RETRY_ENABLED)
* [RETRY\_HTTP\_CODES](https://doc.scrapy.org/en/latest/topics/downloader-middleware.html#std:setting-RETRY_HTTP_CODES)
* [RETRY\_TIMES](https://doc.scrapy.org/en/latest/topics/downloader-middleware.html#std:setting-RETRY_TIMES)
* [TELNETCONSOLE\_HOST](https://doc.scrapy.org/en/latest/topics/telnetconsole.html#std:setting-TELNETCONSOLE_HOST)
* [TELNETCONSOLE\_PORT](https://doc.scrapy.org/en/latest/topics/telnetconsole.html#std:setting-TELNETCONSOLE_PORT)

[下一个](https://doc.scrapy.org/en/latest/topics/exceptions.html)[以前](https://doc.scrapy.org/en/latest/topics/link-extractors.html)

# 错误（Exceptions）

## 内置的错误参考

## 这里列出了所有的例外情况，包括在擦伤和他们的使用。

### DropItem

## Exception scrapy.exceptions.DropItem

## 项目更多信息见管道阶段必须提高的异常，以停止处理项目。更多信息见 [Item Pipeline](https://doc.scrapy.org/en/latest/topics/item-pipeline.html#topics-item-pipeline).

### CloseSpider

**Exception scrapy.exceptions.CloseSpider(reason='cancelled')**

这个异常可以从蜘蛛回调中提出来请求蜘蛛被关闭/停止。支持的参数：

|  |  |
| --- | --- |
| **参数：** | **reason** （*str*） - 关闭的原因 |

例如：



### DontCloseSpider

**exceptionscrapy.exceptions.DontCloseSpider**

这个异常可以在[**spider\_idle**](https://doc.scrapy.org/en/latest/topics/signals.html#std:signal-spider_idle)信号处理程序中引发，以防止蜘蛛被关闭。

### IgnoreRequest

**Exception scrapy.exceptions.IgnoreRequest**

计划程序或任何下载器中间件都会引发此异常，以指示请求应被忽略

### NotConfigured

**Exception scrapy.exceptions.NotConfigured**

某些组件可能会引发此异常，以表明它们将保持禁用状态。这些组件包括：

* 扩展
* 物品管道
* 下载中间件
* 蜘蛛中间件

必须在组件的\_\_init\_\_方法中引发异常。

### NotSupported

**exceptionscrapy.exceptions.NotSupported**

引发此异常以指示不受支持的功能。

# Logging

**注意**

**scrapy.log**已被弃用，并且支持显式调用Python标准日志记录。继续阅读以了解更多关于新日志记录系统的信息。

Scrapy使用[Python的内置日志记录系统](https://docs.python.org/3/library/logging.html)进行事件日志记录。我们将提供一些简单的示例来帮助您开始，但对于更高级的用例，强烈建议您仔细阅读其文档。

日志功能可以直接使用，并且可以通过[记录设置中](https://doc.scrapy.org/en/latest/topics/logging.html#topics-logging-settings)列出的Scrapy设置进行一定程度的[配置](https://doc.scrapy.org/en/latest/topics/logging.html#topics-logging-settings)。

Scrapy调用[**scrapy.utils.log.configure\_logging()**](https://doc.scrapy.org/en/latest/topics/logging.html#scrapy.utils.log.configure_logging)设置一些合理的默认值并处理这些设置[日志记录设置](https://doc.scrapy.org/en/latest/topics/logging.html#topics-logging-settings)运行命令时，因此我们建议您手动调用它，如果你从脚本中描述的运行Scrapy [从脚本运行Scrapy](https://doc.scrapy.org/en/latest/topics/practices.html#run-from-script)。

## 日志级别

Python的内建日志记录定义了5个不同的级别来指示给定日志消息的严重性。这里是标准的，按递减顺序列出：

1. logging.CRITICAL - 严重错误（最严重）
2. logging.ERROR - 常规错误
3. logging.WARNING - 用于警告消息
4. logging.INFO - 用于参考消息
5. logging.DEBUG - 用于调试消息（最低严重性）

## 如何记录消息

以下是如何使用logging.WARNING 级别记录消息的简单示例：



有任何标准5级发布日志消息的捷径，还有一个logging.log以给定级别为参数的通用方法。如果需要，最后一个例子可以改写为：



最重要的是，您可以创建不同的“记录器”来封装消息。（例如，通常的做法是为每个模块创建不同的记录器）。这些记录器可以独立配置，并且允许分层结构。

前面的示例在后台使用根记录器，这是所有消息传播到的顶级记录器（除非另有说明）。使用logging助手只是显式获取root logger的捷径，所以这也是最后一个片段的等价物：



您可以通过使用logging.getLogger函数获取其名称来使用其他记录器 ：



最后，您可以确保通过使用\_\_name\_\_填充了当前模块路径的变量为您正在处理的任何模块创建自定义记录器：



**也可以看看**

**模块日志，**[**HowTo**](https://docs.python.org/2/howto/logging.html)

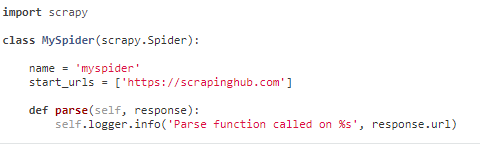
基本记录教程

**模块日志**[**记录器**](https://docs.python.org/2/library/logging.html#logger-objects)

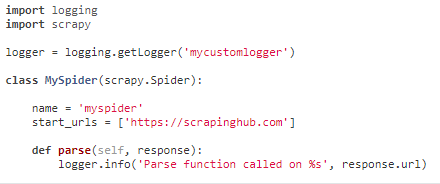
有关记录仪的进一步文件

## 从蜘蛛记录

Scrapy [**logger**](https://doc.scrapy.org/en/latest/topics/spiders.html#scrapy.spiders.Spider.logger)在每个Spider实例中都提供了一个可以像这样访问和使用的实例：



该记录器是使用Spider的名称创建的，但您可以使用任何您想要的自定义Python记录器。例如：



## 记录配置

记录器自己不管理如何显示通过它们发送的消息。对于这项任务，可以将不同的“处理程序”附加到任何记录程序实例，并将这些消息重定向到适当的目标，例如标准输出，文件，电子邮件等。

默认情况下，Scrapy根据下面的设置设置和配置根记录器的处理程序。

### 记录设置

这些设置可用于配置日志记录：

* [**LOG\_FILE**](https://doc.scrapy.org/en/latest/topics/settings.html#std:setting-LOG_FILE)
* [**LOG\_ENABLED**](https://doc.scrapy.org/en/latest/topics/settings.html#std:setting-LOG_ENABLED)
* [**LOG\_ENCODING**](https://doc.scrapy.org/en/latest/topics/settings.html#std:setting-LOG_ENCODING)
* [**LOG\_LEVEL**](https://doc.scrapy.org/en/latest/topics/settings.html#std:setting-LOG_LEVEL)
* [**LOG\_FORMAT**](https://doc.scrapy.org/en/latest/topics/settings.html#std:setting-LOG_FORMAT)
* [**LOG\_DATEFORMAT**](https://doc.scrapy.org/en/latest/topics/settings.html#std:setting-LOG_DATEFORMAT)
* [**LOG\_STDOUT**](https://doc.scrapy.org/en/latest/topics/settings.html#std:setting-LOG_STDOUT)
* [**LOG\_SHORT\_NAMES**](https://doc.scrapy.org/en/latest/topics/settings.html#std:setting-LOG_SHORT_NAMES)

第一对设置定义了日志消息的目的地。如果 [**LOG\_FILE**](https://doc.scrapy.org/en/latest/topics/settings.html#std:setting-LOG_FILE)已设置，通过根记录器发送的消息将被重定向到[**LOG\_FILE**](https://doc.scrapy.org/en/latest/topics/settings.html#std:setting-LOG_FILE)以编码 命名的文件[**LOG\_ENCODING**](https://doc.scrapy.org/en/latest/topics/settings.html#std:setting-LOG_ENCODING)。如果没有设置和[**LOG\_ENABLED**](https://doc.scrapy.org/en/latest/topics/settings.html#std:setting-LOG_ENABLED)是True，日志消息将显示在标准错误。最后，如果 [**LOG\_ENABLED**](https://doc.scrapy.org/en/latest/topics/settings.html#std:setting-LOG_ENABLED)是False，则不会有任何可见的日志输出。

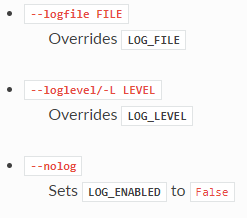
[**LOG\_LEVEL**](https://doc.scrapy.org/en/latest/topics/settings.html#std:setting-LOG_LEVEL)确定要显示的最低严重级别，那些严重程度较低的消息将被过滤掉。它的范围通过[日志级别中](https://doc.scrapy.org/en/latest/topics/logging.html#topics-logging-levels)列出的可能[级别](https://doc.scrapy.org/en/latest/topics/logging.html#topics-logging-levels)。

[**LOG\_FORMAT**](https://doc.scrapy.org/en/latest/topics/settings.html#std:setting-LOG_FORMAT)并[**LOG\_DATEFORMAT**](https://doc.scrapy.org/en/latest/topics/settings.html#std:setting-LOG_DATEFORMAT)指定用作所有消息布局的格式化字符串。这些字符串可以包含[日志的logrecord属性文档](https://docs.python.org/2/library/logging.html#logrecord-attributes)和 [日期时间的strftime和strptime指令中](https://docs.python.org/2/library/datetime.html#strftime-and-strptime-behavior) 分别列出的任何占位符。

如果[**LOG\_SHORT\_NAMES**](https://doc.scrapy.org/en/latest/topics/settings.html#std:setting-LOG_SHORT_NAMES)已设置，则日志将不会显示打印日志的scrapy组件。它在默认情况下是未设置的，因此日志包含负责该日志输出的scrapy组件。

### 命令行选项

有一些命令行参数可用于所有命令，您可以使用它们来覆盖有关日志记录的一些Scrapy设置。



**也可以看看**

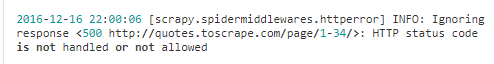
**模块**[**logging.handlers**](https://docs.python.org/2/library/logging.handlers.html)

有关可用处理程序的更多文档

### 高级定制

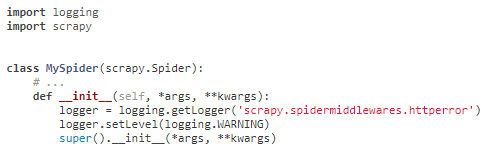
由于Scrapy使用stdlib日志记录模块，因此可以使用stdlib日志记录的所有功能自定义日志记录。

例如，假设您正在抓取一个返回许多HTTP 404和500响应的网站，并且想要隐藏所有这样的消息：



首先要注意的是一个记录器名称 - 它在括号内： [scrapy.spidermiddlewares.httperror]。如果你得到的只是[scrapy]那么 [**LOG\_SHORT\_NAMES**](https://doc.scrapy.org/en/latest/topics/settings.html#std:setting-LOG_SHORT_NAMES)很可能设置为True; 将其设置为False并重新运行爬网。

接下来，我们可以看到该消息具有INFO级别。要隐藏它，我们应该设置scrapy.spidermiddlewares.httperror 高于INFO的日志级别; INFO之后的下一个级别是WARNING。它可以通过例如蜘蛛的\_\_init\_\_方法完成：



如果你再次运行这个蜘蛛，那么来自scrapy.spidermiddlewares.httperror记录器的INFO消息 将会消失。

## scrapy.utils.log模块

**scrapy.utils.log.configure\_logging(settings=None, install\_root\_handler=True)**

初始化Scrapy的日志记录默认值。

|  |  |
| --- | --- |
| **参数：** | * **settings** (dict, [**Settings**](https://doc.scrapy.org/en/latest/topics/api.html#scrapy.settings.Settings) object or **None**)  - 用于为根记录器创建和配置处理程序的设置（默认值：无）。 * **install\_root\_handler**（bool） - 是否安装根日志记录处理程序（默认值：True） |

这个功能确实：

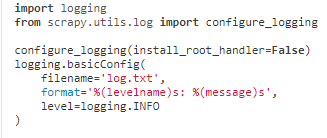
* 通过Python标准日志记录路由警告和扭曲日志记录
* 分别为Scrapy和Twisted记录器分配DEBUG和ERROR级别
* 如果LOG\_STDOUT设置为True，则路由stdout以记录日志

何时**install\_root\_handler**为True（默认）时，此功能还会根据给定设置为根记录器创建处理程序（请参阅[记录设置](https://doc.scrapy.org/en/latest/topics/logging.html#topics-logging-settings)）。您可以使用**settings**参数覆盖默认选项。当**settings**为空或无时，使用默认值。

**configure\_logging**在使用Scrapy命令时会自动调用，但在运行自定义脚本时需要明确调用。在这种情况下，它的使用不是必需的，但建议。

如果您计划自己配置处理程序，仍建议您调用此函数，并传递install\_root\_handler = False。请记住，在这种情况下，默认情况下不会设置任何日志输出。

为了让您开始手动配置日志记录的输出，您可以使用 [logging.basicConfig（）](https://docs.python.org/2/library/logging.html#logging.basicConfig)来设置基本的根处理程序。这是如何将**INFO**消息重定向到更高文件的示例：



请参阅[脚本中的运行Scrapy，以](https://doc.scrapy.org/en/latest/topics/practices.html#run-from-script)获取更多关于如何使用Scrapy的详细信息。

# Stats Collection（统计收集）

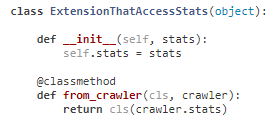
Scrapy提供了一个方便的功能，以键/值的形式收集统计数据，其中值通常是计数器。该工具被称为Stats Collector，可以通过[Crawler API](https://doc.scrapy.org/en/latest/topics/api.html#topics-api-crawler)的[**stats**](https://doc.scrapy.org/en/latest/topics/api.html#scrapy.crawler.Crawler.stats) 属性进行访问，如[Common Stats Collector使用](https://doc.scrapy.org/en/latest/topics/stats.html#topics-stats-usecases)部分中的示例所示。

但是，统计收集器始终可用，因此无论是否启用统计信息收集，您都可以始终将其导入模块并使用其API（用于增加或设置新的统计信息键）。如果它被禁用，API仍然可以工作，但它不会收集任何东西。这是为了简化统计收集器的使用：你应该花费不超过一行代码来收集你的蜘蛛，Scrapy扩展，或者你使用Stats Collector的任何代码。

Stats Collector的另一个功能是它非常高效（启用时），并且在禁用时非常高效（几乎不明显）。

统计收集器为每个打开的蜘蛛保持一个统计表，当蜘蛛打开时它会自动打开，当蜘蛛关闭时会自动打开。

## Common Stats Collector使用

通过[**stats**](https://doc.scrapy.org/en/latest/topics/api.html#scrapy.crawler.Crawler.stats) 属性访问统计收集器。以下是访问统计信息的扩展示例：

Set stat value:



增量属性值：



仅在大于以前时才设置统计值：



仅在低于之前设置统计值：



获取统计值：



获取所有统计信息：



## 可用的统计收集器

除了基本功能外**StatsCollector**，Scrapy中还有其他可用的Stats Collector扩展了基本的Stats Collector。您可以通过[**STATS\_CLASS**](https://doc.scrapy.org/en/latest/topics/settings.html#std:setting-STATS_CLASS)设置选择要使用哪个统计收集器。使用的默认Stats Collector是**MemoryStatsCollector**。

### MemoryStatsCollector

**Class scrapy.statscollectors.MemoryStatsCollector**

一个简单的统计收集器，在关闭后保存最后一次抓取的内容（对于每个蜘蛛）在内存中的统计信息。统计信息可以通过[**spider\_stats**](https://doc.scrapy.org/en/latest/topics/stats.html#scrapy.statscollectors.MemoryStatsCollector.spider_stats)属性进行访问，该属性是一个由蜘蛛域名键入的字典。

这是Scrapy中使用的默认Stats Collector。

**spider\_stats**

一个字典（以蜘蛛名字为键）的字典，包含每个蜘蛛最后一次扫描的统计数据。

### DummyStatsCollector

**classscrapy.statscollectors.DummyStatsCollector**

一个统计收集器什么都不做，但效率很高（因为它什么都不做）。此状态收集器可以通过设置进行[**STATS\_CLASS**](https://doc.scrapy.org/en/latest/topics/settings.html#std:setting-STATS_CLASS) 设置，以禁用状态收集以提高性能。但是，与其他Scrapy工作负载（如解析页面）相比，统计信息收集的性能损失通常很小。

# 发送电子邮件

尽管Python通过[smtplib](https://docs.python.org/2/library/smtplib.html) 库使发送电子邮件相对容易，但Scrapy提供了自己的发送电子邮件的工具，这些邮件非常易于使用，并且使用[Twisted non-blocking IO](https://twistedmatrix.com/documents/current/core/howto/defer-intro.html)来实现，以避免干扰非阻塞IO履带。它还为发送附件提供了一个简单的API，并且使用一些[设置](https://doc.scrapy.org/en/latest/topics/email.html#topics-email-settings)很容易进行配置 。

## 快速示例

有两种方法来实例化邮件发件人。你可以使用标准的构造函数来实例化它：



或者你可以实例化它传递一个Scrapy设置对象，它将遵循这些[设置](https://doc.scrapy.org/en/latest/topics/email.html#topics-email-settings)：



以下是如何使用它发送电子邮件（无附件）：

mailer**.**send(to**=**["someone@example.com"], subject**=**"Some subject", body**=**"Some body", cc**=**["another@example.com"])

## MailSender类参考

MailSender是用于从Scrapy发送电子邮件的首选类，因为它使用[Twisted非阻塞IO](https://twistedmatrix.com/documents/current/core/howto/defer-intro.html)，就像框架的其余部分一样。

**Class scrapy.mail.MailSender（smtphost = None，mailfrom = None，smtpuser = None，smtppass = None，smtpport = None ）**

|  |  |
| --- | --- |
| **参数：** | * **smtphost**（str 或bytes） - 用于发送电子邮件的SMTP主机。如果省略，[**MAIL\_HOST**](https://doc.scrapy.org/en/latest/topics/email.html#std:setting-MAIL_HOST)将使用该 设置。 * **mailfrom**（str） - 用于发送电子邮件的地址（在**From:**标题中）。如果省略，[**MAIL\_FROM**](https://doc.scrapy.org/en/latest/topics/email.html#std:setting-MAIL_FROM)将使用该设置。 * **smtpuser** - SMTP用户。如果省略，[**MAIL\_USER**](https://doc.scrapy.org/en/latest/topics/email.html#std:setting-MAIL_USER) 将使用该设置。如果没有给出，则不会执行SMTP认证。 * **smtppass**（str 或bytes） - 用于验证的SMTP通行证。 * **smtpport**（int） - 要连接到的SMTP端口 * **smtptls**（boolean） - 使用SMTP STARTTLS强制执行 * **smtpssl**（boolean） - 使用安全的SSL连接强制执行 |

**classmethodfrom\_settings（设置）**

使用Scrapy设置对象进行实例化，这将遵循 [这些Scrapy设置](https://doc.scrapy.org/en/latest/topics/email.html#topics-email-settings)。

|  |  |
| --- | --- |
| **参数：** | **设置**（[**scrapy.settings.Settings**](https://doc.scrapy.org/en/latest/topics/api.html#scrapy.settings.Settings)对象） - 电子邮件收件人 |

**send（to，subject，body，cc = None，attachs =（），mimetype ='text / plain'，charset = None ）**

发送电子邮件给给定的收件人。

|  |  |
| --- | --- |
| **参数：** | * **to** (str or list of str)  - 电子邮件收件人 * **subject** (str) - 电子邮件的主题 * **cc**（str 或str 列表） - 给CC的电子邮件 * **body**（str） - 电子邮件正文 * **attachs** (iterable) - 一个元组的迭代，其中 是一个字符串，其名称将出现在电子邮件的附件中，是附件的MIME类型，是一个可读文件对象，包含附件的内容**(attach\_name, mimetype, file\_object)attach\_namemimetypefile\_object** * **mimetype**（str） - 电子邮件的MIME类型 * **charset** (str) - 用于电子邮件内容的字符编码 |

## 邮件设置

这些设置定义了[**MailSender**](https://doc.scrapy.org/en/latest/topics/email.html#scrapy.mail.MailSender) 类的默认构造函数值，并且可以用来在您的项目中配置电子邮件通知，而无需编写任何代码（用于那些使用的扩展和代码[**MailSender**](https://doc.scrapy.org/en/latest/topics/email.html#scrapy.mail.MailSender)）。

### MAIL\_FROM

默认： 'scrapy@localhost'

发件人电子邮件使用（From:标题）发送电子邮件。

### MAIL\_HOST

默认： 'localhost'

用于发送电子邮件的SMTP主机。

### MAIL\_PORT

默认： 25

SMTP端口用于发送电子邮件。

### MAIL\_USER

默认： None

用户用于SMTP验证。如果禁用，则不会执行SMTP验证。

### MAIL\_PASS

默认： None

用于SMTP验证的密码以及[**MAIL\_USER**](https://doc.scrapy.org/en/latest/topics/email.html#std:setting-MAIL_USER)。

### MAIL\_TLS

默认： False

强制使用STARTTLS。STARTTLS是一种采取现有不安全连接的方式，并使用SSL / TLS将其升级到安全连接。

### MAIL\_SSL

默认： False

使用SSL加密连接强制连接

# Telnet控制台

Scrapy带有一个内置的telnet控制台，用于检查和控制Scrapy运行过程。Telnet控制台只是Scrapy过程中运行的常规python shell，因此您可以从中完成任何操作。

Telnet控制台是默认启用的[内置Scrapy扩展](https://doc.scrapy.org/en/latest/topics/extensions.html#topics-extensions-ref)，但如果需要，也可以将其禁用。有关分机本身的更多信息，请参阅 [Telnet控制台扩展](https://doc.scrapy.org/en/latest/topics/extensions.html#topics-extensions-ref-telnetconsole)。

## 如何访问telnet控制台

Telnet控制台侦听[**TELNETCONSOLE\_PORT**](https://doc.scrapy.org/en/latest/topics/telnetconsole.html#std:setting-TELNETCONSOLE_PORT)设置中定义的TCP端口 ，默认为6023。要访问控制台，您需要输入：

telnet localhost 6023

>>>

您需要在Windows中默认安装的telnet程序，以及大多数Linux发行版。

## telnet控制台中的可用变量

telnet控制台就像在Scrapy过程中运行的常规Python shell一样，所以你可以做任何事情，包括导入新模块等。

但是，telnet控制台带有一些为了方便而定义的默认变量：

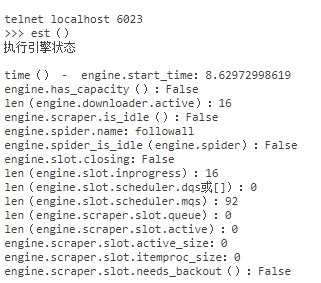


## Telnet控制台使用示例

以下是您可以使用telnet控制台执行的一些示例任务：

### 查看引擎状态

您可以使用est()Scrapy引擎的方法使用telnet控制台快速显示其状态：



### 暂停，恢复并停止Scrapy引擎

暂停：

telnet localhost 6023

>>> engine.pause（）

>>>

恢复：

telnet localhost 6023

>>> engine.unpause（）

>>>

停止：

telnet localhost 6023

>>> engine.stop（）

外部主机关闭连接。

## Telnet控制台信号

**scrapy.extensions.telnet.update\_telnet\_vars（telnet\_vars ）**

在Telnet控制台打开之前发送。您可以连接到该信号以添加，删除或更新telnet本地名称空间中可用的变量。为了做到这一点，你需要更新**telnet\_vars**处理程序中的 字典。

|  |  |
| --- | --- |
| **参数：** | **telnet\_vars**（dict） - telnet变量的字典 |

## Telnet设置

这些是控制telnet控制台行为的设置：

### TELNETCONSOLE\_PORT

默认： [6023, 6073]

用于telnet控制台的端口范围。如果设置为None或0，则使用动态分配的端口。

### TELNETCONSOLE\_HOST

默认： '127.0.0.1'

telnet控制台应该监听的接口

# Web Service

webservice已被移入一个单独的项目。

它的主办地址是：

<https://github.com/scrapy-plugins/scrapy-jsonrpc>

# 经常问的问题

## Scrapy与BeautifulSoup或lxml相比如何？

[BeautifulSoup](https://www.crummy.com/software/BeautifulSoup/)和[lxml](http://lxml.de/)是解析HTML和XML的库。Scrapy是一个用于编写抓取网站并从中提取数据的网络蜘蛛的应用程序框架。

Scrapy提供了一种内置的提取数据的机制（称为 [选择器](https://doc.scrapy.org/en/latest/topics/selectors.html#topics-selectors)），但如果您觉得使用它们更舒适，您可以轻松使用[BeautifulSoup](https://www.crummy.com/software/BeautifulSoup/) （或[lxml](http://lxml.de/)）。毕竟，他们只是解析可从任何Python代码导入和使用的库。

换句话说，比较[BeautifulSoup](https://www.crummy.com/software/BeautifulSoup/)（或[lxml](http://lxml.de/)）和Scrapy就像比较[jinja2](http://jinja.pocoo.org/)和[Django](https://www.djangoproject.com/)。

## 我可以在BeautifulSoup上使用Scrapy吗？

是的你可以。如所提到的[上面](https://doc.scrapy.org/en/latest/faq.html#faq-scrapy-bs-cmp)，[BeautifulSoup](https://www.crummy.com/software/BeautifulSoup/)可用于在Scrapy回调解析HTML响应。您只需将响应的主体提供给BeautifulSoup对象并从中提取所需的任何数据即可。

下面是一个使用BeautifulSoup API的示例蜘蛛，并将其lxml作为HTML解析器：



**注意**

BeautifulSoup支持多种HTML / XML解析器。请参阅[BeautifulSoup官方文档，](https://www.crummy.com/software/BeautifulSoup/bs4/doc/#specifying-the-parser-to-use)了解哪些可用。

## Scrapy支持哪些Python版本？

Scrapy在CPython（默认Python实现）和PyPy（从PyPy 5.9开始）下支持Python 2.7和Python 3.4。从Scrapy 0.20开始，Python 2.6的支持被删除了。Python 3支持已添加到Scrapy 1.1中。在Scrapy 1.4中添加了PyPy支持，在Scrapy 1.5中添加了PyPy3支持。

**注意**

对于Windows上的Python 3支持，建议使用[安装指南中概述的](https://doc.scrapy.org/en/latest/intro/install.html#intro-install-windows) Anaconda / Miniconda 。

## Scrapy是否从Django“偷”了X？

可能吧，但我们不喜欢那个词。我们认为[Django](https://www.djangoproject.com/)是一个很好的开源项目，也是一个可以遵循的例子，所以我们将它用作Scrapy的灵感来源。

我们相信，如果事情已经做得很好，就没有必要重塑它。这个概念除了是开源和自由软件的基础之外，不仅适用于软件，也适用于文档，程序，政策等。因此，我们不是自己去解决每个问题，而是选择从这些项目中复制想法已经解决了这些问题，并专注于我们需要解决的实际问题。

如果Scrapy充当其他项目的灵感，我们会很自豪。随意从我们窃取！

## Scrapy是否支持HTTP代理？

是。通过HTTP代理下载中间件提供了对HTTP代理的支持（从Scrapy 0.8开始）。看[**HttpProxyMiddleware**](https://doc.scrapy.org/en/latest/topics/downloader-middleware.html#scrapy.downloadermiddlewares.httpproxy.HttpProxyMiddleware)。

## 我怎样才能在不同的页面中使用属性来刮一个项目？

请参阅将[附加数据传递给回调函数](https://doc.scrapy.org/en/latest/topics/request-response.html#topics-request-response-ref-request-callback-arguments)。

## Scrapy崩溃：ImportError：没有名为win32api的模块

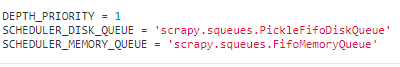
由于[此Twisted错误，](https://twistedmatrix.com/trac/ticket/3707)您需要安装[pywin32](https://sourceforge.net/projects/pywin32/)。

## 我如何模拟蜘蛛中的用户登录？

请参阅[使用FormRequest.from\_response（）模拟用户登录](https://doc.scrapy.org/en/latest/topics/request-response.html#topics-request-response-ref-request-userlogin)。

## Scrapy是以广度优先还是深度优先的顺序抓取？

默认情况下，Scrapy使用[LIFO](https://en.wikipedia.org/wiki/Stack_(abstract_data_type))队列来存储未决请求，这基本上意味着它按[DFO顺序进行](https://en.wikipedia.org/wiki/Depth-first_search)爬网。这个订单在大多数情况下更方便。如果您确实想要以真正的[BFO顺序](https://en.wikipedia.org/wiki/Breadth-first_search)进行爬网，您可以通过设置以下设置来完成此操作：



## 我的Scrapy爬虫有内存泄漏。我能做什么？

请参阅[调试内存泄漏](https://doc.scrapy.org/en/latest/topics/leaks.html#topics-leaks)。

另外，Python有一个内置的内存泄漏问题，这个问题在[没有泄漏的泄漏中](https://doc.scrapy.org/en/latest/topics/leaks.html#topics-leaks-without-leaks)有描述 。

## 我如何让Scrapy消耗更少的内存？

见上一个问题。

## 我可以在我的蜘蛛中使用基本HTTP身份验证吗？

是的，看[**HttpAuthMiddleware**](https://doc.scrapy.org/en/latest/topics/downloader-middleware.html#scrapy.downloadermiddlewares.httpauth.HttpAuthMiddleware)。

## 为什么Scrapy以英文而非我的母语下载页面？

尝试通过覆盖设置来更改默认的[Accept-Language](https://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html#sec14.4)请求标头 [**DEFAULT\_REQUEST\_HEADERS**](https://doc.scrapy.org/en/latest/topics/settings.html#std:setting-DEFAULT_REQUEST_HEADERS)。

## 我在哪里可以找到一些Scrapy项目的例子？

参见[示例](https://doc.scrapy.org/en/latest/intro/examples.html#intro-examples)。

## 我可以在不创建项目的情况下运行蜘蛛吗？

是。您可以使用该[**runspider**](https://doc.scrapy.org/en/latest/topics/commands.html#std:command-runspider)命令。例如，如果你有一个蜘蛛写入my\_spider.py文件，你可以运行它：



请参阅[**runspider**](https://doc.scrapy.org/en/latest/topics/commands.html#std:command-runspider)命令了解更多信息。

## 我收到“Filtered offsite request”消息。我该如何解决它们？

这些消息（使用DEBUG级别登录）并不一定意味着存在问题，因此您可能不需要修复它们。

这些消息由异地蜘蛛中间件引发，该中间件是一个蜘蛛中间件（默认情况下启用），其目的是将请求过滤到蜘蛛覆盖之外的域。

有关更多信息，请参阅： [**OffsiteMiddleware**](https://doc.scrapy.org/en/latest/topics/spider-middleware.html#scrapy.spidermiddlewares.offsite.OffsiteMiddleware)。

## 在生产中部署Scrapy爬虫的建议方式是什么？

请参阅[部署Spider](https://doc.scrapy.org/en/latest/topics/deploy.html#topics-deploy)。

## 我可以使用JSON进行大型出口吗？

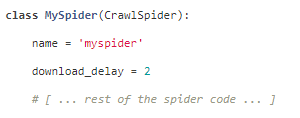
这取决于你的输出有多大。在 文档中看到[这个警告](https://doc.scrapy.org/en/latest/topics/exporters.html#json-with-large-data)[**JsonItemExporter**](https://doc.scrapy.org/en/latest/topics/exporters.html#scrapy.exporters.JsonItemExporter)。

## 我可以从信号处理程序返回（扭曲）延迟吗？

一些信号支持从处理程序返回延迟，其他信号不支持。看到[内置的信号参考](https://doc.scrapy.org/en/latest/topics/signals.html#topics-signals-ref)知道哪些。

## 响应状态码999的含义是什么？

999是雅虎网站用来限制请求的自定义响应状态代码。尝试通过2在蜘蛛中使用下载延迟（或更高）来减慢爬行速度：



或者通过设置在您的项目中设置全局下载延迟 [**DOWNLOAD\_DELAY**](https://doc.scrapy.org/en/latest/topics/settings.html#std:setting-DOWNLOAD_DELAY)。

## 我可以pdb.set\_trace()从我的蜘蛛打电话来调试它们吗？

是的，但你也可以使用Scrapy shell，它可以让你快速分析（甚至修改）蜘蛛正在处理的响应，这通常比普通的更有用pdb.set\_trace()。

欲了解更多信息，请参阅[从蜘蛛调用shell来检查响应](https://doc.scrapy.org/en/latest/topics/shell.html#topics-shell-inspect-response)。

## 最简单的方法将我所有的抓取的项目转储到JSON / CSV / XML文件中？

转储到JSON文件中：





有关更多信息，请参阅[Feed输出](https://doc.scrapy.org/en/latest/topics/feed-exports.html#topics-feed-exports)

## \_\_VIEWSTATE在某些形式中使用这个巨大的神秘参数是什么？

该\_\_VIEWSTATE参数用于使用ASP.NET / VB.NET构建的网站。有关它如何工作的更多信息，请参阅[此页面](http://search.cpan.org/~ecarroll/HTML-TreeBuilderX-ASP_NET-0.09/lib/HTML/TreeBuilderX/ASP_NET.pm)。此外，这里是一个[蜘蛛抓取](https://github.com/AmbientLighter/rpn-fas/blob/master/fas/spiders/rnp.py) 这些网站之一的[例子](https://github.com/AmbientLighter/rpn-fas/blob/master/fas/spiders/rnp.py)。

## 解析大型XML / CSV数据馈送的最佳方式是什么？

使用XPath选择器解析大型提要可能会产生问题，因为它们需要在内存中构建整个提要的DOM，而且这可能非常缓慢并且会消耗大量内存。

为了避免在内存中一次解析所有的全部饲料原料，可以使用的功能xmliter和csviter从scrapy.utils.iterators 模块。事实上，这就是饲料蜘蛛（见[蜘蛛](https://doc.scrapy.org/en/latest/topics/spiders.html#topics-spiders)）在封面下使用的。

## Scrapy会自动管理cookie吗？

是的，Scrapy接收并跟踪由服务器发送的cookies，并将它们发回后续请求，就像任何常规的Web浏览器一样。

欲了解更多信息，请参阅[请求和响应](https://doc.scrapy.org/en/latest/topics/request-response.html#topics-request-response)和[CookiesMiddleware](https://doc.scrapy.org/en/latest/topics/downloader-middleware.html#cookies-mw)。

## 如何查看Scrapy发送和接收的Cookie？

启用[**COOKIES\_DEBUG**](https://doc.scrapy.org/en/latest/topics/downloader-middleware.html#std:setting-COOKIES_DEBUG)设置。

## 我如何指导蜘蛛自行停止？

[**CloseSpider**](https://doc.scrapy.org/en/latest/topics/exceptions.html#scrapy.exceptions.CloseSpider)从回调中引发异常。有关更多信息，请参阅：[**CloseSpider**](https://doc.scrapy.org/en/latest/topics/exceptions.html#scrapy.exceptions.CloseSpider)。

## 我如何防止我的Scrapy bot被禁止？

请参阅[避免遭到禁止](https://doc.scrapy.org/en/latest/topics/practices.html#bans)。

## 我应该使用蜘蛛参数还是设置来配置我的蜘蛛？

这两种[蜘蛛的参数](https://doc.scrapy.org/en/latest/topics/spiders.html#spiderargs)和[设置，](https://doc.scrapy.org/en/latest/topics/settings.html#topics-settings) 可以用于配置您的蜘蛛。没有严格的规则要求使用其中一个或另一个，但设置更适合参数，一旦设置，参数变化不大，而蜘蛛参数要更频繁地更改，即使在每个蜘蛛运行中，有时也是如此所需的蜘蛛运行（例如，设置蜘蛛的起始url）。

举一个例子来说明，假设你有一个蜘蛛需要登录到一个站点来抓取数据，而你只想从网站的某个部分（每次都有所不同）中抓取数据。在这种情况下，登录的凭据将是设置，而要删除的部分的URL将是一个蜘蛛参数。

## 我刮了一个XML文档，我的XPath选择器不返回任何项目

您可能需要删除名称空间。请参阅[除去名称空间](https://doc.scrapy.org/en/latest/topics/selectors.html#removing-namespaces)。

# Debugging Spiders

本文档介绍了调试蜘蛛程序的最常用技术。考虑下面的scrapy spider：

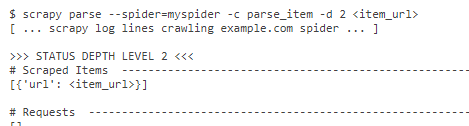


基本上这是一个简单的蜘蛛分析两页的项目（start\_urls）。项目也有详细信息页面，其中包含更多信息，因此我们使用meta功能[**Request**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.Request)来传递部分填充的项目。

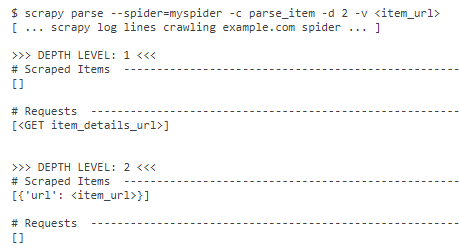
## 解析命令

检查蜘蛛输出的最基本的方法是使用该 [**parse**](https://doc.scrapy.org/en/latest/topics/commands.html#std:command-parse)命令。它允许在方法级别检查蜘蛛不同部分的行为。它具有使用灵活和简单的优点，但不允许在方法内调试代码。

为了查看从特定网址获取的项目：



使用--verbose或-v选项，我们可以看到每个深度级别的状态：



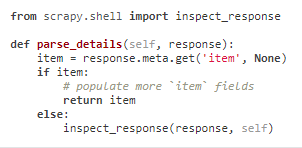
检查从单个start\_url中删除的项目也可以使用以下方法轻松实现：



## Scrapy Shell

虽然该[**parse**](https://doc.scrapy.org/en/latest/topics/commands.html#std:command-parse)命令对于检查蜘蛛的行为非常有用，但除了显示收到的响应和输出外，检查回调内部会发生什么也没有多大帮助。parse\_details有时没有收到任何项目时如何调试情况 ？

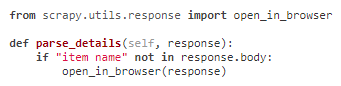
幸运的[**shell**](https://doc.scrapy.org/en/latest/topics/commands.html#std:command-shell)是，在这种情况下，这是你的面包和黄油（请参阅 [从蜘蛛中调用壳来检查响应](https://doc.scrapy.org/en/latest/topics/shell.html#topics-shell-inspect-response)）：



另请参阅：[从蜘蛛调用外壳来检查响应](https://doc.scrapy.org/en/latest/topics/shell.html#topics-shell-inspect-response)。

## 在浏览器中打开

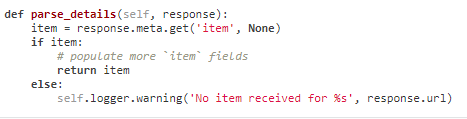
有时候你只想看看某个响应在浏览器中的外观如何，你可以使用该open\_in\_browser功能。这里是你如何使用它的例子：



open\_in\_browser将打开一个浏览器，其中包含Scrapy在此时收到的响应，调整[基本标记](https://www.w3schools.com/tags/tag_base.asp)以便正确显示图像和样式。

## 记录

日志记录是获取有关您的蜘蛛跑的信息的另一个有用的选项。虽然不是那么方便，但它带来的好处是日志在以后的所有运行中都可用，如果它们再次需要：



有关更多信息，请查看[日志记录](https://doc.scrapy.org/en/latest/topics/logging.html#topics-logging)部分。

# 蜘蛛合同

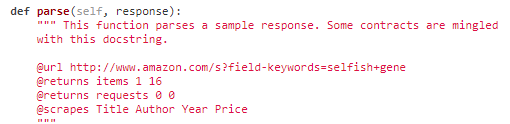
新版本0.15。

**注意**

这是一项新功能（在Scrapy 0.15中引入），可能会受到次要功能/ API更新。检查[发布说明](https://doc.scrapy.org/en/latest/news.html#news)以通知更新。

测试蜘蛛可能变得特别烦人，虽然没有什么能够阻止你编写单元测试，但任务很快就会变得麻烦。Scrapy提供了一种通过契约来测试蜘蛛的综合方法。

这允许您通过对示例url进行硬编码来测试蜘蛛的每个回调，并检查回调处理响应的各种约束。每个合约都以一个@和包含在文档字符串前面。看下面的例子：



此回调使用三个内置合约进行测试：

**Class scrapy.contracts.default.UrlContract**

此契约（**@url**）设置了检查此蜘蛛的其他合同条件时使用的示例url。该合同是强制性的。运行检查时忽略所有缺少此合同的回调：

@url 网址

**Class scrapy.contracts.default.ReturnsContract**

此契约（**@returns**）为蜘蛛所返回的项目和请求设置了下限和上限。上限是可选的：

@returns 项目（S ）**|** request （s ） [ min [ max ]]

**Class scrapy.contracts.default.ScrapesContract**

这个contract（**@scrapes**）检查回调函数返回的所有条目都有指定的字段：

@scrapes field\_1 field\_2 **...**

使用该[**check**](https://doc.scrapy.org/en/latest/topics/commands.html#std:command-check)命令来运行合同检查。

## 自定义合同

如果您发现需要比内置scrapy合约更多的权力，您可以使用以下[**SPIDER\_CONTRACTS**](https://doc.scrapy.org/en/latest/topics/settings.html#std:setting-SPIDER_CONTRACTS)设置在项目中创建和加载自己的合同 ：



每份合同必须继承[**scrapy.contracts.Contract**](https://doc.scrapy.org/en/latest/topics/contracts.html#scrapy.contracts.Contract)并可以覆盖三种方法：

**Class scrapy.contracts.Contract(method, \*args)**

|  |  |
| --- | --- |
| **参数：** | * **method** (function) - 合约所关联的回调函数 * **args**（[*list*](https://doc.scrapy.org/en/latest/topics/api.html#scrapy.loader.SpiderLoader.list)） - 传递给docstring的参数列表（空格分隔） |

**adjust\_request\_args(args)**

这会接收一个**dict**包含[**Request**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.Request)对象默认参数的参数。必须返回相同或修改后的版本。

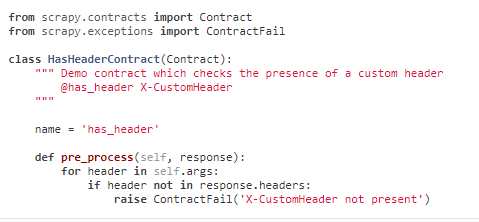
**pre\_process(response)**

这允许在传递给回调之前，对从采样请求接收到的响应进行各种检查。

**post\_process(output)**

这允许处理回调的输出。在传递给此钩子之前，迭代器会进行转换。

这里是一个演示合同，它检查收到的响应中是否存在自定义标题。**scrapy.exceptions.ContractFail**为了获得漂亮的打印失败而升起：



# 常见做法

本节介绍使用Scrapy时的常见做法。这些是涵盖很多主题的东西，并且通常不属于任何其他特定部分。

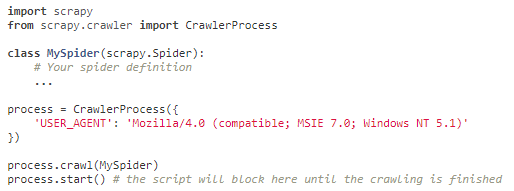
## 从脚本运行Scrapy

您可以使用[API](https://doc.scrapy.org/en/latest/topics/api.html#topics-api)从脚本运行Scrapy，而不是通过运行Scrapy的典型方式。scrapy crawl

请记住，Scrapy建立在Twisted异步网络库之上，因此您需要在Twisted reactor中运行它。

你可以用来运行你的蜘蛛的第一个工具是 [**scrapy.crawler.CrawlerProcess**](https://doc.scrapy.org/en/latest/topics/api.html#scrapy.crawler.CrawlerProcess)。这个类将为你启动一个Twisted reactor，配置日志记录和设置关闭处理程序。这个类是所有Scrapy命令使用的类。

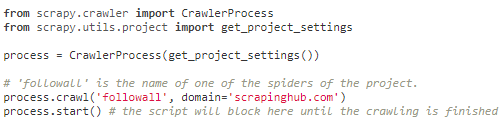
这里有一个例子展示了如何运行一个蜘蛛。



确保检查[**CrawlerProcess**](https://doc.scrapy.org/en/latest/topics/api.html#scrapy.crawler.CrawlerProcess)文档以熟悉其使用细节。

如果您在Scrapy项目中，则可以使用一些其他帮助程序在项目中导入这些组件。您可以自动导入您的蜘蛛传递他们的名字[**CrawlerProcess**](https://doc.scrapy.org/en/latest/topics/api.html#scrapy.crawler.CrawlerProcess)，并使用 您的项目设置get\_project\_settings获得一个[**Settings**](https://doc.scrapy.org/en/latest/topics/api.html#scrapy.settings.Settings)实例。

下面是如何使用[testspiders](https://github.com/scrapinghub/testspiders) 项目作为示例的工作示例。

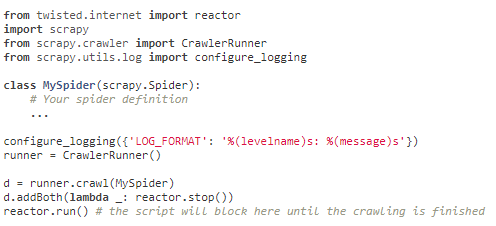


还有另一个Scrapy实用程序可以更好地控制抓取过程：[**scrapy.crawler.CrawlerRunner**](https://doc.scrapy.org/en/latest/topics/api.html#scrapy.crawler.CrawlerRunner)。这个类是一个简单的封装器，封装了一些简单的帮助器来运行多个爬虫，但它不会以任何方式启动或干扰现有的反应器。

使用这个类时，应该在安排好蜘蛛之后明确运行反应堆。建议您使用，[**CrawlerRunner**](https://doc.scrapy.org/en/latest/topics/api.html#scrapy.crawler.CrawlerRunner) 而不是[**CrawlerProcess**](https://doc.scrapy.org/en/latest/topics/api.html#scrapy.crawler.CrawlerProcess)如果您的应用程序已经使用Twisted，并且您想在同一个反应器中运行Scrapy。

请注意，在蜘蛛完成后，您还必须自行关闭Twisted反应堆。这可以通过向该[**CrawlerRunner.crawl**](https://doc.scrapy.org/en/latest/topics/api.html#scrapy.crawler.CrawlerRunner.crawl)方法返回的延迟添加回调来实现。

下面是它的使用示例，以及在*MySpider*完成运行后手动停止反应堆的回调。



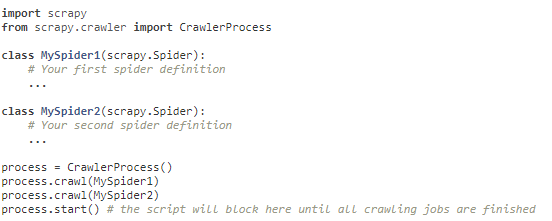
**See also**

[Twisted Reactor Overview](https://twistedmatrix.com/documents/current/core/howto/reactor-basics.html).

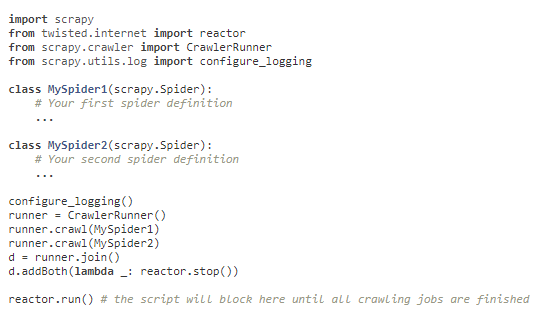
## 在同一个进程中运行多个蜘蛛

默认情况下，Scrapy在运行时为每个进程运行一个蜘蛛。但是，Scrapy支持使用[内部API](https://doc.scrapy.org/en/latest/topics/api.html#topics-api)在每个进程中运行多个蜘蛛。scrapy crawl

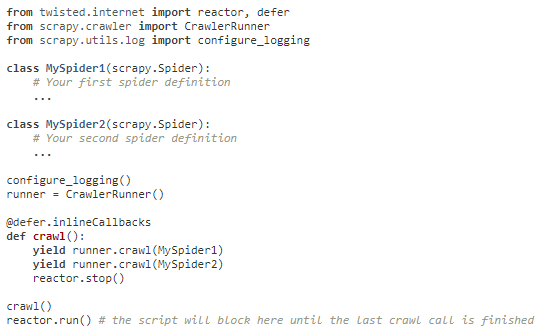
这是一个同时运行多个蜘蛛的例子：



同样的例子使用[**CrawlerRunner**](https://doc.scrapy.org/en/latest/topics/api.html#scrapy.crawler.CrawlerRunner)：



同样的例子，但通过链接延期来依次运行蜘蛛：



**也可以看看**

[从脚本运行Scrapy](https://doc.scrapy.org/en/latest/topics/practices.html#run-from-script)。

## 分布式爬网

Scrapy不提供以分布式（多服务器）方式运行爬网的任何内置工具。但是，有一些分发抓取的方式，根据您打算分发抓取的方式而有所不同。

如果你有很多蜘蛛，分配负载的明显方式是设置许多Scrapyd实例并在这些实例之间分发蜘蛛。

如果你想通过许多机器运行一个（大）蜘蛛，你通常会做的就是划分这些网址来爬行并将它们发送给每个单独的蜘蛛。这是一个具体的例子：

首先，您准备要抓取的网址列表并将其放入单独的文件/网址中：



然后你在三个不同的Scrapyd服务器上运行一个蜘蛛。蜘蛛会收到一个（蜘蛛）参数part与分区的数量爬行：



## 避免被禁止

有些网站采取了一些措施，以防止漫游器爬行，并具有不同程度的复杂性。避开这些措施可能非常困难且棘手，有时可能需要特殊的基础设施。如有疑问，请考虑联系[商业支持](https://scrapy.org/support/)。

在处理这些类型的网站时，请注意以下几点提示：

* 从浏览器中的众所周知的用户代理中旋转你的用户代理（谷歌周围以获得他们的列表）
* 禁用cookie（请参阅[**COOKIES\_ENABLED**](https://doc.scrapy.org/en/latest/topics/downloader-middleware.html#std:setting-COOKIES_ENABLED)），因为某些站点可能使用cookie来识别机器人行为
* 使用下载延迟（2或更高）。见[**DOWNLOAD\_DELAY**](https://doc.scrapy.org/en/latest/topics/settings.html#std:setting-DOWNLOAD_DELAY)设置。
* 如果可能，请使用[Google缓存](http://www.googleguide.com/cached_pages.html)来抓取页面，而不是直接击中网站
* 使用一个旋转的IP池。例如，免费的[Tor项目](https://www.torproject.org/)或付费服务，如[ProxyMesh](https://proxymesh.com/)。开源替代品是[scrapoxy](https://scrapoxy.io/)，一种超级代理，您可以将自己的代理附加到。
* 使用一个高度分布式的下载程序，它可以避免内部禁止，所以你可以专注于解析干净的页面。这种下载器的一个例子是 [Crawlera](https://scrapinghub.com/crawlera)

如果您仍然无法阻止您的机器人被禁止，请考虑联系 [商业支持](https://scrapy.org/support/)。

# 广泛的爬行

Scrapy默认设置针对特定网站进行了优化。这些网站通常由一个单一的Scrapy蜘蛛来处理，虽然这不是必需或不需要的（例如，有通用的蜘蛛可以处理任何给定的网站）。

除了这种“重点抓取”之外，还有另一种常见的抓取方式，涵盖了大量（可能无限）的域，并且仅受时间或其他任意约束的限制，而不是在域被抓取到完成时停​​止或当没有更多的要求执行。这些被称为“广泛抓取”，是搜索引擎使用的典型抓取工具。

这些是广泛爬行中常见的一些常见属性：

* 他们抓取很多域（通常是无界的）而不是一组特定的站点
* 他们不一定要抓取域名才能完成，因为这样做不切实际（或不可能），而是会按时间或爬网页数限制抓取
* 它们在逻辑上更简单（与具有许多提取规则的非常复杂的蜘蛛相反），因为数据通常在单独的阶段中进行后处理
* 他们同时抓取多个域，这使得他们可以通过不受任何特定站点约束的限制实现更快的爬网速度（每个站点缓慢爬取以尊重礼貌，但许多站点并行抓取）

如上所述，Scrapy默认设置针对重点抓取进行了优化，而不是广泛抓取。但是，由于其异步架构，Scrapy非常适合执行快速广泛爬网。本页总结了在使用Scrapy进行广泛爬网时需要注意的一些事项，以及Scrapy设置的具体建议以调整以实现高效的全面爬网。

## 提高并发性

并发性是并行处理的请求数。存在全局限制和每个域限制。

Scrapy中默认的全局并发限制不适合并行爬行许多不同的域，因此您需要增加它。增加多少将取决于您的爬虫有多少CPU可用。一个好的起点是100，但找出最好的方法是通过做一些试验，并确定Scrapy过程在哪些并发情况下使CPU受到限制。为了获得最佳性能，您应该选择CPU使用率为80-90％的并发性。

要增加全局并发使用：



## 增加Twisted IO线程池的最大大小

目前，Scrapy以使用线程池的阻塞方式进行DNS解析。在并发级别较高的情况下，爬网速度可能会很慢，甚至会导致DNS解析器超时失败。增加处理DNS查询的线程数的可能解决方案。DNS队列的处理速度将更快，加快建立连接和整体爬行。

要增加最大线程池大小，请使用：



## 设置你自己的DNS

如果您有多个爬网流程和单个中央DNS，它可能会像DNS服务器上的DoS攻击一样，从而导致整个网络变慢甚至阻塞您的计算机。为了避免这种情况，您可以使用本地缓存设置您自己的DNS服务器，并将其上传到OpenDNS或Verizon等大型DNS。

## 降低日志级别

在进行广泛的抓取时，您通常只对抓取的抓取率和发现的任何错误感兴趣。Scrapy在使用INFO日志级别时会报告这些统计信息 。为了节省CPU（以及日志存储需求），DEBUG在预生成大量广泛爬网时，不应使用日志级别。DEBUG开发您的（广泛）爬虫时使用级别可能没有问题。

设置日志级别使用：



## 禁用Cookie

除非你真的需要，否则禁用cookies 。进行大量抓取时（搜索引擎抓取工具忽略它们）通常不需要Cookie，它们通过节省一些CPU周期并减少Scrapy抓取工具的内存占用量来提高性能。

要禁用Cookie，请使用：



## 禁用重试

重试失败的HTTP请求可能会显着减慢抓取速度，尤其是当站点导致响应非常缓慢（或失败）时，这会导致超时错误，导致多次重试，从而不必要地阻止爬网程序的容量重用于其他域。

要禁用重试，请使用：



## 减少下载超时

除非您从非常慢的连接抓取（对于大量抓取，这不应该是这种情况），请减少下载超时，以便快速丢弃停滞的请求并释放处理下一个请求的容量。

要减少下载超时使用：



## 禁用重定向

考虑禁用重定向，除非您有兴趣关注它们。在进行广泛抓取时，通常会在以后的抓取中重新访问网站时保存重定向并解决它们。这也有助于保持每个爬网批次的请求数量不变，否则重定向循环可能会导致爬虫将任何特定域上的资源投入过多。

要禁用重定向，请使用：



## 启用“Ajax可抓取页面”的抓取

有些页面（根据2013年的经验数据，最高达1％）宣称自己是可以[抓取的](https://developers.google.com/webmasters/ajax-crawling/docs/getting-started)。这意味着他们提供纯粹的HTML版本的内容，通常只能通过AJAX使用。页面可以用两种方式表示它：

1. 通过#!在URL中使用- 这是默认的方式;
2. 通过使用特殊的元标记 - 这种方式是用于“主要”，“索引”网站页面。

Scrapy自动处理（1）; 处理（2）启用 [AjaxCrawlMiddleware](https://doc.scrapy.org/en/latest/topics/downloader-middleware.html#ajaxcrawl-middleware)：



在进行广泛的抓取时，通常抓取很多“索引”网页; AjaxCrawlMiddleware有助于正确地抓取它们。它默认关闭，因为它具有一定的性能开销，并且启用它进行重点抓取没有多大意义。

# 使用Firefox进行刮擦

以下是使用Firefox进行搜索的技巧和建议列表，以及一些有用的Firefox插件列表，以简化搜索过程。

## 注意检查实时浏览器DOM

由于Firefox插件在实时浏览器DOM上运行，因此检查页面源代码时实际看到的内容不是原始HTML代码，而是应用某些浏览器清理并执行Javascript代码后修改后的代码。特别是，Firefox以添加<tbody>元素到表格而闻名。另一方面，Scrapy不会修改原始页面HTML，因此如果您<tbody>在XPath表达式中使用，则无法提取任何数据。

因此，在使用Firefox和XPath时，请记住以下几点：

* 在检查DOM寻找要在Scrapy中使用的XPath时禁用Firefox Javascript
* 从不使用完整的XPath路径，使用基于属性相对和巧妙的人（如id，class，width等），或任何识别特征等 。contains(@href, 'image')
* <tbody>除非你真的知道你在做什么，否则不要在XPath表达式中包含元素

## 有用的Firefox加载项用于抓取

### Firebug

[Firebug](https://getfirebug.com/)是Web开发人员众所周知的工具，它也是非常有用的。特别是，当您需要构建用于提取数据的XPath时，它的[Inspect Element](https://www.youtube.com/watch?v=-pT_pDe54aA)功能非常方便，因为它允许您在将鼠标移到其上时查看每个页面元素的HTML代码。

有关如何在Scrapy中使用Firebug的详细指南，请参阅[使用Firebug进行](https://doc.scrapy.org/en/latest/topics/firebug.html#topics-firebug)拼写。

### XPather

[XPather](https://addons.mozilla.org/en-US/firefox/addon/xpather/)允许您直接在页面上测试XPath表达式。

### XPath检查器

[XPath Checker](https://addons.mozilla.org/en-US/firefox/addon/xpath-checker/)是另一个Firefox插件，用于在您的页面上测试XPath。

### 篡改数据

[Tamper Data](https://addons.mozilla.org/en-US/firefox/addon/tamper-data/)是一个Firefox插件，它允许您查看和修改Firefox发送的HTTP请求标头。Firebug也允许查看HTTP头，但不能修改它们。

### Firecookie

[Firecookie](https://addons.mozilla.org/en-US/firefox/addon/firecookie/)可以更轻松地查看和管理Cookie。您可以使用此扩展程序创建新的Cookie，删除现有的Cookie，查看当前网站的Cookie列表，管理Cookie权限等等。

[下一个](https://doc.scrapy.org/en/latest/topics/firebug.html)[以前](https://doc.scrapy.org/en/latest/topics/broad-crawls.html)

# 使用Firebug进行刮擦

**注意**

Google Directory，本指南中使用的示例网站已不再可用，因为它[已被Google关闭](https://searchenginewatch.com/sew/news/2096661/google-directory-shut)。本指南中的概念仍然有效。如果您想更新本指南以使用新的（工作）网站，您的贡献将超过欢迎！请参阅[贡献Scrapy](https://doc.scrapy.org/en/latest/contributing.html#topics-contributing) 获取有关如何这样做的信息。

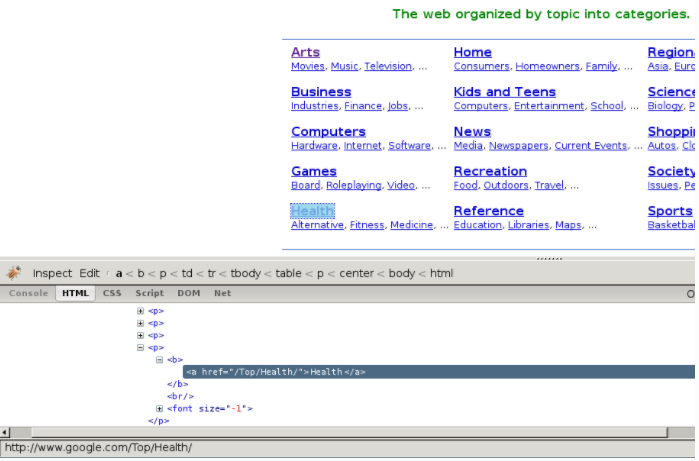
## 介绍

本文档解释了如何使用[Firebug](https://getfirebug.com/)（Firefox附加组件）使拼音过程更轻松，更有趣。对于其他有用的Firefox附加组件，请参阅 [有用的Firefox附加组件](https://doc.scrapy.org/en/latest/topics/firefox.html#topics-firefox-addons)。使用Firefox插件检查页面有一些注意事项，请参阅[警告以检查实时浏览器DOM](https://doc.scrapy.org/en/latest/topics/firefox.html#topics-firefox-livedom)。

在这个例子中，我们将演示如何使用[Firebug](https://getfirebug.com/)从 [Google Directory](http://directory.google.com/)中获取数据，该数据包含与本[教程中](https://doc.scrapy.org/en/latest/intro/tutorial.html#intro-tutorial)使用的[开放目录项目](http://www.dmoz.org/)相同的数据，但具有不同的面孔。

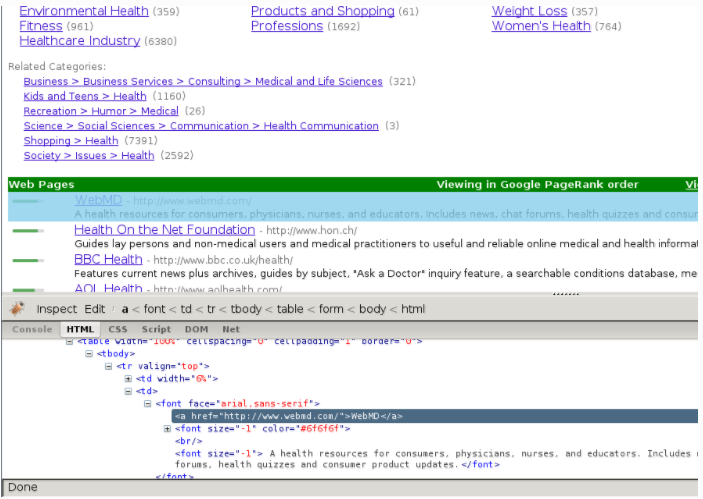
Firebug附带了一个非常有用的功能，名为[Inspect Element](https://www.youtube.com/watch?v=-pT_pDe54aA)，它允许您通过将鼠标悬停在不同的页面元素上来检查HTML代码。否则，您将不得不通过HTML主体手动搜索标签，这可能是一项非常繁琐的任务。

在以下屏幕截图中，您可以看到[Inspect Element](https://www.youtube.com/watch?v=-pT_pDe54aA)工具正在运行。



乍一看，我们可以看到目录按类别划分，这些类别也划分为子类别。

但是，看起来有更多的子类别比在这个页面中显示的子类别，所以我们会继续寻找：



如预期的那样，子类别包含指向其他子类别的链接，并且还链接到实际网站，这是目录的用途。

## 获取链接

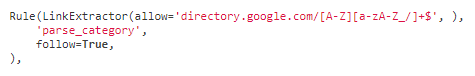
通过查看类别URL，我们可以看到它们共享一个模式：

<http://directory.google.com/Category/Subcategory/Another_Subcategory>

一旦我们知道这一点，我们就可以构建一个正则表达式来跟踪这些链接。例如，以下一个：

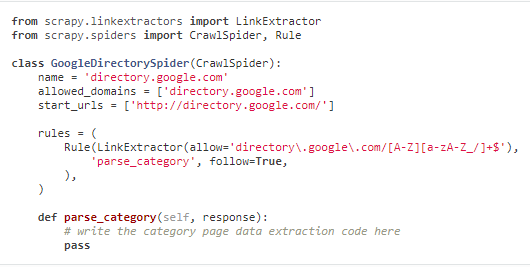


因此，基于该正则表达式，我们可以创建第一个爬网规则：



该[**Rule**](https://doc.scrapy.org/en/latest/topics/spiders.html#scrapy.spiders.Rule)对象指示 [**CrawlSpider**](https://doc.scrapy.org/en/latest/topics/spiders.html#scrapy.spiders.CrawlSpider)基于蜘蛛的如何遵循类别链接。parse\_category将成为蜘蛛的一种方法，它将处理并从这些页面提取数据。

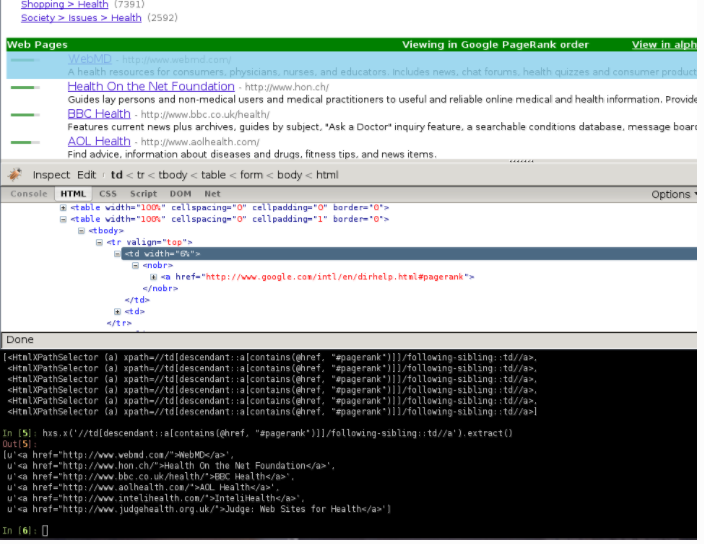
这就是蜘蛛看起来如此遥远的原因：



## 提取数据

现在我们要编写代码来从这些页面提取数据。

在Firebug的帮助下，我们将查看包含指向网站链接的某个页面（比如<http://directory.google.com/Top/Arts/Awards/>），并了解如何使用[选择器](https://doc.scrapy.org/en/latest/topics/selectors.html#topics-selectors)提取这些链接。我们还将使用[Scrapy shell](https://doc.scrapy.org/en/latest/topics/shell.html#topics-shell)来测试这些XPath，并确保它们按照我们的预期工作。



正如你可以看到，页面的标记是不是很描述：元素不包含id，class或者清楚地加以识别，所以我们将使用的排名吧作为参考点来选择数据的任何属性，当我们构建提取我们XPath的。

使用FireBug后，我们可以看到每个链接都在一个td标签内，它本身位于标签中，该tr标签还包含链接的排名栏（在另一个标签栏中td）。

所以我们可以选择排名栏，然后找到它的父母（the tr），最后，链接td（包含我们想要抓取的数据）。

这会导致以下XPath：

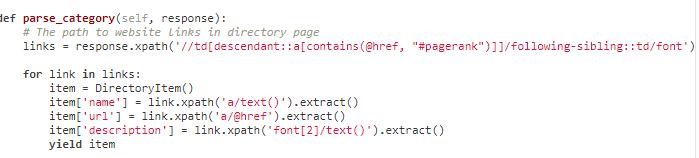


使用[Scrapy shell](https://doc.scrapy.org/en/latest/topics/shell.html#topics-shell)来测试这些复杂的XPath表达式并确保它们按预期工作非常重要。

基本上，该表达式将查找排名栏的td元素，然后选择td具有后代a元素的任何元素，该元素的href属性包含字符串#pagerank“

当然，这不是唯一的XPath，也许不是选择这些数据的更简单的方法。例如，另一种方法可能是找到任何font具有链接灰色的标签，

最后，我们可以编写我们的parse\_category()方法：



请注意，您可能会发现一些出现在Firebug中但不是原始HTML中的<tbody> 元素，例如元素的典型情况。

或者页面HTML源码中的Therefer可能会在Firebug上检查实时DOM

# 调试内存泄漏

在Scrapy中，Requests，Responses和Items等对象具有有限的生命周期：它们被创建，使用一段时间，并最终被销毁。

从所有这些对象中，Request可能是具有最长生命周期的请求，因为它在Scheduler队列中一直处于等待状态，直到需要处理它为止。欲了解更多信息请参阅[架构概述](https://doc.scrapy.org/en/latest/topics/architecture.html#topics-architecture)。

由于这些Scrapy对象具有（相当长的）生命周期，因此总是存在将它们累积在内存中而不正确释放它们并因此引起所谓的“内存泄漏”的风险。

为了帮助调试内存泄漏，Scrapy提供了一种内置机制来跟踪名为[trackref的](https://doc.scrapy.org/en/latest/topics/leaks.html#topics-leaks-trackrefs)对象引用，还可以使用称为[Guppy](https://doc.scrapy.org/en/latest/topics/leaks.html#topics-leaks-guppy)的第三方库进行更高级的内存调试（请参阅下面的更多信息）。这两种机制都必须在[Telnet控制台中使用](https://doc.scrapy.org/en/latest/topics/telnetconsole.html#topics-telnetconsole)。

## 内存泄漏的常见原因

它经常发生（有时是偶然的，有时是故意的）Scrapy开发人员传递请求中引用的对象（例如，使用[**meta**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.Request.meta)属性或请求回调函数），并且有效地将这些引用对象的生存期限限制为请求。到目前为止，这是Scrapy项目中最常见的内存泄漏原因，也是一个很难为新手调试的问题。

在大型项目中，蜘蛛通常由不同的人写成，并且一些蜘蛛可能会“泄漏”，从而在其他（写得很好的）蜘蛛同时运行时影响其他蜘蛛，这反过来会影响蜘蛛整个爬行过程。

如果您没有正确释放（以前分配的）资源，泄漏也可能来自您编写的自定义中间件，管道或扩展。例如，如果您[为每个进程](https://doc.scrapy.org/en/latest/topics/practices.html#run-multiple-spiders)运行[多个蜘蛛](https://doc.scrapy.org/en/latest/topics/practices.html#run-multiple-spiders)，则分配资源[**spider\_opened**](https://doc.scrapy.org/en/latest/topics/signals.html#std:signal-spider_opened) 但不释放它们[**spider\_closed**](https://doc.scrapy.org/en/latest/topics/signals.html#std:signal-spider_closed)可能会导致问题。

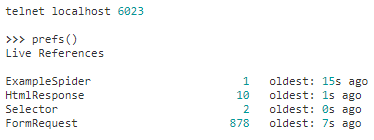
### 请求过多？

默认情况下，Scrapy将请求队列保存在内存中; 它包含 [**Request**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.Request)对象和Request属性中引用的所有对象（例如in [**meta**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.Request.meta)）。虽然不一定是泄漏，但这可能需要很多内存。启用 [持久作业队列](https://doc.scrapy.org/en/latest/topics/jobs.html#topics-jobs)可以帮助控制内存使用情况。

## 使用调试内存泄漏 trackref

**trackref**是Scrapy提供的用于调试最常见的内存泄漏情况的模块。它基本上跟踪对所有活动的请求，响应，项目和选择器对象的引用。

您可以输入telnet控制台并使用该prefs()函数的别名函数检查（上述类中的）对象的当前有多少个对象[**print\_live\_refs()**](https://doc.scrapy.org/en/latest/topics/leaks.html#scrapy.utils.trackref.print_live_refs)：



如您所见，该报告还显示了每个班级中最老的对象的“年龄”。如果您在每个进程中运行多个蜘蛛，您可以通过查看最旧的请求或响应来确定哪个蜘蛛在泄漏。您可以使用该[**get\_oldest()**](https://doc.scrapy.org/en/latest/topics/leaks.html#scrapy.utils.trackref.get_oldest)功能（从telnet控制台）获取每个班级的最老的对象 。

### 跟踪哪些对象？

跟踪的对象trackrefs都来自这些类（及其所有子类）：

* [**scrapy.http.Request**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.Request)
* [**scrapy.http.Response**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.Response)
* [**scrapy.item.Item**](https://doc.scrapy.org/en/latest/topics/items.html#scrapy.item.Item)
* [**scrapy.selector.Selector**](https://doc.scrapy.org/en/latest/topics/selectors.html#scrapy.selector.Selector)
* [**scrapy.spiders.Spider**](https://doc.scrapy.org/en/latest/topics/spiders.html#scrapy.spiders.Spider)

### 一个真实的例子

我们来看一个假设的内存泄漏情况的具体例子。假设我们有一条蜘蛛的线条与这条线相似：



该行在请求中传递响应引用，将响应生命周期有效地与请求的生命周期相关联，并且肯定会导致内存泄漏。

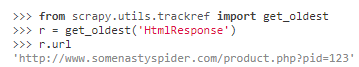
让我们看看我们如何通过使用该trackref工具发现原因（当然，不知道它是先验的）。

搜索器运行几分钟后，我们注意到它的内存使用量已经增长很多，我们可以进入它的telnet控制台并检查实时引用：

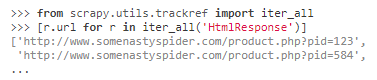


事实上有很多现场响应（并且它们太旧了）肯定是可疑的，因为与请求相比，响应应该具有相对较短的生命周期。响应的数量与请求的数量相似，因此看起来它们以某种方式相关联。我们现在可以去检查蜘蛛的代码来发现产生泄漏的令人讨厌的线（在请求中传递响应引用）。

有时候关于活动对象的额外信息会有所帮助。让我们来检查最老的回应：



如果你想遍历所有的对象，而不是获取最老的对象，你可以使用这个[**scrapy.utils.trackref.iter\_all()**](https://doc.scrapy.org/en/latest/topics/leaks.html#scrapy.utils.trackref.iter_all)函数：



### 蜘蛛太多了？

如果您的项目有太多的蜘蛛并行执行，输出**prefs()**可能难以阅读。出于这个原因，该函数有一个ignore参数可以用来忽略特定的类（及其所有子类）。例如，这不会显示任何对蜘蛛的实时引用：



### scrapy.utils.trackref模块

以下是[**trackref**](https://doc.scrapy.org/en/latest/topics/leaks.html#module-scrapy.utils.trackref)模块中可用的功能。

**Class scrapy.utils.trackref.object\_ref**

如果要使用**trackref**模块跟踪活动实例，则继承此类（而不是对象）。

**scrapy.utils.trackref.print\_live\_refs（*class\_name*，*ignore = NoneType*）**

打印生活参考报告，按类名称分组。

|  |  |
| --- | --- |
| **参数：** | **ignore**(class or classes tuple) （*类或类元组*） - 如果给定，则指定类（或元*组元组*）中的所有对象都将被忽略。 |

**scrapy.utils.trackref.get\_oldest（*class\_name*）**

用给定的类名返回最旧的对象，或者**None**如果没有找到。[**print\_live\_refs()**](https://doc.scrapy.org/en/latest/topics/leaks.html#scrapy.utils.trackref.print_live_refs)首先使用获取每个类名称的所有跟踪活动对象的列表。

**scrapy.utils.trackref.iter\_all（*class\_name*）**

使用给定的类名返回所有活动对象的迭代器，或者 **None**如果找不到任何对象。[**print\_live\_refs()**](https://doc.scrapy.org/en/latest/topics/leaks.html#scrapy.utils.trackref.print_live_refs)首先使用获取每个类名称的所有跟踪活动对象的列表。

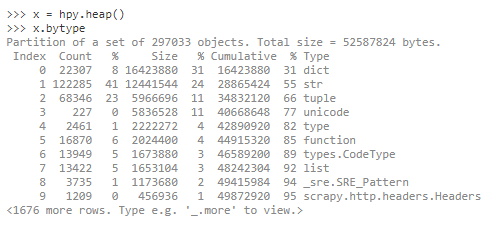
## 用Guppy调试内存泄漏

trackref为跟踪内存泄漏提供了一个非常方便的机制，但它仅跟踪更可能导致内存泄漏的对象（请求，响应，项目和选择器）。但是，还有其他情况下内存泄漏可能来自其他（或多或少模糊）对象。如果这是你的情况，并且你不能找到你的泄漏使用trackref，你仍然有另一个资源：[瓜皮库](https://pypi.python.org/pypi/guppy)。

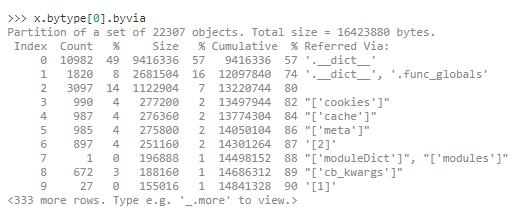
如果您使用pip，您可以使用以下命令安装Guppy：



Telnet控制台还带有一个内置快捷键（hpy），用于访问Guppy堆对象。以下是使用Guppy查看堆中可用的所有Python对象的示例：



你可以看到大多数空间都被字典使用。然后，如果你想查看哪些属性被引用，你可以这样做：



正如你所看到的，Guppy模块非常强大，但也需要一些关于Python内部的深入知识。有关Guppy的更多信息，请参阅 [Guppy文档](http://guppy-pe.sourceforge.net/)。

## 泄漏无泄漏

有时，您可能会注意到Scrapy进程的内存使用量只会增加，但从不会减少。不幸的是，即使Scrapy和您的项目都没有泄漏内存，也可能发生这种情况。这是由于Python（已不是很好）已知的问题，在某些情况下，它可能不会将释放的内存返回给操作系统。有关此问题的更多信息，请参阅：

* [Python内存管理](http://www.evanjones.ca/python-memory.html)
* [Python内存管理第2部分](http://www.evanjones.ca/python-memory-part2.html)
* [Python内存管理第3部分](http://www.evanjones.ca/python-memory-part3.html)

Evan Jones提出的改进，在[本文](http://www.evanjones.ca/memoryallocator/)中详细[介绍](http://www.evanjones.ca/memoryallocator/)，已经在Python 2.5中进行了合并，但这只能减少问题，并不能完全解决问题。引用这篇论文：

不幸的是，如果没有更多的对象被分配，这个补丁只能释放一个竞技​​场。这意味着碎片化是一个大问题。一个应用程序可能有许多兆字节的空闲内存，散布在所有场馆中，但它将无法释放任何内存。这是所有内存分配器遇到的问题。解决这个问题的唯一方法就是转移到一个压缩垃圾收集器，它能够将内存中的对象移动。这将需要对Python解释器进行重大更改。

为了保持内存消耗的合理性，您可以将作业拆分为几个较小的作业，或者 不时启用[持久作业队列](https://doc.scrapy.org/en/latest/topics/jobs.html#topics-jobs)和停止/启动蜘蛛。

[下一个](https://doc.scrapy.org/en/latest/topics/media-pipeline.html)[以前](https://doc.scrapy.org/en/latest/topics/firebug.html)

# 下载和处理文件和图像

Scrapy提供可重复使用的[项目管道，](https://doc.scrapy.org/en/latest/topics/item-pipeline.html)用于下载附加到特定项目的文件（例如，当您刮擦产品并且还想在本地下载其图像时）。这些管道共享一些功能和结构（我们将它们称为介质管道），但通常您可以使用“文件管道”或“图像管道”。

两个管道都实现这些功能：

* 避免重新下载最近下载的媒体
* 指定存储介质的位置（文件系统目录，Amazon S3存储桶，Google云存储存储桶）

图像管道有几个额外的功能来处理图像：

* 将所有下载的图像转换为通用格式（JPG）和模式（RGB）
* 生成缩略图
* 检查图像宽度/高度以确保它们符合最小限制

管道还保留当前正在计划下载的那些媒体URL的内部队列，并将包含相同媒体到达的那些响应连接到该队列。这避免了多个项目共享多次下载相同的媒体。

## 使用文件管道

典型的工作流程**FilesPipeline**如下所示：

1. 在Spider中，您刮取一个项目并将所需的URL放入一个 file\_urls字段中。
2. 该物品从蜘蛛中返回并进入物品管道。
3. 当该项目到达时**FilesPipeline**，该file\_urls字段中的URL 将使用标准的Scrapy调度程序和下载程序（这意味着调度程序和下载程序中间件被重新使用）计划下载，但具有更高的优先级，则在其他页面被删除之前处理​​它们。该项目在该特定管道阶段保持“锁定”状态，直到文件完成下载（或由于某种原因失败）。
4. 下载文件时，另一个字段（files）将填充结果。该字段将包含一个含有下载文件信息的词典列表，例如下载的路径，原始抓取的URL（从file\_urls现场获取）以及文件校验和。该files字段列表中的文件将保留原始file\_urls字段的相同顺序。如果某个文件下载失败，则会记录一个错误，并且该文件不会出现在该files字段中。

## 使用图像管道

使用这[**ImagesPipeline**](https://doc.scrapy.org/en/latest/topics/media-pipeline.html#scrapy.pipelines.images.ImagesPipeline)很像使用**FilesPipeline**，除了使用的默认字段名称不同：您image\_urls用于项目的图像URL，它将填充images字段以获取有关下载图像的信息。

使用[**ImagesPipeline**](https://doc.scrapy.org/en/latest/topics/media-pipeline.html#scrapy.pipelines.images.ImagesPipeline)for图像文件的优点是，您可以配置一些额外的功能，例如生成缩略图和根据图像大小过滤图像。

Images Pipeline使用[Pillow](https://github.com/python-pillow/Pillow)将图像缩略图和规格化为JPEG / RGB格式，因此您需要安装此库才能使用它。 在大多数情况下，[Python成像库](http://www.pythonware.com/products/pil/)（PIL）也应该可以工作，但是在某些设置中会导致麻烦，所以我们建议使用[Pillow](https://github.com/python-pillow/Pillow)而不是PIL。

## 启用媒体管道

要启用媒体管道，您必须先将其添加到您的项目 [**ITEM\_PIPELINES**](https://doc.scrapy.org/en/latest/topics/settings.html#std:setting-ITEM_PIPELINES)设置中。

对于图像管道，请使用：

ITEM\_PIPELINES **=**  { 'scrapy.pipelines.images.ImagesPipeline' ： 1 }

对于文件管道，请使用：

ITEM\_PIPELINES **=**  { 'scrapy.pipelines.files.FilesPipeline' ： 1 }

**注意**

您也可以同时使用文件和图像管道。

然后，将目标存储设置配置为用于存储下载图像的有效值。否则，管道将保持禁用状态，即使您将其包含在[**ITEM\_PIPELINES**](https://doc.scrapy.org/en/latest/topics/settings.html#std:setting-ITEM_PIPELINES)设置中。

对于文件管道，请设置以下[**FILES\_STORE**](https://doc.scrapy.org/en/latest/topics/media-pipeline.html#std:setting-FILES_STORE)设置：



对于图像管线，设置[**IMAGES\_STORE**](https://doc.scrapy.org/en/latest/topics/media-pipeline.html#std:setting-IMAGES_STORE)设置：



## 支持的存储

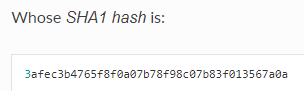
文件系统目前是唯一官方支持的存储，但也支持在[Amazon S3](https://aws.amazon.com/s3/)和[Google云存储中](https://cloud.google.com/storage/)存储文件。

### 文件系统存储

这些文件使用它们的URL 的[SHA1散列](https://en.wikipedia.org/wiki/SHA_hash_functions)来存储文件名。

例如，以下图片网址：





将被下载并存储在以下文件中：



哪里：

* <IMAGES\_STORE>是[**IMAGES\_STORE**](https://doc.scrapy.org/en/latest/topics/media-pipeline.html#std:setting-IMAGES_STORE)图像管道设置中定义的目录。
* full是将完整图像与缩略图分开的子目录（如果使用的话）。有关更多信息，请参阅[图像的缩略图生成](https://doc.scrapy.org/en/latest/topics/media-pipeline.html#topics-images-thumbnails)。

### Amazon S3存储

[**FILES\_STORE**](https://doc.scrapy.org/en/latest/topics/media-pipeline.html#std:setting-FILES_STORE)并[**IMAGES\_STORE**](https://doc.scrapy.org/en/latest/topics/media-pipeline.html#std:setting-IMAGES_STORE)可以代表Amazon S3存储桶。Scrapy会自动将文件上传到存储桶。

例如，这是一个有效的[**IMAGES\_STORE**](https://doc.scrapy.org/en/latest/topics/media-pipeline.html#std:setting-IMAGES_STORE)值：



您可以修改用于存储文件的访问控制列表（ACL）策略，这是由[**FILES\_STORE\_S3\_ACL**](https://doc.scrapy.org/en/latest/topics/media-pipeline.html#std:setting-FILES_STORE_S3_ACL)和[**IMAGES\_STORE\_S3\_ACL**](https://doc.scrapy.org/en/latest/topics/media-pipeline.html#std:setting-IMAGES_STORE_S3_ACL)设置定义的。默认情况下，ACL设置为 private。要使文件公开可用，请使用以下public-read 策略：



有关更多信息，请参阅Amazon S3开发人员指南中的[预装ACL](https://docs.aws.amazon.com/AmazonS3/latest/dev/acl-overview.html#canned-acl)。

### Google云端存储

[**FILES\_STORE**](https://doc.scrapy.org/en/latest/topics/media-pipeline.html#std:setting-FILES_STORE)并[**IMAGES\_STORE**](https://doc.scrapy.org/en/latest/topics/media-pipeline.html#std:setting-IMAGES_STORE)可以代表Google云存储存储分区。Scrapy会自动将文件上传到存储桶。（需要[谷歌云存储](https://cloud.google.com/storage/docs/reference/libraries#client-libraries-install-python)）

例如，这些是有效的[**IMAGES\_STORE**](https://doc.scrapy.org/en/latest/topics/media-pipeline.html#std:setting-IMAGES_STORE)和[**GCS\_PROJECT\_ID**](https://doc.scrapy.org/en/latest/topics/media-pipeline.html#std:setting-GCS_PROJECT_ID)设置：



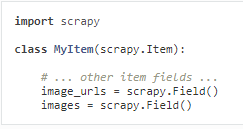
有关身份验证的信息，请参阅此[文档](https://cloud.google.com/docs/authentication/production)。

## 用法示例

为了首先使用媒体管道，[启用它](https://doc.scrapy.org/en/latest/topics/media-pipeline.html#topics-media-pipeline-enabling)。

然后，如果蜘蛛用URLs键（file\_urls或者 image\_urls分别为文件或图像管线）返回字典，管道将把结果放在相应的键（files或images）下。

如果您更喜欢使用[**Item**](https://doc.scrapy.org/en/latest/topics/items.html#scrapy.item.Item)，那么使用必要的字段定义一个自定义项目，例如图像管道的示例：



如果要为URL键或结果键使用另一个字段名称，也可以覆盖它。

对于文件管道，设置[**FILES\_URLS\_FIELD**](https://doc.scrapy.org/en/latest/topics/media-pipeline.html#std:setting-FILES_URLS_FIELD)和/或 [**FILES\_RESULT\_FIELD**](https://doc.scrapy.org/en/latest/topics/media-pipeline.html#std:setting-FILES_RESULT_FIELD)设置：



对于图像管线，设置[**IMAGES\_URLS\_FIELD**](https://doc.scrapy.org/en/latest/topics/media-pipeline.html#std:setting-IMAGES_URLS_FIELD)和/或 [**IMAGES\_RESULT\_FIELD**](https://doc.scrapy.org/en/latest/topics/media-pipeline.html#std:setting-IMAGES_RESULT_FIELD)设置：



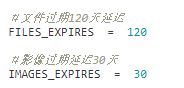
如果您需要更复杂的内容并希望覆盖自定义管道行为，请参阅[扩展介质管道](https://doc.scrapy.org/en/latest/topics/media-pipeline.html#topics-media-pipeline-override)。

如果您有多个图像管道从ImagePipeline继承，并且您希望在不同管道中使用不同的设置，则可以设置以管道类的大写名称开头的设置键。例如，如果您的管道被称为MyPipeline，并且您想定制IMAGES\_URLS\_FIELD，则可以定义设置MYPIPELINE\_IMAGES\_URLS\_FIELD并使用您的自定义设置。

## 附加功能

### 文件到期

图像管道避免了下载最近下载的文件。要调整此保留延迟，请使用[**FILES\_EXPIRES**](https://doc.scrapy.org/en/latest/topics/media-pipeline.html#std:setting-FILES_EXPIRES)设置（或者[**IMAGES\_EXPIRES**](https://doc.scrapy.org/en/latest/topics/media-pipeline.html#std:setting-IMAGES_EXPIRES)在图像管道的情况下），该设置指定延迟天数：



这两个设置的默认值是90天。

如果你有管道的子类FilesPipeline，你想有不同的设置，你可以设置以大写的类名为前缀的设置键。例如，给定管道类名为MyPipeline，您可以设置设置键：

MYPIPELINE\_FILES\_EXPIRES = 180

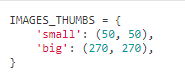
并且管道类MyPipeline将到期时间设置为180。

### 为图像生成缩略图

图像管道可以自动创建下载图像的缩略图。

为了使用此功能，您必须设置[**IMAGES\_THUMBS**](https://doc.scrapy.org/en/latest/topics/media-pipeline.html#std:setting-IMAGES_THUMBS)一个字典，其中的键是缩略图名称，值是它们的尺寸。

例如：



当您使用此功能时，图像管道将使用以下格式创建每个指定尺寸的缩略图：



哪里：

* <size\_name>是在指定的[**IMAGES\_THUMBS**](https://doc.scrapy.org/en/latest/topics/media-pipeline.html#std:setting-IMAGES_THUMBS) 字典键（small，big等）
* <image\_id>是图片网址的[SHA1哈希值](https://en.wikipedia.org/wiki/SHA_hash_functions)

使用small和big缩略图名称存储的图像文件示例：



第一个是从网站下载的完整图像。

### 过滤掉小图片

使用图像管线时，可以通过指定[**IMAGES\_MIN\_HEIGHT**](https://doc.scrapy.org/en/latest/topics/media-pipeline.html#std:setting-IMAGES_MIN_HEIGHT)和 [**IMAGES\_MIN\_WIDTH**](https://doc.scrapy.org/en/latest/topics/media-pipeline.html#std:setting-IMAGES_MIN_WIDTH)设置中允许的最小尺寸来放置太小的图像。

例如：



**注意**

大小限制完全不影响缩略图生成。

可以只设置一个尺寸约束或者两者兼有。设置它们时，只会保存同时满足最小尺寸的图像。对于上述示例，大小（105 x 105）或（105 x 200）或（200 x 105）的图像将全部被删除，因为至少一个维度比约束更短。

默认情况下，没有大小限制，因此所有图像都被处理。

### 允许重定向

默认情况下，媒体管道忽略重定向，即HTTP重定向到媒体文件URL请求将意味着媒体下载被认为失败。

要处理媒体重定向，请将此设置设置为True：



## 扩展媒体管道

在这里看到你可以在自定义文件管道中覆盖的方法：

***class*scrapy.pipelines.files.FilesPipeline**

**get\_media\_requests(*item*, *info*)**

如工作流程所示，管道将获取要从项目下载的图像的URL。为了做到这一点，您可以覆盖该[**get\_media\_requests()**](https://doc.scrapy.org/en/latest/topics/media-pipeline.html#scrapy.pipelines.files.FilesPipeline.get_media_requests)方法并为每个文件URL返回一个请求：

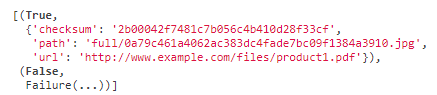


这些请求将由管道处理，并且当它们完成下载时，结果将[**item\_completed()**](https://doc.scrapy.org/en/latest/topics/media-pipeline.html#scrapy.pipelines.files.FilesPipeline.item_completed)作为2元素元组列表发送到 方法。每个元组将包含以下内容：**(success, file\_info\_or\_error)**

* **success**是一个布尔值，**True**如果图像下载成功或者**False**由于某种原因失败了
* **file\_info\_or\_error**是一个包含以下键（如果成功**True**）的词典，或者如果出现问题，则是[扭曲失败](https://twistedmatrix.com/documents/current/api/twisted.python.failure.Failure.html)。
  + **url** - 文件的下载地址。这是从[**get\_media\_requests()**](https://doc.scrapy.org/en/latest/topics/media-pipeline.html#scrapy.pipelines.files.FilesPipeline.get_media_requests) 方法返回的请求的url 。
  + **path**- [**FILES\_STORE**](https://doc.scrapy.org/en/latest/topics/media-pipeline.html#std:setting-FILES_STORE)文件存储位置的路径（相对于）
  + **checksum**- 图像内容的[MD5哈希](https://en.wikipedia.org/wiki/MD5)

接收到的元组列表[**item\_completed()**](https://doc.scrapy.org/en/latest/topics/media-pipeline.html#scrapy.pipelines.files.FilesPipeline.item_completed)保证保持从该[**get\_media\_requests()**](https://doc.scrapy.org/en/latest/topics/media-pipeline.html#scrapy.pipelines.files.FilesPipeline.get_media_requests)方法返回的请求的相同顺序 。

这是一个典型的**results**参数值：



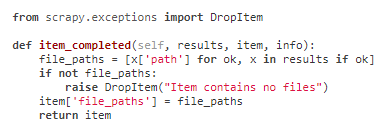
默认情况下，该[**get\_media\_requests()**](https://doc.scrapy.org/en/latest/topics/media-pipeline.html#scrapy.pipelines.files.FilesPipeline.get_media_requests)方法返回**None**，这意味着没有文件要下载的项目。

**item\_completed(results, item, info)**

[**FilesPipeline.item\_completed()**](https://doc.scrapy.org/en/latest/topics/media-pipeline.html#scrapy.pipelines.files.FilesPipeline.item_completed)当单个项目的所有文件请求都已完成（完成下载或由于某种原因失败）时调用此方法。

该[**item\_completed()**](https://doc.scrapy.org/en/latest/topics/media-pipeline.html#scrapy.pipelines.files.FilesPipeline.item_completed)方法必须返回将发送到后续项目管道阶段的输出，因此您必须返回（或丢弃）该项目，就像在任何管道中一样。

以下是[**item\_completed()**](https://doc.scrapy.org/en/latest/topics/media-pipeline.html#scrapy.pipelines.files.FilesPipeline.item_completed)我们将下载的文件路径（传递到结果中）存储在**file\_paths** 项目字段中的方法示例，如果项目不包含任何文件，我们将删除该项目：



默认情况下，该[**item\_completed()**](https://doc.scrapy.org/en/latest/topics/media-pipeline.html#scrapy.pipelines.files.FilesPipeline.item_completed)方法返回该项目。

在这里看到你可以在自定义图像管道中覆盖的方法：

**Class scrapy.pipelines.images.ImagesPipeline**

这[**ImagesPipeline**](https://doc.scrapy.org/en/latest/topics/media-pipeline.html#scrapy.pipelines.images.ImagesPipeline)是**FilesPipeline**对字段名称进行自定义并为图像添加自定义行为的扩展。

**get\_media\_requests（*item*，*info*）**

以与方法相同的方式工作**FilesPipeline.get\_media\_requests()**，但对图像网址使用不同的字段名称。

必须为每个图片网址返回一个请求。

**item\_completed(results, item, info)**

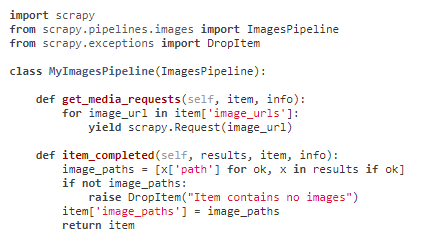
[**ImagesPipeline.item\_completed()**](https://doc.scrapy.org/en/latest/topics/media-pipeline.html#scrapy.pipelines.images.ImagesPipeline.item_completed)当单个项目的所有图像请求都已完成（完成下载或由于某种原因失败）时调用此方法。

以与方法相同的方式工作**FilesPipeline.item\_completed()**，但使用不同的字段名称来存储图像下载结果。

默认情况下，该[**item\_completed()**](https://doc.scrapy.org/en/latest/topics/media-pipeline.html#scrapy.pipelines.images.ImagesPipeline.item_completed)方法返回该项目。

## 自定义图像管道示例

下面是图像管道的完整示例，其示例方法如上所示：



**部署蜘蛛**

本节介绍您为部署Scrapy蜘蛛而定期运行它们的不同选项。在本地机器上运行Scrapy蜘蛛程序对于（早期）开发阶段非常方便，但是当您需要执行长时间运行的蜘蛛或移动蜘蛛以连续运行时，则不是那么重要。这是部署Scrapy蜘蛛解决方案的地方。

部署Scrapy蜘蛛的普遍选择是：

* [Scrapyd](https://doc.scrapy.org/en/latest/topics/deploy.html#deploy-scrapyd)（open source 开源）
* [Scrapy [Cloud](https://doc.scrapy.org/en/latest/topics/deploy.html#deploy-scrapy-cloud)](https://doc.scrapy.org/en/latest/topics/deploy.html#deploy-scrapy-cloud)（cloud-based 基于云）

## 部署到Scrapyd服务器

[Scrapyd](https://github.com/scrapy/scrapyd)是运行Scrapy蜘蛛的开源应用程序。它提供了一个HTTP API的服务器，能够运行和监控Scrapy蜘蛛。

要将Spider部署到Scrapyd，您可以使用由[scrapyd-client](https://github.com/scrapy/scrapyd-client)包提供的scrapyd-deploy工具。请参阅[scrapyd-deploy文档](https://scrapyd.readthedocs.io/en/latest/deploy.html)以获取更多信息。

Scrapyd由一些Scrapy开发人员维护。

## 部署到Scrapy Cloud

[Scrapy Cloud](https://scrapinghub.com/scrapy-cloud)是[Scrapy](https://scrapinghub.com/scrapy-cloud)背后的[Scrapinghub](https://scrapinghub.com/)托管的基于云的服务。

Scrapy Cloud不需要安装和监控服务器，并提供了一个很好的用户界面来管理蜘蛛并查看抓取的项目，日志和统计信息。

要将Scider部署到Scrapy Cloud，您可以使用[shub](https://doc.scrapinghub.com/shub.html)命令行工具。请参阅[Scrapy Cloud文档](https://doc.scrapinghub.com/scrapy-cloud.html)以获取更多信息。

Scrapy Cloud与Scrapyd兼容，并且可以根据需要在它们之间切换 - 从配置scrapy.cfg文件中读取配置scrapyd-deploy。

[下一个](https://doc.scrapy.org/en/latest/topics/autothrottle.html)[以前](https://doc.scrapy.org/en/latest/topics/media-pipeline.html)

# AutoThrottle扩展

这是一个基于Scrapy服务器和您正在爬行的网站的负载自动限制爬网速度的扩展。

## 设计目标

1. 更好的网站，而不是使用默认的下载延迟零
2. 自动将scrapy调整为最佳爬行速度，因此用户不必调整下载延迟以找到最佳爬行速度。用户只需要指定它允许的最大并发请求，其余的则由扩展来完成。

## 怎么运行的

AutoThrottle扩展可动态调整下载延迟，以使蜘蛛向[**AUTOTHROTTLE\_TARGET\_CONCURRENCY**](https://doc.scrapy.org/en/latest/topics/autothrottle.html#std:setting-AUTOTHROTTLE_TARGET_CONCURRENCY)每个远程网站平均发送 并发请求。

它使用下载延迟来计算延迟。其主要思想是：如果一台服务器需要latency秒钟响应，客户端应该发送一个请求的每个latency/N秒，具有N并行处理的请求。

而不是调整延迟，可以设置一个小的固定下载延迟并对并发使用[**CONCURRENT\_REQUESTS\_PER\_DOMAIN**](https://doc.scrapy.org/en/latest/topics/settings.html#std:setting-CONCURRENT_REQUESTS_PER_DOMAIN)或[**CONCURRENT\_REQUESTS\_PER\_IP**](https://doc.scrapy.org/en/latest/topics/settings.html#std:setting-CONCURRENT_REQUESTS_PER_IP)选项施加硬性限制 。它会提供类似的效果，但有一些重要的区别：

* 由于下载延迟很小，偶尔会出现一些请求;
* 通常非200（错误）响应可以比常规响应更快地返回，因此，在服务器开始返回错误时，如果下载延迟较小，并发限制爬网程序将更快地向服务器发送请求。但这与爬虫应该做的事情相反 - 如果发生错误，减慢速度更有意义：这些错误可能是由高请求率造成的。

AutoThrottle没有这些问题。

## 节流算法

AutoThrottle算法根据以下规则调整下载延迟：

1. 蜘蛛总是以下载延迟开始 [**AUTOTHROTTLE\_START\_DELAY**](https://doc.scrapy.org/en/latest/topics/autothrottle.html#std:setting-AUTOTHROTTLE_START_DELAY);
2. 当接收到响应时，目标下载延迟被计算为 其中响应的等待时间，并且是。latency / N latency N [**AUTOTHROTTLE\_TARGET\_CONCURRENCY**](https://doc.scrapy.org/en/latest/topics/autothrottle.html#std:setting-AUTOTHROTTLE_TARGET_CONCURRENCY)
3. 下次请求的下载延迟设置为上次下载延迟的平均值和目标下载延迟;
4. 不允许200个响应的延迟减少延迟;
5. 下载延迟不能小于[**DOWNLOAD\_DELAY**](https://doc.scrapy.org/en/latest/topics/settings.html#std:setting-DOWNLOAD_DELAY)或大于[**AUTOTHROTTLE\_MAX\_DELAY**](https://doc.scrapy.org/en/latest/topics/autothrottle.html#std:setting-AUTOTHROTTLE_MAX_DELAY)

**注意**

AutoThrottle扩展支持标准Scrapy设置的并发性和延迟。这意味着它会尊重[**CONCURRENT\_REQUESTS\_PER\_DOMAIN**](https://doc.scrapy.org/en/latest/topics/settings.html#std:setting-CONCURRENT_REQUESTS_PER_DOMAIN)和 [**CONCURRENT\_REQUESTS\_PER\_IP**](https://doc.scrapy.org/en/latest/topics/settings.html#std:setting-CONCURRENT_REQUESTS_PER_IP)选择，永远不会将下载延迟设置为低于[**DOWNLOAD\_DELAY**](https://doc.scrapy.org/en/latest/topics/settings.html#std:setting-DOWNLOAD_DELAY)。

在Scrapy中，下载延迟时间是根据建立TCP连接和接收HTTP头之间的时间来度量的。

请注意，这些延迟在协作式多任务环境中很难准确测量，因为Scrapy可能正在忙于处理蜘蛛回调，并且无法参加下载。但是，这些延迟仍应该对Scrapy（最终是服务器）的繁忙程度进行合理估计，并且此扩展建立在此前提之上。

## 设置

用于控制AutoThrottle扩展的设置是：

* [**AUTOTHROTTLE\_ENABLED**](https://doc.scrapy.org/en/latest/topics/autothrottle.html#std:setting-AUTOTHROTTLE_ENABLED)
* [**AUTOTHROTTLE\_START\_DELAY**](https://doc.scrapy.org/en/latest/topics/autothrottle.html#std:setting-AUTOTHROTTLE_START_DELAY)
* [**AUTOTHROTTLE\_MAX\_DELAY**](https://doc.scrapy.org/en/latest/topics/autothrottle.html#std:setting-AUTOTHROTTLE_MAX_DELAY)
* [**AUTOTHROTTLE\_TARGET\_CONCURRENCY**](https://doc.scrapy.org/en/latest/topics/autothrottle.html#std:setting-AUTOTHROTTLE_TARGET_CONCURRENCY)
* [**AUTOTHROTTLE\_DEBUG**](https://doc.scrapy.org/en/latest/topics/autothrottle.html#std:setting-AUTOTHROTTLE_DEBUG)
* [**CONCURRENT\_REQUESTS\_PER\_DOMAIN**](https://doc.scrapy.org/en/latest/topics/settings.html#std:setting-CONCURRENT_REQUESTS_PER_DOMAIN)
* [**CONCURRENT\_REQUESTS\_PER\_IP**](https://doc.scrapy.org/en/latest/topics/settings.html#std:setting-CONCURRENT_REQUESTS_PER_IP)
* [**DOWNLOAD\_DELAY**](https://doc.scrapy.org/en/latest/topics/settings.html#std:setting-DOWNLOAD_DELAY)

有关更多信息，请参阅[它如何工作](https://doc.scrapy.org/en/latest/topics/autothrottle.html#autothrottle-algorithm)。

### AUTOTHROTTLE\_ENABLED

默认： False

启用AutoThrottle扩展。

### AUTOTHROTTLE\_START\_DELAY

默认： 5.0

最初的下载延迟（以秒为单位）。

### AUTOTHROTTLE\_MAX\_DELAY

默认： 60.0

在高延迟情况下设置的最大下载延迟（以秒为单位）。

### AUTOTHROTTLE\_TARGET\_CONCURRENCY

*版本1.1中的新功能*

默认： 1.0

Scrapy应平行发送到远程网站的平均请求数量。

默认情况下，AutoThrottle调整延迟以向每个远程网站发送单个并发请求。将此选项设置为更高的值（例如2.0）以增加吞吐量和远程服务器的负载。较低的AUTOTHROTTLE\_TARGET\_CONCURRENCY值（例如0.5）会使抓取工具更加保守和礼貌。

请注意，[**CONCURRENT\_REQUESTS\_PER\_DOMAIN**](https://doc.scrapy.org/en/latest/topics/settings.html#std:setting-CONCURRENT_REQUESTS_PER_DOMAIN) 并[**CONCURRENT\_REQUESTS\_PER\_IP**](https://doc.scrapy.org/en/latest/topics/settings.html#std:setting-CONCURRENT_REQUESTS_PER_IP)在启用自动油门扩展选项仍然尊重。这意味着如果 AUTOTHROTTLE\_TARGET\_CONCURRENCY设置的值高于 [**CONCURRENT\_REQUESTS\_PER\_DOMAIN**](https://doc.scrapy.org/en/latest/topics/settings.html#std:setting-CONCURRENT_REQUESTS_PER_DOMAIN)或者 [**CONCURRENT\_REQUESTS\_PER\_IP**](https://doc.scrapy.org/en/latest/topics/settings.html#std:setting-CONCURRENT_REQUESTS_PER_IP)，搜寻器将不会达到这个并发请求数。

在每个给定的时间点，Scrapy可以发送比或多或少的并发请求AUTOTHROTTLE\_TARGET\_CONCURRENCY; 它是爬虫尝试接近的建议值，而不是硬限制。

### AUTOTHROTTLE\_DEBUG

默认： False

启用AutoThrottle调试模式，该模式将显示收到的每个响应的统计数据，以便您可以看到如何实时调整调节参数。

# Benchmarking

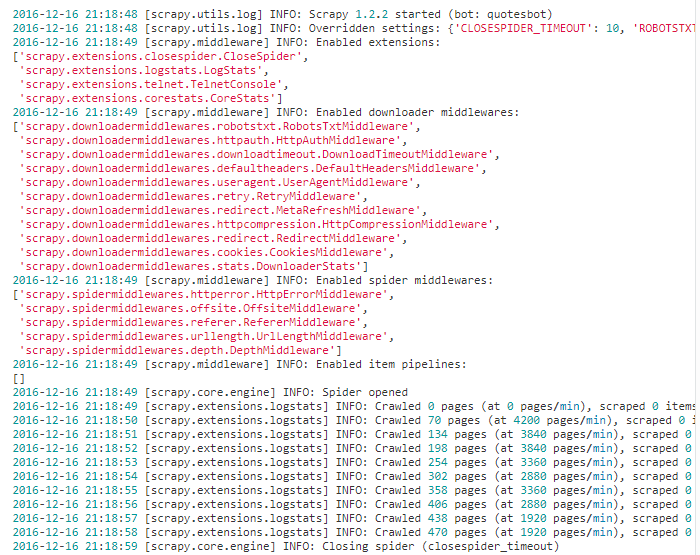
*0.17版本中的新功能。*

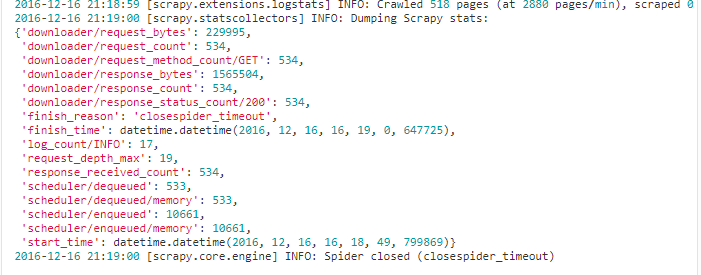
Scrapy带有一个简单的基准测试套件，它产生一个本地HTTP服务器并以最大可能的速度抓取它。这个基准测试的目标是了解Scrapy如何在您的硬件中执行，以便有一个通用的比较基准。它使用一个简单的蜘蛛，它什么都不做，只是在链接之后。

运行它使用：



你应该看到这样的输出：





这告诉你Scrapy能够在你运行硬件的地方每分钟抓取大约3000页。请注意，这是一个非常简单的蜘蛛，旨在跟踪链接，你写的任何自定义蜘蛛可能会做更多的东西，导致爬行速度较慢。多慢取决于你的蜘蛛做多少，写得多好。

未来，更多的案例将被添加到基准测试套件中以涵盖其他常见场景。

**工作：暂停和恢复抓取**

有时，对于大型网站，最好暂停爬网并稍后恢复。

Scrapy通过提供以下功能支持此功能：

* 一个在磁盘上保存预定请求的调度程序
* 一个重复过滤器，它保留磁盘上的访问请求
* 批次间保持一些蜘蛛状态（键/值对）的扩展

**工作目录**

要启用持久性支持，只需通过设置定义一个*作业目录即可*JOBDIR。该目录将用于存储所有必需的数据以保持单个作业的状态（即蜘蛛程序运行）。需要注意的是，该目录不能由不同的蜘蛛共享，甚至不能由同一蜘蛛的不同作业/运行共享，因为它旨在用于存储*单个*作业的状态。

**如何使用它**

要启用支持启用持久性的spider，请像这样运行它：



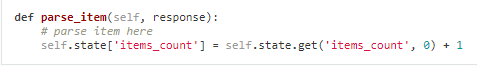
然后，您可以随时安全地停止蜘蛛（通过按Ctrl-C或发送信号），稍后通过发出相同的命令来恢复它：



## 在批次之间保持持久状态

有时你会想要在暂停/恢复批次之间保持一些持久的蜘蛛状态。你可以使用该spider.state属性，这应该是一个字典。当蜘蛛启动和停止时，有一个内置的扩展，负责序列化，存储和加载作业目录中的属性。

下面是一个使用蜘蛛状态的回调示例（为简洁起见，其他蜘蛛代码被省略）：



## 持久性陷阱

如果您希望能够使用Scrapy持久性支持，请注意以下几点：

### Cookies过期

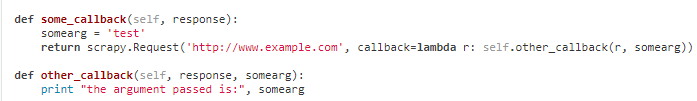
Cookie可能会过期。所以，如果你不能迅速恢复你的蜘蛛，预定的请求可能不再有效。如果你的蜘蛛不依赖cookies，这不会成为问题。

### 请求序列化

请求必须由pickle模块序列化，以便持久化工作，所以你应该确保你的请求是可序列化的。

这里最常见的问题是lambda在请求回调中使用不能被持久化的函数。

所以，例如，这不起作用：



但是这将会：

**def** **some\_callback**(self, response):

somearg **=** 'test'

**return** scrapy**.**Request('http://www.example.com', callback**=**self**.**other\_callback, meta**=**{'somearg': somearg})

**def** **other\_callback**(self, response):

somearg **=** response**.**meta['somearg']

print "the argument passed is:", somearg

如果您希望记录无法序列化的请求，可以将该 [**SCHEDULER\_DEBUG**](https://doc.scrapy.org/en/latest/topics/settings.html#std:setting-SCHEDULER_DEBUG)设置设置为True在项目的设置页面中。这是False默认的。

[下一个](https://doc.scrapy.org/en/latest/topics/architecture.html)[以前](https://doc.scrapy.org/en/latest/topics/benchmarking.html)

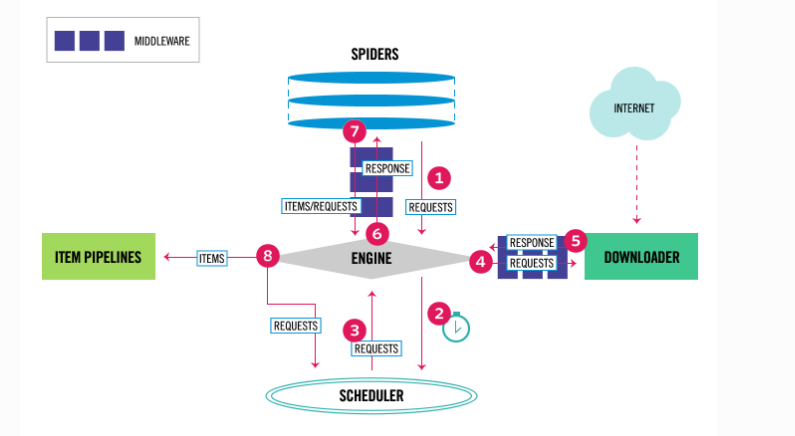
# 架构概述

本文档描述了Scrapy的体系结构以及它的组件如何交互。

## 概观

下图显示了Scrapy体系结构及其组件的概述以及在系统内部发生的数据流概​​述（用红色箭头表示）。下面将对这些组件进行简要说明，并提供有关这些组件的更多详细信息的链接。数据流也在下面描述。

## 数据流



Scrapy中的数据流由执行引擎控制，如下所示：

1. 该[引擎](https://doc.scrapy.org/en/latest/topics/architecture.html#component-engine)获得初始请求从抓取 [蜘蛛](https://doc.scrapy.org/en/latest/topics/architecture.html#component-spiders)。
2. 该[引擎](https://doc.scrapy.org/en/latest/topics/architecture.html#component-engine)安排在请求 [调度程序](https://doc.scrapy.org/en/latest/topics/architecture.html#component-scheduler)和要求下一个请求抓取。
3. 该[计划](https://doc.scrapy.org/en/latest/topics/architecture.html#component-scheduler)返回下一请求的[引擎](https://doc.scrapy.org/en/latest/topics/architecture.html#component-engine)。
4. 该[引擎](https://doc.scrapy.org/en/latest/topics/architecture.html#component-engine)发送请求到 [下载器](https://doc.scrapy.org/en/latest/topics/architecture.html#component-downloader)，通过 [下载器中间件](https://doc.scrapy.org/en/latest/topics/architecture.html#component-downloader-middleware)（见 [**process\_request()**](https://doc.scrapy.org/en/latest/topics/downloader-middleware.html#scrapy.downloadermiddlewares.DownloaderMiddleware.process_request)）。
5. 一旦页面完成下载， [Downloader会](https://doc.scrapy.org/en/latest/topics/architecture.html#component-downloader)生成一个响应（包含该页面）并将其发送到引擎，并通过[Downloader Middlewares](https://doc.scrapy.org/en/latest/topics/architecture.html#component-downloader-middleware)（请参阅 [参考资料](https://doc.scrapy.org/en/latest/topics/architecture.html#component-downloader-middleware)[**process\_response()**](https://doc.scrapy.org/en/latest/topics/downloader-middleware.html#scrapy.downloadermiddlewares.DownloaderMiddleware.process_response)）。
6. 该[引擎](https://doc.scrapy.org/en/latest/topics/architecture.html#component-engine)接收来自响应 [下载器](https://doc.scrapy.org/en/latest/topics/architecture.html#component-downloader)并将其发送到所述 [蜘蛛](https://doc.scrapy.org/en/latest/topics/architecture.html#component-spiders)进行处理，通过[蜘蛛中间件](https://doc.scrapy.org/en/latest/topics/architecture.html#component-spider-middleware)（见[**process\_spider\_input()**](https://doc.scrapy.org/en/latest/topics/spider-middleware.html#scrapy.spidermiddlewares.SpiderMiddleware.process_spider_input)）。
7. [Spider](https://doc.scrapy.org/en/latest/topics/architecture.html#component-spiders)处理响应，并通过 [蜘蛛中间件](https://doc.scrapy.org/en/latest/topics/architecture.html#component-spider-middleware)（请参阅 [参考资料](https://doc.scrapy.org/en/latest/topics/architecture.html#component-spider-middleware)）返回抓取的项目和新的请求（接下来）到 [引擎](https://doc.scrapy.org/en/latest/topics/architecture.html#component-engine)。[**process\_spider\_output()**](https://doc.scrapy.org/en/latest/topics/spider-middleware.html#scrapy.spidermiddlewares.SpiderMiddleware.process_spider_output)
8. 该[引擎](https://doc.scrapy.org/en/latest/topics/architecture.html#component-engine)发送处理的项目，以 [项目管道](https://doc.scrapy.org/en/latest/topics/architecture.html#component-pipelines)，然后把处理的请求的[调度](https://doc.scrapy.org/en/latest/topics/architecture.html#component-scheduler)，并要求今后可能要求抓取。
9. 该过程重复（从第1步开始），直到[调度程序](https://doc.scrapy.org/en/latest/topics/architecture.html#component-scheduler)没有更多请求 。

## 组件

### Scrapy引擎

引擎负责控制系统所有组件之间的数据流，并在发生某些操作时触发事件。有关更多详细信息，请参阅上面的 [数据流](https://doc.scrapy.org/en/latest/topics/architecture.html#data-flow)部分

### 调度

调度程序接收来自引擎的请求，并将它们排入队列，以便在引擎请求它们时将它们提供给它们（也引擎）。

### 下载

下载器负责获取网页并将它们馈送到引擎，然后引擎将它们馈送给蜘蛛。

### 蜘蛛

蜘蛛程序是由Scrapy用户编写的自定义类，用于解析响应并从中提取项目（也称为抓取的项目）或追加的其他请求。欲了解更多信息，请参阅[蜘蛛](https://doc.scrapy.org/en/latest/topics/spiders.html#topics-spiders)。

### 物品管道

物品管道负责处理物品，一旦它们被蜘蛛提取（或刮掉）。典型的任务包括清理，验证和持久性（如将项目存储在数据库中）。欲了解更多信息，请参阅[项目管道](https://doc.scrapy.org/en/latest/topics/item-pipeline.html#topics-item-pipeline)。

### 下载中间件

下载器中间件是位于引擎和下载器之间的特定钩子，当它们从引擎传递到下载器时处理请求，以及从下载器传递到引擎的响应。

如果您需要执行以下操作之一，请使用Downloader中间件：

* 在将请求发送到Downloader之前处理请求（即，在Scrapy将请求发送到网站之前）;
* 在将其传递给蜘蛛之前改变接收到的响应;
* 发送新的请求，而不是将接收到的响应传递给蜘蛛;
* 向蜘蛛传递响应而不需要获取网页;
* 默默地放下一些请求。

欲了解更多信息，请参阅[下载中间件](https://doc.scrapy.org/en/latest/topics/downloader-middleware.html#topics-downloader-middleware)。

### 蜘蛛中间件

蜘蛛中间件是引擎和蜘蛛之间的特定钩子，能够处理蜘蛛输入（响应）和输出（项目和请求）。

如果需要，请使用Spider中间件

* spider回调的后处理输出 - 更改/添加/删除请求或项目;
* 后处理start\_requests;
* 处理蜘蛛异常;
* 根据响应内容为一些请求调用errback而不是回叫。

欲了解更多信息，请参阅[蜘蛛中间件](https://doc.scrapy.org/en/latest/topics/spider-middleware.html#topics-spider-middleware)。

## 事件驱动的网络

Scrapy是用[Twisted](https://twistedmatrix.com/trac/)编写的，这是一个流行的事件驱动的Python网络框架。因此，它使用非阻塞（又称异步）代码来实现并发。

有关异步编程和Twisted的更多信息，请参阅以下链接：

* [[Twisted](https://twistedmatrix.com/documents/current/core/howto/defer-intro.html)的延期介绍](https://twistedmatrix.com/documents/current/core/howto/defer-intro.html)
* [[Twisted](https://twistedmatrix.com/documents/current/core/howto/defer-intro.html)- 你好，异步编程](http://jessenoller.com/2009/02/11/twisted-hello-asynchronous-programming/)
* [[Twisted](https://twistedmatrix.com/documents/current/core/howto/defer-intro.html)的介绍 - Krondo](http://krondo.com/an-introduction-to-asynchronous-programming-and-twisted/)

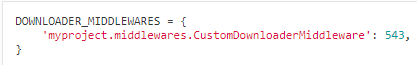
# Downloader Middleware

下载器中间件是Scrapy的请求/响应处理的钩子框架。这是一个轻微的低级系统，用于全球改变Scrapy的请求和响应。

## 激活一个下载中间件

要激活下载器中间件组件，请将其添加到 [**DOWNLOADER\_MIDDLEWARES**](https://doc.scrapy.org/en/latest/topics/settings.html#std:setting-DOWNLOADER_MIDDLEWARES)设置中，该设置是键是中间件类路径的字典，它们的值是中间件订单。

这是一个例子：



该[**DOWNLOADER\_MIDDLEWARES**](https://doc.scrapy.org/en/latest/topics/settings.html#std:setting-DOWNLOADER_MIDDLEWARES)设置与[**DOWNLOADER\_MIDDLEWARES\_BASE**](https://doc.scrapy.org/en/latest/topics/settings.html#std:setting-DOWNLOADER_MIDDLEWARES_BASE)Scrapy中定义的设置合并 （并不意味着被覆盖），然后按顺序排序以获得最终的已启用中间件排序列表：第一个中间件是靠近引擎的中间件，最后一个是靠近引擎的中间件到下载器。换句话说，[**process\_request()**](https://doc.scrapy.org/en/latest/topics/downloader-middleware.html#scrapy.downloadermiddlewares.DownloaderMiddleware.process_request) 每个中间件的方法将以增加中间件的顺序（100,200,300，...）[**process\_response()**](https://doc.scrapy.org/en/latest/topics/downloader-middleware.html#scrapy.downloadermiddlewares.DownloaderMiddleware.process_response)被调用，并且每个中间件的方法将按降序调用。

要决定分配给中间件的顺序，请参阅 [**DOWNLOADER\_MIDDLEWARES\_BASE**](https://doc.scrapy.org/en/latest/topics/settings.html#std:setting-DOWNLOADER_MIDDLEWARES_BASE)设置并根据要插入中间件的位置选择一个值。顺序很重要，因为每个中间件都执行不同的操作，而您的中间件可能依赖于某些以前（或后续）正在应用的中间件。

如果要禁用内置中间件（[**DOWNLOADER\_MIDDLEWARES\_BASE**](https://doc.scrapy.org/en/latest/topics/settings.html#std:setting-DOWNLOADER_MIDDLEWARES_BASE)默认情况下定义和启用的中间件 ），则必须在项目[**DOWNLOADER\_MIDDLEWARES**](https://doc.scrapy.org/en/latest/topics/settings.html#std:setting-DOWNLOADER_MIDDLEWARES)设置 中将其定义并将*None*分配为其值。例如，如果您想禁用用户代理中间件：



最后，请记住，某些中间件可能需要通过特定设置启用。有关更多信息，请参阅每个中间件文档。

## 编写你自己的下载中间件

每个中间件组件都是一个Python类，它定义了以下一种或多种方法：

***Class* scrapy.downloadermiddlewares.DownloaderMiddleware**

**注意**

任何下载器中间件方法也可能返回延迟。.

**process\_request(*request*, *spider*)**

每个通过下载中间件的请求都会调用此方法。

[**process\_request()**](https://doc.scrapy.org/en/latest/topics/downloader-middleware.html#scrapy.downloadermiddlewares.DownloaderMiddleware.process_request)应该：返回**None**，返回一个 [**Response**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.Response)对象，返回一个[**Request**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.Request) 对象，或者提高[**IgnoreRequest**](https://doc.scrapy.org/en/latest/topics/exceptions.html#scrapy.exceptions.IgnoreRequest)。

如果它返回**None**，Scrapy将继续处理这个请求，执行所有其他中间件，直到最后，合适的下载器处理程序被称为执行的请求（以及它的响应下载）。

如果它返回一个[**Response**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.Response)对象，Scrapy不会打扰调用*任何*其他[**process\_request()**](https://doc.scrapy.org/en/latest/topics/downloader-middleware.html#scrapy.downloadermiddlewares.DownloaderMiddleware.process_request)或[**process\_exception()**](https://doc.scrapy.org/en/latest/topics/downloader-middleware.html#scrapy.downloadermiddlewares.DownloaderMiddleware.process_exception)方法，或相应的下载功能; 它会返回该响应。[**process\_response()**](https://doc.scrapy.org/en/latest/topics/downloader-middleware.html#scrapy.downloadermiddlewares.DownloaderMiddleware.process_response) 安装中间件的方法始终在每个响应中调用。

如果它返回一个[**Request**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.Request)对象，Scrapy将停止调用process\_request方法并重新安排返回的请求。一旦执行新返回的请求，将在下载的响应中调用适当的中间件链。

如果引发[**IgnoreRequest**](https://doc.scrapy.org/en/latest/topics/exceptions.html#scrapy.exceptions.IgnoreRequest)异常，[**process\_exception()**](https://doc.scrapy.org/en/latest/topics/downloader-middleware.html#scrapy.downloadermiddlewares.DownloaderMiddleware.process_exception)则会调用已安装的下载器中间件的 方法。如果它们都不处理异常，**Request.errback**则调用request（）的errback函数。如果没有代码处理引发的异常，它将被忽略并且不记录（不像其他异常）。

|  |  |
| --- | --- |
| **参数：** | * **request** ([**Request**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.Request) object)  - 正在处理的请求 * **spider** ([**Spider**](https://doc.scrapy.org/en/latest/topics/spiders.html#scrapy.spiders.Spider) object)  - 这个请求所针对的蜘蛛 |

**process\_response(request, response, spider)**

[**process\_response()**](https://doc.scrapy.org/en/latest/topics/downloader-middleware.html#scrapy.downloadermiddlewares.DownloaderMiddleware.process_response)应该：返回一个[**Response**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.Response) 对象，返回一个[**Request**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.Request)对象或引发[**IgnoreRequest**](https://doc.scrapy.org/en/latest/topics/exceptions.html#scrapy.exceptions.IgnoreRequest)异常。

如果它返回一个[**Response**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.Response)（它可能是相同的响应，或者是一个全新的响应），那么响应将继续与[**process\_response()**](https://doc.scrapy.org/en/latest/topics/downloader-middleware.html#scrapy.downloadermiddlewares.DownloaderMiddleware.process_response)链中的下一个中间件一起处理。

如果它返回一个[**Request**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.Request)对象，则中间件链将暂停，并且返回的请求将被重新计划以备将来下载。这与返回请求的行为相同[**process\_request()**](https://doc.scrapy.org/en/latest/topics/downloader-middleware.html#scrapy.downloadermiddlewares.DownloaderMiddleware.process_request)。

如果引发[**IgnoreRequest**](https://doc.scrapy.org/en/latest/topics/exceptions.html#scrapy.exceptions.IgnoreRequest)异常，**Request.errback**则调用request（）的errback函数。如果没有代码处理引发的异常，它将被忽略并且不记录（不像其他异常）。

|  |  |
| --- | --- |
| **参数：** | * **request** (is a [**Request**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.Request) object)  - 发起响应的请求 * **response** ([**Response**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.Response) object) - 正在处理的响应 * **spider** ([**Spider**](https://doc.scrapy.org/en/latest/topics/spiders.html#scrapy.spiders.Spider) object)  - 这个反应所针对的蜘蛛 |

**process\_exception(request, exception, spider)**

[**process\_exception()**](https://doc.scrapy.org/en/latest/topics/downloader-middleware.html#scrapy.downloadermiddlewares.DownloaderMiddleware.process_exception)当下载处理程序或[**process\_request()**](https://doc.scrapy.org/en/latest/topics/downloader-middleware.html#scrapy.downloadermiddlewares.DownloaderMiddleware.process_request)（从下载程序中间件）引发异常（包括[**IgnoreRequest**](https://doc.scrapy.org/en/latest/topics/exceptions.html#scrapy.exceptions.IgnoreRequest)异常）时，Scrapy会调用它，

[**process\_exception()**](https://doc.scrapy.org/en/latest/topics/downloader-middleware.html#scrapy.downloadermiddlewares.DownloaderMiddleware.process_exception)应该返回：要么**None**，一个[**Response**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.Response)对象或[**Request**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.Request)对象。

如果它返回**None**，Scrapy将继续处理这个异常，执行任何其他[**process\_exception()**](https://doc.scrapy.org/en/latest/topics/downloader-middleware.html#scrapy.downloadermiddlewares.DownloaderMiddleware.process_exception)已安装中间件的方法，直到没有中间件被遗留，并且默认的异常处理开始。

如果它返回一个[**Response**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.Response)对象，那么[**process\_response()**](https://doc.scrapy.org/en/latest/topics/downloader-middleware.html#scrapy.downloadermiddlewares.DownloaderMiddleware.process_response) 已安装中间件的方法链就会启动，Scrapy也不会打扰其他任何[**process\_exception()**](https://doc.scrapy.org/en/latest/topics/downloader-middleware.html#scrapy.downloadermiddlewares.DownloaderMiddleware.process_exception)中间件的调用。

如果它返回一个[**Request**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.Request)对象，则返回的请求将被重新计划以备将来下载。这会停止执行[**process\_exception()**](https://doc.scrapy.org/en/latest/topics/downloader-middleware.html#scrapy.downloadermiddlewares.DownloaderMiddleware.process_exception)中间件的方法，就像返回响应一样。

|  |  |
| --- | --- |
| **参数：** | * **request** (is a [**Request**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.Request) object)  - 生成异常的请求 * **exception** (an **Exception** object)  - 引发的异常 * **spider** ([**Spider**](https://doc.scrapy.org/en/latest/topics/spiders.html#scrapy.spiders.Spider) object) - 这个请求所针对的蜘蛛 |

**from\_crawler(cls, crawler)**

如果存在，这个classmethod被称为从a创建一个中间件实例[**Crawler**](https://doc.scrapy.org/en/latest/topics/api.html#scrapy.crawler.Crawler)。它必须返回一个新的中间件实例。抓取工具对象提供对所有Scrapy核心组件的访问，如设置和信号; 这是中间件访问它们并将其功能挂接到Scrapy的一种方式。

|  |  |
| --- | --- |
| **参数：** | **crawler** ([**Crawler**](https://doc.scrapy.org/en/latest/topics/api.html#scrapy.crawler.Crawler) object)  - 使用此中间件的爬虫 |

## 内置下载中间件参考

本页面描述了Scrapy附带的所有下载器中间件组件。有关如何使用它们以及如何编写自己的下载器中间件的信息，请参阅[下载器中间件使用指南](https://doc.scrapy.org/en/latest/topics/downloader-middleware.html#topics-downloader-middleware)。

有关默认启用的组件列表（及其订单），请参阅该 [**DOWNLOADER\_MIDDLEWARES\_BASE**](https://doc.scrapy.org/en/latest/topics/settings.html#std:setting-DOWNLOADER_MIDDLEWARES_BASE)设置。

### CookiesMiddleware

**Class scrapy.downloadermiddlewares.cookies.CookiesMiddleware**

该中间件可以处理需要cookie的网站，例如那些使用会话的网站。它跟踪由Web服务器发送的cookie，并将其发回（随后从蜘蛛中获取），就像浏览器一样。

以下设置可用于配置Cookie中间件：

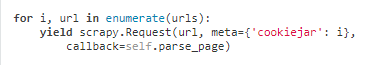
* [**COOKIES\_ENABLED**](https://doc.scrapy.org/en/latest/topics/downloader-middleware.html#std:setting-COOKIES_ENABLED)
* [**COOKIES\_DEBUG**](https://doc.scrapy.org/en/latest/topics/downloader-middleware.html#std:setting-COOKIES_DEBUG)

#### 每个蜘蛛多个Cookie会话

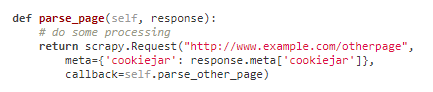
*新版本0.15。*

通过使用[**cookiejar**](https://doc.scrapy.org/en/latest/topics/downloader-middleware.html#std:reqmeta-cookiejar)请求元密钥，支持每个蜘蛛保持多个Cookie会话 。默认情况下，它使用一个cookie jar（会话），但是你可以传递一个标识符来使用不同的标识符。

例如：



请记住，[**cookiejar**](https://doc.scrapy.org/en/latest/topics/downloader-middleware.html#std:reqmeta-cookiejar)元键不是“粘性”的。您需要继续在随后的请求中传递它。例如：



#### COOKIES\_ENABLED

默认： True

是否启用Cookie中间件。如果禁用，则不会将Cookie发送到Web服务器。

注意，如果[**Request**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.Request) 已经meta['dont\_merge\_cookies']评估True。尽管价值[**COOKIES\_ENABLED**](https://doc.scrapy.org/en/latest/topics/downloader-middleware.html#std:setting-COOKIES_ENABLED)饼干将**不会**被发送到Web服务器，并收到饼干 [**Response**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.Response)将**不会**与现有的cookie合并。

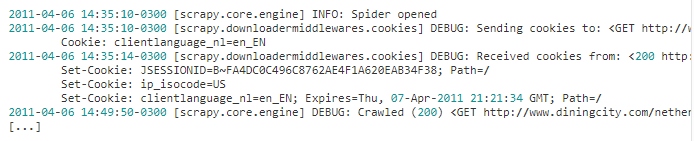
有关更多详细信息，请参阅中的cookies参数 [**Request**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.Request)

#### COOKIES\_DEBUG

默认： False

如果启用，Scrapy会记录在请求中发送的所有Cookie（即Cookie 标题）以及在响应中收到的所有Cookie（即Set-Cookie标题）。

以下是[**COOKIES\_DEBUG**](https://doc.scrapy.org/en/latest/topics/downloader-middleware.html#std:setting-COOKIES_DEBUG)启用日志的示例：



### DefaultHeadersMiddleware

**Class scrapy.downloadermiddlewares.defaultheaders.DefaultHeadersMiddleware**

该中间件设置设置中指定的所有默认请求标头 [**DEFAULT\_REQUEST\_HEADERS**](https://doc.scrapy.org/en/latest/topics/settings.html#std:setting-DEFAULT_REQUEST_HEADERS)。

### DownloadTimeoutMiddleware

**Class scrapy.downloadermiddlewares.downloadtimeout.DownloadTimeoutMiddleware**

该中间件为[**DOWNLOAD\_TIMEOUT**](https://doc.scrapy.org/en/latest/topics/settings.html#std:setting-DOWNLOAD_TIMEOUT)设置或**download\_timeout** spider属性中指定的请求设置下载超时 。

**注意**

您还可以使用[**download\_timeout**](https://doc.scrapy.org/en/latest/topics/request-response.html#std:reqmeta-download_timeout)Request.meta键设置每个请求的下载超时时间 ; 即使在禁用了DownloadTimeoutMiddleware的情况下也支持此功能。

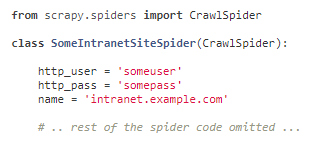
### HttpAuthMiddleware

**Class scrapy.downloadermiddlewares.httpauth.HttpAuthMiddleware**

该中间件使用[基本访问验证](https://en.wikipedia.org/wiki/Basic_access_authentication)（又称HTTP验证）来[验证](https://en.wikipedia.org/wiki/Basic_access_authentication)从某些蜘蛛生成的所有请求。

为了能够从某些蜘蛛HTTP认证，设置**http\_user** 和**http\_pass**那些蜘蛛属性。

例：



### HttpCacheMiddleware

**Class scrapy.downloadermiddlewares.httpcache.HttpCacheMiddleware**

该中间件为所有HTTP请求和响应提供低级缓存。它必须与缓存存储后端以及缓存策略结合使用。

Scrapy附带三个HTTP缓存存储后端：

* [文件系统存储后端（默认）](https://doc.scrapy.org/en/latest/topics/downloader-middleware.html#httpcache-storage-fs)
* [DBM存储后端](https://doc.scrapy.org/en/latest/topics/downloader-middleware.html#httpcache-storage-dbm)
* [LevelDB存储后端](https://doc.scrapy.org/en/latest/topics/downloader-middleware.html#httpcache-storage-leveldb)

您可以使用该[**HTTPCACHE\_STORAGE**](https://doc.scrapy.org/en/latest/topics/downloader-middleware.html#std:setting-HTTPCACHE_STORAGE) 设置更改HTTP缓存存储后端。或者你也可以实现你自己的存储后端。

Scrapy附带两个HTTP缓存策略：

* [RFC2616政策](https://doc.scrapy.org/en/latest/topics/downloader-middleware.html#httpcache-policy-rfc2616)
* [虚拟策略（默认）](https://doc.scrapy.org/en/latest/topics/downloader-middleware.html#httpcache-policy-dummy)

您可以使用该[**HTTPCACHE\_POLICY**](https://doc.scrapy.org/en/latest/topics/downloader-middleware.html#std:setting-HTTPCACHE_POLICY) 设置更改HTTP缓存策略。或者你也可以实施你自己的政策。

您还可以避免使用[**dont\_cache**](https://doc.scrapy.org/en/latest/topics/downloader-middleware.html#std:reqmeta-dont_cache)meta key等于True在每个策略上缓存响应。

#### 虚拟策略（默认）

该策略不了解任何HTTP Cache-Control指令。每个请求及其相应的响应都被缓存。当再次看到相同的请求时，将返回响应而不从Internet传输任何内容。

Dummy策略对于更快地测试蜘蛛（无需每次都等待下载）以及当Internet连接不可用时尝试使蜘蛛离线都很有用。我们的目标是能够像以前一样“重播”蜘蛛跑。

为了使用此策略，请设置：



#### RFC2616政策

该策略提供符合RFC2616的HTTP缓存，即HTTP缓存控制感知，旨在生产并用于连续运行，以避免下载未修改的数据（以节省带宽并加速爬网）。

实施内容：

* 不要尝试存储没有存储缓存控制指令集的响应/请求
* 如果即使对于新的响应设置了no-cache cache-control指令，也不要从高速缓存提供响应
* 根据最大年龄缓存控制指令计算新鲜度寿命
* 计算Expires响应标题的新鲜度生命周期
* 从Last-Modified响应标头计算新鲜生命期（Firefox使用的启发式）
* 从年龄响应标题计算当前年龄
* 从日期标题计算当前时间
* 根据Last-Modified响应标题重新验证陈旧的响应
* 基于ETag响应头重新验证陈旧的响应
* 为任何收到的响应设置日期标题丢失它
* 在请求中支持最大陈旧的缓存控制指令

这使得蜘蛛可以使用完整的RFC2616缓存策略进行配置，但是可以避免在逐个请求的基础上进行重新验证，同时保持与HTTP规范的一致性。

例：

添加缓存控制：max-stale = 600以请求标头接受超过其过期时间不超过600秒的响应。

另见：RFC2616,14.9.3

什么不见​​了：

* Pragma：无缓存支持<https://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html#sec14.9.1>
* 多种头文件支持<https://www.w3.org/Protocols/rfc2616/rfc2616-sec13.html#sec13.6>
* 更新或删除后无效<https://www.w3.org/Protocols/rfc2616/rfc2616-sec13.html#sec13.10>
* ...可能是其他人..

为了使用此策略，请设置：



#### 文件系统存储后端（默认）

文件系统存储后端可用于HTTP缓存中间件。

为了使用此存储后端，请设置：



每个请求/响应对都存储在包含以下文件的不同目录中：

* request\_body - 简单的请求主体
* request\_headers - 请求标题（原始HTTP格式）
* response\_body - 简单的回应机构
* response\_headers - 请求标题（原始HTTP格式）
* meta- Python repr()格式的这种缓存资源的一些元数据（grep-friendly格式）
* pickled\_meta- meta为了更有效的反序列化而腌制相同的元数据

目录名称由请求指纹（请参见参考资料scrapy.utils.request.fingerprint）制作而成 ，并且使用一级子目录以避免将太多文件创建到同一目录（这在许多文件系统中效率低下）。示例目录可以是：



#### DBM存储后端

*0.13版本的新功能。*

一个[DBM](https://en.wikipedia.org/wiki/Dbm)存储后端也可用于HTTP缓存中间件。

默认情况下，它使用[anydbm](https://docs.python.org/2/library/anydbm.html)模块，但可以使用该[**HTTPCACHE\_DBM\_MODULE**](https://doc.scrapy.org/en/latest/topics/downloader-middleware.html#std:setting-HTTPCACHE_DBM_MODULE)设置对其进行更改 。

为了使用此存储后端，请设置：



#### LevelDB存储后端

*0.23版本的新功能。*

一个[性LevelDB](https://github.com/google/leveldb)存储后端也可用于HTTP缓存中间件。

由于只有一个进程可以同时访问LevelDB数据库，因此不推荐将此后端用于开发，因此不能同时为同一个蜘蛛运行爬网并打开scrapy shell。

为了使用这个存储后端：

* 设置[**HTTPCACHE\_STORAGE**](https://doc.scrapy.org/en/latest/topics/downloader-middleware.html#std:setting-HTTPCACHE_STORAGE)为scrapy.extensions.httpcache.LeveldbCacheStorage
* 像安装[LevelDB python绑定](https://pypi.python.org/pypi/leveldb)一样pip install leveldb

#### HTTPCache中间件设置

的[**HttpCacheMiddleware**](https://doc.scrapy.org/en/latest/topics/downloader-middleware.html#scrapy.downloadermiddlewares.httpcache.HttpCacheMiddleware)可以通过以下设置来配置：

##### HTTPCACHE\_ENABLED

*0.11版本的新功能。*

默认： False

是否启用HTTP缓存。

*在版本0.11中更改：*在0.11之前，[**HTTPCACHE\_DIR**](https://doc.scrapy.org/en/latest/topics/downloader-middleware.html#std:setting-HTTPCACHE_DIR)用于启用缓存。

##### HTTPCACHE\_EXPIRATION\_SECS

默认： 0

缓存请求的到期时间，以秒为单位。

比这次更早的缓存请求将被重新下载。如果为零，则高速缓存的请求将永不过期。

*在版本0.11中更改：*在0.11之前，零意味着缓存的请求总是过期。

##### HTTPCACHE\_DIR

默认： 'httpcache'

用于存储（低级别）HTTP缓存的目录。如果为空，则HTTP缓存将被禁用。如果给出了相对路径，则相对于项目数据目录进行。欲了解更多信息，请参阅：[Scrapy项目的默认结构](https://doc.scrapy.org/en/latest/topics/commands.html#topics-project-structure)。

##### HTTPCACHE\_IGNORE\_HTTP\_CODES

*0.10版本中的新功能。*

默认： []

不要用这些HTTP代码缓存响应。

##### HTTPCACHE\_IGNORE\_MISSING

默认： False

如果启用，则在缓存中找不到的请求将被忽略而不是下载。

##### HTTPCACHE\_IGNORE\_SCHEMES

*0.10版本中的新功能。*

默认： ['file']

不要使用这些URI方案缓存响应。

##### HTTPCACHE\_STORAGE

默认： 'scrapy.extensions.httpcache.FilesystemCacheStorage'

实现高速缓存存储后端的类。

##### HTTPCACHE\_DBM\_MODULE

*0.13版本的新功能。*

默认： 'anydbm'

用于[DBM存储后端](https://doc.scrapy.org/en/latest/topics/downloader-middleware.html#httpcache-storage-dbm)的数据库模块。此设置特定于DBM后端。

##### HTTPCACHE\_POLICY

*0.18版本中的新功能。*

默认： 'scrapy.extensions.httpcache.DummyPolicy'

实现缓存策略的类。

##### HTTPCACHE\_GZIP

*1.0版中的新功能。*

默认： False

如果启用，将使用gzip压缩所有缓存的数据。该设置特定于文件系统后端。

##### HTTPCACHE\_ALWAYS\_STORE

*版本1.1中的新功能*

默认： False

如果启用，将无条件缓存页面。

例如，蜘蛛可能希望在缓存中提供所有响应，以便将来与Cache-Control一起使用：max-stale。DummyPolicy缓存所有响应，但从不重新验证它们，有时需要更细致的策略。

该设置仍然遵循Cache-Control：响应中的无存储指令。如果您不希望这样，请在您提供给缓存中间件的响应中过滤缓存控制标头中的no-store。

##### HTTPCACHE\_IGNORE\_RESPONSE\_CACHE\_CONTROLS

*版本1.1中的新功能*

默认： []

缓存控制指令列表中的响应被忽略。

网站通常会设置“无存储”，“无缓存”，“必须重新验证”等，但如果遵守这些指令，蜘蛛可能产生的流量就会变得不安。这允许有选择地忽略已知对于被爬网的站点不重要的Cache-Control指令。

我们假设蜘蛛不会在请求中发出Cache-Control指令，除非它真的需要它们，所以请求中的指令不会被过滤。

### HttpCompressionMiddleware

**Class scrapy.downloadermiddlewares.httpcompression.HttpCompressionMiddleware**

该中间件允许从网站发送/接收压缩（gzip，deflate）流量。

只要安装了[brotlipy](https://pypi.python.org/pypi/brotlipy)，该中间件也支持解码[brotli压缩的](https://www.ietf.org/rfc/rfc7932.txt)响应。

#### HttpCompressionMiddleware Settings

##### COMPRESSION\_ENABLED

默认： True

压缩中间件是否启用。

### HttpProxyMiddleware

*0.8版新增功能*

**Class scrapy.downloadermiddlewares.httpproxy.HttpProxyMiddleware**

该中间件通过设置对象的**proxy**元值来设置HTTP代理以用于请求 [**Request**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.Request)。

像Python标准库模块[urllib](https://docs.python.org/2/library/urllib.html)和[urllib2一样](https://docs.python.org/2/library/urllib2.html)，它遵循以下环境变量：

* **http\_proxy**
* **https\_proxy**
* **no\_proxy**

您还可以将**proxy**每个请求的元键设置为像**http://some\_proxy\_server:port**or 的值**http://username:password@some\_proxy\_server:port**。请记住，该值将优先于**http\_proxy**/ **https\_proxy**环境变量，并且也会忽略**no\_proxy**环境变量。

### RedirectMiddleware

**Class scrapy.downloadermiddlewares.redirect.RedirectMiddleware**

该中间件根据响应状态处理请求的重定向。

请求经过的URL（被重定向）可以在redirect\_urls [**Request.meta**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.Request.meta)密钥中找到。

该[**RedirectMiddleware**](https://doc.scrapy.org/en/latest/topics/downloader-middleware.html#scrapy.downloadermiddlewares.redirect.RedirectMiddleware)可通过以下设置进行配置（详情参见设置文档）：

* [**REDIRECT\_ENABLED**](https://doc.scrapy.org/en/latest/topics/downloader-middleware.html#std:setting-REDIRECT_ENABLED)
* [**REDIRECT\_MAX\_TIMES**](https://doc.scrapy.org/en/latest/topics/settings.html#std:setting-REDIRECT_MAX_TIMES)

如果[**Request.meta**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.Request.meta)将dont\_redirect 键设置为True，则该中间件将忽略该请求。

如果你想在蜘蛛中处理一些重定向状态码，你可以在handle\_httpstatus\_list蜘蛛属性中指定这些。

例如，如果您希望重定向中间件忽略301和302响应（并将它们传递给您的蜘蛛），您可以这样做：



所述handle\_httpstatus\_list的键[**Request.meta**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.Request.meta)也可以被用于指定的响应代码，以允许在每个请求基础。您还可以设置meta键 handle\_httpstatus\_all来True，如果你想以允许请求的任何响应代码。

#### RedirectMiddleware设置

##### REDIRECT\_ENABLED

*0.13版本的新功能。*

默认： True

是否启用重定向中间件。

##### REDIRECT\_MAX\_TIMES

默认： 20

单个请求将遵循的最大重定向次数。

### MetaRefreshMiddleware

**Class scrapy.downloadermiddlewares.redirect.MetaRefreshMiddleware**

该中间件处理基于元刷新html标签的请求重定向。

该[**MetaRefreshMiddleware**](https://doc.scrapy.org/en/latest/topics/downloader-middleware.html#scrapy.downloadermiddlewares.redirect.MetaRefreshMiddleware)可通过以下设置进行配置（详情参见设置文档）：

* [**METAREFRESH\_ENABLED**](https://doc.scrapy.org/en/latest/topics/downloader-middleware.html#std:setting-METAREFRESH_ENABLED)
* [**METAREFRESH\_MAXDELAY**](https://doc.scrapy.org/en/latest/topics/downloader-middleware.html#std:setting-METAREFRESH_MAXDELAY)

该中间件服从[**REDIRECT\_MAX\_TIMES**](https://doc.scrapy.org/en/latest/topics/settings.html#std:setting-REDIRECT_MAX_TIMES)设置，[**dont\_redirect**](https://doc.scrapy.org/en/latest/topics/downloader-middleware.html#std:reqmeta-dont_redirect) 并[**redirect\_urls**](https://doc.scrapy.org/en/latest/topics/downloader-middleware.html#std:reqmeta-redirect_urls)按照所述的要求请求元键[**RedirectMiddleware**](https://doc.scrapy.org/en/latest/topics/downloader-middleware.html#scrapy.downloadermiddlewares.redirect.RedirectMiddleware)

#### MetaRefreshMiddleware设置

##### METAREFRESH\_ENABLED

*0.17版本中的新功能。*

默认： True

是否启用Meta Refresh中间件。

##### METAREFRESH\_MAXDELAY

默认： 100

遵循重定向的最大元刷新延迟（以秒为单位）。某些站点使用元刷新重定向到会话过期页面，因此我们将自动重定向限制为最大延迟。

### RetryMiddleware

**Class scrapy.downloadermiddlewares.retry.RetryMiddleware**

一个中间件，用于重试可能由临时问题（如连接超时或HTTP 500错误）导致的失败请求。

在抓取过程中收集失败页面，并在蜘蛛抓取完所有常规（非失败）页面后结束重新安排。一旦没有更多的失败页面重试，这个中间件发送一个信号（retry\_complete），所以其他扩展可以连接到该信号。

该[**RetryMiddleware**](https://doc.scrapy.org/en/latest/topics/downloader-middleware.html#scrapy.downloadermiddlewares.retry.RetryMiddleware)可通过以下设置进行配置（详情参见设置文档）：

* [**RETRY\_ENABLED**](https://doc.scrapy.org/en/latest/topics/downloader-middleware.html#std:setting-RETRY_ENABLED)
* [**RETRY\_TIMES**](https://doc.scrapy.org/en/latest/topics/downloader-middleware.html#std:setting-RETRY_TIMES)
* [**RETRY\_HTTP\_CODES**](https://doc.scrapy.org/en/latest/topics/downloader-middleware.html#std:setting-RETRY_HTTP_CODES)

如果[**Request.meta**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.Request.meta)将dont\_retry键设置为True，则该中间件将忽略该请求。

#### 重试中间件设置

##### RETRY\_ENABLED

*0.13版本的新功能。*

默认： True

是否启用重试​​中间件。

##### RETRY\_TIMES

默认： 2

除了第一次下载之外，重试的最大次数。

也可以使用[**max\_retry\_times**](https://doc.scrapy.org/en/latest/topics/request-response.html#std:reqmeta-max_retry_times)属性为每个请求指定最大重试次数 [**Request.meta**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.Request.meta)。初始化时，[**max\_retry\_times**](https://doc.scrapy.org/en/latest/topics/request-response.html#std:reqmeta-max_retry_times)元键优先于[**RETRY\_TIMES**](https://doc.scrapy.org/en/latest/topics/downloader-middleware.html#std:setting-RETRY_TIMES)设置。

##### RETRY\_HTTP\_CODES

默认： [500, 502, 503, 504, 408]

哪个HTTP响应代码要重试。其他错误（DNS查找问题，连接丢失等）总是重试。

在某些情况下，您可能需要添加400，[**RETRY\_HTTP\_CODES**](https://doc.scrapy.org/en/latest/topics/downloader-middleware.html#std:setting-RETRY_HTTP_CODES)因为它是用于指示服务器过载的通用代码。它没有默认包含，因为HTTP规范是这样说的。

### RobotsTxtMiddleware

**Class scrapy.downloadermiddlewares.robotstxt.RobotsTxtMiddleware**

该中间件过滤掉robots.txt排除标准禁止的请求。

为确保Scrapy尊重robots.txt，请确保中间件已启用且[**ROBOTSTXT\_OBEY**](https://doc.scrapy.org/en/latest/topics/settings.html#std:setting-ROBOTSTXT_OBEY)设置已启用。

如果[**Request.meta**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.Request.meta)将 dont\_obey\_robotstxt键设置为True，则即使[**ROBOTSTXT\_OBEY**](https://doc.scrapy.org/en/latest/topics/settings.html#std:setting-ROBOTSTXT_OBEY)启用该中间件，该请求也将被忽略 。

### DownloaderStats

**Class scrapy.downloadermiddlewares.stats.DownloaderStats**

存储所有通过它的请求，响应和异常的统计信息的中间件。

要使用此中间件，您必须启用该[**DOWNLOADER\_STATS**](https://doc.scrapy.org/en/latest/topics/settings.html#std:setting-DOWNLOADER_STATS) 设置。

### UserAgentMiddleware

**Class scrapy.downloadermiddlewares.useragent.UserAgentMiddleware**

允许蜘蛛覆盖默认用户代理的中间件。

为了使蜘蛛覆盖默认用户代理， 必须设置其user\_agent属性。

### AjaxCrawlMiddleware

**Class scrapy.downloadermiddlewares.ajaxcrawl.AjaxCrawlMiddleware**

根据meta-fragment html标签查找“AJAX可抓取”页面变体的中间件。有关 更多信息，请参阅<https://developers.google.com/webmasters/ajax-crawling/docs/getting-started>。

**注意**

Scrapy发现了“AJAX可抓取”页面， **'http://example.com/!#foo=bar'**即使没有这个中间件也是如此。当URL不包含时，AjaxCrawlMiddleware是必需的**'!#'**。这通常是'索引'或'主'网站页面的情况。

#### AjaxCrawlMiddleware设置

##### AJAXCRAWL\_ENABLED

*新版本0.21。*

默认： False

是否启用AjaxCrawlMiddleware。您可能需要启用它以进行[广泛抓取](https://doc.scrapy.org/en/latest/topics/broad-crawls.html#topics-broad-crawls)。

#### HttpProxyMiddleware设置

##### HTTPPROXY\_ENABLED

默认： True

是否启用**HttpProxyMiddleware**。

##### HTTPPROXY\_AUTH\_ENCODING

默认： "latin-1"

代理身份验证的默认编码**HttpProxyMiddleware**。

# Spider Middleware

蜘蛛中间件是Scrapy的蜘蛛处理机制的钩子框架，您可以插入自定义功能来处理发送给[蜘蛛](https://doc.scrapy.org/en/latest/topics/spiders.html#topics-spiders)进行处理的响应，并处理从蜘蛛生成的请求和项目。

## 激活蜘蛛中间件

要激活蜘蛛中间件组件，将其添加到 [**SPIDER\_MIDDLEWARES**](https://doc.scrapy.org/en/latest/topics/settings.html#std:setting-SPIDER_MIDDLEWARES)设置中，该设置是一个字典，其键是中间件类路径，它们的值是中间件命令。

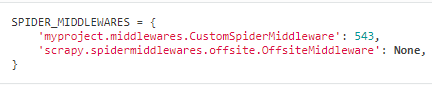
这是一个例子：



该[**SPIDER\_MIDDLEWARES**](https://doc.scrapy.org/en/latest/topics/settings.html#std:setting-SPIDER_MIDDLEWARES)设置与[**SPIDER\_MIDDLEWARES\_BASE**](https://doc.scrapy.org/en/latest/topics/settings.html#std:setting-SPIDER_MIDDLEWARES_BASE)Scrapy中定义的设置合并 （并不意味着被覆盖），然后按顺序排序以获得最终的已启用中间件排序列表：第一个中间件是靠近引擎的中间件，最后一个是靠近引擎的中间件到蜘蛛。换句话说，[**process\_spider\_input()**](https://doc.scrapy.org/en/latest/topics/spider-middleware.html#scrapy.spidermiddlewares.SpiderMiddleware.process_spider_input) 每个中间件的方法将以增加中间件的顺序（100,200,300，...）[**process\_spider\_output()**](https://doc.scrapy.org/en/latest/topics/spider-middleware.html#scrapy.spidermiddlewares.SpiderMiddleware.process_spider_output)被调用，并且每个中间件的 方法将按降序调用。

要决定分配给中间件的顺序，请参阅 [**SPIDER\_MIDDLEWARES\_BASE**](https://doc.scrapy.org/en/latest/topics/settings.html#std:setting-SPIDER_MIDDLEWARES_BASE)设置并根据要插入中间件的位置选择一个值。顺序很重要，因为每个中间件都执行不同的操作，而您的中间件可能依赖于某些以前（或后续）正在应用的中间件。

如果要禁用内置中间件（[**SPIDER\_MIDDLEWARES\_BASE**](https://doc.scrapy.org/en/latest/topics/settings.html#std:setting-SPIDER_MIDDLEWARES_BASE)默认情况下定义的内置中间件 ），则必须在项目[**SPIDER\_MIDDLEWARES**](https://doc.scrapy.org/en/latest/topics/settings.html#std:setting-SPIDER_MIDDLEWARES)设置中定义它，并将*None*指定为其值。例如，如果您想禁用场外中间件：



最后，请记住，某些中间件可能需要通过特定设置启用。有关更多信息，请参阅每个中间件文档。

## 编写你自己的蜘蛛中间件

每个中间件组件都是一个Python类，它定义了以下一种或多种方法：

**Class scrapy.spidermiddlewares.SpiderMiddleware**

**process\_spider\_input(response, spider)**

这种方法被称为每一个通过蜘蛛中间件和蜘蛛进行处理的响应。

[**process\_spider\_input()**](https://doc.scrapy.org/en/latest/topics/spider-middleware.html#scrapy.spidermiddlewares.SpiderMiddleware.process_spider_input)应该返回**None**或引发异常。

如果它返回**None**，Scrapy将继续处理这个响应，执行所有其他中间件，直到最终将响应交给蜘蛛进行处理。

如果它引发异常，Scrapy将不会打扰任何其他蜘蛛中间件，[**process\_spider\_input()**](https://doc.scrapy.org/en/latest/topics/spider-middleware.html#scrapy.spidermiddlewares.SpiderMiddleware.process_spider_input)并会调用请求errback。errback的输出在另一个方向上被链接[**process\_spider\_output()**](https://doc.scrapy.org/en/latest/topics/spider-middleware.html#scrapy.spidermiddlewares.SpiderMiddleware.process_spider_output)以处理它，或者[**process\_spider\_exception()**](https://doc.scrapy.org/en/latest/topics/spider-middleware.html#scrapy.spidermiddlewares.SpiderMiddleware.process_spider_exception)如果它引发异常。

|  |  |
| --- | --- |
| **参数：** | * **response** ([**Response**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.Response) object)  - 正在处理的响应 * **spider** ([**Spider**](https://doc.scrapy.org/en/latest/topics/spiders.html#scrapy.spiders.Spider) object) - 这个反应所针对的蜘蛛 |

**process\_spider\_output(response, result, spider)**

该方法在Spider处理完响应后调用返回的结果。

[**process\_spider\_output()**](https://doc.scrapy.org/en/latest/topics/spider-middleware.html#scrapy.spidermiddlewares.SpiderMiddleware.process_spider_output)必须返回可迭代的 [**Request**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.Request)，字典或[**Item**](https://doc.scrapy.org/en/latest/topics/items.html#scrapy.item.Item) 对象。

|  |  |
| --- | --- |
| **参数：** | * **response** ([**Response**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.Response) object) - 从蜘蛛生成此输出的响应 * **result** (an iterable of [**Request**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.Request), dict or [**Item**](https://doc.scrapy.org/en/latest/topics/items.html#scrapy.item.Item) objects) - 蜘蛛返回的结果 * **spider** ([**Spider**](https://doc.scrapy.org/en/latest/topics/spiders.html#scrapy.spiders.Spider) object)  - 结果正在处理的蜘蛛 |

**process\_spider\_exception(response, exception, spider)**

当蜘蛛或[**process\_spider\_input()**](https://doc.scrapy.org/en/latest/topics/spider-middleware.html#scrapy.spidermiddlewares.SpiderMiddleware.process_spider_input) 方法（来自其他蜘蛛中间件）引发异常时调用此方法。

[**process\_spider\_exception()**](https://doc.scrapy.org/en/latest/topics/spider-middleware.html#scrapy.spidermiddlewares.SpiderMiddleware.process_spider_exception)应该返回一个**None**或一个迭代[**Response**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.Response)，字典或 [**Item**](https://doc.scrapy.org/en/latest/topics/items.html#scrapy.item.Item)对象。

如果它返回**None**，Scrapy将继续处理这个异常，[**process\_spider\_exception()**](https://doc.scrapy.org/en/latest/topics/spider-middleware.html#scrapy.spidermiddlewares.SpiderMiddleware.process_spider_exception)在下面的中间件组件中执行任何其他的事件，直到没有中间件组件被遗留并且异常到达引擎（它被记录和丢弃的地方）。

如果它返回一个可迭代的[**process\_spider\_output()**](https://doc.scrapy.org/en/latest/topics/spider-middleware.html#scrapy.spidermiddlewares.SpiderMiddleware.process_spider_output)管道，[**process\_spider\_exception()**](https://doc.scrapy.org/en/latest/topics/spider-middleware.html#scrapy.spidermiddlewares.SpiderMiddleware.process_spider_exception)则不会调用其他管道。

|  |  |
| --- | --- |
| **参数：** | * **response** ([**Response**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.Response) object) - 引发异常时处理的响应 * **exception** ([Exception](https://docs.python.org/2/library/exceptions.html#exceptions.Exception) object) - 引发的异常 * **spider** ([**Spider**](https://doc.scrapy.org/en/latest/topics/spiders.html#scrapy.spiders.Spider) object) - 引发异常的蜘蛛 |

**process\_start\_requests(start\_requests, spider)**

*新版本0.15。*

该方法使用spider的启动请求调用，并且与该[**process\_spider\_output()**](https://doc.scrapy.org/en/latest/topics/spider-middleware.html#scrapy.spidermiddlewares.SpiderMiddleware.process_spider_output)方法的工作方式类似，只是它没有关联的响应，并且只能返回请求（不是项目）。

它接收一个iterable（在**start\_requests**参数中）并且必须返回另一个可迭代的[**Request**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.Request)对象。

**注意**

在蜘蛛中间件中实现此方法时，应始终返回一个可迭代的（跟随输入的）并且不会消耗所有的**start\_requests**迭代器，因为它可能非常大（甚至无界）并导致内存溢出。Scrapy引擎设计用于在启动请求有能力处理启动请求时启动，因此启动请求迭代器可以在停止蜘蛛的某些其他条件（如时间限制或项目/页数）的情况下实现无限循环。

|  |  |
| --- | --- |
| **参数：** | * **start\_requests** (an iterable of [**Request**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.Request))  - 启动请求 * **spider** ([**Spider**](https://doc.scrapy.org/en/latest/topics/spiders.html#scrapy.spiders.Spider) object) - 启动请求所属的蜘蛛 |

**from\_crawler(cls, crawler)**

如果存在，这个classmethod被称为从a创建一个中间件实例[**Crawler**](https://doc.scrapy.org/en/latest/topics/api.html#scrapy.crawler.Crawler)。它必须返回一个新的中间件实例。抓取工具对象提供对所有Scrapy核心组件的访问，如设置和信号; 这是中间件访问它们并将其功能挂接到Scrapy的一种方式。

|  |  |
| --- | --- |
| **参数：** | **crawler** ([**Crawler**](https://doc.scrapy.org/en/latest/topics/api.html#scrapy.crawler.Crawler) object)  - 使用此中间件的爬虫 |

## 内置蜘蛛中间件参考

本页面描述了Scrapy附带的所有蜘蛛中间件组件。有关如何使用它们以及如何编写自己的蜘蛛中间件的信息，请参阅[蜘蛛中间件使用指南](https://doc.scrapy.org/en/latest/topics/spider-middleware.html#topics-spider-middleware)。

有关默认启用的组件列表（及其订单），请参阅该 [**SPIDER\_MIDDLEWARES\_BASE**](https://doc.scrapy.org/en/latest/topics/settings.html#std:setting-SPIDER_MIDDLEWARES_BASE)设置。

### DepthMiddleware

**Class scrapy.spidermiddlewares.depth.DepthMiddleware**

DepthMiddleware用于跟踪被抓取站点内每个请求的深度。它通过设置request.meta ['depth'] = 0来工作，只要没有先前设置的值（通常只是第一个请求）并以1递增即可。

它可以用来限制最大深度来刮取，根据它们的深度控制请求优先级，以及类似的事情。

该[**DepthMiddleware**](https://doc.scrapy.org/en/latest/topics/spider-middleware.html#scrapy.spidermiddlewares.depth.DepthMiddleware)可通过以下设置进行配置（详情参见设置文档）：

* [**DEPTH\_LIMIT**](https://doc.scrapy.org/en/latest/topics/settings.html#std:setting-DEPTH_LIMIT) - 允许任何网站抓取的最大深度。如果为零，则不会施加限制。
* [**DEPTH\_STATS**](https://doc.scrapy.org/en/latest/topics/settings.html#std:setting-DEPTH_STATS) - 是否收集深度统计。
* [**DEPTH\_PRIORITY**](https://doc.scrapy.org/en/latest/topics/settings.html#std:setting-DEPTH_PRIORITY) - 是否根据深度优先处理请求。

### HttpErrorMiddleware

**Class scrapy.spidermiddlewares.httperror.HttpErrorMiddleware**

过滤掉不成功的（错误的）HTTP响应，这样蜘蛛就不必处理它们，（大部分时间）会产生开销，消耗更多资源，并使蜘蛛逻辑更加复杂。

根据[HTTP标准](https://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html)，成功的响应是那些状态代码在200-300范围内的响应。

如果您仍想处理该范围之外的响应代码，则可以使用handle\_httpstatus\_listspider属性或[**HTTPERROR\_ALLOWED\_CODES**](https://doc.scrapy.org/en/latest/topics/spider-middleware.html#std:setting-HTTPERROR_ALLOWED_CODES)设置来指定蜘蛛能够处理哪些响应代码 。

例如，如果你想让蜘蛛来处理404响应，你可以这样做：



所述handle\_httpstatus\_list的键[**Request.meta**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.Request.meta)也可以被用于指定的响应代码，以允许在每个请求基础。您还可以设置meta键handle\_httpstatus\_all 来True，如果你想以允许请求的任何响应代码。

但请记住，除非你真的知道你在做什么，否则处理非200响应通常是一个坏主意。

有关更多信息，请参阅：[HTTP状态代码定义](https://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html)。

#### HttpErrorMiddleware设置

##### HTTPERROR\_ALLOWED\_CODES

默认： []

通过此列表中包含的非200状态代码的所有响应。

##### HTTPERROR\_ALLOW\_ALL

默认： False

无论其状态码如何，都可以通过所有回复。

### OffsiteMiddleware

**Class scrapy.spidermiddlewares.offsite.OffsiteMiddleware**

过滤蜘蛛所涉及域之外的URL请求。

该中间件过滤掉每个请求的主机名称不在蜘蛛的[**allowed\_domains**](https://doc.scrapy.org/en/latest/topics/spiders.html#scrapy.spiders.Spider.allowed_domains)属性中。列表中任何域的所有子域也都是允许的。例如，规则**www.example.org**也允许**bob.www.example.org** ，但不能**www2.example.com**也不**example.com**。

当你的蜘蛛返回一个不属于蜘蛛覆盖的域的请求时，这个中间件会记录一条类似于这个的调试信息：

DEBUG: Filtered offsite request to 'www.othersite.com': **<**GET http:**//**www**.**othersite**.**com**/**some**/**page**.**html**>**

为了避免过多的噪音填充日志，它只会为每个过滤的新域打印这些消息之一。因此，例如，如果**www.othersite.com**过滤另一个请求，则不会打印日志消息。但是，如果**someothersite.com**过滤请求，则会打印一条消息（但仅限于过滤的第一个请求）。

如果蜘蛛没有定义 [**allowed\_domains**](https://doc.scrapy.org/en/latest/topics/spiders.html#scrapy.spiders.Spider.allowed_domains)属性，或者该属性为空，则非现场中间件将允许所有请求。

如果请求具有**dont\_filter**属性集，则即使其域未在允许的域中列出，异地中间件也将允许该请求。

### RefererMiddleware

**Class scrapy.spidermiddlewares.referer.RefererMiddleware**

**Referer**根据生成它的Response的URL 填充Request 头。

#### RefererMiddleware设置

##### REFERER\_ENABLED

*新版本0.15。*

默认： True

是否启用引用中间件。

##### REFERRER\_POLICY

*1.4版本中的新功能。*

默认： 'scrapy.spidermiddlewares.referer.DefaultReferrerPolicy'

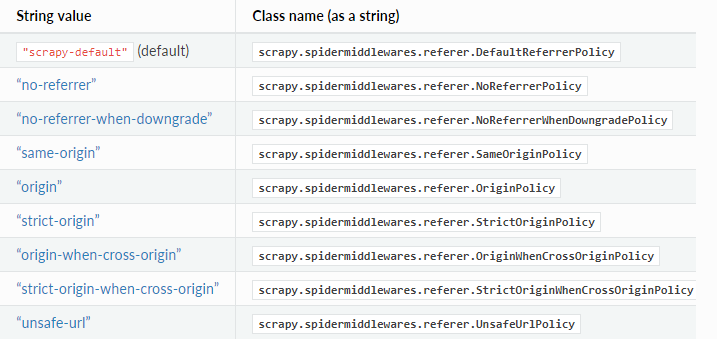
[引荐者策略](https://www.w3.org/TR/referrer-policy)在填充请求“引用者”标题时应用。

**注意**

您还可以使用特殊的"referrer\_policy" [Request.meta](https://doc.scrapy.org/en/latest/topics/request-response.html#topics-request-meta)键为每个请求设置Referrer Policy ，其REFERRER\_POLICY设置的值与可接受的值相同。

###### REFERRER\_POLICY的可接受值

* 要么是一个scrapy.spidermiddlewares.referer.ReferrerPolicy 子类的路径- 一个自定义策略或一个内置的策略（见下面的类），
* 或者标准的W3C定义的字符串值之一，
* 或特殊的"scrapy-default"。



**Class scrapy.spidermiddlewares.referer.DefaultReferrerPolicy**

“降级时不使用引用者”的变体，并且在父请求正在使用**file://**或**s3://**计划时不会发送“引用者” 。

**警告**

Scrapy的默认引用策略 - 就像W3C推荐的浏览器值- [“no-referrer-when-downgrade”一样](https://www.w3.org/TR/referrer-policy/#referrer-policy-no-referrer-when-downgrade)，即使域不同，它也会从http(s)://任意https://URL 向任何URL 发送非空的“Referer”头。

如果您想删除跨域请求的引荐来源信息，[“同源”](https://www.w3.org/TR/referrer-policy/#referrer-policy-same-origin)可能是更好的选择。

**Class scrapy.spidermiddlewares.referer.NoReferrerPolicy**

<https://www.w3.org/TR/referrer-policy/#referrer-policy-no-referrer>

最简单的策略是“禁止引用者”，它指定不引用任何引用信息以及从特定请求客户端向任何源发出的请求。标题将被完全省略。

**Class scrapy.spidermiddlewares.referer.NoReferrerWhenDowngradePolicy**

<https://www.w3.org/TR/referrer-policy/#referrer-policy-no-referrer-when-downgrade>

“no-referrer-when-downgrade”策略将完整的URL以及来自受TLS保护的环境设置对象的请求发送到可能可信的URL以及来自客户端的未受TLS保护的来自任何来源的请求。

另一方面，从受TLS保护的客户端请求非潜在可信的URL将不包含引荐信息。Referer HTTP头将不会被发送。

如果没有另外指定策略，这是用户代理的默认行为。

**注意**

“no-referrer-when-downgrade”策略是W3C推荐的默认策略，并被主要的Web浏览器使用。

但是，这不是Scrapy的默认引用者策略（请参阅参考资料[**DefaultReferrerPolicy**](https://doc.scrapy.org/en/latest/topics/spider-middleware.html#scrapy.spidermiddlewares.referer.DefaultReferrerPolicy)）。

**Class scrapy.spidermiddlewares.referer.SameOriginPolicy**

<https://www.w3.org/TR/referrer-policy/#referrer-policy-same-origin>

“相同来源”政策规定，在从特定请求客户端发出同源请求时，会将作为引用者使用的完整URL作为引用者信息发送。

另一方面，跨境请求将不包含引荐来源信息。Referer HTTP头将不会被发送。

**Class scrapy.spidermiddlewares.referer.OriginPolicy**

<https://www.w3.org/TR/referrer-policy/#referrer-policy-origin>

“起源”策略规定，当从特定请求客户端发出同源请求和跨源请求时，只有请求客户端源的ASCII序列号作为引用者信息发送。

**Class scrapy.spidermiddlewares.referer.StrictOriginPolicy**

<https://www.w3.org/TR/referrer-policy/#referrer-policy-strict-origin>

“严格来源”策略在发出请求时发送请求客户端的源代码的ASCII序列化： - 从受TLS保护的环境设置对象到可能可信的URL，以及 - 从非TLS保护的环境设置对象到任何起源。

另一方面，从TLS保护的请求客户端请求非潜在可信的URL将不包含引荐者信息。Referer HTTP头将不会被发送。

**Class scrapy.spidermiddlewares.referer.OriginWhenCrossOriginPolicy**

<https://www.w3.org/TR/referrer-policy/#referrer-policy-origin-when-cross-origin>

“来源交叉源”策略规定，当从特定的请求客户端发出同源请求时，剥离用作引用者的完整URL作为引用者信息发送，并且只有源请求客户端在发出来自特定请求客户端的跨源请求时作为引用者信息发送。

**Class scrapy.spidermiddlewares.referer.StrictOriginWhenCrossOriginPolicy**

<https://www.w3.org/TR/referrer-policy/#referrer-policy-strict-origin-when-cross-origin>

“严格来源时，交叉来源”政策规定，剥离用作引荐来源的完整URL作为引荐来源信息发送，当从特定请求客户端发出同源请求时，只有ASCII的序列化请求客户端进行跨域请求时的起源：

* 从受TLS保护的环境设置对象到可能可信的URL，以及
* 从非TLS保护的环境设置对象到任何原点。

另一方面，来自受TLS保护的客户端请求的不可信的URL将不包含引荐者信息。Referer HTTP头将不会被发送。

**Class scrapy.spidermiddlewares.referer.UnsafeUrlPolicy**

<https://www.w3.org/TR/referrer-policy/#referrer-policy-unsafe-url>

“unsafe-url”策略指定一个完整的URL被剥离用作引用链接，并且同时发送来自特定请求客户端的跨源请求和同源请求。

注意：保单的名称不是谎言; 这是不安全的。此政策会将来自TLS保护资源的来源和路径泄漏到不安全的来源。仔细考虑为可能敏感的文档设置此策略的影响。

**警告**

不建议使用“不安全URL”策略。

### UrlLengthMiddleware

**Class scrapy.spidermiddlewares.urllength.UrlLengthMiddleware**

筛选出URL长度超过URLLENGTH\_LIMIT的请求

该[**UrlLengthMiddleware**](https://doc.scrapy.org/en/latest/topics/spider-middleware.html#scrapy.spidermiddlewares.urllength.UrlLengthMiddleware)可通过以下设置进行配置（详情参见设置文档）：

* [**URLLENGTH\_LIMIT**](https://doc.scrapy.org/en/latest/topics/settings.html#std:setting-URLLENGTH_LIMIT) - 允许抓取的网址的最大网址长度。

# 扩展

扩展框架提供了一种将您自己的自定义功能插入到Scrapy中的机制。

扩展只是在扩展初始化时在Scrapy启动时实例化的常规类。

## 扩展设置

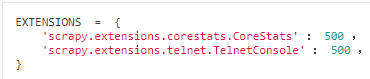
与其他Scrapy代码一样，扩展使用[Scrapy设置](https://doc.scrapy.org/en/latest/topics/settings.html#topics-settings)来管理其设置。

扩展名通常以其自己的名称作为其设置的前缀，以避免与现有（和未来）扩展名冲突。例如，处理[Google Sitemaps](https://en.wikipedia.org/wiki/Sitemaps)的假设性扩展会使用像GOOGLESITEMAP\_ENABLED，GOOGLESITEMAP\_DEPTH等设置 。

## 加载和激活扩展

扩展在启动时通过实例化扩展类的单个实例来加载和激活。因此，所有的扩展初始化代码必须在类构造函数（\_\_init\_\_方法）中执行。

要使扩展程序可用，请将其添加到[**EXTENSIONS**](https://doc.scrapy.org/en/latest/topics/settings.html#std:setting-EXTENSIONS)Scrapy设置中的设置。在中[**EXTENSIONS**](https://doc.scrapy.org/en/latest/topics/settings.html#std:setting-EXTENSIONS)，每个扩展都用一个字符串表示：扩展类名的完整Python路径。例如：



如您所见，该[**EXTENSIONS**](https://doc.scrapy.org/en/latest/topics/settings.html#std:setting-EXTENSIONS)设置是一个字典，其中的键是扩展路径，它们的值是定义扩展加载顺序的顺序。该[**EXTENSIONS**](https://doc.scrapy.org/en/latest/topics/settings.html#std:setting-EXTENSIONS)设置与[**EXTENSIONS\_BASE**](https://doc.scrapy.org/en/latest/topics/settings.html#std:setting-EXTENSIONS_BASE)Scrapy中定义的设置合并 （并不意味着被覆盖），然后按顺序排序以获得最终的已启用扩展的排序列表。

由于扩展通常不相互依赖，因此在大多数情况下，加载顺序无关紧要。这就是为什么该[**EXTENSIONS\_BASE**](https://doc.scrapy.org/en/latest/topics/settings.html#std:setting-EXTENSIONS_BASE)设置使用相同的顺序（0）定义所有扩展。但是，如果您需要添加取决于已加载的其他扩展的扩展，则可以利用此功能。

## 可用，启用和禁用的扩展

并非所有可用的分机都将启用。其中一些通常取决于特定的设置。例如，默认情况下，HTTP Cache扩展可用，但除非[**HTTPCACHE\_ENABLED**](https://doc.scrapy.org/en/latest/topics/downloader-middleware.html#std:setting-HTTPCACHE_ENABLED)设置已设置，否则禁用。

## 禁用扩展

为了禁用默认启用的扩展（即[**EXTENSIONS\_BASE**](https://doc.scrapy.org/en/latest/topics/settings.html#std:setting-EXTENSIONS_BASE)设置中包含的扩展），您必须将其顺序设置为None。例如：



## 编写自己的扩展

每个扩展都是一个Python类。Scrapy扩展的主要入口点（这也包括中间件和管道）是from\_crawler接收Crawler实例的类方法。通过Crawler对象，您可以访问设置，信号，统计信息并控制爬行行为。

通常，分机连接到[信号](https://doc.scrapy.org/en/latest/topics/signals.html#topics-signals)并执行由它们触发的任务。

最后，如果该from\_crawler方法引发 [**NotConfigured**](https://doc.scrapy.org/en/latest/topics/exceptions.html#scrapy.exceptions.NotConfigured)异常，则扩展将被禁用。否则，该扩展将被启用。

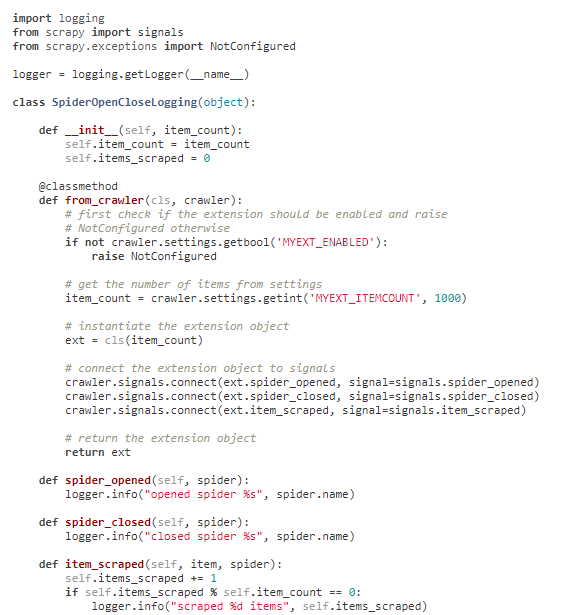
### 示例扩展

这里我们将实现一个简单的扩展来说明上一节中描述的概念。每次该扩展都会记录一条消息：

* 一只蜘蛛被打开
* 一只蜘蛛关闭
* 特定数量的项目被刮掉

该分机将通过MYEXT\_ENABLED设置启用，并且项目数量将通过MYEXT\_ITEMCOUNT设置指定。

这是这种扩展的代码：



## 内置扩展参考

### 通用扩展

#### 日志统计扩展

**Class scrapy.extensions.logstats.LogStats**

记录抓取的页面和抓取的项目等基本统计信息。

#### 核心统计扩展

**Class scrapy.extensions.corestats.CoreStats**

如果启用了统计信息收集，请启用核心统计信息收集（请参阅[统计信息收集](https://doc.scrapy.org/en/latest/topics/stats.html#topics-stats)）。

#### Telnet控制台扩展

**Class scrapy.extensions.telnet.TelnetConsole**

提供一个Telnet控制台，用于在当前运行的Scrapy过程中进入Python解释器，这对调试非常有用。

远程登录控制台必须通过该[**TELNETCONSOLE\_ENABLED**](https://doc.scrapy.org/en/latest/topics/settings.html#std:setting-TELNETCONSOLE_ENABLED) 设置启用，服务器将监听端口中指定的端口[**TELNETCONSOLE\_PORT**](https://doc.scrapy.org/en/latest/topics/telnetconsole.html#std:setting-TELNETCONSOLE_PORT)。

#### 内存使用扩展

***Class* scrapy.extensions.memusage.MemoryUsage**

**注意**

此扩展在Windows中不起作用。

监控运行蜘蛛的Scrapy进程使用的内存，并：

1. 超过某个值时发送通知电子邮件
2. 超过一定值时关闭蜘蛛

当达到某个警告值（[**MEMUSAGE\_WARNING\_MB**](https://doc.scrapy.org/en/latest/topics/settings.html#std:setting-MEMUSAGE_WARNING_MB)）并且达到最大值（）时，通知电子邮件可以被触发，[**MEMUSAGE\_LIMIT\_MB**](https://doc.scrapy.org/en/latest/topics/settings.html#std:setting-MEMUSAGE_LIMIT_MB)这也会导致蜘蛛被关闭并且Scrapy进程被终止。

该扩展程序由该[**MEMUSAGE\_ENABLED**](https://doc.scrapy.org/en/latest/topics/settings.html#std:setting-MEMUSAGE_ENABLED)设置启用，并且可以使用以下设置进行配置：

* [**MEMUSAGE\_LIMIT\_MB**](https://doc.scrapy.org/en/latest/topics/settings.html#std:setting-MEMUSAGE_LIMIT_MB)
* [**MEMUSAGE\_WARNING\_MB**](https://doc.scrapy.org/en/latest/topics/settings.html#std:setting-MEMUSAGE_WARNING_MB)
* [**MEMUSAGE\_NOTIFY\_MAIL**](https://doc.scrapy.org/en/latest/topics/settings.html#std:setting-MEMUSAGE_NOTIFY_MAIL)
* [**MEMUSAGE\_CHECK\_INTERVAL\_SECONDS**](https://doc.scrapy.org/en/latest/topics/settings.html#std:setting-MEMUSAGE_CHECK_INTERVAL_SECONDS)

#### 内存调试器扩展

**Class scrapy.extensions.memdebug.MemoryDebugger**

调试内存使用的扩展。它收集有关以下信息：

* Python垃圾收集器未收集的对象
* 不应该留下活着的物体。有关更多信息，请参阅[使用trackref调试内存泄漏](https://doc.scrapy.org/en/latest/topics/leaks.html#topics-leaks-trackrefs)

要启用此扩展程序，请打开[**MEMDEBUG\_ENABLED**](https://doc.scrapy.org/en/latest/topics/settings.html#std:setting-MEMDEBUG_ENABLED)设置。信息将存储在统计信息中。

#### 关闭蜘蛛扩展

**Class scrapy.extensions.closespider.CloseSpider**

在满足某些条件时自动关闭蜘蛛，并针对每种情况使用特定的关闭原因。

关闭蜘蛛的条件可以通过以下设置进行配置：

* [**CLOSESPIDER\_TIMEOUT**](https://doc.scrapy.org/en/latest/topics/extensions.html#std:setting-CLOSESPIDER_TIMEOUT)
* [**CLOSESPIDER\_ITEMCOUNT**](https://doc.scrapy.org/en/latest/topics/extensions.html#std:setting-CLOSESPIDER_ITEMCOUNT)
* [**CLOSESPIDER\_PAGECOUNT**](https://doc.scrapy.org/en/latest/topics/extensions.html#std:setting-CLOSESPIDER_PAGECOUNT)
* [**CLOSESPIDER\_ERRORCOUNT**](https://doc.scrapy.org/en/latest/topics/extensions.html#std:setting-CLOSESPIDER_ERRORCOUNT)

##### CLOSESPIDER\_TIMEOUT

默认： 0

一个指定秒数的整数。如果蜘蛛保持打开的时间超过这个秒数，它将会自动关闭closespider\_timeout。如果为零（或未设置），则蜘蛛将不会因超时而关闭。

##### CLOSESPIDER\_ITEMCOUNT

默认： 0

一个指定项目数量的整数。如果蜘蛛的刮伤量超过了这个数量，并且这些物品通过了物品管道，蜘蛛将会被关闭closespider\_itemcount。目前处于下载队列中的[**CONCURRENT\_REQUESTS**](https://doc.scrapy.org/en/latest/topics/settings.html#std:setting-CONCURRENT_REQUESTS)请求（直至 请求）仍然处理。如果为零（或未设置），蜘蛛将不会按传递项目的数量关闭。

##### CLOSESPIDER\_PAGECOUNT

*0.11版本的新功能。*

默认： 0

一个整数，指定要抓取的最大响应数。如果蜘蛛爬得比这更多，蜘蛛就会被原因关闭closespider\_pagecount。如果为零（或未设置），则爬网的响应数量将不会关闭蜘蛛。

##### CLOSESPIDER\_ERRORCOUNT

*0.11版本的新功能。*

默认： 0

一个整数，指定在关闭蜘蛛之前接收的最大错误数。如果蜘蛛生成的错误数量超过了这个数量，它将会因为原因而被关闭closespider\_errorcount。如果为零（或未设置），则蜘蛛将不会按错误数量关闭。

#### StatsMailer扩展

**Class scrapy.extensions.statsmailer.StatsMailer**

这个简单的扩展可用于每次域名完成抓取时发送通知电子邮件，包括收集的Scrapy统计信息。电子邮件将发送给[**STATSMAILER\_RCPTS**](https://doc.scrapy.org/en/latest/topics/settings.html#std:setting-STATSMAILER_RCPTS) 设置中指定的所有收件人。

### 调试扩展

#### 堆栈跟踪转储扩展

**Class scrapy.extensions.debug.StackTraceDump**

在 收到[SIGQUIT](https://en.wikipedia.org/wiki/SIGQUIT)或[SIGUSR2](https://en.wikipedia.org/wiki/SIGUSR1_and_SIGUSR2)信号时转储有关正在运行的进程的信息。所甩的信息如下：

1. 引擎状态（使用scrapy.utils.engine.get\_engine\_status()）
2. 活动引用（请参阅[使用trackref调试内存泄漏](https://doc.scrapy.org/en/latest/topics/leaks.html#topics-leaks-trackrefs)）
3. 所有线程的堆栈跟踪

堆栈跟踪和引擎状态转储后，Scrapy进程继续正常运行。

此扩展只适用于符合POSIX的平台（即非Windows），因为[SIGQUIT](https://en.wikipedia.org/wiki/SIGQUIT)和[SIGUSR2](https://en.wikipedia.org/wiki/SIGUSR1_and_SIGUSR2)信号在Windows上不可用。

至少有两种方式向Scrapy发送[SIGQUIT](https://en.wikipedia.org/wiki/SIGQUIT)信号：

1. 在Scrapy进程运行时按下Ctrl键（仅限Linux？）
2. 通过运行这个命令（假设<pid>是Scrapy进程的进程ID）：



#### 调试器扩展

**Class scrapy.extensions.debug.Debugger**

在 收到[SIGUSR2](https://en.wikipedia.org/wiki/SIGUSR1_and_SIGUSR2)信号时，在正在运行的Scrapy过程中调用[Python调试器](https://docs.python.org/2/library/pdb.html)。调试器退出后，Scrapy进程继续正常运行。

有关更多信息，请参阅Python中的调试。

此扩展只适用于符合POSIX的平台（即非Windows）。

# Signals（信号）

Scrapy广泛使用信号来通知特定事件发生的时间。您可以捕获Scrapy项目中的一些信号（例如使用[扩展](https://doc.scrapy.org/en/latest/topics/extensions.html#topics-extensions)）来执行其他任务或扩展Scrapy以添加不提供的功能。

即使信号提供了多个参数，捕获它们的处理程序也不需要接受所有这些参数 - 信号分派机制只会传递处理程序接收到的参数。

您可以通过[Signals API](https://doc.scrapy.org/en/latest/topics/api.html#topics-api-signals)连接到信号（或发送自己的 [信号）](https://doc.scrapy.org/en/latest/topics/api.html#topics-api-signals)。

下面是一个简单的例子，展示如何捕捉信号并执行一些操作：



## 延迟信号处理程序

某些信号支持从处理程序返回[Twisted延迟](https://twistedmatrix.com/documents/current/core/howto/defer.html)，请参阅下面的[内置信号参考](https://doc.scrapy.org/en/latest/topics/signals.html#topics-signals-ref)以了解哪些[信号](https://doc.scrapy.org/en/latest/topics/signals.html#topics-signals-ref)。

## 内置信号参考

以下是Scrapy内置信号列表及其含义。

### engine\_started

**scrapy.signals.engine\_started（）**

当Scrapy引擎开始爬行时发送。

该信号支持从处理程序返回延迟。

**注意**

这个信号可能被解雇后的[**spider\_opened**](https://doc.scrapy.org/en/latest/topics/signals.html#std:signal-spider_opened)信号，这取决于蜘蛛是如何开始。所以**不要**依赖这个信号之前被解雇[**spider\_opened**](https://doc.scrapy.org/en/latest/topics/signals.html#std:signal-spider_opened)。

### engine\_stopped

**scrapy.signals.engine\_stopped（）**

Scrapy引擎停止时发送（例如，爬网过程完成时）。

该信号支持从处理程序返回延迟。

### item\_scraped

**scrapy.signals.item\_scraped(item, response, spider)**

当一件物品已经通过所有 [物品管线](https://doc.scrapy.org/en/latest/topics/item-pipeline.html#topics-item-pipeline)阶段（没有被丢弃）后被发送。

该信号支持从处理程序返回延迟。

|  |  |
| --- | --- |
| **参数：** | * **item** (dict or [**Item**](https://doc.scrapy.org/en/latest/topics/items.html#scrapy.item.Item) object) - 项目被刮掉 * **spider** ([**Spider**](https://doc.scrapy.org/en/latest/topics/spiders.html#scrapy.spiders.Spider) object)  - 刮掉物品的蜘蛛 * **response** ([**Response**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.Response) object)  - 项目被刮取的响应 |

### item\_dropped

**scrapy.signals.item\_dropped(item, response, exception, spider)**

当某个阶段发生异常时，从[项目管道中](https://doc.scrapy.org/en/latest/topics/item-pipeline.html#topics-item-pipeline)删除某个项目后发送[**DropItem**](https://doc.scrapy.org/en/latest/topics/exceptions.html#scrapy.exceptions.DropItem)。

该信号支持从处理程序返回延迟。

|  |  |
| --- | --- |
| **参数：** | * **item** (dict or [**Item**](https://doc.scrapy.org/en/latest/topics/items.html#scrapy.item.Item) object)  - 从[项目管道中](https://doc.scrapy.org/en/latest/topics/item-pipeline.html#topics-item-pipeline)删除的[项目](https://doc.scrapy.org/en/latest/topics/item-pipeline.html#topics-item-pipeline) * **spider** ([**Spider**](https://doc.scrapy.org/en/latest/topics/spiders.html#scrapy.spiders.Spider) object)  - 刮掉物品的蜘蛛 * **response** ([**Response**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.Response) object)  - 项目被放置的响应 * **exception** ([**DropItem**](https://doc.scrapy.org/en/latest/topics/exceptions.html#scrapy.exceptions.DropItem) exception)  - [**DropItem**](https://doc.scrapy.org/en/latest/topics/exceptions.html#scrapy.exceptions.DropItem)导致项目被删除的异常（必须是 子类） |

### spider\_closed

**scrapy.signals.spider\_closed(spider, reason)**

蜘蛛关闭后发送。这可以用来释放保留的每个蜘蛛资源[**spider\_opened**](https://doc.scrapy.org/en/latest/topics/signals.html#std:signal-spider_opened)。

该信号支持从处理程序返回延迟。

|  |  |
| --- | --- |
| **参数：** | * **spider** ([**Spider**](https://doc.scrapy.org/en/latest/topics/spiders.html#scrapy.spiders.Spider) object) - 已关闭的蜘蛛 * **reason**（*str*） - 一个描述蜘蛛关闭原因的字符串。如果因为蜘蛛已经完成了抓取而关闭，原因是**'finished'**。否则，如果通过调用**close\_spider**引擎方法手动关闭了蜘蛛，则原因是该**reason**方法参数中传递的原因（默认为 **'cancelled'**）。如果引擎被关闭（例如，通过按Ctrl-C来停止它）原因将会是**'shutdown'**。 |

### spider\_opened

**scrapy.signals.spider\_opened(spider)**

在蜘蛛被打开爬行后发送。这通常用于保留每个蜘蛛资源，但可以用于打开蜘蛛时需要执行的任何任务。

该信号支持从处理程序返回延迟。

|  |  |
| --- | --- |
| **参数：** | **spider** ([**Spider**](https://doc.scrapy.org/en/latest/topics/spiders.html#scrapy.spiders.Spider) object)  - 已被打开的蜘蛛 |

### spider\_idle

**scrapy.signals.spider\_idle(spider)**

当蜘蛛闲置时发出，这意味着蜘蛛没有进一步的发现：

* 等待下载的请求
* 请求安排
* 项目流水线中正在处理的项目

如果在此信号的所有处理程序完成后，空闲状态仍然存在，则引擎开始关闭蜘蛛。蜘蛛完成关闭后，[**spider\_closed**](https://doc.scrapy.org/en/latest/topics/signals.html#std:signal-spider_closed)发送信号。

你可以举一个[**DontCloseSpider**](https://doc.scrapy.org/en/latest/topics/exceptions.html#scrapy.exceptions.DontCloseSpider)例外来防止蜘蛛被关闭。

该信号不支持从处理程序返回延迟。

|  |  |
| --- | --- |
| **参数：** | **spider** ([**Spider**](https://doc.scrapy.org/en/latest/topics/spiders.html#scrapy.spiders.Spider) object) - 空闲的蜘蛛 |

**注意**

在您的[**spider\_idle**](https://doc.scrapy.org/en/latest/topics/signals.html#std:signal-spider_idle)处理程序中调度某些请求并 **不能**保证它可以防止蜘蛛被关闭，尽管有时可以。这是因为如果调度程序拒绝所有计划的请求（例如，由于重复进行过滤），则蜘蛛可能仍保持空闲状态。

### spider\_error

**scrapy.signals.spider\_error（failure, response, spider）**

当一个蜘蛛回调产生一个错误（即引发一个异常）时发送。

该信号不支持从处理程序返回延迟。

|  |  |
| --- | --- |
| **参数：** | * **failure** ([Failure](https://twistedmatrix.com/documents/current/api/twisted.python.failure.Failure.html) object)  - 作为Twisted [Failure](https://twistedmatrix.com/documents/current/api/twisted.python.failure.Failure.html)对象引发的异常 * **response** ([**Response**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.Response) object) - 引发异常时处理的响应 * **spider** ([**Spider**](https://doc.scrapy.org/en/latest/topics/spiders.html#scrapy.spiders.Spider) object)  - 引发异常的蜘蛛 |

### request\_scheduled

**scrapy.signals.request\_scheduled(request, spider)**

当引擎安排一个[**Request**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.Request)时发送**，**稍后下载。

该信号不支持从处理程序返回延迟。

|  |  |
| --- | --- |
| **参数：** | * **request** ([**Request**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.Request) object) - 到达调度程序的请求 * **spider** ([**Spider**](https://doc.scrapy.org/en/latest/topics/spiders.html#scrapy.spiders.Spider) object) – - 产生请求的蜘蛛 |

### request\_dropped

**scrapy.signals.request\_dropped(request, spider)**

当[**Request**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.Request)稍后下载的引擎安排的a 被调度程序拒绝时发送。

该信号不支持从处理程序返回延迟。

|  |  |
| --- | --- |
| **参数：** | * **request** ([**Request**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.Request) object) - 到达调度程序的请求 * **spider** ([**Spider**](https://doc.scrapy.org/en/latest/topics/spiders.html#scrapy.spiders.Spider) object) - 产生请求的蜘蛛 |

### response\_received

**scrapy.signals.response\_received(response, request, spider)**

当引擎收到[**Response**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.Response)来自下载器的新消息时发送。

该信号不支持从处理程序返回延迟。

|  |  |
| --- | --- |
| **参数：** | * **response** ([**Response**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.Response) object) - 收到的响应 * **request** ([**Request**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.Request) object) - 生成响应的请求 * **spider** ([**Spider**](https://doc.scrapy.org/en/latest/topics/spiders.html#scrapy.spiders.Spider) object)  - 反应意图的蜘蛛 |

### response\_downloaded

**scrapy.signals.response\_downloaded(response, request, spider)**

下载后立即由下载器发送**HTTPResponse**。

该信号不支持从处理程序返回延迟。

|  |  |
| --- | --- |
| **参数：** | * **response** ([**Response**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.Response) object) - 下载的响应 * **request** ([**Request**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.Request) object)  - 生成响应的请求 * **spider** ([**Spider**](https://doc.scrapy.org/en/latest/topics/spiders.html#scrapy.spiders.Spider) object)  - 反应意图的蜘蛛 |

# Item Exporters

一旦你刮掉了你的物品，你通常会想要坚持或导出这些物品，以便在其他应用程序中使用这些数据。毕竟，这是刮削过程的全部目的。

为此，Scrapy为不同的输出格式提供了一组项目导出器，如XML，CSV或JSON。

## Using Item Exporters

如果您很匆忙，只想使用项目导出器来输出抓取的数据，请参阅[Feed输出](https://doc.scrapy.org/en/latest/topics/feed-exports.html#topics-feed-exports)。否则，如果您想知道Item Exporters如何工作或需要更多自定义功能（不包括默认导出），请继续阅读以下内容。

为了使用Item Exporter，你必须用它需要的参数来实例化它。每个项目导出器需要不同的参数，因此请检查每个导出器文档以确保在“ [内置项目导出器”参考中](https://doc.scrapy.org/en/latest/topics/exporters.html#topics-exporters-reference)。在您实例化出口商后，您必须：

1.调用该方法[**start\_exporting()**](https://doc.scrapy.org/en/latest/topics/exporters.html#scrapy.exporters.BaseItemExporter.start_exporting)以指示输出过程的开始

2.调用[**export\_item()**](https://doc.scrapy.org/en/latest/topics/exporters.html#scrapy.exporters.BaseItemExporter.export_item)您想要导出的每个项目的方法

3.最后调用该[**finish\_exporting()**](https://doc.scrapy.org/en/latest/topics/exporters.html#scrapy.exporters.BaseItemExporter.finish_exporting)信号表示导出过程结束

在这里您可以看到一个[项目管道](https://doc.scrapy.org/en/latest/topics/item-pipeline.html)，它使用多个项目导出器根据其中一个字段的值将分组项目分组到不同的文件中：



## 项目字段的序列化

默认情况下，字段值不加修改地传递给底层的序列化库，并且如何序列化它们的决定被委托给每个特定的序列化库。

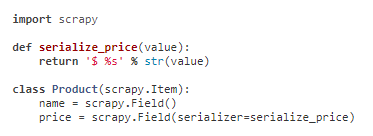
但是，您可以在将每个字段值序列化为序列化之前自定义其序列化库。

有两种方法可以自定义字段序列化的方式，这将在下面介绍。

### 1.在现场声明一个序列化器

如果你使用[**Item**](https://doc.scrapy.org/en/latest/topics/items.html#scrapy.item.Item)你可以在[字段元数据中](https://doc.scrapy.org/en/latest/topics/items.html#topics-items-fields)声明一个序列化器 。序列化程序必须是可调用的，它接收一个值并返回其序列化形式。

例：

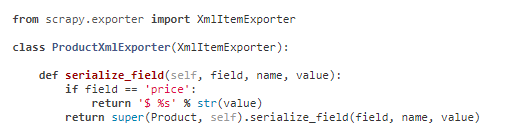


### 2.重写serialize\_field（）方法

您也可以重写该[**serialize\_field()**](https://doc.scrapy.org/en/latest/topics/exporters.html#scrapy.exporters.BaseItemExporter.serialize_field)方法来自定义字段值的导出方式。

确保[**serialize\_field()**](https://doc.scrapy.org/en/latest/topics/exporters.html#scrapy.exporters.BaseItemExporter.serialize_field)在自定义代码之后调用基类方法。

例：



## 内置项目导出器参考

以下是与Scrapy捆绑在一起的Item Exporters列表。其中一些包含输出示例，它们假设您正在导出这两个项目：



### BaseItemExporter

**classscrapy.exporters.BaseItemExporter（fields\_to\_export = None，export\_empty\_fields = False，encoding ='utf-8'，indent = 0 ）**

这是所有项目导出器的（抽象）基类。它提供对所有（具体）项目导出器使用的常用功能的支持，例如定义要导出的字段，是否导出空字段或使用哪种编码。

这些功能可以通过该填充它们各自的实例属性的构造器参数进行配置：[**fields\_to\_export**](https://doc.scrapy.org/en/latest/topics/exporters.html#scrapy.exporters.BaseItemExporter.fields_to_export)，[**export\_empty\_fields**](https://doc.scrapy.org/en/latest/topics/exporters.html#scrapy.exporters.BaseItemExporter.export_empty_fields)，[**encoding**](https://doc.scrapy.org/en/latest/topics/exporters.html#scrapy.exporters.BaseItemExporter.encoding)，[**indent**](https://doc.scrapy.org/en/latest/topics/exporters.html#scrapy.exporters.BaseItemExporter.indent)。

**export\_item(item)**

导出给定的项目。这个方法必须在子类中实现。

**serialize\_field(field, name, value)**

返回给定字段的序列化值。如果要控制如何序列化/导出特定字段或值，则可以覆盖此方法（在自定义项目导出器中）。

默认情况下，此方法查找[在项目字段中声明](https://doc.scrapy.org/en/latest/topics/exporters.html#topics-exporters-serializers)的序列化程序[，](https://doc.scrapy.org/en/latest/topics/exporters.html#topics-exporters-serializers)并返回将该序列化程序应用于该值的结果。如果没有找到序列化程序，则返回值不变，除了使用在属性中声明的编码**unicode**进行编码的值 。**str**[**encoding**](https://doc.scrapy.org/en/latest/topics/exporters.html#scrapy.exporters.BaseItemExporter.encoding)

|  |  |
| --- | --- |
| **参数：** | * **field** ([**Field**](https://doc.scrapy.org/en/latest/topics/items.html#scrapy.item.Field) object or an empty dict) - 字段被序列化。如果原始字典正在导出（不[**Item**](https://doc.scrapy.org/en/latest/topics/items.html#scrapy.item.Item)）*字段*值是一个空的字典。 * **name**（*str*） - 被序列化的字段的名称 * **value** - 正在序列化的值 |

**start\_exporting（）**

指示出口过程的开始。一些出口商可能会使用它来生成一些必需的标题（例如，[**XmlItemExporter**](https://doc.scrapy.org/en/latest/topics/exporters.html#scrapy.exporters.XmlItemExporter)）。您必须在导出任何项目之前调用此方法。

**finish\_exporting（）**

指示出口过程的结束。一些出口商可能会使用它来生成一些必需的页脚（例如，[**XmlItemExporter**](https://doc.scrapy.org/en/latest/topics/exporters.html#scrapy.exporters.XmlItemExporter)）。在没有更多项目要导出后，您必须始终调用此方法。

**fields\_to\_export**

包含要导出的字段名称的列表，如果您要导出所有字段，则为None。默认为None。

一些出口商（如[**CsvItemExporter**](https://doc.scrapy.org/en/latest/topics/exporters.html#scrapy.exporters.CsvItemExporter)）尊重该属性中定义的字段的顺序。

一些出口商可能需要fields\_to\_export列表，以便在蜘蛛返回字典（不是**Item**实例）时正确导出数据。

**export\_empty\_fields**

是否在导出的数据中包含空白/未填充的项目字段。默认为**False**。一些出口商（如[**CsvItemExporter**](https://doc.scrapy.org/en/latest/topics/exporters.html#scrapy.exporters.CsvItemExporter)）忽略这个属性，并始终导出所有空字段。

字典项目将忽略此选项。

**encoding**

将用于编码unicode值的编码。这只会影响unicode值（它总是使用此编码序列化为str）。其他值类型不变地传递给特定的序列化库。

**indent**

用于缩进每个级别的输出的空间量。默认为**0**。

* **indent=None** 选择最紧凑的表示形式，同一行中的所有项目都没有缩进
* **indent<=0** 每个项目在其自己的行，没有缩进
* **indent>0** 每个项目在它自己的行上，用提供的数字值缩进

### XmlItemExporter

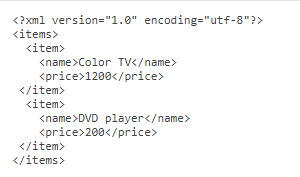
**classscrapy.exporters.XmlItemExporter（file，item\_element ='item'，root\_element ='items'，\*\* kwargs ）**

将XML格式的项目导出到指定的文件对象。

|  |  |
| --- | --- |
| **参数：** | * **file**- 用于导出数据的文件类对象。它的**write**方法应该接受**bytes**（以二进制模式打开的磁盘文件，**io.BytesIO**对象等） * **root\_element**（str） - 导出的XML中的根元素的名称。 * **item\_element**（str） - 导出的XML中每个项目元素的名称。 |

此构造函数的其他关键字参数传递给 [**BaseItemExporter**](https://doc.scrapy.org/en/latest/topics/exporters.html#scrapy.exporters.BaseItemExporter)构造函数。

这个出口商的典型产出是：

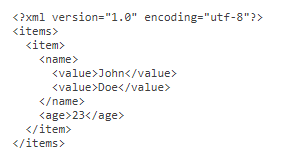


除非在**serialize\_field()**方法中重写，否则通过序列化**<value>**元素内的每个值来导出多值字段。这是为了方便，因为多值字段非常常见。

例如，该项目：



将被序列化为：



### CsvItemExporter

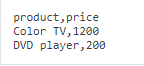
**classscrapy.exporters.CsvItemExporter(file, include\_headers\_line=True, join\_multivalued=', ', \*\*kwargs)**

将CSV格式的项目导出到给定的类文件对象。如果该 **fields\_to\_export**属性已设置，则将用于定义CSV列及其顺序。该**export\_empty\_fields**属性对此导出器没有影响。

|  |  |
| --- | --- |
| **参数：** | * **file**  - 用于导出数据的文件类对象。它的**write**方法应该接受**bytes**（以二进制模式打开的磁盘文件，**io.BytesIO**对象等） * **include\_headers\_line**（str） - 如果启用，使导出器输出包含从[**BaseItemExporter.fields\_to\_export**](https://doc.scrapy.org/en/latest/topics/exporters.html#scrapy.exporters.BaseItemExporter.fields_to_export)第一个导出项目字段获取的字段名称的标题行。 * **join\_multivalued** - 将用于加入多值字段的char（或字符），如果找到。 |

此构造函数的其他关键字参数传递给 [**BaseItemExporter**](https://doc.scrapy.org/en/latest/topics/exporters.html#scrapy.exporters.BaseItemExporter)构造函数，以及[csv.writer](https://docs.python.org/2/library/csv.html#csv.writer)构造函数的剩余参数 ，因此您可以使用任何csv.writer构造函数参数来定制此导出器。

这个出口商的典型产出是：



### PickleItemExporter

**Class scrapy.exporters.PickleItemExporter(file, protocol=0, \*\*kwargs)**

将项目以pickle格式导出到给定的类文件对象。

|  |  |
| --- | --- |
| **参数：** | * **file** - 用于导出数据的文件类对象。它的**write**方法应该接受**bytes**（以二进制模式打开的磁盘文件，**io.BytesIO**对象等） * **protocol**（int） - 使用的pickle协议。 |

有关更多信息，请参阅[pickle模块文档](https://docs.python.org/2/library/pickle.html)。

此构造函数的其他关键字参数传递给 [**BaseItemExporter**](https://doc.scrapy.org/en/latest/topics/exporters.html#scrapy.exporters.BaseItemExporter)构造函数。

Pickle不是一种人们可读的格式，所以没有提供输出示例。

### PprintItemExporter

**Class scrapy.exporters.PprintItemExporter(file, \*\*kwargs)**

以漂亮打印格式将项目导出到指定的文件对象。

|  |  |
| --- | --- |
| **参数：** | **file** - 用于导出数据的文件类对象。它的**write**方法应该接受**bytes**（以二进制模式打开的磁盘文件，**io.BytesIO**对象等） |

此构造函数的其他关键字参数传递给 [**BaseItemExporter**](https://doc.scrapy.org/en/latest/topics/exporters.html#scrapy.exporters.BaseItemExporter)构造函数。

这个出口商的典型产出是：



较长的行（当存在时）格式很好。

### JsonItemExporter

**Class scrapy.exporters.JsonItemExporter(file, \*\*kwargs)**

将JSON格式的项目导出到指定的类文件对象，将所有对象写为对象列表。额外的构造函数参数传递给[**BaseItemExporter**](https://doc.scrapy.org/en/latest/topics/exporters.html#scrapy.exporters.BaseItemExporter)构造函数，以及[JSONEncoder](https://docs.python.org/2/library/json.html#json.JSONEncoder)构造函数的剩余参数，因此您可以使用任何 [JSONEncoder](https://docs.python.org/2/library/json.html#json.JSONEncoder)构造函数参数来自定义此导出器。

|  |  |
| --- | --- |
| **参数：** | **file** - 用于导出数据的文件类对象。它的**write**方法应该接受**bytes**（以二进制模式打开的磁盘文件，**io.BytesIO**对象等） |

这个出口商的典型产出是：



**警告**

JSON是非常简单和灵活的序列化格式，但是对于大量数据来说它不能很好地扩展，因为在JSON解析器（任何语言）之间，增量（即流模式）解析得不到很好的支持（如果有的话），并且他们中的大多数只是解析整个对象在内存中。如果您希望JSON的强大功能和简单性以更流的格式，请考虑使用[**JsonLinesItemExporter**](https://doc.scrapy.org/en/latest/topics/exporters.html#scrapy.exporters.JsonLinesItemExporter) ，或者将输出分成多个块。

### JsonLinesItemExporter

**Class scrapy.exporters.JsonLinesItemExporter(file, \*\*kwargs)**

将JSON格式的项目导出到指定的类文件对象，每行写入一个JSON编码的项目。额外的构造函数参数传递给[**BaseItemExporter**](https://doc.scrapy.org/en/latest/topics/exporters.html#scrapy.exporters.BaseItemExporter)构造函数，以及[JSONEncoder](https://docs.python.org/2/library/json.html#json.JSONEncoder)构造函数的剩余参数，因此您可以使用任何[JSONEncoder](https://docs.python.org/2/library/json.html#json.JSONEncoder) 构造函数参数来自定义此导出器。

|  |  |
| --- | --- |
| **参数：** | **file** - 用于导出数据的文件类对象。它的**write**方法应该接受**bytes**（以二进制模式打开的磁盘文件，**io.BytesIO**对象等） |

这个出口商的典型产出是：



与生成的不同[**JsonItemExporter**](https://doc.scrapy.org/en/latest/topics/exporters.html#scrapy.exporters.JsonItemExporter)，此导出器生成的格式非常适合序列化大量数据。